**Constructor**

**🧱 What is a Constructor?**

A constructor is a special method that gets called when an object is created.

**Its job?**

To initialize the object.

**What is a Constructor really doing?**

When you do: Car c = new Car();

Three things happen:

* Memory is allocated on the heap for the object.
* Java calls the appropriate constructor (default or parameterized).
* Fields inside the object are initialized.

Constructor != method. It does not return anything, not even void.

**Key Points:**

* Name must match the class name
* No return type (not even void)
* Called automatically when an object is created using new
* Can be overloaded (multiple constructors in a class)

**Types of Constructors:**

1. **Default Constructor** (provided by Java if you don’t write one)
2. **No-arg Constructor** (you write it with no parameters)
3. **Parameterized Constructor**

🔧 Examples:

1️.**Default** constructor (Java provides it if none is written): class Car { int speed; }

Car c = new Car(); // Default constructor is called

If you write no constructor at all, Java gives you one.

**Example**:

class Student {

int id;

String name;

}

Student s = new Student(); // Default constructor used

**Use Case:** When you want to create a basic object and set values later using setters.

2️.No-arg constructor (you define it yourself): class Car

{ Car() { System.out.println("Car created"); } }

You write a constructor with no parameters.

**Example**:

class Student {

Student()

{

System.out.println("Student created");

}

}

Student s = new Student();

**Use Case**: If you want to log something or initialize default values on object creation.

3️.**Parameterized constructor**:  
You write a constructor that takes parameters.

Example:

class Student

{

String name;

Student(String n)

{

name = n;

}

}

Student s = new Student("Tanvi");

**Use Case:** To initialize values at the time of object creation — saves code and reduces errors.

**Constructor Overloading**  
Multiple constructors with different parameter types or count.

**Example**:

class Book

{

Book()

{

System.out.println("Default");

}

Book(String name)

{

System.out.println("Book name: " + name);

}

}

Book b1 = new Book();

Book b2 = new Book("Java");

**Use Case**: When you want flexibility — create an object in different ways.

**Constructor Chaining in Java**

Constructor Chaining is when one constructor calls another constructor in the same class or parent class**.**

**There are two types:**

* 1. Within same class → this()
  2. From parent class → super()
* Calling constructor with another constructor
* this() and super() should be always first statement
* super() inheritance is mandatory
* super() and this() cannot be written in same constructor

**Example:**

Car() {

this("Maruti", 500000); // calls constructor with 2 params

System.out.println("Default constructor");

}

Car(String brand, int price) {

this(brand, price, "Petrol"); // calls constructor with 3 params

System.out.println("2-arg constructor");

}

Car(String brand, int price, String fuel) {

this.brand = brand;

this.price = price;

this.fuel = fuel;

System.out.println("3-arg constructor");

}

void show() {

System.out.println(brand + " - " + price + " - " + fuel);

}

**Super():**

Class Demo

{

Demo(){

s.o.p(“demo”);

}

}

Public class test{

Psvm(String arg[])

{

Test()

{

Super();

S.O.P(“Test constructor”);

}

}

}

o/p:

demo

test constructor

**Copy Constructor in Java**

Java doesn’t provide it by default, but you can create one manually.

It is used to create a new object that is a copy of an existing object.

**Example**: Copy Constructor

class Student

{

String name;

int age;

Student(String name, int age) {

this.name = name;

this.age = age;

}

// Copy constructor

Student(Student s) {

this.name = s.name;

this.age = s.age;

}

void display() {

System.out.println(name + " - " + age);

}

🧠 **Tricky Constructor Questions with Answers**

🔹 **Q1: Can a constructor be private? Why would you use it?**

✅ **Answer**: Yes, a constructor can be private. It is commonly used in **Singleton** patterns to prevent instantiation from outside the class.

**Use case**: Ensures only one instance of Singleton is created in the app.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q2: What happens if you don’t define any constructor in your class?**

✅ **Answer**: Java provides a default no-argument constructor automatically. But if you define any constructor (even with parameters), Java won’t generate the default one.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q3: Can you call this() and super() in the same constructor?**

✅ Answer: No. It leads to a compilation error. Only one of this() or super() can be used, and it must be the first statement in the constructor.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q4: What’s the output?**

class Test { int x;

Test() {

this(10); // calls constructor with 1 argument

System.out.println("No-arg constructor");

}

Test(int x) {

System.out.println("One-arg constructor");

}

public static void main(String[] args) {

Test t = new Test();

}

}

✅ **Output**: One-arg constructor  
No-arg constructor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

🔹 Q5: What’s the output?

class Test {

Test()

{

System.out.println("Constructor");

return; // valid

}

void Test() {

System.out.println("This is NOT a constructor");

}

public static void main(String[] args) {

Test t = new Test(); // constructor is called

t.Test(); // this is a method

}

}

✅ **Output**: Constructor  
This is NOT a constructor

🧠 **Trick**: A method with the same name as class but with return type (even void) is just a method, not a constructor.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q6: Can a constructor be final, static, or abstract?**

✅ **Answer**: No. Constructors cannot be:

* final → because it can’t be inherited or overridden anyway.
* static → because they belong to an instance, not the class.
* abstract → because constructors can’t be abstract (they must be implemented)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**7: What happens if you write return inside a constructor?**

✅ **Answer**: You can use return; (no value) in a constructor. But return <value>; gives a compile-time error because constructors do not return anything.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q8: **Can constructor be overloaded?**

✅ Answer: Yes. Constructor overloading is supported.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q9**: Can constructor call a method?**

✅ Answer: Yes, a constructor can call methods — just like regular code.

But beware: if you call an overridden method inside the constructor, and it uses subclass state, it may behave unexpectedly because the subclass isn't fully constructed yet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q10**: Can we overload constructors and also use inheritance?**

✅ Answer: Yes. You can overload constructors in a subclass and use super() to call parent constructors. You must handle which constructor to call using super().