Exploring Encapsulation, Version 1

Note – This exercise is part of the ongoing `Banking` project. You must make sure that you have completed all prior exercises related to this project before attempting this exercise. If you encountered difficulty in completing the prior exercise related to this project, you may obtain the prior exercise's solution files from the `projects\StarterFiles` directory.

In the next two exercises, you explore the purpose of proper *object encapsulation*. You create a class in two steps demonstrating the use of information hiding. In this version you will create an `Account` class with public data members. You will then create a test program that demonstrates the danger of using the public data directly.

Figure 2-1 shows the UML class diagram of the `Account` class that you will create in this exercise. This class will have one public data member (or instance variable), called `balance`, that maintains the monetary value of the customer's bank account.

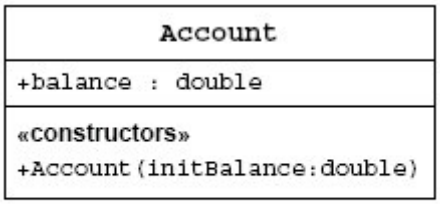

Figure 2-1 UML Class Diagram of `Account` With No Hiding

There is only one business rule that applies to the `Account` class: *The balance of the bank account must never go below zero*. In this exercise, you will discover that the `Account` class cannot ensure this business rule.

Preparation

1. Go to the `projects\BankPrj` directory.

Task 1 – Creating the `Account` Class

Using a text editor, create the `Account` class source file. This class must satisfy the UML diagram in Figure 2-1.

2. Declare the `Account` class.

```
public class Account {
  // code here
}
```

3. Add the `balance` instance variable.

```
public double balance;
```

4. Add a constructor that sets the balance to the initial balance argument passed to the constructor.

```
public Account(double initBalance) {
  balance = initBalance;
}
```

Task 2 – Creating the `TestAccount2` Class

Using a text editor, create the `TestAccount2` program source file. This class acts as a program to create an `Account` object with an initial balance of 100. The test program will then add 47 and then subtract 150. Finally, the test program must print out the balance of the object to the standard output stream.

5. Declare the `TestAccount2` class.

```
public class TestAccount2 {
  // code here
}
```

6. Add the `main` method:

```
public static void main(String[] args) {
  // code here
}
```

a. Declare a variable within the `main` method of type `Account` named `acct`. Also, in the same statement, initialize the variable `acct` to a new instance of `Account` by passing 100.00 to the constructor as the initial balance.

```
Account acct = new Account(100.0);
```

b. Use the addition operator to add 47 to the account object's balance.

```
acct.balance = acct.balance + 47.0;
```

c. Use the subtraction operator to subtract 150 from the account object's balance.

```
acct.balance = acct.balance - 150.0;
```

d. Use the `System.out.println` method to display the balance to the standard output stream.

```
System.out.println("Final account balance is " + acct.balance);
```

Task 3 – Compiling the `TestAccount2` Program

7. On the command line, use the `javac` command to compile the test program and the `Account` class.

`javac TestAccount2.java`

Discussion – Why only compile the test program? Why not also compile the `Account` class?

Task 4 – Running the `TestAccount2` Program

8. On the command line, use the `java` command to run the test program.

`java TestAccount2`

The output should be:

```
The final balance is -3.0
```