EXERCISE 1

Inheritance can be used to model relationships among classes. The extends and implements keywords are used to define inheritance relationships among classes in Java. In this exercise, you will apply the concept of inheritance by designing a simple inheritance tree as described below.

A stopwatch and a timer are very similar in that both rely on the concept of time. The difference between them is that a timer counts down while a stopwatch counts up.

In this exercise, you are required to create a class, `Time,` that represents a time, a `Timeable` class that represents an object that can be timed and `Stopwatch` and `Timer` classes that represent a stopwatch and timer respectively.

The `Time` class will have the following fields and methods and constructors

1. `hour: int` – the hour of the time
2. `minute: int` – the minute of the time
3. `second: int` – the second of the time
4. `add(t: Time): Time` – that adds the provided `Time` argument to this `Time` and returns the resulting `Time`.
5. `subtract(t: Time): Time` – that subtracts the provided `Time` argument from this `Time` and returns the resulting `Time`.
6. `Time()`
7. `Time(hour: int, minute: int, second: int)`

You are to provide getters and setters for each of the fields of the `Time` class and provide an implementation for the `subtract` and `add` methods. You can see how to add and subtract time here.

The `Timeable` class will have the following fields and methods and constructors:

1. `time: Time` – that represents the underlying time of the `Timeable`.
2. `reset(): void` – that resets the time, that is, it sets the hour, minute and second of the time to 0.
3. `Timeable(t: Time)`

You are to provide a getter and setter for the `time` field of the `Timeable` class. The `time` field should have protected access and not private. Can you think of a reason why?

Since both the `Stopwatch` and `Timer` are `Timeable`, they will inherit from the `Timeable` class you will create. Create the `Stopwatch` and `Timer` classes too.

The `Timer` class will have the followings methods and constructors

1. `countDown(): void` - which counts down the time.
2. `Timer(t: Time) – which instantiates a Timer with the provided Time argument`

`The Stopwatch` class will have a methods and constructors

1. `countUp(): void` - which counts up the time.
2. `Stopwatch(t: Time) – which instantiates a Timer with the provided Time argument`

For now. Don’t worry about implementing the `countDown` and `countUp` methods for now. You can simply leave them blank.

You will notice that we might as well have the `countUp` and `countDown` methods in the `Timeable` class. Why do you think this is not a good idea?

EXERCISE 2

Create a class called `Palindrome`. The class should have the following static method:

`isPalindrome(s: String): boolean`

the `isPalindrome` method should check if a string is a palindrome. A palindrome is any string that stays the same when written backwards. For example, refer, madam, and rotor are all palindromes but long is not. Ask the user for input and then check if the provided word is a palindrome. This includes numbers. Print true to the console if the word is a palindrome, and false otherwise.

NOTE: Ensure that your methods and fields are named exactly as they appear in this exercise otherwise your tests will fail. Also take not of the following colour codes on methods, fields or classes.

Red – represents a public field, method or class

Green – represents a private field, method or class

Orange – represents a protected field or method

Underline – represents a static field or method

Ensure that you follow these rules strictly as the tests will fail if even just the access modifier is different.

Lastly, you need to add the Junit libraries provided to your classpath in order to run the tests. All your classes should be in the default package. Add the test classes provided to the default package of your project and run them to test your code.