COMP1006/1406 - Summer 2016 - Tutorial 3

Objectives: practice using inheritance, abstract classes and interfaces.

In this tutorial you will use the following class.

```
public abstract class AStudent{
   private String name;
   private int    id;

   public AStudent(String name, int id){
      this.name = name;
      this.id = id;
   }

   public String getName(){ return name; }
   public int getID(){ return id; }
}
```

1: Abstract

Write a new concrete class called `Student` that extends the `AStudent` class.

Run the testing program `Tutorial3` with command line argument "one".

```
java Tutorial3 one
```

2: Override

In the `Student` class, override the `toString` method that is inherited from the `AStudent` class (which inherited it from the `Object` class). The method should return a string that has the student's name and id.

For example,

```
AStudent s = new Student("cat", 3);
System.out.println(s);
```
displays `"<cat, 3>"`

Run the testing program `Tutorial3` with command line argument "two".

```
java Tutorial3 two
```

3: Extends

Create a new class called `ExchangeStudent` that extends the student class. You should add a constructor that has three input arguments; name, id and home university. You should also redefine the `toString` method so that it returns the name, id and home university.

For example,

```
AStudent s = new ExchangeStudent("cat", 3, "UBC");
System.out.println(s);
```
displays
```
"<cat, 3> [UBC]"
```

Run the testing program `Tutorial3` with command line argument "three".

```
java Tutorial3 three
```

4: implements

Modify the `AStudent` class so that it implements the `Comparable` interface. You should be comparing `AStudent` objects first based on the length of a student's name and then by id. Let `s` and `t` be two students.

- $s < t$ if `s`'s name is shorter than `t`'s name.
- $s > t$ if `s`'s name is longer than `t`'s name.
- when `s`'s name is the same length as `t`'s name, then $s < t$ if `s`'s id is smaller than `t`'s id, $s > t$ if `s`'s id is larger than `t`'s id, and s is equal to t otherwise.

In the `Arrays` class there is a sort method that allows you sort the contents of an array. The sort method relies on the fact that the objects in the array have implemented the Comparable interface.

Run the testing program `Tutorial3` with command line argument "four". Be sure you read and *understand* what this testing program does.

```
java Tutorial3 four
```

5: Another ordering

Redefine the `compareTo` method in the `AStudent` class so that it orders students first by name (alphabetically) and then by id.

6: Yet another ordering

Suppose we wanted to order AStudent objects so after sorting (with `Arrays.sort()` all the non-exchange students come first sorted alphabetically by name and then by id for ties, and then all exchange students come order alphabetically by name first and then by id.

How would you approach this? Which classes need to be modified? Suppose we further want the exchange students to be separated by Canadian universities, North American universities and outside of North America universities?

Extra Coding 

Download the `Game.java`, `Player.java` and `RandomPlayer.java` classes. Read the code and run the Game program.

➨ Create two new players for the game: `IncrementPlayer` and `SmartPlayer`. Both of these new classes must `extends` the `Player` class. The new player classes should behave as follows:

IncrementPlayer This players current guess should be the previous guess ± 1, depending on if the previous guess was greater or less than the secret. For example, if the previous guess was 10 and it was less than the secret, the current guess should be 11. Thus, this player should get closer (by one) to the secret than the previous guess.

SmartPlayer This player should be more clever than the other two players. In particular, this player should perform a binary search for the game. Or something like it. Don't spend too much time getting the binary search working.

You can add whatever attributes and helper methods that you need in these subclasses.

➨ Once you have your new players created, run the `Game2` program. This requires all three player subclasses.

➨ You'll notice that `Player` has an `abstract` method called `updateStats`. Fill in the code for this. This method should update all the statistical attributes that `Player` stores.

➨ Add a method, `public long[] getStats()`, to the `Player` class. This method should return an array of four longs that contain all four statistical attributes in the same order as they appear in the class definition. You can use this method to ask about the statistics of a given player.

➨ Modify the `Player` class so that it `implements` the `Comparable` interface. The ordering should be based on the average number of guesses the player used for all games the player played.

➨ Once your `Player` class has implemented the `Comparable` interface, use the `Arrays.sort` method to sort the array of players. Modify the `Game2` program as follows:

- Generate an array of about 9 players (3 of each kind). Sort the array and print it to standard output in a nice format.

  Note: You'll have to override `Object`'s `toString` method so that you can print the players nicely. You can print something like

  ```
  I am a RandomPlayer: 10 : 2 : 1001 : 134433 : 123.43
  ```

  where the numbers are number of games:best:worst:total:average guesses)

- have each player play 50 games. Resort the array and print it again.

- have each player play another 950 games. Resort the array and print it again.