AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956
SCHOOL OF COMPUTING

PAMISHETTY KEERMANI

CH.SC.U4CSE24113

# OBJECT ORIENTED PROGRAMMING (23CSE111)

LAB RECORD

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING, CHENNAI

## <u>BONAFIDE CERTIFICATE</u>

This is to certify that the Lab Record work for 23CSE111-Object Oriented Programming Subject submitted by **<u>CH.SC.U4CSE24135 – PAMISHETTY KEERMANI</u>** in "Computer Science and Engineering" is a Bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Chennai.

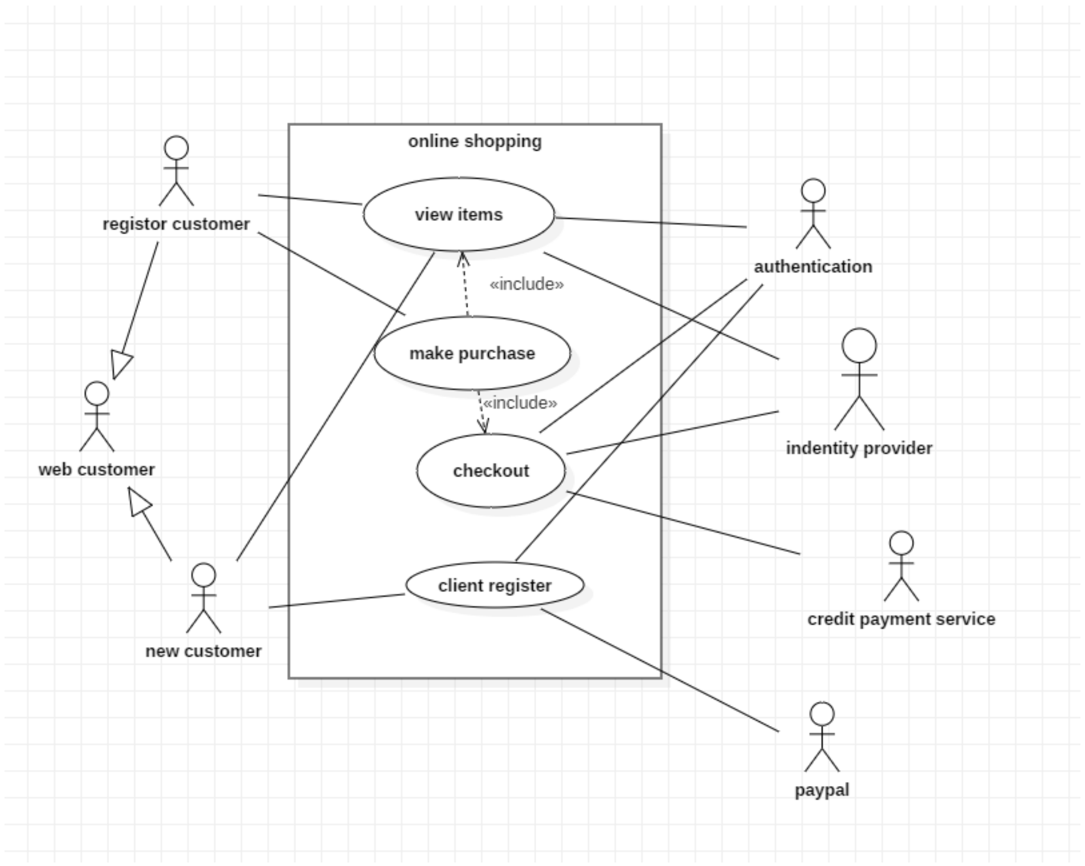
This Lab examination on held on

Internal Examiner 1                    Internal Examiner 2

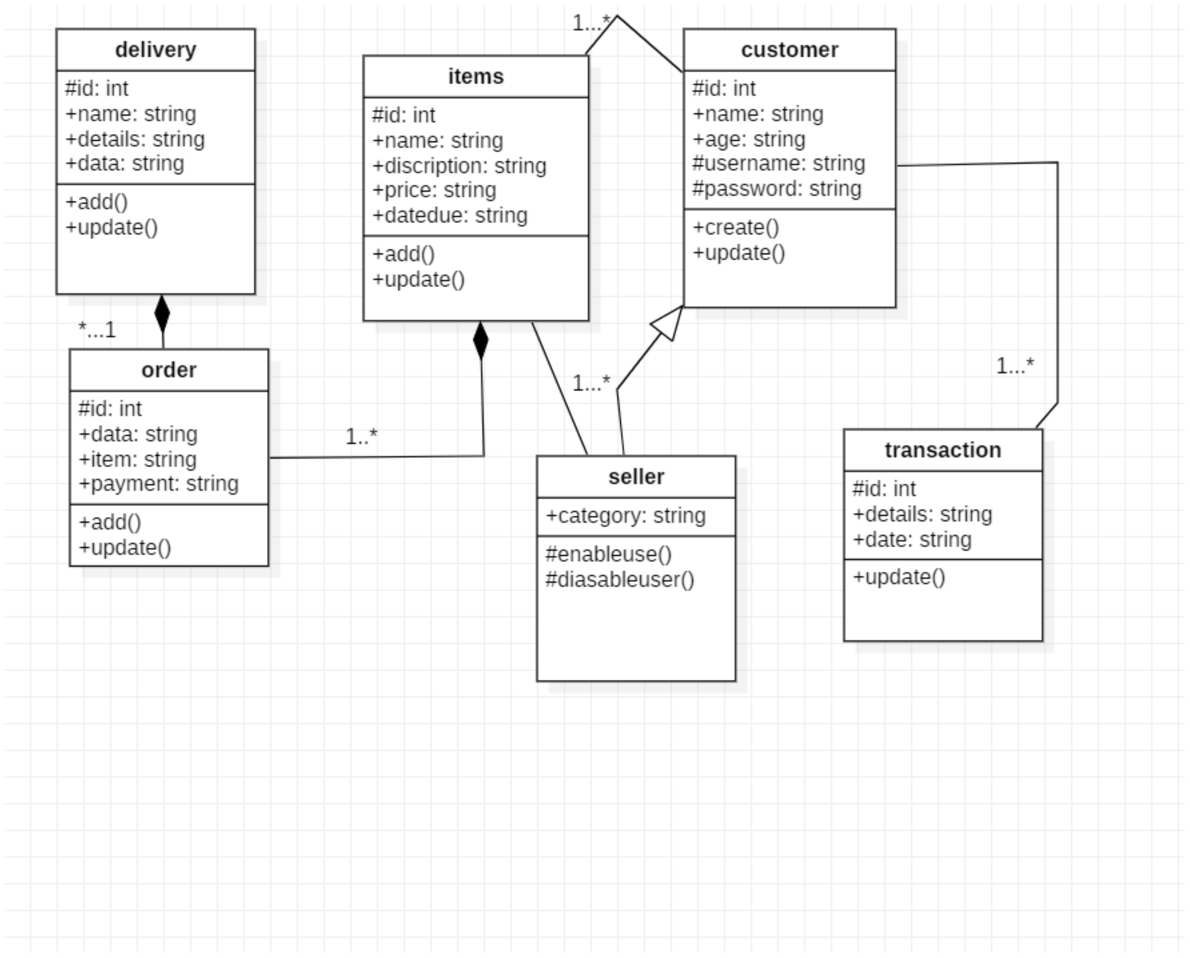
# UML DIAGRAMS

ONLINE SHOPPING

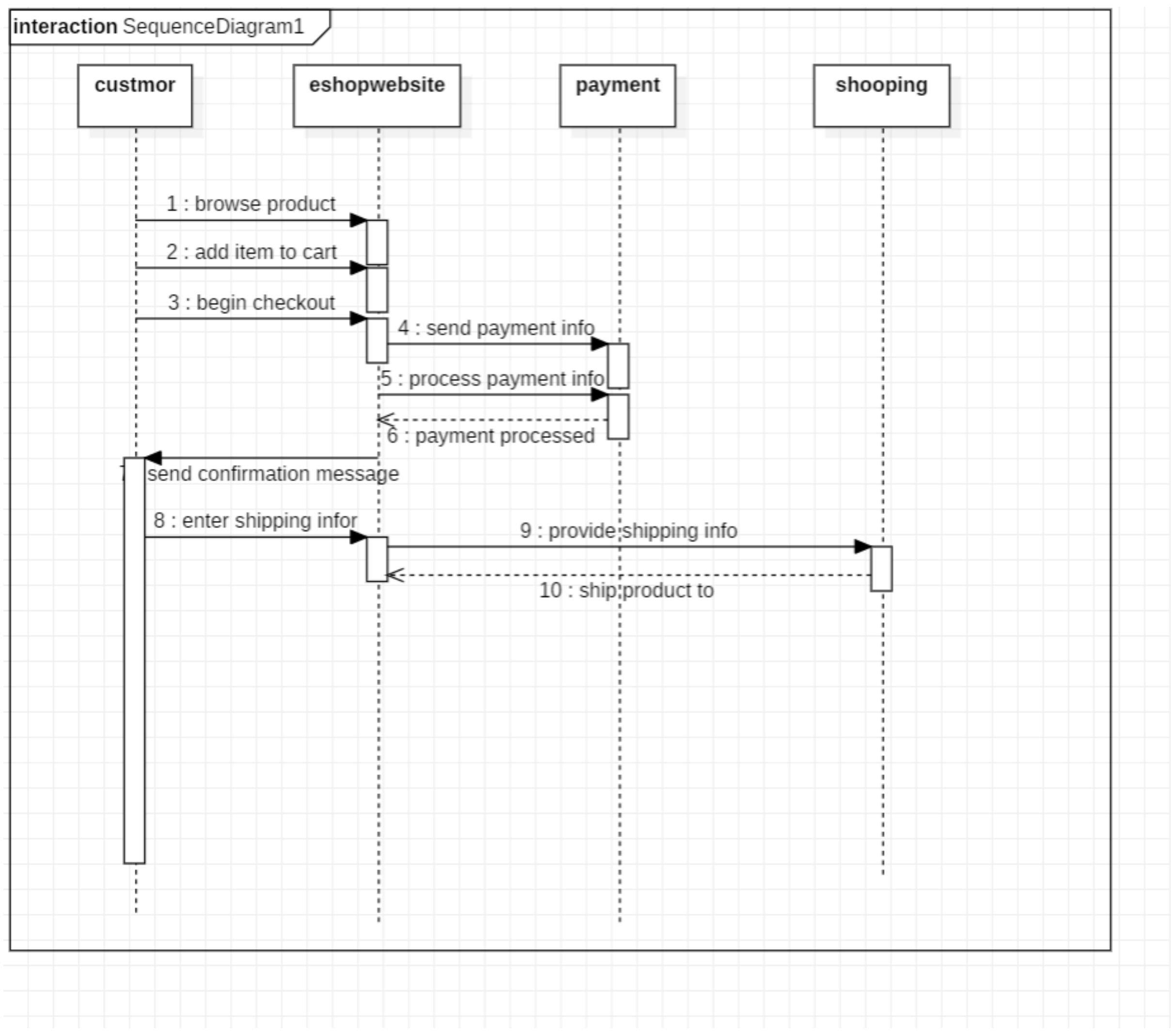
## 1 A) USE CASE DIAGRAM:



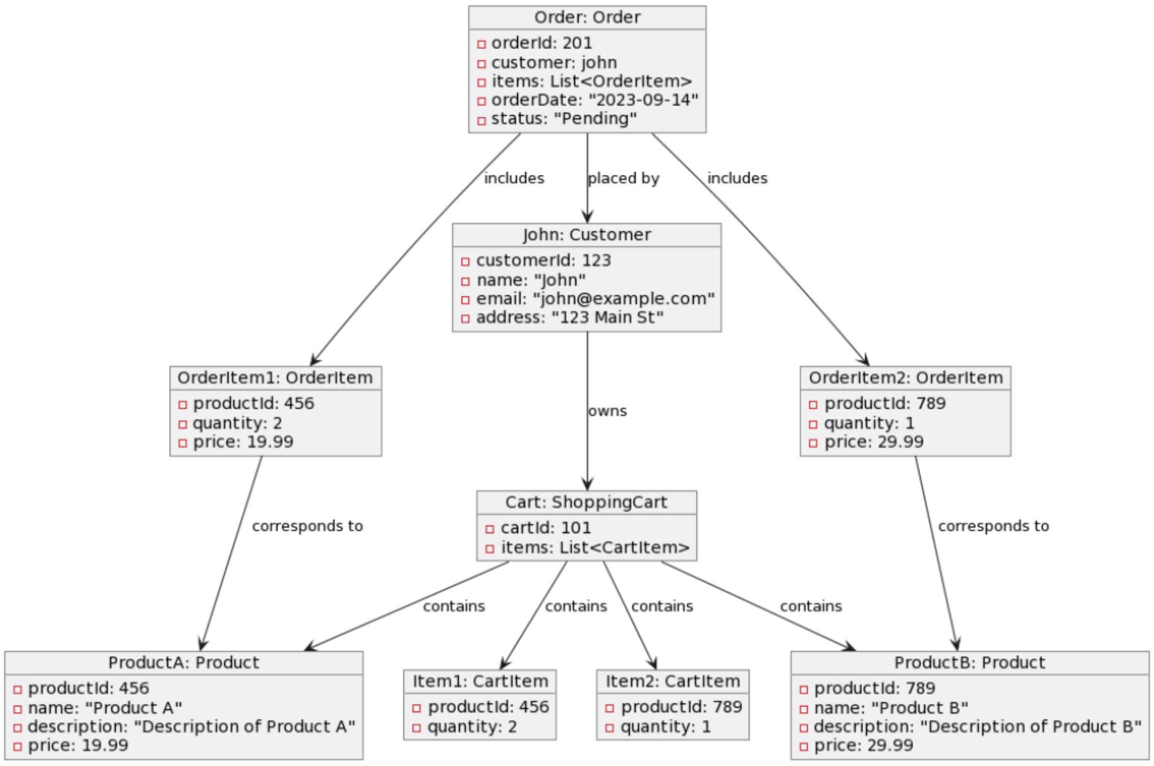
## 1 B) CLASS DIAGRAM:



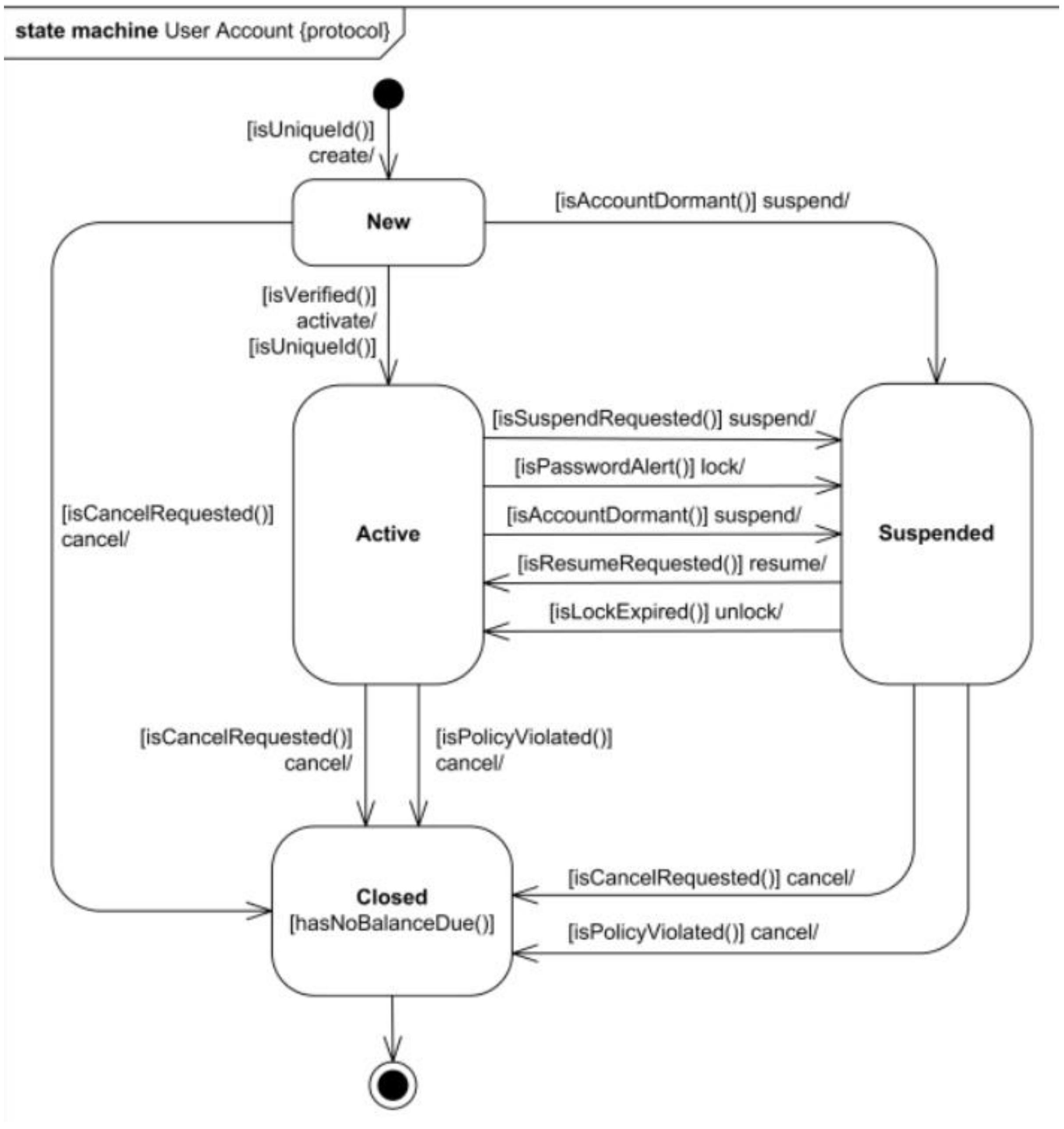
## 1 C) SEQUENCE DIAGRAM:



## 1 D) OBJECT DIAGRAM:

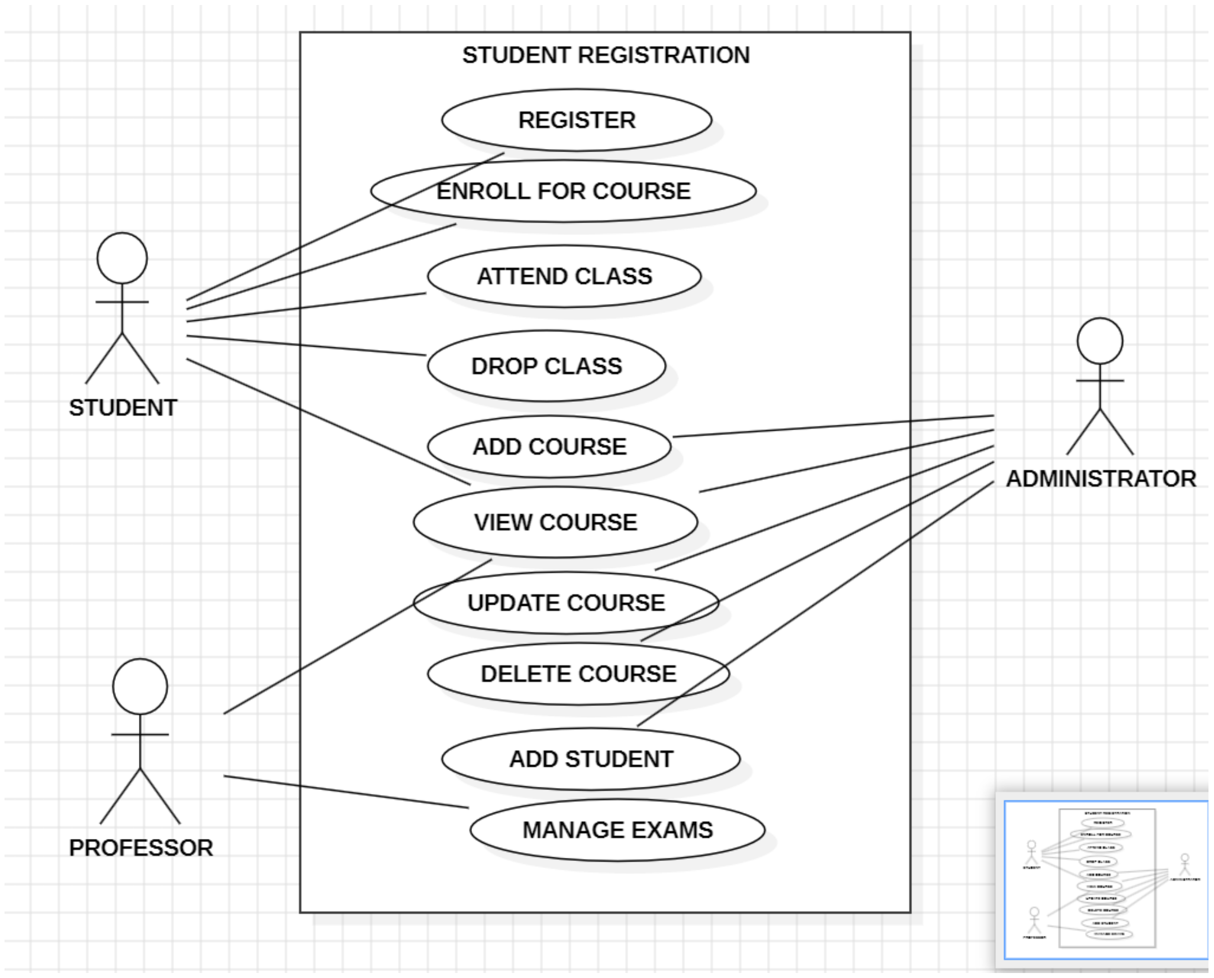


## 1 E) STATE ACTIVITY DIAGRAM:

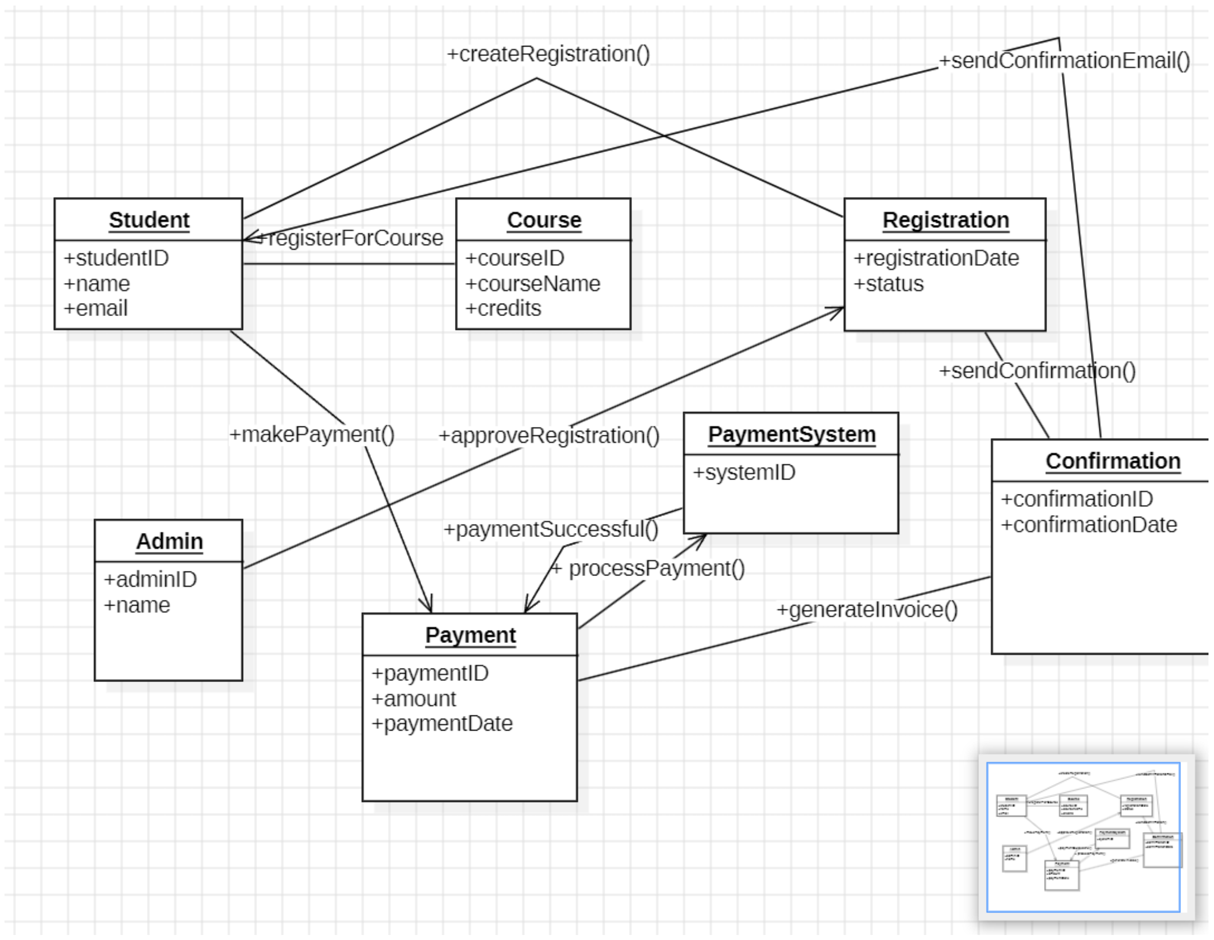

state machine User Account {protocol}
[isUniqueId()] create/
New
[isAccountDormant()] suspend/
[isVerified()] activate/ [isUniqueId()]
Active
Suspended
[isSuspendRequested()] suspend/
[isPasswordAlert()] lock/
[isAccountDormant()] suspend/
[isResumeRequested()] resume/
[isLockExpired()] unlock/
[isCancelRequested()] cancel/
[isCancelRequested()] cancel/
[isPolicyViolated()] cancel/
Closed [hasNoBalanceDue()]
[isCancelRequested()] cancel/
[isPolicyViolated()] cancel/

# STUDENT REGISTRATION

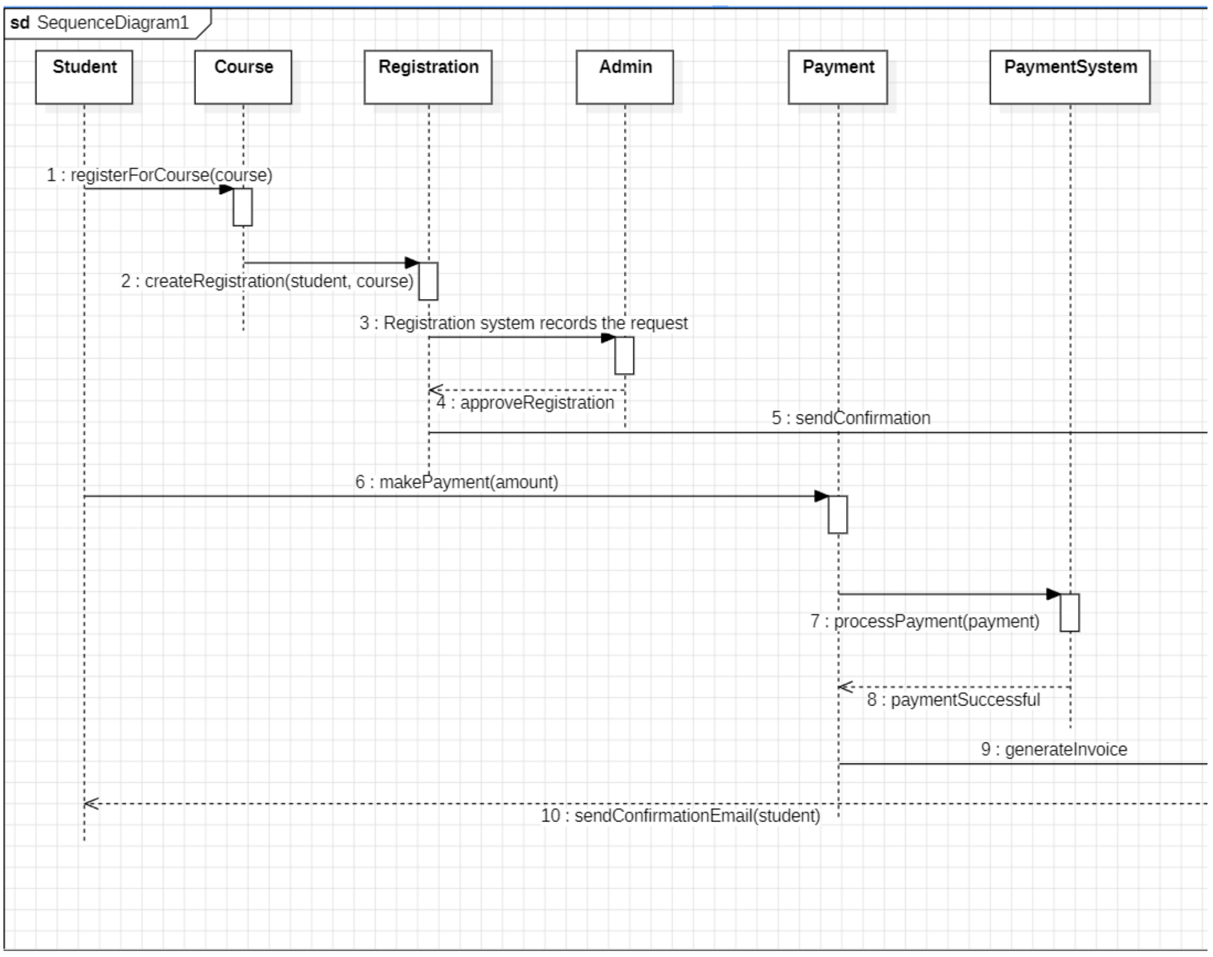
## 1 A) USE CASE DIAGRAM:



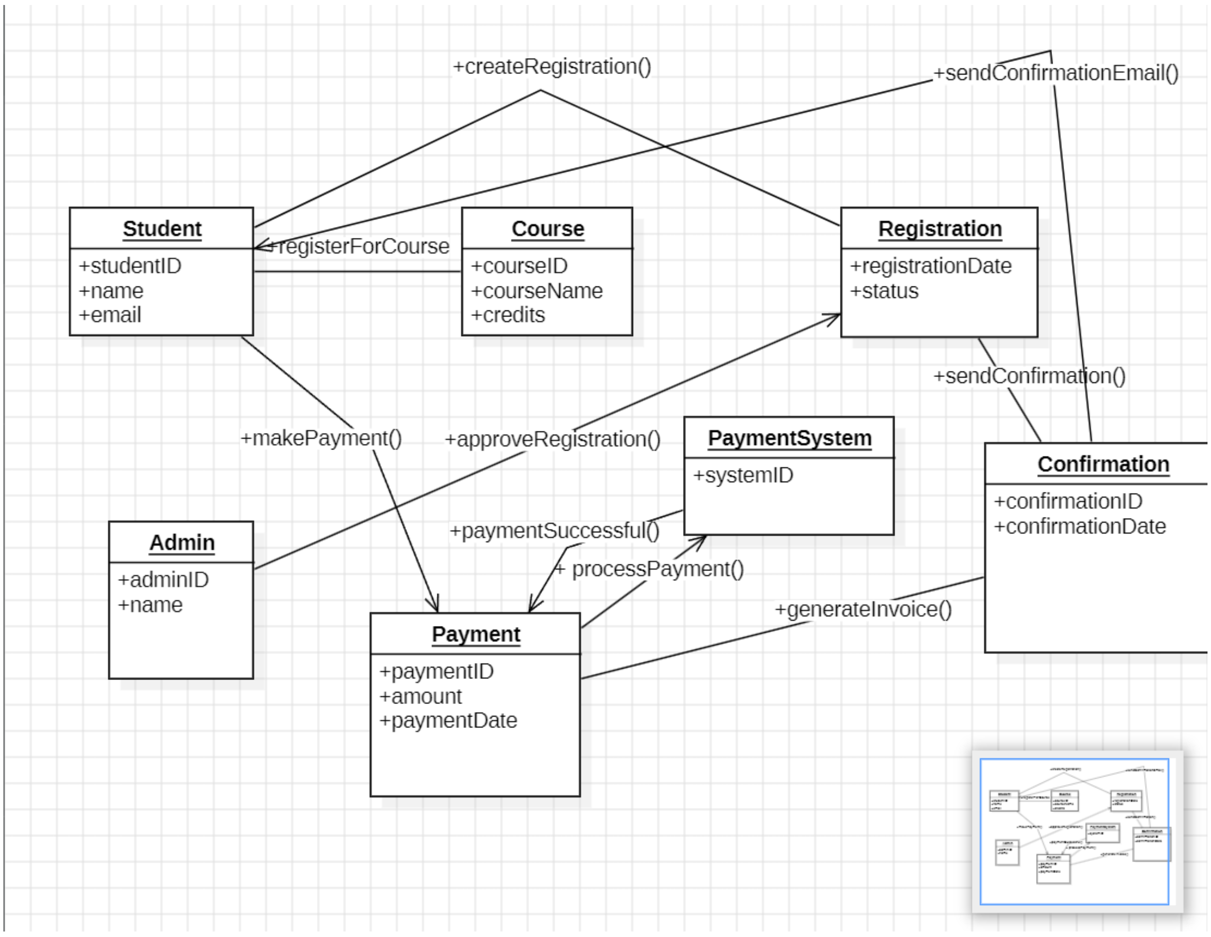
## 1 B) CLASS DIAGRAM:



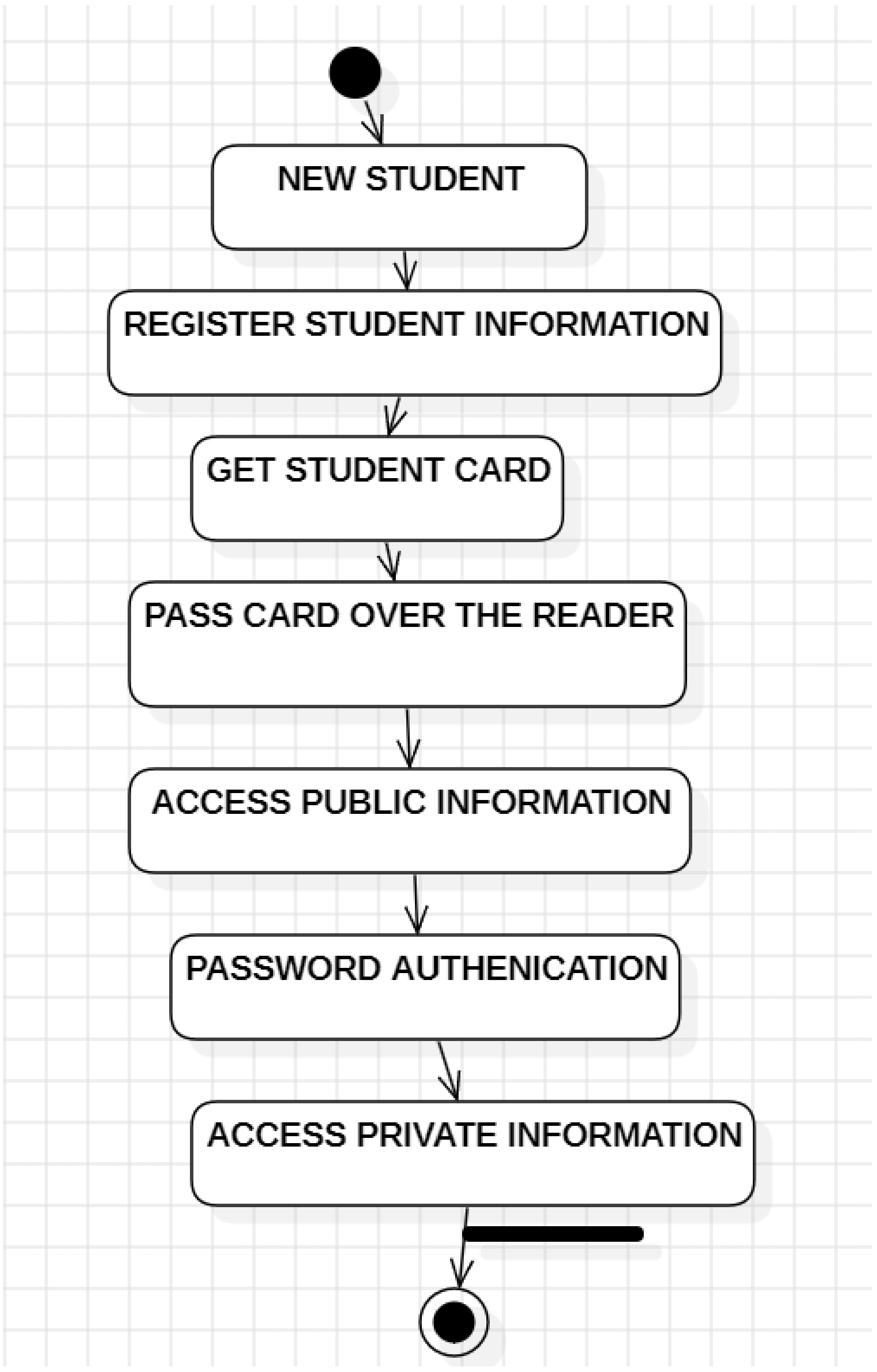
## 1 C) SEQUENCE DIAGRAM:



## 1 D) OBJECT DIAGRAM



## 1 E) STATE DIAGRAM

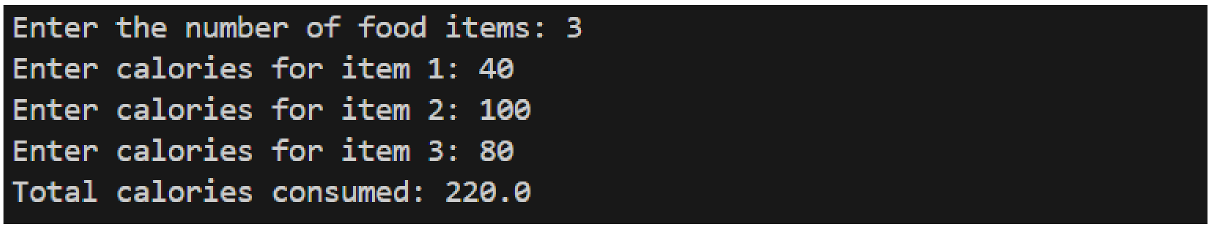

NEW STUDENT
REGISTER STUDENT INFORMATION
GET STUDENT CARD
PASS CARD OVER THE READER
ACCESS PUBLIC INFORMATION
PASSWORD AUTHENICATION
ACCESS PRIVATE INFORMATION

## 3.BASIC JAVA PROGRAMS

### 3 A) CALORIE COUNTER:

```
import java.util.Scanner;
public class CalorieCounter {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 System.out.print("Enter the number of food items: ");
 int items = scanner.nextInt();

 double totalCalories = 0;

 for (int i = 1; i <= items; i++) {
 System.out.print("Enter calories for item " + i + ": ");
 double calories = scanner.nextDouble();
 totalCalories += calories;
 }

 System.out.println("Total calories consumed: " + totalCalories);

 scanner.close();
 }
}
```

OUTPUT:

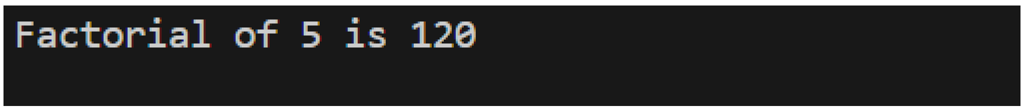


```
Enter the number of food items: 3
Enter calories for item 1: 40
Enter calories for item 2: 100
Enter calories for item 3: 80
Total calories consumed: 220.0
```

3 B) FACTORIAL:

```
public class Factorial {
 public static void main(String[] args) {
 int number = 5;
 int factorial = 1;
 for (int i = 1; i <= number; i++) {
 factorial *= i;
 }
 System.out.println("Factorial of " + number + " is " + factorial);
 }
}
```

OUTPUT:

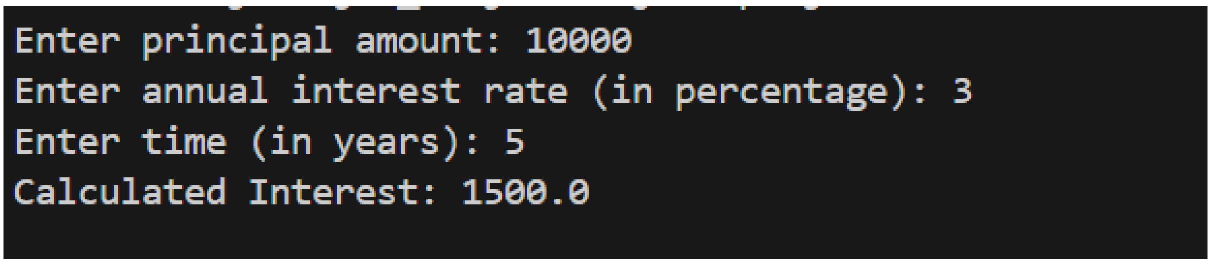


```
Factorial of 5 is 120
```

3 C) INTEREST CALCULATOR:

```
import java.util.Scanner;
public class InterestCalculator {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);

 System.out.print("Enter principal amount: ");
 double principal = scanner.nextDouble();

 System.out.print("Enter annual interest rate (in percentage): ");
 double rate = scanner.nextDouble();

 System.out.print("Enter time (in years): ");
 double time = scanner.nextDouble();

 double interest = (principal * rate * time) / 100;

 System.out.println("Calculated Interest: " + interest);

 scanner.close();
 }
}
```

OUTPUT:

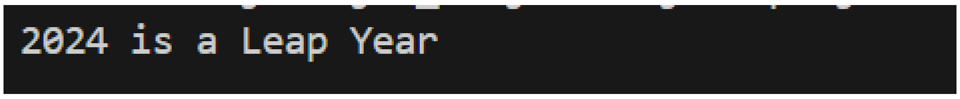


```
Enter principal amount: 10000
Enter annual interest rate (in percentage): 3
Enter time (in years): 5
Calculated Interest: 1500.0
```

## 3 D) LEAP YEAR CHECK:

```
public class LeapYearCheck {
 public static void main(String[] args) {
 int year = 2024;
 if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
 System.out.println(year + " is a Leap Year");
 }
else {
 System.out.println(year + " is NOT a Leap Year");
 }
}
}
```

OUTPUT:

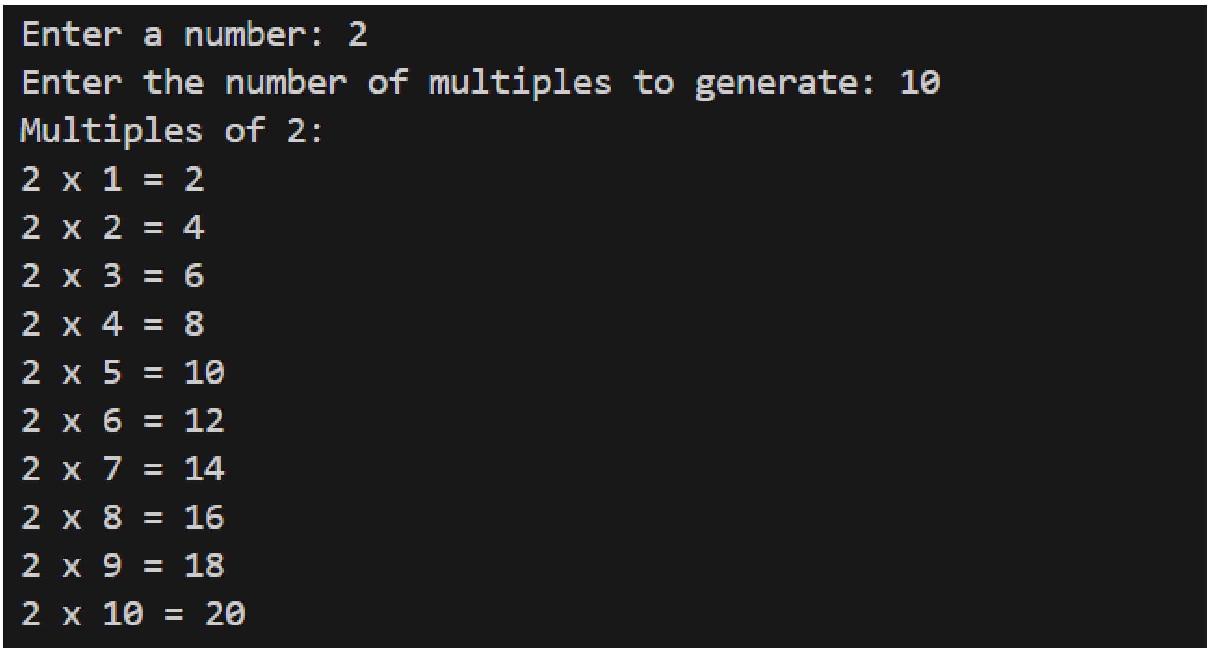


```
2024 is a Leap Year
```

## 3 E) MULTIPLICATION CALCULATOR:

```
import java.util.Scanner;
public class MultiplicationCalculator {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);


 System.out.print("Enter a number: ");
 int number = scanner.nextInt();

 System.out.print("Enter the number of multiples to generate: ");
 int multiplesCount = scanner.nextInt();

 System.out.println("Multiples of " + number + ":");
 for (int i = 1; i <= multiplesCount; i++) {
 System.out.println(number + " x " + i + " = " + (number * i));
 }

 scanner.close();
 }
}
```

OUTPUT:

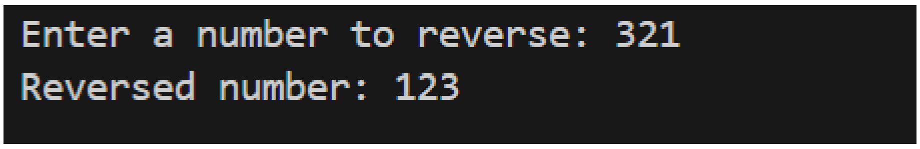


```
Enter a number: 2
Enter the number of multiples to generate: 10
Multiples of 2:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

3 F) NUMBER REVERSE:

```
import java.util.Scanner;
public class NumberReverser {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);

 System.out.print("Enter a number to reverse: ");
 int number = scanner.nextInt();

 int reversedNumber = 0;
 while (number != 0) {
 int digit = number % 10;
 reversedNumber = reversedNumber * 10 + digit;
 number /= 10;
 }

 System.out.println("Reversed number: " + reversedNumber);

 scanner.close();
 }
}
```

OUTPUT:

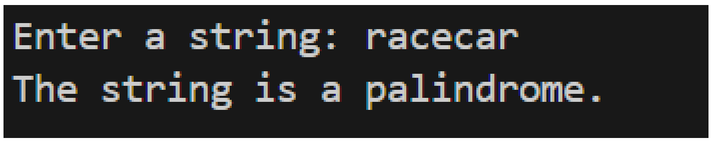


```
Enter a number to reverse: 321
Reversed number: 123
```

3 G) PALINDROME CHECKER:

```
public class PalindromeChecker {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);

 System.out.print("Enter a string: ");
 String input = scanner.nextLine();

 if (isPalindrome(input)) {
 System.out.println("The string is a palindrome.");
 }
 else {
 System.out.println("The string is not a palindrome.");
 }

 scanner.close();
 }

 public static boolean isPalindrome(String str) {
 str = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
 int left = 0, right = str.length() - 1;

 while (left < right) {
 if (str.charAt(left) != str.charAt(right)) {
 return false;
 }
 left++;
 right--;
 }

 return true;
 }
}
```

OUTPUT:

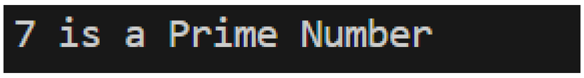


```
Enter a string: racecar
The string is a palindrome.
```

3 H) PRIME CHECK:

```
public class PrimeCheck {
 public static void main(String[] args) {
 int number = 7;
 boolean isPrime = true;
 if (number <= 1) {
 isPrime = false;
 } else {
 for (int i = 2; i <= number / 2; i++) {
 if (number % i == 0) {
 isPrime = false;
 break;
 }
 }
 }
 if (isPrime)
 System.out.println(number + " is a Prime Number");
 else
 System.out.println(number + " is not a Prime Number");
 }
}
```

OUTPUT:

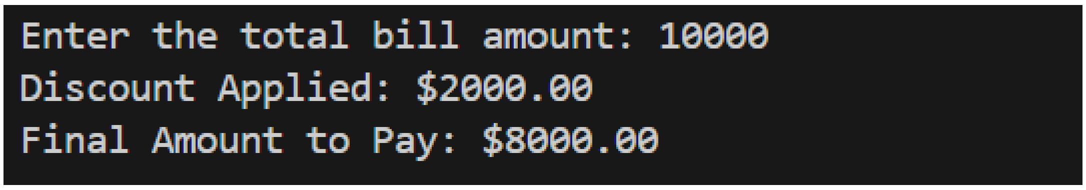


```
7 is a Prime Number
```

## 3 I) SHOPPING DISCOUNT:

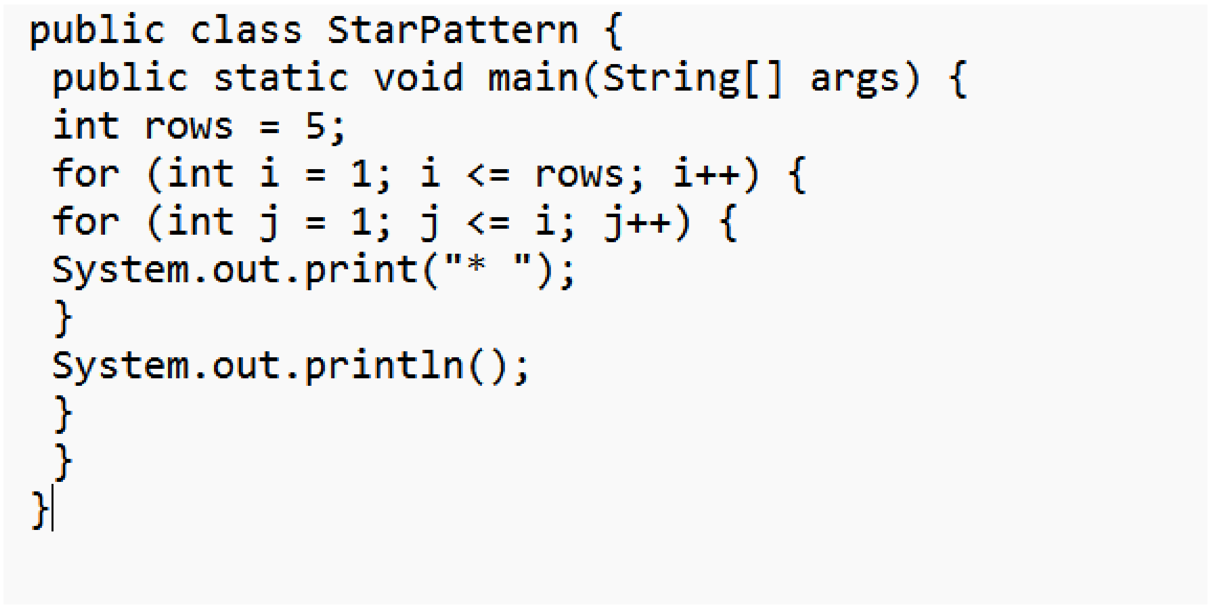
```
import java.util.Scanner;
public class ShoppingDiscount {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);


 System.out.print("Enter the total bill amount: ");
 double billAmount = scanner.nextDouble();

 if (billAmount < 0) {
 System.out.println("Invalid bill amount. Please enter a positive number.");
 } else {
 double discount;

 if (billAmount >= 500) {
 discount = billAmount * 0.20;
 } else if (billAmount >= 200) {
 discount = billAmount * 0.10;
 } else {
 discount = billAmount * 0.05;
 }

 double finalAmount = billAmount - discount;

 System.out.printf("Discount Applied: $%.2f%n", discount);
 System.out.printf("Final Amount to Pay: $%.2f%n", finalAmount);
 }

 scanner.close();
 }
}
```

## OUTPUT:



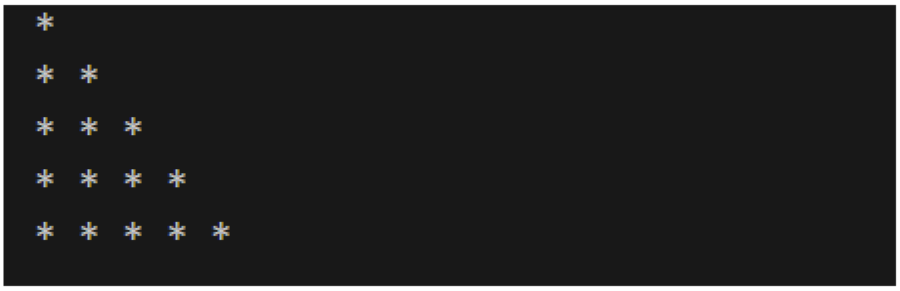
```
Enter the total bill amount: 10000
Discount Applied: $2000.00
Final Amount to Pay: $8000.00
```

3 J) STAR PATTERN:



```
public class StarPattern {
 public static void main(String[] args) {
 int rows = 5;
 for (int i = 1; i <= rows; i++) {
 for (int j = 1; j <= i; j++) {
 System.out.print("* ");
 }
 System.out.println();
 }
 }
}
```

OUTPUT:



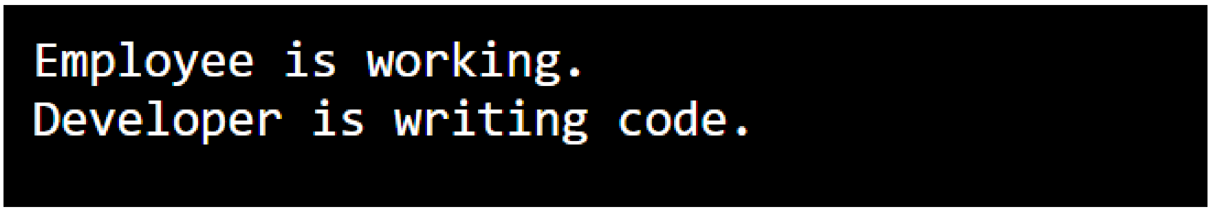
```
*
* *
* * *
* * * *
* * * * *
```

## 4. SINGLE INHERITANCE PROGRAMS

### 4 A) EMPLOYEE-DEVELOPER

```java
class Employee {
    void work() {
        System.out.println("Employee is working.");
    }
}

class Developer extends Employee {
    void code() {
        System.out.println("Developer is writing code.");
    }
}

public class Main{
    public static void main(String[] args) {
        Developer dev = new Developer();
        dev.work();
        dev.code();
    }
}
```

OUTPUT:

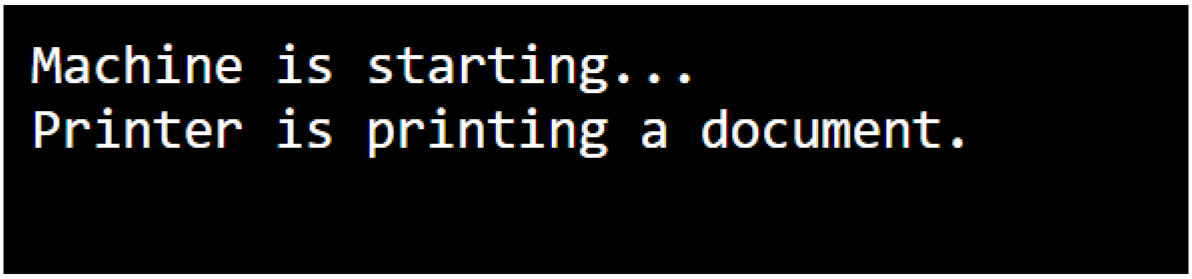


```
Employee is working.
Developer is writing code.
```

## 4 B) MACHINE-PRINTER

```java
class Machine {
    void start() {
        System.out.println("Machine is starting...");
    }
}

class Printer extends Machine {
    void printDocument() {
        System.out.println("Printer is printing a document.");
    }
}

public class SingleInheritance2 {
    public static void main(String[] args) {
        Printer p = new Printer();
        p.start();          // Inherited method
        p.printDocument(); // Own method
    }
}
```

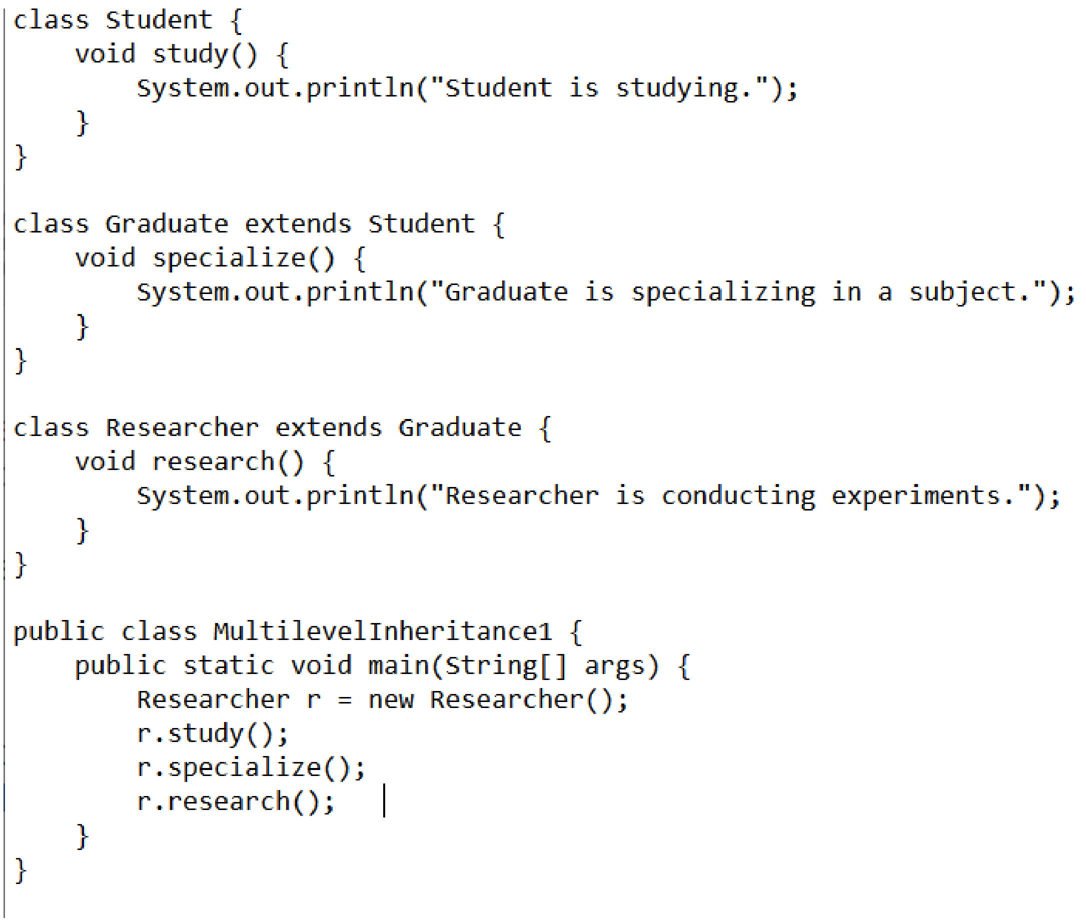
OUTPUT:



```
Machine is starting...
Printer is printing a document.
```

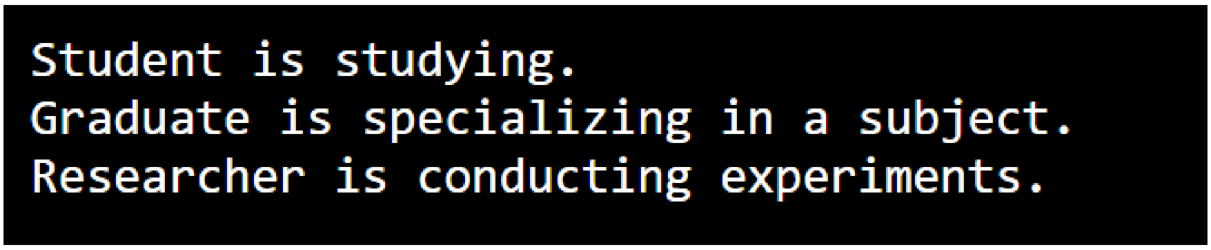
# 5. MULTILEVEL INHERITANCE PROGRAMS

## 5 A) STUDENT-GRADUATE-RESEARCHER



```java
class Student {
    void study() {
        System.out.println("Student is studying.");
    }
}

class Graduate extends Student {
    void specialize() {
        System.out.println("Graduate is specializing in a subject.");
    }
}

class Researcher extends Graduate {
    void research() {
        System.out.println("Researcher is conducting experiments.");
    }
}

public class MultilevelInheritance1 {
    public static void main(String[] args) {
        Researcher r = new Researcher();
        r.study();
        r.specialize();
        r.research();
    }
}
```

OUTPUT:

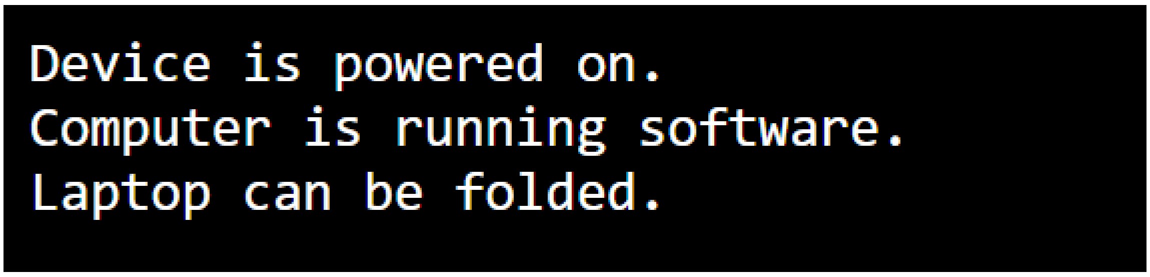


```
Student is studying.
Graduate is specializing in a subject.
Researcher is conducting experiments.
```

## 5 B) DEVICE-COMPUTER-LAPTOP

```java
class Device {
    void powerOn() {
        System.out.println("Device is powered on.");
    }
}

class Computer extends Device {
    void runSoftware() {
        System.out.println("Computer is running software.");
    }
}

class Laptop extends Computer {
    void fold() {
        System.out.println("Laptop can be folded.");
    }
}

public class MultilevelInheritance2 {
    public static void main(String[] args) {
        Laptop myLaptop = new Laptop();
        myLaptop.powerOn();    // From Device
        myLaptop.runSoftware(); // From Computer
        myLaptop.fold();        // Own method
    }
}
```

OUTPUT:



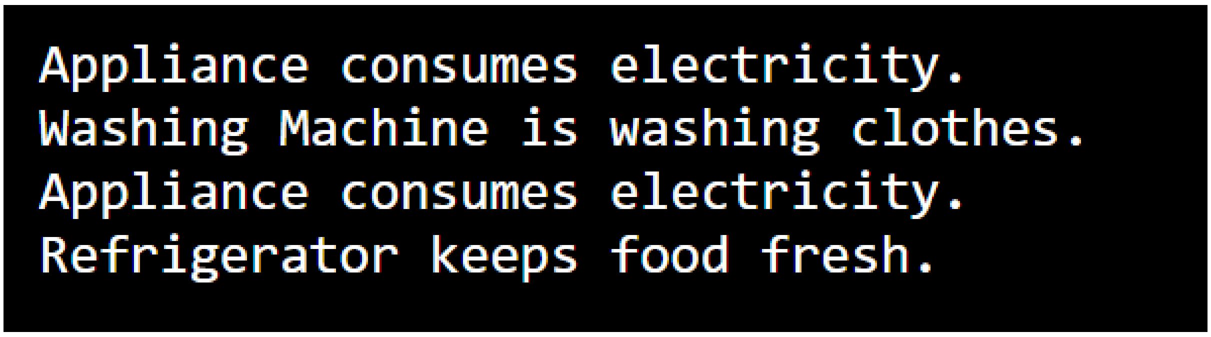
```
Device is powered on.
Computer is running software.
Laptop can be folded.
```

# 6. HIERARCHICAL INHERITANCE PROGRAMS

## 6 A) APPLIANCE – WASHING MACHINE / REFRIGERATOR

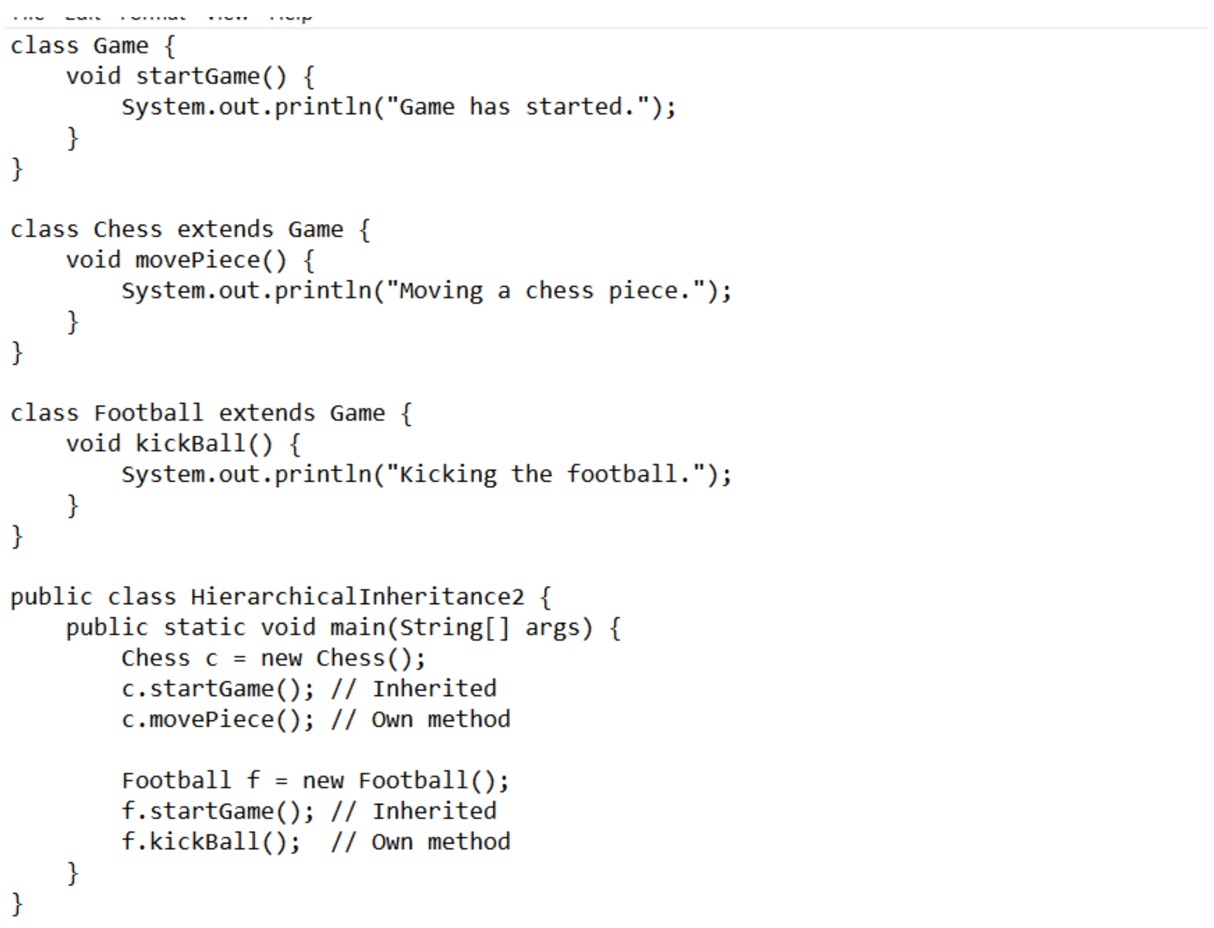
```java
class Appliance {
    void consumeElectricity() {
        System.out.println("Appliance consumes electricity.");
    }
}

class WashingMachine extends Appliance {
    void washClothes() {
        System.out.println("Washing Machine is washing clothes.");
    }
}

class Refrigerator extends Appliance {
    void keepFoodFresh() {
        System.out.println("Refrigerator keeps food fresh.");
    }
}

public class HierarchicalInheritance1 {
    public static void main(String[] args) {
        WashingMachine wm = new WashingMachine();
        wm.consumeElectricity(); // Inherited
        wm.washClothes();        // Own method

        Refrigerator fridge = new Refrigerator();
        fridge.consumeElectricity(); // Inherited
        fridge.keepFoodFresh();      // Own method
    }
}
```

OUTPUT:



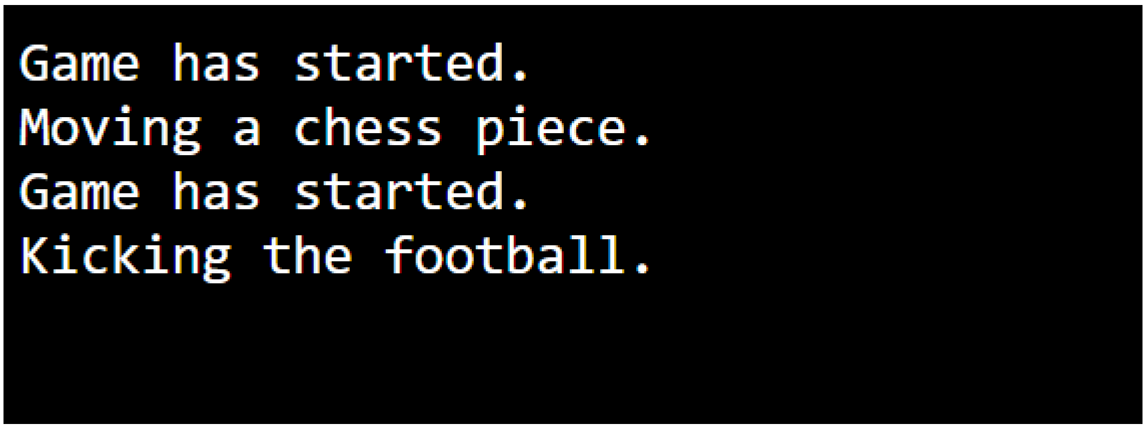
```
Appliance consumes electricity.
Washing Machine is washing clothes.
Appliance consumes electricity.
Refrigerator keeps food fresh.
```

## 6 B) GAME- CHESS \ FOOTBALL



```
class Game {
    void startGame() {
        System.out.println("Game has started.");
    }
}

class Chess extends Game {
    void movePiece() {
        System.out.println("Moving a chess piece.");
    }
}

class Football extends Game {
    void kickBall() {
        System.out.println("Kicking the football.");
    }
}

public class HierarchicalInheritance2 {
    public static void main(String[] args) {
        Chess c = new Chess();
        c.startGame(); // Inherited
        c.movePiece(); // Own method

        Football f = new Football();
        f.startGame(); // Inherited
        f.kickBall();  // Own method
    }
}
```

OUTPUT:



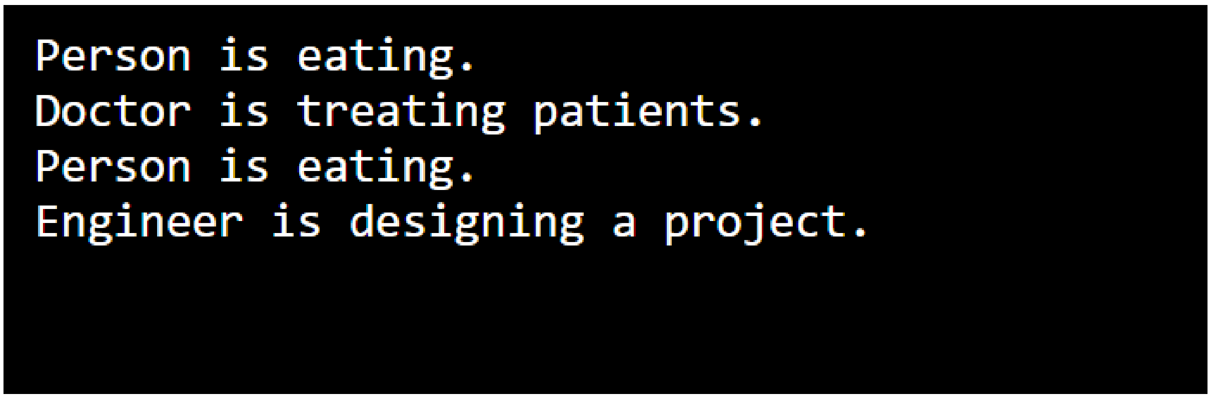
```
Game has started.
Moving a chess piece.
Game has started.
Kicking the football.
```

# 7. HYBRID INHERITANCE PROGRAMS

## 7 A) PERSON-DOCTOR\ ENGINEER

```
interface Worker {
    void performDuties();
}

class Person {
    void eat() {
        System.out.println("Person is eating.");
    }
}

class Doctor extends Person implements Worker {
    public void performDuties() {
        System.out.println("Doctor is treating patients.");
    }
}

class Engineer extends Person implements Worker {
    public void performDuties() {
        System.out.println("Engineer is designing a project.");
    }
}

public class HybridInheritance1 {
    public static void main(String[] args) {
        Doctor d = new Doctor();
        d.eat();          // From Person
        d.performDuties(); // From Worker

        Engineer e = new Engineer();
        e.eat();          // From Person
        e.performDuties(); // From Worker
    }
}
```

OUTPUT:

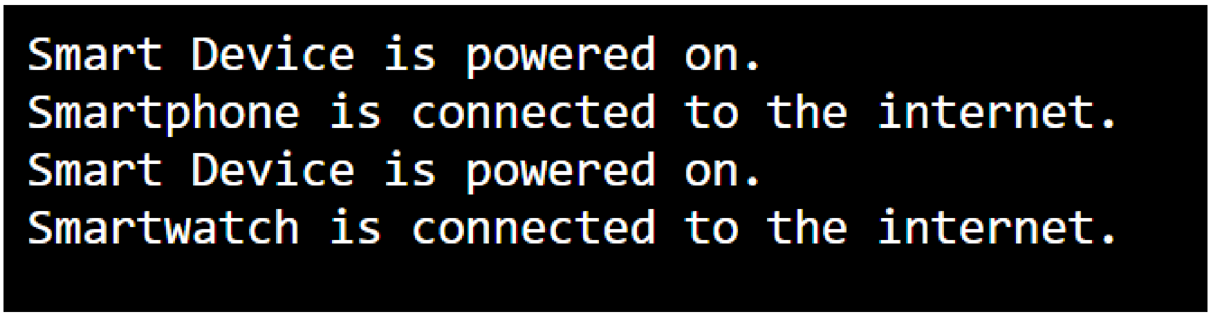


```
Person is eating.
Doctor is treating patients.
Person is eating.
Engineer is designing a project.
```

## 7 B) SMART DEVICE- SMART PHONE \ SMART WATCH

```java
interface Connectivity {
    void connectToInternet();
}

class SmartDevice {
    void powerOn() {
        System.out.println("Smart Device is powered on.");
    }
}

class Smartphone extends SmartDevice implements Connectivity {
    public void connectToInternet() {
        System.out.println("Smartphone is connected to the internet.");
    }
}

class SmartWatch extends SmartDevice implements Connectivity {
    public void connectToInternet() {
        System.out.println("Smartwatch is connected to the internet.");
    }
}

public class HybridInheritance2 {
    public static void main(String[] args) {
        Smartphone phone = new Smartphone();
        phone.powerOn();
        phone.connectToInternet();

        SmartWatch watch = new SmartWatch();
        watch.powerOn();
        watch.connectToInternet();
    }
}
```

OUTPUT:



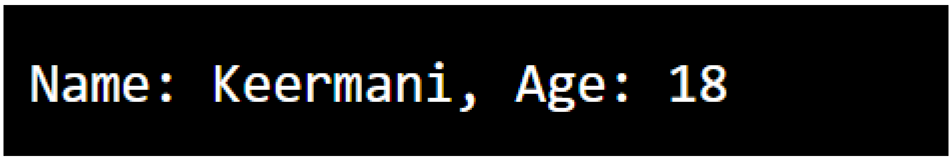
```
Smart Device is powered on.
Smartphone is connected to the internet.
Smart Device is powered on.
Smartwatch is connected to the internet.
```

## 8. CONSTRUCTOR PROGRAMS

### 8 A) STUDENT CONSTRUCTOR

```
class Student {
    String name;
    int age;

    // Constructor
    Student(String n, int a) {
        name = n;
        age = a;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }

    public static void main(String[] args) {
        Student s1 = new Student("Keermani", 18);
        s1.display();
    }
}
```

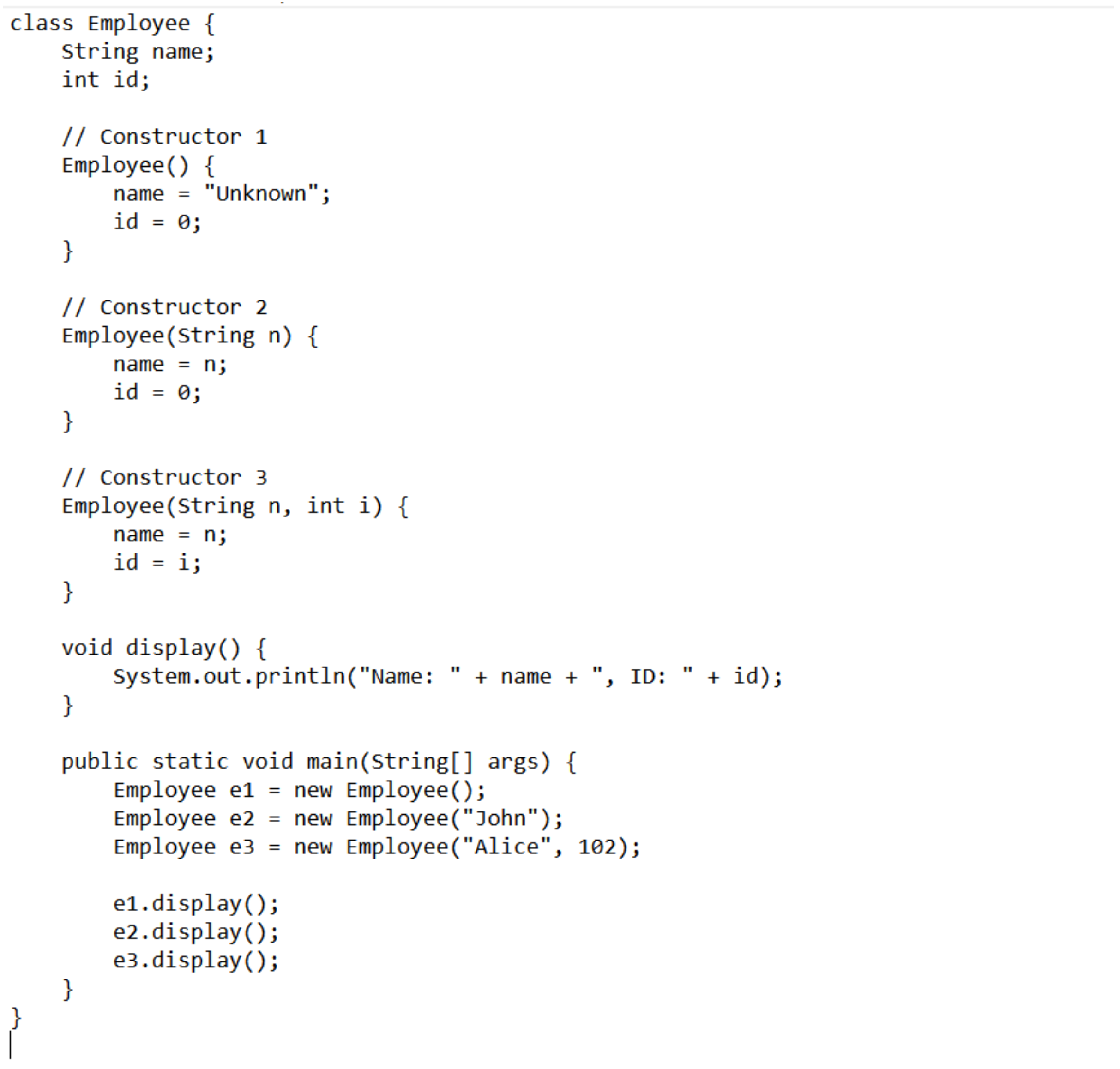
OUTPUT:



```
Name: Keermani, Age: 18
```

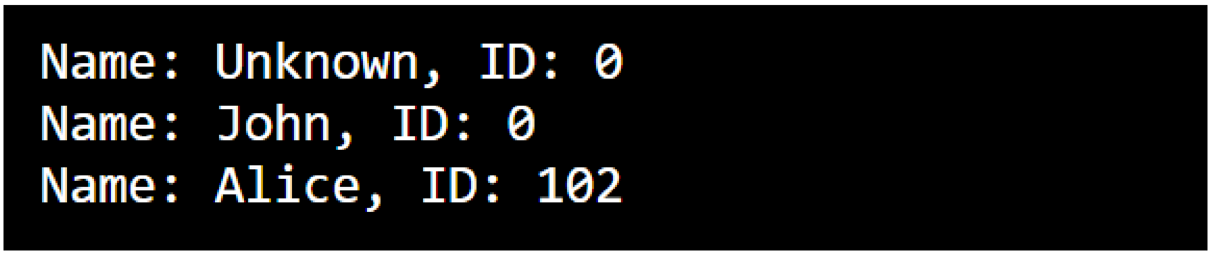
# 9. CONSTRUCTOR OVERLOADING PROGRAMS

## 9 A) EMPLOYEE CONSTRUCTOR OVERLOADING



```
class Employee {
    String name;
    int id;

    // Constructor 1
    Employee() {
        name = "Unknown";
        id = 0;
    }

    // Constructor 2
    Employee(String n) {
        name = n;
        id = 0;
    }

    // Constructor 3
    Employee(String n, int i) {
        name = n;
        id = i;
    }

    void display() {
        System.out.println("Name: " + name + ", ID: " + id);
    }

    public static void main(String[] args) {
        Employee e1 = new Employee();
        Employee e2 = new Employee("John");
        Employee e3 = new Employee("Alice", 102);

        e1.display();
        e2.display();
        e3.display();
    }
}
```

OUTPUT:



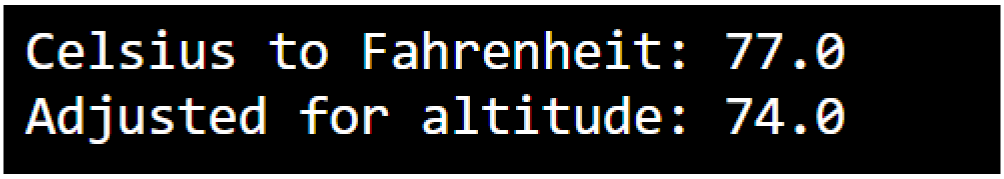
```
Name: Unknown, ID: 0
Name: John, ID: 0
Name: Alice, ID: 102
```

## 10. METHOD OVERLOADING PROGRAMS

### 10 A) TEMPERATURE CONVERTER OVERLOADING

```java
class Employee {
    String name;
    int id;

    // Constructor 1
    Employee() {
        name = "Unknown";
        id = 0;
    }

    // Constructor 2
    Employee(String n) {
        name = n;
        id = 0;
    }

    // Constructor 3
    Employee(String n, int i) {
        name = n;
        id = i;
    }

    void display() {
        System.out.println("Name: " + name + ", ID: " + id);
    }

    public static void main(String[] args) {
        Employee e1 = new Employee();
        Employee e2 = new Employee("John");class TemperatureConverter {
    // Convert Celsius to Fahrenheit
    double convert(double celsius) {
        return (celsius * 9/5) + 32;
    }

    // Convert Celsius and adjust for altitude
    double convert(double celsius, int altitude) {
        return ((celsius * 9/5) + 32) - (altitude * 0.003);
    }

    public static void main(String[] args) {
        TemperatureConverter converter = new TemperatureConverter();
        System.out.println("Celsius to Fahrenheit: " + converter.convert(25));
        System.out.println("Adjusted for altitude: " + converter.convert(25, 1000));
    }
}

        Employee e3 = new Employee("Alice", 102);

        e1.display();
        e2.display();
        e3.display();
    }
}
```

OUTPUT:

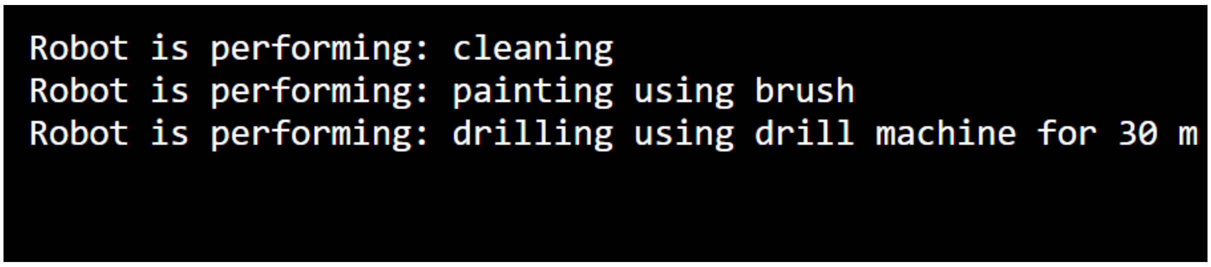


```
Celsius to Fahrenheit: 77.0
Adjusted for altitude: 74.0
```

## 10 B) ROBOT TASK EXECUTION OVERLOADING

```
class Robot {
    // Perform a task without a tool
    void performTask(String task) {
        System.out.println("Robot is performing: " + task);
    }

    // Perform a task with a tool
    void performTask(String task, String tool) {
        System.out.println("Robot is performing: " + task + " using " + tool);
    }

    // Perform a task with a tool and duration
    void performTask(String task, String tool, int duration) {
        System.out.println("Robot is performing: " + task + " using " + tool + " for " + duration + " minutes.");
    }

    public static void main(String[] args) {
        Robot r = new Robot();
        r.performTask("cleaning");
        r.performTask("painting", "brush");
        r.performTask("drilling", "drill machine", 30);
    }
}
```

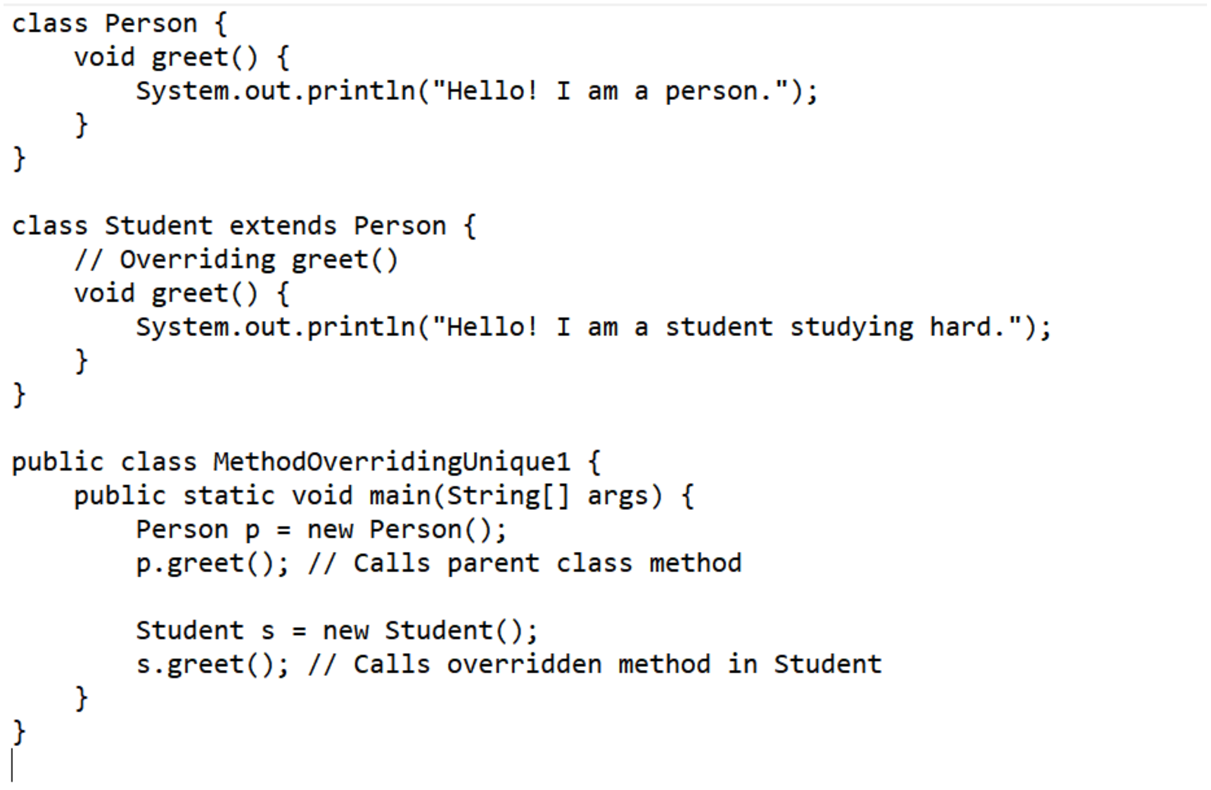
## OUTPUT:



```
Robot is performing: cleaning
Robot is performing: painting using brush
Robot is performing: drilling using drill machine for 30 m
```

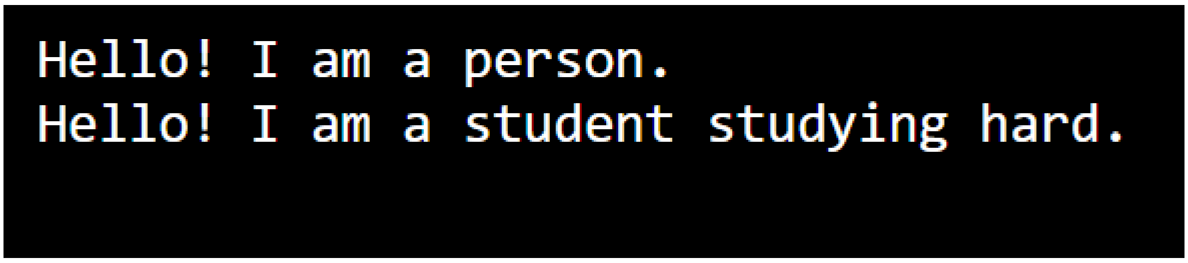
## 11. METHOD OVERRIDING PROGRAMS

### 11 A) PARENT-CHILD GREETING



```java
class Person {
    void greet() {
        System.out.println("Hello! I am a person.");
    }
}

class Student extends Person {
    // Overriding greet()
    void greet() {
        System.out.println("Hello! I am a student studying hard.");
    }
}

public class MethodOverridingUnique1 {
    public static void main(String[] args) {
        Person p = new Person();
        p.greet(); // Calls parent class method

        Student s = new Student();
        s.greet(); // Calls overridden method in Student
    }
}
```

OUTPUT:

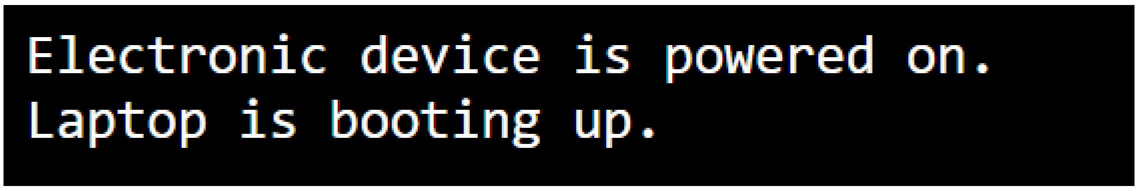


```
Hello! I am a person.
Hello! I am a student studying hard.
```

## 11 B) ELECTRONIC DEVICE POWER

```
class ElectronicDevice {
    void powerOn() {
        System.out.println("Electronic device is powered on.");
    }
}

class Laptop extends ElectronicDevice {
    // Overriding powerOn()
    void powerOn() {
        System.out.println("Laptop is booting up.");
    }
}

public class MethodOverridingUnique2 {
    public static void main(String[] args) {
        ElectronicDevice device = new ElectronicDevice();
        device.powerOn(); // Calls parent class method

        Laptop myLaptop = new Laptop();
        myLaptop.powerOn(); // Calls overridden method in Laptop
    }
}
```

OUTPUT:



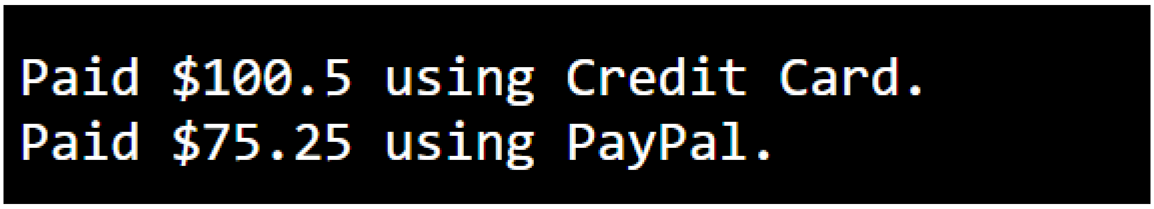
```
Electronic device is powered on.
Laptop is booting up.
```

# 12. INTERFACE PROGRAMS

## 12 A) PAYMENT SYSTEM

```
interface Payment {
    void makePayment(double amount);
}

class CreditCardPayment implements Payment {
    public void makePayment(double amount) {
        System.out.println("Paid $" + amount + " using Credit Card.");
    }
}

class PayPalPayment implements Payment {
    public void makePayment(double amount) {
        System.out.println("Paid $" + amount + " using PayPal.");
    }
}

public class InterfaceExample1 {
    public static void main(String[] args) {
        Payment payment1 = new CreditCardPayment();
        payment1.makePayment(100.50);

        Payment payment2 = new PayPalPayment();
        payment2.makePayment(75.25);
    }
}
```

OUTPUT:

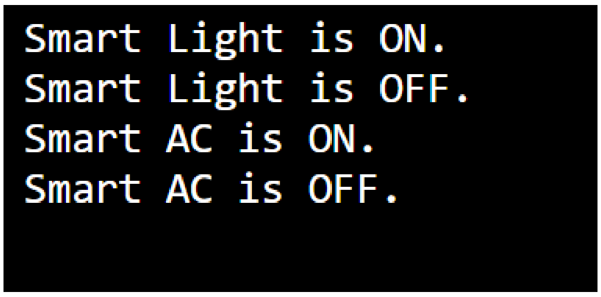


```
Paid $100.5 using Credit Card.
Paid $75.25 using PayPal.
```

## 12 B) SMART HOME DEVICES

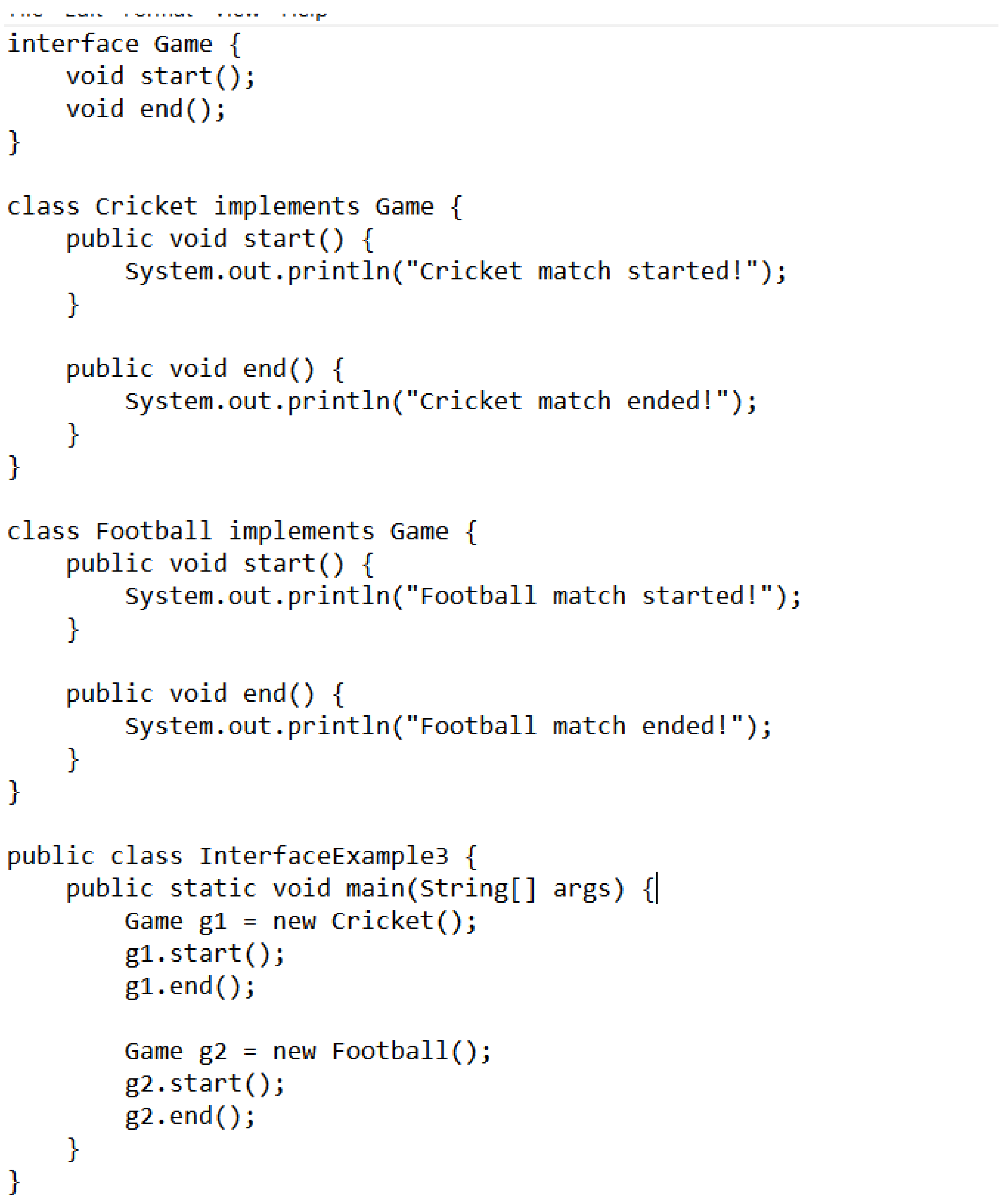
```java
interface SmartDevice {
    void turnOn();
    void turnOff();
}

class SmartLight implements SmartDevice {
    public void turnOn() {
        System.out.println("Smart Light is ON.");
    }

    public void turnOff() {
        System.out.println("Smart Light is OFF.");
    }
}

class SmartAC implements SmartDevice {
    public void turnOn() {
        System.out.println("Smart AC is ON.");
    }

    public void turnOff() {
        System.out.println("Smart AC is OFF.");
    }
}

public class InterfaceExample2 {
    public static void main(String[] args) {
        SmartDevice light = new SmartLight();
        light.turnOn();
        light.turnOff();

        SmartDevice ac = new SmartAC();
        ac.turnOn();
        ac.turnOff();
    }
}
```

OUTPUT:



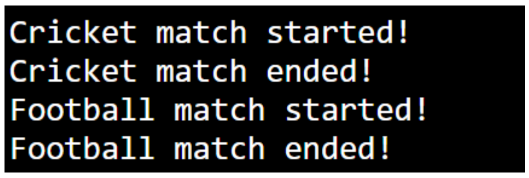
```
Smart Light is ON.
Smart Light is OFF.
Smart AC is ON.
Smart AC is OFF.
```

## 12 C) SPORTS GAME



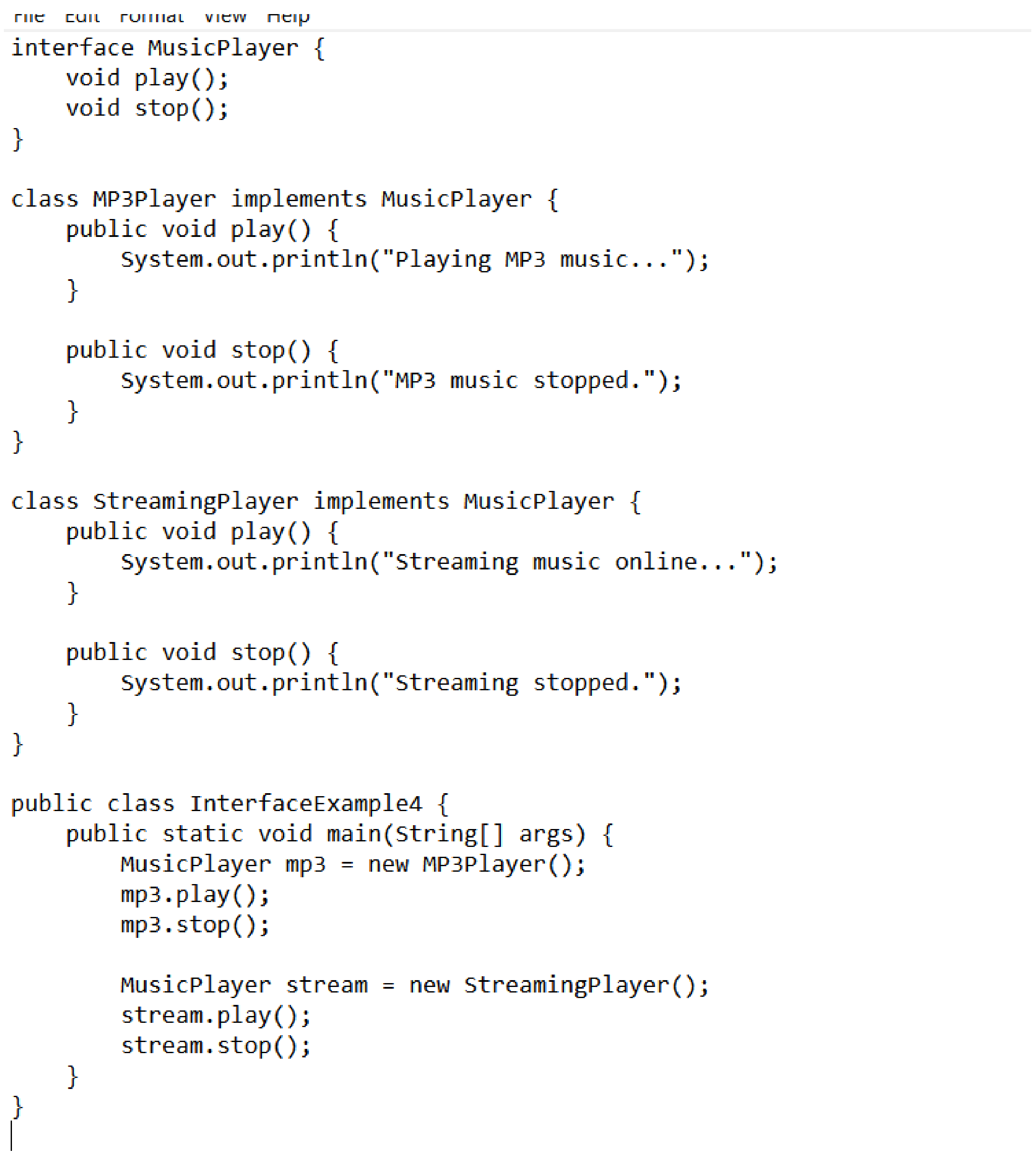
```
interface Game {
    void start();
    void end();
}

class Cricket implements Game {
    public void start() {
        System.out.println("Cricket match started!");
    }

    public void end() {
        System.out.println("Cricket match ended!");
    }
}

class Football implements Game {
    public void start() {
        System.out.println("Football match started!");
    }

    public void end() {
        System.out.println("Football match ended!");
    }
}

public class InterfaceExample3 {
    public static void main(String[] args) {
        Game g1 = new Cricket();
        g1.start();
        g1.end();

        Game g2 = new Football();
        g2.start();
        g2.end();
    }
}
```

OUTPUT:



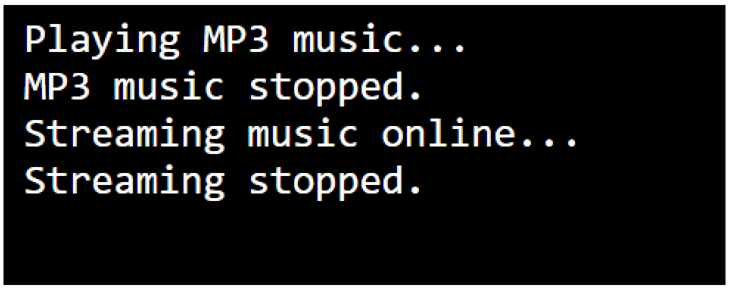
```
Cricket match started!
Cricket match ended!
Football match started!
Football match ended!
```

12 D) MUSIC PLAYER



```
interface MusicPlayer {
    void play();
    void stop();
}

class MP3Player implements MusicPlayer {
    public void play() {
        System.out.println("Playing MP3 music...");
    }

    public void stop() {
        System.out.println("MP3 music stopped.");
    }
}

class StreamingPlayer implements MusicPlayer {
    public void play() {
        System.out.println("Streaming music online...");
    }

    public void stop() {
        System.out.println("Streaming stopped.");
    }
}

public class InterfaceExample4 {
    public static void main(String[] args) {
        MusicPlayer mp3 = new MP3Player();
        mp3.play();
        mp3.stop();

        MusicPlayer stream = new StreamingPlayer();
        stream.play();
        stream.stop();
    }
}
```

OUTPUT:



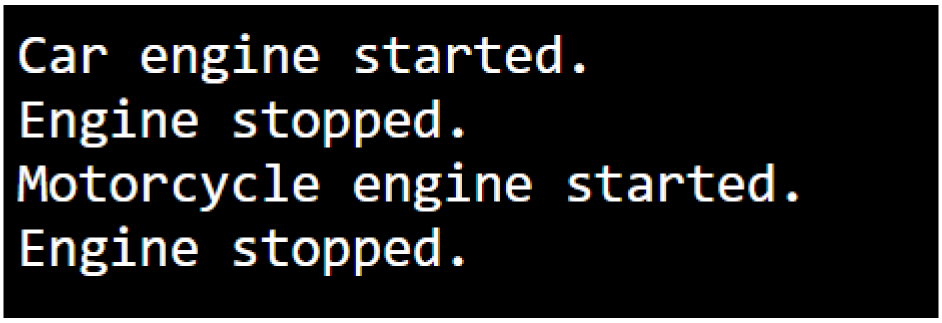
```
Playing MP3 music...
MP3 music stopped.
Streaming music online...
Streaming stopped.
```

## 13. ABSTRACT CLASS PROGRAMS

### 13 A) VEHICLE

```
abstract class Vehicle {
    abstract void startEngine();
    void stopEngine() {
        System.out.println("Engine stopped.");
    }
}

class Car extends Vehicle {
    void startEngine() {
        System.out.println("Car engine started.");
    }
}

class Motorcycle extends Vehicle {
    void startEngine() {
        System.out.println("Motorcycle engine started.");
    }
}

public class AbstractClassExample1 {
    public static void main(String[] args) {
        Vehicle car = new Car();
        car.startEngine();
        car.stopEngine();

        Vehicle bike = new Motorcycle();
        bike.startEngine();
        bike.stopEngine();
    }
}
```

OUTPUT:

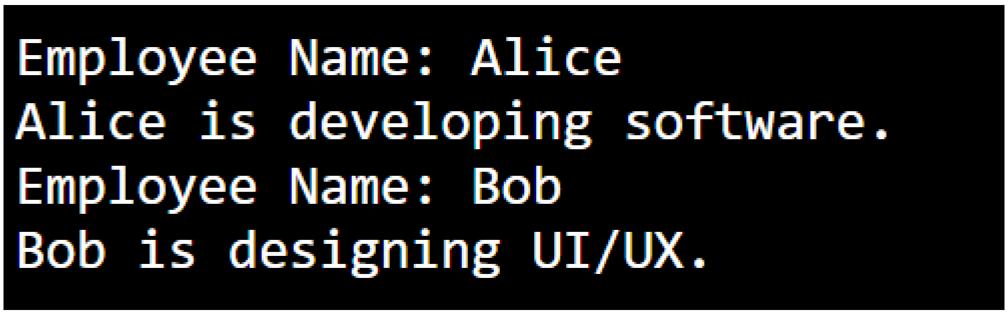


```
Car engine started.
Engine stopped.
Motorcycle engine started.
Engine stopped.
```

## 13 B) EMPLOYEE

```java
abstract class Employee {
    String name;
    Employee(String name) {
        this.name = name;
    }
    abstract void work();

    void showDetails() {
        System.out.println("Employee Name: " + name);
    }
}

class Developer extends Employee {
    Developer(String name) {
        super(name);
    }

    void work() {
        System.out.println(name + " is developing software.");
    }
}

class Designer extends Employee {
    Designer(String name) {
        super(name);
    }

    void work() {
        System.out.println(name + " is designing UI/UX.");
    }
}

public class AbstractClassExample2 {
    public static void main(String[] args) {
        Employee dev = new Developer("Alice");
        dev.showDetails();
        dev.work();

        Employee des = new Designer("Bob");
        des.showDetails();
        des.work();
    }
}
```

OUTPUT:

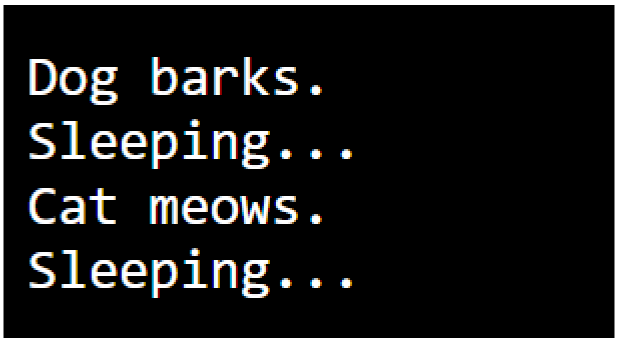


```
Employee Name: Alice
Alice is developing software.
Employee Name: Bob
Bob is designing UI/UX.
```

13 C) ANIMAL

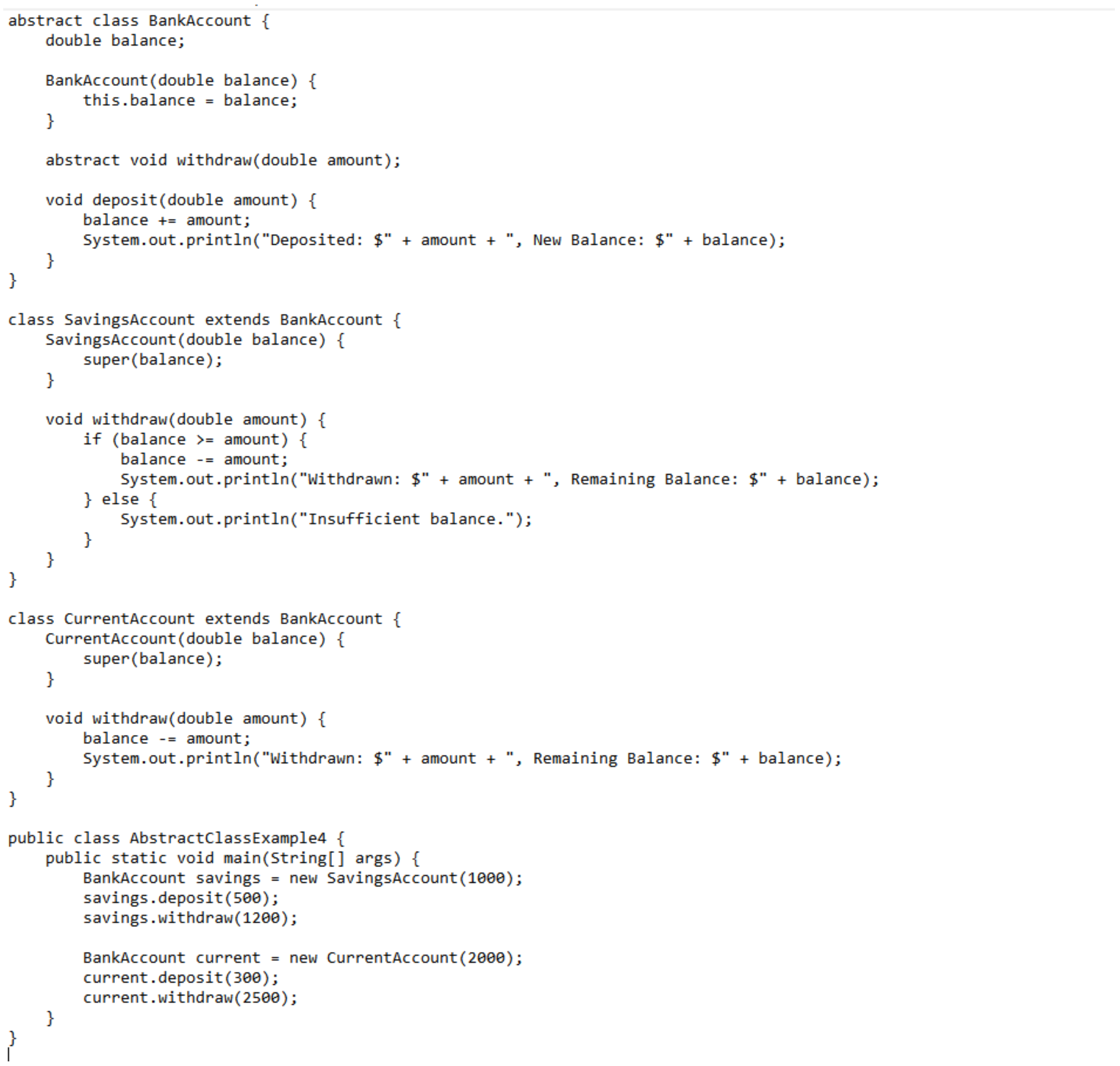
```
abstract class Animal {
    abstract void makeSound();
    void sleep() {
        System.out.println("Sleeping...");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks.");
    }
}

class Cat extends Animal {
    void makeSound() {
        System.out.println("Cat meows.");
    }
}

public class AbstractClassExample3 {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.makeSound();
        dog.sleep();

        Animal cat = new Cat();
        cat.makeSound();
        cat.sleep();
    }
}
```

OUTPUT:



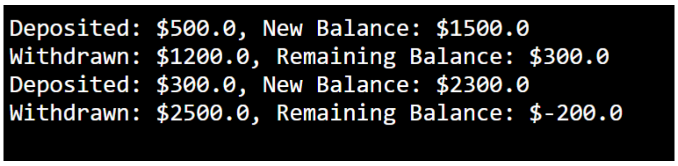
```
Dog barks.
Sleeping...
Cat meows.
Sleeping...
```

## 13 D) BANK ACCOUNT



```java
abstract class BankAccount {
    double balance;

    BankAccount(double balance) {
        this.balance = balance;
    }

    abstract void withdraw(double amount);

    void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: $" + amount + ", New Balance: $" + balance);
    }
}

class SavingsAccount extends BankAccount {
    SavingsAccount(double balance) {
        super(balance);
    }

    void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount + ", Remaining Balance: $" + balance);
        } else {
            System.out.println("Insufficient balance.");
        }
    }
}

class CurrentAccount extends BankAccount {
    CurrentAccount(double balance) {
        super(balance);
    }

    void withdraw(double amount) {
        balance -= amount;
        System.out.println("Withdrawn: $" + amount + ", Remaining Balance: $" + balance);
    }
}

public class AbstractClassExample4 {
    public static void main(String[] args) {
        BankAccount savings = new SavingsAccount(1000);
        savings.deposit(500);
        savings.withdraw(1200);

        BankAccount current = new CurrentAccount(2000);
        current.deposit(300);
        current.withdraw(2500);
    }
}
```

OUTPUT:



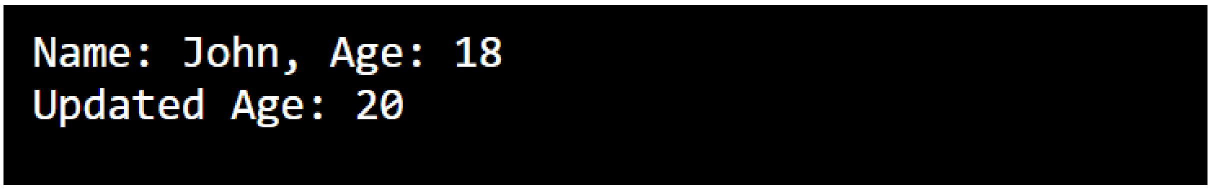
```
Deposited: $500.0, New Balance: $1500.0
Withdrawn: $1200.0, Remaining Balance: $300.0
Deposited: $300.0, New Balance: $2300.0
Withdrawn: $2500.0, Remaining Balance: $-200.0
```

# 14. ENCAPSULATION PROGRAMS

## 14 A) STUDENT DATA

```java
class Student {
    private String name;
    private int age;

    // Constructor
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // Setter methods
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        } else {
            System.out.println("Invalid age!");
        }
    }
}

public class EncapsulationExample1 {
    public static void main(String[] args) {
        Student s1 = new Student("John", 18);
        System.out.println("Name: " + s1.getName() + ", Age: " + s1.getAge());

        s1.setAge(20);
        System.out.println("Updated Age: " + s1.getAge());
    }
}
```

## OUTPUT:

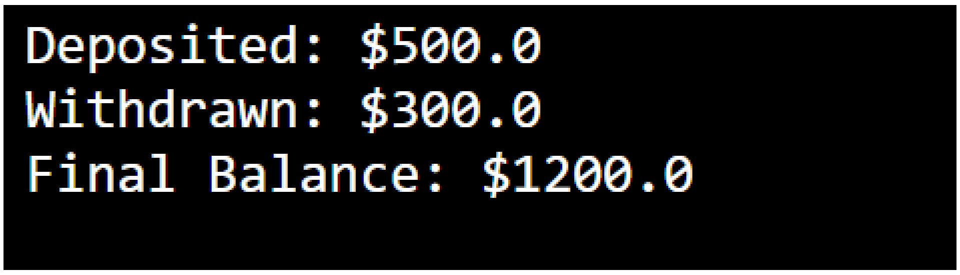


```
Name: John, Age: 18
Updated Age: 20
```

## 14 B) BANK ACCOUNT

```
class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        if (initialBalance > 0) {
            balance = initialBalance;
        } else {
            System.out.println("Invalid balance.");
        }
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount);
        } else {
            System.out.println("Insufficient funds or invalid amount.");
        }
    }
}

public class EncapsulationExample2 {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
        account.deposit(500);
        account.withdraw(300);
        System.out.println("Final Balance: $" + account.getBalance());
    }
}
```

OUTPUT:

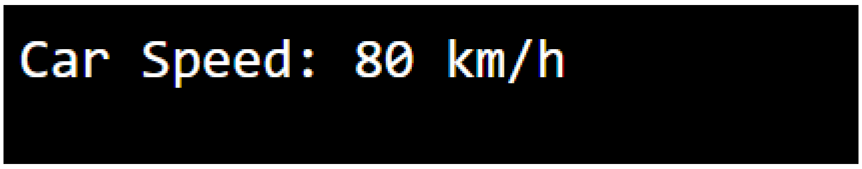


```
Deposited: $500.0
Withdrawn: $300.0
Final Balance: $1200.0
```

## 14 C) CAR CONTROL SPEED

```java
class Car {
    private int speed;

    public void setSpeed(int speed) {
        if (speed >= 0) {
            this.speed = speed;
        } else {
            System.out.println("Speed cannot be negative.");
        }
    }

    public int getSpeed() {
        return speed;
    }
}

public class EncapsulationExample3 {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.setSpeed(80);
        System.out.println("Car Speed: " + myCar.getSpeed() + " km/h");
    }
}
```

## OUTPUT:

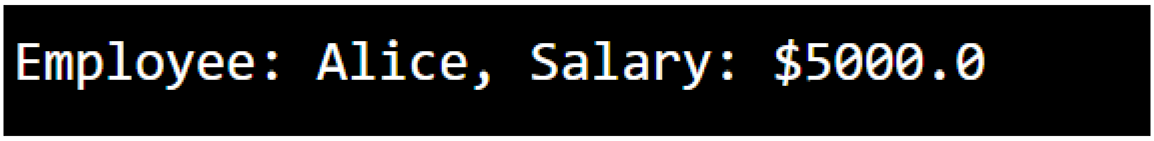


```
Car Speed: 80 km/h
```

## 14 D) EMPLOYEE DETAILS

```java
class Employee {
    private String empName;
    private double salary;

    public void setEmpName(String name) {
        this.empName = name;
    }

    public String getEmpName() {
        return empName;
    }

    public void setSalary(double salary) {
        if (salary > 0) {
            this.salary = salary;
        } else {
            System.out.println("Invalid salary.");
        }
    }

    public double getSalary() {
        return salary;
    }
}

public class EncapsulationExample4 {
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setEmpName("Alice");
        emp.setSalary(5000);
        System.out.println("Employee: " + emp.getEmpName() + ", Salary: $" + emp.getSalary());
    }
}
```

OUTPUT:



```
Employee: Alice, Salary: $5000.0
```

## 15. PACKAGES PROGRAMS

15 A) USER DEFINED PACKAGE

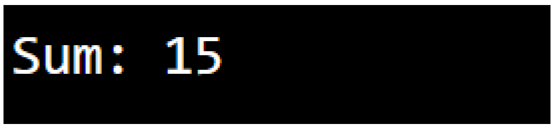
PACKAGE FILE:

```
package mathoperations;

public class Addition {
    public int add(int a, int b) {
        return a + b;
    }
}
```

MAIN CLASS:

```
import mathoperations.Addition;

public class UserPackageExample1 {
    public static void main(String[] args) {
        Addition obj = new Addition();
        System.out.println("Sum: " + obj.add(5, 10));
    }
}
```

OUTPUT:



```
Sum: 15
```

## 15 B) USER DEFINED PACKAGE

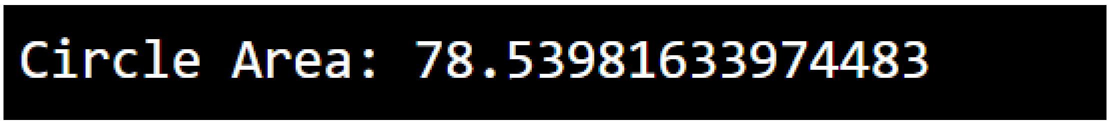
PACKAGE FILE:

```
package shapes;

public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double area() {
        return Math.PI * radius * radius;
    }
}
```

MAIN CLASS:

```
import shapes.Circle;

public class UserPackageExample2 {
    public static void main(String[] args) {
        Circle c = new Circle(5);
        System.out.println("Circle Area: " + c.area());
    }
}
```

OUTPUT:

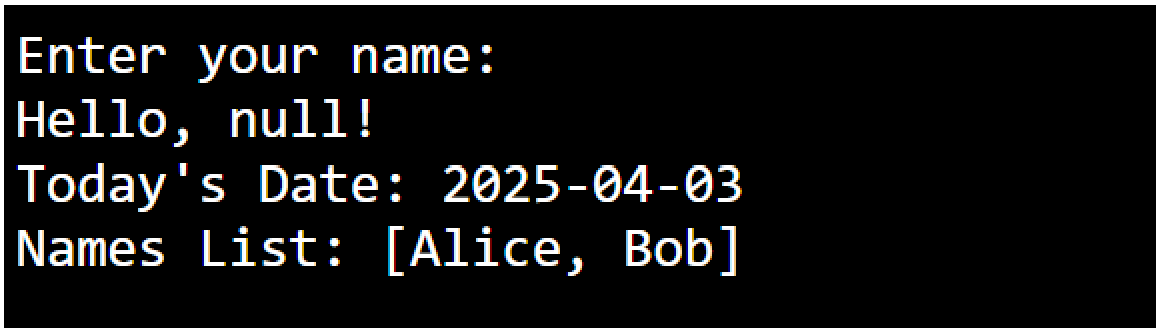


```
Circle Area: 78.53981633974483
```

## 15 C) BUILT IN PACKAGES

```java
import java.util.ArrayList;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.time.LocalDate;

public class BuiltInPackageExample1 {
    public static void main(String[] args) throws Exception {
        // Using java.util.ArrayList
        ArrayList<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");

        // Using java.io.BufferedReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter your name: ");
        String userName = br.readLine();

        // Using java.time.LocalDate
        LocalDate today = LocalDate.now();

        System.out.println("Hello, " + userName + "!");
        System.out.println("Today's Date: " + today);
        System.out.println("Names List: " + names);
    }
}
```

OUTPUT:

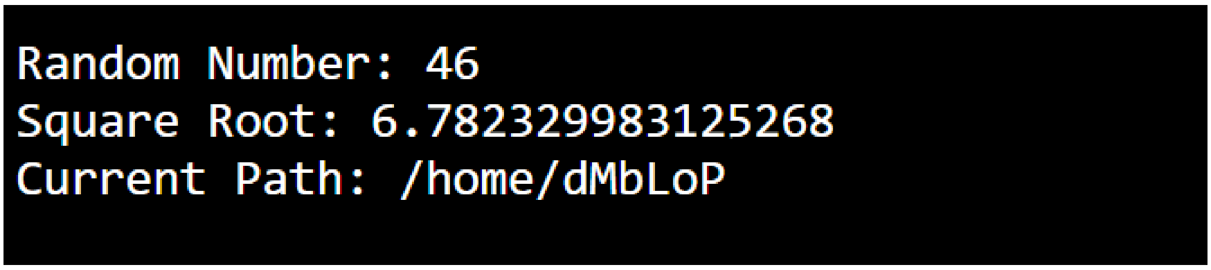


```
Enter your name:
Hello, null!
Today's Date: 2025-04-03
Names List: [Alice, Bob]
```

## 15 D) BUILT IN PACKAGES

```
import java.util.Random;
import java.lang.Math;
import java.nio.file.Paths;

public class BuiltInPackageExample2 {
    public static void main(String[] args) {
        // Using java.util.Random
        Random rand = new Random();
        int randomNum = rand.nextInt(100);
        System.out.println("Random Number: " + randomNum);

        // Using java.lang.Math
        double squareRoot = Math.sqrt(randomNum);
        System.out.println("Square Root: " + squareRoot);

        // Using java.nio.file.Paths
        System.out.println("Current Path: " + Paths.get("").toAbsolutePath());
    }
}
```

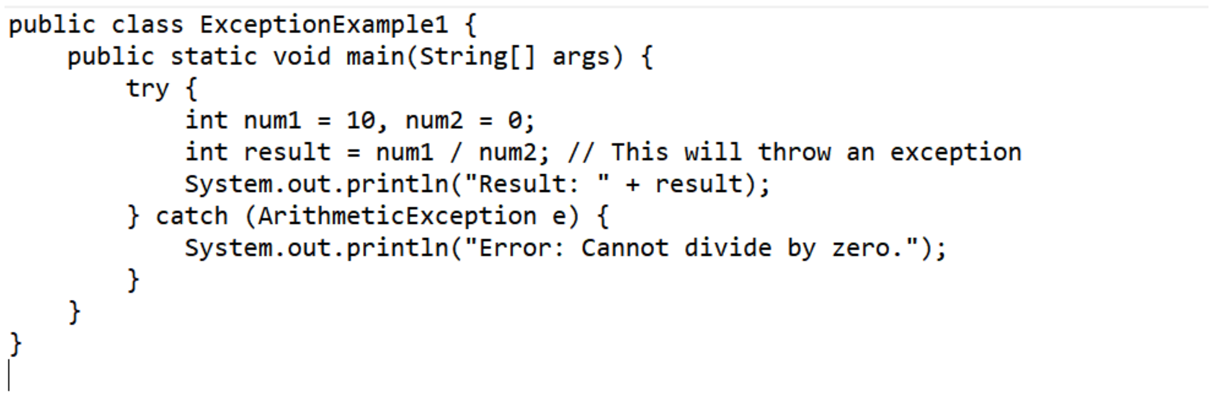
OUTPUT:



```
Random Number: 46
Square Root: 6.782329983125268
Current Path: /home/dMbLoP
```

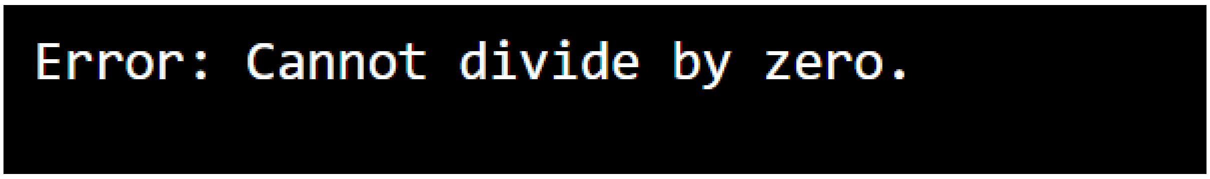
# 16. EXCEPTION HANDLING PROGRAMS

## 16 A) DIVIDE BY ZERO



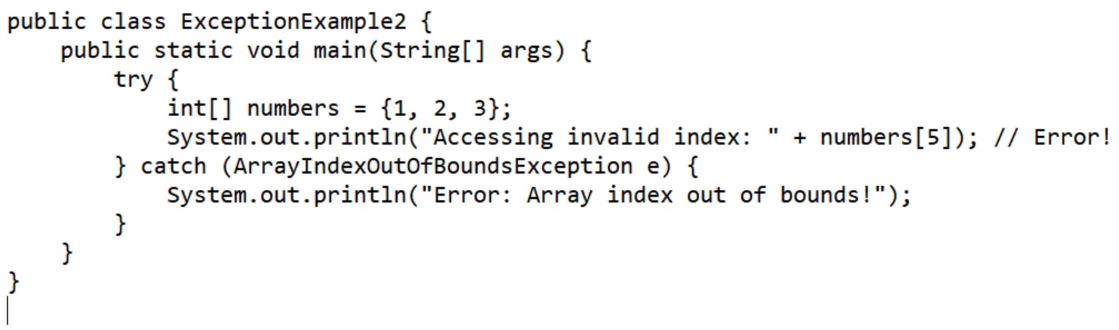
```
public class ExceptionExample1 {
    public static void main(String[] args) {
        try {
            int num1 = 10, num2 = 0;
            int result = num1 / num2; // This will throw an exception
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        }
    }
}
```

OUTPUT:



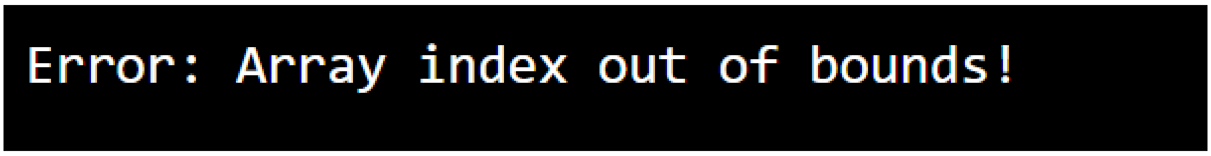
```
Error: Cannot divide by zero.
```

## 16 B) ARRAY INDEX OUT OF BOUND



```
public class ExceptionExample2 {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
            System.out.println("Accessing invalid index: " + numbers[5]); // Error!
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Array index out of bounds!");
        }
    }
}
```

OUTPUT:

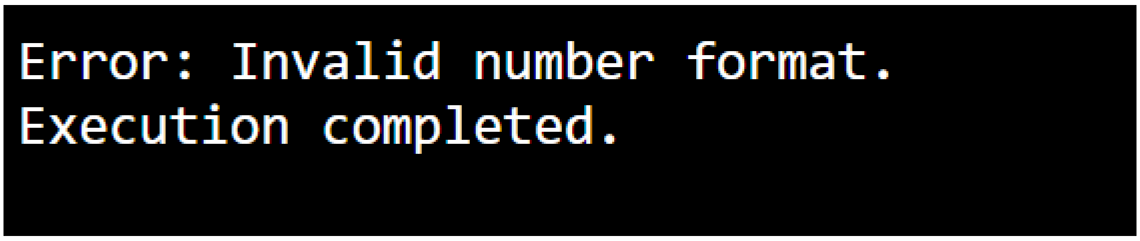


```
Error: Array index out of bounds!
```

## 16 C) INVALID NUMBER FORMAT

```java
public class ExceptionExample3 {
    public static void main(String[] args) {
        try {
            int num = Integer.parseInt("abc"); // NumberFormatException
            int result = 10 / 0; // ArithmeticException
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid number format.");
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero.");
        } finally {
            System.out.println("Execution completed.");
        }
    }
}
```

OUTPUT:

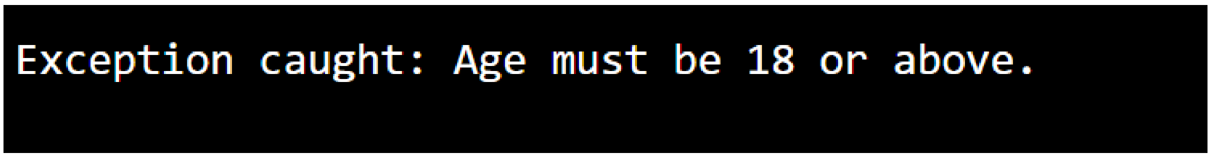


```
Error: Invalid number format.
Execution completed.
```

## 16 D) AGE EXCEPTION

```java
class AgeException extends Exception {
    public AgeException(String message) {
        super(message);
    }
}

public class ExceptionExample4 {
    public static void validateAge(int age) throws AgeException {
        if (age < 18) {
            throw new AgeException("Age must be 18 or above.");
        } else {
            System.out.println("Valid age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(15);
        } catch (AgeException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

OUTPUT:



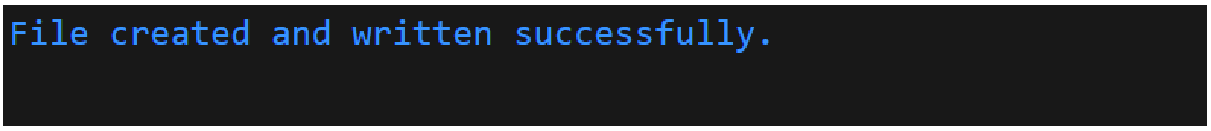
```
Exception caught: Age must be 18 or above.
```

# 17. FILE HANDLING PROGRAMS

## 17 A) CREATE AND WRITE TO A FILE

```
import java.io.FileWriter;
import java.io.IOException;

public class FileHandlingExample1 {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("sample.txt");
            writer.write("Hello, this is a sample file!");
            writer.close();
            System.out.println("File created and written successfully.");
        } catch (IOException e) {
            System.out.println("Error writing to the file.");
        }
    }
}
```

OUTPUT:

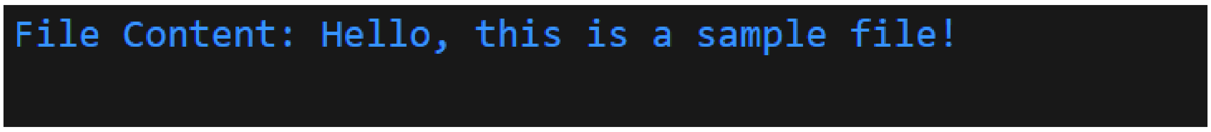


```
File created and written successfully.
```

## 17 B) READ A FILE

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileHandlingExample2 {
    public static void main(String[] args) {
        try {
            File file = new File("sample.txt");
            Scanner reader = new Scanner(file);
            while (reader.hasNextLine()) {
                String data = reader.nextLine();
                System.out.println("File Content: " + data);
            }
            reader.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
        }
    }
}
```

OUTPUT:

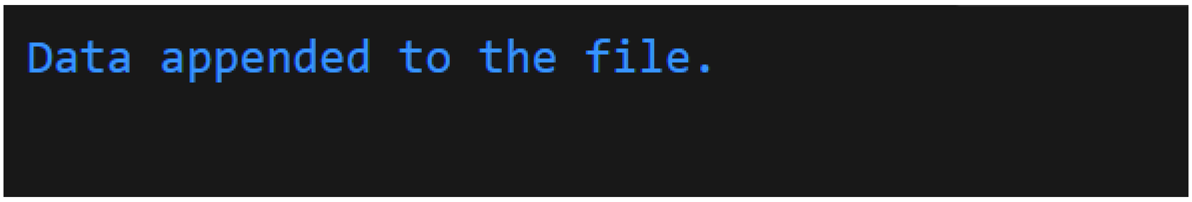


```
File Content: Hello, this is a sample file!
```

17 C)

```
import java.io.FileWriter;
import java.io.IOException;

public class FileHandlingExample3 {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("sample.txt", true);
            writer.append("\nAppending new text.");
            writer.close();
            System.out.println("Data appended to the file.");
        } catch (IOException e) {
            System.out.println("Error appending to the file.");
        }
    }
}
```

OUTPUT:

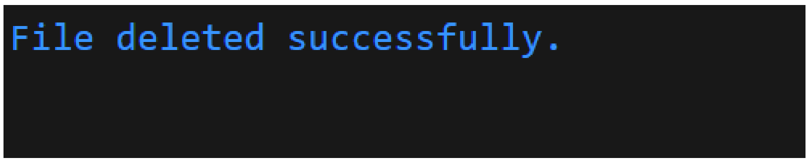


```
Data appended to the file.
```

17 D)

```
import java.io.File;

public class FileHandlingExample4 {
    public static void main(String[] args) {
        File file = new File("sample.txt");
        if (file.delete()) {
            System.out.println("File deleted successfully.");
        } else {
            System.out.println("Failed to delete the file.");
        }
    }
}
```

OUTPUT:



```
File deleted successfully.
```