

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Essentials of the Java™ Programming Language: A Hands-On Guide, Part 2

by Monica Pawlan[\[CONTENTS\]](#) [\[NEXT>>\]](#)

This series of lessons builds on the material presented in [Java™ Programming Language Basics, Part 1](#), which introduced applications, applets, and servlets; simple file and database access operations; and remote method invocation (RMI).

The lessons and code examples for Part 2 are somewhat more complex. They walk you through network communications, building a user interface using more components, data encryption and decryption (pseudo code only), grouping multiple data elements into one object (collections), and internationalizing a program. Part 2 concludes with some object-oriented programming concepts.

Contents

Lesson 1: [Socket Communications](#)

- [What are Sockets and Threads?](#)
- [About the Examples](#)
- [Example 1: Server-Side Program](#)
- [Example 1: Client-Side Program](#)
- [Example 2: Multithreaded Server Example](#)
- [More Information](#)

Lesson 2: [User Interfaces Revisited](#)

- [About the Example](#)
- [Fruit Order Client Code](#)
 - [Global Variables](#)
 - [Constructor](#)
 - [Event Handling](#)
 - [Cursor Focus](#)
 - [Converting Strings to Numbers and Back](#)
- [Server Program Code](#)
- [View Order Client Code](#)
- [Program Improvements](#)

- [More Information](#)

Lesson 3: [Cryptography](#)

- [About the Example](#)
- [Running the Example](#)
- [Pseudo Code](#)
- [More Information](#)

Lesson 4: [Serialization](#)

- [About the Example](#)
- [Wrapping the Data](#)
- [Sending Data](#)
- [Server Program](#)
- [Receiving Data](#)
- [More Information](#)

Lesson 5: [Collections](#)

- [About Collections](#)
- [Creating a Set](#)
- [Printing](#)
- [More Information](#)

Lesson 6: [Internationalization](#)

- [Identify Culturally Dependent Data](#)
- [Create Keyword and Value Pair Files](#)
- [Internationalize Application Text](#)
- [Localize Numbers](#)
- [Compile and Run the Application](#)
- [Program Improvements](#)
- [More Information](#)

Lesson 7: [Packages and Java Archive File Format](#)

- [Setting up Class Packages](#)
 - [Create the Directories](#)
 - [Declare the Packages](#)
 - [Make Classes and Fields Accessible](#)
 - [Change Client Code to Find the Properties File](#)
 - [Compile and Run the Example](#)
- [Using JAR Files to Deploy](#)
 - [Server Set of Files](#)
 - [Fruit Order Client Set of Files](#)
 - [View Order Client Set of Files](#)
 - [More Information](#)

Lesson 8: [Object-Oriented Programming](#)

- [Object-Oriented Programming Defined](#)
- [Classes](#)
- [Objects](#)
- [Well-Defined Boundaries and Cooperation](#)
- [Inheritance](#)
- [Polymorphism](#)
- [Data Access Levels](#)
- [Your Own Classes](#)
- [Program Improvements](#)
- [More Information](#)

[In Closing](#)

Reader Feedback

Tell us what you think of this training book and earn two DukeDollars.

<input type="text" value="[Duke]"/>	<input type="radio"/> Very worth reading	<input type="radio"/> Worth reading	<input type="radio"/> Not worth reading
	If you have other comments or ideas for future training books, please type them here:		

[\[TOP\]](#)

[Print Button](#)

[This page was updated: 6-Apr-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)
[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: **(800) 786-7638**
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.

**Sun
Microsystem**

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 1: Socket Communications

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)

[Java™ Programming Language Basics, Part 1](#), finished with a simple network communications example using the Remote Method Invocation (RMI) application programming interface (API). The RMI example allows multiple client programs to communicate with the same server program without any explicit code to do this because the RMI API is built on sockets and threads.

This lesson presents a simple sockets-based program to introduce the concepts of sockets and multi-threaded programming. A multi-threaded program performs multiple tasks at one time such as fielding simultaneous requests from many client programs.

- [What are Sockets and Threads?](#)
- [About the Examples](#)
- [Example 1: Server-Side Program](#)
- [Example 1: Client-Side Program](#)
- [Example 2: Multithreaded Server Example](#)
- [More Information](#)

What are Sockets and Threads?

A socket is a software endpoint that establishes bidirectional communication between a server program and one or more client programs. The socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program.



A server program typically provides resources to a network of client programs. Client programs send requests to the server program, and the server program responds to the request.

One way to handle requests from more than one client is to make the server program multi-threaded. A multi-threaded server creates a thread for each communication it accepts from a client. A thread is a sequence of instructions that

run independently of the program and of any other threads.

Using threads, a multi-threaded server program can accept a connection from a client, start a thread for that communication, and continue listening for requests from other clients.

About the Examples

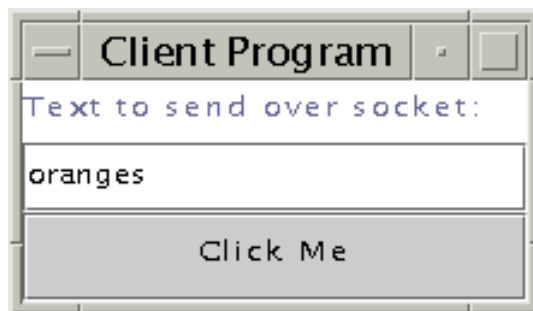
The examples for this lesson consist of two versions of the client and server program pair adapted from the [FileIO.java](#) application presented in [Part 1, Lesson 6: File Access and Permissions](#).

Example 1 sets up a client and server communication between one server program and one client program. The server program is not multi-threaded and cannot handle requests from more than one client.

Example 2 converts the server program to a multi-threaded version so it can handle requests from more than one client.

Example 1: Client-Side Behavior

The [client program](#) presents a simple user interface and prompts for text input. When you click the Click Me button, the text is sent to the server program. The client program expects an echo from the server and prints the echo it receives on its standard output.



Example 1: Server-Side Behavior

The [server program](#) presents a simple user interface, and when you click the Click Me button, the text received from the client is displayed. The server echoes the text it receives whether or not you click the Click Me button.



Example 1: Compile and Run

To run the example programs, start the server program first. If you do not, the client program cannot establish the socket connection.

Here are the compiler and interpreter commands to compile and run the example.

```
javac SocketServer.java
javac SocketClient.java

java SocketServer
java SocketClient
```

Example 1: Server-Side Program

The [server program](#) establishes a socket connection on Port 4321 in its `listenSocket` method. It reads data sent to it and sends that same data back to the server in its `actionPerformed` method.

listenSocket Method

The `listenSocket` method creates a `ServerSocket` object with the port number on which the server program is going to listen for client communications. The port number must be an available port, which means the number cannot be reserved or already in use. For example, Unix systems reserve ports 1 through 1023 for administrative functions leaving port numbers greater than 1024 available for use.

```
public void listenSocket(){
    try{
        server = new ServerSocket(4321);
    } catch (IOException e) {
        System.out.println("Could not listen on port 4321");
        System.exit(-1);
    }
}
```

Next, the `listenSocket` method creates a `Socket` connection for the requesting client. This code executes when a client starts up and requests the connection on the host and port where this server program is running. When the connection is successfully established, the `server.accept` method returns a new `Socket` object.

```
try{
    client = server.accept();
} catch (IOException e) {
    System.out.println("Accept failed: 4321");
    System.exit(-1);
}
```

Then, the `listenSocket` method creates a `BufferedReader` object to read the data sent over the socket connection from the client program. It also creates a `PrintWriter` object to send the data received from the client back to the server.

```
try{
    in = new BufferedReader(new InputStreamReader(
        client.getInputStream()));
    out = new PrintWriter(client.getOutputStream(),
```

```

        true);
    } catch (IOException e) {
        System.out.println("Read failed");
        System.exit(-1);
    }
}

```

Lastly, the `listenSocket` method loops on the input stream to read data as it comes in from the client and writes to the output stream to send the data back.

```

    while(true){
        try{
            line = in.readLine();
//Send data back to client
            out.println(line);
        } catch (IOException e) {
            System.out.println("Read failed");
            System.exit(-1);
        }
    }
}

```

actionPerformed Method

The `actionPerformed` method is called by the Java platform for action events such as button clicks. This `actionPerformed` method uses the text stored in the `line` object to initialize the `textArea` object so the retrieved text can be displayed to the end user.

```

public void actionPerformed(ActionEvent event) {
    Object source = event.getSource();

    if(source == button){
        textArea.setText(line);
    }
}

```

Example 1: Client-Side Program

The [client program](#) establishes a connection to the server program on a particular host and port number in its `listenSocket` method, and sends the data entered by the end user to the server program in its `actionPerformed` method. The `actionPerformed` method also receives the data back from the server and prints it to the command line.

listenSocket Method

The `listenSocket` method first creates a `Socket` object with the computer name (`kq6py`) and port number (`4321`) where the server program is listening for client connection requests. Next, it creates a `PrintWriter` object to send data over the socket connection to the server program. It also creates a `BufferedReader` object to read the text sent by the server back to the client.

```

public void listenSocket(){

```

```
//Create socket connection
try{
    socket = new Socket("kq6py", 4321);
    out = new PrintWriter(socket.getOutputStream(),
        true);
    in = new BufferedReader(new InputStreamReader(
        socket.getInputStream()));
} catch (UnknownHostException e) {
    System.out.println("Unknown host: kq6py");
    System.exit(1);
} catch (IOException e) {
    System.out.println("No I/O");
    System.exit(1);
}
}
```

actionPerformed Method

The actionPerformed method is called by the Java platform for action events such as button clicks. This actionPerformed method code gets the text in the Textfield object and passes it to the PrintWriter object, which then sends it over the socket connection to the server program.

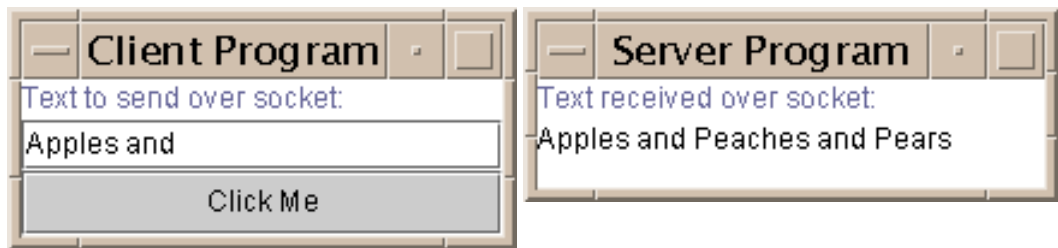
The actionPerformed method then makes the Textfield object blank so it is ready for more end user input. Lastly, it receives the text sent back to it by the server and prints the text out.

```
public void actionPerformed(ActionEvent event){
    Object source = event.getSource();

    if(source == button){
//Send data over socket
        String text = textField.getText();
        out.println(text);
        textField.setText(new String(""));
        out.println(text);
    }
//Receive text from server
    try{
        String line = in.readLine();
        System.out.println("Text received: " + line);
    } catch (IOException e){
        System.out.println("Read failed");
        System.exit(1);
    }
}
```

Example 2: Multithreaded Server Example

The example in its current state works between the server program and one client program only. To allow multiple client connections, the server program has to be converted to a [multithreaded server program](#).



First Client



Second Client



Third Client

The multithreaded server program creates a new thread for every client request. This way each client has its own connection to the server for passing data back and forth. When running multiple threads, you have to be sure that one thread cannot interfere with the data in another thread.

In this example the `listenSocket` method loops on the `server.accept` call waiting for client connections and creates an instance of the `ClientWorker` class for each client connection it accepts. The `textArea` component that displays the text received from the client connection is passed to the `ClientWorker` instance with the accepted client connection.

```
public void listenSocket(){
    try{
        server = new ServerSocket(4444);
    } catch (IOException e) {
        System.out.println("Could not listen on port 4444");
        System.exit(-1);
    }
    while(true){
        ClientWorker w;
        try{
            //server.accept returns a client connection
            w = new ClientWorker(server.accept(), textArea);
            Thread t = new Thread(w);
            t.start();
        } catch (IOException e) {
            System.out.println("Accept failed: 4444");
            System.exit(-1);
        }
    }
}
```

```

    }
  }
}

```

The important changes in this version of the server program over the non-threaded server program are the `line` and `client` variables are no longer instance variables of the server class, but are handled inside the `ClientWorker` class.

The `ClientWorker` class implements the `Runnable` interface, which has one method, `run`. The `run` method executes independently in each thread. If three clients request connections, three `ClientWorker` instances are created, a thread is started for each `ClientWorker` instance, and the `run` method executes for each thread.

In this example, the `run` method creates the input buffer and output writer, loops on the input stream waiting for input from the client, sends the data it receives back to the client, and sets the text in the text area.

```

class ClientWorker implements Runnable {
    private Socket client;
    private JTextArea textArea;

    //Constructor
    ClientWorker(Socket client, JTextArea textArea) {
        this.client = client;
        this.textArea = textArea;
    }

    public void run(){
        String line;
        BufferedReader in = null;
        PrintWriter out = null;
        try{
            in = new BufferedReader(new
                InputStreamReader(client.getInputStream()));
            out = new
                PrintWriter(client.getOutputStream(), true);
        } catch (IOException e) {
            System.out.println("in or out failed");
            System.exit(-1);
        }

        while(true){
            try{
                line = in.readLine();
                //Send data back to client
                out.println(line);
                //Append data to text area
                textArea.append(line);
            } catch (IOException e) {
                System.out.println("Read failed");
                System.exit(-1);
            }
        }
    }
}

```

```

    }
  }
}

```

The `JTextArea.append` method is thread safe, which means its implementation includes code that allows one thread to finish its append operation before another thread can start an append operation. This prevents one thread from overwriting all or part of a string of appended text and corrupting the output. If the `JTextArea.append` method were not thread safe, you would need to wrap the call to `textArea.append(line)` in a synchronized method and replace the run method call to `textArea.append(line)` with a call to `appendText(line)`:

```

public synchronized void appendText(line){
    textArea.append(line);
}

```

The `synchronized` keyword means this thread has a lock on the `textArea` and no other thread can change the `textArea` until this thread finishes its append operation.

The `finalize()` method is called by the Java virtual machine (JVM)* before the program exits to give the program a chance to clean up and release resources. Multi-threaded programs should close all Files and Sockets they use before exiting so they do not face resource starvation. The call to `server.close()` in the `finalize()` method closes the Socket connection used by each thread in this program.

```

protected void finalize(){
//Objects created in run method are finalized when
//program terminates and thread exits
    try{
        server.close();
    } catch (IOException e) {
        System.out.println("Could not close socket");
        System.exit(-1);
    }
}
}

```

More Information

You can find more information on sockets in the [All About Sockets](#) section in [The Java Tutorial](#).

[\[TOP\]](#)

Print Button

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)
[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:

(800) 786-7638

Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 2: User Interfaces Revisited

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)

In [Java™ Programming Language Basics, Part 1](#), you learned how to use Java Foundation Classes (JFC) Project Swing (Project Swing) components to build a simple user interface with very basic backend functionality. You also learned how to use the Remote Method Invocation (RMI) application programming interface (API) to send data from a client program to a server program on the net where the data can be accessed by other client programs.

This lesson takes the RMI application from [Part 1, Lesson 8: Remote Method Invocation](#), creates a more involved user interface, and uses a different layout manager. These changes give you the beginnings of a very simple electronic-commerce application that consists of two types of client programs: one lets end users place purchase orders and the other lets order processors view the orders.

- [About the Example](#)
- [Fruit Order Client Code](#)
 - [Instance Variables](#)
 - [Constructor](#)
 - [Event Handling](#)
 - [Cursor Focus](#)
 - [Converting Strings to Numbers and Back](#)
- [Server Program Code](#)
- [View Order Client Code](#)
- [Program Improvements](#)
- [More Information](#)

About the Example

This is a very simple electronic commerce example for instructional purposes only. It consists of three programs: two client programs, one for ordering fruit and another for viewing the order, and one server program that makes order information available to clients that view the orders.

Fruit Order Client

The [FruitClient](#) program presents a user interface and prompts the end user to order apples, peaches, and pears at \$1.25 each.

Select Items	Specify Quantity
Apples	2
Peaches	1
Pears	4
Total Items:	7
Total Cost:	8.75
Credit Card:	1234-4321-1234-3
Customer ID:	munchkin

Reset Purchase

After the end user enters the number of each item to order, he or she presses the Return key to commit the order and update the running total.

The Tab key or mouse moves the cursor to the next field. At the bottom, the end user provides a credit card number and customer ID.

When the end user clicks Purchase, all values entered into the form are sent to the server program.

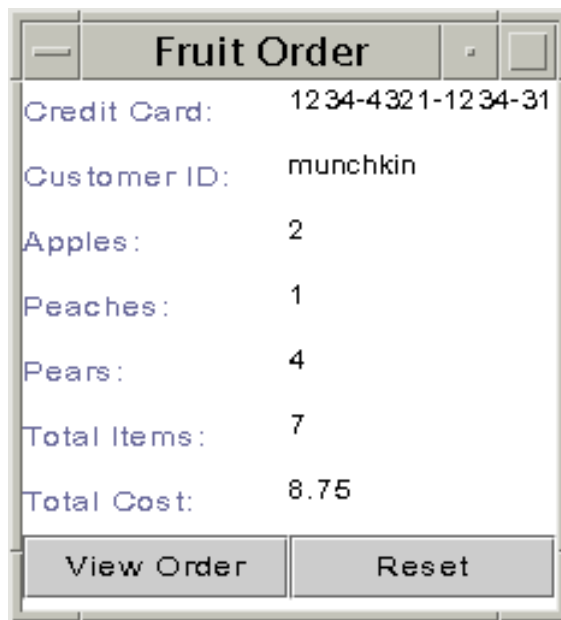
The end user must press the Return key for the total to update. If the Return key is not pressed, an incorrect total is sent across the net with the order. The end of this lesson asks you to change the code so there is no danger incorrect totals are sent across the net because the end user did not press the Return key.

Server Program

The [RemoteServer](#) program provides remotely accessible send and get methods. Fruit order clients call send methods to send data to the server, and view order clients call the get methods to retrieve the data. In this example, the server program has no user interface.

View Order Client

The [OrderClient](#) program presents a user interface, and when the end user clicks View Order, the program gets the order data from the server program and puts it on the screen.



Compile and Run the Example

See [Part 1, Lesson 8: Remote Method Invocation](#), for information on how to run the example. Use the Part 1, Lesson 8 instructions, but use the source code provided in this lesson. Here is a summarized version of those steps:

Compile: These instructions assume development is in the zelda home directory.

Unix:

```
cd /home/zelda/classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
cp RemoteServer*.class /home/zelda/public_html/classes
cp Send.class /home/zelda/public_html/classes
```

Win32:

```
cd \home\zelda\classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
copy RemoteServer*.class \home\zelda\public_html\classes
copy Send.class \home\zelda\public_html\classes
```

Start rmi Registry:

Unix:

```
cd /home/zelda/public_html/classes
unsetenv CLASSPATH
rmiregistry &
```

Win32:

```
cd \home\zelda\public_html\classes
set CLASSPATH=
start rmiregistry
```

Start Remote Server:**Unix:**

```
cd /home/zelda/public_html/classes
java
-Djava.rmi.server.codebase=http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RemoteServer
```

Win32:

```
cd \home\zelda\public_html\classes
java -Djava.rmi.server.codebase=
    file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RemoteServer
```

Start RMI Client 1:**Unix:**

```
cd /home/zelda/classes

java -Djava.rmi.server.codebase=
    http://kq6py/~zelda/classes/
-Djava.security.policy=java.policy RMIClient1
    kq6py.eng.sun.com/~zelda
```

Win32:

```
cd \home\zeldzeldaa\classes

java -Djava.rmi.server.codebase=
    file:c:\home\zelda\classes\
-Djava.security.policy=java.policy RMIClient1
    kq6py.eng.sun.com\home\zelda\public\html
```

Start RMI Client 2:**Unix:**

```
cd /home/zelda/classes
java -Djava.rmi.server.codebase=
    http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RMIClient2
    kq6py.eng.sun.com
```

Win32:

```
cd \home\zelda\classes
java -Djava.rmi.server.codebase=
    file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
```

```
-Djava.security.policy=java.policy RMIClient2
kq6py.eng.sun.com
```

Fruit Order Client Code

The [RMIClient1.java](#) code uses label, text field, text area, and button components to create the user interface for ordering fruit.

Select Items	Specify Quantity
Apples	2
Peaches	1
Pears	4
Total Items:	7
Total Cost:	8.75
Credit Card:	1234-4321-1234-3
Customer ID:	munchkin
<input type="button" value="Reset"/> <input type="button" value="Purchase"/>	

On the display, user interface components are arranged in a 2-column grid with labels in the left column, and the input and output data fields (text fields and text areas) aligned in the right column.

The end user enters his or her apples, peaches, and pears order into the text fields and presses the Return key after each fruit entry. When the Return key is pressed, the text field behavior updates the item and cost totals displayed in the text areas.

The Reset button behavior clears the display, and the underlying total cost and items variables. The Purchase button behavior sends the order data to the server program. If the Reset button is clicked before the Purchase button, null values are sent over the network.

Instance Variables

These next lines declare the Project Swing component classes the SwingUI class uses. These are instance variables that can be accessed by any method in the instantiated class. In this example, they are built in the SwingUI constructor and accessed in the actionPerformed method implementation.

```
JLabel col1, col2;
JLabel totalItems, totalCost;
JLabel cardNum, custID;
JLabel applechk, peachchk, peachchk;

JButton purchase, reset;
JPanel panel;

JTextField applegnt, pearqnt, peachqnt;
JTextField creditCard, customer;
JTextArea items, cost;

static Send send;
int itotal=0;
double icost=0;
```

Constructor

The constructor is fairly long because it creates all the components, sets the layout to a 2-column grid, and places the components in the grid on a panel. A panel is a container component that holds other components.

The Reset and Purchase buttons and the appleQnt, pearQnt, and peachQnt text fields are added as action listeners. This means when the end user clicks one of the buttons or presses Return in one of the text fields, an action event occurs that causes the platform to call the FruitClient.actionPerformed method where the behaviors for these components are defined.

As explained in [Part 1, Lesson 4: Building a User Interface](#), a class declares the ActionListener interface and implements the actionPerformed method if it needs to handle action events such as button clicks and text field Returns. Other user interface components generate some different action events, and as a result, require you to implement different interfaces and methods.

```
//Create left and right column labels
col1 = new JLabel("Select Items");
col2 = new JLabel("Specify Quantity");

//Create labels and text field components
applechk = new JLabel("  Apples");
appleqnt = new JTextField();
appleqnt.addActionListener(this);

pearchk = new JLabel("  Pears");
pearqnt = new JTextField();
pearqnt.addActionListener(this);

peachchk = new JLabel("  Peaches");
peachqnt = new JTextField();
peachqnt.addActionListener(this);

cardNum = new JLabel("  Credit Card:");
creditCard = new JTextField();

customer = new JTextField();
custID = new JLabel("  Customer ID:");

//Create labels and text area components
totalItems = new JLabel("Total Items:");
totalCost = new JLabel("Total Cost:");
items = new JTextArea();
cost = new JTextArea();

//Create buttons and make action listeners
purchase = new JButton("Purchase");
purchase.addActionListener(this);
```

```

reset = new JButton("Reset");
reset.addActionListener(this);

```

In the next lines, a JPanel component is created and added to the top-level frame, and the layout manager and background color for the panel are specified. The layout manager determines how user interface components are arranged on the panel.

The example in [Part 1, Lesson 4: Building a User Interface](#), used the BorderLayout layout manager. This example uses the GridLayout layout manager, which arranges components in a grid or the number of rows and columns you specify. The example uses a 2-column grid with an unlimited number of rows as indicated by the zero (unlimited rows) and two (two columns) in the statement `panel.setLayout(new GridLayout(0,2));`.

The layout manager and color are set on the panel, and the panel is added to the content pane with a call to the `getContentPane` method of the JFrame class. A content pane lets different types of components work together in Project Swing.

```

//Create a panel for the components
panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
panel.setLayout(new GridLayout(0,2));
panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
getContentPane().add(panel);
panel.add(col1);
panel.add(col2);

panel.add(applechk);
panel.add(appleqnt);

panel.add(peachchk);
panel.add(peachqnt);

panel.add(pearchk);
panel.add(pearqnt);

panel.add(totalItems);
panel.add(items);

panel.add(totalCost);
panel.add(cost);

panel.add(cardNum);
panel.add(creditCard);

panel.add(custID);
panel.add(customer);

```

```
panel.add(reset);
panel.add(purchase);
```

Event Handling

The `actionPerformed` method provides behavior for each of the following possible application events:

- The mouse is clicked on the Purchase or Reset button.
- The Return key is pressed inside the `appleQnt`, `peachQnt`, or `pearQnt` field.

Rather than show the entire `actionPerformed` method here, this section describes the purchase button and `pearQnt` text field behaviors only. The Reset button is similar to the purchase button, and the other text fields are similar to `pearQnt`.

Purchase Button: The Purchase button behavior involves retrieving data from the text fields and text areas, and sending that data to the server program. The server program is available to the `FruitClient` program through its `Send` interface, which declares the remote server methods for sending and getting data.

The `send` variable is an instance of the `Send` interface. This instance is created in the `FruitClient` program's main method. The `send` variable is declared static and global in the `FruitClient` program so the static main method can instantiate it, and to make it accessible to the `actionPerformed` method.

```
if(source == purchase){
    cardnum = creditCard.getText();
    custID = customer.getText();
    apples = appleqnt.getText();
    peaches = peachqnt.getText();
    pears = pearqnt.getText();
    try{
        send.sendCreditCard(cardnum);
        send.sendCustID(custID);
        send.sendAppleQnt(apples);
        send.sendPeachQnt(peaches);
        send.sendPearQnt(pears);
        send.sendTotalCost(icost);
        send.sendTotalItems(itotal);
    } catch (Exception e) {
        System.out.println("Cannot send data to server");
    }
}
```

pearQnt Text Field: The `pearQnt` text field behavior involves retrieving the number of pears the end user wants to order, adding the number to the items total, using the number to calculate the cost, and adding the cost for pears to the total cost. Two interesting things in this code involve managing the cursor focus and converting strings to numbers for the calculations. Both topics are covered below.

```

if(source == pearqnt){
    number = pearqnt.getText();
    if(number.length() > 0){
        pearsNo = Integer.valueOf(number);
        itotal += pearsNo.intValue();
        pearqnt.setNextFocusableComponent(creditCard);
    } else {
        itotal += 0;
        pearqnt.setNextFocusableComponent(creditCard);
    }
}
}

```

Cursor Focus

End users can use the Tab key to move the cursor from one component to another within the user interface. The default Tab key movement steps through all user interface components including the text areas.

Because the end user does not interact with the text areas, there is no reason for the cursor to go there. The example program includes a call in its constructor to `pearqnt.setNextFocusableComponent` to make the cursor move from the `pearqnt` text field to the `creditcard` text field bypassing the total cost and total items text areas when the Tab key is pressed.

```

applechk = new JLabel(" Apples");
appleqnt = new JTextField();
appleqnt.addActionListener(this);

pearchk = new JLabel(" Pears");
pearqnt = new JTextField();
pearqnt.addActionListener(this);

peachchk = new JLabel(" Peaches");
peachqnt = new JTextField();
peachqnt.addActionListener(this);

cardNum = new JLabel(" Credit Card:");
creditCard = new JTextField();
//Make cursor go to creditCard component
pearqnt.setNextFocusableComponent(creditCard);

customer = new JTextField();
custID = new JLabel(" Customer ID:");

```

Converting Strings to Numbers and Back

To calculate the items ordered and their cost, the string values retrieved from the `appleQnt`, `peachQnt`, and `pearQnt` text fields have to be converted to their number equivalents.

The string value is returned in the number variable. To be sure the user actually entered a value, the string length is checked. If the length is not greater than zero, the end user pressed Return

without entering a value. In this case, the else statement adds zero to the running total and the cursor focus is set for the creditCard text field. Adding zero is not really necessary, but does make the code more understandable for someone reading it.

If the length is greater than zero, an instance of the java.lang.Integer class is created from the string. Next, the Integer.intValue() method is called to produce the integer (int) equivalent of the string value so it can be added to the items total kept in the itotal integer variable.

```
if(number.length() > 0){
    pearsNo = Integer.valueOf(number);
    itotal += pearsNo.intValue();
} else {
    itotal += 0;
}
```

To display the running item and cost totals in their respective text areas, the totals have to be converted back to strings. The code at the end of the actionPerformed method shown below does this.

To display the total items, a java.lang.Integer object is created from the itotal integer variable. The Integer.toString method is called to produce the String equivalent of the integer (int). This string is passed to the call to this.cost.setText(text2) to update the Total Cost field in the display.

Note: The cost text area variable is referenced as this.cost because the actionPerformed method also has a cost variable of type Double. To reference the global text area and not the local Double by the same name, you have to reference it as this.cost.

```
num = new Integer(itotal);
text = num.toString();
this.items.setText(text);

icost = (itotal * 1.25);
cost = new Double(icost);
text2 = cost.toString();
this.cost.setText(text2);
```

Until now, all data types used in the examples have been classes. But, the int and double data types are not classes. They are primitive data types.

The int data type contains a single whole 32-bit integer value that can be positive or negative. You can use the standard arithmetic operators (+, -, *, and /) to perform arithmetic operations on the integer.

The Integer class, not only contains a whole 32-bit integer value that can be positive or negative, but provides methods for working on the value. For example, the Integer.intValue method lets you convert an Integer to an int to perform arithmetic operations.

The double data type contains a 64-bit double-precision floating

point value. The Double class not only contains a 64-bit double-precision floating point value, but provides methods for working on the value. For example, the Double.doubleValue method lets you convert a Double to a double to perform arithmetic operations.

Server Program Code

The server program consists of the [RemoteServer.java](#) class that implements the methods declared in the [Send.java](#) interface. These classes are described in [Part 1, Lesson 8: Remote Method Invocation](#) with the only difference being in this lesson there are many more sendXXX and getXXX methods to declare and implement. Here is the list:

- `public void sendCreditCard(String creditcard){ cardnum = creditcard; }`
- `public String getCreditCard(){ return cardnum; }`
- `public void sendCustID(String cust){ custID = cust; }`
- `public String getCustID(){ return custID; }`
- `public void sendAppleQnt(String apps){ apples = apps; }`
- `public String getAppleQnt(){ return apples; }`
- `public void sendPeachQnt(String pchs){ peaches = pchs; }`
- `public String getPeachQnt(){ return peaches; }`
- `public void sendPearQnt(String prs){ pears = prs; }`
- `public String getPearQnt(){ return pears; }`
- `public void sendTotalCost(double cst){ cost = cst; }`
- `public double getTotalCost(){ return cost; }`
- `public void sendTotalItems(int itm){ items = itm; }`
- `public int getTotalItems(){ return items; }`

The important thing to note is data of any type and size can be easily passed from one client through the server to another client using the RMI API. No special handling is needed for large amounts of data or special considerations for different data types, which can sometimes be issues when using socket communications.

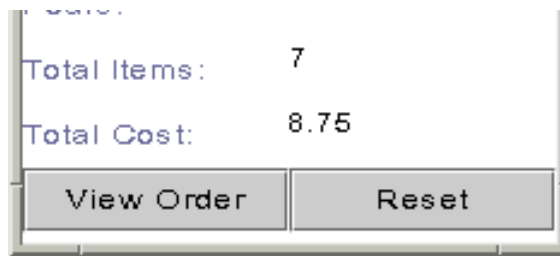
View Order Client Code

The [OrderClient.java](#) class uses text areas and buttons to display the order information.



The code is very similar to the FruitOrder.java class so rather than repeat much of what you have read above, this section highlights two parts of the actionPerformed method behavior for viewing an order.

The first part retrieves the credit card number, and the



number of apples, peaches, and pears ordered from the server and sets those values in the corresponding text areas.

The second part retrieves the cost and item totals, which are double and integer, respectively.

It then converts the total cost to a `java.lang.Double` object, and the total items to a `java.lang.Integer` object, and calls the `toString` method on each to get the string equivalents. Finally, the strings can be used to set the values for the corresponding text areas.

```

if(source == view){
    try{
//Retrieve and display text
        text = send.getCreditCard();
        creditNo.setText(text);

        text = send.getCustID();
        customerNo.setText(text);

        text = send.getAppleQnt();
        applesNo.setText(text);

        text = send.getPeachQnt();
        peachesNo.setText(text);

        text = send.getPearQnt();
        pearsNo.setText(text);

//Convert Numbers to Strings
        cost = send.getTotalCost();
        price = new Double(cost);
        unit = price.toString();
        icost.setText(unit);

        items = send.getTotalItems();
        itms = new Integer(items);
        i = itms.toString();
        itotal.setText(i);

    } catch (Exception e) {
        System.out.println("Cannot send data to server");
    }
}

```

Program Improvements

The example program as it is currently written has two major design flaws in the fruit order client. The first one involves the need to press the Return key for calculations to happen, and the second involves handling the error condition if the end user enters a character that is not a number when ordering apples, peaches, and

pears.

Calculations and Pressing Return: If the end user enters a value for apples, peaches, or pears and moves to the next field without pressing the Return key, no calculation is made. This means when the end user clicks the Purchase key, the order is sent, but the item and cost totals will be incorrect. So, in this particular application relying on the Return key action event is not good design.

Modify the `actionPerformed` method so this does not happen. Here is one possible [solution](#). Give it a try before taking a look.

Non-Number Errors: If the end user enters a non-number value for apples, peaches, or pears the program will present a stack trace indicating an illegal number format. A good program will catch and handle the error, rather than produce a stack trace.

Hint: You need to figure out which part of the code throws the error and enclose it in a try and catch block. try and catch blocks were first introduced in [Part 1, Lesson 6: File Access and Permissions](#).

The error you need to catch is `java.lang.NumberFormatException`.

Give it a try before taking a look at the [solution](#).

More Information

You can find more information on event listening in the [Writing Event Listeners](#) lesson in [The Java Tutorial](#).

The [Variables and Data Types](#) trail in [The Java Tutorial](#) provides more information on primitive data types.

See [The JFC Swing Tutorial: A Guide to Constructing GUIs](#) for more information on Project Swing.

* As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[\[TOP\]](#)

Print Button

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638

Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
 All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 3: Cryptography

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)]

Many people are protective of their credit card numbers, and for good reason. A stolen credit card number with other personal information can give a thief all he or she needs to create serious mayhem in someone's life. One way to keep credit card and other proprietary information secure when sending it over the net is to encrypt it.

Encryption is the process of applying a key to plain text that transforms that plain text into unintelligible (cipher) text. Only programs with the key to turn the cipher text back to original text can decrypt the protected information.

This lesson adapts the [Part 2, Lesson 2: User Interfaces Revisited](#) example to encrypt the credit card number before sending it over the net, and decrypt it on the other side.

Note: Because cryptography software is not exportable outside the United States and Canada, the example in this lesson is in pseudo code rather than source code.

- [About the Example](#)
- [Running the Example](#)
- [Pseudo Code](#)
 - [Server](#)
 - [Generating the Public and Private Key](#)
 - [Sealing the Symmetric Key](#)
 - [Encrypting the Symmetric Key with the RSA Algorithm](#)
- [More Information](#)

About the Example

To safely send the credit card number over the net, the example program gets the plain text credit card number entered by the end user and passes the credit card number to its encrypt method.



The encrypt method creates a cipher and session key, and uses the session key with the cipher to encrypt the credit card

Apples	2
Peaches	1
Pears	4
Total Items:	7
Total Cost:	8.75
Credit Card:	1234-4321-1234-3
Customer ID:	munchkin
Reset	Purchase

number.

A session key is a secret key that is generated new each time the Purchase button is clicked. Changing the session key protects against an unauthorized program getting the key and decrypting hundreds and thousands of credit card numbers with it.

The credit card number is encrypted and decrypted with the same session key. This type of cryptography is called

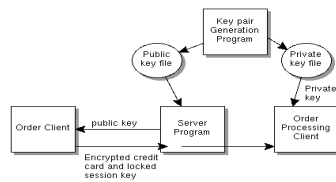
symmetric key encryption, and in our example, requires the session key and encrypted credit card number be sent over the net to the receiving program. Because the session key is sent over the net, it too should be protected against unauthorized access.

To protect the session key, it is encrypted with or wrapped under the public key of the recipient. Even if an unauthorized program gets the wrapped session key and credit card number, he or she would have to recover the session key with the intended recipient's private key to be able to decrypt the credit card number with the session key.

Anything encrypted with a public key, can only be decrypted with the private key corresponding to the public key that originally encrypted it. This type of cryptography is called asymmetric key encryption. In the example, the public key is made readily available to any client program that requests it, and the private key is kept secret and made available to specific, trusted clients only.

As shown in the diagram, this example uses a separate program to generate the public and private key pair. The public key is stored in one file, and the private key is stored in another. The file with the private key must be kept in a very secure place. Many companies keep the private key file on an external storage medium such as tape or disk to prevent an unauthorized person or program from breaking into the system and getting the private key.

The server program loads the public key from the public key file, and makes it available to order clients for encrypting the session key. Order processing clients get the encrypted session key and credit card number, load the private key, use the private key to decrypt the session key, and use the session key to decrypt the credit card number.



Running the Example

If you are within the United States or Canada, you can download the `javax.crypto` package from the [Products & APIs](#) page. It contains documentation and a Java™ Archive (JAR) file with the cryptographic APIs and a cryptographic service provider. A cryptographic service provider is a package or set of packages that supplies a concrete implementation of a cryptographic algorithm.

Copy the JAR file to the `jdk1.2/jre/lib/ext` directory of your Java 2 SDK, Standard Edition, installation or to the `jre1.2/lib/security` directory of your Java Runtime Environment (JRE) 1.2 installation.

Make sure you have the following entries in the `jdk1.2/jre/lib/security/java.security` or `jre1.2/lib/security/java.security` file:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.crypto.provider.SunJCE
```

You also need to install a package with an asymmetric algorithm such as the Rivest, Shamir, and Adleman (RSA) Asymmetric-Cipher algorithm.

The asymmetric algorithm is needed to create the asymmetric cipher for the public and private key encryption. Add the asymmetric algorithm package to `jdk1.2/jre/lib/security/java.security` or `jre1.2/lib/security/java.security` as `security.provider.3=` and put it in the `jdk1.2/jre/lib/ext` or `jre1.2/lib/ext` directory with the other JAR files.

Using the documentation in the download, convert the pseudo code to source code.

Compile and run the example as usual.

Pseudo Code

A cipher object is used in the encryption and decryption process. The cipher object is created with a specific cryptographic algorithm

depending on the type of encryption in use. In this example, two types of encryption are used: symmetric and asymmetric.

Symmetric key encryption uses a symmetric algorithm such as Data Encryption Standard (DES). The asymmetric key encryption uses an asymmetric algorithm such as Rives, Shamir, and Adleman (RSA) Asymmetric-Cipher algorithm.

The `javax.crypto` package defines the framework for both symmetric and asymmetric encryption into which concrete cipher implementations can be plugged. The SunJCE provider that comes standard with JCE 1.2 supplies only implementations of symmetric encryption algorithms such as DES. For an implementation of an asymmetric encryption algorithm such as RSA, you need to install a different provider.

The pseudo code shows two ways to do the asymmetric encryption of the session key. One way uses an RSA key to encrypt the symmetric key. The other way uses another asymmetric algorithm to seal (encrypt) the symmetric key. Sealing is the preferred way, but presents a problem when you use the RSA key because the RSA algorithm imposes a size restriction (discussed below) on the object being encrypted and sealing makes the object too large for RSA encryption.

After the cipher is created with the correct symmetric or asymmetric algorithm, it is initialized for encryption or decryption with a key. In the case of symmetric encryption, the key is a secret key, and in the case of asymmetric encryption, the key is either the public or private key.

Server

The `Send` interface declares and the `RemoteServer` class implements methods to handle the encrypted credit card number and the encrypted secret key. It also defines a method to return the public key when a client requests it. In pseudo code, this is what the server interface and class need to declare and implement:

```
A method to get the public key
A method to send the encryped credit card number
A method to get the encrypted credit card number
A method to send the encrypted symmetric key
A method to get the encrypted symmetric key
```

Generating the Public and Private Key Pair

You need a program to generate a public and private key pair and store them to separate files. The public key is read from its file when a client calls the method to get the public key. The private key is read from its file when `RMIClient2` needs it to decrypt the secret key.

```
Generate public and private key pair
    using asymmetric algorithm
Store private Key in very safe place
```

Store public key in accessible place

Sealing the Symmetric Key

Sealing the symmetric key involves creating a sealed object that uses an asymmetric cipher to seal (encrypt) the session key. The RSA asymmetric algorithm cannot be used because it has the size restrictions described in the next section, and the sealing process makes the session key too large to use with the RSA algorithm.

RMIClient1Sealed.java: The [RMIClient1.java](#) code has an encrypt method to encrypt the credit card number, seal the symmetric key, and send the encrypted credit card number and sealed key to the server. Here is the pseudo code to do it:

```
private void encrypt(credit card number){
    Create cipher for symmetric key encryption (DES)
    Create a key generator
    Create a secret (session) key with key generator
    Initialize cipher for encryption with session key
    Encrypt credit card number with cipher
    Get public key from server
    Create cipher for asymmetric encryption
                                (do not use RSA)
    Initialize cipher for encryption with public key
    Seal session key using asymmetric Cipher
    Send encrypted credit card number and sealed
    session key to server
}
```

RMIClient2Sealed.java: The [RMIClient2.java](#) code has a decrypt method to unseal the symmetric key and decrypt the credit card number. Here is the pseudo code to do it:

```
public byte[] decrypt(encrypted key,
                    encrypted credit card number){
    Get private key from file
    Create asymmetric cipher (do not use RSA)
    Initialize cipher for decryption with private key
    Unseal wrapped session key using asymmetric cipher
    Create symmetric cipher
    Initialize cipher for decryption with session key
    Decrypt credit card number with symmetric cipher
}
```

Encrypting the Symmetric Key with the RSA Algorithm

The RSA algorithm imposes size restrictions on the object being encrypted. RSA encryption uses the PKCS#1 standard with PKCS#1 block type 2 padding. The PKCS RSA encryption padding scheme needs 11 spare bytes to work. So, if you generate an RSA key pair with a key size of 512 bits, you cannot use the keys to encrypt more than 53 bytes ($53 = 64 - 11$).

So, if you have a session key that is only 8 bytes long, sealing

expands it to 3644 bytes, which is way over the size restriction imposed by the RSA algorithm. In the process of sealing, the object to be sealed (the session key, in this case) is first serialized, and then the serialized contents are encrypted. Serialization adds more information to the session key such as the class of the session key, the class signature, and any objects referenced by the session key. The additional information makes the session key too large to be encrypted with an RSA key, and the result is a `javax.crypto.IllegalBlockSizeException` run time error.

`RMIClient1.java`: The [RMIClient1.java](#) code has an `encrypt` method to encrypt the credit card number, seal (encrypt) the session key, and send the encrypted credit card number and sealed session key to the server. Here is the pseudo code to do it:

```
private void encrypt(credit card number){
    Create cipher for symmetric key encryption (DES)
    Create a key generator
    Create a secret (session) key with key generator
    Initialize cipher for encryption with session key
    Encrypt credit card number with cipher
    Get public key from server
    Create cipher for asymmetric encryption (RSA)
    Initialize cipher for encryption with public key
    Encrypt session key
    Send encrypted credit card number and session
                                     key to server
}
```

`RMIClient2.java`: The [RMIClient2.java](#) code has a `decrypt` method to unseal (decrypt) the symmetric key and decrypt the credit card number. Here is the pseudo code to do it:

```
public String decrypt(encrypted key,
                    encrypted credit card number){
    Decrypt credit card number
    Get private key from file
    Create asymmetric cipher (RSA)
    Initialize cipher for decryption with private key
    Decrypt symmetric key
    Instantiate symmetric key
    Create symmetric cipher
    Initialize Cipher for decryption with session key
    Decrypt credit card number with symmetric Cipher
}
```

More Information

You can find more information on key encryption on the [Security Dynamics](#) Web site (for RSA encryption), or by using a search engine and searching on RSA Cryptography, asymmetric key encryption, or symmetric key encryption.

[\[TOP\]](#)

[Print Button](#)

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 4: Serialization

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)

One big problem with the example program in its current form is the fact that sending clients can overwrite each other's data before receiving clients have a chance to get and process it. This lesson adapts the server code to ensure all orders are processed (nothing is overwritten), and all orders are processed in the order they are received by the server.

- [About the Example](#)
- [Wrapping the Data](#)
- [Sending Data](#)
- [Server Program](#)
- [Receiving Data](#)
- [More Information](#)

About the Example

The example adapts the [Part 2, Lesson 2: User Interfaces Revisited](#) example to wrap the fruit order data into a single data object and send the data object over the network to the server. This is more efficient than sending each unit of data separately.

Wrapping the Data

The [DataOrder.java](#) class is very simple. It defines the fields that wrap and store the fruit order data. It has no methods. It implements the `Serializable` interface so its data can be serialized, and written to and read from a file as a single unit.

Object serialization transforms an object's data to a bytestream that represents the state of the data. The serialized form of the data contains enough information to recreate the object with its data in a similar state to what it was when saved.

```
import java.io.*;

class DataOrder implements Serializable{
    String apples, peaches, pears, cardnum, custID;
    double icost;
    int itotal;
```

}

Sending Data

The [RMIClient1.java](#) program is modified to use the `DataOrder` class to send the order data over the net. The `RMIClient1.actionPerformed` method creates an instance of the `DataOrder` class and initializes its fields with order data retrieved from the user interface text fields and areas.

```
public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String number, text, text2;
    DataOrder order = new DataOrder();

    if(source == purchase){
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();
```

The total number of items is calculated using the `order.icost` field.

```
if(order.apples.length() > 0){
    try{
        applesNo = Integer.valueOf(order.apples);
        order.itotal += applesNo.intValue();
    } catch (java.lang.NumberFormatException e) {
        appleqnt.setText("Invalid Value");
    }
} else {
    order.itotal += 0;
}
```

The total number of items is retrieved from the `order.itotal` field and displayed in the user interface.

```
num = new Integer(order.itotal);
text = num.toString();
this.items.setText(text);
```

Similarly, the total cost is calculated and displayed in the user interface using the `order.icost` field.

```
order.icost = (order.itotal * 1.25);
cost = new Double(order.icost);
text2 = cost.toString();
this.cost.setText(text2);

try{
    send.sendOrder(order);
} catch (Exception e) {
```

```

        System.out.println("Cannot send data to server");
    }

```

After the totals are calculated, the order object is sent over the net to the server program.

Server Program

The [Send.java](#) and [RemoteServer.java](#) classes are much simpler in this lesson. They have one `getXXX` method that returns an instance of `DataOrder`, and one `setXXX` method that accepts an instance of `DataOrder`.

Send.java

```

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Send extends Remote {
    public void sendOrder(DataOrder order)
        throws RemoteException;
    public DataOrder getOrder() throws RemoteException;
}

```

RemoteServer.java

The `RemoteServer.sendOrder` method accepts a `DataOrder` instance as input, and stores each order in a separate file where the file name is a number. The first order received is stored in a file named 1, the second order is stored in a file named 2, and so forth.

To keep track of the file names, the value variable is incremented by 1 each time the `sendOrder` method is called, converted to a `String`, and used for the file name in the serialization process.

Objects are serialized by creating a serialized output stream and writing the object to the output stream. In the code, the first line in the `try` block creates a `FileOutputStream` with the file name to which the serialized object is to be written.

The next line creates an `ObjectOutputStream` from the file output stream. This is the serialized output stream to which the order object is written in the last line of the `try` block.

RemoteServer.java

```

public void sendOrder(DataOrder order){

    value += 1;
    num = new Integer(value);
    orders = num.toString();
    try{
        FileOutputStream fos =
            new FileOutputStream(orders);
        ObjectOutputStream oos =
            new ObjectOutputStream(fos);
        oos.writeObject(order);
    }
}

```

```

    }catch (java.io.FileNotFoundException e){
        System.out.println(e.toString());
    }catch (java.io.IOException e){
        System.out.println(e.toString());
    }
}

```

The `RemoteServer.getOrder` method does what the `sendOrder` method does in reverse using the `get` variable to keep track of which orders have been viewed.

But first, this method checks the `value` variable. If it is equal to zero, there are no orders to get from a file and view, and if it is greater than the value in the `get` variable, there is at least one order to get from a file and view. As each order is viewed, the `get` variable is incremented by 1.

```

public DataOrder getOrder(){

    DataOrder order = null;

    if(value == 0){
        System.out.println("No Orders To Process");
    }

    if(value > get){
        get += 1;
        num = new Integer(get);
        orders = num.toString();
        try{
            FileInputStream fis =
                new FileInputStream(orders);
            ObjectInputStream ois =
                new ObjectInputStream(fis);
            order = (DataOrder)ois.readObject();
        }catch (java.io.FileNotFoundException e){
            System.out.println(e.toString());
        }catch (java.io.IOException e){
            System.out.println(e.toString());
        }catch (java.lang.ClassNotFoundException e){
            System.out.println(e.toString());
        }
    }else{
        System.out.println("No Orders To Process");
    }
    return order;
}

```

Receiving Data

The [RMIClient2.actionPerformed](#) method gets an order object and references its fields to display data in the user interface.

```

if(source == view){
    try{

```

```
        order = send.getOrder();
        creditNo.setText(order.cardnum);
        customerNo.setText(order.custID);
        applesNo.setText(order.apples);
        peachesNo.setText(order.peaches);
        pearsNo.setText(order.pears);

        cost = order.icost;
        price = new Double(cost);
        unit = price.toString();
        icost.setText(unit);

        items = order.itotal;
        itms = new Integer(items);
        i = itms.toString();
        itotal.setText(i);
    } catch (Exception e) {
        System.out.println("Cannot send data to server");
    }
}
```

More Information

You can find more information on serialization in the [Reading and Writing \(but no 'rithmetic\)](#) lesson in [The Java™ Tutorial](#).

[[TOP](#)]

[Print Button](#)

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

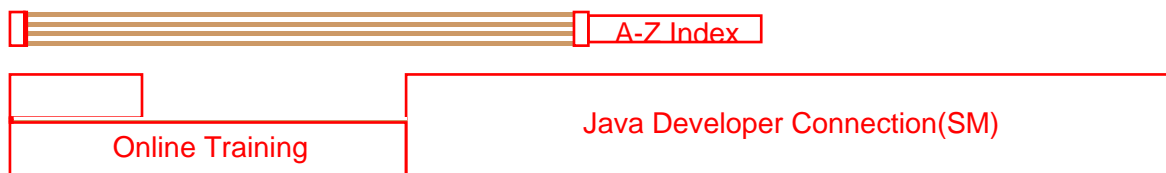
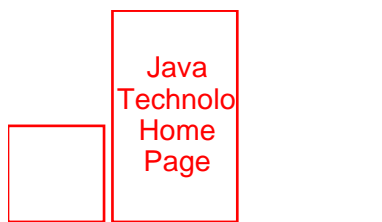
For more information on Java technology
and other software from Sun Microsystems, call:

(800) 786-7638

Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Downloads, APIs,

Java Developer

Tutorials, Tech Articles,

Online Support

Community Discussion

News & Events from

Products from

How Java Technology

Print Button

Java™ Programming Language Basics, Part 2

Lesson 5: Collections

[<<BACK] [CONTENTS] [NEXT>>]

A collection is an object that contains other objects and provides methods for working on the objects it contains. A collection can consist of the same types of objects, but can contain objects of different types too.

This lesson adapts the [RMIClient2](#) program from [Part 2, Lesson 2: User Interfaces Revisited](#) to use the Collections application programming interface (API) to maintain and print a list of unique customer IDs. The customer IDs are all objects of type String and represent the same type of information, a customer ID. You could, however, have a collection object that contains objects of type String, Integer, and Double if it makes sense in your application.

- [About Collections](#)
- [Creating a Set](#)
- [Printing](#)
- [More Information](#)

About Collections

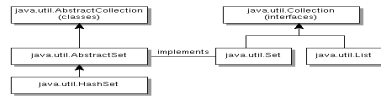
The Collection classes available to use in programs implement Collection interfaces. Interfaces are abstract data types that let collections be manipulated independently of their representation details. There are three primary types of collection interfaces: List, Set, and Map. This lesson focuses on the List and Set collections.

Set implementations do not permit duplicate elements, but List implementations do. Duplicate elements have the same data type and value. For example, two customer IDs of type String containing the value Zelda are duplicate; whereas, an element of type String containing the value 1 and an element of type Integer containing the value 1 are not duplicate.

The API provides two general-purpose Set implementations. HashSet, which stores its elements in a hash table, and TreeSet, which stores its elements in a balanced binary tree called a red-black tree. The example for this lesson uses the HashSet implementation because it currently has the best performance.

This diagram shows the Collection interfaces on the right and the class hierarchy for the java.util.HashSet on the left. You can see that

the `HashSet` class implements the `Set` interface.



Creating a Set

This example adapts the [RMIClient2.java](#) class to collect customer IDs in a `Set` and print the list of customer IDs whenever the `View` button is clicked.

The collection object is a `Set` so if the same customer enters multiple orders, there is only one element for that customer in the list of customer IDs. If the program tries to add an element that is the same as an element already in the set, the second element is simply not added. No error is thrown and there is nothing you have to do in your code.

The [RMIClient2.actionPerformed](#) method calls the `addCustomer` method to add a customer ID to the set when the order processor clicks the `View` button.

The `addCustomer` method shown below adds the customer ID to the set and prints a notice that the customer ID has been added.

```
//Create list of customer IDs
public void addCustomer(String custID){
    s.add(custID);
    System.out.println("Customer ID added");
}
```

Printing

The `print` method is called from the [RMIClient2.actionPerformed](#) method when the order processor clicks the `View` button. The `print` method prints the elements currently in the set to the command line.

Note: A `HashSet` does not guarantee the order of the elements in the set. Elements are printed in the order they occur in the set, but that order is not necessarily the same as the order in which the elements were placed in the set.

To traverse the set, an object of type `Iterator` is returned from the

set. The Iterator object has a hasNext method that lets you test if there is another element in the set, a next that lets you move over the elements in the set, and a remove method that lets you remove an element.

The example print method shows two ways to print the set. The first way uses an iterator and the second way simply calls System.out.println on the set. In the iterator approach, the element returned by the next method is printed to the command line until there are no more elements in the set.

```
//Print customer IDs
public void print(){
//Iterator approach
if(s.size()!=0){
    Iterator it = s.iterator();
    while(it.hasNext()){
        System.out.println(it.next());
    }
//Call System.out.println on the set
    System.out.println(s);
}else{
    System.out.println("No customer IDs available");
}
}
```

More Information

You can find more information on Collections in the [Collections](#) trail in [The Java™ Tutorial](#).

[\[TOP\]](#)

Print Button

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 6: Internationalization

[<< BACK](#) [CONTENTS](#) [NEXT >>](#)

More and more companies, large and small, are doing business around the world using many different languages. Effective communication is always good business, so it follows that adapting an application to a local language adds to profitability through better communication and increased satisfaction.

The Java™ 2 platform provides internationalization features that let you separate culturally dependent data from the application (internationalization) and adapt it to as many cultures as needed (localization).

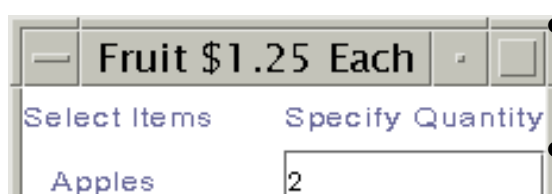
This lesson takes the two client programs from [Part 2, Lesson 5: Collections](#), internationalizes them and adapts the text to France, Germany, and the United States.

- [Identify Culturally Dependent Data](#)
- [Create Keyword and Value Pair Files](#)
- [Internationalize Application Text](#)
- [Internationalize Numbers](#)
- [Compile and Run the Application](#)
- [Program Improvements](#)
- [More Information](#)

Identify Culturally Dependent Data

The first thing you need to do is identify the culturally dependent data in your application. Culturally-dependent data is any data that varies from one culture or country to another. Text is the most obvious and pervasive example of culturally dependent data, but other things like number formats, sounds, times, and dates must be considered too.

The [RMIClient1.java](#) and [RMIClient2.java](#) classes have the following culturally-dependent data visible to the end user:



- Titles and labels (window titles, column heads, and left column labels)
- Buttons (Purchase, Reset, View)

Peaches	<input type="text" value="1"/>
Pears	<input type="text" value="4"/>
Total Items:	7
Total Cost:	8.75
Credit Card:	<input type="text" value="1234-4321-1234-3"/>
Customer ID:	<input type="text" value="munchkin"/>
<input type="button" value="Reset"/> <input type="button" value="Purchase"/>	

- Numbers (values for item and cost totals)

- Error messages

Although the application has a server program, the server program is not being internationalized and localized.

The only visible culturally-dependent data in the server program is the error message text.

The server program runs in one place and the assumption is that

it is not seen by anyone other than the system administrator who understands the language in which the error messages is hard coded. In this example, it is English.

All error messages in `RMIClient1` and `RMIClient2` are handled in `try` and `catch` blocks, as demonstrated by the `print` method below. This way you have access to the error text `No data available` for translation into another language.

```
public void print(){
    if(s!=null){
        Iterator it = s.iterator();
        while(it.hasNext()){
            try{
                String customer = (String)it.next();
                System.out.println(customer);
            }catch (java.util.NoSuchElementException e){
                System.out.println("No data available");
            }
        }
    }else{
        System.out.println("No customer IDs available");
    }
}
```

The `print` method could have been coded to declare the exception in its `throws` clause as shown below, but this way you cannot access the error message text thrown when the method tries to access unavailable data in the set.

In this case, the system-provided text for this error message is sent to the command line regardless of the locale in use for the application. The point here is it is always better to use `try` and `catch` blocks wherever possible if there is any chance the application will be internationalized so you can localize the error message text.

```
public void print()
    throws java.util.NoSuchElementException{
    if(s!=null){
        Iterator it = s.iterator();
        while(it.hasNext()){
            String customer = (String)it.next();
```

```

        System.out.println(customer);
    }
    }else{
        System.out.println("No customer IDs available");
    }
}

```

Here is a list of the title, label, button, number, and error text visible to the user, and therefore, subject to internationalization and localization. This data was taken from both [RMIClient1.java](#) and [RMIClient2.java](#).

- Labels: Apples, Peaches, Pears, Total Items, Total Cost, Credit Card, Customer ID
- Titles: Fruit \$1.25 Each, Select Items, Specify Quantity
- Buttons: Reset, View, Purchase
- Number Values: Value for total items, Value for total cost
- Errors: Invalid Value, Cannot send data to server, Cannot look up remote server object, No data available, No customer IDs available, Cannot access data in server

Create Keyword and Value Pair Files

Because all text visible to the user will be moved out of the application and translated, your application needs a way to access the translated text during execution. This is done with keyword and value pair files, where this is a file for each language. The keywords are referenced from the application instead of the hard-coded text and used to load the appropriate text from the file for the language in use.

For example, you can map the keyword purchase to Kaufen in the German file, Achetez in the French file, and Purchase in the United States English file. In your application, you reference the keyword purchase and indicate the language to use.

Keyword and value pairs are stored in files called properties files because they store information about the programs properties or characteristics. Property files are plain-text format, and you need one file for each language you intend to use.

In this example, there are three properties files, one each for the English, French, and German translations. Because this application currently uses hard-coded English text, the easiest way to begin the internationalization process is to use the hard-coded text to set up the key and value pairs for the English properties file.

The properties files follow a naming convention so the application can locate and load the correct file at run time. The naming convention uses language and country codes which you should make part of the file name. The language and country are both included because the same language can vary between countries. For example, United States English and Australian English are a little different, and Swiss German and Austrian German both differ from each other and from the German spoken in Germany.

These are the names of the properties files for the German (de_DE),

French (fr_FR), and American English (en_US) translations where de, fr, and en indicate the German (Deutsche), French, and English languages; and DE, FR, and US indicate Germany (Deutschland), France, and the United States:

- MessagesBundle_de_DE.properties
- MessagesBundle_en_US.properties
- MessagesBundle_fr_FR.properties

Here is the English language properties file. Keywords appear to the left of the equals (=) sign, and text values appear to the right.

[MessagesBundle_en_US.properties](#)

```
apples=Apples:
peaches=Peaches:
pears=Pears:
items=Total Items:
cost=Total Cost:
card=Credit Card:
customer=Customer ID:

title=Fruit 1.25 Each
1col=Select Items
2col=Specify Quantity

reset=Reset
view=View
purchase=Purchase

invalid=Invalid Value
send=Cannot send data to server
nolookup=Cannot look up remote server object

nodata=No data available
noID=No customer IDs available
noserver=Cannot access data in server
```

With this file complete, you can hand it off to your French and German translators and ask them to provide the French and German equivalents for the text to the right of the equals (=) sign. Keep a copy for yourself because you will need the keywords to internationalize your application text.

The properties file with the [German](#) translations produces this user interface for the fruit order client:

The properties file with the [French](#) translations produces this user interface for the fruit order client:

Internationalize Application Text

This section walks through internationalizing the [RMIClient1.java](#) program. The [RMIClient2.java](#) code is almost identical so you can apply the same steps to that program on your own.

Instance Variables

In addition to adding an import statement for the `java.util.*` package where the internationalization classes are, this program needs the following instance variable declarations for the internationalization process:

```
//Initialized in main method
    static String language, country;
```

```

    Locale currentLocale;
    static ResourceBundle messages;

//Initialized in actionPerformed method
    NumberFormat numFormat;

```

main Method

The program is designed so the user specifies the language to use at the command line. So, the first change to the main method is to add the code to check the command line parameters. Specifying the language at the command line means once the application is internationalized, you can easily change the language without any recompilation.

The `String[] args` parameter to the main method contains arguments passed to the program from the command line. This code expects 3 command line arguments when the end user wants a language other than English. The first argument is the name of the machine on which the program is running. This value is passed to the program when it starts and is needed because this is a networked program using the Remote Method Invocation (RMI) API.

The other two arguments specify the language and country codes. If the program is invoked with 1 command line argument (the machine name only), the country and language are assumed to be United States English.

As an example, here is how the program is started with command line arguments to specify the machine name and German language (de DE). Everything goes on one line.

```

java -Djava.rmi.server.codebase=
    http://kq6py/~zelda/classes/
    -Djava.security.policy=java.policy
    RMIClient1 kq6py.eng.sun.com de DE

```

And here is the main method code. The `currentLocale` instance variable is initialized from the language and country information passed in at the command line, and the `messages` instance variable is initialized from the `currentLocale`.

The `messages` object provides access to the translated text for the language in use. It takes two parameters: the first parameter "MessagesBundle" is the prefix of the family of translation files this application uses, and the second parameter is the `Locale` object that tells the `ResourceBundle` which translation to use.

Note: This style of programming makes it possible for the same user to run the program in different languages, but in most cases, the program will use one language and not rely on command-line arguments to set the country and language.

If the application is invoked with `de DE` command line parameters, this code creates a `ResourceBundle` variable to access the `MessagesBundle_de_DE.properties` file.

```

    public static void main(String[] args){
//Check for language and country codes
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

//Create locale and resource bundle
    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("MessagesBundle",
        currentLocale);

    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

//Create the RMIclient1 object
    RMIclient1 frame = new RMIclient1();

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(
            new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println(messages.getString(
            "nolookup"));
    } catch (java.rmi.RemoteException e){
        System.out.println(messages.getString(
            "nolookup"));
    } catch (java.net.MalformedURLException e) {
        System.out.println(messages.getString(
            "nolookup"));
    }
    }
}

```

The applicable error text is accessed by calling the `getString` method on the `ResourceBundle`, and passing it the keyword that maps to the applicable error text.

```

try {
    String name = "//" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println(messages.getString(
        "nolookup"));
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString(
        "nolookup"));
} catch (java.net.MalformedURLException e) {
    System.out.println(messages.getString(
        "nolookup"));
}

```

Constructor

The window title is set by calling the `getString` method on the `ResourceBundle`, and passing it the keyword that maps to the title text. You must pass the keyword exactly as it appears in the translation file, or you will get a runtime error indicating the resource is unavailable.

```

RMIClient1(){

//Set window title
    setTitle(messages.getString("title"));

```

The next thing the constructor does is use the `args` parameter to look up the remote server object. If there are any errors in this process, the `catch` statements get the applicable error text from the `ResourceBundle` and print it to the command line. User interface objects that display text, such as `JLabel` and `JButton`, are created in the same way:

```

//Create left and right column labels
    col1 = new JLabel(messages.getString("1col"));
    col2 = new JLabel(messages.getString("2col"));
...
//Create buttons and make action listeners
    purchase = new JButton(messages.getString(
        "purchase"));
    purchase.addActionListener(this);

    reset = new JButton(messages.getString("reset"));
    reset.addActionListener(this);

```

actionPerformed Method

In the `actionPerformed` method, the `Invalid Value` error is caught and translated:

```

    if(order.apples.length() > 0){
//Catch invalid number error
        try{
            applesNo = Integer.valueOf(order.apples);

```

```

        order.itotal += applesNo.intValue();
    } catch (java.lang.NumberFormatException e) {
        appleqnt.setText(messages.getString("invalid"));
    }
} else {
    order.itotal += 0;
}

```

The `actionPerformed` method calculates item and cost totals, translates them to the correct format for the language currently in use, and displays them in the user interface.

Internationalize Numbers

A `NumberFormat` object is used to translate numbers to the correct format for the language currently in use. To do this, a `NumberFormat` object is created from the `currentLocale`. The information in the `currentLocale` tells the `NumberFormat` object what number format to use.

Once you have a `NumberFormat` object, all you do is pass in the value you want translated, and you receive a `String` that contains the number in the correct format. The value can be passed in as any data type used for numbers such as `int`, `Integer`, `double`, or `Double`. No code such as to convert an `Integer` to an `int` and back again is needed.

```

//Create number formatter
numFormat = NumberFormat.getNumberInstance(
    currentLocale);

//Display running total
text = numFormat.format(order.itotal);
this.items.setText(text);

//Calculate and display running cost
order.icost = (order.itotal * 1.25);
text2 = numFormat.format(order.icost);
this.cost.setText(text2);

try{
    send.sendOrder(order);
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("send"));
}

```

Compile and Run the Application

Here are the summarized steps for compiling and running the example program. The important thing to note is that when you start the client programs, you need to include language and country codes if you want a language other than United States English.

Compile

These instructions assume development is in the `zelda` home directory.

Unix:

```
cd /home/zelda/classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
cp RemoteServer*.class /home/zelda/public_html/classes
cp Send.class /home/zelda/public_html/classes
cp DataOrder.class /home/zelda/public_html/classes
```

Win32:

```
cd \home\zelda\classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
copy RemoteServer*.class
                        \home\zelda\public_html\classes
copy Send.class \home\zelda\public_html\classes
copy DataOrder.class \home\zelda\public_html\classes
```

Start rmi Registry**Unix:**

```
cd /home/zelda/public_html/classes
unsetenv CLASSPATH
rmiregistry &
```

Win32:

```
cd \home\zelda\public_html\classes
set CLASSPATH=
start rmiregistry
```

Start the Server**Unix:**

```
cd /home/zelda/public_html/classes
java -Djava.rmi.server.codebase=
      http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RemoteServer
```

Win32:

```
cd \home\zelda\public_html\classes
java -Djava.rmi.server.codebase=
      file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RemoteServer
```

Start RMIClient1 in German

Note the addition of de DE for the German language and country at the end of the line.

Unix:

```
cd /home/zelda/classes

java -Djava.rmi.server.codebase=
      http://kq6py/~zelda/classes/
-Djava.security.policy=java.policy
      RMIClient1 kq6py.eng.sun.com de DE
```

Win32:

```
cd \home\zelda\classes

java -Djava.rmi.server.codebase=
      file:c:\home\zelda\classes\
-Djava.security.policy=java.policy RMIClient1
      kq6py.eng.sun.com de DE
```

Start RMIClient2 in French

Note the addition of fr FR for the French language and country at the end of the line.

Unix:

```
cd /home/zelda/classes

java -Djava.rmi.server.codebase=
      http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy
      RMIClient2 kq6py.eng.sun.com fr FR
```

Win32:

```
cd \home\zelda\classes

java -Djava.rmi.server.codebase=
      file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=java.policy RMIClient2
      kq6py.eng.sun.com/home/zelda/public_html fr FR
```

Program Improvements

A real-world scenario for an ordering application like this might be that [RMIClient1](#) is an applet embedded in a web page. When orders are submitted, order processing staff run [RMIClient2](#) as applications from their local machines.

So, an interesting exercise is to convert RMIClient1.java to its applet equivalent. The translation files would be loaded by the applet from the same directory from which the browser loads the applet class.

One way is to have a separate applet for each language with the language and country codes hard coded. Your web page can let them choose the language by clicking a link that launches the appropriate applet. Here are the source code files for the [English](#), [French](#), and [German](#) applets.

Here is the HTML code to load the French applet on a Web page.

```
<HTML>
<BODY>
<APPLET CODE=RMIFrenchApp.class WIDTH=300 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

Note: To run an applet written with Java™ 2 APIs in a browser, the browser must be enabled for the Java 2 Platform. If your browser is not enabled for the Java 2 Platform, you have to use appletviewer to run the applet or install [Java Plug-in](#). Java Plug-in lets you run applets on web pages under the 1.2 version of the Java¹ virtual machine (VM) instead of the web browser's default Java VM.

To use applet viewer, type the following where rmiFrench.html is the HTML file for the French applet.

```
appletviewer rmiFrench.html
```

Another improvement to the program as it currently stands would be enhancing the error message text. You can locate the errors in the [Java API docs](#) and use the information there to make the error message text more user friendly by providing more specific information.

You might also want to adapt the client programs to catch and handle the error thrown when an incorrect keyword is used. Here are the error and stack trace provided by the system when this type of error occurs:

```
Exception in thread "main"
  java.util.MissingResourceException:
Can't find resource
  at java.util.ResourceBundle.getObject(Compiled Code)
  at java.util.ResourceBundle.getString(Compiled Code)
  at RMIClient1.<init>(Compiled Code)
  at RMIClient1.main(Compiled Code)
```

More Information

You can find more information on Internationalization in the [Internationalization](#) trail in [The Java Tutorial](#).

You can find more information on applets in the [Writing Applets](#) trail in [The Java Tutorial](#).

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform

[\[TOP\]](#)

[Print Button](#)

[This page was updated: 3-Apr-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:

(800) 786-7638

Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

**Sun
Microsystem**

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 7: Packages and Java™ Archive File Format

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

Until now, you have used classes from the Java API library by importing the package containing the class or classes you need. A package is a convenient way to organize groups of related classes, and in development, you should organize your application files into packages too. Packages make it easier to locate and use the class files and help you control access to class data at run time.

When your application is fully tested, debugged, and ready for deployment, use the Java™ Archive file format to deploy the application. JAR file format is a compression and file packaging format and tool for bundling executable files with any other related application files so they can be deployed as one unit.

This lesson shows you how to organize the program files from [Part 2, Lesson 6: Internationalization](#) into packages and deploy the executable and other related files to production using JAR file format. Normally, you would use packages from the beginning of development.

- [Setting up Class Packages](#)
 - [Create the Directories](#)
 - [Declare the Packages](#)
 - [Make Classes and Fields Accessible](#)
 - [Change Client Code to Find the Properties File](#)
 - [Compile and Run the Example](#)
- [Using JAR Files to Deploy](#)
 - [Server Set of Files](#)
 - [Fruit Order Client Set of Files](#)
 - [View Order Client Set of Files](#)
 - [More Information](#)

Setting up Class Packages

It is easy to organize class files into packages. All you do is put related class files in the same directory, give the directory a name that relates to the purpose of the classes, and add a line to the top of each class file that declares the package name, which is the same as the directory name where they reside.

For example, the class and other related files for the program files from [Part 2, Lesson 6: Internationalization](#) can be divided into three groups of files: fruit order client, view order client, and server files. Although these three sets of classes are related to each other, they have different functions and are to be deployed separately.

Create the Directories

To organize the internationalization program into three packages, you could create the following three directories and move the listed source files into them:

- client1
 - RMIEnglishApp.java
 - RMIFrenchApp.java
 - RMIGermanApp.java
 - MessagesBundle_de_DE.properties
 - MessagesBundle_en_US.properties
 - MessagesBundle_fr_FR.properties
 - index.html
 - rmiFapp.html
 - rmiGapp.html
 - rmiEapp.html
 - java.policy
- client2
 - RMIClient2.java
 - MessagesBundle_de_DE.properties
 - MessagesBundle_en_US.properties
 - MessagesBundle_fr_FR.properties
 - java.policy
- server
 - DataOrder.java
 - RemoteServer.java
 - Send.java
 - java.policy

Declare the Packages

Each *.java file needs a package declaration at the top that reflects the name of the directory. Also, the fruit order (client1 and view order (client2) client class files need an import statement for the server package because they have to access the remote server object at runtime.

As an example, the package declaration and import statements for the [RMIClient2.java](#) class file look like this:

```
//package declaration
package client2;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;
```

```
import java.util.*;
import java.text.*;

//Import server package
import server.*;
```

Make Classes and Fields Accessible

With class files organized into packages, you have to declare the server classes in the server directory public so they can be instantiated by client programs, which are created from classes in the client1 and client2 directories. If you do not specify public, a class can only be instantiated by an object created from a class in the same package.

So client programs can access the fruit order data, the fields of the DataOrder class have to be public too. The [RemoteServer](#) class and [Send](#) interface need to be public classes, but their fields do not need to be public because they do not have public data.

Fields and methods without an access specifier such as public can only be accessed by objects created from classes in the same package.

Here is the new [DataOrder](#) class.

```
package server;

import java.io.*;

//Make class public
public class DataOrder implements Serializable{

//Make fields public
    public String apples, peaches, pears, cardnum, custID;
    public double icost;
    public int itotal;
}
```

Change Client Code to Find the Properties Files

In the example, the properties files (Messages_*) are stored in the directories with the client source files. This makes it easier to package and deploy the files later. So the programs can find the properties files, you have to make one small change to the client source code.

The code that creates the messages variable needs to include the directory (package name) as follows:

```
messages = ResourceBundle.getBundle(
    "client2" +
    File.separatorChar +
    "MessagesBundle", currentLocale);
```

Compile and Run the Example

Compiling and running the example organized into packages is a little different from compiling and running the example in previous lessons. First, you have to execute the compiler and interpreter commands from one directory above the package directories, and second, you have to

specify the package directories to the compiler and interpreter commands.

Compile

These instructions assume development occurs in the zelda home directory.

Unix:

```
cd /home/zelda/classes

javac server/Send.java
javac server/RemoteServer.java
javac client2/RMIClient2.java
javac client1/RMIFrenchApp.java
javac client1/RMIGermanApp.java
javac client1/RMIEnglishApp.java
rmic -d . server.RemoteServer
cp server/RemoteServer*.class
    /home/zelda/public_html/classes
cp server/Send.class
    /home/zelda/public_html/classes
cp server/DataOrder.class
    /home/zelda/public_html/classes
```

Win32:

```
cd \home\zelda\classes

javac server\Send.java
javac server\RemoteServer.java
javac client2\RMIClient2.java
javac client1\RMIFrenchApp.java
javac client1\RMIGermanApp.java
javac client1\RMIEnglishApp.java
rmic -d . server.RemoteServer
copy server\RemoteServer*.class
    \home\zelda\public_html\classes
copy server\Send.class
    \home\zelda\public_html\classes
copy server\DataOrder.class
    \home\zelda\public_html\classes
```

Note: The `rmic -d . server.RemoteServer` line uses `server.RemoteServer` instead of `server/RemoteServer` so the `_stub` and `_skel` classes are generated properly with the package.

Start rmi Registry:

Unix:

```
cd /home/zelda/public_html/classes
unsetenv CLASSPATH
rmiregistry &
```

Win32:

```
cd \home\zelda\public_html\classes
set CLASSPATH=
start rmiregistry
```

Start the Server

Unix:

```
cd /home/zelda/public_html/classes

java -Djava.rmi.server.codebase=
      http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=
      server/java.policy server/RemoteServer
```

Win32:

```
cd \home\zelda\public_html\classes

java -Djava.rmi.server.codebase=
      file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=
      server\java.policy server\RemoteServer
```

Start RMI GermanApp Here is the HTML code to load the German applet, Note the directory/package name prefixed to the applet class name (client1/RMIGermanApp.class).

```
<HTML>
<BODY>
<APPLET CODE=client1/RMIGermanApp.class WIDTH=300 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

To run the applet with appletviewer, invoke the HTML file from the directory just above client1 as follows:

```
cd /home/zelda/classes

appletviewer rmiGapp.html
```

Start RMI Client2 in French

Unix:

```
cd /home/zelda/classes

java -Djava.rmi.server.codebase=
      http://kq6py/~zelda/classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=client2/java.policy
      client2/RMIClient2 kq6py.eng.sun.com fr FR
```

Win32:

```
cd \home\zelda\classes

java -Djava.rmi.server.codebase=
      file:c:\home\zelda\public_html\classes
-Djava.rmi.server.hostname=kq6py.eng.sun.com
-Djava.security.policy=client2\java.policy
      client2\RMIClient2 kq6py.eng.sun.com fr FR
```

Using JAR Files to Deploy

After testing and debugging, the best way to deploy the two client and server files is to bundle the executables and other related application files into three separate JAR files, one JAR file for each client program, and one JAR file for the server program.

JAR files use the ZIP file format to compress and pack files into, and decompress and unpack files from, the JAR file. JAR files make it easy to deploy programs that consist of many files. Browsers can easily download applets bundled into JAR files, and the download goes much more quickly than if the applet and its related files were not bundled into a JAR file.

Server Set of Files

Here are the server files:

- RemoteServer.class
- RemoteServer_skel.class
- RemoteServer_stub.class
- Send.class
- DataOrder.class
- java.policy

Compress and Pack Server Files

To compress and pack the server files into one JAR file, type the following command on one line. This command is executed in the same directory with the files. If you execute the command from a directory other than where the files are, you have to specify the full pathname.

```
jar cf server.jar
    RemoteServer.class
    RemoteServer_skel.class
    RemoteServer_stub.class
    Send.class
    DataOrder.class
    java.policy
```

`jar` is the `jar` command. If you type `jar` with no options, you get the following help screen. You can see from the help screen that the `cf` options to the `jar` command mean create a new JAR file named `server.jar` and put the list of files that follows into it. The new JAR file is placed in the current directory.

```
kq6py% jar
Usage: jar {ctxu}[vfmOM] [jar-file] [manifest-file]
    [-C dir] files ...
Options:
-c   create new archive
-t   list table of contents for archive
-x   extract named (or all) files from archive
-u   update existing archive
-v   generate verbose output on standard output
-f   specify archive file name
-m   include manifest information from specified
```

```

        manifest file
    -0  store only; use no ZIP compression
    -M  Do not create a manifest file for the entries
    -C  change to the specified directory and
        include the following file
If any file is a directory then it is processed
recursively.
The manifest file name and the archive file name
needs to be specified in the same order the
'm' and 'f' flags are specified.

```

Example 1: to archive two class files into an archive called classes.jar:

```
jar cvf classes.jar Foo.class Bar.class
```

Example 2: use an existing manifest file 'mymanifest' and archive all the files in the foo/ directory into 'classes.jar':

```
jar cvfm classes.jar mymanifest -C foo/ .
```

To deploy the server files, all you have to do is move the server.jar file to a publicly accessible directory on the server where they are to execute.

Decompress and Unpack Server Files

After moving the JAR file to its final location, the compressed and packed files need to be decompressed and unpacked so you can start the server. The following command means extract (x) all files from the server.jar file (f).

```
jar xf server.jar
```

Fruit Order Set of Files

The fruit order set of files (below) consists of applet classes, web pages, translation files, and the policy file. Because they live on the web, they need to be in a directory accessible by the web server. The easiest way to deploy these files is to bundle them all into a JAR file and copy them to their location.

- RMIEnglishApp.class
- RMIFrenchApp.class
- RMIGermanApp.class
- index.html (top-level web page where user chooses language)
- rmiEapp.html (second-level web page for English)
- rmiFapp.html (second-level web page for French)
- rmiGapp.html (second-level web page for German)
- MessagesBundle_de_DE.properties
- MessagesBundle_en_US.properties
- MessagesBundle_fr_FR.properties
- java.policy

Compress and Pack Files

```

jar cf applet.jar
    RMIEnglishApp.class
    RMIFrenchApp.class

```

```

RMIGermanApp.class
index.html
rmiEapp.html
rmiFapp.html
rmiGapp.html
MessagesBundle_de_DE.properties
MessagesBundle_en_US.properties
MessagesBundle_fr_FR.properties
java.policy

```

To deploy the fruit order client files, copy the `applet.jar` file to its final location.

Decompress and Unpack Files

An applet in a JAR file can be invoked from an HTML file without being unpacked. All you do is specify the `ARCHIVE` option to the `APPLET` tag in your web page, which tells `appletviewer` the name of the JAR file containing the class file. Be sure to include the package directory when you specify the applet class to the `CODE` option.

You can leave the translation files and policy file in the JAR file. When using `appletviewer`, the applet invoked from the JAR file will find them in the JAR file.

```

<HTML>
<BODY>
<APPLET CODE=client1/RMIFrenchApp.class
  ARCHIVE="applet.jar"
  WIDTH=300
  HEIGHT=300>
</APPLET>
</BODY>
</HTML>

```

However, you do need to unpack the web pages so you can move them to their final location. The following command does this. Everything goes on one line.

```

jar xv applet.jar index.html
    rmiEapp.html
    rmiFapp.html
    rmiGapp.html

```

Note: To run the HTML files from a browser, you need to unpack the JAR file, copy the `java.policy` file to your home directory and make sure it has the right name (`.java.policy` for Unix and `java.policy` for Windows), and install Java Plug-In.

View Order Set of Files

The view order set of files (below) consists of the application class file and the policy file.

- `RMIClient2.class`
- `java.policy`

Compress and Pack Files

```
jar cf vieworder.jar RMIClient2.class java.policy
```

To deploy the view order client files, copy the vieworder.jar file to its final location.

Decompress and Unpack Files

```
jar xf vieworder.jar
```

More Information

You can find more information on packages in the [Creating and Using Packages](#) lesson in [The Java Tutorial](#).

You can find more information on these and other JAR file format topics in the [JAR File Format](#) trail in [The Java Tutorial](#).

[\[TOP\]](#)

Print Button

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics, Part 2

Lesson 8: Object-Oriented Programming

[<<BACK](#) [CONTENTS](#) [NEXT>>](#)

You have probably heard a lot of talk about object-oriented programming. And, if the Java™ programming language is your first experience with an object-oriented language, you are probably wondering what all the talk is about.

You already know a little about object-oriented programming because after working the example programs in [Java Programming Language Basics, Part 1](#) and [Part 2](#), you are somewhat familiar with the object-oriented concepts of class, object, instance, and inheritance plus the access levels public and private. But mostly, you have been doing object-oriented programming without really thinking about it.

And that is one of the great things about the Java programming language. It is inherently object oriented.

To help you gain a deeper understanding of object-oriented programming and its benefits, this lesson presents a very brief overview of object-oriented concepts and terminology as they relate to some of the example code presented in this tutorial.

- [Object-Oriented Programming Defined](#)
- [Classes](#)
- [Objects](#)
- [Well-Defined Boundaries and Cooperation](#)
- [Inheritance](#)
- [Polymorphism](#)
- [Data Access Levels](#)
- [Your Own Classes](#)
- [Program Improvements](#)
- [More Information](#)

Object-Oriented Programming Defined

Object-oriented programming is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects.

Classes

A class is a structure that defines the data and the methods to work on that data. When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries.

The `ExampleProgram` class from the simple program in the first lesson of Part 1 is a programmer-written class that uses the `java.lang.System` class from the Java platform API libraries to print a character string to the command line.

```
class ExampleProgram {
    public static void main(String[] args){
        System.out.println("I'm a simple Program");
    }
}
```

Classes in the Java platform API libraries define a set of objects that share a common structure and behavior. The `java.lang.System` class used in the example defines such things as standard input, output, and error streams, and access to system properties. In contrast, the `java.lang.String` class defines character strings.

In the example, you do not see an explicit use of the `String` class, but in the Java language, a character string can be used anywhere a method expects to receive a `String` object. During execution, the Java platform creates a `String` object from the character string passed to the `System.out.println` call, but your program cannot call any of the `String` class methods because it did not instantiate the `String` object.

If you want access to the `String` methods, you can rewrite the example program to create a `String` object as follows. This way, you can call a method such as the `String.concat` method that adds text to the original string.

```
class ExampleProgram {
    public static void main(String[] args){
        String text = new String("I'm a simple Program ");
        System.out.println(text);
        String text2 = text.concat(
            "that uses classes and objects");
        System.out.println(text2);
    }
}
```

The output looks like this:

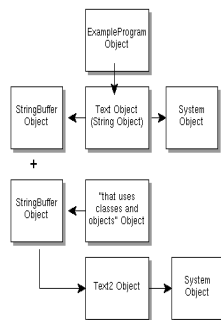
```
I'm a simple Program
I'm a simple Program that uses classes and objects
```

Objects

An instance is an executable copy of a class. Another name for instance is object. There can be any number of objects of a given

class in memory at any one time.

In the last example, four different String objects are created for the concatenation operation, text object, text2 object, and a String object created behind the scenes from the " that uses classes and objects" character string passed to the String.concat method.



Also, because String objects cannot be edited, the java.lang.String.concat method converts the String objects to StringBuffer (editable) string objects to do the concatenation.

Besides the String object, there is an instance of the ExampleProgram.java class in memory as well.

The System class is never instantiated by the ExampleProgram class because it contains only static variables and methods, and therefore, cannot be instantiated by a program, but it is instantiated behind the scenes by the Java™ virtual machine¹ (VM).

Well-Defined Boundaries and Cooperation

Class definitions must allow objects to cooperate during execution. In the previous section, you saw how the System, String, and StringBuffer objects cooperated to print a concatenated character string to the command line.

This section changes the example program to display the concatenated character string in a JLabel component in a user interface to further illustrate the concepts of well-defined class boundaries and object cooperation.

The program code to place the text in a label to display it in a user interface uses a number of cooperating classes. Each class has its

own function and purpose as summarized below, and where appropriate, the classes are defined to work with objects of another class.

- ExampleProgram defines the program data and methods to work on that data.
- JFrame defines the top-level window including the window title and frame menu.
- WindowEvent defines behavior for (works with) the Close option on the frame menu.
- String defines a character string to create the label.
- JLabel defines a user interface component to display static text.
- JPanel defines the background color, contains the label, and uses the default layout manager (java.awt.FlowLayout) to position the label on the display.

While each class has its own specific purpose, they all work together to create the simple user interface you see here.



```
import javax.swing.*;
import java.awt.Color;
import java.awt.event.*;

class ExampleProgram extends JFrame {

    public ExampleProgram(){
        String text = new String("I'm a simple Program ");
        String text2 = text.concat(
            "that uses classes and objects");

        JLabel label = new JLabel(text2);
        JPanel panel = new JPanel();
        panel.setBackground(Color.white);

        getContentPane().add(panel);
        panel.add(label);
    }

    public static void main(String[] args){
        ExampleProgram frame = new ExampleProgram();

        frame.setTitle("Fruit $1.25 Each");
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };

        frame.addWindowListener(l);
        frame.pack();
    }
}
```

```

        frame.setVisible(true);
    }
}

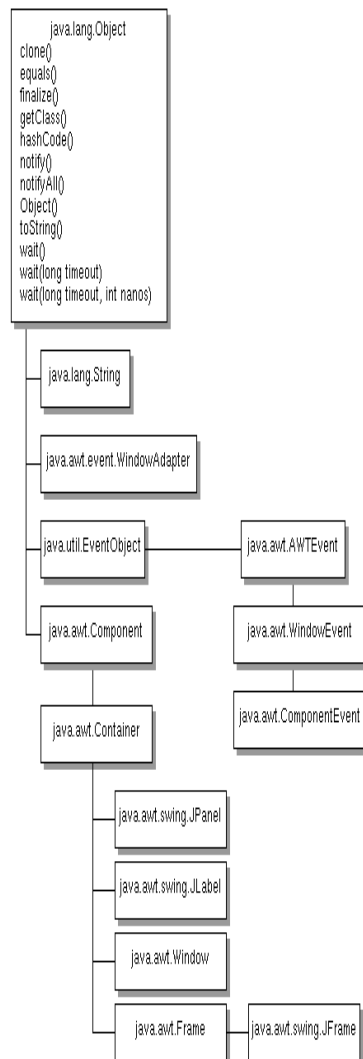
```

Inheritance

One object-oriented concept that helps objects work together is inheritance. Inheritance defines relationships among classes in an object-oriented language. In the Java programming language, all classes descend from `java.lang.Object` and implement its methods.

The following diagram shows the class hierarchy as it descends from `java.lang.Object` for the classes in the user interface example above. The `java.lang.Object` methods are also shown because they are inherited and implemented by all of its subclasses, which is every class in the Java API libraries. `java.lang.Object` defines the core set of behaviors that all classes have in common.

As you move down the hierarchy, each class adds its own set of class-specific fields and methods to what it inherits from its superclass or superclasses. The `java.awt.swing.JFrame` class inherits fields and methods from `java.awt.Frame`, which inherits fields and methods from `java.awt.Container`, which inherits fields and methods from `java.awt.Component`, which finally inherits from `java.lang.Object`, and each subclass adds its own fields and methods as needed.



Polymorphism

Another way objects work together is to define methods that take other objects as parameters. You get even more cooperation and efficiency when the objects are united by a common superclass. All classes in the Java programming language have an inheritance relationship.

For example, if you define a method that takes a `java.lang.Object` as a parameter, it can accept any object in the entire Java platform. If you define a method that takes a `java.awt.Component` as a parameter, it can accept any component object. This form of cooperation is called **polymorphism**.

You saw an example of polymorphism in [Part 2, Lesson 5: Collections](#) where a collection object can contain any type of object as long as it descends from `java.lang.Object`. It is repeated here to show you that `Set` collection can add a `String` object and an `Integer` object to the `Set` because the `Set.add` method is defined to accept any class instance that traces back to the `java.lang.Object` class.

```
String custID = "munchkin";
Integer creditCard = new Integer(25);

Set s = new HashSet();
s.add(custID);
s.add(creditCard);
```

Data Access Levels

Another way classes work together is through access level controls. Classes, and their fields and methods have access levels to specify how they can be used by other objects during execution. While cooperation among objects is desirable, there are times when you will want to explicitly control access, and specifying access levels is the way to gain that control. When you do not specify an access level, the default access level is in effect.

Classes

By default, a class can be used only by instances of other classes in

the same package. A class can be declared public to make it accessible to all class instances regardless of what package its class is in. You might recall that in Part 1, [Part 1, Lesson 3: Building Applets](#), the applet class had to be declared public so it could be accessed by the appletviewer tool because the appletviewer program is created from classes in another package.

Here is an applet class declared to have a public access level:

```
public class DbApplet extends Applet
    implements ActionListener {
```

Without the public access level (shown below), its access level is package by default. You get an error when you try to interpret a class with an access level of package with the appletviewer tool. The same is true if the access level is protected or private.

```
class DbApplet extends Applet
    implements ActionListener {
```

Also, in [Part 2, Lesson 6: Internationalization](#) the server classes are made public so client classes can access them.

Fields and Methods

Fields and methods can be declared private, protected, public, or package. If no access level is specified, the field or method access level is package by default.

private: A private field or method is accessible only to the class in which it is defined. In Part 1, [Lesson 7: Database Access and Permissions](#) the connection, user name, and password for establishing the database access are all private. This is to prevent an outside class from accessing them and jeopardizing the database connection, or compromising the secret user name and password information.

```
private Connection c;
```

protected: A protected field or method is accessible to the class itself, its subclasses, and classes in the same package.

public: A public field or method is accessible to any class of any parentage in any package. In [Part 2, Lesson 6: Internationalization](#) server data accessed by client programs is made public.

package: A package field or method is accessible to other classes in the same package.

Your Own Classes

When you use the Java API library classes, they have already been designed with the above concepts in mind. They all descend from `java.lang.Object` giving them an inheritance relationship; they have well-defined boundaries; and they are designed to cooperate with each other where appropriate.

For example, you will not find a `String` class that takes an `Integer` object as input because that goes beyond the well-defined boundary for a `String`. You will, however, find the `Integer` class has a method for converting its integer value to a `String` so its value can be displayed in a user interface component, which only accepts `String` objects.

But what about when you write your own classes? How can you be sure your classes have well-defined boundaries, cooperate, and make use of inheritance? One way is to look at the functions you need a program to perform and separate them into distinct modules where each functional module is defined by its own class or group of classes.

Well-Defined and Cooperating Classes

Looking at the [RMIClient2](#) class from the [Part 2, Lesson 5: Collections](#) lesson, you can see it performs the following functions: Get data, display data, store customer IDs, print customer IDs, and reset the display.

Getting data, displaying the data, and resetting the display are closely related and easily form a functional module. But in a larger program with more data processing, the storing and printing of customer IDs could be expanded to store and print a wider range of data. In such a case, it would make sense to have a separate class for storing data, and another class for printing it in various forms.

You could, for example, have a class that defines how to store customer IDs, and tracks the number of apples, peaches, and pears sold during the year. You could also have another class that defines report printing. It could access the stored data to print reports on apples, peaches, and pears sold by the month, per customer, or throughout a given season.

Making application code modular by separating out functional units makes it easier to update and maintain the source code. When you change a class, as long as you did not change any part of its public interface, you only have to recompile that one class.

Inheritance

Deciding what classes your program needs means separating functions into modules, but making your code more efficient and easier to maintain means looking for common functions where you can use inheritance. If you need to write a class that has functionality similar to a class in the Java API libraries, it makes sense to extend that API library class and use its methods rather than write everything from scratch.

The [RMIClient2](#) class from the [Part 2, Lesson 5: Collections](#) lesson extends `JFrame` to leverage the ready-made functionality it provides for a program's top-level window including, frame menu closing behavior, background color setting, and a customized title.

Likewise, if you want to add customized behavior to an existing

class, you can extend that class and add the functionality you want. For example, you might want to create your own JButton class with a different look. To do this, you can write your own class that extends JButton and implement it to appear the way you want. Then your program can instantiate your button class instead of the JButton class whenever you need a button with the new look you created.

Access Levels

You should always keep access levels in mind when you declare classes, fields, and methods. Consider which objects really need access to the data, and use packages and access levels to protect your application data from all other objects executing in the system.

Most object-oriented applications do not allow other objects to access their fields directly by declaring them private. Then they make their methods protected, public, or package as needed and allow other objects to manipulate their private data by calling the methods only. This way, you can update your class by changing a field definition and the corresponding method implementation, but other objects that access that data do not need to be changed because their interface to the data (the method signature) has not changed.

Program Improvements

It is always best to restrict access as much as possible. Going back to [Part 2, Lesson 7: Packages and JAR Files](#), the server classes had to be made public and the DataOrder class fields also had to be made public so the client programs can access them.

At that time, no access level was specified for the other classes and fields so they are all package by default. All methods have an access level of public.

A good exercise would be to go back to the client classes and give the classes, fields, and methods an access level so they are not accessed inappropriately by other objects.

Here is one possible solution for the [RMIClient1.java](#) and [RMIClient2.java](#) client programs. Can you explain why the actionPerformed method cannot be made private? If not, make it private, run the javac command to compile, and see what the compiler has to say about it.

More Information

You can find more information on Object-oriented programming concepts files in the [Object-Oriented Programming Concepts](#) trail in [The Java Tutorial](#).

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[Print Button](#)

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)
[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Downloads, APIs,](#)[Java Developer](#)[Tutorials, Tech Articles,](#)[Online Support](#)[Community Discussion](#)[News & Events from](#)[Products from](#)[How Java Technology](#)[Print Button](#)

Java™ Programming Language Basics

In Closing

[<<BACK](#) [CONTENTS](#)

After completing [Java™ Programming Language Basics, Part 1](#) and Part 2 of this training series you should have a solid understanding of Java™ programming basics and how to use some of the more common application programming interfaces (APIs) available in the Java platform.

With this foundation, you can explore Java programming on your own with the help of the [articles](#) and [training](#) materials available on the JDC, or the documents available on the java.sun.com site.

In particular, one tutorial you might find the [Writing Advanced Applications for the Java™ Platform](#) a good place to continue your studies.

[Monica Pawlan](#) is a staff writer on the JDC team. She has a background in 2D and 3D graphics, security, database products, and loves to explore emerging technologies.
monica.pawlan@eng.sun.com

[Print Button](#)

[This page was updated: 31-Mar-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) - [Applets](#) - [Tutorial](#) - [Employment](#) - [Business & Licensing](#) - [Java Store](#) - [Java in the Real World](#)

[FAQ](#) | [Feedback](#) | [Map](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.

Sun
Microsystem

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

import java.util.*;

class RMIClient2 extends JFrame
    implements ActionListener{

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    static Send send;
    String customer;
    HashSet s = new HashSet();

    RMIClient2(){ //Begin Constructor
//Create labels
        creditCard = new JLabel("Credit Card:");
        custID = new JLabel("Customer ID:");
        apples = new JLabel("Apples:");
        peaches = new JLabel("Peaches:");
        pears = new JLabel("Pears:");
        total = new JLabel("Total Items:");
        cost = new JLabel("Total Cost:");

//Create text areas
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton("View Order");
        view.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create panel for 2-column layout
//Set white background color
        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(creditCard);
        panel.add(creditNo);

        panel.add(custID);
        panel.add(customerNo);

        panel.add(apples);
```

```

    panel.add(applesNo);

    panel.add(peaches);
    panel.add(peachesNo);

    panel.add(pears);
    panel.add(pearsNo);

    panel.add(total);
    panel.add(itotal);

    panel.add(cost);
    panel.add(icost);

    panel.add(view);
    panel.add(reset);

} //End Constructor

//Create list of customer IDs
public void addCustomer(String custID){
    s.add(custID);
    System.out.println("Customer ID added");
}

//Print customer IDs
public void print(){
    if(s.size()!=0){
        Iterator it = s.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
        System.out.println(s);
    }else{
        System.out.println("No customer IDs available");
    }
}

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String unit, i;
    double cost;
    Double price;
    int items;
    Integer itms;
    DataOrder order = new DataOrder();

//If View button pressed
//Get data from server and display it
    if(source == view){
        try{
            order = send.getOrder();
            creditNo.setText(order.cardnum);
            customerNo.setText(order.custID);
//Get customer ID and add to list
            addCustomer(order.custID);
            applesNo.setText(order.apples);
            peachesNo.setText(order.peaches);
            pearsNo.setText(order.pears);

            cost = order.icost;
            price = new Double(cost);
            unit = price.toString();
            icost.setText(unit);

```

```
        items = order.itotal;
        itms = new Integer(items);
        i = itms.toString();
        itotal.setText(i);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot access data in server");
    }
}
```

```
//Print
```

```
    print();
}
```

```
//If Reset button pressed
```

```
//Clear all fields
```

```
    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}
```

```
public static void main(String[] args){
    RMIClient2 frame = new RMIClient2();
    frame.setTitle("Fruit Order");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
}
```

```
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
```

```
if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}
```

```
try {
    String name = "://" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println("Cannot access data in server");
} catch (java.rmi.RemoteException e){
    System.out.println("Cannot access data in server");
} catch (java.net.MalformedURLException e) {
    System.out.println("Cannot access data in server");
}
}
```

```
}
```

```
package client1;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

import server.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    private JLabel col1, col2;
    private JLabel totalItems, totalCost;
    private JLabel cardNum, custID;
    private JLabel applechk, peachchk, peachchk;

    private JButton purchase, reset;
    private JPanel panel;

    private JTextField appleqnt, pearqnt, peachqnt;
    private JTextField creditCard, customer;
    private JTextArea items, cost;

    private static Send send;

//Internationalization variables
    private static Locale currentLocale;
    private static ResourceBundle messages;
    private static String language, country;
    private NumberFormat numFormat;

    private RMIClient1(){ //Begin Constructor
        setTitle(messages.getString("title"));
    }

//Create left and right column labels
    col1 = new JLabel(messages.getString("1col"));
    col2 = new JLabel(messages.getString("2col"));

//Create labels and text field components
    applechk = new JLabel(" " + messages.getString("apples"));
    appleqnt = new JTextField();
    appleqnt.addActionListener(this);

    peachchk = new JLabel(" " + messages.getString("pears"));
    pearqnt = new JTextField();
    pearqnt.addActionListener(this);

    peachchk = new JLabel(" " + messages.getString("peaches"));
    peachqnt = new JTextField();
    peachqnt.addActionListener(this);

    cardNum = new JLabel(" " + messages.getString("card"));
    creditCard = new JTextField();
    pearqnt.setNextFocusableComponent(creditCard);
```

```

        customer = new JTextField();
        custID = new JLabel("    " + messages.getString("customer"));

//Create labels and text area components
        totalItems = new JLabel("    " + messages.getString("items"));
        totalCost = new JLabel("    " + messages.getString("cost"));

        items = new JTextArea();
        cost = new JTextArea();

//Create buttons and make action listeners
        purchase = new JButton(messages.getString("purchase"));
        purchase.addActionListener(this);

        reset = new JButton(messages.getString("reset"));
        reset.addActionListener(this);

//Create a panel for the components
        panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(col1);
        panel.add(col2);

        panel.add(applechk);
        panel.add(appleqnt);

        panel.add(peachchk);
        panel.add(peachqnt);

        panel.add(pearchk);
        panel.add(pearqnt);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);

    } //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

```

```
//If Purchase button pressed . . .
    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();

//Calculate total items
        if(order.apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(order.apples);
                order.itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.peaches.length() > 0){
//Catch invalid number error
            try{
                peachesNo = Integer.valueOf(order.peaches);
                order.itotal += peachesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                peachqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.pears.length() > 0){
//Catch invalid number error
            try{
                pearsNo = Integer.valueOf(order.pears);
                order.itotal += pearsNo.intValue();
            }catch(java.lang.NumberFormatException e){
                pearqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

//Create number formatter
        numFormat = NumberFormat.getNumberInstance(currentLocale);

//Display running total
        text = numFormat.format(order.itotal);
        this.items.setText(text);

//Calculate and display running cost
        order.icost = (order.itotal * 1.25);
        text2 = numFormat.format(order.icost);
        this.cost.setText(text2);

        try{
            send.sendOrder(order);
        } catch (java.rmi.RemoteException e) {
            System.out.println(messages.getString("send"));
        }
    }
}
```

```

//If Reset button pressed
//Clear all fields
    if(source == reset){
        creditCard.setText("");
        appleqnt.setText("");
        peachqnt.setText("");
        pearqnt.setText("");
        creditCard.setText("");
        customer.setText("");

        order.icost = 0;
        cost = new Double(order.icost);
        text2 = cost.toString();
        this.cost.setText(text2);

        order.itotal = 0;
        num = new Integer(order.itotal);
        text = num.toString();
        this.items.setText(text);
    }
}

public static void main(String[] args){
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("client1" + File.separatorChar +
"MessagesBundle", currentLocale);

    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    RMIClient1 frame = new RMIClient1();
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println(messages.getString("nolookup"));
    } catch (java.rmi.RemoteException e){
        System.out.println(messages.getString("nolookup"));
    } catch (java.net.MalformedURLException e) {
        System.out.println(messages.getString("nollookup"));
    }
}
}

```

```
package client2;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.io.FileInputStream.*;
import java.io.RandomAccessFile.*;
import java.io.File;

import java.util.*;
import java.text.*;

import server.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    private JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    private JButton view, reset;
    private JPanel panel;
    private JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    private static Send send;
    private String customer;
    private Set s = null;

//Internationalization variables
    private static Locale currentLocale;
    private static ResourceBundle messages;
    private static String language, country;
    private NumberFormat numFormat;

    private RMIClient2(){ //Begin Constructor

        setTitle(messages.getString("title"));

//Create labels
        creditCard = new JLabel(messages.getString("card"));
        custID = new JLabel(messages.getString("customer"));
        apples = new JLabel(messages.getString("apples"));
        peaches = new JLabel(messages.getString("peaches"));
        pears = new JLabel(messages.getString("pears"));
        total = new JLabel(messages.getString("items"));
        cost = new JLabel(messages.getString("cost"));

//Create text areas
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton(messages.getString("view"));
        view.addActionListener(this);
```

```

    reset = new JButton(messages.getString("reset"));
    reset.addActionListener(this);

```

```

//Create panel for 2-column layout
//Set white background color
    panel = new JPanel();
    panel.setLayout(new GridLayout(0,2));
    panel.setBackground(Color.white);

```

```

//Add components to panel columns
//going left to right and top to bottom
    getContentPane().add(panel);
    panel.add(creditCard);
    panel.add(creditNo);

```

```

    panel.add(custID);
    panel.add(customerNo);

```

```

    panel.add(apples);
    panel.add(applesNo);

```

```

    panel.add(peaches);
    panel.add(peachesNo);

```

```

    panel.add(pears);
    panel.add(pearsNo);

```

```

    panel.add(total);
    panel.add(itotal);

```

```

    panel.add(cost);
    panel.add(icost);

```

```

    panel.add(view);
    panel.add(reset);

```

```

} //End Constructor

```

```

//Create list of customer IDs
private void customerList(String custID){
    if(s==null){
        s = new HashSet();
        s.add(custID);
    }else{
        s.add(custID);
    }
}

```

```

//Print customer IDs
private void print(){
    if(s!=null){
        Iterator it = s.iterator();
        while(it.hasNext()){
            try{
                String customer = (String)it.next();
                System.out.println(customer);
            }catch (java.util.NoSuchElementException e){
                System.out.println("No data available");
            }
        }
    }else{
        System.out.println("No customer IDs available");
    }
}

```

```

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String unit, i;
    double cost;
    Double price;
    int items;
    Integer itms;
    DataOrder order = new DataOrder();

//If View button pressed
//Get data from server and display it
    if(source == view){
        try{
            order = send.getOrder();
            creditNo.setText(order.cardnum);
            customerNo.setText(order.custID);
//Get customer ID and add to list
            customerList(order.custID);
            applesNo.setText(order.apples);
            peachesNo.setText(order.peaches);
            pearsNo.setText(order.pears);

//Create number formatter
            numFormat = NumberFormat.getNumberInstance(currentLocale);

            price = new Double(order.icost);
            unit = numFormat.format(price);
            icost.setText(unit);

            itms = new Integer(order.itotal);
            i = numFormat.format(order.itotal);
            itotal.setText(i);
        } catch (java.rmi.RemoteException e) {
            System.out.println("Cannot access data in server");
        }
//Print
        print();
    }

//If Reset button pressed
//Clear all fields
    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

public static void main(String[] args){
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("client2" + File.separatorChar +

```

```
"MessagesBundle", currentLocale);

WindowListener l = new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
};

RMIClient2 frame = new RMIClient2();
frame.addWindowListener(l);
frame.pack();
frame.setVisible(true);

if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}

try {
    String name = "/" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println(messages.getString("nolookup"));
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("nolookup"));
} catch (java.net.MalformedURLException e) {
    System.out.println(messages.getString("nolookup"));
}
}
}
```

```
package client2;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

import server.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    static Send send;
    String customer;
    Set s = null;

//Internationalization variables
    static Locale currentLocale;
    static ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    RMIClient2(){ //Begin Constructor

        setTitle(messages.getString("title"));

//Create labels
        creditCard = new JLabel(messages.getString("card"));
        custID = new JLabel(messages.getString("customer"));
        apples = new JLabel(messages.getString("apples"));
        peaches = new JLabel(messages.getString("peaches"));
        pears = new JLabel(messages.getString("pears"));
        total = new JLabel(messages.getString("items"));
        cost = new JLabel(messages.getString("cost"));

//Create text areas
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton(messages.getString("view"));
        view.addActionListener(this);

        reset = new JButton(messages.getString("reset"));
        reset.addActionListener(this);

//Create panel for 2-column layout
```

```
//Set white background color
    panel = new JPanel();
    panel.setLayout(new GridLayout(0,2));
    panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
    getContentPane().add(panel);
    panel.add(creditCard);
    panel.add(creditNo);

    panel.add(custID);
    panel.add(customerNo);

    panel.add(apples);
    panel.add(applesNo);

    panel.add(peaches);
    panel.add(peachesNo);

    panel.add(pears);
    panel.add(pearsNo);

    panel.add(total);
    panel.add(itotal);

    panel.add(cost);
    panel.add(icost);

    panel.add(view);
    panel.add(reset);

} //End Constructor

//Create list of customer IDs
public void addCustomer(String custID){
    s.add(custID);
    System.out.println("Customer ID added");
}

//Print customer IDs
public void print(){
    if(s.size()!=0){
        Iterator it = s.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
        System.out.println(s);
    }else{
        System.out.println("No customer IDs available");
    }
}

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String unit, i;
    double cost;
    Double price;
    int items;
    Integer itms;
    DataOrder order = new DataOrder();

//If View button pressed
//Get data from server and display it
    if(source == view){
```

```

    try{
        order = send.getOrder();
        creditNo.setText(order.cardnum);
        customerNo.setText(order.custID);
//Get customer ID and add to list
        addCustomer(order.custID);
        applesNo.setText(order.apples);
        peachesNo.setText(order.peaches);
        pearsNo.setText(order.pears);

//Create number formatter
        numFormat = NumberFormat.getNumberInstance(currentLocale);

        price = new Double(order.icost);
        unit = numFormat.format(price);
        icost.setText(unit);

        itms = new Integer(order.itotal);
        i = numFormat.format(order.itotal);
        itotal.setText(i);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot access data in server");
    }
//Print
    print();
}

//If Reset button pressed
//Clear all fields
    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

public static void main(String[] args){
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("client2" + File.separatorChar +
"MessagesBundle", currentLocale);

    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    RMIClient2 frame = new RMIClient2();
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
}

```

```
if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}

try {
    String name = "//" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println(messages.getString("nolookup"));
} catch (java.rmi.RemoteException e){
    System.out.println(messages.getString("nolookup"));
} catch (java.net.MalformedURLException e) {
    System.out.println(messages.getString("nolookup"));
}
}
}
```

```
package server;

import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

public class RemoteServer extends UnicastRemoteObject
    implements Send {

    Integer num = null;
    String orders = null;
    int value = 0, get = 0;

    public RemoteServer() throws RemoteException {
        super();
    }

    public void sendOrder(DataOrder order){
        value += 1;
        num = new Integer(value);
        orders = num.toString();
        try{
            FileOutputStream fos = new FileOutputStream(orders);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(order);
        }catch (java.io.FileNotFoundException e){
            System.out.println("File not found");
        }catch (java.io.IOException e){
            System.out.println("Unable to write to file");
        }
    }

    public DataOrder getOrder(){

        DataOrder order = null;

        if(value == 0){
            System.out.println("No Orders To Process");
        }

        if(value > get){
            get += 1;
            num = new Integer(get);
            orders = num.toString();
            try{
                FileInputStream fis = new FileInputStream(orders);
                ObjectInputStream ois = new ObjectInputStream(fis);
                order = (DataOrder)ois.readObject();
            }catch (java.io.FileNotFoundException e){
                System.out.println("File not found");
            }catch (java.io.IOException e){
                System.out.println("Unable to read file");
            }catch (java.lang.ClassNotFoundException e){
                System.out.println("Data class unavailable");
            }
        }else{
            System.out.println("No Orders To Process");
        }
        return order;
    }
}
```

```
}  
  
public static void main(String[] args){  
    if(System.getSecurityManager() == null) {  
        System.setSecurityManager(new RMISecurityManager());  
    }  
    String name = "//kq6py.eng.sun.com/Send";  
    try {  
        Send remoteServer = new RemoteServer();  
        Naming.rebind(name, remoteServer);  
        System.out.println("RemoteServer bound");  
    } catch (java.rmi.RemoteException e) {  
        System.out.println("Cannot create remote server object");  
    } catch (java.net.MalformedURLException e) {  
        System.out.println("Cannot look up server object");  
    }  
}  
}
```

```
package server;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Send extends Remote {
    public void sendOrder(DataOrder order) throws RemoteException;
    public DataOrder getOrder() throws RemoteException;
}
```

```
package server;  
  
import java.io.*;  
  
public class DataOrder implements Serializable{  
  
    public String apples, peaches, pears, cardnum, custID;  
    public double icost;  
    public int itotal;  
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;

//Internationalization variables
    static Locale currentLocale;
    static ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    RMIClient1(){ //Begin Constructor
        setTitle(messages.getString("title"));
    }

//Create left and right column labels
    col1 = new JLabel(messages.getString("1col"));
    col2 = new JLabel(messages.getString("2col"));

//Create labels and text field components
    applechk = new JLabel(" " + messages.getString("apples"));
    appleqnt = new JTextField();
    appleqnt.addActionListener(this);

    peachchk = new JLabel(" " + messages.getString("pears"));
    peachqnt = new JTextField();
    peachqnt.addActionListener(this);

    peachchk = new JLabel(" " + messages.getString("peaches"));
    peachqnt = new JTextField();
    peachqnt.addActionListener(this);

    cardNum = new JLabel(" " + messages.getString("card"));
    creditCard = new JTextField();
    peachqnt.setNextFocusableComponent(creditCard);

    customer = new JTextField();
    custID = new JLabel(" " + messages.getString("customer"));
```

```
//Create labels and text area components
totalItems = new JLabel("    " + messages.getString("items"));
totalCost = new JLabel("    " + messages.getString("cost"));

items = new JTextArea();
cost = new JTextArea();

//Create buttons and make action listeners
purchase = new JButton(messages.getString("purchase"));
purchase.addActionListener(this);

reset = new JButton(messages.getString("reset"));
reset.addActionListener(this);

//Create a panel for the components
panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
panel.setLayout(new GridLayout(0,2));
panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
getContentPane().add(panel);
panel.add(col1);
panel.add(col2);

panel.add(applechk);
panel.add(appleqnt);

panel.add(peachchk);
panel.add(peachqnt);

panel.add(pearchk);
panel.add(pearqnt);

panel.add(totalItems);
panel.add(items);

panel.add(totalCost);
panel.add(cost);

panel.add(cardNum);
panel.add(creditCard);

panel.add(custID);
panel.add(customer);

panel.add(reset);
panel.add(purchase);

} //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

//If Purchase button pressed . . .
    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
```

```
order.custID = customer.getText();
order.apples = appleqnt.getText();
order.peaches = peachqnt.getText();
order.pears = pearqnt.getText();
```

```
//Calculate total items
if(order.apples.length() > 0){
//Catch invalid number error
try{
    applesNo = Integer.valueOf(order.apples);
    order.itotal += applesNo.intValue();
} catch (java.lang.NumberFormatException e){
    appleqnt.setText(messages.getString("invalid"));
}
} else {
    order.itotal += 0;
}

if(order.peaches.length() > 0){
//Catch invalid number error
try{
    peachesNo = Integer.valueOf(order.peaches);
    order.itotal += peachesNo.intValue();
} catch (java.lang.NumberFormatException e){
    peachqnt.setText(messages.getString("invalid"));
}
} else {
    order.itotal += 0;
}

if(order.pears.length() > 0){
//Catch invalid number error
try{
    pearsNo = Integer.valueOf(order.pears);
    order.itotal += pearsNo.intValue();
} catch (java.lang.NumberFormatException e){
    pearqnt.setText(messages.getString("invalid"));
}
} else {
    order.itotal += 0;
}

//Create number formatter
numFormat = NumberFormat.getNumberInstance(currentLocale);

//Display running total
text = numFormat.format(order.itotal);
this.items.setText(text);

//Calculate and display running cost
order.icost = (order.itotal * 1.25);
text2 = numFormat.format(order.icost);
this.cost.setText(text2);

try{
    send.sendOrder(order);
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("send"));
}
}

//If Reset button pressed
//Clear all fields
if(source == reset){
    creditCard.setText("");
}
```

```
appleqnt.setText("");
peachqnt.setText("");
pearqnt.setText("");
creditCard.setText("");
customer.setText("");

order.icost = 0;
cost = new Double(order.icost);
text2 = cost.toString();
this.cost.setText(text2);

order.itotal = 0;
num = new Integer(order.itotal);
text = num.toString();
this.items.setText(text);
}
}

public static void main(String[] args){
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);

    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    RMIClient1 frame = new RMIClient1();
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMI SecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println(messages.getString("nolookup"));
    } catch (java.rmi.RemoteException e){
        System.out.println(messages.getString("nolookup"));
    } catch (java.net.MalformedURLException e) {
        System.out.println(messages.getString("nolookup"));
    }
}
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    static Send send;
    String customer;
    Set s = null;

//Internationalization variables
    static Locale currentLocale;
    static ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    RMIClient2(){ //Begin Constructor

        setTitle(messages.getString("title"));

//Create labels
        creditCard = new JLabel(messages.getString("card"));
        custID = new JLabel(messages.getString("customer"));
        apples = new JLabel(messages.getString("apples"));
        peaches = new JLabel(messages.getString("peaches"));
        pears = new JLabel(messages.getString("pears"));
        total = new JLabel(messages.getString("items"));
        cost = new JLabel(messages.getString("cost"));

//Create text areas
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton(messages.getString("view"));
        view.addActionListener(this);

        reset = new JButton(messages.getString("reset"));
        reset.addActionListener(this);

//Create panel for 2-column layout
//Set white background color
        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);
```

```

//Add components to panel columns
//going left to right and top to bottom
    getContentPane().add(panel);
    panel.add(creditCard);
    panel.add(creditNo);

    panel.add(custID);
    panel.add(customerNo);

    panel.add(apples);
    panel.add(applesNo);

    panel.add(peaches);
    panel.add(peachesNo);

    panel.add(pears);
    panel.add(pearsNo);

    panel.add(total);
    panel.add(itotal);

    panel.add(cost);
    panel.add(icost);

    panel.add(view);
    panel.add(reset);

} //End Constructor

//Create list of customer IDs
public void addCustomer(String custID){
    s.add(custID);
    System.out.println("Customer ID added");
}

//Print customer IDs
public void print(){
    if(s.size()!=0){
        Iterator it = s.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
        System.out.println(s);
    }else{
        System.out.println("No customer IDs available");
    }
}

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String unit, i;
    double cost;
    Double price;
    int items;
    Integer itms;
    DataOrder order = new DataOrder();

//If View button pressed
//Get data from server and display it
    if(source == view){
        try{
            order = send.getOrder();
            creditNo.setText(order.cardnum);
            customerNo.setText(order.custID);

```

```
//Get customer ID and add to list
    addCustomer(order.custID);
    applesNo.setText(order.apples);
    peachesNo.setText(order.peaches);
    pearsNo.setText(order.pears);

//Create number formatter
    numFormat = NumberFormat.getNumberInstance(currentLocale);

    price = new Double(order.icost);
    unit = numFormat.format(price);
    icost.setText(unit);

    itms = new Integer(order.itotal);
    i = numFormat.format(order.itotal);
    itotal.setText(i);
} catch (java.rmi.RemoteException e) {
    System.out.println("Cannot access data in server");
}
}
//Print
    print();
}

//If Reset button pressed
//Clear all fields
    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

public static void main(String[] args){
    if(args.length != 3) {
        language = new String("en");
        country = new String ("US");
        System.out.println("English");
    }else{
        language = new String(args[1]);
        country = new String(args[2]);
        System.out.println(language + country);
    }

    currentLocale = new Locale(language, country);
    messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);

    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    RMIClient2 frame = new RMIClient2();
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
}
```

```
try {
    String name = "//" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println(messages.getString("nolookup"));
} catch (java.rmi.RemoteException e){
    System.out.println(messages.getString("nolookup"));
} catch (java.net.MalformedURLException e) {
    System.out.println(messages.getString("nolookup"));
}
}
}
```

apples=Apples:
peaches=Peaches:
pears=Pears:
items=Total Items:
cost=Total Cost:
card=Credit Card:
customer=Customer ID:

title=Fruit 1.25 Each
1col=Select Items
2col=Specify Quantity

reset=Reset
view=View
purchase=Purchase

invalid=Invalid Value
send=Cannot send data to server
nolookup=Cannot look up remote server object

nodata=No data available
noID=No customer IDs available
noserver=Cannot access data in server

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

import java.applet.Applet;

//Make public
public class RMIEnglishApp extends Applet
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;

//Internationalization variables
    Locale currentLocale;
    ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    public void init(){
        language = new String("en");
        country = new String ("US");

        if(System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        currentLocale = new Locale(language, country);
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        Locale test = messages.getLocale();

        try {
//Path to host where remote Send object is running
            String name = "///kq6py.eng.sun.com/Send";
            send = ((Send) Naming.lookup(name));
        } catch (java.rmi.NotBoundException e) {
            System.out.println(messages.getString("nolookup"));
        } catch (java.rmi.RemoteException e){
            System.out.println(messages.getString("nolookup"));
        } catch (java.net.MalformedURLException e) {
            System.out.println(messages.getString("nolookup"));
        }
    }

//Create left and right column labels
```

```
col1 = new JLabel(messages.getString("1col"));
col2 = new JLabel(messages.getString("2col"));
```

```
//Create labels and text field components
```

```
applechk = new JLabel(" " + messages.getString("apples"));
appleqnt = new JTextField();
appleqnt.addActionListener(this);
```

```
pearchk = new JLabel(" " + messages.getString("pears"));
pearqnt = new JTextField();
pearqnt.addActionListener(this);
```

```
peachchk = new JLabel(" " + messages.getString("peaches"));
peachqnt = new JTextField();
peachqnt.addActionListener(this);
```

```
cardNum = new JLabel(" " + messages.getString("card"));
creditCard = new JTextField();
pearqnt.setNextFocusableComponent(creditCard);
```

```
customer = new JTextField();
custID = new JLabel(" " + messages.getString("customer"));
```

```
//Create labels and text area components
```

```
totalItems = new JLabel(" " + messages.getString("items"));
totalCost = new JLabel(" " + messages.getString("cost"));
```

```
items = new JTextArea();
cost = new JTextArea();
```

```
//Create buttons and make action listeners
```

```
purchase = new JButton(messages.getString("purchase"));
purchase.addActionListener(this);
```

```
reset = new JButton(messages.getString("reset"));
reset.addActionListener(this);
```

```
//Set panel layout to 2-column grid
```

```
//on a white background
```

```
setLayout(new GridLayout(0,2));
setBackground(Color.white);
```

```
//Add components to panel columns
```

```
//going left to right and top to bottom
```

```
add(col1);
add(col2);
```

```
add(applechk);
add(appleqnt);
```

```
add(peachchk);
add(peachqnt);
```

```
add(pearchk);
add(pearqnt);
```

```
add(totalItems);
add(items);
```

```
add(totalCost);
add(cost);
```

```
add(cardNum);
add(creditCard);
```

```

    add(custID);
    add(customer);

```

```

    add(reset);
    add(purchase);

```

```

} //End Constructor

```

```

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

```

```

//If Purchase button pressed . . .

```

```

    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();

```

```

//Calculate total items

```

```

    if(order.apples.length() > 0){

```

```

//Catch invalid number error

```

```

        try{
            applesNo = Integer.valueOf(order.apples);
            order.itotal += applesNo.intValue();
        }catch(java.lang.NumberFormatException e){
            appleqnt.setText(messages.getString("invalid"));
        }

```

```

    } else {
        order.itotal += 0;
    }

```

```

    if(order.peaches.length() > 0){

```

```

//Catch invalid number error

```

```

        try{
            peachesNo = Integer.valueOf(order.peaches);
            order.itotal += peachesNo.intValue();
        }catch(java.lang.NumberFormatException e){
            peachqnt.setText(messages.getString("invalid"));
        }

```

```

    } else {
        order.itotal += 0;
    }

```

```

    if(order.pears.length() > 0){

```

```

//Catch invalid number error

```

```

        try{
            pearsNo = Integer.valueOf(order.pears);
            order.itotal += pearsNo.intValue();
        }catch(java.lang.NumberFormatException e){
            pearqnt.setText(messages.getString("invalid"));
        }

```

```

    } else {
        order.itotal += 0;
    }

```

```

//Create number formatter

```

```

    numFormat = NumberFormat.getNumberInstance(currentLocale);

```

```

//Display running total

```

```
text = numFormat.format(order.itotal);
this.items.setText(text);
```

```
//Calculate and display running cost
order.icost = (order.itotal * 1.25);
text2 = numFormat.format(order.icost);
this.cost.setText(text2);
```

```
try{
    send.sendOrder(order);
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("send"));
}
}
```

```
//If Reset button pressed
```

```
//Clear all fields
```

```
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");
```

```
order.icost = 0;
cost = new Double(order.icost);
text2 = cost.toString();
this.cost.setText(text2);
```

```
order.itotal = 0;
num = new Integer(order.itotal);
text = num.toString();
this.items.setText(text);
```

```
    }
}
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

import java.applet.Applet;

//Make public
public class RMIFrenchApp extends Applet
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, pearchk, peachchk;

    JButton purchase, reset;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;

//Internationalization variables
    Locale currentLocale;
    ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    public void init(){
        language = new String("fr");
        country = new String ("FR");

        if(System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        currentLocale = new Locale(language, country);
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        Locale test = messages.getLocale();

        try {
//Path to host where remote Send object is running
            String name = "///kq6py.eng.sun.com/Send";
            send = ((Send) Naming.lookup(name));
        } catch (java.rmi.NotBoundException e) {
            System.out.println(messages.getString("nolookup"));
        } catch (java.rmi.RemoteException e){
            System.out.println(messages.getString("nolookup"));
        } catch (java.net.MalformedURLException e) {
            System.out.println(messages.getString("nollookup"));
        }
    }

//Create left and right column labels

```

```
col1 = new JLabel(messages.getString("1col"));
col2 = new JLabel(messages.getString("2col"));

//Create labels and text field components
applechk = new JLabel(" " + messages.getString("apples"));
appleqnt = new JTextField();
appleqnt.addActionListener(this);

pearchk = new JLabel(" " + messages.getString("pears"));
pearqnt = new JTextField();
pearqnt.addActionListener(this);

peachchk = new JLabel(" " + messages.getString("peaches"));
peachqnt = new JTextField();
peachqnt.addActionListener(this);

cardNum = new JLabel(" " + messages.getString("card"));
creditCard = new JTextField();
pearqnt.setNextFocusableComponent(creditCard);

customer = new JTextField();
custID = new JLabel(" " + messages.getString("customer"));

//Create labels and text area components
totalItems = new JLabel(" " + messages.getString("items"));
totalCost = new JLabel(" " + messages.getString("cost"));

items = new JTextArea();
cost = new JTextArea();

//Create buttons and make action listeners
purchase = new JButton(messages.getString("purchase"));
purchase.addActionListener(this);

reset = new JButton(messages.getString("reset"));
reset.addActionListener(this);

//Set panel layout to 2-column grid
//on a white background
setLayout(new GridLayout(0,2));
setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
add(col1);
add(col2);

add(applechk);
add(appleqnt);

add(peachchk);
add(peachqnt);

add(pearchk);
add(pearqnt);

add(totalItems);
add(items);

add(totalCost);
add(cost);

add(cardNum);
add(creditCard);
```

```

        add(custID);
        add(customer);

        add(reset);
        add(purchase);

    } //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

//If Purchase button pressed . . .
    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();

//Calculate total items
        if(order.apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(order.apples);
                order.itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.peaches.length() > 0){
//Catch invalid number error
            try{
                peachesNo = Integer.valueOf(order.peaches);
                order.itotal += peachesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                peachqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.pears.length() > 0){
//Catch invalid number error
            try{
                pearsNo = Integer.valueOf(order.pears);
                order.itotal += pearsNo.intValue();
            }catch(java.lang.NumberFormatException e){
                pearqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

//Create number formatter
        numFormat = NumberFormat.getNumberInstance(currentLocale);

//Display running total

```

```
text = numFormat.format(order.itotal);
this.items.setText(text);
```

```
//Calculate and display running cost
order.icost = (order.itotal * 1.25);
text2 = numFormat.format(order.icost);
this.cost.setText(text2);
```

```
try{
    send.sendOrder(order);
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("send"));
}
}
```

```
//If Reset button pressed
```

```
//Clear all fields
```

```
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");
```

```
order.icost = 0;
cost = new Double(order.icost);
text2 = cost.toString();
this.cost.setText(text2);
```

```
order.itotal = 0;
num = new Integer(order.itotal);
text = num.toString();
this.items.setText(text);
```

```
    }
}
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.util.*;
import java.text.*;

import java.applet.Applet;

//Make public
public class RMIGermanApp extends Applet
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;

//Internationalization variables
    Locale currentLocale;
    ResourceBundle messages;
    static String language, country;
    NumberFormat numFormat;

    public void init(){
        language = new String("de");
        country = new String ("DE");

        if(System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        currentLocale = new Locale(language, country);
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        Locale test = messages.getLocale();

        try {
//Path to host where remote Send object is running
            String name = "///kq6py.eng.sun.com/Send";
            send = ((Send) Naming.lookup(name));
        } catch (java.rmi.NotBoundException e) {
            System.out.println(messages.getString("nolookup"));
        } catch (java.rmi.RemoteException e){
            System.out.println(messages.getString("nolookup"));
        } catch (java.net.MalformedURLException e) {
            System.out.println(messages.getString("nollookup"));
        }
    }

//Create left and right column labels

```

```
col1 = new JLabel(messages.getString("1col"));
col2 = new JLabel(messages.getString("2col"));
```

```
//Create labels and text field components
```

```
applechk = new JLabel(" " + messages.getString("apples"));
appleqnt = new JTextField();
appleqnt.addActionListener(this);
```

```
pearchk = new JLabel(" " + messages.getString("pears"));
pearqnt = new JTextField();
pearqnt.addActionListener(this);
```

```
peachchk = new JLabel(" " + messages.getString("peaches"));
peachqnt = new JTextField();
peachqnt.addActionListener(this);
```

```
cardNum = new JLabel(" " + messages.getString("card"));
creditCard = new JTextField();
pearqnt.setNextFocusableComponent(creditCard);
```

```
customer = new JTextField();
custID = new JLabel(" " + messages.getString("customer"));
```

```
//Create labels and text area components
```

```
totalItems = new JLabel(" " + messages.getString("items"));
totalCost = new JLabel(" " + messages.getString("cost"));
```

```
items = new JTextArea();
cost = new JTextArea();
```

```
//Create buttons and make action listeners
```

```
purchase = new JButton(messages.getString("purchase"));
purchase.addActionListener(this);
```

```
reset = new JButton(messages.getString("reset"));
reset.addActionListener(this);
```

```
//Set panel layout to 2-column grid
```

```
//on a white background
```

```
setLayout(new GridLayout(0,2));
setBackground(Color.white);
```

```
//Add components to panel columns
```

```
//going left to right and top to bottom
```

```
add(col1);
add(col2);
```

```
add(applechk);
add(appleqnt);
```

```
add(peachchk);
add(peachqnt);
```

```
add(pearchk);
add(pearqnt);
```

```
add(totalItems);
add(items);
```

```
add(totalCost);
add(cost);
```

```
add(cardNum);
add(creditCard);
```

```
        add(custID);
        add(customer);

        add(reset);
        add(purchase);

    } //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

//If Purchase button pressed . . .
    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();

//Calculate total items
        if(order.apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(order.apples);
                order.itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.peaches.length() > 0){
//Catch invalid number error
            try{
                peachesNo = Integer.valueOf(order.peaches);
                order.itotal += peachesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                peachqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

        if(order.pears.length() > 0){
//Catch invalid number error
            try{
                pearsNo = Integer.valueOf(order.pears);
                order.itotal += pearsNo.intValue();
            }catch(java.lang.NumberFormatException e){
                pearqnt.setText(messages.getString("invalid"));
            }
        } else {
            order.itotal += 0;
        }

//Create number formatter
        numFormat = NumberFormat.getNumberInstance(currentLocale);

//Display running total
```

```
text = numFormat.format(order.itotal);
this.items.setText(text);
```

```
//Calculate and display running cost
order.icost = (order.itotal * 1.25);
text2 = numFormat.format(order.icost);
this.cost.setText(text2);
```

```
try{
    send.sendOrder(order);
} catch (java.rmi.RemoteException e) {
    System.out.println(messages.getString("send"));
}
}
```

```
//If Reset button pressed
```

```
//Clear all fields
```

```
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");
```

```
order.icost = 0;
cost = new Double(order.icost);
text2 = cost.toString();
this.cost.setText(text2);
```

```
order.itotal = 0;
num = new Integer(order.itotal);
text = num.toString();
this.items.setText(text);
```

```
    }
}
}
```

```
import java.io.*;

class DataOrder implements Serializable{

    String apples, peaches, pears, cardnum, custID;
    double icost;
    int itotal;
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, pearchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        pearchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

//Create buttons and make action listeners
        purchase = new JButton("Purchase");
        purchase.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);
```

```

//Create a panel for the components
    panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
    panel.setLayout(new GridLayout(0,2));
    panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
    getContentPane().add(panel);
    panel.add(col1);
    panel.add(col2);

    panel.add(applechk);
    panel.add(appleqnt);

    panel.add(peachchk);
    panel.add(peachqnt);

    panel.add(pearchk);
    panel.add(pearqnt);

    panel.add(totalItems);
    panel.add(items);

    panel.add(totalCost);
    panel.add(cost);

    panel.add(cardNum);
    panel.add(creditCard);

    panel.add(custID);
    panel.add(customer);

    panel.add(reset);
    panel.add(purchase);

} //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String text, text2;
    DataOrder order = new DataOrder();

//If Purchase button pressed . . .
    if(source == purchase){
//Get data from text fields
        order.cardnum = creditCard.getText();
        order.custID = customer.getText();
        order.apples = appleqnt.getText();
        order.peaches = peachqnt.getText();
        order.pears = pearqnt.getText();

//Calculate total items
        if(order.apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(order.apples);
                order.itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText("Invalid Value");
            }
        }
    }
}

```

```
    }
    } else {
        order.itotal += 0;
    }

    if(order.peaches.length() > 0){
//Catch invalid number error
        try{
            peachesNo = Integer.valueOf(order.peaches);
            order.itotal += peachesNo.intValue();
        }catch(java.lang.NumberFormatException e){
            peachqnt.setText("Invalid Value");
        }
    } else {
        order.itotal += 0;
    }

    if(order.pears.length() > 0){
//Catch invalid number error
        try{
            pearsNo = Integer.valueOf(order.pears);
            order.itotal += pearsNo.intValue();
        }catch(java.lang.NumberFormatException e){
            pearqnt.setText("Invalid Value");
        }
    } else {
        order.itotal += 0;
    }

//Display running total
    num = new Integer(order.itotal);
    text = num.toString();
    this.items.setText(text);

//Calculate and display running cost
    order.icost = (order.itotal * 1.25);
    cost = new Double(order.icost);
    text2 = cost.toString();
    this.cost.setText(text2);

    try{
        send.sendOrder(order);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot send data to server");
    }
}

//If Reset button pressed
//Clear all fields
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");

    order.icost = 0;
    cost = new Double(order.icost);
    text2 = cost.toString();
    this.cost.setText(text2);

    order.itotal = 0;
    num = new Integer(order.itotal);
    text = num.toString();
```

```
        this.items.setText(text);
    }
}

public static void main(String[] args){
    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "//" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot look up remote server object");
    } catch (java.rmi.RemoteException e){
        System.out.println("Cannot look up remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up remote server object");
    }
}
}
```

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Send extends Remote {  
    public void sendOrder(DataOrder order) throws RemoteException;  
    public DataOrder getOrder() throws RemoteException;  
}
```

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RemoteServer extends UnicastRemoteObject
    implements Send {

    Integer num = null;
    String orders = null;
    int value = 0, get = 0;

    public RemoteServer() throws RemoteException {
        super();
    }

    public void sendOrder(DataOrder order){

        value += 1;
        num = new Integer(value);
        orders = num.toString();
        try{
            FileOutputStream fos = new FileOutputStream(orders);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(order);
        }catch (java.io.FileNotFoundException e){
            System.out.println("File not found");
        }catch (java.io.IOException e){
            System.out.println("Unable to write to file");
        }
    }

    public DataOrder getOrder(){

        DataOrder order = null;

        if(value == 0){
            System.out.println("No Orders To Process");
        }

        if(value > get){
            get += 1;
            num = new Integer(get);
            orders = num.toString();
            try{
                FileInputStream fis = new FileInputStream(orders);
                ObjectInputStream ois = new ObjectInputStream(fis);
                order = (DataOrder)ois.readObject();
            }catch (java.io.FileNotFoundException e){
                System.out.println("File not found");
            }catch (java.io.IOException e){
                System.out.println("Unable to read file");
            }catch (java.lang.ClassNotFoundException e){
                System.out.println("No data available");
            }
        }else{
            System.out.println("No Orders To Process");
        }
        return order;
    }
}
```

```
public static void main(String[] args){
    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    String name = "//kq6py.eng.sun.com/Send";
    try {
        Send remoteServer = new RemoteServer();
        Naming.rebind(name, remoteServer);
        System.out.println("RemoteServer bound");
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up server object");
    }
}
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.io.FileInputStream.*;
import java.io.RandomAccessFile.*;
import java.io.File;

import java.util.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    static Send send;
    String customer;
    Set s = null;

    RMIClient2(){ //Begin Constructor
//Create labels
        creditCard = new JLabel("Credit Card:");
        custID = new JLabel("Customer ID:");
        apples = new JLabel("Apples:");
        peaches = new JLabel("Peaches:");
        pears = new JLabel("Pears:");
        total = new JLabel("Total Items:");
        cost = new JLabel("Total Cost:");

//Create text areas
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton("View Order");
        view.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create panel for 2-column layout
//Set white background color
        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(creditCard);
        panel.add(creditNo);
```

```

panel.add(custID);
panel.add(customerNo);

```

```

panel.add(apples);
panel.add(applesNo);

```

```

panel.add(peaches);
panel.add(peachesNo);

```

```

panel.add(pears);
panel.add(pearsNo);

```

```

panel.add(total);
panel.add(itotal);

```

```

panel.add(cost);
panel.add(icost);

```

```

panel.add(view);
panel.add(reset);

```

```

} //End Constructor

```

```

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String unit, i;
    double cost;
    Double price;
    int items;
    Integer itms;
    DataOrder order = new DataOrder();

```

```

//If View button pressed

```

```

//Get data from server and display it

```

```

    if(source == view){
        try{
            order = send.getOrder();
            creditNo.setText(order.cardnum);
            customerNo.setText(order.custID);
            applesNo.setText(order.apples);
            peachesNo.setText(order.peaches);
            pearsNo.setText(order.pears);

            cost = order.icost;
            price = new Double(cost);
            unit = price.toString();
            icost.setText(unit);

            items = order.itotal;
            itms = new Integer(items);
            i = itms.toString();
            itotal.setText(i);
        } catch (java.rmi.RemoteException e) {
            System.out.println("Cannot access data in server");
        }
    }
}

```

```

//If Reset button pressed

```

```

//Clear all fields

```

```

    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
    }
}

```

```
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

public static void main(String[] args){
    RMIClient2 frame = new RMIClient2();
    frame.setTitle("Fruit Order");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up server object");
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot access data in server");
    }
}
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
//import javax.swing.*;
import com.sun.java.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import javax.crypto.*;
import java.security.*;
import java.security.interfaces.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, pearchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;
    int itotal=0;
    double icost=0;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        pearchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

```

```

//Create buttons and make action listeners
purchase = new JButton("Purchase");
purchase.addActionListener(this);

reset = new JButton("Reset");
reset.addActionListener(this);
creditCard.setNextFocusableComponent(reset);

//Create a panel for the components
panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
panel.setLayout(new GridLayout(0,2));
panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
getContentPane().add(panel);
panel.add(col1);
panel.add(col2);

panel.add(applechk);
panel.add(appleqnt);

panel.add(peachchk);
panel.add(peachqnt);

panel.add(pearchk);
panel.add(pearqnt);

panel.add(totalItems);
panel.add(items);

panel.add(totalCost);
panel.add(cost);

panel.add(cardNum);
panel.add(creditCard);

panel.add(custID);
panel.add(customer);

panel.add(reset);
panel.add(purchase);

} //End Constructor

/*
private void encrypt(credit card number){
// Create cipher for symmetric key encryption (DES)
// Create a key generator
// Create a secret (session) key with key generator
// Initialize cipher for encryption with session key
// Encrypt credit card number with cipher
// Get public key from server
// Create cipher for asymmetric encryption (so not use RSA)
// Initialize cipher for encryption with public key
// Seal session key using asymmetric cipher
// Serialize sealed session key
// Send encrypted credit card number and
// sealed session key to server
}
*/

```

```

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String number, cardnum, custID, apples, peaches, pears, text, text2;

//If Purchase button pressed
    if(source == purchase){
//Get data from text fields
        cardnum = creditCard.getText();
        custID = customer.getText();
        apples = appleqnt.getText();
        peaches = peachqnt.getText();
        pears = pearqnt.getText();

//Calculate total items
        if(apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(apples);
                itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText("Invalid Value");
            }
        } else {
            itotal += 0;
        }

        if(peaches.length() > 0){
//Catch invalid number error
            try{
                peachesNo = Integer.valueOf(peaches);
                itotal += peachesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                peachqnt.setText("Invalid Value");
            }
        } else {
            itotal += 0;
        }

        if(pears.length() > 0){
//Catch invalid number error
            try{
                pearsNo = Integer.valueOf(pears);
                itotal += pearsNo.intValue();
            }catch(java.lang.NumberFormatException e){
                pearqnt.setText("Invalid Value");
            }
        } else {
            itotal += 0;
        }

//Display running total
        num = new Integer(itotal);
        text = num.toString();
        this.items.setText(text);

//Calculate and display running cost
        icost = (itotal * 1.25);
        cost = new Double(icost);
        text2 = cost.toString();
        this.cost.setText(text2);

//Encrypt credit card number

```

```

        encrypt(cardnum);

//Send data over net
    try{
        send.sendCustID(custID);
        send.sendAppleQnt(apples);
        send.sendPeachQnt(peaches);
        send.sendPearQnt(pears);
        send.sendTotalCost(icost);
        send.sendTotalItems(itotal);
    } catch (java.rmi.RemoteException e) {
        System.out.println("sendData exception: " + e.getMessage());
    }
}

//If Reset button pressed
//Clear all fields
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");

    icost = 0;
    cost = new Double(icost);
    text2 = cost.toString();
    this.cost.setText(text2);

    itotal = 0;
    num = new Integer(itotal);
    text = num.toString();
    this.items.setText(text);
}
}

public static void main(String[] args){
    try{
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
    }catch (Exception e) {
        System.out.println("Couldn't use the cross-platform"
            + "look and feel: " + e);
    }

    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    }
}

```

```
    } catch (Exception e) {  
        System.out.println("RMIClient1 exception: " + e.getMessage());  
        e.printStackTrace();  
    }  
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
//import javax.swing.*;
import com.sun.java.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.io.FileInputStream.*;
import java.io.RandomAccessFile.*;
import java.io.File;

import java.util.*;

import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    Socket socket = null;
    PrintWriter out = null;
    static Send send;
    String customer;

    RMIClient2(){ //Begin Constructor
//Create labels
        creditCard = new JLabel("Credit Card:");
        custID = new JLabel("Customer ID:");
        apples = new JLabel("Apples:");
        peaches = new JLabel("Peaches:");
        pears = new JLabel("Pears:");
        total = new JLabel("Total Items:");
        cost = new JLabel("Total Cost:");

//Create text area components
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton("View Order");
        view.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create panel for 2-column layout
//Set white background color
        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);
```

```

//Add components to panel columns
//going left to right and top to bottom
    getContentPane().add(panel);
    panel.add(creditCard);
    panel.add(creditNo);

    panel.add(custID);
    panel.add(customerNo);

    panel.add(apples);
    panel.add(applesNo);

    panel.add(peaches);
    panel.add(peachesNo);

    panel.add(pears);
    panel.add(pearsNo);

    panel.add(total);
    panel.add(itotal);

    panel.add(cost);
    panel.add(icost);

    panel.add(view);
    panel.add(reset);

} //End Constructor

/*
public String decrypt(sealed key,
                    encrypted credit card number){
    // Get private key from file
    // Create asymmetric cipher (do not use RSA)
    // Initialize cipher for decryption with private key
    // Unseal wrapped session key using asymmetric cipher
    // Create symmetric cipher
    // Initialize cipher for decryption with session key
    // Decrypt credit card number with symmetric cipher
}
*/

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String text=null, unit, i;
    byte[] wrappedKey;
    byte[] encrypted;
    double cost;
    Double price;
    int items;
    Integer itms;

//If View button pressed
//Get data from server and display it
    if(source == view){
        try{
            wrappedKey = send.getKey();
            encrypted = send.getEncrypted();
//Decrypt credit card number
            String credit = decrypt(sealed, encrypted);
            creditNo.setText(new String(credit));

            text = send.getCustID();
            customerNo.setText(text);

```

```

    text = send.getAppleQnt();
    applesNo.setText(text);

```

```

    text = send.getPeachQnt();
    peachesNo.setText(text);

```

```

    text = send.getPearQnt();
    pearsNo.setText(text);

```

```

    cost = send.getTotalCost();
    price = new Double(cost);
    unit = price.toString();
    icost.setText(unit);

```

```

    items = send.getTotalItems();
    itms = new Integer(items);
    i = itms.toString();
    itotal.setText(i);

```

```

    } catch (java.rmi.RemoteException e) {
        System.out.println("sendData exception: " + e.getMessage());
    }
}

```

```
//If Reset button pressed
```

```
//Clear all fields
```

```

    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

```

```

public static void main(String[] args){
    RMIClient2 frame = new RMIClient2();
    frame.setTitle("Fruit Order");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
}

```

```

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

```

```

if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}

```

```

try {
    String name = "/" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
    System.out.println("Cannot access data in server");
} catch (java.rmi.RemoteException e){
    System.out.println("Cannot access data in server");
} catch (java.net.MalformedURLException e) {
    System.out.println("Cannot access data in server");
}
}

```

```
}  
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
//import javax.swing.*;
import com.sun.java.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import javax.crypto.*;
import java.security.*;
import java.security.interfaces.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, pearchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;
    int itotal=0;
    double icost=0;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        pearchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

```

```

//Create buttons and make action listeners
purchase = new JButton("Purchase");
purchase.addActionListener(this);

reset = new JButton("Reset");
reset.addActionListener(this);
creditCard.setNextFocusableComponent(reset);

//Create a panel for the components
panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
panel.setLayout(new GridLayout(0,2));
panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
getContentPane().add(panel);
panel.add(col1);
panel.add(col2);

panel.add(applechk);
panel.add(appleqnt);

panel.add(peachchk);
panel.add(peachqnt);

panel.add(pearchk);
panel.add(pearqnt);

panel.add(totalItems);
panel.add(items);

panel.add(totalCost);
panel.add(cost);

panel.add(cardNum);
panel.add(creditCard);

panel.add(custID);
panel.add(customer);

panel.add(reset);
panel.add(purchase);

} //End Constructor

private void encrypt(credit card number){
//Create cipher for symmetric key encryption (DES)
//Create a key generator
//Create a secret (session) key with key generator
//Initialize cipher for encryption with session key
//Encrypt credit card number with cipher
//Get public key from server
//Create cipher for asymmetric encryption (RSA)
//Initialize cipher for encryption with public key
//Encrypt session key
//Send encrypted credit card number and session key to server
}

public void actionPerformed(ActionEvent event){
Object source = event.getSource();
Integer applesNo, peachesNo, pearsNo, num;

```

```

Double cost;
String number, cardnum, custID, apples, peaches, pears, text, text2;

```

```

if(source == purchase){
//Retrieve order information
    cardnum = creditCard.getText();
    custID = customer.getText();
    apples = appleqnt.getText();
    peaches = peachqnt.getText();
    pears = pearqnt.getText();

//Calculate totals
    number = appleqnt.getText();
    if(number.length() > 0){
        try{
            applesNo = Integer.valueOf(number);
            itotal += applesNo.intValue();
        } catch (java.lang.NumberFormatException e) {
            appleqnt.setText("Invalid Value");
        }
    } else {
        itotal += 0;
    }

    number = peachqnt.getText();
    if(number.length() > 0){
        try{
            peachesNo = Integer.valueOf(number);
            itotal += peachesNo.intValue();
        } catch (java.lang.NumberFormatException e) {
            peachqnt.setText("Invalid Value");
        }
    } else {
        itotal += 0;
    }

    number = pearqnt.getText();
    if(number.length() > 0){
        try{
            pearsNo = Integer.valueOf(number);
            itotal += pearsNo.intValue();
        } catch (java.lang.NumberFormatException e) {
            pearqnt.setText("Invalid Value");
        }
    } else {
        itotal += 0;
    }

    num = new Integer(itotal);
    text = num.toString();
    this.items.setText(text);

    icost = (itotal * 1.25);
    cost = new Double(icost);
    text2 = cost.toString();
    this.cost.setText(text2);

//Encrypt credit card number
    encrypt(cardnum);

    try{
        send.sendCustID(custID);
        send.sendAppleQnt(apples);
        send.sendPeachQnt(peaches);
        send.sendPearQnt(pears);
    }

```

```

        send.sendTotalCost(icost);
        send.sendTotalItems(itotal);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot send data to server");
    }
}

if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");

    icost = 0;
    cost = new Double(icost);
    text2 = cost.toString();
    this.cost.setText(text2);

    itotal = 0;
    num = new Integer(itotal);
    text = num.toString();
    this.items.setText(text);
}
}

public static void main(String[] args){
    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMI SecurityManager());
    }

    try {
        String name = "://" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot look up remote server object");
    } catch (java.rmi.RemoteException e){
        System.out.println("Cannot look up remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up remote server object");
    }
}
}
}

```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
//import javax.swing.*;
import com.sun.java.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.io.FileInputStream.*;
import java.io.RandomAccessFile.*;
import java.io.File;

import java.util.*;

import javax.crypto.*;
import java.security.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    Socket socket = null;
    PrintWriter out = null;
    static Send send;
    String customer;

    RMIClient2(){ //Begin Constructor
        creditCard = new JLabel("Credit Card:");
        custID = new JLabel("Customer ID:");
        apples = new JLabel("Apples:");
        peaches = new JLabel("Peaches:");
        pears = new JLabel("Pears:");
        total = new JLabel("Total Items:");
        cost = new JLabel("Total Cost:");

        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

        view = new JButton("View Order");
        view.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);

        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);
        getContentPane().add(panel);

        panel.add(creditCard);
        panel.add(creditNo);
```

```

    panel.add(custID);
    panel.add(customerNo);

    panel.add(apples);
    panel.add(applesNo);

    panel.add(peaches);
    panel.add(peachesNo);

    panel.add(pears);
    panel.add(pearsNo);

    panel.add(total);
    panel.add(itotal);

    panel.add(cost);
    panel.add(icost);

    panel.add(view);
    panel.add(reset);

} //End Constructor

public void writeToFile(String custID) throws java.io.IOException{
    byte b[] = custID.getBytes();
    File outputFile = new File(File.separatorChar +
        "home" + File.separatorChar +
        "monicap" + File.separatorChar +
        "text.txt");
    RandomAccessFile out = new RandomAccessFile(outputFile, "rw");
    out.seek(outputFile.length());
    out.write(b);
    out.writeByte(10);
    out.close();
}

public String decrypt(symmetric key,
                    encrypted credit card number){
    //Decrypt credit card number
    //Get private key from file
    //Create asymmetric cipher (RSA)
    //Initialize cipher for decryption with private key
    //Decrypt symmetric key
    //Instantiate symmetric key
    //Create symmetric cipher
    //Initialize Cipher for decryption with session key
    //Decrypt credit card number with symmetric Cipher
}

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    String text=null, unit, i;
    SealedObject sealed;
    byte[] encrypted;
    double cost;
    Double price;
    int items;
    Integer itms;

    if(source == view){
        try{
            sealed = send.getCreditCard();
            encrypted = send.getEncrypted();
            String credit = decrypt(sealed, encrypted);
            creditNo.setText(credit);
        }
    }
}

```

```

        text = send.getCustID();
        customerNo.setText(text);
        writeToFile(text);

        text = send.getAppleQnt();
        applesNo.setText(text);

        text = send.getPeachQnt();
        peachesNo.setText(text);

        text = send.getPearQnt();
        pearsNo.setText(text);

        cost = send.getTotalCost();
        price = new Double(cost);
        unit = price.toString();
        icost.setText(unit);

        items = send.getTotalItems();
        itms = new Integer(items);
        i = itms.toString();
        itotal.setText(i);

    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot access data in server");
    }
}

if(source == reset){
    creditNo.setText("");
    customerNo.setText("");
    applesNo.setText("");
    peachesNo.setText("");
    pearsNo.setText("");
    itotal.setText("");
    icost.setText("");
}
}

public static void main(String[] args){
    RMIClient2 frame = new RMIClient2();
    frame.setTitle("Fruit Order");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up server object");
    } catch (java.rmi.NotBoundException e) {

```

```
        System.out.println("Cannot access data in server");  
    }  
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;
    int itotal=0;
    double icost=0;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        pearchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

//Create buttons and make action listeners
        purchase = new JButton("Purchase");
        purchase.addActionListener(this);

```

```
        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create a panel for the components
        panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(col1);
        panel.add(col2);

        panel.add(applechk);
        panel.add(appleqnt);

        panel.add(peachchk);
        panel.add(peachqnt);

        panel.add(pearchk);
        panel.add(pearqnt);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);

    } //End Constructor

    public void actionPerformed(ActionEvent event){
        Object source = event.getSource();
        Integer applesNo, peachesNo, pearsNo, num;
        Double cost;
        String number, cardnum, custID, apples, peaches, pears, text, text2;

//If Purchase button pressed . . .
        if(source == purchase){
//Get data from text fields
            cardnum = creditCard.getText();
            custID = customer.getText();
            apples = appleqnt.getText();
            peaches = peachqnt.getText();
            pears = pearqnt.getText();
            try{
//Send data over net
                send.sendCreditCard(cardnum);
                send.sendCustID(custID);
                send.sendAppleQnt(apples);
                send.sendPeachQnt(peaches);
                send.sendPearQnt(pears);
                send.sendTotalCost(icost);
```

```
        send.sendTotalItems(itotal);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot send data to server");
    }
}

//If Reset button pressed
//Clear all fields
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");
    icost = 0;
    itotal = 0;
}

//If Return in apple quantity text field
//Calculate totals
if(source == appleqnt){
    number = appleqnt.getText();
    if(number.length() > 0){
        applesNo = Integer.valueOf(number);
        itotal += applesNo.intValue();
    } else {
        itotal += 0;
    }
}

//If Return in peach quantity text field
//Calculate totals
if(source == peachqnt){
    number = peachqnt.getText();
    if(number.length() > 0){
        peachesNo = Integer.valueOf(number);
        itotal += peachesNo.intValue();
    } else {
        itotal += 0;
    }
}

//If Return in pear quantity text field
//Calculate totals
if(source == pearqnt){
    number = pearqnt.getText();
    if(number.length() > 0){
        pearsNo = Integer.valueOf(number);
        itotal += pearsNo.intValue();
    } else {
        itotal += 0;
    }
}

num = new Integer(itotal);
text = num.toString();
this.items.setText(text);

icost = (itotal * 1.25);
cost = new Double(icost);
text2 = cost.toString();
this.cost.setText(text2);
}
```

```
public static void main(String[] args){
    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "//" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot look up remote server object");
    } catch (java.rmi.RemoteException e){
        System.out.println("Cannot look up remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up remote server object");
    }
}
}
```

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RemoteServer extends UnicastRemoteObject
    implements Send {

    private String cardnum;
    private String custID;
    private String apples;
    private String peaches;
    private String pears;
    private int items;
    private double cost;

    public RemoteServer() throws RemoteException {
        super();
    }

    public void sendCreditCard(String creditcard){
        cardnum = creditcard;
    }

    public String getCreditCard(){
        return cardnum;
    }

    public void sendCustID(String cust){
        custID = cust;
    }
    public String getCustID(){
        return custID;
    }

    public void sendAppleQnt(String apps){
        apples = apps;
    }
    public String getAppleQnt(){
        return apples;
    }

    public void sendPeachQnt(String pchs){
        peaches = pchs;
    }
    public String getPeachQnt(){
        return peaches;
    }

    public void sendPearQnt(String prs){
        pears = prs;
    }
    public String getPearQnt(){
        return pears;
    }

    public void sendTotalCost(double cst){
        cost = cst;
    }
}
```

```
public double getTotalCost(){
    return cost;
}

public void sendTotalItems(int itm){
    items = itm;
}
public int getTotalItems(){
    return items;
}

public static void main(String[] args){
    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    String name = "//kq6py.eng.sun.com/Send";
    try {
        Send remoteServer = new RemoteServer();
        Naming.rebind(name, remoteServer);
        System.out.println("RemoteServer bound");
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up server object");
    }
}
}
```

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

import java.io.FileInputStream.*;
import java.io.RandomAccessFile.*;
import java.io.File;

import java.util.*;

class RMIClient2 extends JFrame
    implements ActionListener {

    JLabel creditCard, custID, apples, peaches, pears, total, cost, clicked;
    JButton view, reset;
    JPanel panel;
    JTextArea creditNo, customerNo, applesNo, peachesNo, pearsNo, itotal, icost;
    static Send send;
    String customer;

    RMIClient2(){ //Begin Constructor
//Create labels
        creditCard = new JLabel("Credit Card:");
        custID = new JLabel("Customer ID:");
        apples = new JLabel("Apples:");
        peaches = new JLabel("Peaches:");
        pears = new JLabel("Pears:");
        total = new JLabel("Total Items:");
        cost = new JLabel("Total Cost:");

//Create text area components
        creditNo = new JTextArea();
        customerNo = new JTextArea();
        applesNo = new JTextArea();
        peachesNo = new JTextArea();
        pearsNo = new JTextArea();
        itotal = new JTextArea();
        icost = new JTextArea();

//Create buttons
        view = new JButton("View Order");
        view.addActionListener(this);

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create panel for 2-column layout
//Set white background color
        panel = new JPanel();
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(creditCard);
        panel.add(creditNo);
```

```

panel.add(custID);
panel.add(customerNo);

```

```

panel.add(apples);
panel.add(applesNo);

```

```

panel.add(peaches);
panel.add(peachesNo);

```

```

panel.add(pears);
panel.add(pearsNo);

```

```

panel.add(total);
panel.add(itotal);

```

```

panel.add(cost);
panel.add(icost);

```

```

panel.add(view);
panel.add(reset);

```

```

} //End Constructor

```

```

public void actionPerformed(ActionEvent event){

```

```

    Object source = event.getSource();

```

```

    String text=null, unit, i;

```

```

    double cost;

```

```

    Double price;

```

```

    int items;

```

```

    Integer itms;

```

```

//If View button pressed

```

```

//Get data from server and display it

```

```

    if(source == view){

```

```

        try{

```

```

            text = send.getCreditCard();

```

```

            creditNo.setText(text);

```

```

            text = send.getCustID();

```

```

            customerNo.setText(text);

```

```

            text = send.getAppleQnt();

```

```

            applesNo.setText(text);

```

```

            text = send.getPeachQnt();

```

```

            peachesNo.setText(text);

```

```

            text = send.getPearQnt();

```

```

            pearsNo.setText(text);

```

```

            cost = send.getTotalCost();

```

```

            price = new Double(cost);

```

```

            unit = price.toString();

```

```

            icost.setText(unit);

```

```

            items = send.getTotalItems();

```

```

            itms = new Integer(items);

```

```

            i = itms.toString();

```

```

            itotal.setText(i);

```

```

        } catch (java.rmi.RemoteException e) {

```

```

            System.out.println("Cannot access data in server");

```

```

        }

```

```

    }

```

```

//If Reset button pressed . . .

```

```
//Clear all fields
    if(source == reset){
        creditNo.setText("");
        customerNo.setText("");
        applesNo.setText("");
        peachesNo.setText("");
        pearsNo.setText("");
        itotal.setText("");
        icost.setText("");
    }
}

public static void main(String[] args){
    RMIClient2 frame = new RMIClient2();
    frame.setTitle("Fruit Order");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "/" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot access data in server");
    } catch (java.rmi.RemoteException e){
        System.out.println("Cannot access data in server");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot access data in server");
    }
}
}
```

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Send extends Remote {

    public void sendCreditCard(String cardnum) throws RemoteException;
    public String getCreditCard() throws RemoteException;

    public void sendCustID(String cust) throws RemoteException;
    public String getCustID() throws RemoteException;

    public void sendAppleQnt(String apples) throws RemoteException;
    public String getAppleQnt() throws RemoteException;

    public void sendPeachQnt(String peaches) throws RemoteException;
    public String getPeachQnt() throws RemoteException;

    public void sendPearQnt(String pears) throws RemoteException;
    public String getPearQnt() throws RemoteException;

    public void sendTotalCost(double cost) throws RemoteException;
    public double getTotalCost() throws RemoteException;

    public void sendTotalItems(int items) throws RemoteException;
    public int getTotalItems() throws RemoteException;
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;
    int itotal=0;
    double icost=0;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        pearchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

//Create buttons and make action listeners
        purchase = new JButton("Purchase");
        purchase.addActionListener(this);

```

```

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create a panel for the components
        panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(col1);
        panel.add(col2);

        panel.add(applechk);
        panel.add(appleqnt);

        panel.add(peachchk);
        panel.add(peachqnt);

        panel.add(pearchk);
        panel.add(pearqnt);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);

    } //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String cardnum, custID, apples, peaches, pears, text, text2;

//If Purchase button pressed . . .
    if(source == purchase){
        cardnum = creditCard.getText();
        custID = customer.getText();
        apples = appleqnt.getText();
        peaches = peachqnt.getText();
        pears = pearqnt.getText();

//Calculate total items
        if(apples.length() > 0){
            applesNo = Integer.valueOf(apples);
            itotal += applesNo.intValue();
        } else {
            itotal += 0;
        }
    }
}

```

```
if(peaches.length() > 0){
    peachesNo = Integer.valueOf(peaches);
    itotal += peachesNo.intValue();
} else {
    itotal += 0;
}
```

```
if(pears.length() > 0){
    pearsNo = Integer.valueOf(pears);
    itotal += pearsNo.intValue();
} else {
    itotal += 0;
}
```

```
//Display running total
num = new Integer(itotal);
text = num.toString();
this.items.setText(text);
```

```
//Calculate and display running cost
icost = (itotal * 1.25);
cost = new Double(icost);
text2 = cost.toString();
this.cost.setText(text2);
```

```
//Send data over net
try{
    send.sendCreditCard(cardnum);
    send.sendCustID(custID);
    send.sendAppleQnt(apples);
    send.sendPeachQnt(peaches);
    send.sendPearQnt(pears);
    send.sendTotalCost(icost);
    send.sendTotalItems(itotal);
} catch (java.rmi.RemoteException e) {
    System.out.println("Cannot send data to server");
}
}
```

```
//If Reset button pressed
//Clear all fields
if(source == reset){
    creditCard.setText("");
    appleqnt.setText("");
    peachqnt.setText("");
    pearqnt.setText("");
    creditCard.setText("");
    customer.setText("");

    icost = 0;
    cost = new Double(icost);
    text2 = cost.toString();
    this.cost.setText(text2);

    itotal = 0;
    num = new Integer(itotal);
    text = num.toString();
    this.items.setText(text);
}
}
```

```
public static void main(String[] args){
    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
```

```
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
};
```

```
frame.addWindowListener(l);  
frame.pack();  
frame.setVisible(true);
```

```
if(System.getSecurityManager() == null) {  
    System.setSecurityManager(new RMISecurityManager());  
}
```

```
try {  
    String name = "//" + args[0] + "/Send";  
    send = ((Send) Naming.lookup(name));  
} catch (java.rmi.NotBoundException e) {  
    System.out.println("Cannot look up remote server object");  
} catch (java.rmi.RemoteException e){  
    System.out.println("Cannot look up remote server object");  
} catch (java.net.MalformedURLException e) {  
    System.out.println("Cannot look up remote server object");  
}  
}  
}
```

```

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient1 extends JFrame
    implements ActionListener{

    JLabel col1, col2;
    JLabel totalItems, totalCost;
    JLabel cardNum, custID;
    JLabel applechk, peachchk, peachchk;

    JButton purchase, reset;
    JPanel panel;

    JTextField appleqnt, pearqnt, peachqnt;
    JTextField creditCard, customer;
    JTextArea items, cost;

    static Send send;
    int itotal=0;
    double icost=0;

    RMIClient1(){ //Begin Constructor
//Create left and right column labels
        col1 = new JLabel("Select Items");
        col2 = new JLabel("Specify Quantity");

//Create labels and text field components
        applechk = new JLabel(" Apples");
        appleqnt = new JTextField();
        appleqnt.addActionListener(this);

        peachchk = new JLabel(" Pears");
        pearqnt = new JTextField();
        pearqnt.addActionListener(this);

        peachchk = new JLabel(" Peaches");
        peachqnt = new JTextField();
        peachqnt.addActionListener(this);

        cardNum = new JLabel(" Credit Card:");
        creditCard = new JTextField();
        pearqnt.setNextFocusableComponent(creditCard);

        customer = new JTextField();
        custID = new JLabel(" Customer ID:");

//Create labels and text area components
        totalItems = new JLabel("Total Items:");
        totalCost = new JLabel("Total Cost:");
        items = new JTextArea();
        cost = new JTextArea();

//Create buttons and make action listeners
        purchase = new JButton("Purchase");
        purchase.addActionListener(this);

```

```

        reset = new JButton("Reset");
        reset.addActionListener(this);

//Create a panel for the components
        panel = new JPanel();

//Set panel layout to 2-column grid
//on a white background
        panel.setLayout(new GridLayout(0,2));
        panel.setBackground(Color.white);

//Add components to panel columns
//going left to right and top to bottom
        getContentPane().add(panel);
        panel.add(col1);
        panel.add(col2);

        panel.add(applechk);
        panel.add(appleqnt);

        panel.add(peachchk);
        panel.add(peachqnt);

        panel.add(pearchk);
        panel.add(pearqnt);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);

    } //End Constructor

public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    Integer applesNo, peachesNo, pearsNo, num;
    Double cost;
    String number, cardnum, custID, apples, peaches, pears, text, text2;

//If Purchase button pressed . . .
    if(source == purchase){
        cardnum = creditCard.getText();
        custID = customer.getText();
        apples = appleqnt.getText();
        peaches = peachqnt.getText();
        pears = pearqnt.getText();

//Calculate total items
        if(apples.length() > 0){
//Catch invalid number error
            try{
                applesNo = Integer.valueOf(apples);
                itotal += applesNo.intValue();
            }catch(java.lang.NumberFormatException e){
                appleqnt.setText("Invalid Value");
            }
        }
    }
}

```

```
    }
} else {
    itotal += 0;
}

if(peaches.length() > 0){
//Catch invalid number error
    try{
        peachesNo = Integer.valueOf(peaches);
        itotal += peachesNo.intValue();
    }catch(java.lang.NumberFormatException e){
        peachqnt.setText("Invalid Value");
    }
} else {
    itotal += 0;
}

if(pears.length() > 0){
//Catch invalid number error
    try{
        pearsNo = Integer.valueOf(pears);
        itotal += pearsNo.intValue();
    }catch(java.lang.NumberFormatException e){
        pearqnt.setText("Invalid Value");
    }
} else {
    itotal += 0;
}

//Display running total
    num = new Integer(itotal);
    text = num.toString();
    this.items.setText(text);

//Calculate and display running cost
    icost = (itotal * 1.25);
    cost = new Double(icost);
    text2 = cost.toString();
    this.cost.setText(text2);

//Send data over net
    try{
        send.sendCreditCard(cardnum);
        send.sendCustID(custID);
        send.sendAppleQnt(apples);
        send.sendPeachQnt(peaches);
        send.sendPearQnt(pears);
        send.sendTotalCost(icost);
        send.sendTotalItems(itotal);
    } catch (java.rmi.RemoteException e) {
        System.out.println("Unable to send data");
    }
}

//If Reset button pressed
//Clear all Fields
    if(source == reset){
        creditCard.setText("");
        appleqnt.setText("");
        peachqnt.setText("");
        pearqnt.setText("");
        creditCard.setText("");
        customer.setText("");

        icost = 0;
    }
}
```

```
cost = new Double(icost);
text2 = cost.toString();
this.cost.setText(text2);

itotal = 0;
num = new Integer(itotal);
text = num.toString();
this.items.setText(text);
```

```
    }
}

public static void main(String[] args){
    RMIClient1 frame = new RMIClient1();
    frame.setTitle("Fruit $1.25 Each");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);

    if(System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    try {
        String name = "://" + args[0] + "/Send";
        send = ((Send) Naming.lookup(name));
    } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
        System.out.println("Cannot look up server object");
    } catch (java.rmi.NotBoundException e) {
        System.out.println("Cannot access data in server");
    }
}
}
```

```

import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

class SocketClient extends JFrame
    implements ActionListener {

    JLabel text, clicked;
    JButton button;
    JPanel panel;
    JTextField textField;
    Socket socket = null;
    PrintWriter out = null;
    BufferedReader in = null;

    SocketClient() { //Begin Constructor
        text = new JLabel("Text to send over socket:");
        textField = new JTextField(20);
        button = new JButton("Click Me");
        button.addActionListener(this);

        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(Color.white);
        getContentPane().add(panel);
        panel.add("North", text);
        panel.add("Center", textField);
        panel.add("South", button);
    } //End Constructor

    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();

        if(source == button) {
//Send data over socket
            String text = textField.getText();
            out.println(text);
            textField.setText(new String(""));
//Receive text from server
            try {
                String line = in.readLine();
                System.out.println("Text received : " + line);
            } catch (IOException e) {
                System.out.println("Read failed");
                System.exit(1);
            }
        }
    }

    public void listenSocket() {
//Create socket connection
        try {
            socket = new Socket("kq6py", 4444);
            out = new PrintWriter(socket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        } catch (UnknownHostException e) {
            System.out.println("Unknown host: kq6py.eng");
            System.exit(1);
        } catch (IOException e) {
            System.out.println("No I/O");
            System.exit(1);
        }
    }
}

```

```
    }  
}  
  
public static void main(String[] args){  
    SocketClient frame = new SocketClient();  
    frame.setTitle("Client Program");  
    WindowListener l = new WindowAdapter() {  
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
    };  
  
    frame.addWindowListener(l);  
    frame.pack();  
    frame.setVisible(true);  
    frame.listenSocket();  
}
```

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

class SocketServer extends JFrame
    implements ActionListener {

    JButton button;
    JLabel label = new JLabel("Text received over socket:");
    JPanel panel;
    JTextArea textArea = new JTextArea();
    ServerSocket server = null;
    Socket client = null;
    BufferedReader in = null;
    PrintWriter out = null;
    String line;

    SocketServer(){ //Begin Constructor
        button = new JButton("Click Me");
        button.addActionListener(this);

        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(Color.white);
        getContentPane().add(panel);
        panel.add("North", label);
        panel.add("Center", textArea);
        panel.add("South", button);

    } //End Constructor

    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();

        if(source == button){
            textArea.setText(line);
        }
    }

    public void listenSocket(){

        try{
            server = new ServerSocket(4444);
        } catch (IOException e) {
            System.out.println("Could not listen on port 4444");
            System.exit(-1);
        }

        try{
            client = server.accept();
        } catch (IOException e) {
            System.out.println("Accept failed: 4444");
            System.exit(-1);
        }

        try{
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream(), true);
        } catch (IOException e) {
            System.out.println("Accept failed: 4444");
            System.exit(-1);
        }
    }
}
```

```
    }

    while(true){
        try{
            line = in.readLine();
//Send data back to client
            out.println(line);
        } catch (IOException e) {
            System.out.println("Read failed");
            System.exit(-1);
        }
    }
}

protected void finalize(){
//Clean up
    try{
        in.close();
        out.close();
        server.close();
    } catch (IOException e) {
        System.out.println("Could not close.");
        System.exit(-1);
    }
}

public static void main(String[] args){
    SocketServer frame = new SocketServer();
    frame.setTitle("Server Program");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
    frame.listenSocket();
}
}
```

```

import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

class ClientWorker implements Runnable {
    private Socket client;
    private JTextArea textArea;

    ClientWorker(Socket client, JTextArea textArea) {
        this.client = client;
        this.textArea = textArea;
    }

    public void run(){
        String line;
        BufferedReader in = null;
        PrintWriter out = null;
        try{
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream(), true);
        } catch (IOException e) {
            System.out.println("in or out failed");
            System.exit(-1);
        }

        while(true){
            try{
                line = in.readLine();
//Send data back to client
                out.println(line);
                textArea.append(line);
            } catch (IOException e) {
                System.out.println("Read failed");
                System.exit(-1);
            }
        }
    }
}

class SocketThrdServer extends JFrame{

    JLabel label = new JLabel("Text received over socket:");
    JPanel panel;
    JTextArea textArea = new JTextArea();
    ServerSocket server = null;

    SocketThrdServer(){ //Begin Constructor
        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(Color.white);
        getContentPane().add(panel);
        panel.add("North", label);
        panel.add("Center", textArea);
    } //End Constructor

    public void listenSocket(){
        try{
            server = new ServerSocket(4444);
        } catch (IOException e) {
            System.out.println("Could not listen on port 4444");
            System.exit(-1);
        }
    }
}

```

```
    }
    while(true){
        ClientWorker w;
        try{
            w = new ClientWorker(server.accept(), textArea);
            Thread t = new Thread(w);
            t.start();
        } catch (IOException e) {
            System.out.println("Accept failed: 4444");
            System.exit(-1);
        }
    }
}

protected void finalize(){
//Objects created in run method are finalized when
//program terminates and thread exits
    try{
        server.close();
    } catch (IOException e) {
        System.out.println("Could not close socket");
        System.exit(-1);
    }
}

public static void main(String[] args){
    SocketThrdServer frame = new SocketThrdServer();
    frame.setTitle("Server Program");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
    frame.listenSocket();
}
}
```