Java Refresher

Module 2 - Using classes and objects

After completing this module, you will have written basic programs that create objects from classes and use methods of those objects. You will also have written programs that use directly use methods of the class itself, or so-called "static" methods. This module will also give you experience in reading Java documentation and learning how to use classes and objects based on that documentation.

Background Material

Java provides you with an enormous library of classes that you may use in your own programs. These classes perform various functions ranging from creating windows and buttons in a graphical user interface, to managing network connections, to managing data structures. Given the plethora of classes available, it is critical to understand how to read Java documentation for any particular class.

The standard format for Java documentation is called *Javadoc*. You can look up the Javadoc for any class in the standard library at the official website here.

In this series of modules, we will also provide you with some classes of our own which you will need to use, the most important of which is the `Input` class, whose Javadoc is below:

Javadoc

Class Input

The Input class provides a set of static methods for easily asking the user questions and reading their answers.

Method Summary

Modifier and Type	Method and Description
`static boolean`	**`askBoolean`**`(java.lang.String question)` Asks the user the given question, waits for the user to enter a boolean at the keyboard, and then returns this boolean.
`static char`	**`askChar`**`(java.lang.String question)` Asks the user the given question, waits for the user to enter a single character at the keyboard, and then returns this character.
`static double`	**`askDouble`**`(java.lang.String question)` Asks the user the given question, waits for the user to enter a double at the keyboard, and then returns this double.
`static int`	**`askInt`**`(java.lang.String question)` Asks the user the given question, waits for the user to enter a integer at the keyboard, and then returns this integer.
`static java.lang.String`	**`askString`**`(java.lang.String question)` Asks the user the given question, waits for the user to enter a string at the keyboard, and then returns this string.

As you can read from the above documentation, this class provides useful methods for reading input from the user. Notice that each of the provided methods are `static` which is quite unusual when compared to most Java classes. What this means is that it is not necessary to create a new instance of the class. Rather, you can invoke these methods directly on the class itself. Let's read the documentation for the `askInt` method. This method will ask the user a question, wait for the user to type an answer, and will then return the answer. Notice also that the return type is an integer as indicated in the first column of the table. Also

note that the question must be supplied as a parameter, and it's type must be a String. To use this method, we can write code such as that below:

```
int number = Input.askInt("What is your favourite number?");
```

Because the method is static, we have invoked the method directly on the class by writing the class name `Input` before the dot. This is easily recognisable to anyone reading this code, because class names always begin with an uppercase letter. If this were not a static method, we would need to have first created an object and then put the name of that object before the dot. This would also be easily recognisable by the reader, since object names always begin with a lowercase letter. Next, notice that we have passed a string question as the parameter. Finally, notice that we have stored the integer result into a variable of the same type, i.e. `int`. When calling a method that returns something, it is a good general rule of them to always store that result into a variable. There are other ways to deal with a return value (including even "ignoring" it in some rare cases), but usually we want to save the result so that we can refer to it in a later step in the program, such as:

```
System.out.println("You entered " + number);
if (number == 7) {
  System.out.println("That is my favourite number too!");
}
else {
  System.out.println(number + " is an okay number, but not as good as 7.");
}
```

Another way to use the return value of a method (which is often not a good idea) is to directly embed the method call where you want to use that value:

```
System.out.println("Your favourite number is " + Input.askInt("What is your favourite number?"));
```

This will work, but note that the return value can only be used once, and is promptly forgotten. If you want to remember the number because you want to refer to it again in the future, you need to remember it by storing it into a variable.

Creating an instance and using instance methods

Now let's take a look at the documentation for a class that has instance methods rather than static methods:

Javadoc

Class Rectangle

Constructor Summary

Constructor and Description
`Rectangle(int width, int height)` Creates a new Rectangle with the given width and height.

Method Summary

Modifier and Type	Method and Description
`void`	**`showDimensions()`** Prints the dimensions of this rectangle to the terminal.
`void`	**`showArea()`** Prints the area of this circle to the terminal.

Notice that unlike in the case of the Input class, none of the methods above are declared static which means they should be assumed to be instance methods. While static methods are invoked directly on the class, instance methods must be invoked on a particular instance of the class. The code below shows how to create two instances of the Rectangle class:

```
Rectangle rect1 = new Rectangle(10, 5);
Rectangle rect2 = new Rectangle(3, 8);
```

The two objects `rect1` and `rect2` were created from the `Rectangle` class and are therefore *instances* of the `Rectangle` class. Notice that we name these objects beginning with a lowercase letter in accordance with Java naming conventions.

To invoke one of the instance methods such as `showDimensions`, you cannot simply write `Rectangle.showDimensions();` because how will it know which rectangle to show? Instance methods only work in reference to a particular instance and we must mention the name of the instance before the dot. For example:

```
rect1.showDimensions();
```

will show the dimensions of the first rectangle which are 10x5.

If you would like to read more information, refer to the section on objects in the Java tutorials (this section is 3 pages).

Exercises

Each exercise below will involve writing a short program with the following structure:

```
public class ExerciseN
{
  public static void main(String[] args)
  {
    YOUR LINES OF CODE GO HERE
  }
}
```

In the skeleton code for this module, you will find 9 ready-made skeleton classes like this, named Exercise1, Exercise2, ... up to Exercise9. Your solution to each exercise should be written inside the corresponding ready-made class.

Exercise 1.

Learning objective: Learn how to use static methods by reading documentation.

Class to edit: `Exercise1`

When attempting this exercise, you should refer to the Java documentation for class `Input`, which was presented in the Background Material section above.

Write a program that asks the user to pick a card from a standard playing deck and then prints out the picked card. Your program should make use of the `Input` class above and should behave according to the following sample I/O:

```
Pick a number (1-13):  « user inputs 7 »
Pick a suit:           « user inputs hearts »
You picked the 7 of hearts
```

Now, add an if/else statement to your program to check if the user inputs a number greater than 13. In this case, your program should instead behave according to the following sample I/O:

```
Pick a number (1-13):  « user inputs 15 »
Pick a suit:           « user inputs spades »
The number you picked is too high
```

This is the only special case you should handle for this exercise.

Hint: read the number using `askInteger` and read the suit using `askString`. Both are static methods of the `Input` class documented in the Background Material section above.

After compiling, running and testing your program, export your project to a jar file being sure to "Include the source files", and then submit this to PLATE to confirm whether or not your solution is correct. If there are no compile errors, PLATE will either give you a PASS, or will otherwise show your program's output alongside the expected benchmark output highlighting any differences.

If your program is incorrect, you may have unlimited attempts to edit your code, re-export to a jar file and resubmit.

Passed? Excellent! As you progress through each exercise below, be sure to continually submit to PLATE to confirm whether your solution is correct. If your solution is not correct, let PLATE's feedback report inform you on what you need to fix.

Exercise 2.

Learning objective: Learn how to create an object with parameters, and use its methods, by reading documentation.

Class to edit: `Exercise2`

When attempting this exercise, you should refer to the following Java documentation for class `Circle`:

Javadoc

Class Circle

Constructor Summary

Constructor and Description
`Circle(double radius)` Creates a new Circle with the given radius.

Method Summary

Modifier and Type	Method and Description
`double`	**`calculateArea()`** Calculates and returns the area of this circle.
`void`	**`showArea()`** Prints the area of this circle to the terminal.

Write a program that creates a new circle of radius 2.0 and then shows the area of that circle. Your solution should be two lines of code:

1. Use the constructor to create a circle with radius 2.0 as the parameter, and store this new circle into a variable. The name of this variable will be used as the name of the object in the next step.

2. Use the `showArea` method to show the circle's area. Since this is a non-static method, you should put the name of the object before the dot, rather than the name of the class (see the discussion of what comes before the dot in Background Material).

Hint: For an example of creating and using an object, refer to the `Rectangle` example at the end of the Background Material section.

After compiling and testing, export and submit your project to PLATE to confirm that the output of your program is correct.

Exercise 3.

Learning objective: Learn how to create an object using the contents of a variable as the constructor argument.

Class to edit: `Exercise3`

Write a program that asks the user "What is the circle radius?", then creates a new circle with that radius and shows its area. The program should behave according to the following sample I/O:

```
What is the circle radius?   « user inputs 3.7 »
The area of the circle is 43.00840342764427
```

Your program should be 3 lines of code:

1. Ask the user for the radius.
2. Create a new circle with that radius.
3. Show the area of that circle using its `showArea` method.

Exercise 4.

Learning objective: Make a decision based on the return value of a method

Class to edit: `Exercise4`

Write a program that asks the user "What is the circle radius?", then creates a new circle with that radius. If the area of the circle is >= 10, then print the message "This circle is big." otherwise print the message "This circle is small." taking note that the message has a fullstop at the end of the sentence.

The following sample I/O shows how your program should behave for a big circle:

```
What is the circle radius?   « user inputs 3.7 »
This circle is big.
```

And for a small circle:

```
What is the circle radius?   « user inputs 1.2 »
This circle is small.
```

Hint: After creating the circle, you can obtain its area by calling its `calculateArea` method. As explained in the Background Material, if you will only be using the return value of this method once, it is possible to embed the method call directly where you need to use its value. For example: `if (circle.calculateArea() >= 10)`. Try it!

Exercise 5.

Learning objective: Learn how to import a class from the standard Java class library.

Class to edit: `Exercise5`

When attempting this exercise, you should refer to the following Java documentation for class `Random`:

Javadoc

java.util

Class Random

Constructor Summary

Constructor and Description
`Random()` Creates a new random number generator.
`Random(long seed)` Creates a new random number generator using a single `long` seed.

Method Summary

Modifier and Type	Method and Description
`boolean`	**`nextBoolean()`** Returns the next pseudorandom, uniformly distributed `boolean` value from this random number generator's sequence.
`double`	**`nextDouble()`** Returns the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence.
`float`	**`nextFloat()`** Returns the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence.
`double`	**`nextGaussian()`** Returns the next pseudorandom, Gaussian ("normally") distributed `double` value with mean `0.0` and standard deviation `1.0` from this random number generator's sequence.
`int`	**`nextInt()`** Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.
`int`	**`nextInt(int bound)`** Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
`long`	**`nextLong()`** Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.
`void`	**`setSeed(long seed)`** Sets the seed of this random number generator using a single `long` seed.

Write a program that creates a random number generator from class `Random` using the seed 7777, then generates a random integer between 0 (inclusive) and 100 (exclusive). While there are several ways you could generate a number in this range, one of the methods documented above is designed specifically for this purpose, and you should use that method. Note that the Javadoc documentation above indicates that class `Random` is provided by the package `java.util` (see the very first line of the Javadoc). This means that in order to use it, you need to import this class, as shown by the code snippet below:

```
import java.util.Random;  ← Like this
```

```
public class Exercise2 {
  public static void main(String[] args) {
    // INSERT YOUR CODE HERE
  }
}
```

Your program should behave according to the following sample I/O:

```
Here is a random number: xxx
```

where `xxx` should be replaced by the actual random number that your program generated.

Your program should be 3 lines:

1. Create a new random number generator using the constructor that accepts a seed as a parameter. Pass in 7777 as the seed.
2. Call the appropriate method to generate a random integer between 0 (inclusive) to 100 (exclusive) and save it into a variable. Use the most appropriate method; There is only one method designed for this purpose.
3. Print `"Here is a random number: xxx"` where `xxx` is the random number that you generated in the previous step.

Exercise 6.

Learning objective: Take multiple actions based on a decision.

Class to edit: `Exercise6`

When attempting this exercise, you should refer to the following Java documentation for class `BankAccount`:

Javadoc

Class BankAccount

Constructor Summary

Constructor and Description
`BankAccount()` Creates a new bank account, asking the user to enter the account name and initial balance.

Method Summary

Modifier and Type	Method and Description
`void`	**`deposit`**`(double amount)` Deposits the specified amount of money into this account.
`double`	**`getBalance`**`()` Gets the balance of this account.
`void`	**`showBalance`**`()` Shows the balance of this account.
`void`	**`transfer`**`(double amount, BankAccount targetAccount)` Transfers the specified amount of money from this account into the target account.
`void`	**`withdraw`**`(double amount)` Withdraws the specified amount of money into this account.

Write a program that executes the following steps:

1. Create a new bank account.
2. Show the balance of the bank account.
3. If the balance of the bank account is >= 20 dollars, do the following steps:
    1. Print "Purchasing movie ticket." including the full stop at the end of the sentence.
    2. Withdraw $20 from the account.
    3. Show the balance of the bank account again.
4. Otherwise, print "You need more money to buy a movie ticket." including the full stop at the end of the sentence.

The following sample I/O shows how your program should behave when the account has at least $20:

```
Creating a new bank account
Enter account name:         « user inputs Shelly Martin »
Enter initial balance: $    « user inputs 567.00 »
Shelly Martin's account has $567.00
Purchasing movie ticket.
Shelly Martin's account has $547.00
```

Otherwise, your program should behave as follows:

```
Creating a new bank account
Enter account name:         « user inputs Shelly Martin »
Enter initial balance: $    « user inputs 7.00 »
Shelly Martin's account has $7.00
You need more money to buy a movie ticket.
```

Hint: Apply a pair of "`{`" and "`}`" braces around the entire 3 steps of the `if` case, and then a separate pair of braces around the 1 step of the `else` case.

Exercise 7. *(Challenge Question)*

Learning objective: Use an object as a method argument.

Class to edit: `Exercise7`

Write a program that executes the following steps:

1. Create a new bank account in a variable called account1.
2. Create a second bank account in a variable called account2.
3. Show the balance of account1.
4. Show the balance of account2.
5. Transfer 10 dollars from account1 into account2.
6. Show the balance of account1.
7. Show the balance of account2.

Hint: In step 5 of the program, you need to call the `transfer` method. This method takes two parameters, which should be separated by a comma. The format of the line of code would be: `sourceAccount.transfer(10.0, targetAccount);` where `sourceAccount` is where the money is coming from, `10.0` is the amount of money to transfer, and `targetAccount` is where the money is being transferred to. Hence, in order to perform a transfer, you need to have two accounts. Conveniently, you DO have two accounts which you created in steps 1 and 2, called `account1` and `account2`. Think carefully about which one should be used as the source account and which should be used as the target account.