We will now design an application program that creates the GUI shown in Figure 8-22.

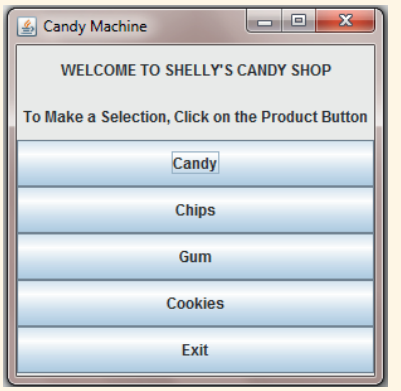

FIGURE 8-22 GUI for the candy machine

The program should do the following:

1. Show the customer the above GUI.
2. Let the customer make the selection.
3. When the user clicks on a product, show the customer its cost, and prompt the customer to enter the money for the product using an input dialog box, as shown in Figure 8-23.

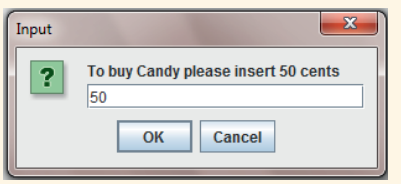

FIGURE 8-23 Input dialog box to enter money for the candy machine

4. Accept the money from the customer.
5. Make the sale and display a dialog box, as shown in Figure 8-24.

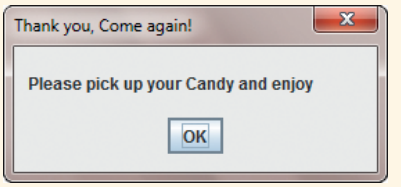

FIGURE 8-24 Output dialog box to show the output of the candy machine

In the first part of this programming example, we designed and implemented the classes `CashRegister` and `Dispenser`. Our final step is to revise the main program of the first part to create a GUI.

MAIN PROGRAM

We now describe how to create the candy machine using the classes `CashRegister` and `Dispenser` and the GUI components. When the program executes, it must display the GUI shown earlier in Figure 8-22.

The GUI contains a window, two labels, and five buttons. The labels and buttons are placed in the content pane of the window. As you learned in Chapter 6, to create the window, the application program is created by extending the definition of the class `JFrame`. Thus, we need the following GUI components:

```
private JLabel headingMainL;    //label for the first line

private JLabel selectionL;      //label for the second line

private JButton exitB, candyB, chipsB, gumB, cookiesB;
```

The following statements create and instantiate these labels and button objects:

```
headingMainL = new JLabel("WELCOME TO SHELLY'S CANDY SHOP",
                          SwingConstants.CENTER);

selectionL = new JLabel("To Make a Selection, "
                      + "Click on the Product Button",
                        SwingConstants.CENTER);

candyB = new JButton("Candy");

chipsB = new JButton("Chips");

gumB = new JButton("Gum");

cookiesB = new JButton("Cookies");

exitB = new JButton("Exit");
```

These components are to be placed in the content pane of the window. The seven components—labels and buttons—are arranged in seven rows. Therefore, the content pane layout will be a grid of 7 rows and 1 column. The following statements get the content pane and add these components to the content pane:

```
Container pane = getContentPane();
setSize(300, 300);

pane.setLayout(new GridLayout(7,1));

pane.add(headingMainL);
pane.add(selectionL);
pane.add(candyB);
pane.add(chipsB);
pane.add(gumB);
pane.add(cookiesB);
pane.add(exitB);
```

EVENT HANDLING

When the user clicks on a product button, it generates an action event. There are five buttons, each generating an action event. To handle these action events, we use the same process that we used in Chapter 6. That is:

1. Create a class implementing the `interface` `ActionListener`.
2. Provide the definition of the method `actionPerformed`.
3. Create and instantiate an object, action listener, of the class type created in Step 1.
4. Register the listener of Step 3 to each button.

In Chapter 6, we created a separate class for each of the buttons and then created a separate listener for each button. In this new program, rather than create a separate class for each button, we create only one class. Recall that the heading of the method `actionPerformed` is:

```
public void actionPerformed(ActionEvent e)
```

In Chapter 6, while providing the definition of this method, we ignored the formal parameter `e`. The formal parameter `e` is a reference variable of the `ActionEvent` type. The `class` `ActionEvent` contains `getActionCommand` (a method without parameters), which can be used to identify which button generated the event. For example, the expression:

```
e.getActionCommand()
```

returns the string containing the label of the component generating the event. We can now use the appropriate `String` method to determine the button generating the event.

If the user clicks on one of the product buttons, then the candy machine attempts to sell the product. Therefore, the action of clicking on a product button is to sell. For

this, we write the method `sellProduct` (discussed later in this Programming Example). If the user clicks on the `Exit` button, the program should terminate. Let's call the class to handle these events `ButtonHandler`. Its definition is:

```
private class ButtonHandler implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        if (e.getActionCommand().equals("Exit"))
            System.exit(0);
        else if (e.getActionCommand().equals("Candy"))
            sellProduct(candy, "Candy");
        else if (e.getActionCommand().equals("Chips"))
            sellProduct(chips, "Chips");
        else if (e.getActionCommand().equals("Gum"))
            sellProduct(gum, "Gum");
        else if (e.getActionCommand().equals("Cookies"))
            sellProduct(cookies, "Cookies");
    }
}
```

You can now declare, instantiate, and register the listener as follows:

```
private ButtonHandler pbHandler;  //declare the listener

pbHandler = new ButtonHandler();  //instantiate the object

    //register the listener with each button
candyB.addActionListener(pbHandler);
chipsB.addActionListener(pbHandler);
gumB.addActionListener(pbHandler);
cookiesB.addActionListener(pbHandler);
exitB.addActionListener(pbHandler);
```

Next, we describe the method `sellProduct`.

Method `sellProduct`

The definition of this method is similar to the one we designed for the non-GUI program. (We give the definition here for the sake of completeness.) This method attempts to sell a particular product selected by the customer. The candy machine contains four dispensers, which correspond to the four products. These dispensers will be declared as instance variables. Therefore, the dispenser of the product to be sold and the name of the product are passed as parameters to this method. Because the cash register will be declared as an instance variable, this method can directly access the cash register.

This definition of the method `sellProduct` is:

```
private void sellProduct(Dispenser product, String productName)
{
    int coinsInserted = 0;
    int price;
    int coinsRequired;
```

```
        String str;

        if (product.getCount() > 0)
        {
            price = product.getProductCost();
            coinsRequired = price - coinsInserted;

            while (coinsRequired > 0)
            {
                str = JOptionPane.showInputDialog("To buy "
                                        + productName
                                        + " please insert "
                                        + coinsRequired + " cents");
                coinsInserted = coinsInserted
                                + Integer.parseInt(str);
                coinsRequired = price - coinsInserted;
            }

            cashRegister.acceptAmount(coinsInserted);
            product.makeSale();

            JOptionPane.showMessageDialog(null,"Please pick up your "
                                    + productName + " and enjoy",
                                     "Thank you, Come again!",
                                    JOptionPane.PLAIN_MESSAGE);
        }
        else           //dispenser is empty
            JOptionPane.showMessageDialog(null,"Sorry "
                                    + productName
                                    + " is sold out\n" +
                                    "Make another selection",
                                    "Thank you, Come again!",
                                    JOptionPane.PLAIN_MESSAGE);
    }//end sellProduct
```

We have described the method `sellProduct` and the other necessary components, so next we write the Java application program for the candy machine.

The algorithm is as follows:

1. Create the cash register—that is, declare a reference variable of type `CashRegister` and instantiate the object.
2. Create four dispensers—that is, declare four reference variables of type `Dispenser` and instantiate the appropriate `Dispenser` objects. For example, the statement:

   ```
   Dispenser candy = new Dispenser(100, 50);
   ```

declares `candy` to be a reference variable of the `Dispenser` type and instantiates the object `candy` to hold the candies. The number of items in the object `candy` is `100`, and the cost of a candy is `50` cents.

3. Create the other objects, such as labels and buttons, as previously described.
4. Display the GUI showing the candy machine, as described at the beginning of this programming example.
5. Get and process the selection.

The complete list of the class containing the application program is:

```
//Candy Machine Program

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

public class CandyMachine extends JFrame
{
   private static final int WIDTH = 300;
   private static final int HEIGHT = 300;

        //Instance variables
   private CashRegister cashRegister = new CashRegister();
   private Dispenser candy = new Dispenser(100, 50);
   private Dispenser chips = new Dispenser(100, 65);
   private Dispenser gum = new Dispenser(75, 45);
   private Dispenser cookies = new Dispenser(100, 85);

   private JLabel headingMainL;
   private JLabel selectionL;

   private JButton exitB, candyB, chipsB, gumB, cookiesB;
   private ButtonHandler pbHandler;

   public CandyMachine()
   {
      setTitle("Candy Machine");      //set the window title
      setSize(WIDTH, HEIGHT);         //set the window size

      Container pane  = getContentPane();  //get the container
      pane.setLayout(new GridLayout(7,1));//set the pane layout

      pbHandler = new ButtonHandler();  //instantiate listener
                                         //object

      headingMainL =
               new JLabel("WELCOME TO SHELLY'S CANDY SHOP",
                          SwingConstants.CENTER); //instantiate
                                                  //the first label
      selectionL = new JLabel("To Make a Selection, "
                           + "Click on the Product Button",
                     SwingConstants.CENTER); //instantiate
                                             //the second label

      pane.add(headingMainL); //add the first label to the pane
      pane.add(selectionL);  //add the second label to the pane
```

```
        candyB = new JButton("Candy");  //instantiate the candy
                                        //button
        candyB.addActionListener(pbHandler);   //register the
                                               //listener to the
                                               //candy button
        chipsB = new JButton("Chips");  //instantiate the chips
                                        //button
        chipsB.addActionListener(pbHandler);  //register the
                                              //listener to the
                                              //chips button
        gumB = new JButton("Gum");  //instantiate the gum button
        gumB.addActionListener(pbHandler);  //register the
                                    //listener to the gum button

        cookiesB = new JButton("Cookies");   //instantiate the
                                             //cookies button
        cookiesB.addActionListener(pbHandler); //register the
                                //listener to the cookies button

        exitB = new JButton("Exit");//instantiate the exit button
        exitB.addActionListener(pbHandler); //register the
                                   //listener to the exit button

        pane.add(candyB);   //add the candy button to the pane
        pane.add(chipsB);   //add the chips button to the pane
        pane.add(gumB);     //add the gum button to the pane
        pane.add(cookiesB); //add the cookies button to the pane
        pane.add(exitB);    //add the exit button to the pane

        setVisible(true);     //show the window and its contents
        setDefaultCloseOperation(EXIT_ON_CLOSE);

    } //end constructor

           //class to handle button events
    private class ButtonHandler implements ActionListener
    {
        public void actionPerformed (ActionEvent e)
        {
            if (e.getActionCommand().equals("Exit"))
                System.exit(0);
            else if (e.getActionCommand().equals("Candy"))
                sellProduct(candy, "Candy");
            else if (e.getActionCommand().equals("Chips"))
                sellProduct(chips, "Chips");
            else if (e.getActionCommand().equals("Gum"))
                sellProduct(gum, "Gum");
            else if (e.getActionCommand().equals("Cookies"))
                sellProduct(cookies, "Cookies");
        }
    }

         //Method to sell a product
    private void sellProduct(Dispenser product,
                             String productName)
    {
      int coinsInserted = 0;
      int price;
```

```
        int coinsRequired;

        String str;

        if (product.getCount() > 0)
        {
           price = product.getProductCost();
           coinsRequired = price - coinsInserted;

           while (coinsRequired > 0)
           {
               str = JOptionPane.showInputDialog("To buy "
                                    + productName
                                    + " please insert "
                                    + coinsRequired + " cents");
               coinsInserted = coinsInserted
                             + Integer.parseInt(str);
               coinsRequired = price - coinsInserted;
           }

           cashRegister.acceptAmount(coinsInserted);
           product.makeSale();

           JOptionPane.showMessageDialog(null, "Please pick up "
                                + "your "
                                + productName + " and enjoy",
                                 "Thank you, Come again!",
                                JOptionPane.PLAIN_MESSAGE);
        }
        else           //dispenser is empty
           JOptionPane.showMessageDialog(null, "Sorry "
                                + productName
                                + " is sold out\n" +
                                "Make another selection",
                                "Thank you, Come again!",
                                JOptionPane.PLAIN_MESSAGE);
     }//end sellProduct

     public static void main(String[] args)
     {
          CandyMachine candyShop = new CandyMachine();
     }
}
```

Sample Run: (Figure 8-25 shows the sample run.)

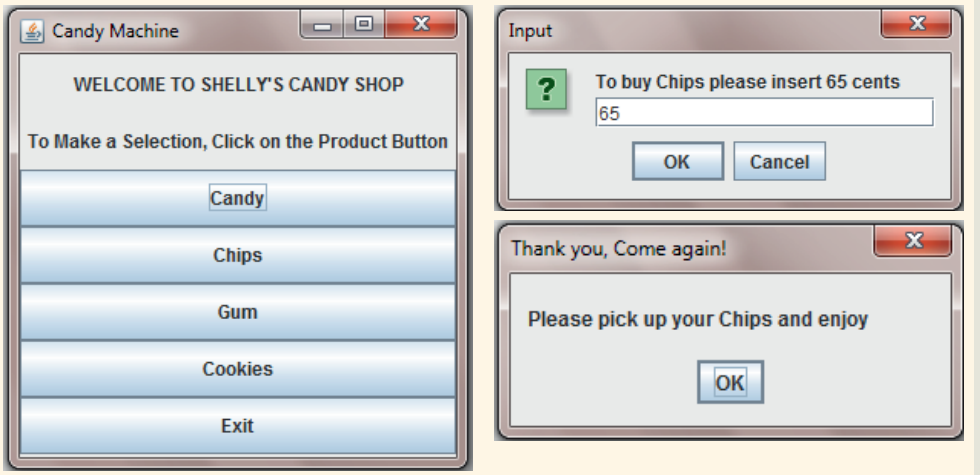

FIGURE 8-25 Sample run for `CandyMachine`