Worksheet 7: Static Classes

Updated: 4th May, 2020

- Understand and Implement Static Classes
- Understand How to Invoke Method Overloading
- Write a generic PDI Math Class.
- Write a generic User Interface.

Note: You must submit this practical worksheet by:

Sunday, 10th May 2020, 5pm (local time)

And have it signed off during your next **registered** practical session.

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** `P07` directory in **`.tar.gz`** format). You must submit everything you have completed in this practical, both in class and at home.

You must also not modify the files in `~/Documents/PDI/P07` until after your submission has been marked. During your practical your tutor will check the modified dates of the files using `ls -l` (el) before marking your work. To create a gzipped tarball use the following command from your **`PDI`** folder:

```
[user@pc]$ tar -cvzf <studentID>_P07.tar.gz P07
```

For more information on what each argument of the above command does use:

```
[user@pc]$ man tar
```

To get your **`.bash_history`** into a submittable format, first close all terminals down, using <ctrl>-d. Then open a new terminal, and type this command from anywhere:

```
[user@pc]$ history >~/Documents/PDI/P07/BashHistoryP07.txt
```

Your submission will be marked during your next **registered** practical session via an online interview (in `Blackboard Collaborate Ultra`) from your tutor, comprised of a few questions.

Please note that the questions asked in the interview may cover the entirety of the worksheet, not just the material in your Assignment Task. Your submitted work will be assessed and marks will be allocated accordingly.

1. What is a Static Class?

Static keyword can be used with class, variable, method and block. In PDI we will be using it with just the variables and methods. Static members belong to the class instead of a specific instance, this means if you make a member static, you can access it without an object. You have been using "Static Classes" already in this unit, any time we created a class that had the purpose for being used once per execution was a Static Class.

Let's look at an example:

```java
public class PIExample
{
    public static void printPI(double pi, int deciPlaces)
    {
        // Calculate the number to multiply by
        int places = (int)Math.pow(10.0, (double)deciPlaces);
        // The Math Class is static, thus we can just reference it with "."
        // Apply the truncation to pi
        double truncPI = ((double)((int)(pi * places))) / places;
        // Print the result
        System.out.println("PI to " + deciPlaces + "dp is: " + truncPI);
    }
}
```

We do not need to create an instance of this class in order to run it. In this same fashion, because we made `printPI()` a `public` method we can access it from outside of the class.

```java
public class OtherPIExample
{
    public static void main(String[] args)
    {
        double someValue;
        someValue = Math.PI;
        PIExample.printPI(someValue, 6);
    }
}
```

Obviously this code is quite pointless and trivial, but it shows how we can break up some of our submodules into other classes and just call them. This way we can start to think about "Class Responsibility" when we design our algorithms. This is especially good if we have parts of our code that relate to certain topics and we can store them within those categories.

For example the Java `Math` class (`java.lang.Math`) does this. It stores all "Math" related submodules/methods in it as static, so that they can be used by anyone to compute its functions.

2. The PDIMath Class

Now it's time to create one of our own classes that we can use and refer to throughout the rest of this unit. We will be using Method Overloading, a concept that allows methods of the same name to be used as long as they have different parameter types.

Methods that you are required to implement in `PDIMath` are:

- `min(X a, X b)` – A method that will take in the values of `a` and `b` and return the lesser of the two values. Note that the `X` will need to be replaced with multiple types: `int, double, long, float`
- `max(X a, X b)` – A method that will take in the values of `a` and `b` and return the greater of the two values. Note that the `X` will need to be replaced with multiple types: `int, double, long, float`
- `abs(X a)` – A method that will return the absolute value of `a`. Note that the `X` will need to be replaced with multiple types: `int, double, long, float`
- `floor(double a)` – A method that will take in the value `a` and return the value that has been rounded to the lesser whole value. e.g., `1.9` will return `1`.
- `ceil(double a)` – A method that will take in the value `a` and return the value that has been rounded to the greater whole value. e.g., `1.1` will return `2`.
- `pow(double base, int exponent)` – A method that will return the value of the first argument raised to the power of the second argument.
- `pi(int precision)` – A method that will take in a precision and calculate `pi` it to its precision. You may use either the method in the lecture slides or the method from P05. Note that P05 method was $\pi/4$.
- `e(int precision)` – A method that will take in a precision and calculate `e` it to its precision. You may use the implementation of this in P05.

Note: You must not use any other implementation these methods in your algorithms (apart from where stated). You are required to think logically about how you can implement each of these methods.

3. Testing PDIMath Class

Using the Java Math class, test to see if your implementations are accurate. You will need to write some driver code to test this (like the test harness last week).

Note: More than likely some of your calculations may be off, but that is because Java has implemented the "best" possible versions of these functions. When you test, in some cases you may need to test with a tolerance.

4. Assignment Task: UserInterface Class

Your last task for this practical is to complete the following task **in your own time**.

Warning: This question goes towards your portfolio/assignment mark and thus any collusion will be dealt with as per university policy.

Create a subdirectory in this week's `P07` directory and call it **`AssignmentTask`** your files should be called **`UserInterface.txt`** and **`UserInterface.java`** respectively.

Next you need to develop **`UserInterface`**, this class should now be the only point of contact in your programs between the user and the program. For example, if you wish to print something out to the user, you will be required to go through the **`UserInterface`** to do it.

You may think that this is pointless and obsolete, but it is very good practice to keep all similar functionality together inside of a class. So that when something goes wrong or an update needs to be made, you know where to go to modify your algorithm.

Methods that you are required to implement in **`UserInterface`**:

- `int userInput(String prompt, int lower, int upper)`
- `double userInput(String prompt, double lower, double upper)`
- `long userInput(String prompt, long lower, long upper)`
- `float userInput(String prompt, float lower, float upper)`
- `char userInput(String prompt, char lower, char upper)`
- `String userInput(String prompt)`
- `void displayError(String error)`

  Yes, this method will be very tedious, but what if we wanted to print all errors to a file instead of to the terminal? Later on, we can just change this method.

You **are** required to do exception handling here, so from now on you will not get any errors/crashes to the screen during any user interactions.

Do not forget to create a test harness (just using some driver code) for this class as well. This will serve as our testing portion of this task.

Note: Driver code refers to code that is written just to "drive" or to test functionality that otherwise couldn't be tested. Your driver code can be as simple as creating a main and testing to see if the methods you wish to test, work.

End of Worksheet