Day_4_OOPJ_Saket_Shalukar

Friday, August 29, 2025 10:09 AM

```
4. Access modifiers can be different but not playing any role in distinguishing

Ex: Not overloading

int sum(int a, int b, int c){
    return a+b+c;
}

double sum(double a, double b){
    return a+b; //CE :
}


Constructor:
------------------
-special method in java
-used to initalize objects
-It is automatically called when an object is created.
-mainly used to set intial values for instance variables.

Features:
------------
-name has to be same as class
```

```java
class Demo {
    int m = 25;
}
---
class InstanceDemo {
    int x = 10;
    int y = 20;
    void display() {
        System.out.println("Display call kiya hai !");
    }
    public static void main(String args[]) {
        int z = 100;
        InstanceDemo i1 = new InstanceDemo();
        System.out.println(i1.x);
        System.out.println(i1.y);
        System.out.println(z);
        Demo d1 = new Demo();
    }
}
```

Why Static Executes First?

Static members of a class, including **static blocks** and **static variables**, are initialized and executed when the class is loaded into the Java Virtual Machine (JVM), which happens before any objects of that class are created. This is why the `main` method, which is also static, can be called without creating an instance of the class it belongs to.

How it Works

When the JVM starts, it needs a starting point, which is the **`main` method**. The `main` method must be accessible without creating an object, so it's declared as `static`. Here's a step-by-step breakdown:

- **Class Loading:** The JVM finds and loads the class file containing the `main` method.
- **Static Initialization:** During the loading process, the JVM initializes all static members. This includes:
- **Static variables:** They are allocated memory and given their default or assigned values.
- **Static blocks:** Any code inside a `static { ... }` block is executed. This is a common way to perform one-time setup for a class.
- **`main` method execution:** After all static members are initialized, the JVM can then call the `main` method. Since the `main` method is static, it belongs to the class itself, not to an object, and is therefore available immediately after the class is loaded

Driver class :

A driver class in Java is a class that contains the `main` method. It acts as the **entry point** for the program, initiating the execution flow. Its primary purpose is to **test and use** the other classes in the application by creating objects and calling their methods.

```java
class Employee{
    private int empid;
    private String empname;

    void setdata(){

    }

    void getdata(){
    }


}
 //Driver class
class EmployeeDemo{


    public static void main(String args[]){
```

DEmo
fields
methods

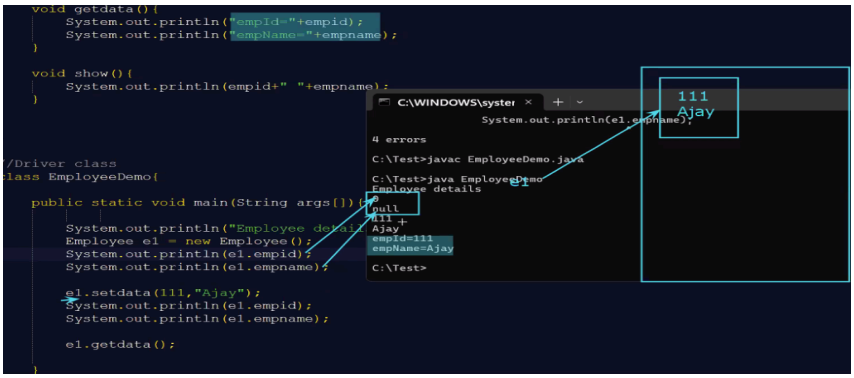
DemoApp
main()

Getter Methods (Accessors)

- **Purpose:** To retrieve or "get" the value of a private variable. They provide read-only access to the data.
- **Naming Convention:** By convention, a getter method's name starts with `get`, followed by the variable name with its first letter capitalized (e.g., `getName()` for a `name` variable).
- **Functionality:** It typically returns the value of the corresponding private variable without modifying it.

Setter Methods (Mutators)

- **Purpose:** To set or "mutate" the value of a private variable. They provide controlled write access to the data.
- **Naming Convention:** By convention, a setter method's name starts with `set`, followed by the variable name with its first letter capitalized (e.g., `setName()` for a `name` variable).
- **Functionality:** It takes a parameter and assigns that value to the corresponding private variable. A key advantage is that you can add **validation logic** inside a setter. For example, before setting an age, you can check if the value is positive. This prevents invalid data from being assigned to the object's state.



constructor :

A **constructor** is a special method in a class that's automatically called when you create an object of that class. Its main job is to **initialize the object's state**, meaning it sets the starting values for the object's variables.

Key Characteristics

- **Same Name as the Class**: A constructor must have the exact same name as the class it belongs to.
- **No Return Type**: Unlike regular methods, a constructor does not have a return type, not even `void`.
- **Automatic Invocation**: You can't call a constructor directly. It is automatically invoked when you use the `new` keyword to create an object.
-

```
int age;
double cost;
Customer()
{
    cost = 100;
}
//Use of this in a constructoe : used to differentiate between parameteras
instance variable
Customer(String name, int age, double cost){
    this.name = name;//this.name : refere to instance variable
    this.age = age;//age : parameter
    this.cost = cost;
}
Customer(String name, int age){
    this.name = name;
    this.age = age;

}

void display(){
    System.out.println("Name="+name+"Age="+age+"Cost="+cost);
}
```

What a Constructor Does

- **Initializes Variables**: A constructor takes data (often passed as parameters) and assigns it to the object's fields. This is how you set up the object with its starting properties. For example, a `Car` constructor might set the `color` and `make` of the new car object.
- **Ensures a Valid State**: By forcing you to provide certain data at creation, a constructor ensures the object is always in a valid, usable state from the moment it's created. This prevents bugs that could arise from using an uninitialized or partially initialized object.
- **Performs Initial Setup**: A constructor can also contain logic for other initial setup tasks, like opening a file, creating a database connection, or setting up other related objects.

Constructor chaining : is the process of calling one constructor from another constructor within the same class or from a base class. It's used to avoid duplicating initialization code and to ensure that all parts of an object, including inherited parts, are properly set up.