COMP503/ENSE502/ENSE602: Week 3 – Exercises

➢ Getting started with The Grade Class

Encapsulation is a fundamental idea in Object-Oriented Programming. It is a way for objects to be kept simple and avoid unnecessary complexities to users. An object is protected from unwanted access to attributes and unwanted changes to attributes.

a) Create the following classes in two separate classes

```java
public class Grade {
	int percentage;//a value in the range 0..100.
	public Grade(int percentage){

		this.percentage = percentage;
	}
}
```

```java
public class UniversityGradeApplication {

	public static void main(String[] args) {
		Grade p2 = new Grade(75);//create a grade object
		p2.percentage = 110;
	}
}
```

Note that we expected a programmer using our **Grade** class to provide a value between 0 and 100. How can we be sure that we do not set **percentage** to a value such as 110%? First, we restrict access to the percentage attribute using the **private** keyword. Next, we add a get and a set method to **Grade**:

b) Modify the **Grade** class as follows:

```
public class Grade {
      private int percentage;//a value in the range 0..100.
      public Grade(int percentage){

            this.setPercentage(percentage);
      }

      public void setPercentage(int percentage){

            if((percentage >= 0)&&(percentage<=100))
                  this.percentage = percentage;
            else
                  this.percentage = 0;
      }

      public int getPercentage(){
            return this.percentage;
      }
}
```

When you try to run the **UniversityGradeApplication** class, you now get a **syntax error**:

The field Grade.percentage is not visible!

This means that we have modified the access of **percentage** to **private**. It can only be accessed within the **Grade** class!

c) Update the **main** method of the **UniversityGradeApplication** to change the percentage of **p2** object to 95% using the **setPercentage(95)**method. Print out the percentage value to the console using the **getPercentage()** method.

COMP503/ENSE502/ENSE602: Week 3 – Exercises

➢ Exercise 01

The university keeps a database of students, recording each student's first name and last name and their ID code (e.g. 0980133).

a) Define the Java class **Student** by
   - choosing appropriate object attributes and access modifiers (e.g. **private**),
   - creating two constructors with:
     1. Input parameters for the first and last name and ID
     2. Input parameter for the ID. The student’s first and last names are initialised to the empty string
   - creating get and set methods in the correct format for each object attribute

b) Create a class **StudentApplication** with the following **main** method:

```java
public class StudentApplication {

      public static void main(String[] args) {
            Student aStudent = new Student("Justin","Case","01234");
            System.out.println(aStudent);
      }
}
```

Override the **toString()** method in the Student class such that the **main** method outputs the object **aStudent** to the console as follows:

Sample output:

```
Student's first name: Justin, last name: Case, ID: 01234
```

COMP503/ENSE502/ENSE602: Week 3 – Exercises

- **Exercise 02**

answer one of these. (Note: if you complete all exercises, you may need to finish all!)

A. Design and implement a class called **Dog** that contains instance data that represent the dog's name and age. Define the **Dog** constructor to accept and initialize instance data. Include getter and setter methods for the name and age. Include a method to compute and return the age of the dog in "person years" (seven times the dog's age). Include a **toString** method that returns a one-line description of the dog.
Create a class called **Kennel**, whose main method instantiates and updates several **Dog** objects.

B. Design and implement a class called **Box** that contains instance data that represent the height, width, and the depth of the box. Also include a **boolean** variable called **full** to represent whether the box is full or not. Define the **Box** constructor to accept and initialize the height, width, and depth of the box. Each newly created **Box** is empty (the constructor should initialize **full** to false). Include getter and setter methods for all instance data. Include a **toString** method that returns a one-line description of the box.
Create a class called **BoxTest**, whose main method instantiates and updates several **Box** objects.

C. Design and implement a class called **Sphere** that contains instance data that represent the sphere's diameter. Define the **Sphere** constructor to accept and initialize the diameter, and include getter and setter methods for the diameter. Include methods that calculate and return the volume and surface area of the sphere. Include a **toString** method that returns a one-line description of the sphere.
Create a class called **MultiSphere**, whose **main** method instantiates and updates several **Sphere** objects.

Volume = $\frac{4}{3}\pi r^3$ (***r*** represents the sphere's radius)

Surface area = $4\,\pi\,r^2$

COMP503/ENSE502/ENSE602: Week 3 – Exercises

➢ **Exercise 03**

Shops often have a coin changer machine, which determines the value of a collection of coins and prints a receipt for the customer. We write a Java program which simulates coin changer machine functionality. The machine accepts a collection of coins and computes the value of the coins in dollar and cents.

Example: if a collection contains five 20c pieces, two 10c pieces and a $1 coin, its value is $2.20

We define the program input and output as follows:

- Program inputs: the number of 10c, 20c, 50c, $1 and $2 coins
- Program output: the value of the coins in dollars and change.

a) Develop the **CoinChanger** class that will calculate the value of the coin collection. The class will have
   I. attributes int ten, int twenty, int fifty, int oneDollar, int twoDollar that store the number of each kind of coin described above, with appropriate access modifiers
   II. get methods for each attribute listed above

b) Complete the following public methods in **CoinChanger**
   I. int **computeChange()**, calculates the total value of the coins in cents according to the formula

   totalCents = ten*10 + twenty*20 + fifty*50 + oneDollar*100 + twoDollar*200

   II. int **dollars()**get the integer quotient of the total value in cents divided by 100 (integer quotient of 320 divided by 100 is 3)
   III. int **cents()**get the integer remainder of the total value in cents divided by 100. (integer remainder of 320 divided by 100 is 20)
   IV. **String toString()** returns a string representing the state of the **CoinChanger** object; e.g. the number of each coin input.

c) Design the **CoinChangerApplication** with a main method that creates a **CoinChanger** object and reads amounts of coins input from the console, computing the dollar and cents. The result is printed to the console. Your program should have output similar to the console printed below.

Sample output:

```
Welcome to the Coin Changer Machine. Please input your coins:
Number of 10c coins:
4
Number of 20c coins:
9
Number of 50c coins:
1
Number of dollar coins:
2
Number of two dollar coins:
1
The total value of the coin collection is: $6.70
Would you like to continue? (Y?)
Y
Welcome to the Coin Changer Machine. Please input your coins:
...
```

COMP503/ENSE502/ENSE602: Week 3 – Exercises

- **Exercise 04**

Create the class **RugbyScore** which keeps track of a rugby team's points in a game and has attributes for the name of the team and their current points.

a) Write a constructor for the **RugbyScore** class with an input parameter for the name of the team. The constructor initialises the team's points with a default value (e.g. point = 0).

b) write a toString() method which returns a string representation of a RugbyScore object.

c) In rugby, points can be scored in four ways. Create the **ScoreAction** enumerated type with the following enumerated literals. Each literal is associated with an integer value which keep struck of the number of points each action is worth:
   - TRY, increments the score by 5 points
   - CONVERSIONKICK, increments the score by 2 points
   - PENALTYKICK, increments the score by 3 points
   - DROPGOAL, increments the score by 3 points

d) In the **RugbyScore** class, write a method **score** which takes as input a **ScoreAction** and updates the team's score with the number of points associated with that action.

e) Create **RugbyScoreTest** class and use the following main method to check the functionality of your class.

```java
public static void main(String args[]) {
            RugbyScore blues = new RugbyScore("Blues");
            System.out.println(blues);
            blues.score(ScoreAction.TRY);
            System.out.println(blues);
            blues.score(ScoreAction.DROPGOAL);
            System.out.println(blues);
}
```

f) create the class **RugbyMatch** which comprises two **RugbyScore** objects as attributes that keep track of the score for two teams: the home team and the opposition. Write the following methods:
   - a constructor with input two **RugbyScore** objects
   - a method **homeScore** that takes a **ScoreAction** which updates the score for the home team
   - a method **oppositionScore** with a **ScoreAction** input parameter which updates the score the opposition team

- a method with signature **RugbyScore winner()** which returns the RugbyScore object of the team with the highest score
- write a toString() method which prints the status of the match. Some sample output is given.

```
Highlanders won over Reds 10 - 9

Bulls tied with the Cheetas 5 points
```

- In the first example, the game has completed with a winner, where as in the second example the game has completed with tied point. In the first case, the team with the higher score is listed first.

COMP503/ENSE502/ENSE602: Week 3 – Exercises

- **Exercise 05: Drawing the UML diagrams**

  Answer this exercise if you have time in the lab. Otherwise, work at home!

  a) Consider the following **Student** class:

```
public class Student
{
	private String name;
	private int id;

	private static int NEXT_STUDENT_ID=0;

	public Student(String name)
	{
		this.setName(name);
		this.id = NEXT_STUDENT_ID++;
	}

	public String getName()
	{
		return name;
	}

	public void setName(String name)
	{
		this.name = name;
	}
	public int getId()
	{
		return id;
	}

	public String toString()
	{
		return this.getName()+"("+this.getId()+")";
	}
}
```

Draw the UML class diagram for the **Student** class.

b) Consider the following **Person** and **Address** classes:

```
public class Person
{
	private String name;
	private Address address;
	public Person(String name,Address address){
		this.setName(name);
		this.setAddress(address);
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public Address getAddress() {
		return address;
	}
	public void setAddress(Address address) {
		this.address = address;
	}
}
```

```
public class Address
{
	private int number;
	private String street;
	public int getNumber() {
		return number;
	}
	public void setNumber(int number) {
		this.number = number;
	}
	public String getStreet() {
		return street;
	}
	public void setStreet(String street) {
		this.street = street;
	}
	public Address(int number, String street) {
		this.setNumber(number);
		this.setStreet(street);
	}
}
```

Draw the UML class diagram for the **Person** and **Address** classes.