13.2 Simple Uses of Inner Classes

The ruling ideas of each age have ever been the ideas of its ruling class.

KARL MARX and FRIEDRICH ENGELS, The Communist Manifesto

Inner classes are classes defined within other classes. In this section, we will describe one of the most useful applications of inner classes, namely, inner classes used as helping classes.

Helping Classes

Defining an inner class is straightforward; simply include the definition of the inner class within another class, as follows:

```java
public class OuterClass {
    private class InnerClass {
        // Declarations_of_InnerClass_Instance_Variables
        // Definitions_of_InnerClass_Methods
    }
}
```

As this outline suggests, the class that includes the inner class is called an outer class. The definition of the inner class (or classes) need not be the first item(s) of the outer class, but it is good to place it either first or last so that it is easy to find. The inner class need not be private, but that is the only case we will consider in this section. We will consider other modifiers besides private in Section 13.3.

An inner class definition is a member of the outer class in the same way that the instance variables of the outer class and the methods of the outer class are members of the outer class. Thus, an inner class definition is local to the outer class definition. So you may reuse the name of the inner class for something else outside the definition of the outer class. If the inner class is private, as ours will always be in this section, then the inner class cannot be accessed by name outside the definition of the outer class.

There are two big advantages to inner classes. First, because they are defined within a class, they can be used to make the outer class self-contained or more self-contained than it would otherwise be. The second advantage is that the inner and outer classes’ methods have access to each other’s private methods and private instance variables.
TIP: Inner and Outer Classes Have Access to Each Other's Private Members

Within the definition of a method of an inner class, it is legal to reference a private instance variable of the outer class and to invoke a private method of the outer class. To facilitate this, Java follows this convention: If a method is invoked in an inner class and the inner class has no such method, then it is assumed to be an invocation of the method by that name in the outer class. (If the outer class also has no method by that name, that is, of course, an error.) Similarly, an inner class can use the name of an instance variable of the outer class.

The reverse situation, invoking a method of the inner class from the outer class, is not so simple. To invoke a (nonstatic) method of the inner class from within a method of the outer class, you need an object of the inner class to use as a calling object, as we did in Display 13.9 (Part 2).

As long as you are within the definition of the inner or outer classes, the modifiers public and private (used within the inner or outer classes) are equivalent.

These sorts of invocations and variable references that cross between inner and outer classes can get confusing. So, it is best to confine such invocations and variable references to cases that are clear and straightforward. It is easy to tie your code in knots if you get carried away with this sort of thing.

Access Privileges between Inner and Outer Classes

Inner and outer classes have access to each other's private members.

EXAMPLE: A Bank Account Class

Display 13.9 contains a simplified bank account program with an inner class for amounts of money. The bank account class uses values of type String to obtain or return amounts of money, such as the amount of a deposit or the answer to a query for the account balance. However, inside the class it stores amounts of money as values of type Money, which is an inner class. Values of type Money are not stored as Strings, which would be difficult to do arithmetic on, nor are they stored as values of type double, which would allow round-off errors that would not be acceptable in banking transactions. Instead, the class Money stores amounts of money as two integers, one for the dollars and one for the cents. In a real banking program, the class Money might have a larger collection of methods, such as methods to do addition, subtraction, and compute percentages, but in this simple example we have only included the method for adding an amount of money to the calling object. The outer class BankAccount would also have more methods in a real class, but here we have only included methods to deposit an amount of money to the account and to obtain the account balance. Display 13.10 contains a simple demonstration program using the class BankAccount.

(continued)
EXAMPLE: (continued)

The class Money is a private inner class of the class BankAccount. So, the class Money cannot be used outside of the class BankAccount. (Public inner classes are discussed in Section 13.3 and have some subtleties involved in their use.) Since the class Money is local to the class BankAccount, the name Money can be used for the name of another class outside of the class BankAccount. (This would be true even if Money were a public inner class.)

We have made the instance variables in the class Money private following our usual conventions for class members. When we discuss public inner classes, this will be important. However, for use within the outer class (and a private inner class cannot be used anywhere else), there is no difference between public and private or other member modifiers. All instance variables and all methods of the inner class are public to the outer class no matter whether they are marked public or private or anything else. Notice the method closeAccount of the outer class. It uses the private instance variables dollars and cents of the inner class.

This is still very much a toy example, but we will have occasion to make serious use of private inner classes when we discuss linked lists in Chapter 15 and when we study Swing GUIs starting in Chapter 17.

Display 13.9 Class with an Inner Class (part 1 of 2)

```java
1    public class BankAccount
2    {
3        private class Money
4            {
5                private long dollars;
6                private int cents;
7
8                    public Money(String stringAmount)
9                    {
10                        abortOnNull(stringAmount);
11                        int length = stringAmount.length();
12                        dollars = Long.parseLong(
13                            stringAmount.substring(0, length - 3));
14                        cents = Integer.parseInt(  
15                            stringAmount.substring(length - 2, length));
16                    }
17
18                    public String getAmount()
19                    {
20                        if (cents > 9)
```
Display 13.9 Class with an Inner Class (part 2 of 2)

```java
return (dollars + "." + cents);
else
    return (dollars + ".0" + cents);
}

public void addIn(Money secondAmount)
{
    abortOnNull(secondAmount);
    int newCents = (cents + secondAmount.cents)%100;
    long carry = (cents + secondAmount.cents)/100;
    cents = newCents;
    dollars = dollars + secondAmount.dollars + carry;
}
private void abortOnNull(Object o)
{
    if (o == null)
    {
        System.out.println("Unexpected null argument.");
        System.exit(0);
    }
}

private Money balance;

public BankAccount()
{
    balance = new Money("0.00");
}

public String getBalance()
{
    return balance.getAmount();
}

public void makeDeposit(String depositAmount)
{
    balance.addIn(new Money(depositAmount));
}

public void closeAccount()
{
    balance.dollars = 0;
    balance.cents = 0;
}
```

The definition of the inner class ends here, but the definition of the outer class continues in Part 2 of this display.

To invoke a nonstatic method of the inner class outside of the inner class, you need to create an object of the inner class.

This invocation of the inner class method `getAmount()` would be allowed even if the method `getAmount()` were marked as private.

Notice that the outer class has access to the private instance variables of the inner class.

This class would normally have more methods, but we have only included the methods we need to illustrate the points covered here.
public class BankAccountDemo
{
    public static void main(String[] args)
    {
        System.out.println("Creating a new account.");
        BankAccount account = new BankAccount();
        System.out.println("Account balance now = $" + account.getBalance());
        System.out.println("Depositing $100.00");
        account.makeDeposit("100.00");
        System.out.println("Account balance now = $" + account.getBalance());
        System.out.println("Depositing $99.99");
        account.makeDeposit("99.99");
        System.out.println("Account balance now = $" + account.getBalance());
        System.out.println("Depositing $0.01");
        account.makeDeposit("0.01");
        System.out.println("Account balance now = $" + account.getBalance());
        System.out.println("Closing account.");
        account.closeAccount();
        System.out.println("Account balance now = $" + account.getBalance());
    }
}

Sample Dialogue

Creating a new account.
Account balance now = $0.00
Depositing $100.00
Account balance now = $100.00
Depositing $99.99
Account balance now = $199.99
Depositing $0.01
Account balance now = $200.00
Closing account.
Account balance now = $0.00
Helping Inner Classes

You may define a class within another class. The inside class is called an **inner class**. A common and simple use of an inner class is to use it as a helping class for the outer class, in which case the inner class should be marked **private**.

### Self-Test Exercises

20. Would the following invocation of `getAmount` in the method `getBalance` of the outer class `BankAccount` still be legal if we change the method `getAmount` of the inner class `Money` from public to private?

```java
public String getBalance()
{
    return balance.getAmount();
}
```

21. Since it does not matter if we make the members of a private inner class public or private, can we simply omit the public or private modifiers from the instance variables and methods of a private inner class?

22. Would it be legal to add the following method to the inner class `Money` in Display 13.9? Remember, the question is would it be legal, not would it be sensible.

```java
public void doubleBalance()
{
    balance.addIn(balance);
}
```

23. Would it be legal to add the following method to the inner class `Money` in Display 13.9? Remember, the question is would it be legal, not would it be sensible.

```java
public void doubleBalance2()
{
    makeDeposit(balance.getAmount());
}
```

### The `.class` File for an Inner Class

When you compile any class in Java, that produces a `.class` file. When you compile a class with an inner class, that compiles both the outer class and the inner class and produces two `.class` files. For example, when you compile the class `BankAccount` in Display 13.9, that produces the following two `.class` files:

`BankAccount.class` and `BankAccount$Money.class`

If `BankAccount` had two inner classes, then three `.class` files would be produced.
Listeners as Inner Classes ★

In all of our previous examples, our GUIs had only one action listener object to deal with all action events from all buttons and menus in the GUI. The opposite extreme also has much to recommend it. You can have a separate ActionListener class for each button or menu item, so that each button or menu item has its own unique action listener. There is then no need for a multiway if-else statement. The listener knows which button or menu item was clicked because it listens to only one button or menu item.

The approach outlined in the previous paragraph does have one downside: You typically need to give a lot of definitions of ActionListener classes. Rather than putting each of these classes in a separate file, it is much cleaner to make them private inner classes. This has the added advantage of allowing the ActionListener classes to have access to private instance variables and methods of the outer class.

In Display 17.16 we have redone the GUI in Display 17.14 using the techniques of this subsection.

Display 17.16  Listeners as Inner Classes (part 1 of 3)

```
<Import statements are the same as in Display 17.14.>

public class InnerListenersDemo extends JFrame {

  public static final int WIDTH = 300;
  public static final int HEIGHT = 200;

  private JPanel greenPanel;
  private JPanel whitePanel;
  private JPanel grayPanel;

  private class greenListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      greenPanel.setBackground(Color.GREEN);
    }
  }  //End of greenListener inner class

  private class WhiteListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      whitePanel.setBackground(Color.WHITE);
    }
  }  //End of WhiteListener inner class
```

(continued)
private class grayListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        grayPanel.setBackground(Color.GRAY);
    }
}

//End of grayListener inner class

public static void main(String[] args)
{
    InnerListenersDemo gui = new InnerListenersDemo();
    gui.setVisible(true);
}

public InnerListenersDemo()
{
    super("Menu Demonstration");
    setSize(WIDTH, HEIGHT);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new GridLayout(1, 3));

    greenPanel = new JPanel();
greenPanel.setBackground(Color.LIGHT_GRAY);
add(greenPanel);

    whitePanel = new JPanel();
    whitePanel.setBackground(Color.LIGHT_GRAY);
add(whitePanel);

    grayPanel = new JPanel();
    grayPanel.setBackground(Color.LIGHT_GRAY);
add(grayPanel);

    JMenu colorMenu = new JMenu("Add Colors");

    JMenuItem greenChoice = new JMenuItem("Green");
greenChoice.addActionListener(new grayListener());
colorMenu.add(greenChoice);

    JMenuItem whiteChoice = new JMenuItem("White");
whiteChoice.addActionListener(new whiteListener());
colorMenu.add(whiteChoice);
Display 17.16  Listeners as Inner Classes (part 3 of 3)

```java
56  JMenuItem grayChoice = new JMenuItem("Gray");
57  grayChoice.addActionListener(new grayListener());
58  colorMenu.add(grayChoice);
59  
60  JMenuBar bar = new JMenuBar();
61  bar.add(colorMenu);
62  setJMenuBar(bar);
63 }
```

Self-Test Exercises

25. What type of event is fired when you click a JMenuItem? How does it differ from the type of event fired when you click a JButton?

26. Write code to create a JButton with "Hello" written on it but with "Bye" as its action command.

27. Write code to create a JMenuItem with "Hello" as its displayed text (when it is a choice in a menu) but with "Bye" as its action command.

28. If you want to change the action command for a JButton, you use the method `setActionCommand`. What method do you use to change the action command for a JMenuItem?

29. Is the following legal in Java?

```java
JMenu oMenu = new JMenu();
...
JMenu oSubMenu = new JMenu();
...
oMenu.add(oSubMenu);
```

30. How many JMenuBar objects can you have in a JFrame?

31. A JFrame has a private instance variable of type JMenuBar. What is the name of the mutator method to change the value of this instance variable?

32. Write code to create a new menu item named oChoice that has the label "Exit".

33. Suppose you are defining a class called MenuGUI that is a derived class of the class JFrame. Write code to add the menu item pItem to the menu m. Then add m to the menu bar mBar, and then add the menu bar to the JFrame MenuGUI. Assume that this all takes place inside a constructor for MenuGUI. Also assume that everything has already been constructed with new, and that all necessary listeners are registered. You just need to do the adding of things.