CS 249: Assignment 10

Generics

Theory Questions (14%)

1. (2%) In Java, write a **generic** class MyClass that has a type parameter E.

2. (2%) In Java, write a **generic** class YourClass that has a type parameter E **that extends Number**.

3. (2%) In Java, write a **generic** class OurClass that has a type parameter E **that implements Comparable**.

4. (2%) In Java, write a **generic** method doNothing() that is public, non-static, returns void, takes an array of type E, and **has an empty body**.

5. (2%) Is the following Java code correct? If not, why not?

```
ArrayList<int> list = new ArrayList<>();
```

6. (2%) Which of the following is **TRUE** about Generics?
   (a) Generic information is ONLY available at compile time.
   (b) Generic information is available at compile time AND runtime.
   (c) Given a generic type E, the following is legal: E data = new E();
   (d) Given a generic type E, the following is legal: public static E data;
7. (2%) You CANNOT write a class that extends Throwable and uses generic types.
   (a) True
   (b) False

Programming Assignments (86%)

Ensure you are enforcing encapsulation!!!

For this assignment, you will have ONE TEST PROGRAM: “Assign10.java”

Note that you will also create other classes and interfaces; *unless they are inner classes,* these should be named “ClassName.java”, where ClassName is the name of the public class inside the .java file.

#	*Questions*	
1	**Create an abstract class Unit**	
	Fields: • int attack • int health • boolean alive • String name	
	Constructor: takes name, attack, and health, and sets alive to true.	
	Methods: • Getter methods for all fields • Override toString() to return name • int attack(Unit other) – Make sure both the current Unit and the other Unit are alive. – Get a random number from [1, attack] – Subtract this damage from the other Unit’s health – If the health of the other Unit drops to OR below zero, set other Unit’s health to zero AND set alive to false. – Return the amount of damage inflicted.	
2	**Create two classes that inherit from Unit: Human and Orc**	

	Human: Calls super constructor with name = “Human”, attack = 20, health = 100	
	Orc: Calls super constructor with name = “Orc”, attack = 10, health = 50	
3	**Create a generic class Army that has a type parameter E that extends Unit**	
	Fields: • ArrayList of type E –> soldiers • String name	
	Constructor: takes name	
	Methods: • Getter method for name • Methods to add, remove, and get soldiers (of type E) • Method to get soldier count • boolean isDefeated() –> returns true if 0 soldiers • Override toString() to print name + “: “ + (health of every soldier separated by a space) • <T extends Unit> void attacks(Army<T> other) – Make an empty ArrayList of type E called deathToll. This is for any soldiers who die from your Army. – For each soldier in your Army: * If the other Army is defeated, break out of the for loop. * Get a random soldier from the other Army * Have the current soldier attack the other soldier * If the other soldier is not alive, remove them from the other Army. * Otherwise, have the other soldier attack your soldier. * If your soldier dies, add the soldier to death toll. – Use removeAll() on your soldiers to remove the soldiers in deathToll from your soldiers.	

4	**Create a class Assign10; in its main() method:**	
	Ask the user to enter the number of Human and Orc troops.	
	Create an Army of Humans called “Gondor”, and add the correct number of Human troops.	
	Create an Army of Orcs called “Mordor”, and add the correct number of Orc troops.	
	Create a variable battleCnt and set it to 0.	
	While both Armies are NOT defeated:	
	Print out “BATTLE “ + battleCnt	
	Have gondor attack mordor	
	Have mordor attack gondor	
	Print out gondor (remember: this will invoke its toString() method)	
	Print out mordor	
	Increment battleCnt	
	Print out the number of battles fought.	
	If gondor is NOT defeated, print out gondor.getName() + “ is victorious!”	
	Else, if mordor is NOT defeated, print out mordor.getName() + “ is victorious!”	
	Else, print out “Both sides lost!”	

SAMPLE OUTPUT:

```
Enter number of Human and Orc Troops:
10 35
BATTLE 0
Gondor: 51 71 80 95 61 80 72 94 96 46
Mordor: 35 38 30 30 45 31 31 34 37 46 42 23 49 43 32 48 43 37 43
   18 33 32 47 17 45 38 35 33 34 38 37 25 35 47 36
```

```
BATTLE 1
Gondor: 37 57 59 89 14 38 53 69 73 16
Mordor: 2 15 12 1 26 16 18 14 18 44 34 32 30 12 46 25 21 24 5 5
   39 4 26 31 24 29 22 37 22 25 46 26
BATTLE 2
Gondor: 29 31 40 60 22 19 62 21
Mordor: 6 7 26 3 9 15 44 33 16 12 6 10 6 32 13 16 13 9 9 28 2 7
   36 12
BATTLE 3
Gondor: 11 16 29 38 2 42
Mordor: 2 3 4 41 19 11 10 3 16 11 3 15 3 36
BATTLE 4
Gondor: 11 16 15 16
Mordor: 2 22 4 4 3 14 15 23
BATTLE 5
Gondor: 7 13 5 6
Mordor: 18 6 6
BATTLE 6
Gondor: 7 13 5 6
Mordor:
7 BATTLES FOUGHT
Gondor is victorious!
```

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