Lab Assignment 09

HOMEWORK

Task 1

Design the **SmartSecurityCamera** class derived from SmartDevice class to generate the following output.

Tester Code and Parent Class

```java
public class SmartHomeTester {
 public static void main(String[] args) {
  SmartSecurityCamera cam1 = new
SmartSecurityCamera("Garden-Cam", 100, 64);
  cam1.powerOn();
  System.out.println("=====================");
  cam1.record(true);
  System.out.println("=====================");
  cam1.powerOff();
  System.out.println("=====================");
  cam1.powerOn();
  System.out.println("=====================");
  cam1.record();
  System.out.println("=====================");
  cam1.formatCard("0000");
  System.out.println("=====================");
  cam1.formatCard("ADMIN123");
  System.out.println("=====================");
  SmartSecurityCamera cam2 = new
SmartSecurityCamera("Indoor-Cam", 80, 1);
  cam2.powerOn();
  System.out.println("=====================");
  cam2.record();
  System.out.println("=====================");
  cam2.powerOff();
  System.out.println("=====================");
  cam2.powerOn();
  System.out.println("=====================");
  cam2.formatCard("ADMIN123");
  System.out.println("=====================");
  cam2 = new SmartSecurityCamera("Indoor-Cam",
2, 10);
```

Output

```
Garden-Cam is now ONLINE.
=====================
[IR SENSORS ACTIVE]
Recording standard footage.
=====================
Turning off Night Vision.
Garden-Cam has shut down.
=====================
Garden-Cam is now ONLINE.
=====================
Recording standard footage.
=====================
REQUEST: Format SD Card
initiated.
ACCESS DENIED: Incorrect PIN.
=====================
REQUEST: Format SD Card
initiated.
Auth Success. Wiping data.
SUCCESS: Storage restored to
64GB.
=====================
Indoor-Cam is now ONLINE.
=====================
Recording standard footage.
=====================
Indoor-Cam has shut down.
=====================
Error: Indoor-Cam storage full.
Recording disabled.
=====================
REQUEST: Format SD Card
```

```java
  cam2.powerOn();
 }
}


class SmartDevice {
 public String deviceName;
 private double batteryLevel;
 protected boolean isActive;

 SmartDevice(String name, double battery) {
  this.deviceName = name;
  this.batteryLevel = battery;
  this.isActive = false;
 }

 public void powerOn() {
  if (batteryLevel > 5) {
   isActive = true;
   batteryLevel -= 2;
   System.out.println(deviceName + " is now
ONLINE.");
  } else {
   System.out.println("Power Low: " +
deviceName + " cannot start.");
  }
 }

 public void powerOff() {
  this.isActive = false;
  System.out.println(deviceName + " has shut
down.");
 }

 public double getBattery() {
  return batteryLevel;
 }
}
```

```
initiated.
ERROR: Device must be ON to
format.
=====================
Power Low: Indoor-Cam cannot
start.
```

Task 2

Your task is to design the **UpsideDown** class with appropriate variables and methods such that the following tester code produces the expected output. Note:

➢ Assume that each gate of **UpsideDown** can connect with two bridges.
➢ You cannot use any arrays in the **UpsideDown** class.
➢ You should use the given **Hawkins** and **DarkDimension** classes' variables and methods as needed.
➢ You cannot modify the given **Hawkins** and **DarkDimension** classes.

Tester Code

```java
public class HawkinsLabTester {
  public static void main(String[] args) {
    Hawkins place1 = new Hawkins("Hawkins Lab");
    Hawkins place2 = new Hawkins("Palace Arcade");
    UpsideDown gate1 = new UpsideDown("The Nina Project");
    UpsideDown gate2 = new UpsideDown("Brimborn Steel Works");
    DarkDimension world = new DarkDimension("The Dark World");
    gate1.open();
    System.out.println("Total bridges: " +
UpsideDown.totalBridges);
    System.out.println("======== [1] ========");
    gate1.connect(place1);
    gate1.connect(place2);
    Hawkins place3 = new Hawkins("Starcourt Mall");
    gate2.connect(place3);
    gate1.details();
    gate2.details();
    System.out.println("======== [2] ========");
    world.runExperiment(gate1);
    world.runExperiment(gate2);
    System.out.println("======== [3] ========");
    System.out.println("Total bridges: " +
UpsideDown.totalBridges);
    System.out.println("======== [4] ========");
    Hawkins place4 = new Hawkins("Byers new house");
    gate1.connect(place4);
    gate1.disconnect(2);
    gate2.disconnect(3);
    System.out.println("======== [5] ========");
    gate1.details();
  }
}
```

Expected Output

```
Bridge from The Nina Project is Open
Total bridges: 0
======== [1] ========
The Nina Project Details:
Bridge 1: Hawkins Lab
Bridge 2: Palace Arcade
Brimborn Steel Works Details:
Bridge 1: Starcourt Mall
======== [2] ========
Bridge present at The Nina Project
Activating the door of Hawkins Lab
Experiment executed successfully!
No Bridge present at Brimborn Steel
Works
Cannot run experiment.
======== [3] ========
Total bridges: 3
======== [4] ========
No further bridges with The Nina Project
Invalid bridge number!
======== [5] ========
The Nina Project Details:
Bridge 1: Hawkins Lab
```

```java
// Grand Parent Class
class Hawkins{
    public String name;
    public boolean status=false;

    public Hawkins(String name) {
        this.name = name;
    }

    public boolean checkBridge(Hawkins h) {
        if (h.status==true) {
            System.out.println("Bridge present at " + h.name);
            return true;
        } else {
            System.out.println("No Bridge present at " + h.name);
            return false;
        }
    }

    public void open() {
      if (status==false){
        status = true;
        System.out.println("Bridge from "+name+" is Open");
      }
    }
}
```

```java
// Parent Class
class UpsideDown extends Hawkins{
     // Write Your Code Here
}
```

```java
// Child Class
class DarkDimension extends UpsideDown {
  public DarkDimension(String name) {
    super(name);
  }

  public void runExperiment(UpsideDown portal) {
    if (!this.checkBridge(portal)) {
      System.out.println("Cannot run experiment.");
    }
    else {
      if (portal.getBridge1() != null) {
        portal.activate(portal.getBridge1());
        System.out.println("Experiment executed successfully!");
      } else if (portal.getBridge2() != null) {
        portal.activate(portal.getBridge2());
        System.out.println("Experiment executed successfully!");
      } else {
        System.out.println("No experiment found!");
      }
    }
  }
}
```

Task 3

Write the Garage, Bike and Car class. **Car, Bike** are child classes of **Vehicle** class. But **Garage** is neither a parent nor a child class. The Garage class has **two arrays as instance variables** called *cars* and *bikes* that can store **Car and Bike objects**.

Hint: In this task you'll need to use the **instanceof** keyword and **downcasting**.

Parent Class

```
public class Vehicle {

    private String brand;
    private int year, wheels;

    public Vehicle(String b, int y){
        this.brand = b;
        this.year = y;
    }

    public String getBrand(){
        return this.brand;
    }

    public int getYear(){
        return this.year;
    }

    public void setWheels( int w ){
        this.wheels = w;
    }

    public int getWheels(){
        return this.wheels;
    }

    public String toString(){
        return "Brand: "+this.brand+", Year: "+this.year+", Wheels: "+this.wheels;
    }

}
```

DRIVER CODE	**OUTPUT**

Garage g = new Garage(2, 3);
System.out.println("==========0===========");
Vehicle vC1 = new Car("Ford", "Mustang", 2022, 2, 4,
false);
Vehicle vC2 = new Car("Tesla", "Model S", 2025, 4, 4,
true);
Vehicle vC3 = new Car("Reliant", "Robin", 1981, 2, 3,
false);
System.out.println("==========1===========");
System.out.println(vC1);
System.out.println("==========2===========");
g.addVehicle(vC1);
g.addVehicle(vC2);
g.addVehicle(vC3);
System.out.println(g.cars[1]);
System.out.println("==========3===========");
g.cars[0].startAutoPilot();
g.cars[1].startAutoPilot();
System.out.println("==========4===========");
Vehicle vB1 = new Bike("Honda", "Gold Wing", 2022, 3,
true);
System.out.println(vB1);
g.addVehicle(vB1);
System.out.println("==========5===========");
Vehicle vB2 = new Bike("Royal Enfield", "Classic 350",
2021, 2, false);
g.addVehicle(vB2);
System.out.println(g.bikes[1]);
System.out.println("==========6===========");
Vehicle vB3 = new Bike("Harley-Davidson", "Street 750",
2022, 2, false);
g.addVehicle(vB3);
Vehicle vB4 = new Bike("Yamaha", "MT-15", 2023, 2,
false);
g.addVehicle(vB4);
System.out.println("=========7============");
g.bikes[0].doAWheelie();
g.bikes[1].doAWheelie();
	Welcome to the Garage!
	Car Capacity: 2
	Bike Capacity: 3
	==========0===========
	==========1===========
	Car Brand: Ford, Year: 2022, Wheels:
	4, Model: Mustang, Doors: 2, AI: false
	==========2===========
	A Ford CAR has been added to the
	Garage
	A Tesla CAR has been added to the
	Garage
	Can't add more Cars! Capacity: 2
	Car Brand: Tesla, Year: 2025, Wheels:
	4, Model: Model S, Doors: 4, AI: true
	==========3===========
	Ford:Mustang has NO AutoPilot
	Tesla:Model S AutoPilot Started
	==========4===========
	Bike Brand: Honda, Year: 2022, Wheels:
	3, Model: Gold Wing, SideCar: true
	A Honda BIKE has been added to the
	Garage
	==========5===========
	A Royal Enfield BIKE has been added to
	the Garage
	Bike Brand: Royal Enfield, Year: 2021,
	Wheels: 2, Model: Classic 350,
	SideCar: false
	==========6===========
	A Harley-Davidson BIKE has been added
	to the Garage
	Can't add more bikes! Capacity: 3
	=========7============
	Wheelie Failed. Honda:Gold Wing has
	SideCar
	Royal Enfield:Classic 350 is doing
	Wheelie!!

Task 4

```java
public class Sue {
 void method1() {
   System.out.println("sue 1");
 }
 void method3() {
   System.out.println("sue 3");
 }
}

public class Blue {
 void method1() {
   System.out.println("blue 1");
   method3();
 }
 void method3() {
   System.out.println("blue 3");
 }
}

public class Moo extends Blue {
 void method2() {
   super.method3();
   System.out.println("moo 2");
   this.method3();
 }
 void method3() {
   System.out.println("moo 3");
 }
}

public class Crew extends Moo {
 void method1() {
   System.out.println("crew 1");
 }
 void method3() {
   System.out.println("crew 3");
 }
}
```

Assuming the following variables have been defined:

```
Moo var1 = new Crew();
Blue var2 = new Moo();
Object var3 = new Sue();
Sue var4 = new Sue();
Blue var5 = new Crew();
Blue var6 = new Blue();
```

In the table below,

- The output produced by the statement in the left-hand column, should be written in the right-hand column
- If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c".
- If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

	Statement	**Output**
1	var1.method1();	
2	var2.method1();	
3	var3.method1();	
4	var4.method1();	
5	var5.method1();	
6	var6.method1();	
7	var1.method3();	
8	var2.method3();	
9	var3.method3();	
10	((Blue)var1).method1();	
11	((Crew)var1).method2();	
12	((Sue)var1).method3();	
13	((Blue)var3).method1();	
14	((Crew)var3).method1();	
15	((Sue)var3).method3();	
16	((Moo)var2).method2();	
17	((Crew)var3).method2();	
18	((Moo)var5).method2();	
19	((Moo)var6).method2();	
20	((Moo)var2).method1();	

Task 5

```
public class Foo {
    String name = "foo";
    public void call1() {
        System.out.println("Foo 1");
    }
    public void call2() {
        call1();
        System.out.println("Foo 2");
    }
}

public class Bar extends Foo {
    public void call2() {
        System.out.println("Bar 2");
    }
    public void call3() {
        System.out.println("Bar 3");
    }
}

public class Buzz extends Bar {
    String name = "Buzz";
    public void call1() {
        System.out.println("Buzz 1");
    }
    public void call4() {
        call3();
        System.out.println("Buzz 4");
    }
}
public class Bux extends Foo {
    String name = "Bux";
    public void call1() {
        System.out.println("Bux 1");
    }
    public void call3() {
        System.out.println("Bux 3");
    }
}
```

Assuming the following variables have been defined:

```
Foo foo1 = new Foo();
Bar bar1 = new Bar();
Bux bux1 = new Bux();
Foo foo2 = new Buzz();
Bar bar2 = new Buzz();
Object obj1 = new Foo();
```

In the table below,

- The output produced by the statement in the left-hand column, should be written in the right-hand column
- If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c".
- If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

	Statement	**Output**
1	bar1.call1();	
2	foo2.call1();	
3	foo2.call2();	
4	bar2.call3();	
5	System.out.println(bar1.name);	
6	System.out.println(bar2.name);	
7	System.out.println(((Buzz)bar2).name);	
8	((Buzz)bar1).call4();	
9	((Bar)foo1).call3();	
10	((Foo)bux1).call1();	
11	((Bux)foo1).call1();	
12	bux1.call1();	
13	bux1.call2();	
14	((Foo)foo2).call2();	
15	((Buzz)obj1).call3();	
16	((Buzz)obj1).call2();	
17	((Bux)foo2).call2();	
18	((Buzz)obj1).call1();	
19	System.out.println(foo2.name);	
20	System.out.println(((Bux)foo2).name);	