Method overloading

Road Racers do time trials. Sometimes they do **2** trials a day, sometimes they do **3**.

Here is an abstraction of a Road Racer, presented in the form of a class:

```
public class RoadRacer{
  //first declare the class properties
  double average;
  //constructor method
  public RoadRace(){
    this.average = 0;
  }
  public void addTwo(int time1, int time2){
    int total = time1+time2;
    this.average = (double)total/2;
  }
  public void addThree(int time1, int time2, int time3){
    int total = time1+time2+time3;
    this.average = (double)total/3;
  }
  public double getAverage(){
    return average;
  }
}
```

Task 1 - Code Analysis

Complete the following table:

Name of a class attribute	
Name of a class method	
How many parameters does the constructor method have?	
How many accessor methods are there?	

The `RoadRacer` class has two methods which do similar things - they add up numbers and determine the average value.

The purpose of the methods is the same, the only difference is the parameter list.

`addTwo` requires 2 parameters in order to work properly.

`addThree` requires 3 parameters in order to work properly.

This is a perfect scenario for **method overloading**:

1. the purpose of the methods is the same.
2. the parameter lists are somehow different eg the data types of parameters are different or the number of parameters is different

Method overloading allows programmers to design methods with the same name as long as the parameter list used by the methods is different, either because the number of parameters is different or that the data types are different.

For example, a class has the following methods (method code not shown):

```
public void add(int x, int y)
public void add(int x, int y, int z)
public void add(int x, double y)
```

Look carefully at the parameter lists. Each are different and so **at run-time** the program will know which method to call.

Here is the updated RoadRacer class using the **method overloading** strategy:

```
public class RoadRacer{
  //first declare the class properties
  double average;
  //constructor method
  public RoadRace(){
    this.average = 0;
  }
  public void add(int time1, int time2){
    int total = time1+time2;
    this.average = (double)total/2;
  }
  public void add(int time1, int time2, int time3){
    int total = time1+time2+time3;
    this.average = (double)total/3;
  }
  public double getAverage(){
    return average;
  }
}
```

Task 2

Which of the following add methods can we be included in the RoadRacer class?

`public void add(double a, int b)`	▢
`public void add(int a, int b)`	▢
`public void add(int a, double b, int c)`	▢

Task 3

A Person has 3 attributes:

- A name
- An age
- An occupation

We can create an instance of a Person as follows:

Person p = new Person(“Janice Joplin”, 36, “musician”);

The following method calls will produce the results described:

p.displayInfo()	Displays all of the object data in one sentence
p.displayInfo(true)	Displays all of the object data in 3 separate sentences (or in one sentence if the parameter is false)
p.displayInfo(true, 2)	Displays all of the object data in 3 sentences (or in one sentence if the first parameter is false), twice.

Design the full `Person` class.

Advantages of Method Overloading

As the Person class demonstrates, 3 methods behave in very similar ways. To have to imagine 3 different method names can be time-consuming. Method overloading allows programmers to use a method name that is suitable for the behaviour of the method. This method name can be used more than once as long as the parameter list is somehow different. In conclusion, **method overloading helps to keep our code readable**.

Sample Exam Question

Define polymorphism. [2]