Practical-Exam-02: Java and Object-Oriented Programming

Due 30 Aug by 12:00 **Points** 100 **Available** 30 Aug at 10:15 - 30 Aug at 12:00 about 2 hours

This assignment was locked 30 Aug at 12:00.

Introduction

This session ends 12 pm

Make sure to complete your submission in time.

Submit to SVN & Websubmission (for every problem solved)

Submission

- Create the repository on the SVN server
- You must checkout only the folder practical-exam-02 from your server
- No other folders from your SVN will be allowed during the practical exam.
- Check if your repository is being track by WebSubmission before start solving the exam.

```
svn mkdir -m "first assessment commit" --parents https://version-control.adelaide.edu.au/svn/<YOUR-ID>/2019/s2/fcs/week-05/practical-exam-02
```

Assessment

- **This is a practical exam - your work must be entirely your own.**
- Marks for this practical exam contribute 2 marks for the final mark of this course.
- Style marks will be awarded later by your workshop supervisor (40%) and websubmission marker (60%).
- **Due date: the end of this section (12 pm).**
- **Do Not Forget** To Submit on **WebSubmission**
- **Late penalties:** Only work submitted during your enrolled practical session from a Linux system in the practical lab will be marked.

Regarding functional marks, please consider:

```
(1) only codes that can compile will be marked;
(2) only codes that are in the suggested directory will be marked;
(3) only codes submitted to SVN and WebSubmission before the deadline will be marked;
(4) only codes containing your signature on the top of the file will be marked by tutors;
(5) you will have your markers decreased in 3 points if *.class file present in your folders;
```

Signature on your files

Note that all your coding files must contain on the top of it this information:

```
//==================================
// Foundations of Computer Science
// Student: you name
// id: your id
// Semester:
// Year:
// Practical Exam Number:
//==================================
```

Note

- To acquire full marks **(1)** all your functionalities must work perfectly, **(2)** your code has to be well and proportionally commented, and **(3)** your code must follow correct indentation (4 spaces for every new indentation level) and **(4)** you have to use all the content from latest lectures.
- We argue that you are not just asked to solve a problem but use the more sophisticated way to solve it. For instance, you can solve a problem using ten variables, but it will always be better to solve the same problem with an array.

Practical Exam 02

Part 01 - Basic Programming using Java

Problem 01 - Function Sum, Subtraction, Division, and Multiplication,

Define and implement a class **Calculator**, which has the functions of **sum**, **subtraction**, **division** and **multiplication**. They should have the following signatures:

```
'''

Signature:

public class Calculator

public int sum(int numA, int numB);

public int sub(int numA, int numB);

public float multiply(float numA, float numB);

public float division(float numA, float numB);


Repository

save this project as practical-exam-02/problem-01/Calculator.java

add to your svn repository

Important
```

```
1. The class Calculator MUST NOT contain a public static main function.
2. If you want to test your code, create an extra class to test it. This
   class should not be submitted to your svn repository;

Constraints
1. If a number is divided by 0, the result should be -99.0;
2. You are NOT asked to print any information on the screen. The
testing scripts will perform this task

'''
```

Problem 02 - Debugging

A runtime error is a program error that occurs while the program is running. This kind of error is different from other types of program errors, such as syntax errors and compile-time errors. There are many different types of runtime errors. One example is a logic error, which produces the incorrect output.

There is a RunTime error in the bugMethod() code and a compile-time issue in the bugMethod2(). You are asked to fix the following code, and submit to your SVN.

Tips: You should fix the runtime error of bugMethod() and the compile-time error in the bugMethod2() based on the output message of the websubmission automarker.

```
//Code:

public class DebuggingDemo {

    public void bugMethod() {
         int num[] = {1, 2, 3, 4};
         System.out.println(num[5]);
     }

    public int bugMethod2() {
            float ans = 0.0;
            System.out.println("This method had a bug!");
            return ans;
     }


}
```

Current runtime error message in the websubmission system:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5

at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

Repository

```
save this project as practical-exam-02/problem-02/DebuggingDemo.java

add to your svn repository

Requirements

1. Fix the bugs;
2. Provide comments explaining what caused the bug;
```

```
3. The classes MUST NOT contain a public static main function.

'''
```

Problem 03 - Handling Arrays

Define and implement a code to **Handle Arrays**.

```
public class HandlingArrays {

    public static void printArray(double [] testArray) {
          // your code goes here
     }

    public static double[] sumElements(double [] firstArray, double [] secondArray) {
          // your code goes here
     }

    public static double[] reverseArray(double [] testArray) {
          // your code goes here
     }

}
```

You are asked to define:

- Define and implement the `printArray`, for a given array;
- Define and implement `sumElements`, which sum **element-wise** two arrays; The term element-wise means that you want to perform an operation on each element of a vector while doing a computation. For example, you may want to add two vectors by adding all of the corresponding elements.
- You might need to use java.lang.RuntimeException **[link] (https://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html)** to handle exceptions. If you found a compile-time error (something like "error: illegal character: '\uff1b'") when using the following quick tip, please remove it from your code, and then type-in it using keyboard (Don't just copy and paste).

```
Quick tip:

if( // condition checking whether an array is longer than the other ){
            throw new RuntimeException("Error - Arrays different shape");
}else{
 // do something
}
```

Output Expected:
PrintArray method:
Given array {1.0,2.0,3.0,4.0}, for instance. The output of *printArray should* be:
printing Array: [1.0,2.0,3.0,4.0]

sumElements method:
Given array {1.0, 2.0, 3.0} and {3.0, 4.0, 6.0}, the output of **sumElements** should be:
{4.0, 6.0, 9.0}
Given two arrays of different size, the output of **sumElements** should be:
Error - Arrays different shape.

reverseArray method:
Given array {1.0, 2.0, 3.0, 4.0}, the output of reverseArray should be:
{4.0, 3.0, 2.0, 1.0}

Important
1. The class HandlingArrays **MUST NOT** contain a ***public static main*** function.
2. In order to test your code, please create another class and perform your test from this class.
3. The method printArray display information on the screen. The other methods just return an array, no message should be pri
nted;

Repository
save this project as practical-exam-02/problem-03/HandlingArrays.java
add to your svn repository

Part 02 - Object-Oriented Programming

Problem 04 - Creating Classes, Objects, Accessors, and Mutator

Object Oriented programming is a programming style which is associated with the concepts like class, object, Inheritance, Abstraction, Polymorphism. An object-based application in Java is based on declaring classes, creating objects from them and interacting between these objects.

In this problem, you are required to create a few basic classes and set accessors and mutators for their attributes.

```
// filename: Character.java
public class Character
+ properties:
private:
+++ (String) name; //this attribute storage the character name;
+++ (int) age; // this attribute storage the character age;
+++ (String) gender; // this attribute storage the character gender
+++ (String) occupation; // this attribute storage the character occupation
+++ (String) familyRole; // this attribute storage the family role
+++ (float) rate;  // this attribute storage the character overall rate by fans;

+ methods:
public:
++ Constructor: create (1) default constructor, and (2) a Constructor with all parameters;
++ accessors: for all the attributes;
++ mutator: for all the attributes;
```

```
// filename: Cake.java
public class Cake
+ properties:
private:
+++ (String) name;
+++ (float) qtSugar;
+++ (float) qtFlour
+++ (float) qtYeast;
+++ (double) timePrepare;

+ methods:
public:
++ Constructor: create (1) default constructor, and (2) a Constructor with all parameters;
++ accessors: for all the attributes;
++ mutator: for all the attributes;


// filename: Car.java
public class Car
+ properties:
private:
+++ (String) model;
+++ (int) numGears;
+++ (float) literTank;
+++ (int) yearManufacture;

+ methods:
public:
++ Constructor: create (1) default constructor, and (2) a Constructor with all parameters;
++ accessors: for all the attributes;
++ mutator: for all the attributes;
```

````
''''

**Repository**

save this project as practical-exam-02/problem-04/Character.java
                     practical-exam-02/problem-04/Cake.java
                     practical-exam-02/problem-04/Car.java

add to your svn repository

**Requirements**

1. Create classes;
2. Create Constructors;
3. Create Accessors and Mutators;

The classes MUST NOT contain a ***public static main*** function.
'''
````

Read carefully before proceeding, if you have solved all the previous problems correctly, this most likely helped you to acquire 60/100 marks. Now you have two options: 1) finish Problem-05 and acquire the remaining 10 functional marks; or 2) improve the quality of the previous solutions and the 10 functional marks will be given based on how sophisticated your solutions are

for the 4 previous problems. **Note that,** the remaining 30 points is the code style mark, like the previous exam.

Problem 05 - Simple Classes (Optional -- for remaining 10 points)

Define and implement a class named ***ParkingPlace*** that has the following public constructors and behaviors:

```
// creates a parking place of type, vehicle type,
ParkingPlace(String owner, String address, String type, string vehicle)   //constructor

void setOwner(string tmpOwner)                        // sets the owner's name of a parking place
string getOwner()                                     // returns the owner's name of a parking place
void setAddress(string tmpAddress)                    // set the address
string getAddress()                                   // returns the address of a parking place
string getType()                                      // returns the type of a parking place
string getVehicleType()                               // returns the vehicle's type that a parking place may accommodate
void setPrice(int tmpPrice)                            // sets the price of a packing place;
                                                      // price is to be non-negative integer value
int getPrice()                                        // returns the price of a packing place
```

Your program may create only two types of parking places: "on-street" and "off-street". Parking places may accommodate vehicles either of type "car" or of type "motorcycle". Every new off-street parking place must have a price of 15, which does not depend on the type of a vehicle. In other words, parking a car or motorcycle costs 15 for any parking place of the "off-street" type. Every new on-street parking place is free for a motorcycle, therefore its price is 0 by default. Every second new on-street parking place for a car is to be free; otherwise, its price is 10. In other words, when your program creates a new on-street parking place for a car and the number of times this is done by the program is **even**, the price for the parking place must be 0. When your program creates a new on-street parking place for a car and the number of times this is done by the program is **odd**, the price for the parking place must be 10.

Your main program should create several objects of class ***ParkingPlace*** to test the implementation of the class. The address, owner, type, and vehicle type of each of them may be arbitrary. Also, your main program should demonstrate that the price and the owner of a ***ParkingPlace*** object can be changed using the public behaviors shown above.

```
'''

save this project as practical-exam-02/problem-05/ParkingPlace.java

add to your svn repository

'''
```

Basic Marking Scheme (1)

Criteria	Ratings	Pts
Compilation In order to achieve full marks - your code must compile and run;		10.0 pts
Basic Functionality Your code (1) perform all the functions correctly, (2) use latest concepts learned in class, (3) has a clear, creative and sophisticated way of solving the problem.		40.0 pts
Functionality Extension Your code (1) perform all the functions correctly, (2) use latest concepts learned in class, (3) has a clear, creative and sophisticated way of solving the problem, and (4) you propose novel ways to solve the problems - or extra functionalities.		10.0 pts
Code Formatting Your code (1) has the right proportion of comments and place line and block comments correctly, (2) follow correct indentation every new indentation level, (3) has good variable naming, (4) has clear organization between tabs and is easy to read.		40.0 pts
		Total points: 100.0