PROGRAMMING EXAMPLE: Candy Machine

A common place to buy candy is from a machine. A new candy machine has been purchased for the gym, but it is not working properly. The machine sells candies, chips, gum, and cookies. You have been asked to write a program for this candy machine so that it can be put into operation.

The program should do the following:

1. Show the customer the different products sold by the candy machine.
2. Let the customer make the selection.
3. Show the customer the cost of the item selected.
4. Accept money from the customer.
5. Release the item.

Input The item selection and the cost of the item.

Output The selected item.

PROBLEM ANALYSIS AND ALGORITHM DESIGN

A candy machine has two main components: a built-in cash register and several dispensers to hold and release the products.

Cash Register

Let us first discuss the properties of a cash register. The register has some cash on hand, it accepts the amount from the customer, and if the amount deposited is more than the cost of the item, then—if possible—it returns the change. For simplicity, we assume that the user deposits the money greater than or equal to the cost of the product. The cash register should also be able to show to the candy machine's owner the amount of money in the register at any given time. The following class defines the properties of a cash register:

```
class cashRegister
{
public:
    int getCurrentBalance() const;
      //Function to show the current amount in the cash
      //register.
      //Postcondition: The value of cashOnHand is returned.

    void acceptAmount(int amountIn);
      //Function to receive the amount deposited by
      //the customer and update the amount in the register.
      //Postcondition: cashOnHand = cashOnHand + amountIn;

    cashRegister(int cashIn = 500);
      //Constructor
      //Sets the cash in the register to a specific amount.
```

```
      //Postcondition: cashOnHand = cashIn;
      //                If no value is specified when the
      //                object is declared, the default value
      //                assigned to cashOnHand is 500.

private:
     int cashOnHand;  //variable to store the cash
                      //in the register
};
```

Figure 12-15 shows the UML class diagram of the `class` `cashRegister`.

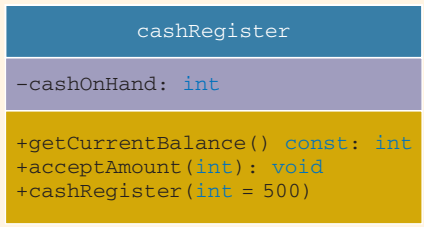

FIGURE 12-15 UML class diagram of the `class` `cashRegister`

Next, we give the definitions of the functions to implement the operations of the `class` `cashRegister`. The definitions of these functions are very simple and easy to follow.

The function `getCurrentBalance` shows the current amount in the cash register. It returns the value of the `private` member variable `cashOnHand`. So its definition is:

```
int cashRegister::getCurrentBalance() const
{
    return cashOnHand;
}
```

The function `acceptAmount` accepts the amount of money deposited by the customer. It updates the cash in the register by adding the amount deposited by the customer to the previous amount in the cash register. Essentially, the definition of this function is:

```
void cashRegister::acceptAmount(int amountIn)
{
    cashOnHand = cashOnHand + amountIn;
}
```

In the definition of the `class` `cashRegister`, the constructor is declared with a default value. Therefore, if the user does not specify any value when the object is declared, the default value is used to initialize the member variable `cashOnHand`. Recall that because we have specified the default value for the constructor's parameter in the

definition of the class, in the heading of the definition of the constructor, we do not specify the default value. The definition of the constructor is as follows:

```
cashRegister::cashRegister(int cashIn)
{
    if (cashIn >= 0)
        cashOnHand = cashIn;
    else
        cashOnHand = 500;
}
```

Note that the definition of the constructor checks for valid values of the parameter `cashIn`. If the value of `cashIn` is less than `0`, the value assigned to the member variable `cashOnHand` is `500`.

Dispenser

The dispenser releases the selected item if it is not empty. It should show the number of items in the dispenser and the cost of the item. The following class defines the properties of a dispenser. Let us call this `class dispenserType`:

```
class dispenserType
{
public:
    int getNoOfItems() const;
      //Function to show the number of items in the machine.
      //Postcondition: The value of numberOfItems is returned.

    int getCost() const;
      //Function to show the cost of the item.
      //Postcondition: The value of cost is returned.

    void makeSale();
      //Function to reduce the number of items by 1.
      //Postcondition: numberOfItems--;

    dispenserType(int setNoOfItems = 50, int setCost = 50);
      //Constructor
      //Sets the cost and number of items in the dispenser
      //to the values specified by the user.
      //Postcondition: numberOfItems = setNoOfItems;
      //               cost = setCost;
      //               If no value is specified for a
      //               parameter, then its default value is
      //               assigned to the corresponding member
      //               variable.

private:
    int numberOfItems;   //variable to store the number of
                         //items in the dispenser
    int cost;  //variable to store the cost of an item
};
```

Figure 12-16 shows the UML class diagram of the **class** **dispenserType**.

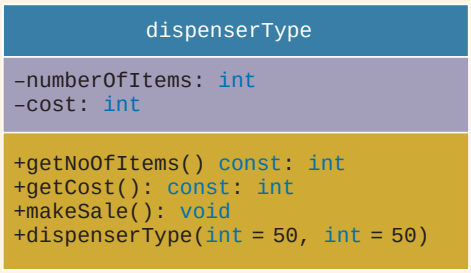

FIGURE 12-16 UML class diagram of the **class** dispenserType

Because the candy machine sells four types of items, we shall declare four objects of type **dispenserType**. For example, the statement:

```
dispenserType chips(100, 65);
```

declares **chips** to be an object of type **dispenserType**, sets the number of chip bags in the dispenser to **100**, and sets the cost of each chip bag to **65** cents (see Figure 12-17).

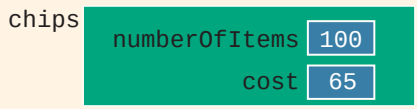

FIGURE 12-17 Object chips

Next, we discuss the definitions of the functions to implement the operations of the **class** **dispenserType**.

The function **getNoOfItems** returns the number of items of a particular product. Because the number of items currently in the dispenser is stored in the **private** member variable **numberOfItems**, the function returns the value of **numberOfItems**. The definition of this function is:

```
int dispenserType::getNoOfItems() const
{
    return numberOfItems;
}
```

The function `getCost` returns the cost of a product. Because the cost of a product is stored in the `private` member variable `cost`, the function returns the value of `cost`. The definition of this function is:

```
int dispenserType::getCost() const
{
    return cost;
}
```

When a product is sold, the number of items in that dispenser is reduced by `1`. Therefore, the function `makeSale` reduces the number of items in the dispenser by `1`. That is, it decrements the value of the `private` member variable `numberOfItems` by `1`. The definition of this function is:

```
void dispenserType::makeSale()
{
    numberOfItems--;
}
```

The definition of the constructor checks for valid values of the parameters. If these values are less than `0`, the default values are assigned to the member variables. The definition of the constructor is:

```
     //constructor
dispenserType::dispenserType(int setNoOfItems, int setCost)
{
    if (setNoOfItems >= 0)
        numberOfItems = setNoOfItems;
    else
        numberOfItems = 50;

    if (setCost >= 0)
        cost = setCost;
    else
        cost = 50;
}
```

MAIN PROGRAM

When the program executes, it must do the following:

1. Show the different products sold by the candy machine.
2. Show how to select a particular product.
3. Show how to terminate the program.

Furthermore, these instructions must be displayed after processing each selection (except exiting the program) so that the user need not remember what to do if he or she wants to buy two or more items. Once the user has made the appropriate selection, the candy machine must act accordingly. If the user has opted to buy a product and that product is available, the candy machine should show the cost of the

product and ask the user to deposit the money. If the amount deposited is at least the cost of the item, the candy machine should sell the item and display an appropriate message.

This discussion translates into the following algorithm:

1. Show the selection to the customer.
2. Get the selection.
3. If the selection is valid and the dispenser corresponding to the selection is not empty, sell the product.

We divide this program into three functions: `showSelection`, `sellProduct`, and `main`.

showSelection

This function displays the information necessary to help the user select and buy a product. This definition of the function `showSelection` is:

```
void showSelection()
{
    cout << "*** Welcome to Shelly's Candy Shop ***" << endl;
    cout << "To select an item, enter " << endl;
    cout << "1 for Candy" << endl;
    cout << "2 for Chips" << endl;
    cout << "3 for Gum" << endl;
    cout << "4 for Cookies" << endl;
    cout << "9 to exit" << endl;
}//end showSelection
```

sellProduct

This function attempts to sell the product selected by the customer. Therefore, it must have access to the dispenser holding the product. The first thing that this function does is check whether the dispenser holding the product is empty. If the dispenser is empty, the function informs the customer that this product is sold out. If the dispenser is not empty, it tells the user to deposit the necessary amount to buy the product.

If the user does not deposit enough money to buy the product, `sellProduct` tells the user how much additional money must be deposited. If the user fails to deposit enough money in two tries to buy the product, the function simply returns the money. (Programming Exercise 9, at the end of this chapter, asks you to revise the definition of the method `sellProduct` so that it keeps asking the user to enter the additional amount as long as the user has not entered enough money to buy the product.) If the amount deposited by the user is sufficient, it accepts the money and sells the product. Selling the product means to decrement the number of items in the dispenser by `1` and to update the money in the cash register by adding the cost of the product. (Because this program does not return the extra money

deposited by the customer, the cash register is updated by adding the money entered by the user.)

From this discussion, it is clear that the function `sellProduct` must have access to the dispenser holding the product (to decrement the number of items in the dispenser by `1` and to show the cost of the item) as well as the cash register (to update the cash). Therefore, this function has two parameters: one corresponding to the dispenser and the other corresponding to the cash register. Furthermore, both parameters must be referenced.

In pseudocode, the algorithm for this function is:

1. If the dispenser is not empty,
    a. Show and prompt the customer to enter the cost of the item.
    b. Get the amount entered by the customer.
    c. If the amount entered by the customer is less than the cost of the product,
        i. Show and prompt the customer to enter the additional amount.
        ii. Calculate the total amount entered by the customer.
    d. If the amount entered by the customer is at least the cost of the product,
        i. Update the amount in the cash register.
        ii. Sell the product—that is, decrement the number of items in the dispenser by `1`.
        iii. Display an appropriate message.
    e. If the amount entered by the user is less than the cost of the item, return the amount.
2. If the dispenser is empty, tell the user that this product is sold out.

This definition of the function `sellProduct` is:

```
void sellProduct(dispenserType& product,
                 cashRegister& pCounter)
{
    int amount;  //variable to hold the amount entered
    int amount2; //variable to hold the extra amount needed

    if (product.getNoOfItems() > 0) //if the dispenser is not
                                    //empty
    {
        cout << "Please deposit " << product.getCost()
             << " cents" << endl;
        cin >> amount;
```

```
            if (amount < product.getCost())
            {
                cout << "Please deposit another "
                     << product.getCost()- amount
                     << " cents" << endl;
                cin >> amount2;
                amount = amount + amount2;
            }

            if (amount >= product.getCost())
            {
                pCounter.acceptAmount(amount);
                product.makeSale();
                cout << "Collect your item at the bottom and "
                     << "enjoy." << endl;
            }
            else
                cout << "The amount is not enough. "
                     << "Collect what you deposited." << endl;

            cout << "*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*"
                 << endl << endl;
        }
        else
            cout << "Sorry, this item is sold out." << endl;
    }//end sellProduct
```

Now that we have described the functions `showSelection` and `sellProduct`, the function `main` is described next.

main

The algorithm for the function `main` is as follows:

1. Create the cash register—that is, declare an object of type `cashRegister`.
2. Create four dispensers—that is, declare four objects of type `dispenserType` and initialize these objects. For example, the statement:

   ```
   dispenserType candy(100, 50);
   ```

   creates a dispenser object, `candy`, to hold the candies. The number of items in the dispenser is `100`, and the cost of an item is `50` cents.
3. Declare additional variables as necessary.
4. Show the selection; call the function `showSelection`.
5. Get the selection.
6. While not done (a selection of `9` exits the program),
   a. Sell the product; call the function `sellProduct`.
   b. Show the selection; call the function `showSelection`.
   c. Get the selection.

The definition of the function `main` is as follows:

```
int main()
{
    cashRegister counter;
    dispenserType candy(100, 50);
    dispenserType chips(100, 65);
    dispenserType gum(75, 45);
    dispenserType cookies(100, 85);

    int choice;  //variable to hold the selection

    showSelection();
    cin >> choice;

    while (choice != 9)
    {
        switch (choice)
        {
        case 1:
            sellProduct(candy, counter);
            break;
        case 2:
            sellProduct(chips, counter);
            break;
        case 3:
            sellProduct(gum, counter);
            break;
        case 4:
            sellProduct(cookies, counter);
            break;
        default :
            cout << "Invalid selection." << endl;
        }//end switch

        showSelection();
        cin >> choice;
    }//end while

    return 0;

}//end main
```

COMPLETE PROGRAM LISTING

In the previous sections, we designed the classes to implement cash registers and dispensers to implement a candy machine. In this section, for the sake of completeness, we give complete definitions of the classes, the implementation file, and the user program to implement a candy machine.

```
//*************************************************************
// Author: D.S. Malik
//
// class cashRegister
// This class specifies the members to implement a cash
// register.
//*************************************************************

class cashRegister
{
public:
    int getCurrentBalance() const;
      //Function to show the current amount in the cash
      //register.
      //Postcondition: The value of cashOnHand is returned.

    void acceptAmount(int amountIn);
      //Function to receive the amount deposited by
      //the customer and update the amount in the register.
      //Postcondition: cashOnHand = cashOnHand + amountIn;

    cashRegister(int cashIn = 500);
      //Constructor
      //Sets the cash in the register to a specific amount.
      //Postcondition: cashOnHand = cashIn;
      //               If no value is specified when the
      //               object is declared, the default value
      //               assigned to cashOnHand is 500.

private:
     int cashOnHand;   //variable to store the cash
                       //in the register
};

//*************************************************************
// Author: D.S. Malik
//
// class dispenserType
// This class specifies the members to implement a dispenser.
//*************************************************************

class dispenserType
{
public:
    int getNoOfItems() const;
      //Function to show the number of items in the machine.
      //Postcondition: The value of numberOfItems is returned.

    int getCost() const;
      //Function to show the cost of the item.
      //Postcondition: The value of cost is returned.

    void makeSale();
      //Function to reduce the number of items by 1.
      //Postcondition: numberOfItems--;
```

```
    dispenserType(int setNoOfItems = 50, int setCost = 50);
      //Constructor
      //Sets the cost and number of items in the dispenser
      //to the values specified by the user.
      //Postcondition: numberOfItems = setNoOfItems;
      //               cost = setCost;
      //               If no value is specified for a
      //               parameter, then its default value is
      //               assigned to the corresponding member
      //               variable.

private:
    int numberOfItems;     //variable to store the number of
                           //items in the dispenser
    int cost;  //variable to store the cost of an item
};


//***********************************************************
// Author: D.S. Malik
//
// Implementation file candyMachineImp.cpp
// This file contains the definitions of the functions to
// implement the operations of the classes cashRegister and
// dispenserType.
//***********************************************************

#include <iostream>
#include "candyMachine.h"

using namespace std;

int cashRegister::getCurrentBalance() const
{
    return cashOnHand;
}

void cashRegister::acceptAmount(int amountIn)
{
    cashOnHand = cashOnHand + amountIn;
}

cashRegister::cashRegister(int cashIn)
{
    if (cashIn >= 0)
        cashOnHand = cashIn;
    else
        cashOnHand = 500;
}

int dispenserType::getNoOfItems() const
{
    return numberOfItems;
}
```

```
int dispenserType::getCost() const
{
    return cost;
}

void dispenserType::makeSale()
{
    numberOfItems--;
}

dispenserType::dispenserType(int setNoOfItems, int setCost)
{
    if (setNoOfItems >= 0)
         numberOfItems = setNoOfItems;
    else
        numberOfItems = 50;

    if (setCost >= 0)
        cost = setCost;
    else
        cost = 50;
}
```

Main Program

```
//*****************************************************
// Author: D.S. Malik
//
// This program uses the classes cashRegister and
// dispenserType to implement a candy machine.
// *****************************************************

#include <iostream>
#include "candyMachine.h"

using namespace std;

void showSelection();
void sellProduct(dispenserType& product,
                 cashRegister& pCounter);

int main()
{
    cashRegister counter;
    dispenserType candy(100, 50);
    dispenserType chips(100, 65);
    dispenserType gum(75, 45);
    dispenserType cookies(100, 85);

    int choice;  //variable to hold the selection

    showSelection();
    cin >> choice;

    while (choice != 9)
    {
        switch (choice)
```

```
        {
        case 1:
            sellProduct(candy, counter);
            break;
        case 2:
            sellProduct(chips, counter);
            break;
        case 3:
            sellProduct(gum, counter);
            break;
        case 4:
            sellProduct(cookies, counter);
            break;
        default:
            cout << "Invalid selection." << endl;
        }//end switch
        showSelection();
        cin >> choice;
    }//end while

    return 0;
}//end main

void showSelection()
{
    cout << "*** Welcome to Shelly's Candy Shop ***" << endl;
    cout << "To select an item, enter " << endl;
    cout << "1 for Candy" << endl;
    cout << "2 for Chips" << endl;
    cout << "3 for Gum" << endl;
    cout << "4 for Cookies" << endl;
    cout << "9 to exit" << endl;
}//end showSelection

void sellProduct(dispenserType& product,
                 cashRegister& pCounter)
{
    int amount;  //variable to hold the amount entered
    int amount2; //variable to hold the extra amount needed

    if (product.getNoOfItems() > 0) //if the dispenser is not
                                    //empty
    {
        cout << "Please deposit " << product.getCost()
             << " cents" << endl;
        cin >> amount;

        if (amount < product.getCost())
        {
            cout << "Please deposit another "
                 << product.getCost()- amount
                 << " cents" << endl;
            cin >> amount2;
            amount = amount + amount2;
        }
```

```
        if (amount >= product.getCost())
        {
            pCounter.acceptAmount(amount);
            product.makeSale();
            cout << "Collect your item at the bottom and "
                 << "enjoy." << endl;
        }
        else
            cout << "The amount is not enough. "
                 << "Collect what you deposited." << endl;

        cout << "*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*"
             << endl << endl;
    }
    else
        cout << "Sorry, this item is sold out." << endl;
}//end sellProduct
```

Sample Run: In this sample run, the user input is shaded.

```
*** Welcome to Shelly's Candy Shop ***
To select an item, enter
1 for Candy
2 for Chips
3 for Gum
4 for Cookies
9 to exit
1
Please deposit 50 cents
50
Collect your item at the bottom and enjoy.
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

*** Welcome to Shelly's Candy Shop ***
To select an item, enter
1 for Candy
2 for Chips
3 for Gum
4 for Cookies
9 to exit
9
```

NOTE

We placed the definitions of the `class`es `cashRegister` and `dispenserType` in the same header file `candyMachine.h`. However, you can also place the definitions of these classes in separate header files and include those header files in the files that use these classes, such as the implementation file of these classes and the file that contains the main program. Similarly, you can also create separate implementation files for these classes. The Web site accompanying this book contains these header and implementation files.