# PRO192: Exercises for Practical Exam

I. One class: constructor, getX/setX, toString
II. SuperClass & SubClass (Inheritance):

## III. Interface & ArrayList

Create a class used to store information about a **Person** are:

- name - the name of a person (character string).
- age - the age of a person (integer value).

1. void f1() – The **oldList** ArrayList<Person> is given. Create **newList**, and add to newList all elements having **age>4** and the first character of the **name** is different from **'A'**.
   With the given data, **the output should be the following:**
   - oldList: (C4,9) **(C6,3)** (C3,7) **(A3,11)** (C7,9) **(C5,2)** (C1,5) (C2,6) **(A4,9)**
   - newList: (C4,9) (C3,7) (C7,9) (C1,5) (C2,6)
2. void f2() - delete the second node having age<6.
   With the given data in the first line, **the output should be the following:**
   (C4,9) (C6,3) (C3,7) **(C6,1)** (C7,9) (C5,2) (C1,5) (C2,6)
   (C4,9) (C6,3) (C3,7) (C7,9) (C5,2) (C1,5) (C2,6)
3. void f3() – sort the list ascendingly by name (if the names are equal then compare ages) for 5 first elements.
   With the given data in the first line, **the output should be the following:**
   **(C4,9) (C6,3) (C3,7) (C6,1) (C7,9)** (C5,2) (C1,5) (C2,6)
   **(C3,7) (C4,9) (C6,1) (C6,3) (C7,9)** (C5,2) (C1,5) (C2,6)
4. void f4() – reverse the list except the last element.
   With the given data in the first line, **the output should be the following:**
   **(C4,9) (C6,3) (C3,7) (C6,1) (C7,9) (C5,2) (C1,5)** (C2,6)
   **(C1,5) (C5,2) (C7,9) (C6,1) (C3,7) (C6,3) (C4,9**) (C2,6)
   void f5() – delete the element right before the end element.
   With the given data in the first line, **the output should be the following:**
   (C4,9) (C6,3) (C3,7) (C6,1) (C7,9) (C5,2) **(C1,5)** (C2,6)
   (C4,9) (C6,3) (C3,7) (C6,1) (C7,9) (C5,2) (C2,6)
5. void f6() – Change the name of the first node having name = "C6" to "CX".
   With the given data in the first line, **the output should be the following:**
   (C4,9) (**C6**,3) (C3,7) (C6,1) (C7,9) (C5,2) (C1,5) (C2,6)
   (C4,9) (**CX**,3) (C3,7) (C6,1) (C7,9) (C5,2) (C1,5) (C2,6)

## IV. String.

Tách từ: "ab12 bcc de685 ca ba4 xyzw m1 a5 zz933 um8"

1. Tách thành các từ, đếm xem có bao nhiêu từ? // 10
2. Tách từ, in ra tất cả các từ độ dài ngắn nhất? // ca, m1, a5
3. Xóa tất cả các từ không có chữ số nào? // bcc, ca, xyzw
4. Tìm từ có độ dài ngắn nhất (đầu tiên) và xóa đi? // ca
5. In ra tất cả các từ có chữ cái 'a' và 'b'? // ab12, ba4
6. In ra tất cả các từ kết thúc bằng 2 chữ số? // ab12, de685, zz933
7. Sắp xếp 6 từ cuối cùng? // ab12 bcc de685 ca **a5 ba4 m1 xyzw zz933 um8**
8. Tìm từ dài nhất (đầu tiên) và thay thế nó bằng số ghi độ dài của nó // de685 => 5
9. Đổi chỗ từ dài nhất (đầu tiên) và ngắn nhất (cuối cùng) cho nhau. // de685 ⇔ a5
10. Xóa hết các chữ cái, tách thành các từ-số, sắp xếp các số giảm dần? //933 685 12 8 5 4 1

## V. The real PE samples

**Paper 1:**

**Question 1:**

**(2 Point)** Write a class placed `Lake` that holds information of a lake.

| Lake |
| --- |
| -place:String<br>-depth:int |
| +Lake()<br>+Lake(place:String, depth:int)<br>+getPlace():String<br>+getDepth():int<br>+setDepth(depth:int):void |

Where:

- Lake() - default constructor.
- Lake(place:String, depth:int) - constructor, which sets values to place and depth.
- getPlace():String – return place in **uppercase** format.
- getDepth():int – return depth.
- setDepth(depth:int):void – update depth.

*Do not format the result.*

The program output might look something like:

| | |
| --- | --- |
| Enter place: atlanta<br>Enter depth: 10<br>1. Test getPlace()<br>2. Test setDepth()<br>Enter TC (1 or 2): 1<br>OUTPUT:<br>ATLANTA | Enter place: atlanta<br>Enter depth: 10<br>1. Test getPlace()<br>2. Test setDepth()<br>Enter TC (1 or 2): 2<br>Enter new depth: 20<br>OUTPUT:<br>20 |

## Question 2:

**(3 Point)** Write a class named **Mango** that holds information about a mango and class named **SpecMango** which is derived from **Mango** (i.e. Mango is super class and SpecMango is sub class).

| Mango |
| --- |
| -source:String<br>-price:int |

Where:

- getSource():String – return source.
- getPrice():int – return price.

| |
| --- |
| +Mango()<br>+Mango(source:String, price:int)<br>+getSource():String<br>+getPrice():int<br>+toString():String |

- toString():String – return the string of format:
  *source, price*

| SpecMango |
| --- |
| -type:int |
| +SpecMango()<br>+SpecMango(source:String, price:int, type:int)<br>+getNewPrice():int<br>+toString():String |

Where:

- getNewPrice():int – return the value price+inc, where if type>10 then inc=15, otherwise inc=10.
- toString():String – return the string of format:
  *source, price, type*

The program output might look something like:

| | | |
| --- | --- | --- |
| Enter source: bala | Enter source: bala | Enter source: bala |
| Enter price: 20 | Enter price: 40 | Enter price: 40 |
| Enter type: 8 | Enter type: 12 | Enter type: 8 |
| 1. Test toString() | 1. Test toString() | 1. Test toString() |
| 2. Test getNewPrice() | 2. Test getNewPrice() | 2. Test getNewPrice() |
| Enter TC (1 or 2): 1 | Enter TC (1 or 2): 2 | Enter TC (1 or 2): 2 |
| OUTPUT: | OUTPUT: | OUTPUT: |
| bala, 20 | 55 | 50 |
| bala, 20, 8 | | |

## Question 3:

**(3 Point)** Write a class named **Mango** that holds information about a mango.

| Mango |
| --- |
| -source:String<br>-price:int |

Where:

- getSource():String – return source.
- getPrice():int – return price.

```
+Mango ()
+Mango (source:String, price:int)
+getSource():String
+getPrice():int
+setSource(source:String):void
+setPrice(price:int):void
```

- setSource(source:String): void – update source.
- setPrice(price:int): void – update price.

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

```
import java.util.List;

public interface IMango {

    public int f1(List<Mango> t);

    public void f2(List<Mango> t);

    public void f3(List<Mango> t);

}
```

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

- f1: count and return the number of mangos with price < 7.
- f2: Sort the first 4 elements in the list descendingly by price (other elements are unchanged).
- f3: Check if in the list there are at least 2 elements having maximum price then remove the second one (thus if only one element with maximum price then do nothing).

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

```
Add how many elements: 0

Enter TC(1-f1;2-f2;3-f3): 1

The list before running f1:
```

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

```
Add how many elements: 0

Enter TC(1-f1;2-f2;3-f3): 1

The list before running f1:

(A,7) (B,3) (C,8) (D,4) (E,6) (F,9) (G,2) (H,9) (I,2) (J,5)

Output used for grading f1:
```



```
OUTPUT:

6
```

```
Add how many elements: 0
Enter TC(1-f1;2-f2;3-f3): 2
The list before running f2:
(A,7) (B,3) (C,8) (D,4) (E,6) (F,9) (G,2) (H,9) (I,2) (J,5)
Output used for grading f2:
OUTPUT:
(C,8) (A,7) (D,4) (B,3) (E,6) (F,9) (G,2) (H,9) (I,2) (J,5)
```

```
Add how many elements: 0
Enter TC(1-f1;2-f2;3-f3): 3
The list before running f3:
(A,7) (B,3) (C,8) (D,4) (E,6) (F,9) (G,2) (H,9) (I,2) (J,5)
Output used for grading f3:
OUTPUT:
(A,7) (B,3) (C,8) (D,4) (E,6) (F,9) (G,2) (I,2) (J,5)
```

**Question 4:**

**(2 Point)** 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 words starting with even digit (word = a string without space(s).
- f2: Return the string which is made from the string str by removing the first palindrom word. (A string is called palindrom if it and its' reverse are the same).

The program output might look something like:

| 1. Test f1() | 1. Test f1() |
|---|---|



| 2. Test f2() | 2. Test f2() |
|---|---|
| Enter TC (1 or 2): 1 | Enter TC (1 or 2): 2 |
| Enter a string: | Enter a string: |
| ab2 **2ab 4c** 1uv xy4 | hoa **abcdcba** la abcba canh |
| OUTPUT: | OUTPUT: |
| 2 | hoa la abcba canh |

## Paper 2:

**Question 2:**

**(3 Point)** Write a class named **Vase** that holds information about a vase and class named **SpecVase** which is derived from **Vase** (i.e. Vase is super class and SpecVase is sub class).

| Vase |
| --- |
| -maker:String<br>-type:int |

Where:

- getMaker():String – return maker.
- getType():int – return type.



| |
| --- |
| +Vase()<br>+Vase(maker:String, type:int)<br>+getMaker():String<br>+getType():int<br>+toString():String |

- toString():String – return the string of format:
  *maker type*

| SpecVase |
| --- |
| -color:int |
| +SpecVase()<br>+SpecVase(maker:String, type:int, color:int)<br>+getNewType():int<br>+toString():String |

Where:

- getNewType():int – return the value type+inc, where if maker starts with "BA" then inc = 20, otherwise inc=10.
- toString():String – return the string of format:
  *maker type color*

The program output might look something like:

| | | |
| --- | --- | --- |
| Enter maker: bala<br>Enter type: 12<br>Enter color: 50<br>1. Test toString()<br>2. Test getNewType()<br>Enter TC (1 or 2): 1<br>OUTPUT:<br>bala 12<br>bala 12 50 | Enter maker: BAbala<br>Enter type: 12<br>Enter color: 50<br>1. Test toString()<br>2. Test getNewType()<br>Enter TC (1 or 2): 2<br>OUTPUT:<br>32 | Enter maker: bala<br>Enter type: 12<br>Enter color: 50<br>1. Test toString()<br>2. Test getNewType()<br>Enter TC (1 or 2): 2<br>OUTPUT:<br>22 |

**Question 3:**

**(3 Point)** Write a class named **Vase** that holds information about a vase.

| Vase |
| --- |
| -maker:String<br>-type:int |

Where:

- getMaker():String – return maker.
- getType():int – return type.



| +Vase ()<br>+Vase (maker:String, type:int)<br>+getMaker():String<br>+getType():int<br>+setMaker(maker:String):void<br>+setType(type:int):void |
| --- |

- setMaker(maker:String): void – update maker.
- setType(type:int): void – update type.

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

```
import java.util.List;

public interface IVase {

   public int f1(List<Vase> t);

   public void f2(List<Vase> t);

   public void f3(List<Vase> t);

}
```

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

- f1: count and return the number of vases with maker starting with 'B' and ending with 'A'.
- f2: Reverse the first 5 elements in the list (other elements are unchanged).
- f3: Remove the second element having type > 5.

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

```
Add how many elements: 0
Enter TC(1-f1;2-f2;3-f3): 1
The list before running f1:
(B1A,9) (B2A,1) (A1B,8) (A,2) (A2B,7) (B3A,6) (B,5)
Output used for grading f1:
OUTPUT:
```

```
Add how many elements: 0
Enter TC(1-f1;2-f2;3-f3): 2
The list before running f2:
(B1A,9) (B2A,1) (A1B,8) (A,2) (A2B,7) (B3A,6) (B,5)
Output used for grading f2:
OUTPUT:
(A2B,7) (A,2) (A1B,8) (B2A,1) (B1A,9) (B3A,6) (B,5)
```

```
Add how many elements: 0
Enter TC(1-f1;2-f2;3-f3): 3
The list before running f3:
(B1A,9) (B2A,1) (A1B,8) (A,2) (A2B,7) (B3A,6) (B,5)
Output used for grading f3:
OUTPUT:
(B1A,9) (B2A,1) (A,2) (A2B,7) (B3A,6) (B,5)
```

**Question 4:**

**(2 Point)** 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 words ending with digit (word = a string without space(s).
- f2: Return the string by replacing the first shortest word in str by it's length.

The program output might look something like:

| | |
|---|---|
| 1. Test f1()<br><br>2. Test f2() | 1. Test f1()<br><br>2. Test f2() |

| | |
|---|---|
| Enter TC (1 or 2): 1<br><br>Enter a string:<br><br>ab **cd3 uv5** 3xy<br><br>OUTPUT:<br><br>2 | Enter TC (1 or 2): 2<br><br>Enter a string:<br><br>hoa **ab** la canh<br><br>OUTPUT:<br><br>hoa 2 la canh |