Chapter 4: Libraries and Error Handling

Learning Outcomes:

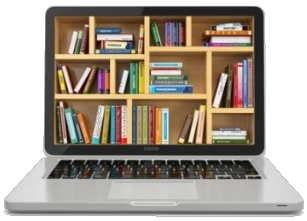
- ✓ Identify the importance of the Java API
- ✓ Implement a static method in a program
- ✓ Implement a non-static method in a program
- ✓ Implement the readLine() method and the 'TRY/CATCH' technique into an interactive program

Prerequisite Knowledge:

- ✓ Complete Chapter 3
- ✓ Be familiar with the terms 'class' and 'method', as well as their role in the Java code structure
- ✓ Be able to confidently implement selection techniques in Java, especially IF statements

Keywords: Class, Java API, Package, Static, Instantiated, Import

4.1 The Theory: Programming Libraries



Most modern-day programming languages come equipped with extensive libraries – in other words, lots of pre-written code to speed up the development time of projects. Java is no different; the Java API (Application Programming Interface – a fancy name for a library) comes complete with nearly 4,000 pre-written classes, containing thousands of methods. There is pretty much a class for everything; the hardest part is often finding the class that is needed. To help programmers find what they are looking for, Oracle provide online documentation which can be found at the following web address: *http://docs.oracle.com/javase/7/docs/api/*

Using the Java API

Although it was not mentioned in the former chapters, the prior exercises made use of pre-written classes from the Java library; for example, the [`equals()`], [`toUpperCase()`], [`toLowerCase()`] and [`length()`] methods, as demonstrated in Chapter 2, all belong to the `String` class – the same way that the [`parseInt()`] and [`toString()`] methods (also used in Chapter 2) belong to the `Integer` class. These methods are like recipes out of a cook book; there is no need to 'reinvent the wheel' if it already exists. There is no logical reason for a programmer to physically write the code to turn a word to upper-case letters, when the [`toUpperCase()`] method already exists and is ready to be used. Both the `String` and `Integer` class belong to the `java.lang` package; in Java, a package is the name that is given to a collection of classes. Packages help to organise the Java library by grouping related classes together.

Static Methods

When using a predefined method from the Java API, a programmer needs to understand how the method is implemented; more specifically, the programmer needs to know whether it is static or non-static. This can be determined by simply looking up the chosen class using the online Java documentation. The difference between static and non-static is that a static method can be called directly from a class without the need to instantiate it first. The [`round()`] method is an example of a static method, and thus can be used directly from the uninstantiated `Math` class, to round a value to the nearest whole number:

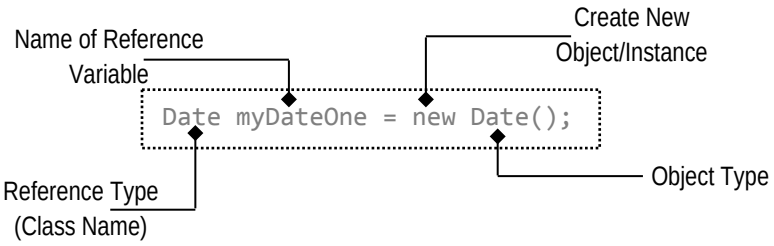
The Math class

```
double x = Math.round(34.54456456);
System.out.println(x);
```

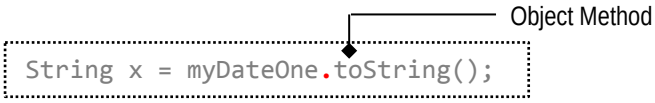
```
C:\Windows\system32\cmd.exe
35.0
Press any key to continue . . .
```


Non-Static Methods

A non-static method requires the class to be instantiated first before it can be used; it needs an instance of the class known as an object. Unfortunately, the whole 'object/class relationship' is a difficult 'chalice to grasp', especially for beginners, but it is something that will become clearer through experience. For now, however, think of a class as a blueprint, template or even a jelly mould that can be used to create many instances, called objects. Once an object is created in memory it can be manipulated, including the use of its methods.



Once an object has been created (using the [new] keyword), a programmer can refer to the reference variable's name to invoke its contained methods. In Java, the full stop (.) is used to refer to an object's method (or property – this is discussed in Chapter 7):



Programming Tips!

During this chapter there is likelihood that the error below may be encountered when working with variables:

```
myOperation might not have been initialized
```

The error is implying that the variable is being referenced to in the program code, but appears not to have a value. This is most probably caused by the assignment of the variable's value occurring inside of either a loop, IF statement, SWITCH statement or TRY/CATCH structure. As these structures are conditional, this means they may or may not execute; which means the variable may or may not have a value.

To resolve the error, simply assign a default value. For example, replace this code:

```
String myOperation;
int x;
```

With this code:

```
String myOperation = null;
int x = 0;
```


4.2 Practical: Using Libraries

Using Methods in a Program

The following example creates two objects (reference variables) from the `Date` class, called `myDateOne` and `myDateTwo`. Note that the keyword [`new`] is used to instantiate each object from the `Date` class. The numbers placed inside of the brackets [`(86,2,28)`] are referred to as 'arguments', and are used to set the date value of the variable – in this instance, to '28 March 1986'. The [`after()`] method, from the `myDateOne` object, is used to compare both dates to determine whether the value of `myDateOne` occurred or occurs after `myDateTwo`. Be aware that to use the [`after()`] method, an object must be created from the class because the method is non-static. The method also returns a boolean value to signify whether the condition is true or false; in the example below, this returned value is stored into the variable `x` and later printed to the command-line.

Activity 4.1

Change the `myDateOne` variable arguments to 84, 0, 29. Check the results in the command-line.

Try the code

```
import java.util.*;
public class UsingMethods{

	public static void main(String[] args){
		Date myDateOne = new Date(86,2,28); //creates object
		Date myDateTwo = new Date(85,1,15);
		boolean x = myDateOne.after(myDateTwo);
		System.out.println(x);
	}
}
```

Step by step

1. Type out the above code
2. Save the program as UsingMethods.java
3. Compile the program
4. Run the UsingMethods.class file
5. Check the results of the program

```
C:\Windows\system32\cmd.exe
true
Press any key to continue . . .
```

Programming Tips!

When using the `Date` class, note that the month value starts at 0 for January, 1 for February and so on!

Understanding the reasons as to why some classes do need to be instantiated and why some classes do not, involves examining Object-Orientated Programming (OOP) in depth. However, this is beyond the scope of this text. For now, remember to use the class name for static methods and create an object, using the [`new`] keyword, for non-static methods. Use the online Java documents to determine whether a method is static or not.

4.3 The Theory: Importing Classes and Readlines

Importing Classes

In the previous program there was this line of code [`import java.util.*;`]. The line of code is used to import the `java.util` package (a collection of classes) that contains the `Date` class referenced by the code; to use a class (and its respective methods) it must first be imported. One may ask, 'Why does the `Date` class have to be imported and the `Math` class does not?' This is because certain classes (classes that are likely to be used, such as the `Math` class) are automatically imported ready for use by a programmer. Another obvious question to ask is 'Why aren't all classes imported ready for use?' There are nearly 4,000 classes available in the Java API, and importing all of them would make programs unnecessarily large; it is much more efficient to only import the classes that are actually used by a program.

Using the Readline Method

For programs to be interactive, programmers need to be able to monitor responses from a user, whether it is a mouse-click, a touch-screen or a key press. Command-line programs can be interactive too by recording the typed response from a user and using selection (i.e. IF statements) to perform chosen tasks. One way to record a response in the command-line is to use the [`readLine()`] method from the `BufferedReader` class. To use the `BufferedReader` class it too must be imported; this can be achieved by using the following line of code at the top of a program [`import java.io.*;`].

The example below uses the [`readLine()`] method to collect a string answer from a user, before printing it to the command-line. The most interesting line of code in the program is [`BufferedReader x = new BufferedReader(new InputStreamReader(System.in));`]; to some it may look complex, but in actuality it is quite simple. The [`System.in`] refers to the standard input, the command-line. The [`InputStreamReader`] is an object that creates a 'stream' to read from a specified input, in this case the command-line. The [`BufferedReader`] is an object that allows a programmer to easily work with the stream reader by providing a set of tools, tools such as [`readLine()`] which allows an entire line to be read – something that is much quicker than reading character by character using the stream reader. In short, the whole line creates an object ready to read text from the command-line and return the entered value as a simple string. The [`readLine()`] method itself is used to wait for and assign the response from a user to a variable.

Try the code

```
import java.io.*;
public class UsingReadlines{

       public static void main(String[] args){
           String myText = null;
           System.out.println("Please enter a word:");
           BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
           myText = x.readLine();
           System.out.println(myText);
       }
}
```


The Big Problem

The example previously demonstrates how to use the [readLine()] method to record a response from a user. There is just one problem: the code will not compile; it will return an error stating that there is an 'unreported exception, and that it must be caught!

```
error: unreported exception
```

TRY/CATCH

Making programs as robust as possible is important in software development; nobody wants a reputation for writing buggy software! Applications should be bug-free and also flexible enough to handle any unexpected exceptions that may occur. There is nothing more frustrating to a software user than an application that continuously 'crashes'.

In Java, 'TRY/CATCH' is a programming technique that can be used to handle any exceptions that may arise during the execution of an application; handling exceptions will help to prevent annoying run-time errors. For example, this technique can be used to catch exceptions such as string-to-integer conversion-type errors or even file handling errors (if a file cannot be found) – errors which would normally result in an application 'crashing' during run-time.

One aspect that sets Java apart from other programming languages is that it will insist that any code deemed hazardous is appropriately handled; hazardous code must be placed inside of a TRY/CATCH technique, to ensure the application will not crash during run-time. In the earlier example, the compiler did not like the use of the [readline()] method – it was 'risky' because it was not handled; the user could return anything or nothing, both of which can cause an error if this is unexpected by a program. The amended code is shown below. The code now includes a TRY/CATCH technique so that the program can be compiled:

Programming Tips!

Using TRY/CATCH – the code to be 'tried' follows the [try] keyword and is typed between the first set of curly brackets [{}]. Any code that is to be executed in the event of an error, follows the [catch] keyword and is typed between the second set of curly brackets. The [IOException e] is an object that contains information about an IO (Input/Output) error should it occur; the object can then be used to print details about the error to the user.

Try the code

```
import java.io.*;
public class ReadlinesWithTryCatch{

       public static void main(String[] args){
           String myText = null;
           System.out.println("Please enter a word:");
           BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
           try{
                myText = x.readLine();
           }catch(IOException e){
               System.out.println("Error: " + e);
           }
           System.out.println(myText);
       }
}
```

Activity 4.2

Type out and compile the amended program code. Enter a range of values and check the results that are printed to the command-line.

Putting it all Together

The program below combines previous techniques learnt from the former chapters, to produce an interactive program. Note that the [readLine()] method is used to gain a response from the user and store it inside of the myGuess variable. An IF statement is then used to test the content of the variable to determine whether the guess is correct or not. The 'OR' [||] operator is used to test the variable for two conditions: lower-case and upper-case.

Activity 4.3

The Quiz – write a 5 question, multiple-choice quiz. The program should inform the user if each guess is correct or not. The code shown below will help.

Try the code

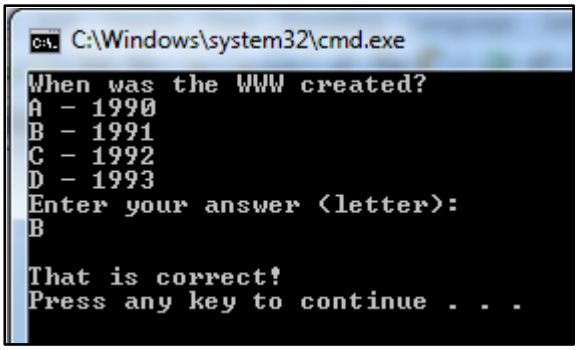
```
import java.io.*; //import the BufferedReader class
public class TheQuiz{

      public static void main(String[] args){
          String myGuess = null; //my variables
          BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
          System.out.println("When was the WWW created?"); //my question
          System.out.println("A - 1990");                  //my choices
          System.out.println("B - 1991");
          System.out.println("C - 1992");
          System.out.println("D - 1993");
          System.out.println("Enter your answer (letter):");
          try{
               myGuess = x.readLine();//read and place value into myGuess variable
          }catch(IOException e){
              System.out.println("Error: " + e);
          }
          if(myGuess.equals("A") || myGuess.equals("a")){ //selection to identify
                System.out.println("");                   //variable value
                System.out.println("That is incorrect!");
          } else if(myGuess.equals("B") || myGuess.equals("b")){
                System.out.println("");
                System.out.println("That is correct!");
          } else if(myGuess.equals("C") || myGuess.equals("d")){
                System.out.println("");
                System.out.println("That is incorrect!");
          } else if(myGuess.equals("D") || myGuess.equals("d")){
                System.out.println("");
                System.out.println("That is incorrect!");
          } else {System.out.println("Error; Invalid Answer!");}
      }
}
```

Activity 4.4

Add a points variable to the quiz program. The points variable should be used to add up the user's score, before printing the total to the command-line at the end of the game. The line of code below can be used to increment the points variable if a guess is correct:

```
points = points + 1;
```



The Big Two

The example below records two inputs from the user; this time, however, the program stores the recorded values into two separate variables, as opposed to reusing the same variable. The program then prints the value of both variables to the command-line. Notice that concatenation is used to add a space between both variables [`+ " " +`], as they are printed to the command-line.

Try the code

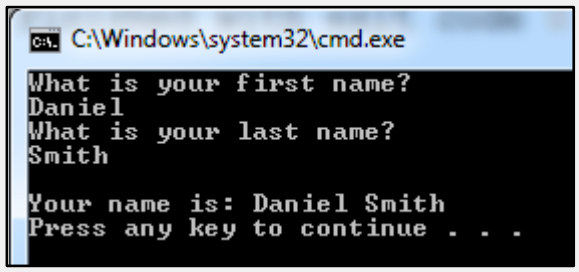
```
import java.io.*;
public class TheBigTwo{

	public static void main(String[] args){
	    String firstName = null;
	    String lastName = null;
	    BufferedReader x = new BufferedReader(new InputStreamReader(System.in));

	    System.out.println("What is your first name?");
	    try{
	         firstName = x.readLine();
	    }catch(IOException e){
	        System.out.println("Error: " + e);
	    }
	    System.out.println("What is your last name?");
	    try{
	         lastName = x.readLine();
	    }catch(IOException e){
	        System.out.println("Error: " + e);
	    }
	    System.out.println("");
	    System.out.println("Your name is: " + firstName + " " + lastName);
	}
}
```

Step by step

1. Type out the above code
2. Save the program as TheBigTwo.java
3. Compile the program
4. Run the TheBigTwo.class file
5. Check the results of the program



Activity 4.5

The Password App – write a program that has two variables, `username` and `password`, with the following values respectively, 'admin' and 'letmein'.

The program should record a username and password attempt from a user and store them into the `usernameGuess` and `passwordGuess` variables. If the username and password guesses match, the program should print 'Access Granted' to the command-line. Else the program should print 'Access Denied.'

Programming Tips!

The [`trim()`] method can be used to remove any added white space in a string; for example, " hello " to "hello".

```
myText = myText.trim();
```


Readlines and Conversion

The data type returned by the [`readLine()`] method is string. However, conversion techniques (as discussed in Chapter 2) can be used to convert the returned data type to more useful ones, including integer and double. The code below reads the value from a user and converts it to an integer value, ready for use in a calculation.

Try the code

```
import java.io.*;
public class TheConversion{

      public static void main(String[] args){
          String myNumber = null;
          int newNumber;
          BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
          System.out.println("Enter a number:");
          try{
               myNumber = x.readLine();
          }catch(IOException e){
              System.out.println("Error: " + e);
          }
          newNumber = Integer.parseInt(myNumber);
          System.out.println(newNumber + " + 10 = " + (newNumber + 10));
      }
}
```

```
C:\Windows\system32\cmd.exe
Enter a number:
50
50 + 10 = 60
Press any key to continue . . .
```

The String Error

There is a problem with this program; if the user enters a string value (text) it crashes. This is because the program is trying to convert the string to an integer and it cannot do it. For now, just enter numeric values; however, techniques to avoid this are demonstrated in Chapter 7.

```
Exception in thread "main"
```

There is also a problem when using division with integers; if an answer has decimal places, these are discarded and the number is rounded to the nearest whole. This, however, can be resolved by using `double` as opposed to `int`. This also means that the input should be converted to double too and therefore this statement should be used:

```
myNumber = Double.parseDouble(input);
```

Instead of:

```
myNumber = Integer.parseInt(input);
```

Activity 4.6

The Driving Licence – write a program that asks the user two questions: 'How long have you been driving?' and 'How many points do you have?'

The program should determine the licence status, before outputting the result to the command-line. The status should be based on:

- 2 years or less and 6 or more points will result in a disqualification
- More than 2 years and 12 or more points will result in a time-ban
- Any other combinations will result in the licence status being safe

Note: all the skills needed to complete this activity have been learnt in the previous chapters; now apply those skills to this new scenario!

Activity 4.7 – The Challenge

The Calculator – write a calculator program that asks the user for two whole numbers. The program should then ask the user for an operation to perform: addition, subtraction, multiplication or division.

The program should then perform the chosen operation on the two given numbers, and output the result to the command-line. The result should show both the answer and the operation performed, for example '2 + 2 = 4'.

