**Question 1: (2 marks)**

Write a class named **Book** that holds information of a book.

| **Book** |
|---|
| -title:String<br>-price:int |
| +Book()<br>+Book(title:String, price:int)<br>+getTitle():String<br>+getPrice():int<br>+setPrice(price:int):void |

Where:

- Book() - default constructor.
- Book(title:String, price:int) - constructor, which sets values to title and price.
- getTitle():String – return title in **uppercase** format.
- getPrice():int – return price.
- setPrice(price:int):void – update price.

*Do not format the result.*

The program output might look something like:

| | |
|---|---|
| Enter title: atlanta<br>Enter price: 12<br>1. Test getTitle()<br>2. Test setPrice()<br>Enter TC (1 or 2): 1<br>OUTPUT:<br>ATLANTA | Enter title: atlanta<br>Enter price: 12<br>1. Test getTitle()<br>2. Test setPrice()<br>Enter TC (1 or 2): 2<br>Enter new price: 20<br>OUTPUT:<br>20 |

**Question 2: (3 marks)**

Write a class named **Car** that holds information about a car and class named **SpecCar** which is derived from **Car** (i.e. Car is super class and SpecCar is sub class).

| **Car** |
|---|
| -maker:String<br>-price:int |
| +Car()<br>+Car(maker:String, price:int)<br>+getMaker():String<br>+getPrice():int<br>+setMaker(maker:String):void<br>+toString():String |

Where:

- getMaker():String – return maker.
- getPrice():int – return price.
- setMaker(maker:String):void – update maker.
- toString():String – return the string of format:
  **maker, price**

| **SpecCar** |
|---|
| -type:int |
| +SpecCar()<br>+SpecCar(maker:String, price:int, type:int)<br>+toString():String<br>+setData():void<br>+getValue():int |

Where:

- toString():String – return the string of format:
  **maker, price, type**
- setData():void – Add string "XZ" to the head of maker and increase price by 20.
- getValue():int – Return price+inc, where if type<7 then inc=10, otherwise inc=15.

The program output might look something like:

| | | | |
|---|---|---|---|
| Enter maker: hala<br>Enter price: 500<br>Enter type: 7<br>1. Test toString()<br>2. Test setData()<br>3. Test getValue()<br>Enter TC (1,2,3): 1<br>OUTPUT:<br>hala, 500<br>hala, 500, 7 | Enter maker: hala<br>Enter price: 500<br>Enter type: 7<br>1. Test toString()<br>2. Test setData()<br>3. Test getValue()<br>Enter TC (1,2,3): 2<br>OUTPUT:<br>XZhala, 520 | Enter maker: hala<br>Enter price: 500<br>Enter type: 6<br>1. Test toString()<br>2. Test setData()<br>3. Test getValue()<br>Enter TC (1,2,3): 3<br>OUTPUT:<br>510 | Enter maker: hala<br>Enter price: 500<br>Enter type: 8<br>1. Test toString()<br>2. Test setData()<br>3. Test getValue()<br>Enter TC (1,2,3): 3<br>OUTPUT:<br>515 |

**Question 3: (3 marks)**

Write a class named **Car** that holds information about a car.

| **Car** |
|---|
| -maker:String<br>-rate:int |
| +Car ()<br>+Car (maker:String, rate:int)<br>+getMaker():String<br>+getRate():int<br>+setMaker(maker:String):void<br>+setRate(rate:int):void |

Where:

- getMaker():String – return maker.
- getRate():int – return rate.
- setMaker(maker:String): void – update maker.
- setRate(rate:int): void – update rate.

The interface **ICar** below is already compiled and given in byte code format, thus **you can use it without creating ICar.java file**.

```
import java.util.List;
public interface ICar {
    public int f1(List<Car> t);
    public void f2(List<Car> t);
    public void f3(List<Car> t);
}
```

Write a class named **MyCar**, which implements the interface **ICar**. The class MyCar implements methods f1, f2 and f3 in ICar as below (you can add other functions in MyCar class):

- f1: Return the whole part of average rate of all cars (e.g. the whole part of 3.7 is 3).
- f2: Find the first max and min rates in the list and swap their positions.
- f3: Sort the list by maker alphabetically, in case makers are the same, sort them descendingly by rate.

When running, the program will add some data to the list. Sample output might look something like:

| | |
|---|---|
| Add how many elements: 0<br>Enter TC(1-f1;2-f2;3-f3): 1<br>The list before running f1:<br>(A,3) (B,7) (C,6) (D,7) (E,6)<br>OUTPUT:<br>5 | Add how many elements: 0<br>Enter TC(1-f1;2-f2;3-f3): 2<br>The list before running f2:<br>(A,6) **(B,2)** (C,9) **(D,17)** (E,8) (F,17) (G,2)<br>OUTPUT:<br>(A,6) **(D,17)** (C,9) **(B,2)** (E,8) (F,17) (G,2) |

| |
|---|
| Add how many elements: 0<br>Enter TC(1-f1;2-f2;3-f3): 3<br>The list before running f3:<br>(H,1) (G,2) **(E,3)** (F,4) **(E,15)** (C,6) (B,7) (A,8) |

| OUTPUT:<br>(A,8) (B,7) (C,6) **(E,15) (E,3)** (F,4) (G,2) (H,1) |
|---|

**Question 4: (2 marks)**

The interface **IString** below is already compiled and given in byte code format, thus **you can use it without creating IString.java file**.

```
public interface IString {
    public int f1(String str);
    public String f2(String str);
}
```

Write a class named **MyString**, which implements the interface **IString**. The class MyString implements methods f1 and f2 in IString as below:

- f1: Count and return number of prime digits in str.
- f2: Reverse order of all words in str (word = a string without space).

The program output might look something like:

| 1. Test f1()<br>2. Test f2()<br>Enter TC (1 or 2): 1<br>Enter a string:<br>a**32**b 9**5**cd b67<br>OUTPUT:<br>4 | 1. Test f1()<br>2. Test f2()<br>Enter TC (1 or 2): 2<br>Enter a string:<br>a9 b1 a8 a7 a6 a5<br>OUTPUT:<br>a5 a6 a7 a8 b1 a9 |
|---|---|