Java Programming Language, Java SE 6

Activity Guide - Volume 1

SL-275-SE6 G.2

D61748GC11
Edition 1.1
June 2010
D67982

ORACLE

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Workbook

Lab Goals

Upon completion of this workbook, you should be able to:

- Write a Java™ technology program using the fundamental language elements: primitive types, reference types, arithmetic operators, relational operators, conditional statements, and iterative statements
- Write a Java technology program using good object-oriented programming concepts and principles: encapsulation, inheritance, interfaces, polymorphism, object association, and multiplicity in associations
- Write a robust Java technology program using exception handling, generic collections, and concurrency control
- Write a rich Java technology program using the graphical user interface (GUI) application programming interfaces (APIs), input/output (I/O) APIs, and networking APIs

This workbook presents the lab exercises for each module of the Student Guide.

Performing the Exercises

You have the option to complete any one of three versions of a lab. To decide which to choose, consult the following descriptions of the levels:

- Level 1 – This version of the lab provides the least amount of guidance. Each bulleted paragraph provides a task description, but you must determine your own way of accomplishing each task.
- Level 2 – This version of the lab provides more guidance. Although each step describes what you should do, you must determine which commands (and options) to input.
- Level 3 – This version of the lab is the easiest to accomplish because each step provides exactly what you should input to the system. This level also includes the task solutions for all three levels.

Also, several modules have advanced labs. These labs are optional. These labs are intended for more advanced students who complete the primary labs easily.

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

Use `ls -al` to list all files.
`system% You have mail.`

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

The `getServletInfo` method is used to get author information.
The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

To list the files in this directory, type:
`# ls`

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

Notice the `javax.servlet` interface is imported to allow access to its life cycle methods (Line 2).

`Courier italics` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for filename to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

Read Chapter 6 in the *User's Guide*.
These are called *class* options.

Additional Conventions

Java programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris™ Operating System (Solaris OS) is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris OS

  ```
  $ cd $SERVER_ROOT/bin
  ```

  If working in Microsoft Windows

  ```
  C:\> cd %SERVER_ROOT%\bin
  ```

Lab 1

Getting Started

Objectives

Upon completion of this lab, you should be able to:

- Diagnose simple compilation and runtime errors
- Create a test program for an existing class

Exercise 1: Exploring Java™ Program Errors

In this exercise, you view the source of several simple Java programs, and correct their compilation and runtime errors.

This exercise contains the following sections:

- "Task – Correcting Compilation and Runtime Errors"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Opening Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `TestProject` project in the `exercises/01_intro/exercise1` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/01_intro/exercise1` directory.

Task – Correcting Compilation and Runtime Errors

In this task, you are presented with four Java programs that contain errors. The errors can be either compilation or runtime errors. Your job is to diagnose and fix those errors so the programs will execute.

Complete the following steps for the files `Test1.java`, `Test2.java`, `Test3.java` and `Test4.java`:

Tool Reference – Java Development: Java Classes: Opening Java Classes

1. Open the Java file.

Tool Reference – Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes

2. Compile the Java file.
3. If a compilation error occurs, identify the source of the error and fix it.
4. Repeat step 2 and step 3 until there are no more errors.

Tool Reference – Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

5. Execute the Java program.
6. If a runtime error occurs, identify the source of the error and fix it.
7. Repeat step 5 and step 6 until there are no more errors.

Exercise 2: Creating a Test Program (Level 1)

In this exercise, you create a test program (also known as a *test harness*) to exercise a pre-provided class. These are the Level 1 instructions that provide additional hints.

Figure 1-1 shows the definition of the `Account` class using a Unified Modeling Language (UML) Class diagram.

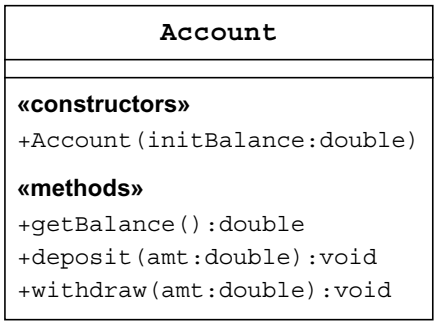

Figure 1-1 Banking Account Class

This exercise contains the following sections:

- "Task 1 – Copying the Account Class"
- "Task 2 – Creating the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

Demonstration – The demonstration for this exercise can be found in the `demos/01_intro/exercise2` directory.

Task 1 – Copying the `Account` Class

In this task, you create the `BankPrj` project, and copy the pre-provided `Account` class to the project.

Tool Reference – Java Development: Java Application Projects: Creating Projects

1. Create the `BankPrj` Java Application Project with the following characteristics:

   Project Name: **`BankPrj`**

   Project Location: **`projects`**

   Project Folder: **`projects/BankPrj`**

   Set as Main Project: **`No`**

   Create Main Class: **`No`**

Tool Reference – Java Development: Java Classes: Modifying Java Classes: Copying Java Classes

2. Copy the pre-provided `Account.java` source file from the `resources/01_intro/exercise2` directory to the source package of the `BankPrj` project.

Task 2 – Creating the `TestAccount` Class

In this task, you complete the following steps to create a new `TestAccount` Java class.

Tool Reference – Java Development: Java Classes: Creating Java Classes

1. Create a new Java class in the `BankPrj` project with the following characteristics:

   Class Name: **`TestAccount`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Edit the source file for the `TestAccount` class to add a `main` method. The `main` method of the `TestAccount` class creates an `Account` object with an initial balance of 100. It deposits 50 to and then withdraws 147 from the `Account` object. Finally, it must print out the balance of the `Account` object to the standard output stream.

Task 3 – Compiling the `TestAccount` Class

Compile the `TestAccount` class, and make necessary changes to correct compilation errors.

Task 4 – Running the `TestAccount` Program

Run the `TestAccount` program. If there are runtime errors, make necessary changes to the `TestAccount` class, recompile it, and run the program again.

The output of the `TestAccount` program should be similar to the following:

```
Final account balance is: 3.0
```

Exercise 2: Creating a Test Program (Level 2)

In this exercise, you create a test harness (a test class) to exercise a pre-provided class. These are the Level 2 instructions that provide additional hints.

This exercise contains the following sections:

- "Task 1 – Copying the Account Class"
- "Task 2 – Creating the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

Demonstration – The demonstration for this exercise can be found in the `demos/01_intro/exercise2` directory.

Task 1 – Copying the `Account` Class

In this task, you create the `BankPrj` project, and copy the pre-provided `Account` class to the project.

Tool Reference – Java Development: Java Application Projects: Creating Projects

1. Create the `BankPrj` Java Application Project with the following characteristics:

   Project Name: **`BankPrj`**

   Project Location: **`projects`**

   Project Folder: **`projects/BankPrj`**

   Set as Main Project: **`No`**

   Create Main Class: **`No`**

Tool Reference – Java Development: Java Classes: Modifying Java Classes: Copying Java Classes

2. Copy the pre-provided `Account.java` source file from the `resources/01_intro/exercise2` directory to the source package of the `BankPrj` project.

Task 2 – Creating the `TestAccount` Class

In this task, you complete the following steps to create a new `TestAccount` Java class.

Tool Reference – Java Development: Java Classes: Creating Java Classes

1. Create a Java class with the following characteristics:

   Class Name: **`TestAccount`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Add the `main` method.
3. Declare a variable in the `main` method. The variable has a type of `Account` and a name of `acct`. Initialize the variable by creating an instance of the `Account` class with an initial balance of 100.

4. Use the `deposit` method to add 50 to the account.
5. Use the `withdraw` method to subtract 147 from the account.
6. Use the `getBalance` method to retrieve the new account balance and use the `System.out.println` method to display the balance to the standard output stream.

Task 3 – Compiling the `TestAccount` Class

Compile the `TestAccount` class, and make necessary changes to correct compilation errors.

Task 4 – Running the `TestAccount` Program

Run the `TestAccount` program. If there are runtime errors, make necessary changes to the `TestAccount` class, recompile it and run the program again.

The output of the TestAccount program should be similar to the following:

```
Final account balance is: 3.0
```

Exercise 2: Creating a Test Program (Level 3)

In this exercise, you create a test harness (a test class) to exercise a pre-provided class. These are the Level 3 instructions that provide additional hints with code snippets.

This exercise contains the following sections:

- "Task 1 – Copying the Account Class"
- "Task 2 – Creating the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

Demonstration – The demonstration for this exercise can be found in the `demos/01_intro/exercise2` directory.

Task 1 – Copying the `Account` Class

In this task, you create the `BankPrj` project, and copy the pre-provided `Account` class to the project.

Tool Reference – Java Development: Java Application Projects: Creating Projects

1. Create the `BankPrj` Java Application Project with the following characteristics:

   Project Name: **`BankPrj`**

   Project Location: **`projects`**

   Project Folder: **`projects/BankPrj`**

   Set as Main Project: **`No`**

   Create Main Class: **`No`**

Tool Reference – Java Development: Java Classes: Modifying Java Classes: Copying Java Classes

2. Copy the pre-provided `Account.java` source file from the `resources/01_intro/exercise2` directory to the source package of the `BankPrj` project.

Task 2 – Creating the `TestAccount` Class

In this task, you complete the following steps to create a new `TestAccount` Java class.

Tool Reference – Java Development: Java Classes: Creating Java Classes

1. Create a Java class with the following characteristics:

   Class Name: **`TestAccount`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Add the `main` method to the `TestAccount` class:

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

3. Declare a variable of type `Account` and initialize that variable by creating an instance of the `Account` class with an initial balance of 100.

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

4. Use the `deposit` method to add 50 to the account.

```
acct.deposit(50.0);
```

5. Use the `withdraw` method to subtract 147 from the account.

```
acct.withdraw(147.0);
```

6. Use the `getBalance` method to retrieve the new account balance and use the `System.out.println` method to display the balance to the standard output stream.

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

Task 3 – Compiling the `TestAccount` Class

Compile the `TestAccount` class, and make necessary changes to correct compilation errors.

Task 4 – Running the `TestAccount` Program

Run the `TestAccount` program. If there are runtime errors, make necessary changes to the `TestAccount` class, recompile it and run the program again.

The output of the TestAccount program should be similar to the following:

```
Final account balance is: 3.0
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 2

Object-Oriented Programming

Objectives

Upon completion of this lab, you should be able to:

- Use the Java API documentation to research a class
- Explore information hiding and encapsulation
- Use packages to organize your classes

Exercise 1: Using the Java API Documentation

In this exercise, you explore the Java™ Platform, Standard Edition 6 (Java SE 6) API documentation to explore the methods of a class.

This exercise contains the following sections:

- "Task – Using the Java API Documentation"

Preparation

You must have a web browser window open.

Task – Using the Java API Documentation

Complete the following steps to use the Java API documentation:

1. Open the Java™ Platform, Standard Edition 6 API Specification Web page in the browser.

Note – The API documentation for Java Platform SE 6 can be found at `http://java.sun.com/javase/6/docs/api`.

2. Select the `java.text` package in the package list in the upper-left corner panel of the API frameset.
3. Select the `NumberFormat` class in the class list in the lower-left panel.
4. Read about the class in the top part of the documentation window on the right panel of the frameset. Review the `format` and `parse` methods.

Exercise 2: Exploring Encapsulation, Version 1 (Level 1)

In this exercise, you explore the purpose of proper *object encapsulation*. You create a class in two steps to demonstrate the use of information hiding. In this version, you 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.

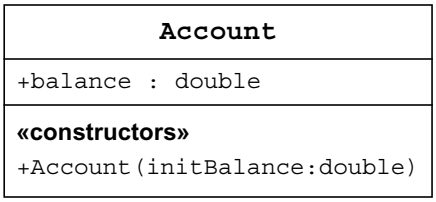

Figure 2-1 UML Class Diagram of `Account` With No Hiding

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.

This exercise contains the following sections:

- "Task 1 – Deleting the Account Class"
- "Task 2 – Creating the Account Class"
- "Task 3 – Creating the TestAccount2 Class"
- "Task 4 – Compiling the TestAccount2 Class"
- "Task 5 – Running the TestAccount2 Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise2` directory.

Task 1 – Deleting the `Account` Class

Tool Reference – Java Development: Other Files: Deleting Files

In this task, you delete the `Account` class copied in Lab 1.

Task 2 – Creating the `Account` Class

In this task, you create the `Account` class according to the UML diagram in Figure 2-1. The class should have the following characteristics:

Class Name: **`Account`**

Project: **`BankPrj`**

Location: **`Source Packages`**

Package: **`default package`**

After creating the `Account` class, add an instance variable `balance` and a constructor according to the UML diagram in Figure 2-1. Initialize the `balance` instance variable with the parameter of the constructor.

Task 3 – Creating the `TestAccount2` Class

In this task, you create the `TestAccount2` class with the following characteristics:

Class Name: **`TestAccount2`**

Project: **`BankPrj`**

Location: **`Source Packages`**

Package: **`default package`**

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.

Task 4 – Compiling the `TestAccount2` Class

In this task, you compile the `TestAccount2` class, and make necessary changes to correct compilation errors.

Task 5 – Running the `TestAccount2` Program

In this task, you run the `TestAccount2` program.

The output should be similar to the following:

```
Final account balance is -3.0
```

Exercise 2: Exploring Encapsulation, Version 1 (Level 2)

In this exercise, you explore the purpose of proper *object encapsulation*. These are the Level 2 instructions, which provide additional hints.

This exercise contains the following sections:

- "Task 1 – Deleting the Account Class"
- "Task 2 – Creating the Account Class"
- "Task 3 – Creating the TestAccount2 Class"
- "Task 4 – Compiling the TestAccount2 Class"
- "Task 5 – Running the TestAccount2 Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise2` directory.

Task 1 – Deleting the `Account` Class

Tool Reference – Java Development: Other Files: Deleting Files

In this task, you delete the `Account` class copied in Lab 1.

Task 2 – Creating the `Account` Class

In this task, you complete the following steps to create a Java class named `Account`:

1. Create the `Account` class with the following characteristics:

   Class Name: **`Account`**

   Project: **`BankPrj`**

   Location: **Source Packages**

   Package: **`default package`**
2. Add the `balance` instance variable.
3. Add a constructor that sets the `balance` instance variable to the initial balance argument passed to the constructor.

Task 3 – Creating the `TestAccount2` Class

In this task, you complete the following steps to create a Java class named `TestAccount2`:

1. Create the `TestAccount2` class with the following characteristics:

   Class Name: **`TestAccount2`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**
2. Add the `main` method:

   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.

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

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

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

Task 4 – Compiling the `TestAccount2` Class

In this task, you compile the `TestAccount2` class and the `Account` class, and make necessary changes to correct compilation errors.

Task 5 – Running the `TestAccount2` Program

In this task, you run the `TestAccount2` program.

The output should be similar to the following:

```
Final account balance is -3.0
```

Exercise 2: Exploring Encapsulation, Version 1 (Level 3)

In this exercise, you explore the purpose of proper *object encapsulation*. These are the Level 3 instructions, which provide additional hints with code snippets.

This exercise contains the following sections:

- "Task 1 – Deleting the Account Class"
- "Task 2 – Creating the Account Class"
- "Task 3 – Creating the TestAccount2 Class"
- "Task 4 – Compiling the TestAccount2 Class"
- "Task 5 – Running the TestAccount2 Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise2` directory.

Task 1 – Deleting the `Account` Class

Tool Reference – Java Development: Other Files: Deleting Files

In this task, you delete the Account class copied in Lab 1.

Task 2 – Creating the `Account` Class

In this task, you complete the following steps to create a Java class named `Account`:

1. Create the `Account` class with the following characteristics:

   Class Name: **`Account`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Add the `balance` instance variable.

```
public double balance;
```

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

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

Task 3 – Creating the `TestAccount2` Class

In this task, you complete the following steps to create a Java class named `TestAccount2`:

1. Create the `TestAccount2` class with the following characteristics:

   Class Name: **`TestAccount2`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Add the `main` method:

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

3. 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);
```

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

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

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

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

6. 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 4 – Compiling the `TestAccount2` Class

In this task, you compile the `TestAccount2` class and the `Account` class, and make necessary changes to correct compilation errors.

Task 5 – Running the `TestAccount2` Program

In this task, you run the `TestAccount2` program.

The output should be similar to the following:

```
Final account balance is -3.0
```

Exercise 3: Exploring Encapsulation, Version 2 (Level 1)

In this exercise, you explore the purpose of proper *object encapsulation*. You modify the `Account` class to hide its data member and provide public methods to manipulate the balance. You then use the test program that you created in Lab 1 to test that the business rule (*balance must not fall below zero*) is satisfied.

Figure 2-2 shows the UML class diagram of the `Account` class that you create. This design for the `Account` class hides the instance variable, `balance`, and supplies public methods to manipulate the account balance. The `deposit` method adds money to the account. The `withdraw` method removes money from the account. The `getBalance` method returns the current value of the `balance` instance variable.

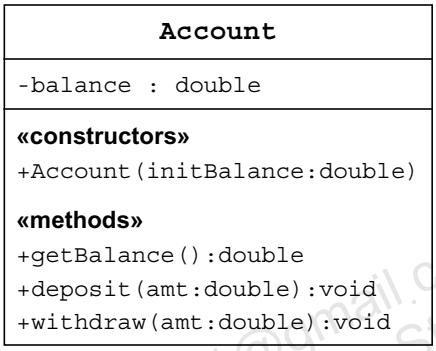

Figure 2-2 UML Class Diagram of `Account` With Information Hiding

Remember, there is still one business rule that must be maintained: *The balance of the bank account must never go below zero.* This business rule should be enforced in the `withdraw` method.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Modifying the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise3` directory.

Task 1 – Modifying the `Account` Class

In this task, you modify the `Account` class source file. This class must satisfy the UML diagram in Figure 2-2.

Task 2 – Modifying the `TestAccount` Class

In this task, you modify the `TestAccount` class to deposit 47 to and withdraw 150 from the `Account` object.

Task 3 – Compiling the `TestAccount` Class

In this task, you compile the `TestAccount` class and the `Account` class.

Task 4 – Running the `TestAccount` Program

In this task, you run the `TestAccount` program. The output should be similar to the following:

```
Final account balance is 147.0
```

The 150 withdraw command did not take effect, because it would have made the balance drop below zero. However, the `Account` object did not tell program that the withdraw command failed, it ignored the command. You will fix this problem in future exercises.

Exercise 3: Exploring Encapsulation, Version 2 (Level 2)

In this exercise, you explore the purpose of proper *object encapsulation*. These are the Level 2 instructions, which provide additional hints.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Modifying the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise3` directory.

Task 1 – Modifying the `Account` Class

In this task, you complete the following steps to modify the `Account` class:

1. Change the `balance` instance variable from `public` to `private`.
2. Add the `deposit` method that takes an amount (of type `double`) and adds that amount to the balance. Save the new balance in the instance variable.

3. Add the `withdraw` method that takes an amount (of type `double`) and subtracts that amount from the balance. Save the new balance in the instance variable.

   This operation might set the balance to a value below zero if the amount to withdraw is larger than the current balance. You must use a conditional statement to verify that the amount is not greater than the balance before doing the subtraction. The conditional statement looks like the following:

   ```
   if ( <boolean_test> ) {
     <statement_when_true>*
   } else {
     <statement_when_false>*
   }
   ```

4. Add the `getBalance` method to return the `balance` instance variable.

Task 2 – Modifying the `TestAccount` Class

In this task, you complete the following steps to modify the `TestAccount` class:

1. Change the amount in the call to the `deposit` method to `47.0`.
2. Change the amount in the call to the `withdraw` method to `150.0`.

Task 3 – Compiling the `TestAccount` Class

In this task, you compile the `TestAccount` class and the `Account` class.

Task 4 – Running the `TestAccount` Program

In this task, you run the `TestAccount` program. The output should be similar to the following:

```
Final account balance is 147.0
```

Exercise 3: Exploring Encapsulation, Version 2 (Level 3)

In this exercise, you explore the purpose of proper *object encapsulation*. These are the Level 3 instructions, which provide additional hints with code snippets.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Modifying the TestAccount Class"
- "Task 3 – Compiling the TestAccount Class"
- "Task 4 – Running the TestAccount Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise3` directory.

Task 1 – Modifying the `Account` Class

In this task, you complete the following steps to modify the `Account` class:

1. Change the `balance` instance variable from `public` to `private`.

```
private double balance;
```

2. Add the `deposit` method that takes an amount (of type `double`) and adds that amount to the balance. Save the new balance in the `balance` instance variable.

```
public void deposit(double amt) {
  balance = balance + amt;
}
```

3. Add the `withdraw` method that takes an amount (of type `double`) and subtracts that amount from the balance. Save the new balance in the `balance` instance variable.

```
public void withdraw(double amt) {
  if ( amt <= balance ) {
    balance = balance - amt;
  }
}
```

4. Add the `getBalance` method to return the `balance` instance variable.

```
public double getBalance() {
  return balance;
}
```

Task 2 – Modifying the `TestAccount` Class

In this task, you complete the following steps to modify the `TestAccount` class:

1. Change the amount in the call to the `deposit` method to `47.0`.

```
acct.deposit(47.0);
```

2. Change the amount in the call to the `withdraw` method to `150.0`.

```
acct.withdraw(150.0);
```

Task 3 – Compiling the `TestAccount` Class

In this task, you compile the `TestAccount` class and the `Account` class.

Task 4 – Running the `TestAccount` Program

In this task, you run the `TestAccount` program. The output should be similar to the following:

```
Final account balance is 147.0
```

Exercise 4: Creating Java Packages

In this exercise, you will place the `Account` class and `TestAccount` class into different packages.

Figure 2-3 shows a UML diagram of the packages that you will create in this exercise. The `Account` class is placed in the `com.mybank.domain` package and the `TestAccount` class is placed in the `com.mybank.test` package.

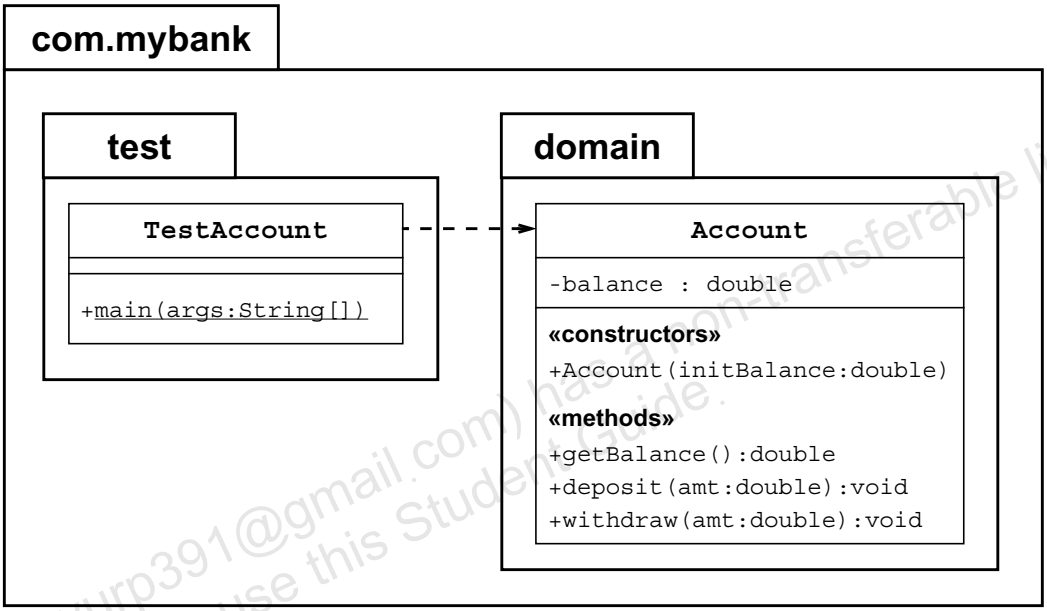

Figure 2-3 The Package Structure of the Banking Project

This exercise contains the following sections:

- "Task 1 – Creating the Java Packages"
- "Task 2 – Moving and Modifying the Account Class"
- "Task 3 – Moving the TestAccount Class"
- "Task 4 – Compiling the TestAccount Class"
- "Task 5 – Running the TestAccount Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Moving Java Classes (without refactoring)
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/02_OOP/exercise4` directory.

Task 1 – Creating the Java Packages

Tool Reference – Java Development: Java Packages: Creating Java Packages

In this task, you create two Java Source Packages in the `BankPrj` project with the following names:

`com.mybank.domain`
`com.mybank.test`

Task 2 – Moving and Modifying the `Account` Class

In this task, you complete the following steps to move the `Account` class and add the appropriate package statement to the class:

Tool Reference – Java Development: Java Classes: Moving Java Classes (without refactoring)

1. Move the `Account` class source file to the `com.mybank.domain` package.
2. Add the following package statement at the top of the `Account` class:

   ```
   package com.mybank.domain;
   ```

Task 3 – Moving the `TestAccount` Class

In this task, you complete the following steps to move the `TestAccount` class and add the appropriate package and import statements to the class:

1. Move the `TestAccount` class source file to the `com.mybank.test` package.
2. Add the following package statement at the top of the `TestAccount` class:

   ```
   package com.mybank.test;
   ```

3. Add the following import statement under the package statement in the `TestAccount` class:

   ```
   import com.mybank.domain.Account;
   ```

Task 4 – Compiling the `TestAccount` Class

In this task, you compile the `TestAccount` class and the `Account` class.

Task 5 – Running the `TestAccount` Program

In this task, you run the `TestAccount` program. The code has not changed, so the output should be similar to the following:

```
Final account balance is 147.0
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 3

Identifiers, Keywords, and Types

Objectives

Upon completion of this lab, you should be able to:

- Explore reference variable assignment
- Use a reference variable to encode an object association

Exercise 1: Investigating Reference Assignment

In this exercise, you will investigate reference variables, object creation, and reference variable assignment.

Figure 3-1 shows a class diagram for the `MyPoint` class that is provided in the exercise directory. Notice that the instance variables, `x` and `y`, are both public so you can access these data members in your test program directly. Also, the `toString` method is used when you print the object using the `System.out.println` method.

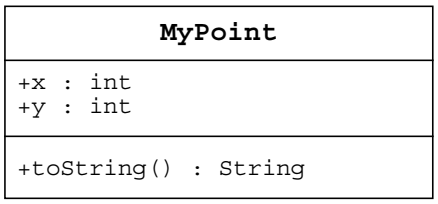

Figure 3-1 UML Class Diagram for the `MyPoint` Class

Your task is to create a test program that explores object references.

This exercise contains the following sections:

- "Task 1 – Creating the TestMyPoint Class"
- "Task 2 – Compiling the TestMyPoint Class"
- "Task 3 – Running the TestMyPoint Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `PointProject` project in the `exercises/03_types/exercise1` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/03_types/exercise1` directory.

Task 1 – Creating the `TestMyPoint` Class

Tool Reference – Java Development: Java Application Projects: Opening Projects

Complete the following steps:

1. Open the `PointProject` project in the `exercises/03_types/exercise1` directory.
2. Create the `TestMyPoint` class with the following characteristics:

   Class Name: **`TestMyPoint`**

   Project: **`PointProject`**

   Location: **`Source Packages`**

   Package: **`default package`**
3. Create a `main` method, and complete the following steps in the `main` method:
   a. Declare two variables of type `MyPoint` and called `start` and `end`. Assign both of these variables a new `MyPoint` object.
   b. Set the `x` and `y` values of `start` to 10. Set the `x` value of `end` to 20 and the `y` value to 30.
   c. Print out both point variables. Use code similar to:

      ```
      System.out.println("Start point is " + start);
      ```

   d. To make sure that you are using the `MyPoint` class correctly, you might want to compile and run `TestMyPoint` now (see "Task 2 – Compiling the TestMyPoint Class" and "Task 3 – Running the TestMyPoint Program"). If you do so, the output will look something like the following:

      ```
      Start point is [10,10]
      End point is [20,30]
      ```

   e. Declare a new variable of type `MyPoint` and call it `stray`. Assign `stray` the reference value of the existing variable `end`.
   f. Print out `stray` and `end`.

g. Assign new values to the `x` (such as 47) and `y` (such as 50) members of the variable `stray`.

h. Print out `stray`, `end`, and `start`.

Task 2 – Compiling the `TestMyPoint` Class

In this task, you compile the `TestMyPoint` class.

Task 3 – Running the `TestMyPoint` Program

In this task, you run the `TestMyPoint` program.

The output should look similar to the following:

```
Start point is [10,10]
End point is [20,30]

Stray point is [20,30]
End point is [20,30]

Stray point is [47,50]
End point is [47,50]
Start point is [10,10]
```

The values reported by `end` reflect the change made in `stray`, indicating that both variables refer to the same `MyPoint` object. However, `start` has not changed, which indicates that it is independent of the other two variables.

Exercise 2: Creating Customer Accounts (Level 1)

In this exercise, you expand the Banking project by adding a `Customer` class.

Figure 3-2 shows the UML class diagram of the `Customer` class and its relationship to the `Account` class. This relationship can be read as: A customer has one account.

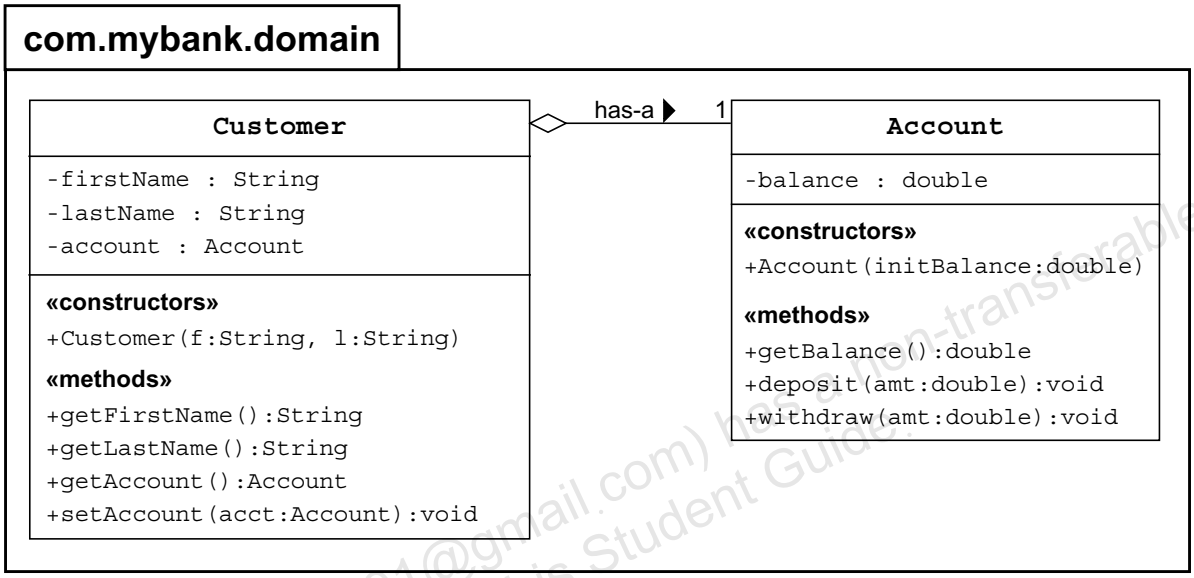

Figure 3-2 UML Class Diagram for the `Customer` Class

This exercise contains the following sections:

- "Task 1 – Creating the Customer Class"
- "Task 2 – Copying the TestBanking Class"
- "Task 3 – Compiling the TestBanking Class"
- "Task 4 – Running the TestBanking Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/03_types/exercise2` directory.

Task 1 – Creating the `Customer` Class

In this task, you create the `Customer` class to satisfy the UML diagram in Figure 3-2. The class has the following characteristics:

Class Name: **`Customer`**

Project: **`BankPrj`**

Location: **`Source Packages`**

Package: **`com.mybank.domain`**

In the constructor, initialize the `firstName` and `lastName` instance variables with the constructor parameters.

Task 2 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/03_types/exercise2` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class. If there are compilation errors, you should correct them by modifying the `Customer` class accordingly and then compile the `TestBanking` class again.

Task 4 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 2: Creating Customer Accounts (Level 2)

In this exercise you expand the Banking project by adding a `Customer` class. These are the Level 2 instructions, which provide additional hints.

This exercise contains the following sections:

- "Task 1 – Creating the Customer Class"
- "Task 2 – Copying the TestBanking Class"
- "Task 3 – Compiling the TestBanking Class"
- "Task 4 – Running the TestBanking Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/03_types/exercise2` directory.

Task 1 – Creating the `Customer` Class

In this task, you create the `Customer` class.

Complete the following steps:

1. Create the `Customer` class with the following characteristics:

   Class Name: **`Customer`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**
2. Declare three private instance variables: `firstName`, `lastName`, and `account`.
3. Declare a public constructor that takes two parameters (`f` and `l`) that populate the object instance variables.
4. Declare two public accessors for the object instance variables; the methods `getFirstName` and `getLastName` return the appropriate instance variable.
5. Declare the `getAccount` method to retrieve the `account` instance variable.
6. Declare the `setAccount` method to assign the `account` instance variable.

Task 2 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/03_types/exercise2` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class. If there are compilation errors, you should correct them by modifying the `Customer` class accordingly and then compile the `TestBanking` class again.

Task 4 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 2: Creating Customer Accounts (Level 3)

In this exercise you will expand the Banking project by adding a `Customer` class. These are the Level 3 instructions, which provide additional hints with code snippets.

This exercise contains the following sections:

- "Task 1 – Creating the Customer Class"
- "Task 2 – Copying the TestBanking Class"
- "Task 3 – Compiling the TestBanking Class"
- "Task 4 – Running the TestBanking Program"

Preparation

No preparation is needed for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/03_types/exercise2` directory.

Task 1 – Creating the `Customer` Class

In this task, you create the `Customer` class.

Complete the following steps:

1. Create the `Customer` class with the following characteristics:

   Class Name: **`Customer`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

2. Declare three private instance variables: `firstName`, `lastName`, and `account`.

```
private String firstName;
private String lastName;
private Account account;
```

3. Declare a public constructor that takes two parameters (`f` and `l`) that populate the object instance variables.

```
public Customer(String f, String l) {
  firstName = f;
  lastName = l;
}
```

4. Declare two public accessors for the object instance variables; the methods `getFirstName` and `getLastName` return the appropriate instance variable.

```
public String getFirstName() {
  return firstName;
}
public String getLastName() {
  return lastName;
}
```

5. Declare the `getAccount` method to retrieve the `account` instance variable.

```
public Account getAccount() {
  return account;
}
```

6. Declare the `setAccount` method to assign the `account` instance variable.

```
public void setAccount(Account acct) {
  account = acct;
}
```

Task 2 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/03_types/exercise2` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class. If there are compilation errors, you should correct them by modifying the `Customer` class accordingly and then compile the `TestBanking` class again.

Task 4 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00
Deposit 22.50
Withdraw 47.62
Customer [Smith, Jane] has a balance of 324.88
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 4

Expressions and Flow Control

Objectives

Upon completion of this lab, you should be able to:

- Use a simple for loop
- Using conditional statements in business logic
- (Optional) Use nested loops to implement a string search operation

Exercise 1: Using Loops and Branching Statements

In this exercise, you will use a simple integer loop and branching statements to play a fictitious game of *foo bar baz*.

Create a program that loops from 1–50 and prints each value on a separate line. Also print `foo` for every multiple of three, `bar` for every multiple of five, and `baz` for every multiple of seven. For example:

Code 4-1 (Partial) Output From the `FooBarBaz` Program

```
1
2
3 foo
4
5 bar
6 foo
7 baz
8
9 foo
10 bar
11
12 foo
13
14 baz
15 foo bar
16
```

and so on.

This exercise contains the following sections:

- "Task 1 – Creating the FooBarBaz Class"
- "Task 2 – Compiling the FooBarBaz Class"
- "Task 3 – Running the FooBarBaz Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `LoopProject` project in the `exercises/04_stmts/exercise1` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/04_stmts/exercise1` directory.

Task 1 – Creating the `FooBarBaz` Class

Complete the following steps:

1. Open the `LoopProject` project in the `exercises/04_stmts/exercise1` directory.
2. Create the `FooBarBaz` class with the following characteristics:

   Class Name: **`FooBarBaz`**

   Project: **`LoopProject`**

   Location: **`Source Packages`**

   Package: **`default package`**
3. Declare the `main` method.
4. Use a `for` loop to iterate from 1–50 in the `main` method.
   a. Print the current number.
   b. Use three `if` statements to test if the current number is divisible by three, five, or seven; if so, then print `foo`, `bar`, and `baz` as necessary.

Task 2 – Compiling the `FooBarBaz` Class

In this task, you compile the `FooBarBaz` program.

Task 3 – Running the `FooBarBaz` Program

In this task, you run the `FooBarBaz` program.

The output should be similar to Code 4-1 on page L4-2.

Hints

These hints might help you to solve this exercise:

- Use the `System.out.print` method to print a string or value without printing a new line character. You can use multiple `print` methods to print a single line of text. You can use a single `println` method, with no arguments, to print a new line character.
- The `%` operator calculates an integer remainder.

Exercise 2: Using Conditional Statements in the Account Class (Level 1)

In this exercise, you modify the withdraw method to return a boolean value to specify whether the operation was successful.

Figure 4-1 shows the UML class diagram of the Banking Project. The Account class now includes a design change to the deposit and withdraw methods; these methods must now return a boolean value to indicate the success (true) or failure (false) of the operation.

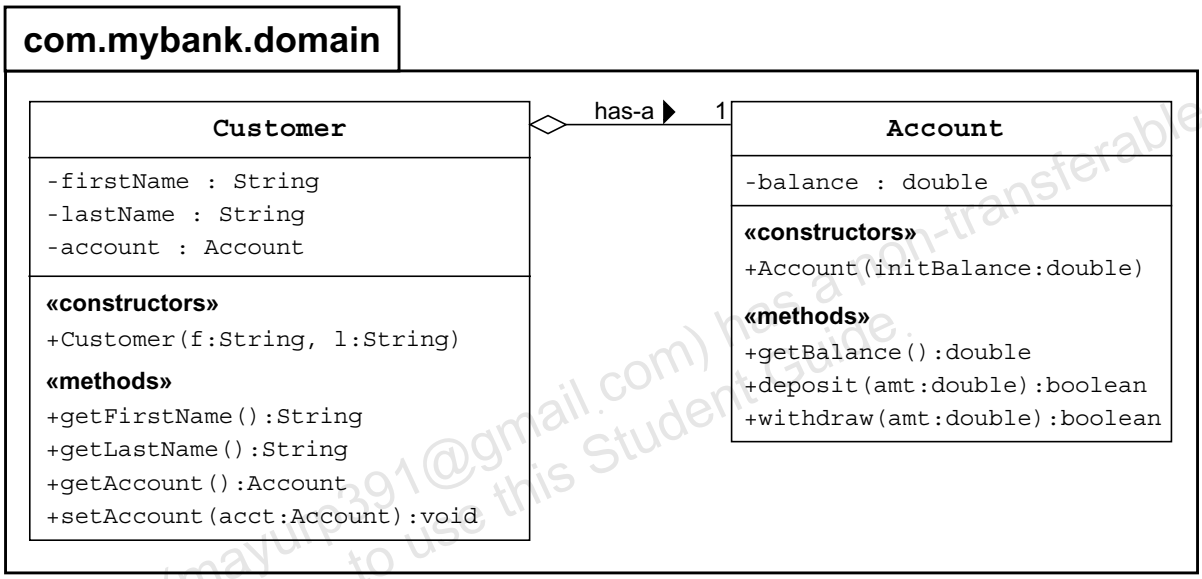

Figure 4-1 UML Class Diagram of Banking Project

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Executing Java Programs

For this exercise, you work in the BankPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/04_stmts/exercise2 directory.

Task 1 – Modifying the Account Class

In this task, you modify the Account class source file. This class must satisfy the UML diagram in Figure 4-1. In particular, the deposit and withdraw methods must return a boolean value as described previously in this exercise.

Task 2 – Deleting the Current TestBanking Class

In this task, you delete the current TestBanking class in the com.mybank.test package of the BankPrj project.

Task 3 – Copying the TestBanking Class

In this task, you copy the new version of the TestBanking.java file from the resources/04_stmts/ directory into the com.mybank.test package of the BankPrj project.

Task 4 – Compiling the TestBanking Class

In this task, you compile the TestBanking class.

Task 5 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 2: Using Conditional Statements in the Account Class (Level 2)

In this exercise, you will modify the withdraw method to return a Boolean value to specify whether the operation was successful.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the BankPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/04_stmts/exercise2 directory.

Task 1 – Modifying the Account Class

In this task, you complete the following steps to modify the Account class:

1. Modify the deposit method. Because an account has no upper limit, this method must always return true to indicate it always succeeds.
2. Modify the withdraw method. Because an account cannot drop below zero, the method must check that the amount of the withdrawal is less than the current balance. If this is the case, then the withdraw operation is successful and must return true; otherwise the operation fails and the method must return false.

Task 2 – Deleting the Current TestBanking Class

In this task, you delete the current TestBanking class in the com.mybank.test package of the BankPrj project.

Task 3 – Copying the TestBanking Class

In this task, you copy the new version of the TestBanking.java file from the resources/04_stmts/ directory into the com.mybank.test package of the BankPrj project.

Task 4 – Compiling the TestBanking Class

In this task, you compile the TestBanking class.

Task 5 – Running the TestBanking Program

In this task, you run the TestBanking program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 2: Using Conditional Statements in the Account Class (Level 3)

In this exercise, you will modify the withdraw method to return a boolean value to specify whether the operation was successful.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the BankPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/04_stmts/exercise2 directory.

Task 1 – Modifying the Account Class

In this task, you complete the following steps to modify the Account class:

1. Modify the deposit method. Because an account has no upper limit, this method must always return true to indicate it always succeeds.

```
public boolean deposit(double amt) {
  balance = balance + amt;
  return true;
}
```

2. Modify the withdraw method. Because an account cannot drop below zero, the method must check that the amount of the withdrawal is less than the current balance. If this is the case, then the withdraw operation is successful and must return true; otherwise the operation fails and the method must return false.

```
public boolean withdraw(double amt) {
  boolean result = false;  // assume operation failure
  if ( amt <= balance ) {
    balance = balance - amt;
    result = true;  // operation succeeds
  }
  return result;
}
```

Task 2 – Deleting the Current TestBanking Class

In this task, you delete the current TestBanking class in the com.mybank.test package of the BankPrj project.

Task 3 – Copying the TestBanking Class

In this task, you copy the new version of the TestBanking.java file from the resources/04_stmts/ directory into the com.mybank.test package of the BankPrj project.

Task 4 – Compiling the TestBanking Class

In this task, you compile the TestBanking class.

Task 5 – Running the TestBanking Program

In this task, you run the TestBanking program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her account with a 500.00 balance.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Smith, Jane] has a balance of 324.88
```

Exercise 3: Using Nested Loops (Advanced)

In this exercise, you use nested loops to implement a string search operation.

This exercise contains the following sections:

- "Task 1 – Writing the isSubString Method"
- "Task 2 – Compiling the TestIsSubString Class"
- "Task 3 – Running the TestIsSubString Program"

Note – There are several advanced exercises in this course. These exercises are optional and should only be attempted if you completed all of the previous exercises for this module.

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `AdvancedLoopProject` project in the `exercises/04_stmts/exercise3` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/04_stmts/exercise3` directory.

Task 1 – Writing the `isSubString` Method

In this task, you open the `AdvancedLoopProject` project in the `exercises/04_stmts/exercise3` directory, open the `TestIsSubString` class, and write the `isSubString` method. The method searches for a specific string within another string; the method must return true if the former exists in the latter string. For example: `isSubString("cat", "The cat in the hat.")` is `true`, but `isSubString("bat", "The cat in the hat.")` is `false`.

Also, verify that the following boundary conditions are met:

- `isSubString("The", "The cat in the hat.")` is `true`
- `isSubString("hat.", "The cat in the hat.")` is `true`

There is an easy way to do this by using the `indexOf` method of the `String` class. You can determine if a `String s` is a substring of some `String x` by evaluating the expression `x.indexOf(s) != -1`. However, the purpose of this exercise is to practice using loops, so you should solve the problem by using only the `charAt` method and nested loops.

Task 2 – Compiling the `TestIsSubString` Class

In this task, you compile the `TestIsSubString` class.

Task 3 – Running the `TestIsSubString` Program

In this task, you run the `TestIsSubString` program. The output of the program should be similar to the example shown in “Task 1 – Writing the isSubString Method” on page L4-14.

Hints

These hints might help you to solve this exercise.

- Use the `charAt(int index)` method in the `String` class to retrieve a specific character from a string; the index starts with zero. For example. `"cat".charAt(0)` is `'c'`, `"cat".charAt(1)` is `'a'`, and `"cat".charAt(2)` is `'t'`.
- The `length` method returns the number of characters in the string; for example, `"cat".length()` is `3`.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 5

Arrays

Objectives

Upon completion of this lab, you should be able to:

- Declare, create, and manipulate one-dimensional primitive arrays
- Use an array to represent a one-to-many object relationship

Exercise 1 – Using Primitive Arrays (Level 2)

In this exercise, you declare, create, and manipulate one-dimensional arrays of primitive types.

This exercise contains the following sections:

- "Task 1 – Creating the TestArrays Class"
- "Task 2 – Compiling the TestArrays Class"
- "Task 3 – Running the TestArrays Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `ArrayProject` project in the `exercises/05_arrays/exercise1` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/05_arrays/exercise1` directory.

Task 1 – Creating the `TestArrays` Class

Complete the following steps:

1. Open the `ArrayProject` project in the `exercises/05_arrays/exercise1` directory.

2. Create the `TestArrays` class with the following characteristics:

   Class Name: **`TestArrays`**

   Project: **`ArrayProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

3. Add a `main` method. In the `main` method, declare two variables called `array1` and `array2`. They should be of type `int[]` (array of `int`).

4. Using the curly-brace notation, {}, initialize `array1` to the first eight prime numbers: 2, 3, 5, 7, 11, 13, 17, and 19.

5. Display the contents of `array1`. You might want to use the `printArray` method (see "Hint" on page L5-4) to display these integer arrays.

6. Assign the `array2` variable equal to the `array1`. Modify the even indexed element in `array2` to be equal to the index value (for example, `array2[0] = 0;` and `array2[2] = 2;` and so on). Print out `array1`.

Task 2 – Compiling the `TestArrays` Class

In this task, you compile the `TestArrays` class.

Task 3 – Running the `TestArrays` Program

In this task, you run the `TestArrays` program. The output should be similar to the following:

```
array1 is <2, 3, 5, 7, 11, 13, 17, 19>
array1 is <0, 3, 2, 7, 4, 13, 6, 19>
```

Discussion – What has happened to `array1`?

Hint

The `printArray` support method might help you to solve this exercise, as follows:

```
public static void printArray(int[] array) {
  System.out.print('<');
  for ( int i = 0; i < array.length; i++ ) {
    // print an element
    System.out.print(array[i]);
    // print a comma delimiter if not the last element
    if ( (i + 1) < array.length ) {
      System.out.print(", ");
    }
  }
  System.out.print('>');
}
```

Note – You can also use the `java.util.Arrays.toString` method to generate a string representation of an array. For example:
`System.out.println(Arrays.toString(array1));`

Exercise 1 – Using Primitive Arrays (Level 3)

In this exercise, you declare, create, and manipulate one-dimensional arrays of primitive types.

This exercise contains the following sections:

- "Task 1 – Creating the TestArrays Class"
- "Task 2 – Compiling the TestArrays Class"
- "Task 3 – Running the TestArrays Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `exercises/05_arrays/exercise1` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/05_arrays/exercise1` directory.

Task 1 – Creating the `TestArrays` Class

Complete the following steps:

1. Open the `ArrayProject` project in the `exercises/05_arrays/exercise1` directory.

2. Create the `TestArrays` class with the following characteristics:

   Class Name: **`TestArrays`**

   Project: **`ArrayProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

3. Add a `main` method. In the `main()` method, declare two variables called `array1` and `array2`. They should be of type `int[]` (array of `int`).

```
public class TestArrays {
  public static void main(String[] args) {
    int[]  array1;
    int[]  array2;
    // insert code here
  }
}
```

4. Using the curly-brace notation, {}, initialize `array1` to the first eight prime numbers: 2, 3, 5, 7, 11, 13, 17, and 19.

```
    int[]  array1 = { 2, 3, 5, 7, 11, 13, 17, 19 };
    int[]  array2;
```

5. Display the contents of `array1`. You might want to use the `printArray` method (see "Hint" on page L5-4) to display these integer arrays.

```
    System.out.print("array1 is ");
    printArray(array1);
    System.out.println();
```

6. Assign the `array2` variable equal to the `array1`. Modify the even indexed element in `array2` to be equal to the index value (for example, `array2[0] = 0;` and `array2[2] = 2;` and so on). Print out `array1`.

```
    array2 = array1;
    // modify array2
    array2[0] = 0;
    array2[2] = 2;
    array2[4] = 4;
    array2[6] = 6;
    // print array 1
    System.out.print("array1 is ");
    printArray(array1);
    System.out.println();
```

Task 2 – Compiling the `TestArrays` Class

In this task, you compile the `TestArrays` class.

Task 3 – Running the `TestArrays` Program

In this task, you run the `TestArrays` program. The output should be similar to the following:

```
array1 is <2, 3, 5, 7, 11, 13, 17, 19>
array1 is <0, 3, 2, 7, 4, 13, 6, 19>
```

Exercise 2 – Using Arrays to Represent One-to-Many Associations (Level 1)

In this exercise, you use arrays to implement the association between a bank and its multiple customers.

Figure 5-1 shows the UML class diagram of the Banking Project. Your assignment is to create the `Bank` class. A bank object keeps track of an association between itself and its customers. You implement this one-to-many association with an array of customer objects. You will also need to keep an integer instance variable that keeps track of how many customers currently exist in the bank.

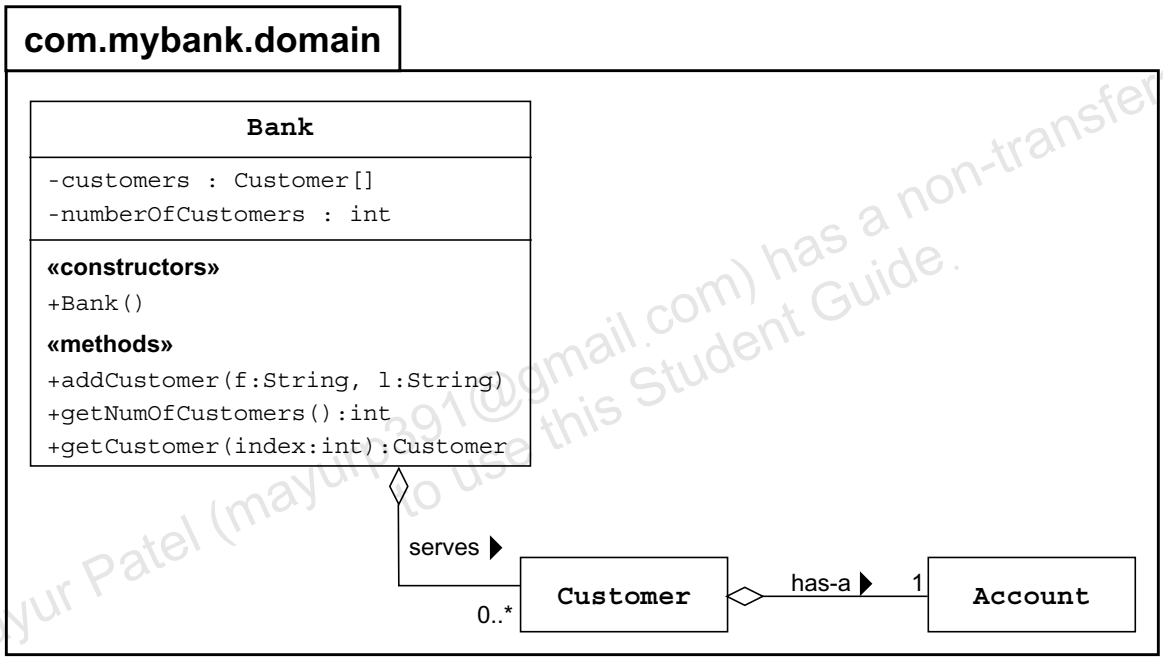

Figure 5-1 UML Class Diagram of Banking Project

This exercise contains the following sections:

- "Task 1 – Creating the Bank Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/05_arrays/exercise2` directory.

Task 1 – Creating the `Bank` Class

In this task, you create the `Bank` class that must satisfy the UML diagram in Figure 5-1 on page L5-8. The class has the following characteristics:

Class Name: **`Bank`**

Project: **`BankPrj`**

Location: **`Source Packages`**

Package: **`com.mybank.domain`**

Task 2 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/05_arrays` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 5 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Customer [1] is Simms, Jane
Customer [2] is Bryant, Owen
Customer [3] is Soley, Tim
Customer [4] is Soley, Maria
```

Exercise 2 – Using Arrays to Represent One-to-Many Associations (Level 2)

In this exercise, you use arrays to implement the association between a bank and its multiple customers.

This exercise contains the following sections:

- "Task 1 – Creating the Bank Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/05_arrays/exercise2` directory.

Task 1 – Creating the `Bank` Class

Complete the following steps:

1. Create the `Bank` class with the following characteristics:

   Class Name: **`Bank`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

2. Add two instance variables to the `Bank` class: `customers` (an array of `Customer` objects) and `numberOfCustomers` (an integer that keeps track of the next `customers` array index).
3. Add a public constructor that initializes the `customers` array with some appropriate maximum size (at least bigger than five).
4. Add the `addCustomer` method. This method must construct a new `Customer` object from the parameters (first name and last name) and place it on the `customers` array. It must also increment the `numberOfCustomers` instance variable.
5. Add the `getNumOfCustomers` accessor method, which returns the `numberOfCustomers` instance variable.
6. Add the `getCustomer` method. This method returns the customer associated with the given `index` parameter.

Task 2 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/05_arrays` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 5 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Customer [1] is Simms, Jane
Customer [2] is Bryant, Owen
Customer [3] is Soley, Tim
Customer [4] is Soley, Maria
```

Exercise 2 – Using Arrays to Represent One-to-Many Associations (Level 3)

In this exercise, you use arrays to implement the association between a bank and its multiple customers.

This exercise contains the following sections:

- "Task 1 – Creating the Bank Class"
- "Task 2 – Deleting the Current TestBanking Class"
- "Task 3 – Copying the TestBanking Class"
- "Task 4 – Compiling the TestBanking Class"
- "Task 5 – Running the TestBanking Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool References used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/05_arrays/exercise2` directory.

Task 1 – Creating the Bank Class

Complete the following steps:

1. Create the Bank class with the following characteristics:

   Class Name: **Bank**

   Project: **BankPrj**

   Location: **Source Packages**

   Package: **com.mybank.domain**

```
package com.mybank.domain;
public class Bank {
  // insert code here
}
```

2. Add two instance variables to the Bank class: customers (an array of Customer objects) and numberOfCustomers (an integer that keeps track of the next customers array index).

```
public class Bank {
  private Customer[] customers;
  private int        numberOfCustomers;
  // insert methods here
}
```

3. Add a public constructor that initializes the customers array with some appropriate maximum size (at least bigger than five).

```
  public Bank() {
    customers = new Customer[10];
    numberOfCustomers = 0;
  }
```

4. Add the addCustomer method. This method must construct a new Customer object from the parameters (first name, last name) and place it on the customers array. It must also increment the numberOfCustomers instance variable.

```
  public void addCustomer(String f, String l) {
    int i = numberOfCustomers++;
    customers[i] = new Customer(f, l);
  }
```

5. Add the getNumOfCustomers accessor method, which returns the numberOfCustomers instance variable.

```
  public int getNumOfCustomers() {
    return numberOfCustomers;
  }
```

6. Add the `getCustomer` method. This method returns the customer associated with the given `index` parameter.

```
public Customer getCustomer(int customer_index) {
  return customers[customer_index];
}
```

Task 2 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 3 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/05_arrays` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 5 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Customer [1] is Simms, Jane
Customer [2] is Bryant, Owen
Customer [3] is Soley, Tim
Customer [4] is Soley, Maria
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 6

Class Design

Objectives

Upon completion of this lab, you should be able to:

- Create subclasses with inheritance and overriding of methods
- Create a heterogeneous collection and use polymorphism

Exercise 1: Creating Bank Account Subclasses (Level 1)

In this exercise, you create two subclasses of the `Account` class in the Banking project: `SavingsAccount` and `CheckingAccount`. These account types have the following business rules:

- A savings account gains interest. The bank permits customers to store money in a savings account and, on a monthly basis, the savings account will accumulate based on the following formula: balance = balance + (interestRate * balance).
- A checking account enables the customer to make any number of deposits and withdrawals. To protect their customers, the bank will permit a fixed amount of *overdraft protection*. This protection enables the customer's balance to drop below zero, but not below the amount of overdraft protection. The account's overdraft amount is decremented as it is used.

Figure 6-1 shows the UML class diagram for a design that satisfies the business rules described above.

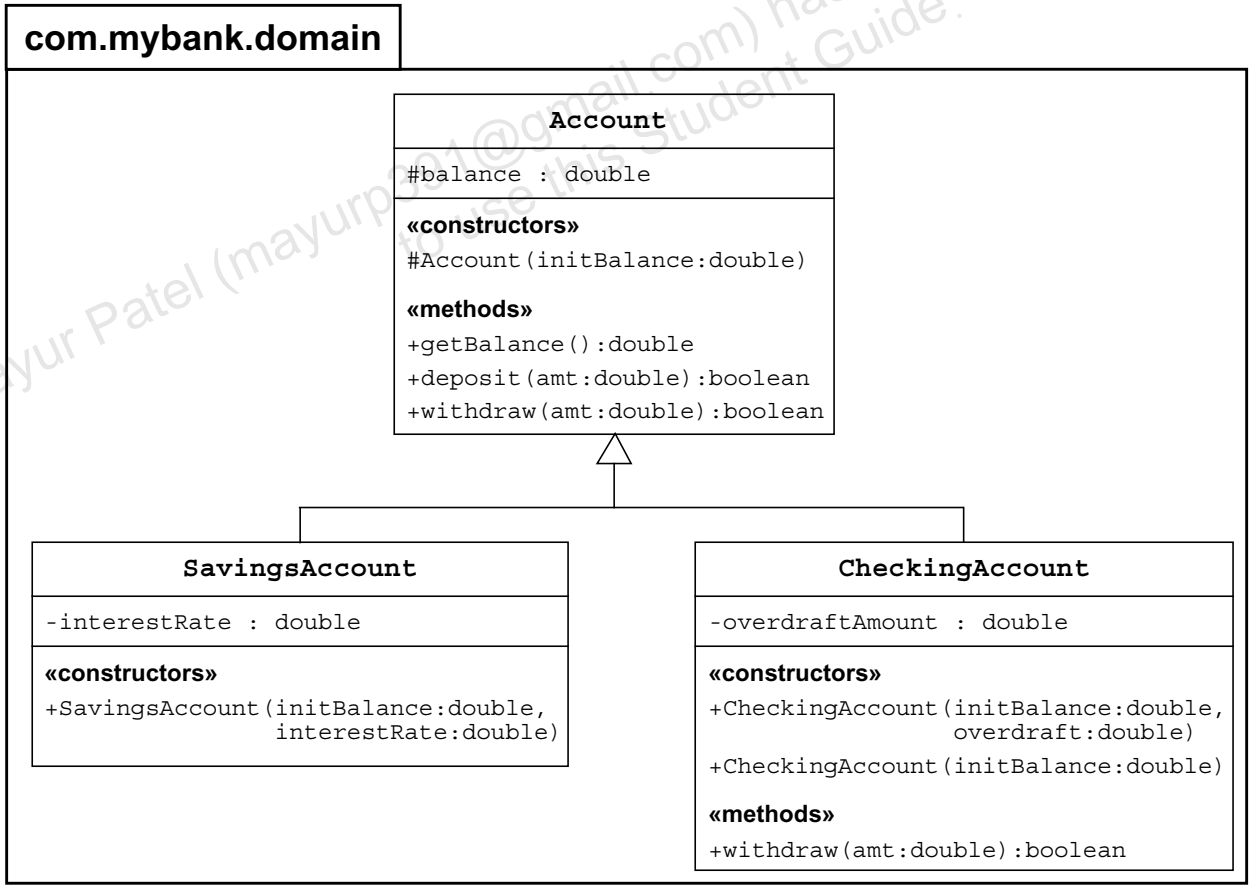

Figure 6-1 Two Subclasses of the `Account` Class

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Creating the SavingsAccount Class"
- "Task 3 – Creating the CheckingAccount Class"
- "Task 4 – Deleting the Current TestBanking Class"
- "Task 5 – Copying the TestBanking Class"
- "Task 6 – Compiling the TestBanking Class"
- "Task 7 – Running the TestBanking Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise1` directory.

Task 1 – Modifying the `Account` Class

In this task, you modify the `Account` class in the `com.mybank.domain` source package of the `BankPrj` project. This class must satisfy the UML diagram in Figure 6-1 on page L6-2; in particular, the `balance` instance variable and `Account` class constructor are now *protected* (indicated by the # character instead of the - character).

Task 2 – Creating the `SavingsAccount` Class

In this task, you create the `SavingsAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

Class Name: **`SavingsAccount`**

Project: **`BankPrj`**

Extends: **`Account`**

Location: **`Source Packages`**

Package: **`com.mybank.domain`**

This class must satisfy the UML diagram in Figure 6-1 on page L6-2 and the business rules defined in the introduction to "Exercise 1: Creating Bank Account Subclasses (Level 1)" on page L6-2.

Task 3 – Creating the `CheckingAccount` Class

In this task, you create the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

Class Name: **`CheckingAccount`**

Project: **`BankPrj`**

Extends: **`Account`**

Location: **`Source Packages`**

Package: **`com.mybank.domain`**

This class must satisfy the UML diagram in Figure 6-1 on page L6-2 and the business rules defined in the introduction to "Exercise 1: Creating Bank Account Subclasses (Level 1)" on page L6-2.

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project. The new `TestBanking` class sets `CheckingAccount` and `SavingsAccount` objects to different customers.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her Savings Account with a 500.00 balance and 3% interest.
Creating the customer Owen Bryant.
Creating his Checking Account with a 500.00 balance and no overdraft
protection.
Creating the customer Tim Soley.
Creating his Checking Account with a 500.00 balance and 500.00 in
overdraft protection.
Creating the customer Maria Soley.
Maria shares her Checking Account with her husband Tim.

Retrieving the customer Jane Smith with her savings account.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Simms, Jane] has a balance of 324.88
```

```
Retrieving the customer Owen Bryant with his checking account with no
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Bryant, Owen] has a balance of 324.88

Retrieving the customer Tim Soley with his checking account that has
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: true
Customer [Soley, Tim] has a balance of 0.0

Retrieving the customer Maria Soley with her joint checking account with
husband Tim.
Deposit 150.00: true
Withdraw 750.00: false
Customer [Soley, Maria] has a balance of 150.0
```

Jane's savings account and Owen's checking account behave fundamentally as a basic bank account. But Tim and Maria's joint checking account has 500.00 worth of overdraft protection. Tim's transactions dip into that protection and therefore his ending balance is 0.00. His account's overdraft protection level is 424.88. Finally, Maria deposits 150.00 into this joint account; raising the balance from 0.00 to 150.00. Then she tries to withdraw 750.00, which fails because neither the balance nor the overdraft protection can cover that requested amount.

Exercise 1: Creating Bank Account Subclasses (Level 2)

In this exercise, you create two subclasses of the `Account` class in the Banking project: `SavingsAccount` and `CheckingAccount`.

This section contains the following sections:

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise1` directory.

Task 1 – Modifying the `Account` Class

In this task, you modify the `Account` class in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Change the `balance` instance variable from `private` to `protected`.
2. Change the `Account` constructor from `public` to `protected`.

Task 2 – Creating the `SavingsAccount` Class

In this task, you create the `SavingsAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Create the `SavingsAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`SavingsAccount`**

   Project: **`BankPrj`**

   Extends: **`Account`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**
2. Add the `interestRate` instance variable to the `SavingsAccount` class.
3. Add a public constructor that takes two arguments: `initBalance` and `interestRate`. Pass the `initBalance` parameter to the super constructor. Save the `interestRate` parameter to the instance variable.

Task 3 – Creating the `CheckingAccount` Class

In this task, you create the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Create the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`CheckingAccount`**

   Project: **`BankPrj`**

   Extends: **`Account`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

2. Add the `overdraftAmount` instance variable to the `CheckingAccount` class.

3. Add a public constructor that takes two arguments: `initBalance` and `overdraftAmount`. Pass the `initBalance` parameter to the super constructor. Save the `overdraftAmount` parameter to the instance variable.

4. Add a second public constructor that takes only one argument: `initBalance`. Call the first constructor with the `initBalance` parameter and use the default value `0.0` for the `overdraftAmount` parameter.

5. Override the `withdraw` method to use the `overdraftAmount` variable. Here is the pseudo-code for the `withdraw` method:

```
if balance < amount
  then
    overdraftNeeded = amount - balance
    if overdraftAmount < overdraftNeeded
      then transaction fails
      else
        balance = 0
        decrement overdraftAmount by overdraftNeeded
  else
    decrement balance by amount
```

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project. The new `TestBanking` class sets `CheckingAccount` and `SavingsAccount` objects to different customers.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her Savings Account with a 500.00 balance and 3% interest.
Creating the customer Owen Bryant.
Creating his Checking Account with a 500.00 balance and no overdraft
protection.
Creating the customer Tim Soley.
Creating his Checking Account with a 500.00 balance and 500.00 in
overdraft protection.
Creating the customer Maria Soley.
Maria shares her Checking Account with her husband Tim.

Retrieving the customer Jane Smith with her savings account.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Simms, Jane] has a balance of 324.88
```

```
Retrieving the customer Owen Bryant with his checking account with no
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Bryant, Owen] has a balance of 324.88

Retrieving the customer Tim Soley with his checking account that has
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: true
Customer [Soley, Tim] has a balance of 0.0

Retrieving the customer Maria Soley with her joint checking account with
husband Tim.
Deposit 150.00: true
Withdraw 750.00: false
Customer [Soley, Maria] has a balance of 150.0
```

Exercise 1: Creating Bank Account Subclasses (Level 3)

In this exercise, you will create two subclasses of the `Account` class in the Banking project: `SavingsAccount` and `CheckingAccount`.

This exercise contains the following sections:

- "Task 1 – Modifying the Account Class"
- "Task 2 – Creating the SavingsAccount Class"
- "Task 3 – Creating the CheckingAccount Class"
- "Task 4 – Deleting the Current TestBanking Class"
- "Task 5 – Copying the TestBanking Class"
- "Task 6 – Compiling the TestBanking Class"
- "Task 7 – Running the TestBanking Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise1` directory.

Task 1 – Modifying the `Account` Class

In this task, you modify the `Account` class in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Change the `balance` instance variable from `private` to `protected`.

```
protected double balance;
```

2. Change the `Account` constructor from `public` to `protected`.

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

Task 2 – Creating the `SavingsAccount` Class

In this task, you create the `SavingsAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Create the `SavingsAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`SavingsAccount`**

   Project: **`BankPrj`**

   Extends: **`Account`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

```
package com.mybank.domain;
public class SavingsAccount extends Account {
  // insert code here
}
```

2. Add the `interestRate` instance variable to the `SavingsAccount` class.

```
private double interestRate;
```

3. Add a public constructor that takes two arguments: `initBalance` and `interestRate`. Pass the `initBalance` parameter to the super constructor. Save the `interestRate` parameter to the instance variable.

```
public SavingsAccount(double initBalance, double interestRate) {
  super(initBalance);
  this.interestRate = interestRate;
}
```

Task 3 – Creating the `CheckingAccount` Class

In this task, you create the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Create the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`CheckingAccount`**

   Project: **`BankPrj`**

   Extends: **`Account`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

```
package com.mybank.domain;
public class CheckingAccount extends Account {
  // insert code here
}
```

2. Add the `overdraftAmount` instance variable to the `CheckingAccount` class.

```
private double overdraftAmount;
```

3. Add a public constructor that takes two arguments: `initBalance` and `overdraftAmount`. Pass the `initBalance` parameter to the super constructor. Save the `overdraftAmount` parameter to the instance variable.

```
public CheckingAccount(double initBalance, double overdraftAmount) {
  super(initBalance);
  this.overdraftAmount = overdraftAmount;
}
```

4. Add a second public constructor that takes only one argument: `initBalance`. Call the first constructor with the `initBalance` parameter and use the default value `0.0` for the `overdraftAmount` parameter.

```
public CheckingAccount(double initBalance) {
  this(initBalance, 0.0);
}
```

5. Override the `withdraw` method to use the `overdraftAmount` variable.

```
public boolean withdraw(double amount) {
  boolean result = true;
  if ( balance < amount ) {
    double overdraftNeeded = amount - balance;
    if ( overdraftAmount < overdraftNeeded ) {
      result = false;
    } else {
      balance = 0.0;
      overdraftAmount -= overdraftNeeded;
    }
  } else {
    balance -= amount;
  }
  return result;
}
```

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project. The new `TestBanking` class sets `CheckingAccount` and `SavingsAccount` objects to different customers.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Creating the customer Jane Smith.
Creating her Savings Account with a 500.00 balance and 3% interest.
Creating the customer Owen Bryant.
Creating his Checking Account with a 500.00 balance and no overdraft
protection.
Creating the customer Tim Soley.
Creating his Checking Account with a 500.00 balance and 500.00 in
overdraft protection.
Creating the customer Maria Soley.
Maria shares her Checking Account with her husband Tim.

Retrieving the customer Jane Smith with her savings account.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Simms, Jane] has a balance of 324.88

Retrieving the customer Owen Bryant with his checking account with no
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Bryant, Owen] has a balance of 324.88

Retrieving the customer Tim Soley with his checking account that has
overdraft protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: true
Customer [Soley, Tim] has a balance of 0.0
```

```
Retrieving the customer Maria Soley with her joint checking account with
husband Tim.
Deposit 150.00: true
Withdraw 750.00: false
Customer [Soley, Maria] has a balance of 150.0
```

Exercise 2: Creating a Heterogeneous Collection of Customer Accounts (Level 1)

In this exercise, you create a heterogeneous array to represent the aggregation of customers to accounts. That is, a given customer can have several accounts of different types.

Figure 6-2 shows the UML class diagram of the relationships between bank, customers, and accounts. What has changed is that a `Customer` object may now have more than one account and these accounts may be of different types, subclasses of the `Account` class.

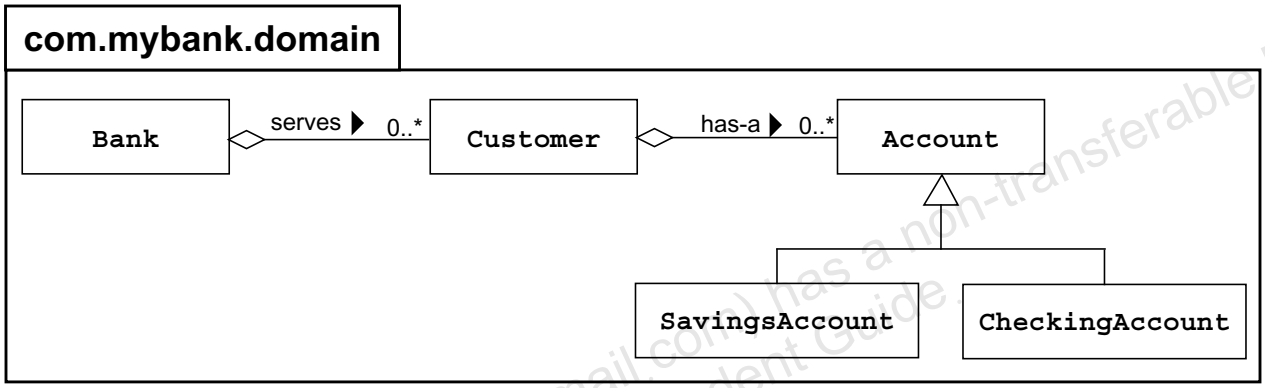

Figure 6-2 Customer With One or More Different Accounts

Your job is to modify the `Customer` class to support a heterogeneous collection of `Account` objects.

This exercise contains the following sections:

- "Task 1 – Modifying the Customer Class"
- "Task 2 – Copying and Completing the CustomerReport Class"
- "Task 3 – Copying the TestReport Class"
- "Task 4 – Compiling the TestReport Class"
- "Task 5 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise2` directory.

Task 1 – Modifying the `Customer` Class

In this task, you modify the `Customer` class in the `com.mybank.domain` source package of the `BankPrj` project. Modify the `Customer` class to handle the association with multiple accounts, just as you did in the Exercise 2 of Lab 5. It must include the public methods: `addAccount(Account)`, `getAccount(int)`, and `getNumOfAccounts()`.

Task 2 – Copying and Completing the `CustomerReport` Class

Complete the following steps:

1. Create the `com.mybank.report` source package in the `BankPrj` project.
2. Copy the `CustomerReport.java` file from the `resources/06_class1` directory into the `com.mybank.report` source package of the `BankPrj` project.

3. Complete the `CustomerReport.java` code. You will find comment blocks that start and end with `/*** ... ***/`. These comments indicate the location in the code that you must supply.

Task 3 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 5 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the following:

```
                    CUSTOMERS REPORT
                    ================


Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Checking Account: current balance is 200.0
    Savings Account: current balance is 150.0
```

Exercise 2: Creating a Heterogeneous Collection of Customer Accounts (Level 2)

In this exercise, you create a heterogeneous array to represent the aggregation of customers to accounts. That is, a given customer can have several accounts of different types.

This exercise contains the following sections:

- "Task 1 – Modifying the Customer Class"
- "Task 2 – Copying and Completing the CustomerReport Class"
- "Task 3 – Copying the TestReport Class"
- "Task 4 – Compiling the TestReport Class"
- "Task 5 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise2` directory.

Task 1 – Modifying the `Customer` Class

In this task, you modify the `Customer` class in the `com.mybank.domain` source package of the `BankPrj` project. Modify the `Customer` class to handle the association with multiple accounts, just as you did in the Exercise 2 of Lab 5. It must include the public methods: `addAccount(Account)`, `getAccount(int)`, and `getNumOfAccounts()`.

Complete the following steps:

1. Add two instance variables to the `Customer` class: `accounts` (an array of `Account` objects) and `numberOfAccounts` (an integer that keeps track of the next `accounts` array index). This replaces the single `account` reference variable, which should be removed.
2. Modify the constructor to initialize the `accounts` array.
3. Add the `addAccount` method. This method takes a single parameter, an `Account` object, and stores it in the `accounts` array. It must also increment the `numberOfAccounts` instance variable. This method replaces the `setAccount` method, which should be removed.
4. Add the `getNumOfAccounts` accessor method, which returns the `numberOfAccounts` instance variable.
5. Add the `getAccount` method. This method returns the account associated with the given `index` parameter. This method replaces the previous `getAccount` method, which should be removed.

Task 2 – Copying and Completing the `CustomerReport` Class

Complete the following steps:

1. Create the `com.mybank.report` source package in the `BankPrj` project.
2. Copy the `CustomerReport.java` file from the `resources/06_class1` directory into the `com.mybank.report` source package of the `BankPrj` project.

3. Complete the `CustomerReport.java` code. You will find comment blocks that start and end with `/*** ... ***/`. These comments indicate the location in the code that you must supply.

   a. Use the `instanceof` operator to test what type of account this is and set `account_type` to an appropriate value, such as `Savings Account` or `Checking Account`.

   b. Print out the type of account and the balance.

Task 3 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 5 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the output listed on page L6-20.

Exercise 2: Creating a Heterogeneous Collection of Customer Accounts (Level 3)

In this exercise, you create a heterogeneous array to represent the aggregation of customers to accounts. That is, a given customer can have several accounts of different types.

This exercise contains the following sections:

- "Task 1 – Modifying the Customer Class"
- "Task 2 – Copying and Completing the CustomerReport Class"
- "Task 3 – Copying the TestReport Class"
- "Task 4 – Compiling the TestReport Class"
- "Task 5 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/06_class1/exercise2` directory.

Task 1 – Modifying the `Customer` Class

In this task, you modify the `Customer` class in the `com.mybank.domain` source package of the `BankPrj` project. Modify the `Customer` class to handle the association with multiple accounts, just as you did in the Exercise 2 of Lab 5. It must include the public methods: `addAccount(Account)`, `getAccount(int)`, and `getNumOfAccounts()`.

Complete the following steps:

1. Add two instance variables to the `Customer` class: `accounts` (an array of `Account` objects) and `numberOfAccounts` (an integer that keeps track of the next `accounts` array index). This replaces the single `account` reference variable, which should be removed.

```
private Account[] accounts;
private int numberOfAccounts;
```

2. Modify the constructor to initialize the `accounts` array.

```
public Customer(String f, String l) {
  firstName = f;
  lastName = l;
  // initialize accounts array
  accounts = new Account[10];
  numberOfAccounts = 0;
}
```

3. Add the `addAccount` method. This method takes a single parameter, an `Account` object, and stores it in the `accounts` array. It must also increment the `numberOfAccounts` instance variable. This method replaces the `setAccount` method, which should be removed.

```
public void addAccount(Account acct) {
  int i = numberOfAccounts++;
  accounts[i] = acct;
}
```

4. Add the `getNumOfAccounts` accessor method, which returns the `numberOfAccounts` instance variable.

```
public int getNumOfAccounts() {
  return numberOfAccounts;
}
```

5. Add the `getAccount` method. This method returns the account associated with the given `index` parameter. This method replaces the previous `getAccount` method, which should be removed.

```
public Account getAccount(int account_index) {
  return accounts[account_index];
}
```

Task 2 – Copying and Completing the `CustomerReport` Class

Complete the following steps:

1. Create the `com.mybank.report` source package in the `BankPrj` project.
2. Copy the `CustomerReport.java` file from the `resources/06_class1` directory into the `com.mybank.report` source package of the `BankPrj` project.
3. Complete the `CustomerReport.java` code. You will find comment blocks that start and end with `/*** ... ***/`. These comments indicate the location of the code that you must supply.
   a. Use the `instanceof` operator to test what type of account this is and set `account_type` to an appropriate value, such as `Savings Account` or `Checking Account`.

```
if ( account instanceof SavingsAccount ) {
  account_type = "Savings Account";
} else if ( account instanceof CheckingAccount ) {
  account_type = "Checking Account";
} else {
  account_type = "Unknown Account Type";
}
```

   b. Print out the type of account and the balance.

```
System.out.println("   " + account_type + ": current balance is "
                   + account.getBalance());
```

Task 3 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 5 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the output listed on page L6-20.

Exercise 3: Creating a Batch Program (Advanced)

In this exercise, you create a batch program to accumulate interest for each savings account on a monthly basis.

Note – This is an advanced exercise. This exercise is optional and should only be attempted if you completed all of the previous exercises for this module.

Figure 6-3 shows the dependencies between the class used by the `TestBatch` program.

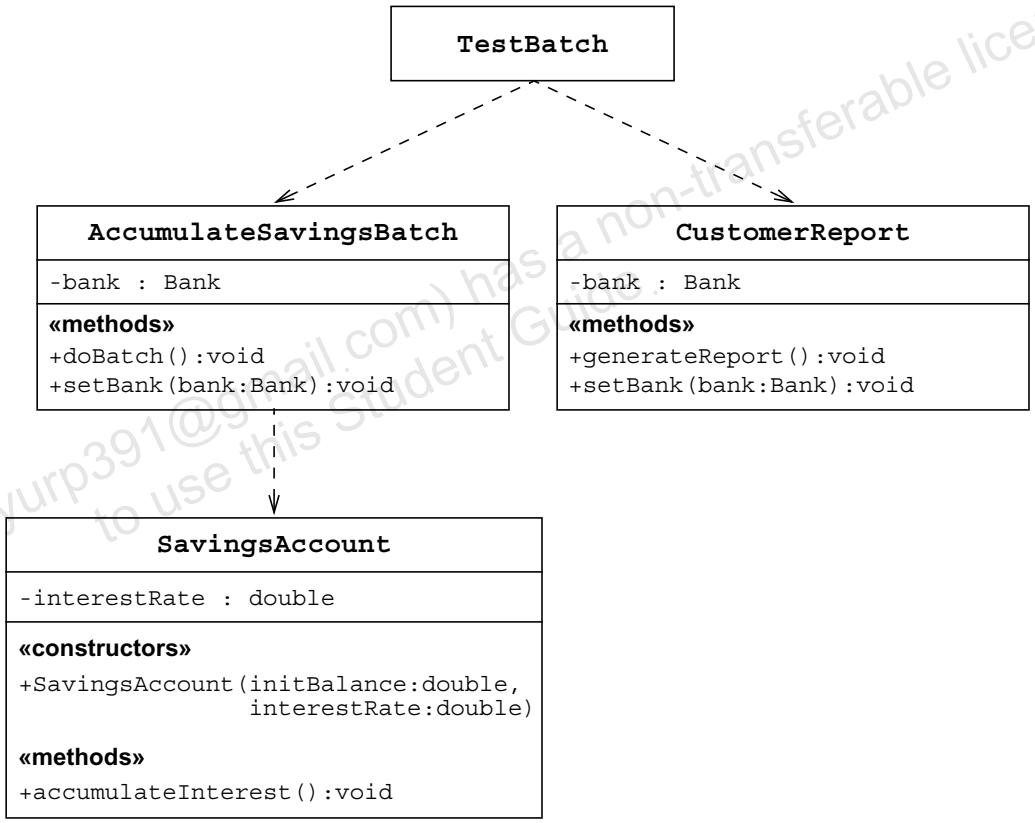

Figure 6-3 Class Dependencies for the `TestBatch` Program

This exercise contains the following sections:

- "Task 1 – Modifying the SavingsAccount Class"
- "Task 2 – Creating the AccumulateSavingsBatch Class"
- "Task 3 – Copying the TestBatch Class"
- "Task 4 – Compiling the TestBatch Class"
- "Task 5 – Running the TestBatch Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Task 1 – Modifying the `SavingsAccount` Class

A savings account gains interest. The bank permits customers to store money in a savings account and, on a monthly basis, the savings account will accumulate based on the following formula: balance = balance + (balance * (interestRate / 12)).

In this task, you add the `accumulateInterest` method to perform this operation.

Task 2 – Creating the `AccumulateSavingsBatch` Class

Complete the following steps:

1. Create the `AccumulateSavingsBatch` class with the following characteristics:

   Class Name: **`AccumulateSavingsBatch`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.batch`**

   The class must conform to the specification defined in Figure 6-3 on page L6-28.

2. Add an instance variable `bank` of the type `Bank`.

3. Add a `setBank` method to set the method parameter to the `bank` instance variable.

4. Add a `doBatch` method using the following hints:

```
for each Customer in the Bank do
  for each Account in the Customer do
    if the Account is a SavingsAccount,
      then call the accumulateInterest method
```

Task 3 – Copying the `TestBatch` Class

In this task, you copy the `TestBatch.java` file from the `resources/06_class1` directory into the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Compiling the `TestBatch` Class

In this task, you compile the `TestBatch` class.

Task 5 – Running the `TestBatch` Program

In this task, you run the `TestBatch` program. The output should be similar to the following:

```
                    CUSTOMERS REPORT
                    ================

Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Checking Account: current balance is 200.0
    Savings Account: current balance is 150.0
```

```
ACCUMULATE SAVINGS BATCH EXECUTED

                        CUSTOMERS REPORT
                        ================

Customer: Simms, Jane
    Savings Account: current balance is 501.25
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1509.375
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Checking Account: current balance is 200.0
    Savings Account: current balance is 150.625
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 7

Advanced Class Features

Objectives

Upon completion of this lab, you should be able to:

- Apply static class members to resolve a design decision.
- Create abstract classes and interfaces, and explore the polymorphic properties of these types of components

Exercise 1: Applying Static Members to a Design (Level 1)

In this exercise, you apply static class members to resolve a design decision. The Banking Project currently uses a concrete class to represent the concept of a bank, which contains the set of customers for the bank. The project team has decided that this is a risky design because it would be possible to instantiate multiple `Bank` objects each with the potential to contain different sets of customers.

The design team has decided to make the `Bank` class a utility class. A utility class is one that is not instantiated and all of its members are static. Figure 7-1 shows the new design for the `Bank` class. Your job is to program these changes to the `Bank` class and to all of the classes that use the `Bank` class.

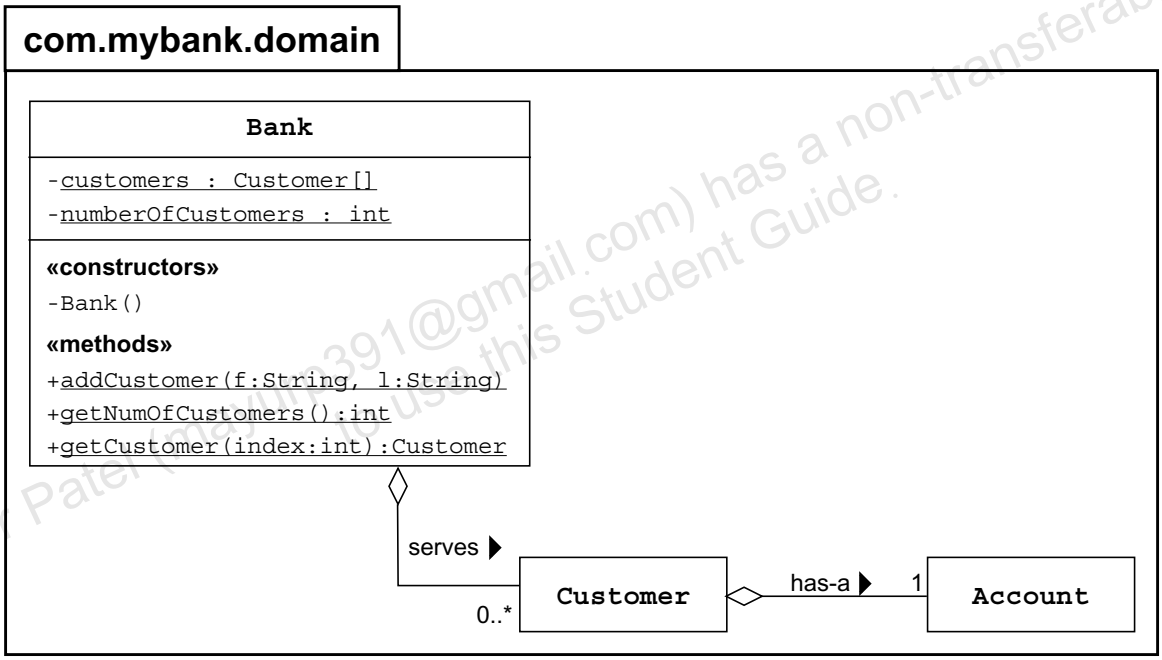

Figure 7-1 UML Diagram of the `Bank` Utility Class

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the CustomerReport Class"
- "Task 3 – Deleting the Current TestReport Class"
- "Task 4 – Copying the TestReport Class"
- "Task 5 – Compiling the TestReport Class"
- "Task 6 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise1` directory.

Task 1 – Modifying the `Bank` Class

In this task, you open the `BankPrj` project in the `projects` directory, and modify the `Bank` class in the `com.mybank.domain` source package of the `BankPrj` project. All the members (both instance variables and methods) should be changed to static, as shown in Figure 7-1 on page L7-2. Also, move the original variable initialization code from the constructor to either a static block or on the static variable declarations.

Task 2 – Modifying the `CustomerReport` Class

In this task, you modify the `CustomerReport` class source file in the `com.mybank.report` source package of the BankPrj project. The updated `CustomerReport` class uses the `Bank` class as a utility class.

Task 3 – Deleting the Current `TestReport` Class

In this task, you delete the current `TestReport` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/07_class2` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 6 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the output from previous tests as shown in Exercise 2, Task 5 of Lab 6 (Level 1).

Exercise 1: Applying Static Members to a Design (Level 2)

In this exercise, you apply static class members to resolve a design decision. The Banking Project currently uses a concrete class to represent the concept of a bank, which contains the set of customers for the bank. The project team has decided that this is a risky design because it would be possible to instantiate multiple `Bank` objects each with the potential to contain different sets of customers.

The design team has decided to make the `Bank` class a utility class. A utility class is one that is not instantiated and all of its members are static. Figure 7-1 on page L7-2 shows the new design for the `Bank` class. Your job is to program these changes to the `Bank` class and to all of the classes that use the `Bank` class.

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the CustomerReport Class"
- "Task 3 – Deleting the Current TestReport Class"
- "Task 4 – Copying the TestReport Class"
- "Task 5 – Compiling the TestReport Class"
- "Task 6 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise1` directory.

Task 1 – Modifying the `Bank` Class

In this task, you modify the `Bank` class in the `com.mybank.domain` source package of the `BankPrj` project. All the members (both instance variables and methods) should be changed to static, as shown in Figure 7-1 on page L7-2.

Complete the following steps:

1. Open the `BankPrj` project in the `projects` directory if you have closed it.
2. Open the `Bank` class in the `com.mybank.domain` source package of the `BankPrj` project.
3. Change all instance variables to static.
4. Move the original variable initialization code from the constructor to either a static block or on the static variable declarations.
5. Change the constructor to be private and remove the body of the constructor.
6. Change all methods to static.

Task 2 – Modifying the `CustomerReport` Class

In this task, you modify the `CustomerReport` class source file in the `com.mybank.report` source package of the BankPrj project. The updated `CustomerReport` class uses the `Bank` class as a utility class.

Complete the following steps:

1. Remove the `bank` instance variable and the `getBank` and `setBank` methods.
2. Modify the `generateReport` method to use the static methods from the new `Bank` utility class design.

Task 3 – Deleting the Current `TestReport` Class

In this task, you delete the current `TestReport` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/07_class2` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 6 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the output from previous tests as shown in Exercise 2, Task 5 of Lab 6 (Level 1).

Exercise 1: Applying Static Members to a Design (Level 3)

In this exercise, you apply static class members to resolve a design decision. The Banking Project currently uses a concrete class to represent the concept of a bank, which contains the set of customers for the bank. The project team has decided that this is a risky design because it would be possible to instantiate multiple `Bank` objects each with the potential to contain different sets of customers.

The design team has decided to make the `Bank` class a utility class. A utility class is one that is not instantiated and all of its members are static. Figure 7-1 on page L7-2 shows the new design for the `Bank` class. Your job is to program these changes to the `Bank` class and to all of the classes that use the `Bank` class.

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the CustomerReport Class"
- "Task 3 – Deleting the Current TestReport Class"
- "Task 4 – Copying the TestReport Class"
- "Task 5 – Compiling the TestReport Class"
- "Task 6 – Running the TestReport Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise1` directory.

Task 1 – Modifying the `Bank` Class

In this task, you modify the `Bank` class in the `com.mybank.domain` source package of the `BankPrj` project. All the members (both instance variables and methods) should be changed to static, as shown in Figure 7-1 on page L7-2.

Complete the following steps:

1. Open the `BankPrj` project in the `projects` directory if you have closed it.
2. Open the `Bank` class in the `com.mybank.domain` source package of the `BankPrj` project.
3. Change all instance variables to static.

```
private static Customer[] customers;
private static int        numberOfCustomers;
```

4. Move the original variable initialization code from the constructor to either a static block or on the static variable declarations.

```
static {
  customers = new Customer[10];
  numberOfCustomers = 0;
}
```

5. Change the constructor to be private and remove the body of the constructor.

```
private Bank() {
  // this constructor should never be called
}
```

6. Change all methods to static.

```
public static void addCustomer(String f, String l) {
  int i = numberOfCustomers++;
  customers[i] = new Customer(f, l);
}
public static int getNumOfCustomers() {
  return numberOfCustomers;
}
public static Customer getCustomer(int customer_index) {
  return customers[customer_index];
}
```

Task 2 – Modifying the `CustomerReport` Class

In this task, you modify the `CustomerReport` class source file in the `com.mybank.report` source package of the BankPrj project. The updated `CustomerReport` class uses the `Bank` class as a utility class.

Complete the following steps:

1. Remove the `bank` instance variable and the `getBank` and `setBank` methods.
2. Modify the `generateReport` method to use the static methods from the new `Bank` utility class design.

```
public void generateReport() {

  // Print report header
  System.out.println("CUSTOMERS REPORT");
  System.out.println("================");

  // For each customer...
  for ( int cust_idx = 0;
        cust_idx < Bank.getNumOfCustomers();
        cust_idx++ ) {
    Customer customer = Bank.getCustomer(cust_idx);
    // and so on...
  }
}
```

Task 3 – Deleting the Current `TestReport` Class

In this task, you delete the current `TestReport` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 4 – Copying the `TestReport` Class

In this task, you copy the `TestReport.java` file from the `resources/07_class2` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 6 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the output from previous tests as shown in Exercise 2, Task 5 of Lab 6 (Level 1).

Exercise 2: Working With Interfaces and Abstract Classes (Level 1)

In this exercise, you create abstract classes and interfaces, and explore the polymorphic properties of these types of components. You create a hierarchy of animals that is rooted in an abstract class `Animal`. Several of the animal classes implement an interface called `Pet`. Figure 7-2 shows a UML class diagram of the animal classes that you create.

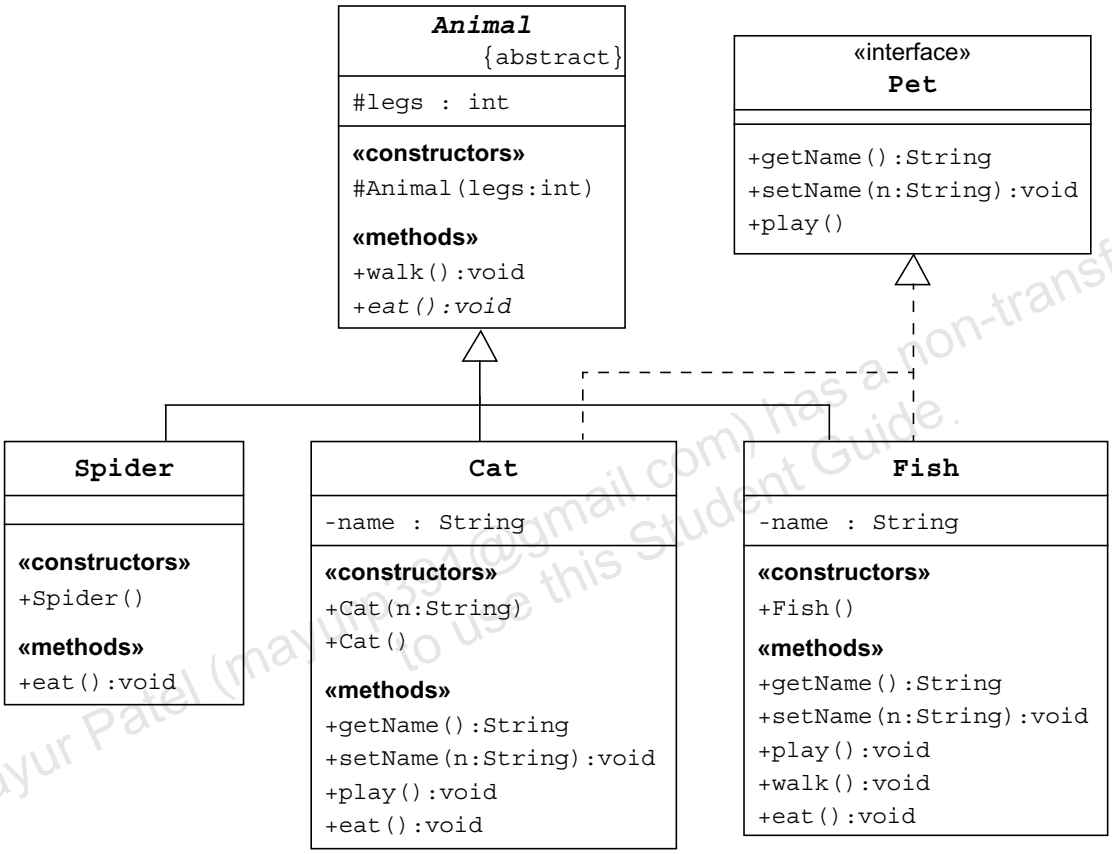

Figure 7-2 Animal and Pet Hierarchy

This exercise contains the following sections:

- "Task 1 – Creating the Pet Interface"
- "Task 2 – Creating the Animal Classes"
- "Task 3 – Creating the TestAnimals Class"
- "Task 4 – Compiling the TestAnimals Class"
- "Task 5 – Running the TestAnimals Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise2` directory.

Task 1 – Creating the `Pet` Interface

Complete the following steps:

1. Open the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.
2. Create the `Pet` interface with the following characteristics:

   Class Name: **`Pet`**

   Project: **`InterfaceProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

The `Pet` interface must satisfy the UML diagram in Figure 7-1 on page L7-2.

Task 2 – Creating the Animal Classes

Complete the following steps:

1. Create the `Animal` class with the following characteristics:

   Class Name: **`Animal`**

   Project: **`InterfaceProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

The `Animal` class must satisfy the UML diagram in Figure 7-1 on page L7-2. The `action` methods (`walk` and `eat`) print a statement to standard output that reflects the animal. For example, the `walk` method for the `Animal` class might say something similar to `This animal walks on 4 legs`, where `4` is the value of the `legs` instance variable.

2. Create the `Spider`, `Cat`, and `Fish` classes in the Source Packages of the `InterfaceProject` project to satisfy the UML diagram in Figure 7-1 on page L7-2.

Task 3 – Creating the `TestAnimals` Class

Complete the following steps:

1. In this task, you create the `TestAnimals` class with the following characteristics:

   Class Name: **`TestAnimals`**

   Project: **`InterfaceProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

2. Add the `main` method to create and manipulate instances of the classes you created previously.

   Start with:

```
Fish d = new Fish();
Cat c = new Cat("Fluffy");
Animal a = new Fish();
Animal e = new Spider();
Pet p = new Cat();
```

Experiment by:

- Calling the methods in each object
- Casting objects
- Using polymorphism
- Using `super` to call superclass methods

Task 4 – Compiling the `TestAnimals` Class

In this task, you compile the `TestAnimals` class.

Task 5 – Running the `TestAnimals` Program

In this task, you run the `TestAnimals` program.

Exercise 2: Working With Interfaces and Abstract Classes (Level 2)

In this exercise, you create abstract classes and interfaces, and explore the polymorphic properties of these types of components. You create a hierarchy of animals that is rooted in an abstract class `Animal`. Several of the animal classes will implement an interface called `Pet`. Use Figure 7-2 on page L7-12 as a reference. You experiment with variations of these animals, their methods, and polymorphism.

This exercise contains the following sections:

- "Task 1 – Creating the Pet Interface"
- "Task 2 – Creating the Animal Classes"
- "Task 3 – Creating the TestAnimals Class"
- "Task 4 – Compiling the TestAnimals Class"
- "Task 5 – Running the TestAnimals Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise2` directory.

Task 1 – Creating the `Pet` Interface

Complete the following steps:

1. Open the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.
2. Create the `Pet` interface with the following characteristics:

   Class Name: **`Pet`**

   Project: **`InterfaceProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

The `Pet` interface must satisfy the UML diagram in Figure 7-1 on page L7-2.

Task 2 – Creating the Animal Classes

Complete the following steps:

1. Create the abstract `Animal` class.
   a. Create the `Animal` class with the following characteristics:

      Class Name: **`Animal`**

      Project: **`InterfaceProject`**

      Location: **`Source Packages`**

      Package: **`default package`**

   b. Declare a protected integer instance variable called `legs`, which records the number of legs for this animal.

   c. Define a protected constructor that initializes the `legs` instance variable.

   d. Declare an abstract method `eat`.

   e. Declare a concrete method `walk` that prints out something about how the animals walks (include the number of legs).

2. Create the `Spider` class.
   a. Create the `Spider` class with the following characteristics:

      Class Name: **`Spider`**

      Project: **`InterfaceProject`**

      Extends: **`Animal`**

      Location: **`Source Packages`**

      Package: **`default package`**

   b. Define a `no-arg` constructor that calls the superclass constructor to specify that all spiders have eight legs.
   c. Implement the `eat` method.
3. Create the `Cat` class.
   a. Create the `Cat` class with the following characteristics:

      Class Name: **`Cat`**

      Project: **`InterfaceProject`**

      Extends: **`Animal`**

      Implements: **`Pet`**

      Location: **`Source Packages`**

      Package: **`default package`**

   b. This class must include a `String` instance variable to store the name of the pet.
   c. Define a constructor that takes one `String` parameter that specifies the cat's name. This constructor must also call the superclass constructor to specify that all cats have four legs.
   d. Define another constructor that takes no parameters. Have this constructor call the previous constructor (using the `this` keyword) and pass an empty string as the argument.
   e. Implement the `Pet` interface methods.
   f. Implement the `eat` method.
4. Create the `Fish` class.
   a. Create the `Fish` class with the following characteristics:

      Class Name: **`Fish`**

      Project: **`InterfaceProject`**

      Extends: **`Animal`**

Implements: **Pet**

Location: **Source Packages**

Package: **default package**

b. This class must include a `String` instance variable to store the name of the pet.

c. Define a `no-arg` constructor that calls the superclass constructor to specify that fish do not have legs.

d. Implement the `Pet` interface methods.

e. Override the `walk` method. This method should call the super method and they print a message that fish do not walk

f. Implement the `eat` method.

Task 3 – Creating the `TestAnimals` Class

Complete the following steps:

1. In this task, you create the `TestAnimals` class with the following characteristics:

   Class Name: **TestAnimals**

   Project: **InterfaceProject**

   Location: **Source Packages**

   Package: **default package**

2. .Add the `main` method to create and manipulate instances of the classes you created previously.

   Start with:

```
Fish d = new Fish();
Cat c = new Cat("Fluffy");
Animal a = new Fish();
Animal e = new Spider();
Pet p = new Cat();
```

   Experiment by:

   - Calling the methods in each object
   - Casting objects
   - Using polymorphism
   - Using `super` to call superclass methods

Task 4 – Compiling the `TestAnimals` Class

In this task, you compile the `TestAnimals` class.

Task 5 – Running the `TestAnimals` Program

In this task, you run the `TestAnimals` program.

Exercise 2: Working With Interfaces and Abstract Classes (Level 3)

In this exercise, you create a hierarchy of animals that is rooted in an abstract class `Animal`. Several of the animal classes implement an interface called `Pet`. Use Figure 7-2 on page L7-12 as a reference. You experiment with variations of these animals, their methods, and polymorphism.

This exercise contains the following sections:

- "Task 1 – Creating the Pet Interface"
- "Task 2 – Creating the Animal Classes"
- "Task 3 – Creating the TestAnimals Class"
- "Task 4 – Compiling the TestAnimals Class"
- "Task 5 – Running the TestAnimals Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/07_class2/exercise2` directory.

Task 1 – Creating the `Pet` Interface

Complete the following steps:

1. Open the `InterfaceProject` project in the `exercises/07_class2/exercise2` directory.
2. Create the `Pet` interface with the following characteristics:

   Class Name: **`Pet`**

   Project: **`InterfaceProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

The `Pet` interface must satisfy the UML diagram in Figure 7-1 on page L7-2.

```
public interface Pet {
  public String getName();
  public void setName(String n);
  public void play();
}
```

Task 2 – Creating the Animal Classes

Complete the following steps:

1. Create the `Animal` class.

   a. Create the `Animal` class with the following characteristics:

      Class Name: **`Animal`**

      Project: **`InterfaceProject`**

      Location: **`Source Packages`**

      Package: **`default package`**

```
public abstract class Animal {
  // more code here
}
```

   b. Declare a protected integer instance variable called `legs`, which records the number of legs for this animal.

```
  protected int legs;
```

   c. Define a protected constructor that initializes the `legs` instance variable.

```
  protected Animal(int legs) {
    this.legs = legs;
  }
```

d. Declare an abstract method `eat`.

```
  public abstract void eat();
```

e. Declare a concrete method `walk` that prints out something about how the animals walks (include the number of legs).

```
  public void walk() {
    System.out.println("This animal walks on " + legs + " legs.");
  }
```

2. Create the `Spider` class.

a. Create the `Spider` class with the following characteristics:

Class Name: **`Spider`**

Project: **`InterfaceProject`**

Extends: **`Animal`**

Location: **`Source Packages`**

Package: **`default package`**

b. The `Spider` class extends the `Animal` class.

```
public class Spider extends Animal {
  // more code here
}
```

c. Define a `no-arg` constructor that calls the superclass constructor to specify that all spiders have eight legs.

```
  public Spider() {
    super(8);
  }
```

d. Implement the `eat` method.

```
  public void eat() {
    System.out.println("The spider eats a fly.");
  }
```

3. Create the `Cat` class.

a. Create the `Cat` class with the following characteristics:

Class Name: **`Cat`**

Project: **`InterfaceProject`**

Extends: **`Animal`**

Implements: **`Pet`**

Location: **Source Packages**

Package: **default package**

b. This class must include a `String` instance variable to store the name of the pet.

```
private String name;
```

c. Define a constructor that takes one `String` parameter that specifies the cat's name. This constructor must also call the superclass constructor to specify that all cats have four legs.

```
public Cat(String n) {
  super(4);
  name = n;
}
```

d. Define another constructor that takes no parameters. Have this constructor call the previous constructor (using the `this` keyword) and pass an empty string as the argument.

```
public Cat() {
  this("");
}
```

e. Implement the `Pet` interface methods.

```
public String getName() {
  return name;
}
public void setName(String n) {
  name = n;
}
public void play() {
  System.out.println(name + " likes to play with string.");
}
```

f. Implement the `eat` method.

```
public void eat() {
  System.out.println("Cats like to eat spiders and mice.");
}
```

4. Create the `Fish` class.

a. Create the `Fish` class with the following characteristics:

Class Name: **Fish**

Project: **InterfaceProject**

Extends: **Animal**

Implements: **Pet**

Location: **`Source Packages`**

Package: **`default package`**

b. This class must include a `String` instance variable to store the name of the pet.

```
private String name;
```

c. Define a `no-arg` constructor that calls the superclass constructor to specify that fish do not have legs.

```
public Fish() {
  super(0);   // this line must be here
}
```

d. Implement the `Pet` interface methods.

```
public void setName(String name) {
  this.name = name;
}
public String getName() {
  return name;
}
public void play() {
  System.out.println("Fish swim in their tanks all day.");
}
```

e. Override the `walk` method. This method should call the super method and they print a message that fish do not walk

```
public void walk() {
  super.walk();
  System.out.println("Fish, of course, can't walk; they swim.");
}
```

f. Implement the `eat` method.

```
public void eat() {
  System.out.println("Fish eat pond scum.");
}
```

Task 3 – Creating the `TestAnimals` Class

Complete the following steps:

1. In this task, you create the `TestAnimals` class with the following characteristics:

   Class Name: **`TestAnimals`**

   Project: **`InterfaceProject`**

Location: **`Source Packages`**

Package: **`default package`**

2. Add the `main` method to create and manipulate instances of the classes you created previously.

   Here is a sample test program:

```
public class TestAnimals {
  public static void main(String[] args) {
    Fish f = new Fish();
    Cat c = new Cat("Fluffy");
    Animal a = new Fish();
    Animal e = new Spider();
    Pet p = new Cat();

    // Demonstrate different implementations of an interface
    f.play();
    c.play();

    // Demonstract virtual method invocation
    e.eat();
    e.walk();

    // Demonstrate calling super methods
    a.walk();
  }
}
```

Task 4 – Compiling the `TestAnimals` Class

In this task, you compile the `TestAnimals` class.

Task 5 – Running the `TestAnimals` Program

In this task, you run the `TestAnimals` program.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 8

Exceptions and Assertions

Objectives

Upon completion of this lab, you should be able to create an application exception and apply the declare-or-handle rule to the account classes.

Exercise: Creating Your Own Exception (Level 1)

In this exercise, you create an `OverdraftException` that is thrown by the `withdraw` method in the `Account` class. In the previous design, the `deposit` and `withdraw` methods return a Boolean flag to indicate whether the operation was successful or not. This design has several flaws, one of which is that a `false` return value does not give the calling client any indication of why the operation was not successful. In the new design, you will use exceptions to indicate operation failure.

Figure 8-1 shows UML class diagram of this new design. In UML parlance, the «send» dependency indicates that the method on the source operation (the `withdraw` method) may throw an `OverdraftException`.

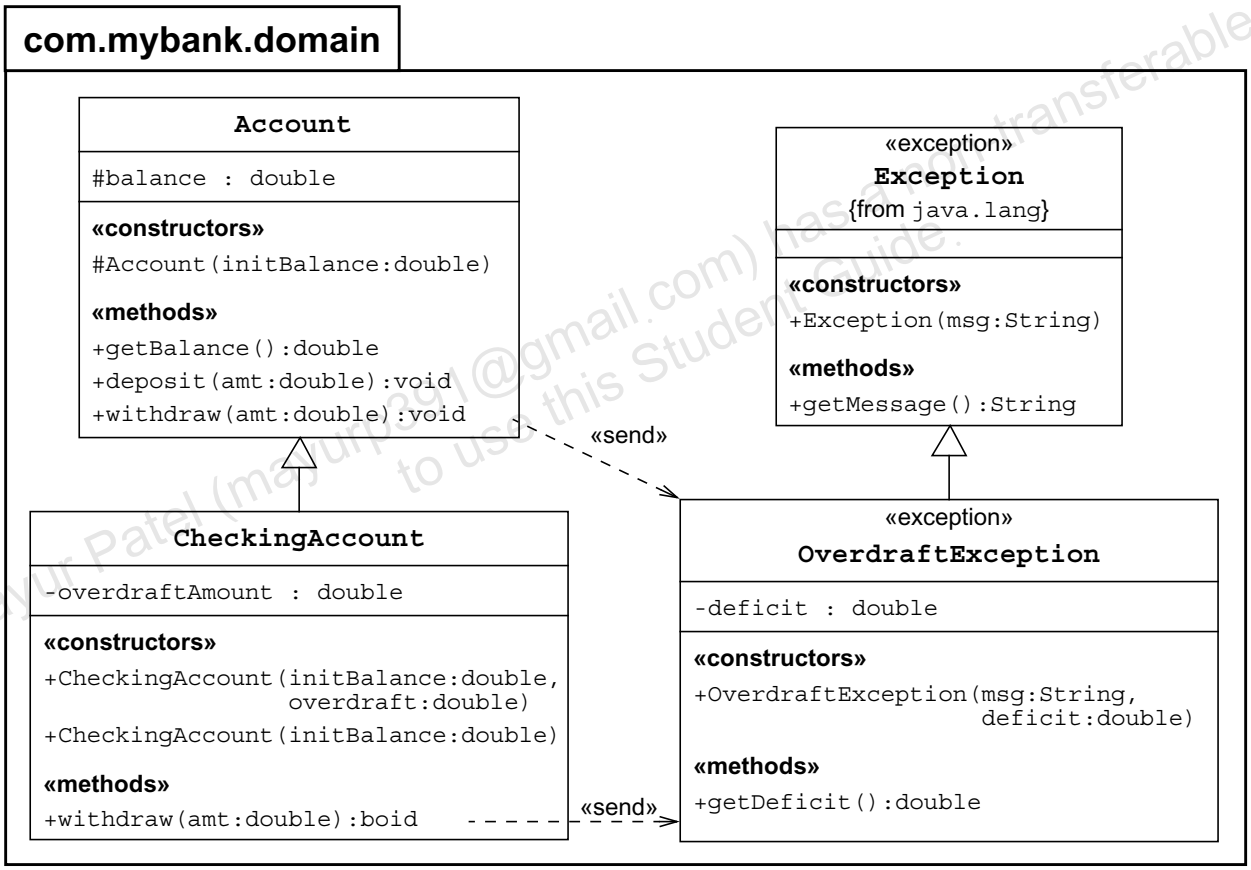

Figure 8-1 The `withdraw` Method Throws the `OverdraftException`

This exercise contains the following sections:

- "Task 1 – Creating the OverdraftException Class"
- "Task 2 – Modifying the Account Class"

- "Task 3 – Modifying the CheckingAccount Class"
- "Task 4 – Deleting the Current TestBanking Class"
- "Task 5 – Copying the TestBanking Class"
- "Task 6 – Compiling the TestBanking Class"
- "Task 7 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/08_except/exercise1` directory.

Task 1 – Creating the `OverdraftException` Class

In this task, you create the `OverdraftException` class in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

Class Name: **`OverdraftException`**

Project: **`BankPrj`**

Extends: **`Exception`**

Location: **`Source Packages`**

Package: **`com.mybank.domain`**

This class must satisfy the UML diagram in Figure 8-1 on page L8-2.

Task 2 – Modifying the `Account` Class

In this task, you modify the `Account` class source file in the `com.mybank.domain` source package of the `BankPrj` project. This class must satisfy the UML diagram in Figure 8-1 on page L8-2. In particular, the `deposit` and `withdraw` methods should not return a `boolean` flag.

Task 3 – Modifying the `CheckingAccount` Class

In this task, you modify the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project. This class must satisfy the UML diagram in Figure 8-1 on page L8-2.

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/08_except` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program. The output should be similar to the following:

```
Customer [Simms, Jane] has a checking balance of 200.0 with a 500.00
overdraft protection.
Checking Acct [Jane Simms] : withdraw 150.00
Checking Acct [Jane Simms] : deposit 22.50
Checking Acct [Jane Simms] : withdraw 147.62
Checking Acct [Jane Simms] : withdraw 470.00
Exception: Insufficient funds for overdraft protection   Deficit: 470.0
Customer [Simms, Jane] has a checking balance of 0.0

Customer [Bryant, Owen] has a savings balance of 200.0
Savings Acct [Owen Bryant] : withdraw 100.00
Savings Acct [Owen Bryant] : deposit 25.00
Savings Acct [Owen Bryant] : withdraw 175.00
Exception: Insufficient funds   Deficit: 50.0
Customer [Bryant, Owen] has a savings balance of 125.0
```

Exercise: Creating Your Own Exception (Level 2)

In this exercise, you create an `OverdraftException` that is thrown by the `withdraw` method in the `Account` class.

This exercise contains the following sections:

- "Task 1 – Creating the OverdraftException Class"
- "Task 2 – Modifying the Account Class"
- "Task 3 – Modifying the CheckingAccount Class"
- "Task 4 – Deleting the Current TestBanking Class"
- "Task 5 – Copying the TestBanking Class"
- "Task 6 – Compiling the TestBanking Class"
- "Task 7 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/08_except/exercise1` directory.

Task 1 – Creating the `OverdraftException` Class

Complete the following steps:

1. Create the `OverdraftException` class in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`OverdraftException`**

   Project: **`BankPrj`**

   Extends: **`Exception`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

2. Add a private instance variable called `deficit` that holds a `double`.
3. Add a public constructor that takes two arguments: message and deficit. The message parameter should be passed to the superclass constructor. The deficit parameter initializes the `deficit` instance variable.
4. Add a public accessor called `getDeficit`.

Task 2 – Modifying the `Account` Class

In this task, you modify the `Account` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Modify the `deposit` method so that it does not return a value (that is, `void`). This operation should never fail, so it does not need to throw any exceptions.
2. Modify the `withdraw` method so that it does not return a value (that is, `void`). Declare that this method throws the `OverdraftException`. Modify the code to throw a new exception that specifies `Insufficient funds` and the deficit (the amount requested subtracted by the current balance).

Task 3 – Modifying the `CheckingAccount` Class

In this task, you modify the `CheckingAccount` class source file in the `com.mybank.domain` source package of the `BankPrj` project. Modify the `withdraw` method so that it does not return a value (that is, `void`). Declare that this method throws the `OverdraftException`. Modify the code to throw an exception when the `overdraftProtection` amount is not sufficient to cover the deficit; use the message `Insufficient funds for overdraft protection` for this exception.

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/08_except` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program.

The output should be similar to the output listed on page L8-5.

Exercise: Creating Your Own Exception (Level 3)

In this exercise, you create an `OverdraftException` that is thrown by the `withdraw` method in the `Account` class.

This exercise contains the following sections:

- "Task 1 – Creating the OverdraftException Class"
- "Task 2 – Modifying the Account Class"
- "Task 3 – Modifying the CheckingAccount Class"
- "Task 4 – Deleting the Current TestBanking Class"
- "Task 5 – Copying the TestBanking Class"
- "Task 6 – Compiling the TestBanking Class"
- "Task 7 – Running the TestBanking Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/08_except/exercise1` directory.

Task 1 – Creating the `OverdraftException` Class

Complete the following steps:

1. Create the `OverdraftException` class in the `com.mybank.domain` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`OverdraftException`**

   Project: **`BankPrj`**

   Extends: **`Exception`**

   Location: **`Source Packages`**

   Package: **`com.mybank.domain`**

```
package com.mybank.domain;
public class OverdraftException extends Exception {
  // insert code here
}
```

2. Add a private instance variable called `deficit` that holds a `double`.

```
  private final double deficit;
```

3. Add a public constructor that takes two arguments: message and deficit. The message parameter should be passed to the superclass constructor. The deficit parameter initializes the `deficit` instance variable.

```
  public OverdraftException(String msg, double deficit) {
    super(msg);
    this.deficit = deficit;
  }
```

4. Add a public accessor called `getDeficit`.

```
  public double getDeficit() {
    return deficit;
  }
```

Task 2 – Modifying the `Account` Class

In this task, you modify the `Account` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Modify the `deposit` method so that it does not return a value (that is, `void`). This operation should never fail, so it does not need to throw any exceptions.

```
public void deposit(double amt) {
  balance = balance + amt;
}
```

2. Modify the `withdraw` method so that it does not return a value (that is, `void`). Declare that this method throws the `OverdraftException`. Modify the code to throw a new exception that specifies `Insufficient funds` and the deficit (the amount requested subtracted by the current balance).

```
public void withdraw(double amt) throws OverdraftException {
  if ( amt <= balance ) {
    balance = balance - amt;
  } else {
    throw new OverdraftException("Insufficient funds", amt - balance);
  }
}
```

Task 3 – Modifying the `CheckingAccount` Class

Using a text editor, modify the `CheckingAccount` class source file in the `src/com/mybank/domain/` directory. This class must satisfy the UML diagram in Figure 8-1 on page L8-2.

Modify the `withdraw` method so that it does not return a value (that is, `void`). Declare that this method throws the `OverdraftException`. Modify the code to throw an exception if necessary. There are two cases that need to be handled. First, there is a deficit with no overdraft protection; use the message `No overdraft protection` for this exception. Second, the `overdraftProtection` amount is not sufficient to cover the deficit; use the message `Insufficient funds for overdraft protection` for this exception.

```
  public void withdraw(double amount) throws OverdraftException {
    if ( balance < amount ) {
      double overdraftNeeded = amount - balance;
      if ( overdraftAmount < overdraftNeeded ) {
        throw new OverdraftException("Insufficient funds for overdraft
protection",
                                     overdraftNeeded);

      } else {
        balance = 0.0;
        overdraftAmount -= overdraftNeeded;
      }
    } else {
      balance = balance - amount;
    }
  }
```

Task 4 – Deleting the Current `TestBanking` Class

In this task, you delete the current `TestBanking` class in the `com.mybank.test` source package of the `BankPrj` project.

Task 5 – Copying the `TestBanking` Class

In this task, you copy the `TestBanking.java` file from the `resources/08_except` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestBanking` Class

In this task, you compile the `TestBanking` class.

Task 7 – Running the `TestBanking` Program

In this task, you run the `TestBanking` program.

The output should be similar to the output listed on page L8-5.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 9

Collections and Generics Framework

Objectives

Upon completion of this lab, you should be able to:

- Use a generic collection to manage a one-to-many association

Exercise 1: Using Collections to Represent Association (Level 1)

In this exercise, you use generic collections to represent class associations in the Bank project domain model.

In your previous design, arrays were used to implement multiplicity in the relationships between the bank and its customers, and between customers and their accounts. This design has several significant limitations; the most significant is that the array, after it is created, has a fixed size. The Collections API was created to solve this and other limitations.

Figure 9-1 shows the domain model of the Bank project with the class associations: a bank serves many customers, and a customer has many accounts. Figure 9-1 also shows the detailed design for `Bank` and `Customer` classes that use a generic `List` to maintain these links.

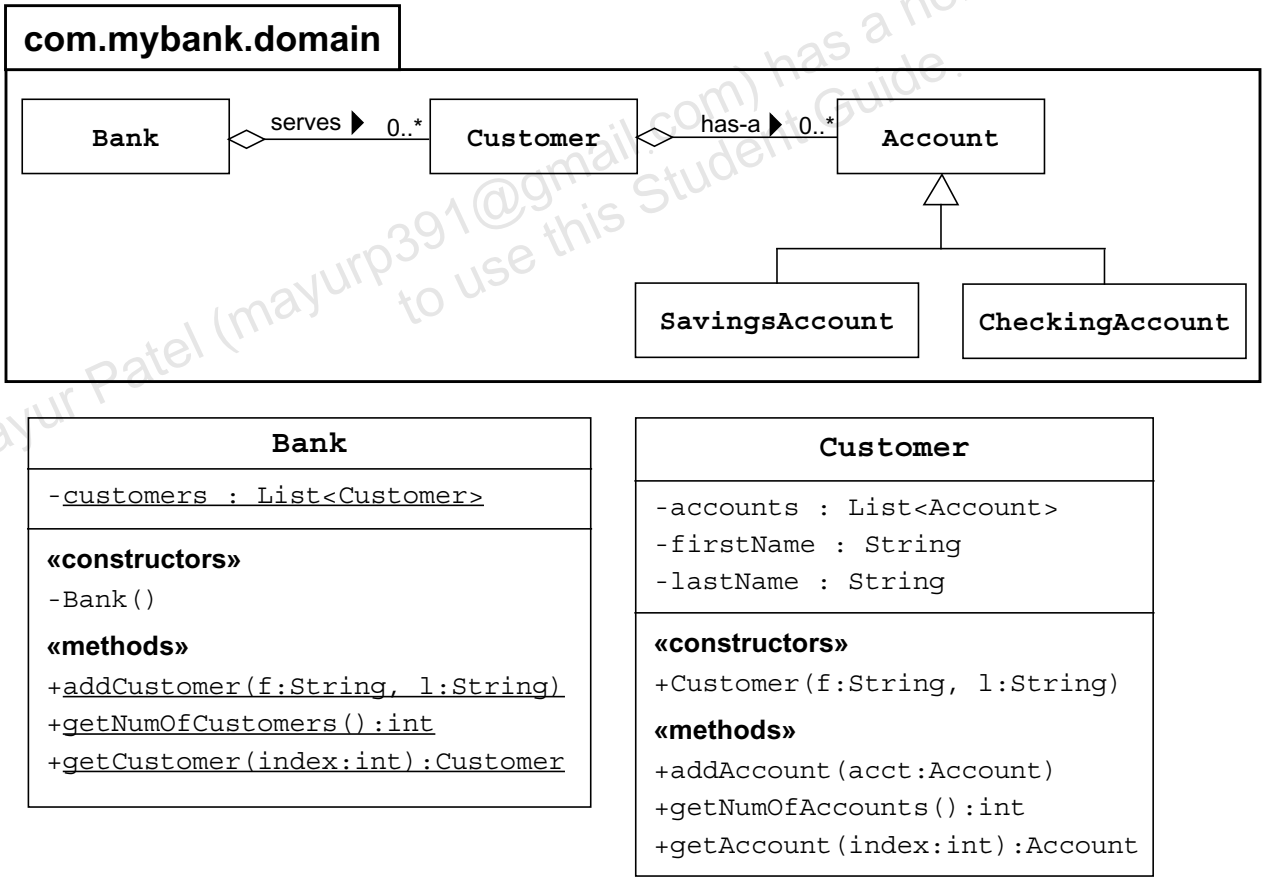

Figure 9-1 Domain Model of the Bank Project with Details on the `Bank` and `Customer` Classes

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the Customer Class"
- "Task 3 – Compiling the TestReport Class"
- "Task 4 – Running the TestReport Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/09_collections/exercise` directory.

Task 1 – Modifying the `Bank` Class

In this task, you modify the `Bank` class source file in the `com.mybank.domain` source package of the `BankPrj` project. This class must satisfy the UML diagram in Figure 9-1 on page L9-2.

Task 2 – Modifying the `Customer` Class

In this task, you modify the `Customer` class source file in the `com.mybank.domain` source package of the `BankPrj` project. This class must satisfy the UML diagram in Figure 9-1 on page L9-2.

Task 3 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 4 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the following:

```
CUSTOMERS REPORT
================

Customer: Simms, Jane
Savings Account: current balance is 500.0
Checking Account: current balance is 200.0

Customer: Bryant, Owen
Checking Account: current balance is 200.0

Customer: Soley, Tim
Savings Account: current balance is 1500.0
Checking Account: current balance is 200.0

Customer: Soley, Maria
Savings Account: current balance is 150.0
```

Exercise 1: Using Collections to Represent Association (Level 2)

In this exercise, you use generic collections to represent class associations in the Bank project domain model.

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the Customer Class"
- "Task 3 – Compiling the TestReport Class"
- "Task 4 – Running the TestReport Program"

Preparation

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/09_collections/exercise` directory.

Task 1 – Modifying the `Bank` Class

In this task, you modify the `Bank` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Modify the declaration for the `customers` instance variable to be of type `List<Customer>`, and drop the `numberOfCustomers` instance variable.

2. Modify the static block to initialize the `customers` instance variable to be a new `ArrayList` object.
3. Modify the `addCustomer` method to use the `add` method.
4. Modify the `getCustomer` method to use the `get` method.
5. Modify the `getNumOfCustomers` method to use the `size` method.

Task 2 – Modifying the `Customer` Class

In this task, you modify the `Customer` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Modify the declaration for the `accounts` instance variable to be of type `List<Account>`, and drop the `numberOfAccounts` instance variable.
2. Modify the constructor to initialize the `accounts` instance variable to be a new `ArrayList` object.
3. Modify the `addAccount` method to use the `add` method.
4. Modify the `getAccount` method to use the `get` method.
5. Modify the `getNumOfAccounts` method to use the `size` method.

Task 3 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 4 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the following:

```
CUSTOMERS REPORT
================

Customer: Simms, Jane
Savings Account: current balance is 500.0
Checking Account: current balance is 200.0

Customer: Bryant, Owen
Checking Account: current balance is 200.0
```

```
Customer: Soley, Tim
Savings Account: current balance is 1500.0
Checking Account: current balance is 200.0

Customer: Soley, Maria
Savings Account: current balance is 150.0
```

Exercise 1: Using Collections to Represent Association (Level 3)

In this exercise, you use generic collections to represent class associations in the Bank project domain model.

This exercise contains the following sections:

- "Task 1 – Modifying the Bank Class"
- "Task 2 – Modifying the Customer Class"
- "Task 3 – Compiling the TestReport Class"
- "Task 4 – Running the TestReport Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/09_collections/exercise` directory.

Task 1 – Modifying the `Bank` Class

In this task, you modify the `Bank` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Add the two import statements to include the collections classes you will use in the Bank class.

```
import java.util.List;
import java.util.ArrayList;
```

2. Modify the declaration for the `customers` instance variable to be of type `List<Customer>`, and drop the `numberOfCustomers` instance variable.

```
  private static List<Customer> customers;
```

3. Modify the static block to initialize the `customers` instance variable to be a new `ArrayList` object.

```
  static {
    customers = new ArrayList<Customer>(10);
  }
```

4. Modify the `addCustomer` method to use the `add` method.

```
  public static void addCustomer(String f, String l) {
    customers.add(new Customer(f, l));
  }
```

5. Modify the `getCustomer` method to use the `get` method.

```
  public static Customer getCustomer(int customer_index) {
    return customers.get(customer_index);
  }
```

6. Modify the `getNumOfCustomers` method to use the `size` method.

```
  public static int getNumOfCustomers() {
    return customers.size();
  }
```

Task 2 – Modifying the `Customer` Class

In this task, you modify the `Customer` class source file in the `com.mybank.domain` source package of the `BankPrj` project.

Complete the following steps:

1. Add the two import statements to include the collections classes you will use in the Bank class.

```
import java.util.List;
import java.util.ArrayList;
```

2. Modify the declaration for the `accounts` instance variable to be of type `List<Account>`, and drop the `numberOfAccounts` instance variable.

```
private List<Account> accounts;
```

3. Modify the constructor to initialize the `accounts` instance variable to be a new `ArrayList` object.

```
public Customer(String f, String l) {
  firstName = f;
  lastName = l;
  // initialize accounts instance variable
  accounts = new ArrayList<Account>(10);
}
```

4. Modify the `addAccount` method to use the `add` method.

```
public void addAccount(Account acct) {
  accounts.add(acct);
}
```

5. Modify the `getAccount` method to use the `get` method.

```
public Account getAccount(int account_index) {
  return accounts.get(account_index);
}
```

6. Modify the `getNumOfAccounts` method to use the `size` method.

```
public int getNumOfAccounts() {
  return accounts.size();
}
```

Task 3 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class.

Task 4 – Running the `TestReport` Program

In this task, you run the `TestReport` program. The output should be similar to the following:

```
CUSTOMERS REPORT
================

Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Checking Account: current balance is 200.0
    Savings Account: current balance is 150.0
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

_____

_____

_____

- Interpretations

_____

_____

_____

- Conclusions

_____

_____

_____

- Applications

_____

_____

_____

Lab 10

I/O Fundamentals

There are no exercises for this module.

Lab 11

Console I/O and File I/O

Objectives

Upon completion of this lab, you should be able to:

- Read a data file using the `Scanner` API
- Use a generic collection to manage a one-to-many association

Exercise 1: Reading a Data File (Level 1)

In this exercise, you create a class that reads customer and account data from a flat file.

Figure 11-1 shows the UML diagram for the `DataSource` class that you create for the `TestReport` program.

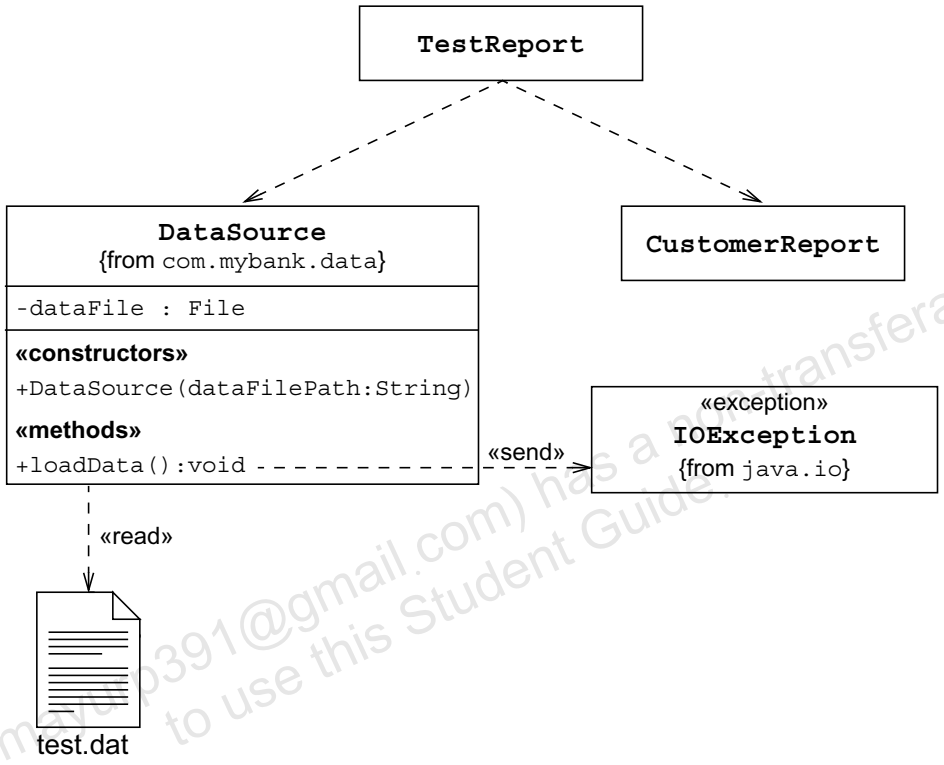

Figure 11-1 The `DataSource` Class Loads Customer Data From a Flat File

Code 11-1 shows an example of the format of the customer data file. The first line contains an integer, which determines the number of customers in the data file. A customer record contains the first name, last name, and the number of accounts, separated by tab characters. Each account record contains a single-character code that determines the type of account and also the data in that record.

Code 11-1 Data File Format

```
<number-of-customers>

<first-name>  <last-name>  <number-of-accounts>
<account-type-code>  <datum1>  <datum2>
```

Code 11-2 shows an example of the format of the customer data file. This data file contains four customer records. The first is for Jane Simms; Jane has two bank accounts. The first account is a savings account, with an initial balance of 500.00 and an interest rate of 5 percent (0.05). The second account is a checking account with an initial balance of 200.00 and overdraft protection of 400.00.

Code 11-2 Example Test Data File

```
4

Jane    Simms   2
S       500.00  0.05
C       200.00  400.00

Owen    Bryant  1
C       200.00  0.00

Tim     Soley   2
S       1500.00 0.05
C       200.00  0.00

Maria   Soley   1
S       150.00  0.05
```

This exercise contains the following sections:

- "Task 1 – Creating a data Directory"
- "Task 2 – Copying the Resource File"
- "Task 3 – Creating the DataSource Class"
- "Task 4 – Deleting Unnecessary Classes"
- "Task 5 – Copying the TestReport Class"
- "Task 6 – Compiling the TestReport Class"
- "Task 7 – Running the BankPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise

- Java Development: Other Files: Creating Folders
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Setting Arguments
- Java Development: Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/11_fileio/exercise` directory.

Task 1 – Creating a `data` Directory

Tool Reference – Java Development: Other Files: Creating Folders

In this project, you create a `data` directory in the `BankPrj` project.

Task 2 – Copying the Resource File

In this task, you copy the `test.dat` file from the `resources/11_fileio` directory into the `data` directory.

Task 3 – Creating the `DataSource` Class

In this task, you create the `DataSource` class in the `com.mybank.data` source package of the `BankPrj` project with the following characteristics:

Class Name: **`DataSource`**

Project: **`BankPrj`**

Location: **`Source Packages`**

Package: **`com.mybank.data`**

The class must satisfy the UML diagram in Figure 11-1 on page L11-2. The `loadData` method must use `Bank` utility methods (`addCustomer` and `getCustomer`) to populate the customers recorded in the data file. Furthermore, the `Customer` class has the `addAccount` method to add the accounts from the data file.

Task 4 – Deleting Unnecessary Classes

In this task, you delete the following Java classes that are no longer used:

```
TestReport
TestAccount
TestAccount2
TestBatch
TestBanking
```

Task 5 – Copying the `TestReport` Class

In this task, you copy the `TestReport` class from the `resources/11_fileio` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class. If there are compilation errors, make necessary changes to the class and recompile it.

Task 7 – Running the `BankPrj` Project

In this task, you set the `TestReport` class as the main class of the `BankPrj` project, and run the `BankPrj` project with an argument `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting Arguments

1. Set the argument of the `BankPrj` project to `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting the Main Class

2. Set the main class of the `BankPrj` project to `com.mybank.test.TestReport`.

Tool Reference – Java Development: Java Application Projects: Running Projects

3. Run the `BankPrj` project. The output should be similar to the following:

```
Reading data file: ../data/test.dat
                        CUSTOMERS REPORT
                        ================

Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Savings Account: current balance is 150.0
```

Exercise 1: Reading a Data File (Level 2)

In this exercise, you create a class that reads customer and account data from a flat file.

This exercise contains the following sections:

- "Task 1 – Creating a data Directory"
- "Task 2 – Copying the Resource File"
- "Task 3 – Creating the DataSource Class"
- "Task 4 – Deleting Unnecessary Classes"
- "Task 5 – Copying the TestReport Class"
- "Task 6 – Compiling the TestReport Class"
- "Task 7 – Running the BankPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Creating Folders
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Setting Arguments
- Java Development: Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/11_fileio/exercise` directory.

Task 1 – Creating a `data` Directory

In this project, you create a `data` directory in the `BankPrj` project.

Task 2 – Copying the Resource File

In this task, you copy the `test.dat` file from the `resources/11_fileio` directory into the `data` directory.

Task 3 – Creating the `DataSource` Class

Complete the following steps:

1. Create the `DataSource` class in the `com.mybank.data` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`DataSource`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.data`**

2. Add the `dataFile` instance variable to the `DataSource` class.
3. Add a public constructor that takes a string argument `dataFilePath` and initializes the `dataFile` instance variable.
4. Add a public method, `loadData`, that populates the `Bank` customer objects and each customer's account objects. Here is the pseudo-code for this method:

```
read <number-of-customers>
for each customer record
  read <first-name> and <last-name>
  add customer to Bank
  read <number-of-accounts>
  for each account
    read <account-type-code>
    switch on <account-type-code>
```

```
Savings:
  read <initial-balance> and <interest-rate)
  create account and add account to customer
Checking:
  read <initial-balance> and <overdraft-amount)
  create account and add account to customer
```

Task 4 – Deleting Unnecessary Classes

In this task, you delete the following Java classes that are no longer used:

```
TestReport
TestAccount
TestAccount2
TestBatch
TestBanking
```

Task 5 – Copying the `TestReport` Class

In this task, you copy the `TestReport` class from the `resources/11_fileio` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class. If there are compilation errors, make necessary changes to the class and recompile it.

Task 7 – Running the `BankPrj` Project

In this task, you set the `TestReport` class as the main class of the `BankPrj` project, and run the `BankPrj` project with an argument `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting Arguments

1. Set the argument of the `BankPrj` project to `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting the Main Class

2. Set the main class of the `BankPrj` project to `com.mybank.test.TestReport`.

Tool Reference – Java Development: Java Application Projects: Running Projects

3. Run the `BankPrj` project. The output should be similar to the following:

```
Reading data file: ../data/test.dat
                        CUSTOMERS REPORT
                        ================

Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Savings Account: current balance is 150.0
```

Exercise 1: Reading a Data File (Level 3)

In this exercise, you create a class that reads customer and account data from a flat file.

This exercise contains the following sections:

- "Task 1 – Creating a data Directory"
- "Task 2 – Copying the Resource File"
- "Task 3 – Creating the DataSource Class"
- "Task 4 – Deleting Unnecessary Classes"
- "Task 5 – Copying the TestReport Class"
- "Task 6 – Compiling the TestReport Class"
- "Task 7 – Running the BankPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Other Files: Creating Folders
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Creating Java Classes
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Setting Arguments
- Java Development: Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/11_fileio/exercise` directory.

Task 1 – Creating a `data` Directory

Create a `data` directory in the `BankPrj` project.

1. With the `BankPrj` project open, click the Files tab.
2. Right-click `BankPrj` in the file tree.
3. Select New -> Other.
4. Select Other in the Categories pane.
5. Select Folder in the File Types pane.
6. Click Next.
7. Enter `data` for the folder name.
8. Click Finish.

Task 2 – Copying the Resource File

In this task, you copy the `test.dat` file from the `resources/11_fileio` directory into the `data` directory.

Task 3 – Creating the `DataSource` Class

Complete the following steps:

1. Create the `DataSource` class in the `com.mybank.data` source package of the `BankPrj` project with the following characteristics:

   Class Name: **`DataSource`**

   Project: **`BankPrj`**

   Location: **`Source Packages`**

   Package: **`com.mybank.data`**

```
package com.mybank.data;
// insert import statements here
public class DataSource {
  // insert code here
```

```
}
```

2. Add an import statement to import necessary class names:

```
import com.mybank.domain.*;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
```

3. Add the `dataFile` instance variable to the `DataSource` class.

```
  private File dataFile;
```

4. Add a public constructor that takes a string argument `dataFilePath` and initializes the `dataFile` instance variable.

```
  public DataSource(String dataFilePath) {
    this.dataFile = new File(dataFilePath);
  }
```

5. Add a public method, `loadData`, that populates the `Bank` customer objects and each customer's account objects, shown as follows:

```
public void loadData() throws IOException {
  // Data source variables
  Scanner input = new Scanner(dataFile);
  // Domain variables
  Customer customer;
  int numOfCustomers = input.nextInt();
  for ( int idx = 0; idx < numOfCustomers; idx++ ) {
    // Create customer object
    String firstName = input.next();
    String lastName = input.next();
    Bank.addCustomer(firstName, lastName);
    customer = Bank.getCustomer(idx);
    // Create customer accounts
    int numOfAccounts = input.nextInt();
    while ( numOfAccounts-- > 0 ) {
      // Create a specific type of account
      char accountType = input.next().charAt(0);
      switch ( accountType ) {
        // Savings account
        case 'S': {
          float initBalance = input.nextFloat();
          float interestRate = input.nextFloat();
          customer.addAccount(new SavingsAccount(initBalance,
                                                 interestRate));
          break;
        }
        // Checking account
        case 'C': {
          float initBalance = input.nextFloat();
          float overdraftProtection = input.nextFloat();
          customer.addAccount(new CheckingAccount(initBalance,
                                                  overdraftProtection));
          break;
        }
      } // END of switch
    } // END of create accounts loop
  } // END of create customers loop
}
```

Task 4 – Deleting Unnecessary Classes

In this task, you delete the following Java classes that are no longer used:

```
TestReport
TestAccount
TestAccount2
TestBatch
TestBanking
```

Task 5 – Copying the `TestReport` Class

In this task, you copy the `TestReport` class from the `resources/11_fileio` directory to the `com.mybank.test` source package of the `BankPrj` project.

Task 6 – Compiling the `TestReport` Class

In this task, you compile the `TestReport` class. If there are compilation errors, make necessary changes to the class and recompile it.

Task 7 – Running the `BankPrj` Project

In this task, you set the `TestReport` class as the main class of the `BankPrj` project, and run the `BankPrj` project with an argument `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting Arguments

1. Set the argument of the `BankPrj` project to `data/test.dat`.

Tool Reference – Java Development: Java Application Projects: Setting the Main Class

2. Set the main class of the `BankPrj` project to `com.mybank.test.TestReport`.

Tool Reference – Java Development: Java Application Projects: Running Projects

3. Run the `BankPrj` project. The output should be similar to the following:

```
Reading data file: ../data/test.dat
                        CUSTOMERS REPORT
                        ================

Customer: Simms, Jane
    Savings Account: current balance is 500.0
    Checking Account: current balance is 200.0

Customer: Bryant, Owen
    Checking Account: current balance is 200.0

Customer: Soley, Tim
    Savings Account: current balance is 1500.0
    Checking Account: current balance is 200.0

Customer: Soley, Maria
    Savings Account: current balance is 150.0
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 12

Building Java GUIs Using the Swing API

Objectives

Upon completion of this lab, you should be able to:

- Create a GUI for the ChatRoom project
- (Optional) Create a GUI for the Banking project

Exercise 1: Creating the ChatClient GUI Part 1 (Level 1)

In this exercise, you create a GUI for a *chat room* application. You use a complex layout to position properly several GUI components in a frame.

Figure 12-1 shows the GUI design that you will attempt to achieve.

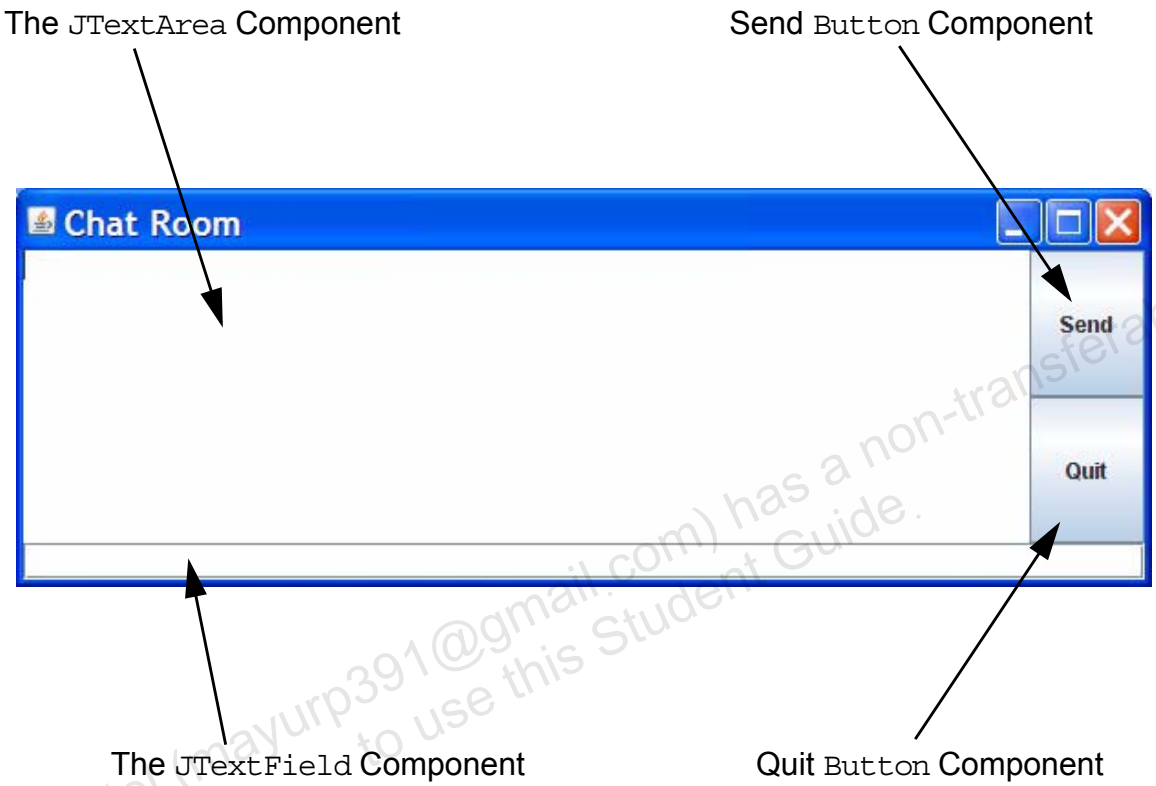

Figure 12-1 GUI Layout for the ChatClient Application

As shown in Figure 12-1, there are four components in this GUI. The main component is a JTextArea. The bottom component is a JTextField. There are two Button components on the right.

This exercise contains the following sections:

- "Task 1 – Creating the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"
- "Task 4 – Terminating the Running ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs
- Java Development: Java Classes: Terminating a Running Process

For this exercise, you first create the `ChatRoomPrj` project in the `projects` directory, and then work in this project for all the following exercises that are part of the `ChatRoomPrj` project.

Demonstration – The demonstration for this exercise can be found in the `demos/12_gui/exercise1` directory.

Task 1 – Creating the `ChatClient` Class

In this task, you create the `ChatClient` class in the source package of the `ChatRoomPrj` project implement the GUI design in Figure 12-1 on page 12-2.

Complete the following steps:

1. Create the `ChatRoomPrj` Java Application Project with the following characteristics:

   Project Name: **`ChatRoomPrj`**

   Project Location: **`projects`**

   Project Folder: **`projects/ChatRoomPrj`**

   Set as Main Project: **`No`**

   Create Main Class: **`No`**

2. Create the `ChatClient` class in the source package of the `ChatRoomPrj` project with the following characteristics:

   Class Name: **`ChatClient`**

Project: **ChatRoomPrj**

Location: **Source Packages**

Package: **default package**

The ChatClient class must implement the GUI design in Figure 12-1 on page 12-2.

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the ChatClient Program

In this task, you run the ChatClient program. You should see the GUI shown in Figure 12-1 on page 12-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Task 4 – Terminating the Running ChatClient Program

Tool Reference – Java Development: Java Classes: Terminating a Running Process

In this task, you terminate the running ChatClient program.

Exercise 1: Creating the ChatClient GUI Part 1 (Level 2)

In this exercise, you create a GUI for a *chat room* application. You use a complex layout to properly position several GUI components in a frame.

This exercise contains the following sections:

- "Task 1 – Creating the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"
- "Task 4 – Terminating the Running ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs
- Java Development: Java Classes: Terminating a Running Process

For this exercise, you first create the `ChatRoomPrj` project in the `projects` directory, and then work in this project for all the following exercises that are part of the `ChatRoomPrj` project.

Demonstration – The demonstration for this exercise can be found in the `demos/12_gui/exercise1` directory.

Task 1 – Creating the ChatClient Class

In this task, you create the ChatClient class in the source package of the ChatRoomPrj project implement the GUI design in Figure 12-1 on page 12-2.

Complete the following steps:

1. Create the ChatRoomPrj Java Application Project with the following characteristics:

   Project Name: **ChatRoomPrj**

   Project Location: **projects**

   Project Folder: **projects/ChatRoomPrj**

   Set as Main Project: **No**

   Create Main Class: **No**

2. Create the ChatClient class with the following characteristics:

   Class Name: **ChatClient**

   Project: **ChatRoomPrj**

   Location: **Source Packages**

   Package: **default package**

3. Add four instance variables to the ChatClient class to hold the GUI components.

4. Add a public constructor that initializes each of the four GUI component instance variables: The text area should be 10 rows tall and 50 columns wide, the text field should be 50 columns wide, the send button should have the word Send in the display, and the quit button should display a similar label.

5. Create a launchFrame method that constructs the layout of the components. Feel free to use nested panels and any layout managers that will help you construct the layout in the GUI design shown above.

6. Create the main method. This method instantiates a new ChatClient object and then calls the launchFrame method.

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the ChatClient Program

In this task, you run the ChatClient program. You should see the GUI shown in Figure 12-1 on page 12-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Task 4 – Terminating the Running ChatClient Program

Tool Reference – Java Development: Java Classes: Terminating a Running Process

In this task, you terminate the running ChatClient program.

Exercise 1: Creating the ChatClient GUI Part 1 (Level 3)

In this exercise, you create a GUI for a *chat room* application. You use a complex layout to properly position several GUI components in a frame.

This exercise contains the following sections:

- "Task 1 – Creating the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"
- "Task 4 – Terminating the Running ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Creating Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs
- Java Development: Java Classes: Terminating a Running Process

For this exercise, you first create the `ChatRoomPrj` project in the `projects` directory, and then work in this project for all the following exercises that are part of the `ChatRoomPrj` project.

Demonstration – The demonstration for this exercise can be found in the `demos/12_gui/exercise1` directory.

Task 1 – Creating the `ChatClient` Class

In this task, you create the ChatClient class in the source package of the `ChatRoomPrj` project implement the GUI design in Figure 12-1 on page 12-2.

Complete the following steps:

1. Create the `ChatRoomPrj` Java Application Project with the following characteristics:

   Project Name: **`ChatRoomPrj`**

   Project Location: **`projects`**

   Project Folder: **`projects/ChatRoomPrj`**

   Set as Main Project: **`No`**

   Create Main Class: **`No`**

2. Create the `ChatClient` class with the following characteristics:

   Class Name: **`ChatClient`**

   Project: **`ChatRoomPrj`**

   Location: **`Source Packages`**

   Package: **`default package`**

3. Import the `java.awt` and `javax.swing` packages.

```
import java.awt.*;
import javax.swing.*;
public class ChatClient {
  // insert code here
}
```

4. Add four instance variables to the `ChatClient` class to hold the GUI components.

```
  private JTextArea output;
  private JTextField input;
  private JButton sendButton;
  private JButton quitButton;
```

5. Add a public constructor that initializes each of the four GUI component instance variables: The text area should be 10 rows tall and 50 columns wide, the text field should be 50 columns wide, the send button should have the word `Send` in the display, and the quit button should display a similar label.

```
  public ChatClient() {
    output = new JTextArea(10,50);
    input = new JTextField(50);
    sendButton = new JButton("Send");
```

```
    quitButton = new JButton("Quit");
  }
```

6. Create a `launchFrame` method, which constructs the layout of the components. Feel free to use nested panels and any layout managers that will help you construct the layout in the GUI design shown above.

```
  public void launchFrame() {
    JFrame frame = new JFrame("Chat Room");

    // Use the Border Layout for the frame
    frame.setLayout(new BorderLayout());

    frame.add(output, BorderLayout.WEST);
    frame.add(input, BorderLayout.SOUTH);

    // Create the button panel
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(2,1));
    p1.add(sendButton);
    p1.add(quitButton);

    // Add the button panel to the center
    frame.add(p1, BorderLayout.CENTER);

    frame.pack();
    frame.setVisible(true);
  }
```

7. Create the `main` method. This method instantiates a new `ChatClient` object and then calls the `launchFrame` method.

```
  public static void main(String[] args) {
    ChatClient c = new ChatClient();
    c.launchFrame();
  }
```

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatClient` Program

In this task, you run the `ChatClient` program. You should see the GUI shown in Figure 12-1 on page 12-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Task 4 – Terminating the Running `ChatClient` Program

Tool Reference – Java Development: Java Classes: Terminating a Running Process

In this task, you terminate the running `ChatClient` program.

Exercise 2: Creating the Bank ATM GUI Part 1 (Advanced)

In this exercise, you create an automated teller machine (ATM) GUI for the Bank project. You use a complex layout to properly position several GUI components in a frame.

Note – This is an advanced exercise. It is optional and should only be attempted if you have already completed the previous exercise for this module.

Figure 12-2 shows the GUI design that you will attempt to achieve.

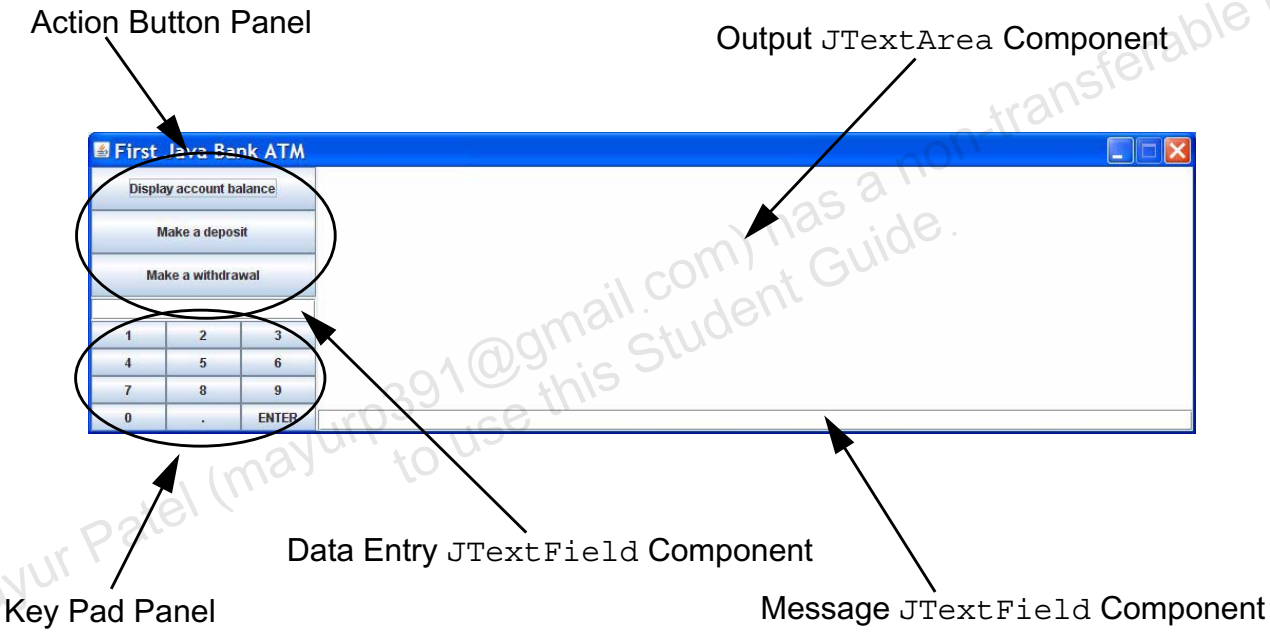

Figure 12-2 GUI Layout for the Bank Project

This exercise contains the following sections:

- "Task 1 – Copying the ATMClient Class"
- "Task 2 – Modifying the ATMClient Class"
- "Task 3 – Compiling the ATMClient Class"
- "Task 4 – Running the ATMClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Packages: Creating Java Packages
- Java Development: Java Classes: Copying Existing Resources
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/12_gui/exercise2` directory.

Task 1 – Copying the `ATMClient` Class

Complete the following steps:

1. Create the `com.mybank.gui` source package in the `BankPrj` project.
2. Copy the `ATMClient.java` template file from the `resources/12_gui` directory into the `com.mybank.gui` source package of the `BankPrj` project. This template code provides the `main` method, which initializes a set of bank customers and then launches the ATM GUI.

Task 2 – Modifying the `ATMClient` Class

In this task, you modify the `ATMClient` to implement the ATM GUI screen, shown in Figure 12-2 on page 12-12. The GUI components must have the following characteristics:

- The Message text field must have a width of 75 characters (called *columns* in the API documentation) and it must be read-only, meaning the user cannot type into the field.
- The Data Entry text field must have a width of 10 characters and it must be read-only. The user will use the key pad buttons to enter data which will be displayed in the Data Entry field. This action will be coded in the next module.
- The Output text area must have a width of 75 characters and a height (called *rows* in the API) of 10 characters and it must be read-only.

Note – In this exercise, you will only be creating the layout of the ATM screen. You will not be creating the code to make the buttons respond to user actions; you will do that in the next module.

Task 3 – Compiling the `ATMClient` Class

In this task, you compile the ATMClient class.

Task 4 – Running the `ATMClient` Program

Complete the following steps:

1. Set the main class of the `BankPrj` project to `com.mybank.gui.ATMClient`.
2. Run the `BankPrj` project. You should see the GUI shown in Figure 12-2 on page 12-12.

Hints

These hints might help you to solve this exercise.

- Use the `setEnabled(false)` method to make a component read-only.
- A grid layout can be used to create a vertical layout by placing a `1` in the `rows` parameters of the `GridLayout(int rows, int columns)` constructor.
- A grid layout can be used to create a horizontal layout by placing a `1` in the `columns` parameters of the `GridLayout(int rows, int columns)` constructor.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 13

Handling GUI-Generated Events

Objectives

Upon completion of this lab, you should be able to:

- Create the GUI event handlers for the ChatRoom project
- (Optional) Create the GUI event handlers for the Banking project

Exercise 1: Creating the ChatClient GUI Part 2 (Level 1)

In this exercise, you implement the basic event handlers for the *chat room* application. At this stage in the development of the ChatClient GUI, you need to create the following event listeners:

- Create an ActionListener that copies the text from the input text field into the output text area when the send button is pressed.
- Create an ActionListener that copies the text from the input text field into the output text area when the Enter key is pressed in the input text field.
- Create an ActionListener that will quit the program when the Quit button is pressed. (Hint – Use System.exit(0).)
- Create a WindowListener that will quit the program when the close widget is pressed on the frame.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the ChatRoomPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/13_events/exercise1 directory.

Task 1 – Modifying the ChatClient Class

In this task, you modify the ChatClient class in the source package of the ChatRoomPrj project. This class must implement event listeners listed in the introduction to this exercise.

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the ChatClient Program

In this task, you run the ChatClient program. Test the behavior of the event listeners you added.

Hints

Use the following hints during this exercise:

- Have the listeners (inner classes) access the instance variables of their containing class to refer to the components like the output text area and the input text field. Remember that you made the components instance variables in the previous lab.
- Remember to import the java.awt.event package.
- To get the text from a JTextArea or JTextField, you can use the getText method; to set the text, use either the setText or append method.

Exercise 1: Creating the ChatClient GUI Part 2 (Level 2)

In this exercise, you implement the basic event handlers for the *chat room* application. At this stage in the development of the ChatClient GUI, you need to create the event listeners listed in "Exercise 1: Creating the ChatClient GUI Part 2 (Level 1)" on page L13-2.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the ChatRoomPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/13_events/exercise1 directory.

Task 1 – Modifying the ChatClient Class

In this task, you modify the ChatClient class in the source package of the ChatRoomPrj project. This class must implement event listeners listed in the introduction to "Exercise 1: Creating the ChatClient GUI Part 2 (Level 1)" on page L13-2.

Complete the following steps:

1. Import the `java.awt.event` package.
2. Add the `ActionListener` for the Send button. This listener must extract the text from the text field and display that text in the text area. Use an inner class for this listener.
3. Add the `ActionListener` for the text field. Can you use the same listener in Step 2?
4. Add the `WindowListener` to the GUI frame. This listener must exit the `ChatClient` program. Use an inner class for this listener.
5. Add the `ActionListener` for the Quit button. This listener must exit the `ChatClient` program. Use an anonymous inner class for this listener.

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatClient` Program

In this task, you run the `ChatClient` program. Test the behavior of the event listeners you added.

Exercise 1: Creating the ChatClient GUI Part 2 (Level 3)

In this exercise, you implement the basic event handlers for the *chat room* application. At this stage in the development of the ChatClient GUI, you need to create the event listeners listed in "Exercise 1: Creating the ChatClient GUI Part 2 (Level 1)" on page L13-2.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the ChatRoomPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/13_events/exercise1 directory.

Task 1 – Modifying the ChatClient Class

In this task, you modify the ChatClient class in the source package of the ChatRoomPrj project. This class must implement event listeners listed in the introduction to "Exercise 1: Creating the ChatClient GUI Part 2 (Level 1)" on page L13-2.

1. Import the java.awt.event package.

```
import java.awt.event.*;
public class ChatClient {
  // your code here
}
```

2. Add the ActionListener for the Send button. This listener must extract the text from the text field and display that text in the text area. Use an inner class for this listener.

```
private void launchFrame() {
  // GUI component initalization code here
  sendButton.addActionListener(new SendHandler());
  // more code
}

private class SendHandler implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    String text = input.getText();
    output.append(text+"\n");
    input.setText("");
  }
}
```

3. Add the ActionListener for the text field. Can you use the same listener in Step 2?

```
private void launchFrame() {
  // GUI component initalization code here
  input.addActionListener(new SendHandler());
  // more code
}
```

In this solution, the SendHandler inner class was reused to reduce redundant code.

4. Add the WindowListener to the GUI frame. This listener must exit the ChatClient program. Use an inner class for this listener.

```
private void launchFrame() {
  // GUI component initalization code here
  frame.addWindowListener(new CloseHandler());
  // more code
}

private class CloseHandler extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}
```

5. Add the ActionListener for the Quit button. This listener must exit the ChatClient program. Use an anonymous inner class for this listener.

```
private void launchFrame() {
  // GUI component initalization code here
  quitButton.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
         System.exit(0);
       }
  });
  // more code
}
```

In this solution, the CloseHandler was not reused because the Quit button requires an ActionListener rather than a WindowListener. How could you have satisfied both listener interfaces using the CloseHandler class?

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the ChatClient Program

In this task, you run the ChatClient program. Test the behavior of the event listeners you added.

Exercise 2: Creating the Bank ATM GUI Part 2 (Advanced)

In this exercise, you modify the `ATMClient` class in the `BankPrj` project by adding necessary event handling capabilities.

Note – This is an advanced exercise. It is optional and should only be attempted if you have already completed the previous exercise for this module.

The simulated ATM must behave in this manner:

1. The ATM screen is displayed.

   At this stage, the Action buttons must be disabled and the following text must be displayed in the Output text area: `Enter your customer ID into the key pad and press the ENTER button.`

2. The user enters the ID using the key pad buttons. Each number is displayed in the Data Entry text field.

   When the `ENTER` button is selected, the application retrieves the specified customer object and displays the following text in the Output text area: `Welcome` *`FIRST-NAME LAST-NAME`* if the customer was found; otherwise, the following text is displayed: `Customer ID was not found`. Finally, if the customer was found, then the Action buttons are enabled.

3. The user selects one of the Action buttons.

   If the action is either deposit or withdraw, then the user is prompted to enter the amount. The user can then enter the amount by clicking the key pad buttons which (again) places digits in the Data Entry text field. When the `ENTER` button is selected, the program executes the action with the amount entered in the Data Entry text field. Finally, the result of the operation must be displayed in the Output text area. At the end of the operation, the GUI should begin again at the top of the ATM operation cycle.

This exercise contains the following sections:

- “Task 1 – Modifying the ATMClient Class”
- “Task 2 – Compiling the ATMClient Class”
- “Task 3 – Running the BankPrj Project”

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `BankPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/13_events/exercise2` directory.

Task 1 – Modifying the `ATMClient` Class

In this task, you modify the `ATMClient` class in the source package of the `BankPrj` project. This class must rely on necessary event listeners to implemented the required behavior.

Task 2 – Compiling the `ATMClient` Class

In this task, you compile the `ATMClient` class.

Task 3 – Running the `BankPrj` Project

In this task, you run the `BankPrj` project.

The following is an example ATM transaction for `Owen Bryant`:

```
Enter your customer ID into the key pad and press the ENTER button.
Welcome Owen Bryant

Your account balance is: 200.0
Enter the amount to deposit into the key pad and press the ENTER button.
Your deposit of 100.0 was successful.
Your account balance is: 300.0
Enter the amount to withdraw into the key pad and press the ENTER button.
Your withdrawal of 250.0 was successful.
Your account balance is: 50.0
```

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 14

GUI-Based Applications

Objectives

Upon completion of this lab, you should be able to add menus to the GUI for the ChatRoom project.

Exercise: Creating the ChatClient GUI, Part 3 (Level 1)

In this exercise, you enhance the GUI for a *chat room* application. You complete the GUI for the ChatClient by adding a JComboBox component, a JScrollPane component, and two menus.

Figure 14-1 shows the GUI design that you will attempt to achieve.

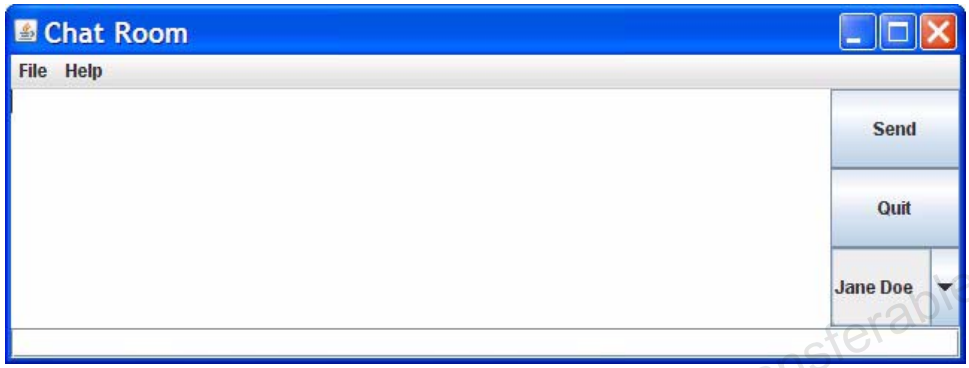

Figure 14-1 GUI Layout for the ChatClient Application

You need to add four features to the existing GUI:

- Add the user name JComboBox component under the Send and Quit buttons. This component enables you to select a name that is posted with every chat line that you enter. You must alter the Send button and JTextField listeners to write the username to the output JTextArea component. Add several user name options including your full name and a few nicknames, such as *1337dud3* or *Java Geek*.
- Put the JTextArea component into a JScrollPane component. Add a vertical scroll bar but no horizontal scroll bar. Auto scroll the text to the bottom of the JTextArea as the user adds text to the text chat window.
- Add the File menu. This menu must include a Quit menu item that terminates the program when selected.
- Add the Help menu. This menu must include an *About* menu item that pops up a simple dialog box, which displays a comment about the program and about you, the developer.

This exercise contains the following sections:

- “Task 1 – Modifying the ChatClient Class”
- “Task 2 – Compiling the ChatClient Class”
- “Task 3 – Running the ChatClient Program”

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `ChatRoomPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/14_guiapps/exercise1` directory.

Task 1 – Modifying the `ChatClient` Class

In this task, you modify the `ChatClient` class in the source package of the `ChatRoomPrj` project. This class must implement the GUI design in Figure 14-1 on page L14-2.

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatClient` Program

In this task, you run the `ChatClient` program. You should see the GUI shown in Figure 14-1 on page L14-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Exercise: Creating the ChatClient GUI, Part 3 (Level 2)

In this exercise, you enhance the GUI for a *chat room* application. You complete the GUI for the ChatClient by adding a JComboBox component and two menus.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the ChatRoomPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/14_guiapps/exercise1 directory.

Task 1 – Modifying the ChatClient Class

In this task, you modify the ChatClient class in the source package of the ChatRoomPrj project. This class must implement the GUI design in Figure 14-1 on page L14-2.

Complete the following steps:

1. Add the user name JComboBox component under the Send and Quit buttons. This component enables you to select a name that is posted with every chat line that you enter. Add several user name options including your full name and a few nicknames, such as *1337dud3* or *Java Geek*.
2. Enhance the listeners for the Send button and JTextField to write the user name to the output JTextArea component.
3. Put the JTextArea component into a JScrollPane component. Add a vertical scroll bar but no horizontal scroll bar. Auto scroll the text to the bottom of the JTextArea as the user adds text to the text chat window.
4. Add the File menu. This menu must include a Quit menu item that terminates the program when selected.
5. Add the Help menu. This menu must include an *About* menu item that pops up a simple dialog box, which displays a comment about the program and about you, the developer.

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the ChatClient Program

In this task, you run the ChatClient program. You should see the GUI, shown in Figure 14-1 on page L14-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Exercise: Creating the ChatClient GUI, Part 3 (Level 3)

In this exercise, you enhance the GUI for a *chat room* application. You complete the GUI for the ChatClient by adding a JComboBox component and two menus.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatClient Program"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the ChatRoomPrj project in the projects directory.

Demonstration – The demonstration for this exercise can be found in the demos/14_guiapps/exercise1 directory.

Task 1 – Modifying the ChatClient Class

In this task, you modify the ChatClient class in the source package of the ChatRoomPrj project. This class must implement the GUI design in Figure 14-1 on page L14-2.

Complete the following steps:

1. Add the user name JComboBox component under the Send and Quit buttons. This component enables you to select a name that is posted with every chat line that you enter. Add several user name options including your full name and a few nicknames, such as *1337dud3* or *Java Geek*.

```
public class ChatClient {
  // existing code here
  private JComboBox usernames;
  // existing code here
  public ChatClient() {
    // more GUI components initialized
    usernames = new JComboBox();
    usernames.addItem("Jane Doe");
    usernames.addItem("1337dud3");
    usernames.addItem("Java Geek");
  }
  public void launchFrame() {
        // existing code here
        // Create the button panel
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(3,1));
        p1.add(sendButton);
        p1.add(quitButton);
        p1.add(usernames);
        // existing code here
  }
  // existing code here
}
```

2. Add the aboutDialog instance variable of type javax.swing.JDialog to the ChatClient class.

3. Add the frame instance variable of type javax.swing.JFrame to the ChatClient class.

4. Modify the declaration and initialization of the frame variable in the launchFrame method to the following:

```
frame = new JFrame("Chat Room");
```

5. Enhance the listeners for the Send button and JTextField to write the user name to the output JTextArea component.

```
  private class SendHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      String text = input.getText();
      output.append(usernames.getSelectedItem() + ": " + text + "\n");
      input.setText("");
    }
```

```
  }
```

6. Add a JScrollPane component to the project and add the JTextArea component to the a JScrollPane. Add a vertical scroll bar but no horizontal scroll bar.

```
// Add declaration
private JScrollPane textPane;
// existing code here

// Put JTextArea into JScrollPane and define scroll bars
textPane = new JScrollPane(output,
    ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
// existing code here

// Add JScrollPane to Layout
frame.add(textPane, BorderLayout.WEST);
```

7. Auto scroll the text to the bottom of the JTextArea as the user adds text to the text chat window.

```
// Add to SendHandler to enable AutoScroll
output.setCaretPosition(output.getDocument().getLength()-1);
```

8. Add the File menu. This menu must include a Quit menu item that terminates the program when selected.

```
  public void launchFrame() {
    // existing code here
    // Create menu bar and File menu
        JMenuBar mb = new JMenuBar();
        JMenu file = new JMenu("File");
        JMenuItem quitMenuItem = new JMenuItem("Quit");
        quitMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        file.add(quitMenuItem);
        mb.add(file);
        frame.setJMenuBar(mb);
  // existing code here
  }
```

9. Add the Help menu. This menu must include an *About* menu item that pops up a simple dialog box, that displays a comment about the program and about you, the developer.

```
  public void launchFrame() {
```

```
    // existing menu bar code here
         // Add Help menu to menu bar
         JMenu help = new JMenu("Help");
         JMenuItem aboutMenuItem = new JMenuItem("About");
         aboutMenuItem.addActionListener(new AboutHandler());
         help.add(aboutMenuItem);
         mb.add(help);
   // existing code here
  }


  private class AboutHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      // Create the aboutDialog when it is requested
      if ( aboutDialog == null ) {
        aboutDialog = new AboutDialog(frame, "About", true);
      }
      aboutDialog.setVisible(true);
    }
  }


    private class AboutDialog extends JDialog implements ActionListener
{
        public AboutDialog(Frame parent, String title, boolean modal) {
            super(parent,title,modal);
            add(new JLabel("The ChatClient is a neat tool that allows you
to talk " +
                       "to other ChatClients via a
ChatServer"),BorderLayout.NORTH);
            JButton b = new JButton("OK");
            add(b,BorderLayout.SOUTH);
            b.addActionListener(this);
            pack();
        }
        // Hide the dialog box when the OK button is pushed
        public void actionPerformed(ActionEvent e) {
            setVisible(false);
        }
    }
```

Task 2 – Compiling the ChatClient Class

In this task, you compile the ChatClient class.

Task 3 – Running the `ChatClient` Program

In this task, you run the `ChatClient` program. You should see the GUI shown in Figure 14-1 on page L14-2. If your GUI does not look exactly like the figure, then edit the code to tweak the design to match this figure.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 15

Threads

Objectives

Upon completion of this lab, you should be able to create a simple multithreaded application.

Exercise: Using Multithreaded Programming (Level 1)

In this exercise, you become familiar with the concepts of multithreading by writing a multithreaded program.

The purpose of this lab is to create three threads and run them at the same time. While they are running, they print out their names to the standard output stream. By observing what is printed, you can observe how the threads run and in what order.

This exercise contains the following sections:

- "Task 1 – Creating the PrintMe Class"
- "Task 2 – Creating the TestThreeThreads Class"
- "Task 3 – Compiling the TestThreeThreads Class"
- "Task 4 – Running the TestThreeThreads Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `ThreadProject` project in the `exercises/15_threads` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/15_threads` directory.

Task 1 – Creating the `PrintMe` Class

Complete the following steps:

1. Open the `ThreadProject` project in the `exercises/15_threads` directory.
2. Create a `PrintMe` class in the source package of the `ThreadProject` with the following characteristics:

   Class Name: **`PrintMe`**

   Project: **`ThreadProject`**

   Implements: **`Runnable`**

   Location: **`Source Packages`**

   Package: **`default package`**

The class implements the `Runnable` interface. The `run` method of the class performs the following actions 10 times: Print the name of the current thread and then sleep for 2 seconds.

Task 2 – Creating the `TestThreeThreads` Class

In this task, you create a `TestThreeThreads` class in the source package of the `ThreadProject` with the following characteristics:

Class Name: **`TestThreeThreads`**

Project: **`ThreadProject`**

Location: **`Source Packages`**

Package: **`default package`**

The main method of the class creates three threads using the `PrintMe` runnable class. Give each thread a unique name using the `setName` method. Start each thread.

Task 3 – Compiling the `TestThreeThreads` Class

In this task, you compile the `TestThreeThreads` class.

Task 4 – Running the `TestThreeThreads` Program

In this task, you run the `TestThreeThreads` program. You should see output similar to this:

```
T3 - Moe
T2 - Curly
T1 - Larry
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
T1 - Larry
T3 - Moe
T2 - Curly
```

Run this program several times. You might see different results for each execution.

Discussion – Can you explain the behavior of your program?

Exercise: Using Multithreaded Programming (Level 2)

In this exercise, you become familiar with the concepts of multithreading by writing a multithreaded program.

This exercise contains the following sections:

- "Task 1 – Creating the PrintMe Class"
- "Task 2 – Creating the TestThreeThreads Class"
- "Task 3 – Compiling the TestThreeThreads Class"
- "Task 4 – Running the TestThreeThreads Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `ThreadProject` project in the `exercises/15_threads` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/15_threads` directory.

Task 1 – Creating the `PrintMe` Class

Complete the following steps:

1. Open the `ThreadProject` project in the `exercises/15_threads` directory.
2. Create the `PrintMe` class with the following characteristics:

   Class Name: **`PrintMe`**

   Project: **`ThreadProject`**

   Implements: **`Runnable`**

   Location: **`Source Packages`**

   Package: **`default package`**

   This class must implement the `Runnable` interface.
3. Create the `run` method to loop the following 10 times: Print the name of the current thread and then sleep for 2 seconds.

Task 2 – Creating the `TestThreeThreads` Class

In this task, you create a `TestThreeThreads` class in the source package of the `ThreadProject` project.

Complete the following steps:

1. Create the `TestThreeThreads` class with the following characteristics:

   Class Name: **`TestThreeThreads`**

   Project: **`ThreadProject`**

   Location: **`Source Packages`**

   Package: **`default package`**
2. Create the `main` method.
   a. Create three `Thread` objects and pass an instance of the `PrintMe` class to each constructor.
   b. Give each thread a unique name using the `setName` method.
   c. Start each thread.

Task 3 – Compiling the `TestThreeThreads` Class

In this task, you compile the `TestThreeThreads` class.

Task 4 – Running the `TestThreeThreads` Program

In this task, you run the `TestThreeThreads` program. You should see output similar to that shown in "Task 4 – Running the TestThreeThreads Program" on page L15-4.

Exercise: Using Multithreaded Programming (Level 3)

In this exercise, you become familiar with the concepts of multithreading by writing a multithreaded program.

This exercise contains the following sections:

- "Task 1 – Creating the PrintMe Class"
- "Task 2 – Creating the TestThreeThreads Class"
- "Task 3 – Compiling the TestThreeThreads Class"
- "Task 4 – Running the TestThreeThreads Program"

Preparation

There is no preparation required for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java Development: Java Classes: Creating Java Classes
- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Development: Java Classes: Modifying Java Classes: Executing Java Programs

For this exercise, you work in the `ThreadProject` project in the `exercises/15_threads` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/15_threads` directory.

Task 1 – Creating the `PrintMe` Class

Complete the following steps:

1. Open the `ThreadProject` project in the `exercises/15_threads` directory.

2. Create the `PrintMe` class with the following characteristics:

   Class Name: **`PrintMe`**

   Project: **`ThreadProject`**

   Implements: **`Runnable`**

   Location: **`Source Packages`**

   Package: **`default package`**

   This class must implement the `Runnable` interface.

```
class PrintMe implements Runnable {
  // more code here
}
```

3. Create the `run` method to loop the following 10 times: Print the name of the current thread and then sleep for 2 seconds.

```
  public void run() {
    for(int x = 0; x < 10; x++) {
      System.out.println(Thread.currentThread().getName());
      try {
        Thread.sleep(2000);
      } catch(Exception e) {}
    }
  }
```

Task 2 – Creating the `TestThreeThreads` Class

In this task, you create a `TestThreeThreads` class in the source package of the `ThreadProject` project.

Complete the following steps:

1. Create the `TestThreeThreads` class with the following characteristics:

   Class Name: **`TestThreeThreads`**

   Project: **`ThreadProject`**

   Location: **`Source Packages`**

   Package: **`default package`**

```
public class TestThreeThreads {
  // more code here
}
```

2. Create the `main` method.

   a. Create three `Thread` objects and pass an instance of the `PrintMe` class to each constructor.

```
public static void main(String[] args) {
  Runnable prog = new PrintMe();
  Thread t1 = new Thread(prog);
  Thread t2 = new Thread(prog);
  Thread t3 = new Thread(prog);

  // more code here
}
```

   b. Give each thread a unique name using the `setName` method.

```
  t1.setName("T1 - Larry");
  t2.setName("T2 - Curly");
  t3.setName("T3 - Moe");
```

   c. Start each thread.

```
  t1.start();
  t2.start();
  t3.start();
```

Task 3 – Compiling the `TestThreeThreads` Class

In this task, you compile the `TestThreeThreads` class.

Task 4 – Running the `TestThreeThreads` Program

In this task, you run the `TestThreeThreads` program. You should see output similar to that shown in "Task 4 – Running the TestThreeThreads Program" on page L15-4.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____

Lab 16

Networking

Objectives

Upon completion of this lab, you should be able to build a GUI client to connect to a remote server by using Transmission Control Protocol / Internet Protocol (TCP / IP).

Exercise: Creating a Socket Client (Level 1)

In this exercise, you write the code to connect the *chat room* client with the chat server.

The chat server is responsible for sending messages received from one client to all connected clients (including the original sender). Figure 16-1 shows an architecture diagram of several clients attached to the single chat server. In this scenario, Simon types **`This is cool!`** into the message `JTextField` component. Simon's client then prepends his name `Simon:` onto the message and sends it to the server over the output stream (Step 1). The server receives the message and then forwards the message to each attached client (Steps 2–4); the order of the forwarded messages is unimportant.

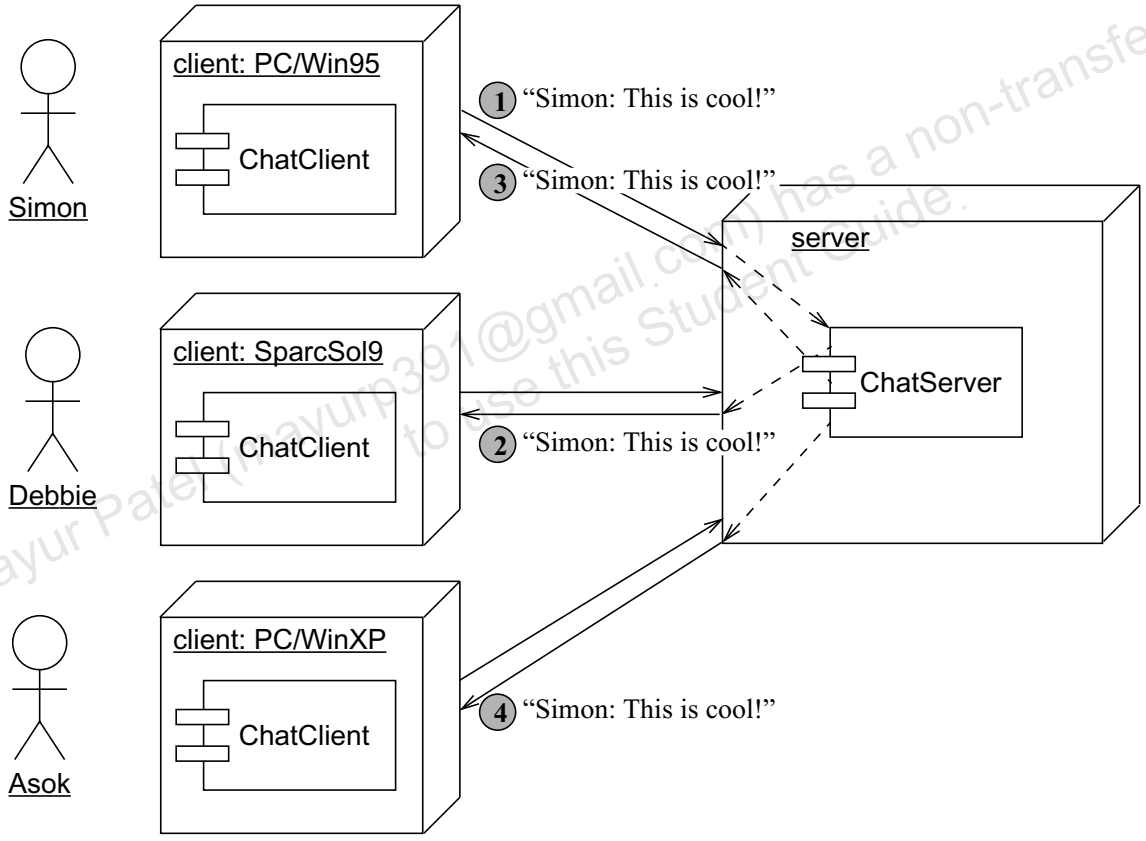

Figure 16-1 The Chat Server Sends Any Message to All Attached Clients

The chat client (your application) must be modified to perform two behaviors: It must send the user's messages to the server, and it must display the messages it receives from the server to the output `JTextArea` component. Figure 16-2 shows a detailed design for the elements of the `ChatClient` application. You will add the `doConnect` method to the `ChatClient` class to initiate the socket connection to the chat server.

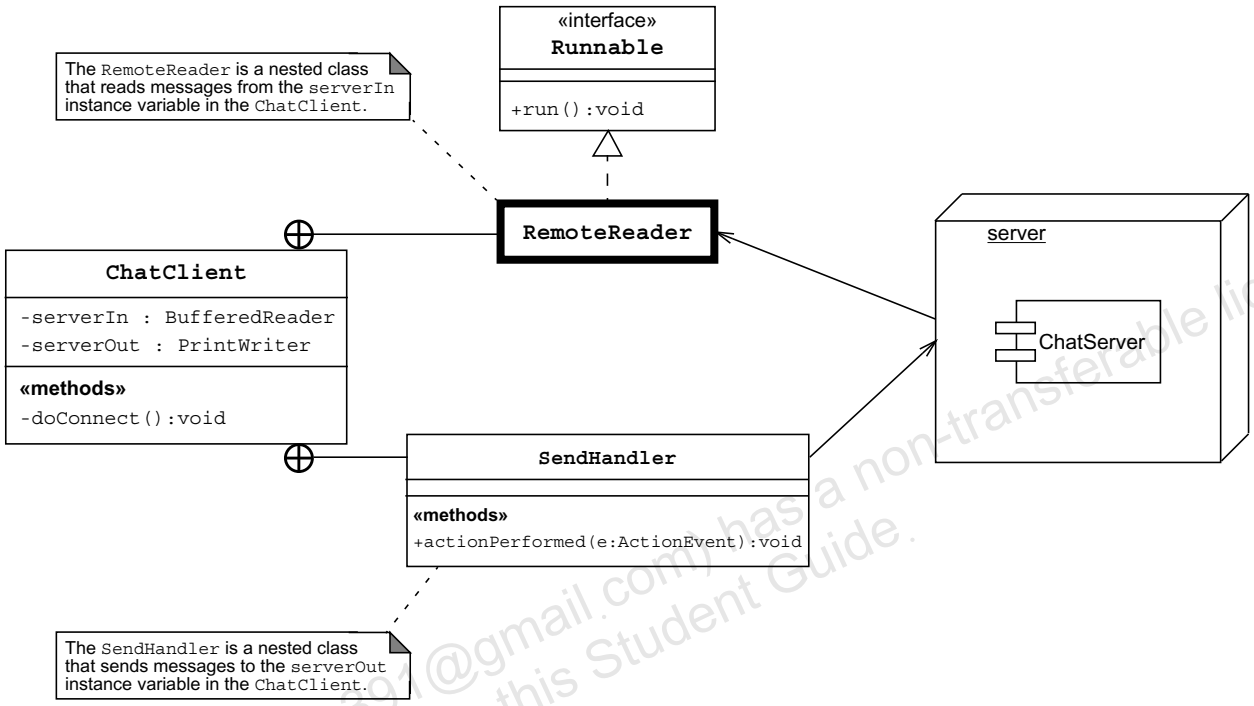

Figure 16-2 Detailed Design of the `ChatClient` Application

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatRoomPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Setting VM Options
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `ChatRoomPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/16_network` directory.

Task 1 – Modifying the `ChatClient` Class

In this task, you modify the `ChatClient` class in the source package of the `ChatRoomPrj` project. Implement the client-server communication based on the detailed design in Figure 16-2 on page L16-3.

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatRoomPrj` Project

Complete the following steps:

1. Set the main class of the `ChatRoomPrj` project to `ChatClient`.

Tool Reference – Java Development: Java Application Projects: Setting VM Options

2. Set the following system properties to the `ChatRoomPrj` project:

```
-DserverIP=<server-host>
-DserverPort=2000
```

Note – Ask your instructor for the value of the server-host.

3. Run the `ChatRoomPrj` project.

   If your program connects to the server successfully, then the output text area will display a message from the server with a secret passphrase. When you send that passphrase to the server, you then trigger an event on the server that plays a gong. This event will indicate that your application is working correctly.

Good luck and have fun!

Exercise: Creating a Socket Client (Level 2)

In this exercise, you write the code to connect the *chat room* client with the chat server.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatRoomPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Setting VM Options
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `ChatRoomPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/16_network` directory.

Task 1 – Modifying the `ChatClient` Class

In this task, you modify the `ChatClient` class in the source package of the `ChatRoomPrj` project. Implement the client-server communication based on the detailed design in Figure 16-2 on page L16-3.

Complete the following steps:

1. Import the `java.net` and `java.io` packages.

2. Add instance variables to hold the input and output streams for the socket connection. You might need additional instance variables, depending on your implementation.
3. Add the `doConnect` method to initiate the TCP/IP socket connection to the server. Ask your instructor for the hostname (or IP address) and port number of the server application.
   a. Initialize server IP and port information.
   b. Create the connection to the chat server.
   c. Prepare the input stream and store it in an instance variable.
   d. Prepare the output stream and store it in an instance variable.
   e. Launch the reader thread.
   f. Use a catch clause to capture any exceptions.
4. Modify the `launchFrame` method to call the `doConnect` method.
5. Modify the `SendHandler` nested class to send the message text (and the user name) to the socket output stream. Delete the code that displayed the message to the output text area.
6. Create the `RemoteReader` nested class that implements the `Runnable` interface. The `run` method must read a line at a time from the socket input stream in an infinite loop.

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatRoomPrj` Project

Complete the following steps:

1. Set the main class of the `ChatRoomPrj` project to `ChatClient`.

Tool Reference – Java Development: Java Application Projects: Setting VM Options

2. Set the following system properties to the `ChatRoomPrj` project:

```
-DserverIP=<server-host>
-DserverPort=2000
```

Note – Ask your instructor for the value of the server-host.

3. Run the `ChatRoomPrj` project.

   If your program connects to the server successfully, then the output text area will display a message from the server with a secret passphrase. When you send that passphrase to the server, you then trigger an event on the server that plays a gong. This event will indicate that your application is working correctly.

Good luck and have fun!

Exercise: Creating a Socket Client (Level 3)

In this exercise, you will write the code to connect the *chat room* client with the chat server.

This exercise contains the following sections:

- "Task 1 – Modifying the ChatClient Class"
- "Task 2 – Compiling the ChatClient Class"
- "Task 3 – Running the ChatRoomPrj Project"

Preparation

There is no preparation for this exercise.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Classes: Modifying Java Classes: Compiling Java Classes
- Java Application Projects: Setting the Main Class
- Java Development: Java Application Projects: Setting VM Options
- Java Development: Java Application Projects: Running Projects

For this exercise, you work in the `ChatRoomPrj` project in the `projects` directory.

Demonstration – The demonstration for this exercise can be found in the `demos/16_network` directory.

Task 1 – Modifying the `ChatClient` Class

In this task, you modify the `ChatClient` class in the source package of the `ChatRoomPrj` project. Implement the client-server communication based on the detailed design in Figure 16-2 on page L16-3.

Complete the following steps:

1. Import the `java.net` and `java.io` packages.

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

2. Add instance variables to hold the input and output streams for the socket connection. You might need additional instance variables, depending on your implementation.

```
public class ChatClient {
  // existing code here
  private Socket connection = null;
  private BufferedReader serverIn = null;
  private PrintStream serverOut = null;
  // existing code here
}
```

3. Add the `doConnect` method to initiate the TCP/IP socket connection to the server. Ask your instructor for the hostname (or IP address) and port number of the server application.

```
  private void doConnect() {
```

a. Initialize server IP and port information.

```
    // Initialize server IP and port information
    String serverIP = System.getProperty("serverIP", "127.0.0.1");
    String serverPort = System.getProperty("serverPort", "2000");
```

Note – You could have hard coded the server IP address and port number in the `Socket` constructor call below. The code for this step demonstrates how to make these values dynamic at runtime. The system properties, `serverIP` and `serverPort`, can be assigned on the `java` command using the `-D` option, as follows:

```
java -DserverIP=myhost.example.com
     -DserverPort=47 ChatClient
```

b. Create the connection to the chat server.

```
    try {
      connection = new Socket(serverIP, Integer.parseInt(serverPort));
```

c. Prepare the input stream and store it in an instance variable.

```
      InputStream is = connection.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      serverIn = new BufferedReader(isr);
```

d. Prepare the output stream and store it in an instance variable.

```
    serverOut = new PrintStream(connection.getOutputStream());
```

e. Launch the reader thread.

```
    Thread t = new Thread(new RemoteReader());
    t.start();
```

f. Use a catch clause to capture any exceptions.

```
  } catch (Exception e) {
    System.err.println("Unable to connect to server!");
    e.printStackTrace();
  } // END of try-catch block
} // END of doConnect method
```

4. Modify the `launchFrame` method to call the `doConnect` method.

```
private void launchFrame() {
  // existing code here
  doConnect();
}
```

5. Modify the `SendHandler` nested class to send the message text (and the user name) to the socket output stream. Delete the code that displayed the message to the output text area.

```
private class SendHandler implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    String text = input.getText();
    text = usernames.getSelectedItem() + ": " + text + "\n";
    serverOut.print(text);
    input.setText("");
  } // END of actionPerformed method
} // END of SendHandler nested class
```

6. Create the `RemoteReader` nested class that implements the `Runnable` interface. The `run` method must read a line at a time from the socket input stream in an infinite loop.

```
private class RemoteReader implements Runnable {
  public void run() {
    try {
      while ( true ) {
        String nextLine = serverIn.readLine();
        output.append(nextLine + "\n");
       output.setCaretPosition(output.getDocument().getLength()-1);
      }
    } catch (Exception e) {
      System.err.println("Error while reading from server.");
      e.printStackTrace();
    }
  } // END of run method
} // END of RemoteReader nested class
```

Task 2 – Compiling the `ChatClient` Class

In this task, you compile the `ChatClient` class.

Task 3 – Running the `ChatRoomPrj` Project

Complete the following steps:

1. Set the main class of the `ChatRoomPrj` project to `ChatClient`.

Tool Reference – Java Development: Java Application Projects: Setting VM Options

2. Set the following system properties to the `ChatRoomPrj` project:

```
-DserverIP=<server-host>
-DserverPort=2000
```

Note – Ask your instructor for the value of the server-host.

3. Run the `ChatRoomPrj` project.

   If your program connects to the server successfully, then the output text area will display a message from the server with a secret passphrase. When you send that pass phrase to the server, you then trigger an event on the server that plays a gong. This event will indicate that your application is working correctly.

Good luck and have fun!

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences

  _____

  _____

  _____

- Interpretations

  _____

  _____

  _____

- Conclusions

  _____

  _____

  _____

- Applications

  _____

  _____

  _____