COSC 1047 – Spring 2017 – Assignment 3
Due: See D2L

This assignment is designed to give you some experience writing Java objects using inheritance. This assignment is done using Eclipse. Import the .zip file that is provided on D2L and when it is complete, export it again as a .zip. All submissions should be done through D2L. You can submit and resubmit as many times as you want up to the due date and time. If you have any problems submitting, email me – alangille@cs.laurentian.ca - in a timely manner. **Make sure your name, student number and a brief description of your program appear at the top of each file in comments. Also make sure that you use comments to clarify your code for marking.** ***Failure to do either of these tasks may result in a mark deduction.***

Note: It is imperative that you import the a3-w17.zip file from D2L properly or you won't be able to work in the proper project environment. Although the screen shots are a bit outdate these instructions are correct:

http://agile.csc.ncsu.edu/SEMaterials/tutorials/import_export/

Submit solutions to question 1 and question 2 only.

1. (20 marks) Complete these classes in the question3 package of the assignment3 eclipse project.

Design a `ship` class that has the following data fields:

- A data field for the name of the ship (a `string`).
- A data field for the year that the ship was built (an `int`).
- A constructor and appropriate accessors and mutators.
- A `toString` method that displays the ship's name and the year it was built.

Design a `CruiseShip` sub class that extends the Ship class. The `CruiseShip` class should have the following:

- An extra data field for the maximum number of passengers (an `int`).
- A constructor and appropriate accessors and mutators.
- A `toString` method that overrides the `toString` method in the base class. The `CruiseShip` class's `toString` method should also include the max passenger limit.

Design a `CargoShip` class that extends the `Ship` class. The `CargoShip` class should have the following:

- An extra data field for the cargo capacity in tonnage (an `int`).
- A constructor and appropriate accessors and mutators.
- A `toString` method that overrides the `toString` method in the base class. The `CargoShip` class's `toString` method should also include the cargo tonnage.

In the appropriate class write an `equals` method to test if two ships are equal - they should be considered equal if they have the same name and were built in the same year.

Demonstrate the classes in a program (`ShipTester`) that has an array of `Ship` objects (at least 5 of them). Assign various `Ship`, `CruiseShip`, and `CargoShip` objects to the array elements (you can hard-code the objects and data) and print out the initial ship configurations. Show that you can use both the accessor and mutator methods on a sampling of the ships (again, you can hard-code the method arguments here if you like). Also show that your `equals` method can determine whether or not two ships are equal (this means creating at least two ships that have "equal" data).

```
Displaying Ships:

Ship[ Name: Queen Annes Revenge Year built: 1701]
Cruise Ship[ Ship[ Name: USS Enterprise Year built: 2245], Passengers: 2400 ]
Cargo Ship[ Ship[ Name: Black Pearl Year built: 1699], Tonnage: 50000 ]
Cruise Ship[ Ship[ Name: USS Voyager Year built: 2371], Passengers: 2800 ]
Cargo Ship[ Ship[ Name: The Victory Year built: 1790], Tonnage: 33100 ]
-------------------------

Checking ship 2 (accessing):

Name: Black Pearl
Year Built: 1699
Cargo Tonnage: 50000
-------------------------

Changing ship 3 (mutating):

Before: Cruise Ship[ Ship[ Name: USS Voyager Year built: 2371], Passengers: 2800 ]
After: Cruise Ship[ Ship[ Name: USS Enterprise Year built: 2245], Passengers: 2800 ]
-------------------------

Comparing ship 0 to ship 1 - Equal?: false
Comparing ship 1 to ship 3 - Equal?: true
Comparing ship 4 to ship 2 - Equal?: false
```

2. (Goblin.java, Orc.java, Sauron.java, MonterTester.java – 30 marks) Consider the abstract Monster class that is given to you in the project file. This class defines the essential characteristics of Monsters that might be used in a game. Your job is to create three specific Monster classes – as described below – and to follow the standard rules of inheritance in doing so (override methods that need to be overridden, implement methods that need to be implemented, do NOT override or implement methods that are adequately implemented in the superclass, etc., etc.).

   Goblin: A goblin starts off with between 10 and 14 health. They aren't very *healthy* and they do only between 7 and 10 damage per turn. They have an affliction called "chewed" that will take an extra amount of damage from the hero at the end of his turn (see below). There is only a 12% chance that a goblin will inflict "chewed" so 88% of the time you'll return nothing (empty string) and return "chewed" otherwise.

   Orc: Orcs are a bit tougher so they start off with health between 12 and 18. They also do a bit more damage – between 9 and 15 per turn. They have an affliction called "scared" that they inflict 39% of the time.

   Sauron: Sauron is kind of a bad guy – he means well, but still… Thanks to the recent finding of the "One Ring" his starting health is between 28 and 36 and he does a damaging 12 to 20 per turn. There is a 50% chance per turn that the hero will be "seen" which will do extra damage.

   In addition to the health data field that is inherited from the superclass, each Monster subclass has an "affliction" data field to make it easier to keep track of. Make sure that each Monster has a toString() method for easy display.

   Afflictions: If the hero is "chewed" he takes 3 extra damage that round. If the hero is "scared" he does half damage this round. If the hero is "seen" he does no damage this round.

   There are two ways to write a tester for your program.

   Easy-way-out: If you don't feel like really digging into the game aspects of this, no worries. Create an array with two goblins, two orcs and a Sauron. Show that you can use for loops to:

   - Print out each Monster's state.
   - Run through a pretend round and show which afflictions would have affected the hero and the total amount of damage the hero would have taken
   - Deal an amount of damage to each Monster

   Fun-way-out: (5 marks, extra work) Create a array of 5 random Monsters with one (guaranteed) Sauron in the last position. Create a hero (prompt the user for the hero's name). Have your hero fight the Monsters. The hero fights each Monster one-at-a-time until either the Monster or the hero are dead. If the Monster dies, the hero moves onto the next one. If the hero dies, the game is over. During each round, the hero and Monsters take turns attacking each other. The hero starts at the beginning of the Monster array and attacks the first Monster that is still alive. If the Monster survives it attacks the hero (and afflicts the hero potentially). Then the hero moves onto the next living Monster and so on until the whole array has been "processed". After each hero's turn (when the hero attacks) afflictions are removed (it might be best to have the monster attack first each time so that afflictions will make more sense). Then the process repeats. This continues until either the hero has died or all of the Monsters have been vanquished.

   Use lots of display lines (from your main() method of course) to show the epic battle as it unfolds. Be sure to declare your winner (hero or Monsters) in the end.

   PLEASE post the output of some of your battles to the forum so that people can see how your tester is working.**

3. (CardGame.java, ComicBook.java, InventoryTest.java + Java Docs – 25 marks) (20 marks) Consider the following interface:

```
interface InventoryItem {
// Return the quantity of that item currently in stock.
public int getQuantity();
// Get the price of an individual of an item.
public double getRetailPrice();
// Change the quantity of an item absolutely.
public void setQuantity(int qty);
// Adjust the retail price of an item
public void setRetailPrice(double price);
}
```

The idea here is to pretend that you are creating a very simple inventory system for a friend who is opening a new store. Write at least two object classes representing card games and comic books that implement this interface. Each item type should have data fields that define the quantity of the item in stock and the per-unit price of that item. A card game should also have a simple description data field while a comic book should have a title and an issue. Write appropriate constructors, toStrings, and any missing accessors or mutators for your classes.

Write an application class called InventoryTracker that demonstrates your inventory objects. Create (hard-code) several different items of each type and store them in a ***single*** array. Show that you can print their state using toString. Also show that you can use a polymorphic loop to determine the total inventory quantity (total number of items), total inventory value as well as the most and least valuable inventory items in the store (value is the quantity × the retail price).

```
Creating a store inventory array of 5 items...
Stocking the inventory items...Printing store inventory:

Comic Book[ Superman, Issue 2, Retail Price: 100.0, Qty in Stock: 2]
Card Game[ Uno, Retail Price: 9.95, Qty in Stock: 3]
Card Game[ Munchkin, Retail Price: 29.95, Qty in Stock: 10]
Comic Book[ Buffy the Vampire Slayer, Issue 6, Retail Price: 9.95, Qty in Stock: 2]
Comic Book[ X-men, Issue 50, Retail Price: 8.95, Qty in Stock: 12]

Assessing value of in-stock items...

Total Inventory Quantity (num items):29
Total Inventory Value:656.65
Inventory Item with Max Value of 299.5 is: Card Game[ Munchkin, Retail Price: 29.95, Qty in Stock: 10]
Inventory Item with Min Value of 19.9 is: Comic Book[ Buffy the Vampire Slayer, Issue 6, Retail Price:
9.95, Qty in Stock: 2]
```

(5 marks) After you have accomplished the work above, change your tester so that it allows the user to enter the inventory items instead of hard-coding them. Your tester:

- Prompts the user for the size of the array
- Uses a loop to prompt the user for each item in the array
- Prompt the user for the type of item and then the necessary data to construct the item
- Then run the remainder of the program as shown above.