CS 249: Assignment 6

Inheritance and Polymorphism

Theory Questions (24%)

Note: for any UML diagrams, you can use a UML drawing utility like UMLet.
For ALL classes coded and designed, make sure they are:

- Clear, consistent, and have central purpose
- Maintaining a good abstraction
- Enforcing encapsulation

1. (2%) Given a superclass ”Animal” and a subclass ”Walrus”, write the CODE for the class definition of ”Walrus”. It should contain NO data or methods; just show the class definition such that ”Walrus” inherits from ”Animal”.

2. (2%) Given the code below, what is this an example of?

   (a) Upcasting
   (b) Downcasting

   ```
   Animal a = new Walrus();
   ```

3. (2%) Is the code in the previous question legal? Briefly explain your answer.

4. (2%) **Design and draw the UML diagrams for the following classes:**
Also draw the inheritance relationship!

Name: **Item**
It should contain the following data:

- Name
- Weight
- Value

It should have the following functionality:

- No-arg constructor (name = ””, weight = 0, value = 0)
- Constructor that takes name, weight, and value
- Getter/setter functions for data
- Override toString() to return a String with the following format:
Example: if name = ”Sword”, weight = 5, and value = 100:

  Name: Sword
  Weight: 5
  Value: 100

Name: **Weapon**
Use the same class from the last assignment, BUT make the following modifications:

- Inherit from Item
- Remove name field
- Add no-args constructor (damage = 0)
- Change existing constructor to take (name, weight, value, damage)
- Call the appropriate super() constructor where necessary
- Remove getter/setter functions for name (already in Item)
- Override toString() to return super.toString() + the damage:
Example: if name = ”Sword”, weight = 5, value = 100, and damage = 30:

  Name: Sword
  Weight: 5
  Value: 100
  Damage: 30

5. (2%) **Modify the Player class and draw the new UML diagram for it (you do not need to draw any relationships here):**

   Add the following data:

   - ArrayList of Items (inventory)

   Add the following functionality:

   - Adding an Item to inventory
   - Removing an Item from inventory
   - Printing inventory (use the toString() function from Item/Weapon)

6. (2%) A child class inherits private methods from the superclass.

   (a) True
   (b) False

7. (2%) In Java, a subclass may only extend ONE superclass.

   (a) True
   (b) False

8. (2%) A protected field in the superclass is accessible by a subclass, EVEN if the subclass is in a DIFFERENT package than the superclass.

   (a) True
   (b) False

9. (2%) What class is the ancestor of ALL other Java classes? (I.e., it is at the very top of the inheritance tree?)

10. (2%) What Java operator allows me to check what class a given object is?

11. (2%) When is the no-args **super()** constructor ***implicitly*** called?

    (a) Never; it is only called when the programmer explicitly states it.

    (b) Only if the current constructor does not explicitly make a call to another constructor or superclass constructor.

    (c) Under all circumstances.

12. (2%) To **override** a method, what must be the same?

    (a) The signature

    (b) The return type

    (c) Both (a) and (b)

Programming Assignments (76%)

Ensure you are enforcing encapsulation!!!

For this assignment, use a SEPARATE Java file for THE TEST PROGRAM of each requirement (not sub-requirements)! Name these Java files "Assign6_N.java", where N is the requirement number.

Note that you will also create other classes; these should be named "ClassName.java", where ClassName is the name of the public class inside the .java file.

#	*Questions*	
1	**Implement the code for the Player, Item, and Weapon classes designed above. Also implement a class Assign6_1 that tests these classes. In its main method:**	
	Create a Player instance, player, with position (3,3).	
	Do the following in a loop:	
	Ask the user for an item name, weight, and value. **You can assume the name is a single word.**	
	Ask the user if it's a weapon. If user enters "Y", then ask for damage.	
	If item is NOT a weapon, create an Item instance and add it to player inventory.	
	If item IS a weapon, create a Weapon instance and add it to player inventory.	
	...while the user doesn't enter a String "None" for the name.	
	Print player inventory.	

	Example run: Enter item name, weight, and value: sword 10 100 Is this a weapon? [Y/N] Y Enter damage: 200 Enter item name, weight, and value: cloak 5 25 Is this a weapon? [Y/N] N Enter item name, weight, and value: dagger 5 45 Is this a weapon? [Y/N] Y Enter damage: 50 Enter item name, weight, and value: None ////////// Name: sword Weight: 10.0 Value: 100.0 Damage: 200.0 ////////// Name: cloak Weight: 5.0 Value: 25.0 ////////// Name: dagger Weight: 5.0 Value: 45.0 Damage: 50.0	
2	**Implement a Map class and a test program, Assign6_2, according to the following specifications:**	
	Implement a class Map with the following:	

	Data fields: • int mapRows • int mapCols • char [][] mapData	
	Methods: • Constructor that takes the number of rows and columns; initializes mapData to that size and fills it with '.' • boolean copyTo(Map other) – Checks if other is null; if it is, return false – Checks if maps are same size; if not, return false – Copy characters from this.mapData to other.mapData – Return true • void draw() - prints characters in mapData to screen. • boolean setMapLocation(int x, int y, char c) – Checks if x and y are within bounds; if not, return false – Set mapData[y][x] = c – Return true	
	In the main() method of Assign6_2, do the following:	
	Create an instance of Map called map with rows = 10 and columns = 20.	
	Create an instance of Map called display with rows = 10 and columns = 20 (same size).	
	Set the position (2,1) in **map** to '%'.	

	Copy map into display.	
	Set the position (3,4) in **display** to ’@’.	
	Draw display.	
	Expected output: ....%..... ....@.....	

Submission

You will submit the following items as a *.tar or *.zip file:

- A plaintext, Word doc, or PDF with your answers to any theory questions
- Your .java file(s)

Submit this tar/zip file on Blackboard under the appropriate assignment.

Do NOT submit:

- Your .class file(s)
- Your project files

Grading

Below is a list of SOME of the grading penalties:

- Submitting ONLY .class files and NOT .java files
- Sloppy or poor coding style
- Bad coding design principles
- Code that does not compile
- Code that crashes, does not run, or takes a VERY long time to complete
- Using code from ANY source other than the course materials
- Collaboration on code of ANY kind; this is an INDIVIDUAL PROJECT
- Sharing code with other people in this class or using code from this or any other related class
- Output that is incorrect
- Output that is NOT generated by the proper algorithms
- Algorithms/implementations that are incorrect
- Submitting improper files
- Failing to submit ALL required files