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 24, 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

Array of objects

Array of Objects

- You can create an array of objects just like primitive types. Each element in the array holds a reference to an object.

Array of Objects - Example

Example (Understanding)

```
class Family {
    String name;
    Family(String name) {
        this.name = name;
    }}
public class Main {
    public static void main(String[] args) {
        Family[] families = new Family[4];
        families[0] = new Family("Premanand");
        families[1] = new Family("Santhalakshmi");
        families[2] = new Family("Nikhilesh");
        families[3] = new Family("Krithiksha");

        for (Family family : families) {
            System.out.println(family.name);}    }}
```

How do you initialize an array of objects?

- An array of objects is an array where each element is a reference to an object. To initialize an array of objects, you need to:
  - Declare the array : Specify the type of objects the array will hold.
  - Instantiate the array : Allocate memory for the array using the new keyword.
  - Initialize each object : Assign individual objects to the elements of the array.
- Key Points About Initializing Arrays of Objects,
  - Declaring and instantiating the array only allocates memory for the references. The actual objects are not created until you explicitly instantiate them.
  - If you skip initializing an element, it remains null, and attempting to access it will result in a NullPointerException

How do you initialize an array of objects?

Example (Understanding)

```
class Student {
    String name;
    int rollNumber;

    // Constructor
    Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }

    void displayInfo() {
        System.out.println("Name: " + name + ",
        Roll Number: " + rollNumber);
    }
}
```

How do you initialize an array of objects?

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        // Step 1: Declare the array
        Student[] students;
        // Step 2: Instantiate the array
        students = new Student[3];
        // Step 3: Initialize each object
        students[0] = new Student("NEHA M", 1011);
        students[1] = new Student("S SANJAY", 1019);
        students[2] = new Student("JAVEED AKTHAAR S", 1022);
        // Access and display information
        for (Student student : students) {
            student.displayInfo();
        }    }}
```

What happens if you try to access an uninitialized element in an array of objects?

- If you try to access an uninitialized element in an array of objects, the program will throw a NullPointerException at runtime.
- This happens because the uninitialized elements in the array are set to null by default, and calling a method or accessing a field on a null reference results in an exception.

What happens if you try to access an uninitialized element in an array of objects?

Example (Understanding)

```
class Car {
    String model;

    Car(String model) {
        this.model = model;
    }

    void displayModel() {
        System.out.println("Model: " + model);
    }
}
```

What happens if you try to access an uninitialized element in an array of objects?

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Car[] cars = new Car[3];
        cars[0] = new Car("Toyota");
        try {
            cars[1].displayModel();
        } catch (NullPointerException e) {
            System.out.println("Error: Attempted
            to access an uninitialized element.");
        }
    }
}
```

Constructors and Destructors

Constructors

- A special method used to initialize an object when it is created.
- It has the same name as the class and does not have a return type, not even void. Constructors are automatically called when an object is instantiated using the new keyword.
- Characteristics,
  - Same Name as the Class : The constructor must have the same name as the class.
  - No Return Type : Unlike regular methods, constructors do not have a return type.
  - Automatically Called : A constructor is automatically invoked when an object is created using new.
  - Can Be Overloaded : You can define multiple constructors with different parameter lists (constructor overloading).
  - Default Constructor : If no constructor is explicitly defined, Java provides a default constructor (a no-argument constructor) automatically. However, if you define any constructor, the default constructor is not provided.

Default Constructor - Type

Example (Understanding)

```
class Car {
    String color;
    // Default Constructor
    Car() {
        color = "Unknown";
    }}
public class Main {
    public static void main(String[] args) {
        // Calls the default constructor
        Car myCar = new Car();
        System.out.println("Car Color: "
        + myCar.color);
    }}
```

Parametrized Constructor - Type

Example (Understanding)

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

Constructor Overloading

- You can define multiple constructors in a class with different parameter lists.

Example (Understanding)

```
class Rectangle {
    int length;
    int width;
    Rectangle() {
        length = 0;
        width = 0;
    }
```

Constructor Overloading

Example (Understanding)

```
    Rectangle(int side) {
        length = width = side;
    }
    Rectangle(int length, int width) {
        this.length = length;
        this.width = width;
    }

    void displayArea() {
        System.out.println("Area: " +
        (length * width));
    }
}
```

Constructor Overloading

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        Rectangle rect1 = new Rectangle();
        rect1.displayArea();

        Rectangle rect2 = new Rectangle(5);
        rect2.displayArea();

        Rectangle rect3 = new Rectangle(4, 6);
        rect3.displayArea();
    }
}
```

Using this Keyword in Constructors

- The this keyword refers to the current object.
- It is often used in constructors to differentiate between instance variables and parameters with the same name.

Using this Keyword in Constructors

Example (Understanding)

```
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }}
public class Main {
    public static void main(String[] args) {
        Person person = new Person("Premanand", 38);
        person.displayInfo();
    }}
```

Copy Constructors

- Initializes an object using another object of the same class.

Copy Constructors

Example (Understanding)

```
class Point {
    int x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    Point(Point other) {
        this.x = other.x;
        this.y = other.y;
    }
```

Copy Constructors

Example (Understanding)

```
    void display() {
        System.out.println("Point (" + x + ", " + y + ")");
    }
}
public class Main {
    public static void main(String[] args) {
        Point p1 = new Point(3, 4);
        Point p2 = new Point(p1);

        p1.display();
        p2.display();
    }
}
```

Chaining Constructors (Using this())

- You can call one constructor from another constructor within the same class using the this() keyword.

Chaining Constructors (Using this())

Example (Understanding)

```
class Box {
    int length, width, height;
    Box() {
        this(0, 0, 0);
    }
    Box(int side) {
        this(side, side, side);
    }
    Box(int length, int width, int height) {
        this.length = length;
        this.width = width;
        this.height = height;
    }
```

Chaining Constructors (Using this())

Example (Understanding)

```
    void displayVolume() {
        System.out.println("Volume: " + (length * width * heig
    }
}
public class Main {
    public static void main(String[] args) {
        Box box1 = new Box();
        box1.displayVolume();
        Box box2 = new Box(5);
        box2.displayVolume();
        Box box3 = new Box(4, 5, 6);
        box3.displayVolume();
    }
}
```

Key Points About Constructors

- Default Constructor : Automatically provided by Java if no constructor is defined.
- Parameterized Constructor : Used to initialize objects with specific values.
- Constructor Overloading : Multiple constructors with different parameter lists.
- this Keyword : Used to refer to the current object and resolve naming conflicts.
- Copy Constructor : Initializes an object using another object of the same class.
- Constructor Chaining : Use this() to call one constructor from another within the same class.

Destructors

- A destructor is a special method that is automatically called when an object is destroyed (e.g., when it goes out of scope or is explicitly deleted).
- Its purpose is to clean up resources, such as closing files, releasing memory, or freeing other system resources.
- However, Java does not have destructors in the same way C++ does. Instead, Java uses a mechanism called garbage collection to manage memory and resource cleanup.

Why Doesn't Java Have Destructors?

- Automatic Garbage Collection
  - Memory management is handled by the garbage collector (GC) .
  - The garbage collector automatically reclaims memory occupied by objects that are no longer in use.
  - This eliminates the need for explicit destructors to free memory.
- No manual memory management
  - Unlike C++, where you manually allocate and deallocate memory using new and delete, Java handles memory allocation and deallocation automatically.

How Does Java Handle Resource Cleanup?

- For managing resources like files, sockets, or database connections, Java provides the try-with-resources statement. This ensures that resources are closed automatically when they are no longer needed.

How Does Java Handle Resource Cleanup? - try with resources

Example (Understanding)

```
import java.io.FileWriter;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try (FileWriter writer = new
        FileWriter("example.txt")) {
            writer.write("Hello, Java!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }}
```

How Does Java Handle Resource Cleanup? - close()

Example (Understanding)

```
class Resource {
    void close() {
        System.out.println("Resource closed");
    }}
public class Main {
    public static void main(String[] args) {
        Resource res = new Resource();
        try {
            // Use resource
        } finally {
            res.close(); // Explicit cleanup
        }   } }
```