U VENKATA ACHYUTH KRISHNA

CH.SC.U4CSE24148

OBJECT ORIENTED PROGRAMMING (23CSE111)

LAB RECORD

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING, CHENNAI

BONAFIDE CERTIFICATE

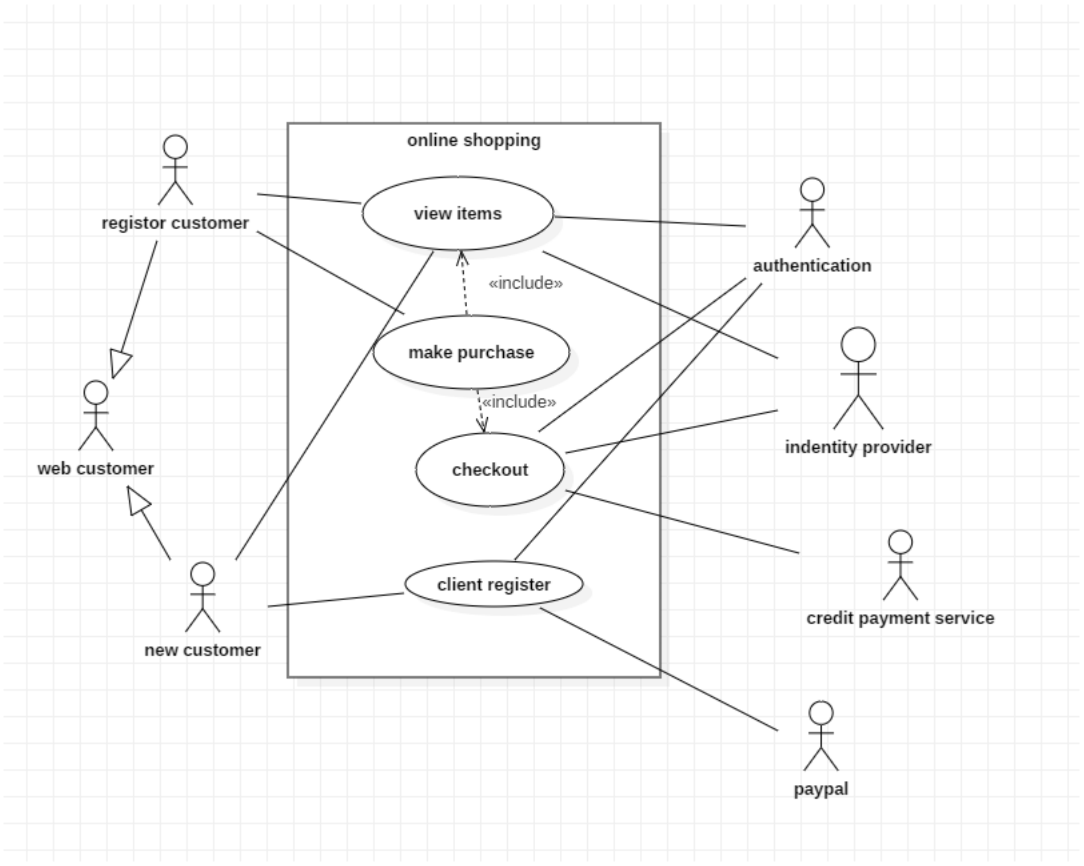
This is to certify that the Lab Record work for 23CSE111-Object Oriented Programming Subject submitted by **CH.SC.U4CSE24148 – U Venkata Achyuth Krishna** in "Computer Science and Engineering" is a Bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Chennai.

This Lab examination on held on

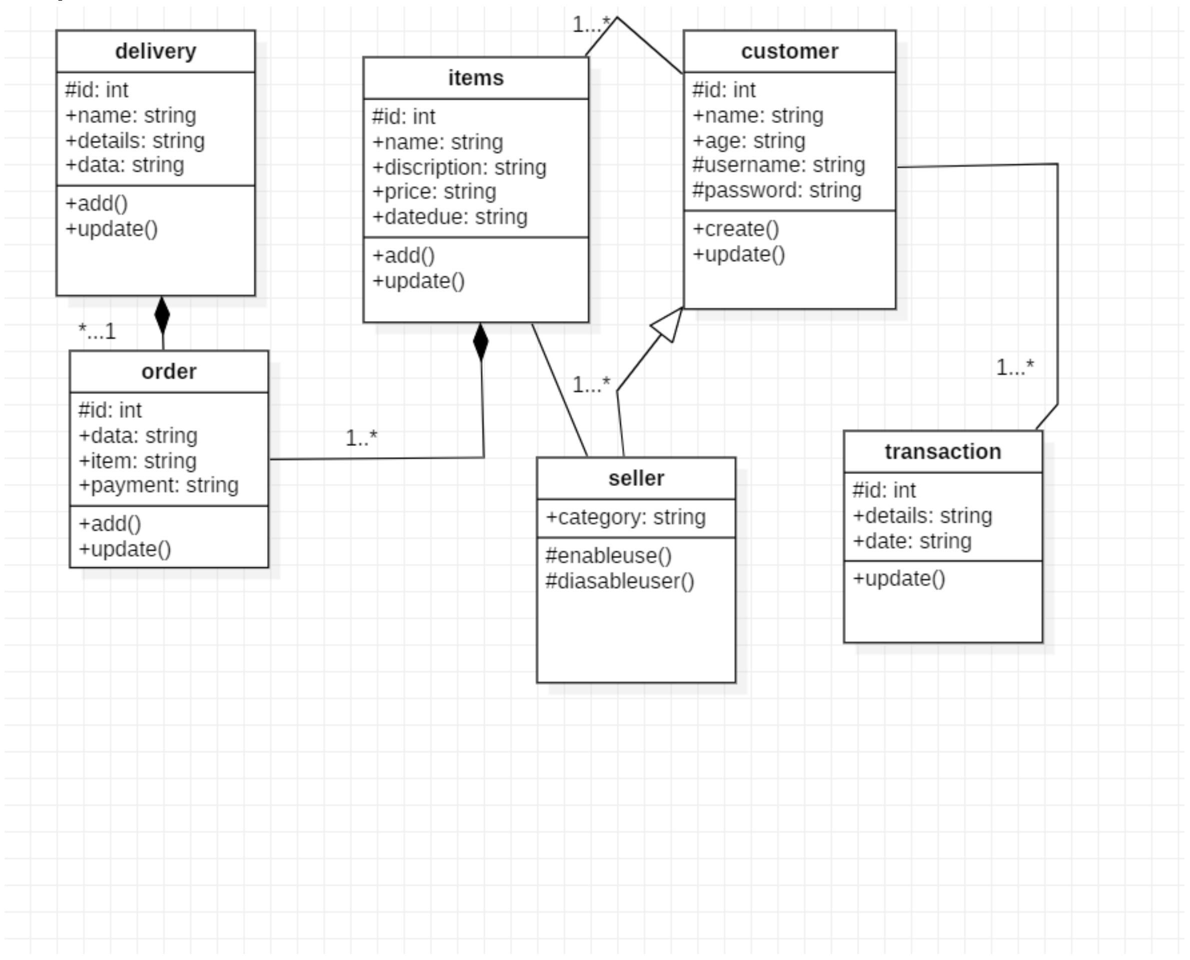
UML DIAGRAMS

ONLINE SHOPPING

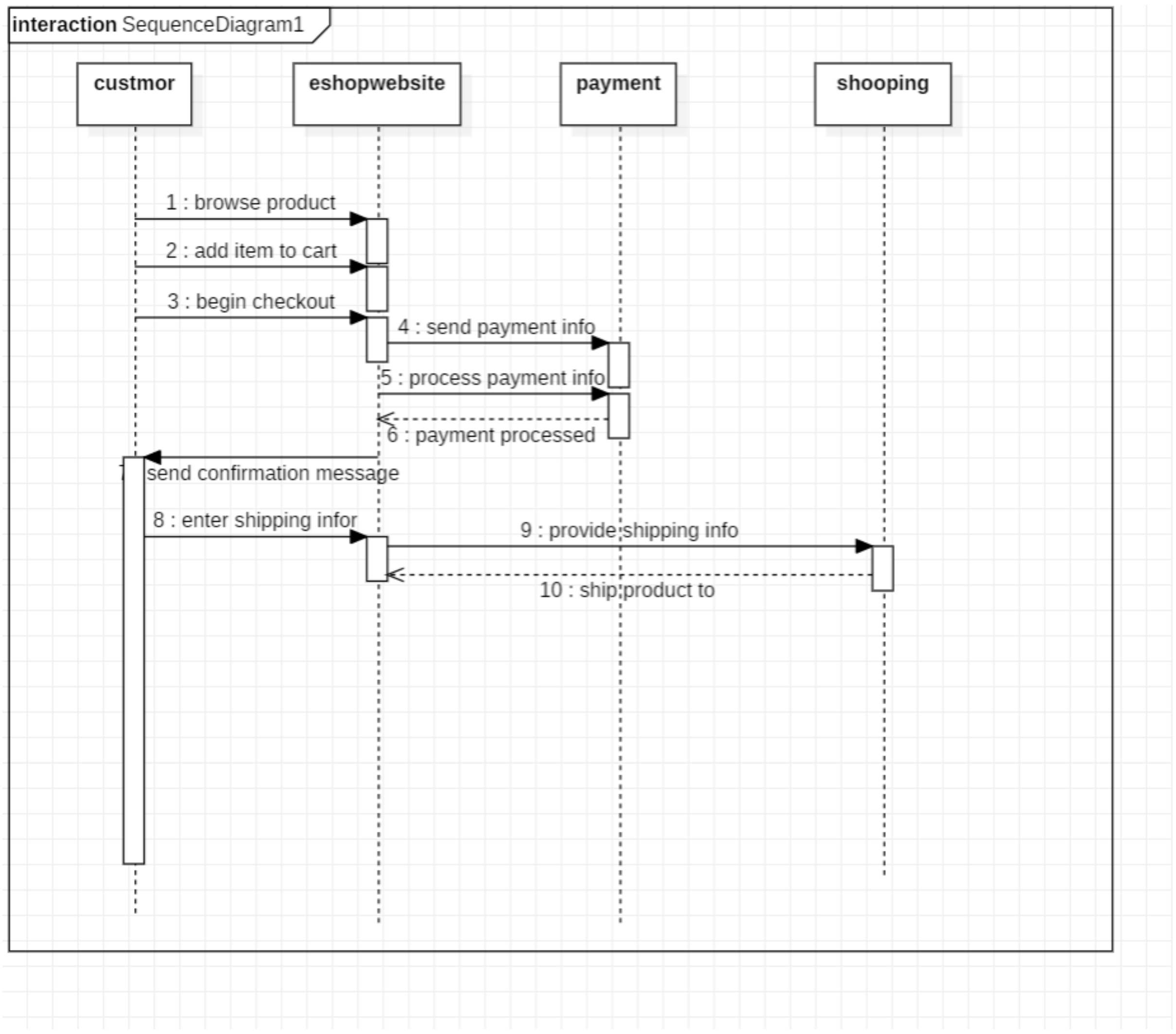
1 A) USE CASE DIAGRAM:



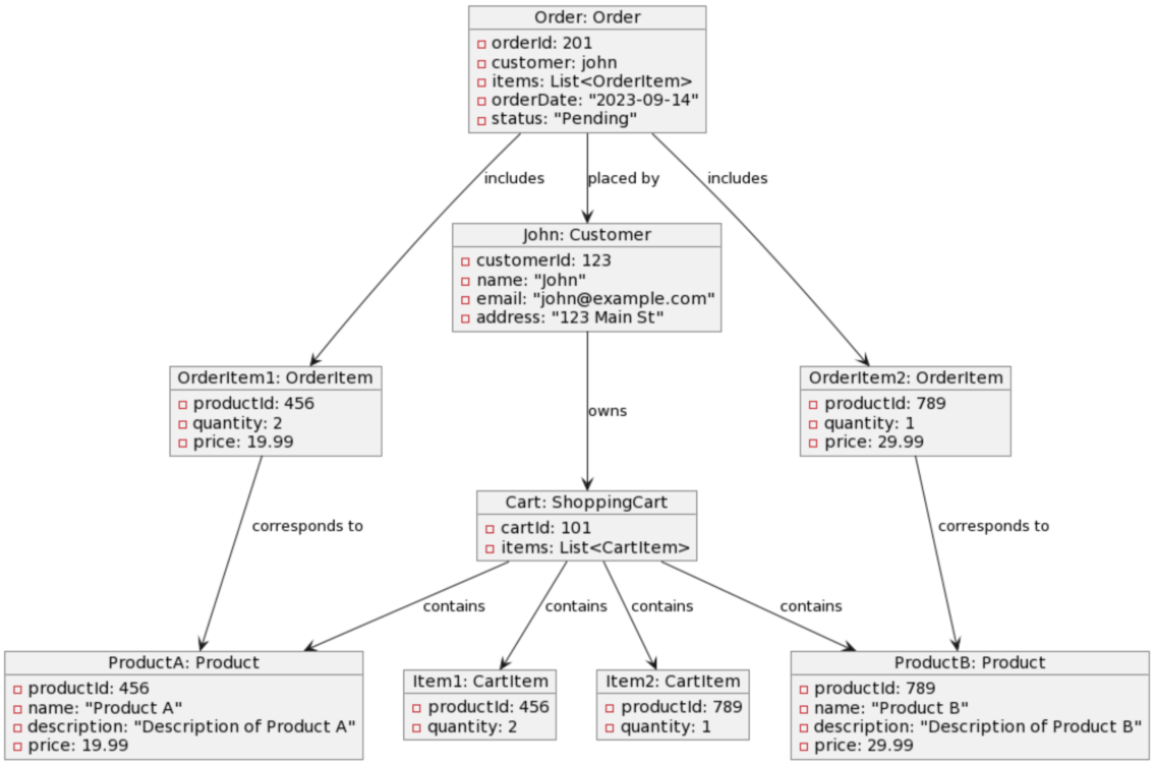
1 B) CLASS DIAGRAM:



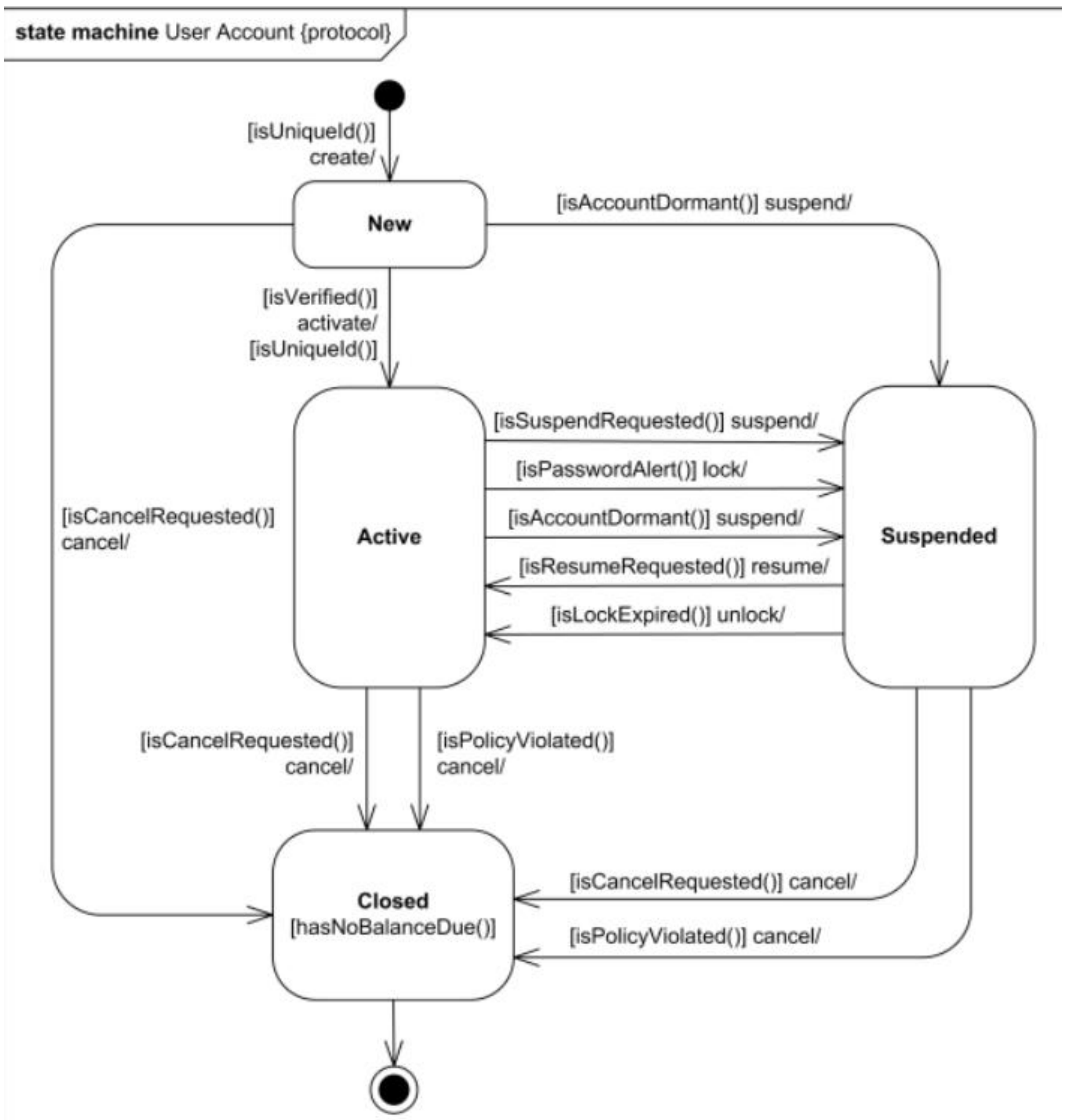
1 C) SEQUENCE DIAGRAM:



1 D) OBJECT DIAGRAM:

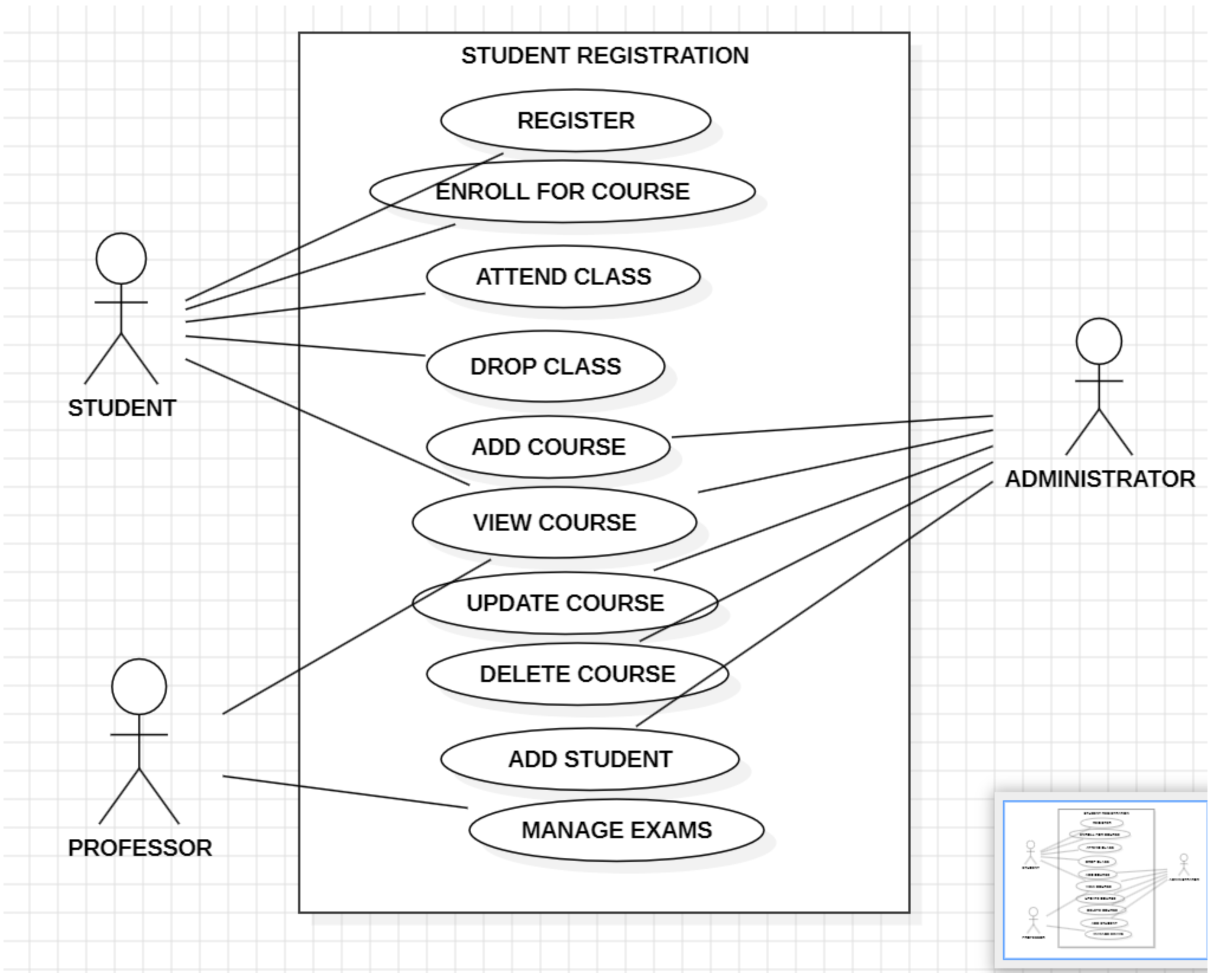


1 E) STATE ACTIVITY DIAGRAM:

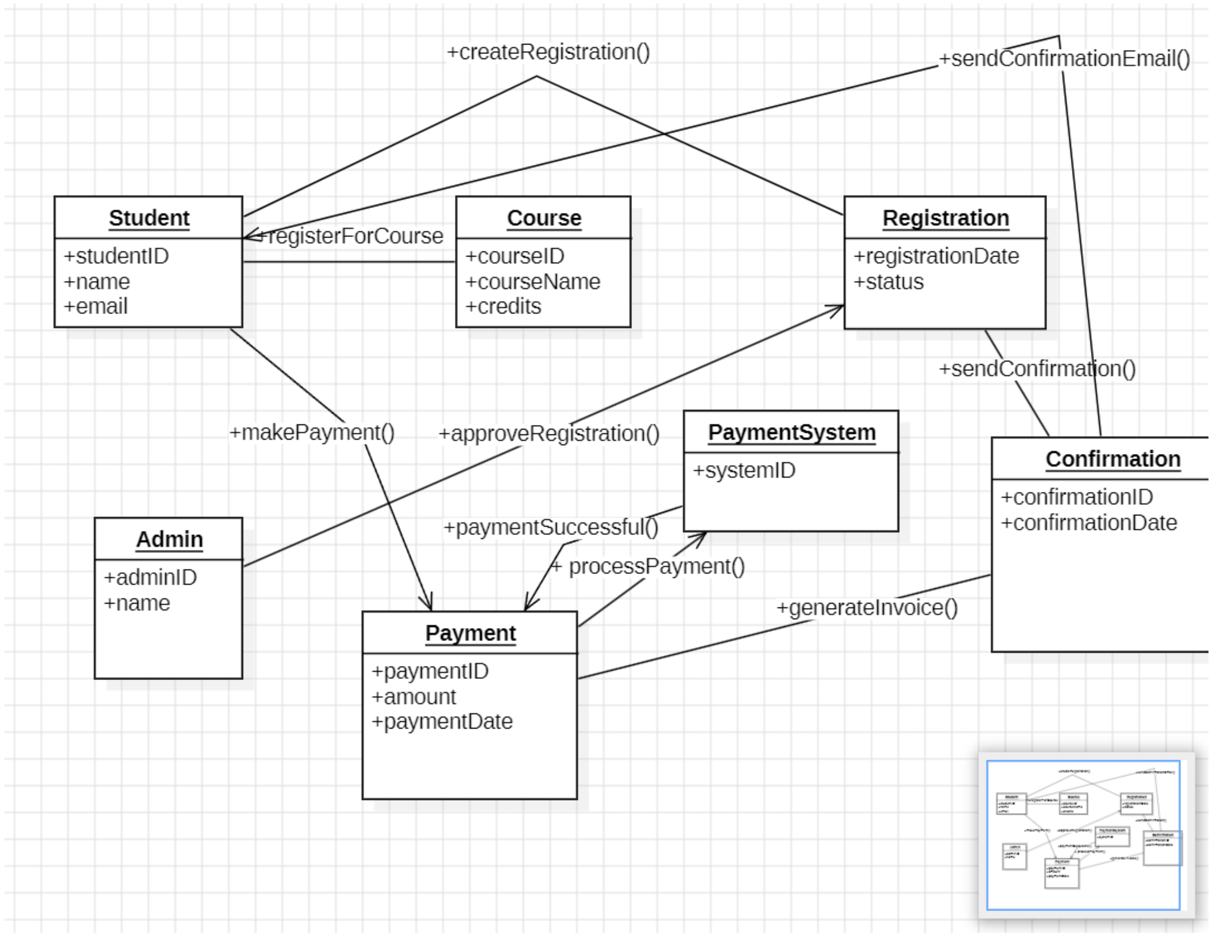

state machine User Account {protocol}
[isUniqueId()] create/
New
[isAccountDormant()] suspend/
[isVerified()] activate/ [isUniqueId()]
Active
Suspended
[isSuspendRequested()] suspend/
[isPasswordAlert()] lock/
[isAccountDormant()] suspend/
[isResumeRequested()] resume/
[isLockExpired()] unlock/
[isCancelRequested()]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

STUDENT REGISTRATION

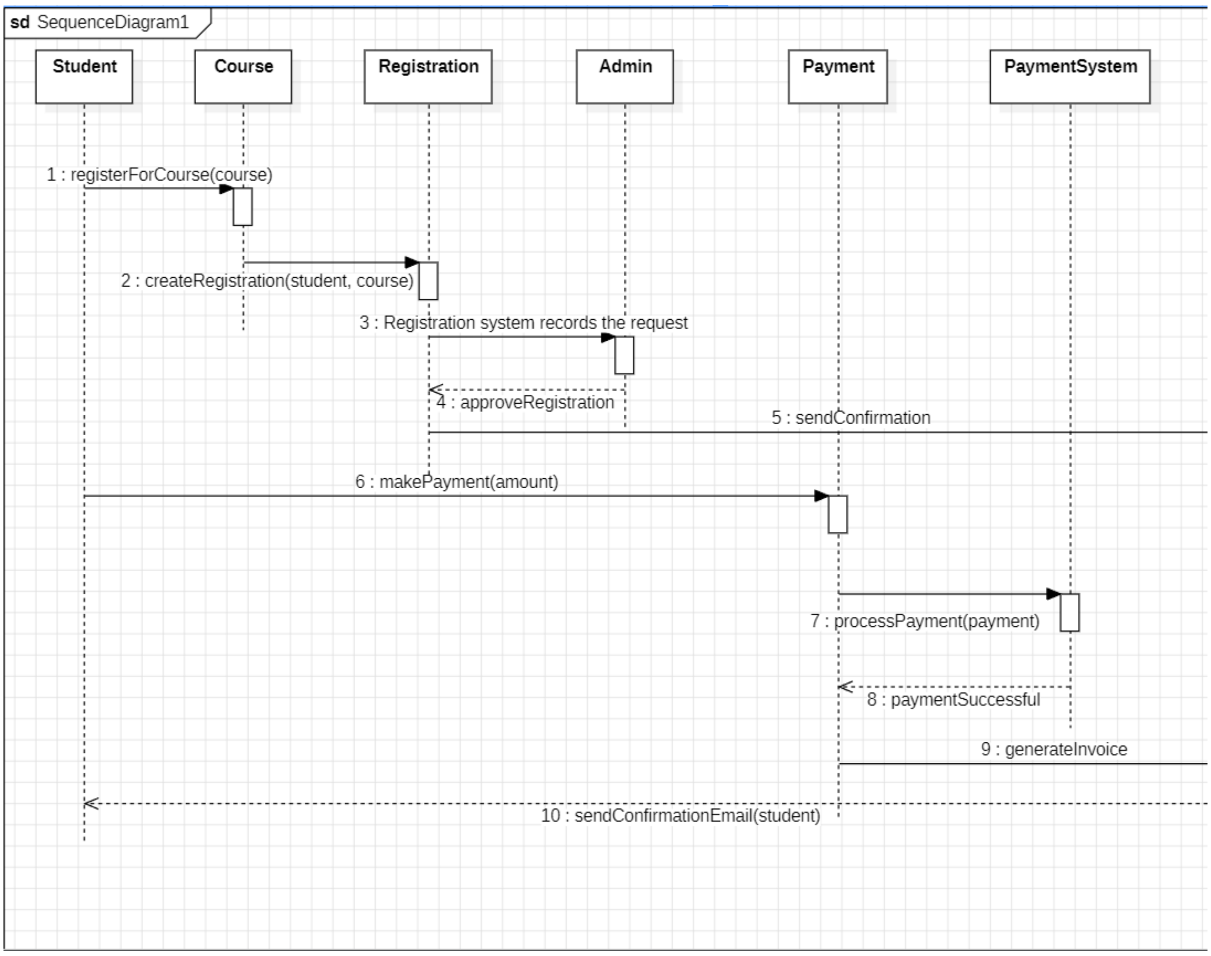
1 A) USE CASE DIAGRAM:



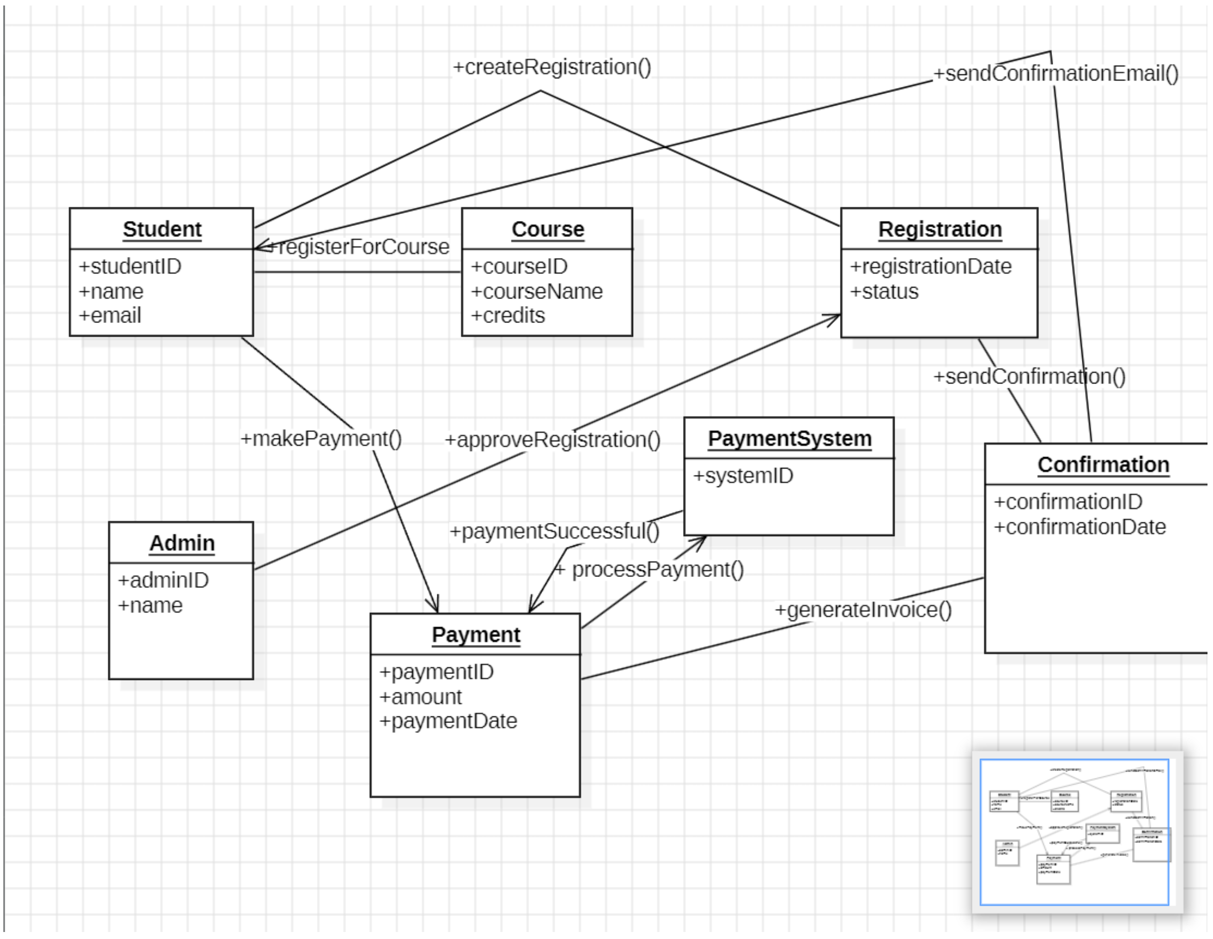
1 B) CLASS DIAGRAM:



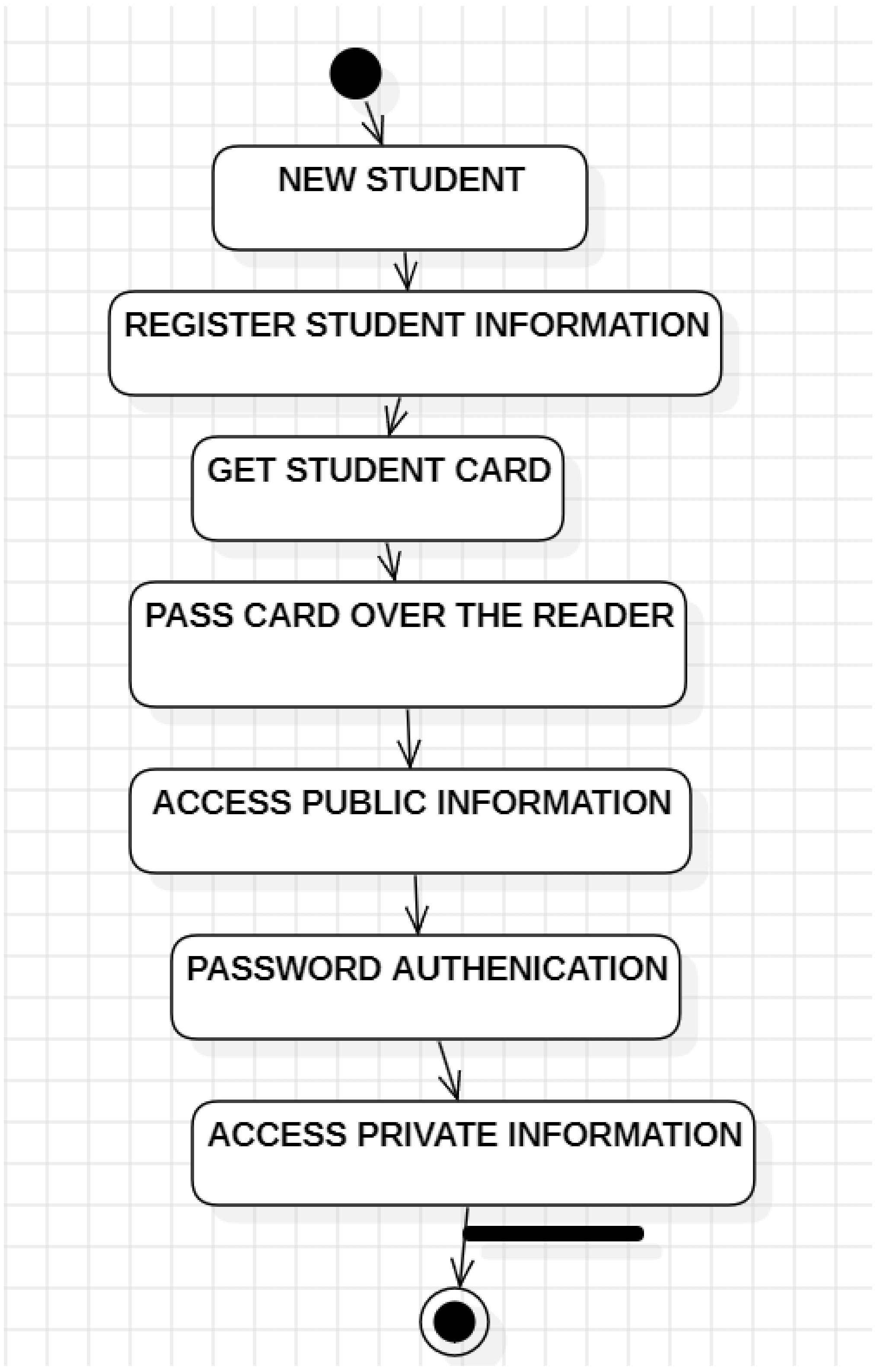
1 C) SEQUENCE DIAGRAM:



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

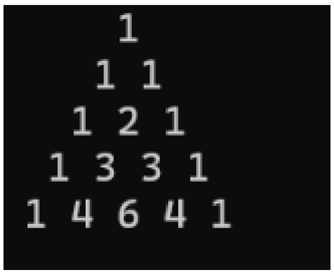
3.Basic Java Programs

3a) PascalTriangle :

Code:

```
public class PascalTriangle {
public static void main(String[] args) {
int rows = 5;
for (int i = 0; i < rows; i++) {
int number = 1;
for (int j = 0; j < rows - i; j++)
System.out.print(" ");
for (int j = 0; j <= i; j++) {
System.out.print(number + " ");
number = number * (i - j) / (j + 1);
}
System.out.println();
}
}
}
```

Output:



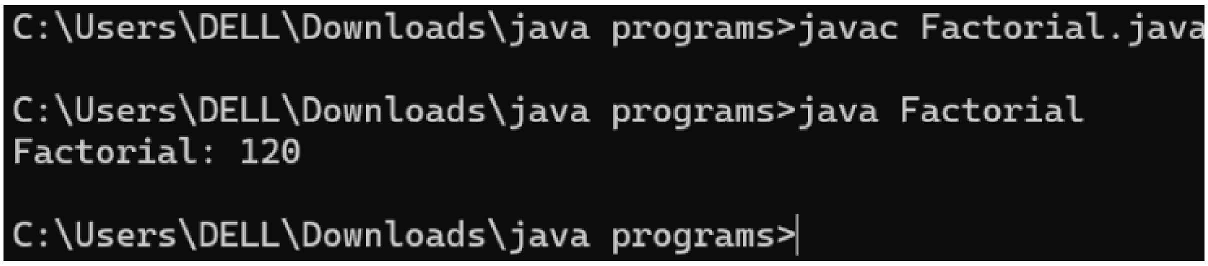
```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

3 b) Factorial:

Code:

```
public class Factorial {
public static void main(String[] args) {
int num = 5, fact = 1;
for (int i = 1; i <= num; i++) {
fact *= i; // Multiplying i with fact
}
System.out.println("Factorial: " + fact);
}
}
```

Output:



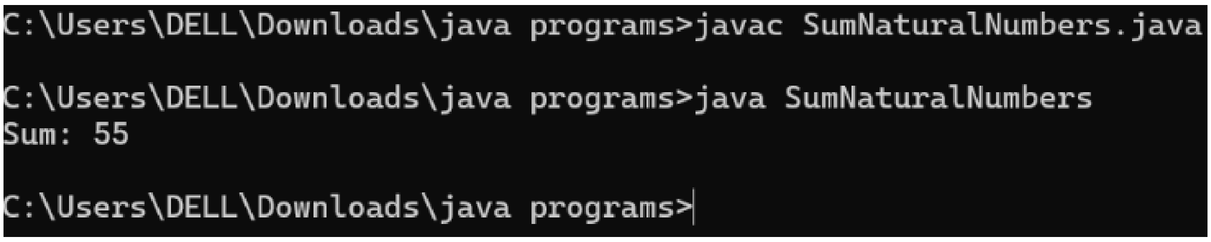
```
C:\Users\DELL\Downloads\java programs>javac Factorial.java

C:\Users\DELL\Downloads\java programs>java Factorial
Factorial: 120

C:\Users\DELL\Downloads\java programs>
```

3C) Sum Natural Numbers :

code:

```
public class SumNaturalNumbers {
public static void main(String[] args) {
int n = 10, sum = 0, i = 1;
while (i <= n) {
sum += i;
i++; // Increments i
}
System.out.println("Sum: " + sum);
}
}
```

Output:



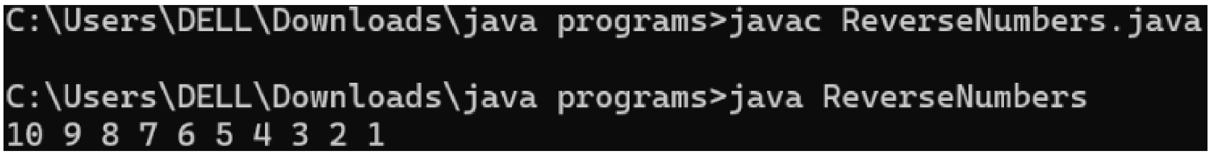
```
C:\Users\DELL\Downloads\java programs>javac SumNaturalNumbers.java

C:\Users\DELL\Downloads\java programs>java SumNaturalNumbers
Sum: 55

C:\Users\DELL\Downloads\java programs>
```

3d) Reverse Numbers :

Code:

```
public class ReverseNumbers {
public static void main(String[] args) {
for (int i = 10; i >= 1; i--) {
System.out.print(i + " ");
}
}
}
```

Output:

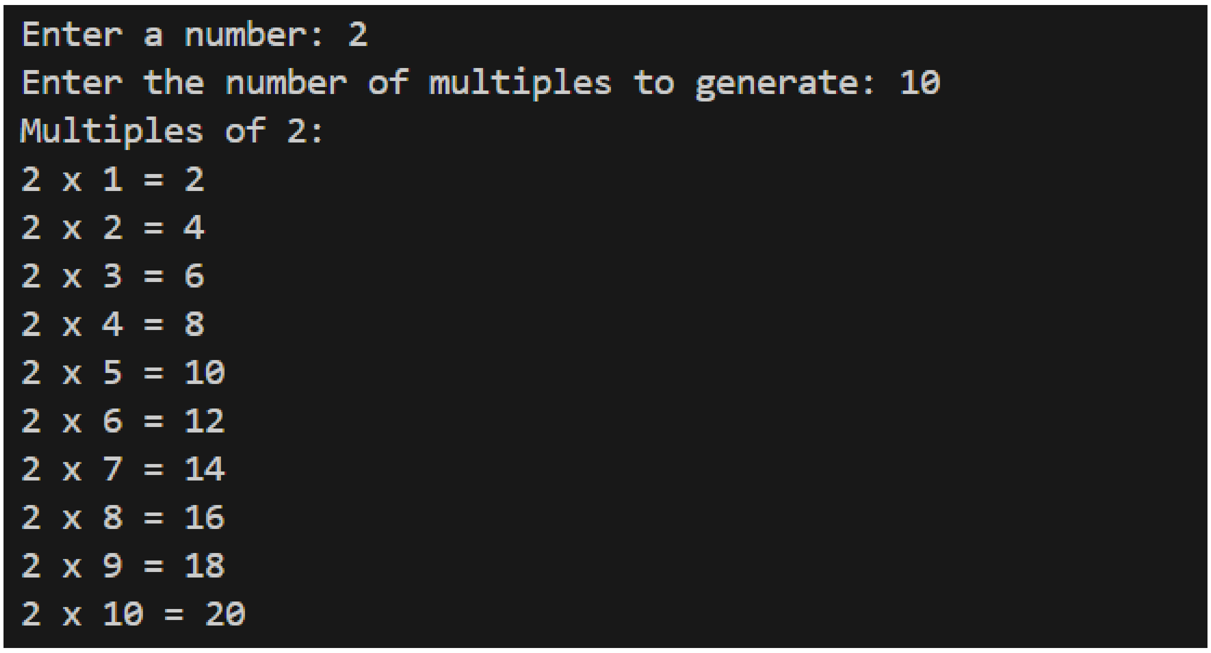


```
C:\Users\DELL\Downloads\java programs>javac ReverseNumbers.java

C:\Users\DELL\Downloads\java programs>java ReverseNumbers
10 9 8 7 6 5 4 3 2 1
```

3 e) Multiplication Calculator:

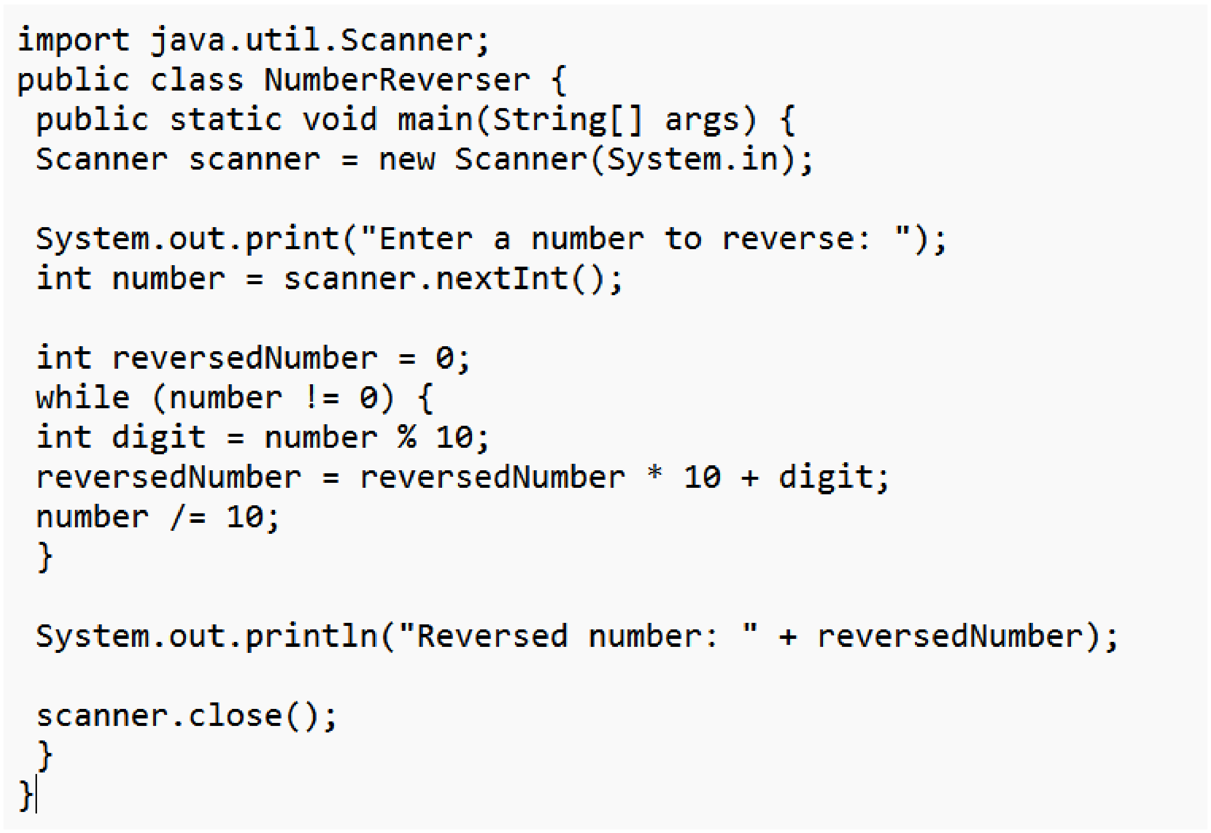
```
import java.util.Scanner;
public class MultiplicationCalculator {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
   int number = scanner.nextInt();
  System.out.print("Enter the number of multiples to generate: ");
 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



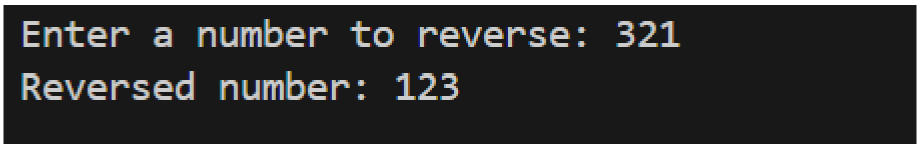
```
Enter a number: 2
Enter the number of multiples to generate: 10
Multiples of 2:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

3f) 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:

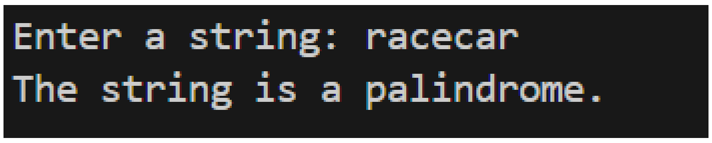


```
Enter a number to reverse: 321
Reversed number: 123
```

3g) Palindrome Checker:

```java
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:

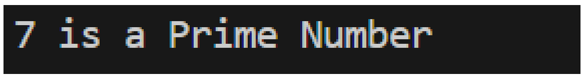


```
Enter a string: racecar
The string is a palindrome.
```

3h) 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
```

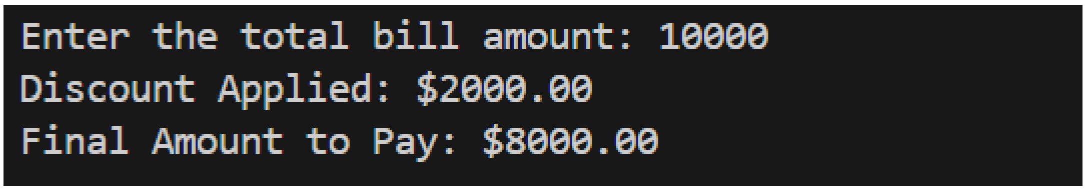
3i) Shopping Discount:

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:

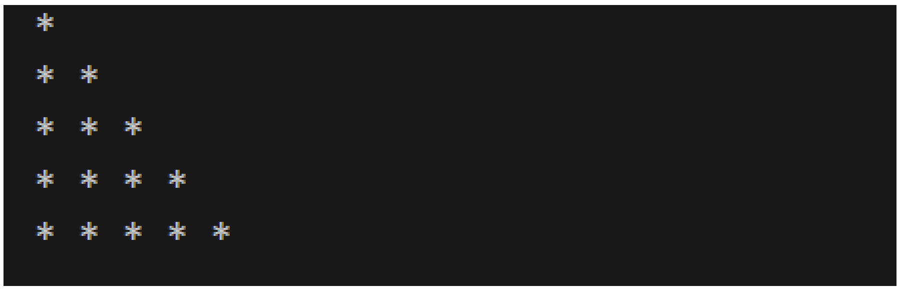


```
Enter the total bill amount: 10000
Discount Applied: $2000.00
Final Amount to Pay: $8000.00
```

3j) 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:



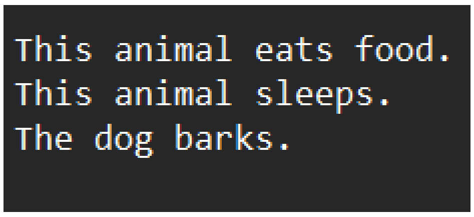
```
*
* *
* * *
* * * *
* * * * *
```

4. Single inheritance Programs

4 a) Animal inheritance

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }

    void sleep() {
        System.out.println("This animal sleeps.");
    }
}

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

public class SingleInheritanceExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
        myDog.sleep();
        myDog.bark();
    }
}
```

OUTPUT:

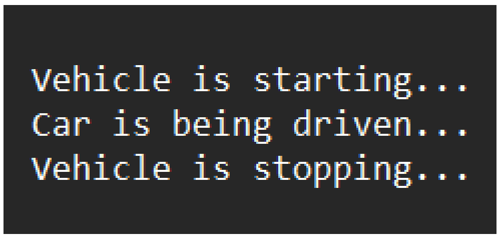


```
This animal eats food.
This animal sleeps.
The dog barks.
```

4b) Vehicle and Car:

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

  void stop() {
    System.out.println("Vehicle is stopping...");
  }
}

class Car extends Vehicle {
  void drive() {
    System.out.println("Car is being driven...");
  }
}

public class SingleInheritanceVehicle {
  public static void main(String[] args) {
    Car myCar = new Car();
    myCar.start();
    myCar.drive();
    myCar.stop();
  }
}
```

Output:



```
Vehicle is starting...
Car is being driven...
Vehicle is stopping...
```

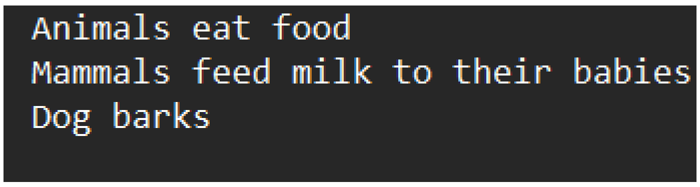
5) <u>Multilevel inheritance Programs</u>

5 a) Animal – mammal- dog

```
class Animal {
  void eat() {
    System.out.println("Animal eats food.");
  }
}

class Mammal extends Animal {
  void walk() {
    System.out.println("Mammals feed mik to their babies.");
  }
}

class Dog extends Mammal {
  void bark() {
    System.out.println("Dog barks.");
  }
}
```

```
public class MultilevelInheritanceExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();   // from Animal
        myDog.walk();  // from Mammal
        myDog.bark();  // from Dog
    }
}
```

Output:

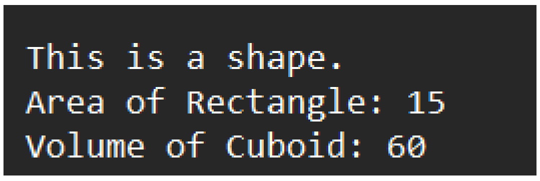


```
Animals eat food
Mammals feed milk to their babies
Dog barks
```

5 B) Shape → Rectangle → Cuboid

```
class Shape {
    void displayShape() {
        System.out.println("This is a shape.");
    }
}

class Rectangle extends Shape {
    int length = 5;
    int breadth = 3;

    void area() {
        int area = length * breadth;
        System.out.println("Area of Rectangle: " + area);
    }
}

class Cuboid extends Rectangle {
    int height = 4;

    void volume() {
        int volume = length * breadth * height;
        System.out.println("Volume of Cuboid: " + volume);
    }
}

public class MultilevelInheritance {
    public static void main(String[] args) {
        Cuboid myCuboid = new Cuboid();
        myCuboid.displayShape();
        myCuboid.area();
        myCuboid.volume();
    }
}
```

Output:



```
This is a shape.
Area of Rectangle: 15
Volume of Cuboid: 60
```

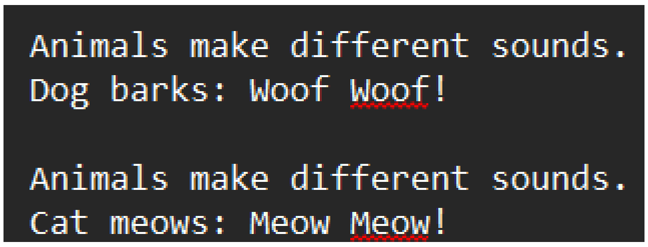
6.Hierarchical inheritance Programs

6 a) Animal → Dog, Cat

```
class Animal {
  void sound() {
    System.out.println("Animals make different sounds.");
  }
}

class Dog extends Animal {
  void bark() {
    System.out.println("Dog barks: Woof Woof!");
  }
}

class Cat extends Animal {
  void meow() {
    System.out.println("Cat meows: Meow Meow!");
  }
}

public class HierarchicalInheritance {
  public static void main(String[] args) {
    Dog dog = new Dog();
    Cat cat = new Cat();

    dog.sound();
    dog.bark();

    System.out.println();
```

```
        cat.sound();

        cat.meow();

    }

}
```

Output:



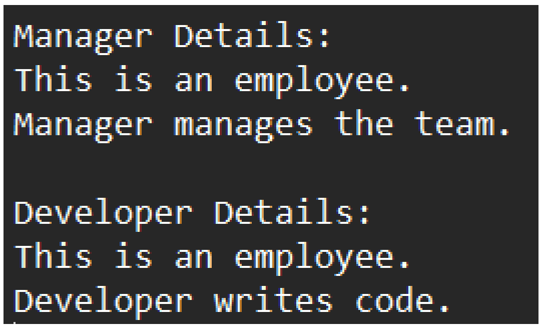
```
Animals make different sounds.
Dog barks: Woof Woof!

Animals make different sounds.
Cat meows: Meow Meow!
```

6 b) Employee → Manager, Developer

```
class Employee {

   void displayDetails() {

      System.out.println("This is an employee.");

   }

}


class Manager extends Employee {

   void manageTeam() {

      System.out.println("Manager manages the team.");

   }

}


class Developer extends Employee {

   void writeCode() {

      System.out.println("Developer writes code.");

   }

}
```

```
public class HierarchicalInheritance {
    public static void main(String[] args) {
        Manager m = new Manager();
        Developer d = new Developer();

        System.out.println("Manager Details:");
        m.displayDetails();
        m.manageTeam();

        System.out.println("\nDeveloper Details:");
        d.displayDetails();
        d.writeCode();
    }
}
```

Output:



```
Manager Details:
This is an employee.
Manager manages the team.

Developer Details:
This is an employee.
Developer writes code.
```

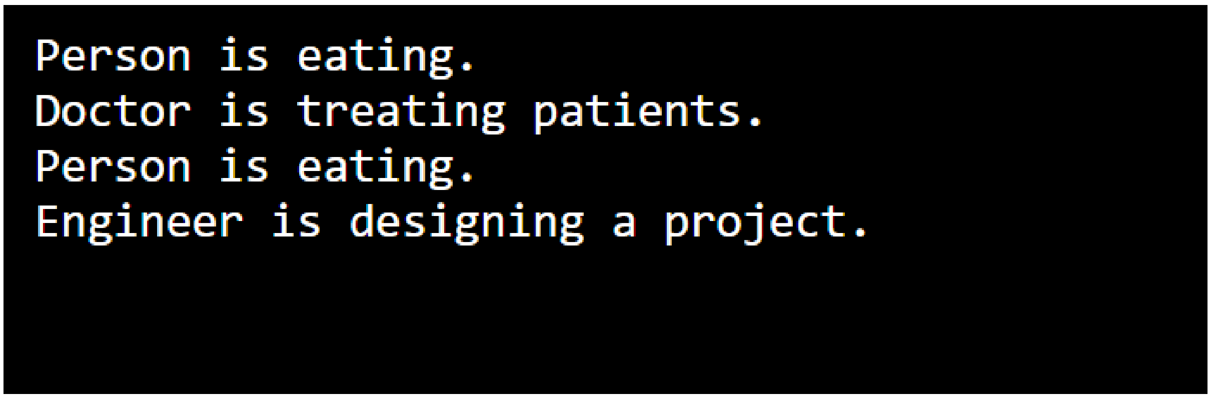
7.Hybrid inheritance Programs

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();
      d.performDuties();

      Engineer e = new Engineer();
      e.eat();
      e.performDuties();
   }
}
```

OUTPUT:



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

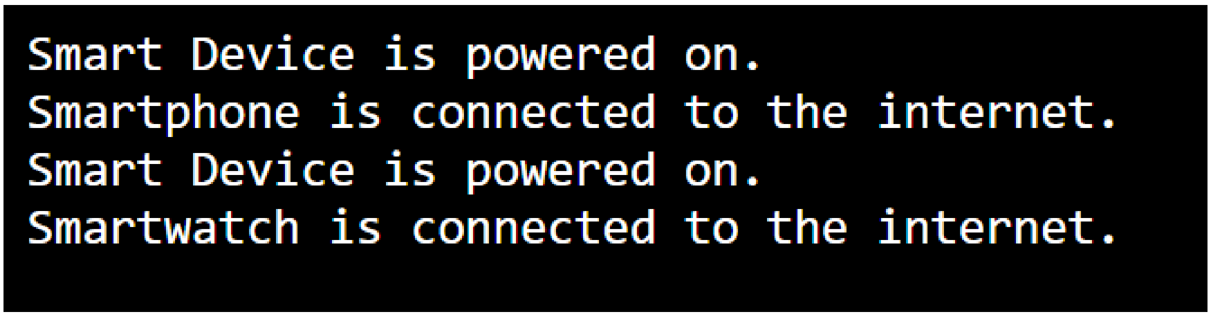
7 b) smart device- smart phone \ smart watch

```
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:



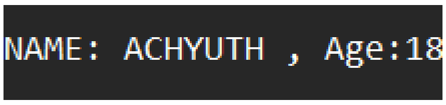
```
Smart Device is powered on.
Smartphone is connected to the internet.
Smart Device is powered on.
Smartwatch is connected to the internet.
```

8.Constructor Programs

8 a) student constructor

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

  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("Achyuth", 18);
    s1.display();
  }
}
```

Output:



```
NAME: ACHYUTH , Age:18
```

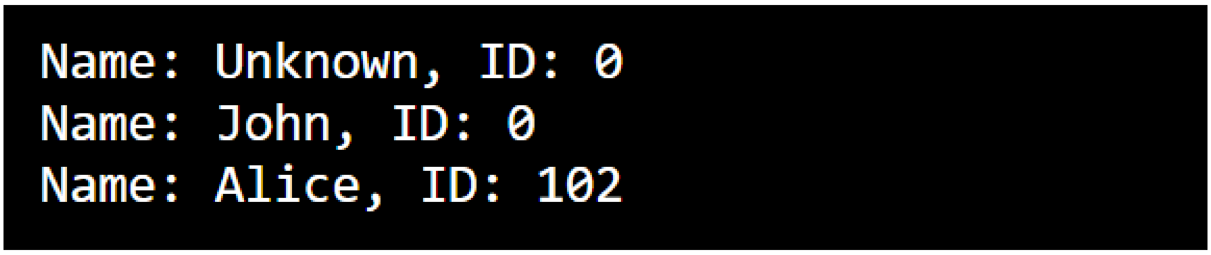
9.Constructor overloading Programs

9 a) Employee Constructor Overloading

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

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

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

  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

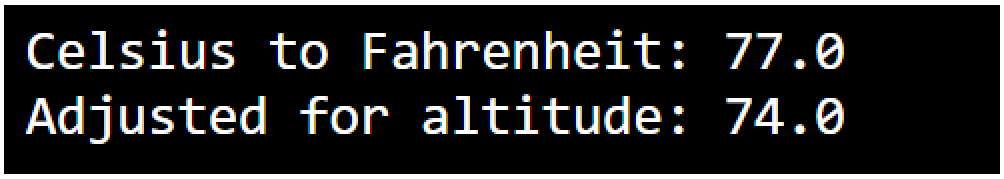
10 a) temperature converter overloading

```
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);
            }
        }

        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:

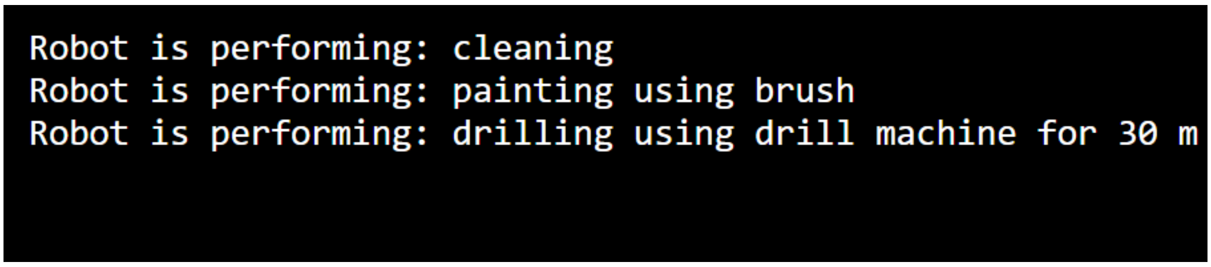


```
Celsius to Fahrenheit: 77.0
Adjusted for altitude: 74.0
```

10 b) robot task execution overloading

```
class Robot {
    void performTask(String task) {
        System.out.println("Robot is performing: " + task);
    }

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

    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
```

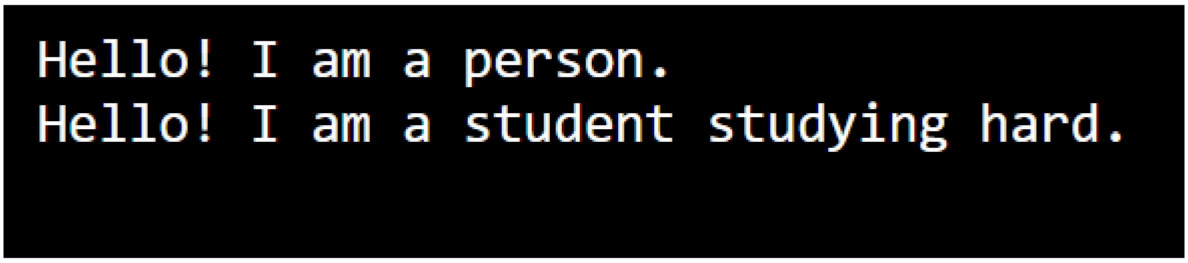
11.Method overriding Programs

11 a) parent-child greeting

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

class Student extends Person {
  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();

    Student s = new Student();
    s.greet();
  }
}
```

Output:

```
}
```

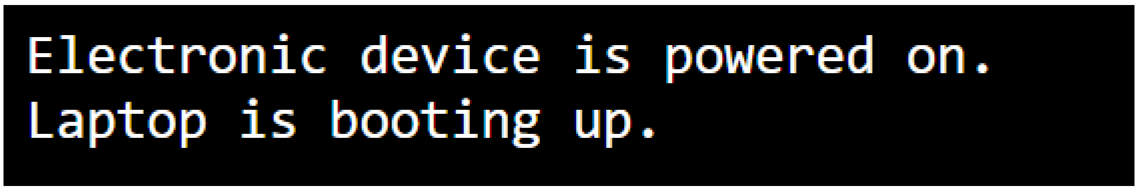


```
Hello! I am a person.
Hello! I am a student studying hard.
```

11 b) electronic device power

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

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

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

    Laptop myLaptop = new Laptop();
    myLaptop.powerOn();
  }
}
```

Output:



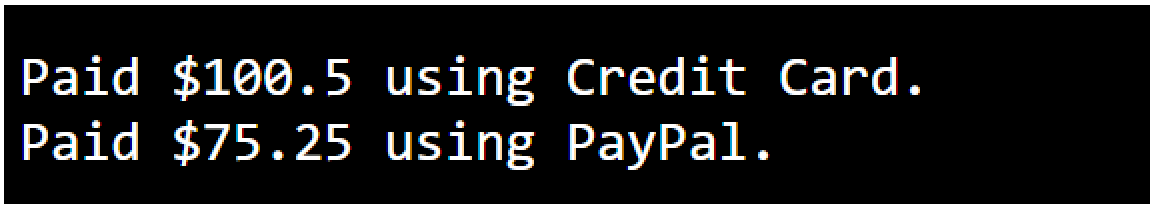
```
Electronic device is powered on.
Laptop is booting up.
```

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:



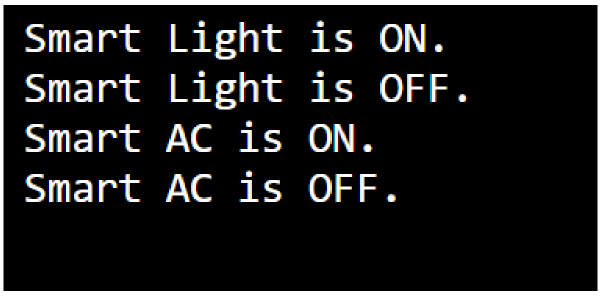
```
Paid $100.5 using Credit Card.
Paid $75.25 using PayPal.
```

12 b) smart home devices

```
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:

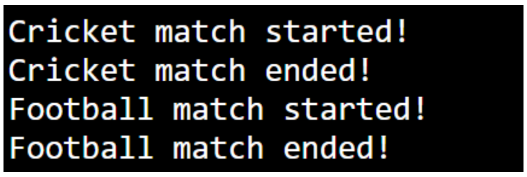


```
Smart Light is ON.
Smart Light is OFF.
Smart AC is ON.
Smart AC is OFF.
```

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:

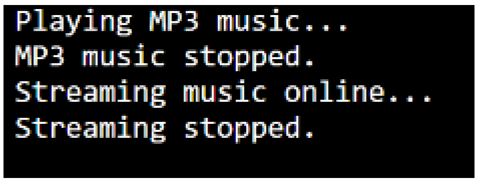


```
Cricket match started!
Cricket match ended!
Football match started!
Football match ended!
```

12 d) music playerinterface

```
 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:



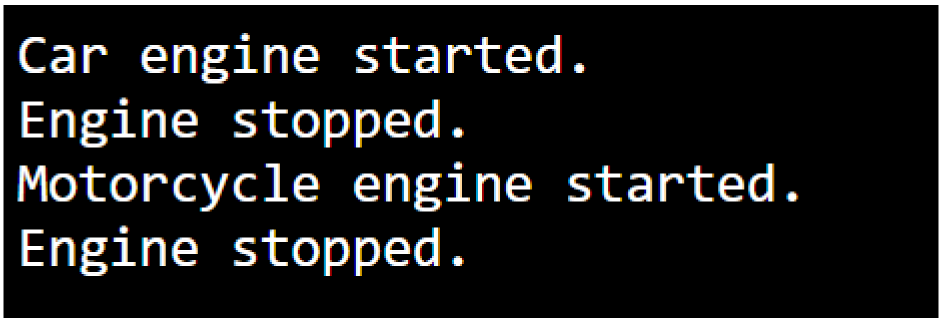
```
Playing MP3 music...
MP3 music stopped.
Streaming music online...
Streaming stopped.
```

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:

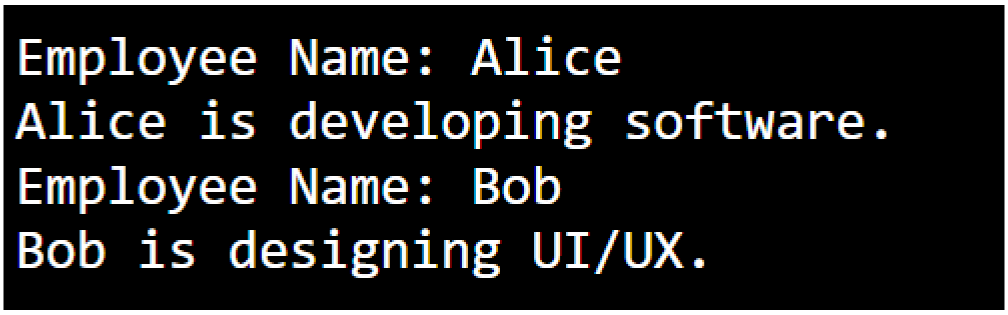


```
Car engine started.
Engine stopped.
Motorcycle engine started.
Engine stopped.
```

13 b) employee

```
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:

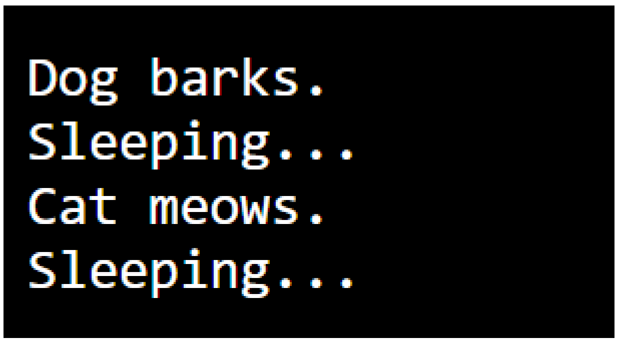


```
Employee Name: Alice
Alice is developing software.
Employee Name: Bob
Bob is designing UI/UX.
```

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:



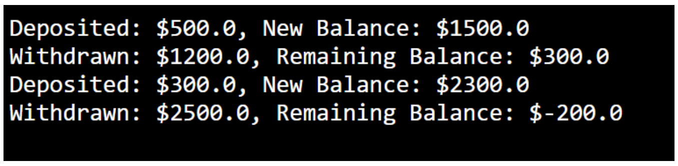
```
Dog barks.
Sleeping...
Cat meows.
Sleeping...
```

13 d) bank account

```
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:



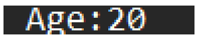
```
Deposited: $500.0, New Balance: $1500.0
Withdrawn: $1200.0, Remaining Balance: $300.0
Deposited: $300.0, New Balance: $2300.0
Withdrawn: $2500.0, Remaining Balance: $-200.0
```

14.Encapsulation Programs

14 a) Encapsulation with Age Validation

```
class Student {
  private int age;

  public void setAge(int a) {
    if (a > 0 && a < 150)
      age = a;
    else
      System.out.println("Invalid age");
  }

  public int getAge() {
        return age;
    }
}

public class Main2 {
    public static void main(String[] args) {
        Student s = new Student();
        s.setAge(20);
        System.out.println("Age: " + s.getAge());
    }
}
```

Output:

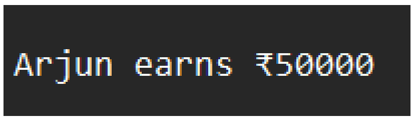


```
Age:20
```

14 b) Multiple Fields

```
class Employee {
    private String name;
    private int salary;

    public void setDetails(String n, int s) {
        name = n;
        salary = s;
    }

    public String getName() {
        return name;
    }
    public int getSalary() {
        return salary;
    }
}

public class Main3 {
    public static void main(String[] args) {
        Employee e = new Employee();
        e.setDetails("Arjun", 50000);
        System.out.println(e.getName() + " earns ₹" + e.getSalary());
    }
}
```

Output:



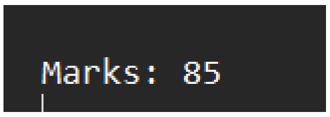
```
Arjun earns ₹50000
```

14 c) Report Card Marks

```
class ReportCard {
    private int marks;

    public void setMarks(int m) {
        if (m >= 0 && m <= 100)
            marks = m;
    }

    public int getMarks() {
        return marks;
    }
}
```

```
public class Main7 {
    public static void main(String[] args) {
        ReportCard r = new ReportCard();
        r.setMarks(85);
        System.out.println("Marks: " + r.getMarks());
    }
}
```

Output:

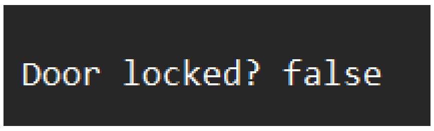


```
Marks: 85
```

14 d) Door Lock System

```
class Door {
    private boolean isLocked = true;
 public void unlock() {
 isLocked = false;
    }

    public boolean isLocked() {
        return isLocked;
    }
}

public class Main8 {
    public static void main(String[] args) {
        Door d = new Door();
        d.unlock();
        System.out.println("Door locked? " + d.isLocked());
    }
}
```

Output:



```
Door locked? false
```

15. packages Programs

15 a) Custom pakage

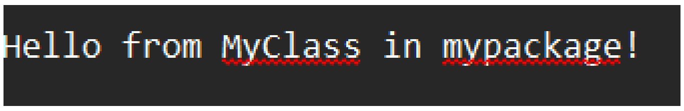
Package file:

```
package mypackage;
public class MyClass {
   public void showMessage() {
      System.out.println("Hello from MyClass in mypackage!");
   }
}
```

Main class:

```
import mypackage.MyClass;
public class Main {
   public static void main(String[] args) {
      MyClass obj = new MyClass();
      obj.showMessage();
   }
}
```

Output:



```
Hello from MyClass in mypackage!
```

15 b) user defined package

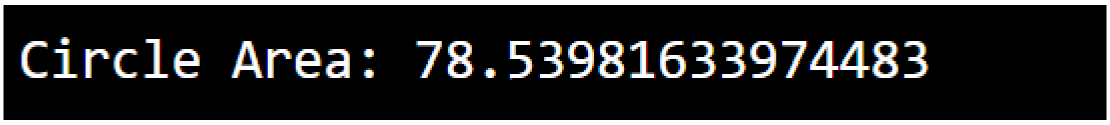
Package file: package shapes;

```
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:

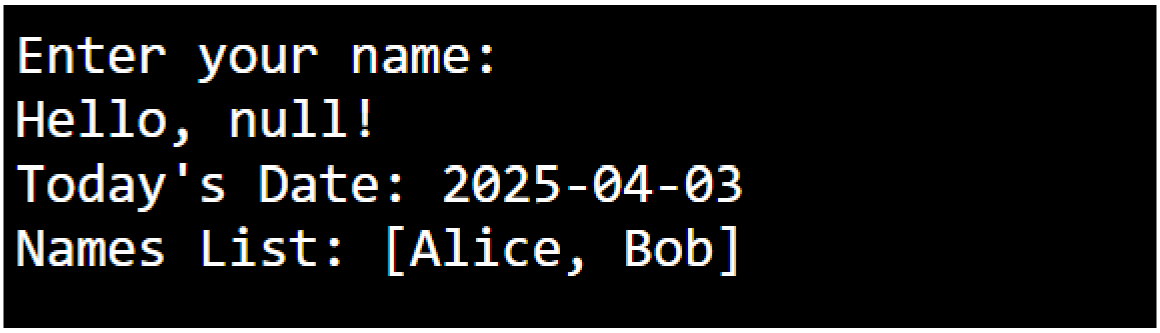


```
Circle Area: 78.53981633974483
```

15 c) built in packages

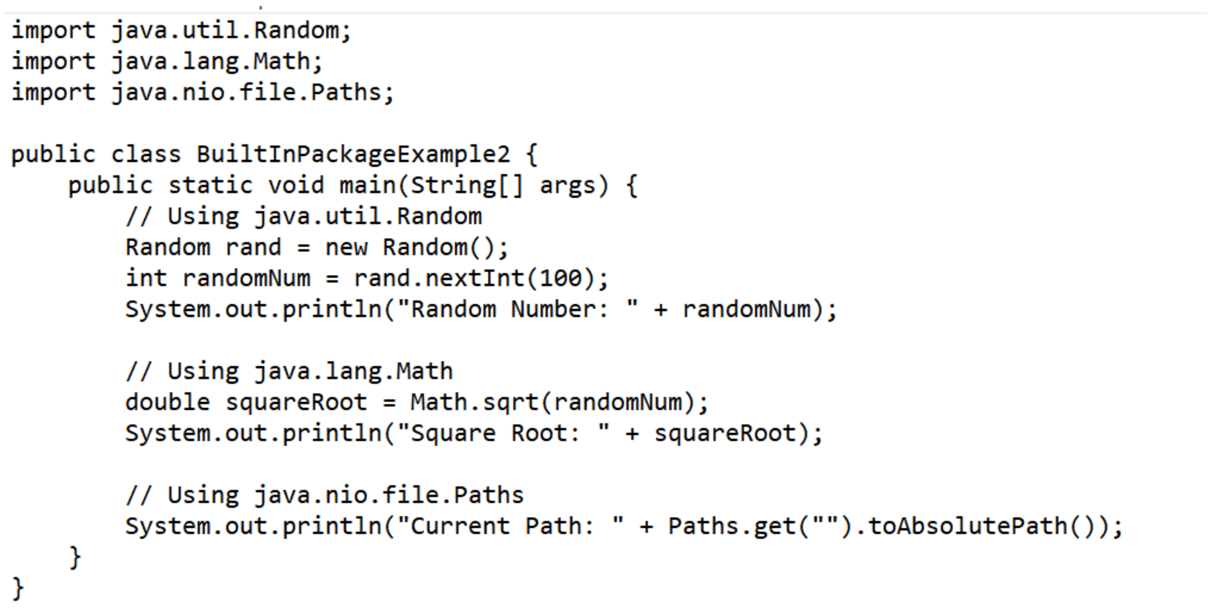
```java
import java.util.ArrayList;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.time.LocalDate;

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:



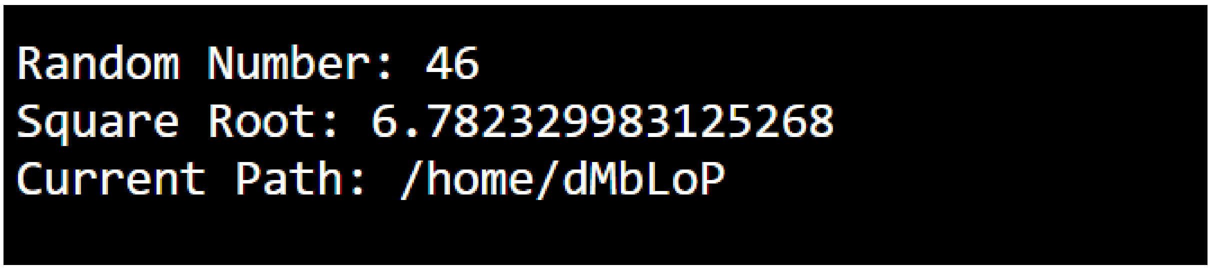
```
Enter your name:
Hello, null!
Today's Date: 2025-04-03
Names List: [Alice, Bob]
```

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
```

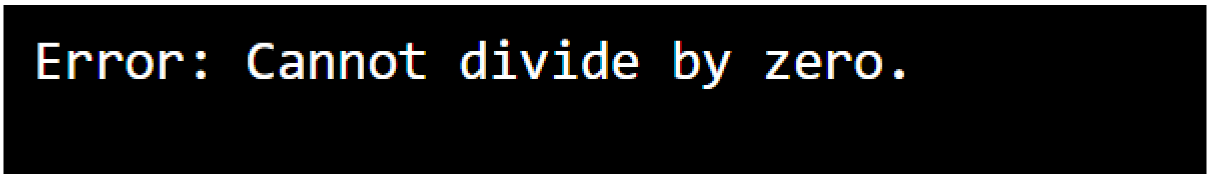
16 <u>exception handling Programs</u>

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:

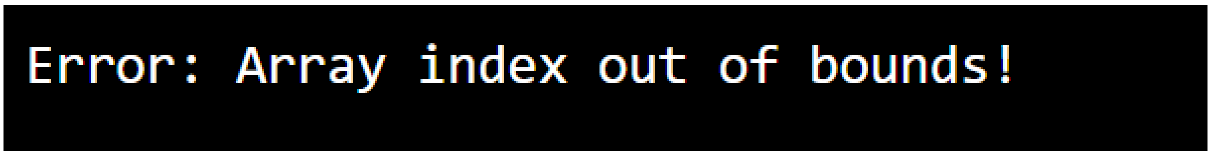


```
Error: Cannot divide by zero.
```

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:

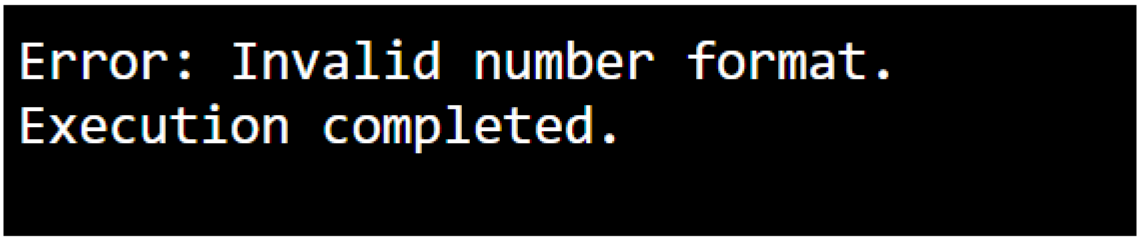


```
Error: Array index out of bounds!
```

16 c) invalid number format

```
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:



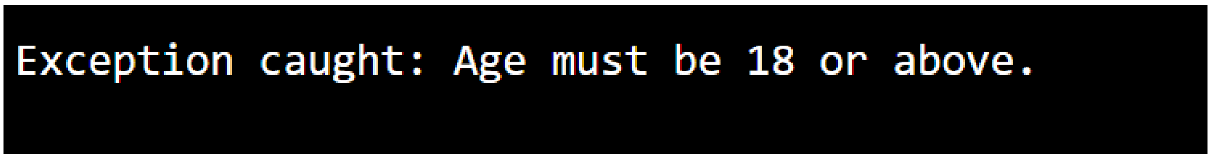
```
Error: Invalid number format.
Execution completed.
```

16 d) age exception

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

```java
    }
}

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:



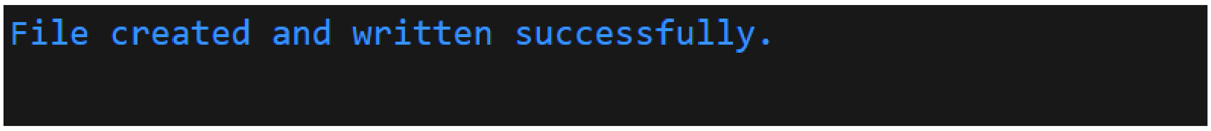
```
Exception caught: Age must be 18 or above.
```

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:



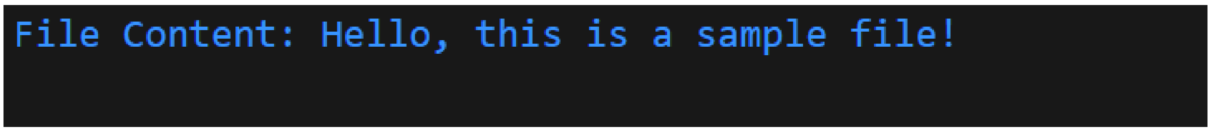
```
File created and written successfully.
```

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:



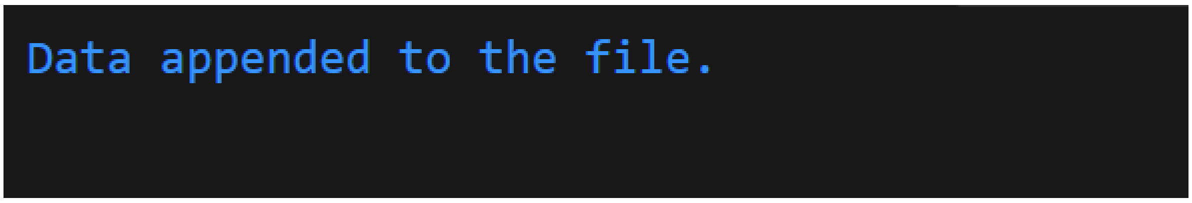
```
File Content: Hello, this is a sample file!
```

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:

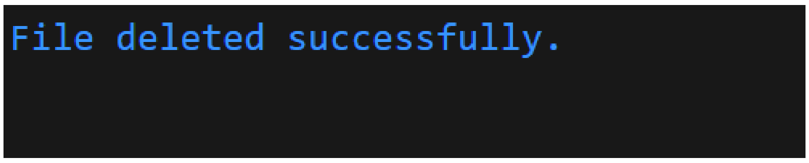


```
Data appended to the file.
```

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.
```