Java Constructors – Complete In-Depth Guide

---

1. What is a Constructor?

A **constructor** is a **specialized setter** whose **name is same as the class name** and **does not have any explicit return type**.

✔ It is executed **automatically during object creation**.
✔ It is mainly used to **initialize instance variables**.

"It is a specialized setter whose name is same as class name and doesn't have any explicit return type"

---

2. Why Constructors Are Needed

- To initialize object data at creation time
- To avoid uninitialized or default values
- To enforce compulsory data while creating objects
- To make object creation meaningful and safe

Without constructor → object exists but may be useless.

---

3. When Is Constructor Invoked?

A constructor is invoked:

- When object is created using `new` keyword

```
Dog d = new Dog();
```

Constructor is **NOT called manually**.

4. Default Constructor (Very Important)

Case 1: No constructor written by programmer

If **developer does NOT include any constructor**, then:

✔ Java compiler **automatically provides** a default zero-parameter constructor.

```
public ClassName() {
}
```

Case 2: At least one constructor is written

If **any constructor is coded**, then:

 Java compiler **will NOT include default constructor**.

"If there is a constructor coded in a class then java compiler would not include any default constructor"

---

5. Parameterized Constructor

A constructor that accepts parameters is called a **parameterized constructor**.

Example (From Your Notes – Dog Class):

```
class Dog {
    private int cost;
    private String name;
    private String color;
.
.
.
.
.
.}

    public Dog(int cost, String name, String color) {
        this.cost = cost;
        this.name = name;
        this.color = color;
    }
}
```

Key Points:

- Used to pass values at object creation

- Acts like a setter but executes only once

---

6. Constructor vs Setter (Your Finisher Example Concept)

Constructor:

- Executes only **once**
- Used at **object creation**
- Initializes object completely

Setter:

- Can be called **multiple times**
- Used to **modify values after object creation**

Example Concept from Your Notes:

```
Finisher f1 = new Finisher(2, "Rohit", "Mumbai");

// Later changes
f1.setId(3);
f1.setName("Nanda");
f1.setCity("Ongole");
```

 Constructor sets **initial state**
 Setters change state later

---

7. Constructor Overloading

Definition:

Having **multiple constructors** in a class with:

- Same name (class name)
- Different parameters

This is called **Constructor Overloading**.

"we can have multiple constructors in a class creating multiple constructor with same name and different parameters is referred as constructor overloading"

Example:

```
class Demo {
    int num1, num2;

    Demo() {
        System.out.println("Zero param constructor");
    }

    Demo(int num1) {
        this.num1 = num1;
        this.num2 = 99;
        System.out.println("1 param constructor");
    }

    Demo(int num1, int num2) {
        this.num1 = num1;
        this.num2 = num2;
        System.out.println("2 param constructor");
    }
}
```

---

8. `this` Keyword (Very Important – Shadowing Solution)

Meaning:

`this` stores the **address of currently running object / instance**.

Used to:

- Differentiate local and instance variables
- Assign data from local variable to instance variable
- Solve **shadowing problem**

"this keyword will have address of currently running instance/object"

Shadowing Example:

```
Dog(int cost) {
    this.cost = cost;
}
```

9. `this()` – Constructor Chaining

Meaning:

`this()` is used to invoke **another constructor of same class**.

Rules:

- Must be **first statement**
- Calls **zero or parameterized constructor**

"this keyword invokes zero parameterized constructor of the class"

---

10. `super()` Keyword (Hidden but Important)

Meaning:

- Used to invoke **parent class constructor**
- Present **behind the code by default**

"for every constructor the first line is super() is present behind the code"

Key Rules:

- `super()` must be first statement
- If not written, compiler inserts it automatically

---

11. Order of Constructor Execution

When child object is created:

1. Parent class constructor executes
2. Child class constructor executes

 Ensures parent data is initialized first

12. Error Case (Zero Parameter Constructor Issue)

If:

- You define **only parameterized constructor**
- And try to create object using zero parameter constructor

 Compile-time error occurs

"If one parameterized constructor exists in code if we want to create one more using zero parameter with same class we got error"

---

13. Constructor Is NOT a Method

Constructor	Method
Same name as class	Any valid name
No return type	Has return type
Invoked automatically	Called explicitly

---

14. Important Properties (Exam Focus)

- Constructor is **not inherited**
- Constructor **cannot be overridden**
- Constructor **can be overloaded**
- Constructor runs **only once per object**
- Every class has **at least one constructor**

---

15. Real-World Analogy (Your Learning Style)

- Aadhaar enrollment → constructor
- Mobile setup → constructor
- Bank account creation → constructor

Setters = profile updates later

---

16. Final Summary

- Constructor initializes object
- Executes during object creation
- Default constructor exists only if none is written
- `this` → current object
- `super` → parent constructor
- Overloading is allowed, overriding is not

EXTRA CONSTRUCTOR TOPICS (ADVANCED + INTERVIEW + REAL-WORLD)

17. Instance Variable Initialization Order (Very Important Internals)

When an object is created, Java follows this **exact order**:

1. Memory allocation in heap
2. Instance variables get **default values**
3. Parent class instance variables initialized
4. Parent class constructor executed
5. Child class instance variables initialized
6. Child class constructor executed

Example:

```
class A {
    int x = 10;
    A() {
        System.out.println("A constructor");
    }
}

class B extends A {
    int y = 20;
    B() {
        System.out.println("B constructor");
    }
}
```

 Output order proves constructor execution flow.

18. Constructor Is NOT Inherited (Clarified)

Even though child class object contains parent data,

 Constructors are NOT inherited

✔ Parent constructor is executed using `super()`

This is why we cannot call parent constructor like a normal method.

---

19. Why Constructor Cannot Be `static`

Reason:

- `static` belongs to class
- Constructor belongs to **object creation**

If constructor were static:
 No object reference (`this`) would exist
 Object initialization concept breaks

Hence, constructor cannot be static.

---

20. Why Constructor Cannot Be `final`

`final` means cannot be overridden.

But:

- Constructors are **never overridden**
- So `final` has **no meaning** for constructors

Therefore Java does not allow it.

---

21. Why Constructor Cannot Be `abstract`

Abstract means:

- Incomplete
- Needs subclass implementation

Constructor job = **complete object initialization**

So constructor **must have body** → cannot be abstract.

---

22. Access Modifiers in Constructors

Constructors can be:

- `public`
- `protected`
- `default`
- `private`

Example Uses:

- `public` → normal object creation
- `protected` → inheritance control
- `private` → singleton / factory pattern

---

23. Private Constructor (Detailed Use Case)

Why private constructor?

- To **restrict object creation**
- To control instances

Example:

```
class Database {
    private Database() {}
}
```

Used in:

- Singleton class
- Utility classes

---

24. Singleton Class Using Constructor

```
class Singleton {
    private static Singleton obj;

    private Singleton() {}

    public static Singleton getInstance() {
        if (obj == null) {
            obj = new Singleton();
        }
        return obj;
    }
}
```

 Constructor access control concept.

---

25. Constructor vs Instance Initialization Block (IIB)

Instance Block:

```
{
    System.out.println("Instance block");
}
```

Execution Order:

1. Instance block
2. Constructor

Used when:

- Common initialization for all constructors

---

26. Constructor and Memory (Heap + Stack)

- Object stored in **heap**
- Reference stored in **stack**
- Constructor initializes heap memory

```
Demo d = new Demo();
```

`d` → stack
Object → heap

---

27. Copy Constructor (Java Style)

Java does not provide copy constructor automatically.

We create it manually:

```
Dog(Dog d) {
    this.cost = d.cost;
    this.name = d.name;
    this.color = d.color;
}
```

Used to:

- Clone object data safely

---

28. Constructor and Garbage Collection

- Constructor → object birth
- Garbage Collector → object death

Constructor initializes
GC destroys unused objects

---

29. Constructor Overloading vs Method Overloading

Constructor Overloading	Method Overloading
Same class name	Same method name
No return type	Has return type
Object creation	Normal behavior

---

30. Common Interview Trap Questions □

1. Can constructor be overridden? □ No
2. Can constructor be inherited? □ No
3. Can constructor be overloaded? □ Yes
4. Can constructor be private? □ Yes
5. Is default constructor always provided? □ No

---

32. Final Master Summary □

Constructor is the **foundation of object-oriented programming**.

- Controls object birth
- Ensures valid initialization
- Works with `this` and `super`
- Supports overloading
- Plays key role in memory management