Due Date: February 16th

Lab 04: Inheritance, and Interfaces

(Works with Java 17)

In this lab, we'll practice inheritance, and interfaces in two parts. Please ensure you create packages for each part within the same project.

Comparator Interface (15)

Create a new **package** called Lab04A. All the code for this part should reside in there.

Create a class that represents a Person in a file called **Person.java**. It should have instance variables, for a name and an age in years (int). Ensure that it has getters for these variables.

Next, let's test out how the Java Comparator interface works. To use it, we need to import the following:

```
import java.util.Comparator;
```

This will allow you to use the Comparator interface like so:

```
public class AgeComparator implements Comparator<Person>
```

Recall from class that abstract methods defined within an interface need to be implemented by the concrete class. In this case, we will need to implement a compare method:

```
public int compare(Person arg0, Person arg1)
```

Create two classes for comparing objects of type Person by age, and by name. Call them as follows, implement their `compare` methods, and place them in the files `AgeComparator.java`, and `LexicographicComparator.java`.

```
public class AgeComparator implements Comparator<Person>

public class LexicographicComparator implements Comparator<Person>
```

With this you can now make use of builtin Java tools for sorting! Create a Main.java file for your main function, and test your methods by creating an array of Person objects. Then, import the Collections package:

```
import java.util.Collections;
```

Now with your new array of type Person, you can sort it without having to implement your own sorting algorithm (so you can make use of high-quality ones):

```
List<People> people = Arrays.asList(p1, p2, p3, p4, p5);

Collections.sort(people, new LexicographicComparator());
```

Print your array before and after sorting lexicographically, and before/after sorting by age.

Note: In some cases, the Collections.sort will become temperamental and will fail to find the correct method. Try restarting your IDE in this case. Otherwise, please see the instructors of the class.

Abstract Instructor Eccentricities (20)

Create a new **package** called Lab04B. All the code for this part should reside in there.

In universities, various many factors determine how much instructors get stressed but the biggest factor is **unread mail**! In this part of the lab, we'll build a model of how Grad students, Lecturers, and Faculty cope with it, along with how it affects their eccentricities (funny behaviours) and respect.

Instructor

All three of these categories have an age, an int representing their unread mail, and a positive number representing the eccentricities. We should take advantage of these common aspects and abstract them into an **abstract class** called **Instructor** (in a file called Instructor.java). Create getters for the three variables, and a constructor for this class. The constructor should only accept settings for the unread mail and the instructor's age.

Add an abstract method there called **cope()** that returns nothing. Anything that inherits from Instructor will need to implement this method.

Add a method called **stress()** that returns an int. The stress is equal to the unread mail, and it's maximum value is 1000.

Add a method called **respect()** that returns an int. Respect is equal to the instructors age minus the eccentricities (age – eccentricities). This value must be positive so if it's negative return 0.

Add a method to reduce the amount of mail the Instructor has. Call it **reduceMail(int readMail)** and it should return nothing. Ensure that unreadMail doesn't become negative.

Add a method to add to eccentricities. Call it **addToEccentricities(int eccentricities)** and it should return nothing. This method should prevent the eccentricities from becoming negative as well (but the parameter eccentricities can be negative).

Notice here how our setters enforce an encapsulation for any Instructor subclasses. In other words, we have **a single common location** for how the properties get setup. Furthermore, we don't have to reimplement these checks in our subclasses.

Lastly, let's implement a method for getting mail. It should be called **getMail(int newMail)**. The method should add this new mail to the unreadMail. Then, 20% of the time (e.g. `Math.random() < 0.2`) either gain 2 eccentricities or lose 2 eccentricities randomly (50% chance for each). After this, if the Instructor finds that their stress is higher than their respect, they must cope with it.

Grad, Lecturer, and Faculty

Let's make use of our Instructor abstraction!

Create 3 public classes called Grad, Lecturer, and Faculty that inherit from Instructor. Remember that each of these need to implement a constructor that calls the super

You will immediately need to implement their cope methods as these are concrete classes. For each instructor type, these are there cope methods:

- **Grad:** Grad students are an ambitious bunch. To cope with stress, they take a seat and immediately go through all of their mail! This has consequences though so they either gain 3 eccentricities or lose 3 eccentricities randomly (50% chance each).

- **Lecturer:** Lecturers "accidentally" delete 60% of their unread mail.

- **Faculty:** Rather than doing something about the emails, faculty gain 30 eccentricities to cope with it.

The instructor types also have some other oddities:

- **Grad:** Their stress is 1.5x higher than other Instructor's so we need to override the superclass's stress() method to implement this behaviour. Ensure that it doesn't exceed 1500.

- **Faculty:** Thankfully for them, the eccentricities they gain from their custom coping mechanism adds to their respect (instead of subtract). You will need to override the superclass's respect() method for this.

Testing

Create a public class called Main and put your main method in there. Initialize an Instructor array that contains a Faculty object, a Grad object, and a Lecturer object.

For at least 10 days, get new mail for each of the instructors by adding the following amount of mail using the getMail method on each of the objects in the array:

```
mail = (days * 10) + 50
```

Grading Criteria:

Style/submission guidelines: https://gmierzwinski.github.io/bishops/cs321/style_guidelines.html

Comments, Formatting, & Readability	**5 Marks**
Submission Guidelines	**5 Marks**
Program	**35 Marks** **See (X) above**
Total	**45 Marks**