**CMT205 Object Oriented Development with Java**

Lab Exercises Week 2

**Learning Outcomes**

After completing these exercises, you should be able to . . .

* initialise, declare and iterate over arrays
* access command line arguments
* write a basic class
* use the Java API Documentation to find appropriate classes and methods
* write overloaded methods
* write exception handlers
* carry out some simple debugging

**Source Code**

The source code for this worksheet is available on Learning Central at:

CMT205 🡪 Learning Materials 🡪 Week 2

**Arrays and Strings**

1. Two ways to declare and initialise an array are. . .

**int** [] nums = {3 ,5 ,7 ,9 ,10};

. . . and. . .

**int**[] nums = **new** **int**[5];

nums[0] = 3;

nums[1] = 5;

nums[2] = 7;

nums[3] = 9;

nums[4] = 10;

Write a program that creates an array containing the above values and then uses a for loop to calculate and print out their sum.

**NOTE**: When writing a loop that will iterate over the elements of an array use the expression **myArray.length** to get the number of elements in the array.

1. Download the source file CountNames.java. The file declares and initialises an array containing the list of actors’ names.
2. Modify the application so that it counts the number of actors whose names end in **Jones**.

**HINT**: Use the Java API Documentation to investigate the **endsWith** method in the **String** class.

1. One of the names has been written in UPPER CASE and might not be counted in your answer to 2a. Modify your application so that it is able to count all instances of Jones regardless of whether they’re written in upper case, lower case or mixed case.

**Command line arguments**

1. Every Java application includes a method:

**public** **static** **void** main ( String [] args )

The parameter **args** is an array of **String** objects containing the command line arguments passed to the method/application by the Operating System. For example, given a Java application **Foo**, executing the following at the command line:

% java Foo bar1 bar2 bar3

will pass the arguments **bar1**, **bar2** and **bar3** to the main method as an array of three elements names **args**.

Write an application **ShowArgs** that outputs (to standard outptut):

1. The number of command line arguments provided.
2. The contents of the arrays args (use an appropriate for loop).

**HINT**: To test the application you will have to pass it some command line arguments.

What happens when you surround a few command line arguments in quotation marks? Try executing the following:

% java ShowArgs " bar 1 bar2 " bar3

% java ShowArgs " bar 1 bar2 bar3 "

1. Arrays in Java support the for-each loop for iterating over elements without using an explicit index (see slides from session 1). The following is an example of a for-each loop that prints out each element of an array of **Strings**:

String [] arr = {" apple ", " pear ", " orange "};

**for** ( String fruit : arr ) {

System .***out*** . println ( fruit );

}

Update your program from Q3 to use a for-each loop to print each command line argument.

**Writing classes**

1. Download the file **BankAccount.java**. Complete the declarations of the data fields (instance variables) by adding the appropriate types. Add:
2. A constructor that takes two arguments specifying the holder’s name and account number.
3. Appropriate accessor and mutator methods (also called setters and getters). For example, for the **name** field you should create an accessor method called **getName**() and a mutator method called **setName**( String newName ).
4. Methods to deposit to, withdraw from, and check the balance of the account.

Test the class by writing a separate application **BankTest** that creates an account with the name *M J W Morgan* and number *0012067* and calls each method (output the state of the object to standard output after each call - hint: override toString() method).

1. Modify your **BankTest** application so that the name and bank account number are taken from two command line arguments. For example:

% java BankTest "W White " 0033548

should create an account with name *W White* and number *0033548*.

1. Modify your answer to Q5 so that the application takes an optional third argument specifying the initial balance of the account. For example:

% java BankTest "W White " 0033548 300.00

will create an account with an initial deposit of £300.00.

**Java API Documentation**

1. The Java API Documentation documents the classes and packages available in the Java Core API. By using the Java API Documentation to identify relevant fields and/or methods in the class **java.lang.Math**, write an application that outputs:
2. The sine of 15.
3. 632.5.
4. The square root of 2498.
5. A random number greater or equal to 0 and less than 1.
6. *π*.

**Method overloading**

1. Given a class containing a method:

**public** **static** **void** foobar ( **float** a, Date b )

which of the following overloaded methods are invalid for the class:

(a) **public** **static** **void** foobar ( Date a, **float** b )

(b) **public** **static** **void** foobar ( **float** a, String b )

(c) **public** **static** **int** foobar ( **float** a, Date b )

(d) **public** **static** **int** foobar ( **float** a, Date b, **int** c )

(e) **public** **static** **void** foobar ( **float** x, Date y )

(f) **public** **void** foobar ( **float** x, Date y )

Verify your answers by implementing each method in the same class.

**Exception handling**

1. One or more statements in a **try** block may cause an exception. In many cases your response to an exception will depend on the cause of the exception. For example, we would wish to deal with an **IllegalArgumentException** differently to a **ClassNotFoundException**. In Java we write a handler for a particular exception using a **catch** block. We can even attach multiple exception handlers to the same **try**, as shown in the following example:

**try** {

// statements here

}

**catch** ( SomeException ex ) {

// statements to handle SomeException here

}

**catch** ( AnotherException ex ) {

// statements to handle AnotherException here

}

**catch** ( YetAnotherException ex ) {

// statements to handle YetAnotherException here

}

Here we have attached three exception handlers to the **try** block. If at some point in the **try** block an exception is thrown, the **catch** block matching that exception will be executed. Download **ReadIntExceptions.java**. This application attempts to read an integer from a file named **an\_integer.txt**. There are a few exceptions that may occur during this process and our application should attempt to handle each of them.

1. Run **ReadIntExceptions** to verify that it works correctly.
2. Try renaming the file **an\_integer.txt** to **oops.txt**. Run **ReadIntExceptions** again and see what happens.
3. Change the files name back to **an\_integer.txt**. Replace the text in the file with 1234five6789. What exception occurs when you now run **ReadIntExceptions**?

Modify **ReadIntExceptions** to add another **catch** block that handles this exception and prints out an appropriate message. Compile and run the application again to verify that it handles this exception correctly.

HINT: Just like any other class, exceptions need to be imported. You should check which package the exception belongs to and import it for use in your application.

**Debugging**

1. Enter the following code:

**public** **class** HelloWorldError

**public** **static** **void** main ( String [] args )

{

System . out. println ( Hello World );

}

}

and save it in a file named **HelloWorldError.java**. Before executing the application it needs to be compiled. Open the command line (**cmd.exe** on Windows) and navigate to the directory where you saved **HelloWorldError.java** by using the command **cd** (change directory). Attempt to compile the application by typing:

> javac HelloWorldError . java

The application is supposed to output the text "Hello World" to **standard output**. However, there are three syntax errors that prevent it from compiling. **javac** will print a list of compilation errors to the command line to help you identify where the problems are. Fix the errors to allow the application to be successfully compiled.

1. Once you manage to successfully compile the application, a **HelloWorldError.class** file will be created in the same directory as **HelloWorldError.java**. This **Java class file** contains the **Java bytecode** for the application. You can now execute the application by typing:

> java HelloWorldError