

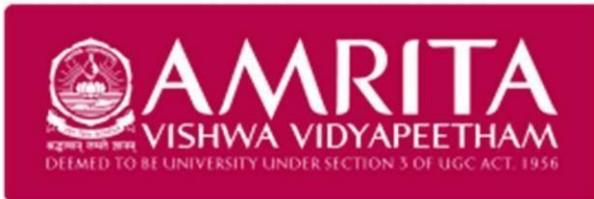
SCHOOL OF
COMPUTING

M.REDDY MANISH

CH.SC.U4CSE24129

**OBJECT ORIENTED PROGRAMMING
(23CSE111)**

LAB RECORD



SCHOOL OF
COMPUTING

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.U4CSE24129 – M. Reddy Manish** 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 held on

Internal Examiner 1

Internal Examiner 2

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	Library Management system	
	1.a) Use Case Diagram	1
	1.b) Class Diagram	2
	1.c) Sequence Diagram	3
2.	Student Management system	
	2.a) Use Case Diagram	4
	2.b) Class Diagram	5
	2.c) Sequence Diagram	6
3.	Restaurant Management system	
	3.a) Use Case Diagram	7
	3.b) Class Diagram	8
	3.c) Sequence Diagram	9
4.	BASIC JAVA PROGRAMS	
	4.a) Hypotenuse of a triangle	10
	4.b) Switch case	11
	4.c) If else statement	13
	4.d) While loop	15
	4.e) For loop	16
	4.f) Shape of a symbol	17
	4.g) Factorial	19
	4.h) Reversing a string	21

	4.i) Prime Number	22
	4.j) Palindrome	24

S.NO	INHERITANCE	PAGE.NO
------	-------------	---------

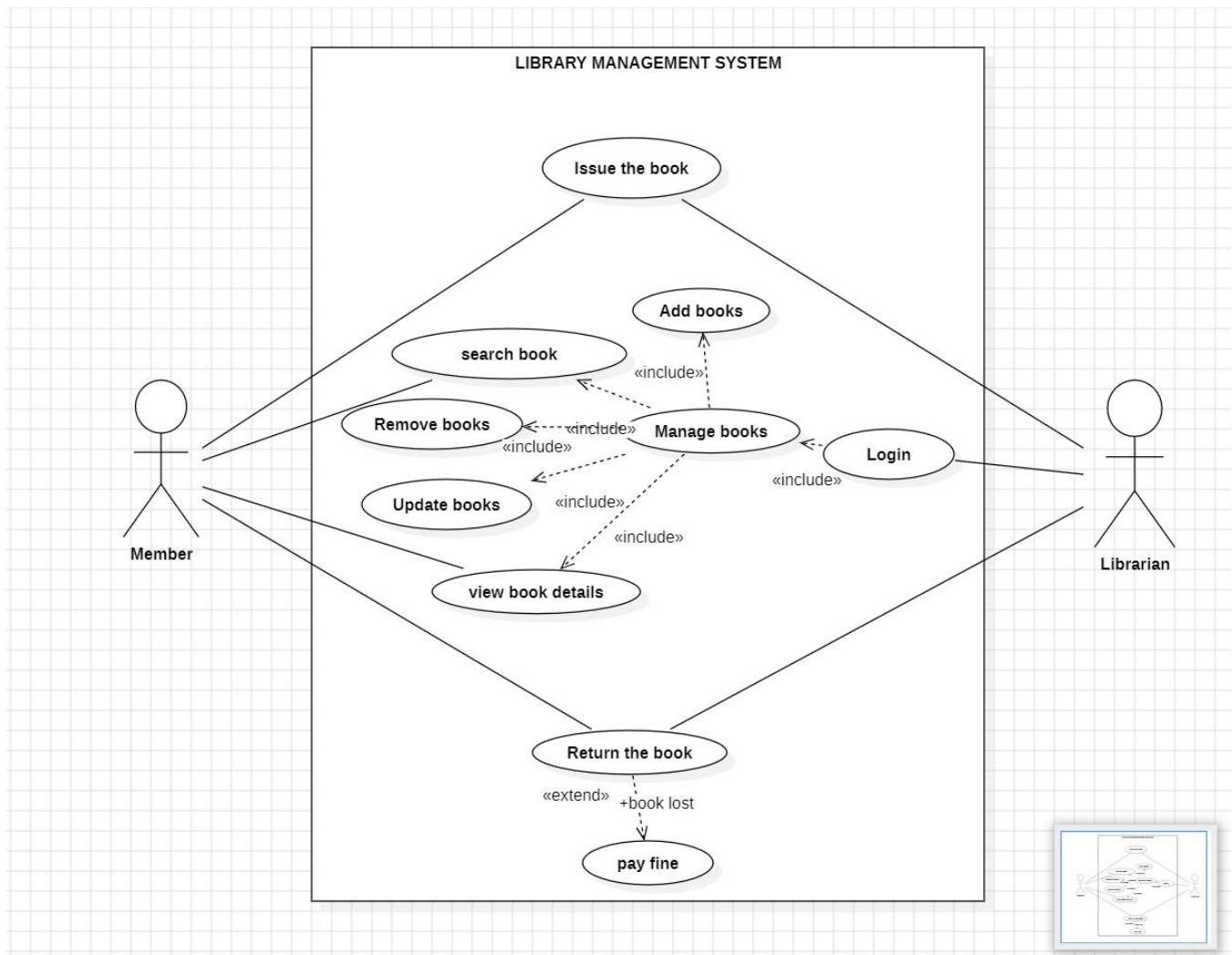
5.	Multilevel INHERITANCE PROGRAMS	38
	5.a) APPLIANCE	
	5.b) VEHICLE	
6.	HYBRID INHERITANCE PROGRAMS	41
	6.a) VEHICLE	
	6.b) ANIMAL	
7.	SINGLE INHERITANCE PROGRAMS	46
	7.a) EMPLOYEE	
	7.b) PERSON	
8.	HIERARCHICAL HYBRID INHERITANCE PROGRAMS	51
	8.a) SHAPE	
	8.b) PERSON	
	POLYMORPHISM	
9.	CONSTRUCTOR PROGRAMS	56
	9.a) USER DETAILS	
10.	CONSTRUCTOR OVERLOADING PROGRAMS	58
	10.a) EMPLOYEE	
11.	METHOD OVERLOADING PROGRAMS	60
	11.a) MULTIPLICATION	
	11.b) PRICE DISCOUNT	
12.	METHOD OVERRIDING PROGRAMS	64
	12.a) VEHICLE	
	12.b) BANK	
	ABSTRACTION	
13.	INTERFACE PROGRAMS	68

	13.a) CALCULATOR	
	13.b) SHOPING CART	
	13.c) TEST PRINTER	
	13.d) INTERFACE4	
14.	ABSTRACT CLASS PROGRAMS	76
	14.a) Game character	
	14.b) Payment	
	14.c) Withdraw	
	14.d) Circle Shape	
	ENCAPSULATION	
15.	ENCAPSULATION PROGRAMS	85
	15.a) STUDENT DETAILS	
	15.b) STUDENT	
	15.c) PERSON AGE	
	15.d) VEHICLE	
16.	PACKAGES PROGRAMS	95
	16.a) User Defined Packages	
	16.b) User Defined Packages	
	16.c) Built – in Package (3 Packages)	
	16.d) Built – in Package (3 Packages)	
17.	EXCEPTION HANDLING PROGRAMS	103
	17.a) Array Exception	
	17.b) Divisibility	
	17.c) Eligibility	
	17.d) Age	
18.	FILE HANDLING PROGRAMS	108
	18.a) Error File	
	18.b) New File	
	18.c) Delete File	
	18.d) Append File	

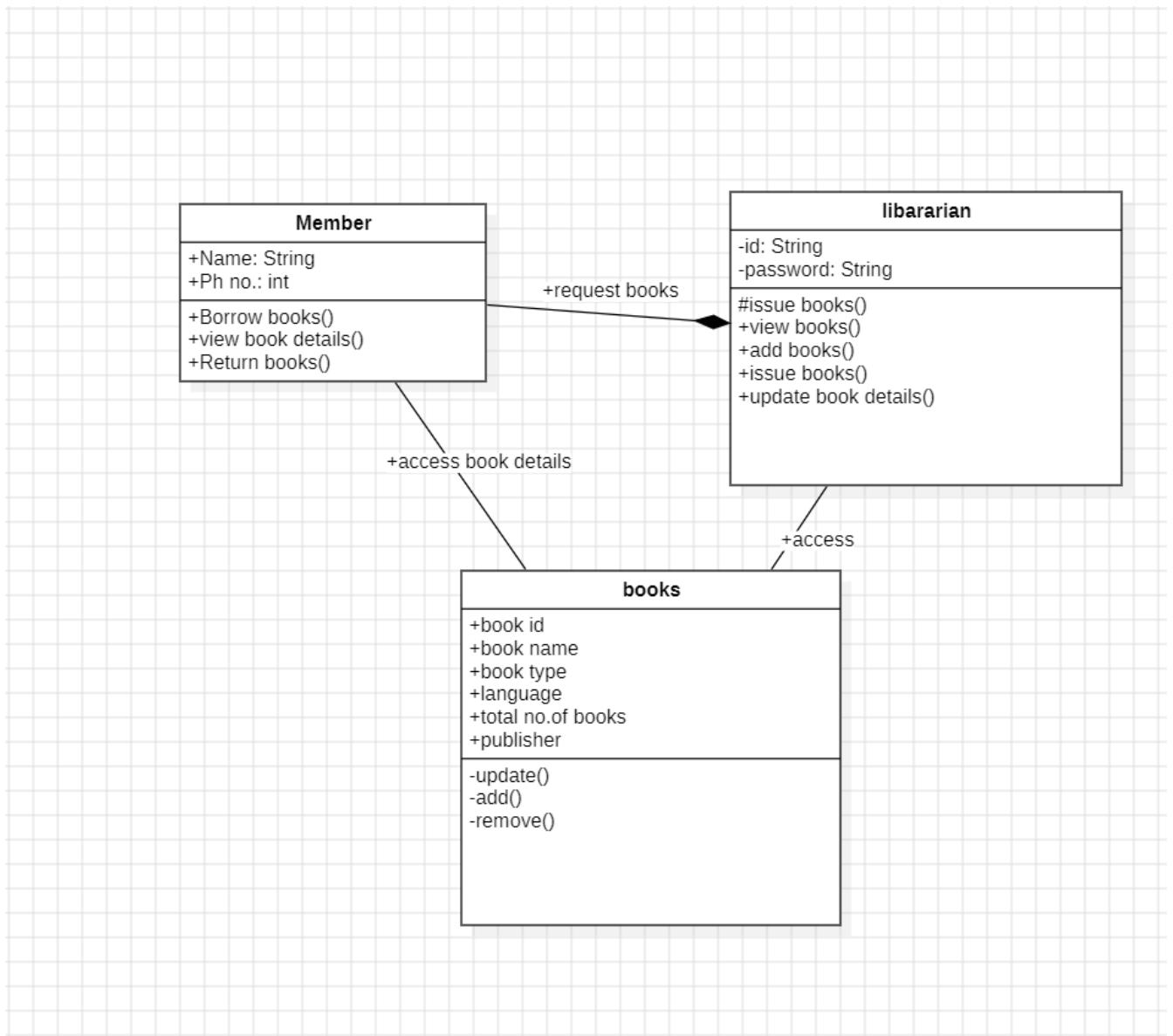
UML DIAGRAMS

1. Library Management system

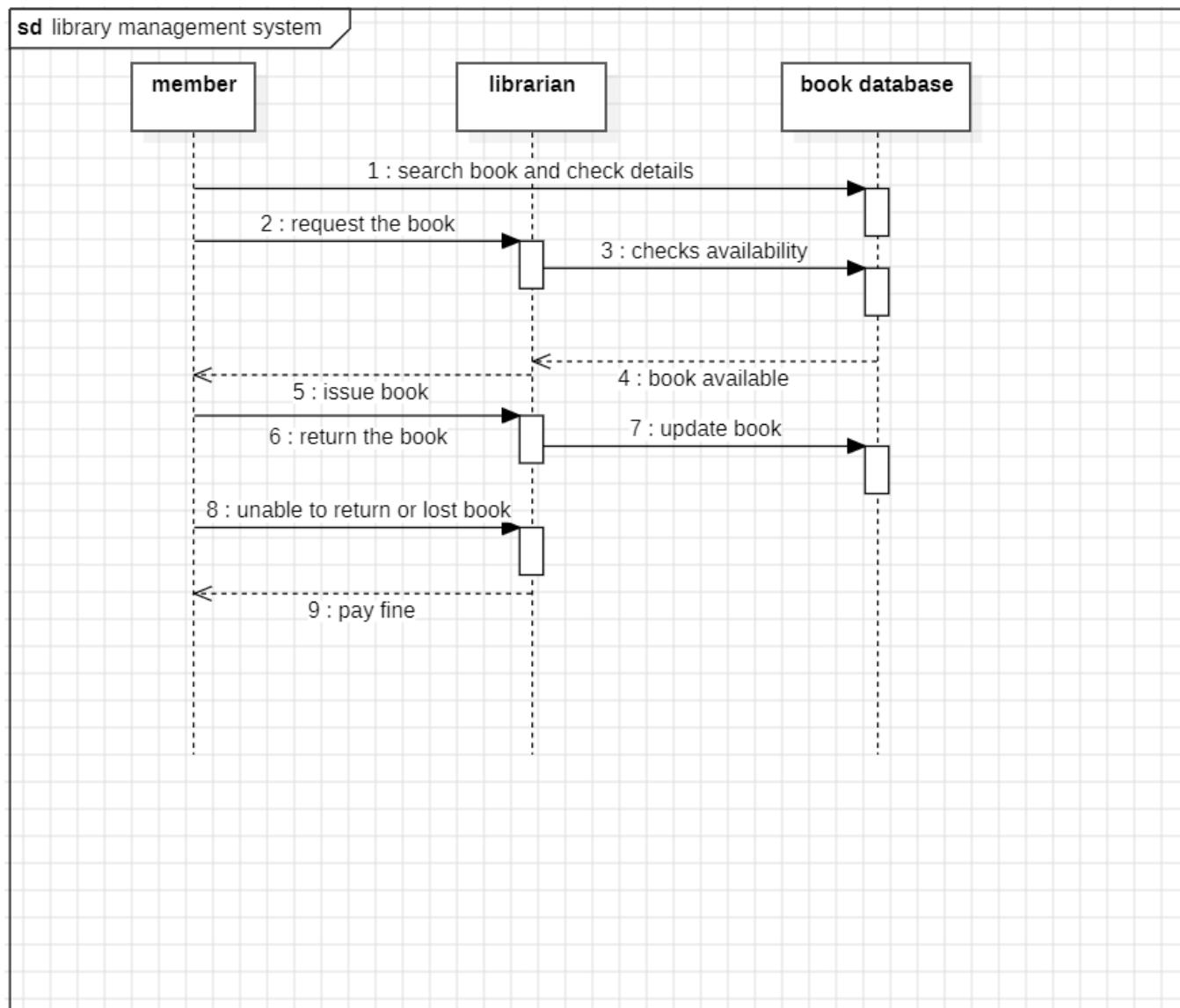
1.a) Use Case Diagram



1.b) Class Diagram

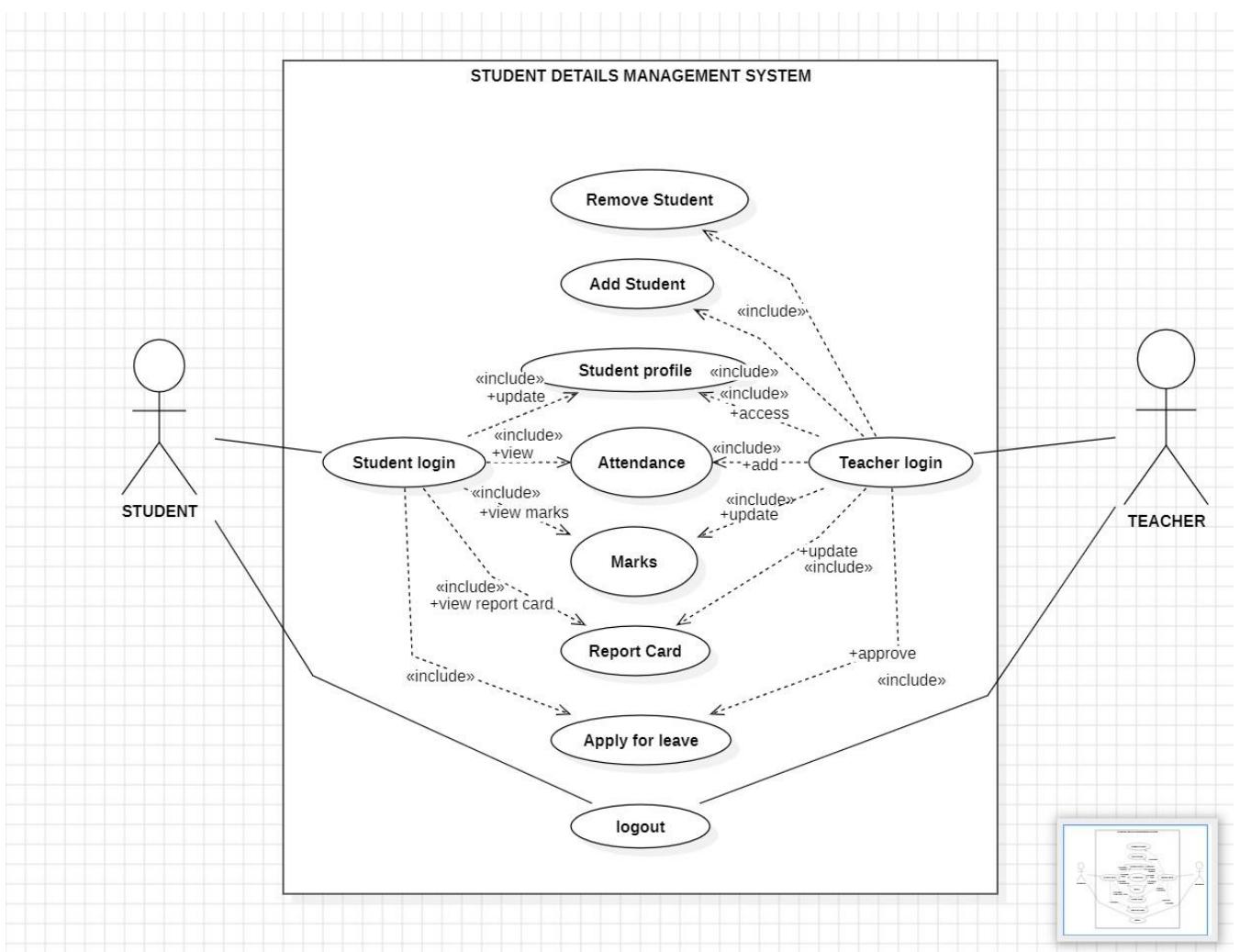


1.c) Sequence Diagram

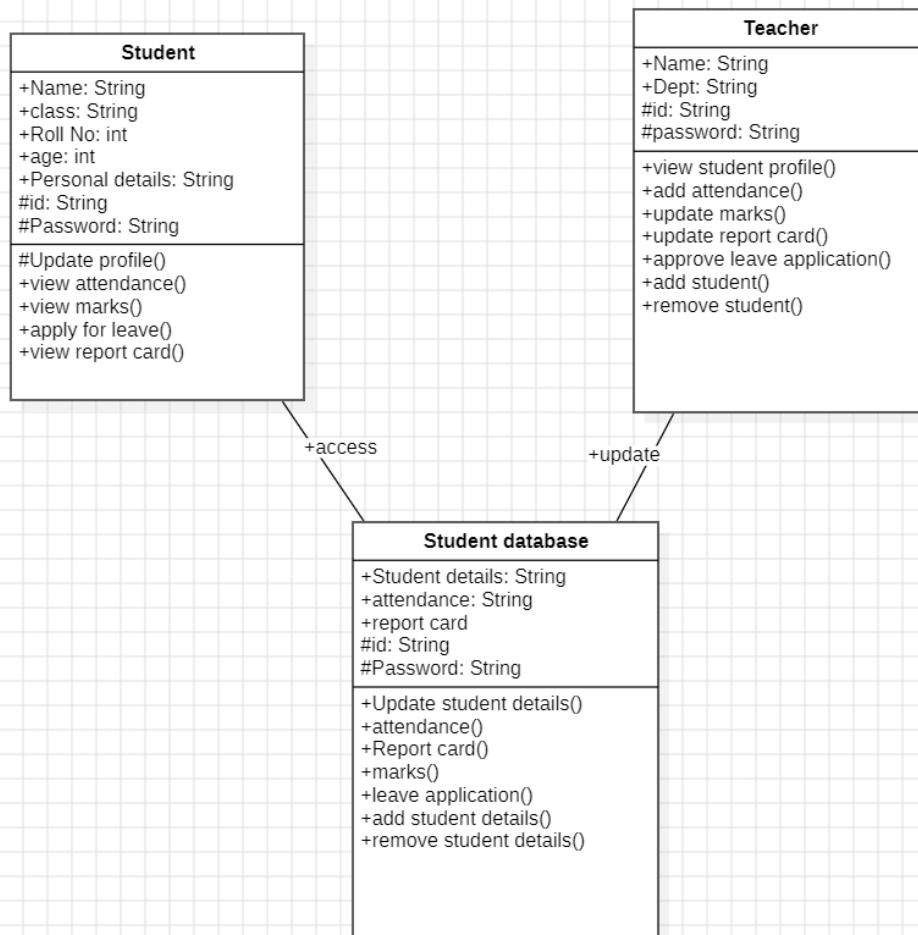


2. Student Management system

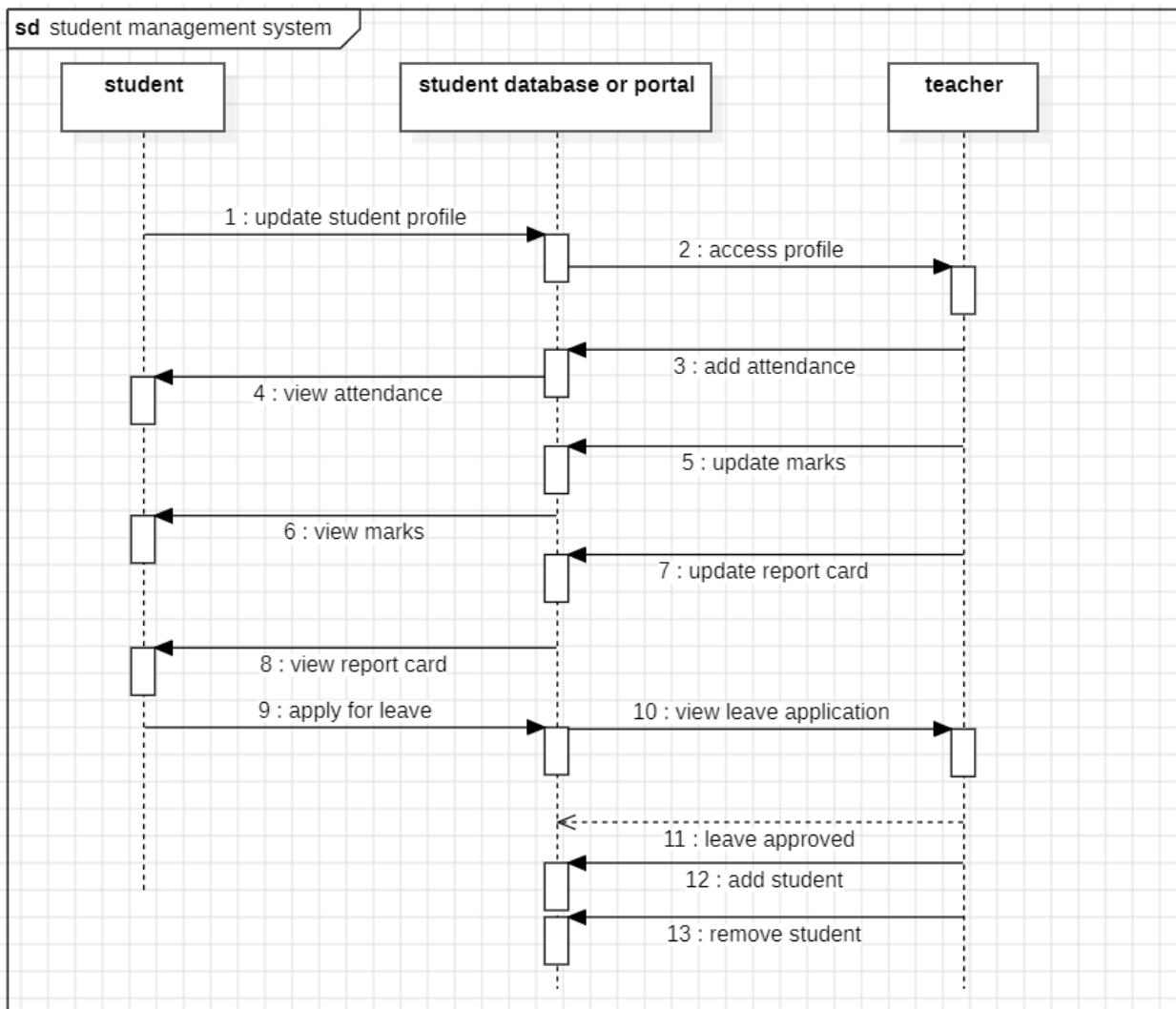
2.a) Use Case Diagram



2.b) Class Diagram

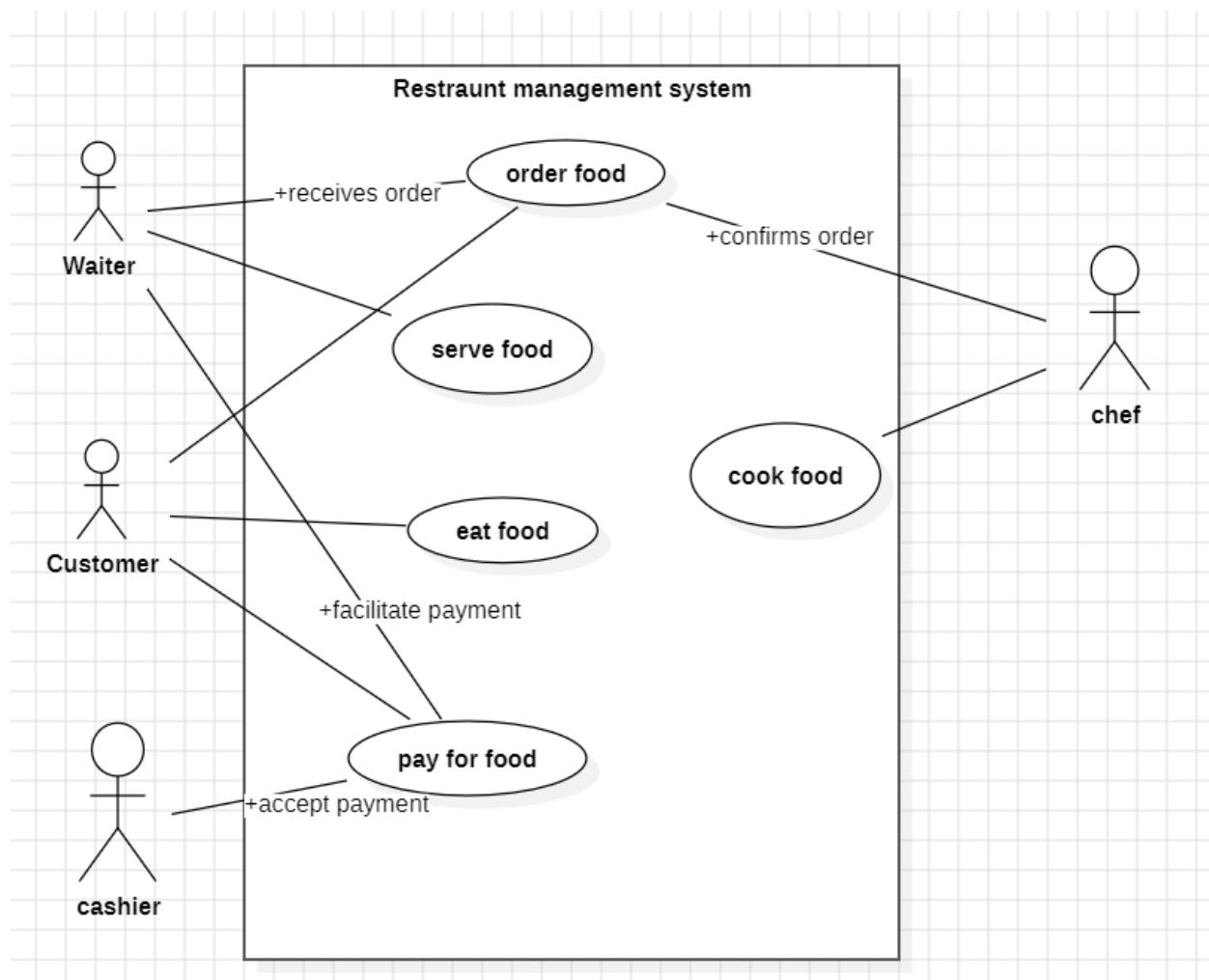


2.c) Sequence Diagram

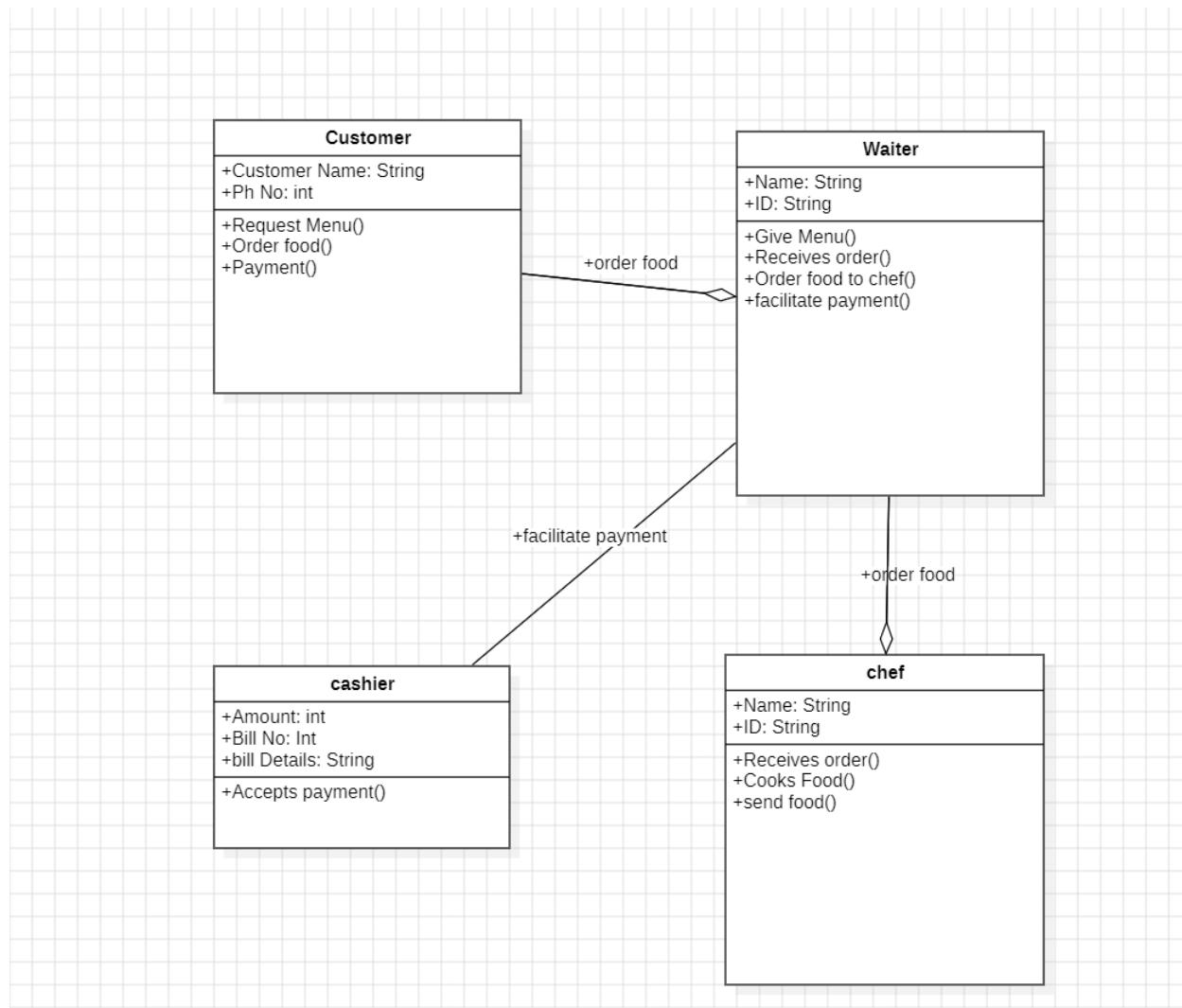


3. Restaurent Management system

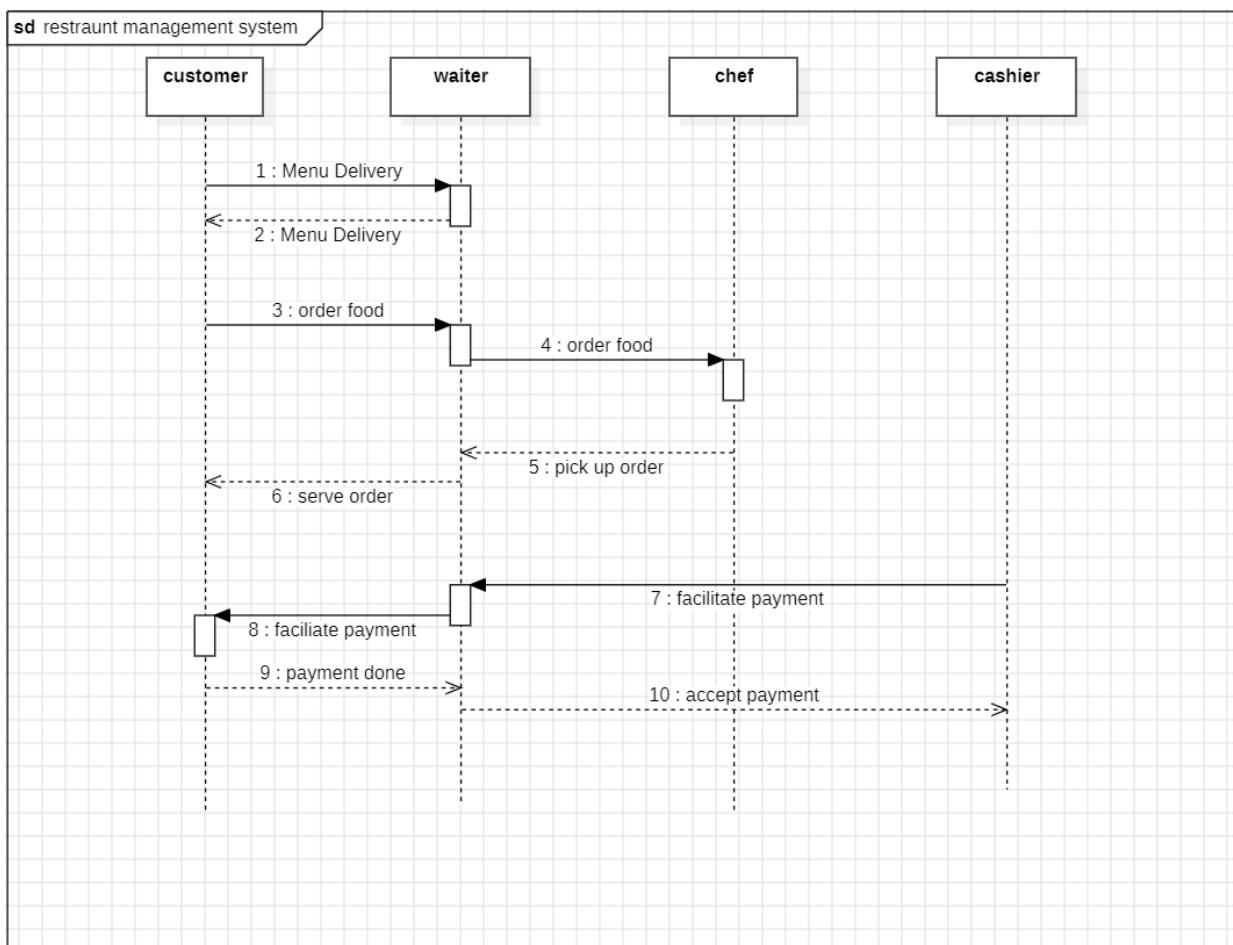
3.a) Use Case Diagram



3.b) Class Diagram



3.c) Sequence Diagram



BASIC JAVA PROGRAMS

4.a. JAVA PROGRAMME TO FIND HYPOTENUSE OF A TRIANGLE:-

AIM:- To Find Hypotenuse of a Triangle

JAVA CODE:-

```
import java.util.Scanner;
```

```
public class Main
```

```
{
```

```
    public static void main(String[] args) {
```

```
        double x;
```

```
        double y;
```

```
        double z;
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println("Enter side x: ");
```

```
        x = scanner.nextDouble();
```

```
        System.out.println("Enter side y: ");
```

```
        y = scanner.nextDouble();
```

```
        z = Math.sqrt((x*x)+(y*y));
```

```
        System.out.println("The hypotenuse is: "+z);
```

```
        scanner.close();
```

```
}
```

```
}
```

SCREENSHOT WITH OUTPUT

The screenshot shows a Java development environment with the following details:

- Code Editor:** The file `Main.java` is open, containing Java code to calculate the hypotenuse of a right-angled triangle using the Pythagorean theorem.
- Output Console:** The console window shows the program's execution. It prompts the user for two sides (x and y), receives input values 4 and 5, and then outputs the calculated hypotenuse value (6.4031242374328485).
- Bottom Status:** The message "...Program finished with exit code 0 Press ENTER to exit console." is displayed.

```
7 ****
8 ****
9 import java.util.Scanner;
10
11 public class Main
12 {
13
14     public static void main(String[] args) {
15         double x;
16         double y;
17         double z;
18
19         Scanner scanner = new Scanner(System.in);
20
21         System.out.println("Enter side x: ");
22         x = scanner.nextDouble();
23         System.out.println("Enter side y: ");
24         y = scanner.nextDouble();
25
26         z = Math.sqrt((x*x)+(y*y));
27
28         System.out.println("The hypotenuse is: "+z);
29
30         scanner.close();
31     }
32 }
33
```

input

```
Enter side x:  
4  
Enter side y:  
5  
The hypotenuse is: 6.4031242374328485  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

4.b. JAVA PROGRAM WITH SWITCH CASE:-

AIM:- To Explore the switch case Element In the java program

JAVA CODE:-

```
public class Main  
{  
    public static void main(String[] args) {  
  
        String day= "Friday";  
        switch(day) {  
  
            case "Sunday": System.out.println("It is Sunday!");  
            break;  
  
            case "Monday": System.out.println("It is Monday!");  
            break;  
  
            case "Tuesday": System.out.println("It is Tuesday!");  
            break;  
  
            case "Wednesday": System.out.println("It is Wednesday!");  
            break;  
  
            case "Thursday": System.out.println("It is Thursday!");  
            break;  
  
            case "Friday": System.out.println("It is Friday!");  
            break;  
  
            case "Saturday": System.out.println("It is Saturday!");  
            break;  
  
        }  
    }  
}
```

SCREENSHOT WITH OUTPUT

The screenshot shows a Java code editor with a dark theme. The file is named Main.java. The code contains a switch statement that prints "It is Friday!" to the console. The output window below the editor shows the printed message and a prompt to exit.

```
6  Code, Compile, Run and Debug online from anywhere in world.
7
8 ****
9 public class Main
10 {
11
12     public static void main(String[] args) {
13         String day= "Friday";
14
15         switch(day) {
16             case "Sunday": System.out.println("It is Sunday!");
17             break;
18             case "Monday": System.out.println("It is Monday!");
19             break;
20             case "Tuesday": System.out.println("It is Tuesday!");
21             break;
22             case "Wednesday": System.out.println("It is Wednesday!");
23             break;
24             case "Thursday": System.out.println("It is Thursday!");
25             break;
26             case "Friday": System.out.println("It is Friday!");
27             break;
28             case "Saturday": System.out.println("It is Saturday!");
29             break;
30
31         }
32     }
33 }
34
```

It is Friday!

...Program finished with exit code 0
Press ENTER to exit console.

PU

4.c. JAVA PROGRAMME USING IF ELSE STATEMENTS:-

AIM:- To Decode the If else statements in java code

JAVA CODE:-

```
import java.util.Scanner;

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

```
System.out.println("you are playing a game! press q or Q to quit");
String response = scanner.next();

if(response.equals("q") || response.equals("Q")){
    System.out.println("you quit the game");
}
else{
    System.out.println("you are still playing the game");
}
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with a dark theme. The top menu bar includes options like Run, Debug, Stop, Share, Save, and Beautify. The code editor window is titled "Main.java" and contains the provided Java code. The output window at the bottom shows the program's execution:

```
you are playing a game! press q or Q to quit
Q
you quit the game

...Program finished with exit code 0
Press ENTER to exit console.[]
```

4.D.JAVA PROGRAMME USING WHILE LOOP:-

AIM:- To Decode the while loops in java code

JAVA CODE:-

```
import java.util.Scanner;

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

        Scanner scanner = new Scanner(System.in);
        String name="";

        while(name.isBlank()){
            System.out.print("Enter your name: ");
            name = scanner.nextLine();
        }
        System.out.println("Hello "+name);

    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with a dark theme. The file is named Main.java. The code is as follows:

```

1 import java.util.Scanner;
2
3 public class Main
4 {
5
6     public static void main(String[] args) {
7
8         Scanner scanner = new Scanner(System.in);
9         String name="";
10
11        while(name.isBlank()){
12            System.out.print("Enter your name: ");
13            name = scanner.nextLine();
14        }
15        System.out.println("Hello "+name);
16    }
17
18 }
19

```

Below the code editor is a terminal window showing the execution of the program. It prompts the user to enter their name three times, and then prints "Hello Peter".

```

Enter your name:
Enter your name:
Enter your name:
Enter your name: Peter
Hello Peter

```

4.e. JAVA PROGRAMME USING FOR LOOP:-

AIM:- To Decode the For loops in java code

JAVA CODE:-

```

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

        for (int i=5; i>=0; i-- ){
            System.out.println(i);

        }
    }
}

```

```
System.out.println("Today we have lab exam");  
}  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface. The top bar includes buttons for Run, Debug, Stop, Share, Save, and Beautify. The left sidebar shows a file named 'Main.java'. The code in the editor is:

```
1 public class Main
2 {
3     public static void main(String[] args) {
4         for (int i=5; i>=0; i--) {
5             System.out.println(i);
6         }
7         System.out.println("Today we have lab exam");
8     }
9 }
10
11 }
```

The bottom window displays the console output:

```
5
4
3
2
1
0
Today we have lab exam

...Program finished with exit code 0
Press ENTER to exit console.
```

4.f.JAVA PROGRAMME FOR SHAPE OF A SYMBOL:-

AIM:- To Find the shape of any symbol using java programme.

JAVA CODE:-

```
import java.util.Scanner;

public class Main
{

    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
int rows;
int columns;
```

```
String symbol = "";  
  
System.out.println("Enter no of rows: ");  
rows = scanner.nextInt();  
  
System.out.println("Enter no of columns: ");  
columns = scanner.nextInt();  
  
System.out.println("Enter symbol to use: ");  
symbol = scanner.next();  
  
  
for(int i=1; i<=rows; i++){  
    System.out.println();  
    for(int j=1; j<=columns; j++){  
        System.out.print(symbol);  
  
    }  
}  
}  
}
```

SCREENSHOT WITH OUTPUT:-

```

Main.java
1 import java.util.Scanner;
2
3 public class Main
4 {
5
6     public static void main(String[] args) {
7
8         Scanner scanner = new Scanner(System.in);
9         int rows;
10        int columns;
11        String symbol = "";
12
13        System.out.println("Enter no of rows: ");
14        rows = scanner.nextInt();
15        System.out.println("Enter no of columns: ");
16        columns = scanner.nextInt();
17        System.out.println("Enter symbol to use: ");
18        symbol = scanner.next();
19
20        for(int i=1; i<=rows; i++){
21            System.out.println();
22            for(int j=1; j<=columns; j++){
23                System.out.print(symbol);
24            }
25        }
}

```

Enter no of rows:
4
Enter no of columns:
5
Enter symbol to use:
\$
\$
\$
\$
\$

4.g. JAVA PROGRAMME FOR FACTORIAL OF A NUMBER:-

AIM:- To Find the Factorial of a Number

JAVA CODE:-

```

import java.util.Scanner;

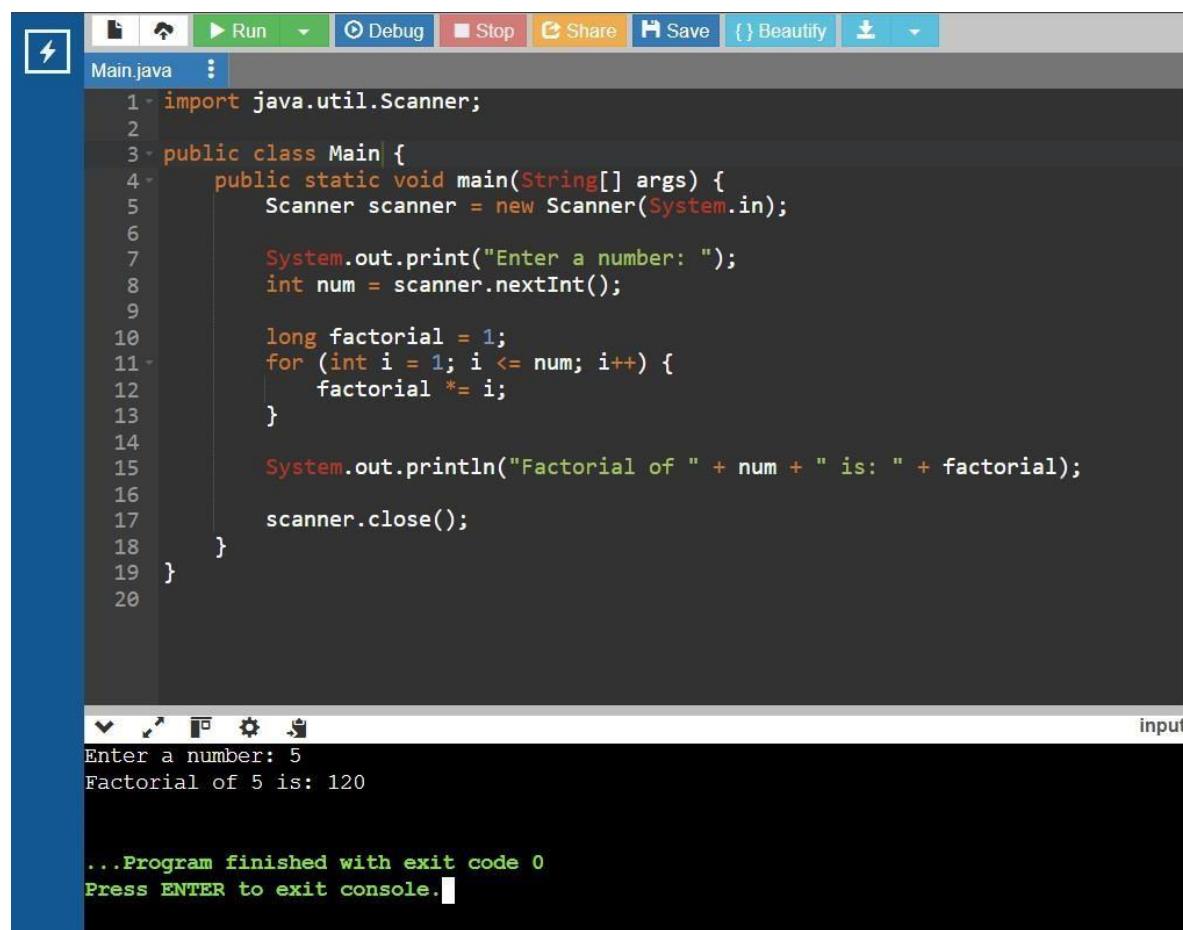
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

```

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

```
long factorial = 1;  
for (int i = 1; i <= num; i++) {  
    factorial *= i;  
}  
  
System.out.println("Factorial of " + num + " is: " + factorial);  
  
scanner.close();  
}  
}
```

SCREENSHOT WITH OUTPUT:-



The screenshot shows a Java development environment with the following details:

- File:** Main.java
- Code:** The code is a Java program that calculates the factorial of a given number. It uses a Scanner to read input from the user and prints the result to the console.
- Output:** The console window shows the following interaction:
 - The user enters "5" as input.
 - The program outputs "Factorial of 5 is: 120".
 - The program concludes with "...Program finished with exit code 0" and "Press ENTER to exit console."

4.h. JAVA PROGRAM FOR REVERSING A STRING:-

AIM:- To Reverse any type of string like names or Anything.

JAVA CODE:-

```
import java.util.Scanner;

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

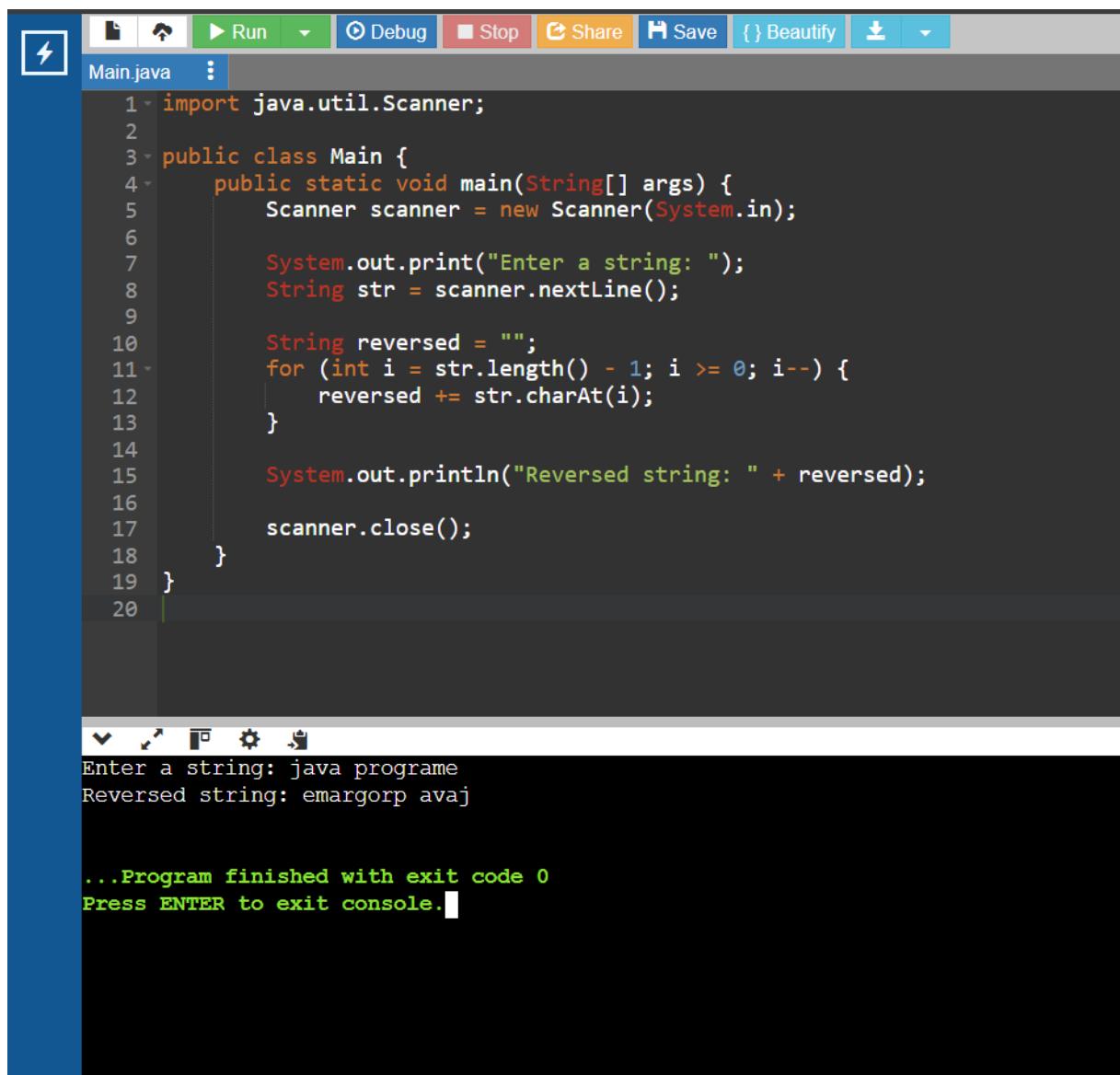
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();

        String reversed = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed += str.charAt(i);
        }

        System.out.println("Reversed string: " + reversed);

        scanner.close();
    }
}
```

SCREENSHOT WITH OUTPUT:-



```

Main.java
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Enter a string: ");
8         String str = scanner.nextLine();
9
10        String reversed = "";
11        for (int i = str.length() - 1; i >= 0; i--) {
12            reversed += str.charAt(i);
13        }
14
15        System.out.println("Reversed string: " + reversed);
16
17        scanner.close();
18    }
19 }
20

```

Enter a string: java programe
 Reversed string: emargorp avaj

...Program finished with exit code 0
 Press ENTER to exit console.

4.i. JAVA PROGRAM TO IDENTIFY PRIME NUMBER:-

AIM:- To Identify weather the given number Is prime number or not.

JAVA CODE:-

```
import java.util.Scanner;
```

```
public class Main{
```

```
    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
```

```
System.out.print("Enter a number: ");
```

```
int num = scanner.nextInt();

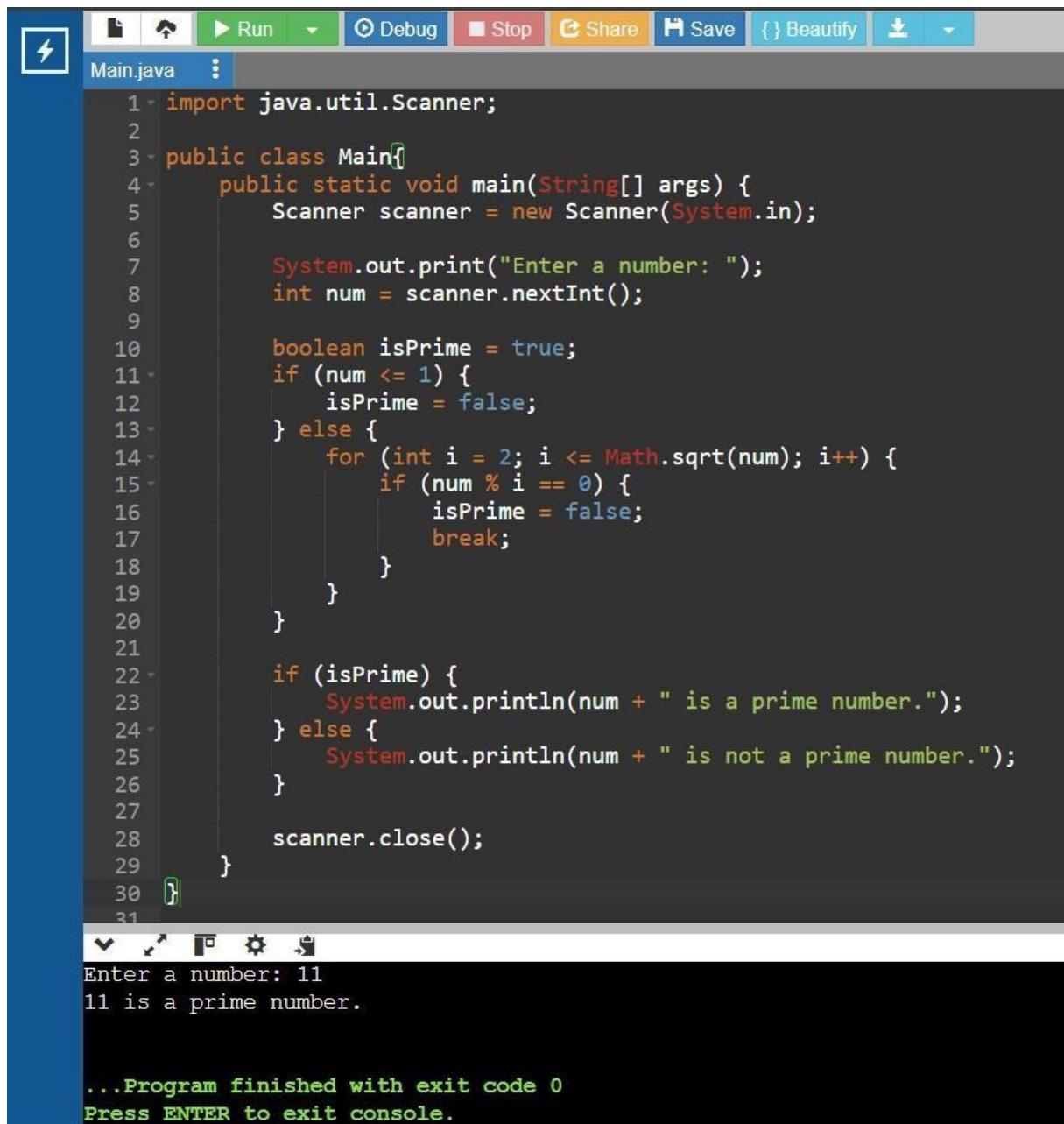
boolean isPrime = true;

if (num <= 1) {
    isPrime = false;
} else {
    for (int i = 2; i <= Math.sqrt(num); i++) {
        if (num % i == 0) {
            isPrime = false;
            break;
        }
    }
}

if (isPrime) {
    System.out.println(num + " is a prime number.");
} else {
    System.out.println(num + " is not a prime number.");
}

scanner.close();
}
```

SCREENSHOT WITH OUTPUT:-



```

1 import java.util.Scanner;
2
3 public class Main{
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Enter a number: ");
8         int num = scanner.nextInt();
9
10        boolean isPrime = true;
11        if (num <= 1) {
12            isPrime = false;
13        } else {
14            for (int i = 2; i <= Math.sqrt(num); i++) {
15                if (num % i == 0) {
16                    isPrime = false;
17                    break;
18                }
19            }
20        }
21
22        if (isPrime) {
23            System.out.println(num + " is a prime number.");
24        } else {
25            System.out.println(num + " is not a prime number.");
26        }
27
28        scanner.close();
29    }
30 }

```

Enter a number: 11
11 is a prime number.

...Program finished with exit code 0
Press ENTER to exit console.

4.j. JAVA PROGRAMME FOR PALINDROME:-

AIM:- To find the palindrome of a given string.

JAVA CODE:-

```
import java.util.Scanner;
```

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

```
System.out.print("Enter a word: ");

String word = scanner.nextLine();

if (isPalindrome(word)) {
    System.out.println(word + " is a palindrome.");
} else {
    System.out.println(word + " is not a palindrome.");
}

scanner.close();
}

public static boolean isPalindrome(String str) {
    int left = 0;
    int right = str.length() - 1;

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

SCREENSHOT OF CODE:-

The screenshot shows a Java code editor with the file 'Main.java' open. The code defines a class 'Main' with a static method 'main'. It uses a Scanner to read a word from standard input, then checks if it's a palindrome by comparing characters from both ends moving towards the center. If all characters match, it prints that the word is a palindrome; otherwise, it prints that it is not.

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Enter a word: ");
8         String word = scanner.nextLine();
9
10        if (isPalindrome(word)) {
11            System.out.println(word + " is a palindrome.");
12        } else {
13            System.out.println(word + " is not a palindrome.");
14        }
15
16        scanner.close();
17    }
18
19    public static boolean isPalindrome(String str) {
20        int left = 0;
21        int right = str.length() - 1;
22
23        while (left < right) {
24            if (str.charAt(left) != str.charAt(right)) {
25                return false;
26            }
27            left++;
28            right--;
29        }
30        return true;
31    }
32 }
```

OUTPUT:-

```
Enter a word: ABCDCBA
ABCDCBA is a palindrome.

...
...Program finished with exit code 0
Press ENTER to exit console.
```

Inheritance:-

5.Multilevel

a)

```
import java.util.Scanner;

class Appliance {
    String brand;

    void getBrand(Scanner sc) {
        System.out.print("Enter appliance brand: ");
        brand = sc.nextLine();
    }

    void displayBrand() {
        System.out.println("Brand: " + brand);
    }
}

class WashingMachine extends Appliance {
    void washClothes() {
        System.out.println(brand + " washing machine is washing clothes.");
    }
}

class SmartWashingMachine extends WashingMachine {
    void smartWash() {
        System.out.println(brand + " smart washing machine optimizes water usage.");
    }
}

public class Multilevel {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        SmartWashingMachine machine = new SmartWashingMachine();
        machine.getBrand(sc);
        machine.displayBrand();
        machine.washClothes();
        machine.smartWash();
    }
}
```

```

        sc.close();
    }
}

```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows the file name "Multilevel.java".
- Toolbar:** Includes icons for Run, Debug, Stop, Share, Save, and Beautify.
- Code Editor:** Displays the Java code for Multilevel.java. The code defines an Appliance class with getBrand() and displayBrand() methods, and two subclasses: WashingMachine and SmartWashingMachine, both extending Appliance. The SmartWashingMachine has a smartWash() method.
- Output Console:** Labeled "input" at the top right. It shows the program's execution:


```

Enter appliance brand: samsung
Brand: samsung
samsung washing machine is washing clothes.
samsung smart washing machine optimizes water usage.

...Program finished with exit code 0
Press ENTER to exit console.

```

b)

```
import java.util.Scanner;
```

```
class Vehicle {
```

```
    String brand;
    int wheels;
```

```
    Vehicle(String brand, int wheels) {
```

```
        this.brand = brand;
        this.wheels = wheels;
    }
```

```
void showVehicleDetails() {
    System.out.println("Brand: " + brand);
    System.out.println("Number of Wheels: " + wheels);
}

class Car extends Vehicle {
    String fuelType;

    Car(String brand, int wheels, String fuelType) {
        super(brand, wheels);
        this.fuelType = fuelType;
    }

    void showCarDetails() {
        showVehicleDetails();
        System.out.println("Fuel Type: " + fuelType);
    }
}

class SportsCar extends Car {
    int maxSpeed;

    SportsCar(String brand, int wheels, String fuelType, int maxSpeed) {
        super(brand, wheels, fuelType);
        this.maxSpeed = maxSpeed;
    }

    void showSportsCarDetails() {
        showCarDetails();
        System.out.println("Max Speed: " + maxSpeed + " km/h");
    }
}

public class Multilevel1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Car Brand: ");
        String brand = sc.nextLine();
        System.out.print("Enter Number of Wheels: ");
        int wheels = sc.nextInt();
        sc.nextLine();
        System.out.print("Enter Fuel Type: ");
        String fuelType = sc.nextLine();
        System.out.print("Enter Max Speed: ");
        int maxSpeed = sc.nextInt();

        SportsCar sportsCar = new SportsCar(brand, wheels, fuelType, maxSpeed);
    }
}
```

```

    System.out.println("\nSports Car Details:");
    sportsCar.showSportsCarDetails();

    sc.close();
}
}

```

SCREENSHOT WITH OUTPUT:-

The screenshot shows an IDE interface with a Java file named `Multilevel1.java` open. The code defines three classes: `Vehicle`, `Car`, and `SportsCar`. The `Vehicle` class has a brand and wheels. The `Car` class extends `Vehicle` and adds fuel type. The `SportsCar` class extends `Car` and adds max speed. The `showVehicleDetails` method prints brand and wheels. The `showCarDetails` method prints brand, wheels, and fuel type. The `showSportsCarDetails` method prints brand, wheels, fuel type, and max speed. The output window shows the results of running the program with input for car brand, number of wheels, fuel type, and max speed.

```

Multilevel1.java ::

1- import java.util.Scanner;
2-
3- class Vehicle {
4-     String brand;
5-     int wheels;
6-
7-     Vehicle(String brand, int wheels) {
8-         this.brand = brand;
9-         this.wheels = wheels;
10    }
11-
12    void showVehicleDetails() {
13        System.out.println("Brand: " + brand);
14        System.out.println("Number of Wheels: " + wheels);
15    }
16 }
17-
18 class Car extends Vehicle {
19     String fuelType;
20-
21     Car(String brand, int wheels, String fuelType) {
22         super(brand, wheels);
23         this.fuelType = fuelType;
24     }
25-
26     void showCarDetails() {
27         showVehicleDetails();
28         System.out.println("Fuel Type: " + fuelType);
29     }
30 }
31-
32 class SportsCar extends Car {

```

input

```

Enter Car Brand: BMW
Enter Number of Wheels: 4
Enter Fuel Type: indian oil
Enter Max Speed: 200kmph

```

6.Hybrid:-

a)

```

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

```

```
    }  
}
```

```
interface ElectricVehicle {  
    void chargeBattery();  
}
```

```
interface FuelVehicle {  
    void refuel();  
}
```

```
class Tesla extends Vehicle implements ElectricVehicle {  
    public void chargeBattery() {  
        System.out.println("Tesla is charging its battery.");  
    }  
  
    void autopilot() {  
        System.out.println("Tesla is in autopilot mode.");  
    }  
}
```

```
class Ford extends Vehicle implements FuelVehicle {  
    public void refuel() {  
        System.out.println("Ford is refueling with gasoline.");  
    }  
  
    void manualDrive() {  
        System.out.println("Ford is being driven manually.");  
    }  
}
```

```
public class Hybrid {  
    public static void main(String[] args) {  
        Tesla myTesla = new Tesla();  
        myTesla.start(); // Inherited from Vehicle  
        myTesla.chargeBattery(); // Implemented from ElectricVehicle  
        myTesla.autopilot(); // Specific to Tesla  
  
        System.out.println("-----");  
  
        Ford myFord = new Ford();
```

```
myFord.start(); // Inherited from Vehicle  
myFord.refuel(); // Implemented from FuelVehicle  
myFord.manualDrive(); // Specific to Ford  
}  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the following code:

```
1 class Vehicle {
2     void start() {
3         System.out.println("Vehicle is starting...");
4     }
5 }
6
7
8 interface ElectricVehicle {
9     void chargeBattery();
10}
11
12
13 interface FuelVehicle {
14     void refuel();
15}
16
17
18 class Tesla extends Vehicle implements ElectricVehicle {
19     public void chargeBattery() {
20         System.out.println("Tesla is charging its battery.");
21     }
22
23     void autopilot() {
24         System.out.println("Tesla is in autopilot mode.");
25     }
26 }
27
28
29 class Ford extends Vehicle implements FuelVehicle {
30     public void refuel() {
```

Below the code editor is a terminal window displaying the output of the program:

```
Vehicle is starting...
Tesla is charging its battery.
Tesla is in autopilot mode.
-----
Vehicle is starting...
Ford is refueling with gasoline.
```

b]

```
class animal {  
void eat(){  
System.out.println("animal eats");  
}  
}  
  
interface carnivore {  
void eatsmeat();  
}  
  
interface herbivore{  
void eatsgrass();  
}  
  
class omnivore extends animal implements carnivore, herbivore {  
public void eatsmeat() {  
  
System.out.println("omnivore eats meat");  
}  
  
public void eatsgrass() {  
System.out.println("omnivore eats grass");  
}  
}  
  
class Hybrid1 {  
public static void main (String[] args) {  
omnivore obj1=new omnivore();  
obj1.eat();  
obj1.eatsmeat();  
obj1.eatsgrass();  
}  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run/debug/stop, share, save, beautify, and download.
- Code Editor:** Displays the content of `Hybrid1.java`. The code defines an `animal` class, a `carnivore` interface, a `herbivore` interface, an `omnivore` class that implements both interfaces, and a `Hybrid1` class that contains a `main` method. The `main` method creates an `omnivore` object and calls its `eat()` method.
- Output Window:** Shows the console output with three lines of text: "animal eats", "omnivore eats meat", and "omnivore eats grass".

```

1- class animal {
2- void eat(){
3 System.out.println("animal eats");
4 }
5 }
6
7- interface carnivore {
8 void eatsmeat();
9 }
10
11- interface herbivore{
12 void eatsgrass();
13 }
14
15- class omnivore extends animal implements carnivore, herbivore {
16 public void eatsmeat() {
17
18 System.out.println("omnivore eats meat");
19
20 }
21
22 public void eatsgrass() {
23 System.out.println("omnivore eats grass");
24
25 }
26 }
27
28- class Hybrid1 {
29 public static void main (String[] args) {
30 omnivore obj1=new omnivore();
31 obj1.eat();

```

7.single Inheritance:-

a)

```

class Employee {
String name;
double salary;

void setDetails(String name, double salary) {
    this.name = name;
    this.salary = salary;
}

```

```
}

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

class Manager extends Employee {
    String department;

    void setDepartment(String department) {
        this.department = department;
    }

    void showManagerDetails() {
        showDetails(); // Call parent class method
        System.out.println("Department: " + department);
    }
}

public class Single {
    public static void main(String[] args) {
        Manager m = new Manager();
        m.setDetails("Alice Johnson", 75000);
        m.setDepartment("IT");

        System.out.println("Manager Details:");
        m.showManagerDetails();
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor and a terminal window. The code editor displays 'Single.java' with the following content:

```
1 class Employee {
2     String name;
3     double salary;
4
5     void setDetails(String name, double salary) {
6         this.name = name;
7         this.salary = salary;
8     }
9
10    void showDetails() {
11        System.out.println("Employee Name: " + name);
12        System.out.println("Salary: $" + salary);
13    }
14 }
15
16
17 class Manager extends Employee {
18     String department;
19
20     void setDepartment(String department) {
21         this.department = department;
22     }
23
24     void showManagerDetails() {
25         showDetails(); // Call parent class method
26         System.out.println("Department: " + department);
27     }
28 }
29
```

The terminal window below shows the output of running the program:

```
Manager Details:
Employee Name: Alice Johnson
Salary: $75000.0
Department: IT

...Program finished with exit code 0
Press ENTER to exit console.
```

b)

```
class Person {
```

```
String name;
int age;

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

class Student extends Person {
    int studentId;

    void showStudentInfo() {
        System.out.println("Student ID: " + studentId);
    }
}

public class Single1 {
    public static void main(String[] args) {
        Student s = new Student();
        s.name = "Alice";
        s.age = 20;
        s.studentId = 101;
        s.display();
        s.showStudentInfo();
    }
}
```

SCREENSHOT WITH OUTPUT:-

```
Single1.java : 1 class Person { 2     String name; 3     int age; 4     void display() { 5         System.out.println("Name: " + name + ", Age: " + age); 6     } 7 } 8 9 10 class Student extends Person { 11     int studentId; 12     void showStudentInfo() { 13         System.out.println("Student ID: " + studentId); 14     } 15 } 16 } 17 18 public class Single1 { 19     public static void main(String[] args) { 20         Student s = new Student(); 21         s.name = "Alice"; 22         s.age = 20; 23         s.studentId = 101; 24         s.display(); 25         s.showStudentInfo(); 26     } 27 } 28
```

Name: Alice, Age: 20
Student ID: 101

...Program finished with exit code 0
Press ENTER to exit console.

8.Hierarchical Inheritance:-

a)

```
class shape {  
  
void display() {  
System.out.println("this is a shape");  
}  
}  
  
class circle extends shape{  
  
int r;  
void areac(int r) {  
System.out.println("area of circle:"+r*3.14*r);  
}  
}  
  
class rect extends shape{  
  
int l;  
int b;  
void arear(int l, int b) {  
System.out.println("area of rect:"+l*b);  
}  
}  
  
class Hierarchical {  
public static void main(String args[])  
{  
circle c1=new circle();  
c1.display();  
c1.areac(2);  
rect r1=new rect();  
r1.display();  
r1.arear(4,2);  
}}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run/debug/stop, share, save, and beautify.
- Code Editor:** The file `Hierarchical.java` is open. The code defines three classes: `shape`, `circle`, and `rect`, which inherit from `shape`. The `circle` class has a private attribute `r` and a public method `areac` that calculates the area of a circle. The `rect` class has private attributes `l` and `b`, and a public method `arear` that calculates the area of a rectangle. The `Hierarchical` class contains a static main method that creates an instance of `circle`, calls its `display` and `areac` methods, and then creates an instance of `rect`, calling its `display` and `arear` methods.
- Output Window:** Displays the console output:

```
this is a shape
area of circle:12.56
this is a shape
area of rect:8
```

b)

```
import java.util.*;
class person{
    String name;
    int age;
```

```
String address;

person(String name,int age, String address){
    this.name=name;
    this.age=age;
    this.address=address;
}

void displayInfo(){
    System.out.println("Name - "+name);
    System.out.println("age - "+age);
    System.out.println("address - "+address);
}
}

class student extends person{
    String[] course = new String[5];
    int id;

    student(String name,int age, String address,String[] course,int id){
        super(name,age,address);
        this.course=course;
        this.id=id;
    }
    void displayStudentInfo(){
        for(String name:course){
            System.out.print(name+", ");
        }
        System.out.println();
        System.out.println("Id - "+id);
        displayInfo();
    }
}

class teacher extends person{
    String subject;

    teacher(String name,int age, String address,String subject){
        super(name,age,address);
        this.subject=subject;
    }
}
```

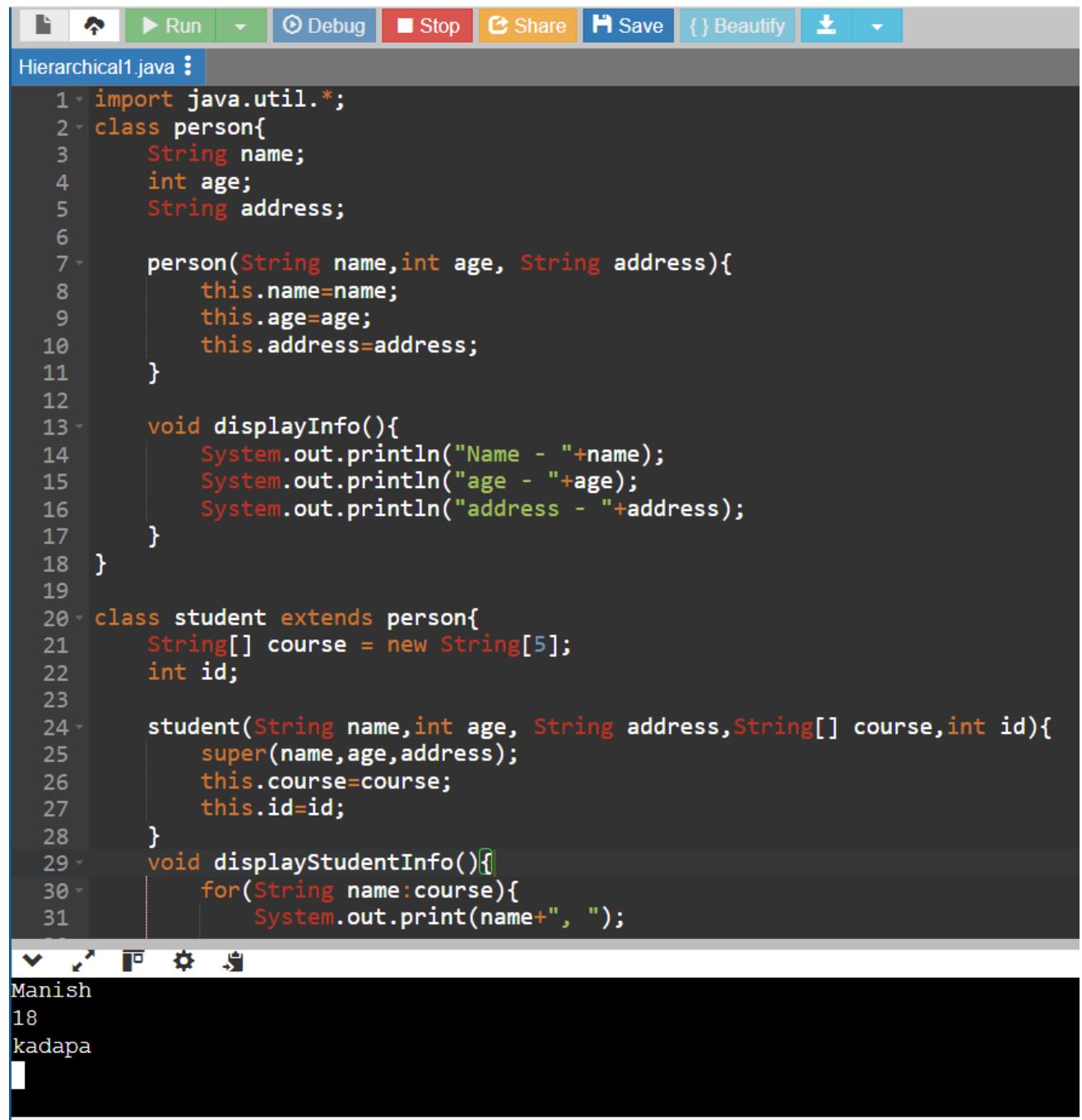
```
void displayTeacherInfo(){
    System.out.println("subject - "+subject);
    displayInfo();
}

}

public class Hierarchical1 {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String[] arr= new String[4];
        for (int i=0 ; i<=3 ; i++){
            String arrele=sc.nextLine();
            arr[i]=arrele;
        }
        String name = sc.nextLine();
        String address = sc.nextLine();
        int age = sc.nextInt();
        int id = sc.nextInt();
        student obj1 = new student(name,age,address,arr,id);
        System.out.println("-----Students details-----");
        obj1.displayStudentInfo();

        sc.nextLine();
        String tname = sc.nextLine();
        String taddress = sc.nextLine();
        String tsubject = sc.nextLine();
        int tage = sc.nextInt();
        teacher obj2 = new teacher(tname,tage,taddress,tsubject);
        System.out.println("-----Teacher details-----");
        obj2.displayTeacherInfo();
    }
}
```

SCREENSHOT WITH OUTPUT:-



```
Hierarchical1.java :  
1 import java.util.*;  
2 class person{  
3     String name;  
4     int age;  
5     String address;  
6  
7     person(String name,int age, String address){  
8         this.name=name;  
9         this.age=age;  
10        this.address=address;  
11    }  
12  
13    void displayInfo(){  
14        System.out.println("Name - "+name);  
15        System.out.println("age - "+age);  
16        System.out.println("address - "+address);  
17    }  
18}  
19  
20 class student extends person{  
21     String[] course = new String[5];  
22     int id;  
23  
24     student(String name,int age, String address,String[] course,int id){  
25         super(name,age,address);  
26         this.course=course;  
27         this.id=id;  
28     }  
29     void displayStudentInfo(){  
30         for(String name:course){  
31             System.out.print(name+", ");  
32         }  
33     }  
34 }
```

9.Constructor:-

```
import java.util.Scanner;

class user{
    String username;
    String email;

    user(String username, String email){
        this.username=username;
        this.email=email;
        System.out.println("name - "+username);
        System.out.println("Email - "+email);
    }

}

public class Constructor {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String username=sc.nextLine();
        String email = sc.nextLine();
        user obj = new user(username,email);
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run/debug/stop, share, and beautify.
- Code Editor:** Displays the `Constructor.java` file. The code defines a `user` class with a constructor that prints the username and email. It also contains a `main` method that creates a `user` object and prints its details.
- Output Console:** Shows the command-line interface output. It starts with the user's email address, followed by the printed details from the program, and ends with a standard Java exit message.

```
Constructor.java :  
1 import java.util.Scanner;  
2  
3 class user{  
4     String username;  
5     String email;  
6  
7     user(String username, String email){  
8         this.username=username;  
9         this.email=email;  
10        System.out.println("name - "+username);  
11        System.out.println("Email - "+email);  
12    }  
13}  
14}  
15  
16 public class Constructor {  
17  
18     public static void main(String[] args) {  
19         Scanner sc = new Scanner(System.in);  
20         String username=sc.nextLine();  
21         String email = sc.nextLine();  
22         user obj = new user(username,email);  
23     }  
24 }  
25  
26  
manish09854@gmail.com  
name - Manish  
Email - manish09854@gmail.com  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

10. Constructor overloading:-

```
class Employee {  
    String name;  
    int id;  
    double salary;  
  
    Employee() {  
        name = "Unknown";  
        id = 0;  
        salary = 0.0;  
    }  
  
    Employee(String empName, int empld) {  
        name = empName;  
        id = empld;  
        salary = 30000.0;  
    }  
  
    Employee(String empName, int empld, double empSalary) {  
        name = empName;  
        id = empld;  
        salary = empSalary;  
    }  
  
    void display() {  
        System.out.println("Employee Name: " + name);  
        System.out.println("ID: " + id);  
        System.out.println("Salary: " + salary);  
    }  
  
    public static void main(String[] args) {  
        Employee e1 = new Employee();  
        Employee e2 = new Employee("John", 102);  
        Employee e3 = new Employee("Emma", 103, 45000.0);  
  
        e1.display();  
        e2.display();  
        e3.display();  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the file `Employee.java` open. The code defines a class `Employee` with three fields: `name`, `id`, and `salary`. It has three constructors: a default constructor setting all values to unknown, a constructor taking `String empName` and `int empId` with `empSalary` set to 30000.0, and another constructor taking all three parameters. A `display()` method prints the employee details. The code ends with a `main` method calling `Employee.display()`.

```
Employee.java : 1 class Employee { 2     String name; 3     int id; 4     double salary; 5 6     Employee() { 7         name = "Unknown"; 8         id = 0; 9         salary = 0.0; 10    } 11 12 13    Employee(String empName, int empId) { 14        name = empName; 15        id = empId; 16        salary = 30000.0; 17    } 18 19 20    Employee(String empName, int empId, double empSalary) { 21        name = empName; 22        id = empId; 23        salary = empSalary; 24    } 25 26    void display() { 27        System.out.println("Employee Name: " + name); 28        System.out.println("ID: " + id); 29        System.out.println("Salary: " + salary); 30    } 31 }
```

The output window below shows the program's execution:

```
Employee Name: Emma
ID: 103
Salary: 45000.0

...Program finished with exit code 0
Press F5/F11 to exit console.
```

11.Method overloading:-

a)

```
import java.util.Scanner;

public class MethodOverloading {

    static int multiply(int a, int b) {
        return a * b;
    }

    static double multiply(double a, double b) {
        return a * b;
    }

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

        System.out.print("Enter first number (integer or decimal): ");
        if (scanner.hasNextInt()) {
            int num1 = scanner.nextInt();
            System.out.print("Enter second number (integer): ");
            int num2 = scanner.nextInt();
            System.out.println("Product of two integers: " + multiply(num1, num2));
        } else {
            double num1 = scanner.nextDouble();
            System.out.print("Enter second number (decimal): ");
            double num2 = scanner.nextDouble();
            System.out.println("Product of two doubles: " + multiply(num1, num2));
        }
        scanner.close();
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the following code:

```
MethodOverloadin... :: 1 import java.util.Scanner; 2 3 public class MethodOverloading { 4 5     static int multiply(int a, int b) { 6         return a * b; 7     } 8 9     static double multiply(double a, double b) {10        return a * b;11    }12 13    public static void main(String[] args) {14        Scanner scanner = new Scanner(System.in);15 16        System.out.print("Enter first number (integer or decimal): ");17        if (scanner.hasNextInt()) {18            int num1 = scanner.nextInt();19            System.out.print("Enter second number (integer): ");20            int num2 = scanner.nextInt();21            System.out.println("Product of two integers: " + multiply(num1, num2));22        } else {23            double num1 = scanner.nextDouble();24            System.out.print("Enter second number (decimal): ");25            double num2 = scanner.nextDouble();26            System.out.println("Product of two doubles: " + multiply(num1, num2));27        }28        scanner.close();29    }30 }31 }
```

The output window shows the following interaction:

```
input
Enter first number (integer or decimal): 10
Enter second number (integer): 10
Product of two integers: 100
```

b)

```
import java.util.*;  
  
class order{  
    int orderid;  
    String discountCode;  
    int price = 1000;  
  
    void displayPrice(int orderid){  
        System.out.println("you order id is "+orderid);  
        System.out.println("your price is - "+price);  
    }  
  
    void displayPrice(int orderid,String discountCode){  
        System.out.println("you order id is "+orderid);  
        price = price-(price/2);  
        System.out.println("your discount price is - "+price);  
    }  
}  
public class MethodOverloading1 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        String discountCode = sc.nextLine();  
        int orderid = sc.nextInt();  
        order o1 = new order();  
        o1.displayPrice(orderid);  
  
        System.out.println("second object");  
        order o2 = new order();  
        o2.displayPrice(orderid,discountCode);  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the following code:

```
1 import java.util.*;
2
3 class order{
4     int orderid;
5     String discountCode;
6     int price = 1000;
7
8     void displayPrice(int orderid){
9         System.out.println("you order id is "+orderid);
10        System.out.println("your price is - "+price);
11    }
12
13    void displayPrice(int orderid, String discountCode){
14        System.out.println("you order id is "+orderid);
15        price = price-(price/2);
16        System.out.println("your discount price is - "+price);
17    }
18
19 }
20 public class MethodOverloading1 {
21     public static void main(String[] args) {
22         Scanner sc = new Scanner(System.in);
23         String discountCode = sc.nextLine();
24         int orderid = sc.nextInt();
25         order o1 = new order();
26         o1.displayPrice(orderid);
27
28         System.out.println("second object");
29         order o2 = new order();
30         o2.displayPrice(orderid, discountCode);
31     }
32 }
```

The output window below the code editor displays the following results:

```
7896541223
2580
you order id is 2580
your price is - 1000
second object
you order id is 2580
your discount price is - 500
```

12.Method overriding:-

a)

```
import java.util.Scanner;

class Vehicle {
    void calculateFare(int distance) {
        System.out.println("Calculating fare for " + distance + " km.");
    }
}

class Car extends Vehicle {
    @Override
    void calculateFare(int distance) {
        System.out.println("Car Fare: " + (distance * 10));
    }
}

class Bike extends Vehicle {
    @Override
    void calculateFare(int distance) {
        System.out.println("Bike Fare: " + (distance * 5));
    }
}

class Auto extends Vehicle {
    @Override
    void calculateFare(int distance) {
        System.out.println("Auto Fare: " + (distance * 7));
    }
}

public class MethodOverriding{
    public static void main(String[] args) {
        Vehicle car = new Car();
        Vehicle bike = new Bike();
        Vehicle auto = new Auto();
        Scanner sc = new Scanner(System.in);
        int distance = sc.nextInt();
        car.calculateFare(distance);
        bike.calculateFare(distance);
        auto.calculateFare(distance);
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the file `MethodOverriding.java` open. The code defines four classes: `Vehicle`, `Car`, `Bike`, and `Auto`. The `Vehicle` class has a method `calculateFare` that prints a generic fare message. The `Car`, `Bike`, and `Auto` classes extend `Vehicle` and override the `calculateFare` method to print specific fares based on distance. The output window below shows the execution of the program, displaying the calculated fares for distances of 100, 50, and 70 respectively.

```
1 import java.util.Scanner;
2
3 class Vehicle {
4     void calculateFare(int distance) {
5         System.out.println("Calculating fare for " + distance + " km.");
6     }
7 }
8
9 class Car extends Vehicle {
10    @Override
11    void calculateFare(int distance) {
12        System.out.println("Car Fare: " + (distance * 10));
13    }
14 }
15
16 class Bike extends Vehicle {
17    @Override
18    void calculateFare(int distance) {
19        System.out.println("Bike Fare: " + (distance * 5));
20    }
21 }
22
23 class Auto extends Vehicle {
24    @Override
25    void calculateFare(int distance) {
26        System.out.println("Auto Fare: " + (distance * 7));
27    }
}
,
10
Car Fare: 100
Bike Fare: 50
Auto Fare: 70

...Program finished with exit code 0
Press ENTER to exit console.
```

b)

```
class Bank {  
    double getInterestRate() {  
        return 2.0; // Default interest rate  
    }  
}  
  
class SavingsAccount extends Bank {  
    double getInterestRate() {  
        return 4.5; // Savings account rate  
    }  
}  
  
class FixedDeposit extends Bank {  
    double getInterestRate() {  
        return 6.5; // Fixed deposit rate  
    }  
}  
  
public class MethodOverriding1 {  
    public static void main(String[] args) {  
        Bank bank = new Bank();  
        Bank savings = new SavingsAccount();  
        Bank fixed = new FixedDeposit();  
  
        System.out.println("Bank Interest Rate: " + bank.getInterestRate() + "%");  
        System.out.println("Savings Account Interest Rate: " + savings.getInterestRate() + "%");  
        System.out.println("Fixed Deposit Interest Rate: " + fixed.getInterestRate() + "%");  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** MethodOverriding1...
- Toolbar:** Run, Debug, Stop, Share, Save, Beautify.
- Code Area:** Contains Java code demonstrating method overriding. It includes three classes: Bank, SavingsAccount, and FixedDeposit, each overriding the getInterestRate() method. The main() method creates instances of these classes and prints their interest rates.

```
1- class Bank {
2-     double getInterestRate() {
3-         return 2.0; // Default interest rate
4-     }
5- }
6
7- class SavingsAccount extends Bank {
8-     double getInterestRate() {
9-         return 4.5; // Savings account rate
10-    }
11- }
12
13- class FixedDeposit extends Bank {
14-     double getInterestRate() {
15-         return 6.5; // Fixed deposit rate
16-     }
17- }
18
19- public class MethodOverriding1 {
20-     public static void main(String[] args) {
21-         Bank bank = new Bank();
22-         Bank savings = new SavingsAccount();
23-         Bank fixed = new FixedDeposit();
24-
25-         System.out.println("Bank Interest Rate: " + bank.getInterestRate() + "%");
26-         System.out.println("Savings Account Interest Rate: " + savings.getInterestRate() + "%");
27-         System.out.println("Fixed Deposit Interest Rate: " + fixed.getInterestRate() + "%");
28-     }
29- }
```

- Output Area:** Shows the console output with the printed interest rates.

```
Bank Interest Rate: 2.0%
Savings Account Interest Rate: 4.5%
Fixed Deposit Interest Rate: 6.5%
```

ABSTRACTION

13.Interface Programmes:-

a)

```
public class Interface1 {  
    public static void main(String[] args) {  
        Calculator add = (a, b) -> a + b;  
        Calculator subtract = (a, b) -> a - b;  
        Calculator multiply = (a, b) -> a * b;  
        Calculator divide = (a, b) -> a / b;  
  
        System.out.println("Addition: " + add.calculate(10, 5));  
        System.out.println("Subtraction: " + subtract.calculate(10, 5));  
        System.out.println("Multiplication: " + multiply.calculate(10, 5));  
        System.out.println("Division: " + divide.calculate(10, 5));  
    }  
}
```

```
interface Calculator {  
    int calculate(int a, int b);  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor with the file `Interface1.java` open. The code defines a `Calculator` interface and its implementation in the `Interface1` class. The output window below shows the results of running the program.

```
1 public class Interface1 {
2     public static void main(String[] args) {
3         Calculator add = (a, b) -> a + b;
4         Calculator subtract = (a, b) -> a - b;
5         Calculator multiply = (a, b) -> a * b;
6         Calculator divide = (a, b) -> a / b;
7
8         System.out.println("Addition: " + add.calculate(10, 5));
9         System.out.println("Subtraction: " + subtract.calculate(10, 5));
10        System.out.println("Multiplication: " + multiply.calculate(10, 5));
11        System.out.println("Division: " + divide.calculate(10, 5));
12    }
13 }
14
15
16 interface Calculator {
17     int calculate(int a, int b);
18 }
```

Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2

...Program finished with exit code 0
Press ENTER to exit console.
```

b)

```
import java.util.Scanner;

interface Product {
    void display();
}

class Electronics implements Product {
    String name;
    double price;

    Electronics(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public void display() {
        System.out.println("Electronics: " + name + " - $" + price);
    }
}

public class Interface2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter product name: ");
        String name = sc.nextLine();
        System.out.print("Enter product price: ");
        double price = sc.nextDouble();

        Product prod = new Electronics(name, price);
        prod.display();

        sc.close();
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run/debug (Run, Debug, Stop, Share), and beautify.
- Code Editor:** Displays the code for `Interface2.java`. The code defines a class `Interface2` with a main method that creates an instance of `Electronics` and prints its details.
- Output Console:** Shows the program's execution. It prompts for product name and price, creates a `Product` object, and prints the result. The output ends with a message about the program finishing.

```
10
11     Electronics(String name, double price) {
12         this.name = name;
13         this.price = price;
14     }
15
16     public void display() {
17         System.out.println("Electronics: " + name + " - $" + price);
18     }
19 }
20
21 public class Interface2 {
22     public static void main(String[] args) {
23         Scanner sc = new Scanner(System.in);
24         System.out.print("Enter product name: ");
25         String name = sc.nextLine();
26         System.out.print("Enter product price: ");
27         double price = sc.nextDouble();
28
29         Product prod = new Electronics(name, price);
30         prod.display();
31
32         sc.close();
33     }
34 }
35
```

```
Enter product name: lays
Enter product price: 20
Electronics: lays - $20.0

...Program finished with exit code 0
Press ENTER to exit console.
```

c)

```
interface Printer {  
    void printDocument(String document);  
}
```

```
class InkjetPrinter implements Printer {  
    public void printDocument(String document) {  
        System.out.println("Inkjet Printer: Printing -> " + document);  
    }  
}
```

```
class LaserPrinter implements Printer {  
    public void printDocument(String document) {  
        System.out.println("Laser Printer: Printing -> " + document);  
    }  
}
```

```
public class Interface3 {  
    public static void main(String[] args) {  
        Printer inkjet = new InkjetPrinter();  
        Printer laser = new LaserPrinter();  
  
        inkjet.printDocument("Report.pdf");  
        laser.printDocument("Invoice.docx");  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run/stop (Run, Debug, Stop), share, and beautify.
- Code Editor:** The file `Interface3.java` is open. The code defines an interface `Printer` with a method `printDocument`, and two implementations: `InkjetPrinter` and `LaserPrinter`. The `main` method creates instances of both printers and calls their `printDocument` methods with specific file names.
- Terminal Output:** Below the code editor, the terminal window displays the printed output:

```
Inkjet Printer: Printing -> Report.pdf
Laser Printer: Printing -> Invoice.docx

...Program finished with exit code 0
Press ENTER to exit console.
```

d)

```
interface a {  
void methoda();  
}  
  
interface b extends a {  
void methodb();  
}  
  
class c implements b {  
public void methoda() {  
System.out.println("methoda");  
}  
  
public void methodb() {  
System.out.println("methodb");  
}  
  
}  
  
}  
  
class Interface4 {  
public static void main(String[] args) {  
c obj= new c();  
obj.methoda();  
obj.methodb();  
}  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, Save, Beautify, and a download icon.
- Code Editor:** Displays the code for `Interface4.java`. The code defines two interfaces, `a` and `b`, and a class `c` that implements `b`. The `main` method creates an object of `c` and calls its methods.

```
1 interface a {  
2     void methoda();  
3 }  
4  
5 interface b extends a {  
6     void methodb();  
7 }  
8  
9 class c implements b {  
10    public void methoda() {  
11        System.out.println("methoda");  
12    }  
13  
14    public void methodb() {  
15        System.out.println("methodb");  
16    }  
17 }  
18  
19 }  
20  
21 class Interface4 {  
22    public static void main(String[] args) {  
23        c obj= new c();  
24        obj.methoda();  
25        obj.methodb();  
26    }  
27 }  
28 }
```

- Console Output:** Shows the printed output from the `System.out.println` statements.

```
methoda  
methodb
```

...Program finished with exit code 0
Press ENTER to exit console.

14. Abstract class Programes:-

a)

```
abstract class GameCharacter {  
    abstract void attack();  
}  
  
class Warrior extends GameCharacter {  
    void attack() {  
        System.out.println("Warrior attacks with a sword!");  
    }  
}  
  
class Mage extends GameCharacter {  
    void attack() {  
        System.out.println("Mage attacks with a fireball!");  
    }  
}  
  
public class Abstract1 {  
    public static void main(String[] args) {  
        GameCharacter warrior = new Warrior();  
        GameCharacter mage = new Mage();  
  
        warrior.attack();  
        mage.attack();  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Contains icons for file operations (New, Open, Save, etc.), a "Run" button, a "Debug" button, a "Stop" button, a "Share" button, a "Save" button, a "Beautify" button, and a download icon.
- Code Editor:** Displays the `Abstract1.java` file content. The code defines an abstract class `GameCharacter` with an abstract method `attack()`. It then defines two concrete classes, `Warrior` and `Mage`, which both inherit from `GameCharacter` and implement the `attack()` method. Finally, it contains a `main` method that creates instances of `Warrior` and `Mage` and calls their `attack` methods.
- Output Console:** Shows the program's output:

```
Warrior attacks with a sword!
Mage attacks with a fireball!

...Program finished with exit code 0
Press ENTER to exit console.
```

b)

```
abstract class OnlinePayment {  
    abstract void processPayment(double amount);  
}  
  
class CreditCardPayment extends OnlinePayment {  
    void processPayment(double amount) {  
        System.out.println("Credit Card Payment of $" + amount + " processed.");  
    }  
}  
  
class PayPalPayment extends OnlinePayment {  
    void processPayment(double amount) {  
        System.out.println("PayPal Payment of $" + amount + " processed.");  
    }  
}  
  
public class Abstract2 {  
    public static void main(String[] args) {  
        OnlinePayment creditCard = new CreditCardPayment();  
        OnlinePayment paypal = new PayPalPayment();  
  
        creditCard.processPayment(100);  
        paypal.processPayment(200);  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Contains icons for file operations (New, Open, Save), run/debug/stop, share, save, beautify, and download.
- Code Editor:** Displays the `Abstract2.java` file content. The code defines an abstract class `OnlinePayment` with an abstract method `processPayment(double amount)`. It then defines two concrete classes, `CreditCardPayment` and `PayPalPayment`, which both extend `OnlinePayment` and implement the `processPayment` method. Finally, it defines a `public class Abstract2` with a `main` method that creates instances of `CreditCardPayment` and `PayPalPayment` and calls their `processPayment` methods with arguments of 100 and 200 respectively.
- Output Console:** Shows the program's output:

```
Credit Card Payment of $100.0 processed.  
PayPal Payment of $200.0 processed.  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

c)

```
abstract class BankAccount {  
    String accountNumber;  
    double balance;  
  
    public BankAccount(String accountNumber, double balance) {  
        this.accountNumber = accountNumber;  
        this.balance = balance;  
    }  
  
    public void deposit(double amount) {  
        balance += amount;  
        System.out.println("Deposited " + amount + ". New balance: " + balance);  
    }  
  
    abstract void withdraw(double amount);  
}  
  
class SavingsAccount extends BankAccount {  
    public SavingsAccount(String accountNumber, double balance) {  
        super(accountNumber, balance);  
    }  
  
    void withdraw(double amount) {  
        if (balance - amount >= 1000) {  
            balance -= amount;  
            System.out.println("Withdrawn " + amount + ". Remaining balance: " + balance);  
        } else {  
            System.out.println("Withdrawal denied! Minimum balance of 1000 must be maintained.");  
        }  
    }  
}  
  
class CurrentAccount extends BankAccount {  
    public CurrentAccount(String accountNumber, double balance) {  
        super(accountNumber, balance);  
    }  
  
    void withdraw(double amount) {
```

```
if (balance >= amount) {  
    balance -= amount;  
    System.out.println("Withdrawn " + amount + ". Remaining balance: " + balance);  
} else {  
    System.out.println("Insufficient balance!");  
}  
}  
}  
  
public class Abstract3 {  
    public static void main(String[] args) {  
        BankAccount savings = new SavingsAccount("SA12345", 5000);  
        BankAccount current = new CurrentAccount("CA67890", 10000);  
  
        savings.deposit(2000);  
        savings.withdraw(6000);  
        savings.withdraw(2000);  
  
        current.withdraw(5000);  
        current.withdraw(6000);  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Contains icons for file operations (New, Open, Save, etc.), Run, Debug, Stop, Share, Save, and Beautify.
- Code Editor:** Displays the content of `Abstract3.java`. The code defines an abstract class `BankAccount` with a constructor and a `deposit` method. It also defines a concrete class `SavingsAccount` that extends `BankAccount` and overrides the `withdraw` method with a balance check.
- Output Window:** Shows the following console output:

```
Deposited 2000.0. New balance: 7000.0
Withdrawn 6000.0. Remaining balance: 1000.0
Withdrawal denied! Minimum balance of 1000 must be maintained.
Withdrawn 5000.0. Remaining balance: 5000.0
Insufficient balance!
```

d)

```
abstract class Shape {  
    Shape() {  
        System.out.println("Shape constructor called");  
    }  
  
    abstract void draw();  
}  
  
class Circle extends Shape {  
    public void draw() {  
        System.out.println("Drawing a circle");  
    }  
}  
  
class Abstract4 {  
    public static void main(String[] args) {  
        Circle obj = new Circle();  
        obj.draw();  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run (Run, Debug, Stop, Share), and other tools (Beautify, Download).
- Code Editor:** Displays the `Abstract4.java` file content. The code defines an abstract class `Shape` with a constructor and an abstract method `draw`. It also defines a concrete class `Circle` that extends `Shape` and implements the `draw` method. Finally, it contains a `main` method that creates a `Circle` object and calls its `draw` method.
- Output Console:** Shows the program's output:

```
Shape constructor called
Drawing a circle

...Program finished with exit code 0
Press ENTER to exit console.
```

The output consists of two lines of text: "Shape constructor called" followed by "Drawing a circle".

ENCAPSULATION

15.Encapsulation Programes:-

a)

```
class Student {  
    private String name;  
  
    public void setName(String newName) {  
        name = newName;  
    }  
  
    public String getName() {  
        return name;  
    }  
}  
  
class Encap1 {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.setName("Manish");  
        System.out.println("Student Name: " + s.getName());  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, and Beautify.
- Code Editor:** Displays the `Encap1.java` file content. The code defines a `Student` class with a private attribute `name` and two methods: `setName` and `getName`. It also defines a `main` method in the `Encap1` class that creates a `Student` object, sets its name to "Manish", and prints the name.
- Output Console:** Shows the program's output:

```
Student Name: Manish
...
...Program finished with exit code 0
Press ENTER to exit console.
```

b)

```
import java.util.Scanner;

class students{
    private String name;
    private int roll;
    private String grade;

    public void setname(String name1){
        name=name1;
    }

    public void setgrade(String grade1){
        grade=grade1;
    }

    public void setroll(int roll1){
        roll=roll1;
    }

    public String getname(){
        return name;
    }

    public String getgrade(){
        return grade;
    }

    public int getroll(){
        return roll;
    }
}

public class Encap2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("name - ");
        String name1 = sc.nextLine();
        System.out.println("");
        System.out.print("grade - ");
        String grade1 = sc.nextLine();
        System.out.println("");
    }
}
```

```
System.out.print("roll - ");
int roll1 = sc.nextInt();
students obj = new students();
obj.setname(name1);

obj.setgrade(grade1);

obj.setroll(roll1);

System.out.println("name="+ " "+obj.getname());
System.out.println("roll="+" "+obj.getroll());
System.out.println("grade="+" "+obj.getgrade());
}

}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, Save, Beautify, and a download icon.
- Code Editor:** The file `Encap2.java` is open. The code defines a class `students` with private fields `name`, `roll`, and `grade`. It contains four methods: `setname`, `setgrade`, `setroll`, and two `get` methods (`getname` and `getgrade`). The code uses Java 8's `String` type and `int` type.
- Output Window:** Displays the following output:

```
name - Manish
grade - 1st year
roll - 129
name= Manish
roll= 129
grade= 1st year
```

c)

```
class Person {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        if (name != null && !name.isEmpty()) {  
            this.name = name;  
        } else {  
            System.out.println("Invalid name.");  
        }  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        if (age > 0) {  
            this.age = age;  
        } else {  
            System.out.println("Invalid age.");  
        }  
    }  
}  
  
public class Encap3 {  
    public static void main(String[] args) {  
        Person person = new Person("Reddy", 25);  
    }  
}
```

```
    System.out.println(person.getName());
    person.setName("Manish");
    System.out.println(person.getName());
}
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java code editor and a terminal window. The code editor displays the file `Encap3.java` with the following content:

```
1 class Person {  
2     private String name;  
3     private int age;  
4  
5     public Person(String name, int age) {  
6         this.name = name;  
7         this.age = age;  
8     }  
9  
10    public String getName() {  
11        return name;  
12    }  
13  
14    public void setName(String name) {  
15        if (name != null && !name.isEmpty()) {  
16            this.name = name;  
17        } else {  
18            System.out.println("Invalid name.");  
19        }  
20    }  
21  
22    public int getAge() {  
23        return age;  
24    }  
25  
26}
```

The terminal window below shows the output of running the program:

```
Reddy  
Manish  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

d)

```
public class Encap4 {  
    public static void main(String[] args) {  
        Car car = new Car("Toyota", "Camry");  
        car.setSpeed(120);  
        System.out.println("Car: " + car.getBrand() + " " + car.getModel() + ", Speed: " +  
        car.getSpeed());  
    }  
}  
  
class Car {  
    private String brand;  
    private String model;  
    private int speed;  
  
    public Car(String brand, String model) {  
        this.brand = brand;  
        this.model = model;  
        this.speed = 0; // Default speed  
    }  
  
    public String getBrand() {  
        return brand;  
    }  
  
    public String getModel() {  
        return model;  
    }  
  
    public int getSpeed() {  
        return speed;  
    }  
  
    public void setSpeed(int speed) {  
        if (speed >= 0 && speed <= 200) {  
            this.speed = speed;  
        } else {  
            System.out.println("Speed must be between 0 and 200.");  
        }  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- File:** Encap4.java
- Code Content:**

```
1 public class Encap4 {
2     public static void main(String[] args) {
3         Car car = new Car("Toyota", "Camry");
4         car.setSpeed(120);
5         System.out.println("Car: " + car.getBrand() + " " + car.getModel() + ", Speed: " + car.getSpeed());
6     }
7 }
8
9 class Car {
10     private String brand;
11     private String model;
12     private int speed;
13
14     public Car(String brand, String model) {
15         this.brand = brand;
16         this.model = model;
17         this.speed = 0; // Default speed
18     }
19
20     public String getBrand() {
21         return brand;
22     }
23
24     public String getModel() {
25         return model;
26     }
27
28     public int getSpeed() {
29         return speed;
30     }
31 }
```
- Output Console:**

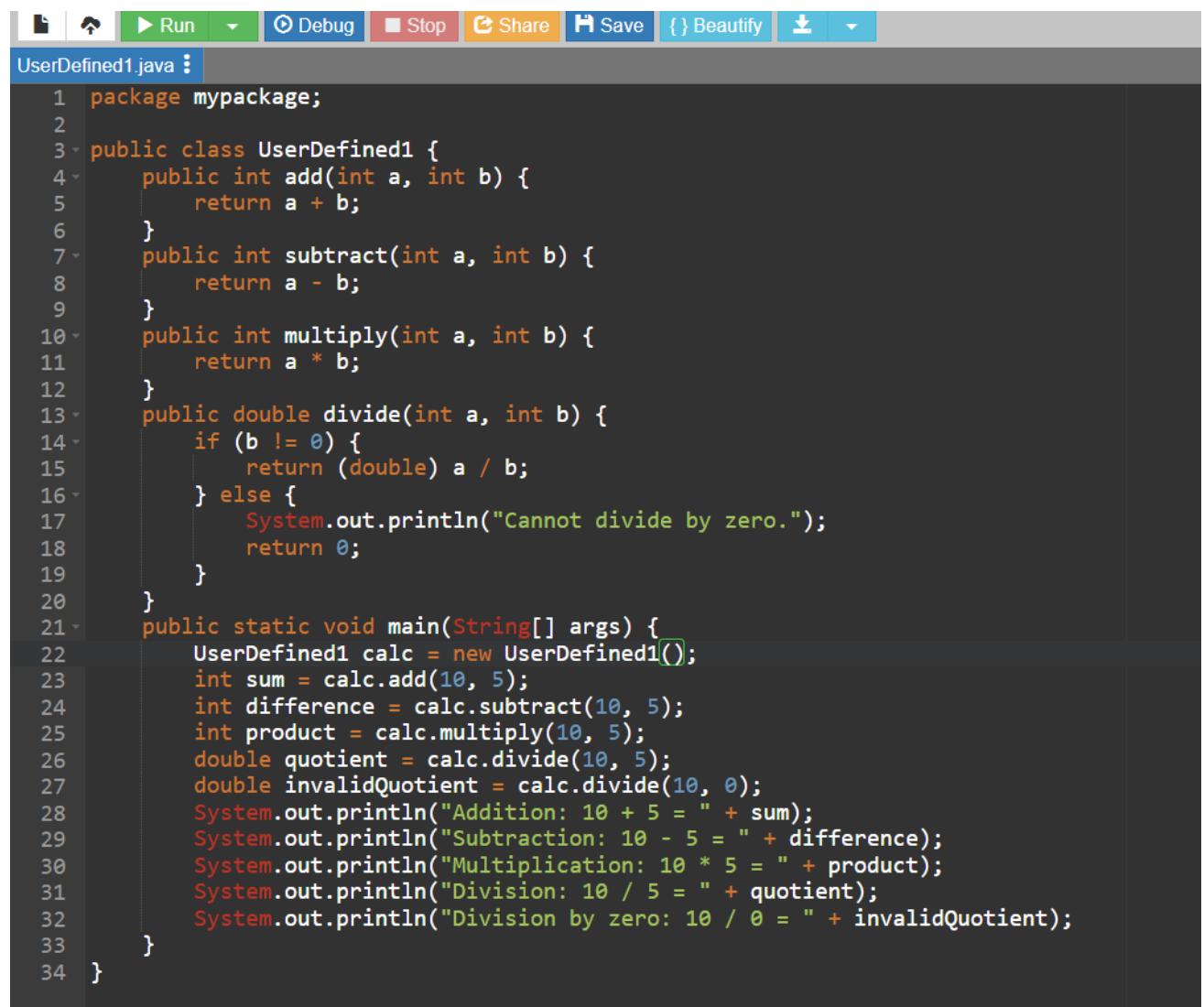
```
Car: Toyota Camry, Speed: 120
...
...Program finished with exit code 0
Press ENTER to exit console.
```

16. Packages Programmes:-

a)

```
package mypackage;

public class UserDefined1 {
    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
    public int multiply(int a, int b) {
        return a * b;
    }
    public double divide(int a, int b) {
        if (b != 0) {
            return (double) a / b;
        } else {
            System.out.println("Cannot divide by zero.");
            return 0;
        }
    }
    public static void main(String[] args) {
        UserDefined1 calc = new UserDefined1();
        int sum = calc.add(10, 5);
        int difference = calc.subtract(10, 5);
        int product = calc.multiply(10, 5);
        double quotient = calc.divide(10, 5);
        double invalidQuotient = calc.divide(10, 0);
        System.out.println("Addition: 10 + 5 = " + sum);
        System.out.println("Subtraction: 10 - 5 = " + difference);
        System.out.println("Multiplication: 10 * 5 = " + product);
        System.out.println("Division: 10 / 5 = " + quotient);
        System.out.println("Division by zero: 10 / 0 = " + invalidQuotient);
    }
}
```

SCREENSHOT WITH OUTPUT:-

```
1 package mypackage;
2
3 public class UserDefined1 {
4     public int add(int a, int b) {
5         return a + b;
6     }
7     public int subtract(int a, int b) {
8         return a - b;
9     }
10    public int multiply(int a, int b) {
11        return a * b;
12    }
13    public double divide(int a, int b) {
14        if (b != 0) {
15            return (double) a / b;
16        } else {
17            System.out.println("Cannot divide by zero.");
18            return 0;
19        }
20    }
21    public static void main(String[] args) {
22        UserDefined1 calc = new UserDefined1();
23        int sum = calc.add(10, 5);
24        int difference = calc.subtract(10, 5);
25        int product = calc.multiply(10, 5);
26        double quotient = calc.divide(10, 5);
27        double invalidQuotient = calc.divide(10, 0);
28        System.out.println("Addition: 10 + 5 = " + sum);
29        System.out.println("Subtraction: 10 - 5 = " + difference);
30        System.out.println("Multiplication: 10 * 5 = " + product);
31        System.out.println("Division: 10 / 5 = " + quotient);
32        System.out.println("Division by zero: 10 / 0 = " + invalidQuotient);
33    }
34 }
```

b)

```
import banking.SavingsAccount;
import banking.CheckingAccount;

public class UserDefined2 {
    public static void main(String[] args) {
        System.out.println("Banking System Example:");

        SavingsAccount savings = new SavingsAccount("SAV001", "John Doe", 1000.00, 2.5);
        CheckingAccount checking = new CheckingAccount("CHK001", "Jane Smith", 500.00, 200.00);

        System.out.println("\n--- Savings Account Details ---");
        System.out.println(savings.getAccountDetails());

        System.out.println("\n--- Checking Account Details ---");
        System.out.println(checking.getAccountDetails());

        System.out.println("\n--- Savings Account Transactions ---");
        savings.deposit(500.00);

        try {
            savings.withdraw(200.00);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

        savings.addInterest();

        System.out.println("\n--- Checking Account Transactions ---");
        checking.deposit(100.00);

        System.out.println("\nAttempt to withdraw within overdraft limit:");
        try {
            checking.withdraw(700.00);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

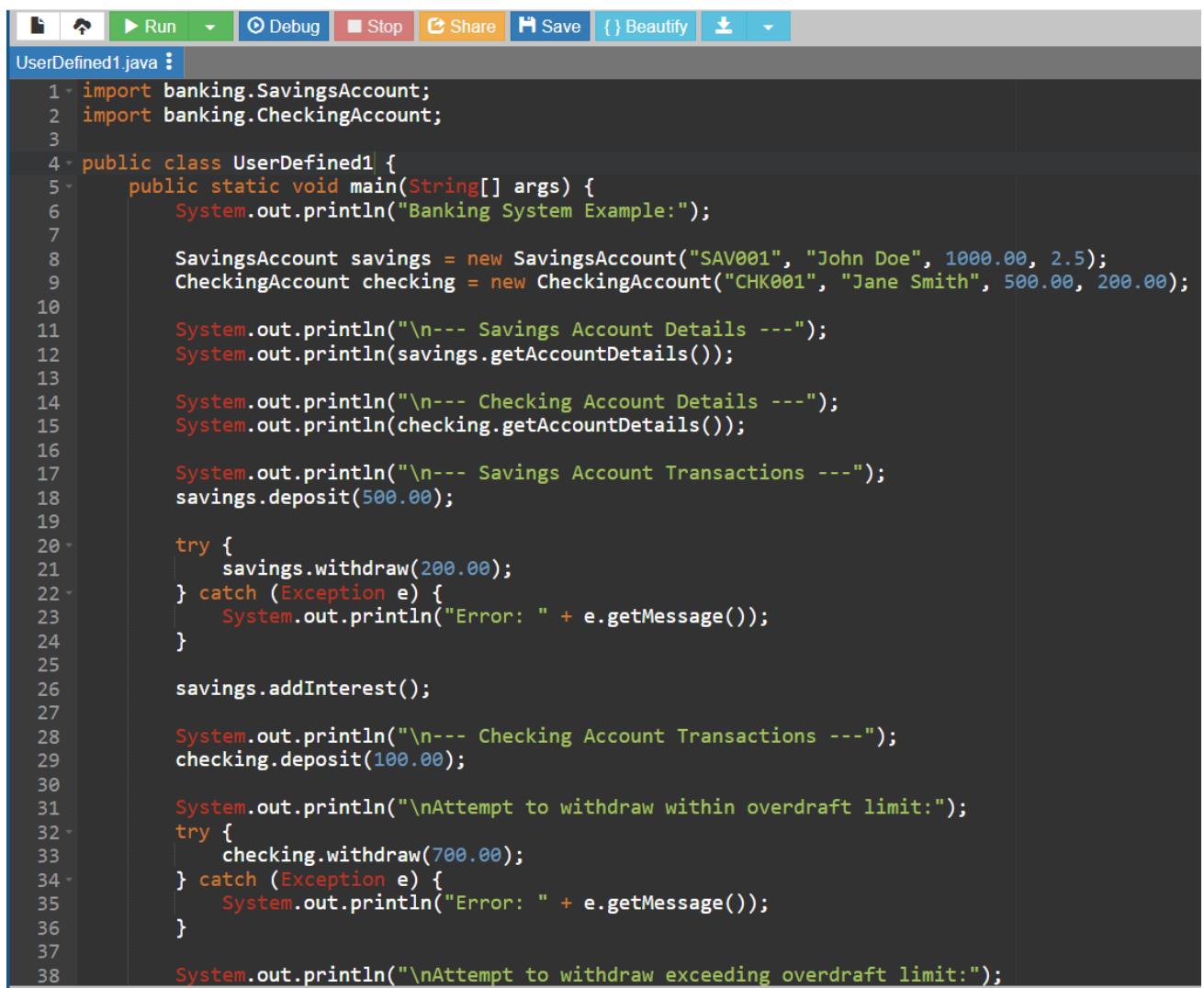
        System.out.println("\nAttempt to withdraw exceeding overdraft limit:");
        try {
            checking.withdraw(1000.00);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

```

System.out.println("\nException Handling Examples:");
try {
    savings.withdraw(-100.00);
} catch (IllegalArgumentException e) {
    System.out.println("Caught exception: " + e.getMessage());
}
}
}
}

```

SCREENSHOT WITH OUTPUT:-



The screenshot shows a Java code editor with the following details:

- Toolbar:** Run, Debug, Stop, Share, Save, Beautify.
- File:** UserDefined1.java
- Code Content:**

```

1 import banking.SavingsAccount;
2 import banking.CheckingAccount;
3
4 public class UserDefined1 {
5     public static void main(String[] args) {
6         System.out.println("Banking System Example:");
7
8         SavingsAccount savings = new SavingsAccount("SAV001", "John Doe", 1000.00, 2.5);
9         CheckingAccount checking = new CheckingAccount("CHK001", "Jane Smith", 500.00, 200.00);
10
11        System.out.println("\n--- Savings Account Details ---");
12        System.out.println(savings.getAccountDetails());
13
14        System.out.println("\n--- Checking Account Details ---");
15        System.out.println(checking.getAccountDetails());
16
17        System.out.println("\n--- Savings Account Transactions ---");
18        savings.deposit(500.00);
19
20        try {
21            savings.withdraw(200.00);
22        } catch (Exception e) {
23            System.out.println("Error: " + e.getMessage());
24        }
25
26        savings.addInterest();
27
28        System.out.println("\n--- Checking Account Transactions ---");
29        checking.deposit(100.00);
30
31        System.out.println("\nAttempt to withdraw within overdraft limit:");
32        try {
33            checking.withdraw(700.00);
34        } catch (Exception e) {
35            System.out.println("Error: " + e.getMessage());
36        }
37
38        System.out.println("\nAttempt to withdraw exceeding overdraft limit:");

```

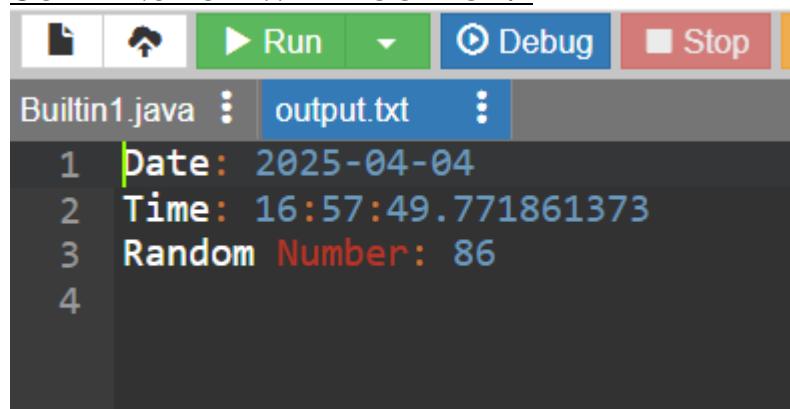
c)

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.Random;
import java.util.Date;
import java.io.FileWriter;
import java.io.IOException;

public class Builtin1 {
    public static void main(String[] args) {
        LocalDate today = LocalDate.now();
        LocalTime now = LocalTime.now();
        System.out.println("Today's Date: " + today);
        System.out.println("Current Time: " + now);

        Random random = new Random();
        int randNumber = random.nextInt(100);
        System.out.println("Random Number: " + randNumber);

        // File
        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Date: " + today + "\n");
            writer.write("Time: " + now + "\n");
            writer.write("Random Number: " + randNumber + "\n");
            writer.close();
            System.out.println("Data successfully written to file.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

SCREENSHOT WITH OUTPUT:-

```
1 Date: 2025-04-04
2 Time: 16:57:49.771861373
3 Random Number: 86
4
```

The screenshot shows a Java development environment with the following details:

- File Bar:** Contains icons for file operations (New, Open, Save, etc.), "Run" (with dropdown), "Debug", "Stop", "Share", "Save", "Beautify", and a download icon.
- Project View:** Shows two files: "Builtin1.java" and "output.txt".
- Code Editor:** Displays the source code for "Builtin1.java". The code imports various Java.time and java.util packages. It prints the current date and time, generates a random number, and writes all data to a file named "output.txt".
- Output Console:** Shows the execution results:
 - Today's Date: 2025-04-04
 - Current Time: 16:57:49.771861373
 - Random Number: 86
 - Data successfully written to file.
- Message:** "...Program finished with exit code 0" followed by "Press ENTER to exit console."

d)

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

public class Builtin2 {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);

        AtomicInteger counter = new AtomicInteger(0);

        System.out.println("Starting concurrent tasks...");

        for (int i = 1; i <= 5; i++) {
            final int taskNum = i;
            executor.submit(() -> {
                try {
                    System.out.println("Task " + taskNum + " started by " +
                        Thread.currentThread().getName());
                    Thread.sleep((long)(Math.random() * 1000));

                    int newCount = counter.incrementAndGet();
                    System.out.println("Task " + taskNum + " completed. Counter: " + newCount);

                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
                return null;
            });
        }

        executor.shutdown();
        try {
            if (!executor.awaitTermination(5, TimeUnit.SECONDS)) {
                System.out.println("Not all tasks completed in time.");
                executor.shutdownNow();
            } else {
                System.out.println("\nAll tasks completed successfully.");
                System.out.println("Final counter value: " + counter.get());
            }
        } catch (InterruptedException e) {
            executor.shutdownNow();
            Thread.currentThread().interrupt();
        }
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Code Editor:** The file `Builtin2.java` is open. The code uses `ExecutorService` to run five tasks concurrently. Each task prints its task number, the thread name, and a random sleep duration. It then increments a shared counter and prints the final counter value.
- Output Terminal:** The terminal window shows the execution of the program. It prints five lines, each starting with "Task" followed by a task number (1-5), the thread name, and a message indicating completion. After all tasks are completed, it prints "All tasks completed successfully." and "Final counter value: 5".
- Status Bar:** The status bar at the bottom right indicates "input".

```
1 import java.util.concurrent.ExecutorService;
2 import java.util.concurrent.Executors;
3 import java.util.concurrent.TimeUnit;
4 import java.util.concurrent.atomic.AtomicInteger;
5
6 public class Builtin2 {
7     public static void main(String[] args) {
8         ExecutorService executor = Executors.newFixedThreadPool(3);
9
10        AtomicInteger counter = new AtomicInteger(0);
11
12        System.out.println("Starting concurrent tasks...");
13
14        for (int i = 1; i <= 5; i++) {
15            final int taskNum = i;
16            executor.submit(() -> {
17                try {
18                    System.out.println("Task " + taskNum + " started by " +
19                        Thread.currentThread().getName());
20                    Thread.sleep((long)(Math.random() * 1000));
21
22                    int newCount = counter.incrementAndGet();
23                    System.out.println("Task " + taskNum + " completed. Counter: " + newCount);
24
25                } catch (InterruptedException e) {
26                    Thread.currentThread().interrupt();
27                }
28            });
29        }
30    }
31 }
```

Task 2 completed. Counter: 4
Task 1 completed. Counter: 5

All tasks completed successfully.
Final counter value: 5

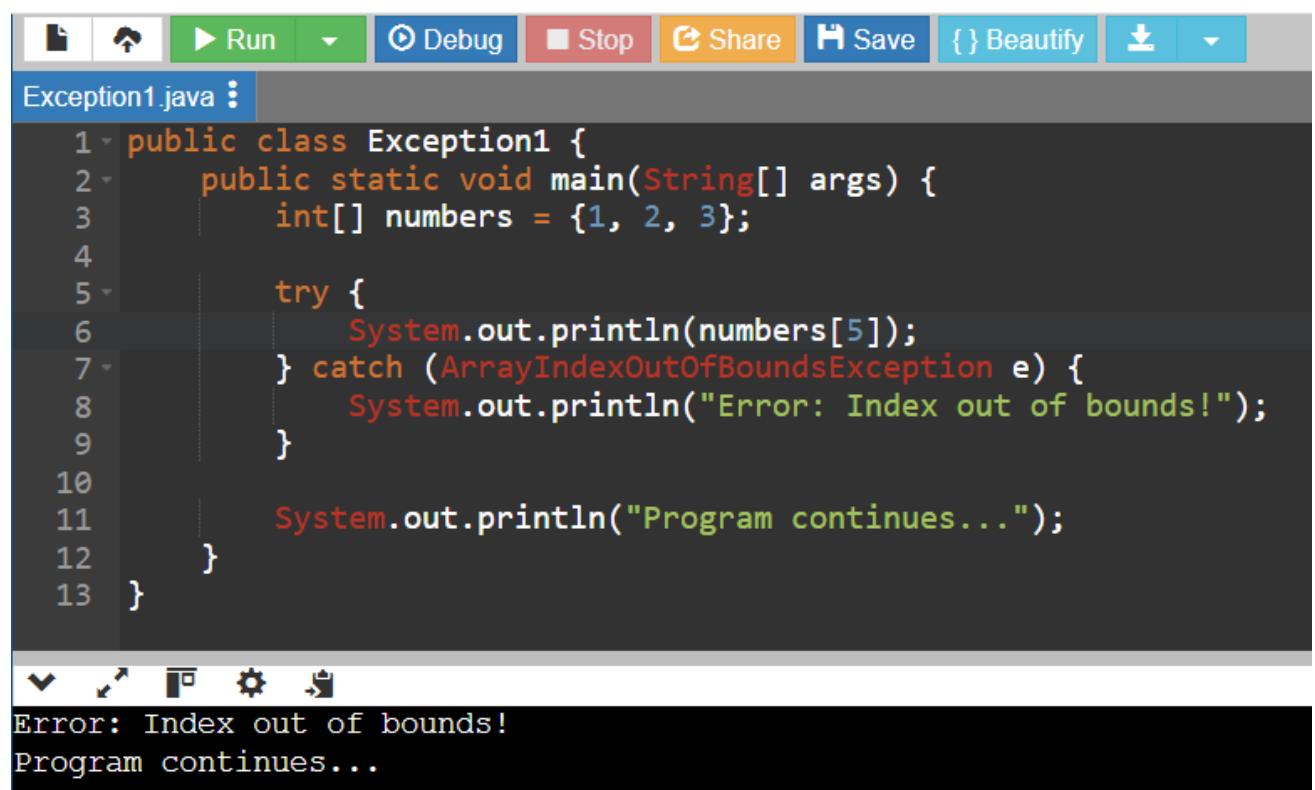
...Program finished with exit code 0
Press ENTER to exit console.

17.Exception Handling Programmes:-

a)

```
public class Exception1 {  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3};  
  
        try {  
            System.out.println(numbers[5]);  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Error: Index out of bounds!");  
        }  
  
        System.out.println("Program continues...");  
    }  
}
```

SCREENSHOT WITH OUTPUT:-



The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, Save, Beautify, and a download icon.
- Code Editor:** Displays the source code for `Exception1.java`. The code is as follows:

```
1  public class Exception1 {  
2      public static void main(String[] args) {  
3          int[] numbers = {1, 2, 3};  
4  
5          try {  
6              System.out.println(numbers[5]);  
7          } catch (ArrayIndexOutOfBoundsException e) {  
8              System.out.println("Error: Index out of bounds!");  
9          }  
10         System.out.println("Program continues...");  
11     }  
12 }  
13 }
```

- Output Console:** Shows the execution results:

```
Error: Index out of bounds!  
Program continues...
```

b)

```
class Exception2 {  
    public static void main(String[] args) {  
        try {  
            int a = 10, b = 0;  
            int result = a / b;  
            System.out.println("Result: " + result);  
        } catch (ArithmaticException e) {  
            System.out.println("Error: Cannot divide by zero!");  
        }  
    }  
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following interface elements:

- Toolbar: Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, and Beautify.
- Code Editor: Displays the Java code for `Exception2.java`. The code defines a class `Exception2` with a `main` method. It attempts to divide 10 by 0, which triggers an `ArithmaticException`, and prints an error message.
- Terminal/Output Area: Shows the execution results:
 - The output of the program: `Error: Cannot divide by zero!`
 - The final message: `...Program finished with exit code 0`
 - The prompt: `Press ENTER to exit console.`

c)

```

class ExceptionExample_3 {
    static void checkAge(int age) throws IllegalArgumentException {
        if (age < 18) {
            throw new IllegalArgumentException("Access Denied: Age must be 18 or above.");
        } else {
            System.out.println("Access Granted.");
        }
    }
}

public class Exception3 {
    public static void main(String[] args) {
        try {
            ExceptionExample_3.checkAge(15);
        } catch (IllegalArgumentException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
        System.out.println("Program continues...");
    }
}

```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for Run, Debug, Stop, Share, Save, and Beautify.
- Code Editor:** Displays the source code for `Exception3.java`. The code consists of two classes: `ExceptionExample_3` and `Exception3`. The `checkAge` method in `ExceptionExample_3` throws an `IllegalArgumentException` if the age is less than 18. The `main` method in `Exception3` calls `checkAge(15)`, catches the exception, prints the error message, and then prints "Program continues...".
- Terminal Output:** Shows the execution results:


```
Exception caught: Access Denied: Age must be 18 or above.
Program continues...
```

d)

```
import java.util.*;
public class Exception4 {
    public static void main(String[] args) {
        detal n= new detal();
        n.mm();
    }
}
class detal{
    void mm(){
        try{
            Scanner inp= new Scanner(System.in);
            System.out.println("enter your age");
            int age=inp.nextInt();
            if (age<1){
                throw new Exception("age should be more than 1");
            }
        }
        catch (Exception e){
            System.out.println(e);
        }
        finally {
            System.out.println("the programm has ended sucessfully");
        }
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), Run, Debug, Stop, Share, and Beautify.
- Code Editor:** Displays the code for `Exception4.java`. The code defines a class `Exception4` with a `main` method. It creates an instance of a nested class `detal`, which contains a method `mm`. This method reads an integer from the user, checks if it's less than 1, and throws an exception if true. It then prints the age and the success message.
- Terminal Output:** Shows the execution of the program. The user enters "18" when prompted "enter your age". The program then prints "the programm has ended sucessfully".
- Bottom Status:** Shows the message "...Program finished with exit code 0" and "Press ENTER to exit console."

18.File Handiling Programes:-

a)

```
import java.io.BufferedReader;
import java.io.FileReader;

import java.util.ArrayList;

public class File1 {
    public static void main(String[] args) {
        ArrayList<String> arr =new ArrayList<>();

        try {
            FileReader fr = new FileReader("output.txt");
            BufferedReader br = new BufferedReader(fr);
            String v= br.readLine();

            int a=0;
            while (v!=null){
                System.out.println(v);
                arr.add(v);
                v= br.readLine();
                a+=1;
                System.out.println(a);
            }
        } catch (Exception e){
            System.out.println("error occured");
        }
        System.out.println(arr);

    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- File Menu:** File1.java, Save, Beautify.
- Toolbar:** Run, Debug, Stop, Share.
- Code Editor:** Contains Java code for reading lines from a file and printing them along with their line numbers. The code uses BufferedReader and ArrayList.
- Output Console:** Shows the error message "error occurred" and an empty array representation "[]".
- Bottom Status:** "...Program finished with exit code 0" and "Press ENTER to exit console."

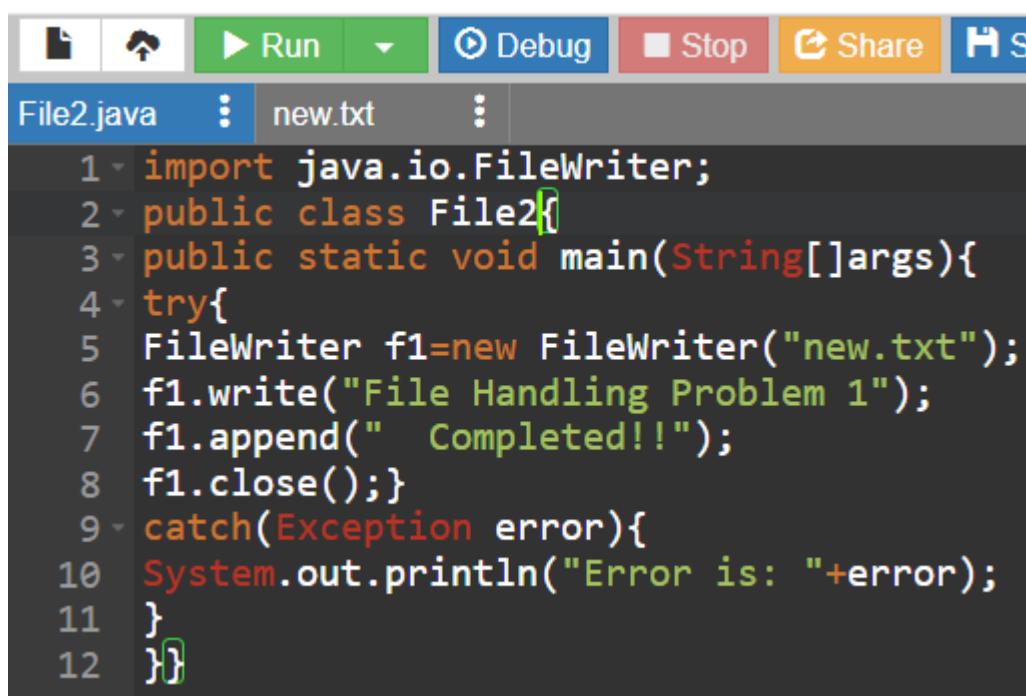
```
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3
4 import java.util.ArrayList;
5
6
7 public class File1 {
8     public static void main(String[] args) {
9         ArrayList<String> arr = new ArrayList<>();
10
11     try {
12         FileReader fr = new FileReader("output.txt");
13         BufferedReader br = new BufferedReader(fr);
14         String v= br.readLine();
15
16         int a=0;
17         while (v!=null){
18             System.out.println(v);
19             arr.add(v);
20             v= br.readLine();
21             a+=1;
22             System.out.println(a);
23         }
24     } catch (Exception e){
25         System.out.println("error occurred");
26     }
27     System.out.println(arr);
28 }
```

error occurred
[]

...Program finished with exit code 0
Press ENTER to exit console.

b)

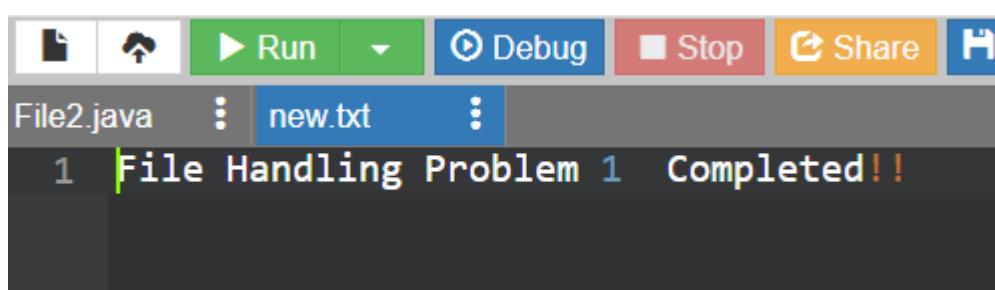
```
import java.io.FileWriter;
public class File2{
public static void main(String[]args){
try{
FileWriter f1=new FileWriter("new.txt");
f1.write("File Handling Problem 1");
f1.append(" Completed!!");
f1.close();
} catch(Exception error){
System.out.println("Error is: "+error);
}
}}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following interface elements:

- Toolbar: Run, Debug, Stop, Share.
- Project Explorer: File2.java, new.txt.
- Code Editor:

```
1 import java.io.FileWriter;
2 public class File2{
3 public static void main(String[]args){
4 try{
5 FileWriter f1=new FileWriter("new.txt");
6 f1.write("File Handling Problem 1");
7 f1.append(" Completed!!");
8 f1.close();
9 } catch(Exception error){
10 System.out.println("Error is: "+error);
11 }
12 }}
```



The screenshot shows the output window of the Java IDE displaying the contents of the new.txt file:

```
1 File Handling Problem 1 Completed!!
```

c)

```
import java.io.File;

public class File3 {
    public static void main(String[] args) {
        File myFile = new File("example.txt");
        if (myFile.delete()) {
            System.out.println("Deleted the file: " + myFile.getName());
        } else {
            System.out.println("Failed to delete the file.");
        }
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- File3.java:** The current file being edited.
- Output Terminal:** Shows the program's execution and output.
 - Line 1: Failed to delete the file.
 - Line 2: ...Program finished with exit code 0
 - Line 3: Press ENTER to exit console.

d)

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

public class File4 {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("system.txt", true);
            writer.write("\nAppending this line to the file.");
            writer.close();
            System.out.println("Successfully appended to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
        }
    }
}
```

SCREENSHOT WITH OUTPUT:-

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save), run (Run, Debug, Stop), share, and beautify.
- Code Editor:** Displays the Java code for `File4.java`. The code uses `FileWriter` to append a line to a file named `system.txt`.
- Output Terminal:** Shows the execution results:
 - The output message: `Successfully appended to the file.`
 - The completion message: `...Program finished with exit code 0`
 - The prompt: `Press ENTER to exit console.`

The screenshot shows a Java development environment. At the top, there's a toolbar with icons for file operations like Open, Save, Run, Debug, Stop, and Share. Below the toolbar, there are two tabs: 'File4.java' and 'system.txt'. The 'system.txt' tab is currently active, displaying the following code:

```
1 Appending this line to the file.
```

Name:-Musali Reddy Manish Reddy
Registration no:-CH.SC.U4CSE24129