1. **Employee Management System**

```
abstract class Employee {
    String name;
    int id;
    double baseSalary;

    Employee(String name, int id, double baseSalary) {
        this.name = name;
        this.id = id;
        this.baseSalary = baseSalary;
    }

    abstract double calculateSalary();
}

class FullTimeEmployee extends Employee {
    double bonus;

    FullTimeEmployee(String name, int id, double baseSalary, double bonus) {
        super(name, id, baseSalary);
        this.bonus = bonus;
    }

    @Override
    double calculateSalary() {
        return baseSalary + bonus;
    }
}

class PartTimeEmployee extends Employee {
    double hourlyRate;
    int hoursWorked;

    PartTimeEmployee(String name, int id, double baseSalary, double hourlyRate, int
hoursWorked) {
        super(name, id, baseSalary);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    @Override
    double calculateSalary() {
        return baseSalary + (hourlyRate * hoursWorked);
```

```
    }
}

public class EmployeeManagement {
    public static void main(String[] args) {
        FullTimeEmployee fullTime = new FullTimeEmployee("John", 1, 50000, 5000);
        PartTimeEmployee partTime = new PartTimeEmployee("Jane", 2, 20000, 15, 120);

        System.out.println("FullTime Employee Salary: " + fullTime.calculateSalary());
        System.out.println("PartTime Employee Salary: " + partTime.calculateSalary());
    }
}
```

2. Banking System

```
interface Account {
    void deposit(double amount);
    void withdraw(double amount);
    void calculateInterest();
    void viewBalance();
}

class SavingsAccount implements Account {
    private double balance;

    @Override
    public void deposit(double amount) {
        balance += amount;
    }

    @Override
    public void withdraw(double amount) {
        balance -= amount;
    }

    @Override
    public void calculateInterest() {
        double interest = balance * 0.04;
        balance += interest;
    }

    @Override
    public void viewBalance() {
```

```
        System.out.println("Savings Account Balance: " + balance);
    }
}

class CurrentAccount implements Account {
    private double balance;

    @Override
    public void deposit(double amount) {
        balance += amount;
    }

    @Override
    public void withdraw(double amount) {
        balance -= amount;
    }

    @Override
    public void calculateInterest() {
        // Current Account does not have interest
    }

    @Override
    public void viewBalance() {
        System.out.println("Current Account Balance: " + balance);
    }
}

class Bank {
    private Account[] accounts = new Account[10];
    private int accountCount = 0;

    void addAccount(Account account) {
        accounts[accountCount++] = account;
    }

    void showAllBalances() {
        for (int i = 0; i < accountCount; i++) {
            accounts[i].viewBalance();
        }
    }
}

public class BankingSystem {
```

```
    public static void main(String[] args) {
        Bank bank = new Bank();
        Account savings = new SavingsAccount();
        Account current = new CurrentAccount();

        savings.deposit(1000);
        current.deposit(2000);

        bank.addAccount(savings);
        bank.addAccount(current);

        bank.showAllBalances();

        savings.calculateInterest();
        savings.viewBalance();
    }
}
```

3. Shape Class with Circle and Rectangle Subclasses

```
// Parent class Shape
class Shape {
    void draw() {
        System.out.println("Drawing shape");
    }
}

// Circle subclass
class Circle extends Shape {
    double radius;

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

    @Override
    void draw() {
        double area = Math.PI * radius * radius;
        System.out.println("Drawing Circle with area: " + area);
    }
}

// Rectangle subclass
class Rectangle extends Shape {
```

```
    double length, width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    void draw() {
        double area = length * width;
        System.out.println("Drawing Rectangle with area: " + area);
    }
}

// Main class to test
public class Main {
    public static void main(String[] args) {
        Shape shape = new Shape();
        shape.draw(); // Drawing shape

        Shape circle = new Circle(5.0);
        circle.draw(); // Drawing Circle with area

        Shape rectangle = new Rectangle(4.0, 6.0);
        rectangle.draw(); // Drawing Rectangle with area
    }
}
```

4. StringConcat Class for Overloaded concat Methods

```
public class StringConcat {

    // Method to concatenate two strings
    public String concat(String a, String b) {
        return a + b;
    }

    // Method to concatenate three strings
    public String concat(String a, String b, String c) {
        return a + b + c;
    }

    // Method to concatenate an array of strings
    public String concat(String[] strings) {
```

```
        String result = "";
        for (int i = 0; i < strings.length; i++) {
            result += strings[i];
        }
        return result;
    }

    public static void main(String[] args) {
        StringConcat sc = new StringConcat();

        // Test the methods
        System.out.println(sc.concat("Hello, ", "World!"));  // Concatenate two strings
        System.out.println(sc.concat("Java ", "is ", "fun!"));  // Concatenate three strings

        String[] arr = {"I ", "am ", "learning ", "Java."};
        System.out.println(sc.concat(arr));  // Concatenate an array of strings
    }
}
```

5. Employee Management System

```
// Superclass Employee
class Employee {
    String name;
    String employeeId;
    double salary;

    public Employee(String name, String employeeId, double salary) {
        this.name = name;
        this.employeeId = employeeId;
        this.salary = salary;
    }

    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Salary: $" + salary);
    }

    public double calculateSalary() {
        return salary;
    }
}

// Subclass FullTimeEmployee
```

```
class FullTimeEmployee extends Employee {
    double benefits;
    double annualBonus;

    public FullTimeEmployee(String name, String employeeId, double salary, double
benefits, double annualBonus) {
        super(name, employeeId, salary);
        this.benefits = benefits;
        this.annualBonus = annualBonus;
    }

    @Override
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Benefits: $" + benefits);
        System.out.println("Annual Bonus: $" + annualBonus);
    }

    @Override
    public double calculateSalary() {
        return salary + benefits + annualBonus;
    }
}

// Subclass PartTimeEmployee
class PartTimeEmployee extends Employee {
    double hourlyRate;
    int hoursWorked;

    public PartTimeEmployee(String name, String employeeId, double hourlyRate, int
hoursWorked) {
        super(name, employeeId, 0);  // Salary will be calculated later
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    @Override
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Hourly Rate: $" + hourlyRate);
        System.out.println("Hours Worked: " + hoursWorked);
    }

    @Override
```

```
    public double calculateSalary() {
        return hourlyRate * hoursWorked;
    }
}

public class EmployeeManagementSystem {

    public static void main(String[] args) {
        // Creating FullTimeEmployee object
        FullTimeEmployee fullTimeEmployee = new FullTimeEmployee("John Doe",
"FT123", 50000, 10000, 5000);
        fullTimeEmployee.displayDetails();
        System.out.println("Total Salary: $" + fullTimeEmployee.calculateSalary());

        System.out.println();

        // Creating PartTimeEmployee object
        PartTimeEmployee partTimeEmployee = new PartTimeEmployee("Jane Smith",
"PT456", 20, 30);
        partTimeEmployee.displayDetails();
        System.out.println("Total Salary: $" + partTimeEmployee.calculateSalary());
    }
}
```

6. Simple School Management System (Using Inheritance)

```
// Base class
class Person {
    String name;
    int age;

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

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

// Derived class Student
class Student extends Person {
    String studentId;
```

```
    String grade;

    Student(String name, int age, String studentId, String grade) {
        super(name, age);
        this.studentId = studentId;
        this.grade = grade;
    }

    void displayStudentInfo() {
        displayPersonInfo();
        System.out.println("Student ID: " + studentId);
        System.out.println("Grade: " + grade);
    }
}

// Derived class GraduateStudent
class GraduateStudent extends Student {
    String thesisTopic;

    GraduateStudent(String name, int age, String studentId, String grade, String
thesisTopic) {
        super(name, age, studentId, grade);
        this.thesisTopic = thesisTopic;
    }

    void displayGraduateStudentInfo() {
        displayStudentInfo();
        System.out.println("Thesis Topic: " + thesisTopic);
    }
}

public class SchoolManagementSystem {
    public static void main(String[] args) {
        // Create an instance of GraduateStudent
        GraduateStudent gradStudent = new GraduateStudent("Alice", 25, "GS12345", "A",
"Artificial Intelligence in Healthcare");
        gradStudent.displayGraduateStudentInfo();
    }
}
```

7. Shopping Cart Simulation (Using Single Inheritance)

```
// Base class Product
class Product {
```

```
    int productId;
    String productName;
    double price;

    Product(int productId, String productName, double price) {
        this.productId = productId;
        this.productName = productName;
        this.price = price;
    }

    void displayProductInfo() {
        System.out.println("Product ID: " + productId);
        System.out.println("Product Name: " + productName);
        System.out.println("Price: $" + price);
    }
}

// Derived class CartItem
class CartItem extends Product {
    int quantity;

    CartItem(int productId, String productName, double price, int quantity) {
        super(productId, productName, price);
        this.quantity = quantity;
    }

    double calculateTotalPrice() {
        return price * quantity;
    }

    void displayCartItemInfo() {
        displayProductInfo();
        System.out.println("Quantity: " + quantity);
        System.out.println("Total Price: $" + calculateTotalPrice());
    }
}

public class ShoppingCart {
    public static void main(String[] args) {
        // Create an instance of CartItem
        CartItem item = new CartItem(101, "Laptop", 999.99, 2);
        item.displayCartItemInfo();
    }
}
```

8. **Create a class called "Person" with a name and age attribute**

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

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

    // Constructor without parameters
    public Person() {
        this.name = "Unknown";
        this.age = 0;
    }

    // Method to display the details
    public void displayDetails() {
        System.out.println("Name: " + name + ", Age: " + age);
    }

    public static void main(String[] args) {
        // Creating instances using different constructors
        Person person1 = new Person("Alice", 30); // Using parameterized constructor
        Person person2 = new Person(); // Using default constructor

        // Display details
        person1.displayDetails();
        person2.displayDetails();
    }
}
```

9. **Display details of five different cities :**

```
class City {
    String name;
    int population;

    // Constructor to initialize City
    public City(String name, int population) {
        this.name = name;
        this.population = population;
    }
```

```
    // Method to display the city details
    public void displayDetails() {
        System.out.println("City: " + name + ", Population: " + population);
    }

    public static void main(String[] args) {
        // Creating 5 City objects
        City city1 = new City("New York", 8419600);
        City city2 = new City("London", 8982000);
        City city3 = new City("Paris", 2148327);
        City city4 = new City("Tokyo", 13929286);
        City city5 = new City("Berlin", 3644826);

        // Display details of each city
        city1.displayDetails();
        city2.displayDetails();
        city3.displayDetails();
        city4.displayDetails();
        city5.displayDetails();
    }
}
```

10. **Banking System for PCCOE Cooperative Bank**:

```
class Account {
    String accountHolder;
    int accountNumber;
    double balance;

    public Account(String accountHolder, int accountNumber, double initialDeposit) {
        this.accountHolder = accountHolder;
        this.accountNumber = accountNumber;
        this.balance = initialDeposit;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount + ", New Balance: " + balance);
        } else {
            System.out.println("Deposit amount must be positive!");
        }
    }
```

```
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount + ", New Balance: " + balance);
        } else {
            System.out.println("Invalid withdrawal amount!");
        }
    }

    public void displayAccountInfo() {
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: " + balance);
    }
}

class Bank {
    private Account[] accounts;
    private int accountCount;

    public Bank(int capacity) {
        accounts = new Account[capacity];
        accountCount = 0;
    }

    public void addAccount(String accountHolder, int accountNumber, double
initialDeposit) {
        if (accountCount < accounts.length) {
            accounts[accountCount++] = new Account(accountHolder, accountNumber,
initialDeposit);
            System.out.println("Account added successfully!");
        } else {
            System.out.println("Bank is full, cannot add more accounts.");
        }
    }

    public void removeAccount(int accountNumber) {
        for (int i = 0; i < accountCount; i++) {
            if (accounts[i].accountNumber == accountNumber) {
                for (int j = i; j < accountCount - 1; j++) {
                    accounts[j] = accounts[j + 1];
                }
                accounts[--accountCount] = null;
                System.out.println("Account removed successfully!");
```

```
                return;
            }
        }
        System.out.println("Account not found!");
    }

    public Account findAccount(int accountNumber) {
        for (int i = 0; i < accountCount; i++) {
            if (accounts[i].accountNumber == accountNumber) {
                return accounts[i];
            }
        }
        return null;
    }
}

public class BankSystem {
    public static void main(String[] args) {
        Bank pccoeBank = new Bank(5); // capacity of 5 accounts
        pccoeBank.addAccount("John Doe", 101, 5000);
        pccoeBank.addAccount("Jane Smith", 102, 3000);

        Account account = pccoeBank.findAccount(101);
        if (account != null) {
            account.deposit(2000);
            account.withdraw(1500);
            account.displayAccountInfo();
        }

        pccoeBank.removeAccount(102);
    }
}
```

11. L**ibrary Catalog System Using Inheritance and Aggregation**:

```
// Base class for all library items
class Item {
    String title;
    String author;

    public Item(String title, String author) {
        this.title = title;
        this.author = author;
    }
```

```
    public void displayItemDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
    }
}

// Derived class for Books
class Book extends Item {
    int pages;

    public Book(String title, String author, int pages) {
        super(title, author);
        this.pages = pages;
    }

    @Override
    public void displayItemDetails() {
        super.displayItemDetails();
        System.out.println("Pages: " + pages);
    }
}

// Derived class for DVDs
class DVD extends Item {
    double duration; // in hours

    public DVD(String title, String author, double duration) {
        super(title, author);
        this.duration = duration;
    }

    @Override
    public void displayItemDetails() {
        super.displayItemDetails();
        System.out.println("Duration: " + duration + " hours");
    }
}

// Library class that aggregates multiple items
class Library {
    private Item[] items;
    private int itemCount;

    public Library(int capacity) {
```

```
        items = new Item[capacity];
        itemCount = 0;
    }

    public void addItem(Item item) {
        if (itemCount < items.length) {
            items[itemCount++] = item;
            System.out.println("Item added to library!");
        } else {
            System.out.println("Library is full, cannot add more items.");
        }
    }

    public void displayLibraryItems() {
        System.out.println("Library Catalog:");
        for (int i = 0; i < itemCount; i++) {
            items[i].displayItemDetails();
            System.out.println();
        }
    }
}

public class LibrarySystem {
    public static void main(String[] args) {
        Library library = new Library(5); // Library can hold 5 items

        Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", 180);
        DVD dvd1 = new DVD("The Godfather", "Francis Ford Coppola", 3.0);

        library.addItem(book1);
        library.addItem(dvd1);

        library.displayLibraryItems();
    }
}
```

12. Geometric Shapes Application (Function Overloading)

```
class Shape {

    // Overloaded method for calculating area of rectangle
    public double area(double length, double width) {
        return length * width;
    }
```

```
    // Overloaded method for calculating perimeter of rectangle
    public double perimeter(double length, double width) {
        return 2 * (length + width);
    }

    // Overloaded method for calculating area of circle
    public double area(double radius) {
        return Math.PI * radius * radius;
    }

    // Overloaded method for calculating perimeter of circle (circumference)
    public double perimeter(double radius) {
        return 2 * Math.PI * radius;
    }

    // Overloaded method for calculating area of triangle
    public double area(double base, double height) {
        return 0.5 * base * height;
    }

    // Overloaded method for calculating perimeter of triangle
    public double perimeter(double side1, double side2, double side3) {
        return side1 + side2 + side3;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape shape = new Shape();

        // Rectangle
        System.out.println("Rectangle Area: " + shape.area(5.0, 3.0));
        System.out.println("Rectangle Perimeter: " + shape.perimeter(5.0, 3.0));

        // Circle
        System.out.println("Circle Area: " + shape.area(4.0));
        System.out.println("Circle Perimeter: " + shape.perimeter(4.0));

        // Triangle
        System.out.println("Triangle Area: " + shape.area(5.0, 3.0));
        System.out.println("Triangle Perimeter: " + shape.perimeter(3.0, 4.0, 5.0));
    }
}
```

13. Social Media Platform (Method Overriding)

```
class Post {
    // Base class method for posting
    public void post() {
        System.out.println("Posting a generic post...");
    }

    // Base class method for commenting
    public void comment() {
        System.out.println("Commenting on a generic post...");
    }
}

class TextPost extends Post {
    // Overriding post method for TextPost
    @Override
    public void post() {
        System.out.println("Posting a text-based post...");
    }

    // Overriding comment method for TextPost
    @Override
    public void comment() {
        System.out.println("Commenting on a text-based post...");
    }
}

class ImagePost extends Post {
    // Overriding post method for ImagePost
    @Override
    public void post() {
        System.out.println("Posting an image-based post...");
    }

    // Overriding comment method for ImagePost
    @Override
    public void comment() {
        System.out.println("Commenting on an image-based post...");
    }
}

public class Main {
    public static void main(String[] args) {
        Post myPost = new Post();
```

```
        myPost.post();
        myPost.comment();

        TextPost textPost = new TextPost();
        textPost.post();
        textPost.comment();

        ImagePost imagePost = new ImagePost();
        imagePost.post();
        imagePost.comment();
    }
}
```

14. University Course Registration System (using abstract classes) :

```
// Abstract class for Course
abstract class Course {
    String courseCode;
    String title;
    int credits;

    // Constructor
    public Course(String courseCode, String title, int credits) {
        this.courseCode = courseCode;
        this.title = title;
        this.credits = credits;
    }

    // Abstract method to enroll students
    public abstract void enrollStudent();

    // Abstract method to display course details
    public abstract void displayCourseDetails();
}

// Derived class for Information Technology course
class InformationTechnology extends Course {

    // Constructor
    public InformationTechnology(String courseCode, String title, int credits) {
        super(courseCode, title, credits);
    }

    @Override
    public void enrollStudent() {
```

```
        System.out.println("Enrolling students in Information Technology course.");
    }

    @Override
    public void displayCourseDetails() {
        System.out.println("Course Code: " + courseCode);
        System.out.println("Course Title: " + title);
        System.out.println("Credits: " + credits);
    }
}

// Derived class for Mechanical Engineering course
class MechanicalEngineering extends Course {

    // Constructor
    public MechanicalEngineering(String courseCode, String title, int credits) {
        super(courseCode, title, credits);
    }

    @Override
    public void enrollStudent() {
        System.out.println("Enrolling students in Mechanical Engineering course.");
    }

    @Override
    public void displayCourseDetails() {
        System.out.println("Course Code: " + courseCode);
        System.out.println("Course Title: " + title);
        System.out.println("Credits: " + credits);
    }
}

public class UniversityCourseRegistration {
    public static void main(String[] args) {
        // Create instances of courses
        Course itCourse = new InformationTechnology("IT101", "Introduction to IT", 3);
        Course mechCourse = new MechanicalEngineering("ME101", "Introduction to
Mechanical Engineering", 4);

        // Display course details and enroll students
        itCourse.displayCourseDetails();
        itCourse.enrollStudent();

        mechCourse.displayCourseDetails();
```

```
        mechCourse.enrollStudent();
    }
}
```

15. Transportation System (using Polymorphism) :

```
// Interface for Transport
interface Transport {
    // Method to calculate fare
    double calculateFare();
}

// Class for Car
class Car implements Transport {
    private double distance;

    public Car(double distance) {
        this.distance = distance;
    }

    @Override
    public double calculateFare() {
        // Fare calculation for car (e.g., $2 per kilometer)
        return distance * 2;
    }
}

// Class for Bus
class Bus implements Transport {
    private double distance;

    public Bus(double distance) {
        this.distance = distance;
    }

    @Override
    public double calculateFare() {
        // Fare calculation for bus (e.g., $1.5 per kilometer)
        return distance * 1.5;
    }
}

// Class for Train
class Train implements Transport {
```

```
    private double distance;

    public Train(double distance) {
        this.distance = distance;
    }

    @Override
    public double calculateFare() {
        // Fare calculation for train (e.g., $3 per kilometer)
        return distance * 3;
    }
}

public class TransportationSystem {
    public static void main(String[] args) {
        // Create transport objects
        Transport car = new Car(10); // 10 kilometers
        Transport bus = new Bus(10); // 10 kilometers
        Transport train = new Train(10); // 10 kilometers

        // Calculate and display fare for each mode of transport
        System.out.println("Car fare: $" + car.calculateFare());
        System.out.println("Bus fare: $" + bus.calculateFare());
        System.out.println("Train fare: $" + train.calculateFare());
    }
}
```

16. Matrix Multiplication Using Multithreading :

```
class MatrixMultiplication implements Runnable {
    private int[][] matrix1;
    private int[][] matrix2;
    private int[][] result;
    private int row;
    private int col;

    public MatrixMultiplication(int[][] matrix1, int[][] matrix2, int[][] result, int row, int col) {
        this.matrix1 = matrix1;
        this.matrix2 = matrix2;
        this.result = result;
        this.row = row;
        this.col = col;
    }

    @Override
```

```
    public void run() {
        try {
            int sum = 0;
            for (int i = 0; i < matrix1[0].length; i++) {
                sum += matrix1[row][i] * matrix2[i][col];
            }
            result[row][col] = sum;
        } catch (Exception e) {
            System.out.println("Error in matrix multiplication: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        int[][] matrix1 = {{1, 2}, {3, 4}};
        int[][] matrix2 = {{5, 6}, {7, 8}};
        int rows = matrix1.length;
        int cols = matrix2[0].length;
        int[][] result = new int[rows][cols];

        Thread[] threads = new Thread[rows * cols];
        int count = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                threads[count] = new Thread(new MatrixMultiplication(matrix1, matrix2, result,
i, j));
                threads[count].start();
                count++;
            }
        }

        try {
            for (Thread t : threads) {
                t.join();
            }
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }

        // Print the resulting matrix
        System.out.println("Result Matrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(result[i][j] + " ");
            }
```

```
            System.out.println();
        }
    }
}
```

17. **Log File Analysis Using I/O Operations :**

```
import java.io.*;

public class LogFileAnalyzer {
    public static void main(String[] args) {
        String fileName = "logfile.txt";
        BufferedReader reader = null;
        int errorCount = 0;
        int warningCount = 0;

        try {
            reader = new BufferedReader(new FileReader(fileName));
            String line;

            while ((line = reader.readLine()) != null) {
                if (line.contains("ERROR")) {
                    errorCount++;
                } else if (line.contains("WARNING")) {
                    warningCount++;
                }
            }

            System.out.println("Error Count: " + errorCount);
            System.out.println("Warning Count: " + warningCount);
        } catch (FileNotFoundException e) {
            System.out.println("Log file not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error reading the file: " + e.getMessage());
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.out.println("Error closing the reader: " + e.getMessage());
            }
        }
    }
```

```
}
```

18. Person, Employee, and Manager Classes :

```
// Base class Person
class Person {
    String name;

    // Constructor to initialize name
    public Person(String name) {
        this.name = name;
    }
}

// Derived class Employee extends Person
class Employee extends Person {
    String employeeId;

    // Constructor to initialize both name and employeeId
    public Employee(String name, String employeeId) {
        super(name);  // Call the parent class constructor
        this.employeeId = employeeId;
    }
}

// Derived class Manager extends Employee
class Manager extends Employee {
    String department;

    // Constructor to initialize name, employeeId, and department
    public Manager(String name, String employeeId, String department) {
        super(name, employeeId);  // Call the parent class constructor
        this.department = department;
    }

    // Method to display all information
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Department: " + department);
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        // Create an instance of Manager
        Manager manager = new Manager("John Doe", "EMP123", "HR");

        // Display all information
        manager.displayInfo();
    }
}
```

19. Write a Java Program to perform following operations using String Class. :

```
public class StringOperations {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";

        // 1. Get length of a string
        System.out.println("Length of str1: " + str1.length());

        // 2. Join two strings
        String joined = str1 + " " + str2;
        System.out.println("Joined String: " + joined);

        // 3. Compare two strings
        int comparison = str1.compareTo(str2);
        if (comparison < 0) {
            System.out.println(str1 + " is lexicographically less than " + str2);
        } else if (comparison > 0) {
            System.out.println(str1 + " is lexicographically greater than " + str2);
        } else {
            System.out.println(str1 + " is equal to " + str2);
        }

        // 4. Replace a string
        String replacedString = str1.replace('l', 'p');
        System.out.println("Replaced String: " + replacedString);

        // 5. Get index of particular character from a string
        int index = str1.indexOf('l');
        System.out.println("Index of character 'l' in str1: " + index);
    }
}
```

20. Student Class :

```
class Student {
```

```
    int id;
    String name;
    double sgpa;

    // Constructor to initialize attributes
    public Student(int id, String name, double sgpa) {
        this.id = id;
        this.name = name;
        this.sgpa = sgpa;
    }

    // Method to display student information
    public void displayStudentInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", SGPA: " + sgpa);
    }

    public static void main(String[] args) {
        // Create and initialize five student objects
        Student student1 = new Student(1, "John Doe", 8.5);
        Student student2 = new Student(2, "Jane Smith", 9.2);
        Student student3 = new Student(3, "Alice Brown", 7.8);
        Student student4 = new Student(4, "Bob White", 6.9);
        Student student5 = new Student(5, "Charlie Green", 8.1);

        // Display student information
        student1.displayStudentInfo();
        student2.displayStudentInfo();
        student3.displayStudentInfo();
        student4.displayStudentInfo();
        student5.displayStudentInfo();
    }
}
```

21. Library Management System :

```
class Book {
    int id;
    String title;
    String author;
    boolean isAvailable;

    // Constructor to initialize attributes
    public Book(int id, String title, String author, boolean isAvailable) {
        this.id = id;
```

```
        this.title = title;
        this.author = author;
        this.isAvailable = isAvailable;
    }

    // Method to display book information
    public void displayBookInfo() {
        System.out.println("ID: " + id + ", Title: " + title + ", Author: " + author + ", Available: "
+ isAvailable);
    }

    public static void main(String[] args) {
        // Create and initialize five book objects
        Book book1 = new Book(1, "The Great Gatsby", "F. Scott Fitzgerald", true);
        Book book2 = new Book(2, "1984", "George Orwell", false);
        Book book3 = new Book(3, "To Kill a Mockingbird", "Harper Lee", true);
        Book book4 = new Book(4, "The Catcher in the Rye", "J.D. Salinger", true);
        Book book5 = new Book(5, "Moby Dick", "Herman Melville", false);

        // Display book information
        book1.displayBookInfo();
        book2.displayBookInfo();
        book3.displayBookInfo();
        book4.displayBookInfo();
        book5.displayBookInfo();
    }
}
```

22. Write a Java Program for a simple payroll system :

```
// Base class Employee
class Employee {
    String name;
    int id;
    double basicSalary;
    double TA;  // Traveling Allowance
    double DA;  // Dearness Allowance

    // Constructor to initialize employee details
    public Employee(String name, int id, double basicSalary, double TA, double DA) {
        this.name = name;
        this.id = id;
        this.basicSalary = basicSalary;
        this.TA = TA;
```

```
        this.DA = DA;
    }

    // Method to calculate salary
    public double calculateSalary() {
        return basicSalary + TA + DA;
    }

    // Method to display employee details and salary
    public void displaySalary() {
        System.out.println("Employee Name: " + name);
        System.out.println("Employee ID: " + id);
        System.out.println("Salary: " + calculateSalary());
    }
}

// Derived class Manager
class Manager extends Employee {
    double bonus;  // Additional bonus for Manager

    // Constructor to initialize manager details
    public Manager(String name, int id, double basicSalary, double TA, double DA, double
bonus) {
        super(name, id, basicSalary, TA, DA);  // Call the parent constructor
        this.bonus = bonus;
    }

    // Overridden method to calculate salary (including bonus)
    @Override
    public double calculateSalary() {
        return super.calculateSalary() + bonus;  // Add bonus to the salary
    }

    // Method to display manager details and salary
    public void displaySalary() {
        System.out.println("Manager Name: " + name);
        System.out.println("Manager ID: " + id);
        System.out.println("Salary (including bonus): " + calculateSalary());
    }
}

public class PayrollSystem {
    public static void main(String[] args) {
        // Create an Employee object
```

```
        Employee emp = new Employee("John Doe", 101, 30000, 5000, 3000);
        emp.displaySalary();  // Display Employee's Salary

        System.out.println();  // Add a blank line for separation

        // Create a Manager object
        Manager mgr = new Manager("Alice Smith", 102, 50000, 7000, 4000, 10000);
        mgr.displaySalary();  // Display Manager's Salary
    }
}
```