Creating Packages and Classes

At the start of the course, we wrote a first program which we called "`Hello World`". By convention, Java packages should be named using the company's or programmer's domain name in reverse order. If we were working at a company that has the domain name "`example.com`", we might name a package "`com.example.chris.fundamentals`".

For your "`Hello World`" program, you were instructed to create a package and, within that package, define a class named `HelloWorld`. Remember that class names in Java should follow Pascal case (also known as upper camel case), which means each word within the name starts with an uppercase letter and the class name contains no spaces. Additionally, Java requires that any public class be saved in a file with the same name as the class, followed by the "`.java`" extension. Therefore, the `HelloWorld` class must be stored in a file named "`HelloWorld.java`".

Example code that satisfies the requirements of the "`Hello World`" program specifications is shown below:

Code Block 1: *HelloWorld class*

```
1 package com.nielsenedu.chris.helloworld;
2 public class HelloWorld {
3    public static void main(String[] args) {
4       System.out.println("Hello World!");
5    }l
6 }
```

Going forward, it is recommended that, for each new assignment in this course, you create a new package with a name that describes the assignment. Then create class(es) within that package, naming each class with a name that describes the functionality of that class.

1. In the box immediately below, write a package name that would be appropriate for your hello world program, following the package name in the code box above.

```
com.bjfles.name.helloworld
```

The Method Header

The ***method header*** is the first line of a ***method declaration***. Below is an example of a method header.

Code Block 2: *Example method declaration*

```
public static boolean isPrimeNumber(int num)
```

The *method header* includes the ***modifiers***, the ***return type***, and the ***method signature*** of the method. For this worksheet, and until we learn about the modifiers, all the methods we will write will have use modifier `public` and the modifier `static`. So when you are asked to write a method, start it by writing "`public static`".

In the *method header* in *Code Block 2*, above, immediately after "`public static`" comes the ***return type***, which in this case is type `boolean`. Compare this to the `main` method *method header* in line `3` of *Code Block 1*, where the return type is `void`. If a Java method does not return any value, we show this by setting `void` as the return type.

Java does not allow a method to return multiple values – only a single value may be returned from a method. The return type of a method may be:

- `void`,
- a ***primitive type*** such as `boolean`, `int`, `double`, `char`, etc.,
- a ***reference type*** (an *object*), such as `String`.

After the return type, the remainder of the *method header* is called the ***method signature***. It consists of the name of the method as well as the ***parameters*** that are required to be passed to the method when the method is called. The *parameters* are given in parentheses and requires both the *type* of the parameter and an *identifier*. The main method always has one parameter of type `String[]`. The *method header* in *Code Block 2* has a single parameter named `num` of type `int`. If there is more than one parameter, each parameter *type-identifier* pair in the list of parameters is separated by a comma. *Code Block 3*, below, gives an example of a method with three parameters, two of type `int` and one of type `double`.

Code Block 3: *Example method declaration with multiple parameters*

```
public static double vectorAngle(int x, int y, double factor)
```

Use the code below to answer the question that follows.

Code Block 4: *Correct method declarations*

```
a public static boolean isEven(int num)
b public static String getLine()
c public static double pow(double x, double y)
d public static int compare(String s1, String s2)
e public static double random()
f public static String subString(String s, int start, int end)
```

2. Fill in the table below with the appropriate values based on the code given in *Code Block 4*.

	method name	return type	Number of Parameters
a)	`isEven`	`boolean`	`1`
b)	`getLine`	`String`	`0`
c)	`pow`	`double`	`2`

	method name	return type	Number of Parameters
d)	`compare`	`int`	`2`
e)	`random`	`double`	`0`
f)	`subString`	`String`	`3`

3. On the lines provided, clearly explain what error(s) are present in each *method header*.

a) `public static void boolean isFactor(int multiple, int factor)`

There are two return values specified, `void` and `boolean`. A method may only return a single value.

b) `public static int area(int length, width)`

Each parameter to a method must have a type. Although `length` is specified to be of type `int`, `width` does not have a type specified, which is an error.

c) `public static printManyTimes(String s, int times)`

A method requires a return type, or if the method does not return a value, the return type must be given as `void`.

d) `public static int(boolean b, double d, String s)`

There is no method name given; `int` is a data type, so it cannot be the method name.

e) `public static int double(int n)`

The name of a method cannot be a reserved word. The word `double` refers to a primitive type, so cannot be used as a method name.

f) `public static int compare(String s, String s)`

An identifier can only refer to one thing. We cannot have two parameters, both named `s`, because how will we distinguish which one we are referring to?

4. For each part of question (3), rewrite the method header below, correcting the error, while keeping as much of the original method header as possible.

a)
```
public static boolean isFactor(int multiple, int factor)
```

b)
```
public static int area(int length, int width)
```

c)
```
public static void printManyTimes(String s, int times)
```

d)
```
public static int myMethod(boolean b, double d String s)
```

e)
```
public static int timesTwo(int n)
```

f)
```
public static int compare(String s1, String s2)
```

Writing Methods

Examine the code in *Code Block 5*, and compare it to the program in *Code Block 1*.

Code Block 5: *HelloWorld2 Class*

```
1  package com.nielsenedu.chris.helloworld;
2  public class HelloWorld2 {
3     public static void main(String[] args) {
4        System.out.println("Hello World!");
5     }
6     public static void sayGoodbye() {
7       System.out.println("See you later!");
8     }
9  }
```

The code defined for the class named `HelloWorld2` starts with the opening curly brace on line `2` and ends with the corresponding closing curly brace on line `9`. Within these curly braces, there are two methods defined: a `main` method, and a method named `sayGoodbye`. The code for each method is enclosed within curly braces. The opening curly brace for a method immediately follows the *method header*. Each method in the example contains a single ***statement*** inside of it (enclosed within the curly braces). When you read example code, take particular note to how the ***indentation*** of code improves the readability, and try to learn the proper indentation of code.

Each method in Java must be declared within a class, and a method cannot be declared within another method. In *Code Block 5*, note how the method header for both the `main` method and the `sayGoodbye` method are at the "same level" within the `HelloWorld2` class, and that the `sayGoodbye` method header is not within the `main` method.

When a Java program is run by the Java Virtual Machine (JVM), the JVM looks for the `main` method, and if it is found, the JVM executes the code within it. The program as written in *Code Block 5* will only execute the code within the `main` method and will not execute the code within the `sayGoodbye` method. In order to execute the code within the `sayGoodbye` method, the `main` method must ***call*** that method. This is the topic of the next section.

Calling Methods

"Calling a method" means telling the program to execute the instructions inside the method; invoking the method using the method's name, followed by parentheses containing the parameters that are required to be passed to the method, as they were defined in the ***method signature***. Examine the method headers given in *Code Block 6*, below.

Code Block 6: *Examples of correct method headers*

```
1 public static void printManyTimes(String s, int times)
2 public static double circumference(double radius)
3 public static String getLine()
```

The first method on line `1` of *Code Block 6* has no return type. To call a method has no return type, the method name is written, followed by parameter values of the types required by the *method signature*. For methods that do have a return value, the returned value is often assigned to a variable of the appropriate type. *Code Block 7* shows an example of how each of the methods from *Code Block 6* could be called. Note each statement in *Code Block 7* is terminated with a semicolon.

Code Block 7: *Calling the methods from Code Block 6*

```
1 printManyTimes("Be quiet!", 3);
2 double rad = 4.0;
3 double c = circumference(rad);
4 String inputLine;
5 inputLine = getLine();
```

As the method named `printManyTimes` has no return value, the call to the method is a statement on its own, and the method does not produce any value that can be used or assigned to any variable. Method `printManyTimes` has two parameters. In this case, the `String` parameter requirement is satisfied with the string literal `"Be quiet!"`, while the `int` parameter requirement is satisfied with the integer literal `3`.

The method named `circumference` requires a parameter of type `double`. In this case, a variable named `rad`, that was previously declared as type `double`, is passed to the method. Notice that the name of the variable passed to the method (`rad`) does not have to be the same as the name of the parameter name of the method (`radius`). The value stored in the variable `rad` is copied into the parameter `radius` when the method is called.

Method `circumference` has a return value of type `double`. Line `3` of *Code Block 7*, in a single statement, defines a variable `c` of type `double`, calls method `circumference` while passing the required parameters, and assigns the return value of that method to the variable `c`.

The `getLine` method requires no parameters. To call a method that requires no parameters, a set of empty parentheses must follow the method name. In line 4 of *Code Block 7*, the method `getLine` is called, and the previously declared variable of type `String` named `inputLine` is set to the return value of the method.

5. For each method header, write a statement that declares a variable of an appropriate type, calls the method defined by the method header, and correctly and assigns the return value of the method to the variable that was declared. Terminate statements with a semicolon. For `void` methods, call the method without any variable declaration or assignment. If any variable is used to satisfy a parameter requirement, that variable must be declared as the appropriate type.

a) `public static int compare(String s1, String s2)`

```
int i = compare("Hello", "hello");
```

b) `public static double random()`

```
double d = random();
```

c) `public static int length(String s)`

```
int len = length("Hello");
```

d) `public static String subString(String s, int start, int end)`

```
String s = subString("Hello", 3, 5);
```

e) `public static double random()`

```
double d = random();
```

f) `public static void setTruth(boolean b)`

```
setTruth(false);
```

g) `public static double length(double x, double y)`

```
double len = length(3.0, 5.0);
```

h) `public static double fill(int x, int y, int color, double opacity)`

```
double d = fill(-1, -2, 1, 0.9);
```

Writing Methods

In the declaration of a method in Java, *method header* is followed by the ***method body***, which is enclosed within curly braces. The method body is the block of code that executes when the method is called.

6. In *Code Block 5*, the code from method `sayGoodbye` will not be executed when the program is run. In the box below, re-write the entire class `HelloWorld2`, changing it such that method `sayGoodbye` is called after the `println` statement in the `main` method. You do not need to copy out the `package` information.

```
1 public class HelloWorld2 {
2    public static void main(String[] args) {
3       System.out.print("Hello World!");
4       sayGoodbye();
5    }
6    public static void sayGoodbye() {
7       System.out.println("See you later!");
8    }
9 }
```

7. In the box below, write a class named `PrintTwice` that includes a `main` method and a method called `printTwice`. The `printTwice` method must return a `boolean` value, which is always set to `true`. It must take a `String` parameter, and use `System.out.println` to print the contents of that parameter to the console twice, each time on a separate line. The `main` method must call `printTwice`, and store the return value in a local variable. The `main` method must then print that `boolean` value to the console.

```
 1 public class PrintTwice {
 2    public static void main(String[] args) {
 3       boolean b;
 4       b = printTwice("Repeat this.");
 5       System.out.println(b);
 6    }
 7    public static boolean printTwice(String s) {
 8       System.out.println(s);
 9       System.out.println(s);
10       return true;
11    }
12 }
```