1.Write a program to create a room class, the attributes of this class is room no,

room type, room area and AC machine. In this class the member functions are set

data and display data. 👇👇👇👇👇👇👇👇👇👇

Certainly! Here's a simple Java program for a `Room` class with the specified attributes and member functions:

```java

public class Room {

// Attributes

private int roomNumber;

private String roomType;

private double roomArea;

private boolean hasAC;

// Member functions: set data

public void setData(int roomNumber, String roomType, double roomArea, boolean hasAC) {

this.roomNumber = roomNumber;

this.roomType = roomType;

this.roomArea = roomArea;

this.hasAC = hasAC;

}

// Member function: display data

public void displayData() {

System.out.println("Room Number: " + roomNumber);

System.out.println("Room Type: " + roomType);

System.out.println("Room Area: " + roomArea + " square meters");

System.out.println("AC Machine: " + (hasAC ? "Yes" : "No"));

}

public static void main(String[] args) {

// Example usage

Room myRoom = new Room();

myRoom.setData(101, "Single", 25.5, true);

myRoom.displayData();

}

}

```

In this example, the `Room` class has private attributes (`roomNumber`, `roomType`, `roomArea`, and `hasAC`) and two member functions (`setData` and `displayData`). The `setData` function is used to set the values of the attributes, and `displayData` is used to display the details of the room. The `main` method demonstrates how to create an instance of the `Room` class, set data, and display the information.

2.a) Write a program to demonstrate static variables, methods, and blocks👇👇👇👇👇

Certainly! Here's a simple Java program demonstrating static variables, methods, and blocks:

```java

public class StaticDemo {

// Static variable

static int staticVariable = 10;

// Static block

static {

System.out.println("This is a static block. Static variable value: " + staticVariable);

}

// Static method

static void staticMethod() {

System.out.println("This is a static method.");

}

public static void main(String[] args) {

// Accessing static variable

System.out.println("Accessing static variable: " + staticVariable);

// Calling static method

staticMethod();

}

}

```

In this example, `staticVariable` is a static variable, the static block is used to initialize and perform actions when the class is loaded, and `staticMethod()` is a static method. The `main` method demonstrates accessing the static variable and calling the static method.

2.b) Write a program for reuse class. For this program use the above ‘room class’

program👇👇👇👇

Certainly! If you have a `Room` class and want to demonstrate reuse in Java, you can create another class that utilizes the `Room` class. Here's a simple example:

```java

// Room class

class Room {

private int length;

private int width;

public Room(int length, int width) {

this.length = length;

this.width = width;

}

public int calculateArea() {

return length \* width;

}

}

// Reuse class

class ReuseExample {

public static void main(String[] args) {

// Creating an instance of the Room class

Room myRoom = new Room(10, 8);

// Calculating and displaying the area of the room

int area = myRoom.calculateArea();

System.out.println("The area of the room is: " + area + " square units");

}

}

```

In this example, the `Room` class is designed to represent a room with length and width attributes. The `ReuseExample` class then creates an instance of the `Room` class and demonstrates the reuse of its functionality by calculating and displaying the area of the room.

3. a) Write a program to give the example for method overriding concepts.👇👇👇👇👇

Certainly! Here's a simple example in Java to illustrate method overriding:

```java

class Animal {

void makeSound() {

System.out.println("Some generic sound");

}

}

class Dog extends Animal {

// Overriding the makeSound method in the subclass

@Override

void makeSound() {

System.out.println("Woof! Woof!");

}

}

public class Main {

public static void main(String[] args) {

Animal genericAnimal = new Animal();

Animal dog = new Dog();

// Calls the makeSound method of the Animal class

genericAnimal.makeSound();

// Calls the overridden makeSound method of the Dog class

dog.makeSound();

}

}

```

In this example, the `Dog` class extends the `Animal` class and overrides the `makeSound` method. When you create a `Dog` object and call the `makeSound` method on it, it will execute the overridden method in the `Dog` class rather than the one in the parent `Animal` class.

3.b Write a program to give the example for ‘super’ keyword👇👇👇👇👇

Certainly! Here's a simple example in Java that demonstrates the use of the `super` keyword:

```java

class Animal {

String sound = "Animal Sound";

void makeSound() {

System.out.println(sound);

}

}

class Dog extends Animal {

String sound = "Bark";

void makeDogSound() {

System.out.println(sound);

}

void displayAnimalSound() {

super.makeSound(); // using 'super' to call the method from the superclass

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.makeDogSound(); // Outputs: Bark

myDog.displayAnimalSound(); // Outputs: Animal Sound

}

}

```

In this example, the `Dog` class extends the `Animal` class. The `Dog` class has its own `sound` variable but uses the `super` keyword to access the `makeSound` method from the superclass (`Animal`).

4.Write a program to create a class named shape. In this class we have three sub

classes circle, triangle and square each class has two member function named

draw () and erase (). Create these using polymorphism concepts.👇👇👇👇

Certainly! Here's a simple Java program that demonstrates polymorphism with a base class `Shape` and three subclasses `Circle`, `Triangle`, and `Square`:

```java

class Shape {

void draw() {

System.out.println("Drawing a shape");

}

void erase() {

System.out.println("Erasing a shape");

}

}

class Circle extends Shape {

@Override

void draw() {

System.out.println("Drawing a circle");

}

@Override

void erase() {

System.out.println("Erasing a circle");

}

}

class Triangle extends Shape {

@Override

void draw() {

System.out.println("Drawing a triangle");

}

@Override

void erase() {

System.out.println("Erasing a triangle");

}

}

class Square extends Shape {

@Override

void draw() {

System.out.println("Drawing a square");

}

@Override

void erase() {

System.out.println("Erasing a square");

}

}

public class Main {

public static void main(String[] args) {

Shape circle = new Circle();

Shape triangle = new Triangle();

Shape square = new Square();

// Polymorphism in action

displayShapeDetails(circle);

displayShapeDetails(triangle);

displayShapeDetails(square);

}

private static void displayShapeDetails(Shape shape) {

shape.draw();

shape.erase();

System.out.println();

}

}

```

This program defines a base class `Shape` with `draw()` and `erase()` methods, and three subclasses (`Circle`, `Triangle`, and `Square`) that override these methods. The `Main` class demonstrates polymorphism by creating instances of the base class and calling methods on them, which are dynamically bound to the appropriate overridden methods in the subclasses.

5. Create an outer class with a function display, again create another class inside the

outer class named inner with a function called display and call the two functions

in the main class.👇👇👇👇👇

Certainly! Here's a simple example in Java:

```java

public class OuterClass {

void display() {

System.out.println("OuterClass display");

}

class Inner {

void display() {

System.out.println("Inner display");

}

}

public static void main(String[] args) {

OuterClass outerObject = new OuterClass();

outerObject.display();

OuterClass.Inner innerObject = outerObject.new Inner();

innerObject.display();

}

}

```

In this example, `OuterClass` has a method `display`. Inside `OuterClass`, there is an inner class `Inner` with its own `display` method. In the `main` method, an instance of `OuterClass` is created, and its `display` method is called. Then, an instance of the inner class `Inner` is created using the outer object (`outerObject.new Inner()`), and its `display` method is called.

6.Create class box and box3d. box3d is extended class of box. The above two classes

going to pull fill following requirement.

•Include constructor.

•set value of length, breadth, height

•Find out area and volume.

Note: Base class and sub classes have respective methods and instance variables👇👇

Certainly! Here's a simple implementation of the `Box` and `Box3D` classes in Java:

```java

class Box {

protected double length;

protected double breadth;

// Constructor for Box class

public Box(double length, double breadth) {

this.length = length;

this.breadth = breadth;

}

// Method to set values of length and breadth

public void setDimensions(double length, double breadth) {

this.length = length;

this.breadth = breadth;

}

// Method to find the area of the box

public double calculateArea() {

return length \* breadth;

}

}

class Box3D extends Box {

protected double height;

// Constructor for Box3D class

public Box3D(double length, double breadth, double height) {

super(length, breadth);

this.height = height;

}

// Method to set values of length, breadth, and height

public void setDimensions(double length, double breadth, double height) {

super.setDimensions(length, breadth);

this.height = height;

}

// Method to find the volume of the box

public double calculateVolume() {

return length \* breadth \* height;

}

}

public class Main {

public static void main(String[] args) {

// Example usage

Box3D myBox = new Box3D(2.0, 3.0, 4.0);

// Set dimensions

myBox.setDimensions(3.0, 4.0, 5.0);

// Calculate and print area

double area = myBox.calculateArea();

System.out.println("Area: " + area);

// Calculate and print volume

double volume = myBox.calculateVolume();

System.out.println("Volume: " + volume);

}

}

```

This code defines a `Box` class with a constructor and methods for setting dimensions and calculating area. The `Box3D` class extends `Box` and adds a height dimension along with its own constructor and methods to set dimensions and calculate volume. The `main` method demonstrates how to create an instance of `Box3D` and use its methods.

7.Write a program for example of multiple catch statements occurring in a program. 👇👇👇👇

Certainly! Here's an example Java program with multiple catch statements:

```java

public class MultipleCatchExample {

public static void main(String[] args) {

try {

// Some code that may cause different types of exceptions

int[] numbers = {1, 2, 3};

System.out.println(numbers[4]); // Accessing an index out of bounds

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index out of bounds exception: " + e.getMessage());

} catch (ArithmeticException e) {

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

} catch (Exception e) {

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

}

}

}

```

In this example, the program attempts to access an index that is out of bounds in an array, which can result in an `ArrayIndexOutOfBoundsException`. The program catches this exception first, then catches an `ArithmeticException` (even though it's not likely to occur in this specific code), and finally, a more general `Exception` catch block is added to handle any other unexpected exceptions. Each catch block has its own handling code.

8.Write a program to illustrate usage of try/catch with finally clause👇👇👇

Certainly! Below is a simple Java program that demonstrates the usage of try/catch with the finally clause:

```java

public class TryCatchFinallyExample {

public static void main(String[] args) {

try {

// Code that may throw an exception

int result = divide(10, 0);

System.out.println("Result: " + result); // This line won't be reached due to exception

} catch (ArithmeticException e) {

// Catching the specific exception

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

} finally {

// Code in the finally block will be executed regardless of whether an exception occurs or not

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

}

}

// A method that may throw an exception

public static int divide(int numerator, int denominator) {

return numerator / denominator;

}

}

```

In this example, the `divide` method attempts to perform a division operation that may result in an `ArithmeticException`. The `try` block contains the code that may throw an exception, the `catch` block catches the specific exception (if it occurs), and the `finally` block contains code that will be executed whether an exception occurs or not.

9.Write a program for creation of user defined exception.👇👇👇

Certainly! Here's an example Java program demonstrating the creation of a user-defined exception:

```java

// Custom exception class

class CustomException extends Exception {

public CustomException(String message) {

super(message);

}

}

// Main class

public class UserDefinedExceptionExample {

// Method that may throw the custom exception

static void validateAge(int age) throws CustomException {

if (age < 18) {

throw new CustomException("Age must be 18 or older.");

} else {

System.out.println("Valid age: " + age);

}

}

// Main method to test the user-defined exception

public static void main(String[] args) {

try {

// Test with an age that triggers the custom exception

validateAge(15);

} catch (CustomException e) {

System.out.println("Caught CustomException: " + e.getMessage());

}

}

}

```

In this example, the `CustomException` class extends the built-in `Exception` class, creating a custom exception. The `validateAge` method checks if the provided age is less than 18 and throws the custom exception if the condition is met. The `main` method demonstrates how to use this custom exception by calling `validateAge` with an age that triggers the exception and catching the exception to handle it

10.Write a program to rename the given file, after renaming the file delete the

renamed file. (Accept the file name using command line arguments.)👇👇👇

Certainly! Below is a simple Java program that takes a file name as a command line argument, renames the file, and then deletes the renamed file:

```java

import java.io.File;

public class FileRenameAndDelete {

public static void main(String[] args) {

// Check if a file name is provided as a command line argument

if (args.length != 1) {

System.out.println("Usage: java FileRenameAndDelete <file\_name>");

return;

}

String fileName = args[0];

File originalFile = new File(fileName);

// Check if the file exists

if (!originalFile.exists()) {

System.out.println("File not found: " + fileName);

return;

}

// Rename the file by adding "\_renamed" to the original file name

String renamedFileName = fileName.replaceFirst("\\.([^.]\*)$", "\_renamed.$1");

File renamedFile = new File(renamedFileName);

if (originalFile.renameTo(renamedFile)) {

System.out.println("File renamed to: " + renamedFileName);

// Delete the renamed file

if (renamedFile.delete()) {

System.out.println("Renamed file deleted successfully.");

} else {

System.out.println("Failed to delete the renamed file.");

}

} else {

System.out.println("Failed to rename the file.");

}

}

}

```

Save this code in a file named `FileRenameAndDelete.java` and compile it using `javac FileRenameAndDelete.java`. Then you can run the program with the desired file name as a command line argument, like this:

```

java FileRenameAndDelete your\_file.txt

```

Replace `your\_file.txt` with the actual file name you want to rename and delete.

11. Write a program to create a file and write data into it using the methods

OutputStream class.👇👇👇

Certainly! Below is a simple Java program that creates a file and writes data into it using the `OutputStream` class:

```java

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStream;

public class FileWriteExample {

public static void main(String[] args) {

String fileName = "example.txt";

String data = "Hello, writing data to a file using OutputStream.";

try (OutputStream outputStream = new FileOutputStream(fileName)) {

byte[] bytes = data.getBytes();

outputStream.write(bytes);

System.out.println("Data has been written to the file successfully.");

} catch (IOException e) {

System.err.println("Error writing to the file: " + e.getMessage());

}

}

}

```

This program uses a `FileOutputStream` to create a file named "example.txt" and writes the string "Hello, writing data to a file using OutputStream." into it. The `try-with-resources` statement is used to automatically close the `OutputStream` after the data has been written or if an exception occurs.

12.Write a program to get the input from the user and store it into file. Using Reader

and Writer file👇👇

Certainly! Below is a simple Java program that uses `BufferedReader` to get input from the user and `BufferedWriter` to store it into a file.

```java

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

public class UserInputToFile {

public static void main(String[] args) {

try {

// Create BufferedReader to read user input

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter text to be stored in the file: ");

String userInput = reader.readLine();

// Specify the file path

String filePath = "output.txt";

// Create BufferedWriter to write to the file

BufferedWriter writer = new BufferedWriter(new FileWriter(filePath));

// Write user input to the file

writer.write(userInput);

// Close the writer

writer.close();

System.out.println("Data successfully written to the file.");

} catch (IOException e) {

System.err.println("Error: " + e.getMessage());

}

}

}

```

This program prompts the user to enter text, reads the input, and then writes it to a file named "output.txt". Note that you might need to handle exceptions and close resources properly in a real-world scenario.

13. Develop a java application with Employee class with Emp\_name, Emp\_id,

Address, Mail\_id, Mobile\_no as members. Inherit the classes, Programmer,

Assistant Professor, Associate Professor and Professor from employee class. Add

Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10

% of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay

slips for the employees with their gross and net salary👇👇

Certainly! Below is a simple Java program that fulfills your requirements. It uses inheritance and encapsulation to create an Employee class and its subclasses with the specified members and calculations for gross and net salary:

```java

import java.util.Scanner;

class Employee {

String Emp\_name, Emp\_id, Address, Mail\_id, Mobile\_no;

// Constructor for Employee class

public Employee(String emp\_name, String emp\_id, String address, String mail\_id, String mobile\_no) {

Emp\_name = emp\_name;

Emp\_id = emp\_id;

Address = address;

Mail\_id = mail\_id;

Mobile\_no = mobile\_no;

}

// Display employee details

public void displayDetails() {

System.out.println("Employee ID: " + Emp\_id);

System.out.println("Employee Name: " + Emp\_name);

System.out.println("Address: " + Address);

System.out.println("Mail ID: " + Mail\_id);

System.out.println("Mobile Number: " + Mobile\_no);

}

}

class Programmer extends Employee {

double basicPay;

// Constructor for Programmer class

public Programmer(String emp\_name, String emp\_id, String address, String mail\_id, String mobile\_no, double basicPay) {

super(emp\_name, emp\_id, address, mail\_id, mobile\_no);

this.basicPay = basicPay;

}

// Calculate gross salary

public double calculateGrossSalary() {

return basicPay + (0.97 \* basicPay) + (0.1 \* basicPay) + (0.12 \* basicPay);

}

// Calculate net salary

public double calculateNetSalary() {

return calculateGrossSalary() - (0.12 \* basicPay);

}

}

// Similar classes for Assistant Professor, Associate Professor, and Professor can be created

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Example usage for Programmer class

System.out.println("Enter Programmer details:");

System.out.print("Employee Name: ");

String name = scanner.nextLine();

System.out.print("Employee ID: ");

String id = scanner.nextLine();

System.out.print("Address: ");

String address = scanner.nextLine();

System.out.print("Mail ID: ");

String mail = scanner.nextLine();

System.out.print("Mobile Number: ");

String mobile = scanner.nextLine();

System.out.print("Basic Pay: ");

double basicPay = scanner.nextDouble();

Programmer programmer = new Programmer(name, id, address, mail, mobile, basicPay);

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

programmer.displayDetails();

System.out.println("Gross Salary: " + programmer.calculateGrossSalary());

System.out.println("Net Salary: " + programmer.calculateNetSalary());

}

}

```

You can extend this program to include classes for Assistant Professor, Associate Professor, and Professor, following the same structure and calculations.

14. Write a program to Check Prime Number using Interface.👇👇👇

Certainly! Here's a simple Java program that uses an interface to check whether a given number is prime:

```java

interface PrimeChecker {

boolean isPrime(int number);

}

class PrimeCheckerImpl implements PrimeChecker {

@Override

public boolean isPrime(int number) {

if (number <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(number); i++) {

if (number % i == 0) {

return false;

}

}

return true;

}

}

public class PrimeNumberCheck {

public static void main(String[] args) {

PrimeChecker primeChecker = new PrimeCheckerImpl();

int numToCheck = 17; // Change this to the number you want to check for primality

if (primeChecker.isPrime(numToCheck)) {

System.out.println(numToCheck + " is a prime number.");

} else {

System.out.println(numToCheck + " is not a prime number.");

}

}

}

```

Replace the `numToCheck` variable with the number you want to check for primality. The program uses an interface `PrimeChecker` and a class `PrimeCheckerImpl` that implements this interface to determine whether a given number is prime or not.

15.Write a program to exhibit simple inheritance, multilevel inheritance and hybrid

inheritance concepts.👇👇👇

Certainly! Below is a simple Java program that demonstrates simple inheritance, multilevel inheritance, and hybrid inheritance concepts:

```java

// Simple Inheritance

class Animal {

void eat() {

System.out.println("Animal is eating");

}

}

// Multilevel Inheritance

class Dog extends Animal {

void bark() {

System.out.println("Dog is barking");

}

}

class Puppy extends Dog {

void play() {

System.out.println("Puppy is playing");

}

}

// Hybrid Inheritance

interface Swimmer {

void swim();

}

class Dolphin extends Animal implements Swimmer {

@Override

public void swim() {

System.out.println("Dolphin is swimming");

}

}

class HybridInheritanceDemo {

public static void main(String[] args) {

// Simple Inheritance

Animal animal = new Animal();

animal.eat();

// Multilevel Inheritance

Puppy puppy = new Puppy();

puppy.eat();

puppy.bark();

puppy.play();

// Hybrid Inheritance

Dolphin dolphin = new Dolphin();

dolphin.eat();

dolphin.swim();

}

}

```

16.a) Write a program to create a dialogbox and menu using JavaFX

Certainly! Here's a simple Java program using JavaFX to create a dialog box and menu:

```java

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Menu;

import javafx.scene.control.MenuBar;

import javafx.scene.control.MenuItem;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

public class DialogBoxAndMenuExample extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Dialog Box and Menu Example");

// Create a menu bar with a menu and menu item

MenuBar menuBar = new MenuBar();

Menu fileMenu = new Menu("File");

MenuItem openItem = new MenuItem("Open");

fileMenu.getItems().add(openItem);

menuBar.getMenus().add(fileMenu);

// Set action for the menu item

openItem.setOnAction(e -> showAlert("Open Dialog", "File Opened"));

// Create a layout and add the menu bar

StackPane layout = new StackPane();

layout.getChildren().add(menuBar);

// Set the scene

Scene scene = new Scene(layout, 300, 200);

primaryStage.setScene(scene);

// Show the stage

primaryStage.show();

}

// Method to display a simple alert dialog

private void showAlert(String title, String content) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(content);

alert.showAndWait();

}

}

```

This program creates a simple JavaFX application with a menu bar containing a "File" menu with an "Open" menu item.

16.b) Write a program to create a grid layout control using JavaFX👇👇👇

Certainly! Below is a simple Java program using JavaFX to create a grid layout control:

```java

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

public class GridLayoutExample extends Application {

@Override

public void start(Stage primaryStage) {

GridPane gridPane = new GridPane();

gridPane.setHgap(10); // Horizontal gap between columns

gridPane.setVgap(10); // Vertical gap between rows

gridPane.setPadding(new Insets(10)); // Padding around the grid

// Adding buttons to the grid

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

Button button = new Button("Button " + (i \* 3 + j + 1));

gridPane.add(button, j, i); // Add button to the grid at specific row and column

}

}

Scene scene = new Scene(gridPane, 300, 200);

primaryStage.setTitle("Grid Layout Example");

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

```

This program creates a 3x3 grid layout using JavaFX's `GridPane` and populates it with buttons. Adjust the dimensions and styling as needed for your specific application.

4.Write a program to create a class named shape. In this class we have three sub

classes circle, triangle and square each class has two member function named

draw () and erase (). Create these using polymorphism concepts.

👇

public class Shape {

public void draw() {

System.out.println("Drawing a generic shape...");

}

public void erase() {

System.out.println("Erasing a generic shape...");

}

}

class Circle extends Shape {

@Override

public void draw() {

System.out.println("Drawing a circle...");

}

@Override

public void erase() {

System.out.println("Erasing a circle...");

}

}

class Triangle extends Shape {

@Override

public void draw() {

System.out.println("Drawing a triangle...");

}

@Override

public void erase() {

System.out.println("Erasing a triangle...");

}

}

class Square extends Shape {

@Override

public void draw() {

System.out.println("Drawing a square...");

}

@Override

public void erase() {

System.out.println("Erasing a square...");

}

}

public class PolymorphismDemo {

public static void main(String[] args) {

Shape[] shapes = {new Circle(), new Triangle(), new Square()};

for (Shape shape : shapes) {

shape.draw();

shape.erase();

}

}

}

11. Write a program to create a file and write data into it using the methods

OutputStream class.