**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB MANUAL**



**Department of computer and communication Engineering**

**Amrita School of Engineering**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name:M.Priya darshini**

**Verified By Roll No: AV.SC.U4CSE24221**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNo | Title | Date | Page No | Signature |
| WEEK 1 |  |  |  |  |
| 1 | Download and Install Java Software |  |  |  |
| 2 | Write a Java program to print the message “Welcome to Java Programming.” |  |  |  |
| WEEK 2 |  |  |  |  |
| 1 | Write a Java program to calculate area of rectangle. |  |  |  |
| 2 | 1. Write a Java program to convert temperature from Fahrenheit to Celsius. 2. Write a Java program to convert temperature from Celsius to Fahrenheit. |  |  |  |
| 3 | Write a Java program to calculate fibonacci of a number. |  |  |  |
| 4 | Write a java program to calculate the area of the triangle. |  |  |  |
| 5 | Write a Java program to calculate factorial of a number. |  |  |  |
| 6 | Write a Java program to calculate Simple interest. |  |  |  |
| WEEK 3 |  |  |  |  |
| 1 | Write a java program with  1. A class with name Car.  2.Create 4 attributes named car\_color , car\_brand ,fuel\_type ,mileage. 3. Create 3 methods named start() ,stop() ,service(). 4.Create 3 objects named car1 ,car2 ,car3. |  |  |  |
| 2 | Write a java program with 1.Create a class named Bankaccount. 2.Create a constructor. 3.Create 2 methods which are withdrawl() and deposit(). |  |  |  |
| WEEK 4 |  |  |  |  |
| 1 | Write a java program with class named book. This class should contain various attributes such as title, author ,year of publication. It should also contain a constructor with parameters which initializers title ,author ,year of publication. Create a method which displays the details of the book .Display the details of 2 books. |  |  |  |
| 2 | Create a java program with class named myclass with a static variable count of int type ,initial value to zero and a constant variable "pi" of type double initialize to 3.14 as attributes of that class ,now define a constructor for "myclass" that increments the count variable each time an object of myclass is created. Finally print the values of count and pi variables . Create 3 objects. |  |  |  |
| WEEK 5 |  |  |  |  |
| 1 | Create a calculator using the operations including addition, subtraction, multiplication, and division using multi-level inheritance and display the desired output. |  |  |  |
| 2 | A vehicle rental company wants to develop a system that maintains information about different types of vechicles available for rent the company rents out cars and bikes, and they need a program to store details about each vehicle, such as brand and speed( should be in super class)  1.cars should have an additional property: no.of doors 2.Bikes should have a property indicating whether they have gears or not. 3.The system should also include a function to display details about each vehicle and indicate when a vehicle is starting. 4.Every class should have a constructor. **Question:**  1.Which oops concept is used in the above program  2.If the company decides to add a new type of vehicle, Truck, how would you modify the program?  a.Truck should include an additional property capacity (in tons)  b.Create a showTruckdetails() method to display the truck’s capacity.  c.Write a constructor for Truck that initializes all properties.Implement the truck class and update the main method to create a Truck object and also create an object for car and bike sub classes Finally, display the details. |  |  |  |
| WEEK 6 |  |  |  |  |
| 1 | Write a java program to create a vehicle class with a method displayinfo(). Override this method in the car subclass to provide specific information about a car. |  |  |  |
| 2 | A college is developing an automated admissions systems that verifies students eligibility for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility. Criteria based on the students percentage in their previous qualification.  1. UG admission require min of 60%  2. PG admission require min of 70% |  |  |  |
| 3 | Create a calculator class with overloaded methods to perform addition.  A. Add two integers  B. Add two double  C. Add three integer |  |  |  |
| 4 | Create a shape class with a method CalculateArea() that is overloaded for different shapes (e.g square, rectangle) then, create a subclass circle that overrides the calculatearea() method for a circle. |  |  |  |
| WEEK 7 |  |  |  |  |
| 1 | Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion amd Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal. |  |  |  |
| 2 | Write a java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extends Shape3D class and implement the respective methods to calculate the Volume and SurfaceArea of each shape. |  |  |  |
| 3 | Write a Java program using an abstract class to define a method for pattern printing.  • Create an abstract class named pattern printer with an abstract method printPattern(int n) and a concrete method to display the pattern title.  • Implement two subclasses :  1. Star Pattern - prints a right angled triangle of stars(\*)  2. Number Pattern – prints a right angled triangle of increasing numbers. In the main() method, Create objects of both subclasses and print the patterns for a given no.of rows. |  |  |  |
| WEEK 8 |  |  |  |  |
| 1 | Write a java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle and Triangle that implement the Shape interface, Implement the getPerimeter() method for each of the three classes. |  |  |  |
| 2 | Write java program to create an interface Playable with a method play() that takes no arguments and returns void. Create 3 classes Football , Volleyball and Basketball that implements the Playable interface and override the play() method to play the respective sports. |  |  |  |
| WEEK 9 |  |  |  |  |
| 1 | Write a java program to create a method that takes integer as parameter on throws an exception if the number is even. |  |  |  |
| 2 | Write a java program to create method that reads a file and throws an exception if the file is not found. |  |  |  |
| 3 | Write a java program to handle arithematic exception using try,catch,finally. |  |  |  |
| 4 | Write a java program to stimulate a university system using inner class.  1.Create an outer class named university with variable university name.  2.Inside it ,define two non-static inner classes  a. Department with variables like deptname and deptcode ,and a method to display department details.  b. Student with variables like studentName and Rollnumber ,and a method to display Student details.  c.Create an object for each class and call their method to display their details along with the university name. |  |  |  |
| WEEK 10 |  |  |  |  |
| 1 | Write a java program to generate a password for a student using his/her initials and age. The password displayed should the string consists of first characters of first name, middle name, last name with age. |  |  |  |
| 2 | Design and implement a java program that will do the following operations to this string “Welcome! You Are Practicing Strings Concept”.  1. Convert all the alphabets to capital letters and print out the results  2. Convert all the alphabets to lower case letters and print out the result.  3.print out the length of the string.  4.print out the index of Course. |  |  |  |
| 3 | Implement a java program using the below array methods.  1.Sorting the elements (numbers and strings) of an array  2.Convert the array elements into string.  3. Copy the elements of one array into another . |  |  |  |
| 4 | Implement a java program using the below array list methods.  1. Insert an element at a particular index array list.  2. Modify an element in the array list.  3. Access an element in the array list  4. Remove an element in the array list.  5. Clear the elements from the array list. |  |  |  |

**WEEK 01**

**PROGRAM-1:**

**AIM:** Download and Install Java Software

**PROCEDURE:**

**Step 1: Download JDK 21**

1.Open chorme wed brower and go to the Oracle JDK download page.

2.Go to down Java SE Development Kit 21 section.

3. Choose the Windows x64 Installer version.

4.Click on Download, then Wait for the download to complete.



**Step 2:** **Install JDK 21**

1.Run the downloaded installer jdk-21\_windows-x64\_bin.exe file.

2. Double-click to launch the installer.

3.click “next” to start the installation process.

4.Accept the license agreement and click next.

5. Choose the installation path (default is C:\Program Files\Java\jdk-21).

6.Wait for the installation to complete.

7.Click Close to the installation is finished.



**Step 3: Setting up the path**

1.Go to “Windows C” Drive on Desktop.

2. Choose Program Files, select Java, then JDK 21, then select Bin.

3. Select and copy the path at the address bar.



**Step 4: Open System Properties**

1.Press Windows + R, type sysdm.cpl , and click Ok-

2.The System Properties window will open.

3.Navigate to the Advanced tab.

4.Click on Environment Variables at the bottom.



**Step 5: Set JAVA\_HOME**

1.Under System Variables, click New.

2.Set the Variable name as JAVA\_HOME.

3.Set Variable value as C:\Program Files\Java\jdk-21 (or your installation path).

4.Click OK.



**Step 6: Update PATH Variable**

1.In System Variables, find Path and click Edit.

2.Click New and add: C:\Program Files\Java\jdk-21\bin

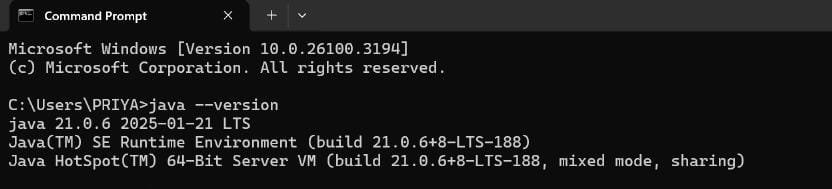
3.Click OK to save.



**Step 7:Verify Installation**

1.Open a command prompt or terminal window.

2. Type java -version: Type java -version to verify that Java is installed correctly.



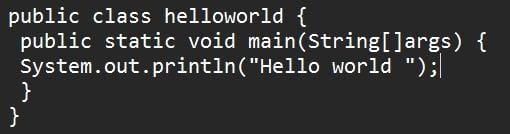
3. To check the java compiler type: javac –version.



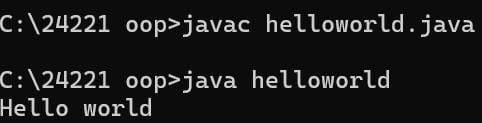
**PROGRAM-2:**

**AIM:** Write a Java program to print the message “Welcome to Java Programming.”

**CODE:**



**Output:**



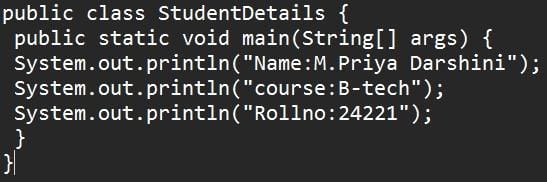
**ERRORS:** None found

**PROGRAM-3:**

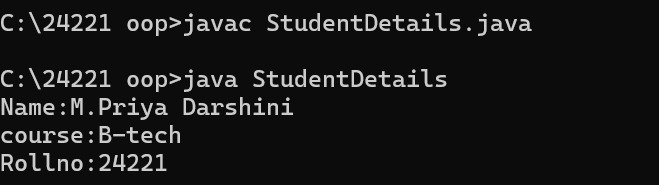
**AIM:** Write a Java Program that prints Name, Roll No, Section of a student.

**CODE:**

**STUDENT DETAILS:**



**OUTPUT:**



**ERROR TABLE:**

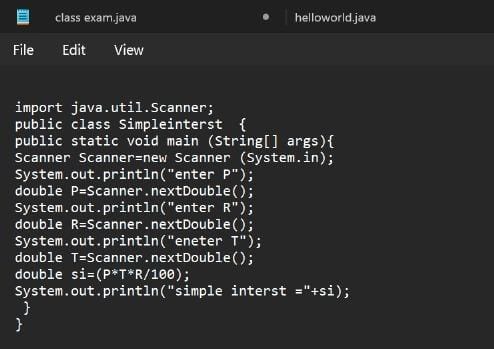
|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1.writing small “S” in place of” S”  In (string [] | 1.code is rectified by keeping capital “S” |

**WEEK – 2**

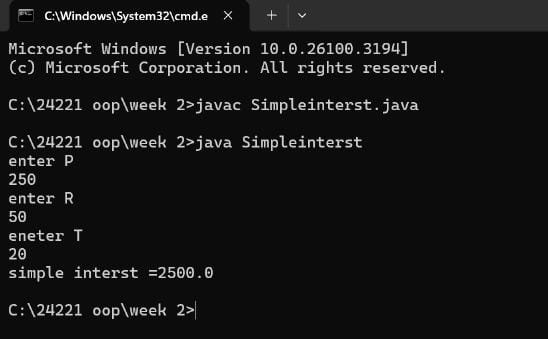
**PROGRAM-1:**

**AIM:** Write a java program to find the simple interest where all the inputs are taken from the user.

**CODE:**



OUTPUT:



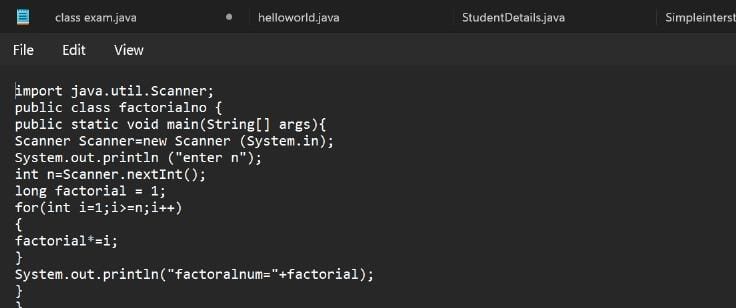
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
|  | ERROR | ERROR RECTIFICATION |
| 1) | ; is expected after input.close() | Corrected by writing a ; after input.close() |
| 2) | Error in the line double principal=input.next.double(); | Corrected by removing “. “after the next. |

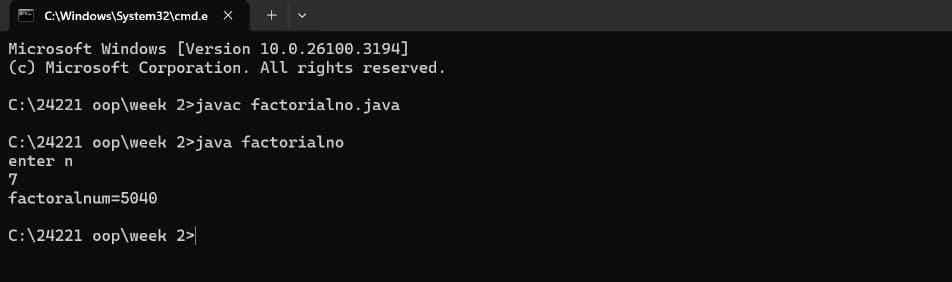
**PROGRAM-2:**

**AIM:** Write a java program to find the factorial of a number where all the inputs are taken from a user.

**CODE:**



**OUTPUT:**



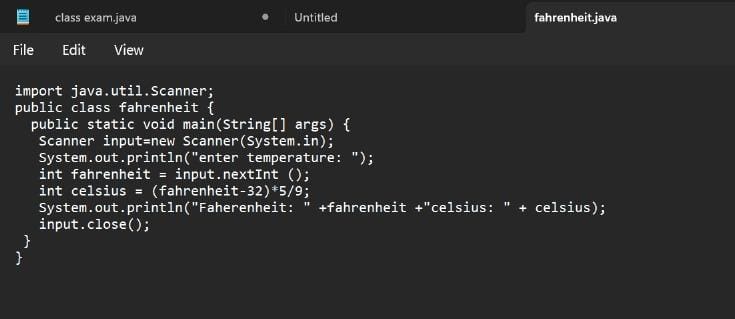
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
|  | **ERROR** | **ERROR RECTIFICATION** |
| 1) | writing small “S” in place of “S” in system .out. println () | code is rectified by keeping capital “S” |
| 2) | “Is missed in the print statement | Corrected by keeping “in the end of print statement. |

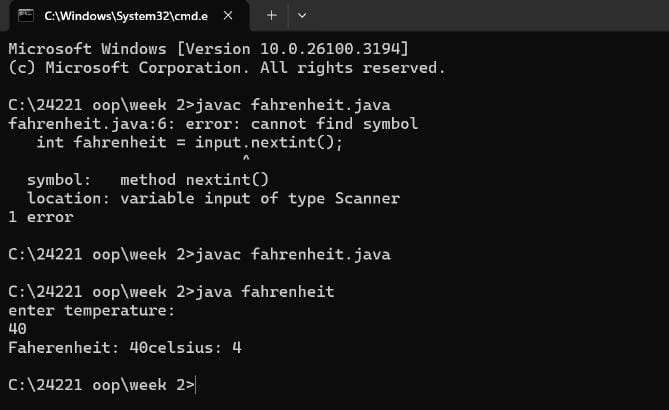
**PROGRAM-3:**

**AIM:** Write a java program to convert the temperature from Celsius to Fahrenheit and Celsius to Fahrenheit.

**CODE:**



**OUTPUT:**



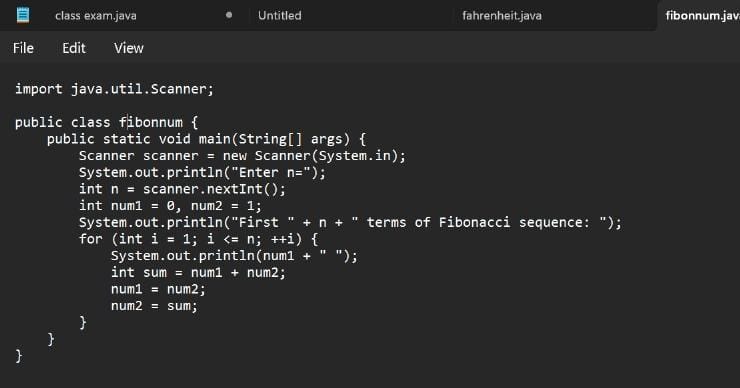
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **ERROR** | **ERROR RECTIFICATION** |
| 1. | Logical error  Missed \* after Celsius in code 1 | Corrected by keeping a \* operator in the appropriate place |

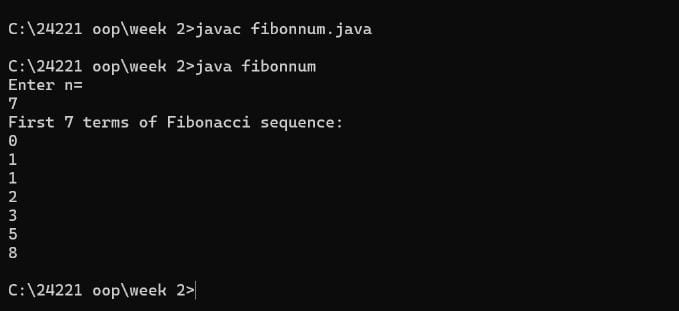
**PROGRAM – 4:**

**AIM:** Write a java program to find the Fibonacci series of a given number where all the inputs are taken from the user.

**CODE:**



**OUTPUT:**

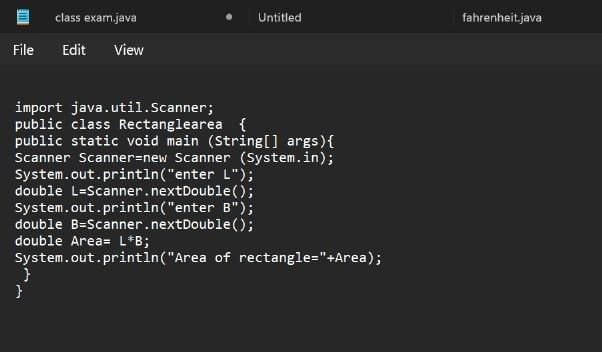


ERROR TABLE:

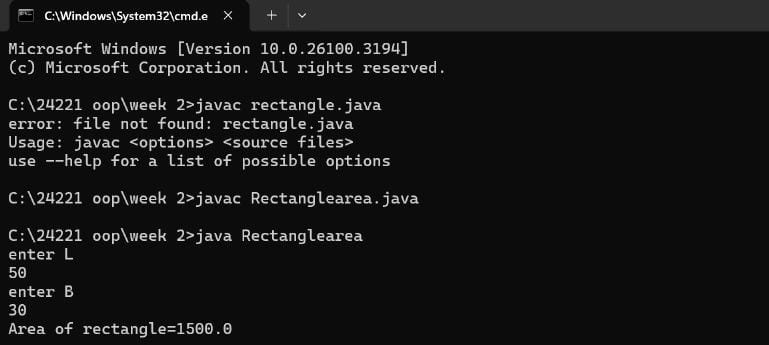
|  |  |  |
| --- | --- | --- |
| S.NO | ERROR | ERROR RECTIFICATION |
| 1. | Expected an; after line 4 | Corrected by keeping an ; after the line |

**PROGRAM-5:**

**AIM:** Write a java program to find the area of rectangle and triangle.

**CODE:** 

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1.Class name is rectangle area and error is small”r”. | 1.code is rectified keeping of capital”R”. |

**WEEK –3**

**PROGRAM-1:**

**AIM:** Write a java program with

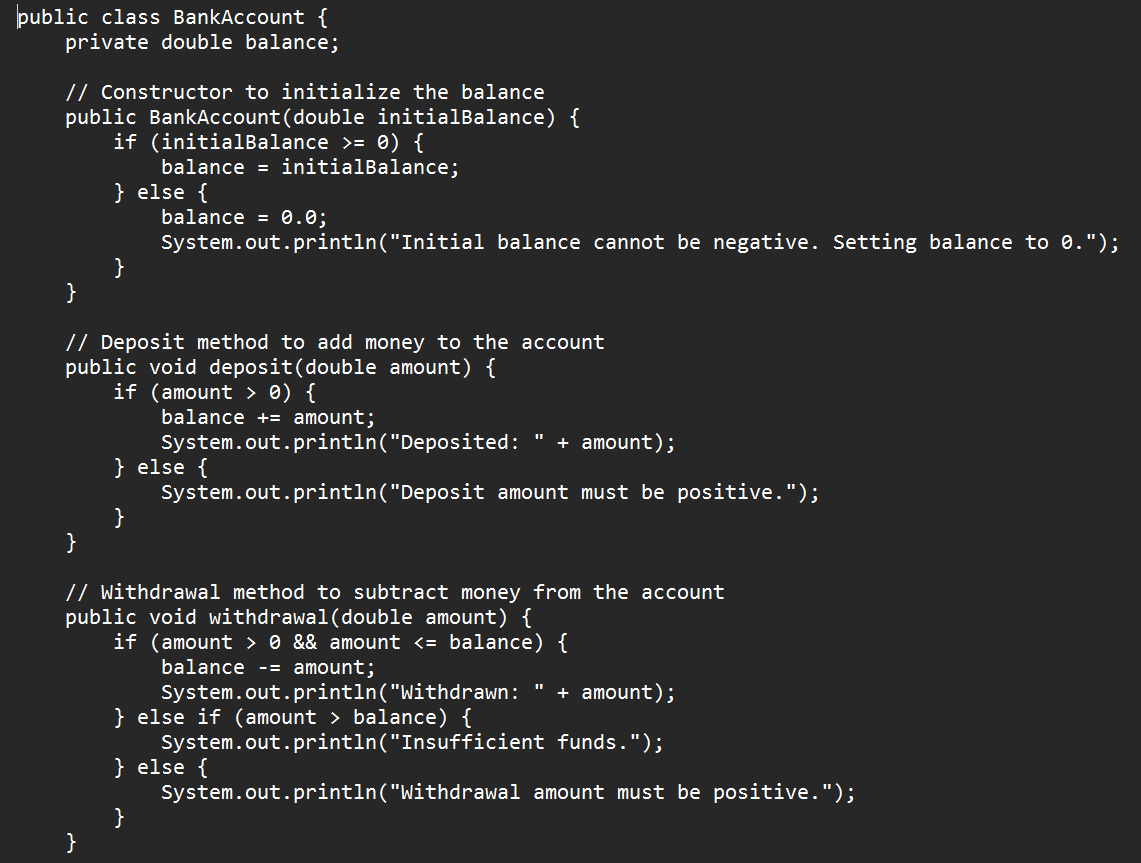
1.Create a class named Bankaccount.

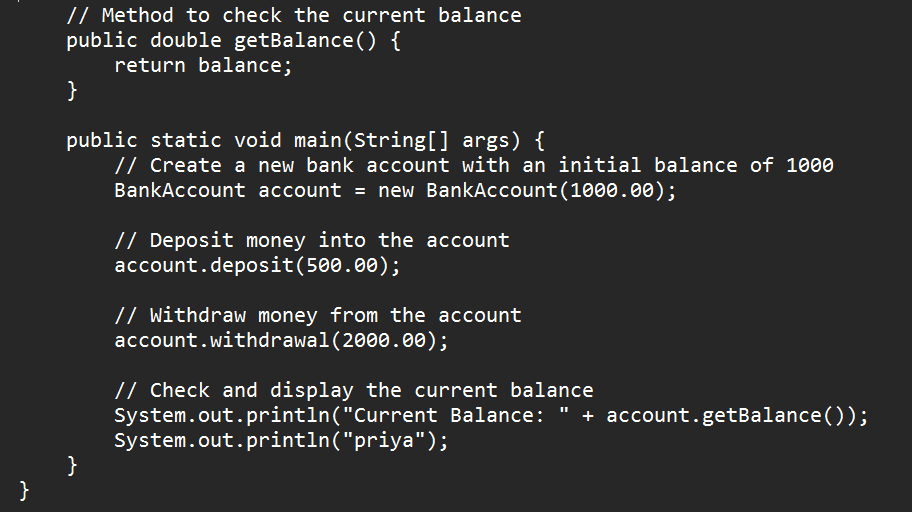
2. Create a constructor.

3. Create 2 methods which are withdrawl() and deposit ()

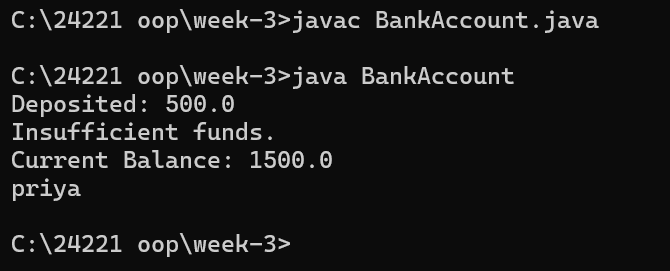
|  |
| --- |
| Bank account |
| -balance:double |
| +bank account initial  Balance:double  +deposit amount:double  +withdrawal:double  +getbalance():double |

**CODE:**

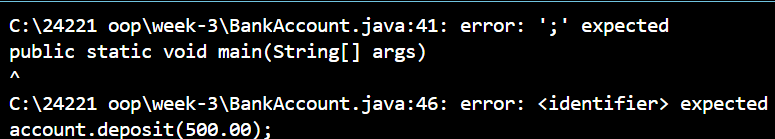




**OUTPUT:**

**POSITIVE CASE:** 

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Non-Existent Methods  Ex: By typing titel instead of title while calling the method | Rectifying the method name as title |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**IMPORTANT POINTS:**

Encapsulation:

* Using private fields and public methods to protect the data and control access.

Conditionals:

* Using if statements to enforce rules and conditions

**Program-2**

**AIM:**To create java program with following instructions

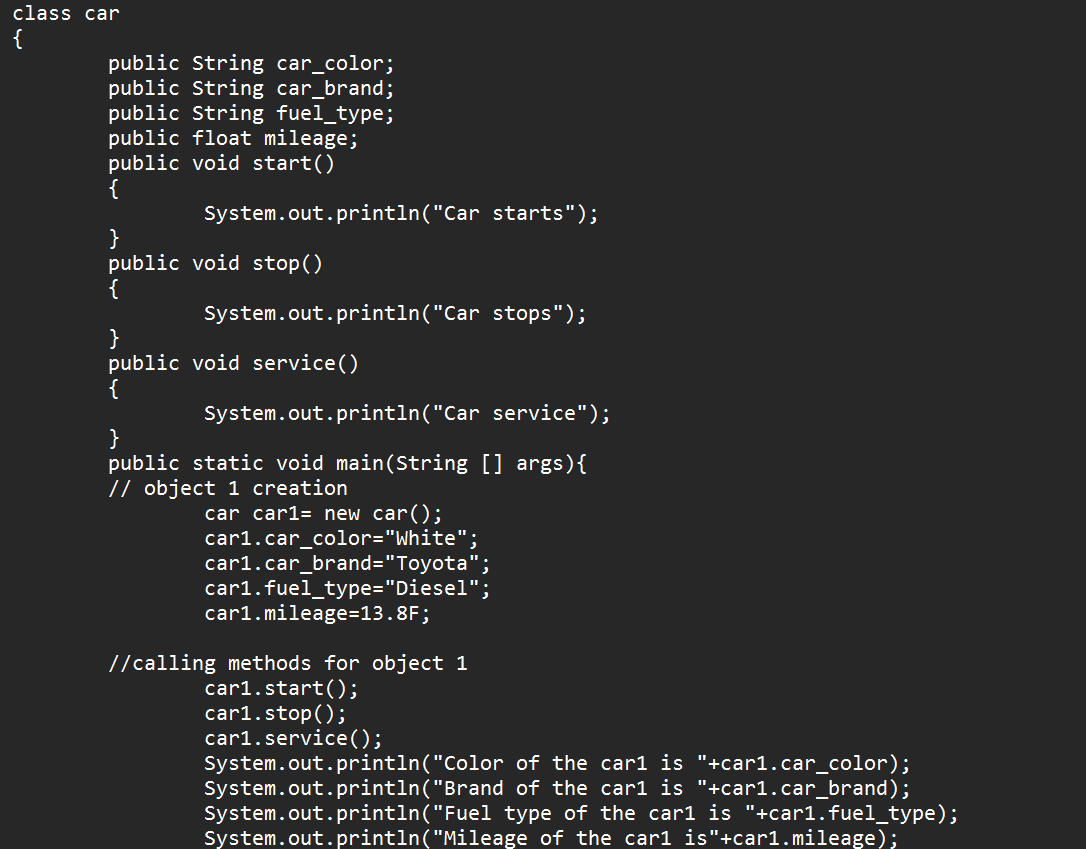
1. Create a class with name Car.
2. Create four attributes named Car\_color, Car\_brand, fuel\_type, mileage.
3. Create three methods named start (), stop (), service ().

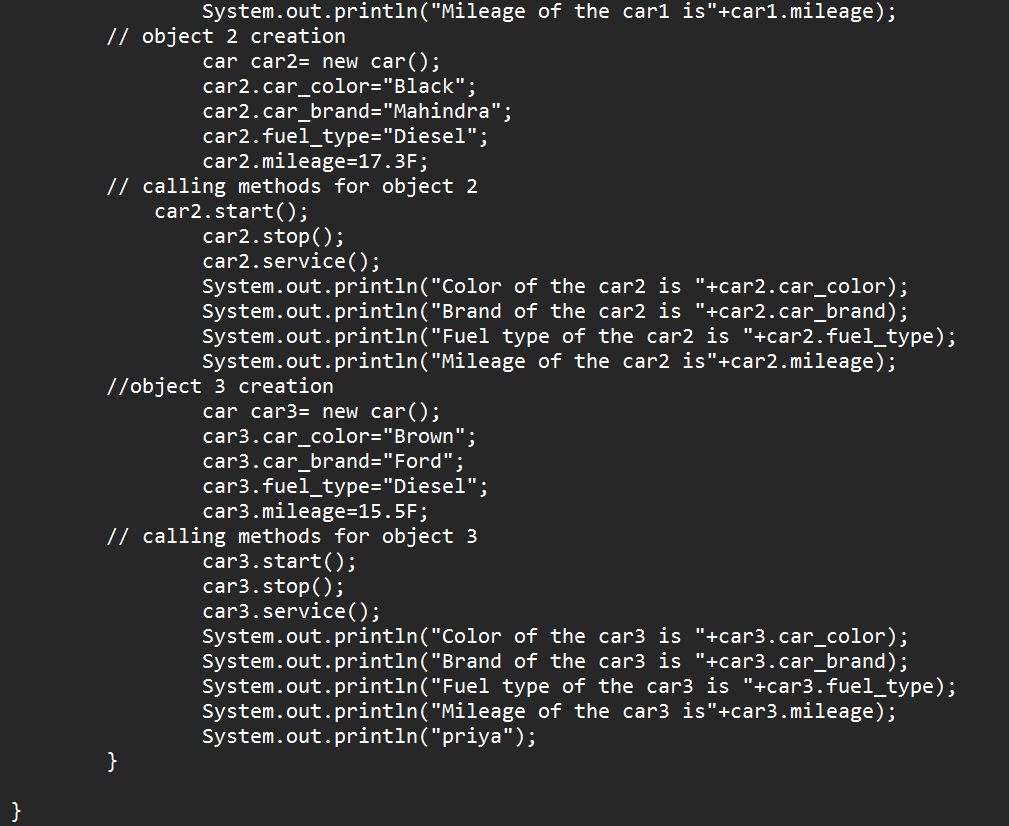
4)Create three objects named Car1, Car2, Car3**.**

**CLASS DIAGRAM:**

|  |
| --- |
| Car |
| + Car\_color: String  + Car\_brand: String  - fuel\_type: String  - mileage: double |
| + start(): void  + stop(): void  - service(): void  + Car1(): void  + Car2(): void  + Car3(): void |

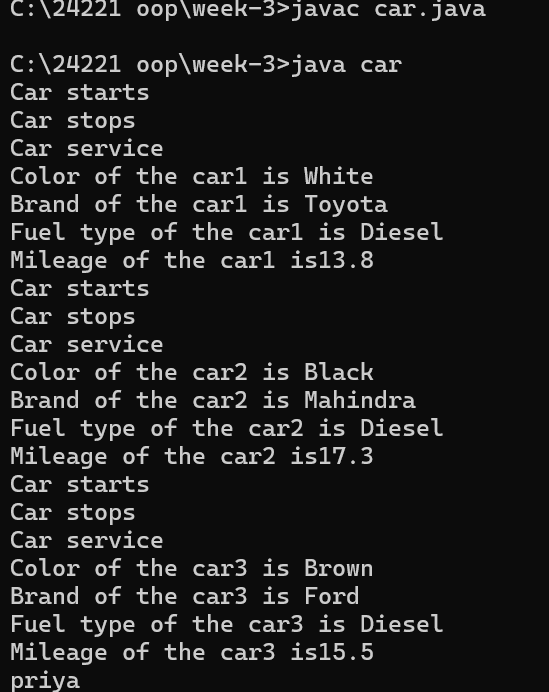
**CODE:**



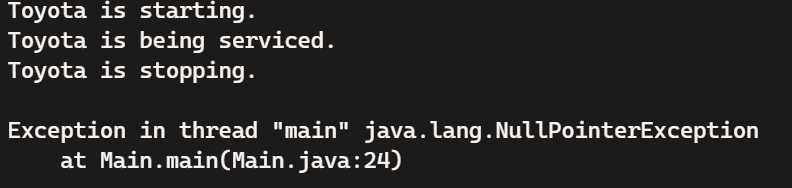


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Null Methods which are not define | Rectifying the correct method names |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**Important Points:**

1. Classes and Objects:
   * Class: A blueprint for creating objects (a particular data structure), containing methods and variables.
   * Object: An instance of a class.
2. Attributes:
   * Variables within a class that represent the state or properties of an object.
3. Constructor:
   * A special method used to initialize objects. It is called when an object of a class is created.
4. Methods:
   * Functions defined within a class that represent the behavior or actions that an object can perform

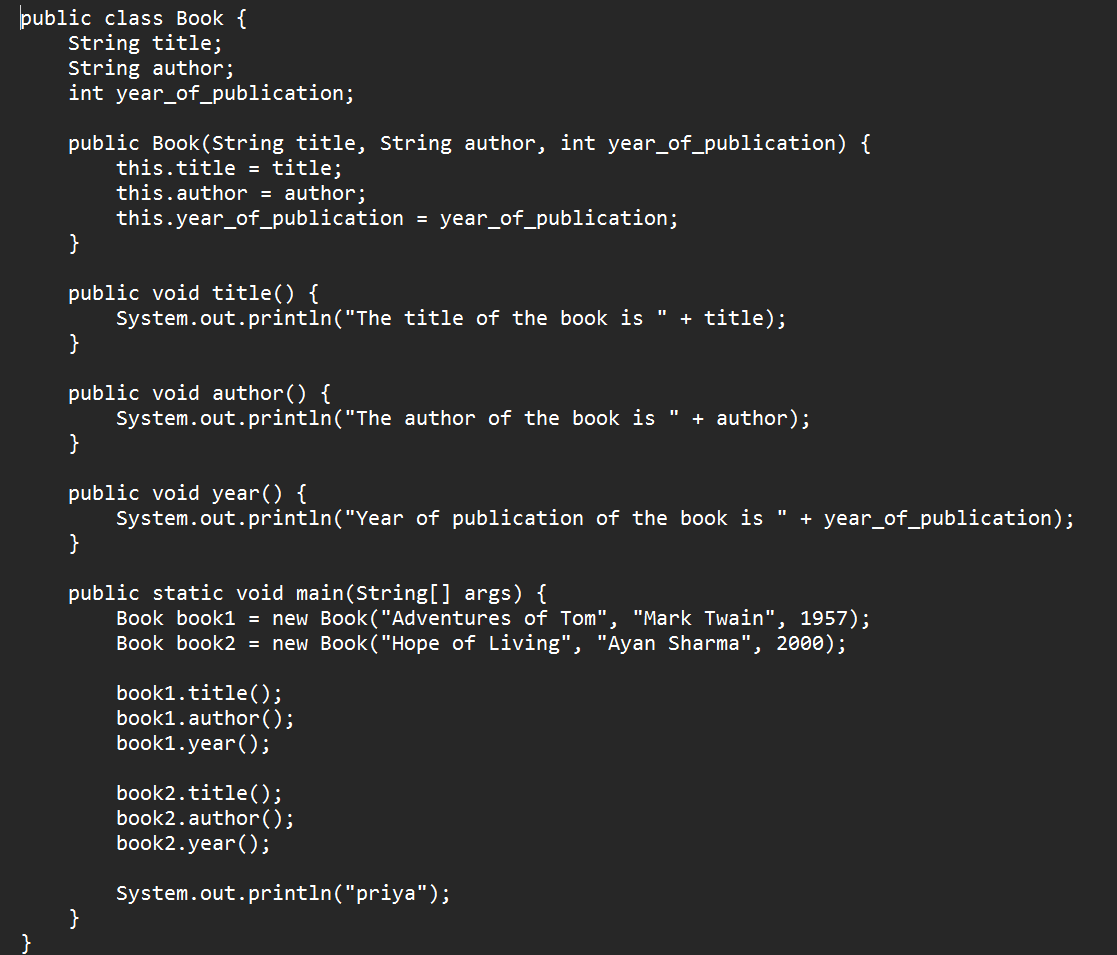
**WEEK-4**

**PROGRAM-1**

**Aim:** Write a java program with class named book. The class should contain various attributes such as title, author, year of publication. It should also contain a constructor with parameters which initializes title, author and year of publication. Create a method which displays the details of the book. Display the details of two books.

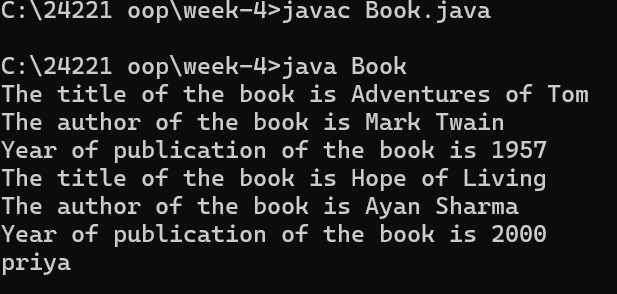
|  |
| --- |
| Book |
| - title: String  - author: String  - year\_of\_publication: int |
| + Book(String, String, int)  + title(): void  + author(): void  + year(): void  + main(String[]): static void |

**CODE:**

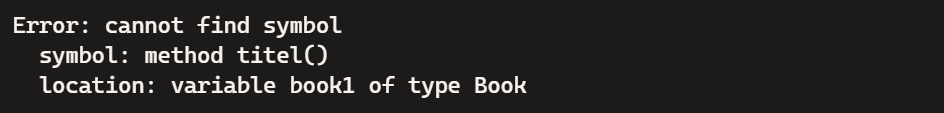


**OUTPUT**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Non-Existent Methods  Ex: By typing titel instead of title while calling the method | Rectifying the method name as title |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**IMPORTANT POINTS:**

Using a constructor to initialize object attributes when a new object is created. Defining methods (title(), author(), year()) to perform specific tasks within the class.Using the main method as the entry point to the program.

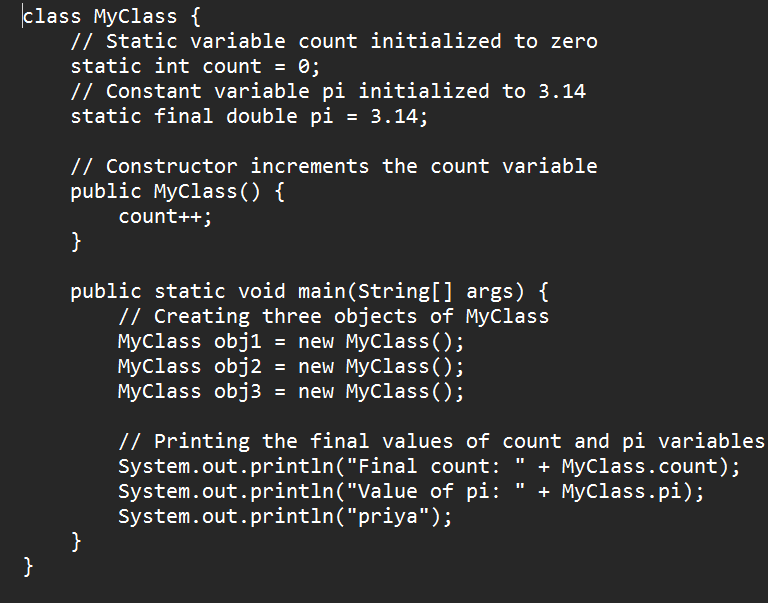
**PROGRAM-2:**

**AIM:** Create a java program with class named “MyClass” with a static variable count of int type initialized to zero and a constant variable ‘pi’ of type double initialized to 3.14 as attributes of that class. Now define a constructor for “MyClass” that increments the count variable each time an object of “MyClass” is created. Finally print the final values of count and pi variables. Create three variables.

**CLASS DIAGRAM:**

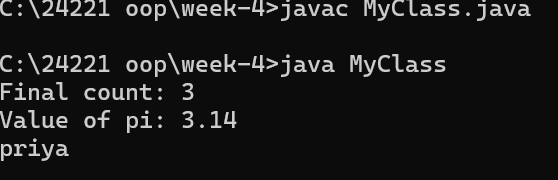
|  |
| --- |
| MyClass |
| - count: static int  - pi: final double |
| + MyClass ()  + main (String []): static void |

**CODE:**



**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Non-Existent Methods  Ex: By typing Myclass instead of MyClass while calling the method | Rectifying the method name as MyClass |
| 2 | S is System.out.println is given in lowercase letters | The error is rectified by replacing the lowercase letter s into uppercase letter S |

**Important Points:**

1. **Static Variables**:
   * count: A static variable shared among all instances of the class. It keeps track of the number of objects created.
   * pi: A static constant variable, its value remains the same for all instances.
2. **Constructor**:
   * Increments the count variable each time an object is created.
3. **Static Context**:
   * Accessing static variables and methods using the class name (MyClass.count, MyClass.pi

**WEEK-5**

**PROGRAM-1**

**AIM:** Create a calculator using the operations including addition, subtraction, multiplication and division using Multilevel Inheritance and display the desired output.

**CLASS DIAGRAM:**

**MULTILEVEL INHERITANNCE**

|  |
| --- |
| Calculator |
| - num1: int  - num2: int |
| + add(): int  + subtract(): int |

|  |
| --- |
| Calculator1 |
| - num1: int  - num2: int |
| + multiply(): int |

|  |
| --- |
| Calculator2 |
| - num1: int  - num2: int |
| + divide(): int |

**CODE:**

class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int subtract(int a, int b) {

        return a - b;

    }

}

class Calculator1 extends Calculator {

    public int multiply(int a, int b) {

        return a \* b;

    }

}

class Calculator2 extends Calculator1 {

    public double divide(int a, int b) {

        return (double) a / b; // Fixed: Ensure division returns a double

    }

}

public class CalculatorTest {

    public static void main(String[] args) {

        Calculator2 calc = new Calculator2();

        int num1 = 10, num2 = 5;

        System.out.println("num1 = " + num1);

        System.out.println("num2 = " + num2);

        System.out.println("Addition: " + calc.add(num1, num2));

        System.out.println("Subtraction: " + calc.subtract(num1, num2));

        System.out.println("Multiplication: " + calc.multiply(num1, num2));

        System.out.println("Division: " + calc.divide(num1, num2));

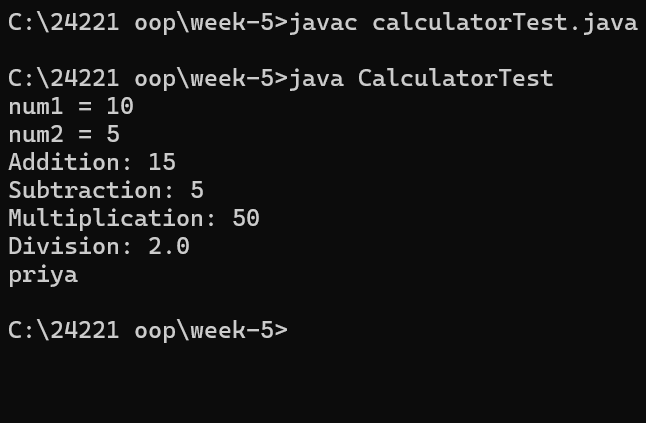
        System.out.println("priya");

    }

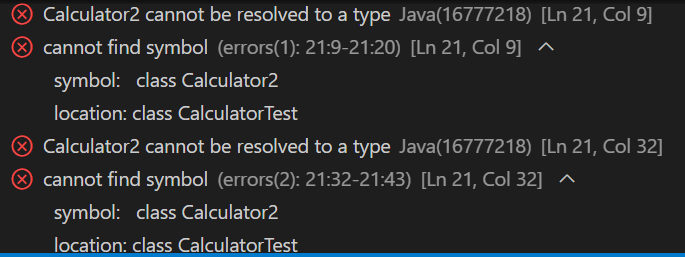
}

**Output:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Given wrong class name while object is created | Rectifying the given wrong class name |

**IMPORTANT POINTS**

1. Multilevel Inheritance:

* Multilevel inheritance occurs when a class is derived from another class, and then a third class is derived from the second class, forming a chain.

2. Code Reusability:

* The add() and subtract() methods are defined once in the Calculator class and can be reused by both the Calculator1 and Calculator2 classes without rewriting the code.
* The multiply() method is defined in Calculator1 and inherited by Calculator2
* The divide() method is added to the Calculator2 class.
* This shows how inheritance helps to reduce redundant code and increases reusability.

3. Polymorphism (Use of Parent Class Reference):

* The code demonstrates polymorphism (though not explicitly in the form of method overriding). Even though the object calc is of the class Calculator2, it can call methods from the base class (Calculator) and intermediate class (Calculator1) due to inheritance.

For example, calling calc.add(num1, num2) and calc.subtract(num1, num2) works because Calculator2 inherits from Calculator

**PROGRAM-2**

**AIM:** A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes and they need a program to store details about each vehicle such as brand and speed

* Cars should have an additional property: number of doors
* Bike should have a property indicating whether they have gears or not
* The system should also include a function to display details about each vehicle and indicate when a vehicle is starting
* Every class should have a constructor

1. Which OOP concept is used in the above program? Explain why it is useful in this scenario.
2. If the company decides to add a new type of vehicle truck, how would you modify the program? Truck should include an additional property capacity(in tons). Create a showTruckDetails() method to display the truck’s capacity. Write a constructor for truck that initializes all properties.
3. Implement the truck class and update the main method to create a truck object, also create an object for car and bike subclassed. Finally display its details.

**CLASS DIAGRAM**:

|  |
| --- |
| Vehicle |
| + brand: String  + speed: int |
| + Vehicle(brand: String, speed: int)  + start(): void  + displayDetails(): void |

|  |
| --- |
| Car |
| +numberOfDoors: int |
| +Car(brand:String, speed: int, numberOfDoors: int)  + displayDetails(): void |

|  |
| --- |
| Bike |
| + hasGears: boolean |
| + Bike(brand: String, speed: int, hasGears: boolean) +displayDetails(): void |

|  |
| --- |
| Truck |
| + Capacity : double |
| +Truck(brand: String, speed: int, Capacity: double)  +showTruckDetails(): void |

**CODE**: class Vehicle {

    String brand;

    int speed;

    public Vehicle(String brand, int speed) {

        this.brand = brand;

        this.speed = speed;

    }

    public void start() {

        System.out.println(brand + " vehicle is starting...");

    }

    public void displayDetails() {

        System.out.println("Brand: " + brand);

        System.out.println("Speed: " + speed + " km/h");

    }

}

class Car extends Vehicle {

    int numberOfDoors;

    public Car(String brand, int speed, int numberOfDoors) {

        super(brand, speed);

        this.numberOfDoors = numberOfDoors;

    }

    @Override

    public void displayDetails() {

        super.displayDetails();

        System.out.println("Number of Doors: " + numberOfDoors);

    }

}

class Bike extends Vehicle {

    boolean hasGears;

    public Bike(String brand, int speed, boolean hasGears) {

        super(brand, speed);

        this.hasGears = hasGears;}

    @Override

    public void displayDetails() {

        super.displayDetails();

        System.out.println("Has Gears: " + (hasGears ? "Yes" : "No"));

    }

}

class Truck extends Vehicle {

    double capacity;

    public Truck(String brand, int speed, double capacity) {

        super(brand, speed);

        this.capacity = capacity;

    }

    public void showTruckDetails() {

        displayDetails();

        System.out.println("Capacity: " + capacity + " tons");

    }

}

public class VehicleTest {

    public static void main(String[] args) {

        // Create car, bike, and truck objects

        Car car = new Car("Toyota", 150, 4);

        Bike bike = new Bike("Yamaha", 80, true);

        Truck truck = new Truck("Volvo", 90, 15.5);

        car.start();

        bike.start();

        truck.start();

        System.out.println("\nCar Details:");

        car.displayDetails();

        System.out.println("\nBike Details:");

        bike.displayDetails();

        System.out.println("\nTruck Details:");

        truck.showTruckDetails();

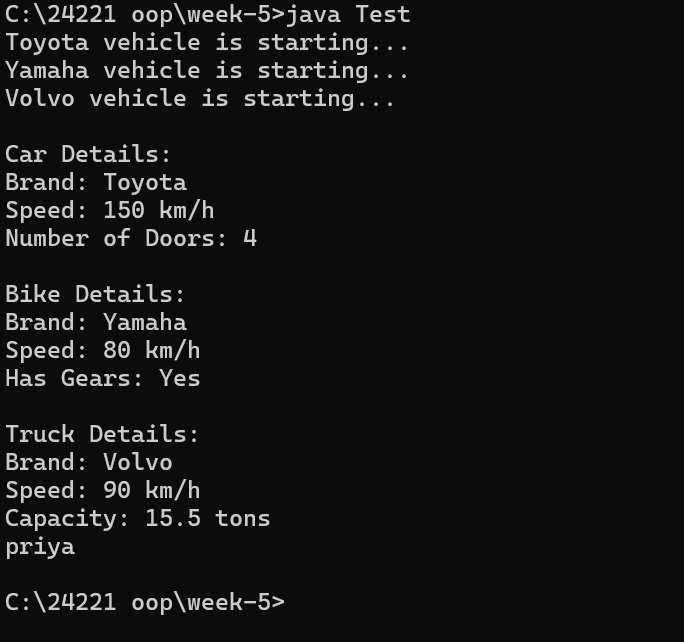
        System.out.println("priya");

    }

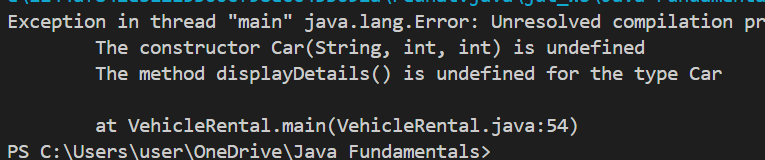
}

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Given wrong class name while object is created | Rectifying the given wrong class name |

**IMPORTANT POINTS:**

1. Class Inheritance:

* Vehicle Class: The Vehicle class serves as the base class. It contains common properties (like brand and speed) and methods (start() and displayDetails()) that are shared by all vehicle types.
* Car, Bike, and Truck Classes: These are subclasses that extend the Vehicle class. They inherit the properties and methods of the Vehicle class but can add their own specific properties (like numberOfDoors for Car, hasGears for Bike, and capacity for Truck).

2. Constructor Inheritance:

* The Car, Bike, and Truck classes use the super() keyword to invoke the constructor of the Vehicle class, ensuring the shared properties (brand and speed) are initialized properly. Each subclass has its own specific constructor to initialize its additional properties.

3. Method Overriding:

* displayDetails() Method: Each subclass (Car, Bike, Truck) overrides the displayDetails() method from the Vehicle class to add more specific details about the subclass. The super.displayDetails() is used to first display the general vehicle information, then additional subclass-specific details are printed.

**WEEK-6**

**PROGRAM-1:**

**AIM:** Write a java program to create a Vehicle class with a method displayInfo(). Override this method in the Car subclass to provide specific information about a car.

CLASS DIAGRAM:

**CODE:** class Vehicle {

    String brand;

    int speed;

    public Vehicle(String brand, int speed) {

        this.brand = brand;

        this.speed = speed;

    }

    public void displayInfo() {

        System.out.println("Vehicle Brand: " + brand);

        System.out.println("Speed: " + speed + " km/h");

    }

}

class Car extends Vehicle {

    int numberOfDoors;

    public Car(String brand, int speed, int numberOfDoors) {

        super(brand, speed);

        this.numberOfDoors = numberOfDoors;

    }

    @Override

    public void displayInfo() {

        super.displayInfo();

        System.out.println("Number Of Doors: " + numberOfDoors);

    }

}

public class Vehicleinfo{

    public static void main(String[] args) {

        Vehicle vehicle = new Vehicle("mahander", 100);

        System.out.println("Vehicle Details:");

        vehicle.displayInfo();

        System.out.println("\nCar Details:");

        Car car = new Car("Thar", 180, 5);

        car.displayInfo();

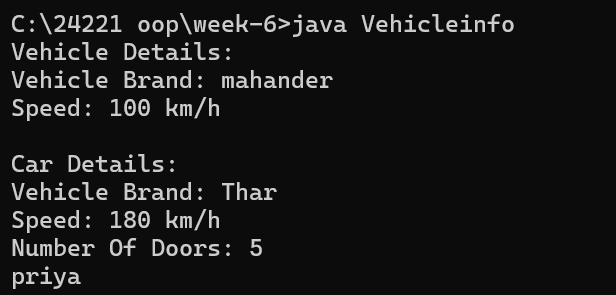
        System.out.println("priya");

    }

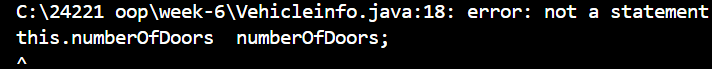
}

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | error: class Main is public, should be declared in a file named Main.java  public class Main { | Save the file name as the name of the main class |

**IMPORTANT POINTS:**

We use the concept of method overriding where the names of the methods in the different classes. The method of the parent class is overridden by the method of the child class

**PROGRAM-2:**

**AIM**: A college is developing an automated admission system that verifies students’ eligibility for under-graduation and post-graduation. Each program has different eligibility criteria base on the percentage of students in their provided qualifications

* Ug requires 60%
* PG requires 70%

|  |
| --- |
| UG |
| UG(String name ,double percentage) |

|  |
| --- |
| **Student** |
| + name: String  + percentage: double |
| + Student (String name, double percentage): void  + Eligibility (): void |

|  |
| --- |
| **PG** |
| PG (String name, double percentage) |

**CODE:** class Student {

  String name;

  double percentage;

  Student(String name, double percentage){

    this.name = name;

    this.percentage = percentage;

  public void Eligibility(){

    System.out.println(name + " must meet the general admission criteria");

  }

}

class UG extends Student{

  UG(String name, double percentage){

    super(name, percentage);

  }

    public void Eligibility(){

      if (percentage>59){

        System.out.println(name + " is elligible for UG admission");

      }

      else {

        System.out.println(name + " is Not elligible for UG admission");

      }

    }

}

class PG extends Student{

  PG(String name, double percentage){

    super(name, percentage);

  }

    public void Eligibility (){

      if (percentage>69){

        System.out.println(name + " is elligible for PG admission");

      }

      else {

        System.out.println(name + " is Not elligible for PG admission");

        System.out.println("priya");

      }

    }

}

public class Admission {

  public static void main (String[] args){

    UG ug = new UG("riya", 69);

    PG pg = new PG("Santhi", 59);

    ug.Eligibility();

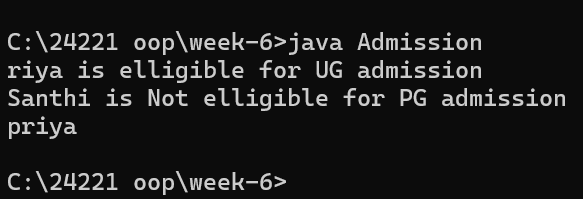
    pg.Eligibility();

  }

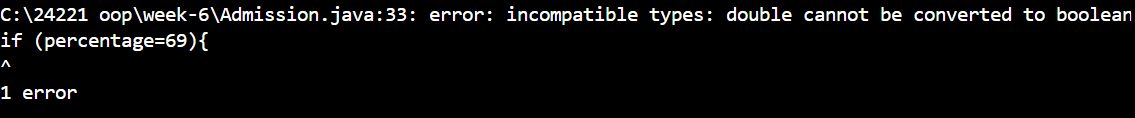
}

OUTPUT:

POSITIVE CASE:



**NEGATIVE CASE:**

****  
**ERRORS TABLE:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: ';' expected System.out.println(name + " is elligible for UG a | Add a ‘;’ after the print statement. |

**PROGRAM-3:  
AIM:** Create a calculator with overloaded methods to perform addition:

i) Add two integers

ii) Add two doubles

iii) Add three integers

**CLASS DIAGRAM:**

|  |
| --- |
| **AddCalculator** |
| + AddCalculator ()  + add (int a, int b): int  + add (double a, double b): double  + add (int a, int b, int c): int |

**CODE:** class AddCalculator {

    AddCalculator() {

        System.out.println("This is a calculator");

    }

    public int add(int a, int b) {

        return a + b;

    }

    public double add(double a, double b) {

        return a + b;

    }

    public int add(int a, int b, int c) {

        return a + b + c;

    }

}

public class AddCal {

    public static void main(String[] args) {

        AddCalculator calc = new AddCalculator();

        System.out.println("The sum of the 2 numbers: 3 and 9 is " + calc.add(4, 7));

        System.out.println("The sum of the 2 numbers: 3.3 and 9.9 is " + calc.add(4.4, 7.7));

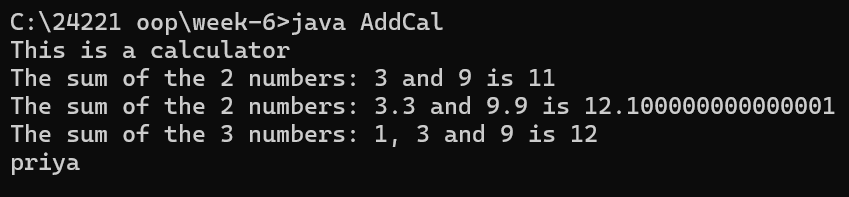
        System.out.println("The sum of the 3 numbers: 1, 3 and 9 is " + calc.add(1, 4, 7));

        System.out.println("priya");

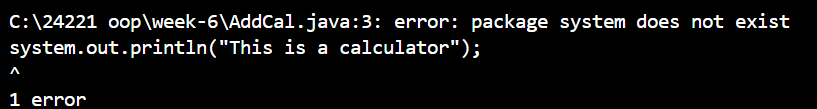
    }

}

**OUTPUT:**

**POSITIVE CASE:**

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | error: Main method not defined | Add public static void main (String [] args) { |
| 2. | error: ';' return type is not expected | Change return type from int to double in its case. |

**IMPORTANT POINTS:**

1. We use the concept of method overloading where the names of the methods in the same class are same but the parameters are given different.

**PROGRAM-4:**

**AIM:** Create a Shape class with a method calculateArea() that is overloaded for different shapes (e.g., square, rectangle).

Then create a subclass Circle that overrides the calculateArea() method for a circle.

**CLASS DIAGRAM:**

|  |
| --- |
| **Circle** |
| + r: double  + pi: double |
| + calcarea(int r): double |

|  |
| --- |
| **Shape** |
| + calcArea(int a): int  + calcArea(int b, int h): int |

**CODE:** class Shape {

    int calcArea(int a) {

        return a \* a;

    }

    int calcArea(int b, int h) {

        return b \* h;

    }

}

class Circle extends Shape {

    double r;

    double pi = 3.141;

    Circle(double r) {

        this.r = r;

    }

  double calcArea(double r) {

        return pi \* r \* r;

    }

}

public class AreaCal {

    public static void main(String[] args) {

        Circle c = new Circle(7);

        System.out.println("The area of circle is " + c.calcArea( 8.5));

        System.out.println("The area of Rectangle is " + c.calcArea( 4,  7));

        System.out.println("The area of Square is " + c.calcArea( 7));

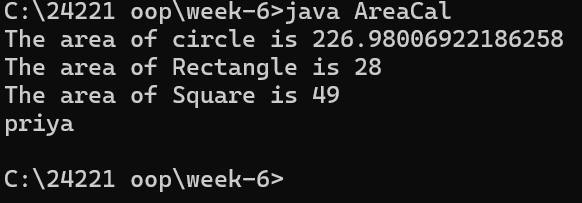
        System.out.println( "priya");

    }

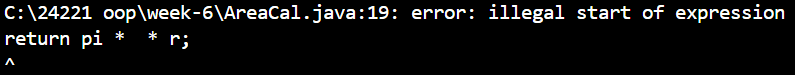
}

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | error: invalid method declaration; return type required  calcArea(int a) { | Enter the return type as per required. Here it is int |
| 2. | error: incompatible types: possible lossy conversion from double to int  return pi\*r\*r; | For calculating area of circle, we need to give return type double. |

**WEEK-7**

**PROGRAM-1:**

**AIM:** Write a Java program to create an abstract class Animal with an abstract method called sound (). Create subclasses Lion & Tiger that extend Animal class and implement the sound () method to make a specific sound for each animal.

**CLASS DIAGRAM:**

|  |
| --- |
| **Animal** |
| **+sound():void** |

|  |  |
| --- | --- |
| **Lion** | **Tiger** |
| **+sound():void** | **+sound():void** |

**CODE:** abstract class animals {

    abstract void sound();

}

class tiger extends animals {

    @Override

    public void sound() {

        System.out.println("tiger growls");

    }

}

class lion extends animals {

    @Override

    public void sound() {

        System.out.println("lion growls");

    }

}

public class Abstractani {

    public static void main(String[] args) {

        tiger tiger=new tiger();

        tiger.sound();

        lion lion=new lion();

        lion.sound();

        System.out.println("priya");

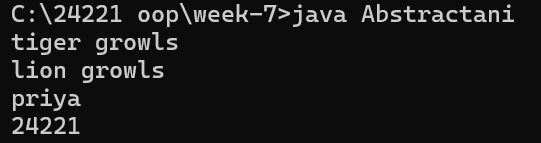
        System.out.println("24221");

    }

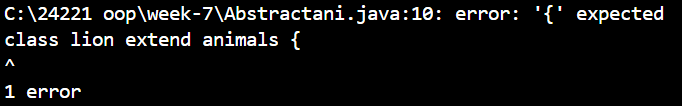
}

**OUTPUT:**

**POSTIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | Implicity declared class not have main method in the form of void main () | Change the public class name based on the abstract class name. |
| 2. | In the given code; expected | Apply the; |

**PROGRAM-2:**

**AIM:** Write a Java program using an abstract class to define a method for pattern printing.

Create an abstract class named PatternPrinter with an abstract method printPattern(int n).

Create concrete subclasses:

1. StarPattern - Prints a right-angled triangle of stars (\*).

2. NumberPattern - Prints a right-angled triangle of numbers

In the main () method, create objects of both subclasses and print the pattern for a given num.

**CLASS DIAGRAM:**

|  |
| --- |
| Patternprinter |
| -rows: int  +displayTitle()  +printPattern() |
| StarPattern  +printPattern()  NumberPattern  +printPattern() |

**CODE:** abstract class PatternPrinter {

    public int n;

    private String title;

    abstract void PrintPattern(int n);

    public void Patterntitle(String title) {

        System.out.println("\n" + title);

        System.out.println("------------------------");

    }

}

class StarPattern extends PatternPrinter {

    public void PrintPattern(int n) {

        this.n = n;

        for (int i = 0; i <= n; i++) {

            for (int j = 0; j <= i; j++) {

                System.out.print("\* ");

            }

            System.out.println(); // Move to the next line after inner loop

        }

    }

}

class NumberPattern extends PatternPrinter {

    public void PrintPattern(int n) {

        this.n = n;

        for (int i = 0; i <= n; i++) {

            for (int j = 0; j <= i; j++) {

                System.out.print(j + 1 + " ");

            }

            System.out.println(); // Move to the next line after inner loop

        }

    }

}

class Pattern {

    public static void main(String[] args) {

        StarPattern s = new StarPattern();

        s.Patterntitle("StarPattern");

        s.PrintPattern(5);

        NumberPattern num = new NumberPattern();

        num.Patterntitle("NumberPattern");

        num.PrintPattern(5);

        System.out.println("priyadarshini");

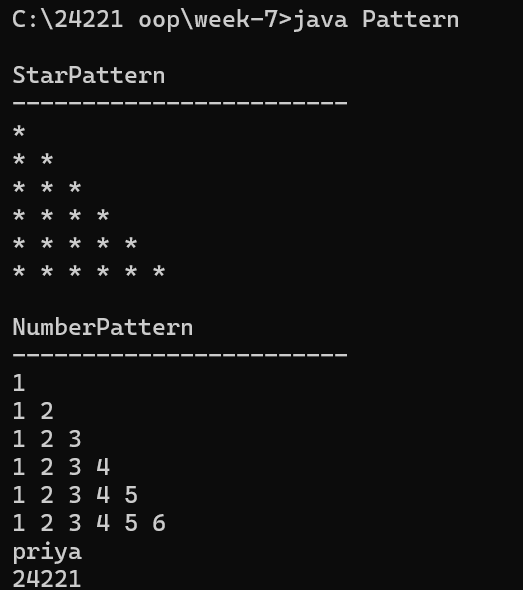
        System.out.println("AV.SC.U4CSE24221");

        System.out.println("CSE-C");

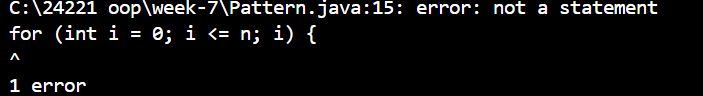
    }

}

**OUTPUT:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | Loop range for (int i=0;i<=n;i++) of the code. | The loop range for (int i=0;i<n;i++) of the rectification. |
| 2. | Method name patterntitle has casing the folder name. | Rename the patterntTitle to follow java folder naming. |
| 3. | Class naming of the pattern title format of the code. | Rename the class variable to patternDemo of the change the class variable. |

**IMPORTANTA POINTS:**

1. Abstract Class Usage:

* PatternPrinter is an abstract class, meaning it cannot be instantiated directly.
* It has an abstract method PrintPattern(int n) which must be implemented by all subclasses.

2. Method Overriding:

* Both StarPattern and NumberPattern classes override the PrintPattern(int n) method to display different patterns.

3. Encapsulation:

* title is a private variable, which is a good practice for encapsulation.
* However, since it’s unused in this code, it might be redundant unless you plan to use it later.

**PROGRAM-03:**

**AIM:**Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.

**CLASS DIAGRAM:**

|  |
| --- |
| Shape3D  +calculatevolume(): double |
|  |

**CODE:** abstract class Shape3d {

    abstract double calculateVolume();

    abstract double calculateSurfaceArea();

}

class Sphere extends Shape3d {

    double radius;

    public Sphere(double radius) {

        this.radius = radius;

    }

    @Override

    double calculateVolume() {

        return (4.0 / 3.0) \* Math.PI \* Math.pow(radius, 3);

    }

    @Override

    double calculateSurfaceArea() {

        return 4 \* Math.PI \* Math.pow(radius, 2);

    }

}

class Cube extends Shape3d {

    double side;

    public Cube(double side) {

        this.side = side;

    }

    @Override

    double calculateVolume() {

        return Math.pow(side, 3);

    }

    @Override

    double calculateSurfaceArea() {

        return 6 \* Math.pow(side, 2);

    }

}

public class shape3D {

    public static void main(String[] args) {

        Shape3d sphere = new Sphere(4.0);

        Shape3d cube = new Cube(2.0);

        System.out.println("Priya Darshini");

        System.out.println("AV.SC.U4CSE24221");

        System.out.println("CSE-C");

        System.out.println("Sphere:");

        System.out.println("Volume: " + sphere.calculateVolume());

        System.out.println("Surface Area: " + sphere.calculateSurfaceArea());

        System.out.println("Cube:");

        System.out.println("Volume: " + cube.calculateVolume());

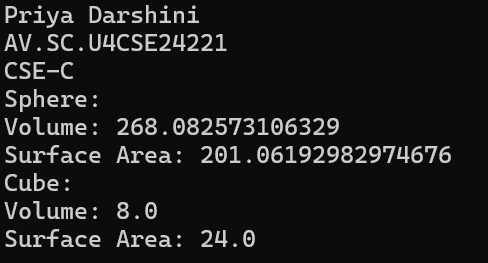
        System.out.println("Surface Area: " + cube.calculateSurfaceArea());

    }

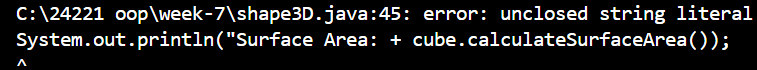
}

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | Missed the; code of the given code. | Corrected by placing; after every object. |
| 2. | } closing is missing | Corrected by the closing the code of the} |

**IMPORTANT POINTS:**

|  |  |
| --- | --- |
|  | 1.Shape3d is declared as an abstract class. It defines the structure (volume and surface area methods) but not the implementation, which must be provided by subclasses. |

|  |  |  |
| --- | --- | --- |
|  |  | 2.The methods calculateVolume() and calculateSurfaceArea() are abstract, enforcing that all 3D shapes must implement them. |

|  |  |  |
| --- | --- | --- |
|  |  | 3.Sphere and Cube extend Shape3d, meaningfrom the abstract class and provide concrete implementations for the methods. |

|  |  |  |
| --- | --- | --- |
|  |  | 4.The main() method uses polymorphism by declaring objects as Shape3d type but instantiating them as Sphere and Cube. |

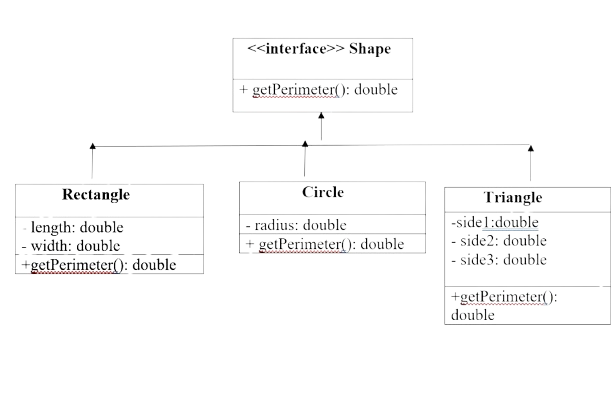
|  |  |
| --- | --- |
|  | 5.Class fields like radius and side are kept within their respective classes. (Could be improved by making them private with getters/setters). |

**WEEK 08**

**PROGRAM-01:**

**AIM:**

Write a Java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getPerimeter() method for each of the three classes.

**CLASS DIAGRAM:** 

**CODE:** interface Shape {

    double getPerimeter();

}

class Rectangle implements Shape {

    private double length;

    private double width;

    public Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }

    @Override

    public double getPerimeter() {

        return 2 \* (length + width);

    }

}

class Circle implements Shape {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public double getPerimeter() {

        return 2 \* Math.PI \* radius;

    }

}

class Triangle implements Shape {

    private double sideA;

    private double sideB;

    private double sideC;

    public Triangle(double sideA, double sideB, double sideC) {

        this.sideA = sideA;

        this.sideB = sideB;

        this.sideC = sideC;

    }

    @Override

    public double getPerimeter() {

        return sideA + sideB + sideC;

    }

}

public class perimeter {

    public static void main(String[] args) {

        Shape rectangle = new Rectangle(23, 4);

        Shape circle = new Circle(4);

        Shape triangle = new Triangle(23, 4, 7);

        System.out.println("priya darshini,AV.SC.U4CSE24221,CSE-C");

        System.out.println("Rectangle Perimeter: " + rectangle.getPerimeter());

        System.out.println("Circle Perimeter: " + circle.getPerimeter());

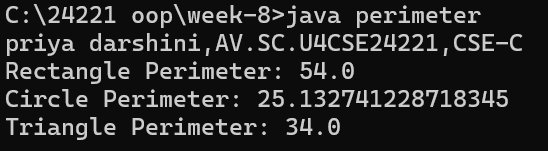
        System.out.println("Triangle Perimeter: " + triangle.getPerimeter());

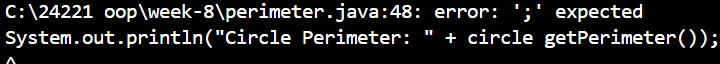
    }

}

**OUTPUT:**

**POSITIVE CASE:**

  
**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | System.out.println(“circleperimeter:”+circle get perimeter ()); syntax error | Corrected the syntax error of the code. |
| 2. | Renamed the folder while not use anywhere of the java programs. | Create the folder with new name of the java program. |

**IMPORTANT POINT:**

1. No Implementation: Interfaces cannot contain method implementations (in many languages), only method signatures.

2. Multiple Inheritance: A class can implement multiple interfaces, providing a way to achieve multiple inheritance.

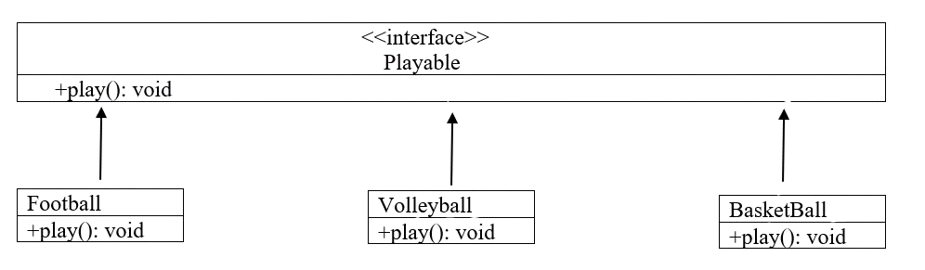
3. Abstract Methods: All methods in an interface are implicitly abstract (in many languages) and must be implemented by the implementing class.

4. No Constructors: Interfaces cannot have constructors because they cannot be instantiated directly.

**PROGRAM-02:**

**AIM:**Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.

**CLASS DIAGRAM**:



**CODE:** interface Playable {

    void play();

}

class Football implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Football: Kicking the ball towards the goal");

    }

}

class Volleyball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Volleyball: Bumping, setting, and spiking the ball");

    }

}

class Basketball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Basketball: Dribbling and shooting the ball");

    }

}

public class Testsport {

    public static void main(String[] args) {

      System.out.println("M.Priya darshini,AV.SC.U4CSE24221,CSE-C");

        Playable football = new Football();

        Playable volleyball = new Volleyball();

        Playable basketball = new Basketball();

        football.play();

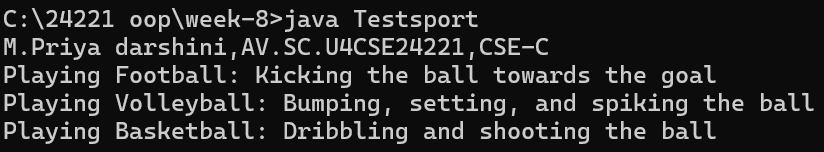
        volleyball.play();

        basketball.play();

    }

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Error** | **Error rectification** |
| 1. | Volley ball=new volley ball(); | VolleyBall v1=new volley ball. |

**IMPORTANT POINTS:**

* The program demonstrates the use of interfaces in Java through the Playable interface, which defines a single method play() that must be implemented by any class that uses it.
* Three different sports classes (FootBall, VolleyBall, and BasketBall) implement the Playable interface, each providing its own version of the play() method, showcasing polymorphism.

**PROGRAM-3:**

**AIM:** Write a java program to implements login System using interfaces.

**CLASS DIAGRAM:**

|  |
| --- |
| interface: Login system |
| + login(String,String): Boolean |

|  |
| --- |
| University\_portal |
| + login(StringString): boolean  + main(String[]): void |

**CODE:**interface LoginSystem {

    boolean login(String id, String pass);

}

class University\_portal implements LoginSystem {

    @Override

    public boolean login(String id, String pass) {

        if (id.equals("Student123") && pass.equals("pass02")) {

            System.out.println("Login successful");

            return true;

        } else {

            System.out.println("Invalid credentials");

            return false;

        }

    }

    public static void main(String[] args) {

        System.out.println("Name:priya,CSE24221");

        University\_portal p1 = new University\_portal();

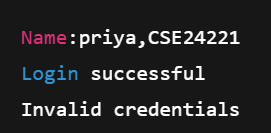
        p1.login("Student123", "pass02");

        p1.login("Student123", "wrongpass");

    }

}

**OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code Rectification |
| 1.Saving file with the interface name is error  2.Removing public will leads to error | 1.save with implement name.  2.add public static void main  (String[]args) |

**IMPORTANT POINTS:**

**WEEK-9**

**PROGRAM-1:**

**AIM:** write a java program to create a method that takes an integer as parameter and throws an exception of the number is even

**CLASS DIAGRAM:**

|  |
| --- |
| EvenNumberExceptionDemo |
| + checkOddNumber(int number) : void  + main(String[] args) : void |

|  |
| --- |
| EvenNumberException  (extends Exception) |
| + EvenNumberException(String msg) |

**CODE:** public class ExceptionNo {

    static class EvenNoException extends Exception{

        public EvenNoException(String message) {

            super(message);

        }

    }

    public static void checkoddno(int no) throws EvenNoException {

        if (no % 2 == 0) {

            throw new EvenNoException("the number " + no + " is even.");

        } else {

            System.out.println("the number" + no +" is odd.");

            }

        }

        public static void main(String[] args) {

            int testnumber = 4;

            try {

              checkoddno(testnumber);

            } catch (EvenNoException e) {

                System.out.println("exception caught:"+ e.getMessage());

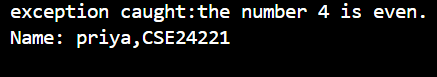
                System.out.println("Name: priya,CSE24221");

            }

   }    }

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code Rectification** |
| 1.Unhandled exception type EvenNumberException  2. Syntax Error  3. Compilation Error | 1. If you call checkOddNumber() without using try-catch or without declaring throws.  2.If missing curly braces {} or wrong method syntax  3.If constructor of EvenNumberException is missing or incorrectly defined. |

**IMPORTANT POINTS:**

1) Created a **custom exception** by extending the Exception class.

2) Used throw keyword to manually throw the custom exception if the number is even.

3)Handled the exception using a **try-catch** block inside main() method.

4) Demonstrates **user-defined exception** handling.

5) Shows clear separation of concerns: checking number and exception message.

**PROGRAM-2:**

**AIM:** write a java program to create a method that takes an integer as parameter and throws an exception of the number is even

**CLASS DIAGRAM:**

|  |
| --- |
| FileReadExample |
| + main(String[] args) : void |

|  |
| --- |
| Uses: |
| - BufferedReader  - FileReader  - FileNotFoundException  - IOException |

**CODE:**

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

public class Filereader {

    public static void main(String[] args) {

        try {

            // Specify the path (make sure "D:\priya" folder already exists)

            String filePath = "D:\\priya\\myfile.txt";

            File file = new File(filePath);

            // Create the file if it doesn't exist

            if (file.createNewFile()) {

                System.out.println("File created: " + file.getAbsolutePath());

            } else {

                System.out.println("File already exists.");

            }

            // Write to the file

            FileWriter writer = new FileWriter(file);

            writer.write("Hello, this is a file created by Java.\n");

            writer.write("Name: Priya Darshini\n");

            writer.write("Roll.no: AV.SC.U4CSE24221\n");

            writer.close();

            System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {

            System.out.println("An error occurred while creating/writing the file.");

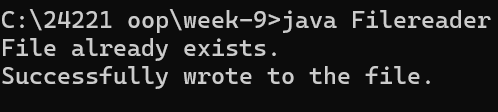
            e.printStackTrace();

        }

    }

}

**OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code Rectification** |
| 1.FileNotFoundException  2.IOException  3.Syntax Error | 1.Occurs if the specified file path is wrong or file does not exist.  2.Occurs while reading file if an input/output error happens.  3.If missing semicolon ;, wrong try-catch block syntax. |

**IMPORTANT POINTS:**

1. Used BufferedReader and FileReader to read text files.

2. FileNotFoundException occurs if the file is missing.

3. IOException occurs for input/output errors during file reading.

4. try-catch block is used for proper exception handling.

5. Always close the reader after reading the file (reader.close()).

**PROGRAM-3:**

**AIM:** write a java program to handle an arthematic exception using try catch finally

**CLASS DIAGRAM:**

|  |
| --- |
| ArithmeticExceptionDemo |
| + main(String[] args) : void |

|  |
| --- |
| Uses: |
| - try-catch-finally block  - ArithmeticException |

**CODE:** public class ArithmeticExceptionDemo {

    public static void main(String[] args) {

        int numerator = 30;

        int denominator = 0;

        try {

            int result = numerator / denominator;

            System.out.println("Result: " + result);

        } catch (ArithmeticException e) {

            System.out.println("ArithmeticException caught: " + e.getMessage());

        } finally {

            System.out.println("Finally block executed.");

        }

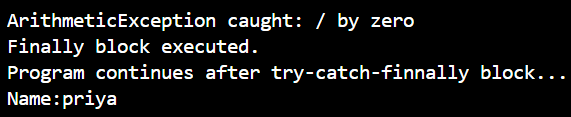
        System.out.println("Program continues after try-catch-finnally block... ");

        System.out.println("Name:priya");

    }

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code Rectification** |
| 1.ArithmeticException  2.Syntax Error  3.Compilation Error | 1.Division by zero operation  2.Missing try-catch or braces  3.Catch block written incorrectly |

**IMPORTANT POINTS:**

1.Division by zero causes ArithmeticException.

2.finally block always executes.

3.Safe program termination even after exceptions.

**PROGRAM-4:**

**AIM:** Write a Java program to simulate a University system using inner classes.

1) Create an outer class named University with a variable universityName. Inside it, define two non-static inner classes:

2) Department with variables like deptName and deptCode, and a method to display department details.

3) Student with variables like studentName and rollNumber, and a method to display student details.

4) Create an object for each class, their methods to display their details along with the university name.

**CLASS DIAGRAM:**

|  |
| --- |
| **Inner Classes:**  - Department  - deptName: String  - deptCode: String  + displayDepartment(): void |
| **-** Student  - studentName: String  **-** rollNumber: int  + displayStudent(): void |

|  |
| --- |
| University |
| -universityName: String  + University (String name) |

|  |
| --- |
| + main(String[] args): void |

**CODE:** public class University {

    String universityName;

    University(String universityName) {

        this.universityName = universityName;

    }

    class Department {

        String deptName;

        String deptCode;

        Department(String deptName, String deptCode) {

            this.deptName = deptName;

            this.deptCode = deptCode;

        }

        void displayDepartmentDetails() {

            System.out.println("University Name: " + universityName);

            System.out.println("Department Name: " + deptName);

            System.out.println("Department Code: " + deptCode);

        }

    }

    class Student {

        String studentName;

        int rollNumber;

        Student(String studentName, int rollNumber) {

            this.studentName = studentName;

            this.rollNumber = rollNumber;

        }

        void displayStudentDetails() {

            System.out.println("University Name: " + universityName);

            System.out.println("Student Name: " + studentName);

            System.out.println("Roll Number: " + rollNumber);

        }

    }

    public static void main(String[] args) {

        University university = new University("AMRITA VISWA VIDYAPEETHAM");

        University.Department department = university.new Department("Computer Science", "CSE01");

        department.displayDepartmentDetails();

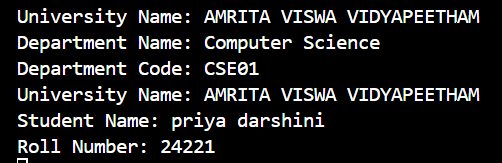
        University.Student student = university.new Student("priya darshini", 24221);

        student.displayStudentDetails();

    }

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code Rectification** |
| 1.Syntax Error  2.Compilation Error  3.Runtime Error | 1.Wrong object creation for inner class  2.Accessing outer class members wrongly  3.NullPointerException if outer object missing |

**IMPORTANT POINTS:**

1.Demonstrates inner class usage.

2.Inner classes access outer class members easily.

3.Separate objects for Department and Student.

4.Good example of encapsulation

**WEEK – 10**

**PROGRAM-1:**

**AIM:** Write a Java Program to generate a password for a student using his/her initials and age. The password displayed should be the string consists of first character of first name, middle name and last name with age.

**CLASS DIAGRAM:**

|  |
| --- |
| **PasswordMaker** |
| + firstName : String  + middleName : String  + lastName : String  + age : int |

**CODE:** import java.util.Scanner;

public class StudPwdGenerator {

    public static void main(String[] args) {

        System.out.println("priya darshini");

        System.out.println("242221");

        System.out.println("CSE-C");

        Scanner scanner = new Scanner(System.in);

        System.out.println("enter the First name: ");

        String firstName = scanner.nextLine();

        System.out.println("enter the Second name: ");

        String middleName = scanner.nextLine();

        System.out.println("enter the last name: ");

        String lastName = scanner.nextLine();

        System.out.println("enter Age: ");

        int age = scanner.nextInt();

        char fInitial = firstName.charAt(0);

        char mInitial = middleName.charAt(0);

        char lInitial = lastName.charAt(0);

        String password = "" + fInitial + mInitial + lInitial + age;

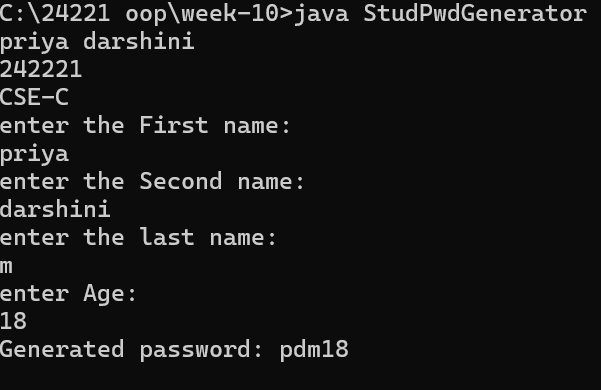
        System.out.println("Generated password: "+ password);

        scanner.close();

    }

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Error rectification** |
| 1. cannot find symbol  String initials = firstName(0,1) + middleName(0,1) + lastName(0,1); | 1.Write String initials = firstName.substring(0,1) + middleName.substring(0,1) + lastName.substring(0,1); |

**IMPORTANT POINTS:**

1. We should use substring keyword to access a part of the string

2. We can also add the substrings with the ‘+’ operator.

**PROGRAM-2:  
AIM:** Design and implement a Java program that will do the following operations to the string " Welcome ! You are practicing String concept "

1. convert all the alphabets to Upper Case and print the result.
2. convert all the alphabets to Lower Case and print the result.
3. print the length of the string.
4. print the index of concept.

**CLASS DIAGRAM:**

|  |
| --- |
| **StringsOp** |
| + s : String |
| + toUpperCase (): void  + toLowerCase (): void  + length (): void  + indexOf (): String |

**CODE:** public class Stringops {

    public static void main(String[] args) {

        System.out.println("Name: priya,cse24221");

        String s = "Welcome! You are practicing String concept ";

        System.out.println(s.toUpperCase());

        System.out.println(s.toLowerCase());

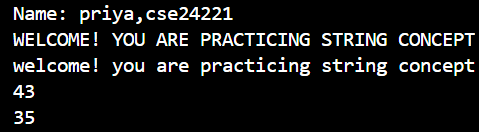
        System.out.println(s.length());

        System.out.println(s.indexOf("concept"));

    }

}

**OUTPUT:**



NEGATIVE CASE:

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Error** | **Error rectification** |
| 1. ';' expected String s = "Welcome! You are practicing String concept "  2.Inaccurate output | 1.Add a semicolon at the end  2.Write “concept” instead of “Concept” |

**IMPORTANT POINTS:**

1. We should import the String class to work with the built-in methods of Strings.

2. We can find the length and also covert a String to Upper case or lower case as per our requirement with the help of the methods.

3. to find the index of a String is Case-Sensitive.

**PROGRAM-3**

**AIM:** Implement a Java program using the below array methods

1. Sorting the elements (Numbers and strings) of an array.
2. Convert the array elements to String.
3. Fill the part of a array.
4. Copy the elements of an array to another
5. 5)cop

**CLASS DIAGRAM:**

|  |
| --- |
| **Array** |
| - y: int[]  - copy: int[]  - numbers: int[] |

**CODE:**import java.util.Arrays;

public class Arraysmeth {

    public static void main(String[] args) {

        System.out.println("Name:priya,CSE24221");

        int[] y = {12, 6, 4, 8};

        Arrays.sort(y);

        for (int i : y) {

            System.out.print(i + ", ");

        }

        System.out.println("\n" + Arrays.toString(y));

        int[] copy = Arrays.copyOf(y, y.length);

        System.out.println(Arrays.toString(copy));

        int[] numbers = new int[4];

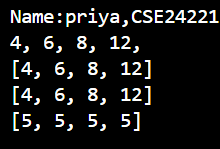
        Arrays.fill(numbers, 5);

        System.out.println(Arrays.toString(numbers));

    }

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Error rectification** |
| 1. illegal start of expression  int[] y = [12, 6, 8, 2]; | 1.Make the use of curly braces ‘{}’ instead of square braces ‘[]’. |
| 2.cannot find symbol  int[] copy = Arrays.copyOf(y,y.length()); | 2.Do not use ‘()’ after length since it is used only for Strings not Arrays. |

**IMPORTANT POINTS:**

1. Arrays are declared in curly braces only

2. they can handle only one type of data type per array. Can e strings, int or floats.

3. We should import the Arrays class to work with the built-in methods of Arrays.

**PROGRAM-4:**

**AIM:** Implement a Java program using the below arrayList methods

1. Modify an element in the arrayList.
2. Access an element from the arrayList.
3. Remove an element from the arrayList.
4. Clear the elements from the arrayList.

**CLASS DIAGRAM:**

|  |
| --- |
| **ArrayList** |
| - list: ArrayList<String> |
| + modifyElement(): void  + accessElement(): void  + removeElement(): void  + clearElements(): void |

**CODE:** import java.util.ArrayList;

import java.util.Arrays;

public class ArrayListEx {

    public static void main(String[] args) {

        System.out.println("Name:Priya,CSE24221");

        ArrayList<String> list = new ArrayList<>(Arrays.asList("Apple", "Mango", "Dragon Fruit"));

        System.out.println("Original ArrayList: " + list);

        list.set(1, "Blueberry");

        System.out.println("After modification: " + list);

        String element = list.get(2);

        System.out.println("Element at index 2: " + element);

        list.remove(0);

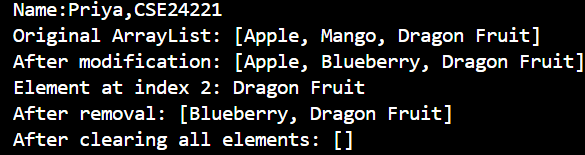
        System.out.println("After removal: " + list);

        list.clear();

        System.out.println("After clearing all elements: " + list);

    }

}

**OUTPUT:** 

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Error rectification** |
| 1.cannot find symbol  ArrayList<String> list = new ArrayList<>(List.of("Apple", "Banana", "Cherry"));  2.Same error | 1.Use Array.asList instead of List.of.  2.Import ArrayList and Arrays both. Importing only one will not work |

**IMPORTANT POINTS:**

1. We should follow the syntax of ArrayList while declaring it.

2. We should import the Arrays and ArrayLists classes to work with the built-in methods of ArrayLists