UIU QUESTION BANK

MID-TERM QUESTION SOLUTIONS

OBJECT ORIENTED PROGRAMMING

CSE 1115

SOLUTION BY

NURUL ALAM ADOR

UPDATED TILL ***SPRING 2024***

nurulalamador.github.io/**UIUQuestionBank**

Index

1. Consider the following codes:

```
class Point2D
{
    int x, y;
    public Point2D(int x, int y){
        this.x = x;
        this.y = y;
        System.out.println("Point2D constructor");
    }
    public String Display(){
        //write codes here
    }
}
class Point3D extends Point2D{
    int z;
    //write codes here
}
```

```
public class Test{
    public static void main(String args[]){
        Point2D p2D = new Point2D(1, 2);
        System.out.println(p2D.Display());
        Point3D p3D = new Point3D(5, 4, 3);
        System.out.println(p3D.Display());
    }
}
```

Now:

I. Complete the "Display" method of the Point2D class that prints all the instance variables,
II. Add a constructor in the Point3D class that uses the base class constructor,
III. Add another method "Display" in the Point3D class. You have to use the parent's "Display" method here, so that the **output** looks like this:

```
Point2D constructor
x = 1, y = 2
Point2D constructor
Point3D constructor
x = 5, y = 4, z = 3
```

Solution:

The code has been rewritten below with the necessary changes:

```
class Point2D {
    int x, y;
    public Point2D(int x, int y) {
        this.x = x;
        this.y = y;
```

```
        System.out.println("Point2D constructor");
    }
    public String Display() {
        return ("x = "+x+", y = "+y);
    }
}

class Point3D extends Point2D{
    int z;
    public Point3D(int x, int y, int z) {
        super(x,y);
        this.z = z;
        System.out.println("Point3D constructor");
    }
    public String Display() {
        return ("x = "+x+", y = "+y+", z = "+z);
    }
}
```

2. Modify the following program by including/excluding the some codes without changing the highlighted parts.

```
public class Myparent {
    private int p;
    public final int myfunction() {
        return p*p;
    }
    public void set_p(int Q) { p = Q; }
    // Write your code here
}
public class Mychild extends Myparent{
    public Mychild(int K){ super(K); }

    public final int myfunction(){
        return p*p+1;
    }
    // write your code of myroot() that finds the square
    // root of p in class Myparent
    // write other necessary codes here
}
```

```
public class Mytest {
    public static void main(String[] args) {
        Myparent c1, c2;
        c1 = new Mychild(2);
        c2 = new Mychild();
        c2.set_p(2);
        int x = c2. myfunction ();
        double y = ((Mychild) c1).myroot();
        // find square root of p in class Myparent
        System.out.println("x = " + x + ", y = " + y);
    }
}
```

Solution:

The code has been rewritten below with the necessary modification:

```
class Myparent {
    private int p;
    public final int myfunction() {
        return p*p;
    }
    public void set_p(int Q) { p = Q; }

    public Myparent(int p) {
        this.p = p;
    }
    public int get_p() { return p; }
}
class Mychild extends Myparent{
    public Mychild(int K){ super(K); }

    public Mychild(){
        super(0);
    }

    // removing myfunction() since it is declared as final in parent

    double myroot() {
        return Math.sqrt(get_p());
    }
}
```

3. Write the output of the following program:

```
public class Person{
    int id;
    String name;
    static int s = 10;
    {
        System.out.println("3");
    }
    public Person(){
        this.id = 1;
        this.name = "M";
        System.out.println("1");
        s++;
    }
    public Person(int id, String name){
        this();
        this.id = id;
        this.name = name;
        System.out.println("2");
        s++;
    }

    public static void main(String args[]){
        Person p = new Person(1, "N");
        Person p1 = new Person();
        System.out.println(p1.s);
        p.s = 11;
```

```
        System.out.println(Person.s);
    }
}
```

Solution:

Output:

```
3
1
2
3
1
13
11
```

4. Suppose that you visit a village market where fresh vegetables and fishes are sold. The sellers sell their items with a profit of $z\%$ of their production cost c. Typical items are given by the following Table:

Food Items	Type t	production cost c per Kg	profit of $z\%$
vegetable	Spinach	20	15
vegetable	Cauliflower	25	18
fish	Carp	300	15
fish	medium	250	20
fish	small	200	25

The class FoodItem that includes type t, production cost c and profit z as public variables and a method findprice() is given as follows:

```
public class FoodItem {
    public double c, z;
    public String t;
    public double getprice(double amount){
        return c*amount*(1+z/100);
    }
}
```

Next, two derived classes Vegetable and Fish are given as follows.

```
public class Vegetable extends FoodItem{
    public void setparameter(){
        if(t == "Spinach"){ c = 20; z = 15; }
        else if(t == "Cauliflower"){c = 25; z = 18;}
    }
    public Vegetable(String t){
        this.t = t;
    }
}
```

```
public class Fish extends FoodItem{
    public void setparameter(){
        if(t == "Carp"){ c = 20; z = 15; }
        else if(t == "medium"){c = 25; z = 20;}
        else if(t == "small"){c = 200; z = 25;}
    }
    public Fish(String t){
        this.t = t;
    }
}
```

Now as a programmer, test the above classes in the main method in a new class *MyTest* by finding your total purchase price if you buy 3Kg fish of type *small* and 2 Kg vegetable of type *Cauliflower*. [Make 2 objects of FoodItems and use the child class references to a FoodItems class object. Then call appropriate methods e.g., **setparameter**, **getprice**.]

Solution:

MyTest class has been written below with the necessary code:

```
public class MyTest {
    public static void main(String[] args) {
        FoodItem fish = new Fish("small");
        FoodItem vegetable = new Vegetable("Cauliflower");

        ((Fish)fish).setparameter();
        ((Vegetable)vegetable).setparameter();

        double fishPrice = fish.getprice(3);
        double vegetablePrice = vegetable.getprice(2);
        double totalPrice = fishPrice + vegetablePrice;

        System.out.println(totalPrice);
    }
}
```

1. a) Logarithms are mathematical functions of the form $\log_b x$ which consists of two parts: base (int b) and argument (double x). The logarithmic value is calculated using the formula:

$$\log_b x = \frac{\log_e x}{\log_e b}$$

Where $\log_b x$ can be evaluated by Java code as Math.log(x).

You have to write a class **Logarithm** and include the **member** variables b and x, different types of **constructors** and a function **myfunc()** to evaluate $\log_b x$ such that the **Main** generates the output provided in the following table:

```
public class Main {
    public static void main(String[] args)
    {
        Logarithm log1 = new Logarithm(2, 9);
        Logarithm log2 = new Logarithm(log1);
        Logarithm log3 = new Logarithm();
        System.out.println(log1.b +" "+log1.x+" "+log1.myfunc());
        System.out.println(log2.b +" "+log2.x+" "+log2. myfunc());
        System.out.println(log3.b +" "+log3.x+" "+log3. myfunc());
    }
}
```

Output

```
2  9.0  3.0
2  9.0  3.0
0  0.0  0.0
```

Solution:

Logarithm class has been written below:

```
class Logarithm {
    int b;
    double x;

    public Logarithm(int b, double x) {
        this.b = b;
        this.x = x;
    }
    public Logarithm(Logarithm log) {
        this.b = log.b;
        this.x = log.x;
    }
    public Logarithm() {
        this.b = 0;
        this.x = 0;
    }
    double myfunc(){
        if (b <= 0 || b == 1 || x <= 0) {
```

```
            return 0.0;
        } else {
            double result = Math.log(x)/Math.log(b);
            return Math.floor(result);
        }
    }
}
```

1. b) What is the output of following java codes:

```
public class Animal {
    public String color;
    public String name;
    public Animal() {
        System.out.println("Default animal");
        color = "Unknown";
    }
    public void showNameColor() {
        System.out.println("Color is: "+ color+" Name is: "+ name);
    }
    {
        System.out.println("Animal instance initialization ");
    }
}
```

```
public class Pokemon extends Animal
{
    public String name = "Pikachu";
    public String color = "Red";
}
public class AnimalTest {
    public static void main(String[] args) {
        Animal defaultAnimal = new Animal();
        Animal pk = new Pokemon();
        defaultAnimal.showNameColor();
        pk.showNameColor();
    }
}
```

Solution:

Output:

```
Animal instance initialization
Default animal
Animal instance initialization
Default animal
Color is: Unknown Name is: null
Color is: Unknown Name is: null
```

2. a) Consider the following codes

```
public class BankAccount {
```

```
    public String name;
    private String account_id;
    private double balance;
    BankAccount(String name, double balance, String gender){
        this.name=name;
        this.balance=balance;
        this.account_id=gender+"-"+name;
    }
    //Add necessary codes here
}
```

```
class Test{
    public static void main(String[] args) {
        BankAccount b=new BankAccount("Mr.Rahman",1000, "M");
        System.out.println("Account id:"+b.account_id);
        System.out.println("balance before:"+b.balance);
        b.balance = b.balance - 2000;
    }
}
```

Rewrite the codes after correcting the errors by adding necessary getter/setter methods to access private variables. ***Do not change access modifiers*** of member variables. While updating the balance, a condition that ***balance cannot be less than 0*** should be considered.

Solution:

The codes has been written below with necessary changes:

```
public class BankAccount {
    public String name;
    private String account_id;
    private double balance;
    BankAccount(String name, double balance, String gender){
        this.name=name;
        this.balance=balance;
        this.account_id=gender+"-"+name;
    }

    public String getAccount_id() {
        return account_id;
    }
    public void setAccount_id(String account_id) {
        this.account_id = account_id;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        if(balance >= 0) {
            this.balance = balance;
        } else {
            System.out.println("Balance cannot be less than 0");
        }
    }
}
```

```
class Test{
    public static void main(String[] args) {
        BankAccount b=new BankAccount("Mr.Rahman",1000, "M");
        System.out.println("Account id:"+b.getAccount_id());
        System.out.println("balance before:"+b.getBalance());
        b.setBalance(b.getBalance() - 2000);
    }
}
```

2. b) Consider the following codes:

```
public class PizzaShop {
    private int pizza_price=320;
    private int drink_price=40;
    private int fries_price=100;
    PizzaShop(){
        // Write necessary codes here
    }
    //Write necessary codes here
}
```

```
class Order{
    public static void main(String[] args) {
        PizzaShop p=new PizzaShop();
        p.order(2,4);
        p.order(1,2,1);
        p.order(3);
    }
}
```

Write the necessary missing codes so that the following output is produced when the program runs:

```
Welcome to pizza shop
You ordered 2 pizzas, 4 drinks
Total bill: 800 taka
You ordered 1 pizzas, 2 drinks, 1 fries
Total bill: 500 taka
You ordered 3 pizzas
Total bill: 960 taka
```

Solution:

The codes has been written below with necessary changes:

```
public class PizzaShop {
    private int pizza_price=320;
    private int drink_price=40;
    private int fries_price=100;

    PizzaShop(){
        System.out.println("Welcome to pizza shop");
    }
```

```
    public void order(int noOfPizza) {
        int totalBill = noOfPizza*pizza_price;
        System.out.println("You ordered "+noOfPizza+" pizzas");
        System.out.println("Total bill: " + totalBill + " taka");
    }
    public void order(int noOfPizza, int noOfDrink) {
        int totalBill = noOfPizza*pizza_price
                                            + noOfDrink*drink_price;
        System.out.println("You ordered " + noOfPizza + " pizzas, "
                                           + noOfDrink + " drinks");
        System.out.println("Total bill: "+totalBill+" taka");
    }
    public void order(int noOfPizza, int noOfDrink, int noOfFries) {
        int totalBill = noOfPizza*pizza_price
                   + noOfDrink*drink_price + noOfFries*fries_pri]ce;
        System.out.println("You ordered " + noOfPizza +" pizzas,"
                  + noOfDrink + " drinks," + noOfFries + " drinks");
        System.out.println("Total bill: "+totalBill+" taka");
    }
}
```

3. Consider the following codes:

```
class Vehicle {
    private String make;
    private String model;
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }
    public void start() {
        System.out.println("[Vehicle] The vehicle is starting.");
    }
    public void stop() {
        System.out.println("[Vehicle] The vehicle is stopping.");
    }
    public void drive() {
        System.out.println("[Vehicle] The vehicle is moving.");
    }
}
```

Create a class named Car that inherits the Vehicle class.

The Car class must have the following attributes/methods:

- An additional attribute named *numberOfDoors* (data type: int, access modifier: private).
- A *constructor* that receives the values of make (String), model (String), numberOfDoors (int) as arguments and initializes the attributes.
- A *method* that overrides the method *drive()* of the parent class. This method invokes *drive()* of the parent class first and then prints “[Car] The car is moving.”.
- Another method named *honk()* (with access modifier: public and return type: void) that prints “[Car] Honk! Honk!”
- Now, consider the class named Main and write the output.

```
public class Main {
    public static void main(String[] args) {
        Vehicle car1 = new Car("make001", "model001", 4);
        car1.drive();
        car1.honk();
    }
}
```

Solution:

The Car class has been created below:

```
class Car extends Vehicle {
    private int numberOfDoors;

    public Car(String make, String model, int numberOfDoors) {
        super(make, model);
        this.numberOfDoors = numberOfDoors;
    }
    public void drive() {
        System.out.println("[Car] The car is moving.");
    }
    public void honk() {
        System.out.println("[Car] Honk! Honk!");
    }
}
```

There will be an error if we execute the code in the given Main class. Here, the variable type of car1 is a Vehicle, and the instance type is Car, which is known as Polymorphism. For this reason, the car1 variable will be able to access only these methods, which are overriding in child class Car from parent class Vehicle. Since the honk() function does not exist in the parent class Vehicle, it will throw an error.

4. Think about your own startup **Uthao** which is a ride sharing program using Java. Now due to our country's rules and regulations, vehicles on your platform must abide by the ***speedLimit*** at 80 km/h.

(a) Create a class name **Ride** that contains an attribute ***speedLimit*** which

- Cannot get changed by its child classes
- Cannot get changed once assigned
- Will contain an Integer value
- Must be **static**

Now create the following child classes of **Ride**:

- Bike
- Car
- Microbus

Each child of Ride (that is, Bike, Car and Microbus) will receive a penalty for exceeding the ***speedLimit***.

All the child classes have the following ***initial_speed*** and ***acceleration***:

- Bike: initial_speed 20 km/h , acceleration 2 km/h
- Car: initial_speed 40 km/h, acceleration 10 km/h
- Microbus: initial_speed 15 km/h, acceleration 5 km/h

Now do the following tasks:

(b) Create a method named ***getHighestAccelerationTime()*** (return type: double) which will find out the time needed for a ride to reach the ***speedLimit*** by using the formula : $v = u + at$

(c) Create a method named ***calculateFine(int hour)*** (return type: double) which will calculate the fine for each vehicle by following the implementation below:

- **Bike:** Base fine will be 50 TK and for each hour exceeding speedLimit, it will add an extra 100 TK
- **Car:** Base fine will be 100 TK and for each hour exceeding speedLimit, it will add an extra 150 TK
- **Microbus:** Total fine will be 3000 TK for just exceeding speedLimit.

(d) Find out the output of the following Code:

```
public class Uthao {
    public static void main(String[] args) {
        Ride car = new Car();
        System.out.println(car.calculateFine(10));
    }
}
```

Solution:

The necessary codes for (a), (b) and (c) have been written below:

```
class Ride {
    static final int speedLimit = 80;
    double initial_speed;
    double acceleration;
    double baseFine;
    double extraFine;

    public Ride(int initial_speed, int acceleration, int baseFine,
                                                        int extraFine) {
        this.initial_speed = initial_speed;
        this.acceleration = acceleration;
        this.baseFine = baseFine;
        this.extraFine = extraFine;
    }

    double getHighestAccelerationTime() {
        /*  From v = u + at,
                  t = (v - u)/a
            Here, u = initial-speed
                  v = speedLimit
                  a = acceleration
        */
        return (speedLimit - initial_speed)/acceleration;
    }

    double calculateFine(int hour) {
        double highestAccelerationTime = getHighestAccelerationTime();
        if(hour >= highestAccelerationTime) {
            double extraHour = Math.floor(hour -
                                            highestAccelerationTime);
            return baseFine + extraHour*extraFine;
```

```
        } else {
            return 0;
        }
    }
}

class Bike extends Ride {
    public Bike() {
        super(20, 2, 50, 100);
    }
}
class Car extends Ride {
    public Car() {
        super(40, 10, 100, 150);
    }
}
class Microbus extends Ride {
    public Microbus() {
        super(15, 5, 3000, 0);
        /* extraFine for Microbus is 0 because total fine will not
           increase for each hour exceeding speedLimit */
    }
}
```

Output for (d) :

```
1000.0
```

Here, For car,

$$t = \frac{v - a}{a} = \frac{80 - 40}{10} = 4\ h$$

Now, For 10 hour,

At $t = 0h$, Fine 0 TK
At $t = 1h$, Fine 0 TK
At $t = 2h$, Fine 0 TK
At $t = 3h$, Fine 0 TK
At $t = 4h$, Fine 100 TK (Base fine)
At $t = 5h$, Fine 250 TK
At $t = 6h$, Fine 400 TK
At $t = 7h$, Fine 550 TK
At $t = 8h$, Fine 700 TK
At $t = 9h$, Fine 850 TK
At $t = 10h$, Fine 1000 TK

5. a) Can you define an abstract method in a non-abstract class? Provide a reason for your answer.

Solution:

No, we cannot define an abstract method in a non-abstract class in Java. This is because an abstract method is inherently a method that lacks an implementation and is meant to be implemented by subclasses. By definition, if a class contains at least one abstract method, the class must be declared as abstract.

5. b) An abstract class cannot contain variables. Do you agree with the statement? Provide a reason for your answer.

Solution:

No, I do not agree with the statement. An abstract class can contain variables. In Java, an abstract class can have variables, and these can be of any type (instance variables, static variables, final variables, etc.). Not only variables, but abstract classes can also contain complete methods. But interfaces cannot contain variables or complete methods in Java. The primary restriction on abstract classes is that they cannot be instantiated directly, without creating any child class.

5. Suppose that you are tasked with modeling a library system for various types of reading materials. You will be working with books, magazines. Each type of reading material has different properties.
Thus, you are required to accomplish the following tasks:

5. c) Write an **abstract class** named **ReadingMaterial** and

i. Add the following **private fields**: title (String), author (String), and year (int).
ii. Add a **constructor** that initializes these fields: (title: String, author: String, year: int).
iii. Define an **abstract method** named **displayDetails()** which should be implemented in subclass (es) to display the details of the reading material.

Solution:

Abstract class ReadingMaterial has been written below:

```
abstract class ReadingMaterial {
    private String title;
    private String author;
    private int year;

    public ReadingMaterial(String title, String author, int year){
        this.title = title;
        this.author = author;
        this.year = year;
    }

    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }
    public int getYear() {
        return year;
    }
    abstract void displayDetails();
}
```

5. d) Create a **subclass** named **Book** that inherits from **ReadingMaterial** and

i. Add an **additional private field** named genre (String) to hold the genre of the book.
ii. Add a **constructor** that takes all parameters (title, author, year, genre) and sets them properly.

iii. Now **override** the **"displayDetails()"** method to display the details of the book, including its title, author, year, and genre.

Solution:

Subclass Book has been written below:

```
class Book extends ReadingMaterial {
    private String genre;

    public Book(String title,String author, int year, String genre){
        super(title,author,year);
        this.genre = genre;
    }

    void displayDetails() {
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Year: " + getYear());
        System.out.println("Genre: " + genre);
    }
}
```

5. e) Create a subclass named **"Magazine"** that inherits from **ReadingMaterial** and

i. Add an **additional private field** named issueNumber (int) to hold the issue number of the magazine.

ii. Add a **constructor** that takes all parameters (title, author, year, issueNumber) and sets them properly.

iii. Now **override** the **"displayDetails()"** method to display the details of the magazine, including its title, author, year, and issue number.

Solution:

Subclass ReadingMaterial has been written below:

```
class Magazine extends ReadingMaterial {
    private int issueNumber;

    public Magazine(String title, String author, int year,
                                          int issueNumber) {
        super(title,author,year);
        this.issueNumber = issueNumber;
    }

    void displayDetails() {
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Year: " + getYear());
        System.out.println("Issue Number: " + issueNumber);
    }
}
```

Summer 2023

1. Write the output of the following program.

```
public class Vehicle {
    private String brand;
    private String model;

    static {
        System.out.println("Initializing Vehicle class...");
    }

    {
        System.out.println("Initializing an instance of the Vehicle
                                                              class...");
    }

    public Vehicle() {
        System.out.println("Creating a default vehicle.");
        brand = "Unknown";
    }

    public Vehicle(String brand, String model) {
        System.out.println("Creating a customized vehicle of brand: " +
                                      brand + " and model: " + model);
        this.brand = brand;
        this.model = model;
    }

    public void honk() {
        System.out.println("The vehicle emits a honking sound.");
    }

    public void honk(String sound) {
        System.out.println("The vehicle emits a custom honking sound: "
                                                                + sound);
    }
    static{
        System.out.println("Making sure of initialization...");
    }

    public void info(){
        System.out.println("model="+model+"brand="+brand);
    }
    public static void main(String[] args) {
        Vehicle defaultVehicle = new Vehicle();
        defaultVehicle.honk();
        defaultVehicle.info();
        Vehicle truck = new Vehicle("Ford", "F-150");
        truck.honk("Loud horn sound");
        truck.info();
    }
}
```

Solution:

Output:

```
Initializing Vehicle class...
Making sure of initialization...
Initializing an instance of the Vehicle class...
Creating a default vehicle.
The vehicle emits a honking sound.
model=null brand=Unknown
Initializing an instance of the Vehicle class…
Creating a customized vehicle of brand: Ford and model: F-150
The vehicle emits a custom honking sound: Loud horn sound
model=F-150 brand=Ford
```

2. Consider the following class named ElectronicDevice representing a generic electronic device.

```
public class ElectronicDevice {
    String brand;
    double price;

    public ElectronicDevice(String brand, double price) {
        this.brand = brand;
        this.price = price;
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Price: $" + price);
    }
}
```

Now write the necessary codes to fulfill the requirements as follows:

1. Create a class named Smartphone - a child class of ElectronicDevice.
2. The Smartphone class has three additional attributes: model (String), operatingSystem (String), and IMEI (String). It must not be possible to set the value of IMEI outside of the class.
3. Create a constructor in Smartphone that takes brand, price, model, operatingSystem, and IMEI as arguments and sets the values of the attributes. This constructor invokes the constructor of ElectronicDevice.
4. There is a method named *displayInfo* in SmartPhone that shows the brand, price, model, and operatingSystem. The method invokes the *displayInfo* of ElectronicDevice.
5. There must be an option to fetch the IMEI outside of the class.

Solution:

The necessary codes has been written below:

```
public class Smartphone extends ElectronicDevice {
    String model;
    String operatingSystem;
    private String IMEI;

    public Smartphone(String brand, double price, String model, String
```

```
    operatingSystem, String IMEI) {
        super(brand, price);
        this.model = model;
        this.operatingSystem = operatingSystem;
        this.IMEI = IMEI;
    }

    public void displayInfo() {
        super.displayInfo();
        System.out.println("Model: " + model);
        System.out.println("Operating System: " + operatingSystem);
    }

    public String getIMEI() {
        return IMEI;
    }
}
```

3. Suppose that you are assigned to compute the volumes of different geometrical objects, i.e., cylinder, sphere and cone. Thus, you are required to do the tasks systemically as follows:

- Write a Java Class called **Myobject** that has a private member variable: r which represents radius of the shape. Add the following function: **findVolume()** that returns -1.0 (since no geometrical object is given).
- Write a child class **Sphere** from **Myobject**. Include the function findVolume() that computes the volume v of a sphere as follows:

$$v = \frac{4}{3}\pi r^3$$

- Write another child class **Cylinder** from **Myobject**. Include a private variable h and the function findVolume() that computes the volume v of a cylinder as follows:

$$v = \pi r^2 h$$

- Write a child class **Cone** from **Cylinder**. Include the function findVolume() that computes the volume v of a cone as follows:

$$v = \frac{1}{3}\pi r^2 h$$

- Add only the necessary Getter methods for each variable in the above classes. Make the necessary parameterized constructors to set the values of the variables.
- Now test your program from main by computing the volumes of different geometrical objects provided in Table 1.

Table 1: List of different geometrical objects and their radius and heights (if applicable)

Myobject	r	h
Sphere1	2.5	
Cone1	1.9	8.9
Cylinder1	1.5	6.5
Cone1	2.7	5.7
Sphere1	3.5	

Hint: Make 5 objects of Myobjects and use each child class reference to each of these objects according to Table 1 using the concept of heterogeneous collection. Next, sum up the volumes which can be obtained by calling the function **findVolume()** through each object.

Solution:

The program has been completed below:

```
public class Myobject {
    private double r;

    public double findVolume(){
        return -1.0;
    }

    public Myobject(double x){
        r = x;
    }

    public double getr(){
        return r;
    }
}

public class Sphere extends Myobject{
    public double findVolume(){
        double r1 = getr();
        double x = 3.0/4.0*Math.PI * Math.pow(r1, 3);
        return x;
    }

    public Sphere(double x){
        super(x);
    }
}

public class Cylinder extends Myobject{
    private double h;

    public double findVolume(){
        double r1 = getr();
        double x = Math.PI * r1*r1*h;
        return x;
    }

    public Cylinder(double x, double y){
        super(x);
        h = y;
    }

    public double geth(){
        return h;
    }
}

public class Cone extends Cylinder{
    public double findVolume(){
```

```
        double r1 = getr();
        double h1 = geth();
        double x = Math.PI * r1*r1*h1/3.0;
        return x;
    }

    public Cone(double x, double y){
        super(x, y);
    }
}

public class JavaApplication5 {
    public static void main(String[] args) {
        Myobject [] sp = new Myobject[5];

        sp[0] = new Sphere(2.5);
        sp[1] = new Cone(1.9, 8.9);
        sp[2] = new Cylinder(1.5, 6.5);
        sp[3] = new Cone(2.7, 5.7);
        sp[4] = new Sphere(3.5);

        double v = 0.0;
        double t;

        for(int i = 0; i < 5; i++){
            t = sp[i].findVolume();
            System.out.println("Individual volume = " + t);
            v += t;
        }

        System.out.println("Total volume: " + v);
    }
}
```

4. Consider the following two classes and the output of the program. Read the comments carefully, **correct errors** in the code of the following **StaticContext** class, and **rewrite** the code for the **StaticContext** class. You can edit, add, or remove any lines excluding the commented ones. You can also write necessary constructors and blocks in the StaticContext class if required.

```
package rollbar;

    //You can't remove or modify this FinalContext class.
    public class FinalContext {
        public final void calculate(){
        System.out.println("calculate method is called");
    }
}
```

```
package rollbar;
public class StaticContext {
    final static int value; //You can't modify or remove this line
                                                        of code
    private double mark;
```

```
    private int count;
    public void calculate(){
        System.out.println("calculate method is called");
    }

    private int getCount() {
        return ++count;
    }

    private static double getMark() {
        return mark;
    }
// You can't modify the following main method.
    public static void main(String... args) {
        count++;
        System.out.println("count= "+getCount());
        System.out.println("value = "+value);
        FinalContext sv = new StaticContext();
        System.out.println("mark= "+((StaticContext)sv).getMark());
        sv.calculate();
    }
}
```

Expected Outcome:

```
count= 2
value = 8
mark= 90.0
calculate method is called
```

Solution:

The program has been written below with the necessary modifications:

```
package rollbar;
public class FinalContext {
    public final void calculate(){
        System.out.println("calculate method is called");
    }
}

package rollbar;
public class StaticContext extends FinalContext {
    private final static int value;
    private double mark;
    private static int count;

    static {
        value = 8;
        count = 0;
    }

    StaticContext(){
        mark = 90;
    }

    private static int getCount() {
```

```
        return ++count;
    }
    private double getMark() {
        return mark;
    }

    // calculate method is removed

    public static void main(String... args) {
        count++;
        System.out.println("count= "+getCount());
        System.out.println("value = "+value);
        FinalContext sv = new StaticContext();
        System.out.println("mark= "+((StaticContext)sv).getMark());
        sv.calculate();
    }
}
```

5.1 a) Can a class be abstract and final simultaneously? Why or why not?

Solution:

No, because by making an abstract class final, you are also restricting it from being inherited by other classes. So, there is no way to be able to even create a subclass object which may be assigned to a variable of that abstract class.

5.1 b) Abstract classes can be created without a single instance variable of method declared inside it. Can you think of any reason why you may want to create such an abstract class?

Solution:

We may create an abstract class without any instance variables or methods to create a class hierarchy, where we do not want to add any features to the top superclass.

5.1 c) Why does a class with an abstract method need to be declared abstract? (Just answering “Compiler will give error” will not get you any marks)

Solution:

Since a method declared as abstract does not have any body, if an object of the class with an abstract method is created, it won’t be able to execute that abstract method and would give error. To prevent making the abstract method get called, we declare the entire class as abstract so that an object of that class cannot be initialized

5.2 Consider the following code:

[P.T.O]

```
abstract class Animal {}

class Baby extends Animal {
    public double age;
}

class Cat extends Animal {
    public void sleep(int time) {
        System.out.println("Sleeping");
    }
}
```

```
public class Main {
    public void speak(Animal target) {
        //write your code here
    }
}
```

Now, complete the speak method such that it prints (You cannot change any other part of the program other the speak method)

i. "WAAHHH" if the variable target is instance of the class Baby,
ii. "Meow" if the variable target is instance of the class Cat,
iii. "Grrrrr" if the variable target is instance of any other subclass of Animal

Solution:

The speak method has been completed below:

```
public void speak(Animal target) {
    if (target instanceof Baby) {
        System.out.println("WAAHHH");
    }
    else if (target instanceof Cat) {
        System.out.println("Meow");
    }
    else if (target instanceof Animal) {
        System.out.println("Grrrr");
    }
}
```

Spring 2023

1. Write the expected output of the following code

```
public class SomeClass {
    SomeClass(){
        System.out.println("I am the base constructor");
    }
    SomeClass(int a){
        this();
        System.out.println("I have an extra value: "+ a);
        this.someMethod(a);
    }
    SomeClass(int a, double b){
        this(a);
        System.out.println("I have more values: " + b);
    }
    public void someMethod(){
        System.out.println("I have no param");
    }
    public void someMethod(int c){
        System.out.println("I borrow "+c+" form a constructor");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        SomeClass s = new SomeClass(5,6.3);
        s.someMethod();
    }
}
```

Solution:

Output:

```
I am the base constructor
I have an extra value: 5
I borrow 5 form a constructor
I have more values: 6.3
I have no param
```

2. Consider the following three classes and the output of the program. Then answer (a),(b) and (c).

```
package M1;

public class Human {
    private int id;
    String intelligence;
    protected boolean bravery;
```

```
}

public class Monster {
    String name;
    double weight;
    public void increaseWeight(double weight) {
      System.out.println("New weight of monster="+(this.weight+weight));
    }

    boolean eat(Human h){
        //check intelligence and return a boolean value
    }

    String scare(boolean bravery){
        //check bravery and return a string
    }
}
```

```
package M1.M2;

public class Test {
    public static void main(String[] args) {

        Human h1=new Human(1,"low",true);
        Human h2=new Human(2,"high",false);

        Monster m1= new Monster ("CookieMonster", 100);

        if (m1.eat(h1)==true) {
            System.out.println("Monster has eaten human "+h1.id);
        }else{
            System.out.println("Human escaped");
        }
        System.out.println(m1.scare(h2.bravery));
    }
}
```

Expected Outcome:

```
To avoid getting scared or eaten be brave or intelligent.
To avoid getting scared or eaten be brave or intelligent.
New weight of monster=110.0
Monster has eaten human 1
Human is too brave to scare
```

a) Correct the given code (don't modify the methods yet) by editing or adding any lines. You cannot remove any line from the code. Also write necessary getter methods (Assume the variables are readonly).

b) Observe the output given and write necessary constructors and blocks accordingly.

c) Implement the eat() and scare() methods. The **eat()** method checks the intelligence of a human- if it's "high", returns true, otherwise it calls the increaseWeight() method and then returns false. The **scare()** method checks a human's bravery- if it's false, it prints a line: "Human scared." otherwise it will print: "Human is too brave to scare."

Solution:

a) The program has been written below with correction:

```
package M1;
public class Human {
    private int id;
    private String intelligence; // Make intelligence private
    protected boolean bravery;

    public Human(int id, String intelligence, boolean bravery) {
        // Adding constructor
        this.id = id;
        this.intelligence = intelligence;
        this.bravery = bravery;
    }
    public int getId() {
        return id;
    }
    public String getIntelligence() {
        return intelligence;
    }
    public boolean isBrave() {
        return bravery;
    }
}

// Monster class remains unchanged


// Test class remains unchanged
```

b) The necessary codes has been written below:

```
package M1;

// Human class with constructor already added in (a)

public class Monster {
    private String name; // Make name private
    private double weight; // Make weight private

    public Monster(String name, double weight) {
        // Add print statement
        System.out.println("To avoid getting scared or eaten be
                                        brave or intelligent.");
        this.name = name;
        this.weight = weight;
    }

    // Other methods remain unchanged
}

// Test class remains unchanged
```

c) The eat() and scare() method have been implemented below:

```
public class Monster {
    // ……other code
    boolean eat(Human h) {
        if (h.getIntelligence().equals("high")) {
            return true;
        } else {
            increaseWeight(10);
            return false;
        }
    }

    String scare(boolean bravery) {
        if (!bravery) {
            System.out.println("Human scared.");
        } else {
            System.out.println("Human is too brave to scare.");
        }
    }
}
```

3. Given the following information write the necessary code to implement AdvancedCalculator class.

```
public class Calculator {
    public int a;
    public int b;
    Calculator(int firstNumber, int secondNumber){
        this.a = firstNumber;
        this.b = secondNumber;
    }
    public int sum(){
        return a+b;
    }
    public int subtract(){
        return a-b;
    }
}
```

```
public class AdvancedCalculator // is a Child of Calculator class
{

}
```

```
public class main {
    public static void main(String[] args) {
        AdvancedCalculator c = new AdvancedCalculator(1,2, new int[]{3,
                                                                    4, 5});
        System.out.println(c.subtract());
        System.out.println(c.sum());
    }
}
```

Expected output

```
Subtraction result: -1
Summation result: 15
```

Following criteria must be fulfilled:

- AdvancedCalculator class has a constructor that uses this keyword to set the value of an attribute called numbers, which is an array of int type.
- Inside AdvancedCalculator class, override the sum() method from its parent. Then use the super keyword to utilize sum() method of its parent class Calculator to find out the sum of first two numbers.Then, add additional lines of code to add the elements of numbers[] with the sum and return the total summation.

Hints:

- 1st line of Expected outcome is the subtraction of 1st two numbers 1 and 2.
- 2nd line of Expected outcome is the summation of 1st two numbers 1 and 2 along with all the elements of **numbers[]** which are 3,4 and 5.

Solution:

The AdvancedCalculator class has been implemented below:

```
public class AdvancedCalculator extends Calculator {
    private int[] numbers;
    public AdvancedCalculator(int firstNumber, int secondNumber, int[]
                                                              numbers) {
        super(firstNumber, secondNumber);
         this.numbers = numbers;
    }


    public int sum() {
        int sum = super.sum();
        /* Use the sum() method of the parent class to find the sum of
        first two numbers */

        for (int number : numbers) {
            sum += number; // Add each element of the array to the sum
        }
        return sum;
    }
}
```

4. Write down the output of the following code:

```
public class Parent {
    public static int count = 0;
    public static void printDetails(){
        count++;
        System.out.println("I am in Parent Class: " + count);
    }
}
public class Child extends Parent{
    public static void printDetails()
    {
```

```
        count++;
        System.out.println("I am in a Child Class: " + count);
    }
}
public class Main {
    public static void main(String[] args) {
        Child x = new Child();
        x.printDetails();
        x.printDetails();
        Parent y = new Parent();
        y.printDetails();
        Child.printDetails();
        Parent.printDetails();
        x.printDetails();
    }
}
```

Solution:

Output:

```
I am in a Child Class: 1
I am in a Child Class: 2
I am in Parent Class: 3
I am in a Child Class: 4
I am in Parent Class: 5
I am in a Child Class: 6
```

5. Find out the errors in the following code and explain the reasons of Errors in those particular lines.

```
class Point{
    int x, y;
    final int f = 10;
    final Point p = new Point(1, 2);
    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }
}
class Check{
    public static void main(String args[]){
        Point point = new Point(5, 5);
        point.f = 5;
        point.p.x = 10;
        point.p = new Point(1, 5);
    }
}
```

Solution:

The program has been written below with correction:

```
class Point {
    int x, y;
```

```
    int f = 10;
    // Removing final keyword to prevent error while changing it

    Point p = null;
    /* Declaring p as null to initialize it later, otherwise
       reinitializing this object will throw error */

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
class Check {
    public static void main(String args[]) {
        Point point = new Point(5, 5);
        point.f = 5;
        point.p = new Point(1, 2);
        // Initialize p to a new Point object
        point.p.x = 10;
        point.p = new Point(1, 5);
    }
}
```

6. Answer the following questions based on the attached code snippet.

```
public class Cake {
    protected String name;
    protected double rate;
    public Cake(String n, double r) {
        name = n;
        rate = r;
    }

    public double calcPrice(){
        System.out.println("Print the calculated price.");
    }
    public printDetails(){
        System.out.println("Prints the detail.");
    }
}

class OrderCake extends Cake{
    double weight;

    public OrderCake(String n, double r, double w){
        super(n, r);
        weight = w;
    }
    //override calcPrice & printDetails
}

class ReadymadeCake extends Cake{
    int quantity;

    public ReadymadeCake(String n, double r, int q){
```

```
        super(n, r);
        quantity = q;
    }
    //override calcPrice & printDetails
}
class Main{
    public static void main(String[] args) {
        Cake cake[];

        // Complete the code

        for (int i = 0; i < cake.length; i++) {
            cake[i].printDetails();
        }
    }
}
```

a) calcPrice() method in the Cake class is used to calculate the total price of a cake. You need to override this method in each of the derived classes: OrderCake and ReadymadeCake. Price is calculated as per the following rules:

- OrderCake: rate * weight
- ReadymadeCake: rate * quantity

Also, override another method printDetails() in each class. This method will print the information about a particular cake according to the following format:

Name: <name>
Rate: <rate>
Weight/Quantity: <value>
Total Price: <price>

Now, complete the OrderCake and ReadymadeCake class by overriding calcPrice() and printDetails() methods.

b) Create an array of three cake objects. First two objects are of OrderCake type and later one object is of ReadymadeCake type in the main() method. Assign some values to the class attributes while creating those objects. If you successfully create the array, your output will look like as follows:

Name: OrderCake
Rate: 150
Weight: 3
Total Price: 450
... .

Name: ReadymadeCake
Rate: 200
Quantity: 2
Total Price: 400
....

Solution:

a) The program has been completed below:

```
class OrderCake extends Cake {
    double weight;
```

```
    public OrderCake(String n, double r, double w) {
        super(n, r);
        weight = w;
    }

    public double calcPrice() {
        return rate * weight;
    }

    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Rate: " + rate);
        System.out.println("Weight: " + weight);
        System.out.println("Total Price: " + calcPrice());
    }
}

class ReadymadeCake extends Cake {
    int quantity;

    public ReadymadeCake(String n, double r, int q) {
        super(n, r);
        quantity = q;
    }

    public double calcPrice() {
        return rate * quantity;
    }

    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Rate: " + rate);
        System.out.println("Quantity: " + quantity);
        System.out.println("Total Price: " + calcPrice());
    }
}
```

b) The necessary codes have been written below:

```
class Main {
    public static void main(String[] args) {
        Cake cake[] = new Cake[3];

        // Create the first two objects of OrderCake type
         cake[0] = new OrderCake("OrderCake", 150.0, 3.0);
         cake[1] = new OrderCake("OrderCake", 150.0, 4.0);

        // Create the third object of ReadymadeCake type
        cake[2] = new ReadymadeCake("ReadymadeCake", 200.0, 2.0);

        for (int i = 0; i < cake.length; i++) {
            cake[i].printDetails();
        }
    }
}
```

1. Write the expected output of the following code

```
public class GoT
{
    {
        System.out.println("Valar dohaeris");
    }
    public String name;
    public String house;
    public double impact;
    public float intent;
    public GoT()
    {
        System.out.println("Best TV series after Breaking Bad");
    }
    public GoT(String name, String house)
    {
        this.name = name;
        this.house = house;
    }
    public GoT(double impact)
    {
        this("Daenerys","Targaryen");
        this.impact = impact;
    }
    public GoT(float intent)
    {
        this("Arya","Stark");
        this.intent = intent;
    }
    public void printFullName()
    {
        System.out.println(name + " " + house);
    }
    public void printDetails()
    {
        printFullName();
        System.out.println("Impact: " + impact);
        System.out.println("Intent: " + intent);
    }
    {
        System.out.println("Valar morghulis");
    }
    public static void main(String[] args) {
        GoT ob1 = new GoT();
        ob1.name = "John Snow";
        ob1.house = "404";
        GoT ob2 = new GoT(4.8);
        ob1.printDetails();
        ob2.printDetails();
    }
}
```

Solution:

Output:

```
Valar dohaeris
Valar morghulis
Best TV series after Breaking Bad
Valar dohaeris
Valar morghulis
John Snow 404
Impact: 0.0
Intent: 0.0
Daenerys Targaryen
Impact: 4.8
Intent: 0.0
```

2. A Class named **Movie** containing following elements: name (String), origin (String), genre (String) and rating (Float). Among them, ***name*** cannot be accessed outside the class and ***origin*** can only be accessed in inherited class. Other two elements can be accessed from anywhere.

   Write down the Movie Class which will contain following members:

   a. Provide appropriate Access Modifier to the elements.
   b. Create Necessary Methods to access name and origin from outside the class.
   c. Two Parameterized Constructors where First Constructor will take all four elements and Second Constructor will take only name and genre.
   d. A method named **void Details()** that will print the details of the Movie in following format:

```
You are watching SHUTTER ISLAND
Origin: USA
Genre: Thriller
Rating: 8.2
```

Solution:

The Movie class has been written below:

```
public class Movie {
    private String name;
    protected String origin;
    public String genre;
    public float rating;

    public Movie(String name, String origin, String genre,
                                                    float rating){
        this.name = name;
        this.origin = origin;
        this.genre = genre;
        this.rating = rating;
    }
    public Movie(String name, String genre){
        this.name = name;
        this.genre = genre;
    }
```

```
    public String getName() {
        return name;
    }
    public String getOrigin(){
        return origin;
    }

    public void Details(){
        System.out.println("You are watching " + name);
        System.out.println("Origin: " + origin);
        System.out.println("Genre: " + genre);
        System.out.println("Rating: " + rating);
    }

    public static void main(String[] args) {
        Movie m = new Movie("SHUTTER  ISLAND", "USA", "Thriller", 8.2F);
        m.Details();
    }
}
```

3.

```
package transport;

public class Vehicle {
    int noOfWheels;

    public Vehicle(int noOfWheels){
        this.noOfWheels = noOfWheels;
    }
}
```

```
package publicTransport;

public class Bus extends Vehicle {
    private int licenseNo;

    Bus(int  licenseNo){
        super.noOfWheels = 4;
        this.licenseNo = licenseNo;
    }
}
```

Rewrite the above blocks of code with the following changes.

- Make sure no other classes have direct access to 'noOfWheels' in 'Vehicle'.
- Make Sure 'noOfWheels' in 'Vehicle' is read-only both from inside and outside of the 'transport' package. (the value can be set only within the constructor)
- Make sure the '`Bus extends Vehicle`' line does not throw any error.
- Resolve any error that might be thrown by the constructor of class '`Bus`'.
- Enable read and write access to 'licenseNo' of the class 'Bus' without changing the given access modifier.

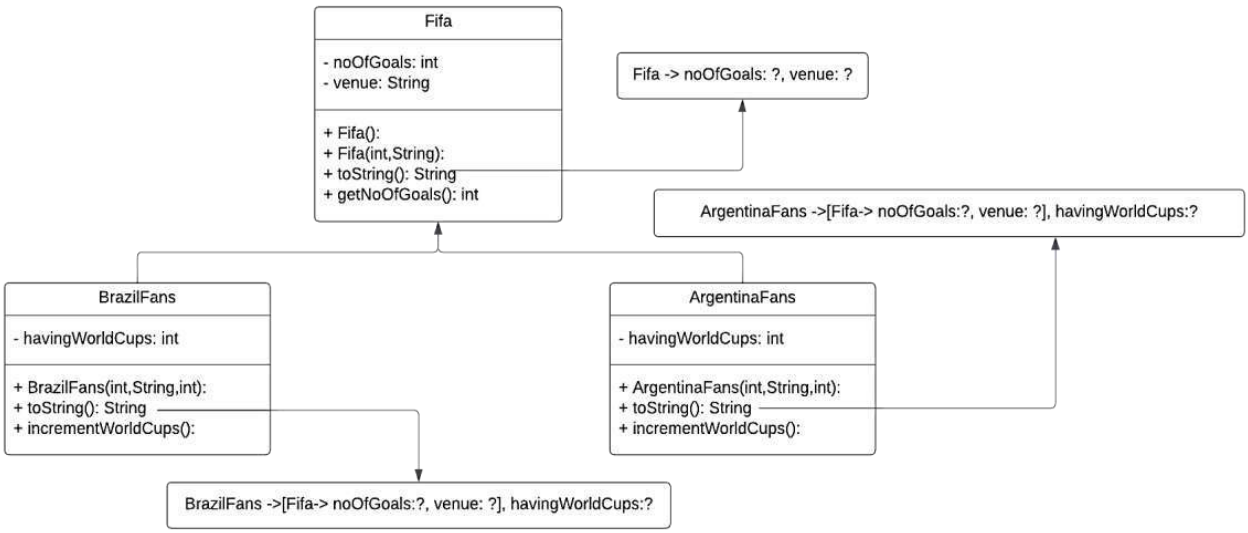
Solution:

The program has been rewritten below with necessary changes:

```
package transport;

public class Vehicle {
    private int noOfWheels;

    public int getNoOfWheels(){
        return  noOfWheels;
    }

    public Vehicle(int noOfWheels){
        this.noOfWheels = noOfWheels;
    }
}
```

```
package publicTransport;

public class Bus extends Vehicle{
    private int licenceNo;

    public Bus(int licenceNo){
        super(4);
        this.licenceNo = licenceNo;
    }

    public int getLicenceNo() {
        return licenceNo;
    }

    public void setLicenceNo(int licenceNo) {
        this.licenceNo = licenceNo;
    }
}
```

4. All of you know that FIFA 2022 with Brazil, Argentina controversy is a trendy topic. Given the following UML diagram, which has three classes: Fifa, BrazilFans, and ArgentinaFans. Although each detail is depicted in the UML diagram, some of the more complex parts are written here for convenience.



There are 3 Classes. **Fifa**, **BrazilFans** and **ArgentinaFans**.

Fifa() constructor prints "Who will be winner?" message. The first line of the **Fifa(int,String)** constructor body is called **Fifa()** constructor with the help of **this** keyword, and the remaining two lines instantiate variables using this keyword. **toString()** methods return a string with the given format.

BrazilFans is a derived class of **Fifa**, which has a constructor with three parameters, the first two of which are passed to the base class, and the last one is used to instantiate variables using **this** keyword. **toString()** methods return a string with the given format. The **incrementWorldCups()** methods increase the value of **havingWorldCups** by 1 when it is invoked.

ArgentinaFans is a derived class of **Fifa**, which has a constructor with three parameters, the first two of which are passed to the base class, and the last one is used to instantiate variables using **this** keyword. **toString()** methods return a string with the given format. The **incrementWorldCups()** methods increase the value of **havingWorldCups** by 1 when it is invoked.

Mid is another class that contains **main()** methods. In the **main()** method, after creating objects of **ArgentinaFans** and **BrazilFans**, if the **noOfGoals** of Argentina is greater than Brazil, then the **incrementWorldCups()** method of **ArgentinaFans** is called; otherwise, the **incrementWorldCups()** method of **BrazilFans** is called. Write the whole program appropriately so that when it executes, you get the following output:

```
Who will be winner?
Argentina will win
Who will be winner?
Brazil will win
ArgentinaFans -> [Fifa -> noofGoals: 10, venue:Qatar], havingWorldCups:2
BrazilFans -> [Fifa -> noofGoals: 7, venue:Qatar], havingWorldCups:5
ArgentinaFans -> [Fifa -> noofGoals: 10, venue:Qatar], havingWorldCups:3
BrazilFans -> [Fifa -> noofGoals: 7, venue:Qatar], havingWorldCups:5
```

Solution:

The program has been written below:

```
class Fifa {
    private int noOfGoals;
    String venue;

    {
        System.out.println("Who will be winner?");
    }

    public Fifa(int noOfGoals, String venue){
        this.noOfGoals=noOfGoals;
        this.venue=venue;
    }

    public String toString(){
        return "Fifa -> noOfGoals: " + noOfGoals+", venue: " + venue;
    }

    public int getNoOfGoals(){
```

```
        return noOfGoals;
    }
}

public class BrazilFans extends Fifa{
    int havingWorldCups;

    {
        System.out.println("Brazil will win");
    }

    public BrazilFans(int noOfGoals, String venue, int havingWorldCups){
        super(noOfGoals, venue);
        this.havingWorldCups = havingWorldCups;
    }

    public String toString(){
        return "BrazilFans -> [" + super.toString() + "],
                                   havingWorldCups: " + havingWorldCups;
    }

    void incrementWordCups(){
        havingWorldCups++;
    }
}

public class ArgentinaFans extends Fifa{
    int havingWorldCups;

    {
        System.out.println("Argentina will win");
    }

    public ArgentinaFans(int noOfGoals, String venue, int
                                                        havingWorldCup){
        super(noOfGoals,venue);
        this.havingWorldCups=havingWorldCup;
    }

    public String toString(){
        return "Argentina -> [" + super.toString() + "],
                                     havingWorldCups: "+havingWorldCups;
    }

    void incrementWordCups(){
        havingWorldCups++;
    }
}

public class Mid {
    public static void main(String[] args) {
        ArgentinaFans argentinaFans= new ArgentinaFans(10,"Qatar",2);
        BrazilFans brazilFans = new BrazilFans(7,"Qatar",5);

        System.out.println(argentinaFans);
        System.out.println(brazilFans);
```

```
        if(argentinaFans.getNoOfGoals() > brazilFans.getNoOfGoals()){
            argentinaFans.incrementWordCups();
        } else{
            brazilFans.incrementWordCups();
        }

        System.out.println(argentinaFans);
        System.out.println(brazilFans);
    }
}
```

5.

```
class PClass {
    void mFnc(){
        System.out.println("Hello from P Class!");
    }

    void mFnc(double d1){
        System.out.println("Double value: " + d1);
    }
}
```

```
class CClass extends PClass {
    void mFnc(){
        System.out.println("Hello from C Class!");
        super.mFnc(11.22);
    }

    void mFnc(int a2, double d2){
        mFnc(d2);
        System.out.println("Integer value: " + a2);
    }
}
```

```
class Main{
    public static void main(String[] args) {
        PClass pObj = new PClass();
        CClass cObj = new CClass();

        pObj.mFnc();
        cObj.mFnc();
        cObj.mFnc(10, 2.99);
        cObj.mFnc(3.145);
    }
}
```

Check the attached Code Snippet and answer following questions:

a) What is the Output of the Main Class?

b) "In the "**CClass**" which is a subclass of "**PClass**", both ***Method Overriding*** and **Method Overloading** has been demonstrated"- Explain the statement using the examples from the code snippet.

Solution:

a) Output:

```
Hello from P Class!
Hello from C Class!
Double value: 11.22
Double value: 2.99
Integer value: 10
Double value: 3.145
```

b) In the “CClass” which is a subclass of “PClass”, both Method Overriding and Method Overloading has been demonstrated.

Method Overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. Here, “CClass” overrides the mFnc() method from “PClass”, providing a new behavior when mFnc() is called on an instance of “CClass”.

Method Overloading occurs when multiple methods have the same name but different parameter lists within the same class. Here, in both “CClass” and “PClass”, the mFnc method is overloaded to accept different parameters such as (void), (double), (int and double), allowing different behaviors based on the arguments passed.