Tutorial 7

You will be expected to engage with the tutor and discuss solutions to the problems below.

1. Explain why a static method cannot refer to an instance variable.

2. Explain why method overloading is useful.

3. Method Overloading

i. Write a method called `average` that accepts two integer parameters and returns their average as a floating point value.

ii. Overload the `average` method of Exercise 3.i such that if three integers are provided as parameters, the method returns the average of all three.

iii. Overload the `average` method of Exercise 3.i to accept four integer parameters and return their average.

iv. Write a method called `multiConcat` that takes a string parameter called `text` and an integer parameter called `count`. Return a string that consists of `text` concatenated with itself `count` times. For example, if the parameter values are `"hi"` and `4`, the return value is `"hihihihi"`. Return the original string if the integer parameter is less than 2.

v. Overload the `multiConcat` method from Exercise 3.iv such that if the `count` parameter is not provided, the method returns the string concatenated with itself. For example, if the parameter is `"test"`, the return value is `"testtest"`

4. Interfaces

i. Can a class implement two interfaces that each contains the same method signature? Explain.

ii. Create an interface called `Visible` that includes two methods:
`makeVisible` and `makeInvisible`. Both methods should take no parameters and should return a `boolean` result. Describe how a class might implement this interface.

iii. Imagine a game in which some game elements can be broken by the player and others cannot. Create an interface called `Breakable` that has a method called `break` that takes no parameters and another method called `broken` that returns a boolean result indicating whether the object is currently broken.

iv. Write a Java interface called `Priority` that includes two methods: `setPriority` and `getPriority`. The interface should define a way to establish numeric priority among a set of objects. Design and implement a class called `Task` that represents a `task` (eg. for a to-do list) that implements the `Priority` interface. Create a driver class to exercise some `Task` objects.