Worksheet 6: Basic Objects

Updated: 27th April, 2020

- Implement the Basic Model Classes concepts
- Understand how a class is dependant on its class fields
- Understand Constructors, Accessors, and Mutators

Note: You must submit this practical worksheet by:

Sunday, 3rd May 2020, 5pm (local time)

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

The workload for this week has been reduced to accomodate for your test and the 1 less day in the week.

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** `P06` 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/P06` 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>_P06.tar.gz P06
```

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/P06/BashHistoryP06.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. Talking About Basic Model Classes

Before we start developing any classes lets talk about Basic Model Classes.

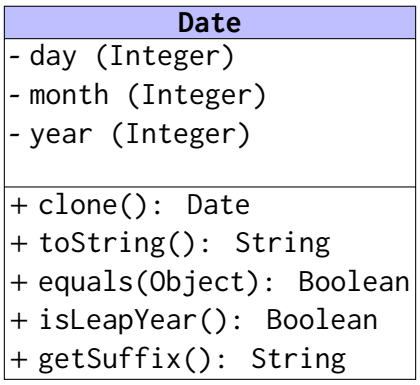
Together as a class, spend 20 minutes discussing these questions. Yes, you can work together for this part of the worksheet. The idea of this exercise is to get you actively thinking about the problems you are about to face. The answers to these questions will also help you design your classes.

Each question also has several sub points that your discussion should address.

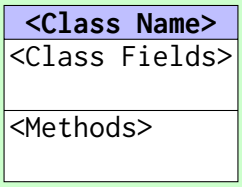
- What is the difference between a class and an object?
- What is the purpose of a constructor?
  - How are they related to class fields?
- Why do we need accessors?
  - What relationship do accessors have with class fields?
  - What class fields would you need for Student class?
- Why do you always write an equals method?
  - What is the parameter an equals method should take?
  - What fields should be compared in an equals method?
  - Is it appropriate for an equals method to FAIL (throw an exception)?
- Why do we need mutators?
  - How are mutators related to class fields?
  - Why do you **not** need a mutator that changes all class fields at once?
- Why might we use submodules for validation?
- Why are class constants declared as public where as declaring class fields as public is very bad (instant zero)?
- Why should model classes not have INPUT and OUTPUT?

2. Date

The first class you should develop is a Date class. The complete class should be implemented as defined in the UML (**U**nified **M**odelling **L**anguage) below (please see the note on interpreting UML). You should maintain the order of the class fields in field declarations, and parameter orders (alternate constructor and setters).



UML:



Notes:

`-` The field/method is **`private`** ie: *cannot* be accessed from other classes.
`+` The field/method is **`public`** ie: can be accessed from other classes.
`<method>(<parameters>):<type>`
`<method>` is the name of the method.
`<parameters>` are the imports of the method (if it has any).
`<type>` is the export datatype of the method (or variable name if they is more meaningful).

Lets think how we can validate each class field in this class:

`day` – How do we calculate the day? First we need the month, we could use a `CASE` statement like we did in P02/3. Then we also need to check to see if it is in a leap year or not.

`month` – What constitutes a valid month?

`year` – How do we validate the year?

What you should do is develop submodules called **`validateX(<DataType>): Boolean`** that take in the variable to test and return whether or not it is in a valid state for your object.

Begin by specifying the class name, constants (there should be a couple), and class-fields, use **`Wheel`** (Blackboard -> Unit Materials -> Resources) as a format guide.

Next develop the **default constructor**, remember the default constructor should take no parameters and **construct a valid object**, what constitutes appropriate values for a default constructor is left to your discretion.

Now complete your **alternate constructor**, remember to deal with validation as necessary and to indicate when/if the submodule can FAIL (`throw Exception` in Java). The parameter order should be the same as the order in which your class fields are declared.

The final constructor to complete is the **copy constructor**. A copy constructor should take in an object of the same class, use the accessors to retrieve the object state and initialise the class fields (see the lecture notes).

The next method to focus on is **clone**. Clone should take no imports and return a copy of the current object. You may do this by using the copy constructor and passing it `this`. Clone is very similar to the copy constructor but without the requirement to specify the class name, this will be extremely useful once we cover inheritance.

Complete the getters, equals, toString (see the note), and mutators (in that order) as per the examples (Blackboard -> Unit Materials -> Resources) and lecture slides.

Once you are confident, you can convert your design to java.

toString: The `toString` method should construct a string in the appropriate format to accurately display the state of the object.

It will be useful for later worksheets to have a second version of the toString method that constructs the string in the file output format. A descriptive name for this method may be `toFileString`. You do not need to worry about this for now. If you wish you can try put it in CSV format, but that is not required for this worksheet.

3. Testing Date Class

Using the lecture notes and WheelTestHarness (Blackboard -> Unit Materials -> Resources) as a guide, you need to develop a test harness for your Date class. Your test harness should test (either directly or indirectly) every submodule you wrote for Date.

You **do not need** to provide pseudocode for your test harnesses, but you should write it anyway as good practice.

4. Assignment Task: Image 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 `P06` directory and call it **`AssignmentTask`** your files should be called **`Image.txt`** and **`Image.java`** respectively.

Next you need to develop **`Image`**, again the details for validation can be found in the assignment specification. you should continue to follow the order of, constructors, accessors, equals, toString, mutators in that order.

Image
- originalImage (int[][])
+ clone(): Image + toString(): String + convolution(int[][]): int[][]

Your **`convolution(int[][]): int[][]`** method will `IMPORT` the `kernel`, and `EXPORT` the `resultArray` as a result of doing a convolution operation with the classfield `originalImage` and the `kernel`.

Your **`toString()`** method can return a string version of the array that you can print out to the screen. For example, instead of using: `System.out.println("What I want to Print");` do: `someString += "What I want to Print";`. It will work the same way if you chose to print out a string.

Follow the same process as above to develop the design and then convert it to Java. You will need more than the methods listed in the UML, some of them have just been omitted to save space.

Do not forget to create a test harness for this class as well. This will serve as our testing portion of this task.

End of Worksheet