1. **Interfaces** The `Comparator` interface provides a way to control how a sort method (such as `Collections.sort`) sorts elements of a collection. For example, the following `main` method sorts a list of strings by their length by using a `StringLengthComparator` object:

```
public static void main(String[] args) {
    List<String> t = new ArrayList<>();
    t.add("a very very very very long string");
    t.add("a short string");
    t.add("a medium length string");
    Collections.sort(t, new StringLengthComparator());
    System.out.println(t);
}
```

The `Comparator` interface is defined as follows:

```
public interface Comparator<T> {
    /**
     * Compares its two arguments for order. Returns a negative
     * integer, zero, or a positive integer as the first argument
     * is less than, equal to, or greater than the second.
     *
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return a negative integer, zero, or a positive integer as
     * the first argument is less than, equal to, or greater than
     * the second.
    public int compare(T o1, T o2);

    // ...
}
```

Implement the class `StringLengthComparator` so that strings are sorted by their length:

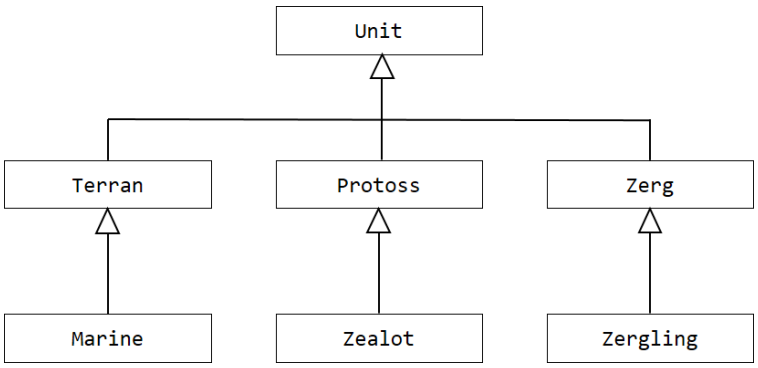
```
public class StringLengthComparator implements Comparator<String> {

    public int compare(String s, String t) {



    }
}
```

Modify your implementation so that `StringLengthComparator` first sorts by string length then by dictionary order (i.e., if two strings have the same length then they are sorted by dictionary order):

```
public class StringLengthComparator implements Comparator<String> {

    public int compare(String s, String t) {




    }
}
```

2. **Inheritance terminology**

   A simplified inheritance hierarchy for the video game Starcraft is shown below:



   (a) Unit is a _____ of Terran

   (b) Protoss is a _____ of Unit

   (c) Zergling is a _____ of Zerg

   (d) Marine is a _____ of Protoss

   (e) Object is a _____ of Unit

3. **Composition instead of inheritance**

   Implement `Stack` using composition instead of inheritance.

```
public class Stack {
    private List<Integer> stack;

    public Stack() {



    }

    public Stack(Stack other) {



    }

    public void push(int value) {


```

```
    }

    public int pop() {




    }
}
```

4. **Subclass constructors**

   Refer back to the figure in Question 2. Every `Unit` has an amount of health; suppose that the constructors for `Unit` are implemented like so:

```
public class Unit {
    private int health;

    public Unit() {
        this.health = 1;
    }

    public Unit(int health) {
        this.health = health;
    }

}
```

   (a) Implement the `Terran` constructors; remember that `Terran` does not have direct access to the field `health`:

```
public class Terran extends Unit {

    public Terran() {


    }

    public Terran(int health) {


    }

}
```

(b) In addition to health, every `Protoss` unit has an amount of shields. Implement the `Protoss` constructors:

```
public class Protoss extends Unit {
    private int shields;

    // initialize unit to 1 health and 1 shields
    public Protoss() {



    }

    public Protoss(int health, int shields) {



    }

}
```

5. Suppose you have a class Y that extends X. X has a method with the following precondition:

   `@pre. value must be a multiple of 2`

   If Y overrides the method which of the following are acceptable preconditions for the overriding method? Provide a brief statement explaining your answer for each precondition.

   (a) `@pre. value must be a multiple of 2`

   (b) `@pre. value must be odd`

   (c) `@pre. value must be a multiple of 2 and must be less than 100`

   (d) `@pre. value must be a multiple of 10`

   (e) `@pre. none`

6. Suppose you have a class Y that extends X. X has a method with the following postcondition:

   ```
   @return a string of length 10
   ```

   If Y overrides the method which of the following are acceptable postconditions for the overriding method? Provide a brief statement explaining your answer for each postcondition.

   (a) `@return a string of length at least equal to 10`

   (b) `@return the string equal to "weimaraner"`

   (c) `@return the empty string`

   (d) `@return a string of length 10`

   (e) `@return a random string of length 10`

7. In the `Mix` class from the lecture slides, which of the following are legal exception specifications? Provide a brief statement explaining your answer for each method header.

   ```
   @Override
   public void someDogMethod() throws BadDogException
   ```

   (a)
   ```
   @Override
   public void someDogMethod() throws Exception
   ```

   (b)
   ```
   @Override
   public void someDogMethod()
   ```

   (c)
   ```
   @Override
   public void someDogMethod() throws DogException, IllegalArgumentException
   ```