Module 3 Classes and Objects

Premanand S

Assistant Professor
School of Electonics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

February 28, 2025

Contents - Module 3

- Class Fundamentals,
- Access and Non-access Specifiers
- Declaring Objects and assigning object reference variables,
- Array of objects,
- Constructor and Destructors,
- usage of 'this' and 'static' keywords

Class fundamentals

- Create a class Book with attributes title and author. Write a method equals(Book other) that checks if two books have the same title and author. Test this method by creating multiple Book objects

Class - Example

Example (Understanding)

```
// Book.java
class Book {
    String title;
    String author;
    boolean equals(Book other) {
        return this.title.equals(other.title) && this.author.
        equals(other.author);
    }
```

Class - Example

Example (Understanding)

```
public static void main(String[] args) {
    Book book1 = new Book();
    book1.title = "Java Programming";
    book1.author = "John Doe";
    Book book2 = new Book();
    book2.title = "Java Programming";
    book2.author = "John Doe";
    Book book3 = new Book();
    book3.title = "Python Basics";
    book3.author = "Jane Smith";
    System.out.println("book1 equals book2? " +
    book1.equals(book2));
    System.out.println("book1 equals book3? " +
    book1.equals(book3));
}}
```

Class fundamentals

- Write a program where a Person class has private fields name and age. Use a parameterized constructor to initialize these fields. Create two objects of the class and compare their ages using a method isOlderThan(Person other).

Class - Example

Example (Understanding)

```
// Person.java
class Person {
    private String name;
    private int age;

    // Parameterized constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to compare ages
    public boolean isOlderThan(Person other) {
        return this.age > other.age;
    }
```

Class - Example

Example (Understanding)

```
    public static void main(String[] args) {
        // Creating Person objects
        Person person1 = new Person("Prem", 25);
        Person person2 = new Person("Anand", 30);

        // Comparing ages
        if (person1.isOlderThan(person2)) {
            System.out.println(person1.name + " is older
            than " + person2.name);
        } else {
            System.out.println(person2.name + " is older
            than " + person1.name);
        }
    }
}
```

Class fundamentals

- Create a class Counter with a static variable count and a non-static variable id. Increment count every time an object is created and assign the current value of count to id. Display the id of multiple objects.

Class - Example

Example (Understanding)

```
// Counter.java
class Counter {
    private static int count = 0;
    private int id;

    // Constructor to increment count and assign it to id
    public Counter() {
        count++;
        this.id = count;
    }

    // Method to display the id
    public void displayId() {
        System.out.println("Object ID: " + id);
    }
```

Class - Example

Example (Understanding)

```
    public static void main(String[] args) {
        // Creating multiple objects
        Counter obj1 = new Counter();
        Counter obj2 = new Counter();
        Counter obj3 = new Counter();

        // Displaying IDs
        obj1.displayId();
        obj2.displayId();
        obj3.displayId();
    }
}
```

Class fundamentals

- Write a program where a Vehicle class has private fields color and model. Use public getter and setter methods to access these fields. Create a subclass Car that inherits from Vehicle and adds a new field numberOfDoors. Display the details of a Car object.

Class - Example

Example (Understanding)

```
class Vehicle {
    private String color;
    private String model;
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }}
```

Class - Example

Example (Understanding)

```
class Car extends Vehicle {
    private int numberOfDoors;
    public int getNumberOfDoors() {
        return numberOfDoors;}
    public void setNumberOfDoors(int numberOfDoors) {
        this.numberOfDoors = numberOfDoors;}
    public void displayCarDetails() {
        System.out.println("Car Model: " + getModel());
        System.out.println("Car Color: " + getColor());
        System.out.println("Number of Doors: " +
        getNumberOfDoors());}
```

Class - Example

Example (Understanding)

```
public static void main(String[] args) {
    Car myCar = new Car();
    myCar.setModel("Toyota Corolla");
    myCar.setColor("Red");
    myCar.setNumberOfDoors(4);
    myCar.displayCarDetails();
}}
```

Class fundamentals

- Write a program to demonstrate the use of null with object reference variables. Create a class Animal with a method makeSound(). Assign null to an Animal reference and try to call the makeSound() method.What happens?

Class - Example

Example (Understanding)

```
class Animal {
    // Method to make sound
    public void makeSound() {
        System.out.println("Some generic animal sound");
    }
}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Create an Animal reference and assign null to it
        Animal animal = null;
        try {
            // Attempt to call the makeSound() method
            on the null reference
            animal.makeSound();
        } catch (NullPointerException e) {
            // Handle the exception
            System.out.println("A NullPointerException
            occurred: " + e.getMessage());
        }
```

Class - Example

Example (Understanding)

```
        // Explanation of what happened
        System.out.println("The reference 'animal' was null,
        so calling makeSound() caused a
        NullPointerException.");
    }
}
```

Class fundamentals

- Create a class Box with attributes length, width, and height. Write a method to calculate the volume. Assign the reference of one Box object to another variable and set the dimensions of the second variable to null. Explain what happens when you try to access the dimensions of the second variable.

Class - Example

Example (Understanding)

```
// Define the Box class
class Box {
    double length;
    double width;
    double height;
    public Box(double length, double width, double height) {
        this.length = length;
        this.width = width;
        this.height = height;
    }
    public double calculateVolume() {
        return length * width * height;
    }}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Box box1 = new Box(5.0, 3.0, 2.0);
        // Assign the reference of box1 to another
        // variable box2
        Box box2 = box1;
        // Print the volume of box1 and box2
        // (both should be the same)
        System.out.println("Volume of box1: " +
        box1.calculateVolume());
        System.out.println("Volume of box2: " +
        box2.calculateVolume());
        // Set box2 to null
        box2 = null;
```

Class - Example

Example (Understanding)

```
// Try to access the dimensions of box2
try {
    if (box2 != null) {
        System.out.println("Dimensions of box2: " +
        box2.length + " x " + box2.width + " x " +
        box2.height);
    } else {
        System.out.println("box2 is null, so its
        dimensions cannot be accessed.");
    }
} catch (NullPointerException e) {
    System.out.println("A NullPointerException
    occurred: " + e.getMessage());
}
```

Class - Example

Example (Understanding)

```
        // Accessing dimensions of box1 (still valid)
        System.out.println("Dimensions of box1: " +
        box1.length + " x " + box1.width + " x " +
        box1.height);
    }}
```

Class fundamentals

- Write a program to demonstrate the difference between assigning an object reference and creating a new object. Use a class Laptop with attributes brand and price.

Class - Example

Example (Understanding)

```
// Define the Laptop class
class Laptop {
    // Attributes
    String brand;
    double price;
    // Constructor to initialize attributes
    public Laptop(String brand, double price) {
        this.brand = brand;
        this.price = price;
    }
    // Method to display laptop details
    public void displayDetails() {
        System.out.println("Brand: " + brand + ",
        Price: $" + price);
    }}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Laptop laptop1 = new Laptop("Dell", 1200.0);

        Laptop laptop2 = laptop1;

        System.out.println("Before modifying laptop2:");
        System.out.print("laptop1: ");
        laptop1.displayDetails();
        System.out.print("laptop2: ");
        laptop2.displayDetails();

        // Modify the attributes using laptop2
        laptop2.brand = "HP";
        laptop2.price = 1500.0;
```

Class - Example

Example (Understanding)

```
        System.out.println("\nAfter modifying laptop2:");
        System.out.print("laptop1: ");
        laptop1.displayDetails();
        System.out.print("laptop2: ");
        laptop2.displayDetails();

        laptop2 = new Laptop("Lenovo", 1000.0);
```

Class - Example

Example (Understanding)

```
        System.out.println("\nAfter creating a new object
        for laptop2:");
        System.out.print("laptop1: ");
        laptop1.displayDetails();
        System.out.print("laptop2: ");
        laptop2.displayDetails();
    }}
```

Class fundamentals

- Create a class Circle with a radius attribute. Write a method to calculate the area of the circle. Assign the reference of one Circle object to another variable and modify the radius through the second variable. Print the area using both references.

Class - Example

Example (Understanding)

```
// Define the Circle class
class Circle {
    // Attribute
    double radius;

    // Constructor to initialize the radius
    public Circle(double radius) {
        this.radius = radius;
    }

    // Method to calculate the area of the circle
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Create a Circle object with an initial radius
        Circle circle1 = new Circle(5.0);

        // Assign the reference of circle1 to circle2
        Circle circle2 = circle1;

        // Print the area using both references
        System.out.println("Before modifying the radius:");
        System.out.println("Area using circle1: " +
        circle1.calculateArea());
        System.out.println("Area using circle2: " +
        circle2.calculateArea());
```

Class - Example

Example (Understanding)

```
        // Modify the radius through circle2
        circle2.radius = 10.0;

        // Print the area using both references again
        System.out.println("\nAfter modifying the radius
        through circle2:");
        System.out.println("Area using circle1: " +
        circle1.calculateArea());
        System.out.println("Area using circle2: " +
        circle2.calculateArea());
    }
}
```

Class fundamentals

- Write a program where two reference variables point to the same object. Modify the object through one reference and demonstrate how the change is reflected when accessed through the other reference

Class - Example

Example (Understanding)

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

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

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

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Create a Person object
        Person person1 = new Person("Prem", 25);

        // Assign the reference of person1 to person2
        Person person2 = person1;

        // Display details using both references
        System.out.println("Before modifying the object:");
        System.out.print("person1: ");
        person1.displayDetails();
        System.out.print("person2: ");
        person2.displayDetails();
```

Class - Example

Example (Understanding)

```
        // Modify the object through person2
        person2.name = "Anand";
        person2.age = 30;

        // Display details using both references again
        System.out.println("\nAfter modifying the object
        through person2:");
        System.out.print("person1: ");
        person1.displayDetails();
        System.out.print("person2: ");
        person2.displayDetails();
    }
}
```

Class fundamentals

- Create a class Person with a private field age. Write a public method isAdult() that returns true if the age is greater than or equal to 18, otherwise false. Test this method in the main method

Class - Example

Example (Understanding)

```
class Person {
    // Private field for age
    private int age;

    // Constructor to initialize the age
    public Person(int age) {
        this.age = age;
    }

    // Public method to check if the person is an adult
    public boolean isAdult() {
        return age >= 18;
    }
}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Create Person objects with different ages
        Person person1 = new Person(20); // Adult
        Person person2 = new Person(16); // Not an adult

        // Test the isAdult() method for person1
        if (person1.isAdult()) {
            System.out.println("Person 1 is an adult.");
        } else {
            System.out.println("Person 1 is not an adult.");
        }
```

Class - Example

Example (Understanding)

```
        // Test the isAdult() method for person2
        if (person2.isAdult()) {
            System.out.println("Person 2 is an adult.");
        } else {
            System.out.println("Person 2 is not an adult.");
        }
    }
}
```

Class fundamentals

- Write a program with a static variable count in a class Counter. Increment count every time an object of the class is created. Display the value of count after creating multiple objects.

Class - Example

Example (Understanding)

```
class Counter {
    // Static variable to keep track of the number of objects
    private static int count = 0;

    // Constructor to increment the count when an object is cr
    public Counter() {
        count++;
    }

    // Static method to get the current value of count
    public static int getCount() {
        return count;
    }
}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Counter c1 = new Counter();
        System.out.println("After creating c1, count: "
        + Counter.getCount());

        Counter c2 = new Counter();
        System.out.println("After creating c2, count: "
        + Counter.getCount());

        Counter c3 = new Counter();
        System.out.println("After creating c3, count: "
        + Counter.getCount());
    }
}
```

Class fundamentals

- Create a class Employee with protected fields name and salary. Write a subclass Manager that inherits from Employee and adds a new field department. Display the details of a Manager object.

Class - Example

Example (Understanding)

```
class Employee {
    // Protected fields
    protected String name;
    protected double salary;

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

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

Class - Example

Example (Understanding)

```
// Define the Manager subclass that inherits from Employee
class Manager extends Employee {
    // Additional field for department
    private String department;

    // Constructor to initialize name, salary, and department
    public Manager(String name, double salary,
    String department) {
        super(name, salary); // Call the superclass constructo
        this.department = department;
    }
```

Class - Example

Example (Understanding)

```
    // Override the displayDetails method to include departmen
    @Override
    public void displayDetails() {
        super.displayDetails(); // Call the superclass method
        System.out.println("Department: " + department);
    }
}

public class Main {
    public static void main(String[] args) {
        Manager manager = new Manager("Ram", 75000,
        "Human Resources");
        System.out.println("Manager Details:");
        manager.displayDetails();
    }}
```

Class fundamentals

- Write a class BankAccount with a private field balance. Provide public getter and setter methods to access and modify the balance. Demonstrate how another class can use these methods to interact with the balance field.

Class - Example

Example (Understanding)

```
class BankAccount {
    private double balance;

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        if (balance >= 0) {
            this.balance = balance;
            System.out.println("Balance updated successfully.'
        } else {
            System.out.println("Invalid balance. Balance
            must be non-negative.");
        }    }
```

Class - Example

Example (Understanding)

```
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("$" + amount + " deposited.
            New balance: $" + balance);
        } else {
            System.out.println("Invalid deposit amount.
            Amount must be positive.");
        }    }
```

Class - Example

Example (Understanding)

```
public void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("$" + amount + " withdrawn.
        New balance: $" + balance);
    } else {
        System.out.println("Invalid withdrawal amount
        or insufficient balance.");
    }    }}
```

Class - Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        account.setBalance(1000);
        System.out.println("Current balance: $" +
        account.getBalance());
        account.deposit(500);
        account.withdraw(300);
        account.withdraw(2000);
        account.setBalance(-100);
        System.out.println("Final balance: $"
        + account.getBalance());
    }}
```