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 19, 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

Class fundamentals

- A class is a blueprint or template for creating objects.
- It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.
- Classes encapsulate data and functions into a single unit.

Example (Understanding)

```
class ClassName {
    // Fields (Attributes)
    // Methods (Behaviors)
}
```

Class - Example

Example (Understanding)

```
class Car {
    String color;
    int year;
    void displayInfo() {
        System.out.println("Car Color: " + color + ",
        Year: " + year);
    }}
public class Main {
    public static void main(String[] args) {
    // Creating an object of Car
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.year = 2020;
        myCar.displayInfo();
    }}
```

What is a class in Java? How does it differ from an object?

- A class is a blueprint or template that defines the structure and behavior of objects. It specifies the attributes (data members) and methods (functions) that the objects created from the class will have.
- Think of a class as a design or a recipe for creating objects. For example, a Car class might define attributes like color, model, and speed, and methods like accelerate() and brake().
- An object is an instance of a class. It represents a specific entity created using the blueprint defined by the class. Each object has its own set of attributes and can perform actions defined by the methods.
- For example, if Car is a class, then myCar could be an object of the Car class with specific values for color and model.

Can a class exist without any attributes or methods?

- Yes, a class can exist without any attributes or methods. Such a class is often referred to as an empty class or a marker class.

Example (Understanding)

```
class EmptyClass {
    // No attributes or methods
}

public class Main {
    public static void main(String[] args) {
    // Creating an object of EmptyClass
        EmptyClass obj = new EmptyClass();
        System.out.println("Object created successfully.");
    }
}
```

Can a class exist without any attributes or methods?

- Even though the class EmptyClass has no attributes or methods, you can still create an object of this class.
- Every class implicitly inherits from the Object class, which provides some basic methods like toString(), equals(), and hashCode(). So, even an empty class has some default functionality inherited from Object.
- Use Cases for Empty Classes:
  - Marker Interfaces : Marker interfaces (like Serializable or Cloneable) are interfaces with no methods. Similarly, an empty class can be used as a marker to indicate a certain type or category.
  - Placeholder : Sometimes, during development, you might define an empty class as a placeholder for future implementation.

Marker Class - Example

Example (Understanding)

```
class MarkerClass {
    // Empty class used as a marker
}

public class Main {
    public static void main(String[] args) {
        MarkerClass marker = new MarkerClass();
        if (marker instanceof MarkerClass) {
            System.out.println("This is a marker class
            instance.");
        }
    }
}
```

What is the purpose of an empty class in Java?

- An empty class can serve as a marker or placeholder.
- It can also be used to group related classes or indicate a specific type without adding any functionality.

Class Vs Sub Class Vs Super Class

- A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.
- For example, a Vehicle class might define common attributes like color, model, and methods like start() and stop().
- A superclass (also known as a parent class or base class) is a class that other classes can inherit from. It contains common attributes and methods that can be shared by its subclasses.
- For example, a Vehicle class could be a superclass, and it might contain general information about vehicles.
- A subclass (also known as a child class or derived class) is a class that inherits from a superclass. It can reuse the attributes and methods of the superclass and also add its own specific attributes and methods.
- For example, a Car class could be a subclass of the Vehicle superclass. The Car class would inherit the general properties of a vehicle but could also have specific attributes like numberOfDoors.

Class Vs Sub Class Vs Super Class

Example (Understanding)

```
// Superclass (Parent Class)
class Vehicle {
    String color;  // Attribute in the superclass
    int year;      // Attribute in the superclass

    // Method in the superclass
    void start() {
        System.out.println("Vehicle started.");
    }

    // Method in the superclass
    void stop() {
        System.out.println("Vehicle stopped.");
    }
}
```

Class Vs Sub Class Vs Super Class

Example (Understanding)

```
// Subclass (Child Class) inheriting from Vehicle
class Car extends Vehicle {
    int numberOfDoors;  // Specific attribute for Car

    // Specific method for Car
    void honk() {
        System.out.println("Honk! Honk!");
    }
}
```

Class Vs Sub Class Vs Super Class

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.year = 2020;
        myCar.start();
        myCar.stop();
        myCar.numberOfDoors = 4;
        myCar.honk();

        System.out.println("Car Color: " + myCar.color);
        System.out.println("Car Year: " + myCar.year);
        System.out.println("Number of Doors: "
        + myCar.numberOfDoors);
    }}
```

Access and Non-Access Specifiers

Access and Non-Access Specifiers

- Access Specifiers control the visibility of class members (fields, methods, constructors).
  - private: Accessible only within the same class.
  - protected: Accessible within the same package and subclasses.
  - public: Accessible from anywhere.
- Non-access Specifiers :
  - static: Belongs to the class rather than any instance.
  - final: Prevents modification (for variables) or inheritance/overriding (for classes/methods).

Access and Non-access Specifiers - Example

Example (Understanding)

```
class Person {
    private String name; // Private field
    protected int age;   // Protected field
    public void setName(String name) { // Public method
        this.name = name;
    }
    public String getName() {
        return name;
    }}
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("Java");
        System.out.println(person.getName());
    }}
```

What happens if you try to access a private field directly from another class?

- The private access specifier restricts the visibility of a field or method to within the same class . This means that only the class in which the private field or method is declared can access it.
- If you try to access a private field directly from another class, you will get a compile-time error .
- Private fields are meant to encapsulate data and prevent direct access from outside the class. They can only be accessed via public methods (getters and setters).

What happens if you try to access a private field directly from another class?

Example (Understanding)

```
class Person {
    private String name; // Private field
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

What happens if you try to access a private field directly from another class?

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("Premanand");
        System.out.println(person.getName());
    }
}
```

What is the difference between protected and default access specifiers?

- Default Access Specifier:
  - If no access specifier is specified, the member (field or method) has default access.
  - Default access means the member is accessible only within the same package .
  - Classes, methods, or fields with default access cannot be accessed from classes in other packages.
- Protected specifier:
  - The protected access specifier allows access to the member within the same package and also to subclasses (even if the subclass is in a different package).
  - This is useful when you want to allow inheritance-based access while still restricting access from unrelated classes in other packages.

What is the difference between protected and default access specifiers?

Example (Understanding)

```
// File: A.java
package pack1;

public class A {
    int defaultField = 10;        // Default access
    protected int protectedField = 20; // Protected access
}
```

What is the difference between protected and default access specifiers?

Example (Understanding)

```
// File: B.java
package pack1;

public class B {
    public void accessFields() {
        A obj = new A();
        System.out.println(obj.defaultField);
        // Accessible (same package)
        System.out.println(obj.protectedField);
        // Accessible (same package)
    }
}
```

What is the difference between protected and default access specifiers?

Example (Understanding)

```
// File: C.java
package pack2;

import pack1.A;

public class C extends A {
    public void accessFields() {
        // System.out.println(defaultField);
        // Not accessible (different package)
        System.out.println(protectedField);
        // Accessible (subclass in different package)
    }
}
```

What is Package?

- A package in Java is a way to group related classes and interfaces together to organize code and avoid name conflicts. It is similar to a folder in a file system.
- Packages help in:
  - Code organization – Makes it easier to manage large projects.
  - Access control – Restricts access using access specifiers.
  - Avoiding name conflicts – Different packages can have classes with the same name.
  - Reusability – Common functionalities can be placed in a package and reused.
- Types of Packages in Java
  - Built-in Packages – Provided by Java (e.g., java.util, java.io, java.net).
  - User-defined Packages – Created by developers to structure code better.

Declaring Objects and Ref

Declaring Objects and Assigning Object Reference Variables

- An object is an instance of a class. To create an object, use the new keyword.
- Reference variables hold the memory address of the object.

Declaring Objects and Assigning Object Reference Variables

Example (Understanding)

```
class Dog {
    String breed;
    void bark() {
        System.out.println("Woof!");
    }}
public class Main {
    public static void main(String[] args) {
        Dog dog1 = new Dog(); // Creating an object
        dog1.breed = "Labrador";
        System.out.println(dog1.breed);
        dog1.bark();
        Dog dog2 = dog1;
        dog2.breed = "Poodle";
        System.out.println(dog1.breed);    }}
```

What is the difference between assigning an object reference and creating a new object?

- Assigning an Object Reference:
  - When you assign an object reference, you are simply copying the memory address of an existing object to another variable.
  - Both variables will point to the same object in memory.
  - No new object is created; instead, both variables refer to the same instance.
- Creating a New Object:
  - When you create a new object using the new keyword, a new instance of the class is created in memory.
  - Each object has its own separate memory space, and changes made to one object do not affect the other.

What is the difference between assigning an object reference and creating a new object?

Example (Understanding)

```
class Dog {
    String name;

    Dog(String name) {
        this.name = name;
    }
}
```

What is the difference between assigning an object reference and creating a new object?

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Creating a new object
        Dog dog1 = new Dog("Buddy");
        // Assigning an object reference
        Dog dog2 = dog1;
        // Creating a new object
        Dog dog3 = new Dog("Max");
        System.out.println("dog1 name: " + dog1.name);
        System.out.println("dog2 name: " + dog2.name);
        System.out.println("dog3 name: " + dog3.name);
    }
}
```

What is the difference between assigning an object reference and creating a new object?

- If two reference variables point to the same object , any modification made to the object through one reference will be reflected when accessing the object through the other reference. This is because both variables are pointing to the same memory location.

What is the difference between assigning an object reference and creating a new object?

Example (Understanding)

```
class Dog {
    String name;

    Dog(String name) {
        this.name = name;
    }
}
```

What is the difference between assigning an object reference and creating a new object?

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Dog dog1 = new Dog("Buddy");

        Dog dog2 = dog1;

        dog2.name = "Charlie";

        System.out.println("dog1 name: " + dog1.name);
        System.out.println("dog2 name: " + dog2.name);
    }
}
```

Questions to understand the concept!

- Write a class Rectangle with attributes length and width. Add methods to calculate the area and perimeter of the rectangle. Create an object of this class and display its area and perimeter.
- Create a class Student with attributes name, rollNumber, and grade. Write a method displayInfo() to print the details of the student. Create two objects of this class and call the displayInfo() method for each object.
- Write a program where a class Car has attributes color and model. Initialize these attributes using a parameterized constructor and display the car's details.

Questions to understand the concept!

- 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.
- 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.
- 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.
- 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.

Questions to understand the concept!

- 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.
- 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.
- 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.
- 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.

Questions to understand the concept!

- 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?

Questions to understand the concept!

- 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.
- 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.
- 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).
- 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.