CS 249: Assignment 05

UML Diagram (10%)

You will submit a SINGLE diagram that contains:

- The class diagram for the Giant class
- The class diagram for the Troll class
- The class diagram for the Tree class
- The class diagram for the Ent class
- The class diagram for the Huorn class
- The relationship between them

Unless a method is overridden, you do NOT need to repeat a parent's method in a child class' diagram.

You do NOT need the diagrams for the Woods, Silmarillion, or testing classes.

Programming Assignments (85%)

Giant.java

Create a java file with a public class Giant. The purpose of this class is to be a Giant of the woods. It will have the following public methods:

- **public Giant(String name)**
  - Store this Giant's name.
- **public String getName()**
  - Returns the name of this Giant.
- **public void setName(String name)**
  - Sets the name of this Giant.
- **public String toString()**
  - Just returns their name

Troll.java

Create a java file with a public class Troll. **Troll INHERITS from Giant.** The purpose of this class is to be a Troll, wandering around and generally terrorizing (and eating, where possible) the local population (however, they often are only able to get mutton). ***Do NOT give this class its own data for name; use the data stored in Giant!*** This class will have the following public methods:

- **public Troll(String name)**
  - Call the super-constructor to make sure the name is stored.

- **public String toString()**
    - Returns "Troll " + the result from calling the super class' toString() method.
    - *Example*: if the name was "Bob", then return "Troll Bob"
- **public String cook()**
    - Returns "Mutton again…"

Tree.java

Create a java file with a public class Tree. **Tree INHERITS from Giant.** The purpose of this class is to be a Tree (that is most likely sentient). ***Do NOT give this class its own data for name; use the data stored in Giant!*** This class will have the following public methods:

- **public Tree(String name)**
    - Call the super-constructor to make sure the name is stored.
- **public String toString()**
    - Returns the result from calling the super class' toString() method + " of the trees"
    - *Example*: if the name was "Bob", then return "Bob of the trees"
- **public String speak()**
    - Returns "<rustling>"

Ent.java

Create a java file with a public class Ent. **Ent INHERITS from Tree.** The purpose of this class is to be an Ent, a shepherd of the trees. They speak in Entish, which is a language that takes some considerable to time to say anything in. ***Do NOT give this class its own data for name; use the data stored in Giant!*** This class will have the following public methods:

- **public Ent(String name)**
    - Call the super-constructor to make sure the name is stored.
- **public String toString()**
    - Returns "Ent " + the result from calling the super class' toString() method.
    - *Example*: if the name was "Treebeard", then return "Ent Treebeard of the trees"
- **public String speak()**
    - Returns "HOOM"

Huorn.java

Create a java file with a public class Huorn. **Huorn INHERITS from Tree.** The purpose of this class is to be a Huorn, a sentient but largely terrifying kind of tree. They don't speak as such, but whatever they're saying is extremely angry. ***Do NOT give this class its own data for name; use the data stored in Giant!*** This class will have the following public methods:

- **public Huorn(String name)**
    - Call the super-constructor to make sure the name is stored.

- **public String toString()**
    - Returns "Huorn " + the result from calling the super class' toString() method.
    - *Example*: if the name was "Grimdark", then return "Huorn Grimdark of the trees"
- **public String speak()**
    - Returns "<angry rustling>"

Woods.java

Create a java file with a public class Woods. The purpose of this class is to represent a wooded area in Middle-Earth. Woods may have any number of Giants in it (*hint*: you should have an ArrayList of Giants). It will have the following public methods:

- **public Giant createGiant(String name, String typeName)**
    - If name is an empty String (i.e., has length of 0), return null
    - Otherwise, you will create a ***specific kind of Giant*** (with the provided name) based on the ***typeName***:
        - "GIANT" → Giant
        - "TROLL" → Troll
        - "TREE" → Tree
        - "ENT" → Ent
        - "HUORN" → Huorn
    - If the typeName does NOT match any of the above, return null.
    - Otherwise, return the newly-created object.
    - **WARNING: This should NOT add this Giant to Woods' list of Giants! It should ONLY create the Giant object and return it!**
- **public boolean addGiant(String name, String typeName)**
    - Call createGiant() to create the Giant object.
    - If that Giant object is NOT null:
        - Add that Giant object to your list of Giants
        - Return true
    - Otherwise:
        - Return false
- **public Giant getGiant(int index)**
    - If the index is GREATER THAN OR EQUAL TO zero and LESS THAN the total number of Giants, return the Giant at that index in the list.
    - Otherwise, return null
- **public void printAllGiants()**
    - Print "ALL GIANTS:" using System.out.println()
    - For each Giant in your list, print "- " + (toString() on that Giant) with System.out.println().

- **public void printAllTrees()**
    - Print "ALL TREES:" using System.out.println()
    - For each Giant in your list who is ***either a Tree or a child class of Tree***, print "- " + (toString() on that Tree) + ": " + (speak() on that Tree) with System.out.println().
- **public void printAllTrolls()**
    - Print "ALL TROLLS:" using System.out.println()
    - For each Giant in your list who is ***either a Troll or a child class of Troll***, print "- " + (toString() on that Troll) + ": " + (cook() on that Troll) with System.out.println().

Silmarillion.java

This program has been provided for you.

Your code MUST work correctly with Silmarillion. **With the exception of the package line, DO NOT MODIFY SILMARILLION.JAVA (TOLKIEN WILL BE UPSET!)**

The program creates a Woods object. It then repeatedly asks the user for name and type until either the user enters a blank String for the name OR enters an unknown type. Each valid name/type combination is added as another Giant to the Woods object. After that, it calls printAllGiants(), printAllTrees(), and printAllTrolls() on the Woods object.

Output of Silmarillion (user input in blue):

```
**********
Enter name:
Bob
Enter type:
Ent
**********
**********
Enter name:
Rowboat Gilligan
Enter type:
Tree
**********
**********
Enter name:
Gork Mork
Enter type:
Troll
**********
**********
Enter name:
Ned
Enter type:
Giant
**********
**********
Enter name:
Sneezle
Enter type:
```

```
Troll
**********
**********
Enter name:
DirkDark Darkwood
Enter type:
Huorn
**********
**********
Enter name:

Enter type:

**********
ALL GIANTS:
- Ent Bob of the trees
- Rowboat Gilligan of the trees
- Troll Gork Mork
- Ned
- Troll Sneezle
- Huorn DirkDark Darkwood of the trees
ALL TREES:
- Ent Bob of the trees: HOOM
- Rowboat Gilligan of the trees: <rustling>
- Huorn DirkDark Darkwood of the trees: <angry rustling>
ALL TROLLS:
- Troll Gork Mork: Mutton again...
- Troll Sneezle: Mutton again...
```

Testing Screenshot (5%)

Submit a screenshot showing the results of running the test program(s).

Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 10% - UML Diagram
- 85% - Programming assignments

For the **PROGRAMMING** portion of the assignment, in addition to the usual penalties:

Issue	***Penalty (in %)***
Giant missing	10
Troll missing	15
Tree missing	15
Ent missing	15
Huorn missing	15
Woods missing	30

Giant not implemented properly	5
Troll not implemented properly (including inheritance issues and redundant data)	7.5
Tree not implemented properly (including inheritance issues and redundant data)	7.5
Ent not implemented properly (including inheritance issues and redundant data)	7.5
Huorn not implemented properly (including inheritance issues and redundant data)	7.5
Woods not implemented properly	15
Modifying Silmarillion.java (beyond the package line)	30