

# Lab Assignment 09



Inspiring Excellence

Course Code:	CSE111
Course Title:	Programming Language II
Topic:	Polymorphism
Number of Tasks:	10 (Classwork: 05, Homework: 05)

*[Submit all the Coding Tasks (Homework: Task 1 to 3) in the Google Form shared on buX before the next lab.]*

[You are not allowed to change the given codes of any of the tasks]

## CLASSWORK

### Task 1

1.a. Write the **Pokemon** class so that the following code generates the output below:

Driver Code	Output
<pre>public class PokemonTester{     public static void main(String[] args){         Pokemon pikachu = new Pokemon("Pikachu");         pikachu.attack();         pikachu.attack("Thunderbolt");         pikachu.attack("Iron Tail", 90);     } }</pre>	<pre>Pikachu attacks with a basic move! Pikachu uses Thunderbolt! Pokemon uses Iron Tail with power 90!</pre>

1.b. What type of polymorphism is depicted in the code above?

### Task 2

Write the **UsedBattery** and the **PlasticBottle** classes derived from **Waste** class so that the following code generates the output below.

You also need to **complete the SorterBot class** for this code to work. You cannot create any additional methods in the **SorterBot** class.

Tester	Output
<pre>public class RecyclingPlant {     public static void main(String[] args) {         SorterBot robot = new SorterBot();         UsedBattery bat1 = new UsedBattery("Duracell-X", true);         PlasticBottle bot1 = new PlasticBottle("Coke-Zero", 500);         Waste unknown = new Waste("Mystery-Box");         System.out.println("#####");         robot.processItem(bat1);         robot.processItem(bot1);         robot.processItem(unknown);         robot.processItem(new UsedBattery("Energizer", false));     } }  //Parent Class class Waste {     String id;      Waste(String id) {         this.id = id;     } }  //Disjoint Class class SorterBot {     public void processItem(Waste item) {         //WRITE YOUR CODE HERE         System.out.println("-----");     } }</pre>	<pre>##### Scanning Item Duracell-X [Type: BATTERY] Duracell-X is leaking. Sealing in concrete container. ----- Scanning Item Coke-Zero [Type: PLASTIC] Coke-Zero compressed from 500ml to flat disk. ----- Scanning Item Mystery-Box [Type: UNKNOWN] Item sent to generic incinerator. ----- Scanning Item Energizer [Type: BATTERY] Energizer stored in dry cell. -----</pre>

### Task 3

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

- Assume that each **StudyRoom** can add two books.
- You cannot use any arrays in the **StudyRoom** class.
- You should use the given **Library** and **SilentStudyRoom** classes' variables and methods as needed.
- You cannot modify the given **Library** and **SilentStudyRoom** classes.

Tester Code	Expected Output
<pre> public class TestLibrary{     public static void main(String[] args) {         Library library = new Library("The Mind Maze");         library.showRoomInfo();         System.out.println("===== 1 =====");         StudyRoom room9 = new StudyRoom("Study Hub");         SilentStudyRoom room9A = new SilentStudyRoom("Focus Room");         room9.addBook("Data Structures");         System.out.println("===== 2 =====");         room9.addBook("Operating Systems");         room9.showRoomInfo();         System.out.println("===== 3 =====");         System.out.println("Total books: " +StudyRoom.totalBooks);         System.out.println("===== Add Book =====");         room9A.addBook(library, room9);         System.out.println("===== 4 =====");         room9A.showRoomInfo();         System.out.println("Total books: " +StudyRoom.totalBooks);         System.out.println("===== Add Book Again =====");         room9A.addBook(library, room9);         System.out.println("===== 5 =====");         room9A.showRoomInfo();         System.out.println("Total books: " + StudyRoom.totalBooks);     } } </pre>	<pre> Library Name: The Mind Maze ===== 1 ===== ===== 2 ===== Study Hub Details: Book 1: Data Structures Book 2: Operating Systems ===== 3 ===== Total books: 2 ===== Add Book ===== Library Name: The Mind Maze Adding book: Data Structures Book 1 Removed from Study Hub Book added successfully! ===== 4 ===== Focus Room Details: Book 1: Data Structures No book added Total books: 1 ===== Add Book Again ===== Library Name: The Mind Maze Adding book: Operating Systems Book 2 Removed from Study Hub Book added successfully! ===== 5 ===== Focus Room Details: Book 1: Data Structures Book 2: Operating Systems Total books: 0 </pre>
<pre> // Grand Parent Class class Library{     public String name;      public Library(String name) {         this.name = name;     }      public void showRoomInfo() {         System.out.println("Library Name: " + name);     } } </pre>	

```
// Parent Class
```

```
class StudyRoom extends Library{  
    // Write Your Code Here  
}
```

```
// Child Class
```

```
class SilentStudyRoom extends StudyRoom{  
  
    public SilentStudyRoom(String name) {  
        super(name);  
    }  
  
    public void addBook(Library lib, StudyRoom room) {  
        lib.showRoomInfo();  
        if (room.getBook1() != null) {  
            System.out.println("Adding book: " + room.getBook1());  
            this.setBook1(room.getBook1());  
            room.removeBook(1);  
            System.out.println("Book added successfully!");  
        }  
        else if (room.getBook2() != null) {  
            System.out.println("Adding book: " + room.getBook2());  
            this.setBook2(room.getBook2());  
            room.removeBook(2);  
            System.out.println("Book added successfully!");  
        }  
        else {  
            System.out.println("No books available in "+room.name);  
        }  
    }  
}
```

## Task 4

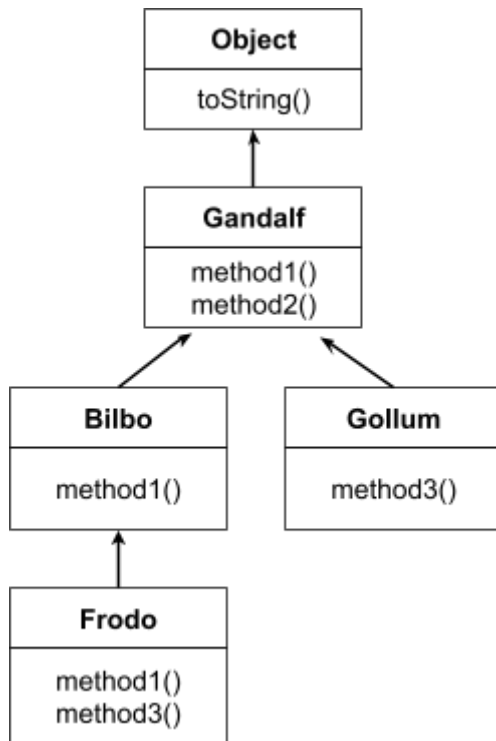
1	public class Gandalf {
2	public void method1(){
3	System.out.println("Gandalf 1");
4	}
5	
6	public void method2(){
7	System.out.println("Gandalf 2");
8	method1();
9	}
10	}
11	public class Bilbo extends Gandalf{
12	public void method1(){
13	System.out.println("Bilbo 1");
14	}
15	}
16	public class Gollum extends Gandalf{
17	public void method3(){
18	System.out.println("Gollum 3");
19	}
20	}
21	public class Frodo extends Bilbo{
22	public void method1(){
23	System.out.println("Frodo 1");
24	super.method1();
25	}
26	
27	public void method3(){
28	System.out.println("Frodo 3");
29	}
30	}

**Assuming the following variables have been defined:**

```
Gandalf var1 = new Frodo();
Gandalf var2 = new Bilbo();
Gandalf var3 = new Gandalf();
Object var4 = new Bilbo();
Bilbo var5 = new Frodo();
Object var6 = new Gollum();
```

A class diagram for the above classes has been added with this question.

**Note:** The diagram may not always be provided, so students should know how to draw it from the code.



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	var4.method1();	
4	var6.method1();	
5	var1.method2();	
6	var3.method2();	
7	var4.method2();	
8	var5.method2();	

9	var6.method2();	
10	((Frodo)var4).method3();	
11	((Frodo)var6).method2();	
12	((Gollum)var1).method3();	
13	((Gollum)var4).method1();	
14	((Gandalf)var1).method2();	
15	((Frodo)var4).method1();	
16	((Gollum)var6).method2();	
17	((Gandalf)var2).method1();	
18	((Bilbo)var6).method2();	
19	((Frodo)var1).method3();	
20	((Gandalf)var5).method3();	

## Task 5

1	public class Caramel extends SilkOreo{
2	String texture = "Softy";
3	public void method1() {
4	System.out.println("Caramel m1");
5	}
6	public void method4() {
7	System.out.println("Caramel m4");
8	}
9	public String toString(){
10	method2();
11	return "Caramel is "+ texture;
12	}
13	}
14	public class Chocolate{
15	String texture = "Chocolaty";
16	public void method1() {
17	method2();
18	System.out.println("Chocolate m1");
19	}
20	public void method2() {
21	System.out.println("Chocolate m2");
22	}
23	public String toString(){
24	method2();
25	return "Chocolate is "+ texture;
26	}
27	}
28	public class DairyMilk extends Chocolate{
29	String texture = "Yummy";
30	public void method2() {
31	System.out.println(this.texture);
32	System.out.println("DairyMilk m2");
33	}
34	public void method3() {
35	System.out.println("DairyMilk m3");
36	}
37	}
38	public class KitKat extends Chocolate{
39	String texture = "Crunchy";
40	public void method1() {
41	System.out.println("KitKat m1");
42	}
43	public void method4() {
44	System.out.println("KitKat m4");



45	}
46	public String toString(){
47	method2();
48	return "KitKat is "+ texture;
49	}
50	}
51	public class SilkOreo extends DairyMilk{
52	String texture = "Silky";
53	public void method1() {
54	super.method1();
55	System.out.println("SilkOreo m1");
56	}
57	public void method3() {
58	System.out.println("SilkOreo m3");
59	System.out.println(this);
60	}
61	}

**Assuming the following variables have been defined:**

```
Chocolate choco1 = new Chocolate();
KitKat kit = new KitKat();
DairyMilk dairyMilk1 = new DairyMilk();
DairyMilk dairyMilk2 = new SilkOreo();
Object obj1 = new DairyMilk();
Object obj2 = new KitKat();
Chocolate caramel1 = new Caramel();
```

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	choco1.method1();	
2	dairyMilk1.method1();	
3	dairyMilk2.method4();	
4	caramel1.method1();	

5	System.out.println(caramel1);	
6	System.out.println(caramel1.texture);	
7	((Chocolate)kit).method2();	
8	((SilkOreo)dairyMilk2).method3();	
9	((DairyMilk)kit).method2();	
10	((Chocolate)kit).method3();	
11	((Chocolate)dairyMilk2).method1();	
12	((Chocolate)obj1).method2();	
13	((Caramel)obj1).method2();	
14	((SilkOreo)obj2).method3();	
15	System.out.println(((Object)choco1).toString());	
16	System.out.println(((Chocolate)kit).texture);	

# HOMEWORK

## Task 1

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

Tester Code and Parent Class	Output
<pre>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);         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 &gt; 5) {</pre>	<pre>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 initiated. ERROR: Device must be ON to format. ===== Power Low: Indoor-Cam cannot start.</pre>

```
        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;
}
}
```

## 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	Expected Output
<pre> 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();     } } </pre>	<pre> 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 </pre>
<pre> // 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); </pre>	

```

        return false;
    }
}

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

```

```

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

```

```

// 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
<pre> 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(); </pre>	<pre> 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!! </pre>



## Task 4

1	public class Sue {
2	void method1() {
3	System.out.println("sue 1");
4	}
5	void method3() {
6	System.out.println("sue 3");
7	}
8	}
9	
10	public class Blue {
11	void method1() {
12	System.out.println("blue 1");
13	method3();
14	}
15	void method3() {
16	System.out.println("blue 3");
17	}
18	}
19	
20	public class Moo extends Blue {
21	void method2() {
22	super.method3();
23	System.out.println("moo 2");
24	this.method3();
25	}
26	void method3() {
27	System.out.println("moo 3");
28	}
29	}
30	
31	public class Crew extends Moo {
32	void method1() {
33	System.out.println("crew 1");
34	}
35	void method3() {
36	System.out.println("crew 3");
37	}
38	}

**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

1	public class Foo {
2	String name = "foo";
3	public void call1() {
4	System.out.println("Foo 1");
5	}
6	public void call2() {
7	call1();
8	System.out.println("Foo 2");
9	}
10	}
11	
12	public class Bar extends Foo {
13	public void call2() {
14	System.out.println("Bar 2");
15	}
16	public void call3() {
17	System.out.println("Bar 3");
18	}
19	}
20	
21	public class Buzz extends Bar {
22	String name = "Buzz";
23	public void call1() {
24	System.out.println("Buzz 1");
25	}
26	public void call4() {
27	call3();
28	System.out.println("Buzz 4");
29	}
30	}
31	public class Bux extends Foo {
32	String name = "Bux";
33	public void call1() {
34	System.out.println("Bux 1");
35	}
36	public void call3() {
37	System.out.println("Bux 3");
38	}
39	}

**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);	

## Ungraded Tasks (Optional)

(You don't have to submit the ungraded tasks)

### Task 1

Write the **Mango** and the **Jackfruit** classes derived from **Fruit** class so that the following code generates the output below:

Parent Class		
<pre>public class Fruit{     private boolean formalin = false;     private String name = "";     public Fruit(boolean formalin, String name){         this.formalin = formalin;         this.name = name;     }     public String getName(){         return name;     }     public boolean hasFormalin(){         return formalin;     } }</pre>		
Driver Code	Output	
<pre>public class FruitTester{     public static void testFruit(Fruit f){         System.out.println("----Printing Detail-----");         if(f.hasFormalin()){             System.out.println("Do not eat the "+f.getName()+".");             System.out.println(f);         }else{             System.out.println("Eat the "+f.getName()+".");             System.out.println(f);         }     }     public static void main(String [] args){         Mango m = new Mango();         testFruit(m);         Jackfruit j = new Jackfruit();         testFruit(j);     } }</pre>	<pre>----Printing Detail----- Do not eat the Mango. Mangos are bad for you ----Printing Detail----- Eat the Jackfruit. Jackfruits are good for you</pre>	

## Task 2

Write the **CSEStudent** and **CSE111Student** classes derived from **Student** class so that the following code generates the output below:

Parent Class	
<pre>public class Student{     public String msg = "I love BU";     public String shout(){         return msg;     } }</pre>	
Driver Code	Output
<pre>public class StudentTester{     public static void printShout(Student s){         System.out.println("-----");         System.out.println(s.msg);         System.out.println(s.shout());     }     public static void main(String [] args){         Student s = new Student();         CSEStudent cs = new CSEStudent();         CSE111Student cs111 = new CSE111Student();         System.out.println(s.msg);         System.out.println(cs.msg);         System.out.println(cs111.msg);         printShout(s);         printShout(cs);         printShout(cs111);     } }</pre>	<pre>I love BU I want to transfer to CSE I love Java Programming ----- I love BU I love BU ----- I love BU I want to transfer to CSE ----- I love BU I love Java Programming</pre>

### Task 3

Design a set of classes for a Fantasy Game Character System. There is a parent class called **Character**. From it, there are two different child classes: **Warrior** and **Mage**. Additionally, there is a subclass called **Paladin** that extends Warrior.

Each character has:

- name (String)
- level (int)

Lastly, you need to Override the .equals() method inside the parent class

Parent Class	
<pre>public class Character {     public String name;     public int level;      public Character(String name, int level) {         this.name = name;         this.level = level;     }      public void specialMove() {         System.out.println("Character uses a generic move.");     }      // Override the .equals() method }</pre>	
Driver Code	Output
<pre>public class GameTester {     public static void main(String[] args) {         Character c1 = new Paladin("Arthur", 10);         Character c2 = new Mage("Merlin", 12);         Character c3 = new Warrior("Leon", 10);          c1.specialMove();         c2.specialMove();         c3.specialMove();          if (c1 instanceof Paladin) {             Paladin p = (Paladin) c1;             p.specialMove();         }          Warrior w1 = new Warrior("Leon", 10);         System.out.println("c3 equals w1? " + c3.equals(w1));          Mage m1 = new Mage("Merlin", 15);         System.out.println("c2 equals m1? " + c2.equals(m1));     } }</pre>	<pre>Arthur unleashes a holy strike! Merlin casts a powerful fireball! Leon performs a heavy sword slash! Arthur unleashes a holy strike! c3 equals w1? true c2 equals m1? false</pre>

## Task 4

Write the **PlatinumCard** and **SignatureCard** classes derived from **CreditCard** class so that the following code generates the output below.

**Note:** Platinum card users initially have 100 reward points and will get 2 reward points for spending 100 taka each. Signature card users initially have 200 reward points and will get 4 reward points for spending 100 taka each. Signature card users are allowed to bring upto 5 companions at lounges.

Parent Class	
<pre>public class CreditCard {     public String cardHolder;     public String accountNo;     public int rewardPoints;     public CreditCard(String cardHolder, String accountNo, int rewardPoints){         this.cardHolder = cardHolder;         this.accountNo = accountNo;         this.rewardPoints = rewardPoints;     }     public void cardDetails(){         System.out.println("Card Holder Name: " + cardHolder);         System.out.println("Account Number: " + accountNo);         System.out.println("Reward point gained: " + rewardPoints);     } }</pre>	
Driver Code	Output
<pre>public class CardTester {     public static void main(String[] args) {         CreditCard card1 = new PlatinumCard("Ali", "345 127");         CreditCard card2 = new SignatureCard("Rahul", "514 123");         CreditCard card3 = new SignatureCard("Rohan", "147 965");         CreditCard [] cards = {card1, card2, card3};         for (int i = 0; i&lt;cards.length; i++)         {             System.out.println("=====");             if (cards[i] instanceof SignatureCard)             {                 SignatureCard new_card = (SignatureCard) cards[i];                 new_card.spendCash(500);             }             else if (cards[i] instanceof PlatinumCard)             {                 PlatinumCard new_card = (PlatinumCard) cards[i];                 new_card.spendCash(200);             }             System.out.println("=====");             cards[i].cardDetails();         }     } }</pre>	<pre>===== Previous Reward Points: 100 Reward points after spending 200 taka: 104 ===== Card Holder Name: Ali Account Number: 345 127 Reward point gained: 104 ===== Previous Reward Points: 200 Reward points after spending 500 taka: 220 ===== Card Holder Name: Rahul Account Number: 514 123 Reward point gained: 220 Possible Number of Companions for Lounge: 5 ===== Previous Reward Points: 200 Reward points after spending 500 taka: 220 ===== Card Holder Name: Rohan Account Number: 147 965 Reward point gained: 220 Possible Number of Companions for Lounge: 5</pre>