

Unit 3: Object Oriented Programming

Classes and Objects: Object Creation, Initializing the Instance Variables

<https://sawin.com.np>

<https://sawin.com.np>

References:

<https://java-tutorial.dev/docs/Java-Object-Oriented-Programming/Classes-And-Objects-In-Java>

<https://java-tutorial.dev/docs/Java-Object-Oriented-Programming/Object-Creation-In-Java>

<https://sawin.com.np>

Define a Class

A class acts as a template that defines the properties (instance variables) and functionalities (methods) of objects.



```
public class Car {  
    // Instance variables (attributes) to describe a car  
    int modelYear;  
    String make;  
    String color;  
}
```

Object Creation

An object is an instance of a class.

You use the new keyword followed by the class name to create an object.

Example:

- `Car myCar = new Car();`

Initialize Instance Variables

Once you have an object, you need to assign values to its instance variables. You can do this in two ways:

- Direct Assignment
- Using a Constructor

Direct Assignment

<https://sawin.com.np>

<https://sawin.com.np>



```
Car myCar = new Car();  
myCar.modelYear = 2023;  
myCar.make = "Honda";  
myCar.color = "Red";
```

<https://sawin.com.np>

Using Constructor



```
// Creating an object using the constructor  
Car myCar = new Car(2023, "Honda", "Red");
```

Access Specifiers

In Java, access specifiers are keywords that define the visibility and accessibility of classes, methods, variables, and constructors within your program. They play a crucial role in encapsulation, a core principle of Object-Oriented Programming (OOP).

Types of Access Specifiers in Java:

- Public
- Private
- Protected
- Default

References: <https://www.javatpoint.com/access-modifiers>

- Public: Members declared as public are accessible from anywhere in your program, including other classes within the same package or different packages. This is the most permissive access level.
- Private: Members declared as private are only accessible within the class they are defined in. Other classes cannot directly access private members. This is the most restrictive access level and promotes data hiding.
- Protected: Members declared as protected are accessible within the class they are defined in, as well as in subclasses (classes that inherit from the current class) regardless of the package they reside in. This provides controlled inheritance access.
- Default: If you don't explicitly declare an access specifier for a member, it becomes package-private. This means it's accessible from all classes within the same package but not from outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Choosing the Right Access Specifier:

The appropriate access specifier depends on the intended use of the member and your program's design:

- Public: Use public for members that need to be accessed from other classes.
- Private: Use private for data members (attributes) to enforce data hiding and controlled access.
- Protected: Use protected for members that should be inherited by subclasses but not directly accessed by other classes.
- Default: Use default for members that only need to be used within the same package, promoting modularity within a package.

By effectively using access specifiers, you can achieve:

- Data Protection: Restrict direct modification of sensitive data.
- Improved Maintainability: Enhance code clarity and organization.
- Controlled Inheritance: Define which members subclasses can access.
- Modular Design: Promote code reusability within packages.

Understanding access specifiers is essential for writing secure, maintainable, and well-structured object-oriented programs in Java.

Encapsulation

Encapsulation is one of the four fundamental Object-Oriented Programming (OOP) concepts, along with inheritance, polymorphism, and abstraction.

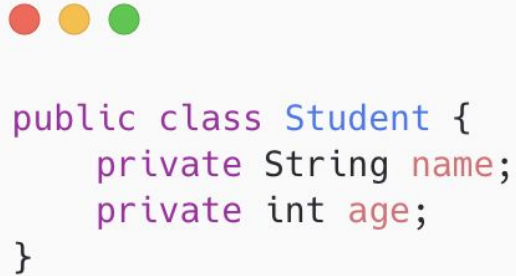
It is a crucial concept in Java, and it involves bundling an object's state (fields) and behavior (methods) into a single unit known as a class.

Encapsulation restricts direct access to some of an object's components, providing better control over data integrity and preventing unintended modifications.

- References:
 - <https://java-tutorial.dev/docs/Java-Object-Oriented-Programming/Encapsulation-In-Java>
 - <https://www.javatpoint.com/encapsulation>
 - <https://www.youtube.com/watch?v=YbqneqDIZh8>

Implementing Encapsulation in java

Private Fields in java: Fields within a class should typically be declared as private to prevent direct external access.



```
public class Student {  
    private String name;  
    private int age;  
}
```

Public Methods in java (Getters and Setters)

Access to the private fields is provided through public methods (getters and setters) within the class.

By providing only a getter method or a setter method for a field, you can create read-only or write-only properties, respectively.

Encapsulation allows you to add validation logic to setter methods, ensuring that the data remains consistent and adheres to specified rules.



```
public class Student {  
    private String name;  
    private int age;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        if (age >= 0) { // Validation  
            this.age = age;  
        }  
    }  
}
```

<https://sawin.com.np>

<https://sawin.com.np>

<https://sawin.com.np>

Constructors

- Constructors in Java are special methods within a class that are used for initializing objects.
- They are called when an object of a class is created using the new keyword.
- Constructors have the same name as the class and may have parameters to set initial values for object attributes.

References:

<https://java-tutorial.dev/docs/Java-Object-Oriented-Programming/Constructors-In-Java>

<https://www.javatpoint.com/java-constructor>

Rules for creating Java constructor

There are three rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor): If you don't provide any constructor in your class, Java automatically provides a default constructor with no parameters. It initializes the object with default values for fields.
2. Parameterized constructor: You can create custom constructors with parameters to set initial values for object attributes. These constructors allow you to create objects with specific data.

Example of Default Constructor



```
class Car {  
    String make;  
    int year;  
}
```

// Default constructor (provided by Java):

```
Car myCar = new Car(); // Initializes 'make' to null and 'year' to 0
```

Example of Parameterized Constructors



```
class Car {  
    String make;  
    int year;  
  
    // Parameterized constructor  
    Car(String make, int year) {  
        this.make = make;  
        this.year = year;  
    }  
}  
  
Car myCar = new Car("Toyota", 2022);
```

Constructor Overloading

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are arranged in a way that each constructor performs a different task.

They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading



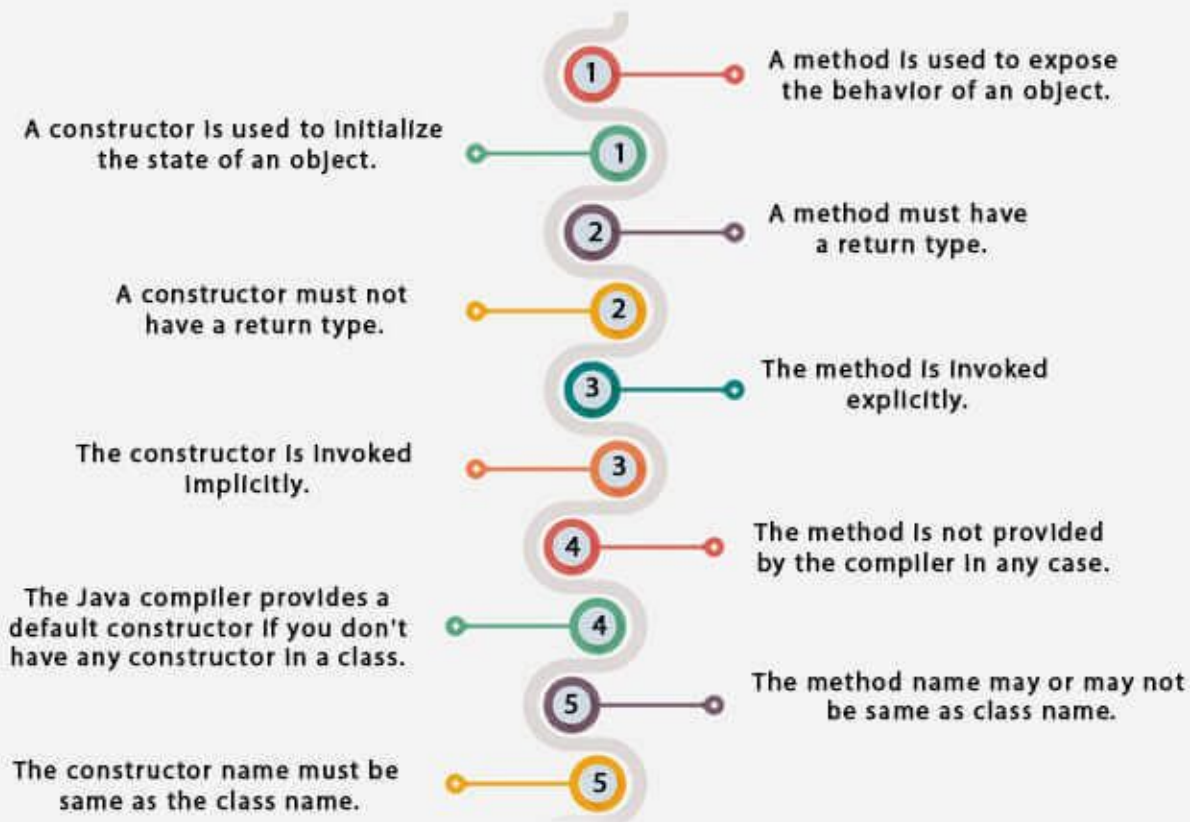
```
class Car {  
    String make;  
    int year;  
  
    Car(String make, int year) {  
        this.make = make;  
        this.year = year;  
    }  
  
    // Overloaded constructor with a different parameter list  
    Car(String make) {  
        this.make = make;  
        this.year = 0; // Default year value  
    }  
}  
  
Car car1 = new Car("Toyota", 2022); // Calls the first constructor  
Car car2 = new Car("Honda");        // Calls the second constructor
```

DIY

WAP to create a class Student (name, age, rollNo) and initialize it with data members using constructor and display the record.

Difference between constructor and method in Java

sawin.com.np



sawin.com.np

Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is the process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java, it is performed automatically. So, java provides better memory management.

References: <https://www.javatpoint.com/Garbage-Collection>

<https://www.geeksforgeeks.org/garbage-collection-java/>

this Keyword

Inside constructors, you can use the this keyword to refer to the current object being created.

This is helpful when the constructor parameters have the same names as the object's fields.

References: <https://www.javatpoint.com/this-keyword>

<https://geeksforgeeks.org/this-reference-in-java/>

Example of this keyword



```
class Person {  
    String name;  
    int age;  
  
    Person(String name, int age) {  
        this.name = name; // 'this' refers to the current object  
        this.age = age;  
    }  
}
```

Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01

this can be used to refer current class instance variable.

04

this can be passed as an argument in the method call.

02

this can be used to invoke current class method (implicity)

05

this can be passed as argument in the constructor call.

03

this() can be used to invoke current class Constructor.

06

this can be used to return the current class instance from the method

Static Members

The static keyword in Java is used for memory management mainly.

We can apply static keyword with variables, methods, blocks and nested classes.

The static keyword belongs to the class than an instance of the class.

It makes your program memory efficient (i.e., it saves memory).

Java static variable; Java static method; Java static block

References: <https://www.javatpoint.com/static-keyword-in-java>



```
class Student{  
    int rollno;  
    String name;  
    String college="ITS";  
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.



```
//Java Program to demonstrate the use of static variable
class Student{
    int rollno;//instance variable
    String name;
    static String college ="ITS";//static variable
    //constructor
    Student(int r, String n){
        rollno = r;
        name = n;
    }
    //method to display the values
    void display (){System.out.println(rollno+" "+name+" "+college);}
}

//Test class to show the values of objects
public class TestStaticVariable1{
    public static void main(String args[]){
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        //we can change the college of all objects by the single line of code
        //Student.college="BBDIT";
        s1.display();
        s2.display();
    }
}
```

Why is the Java main method static?

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Practical

1. Write a Java program to create a class called Vehicle with attributes for color, model, and year. Create an object of the class and display its attributes.
2. Create a Java class called Person with attributes for name, age, and address. Write a method to display the person's details.
3. Define a Java class called Rectangle with attributes for length and width. Write a method to calculate and return the area of the rectangle.
4. Create a Java class called BankAccount with attributes for accountNumber, balance, and accountHolderName. Write a constructor to initialize the attributes and a method to deposit money into the account.
5. Write a Java program to create a class called Employee with attributes for employeeId, name, and salary. Use access specifiers (public, private, protected) to restrict access to the attributes.
6. Define a Java class called Circle with an attribute for radius. Write a method to calculate and return the area of the circle. Use encapsulation to hide the implementation details.
7. Create a Java class called Student with attributes for studentId, name, and grades. Write a method to calculate and return the average grade.
8. Write a Java program to demonstrate garbage collection.
9. Write a Java program to create a class called Car with attributes for make, model, and year. Use the this pointer to access the attributes in a method.
10. Define a Java class called Book with attributes for title, author, and price. Write a method to display the book's details and use static members to count the number of books created.
11. Create a Java class called Hotel with attributes for hotelName, rooms, and roomRates. Write a method to calculate and return the total revenue of the hotel.