**Constructors in Java**

A **constructor** in Java is a special method used to **initialize objects**. It is called automatically when an object of a class is created. The constructor has the same name as the class and **does not have a return type (not even void)**.

**1. Features of Constructors**

* **Same Name as Class**: The constructor name must be the same as the class name.
* **No Return Type**: A constructor does not have a return type.
* **Called Automatically**: It is invoked when an object is created.
* **Used for Object Initialization**: It initializes the object's attributes.

**2. Types of Constructors**

1. **Default Constructor (No-Argument Constructor)**
2. **Parameterized Constructor**
3. **Copy Constructor (Manual Implementation in Java)**

**3. Default Constructor**

A **default constructor** is automatically created by Java if no constructor is defined. It initializes instance variables with their default values.

**Example**

class Student {

String name;

int age;

// Default Constructor

Student() {

System.out.println("Default Constructor is called");

name = "Unknown";

age = 0;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student(); // Default constructor is invoked

s1.display();

}

}

**Output**

Default Constructor is called

Name: Unknown, Age: 0

**4. Parameterized Constructor**

A **parameterized constructor** accepts arguments and initializes the object with specific values.

**Example**

class Student {

String name;

int age;

// Parameterized Constructor

Student(String n, int a) {

name = n;

age = a;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student("Alice", 20);

Student s2 = new Student("Bob", 22);

s1.display();

s2.display();

}

}

**Output**

Name: Alice, Age: 20

Name: Bob, Age: 22

**5. Copy Constructor**

Java does not provide a built-in **copy constructor**, but we can manually create one.

**Example**

class Student {

String name;

int age;

// Parameterized Constructor

Student(String n, int a) {

name = n;

age = a;

}

// Copy Constructor

Student(Student s) {

name = s.name;

age = s.age;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student("Alice", 20);

Student s2 = new Student(s1); // Copy constructor

s1.display();

s2.display();

}

}

**Output**

Name: Alice, Age: 20

Name: Alice, Age: 20

**6. Constructor Overloading**

Constructor overloading means defining multiple constructors with different parameters.

**Example**

class Employee {

String name;

int age;

// Default Constructor

Employee() {

name = "Unknown";

age = 0;

}

// Parameterized Constructor

Employee(String n) {

name = n;

age = 0;

}

// Parameterized Constructor with Two Arguments

Employee(String n, int a) {

name = n;

age = a;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Employee e1 = new Employee();

Employee e2 = new Employee("Alice");

Employee e3 = new Employee("Bob", 25);

e1.display();

e2.display();

e3.display();

}

}

**Output**

Name: Unknown, Age: 0

Name: Alice, Age: 0

Name: Bob, Age: 25

**7. Constructor with this Keyword**

The this keyword is used in a constructor to:

* Refer to the current class instance.
* Invoke another constructor within the same class (Constructor Chaining).

**Example: Using this to Invoke Another Constructor**

class Student {

String name;

int age;

// Constructor with One Parameter

Student(String n) {

this(n, 18); // Calls the second constructor

}

// Constructor with Two Parameters

Student(String n, int a) {

name = n;

age = a;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student("Alice"); // Calls constructor with one parameter

Student s2 = new Student("Bob", 20); // Calls constructor with two parameters

s1.display();

s2.display();

}

}

**Output**

Name: Alice, Age: 18

Name: Bob, Age: 20

**8. Static Constructor (Not Available in Java)**

Java does **not support static constructors** because constructors are used to initialize instances, and static members belong to the class, not instances. However, **static blocks** can be used to initialize static variables.

**Example**

class Example {

static int num;

// Static Block

static {

num = 100;

System.out.println("Static block executed");

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Value of num: " + Example.num);

}

}

**Output**

Static block executed

Value of num: 100

**9. Private Constructor (Singleton Design Pattern)**

A **private constructor** restricts object creation outside the class. This is useful for implementing the **Singleton Pattern**.

**Example**

class Singleton {

private static Singleton instance;

// Private Constructor

private Singleton() {

System.out.println("Singleton Instance Created");

}

// Static Method to Provide Single Instance

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

public class Main {

public static void main(String[] args) {

Singleton obj1 = Singleton.getInstance();

Singleton obj2 = Singleton.getInstance();

}

}

**Output**

Singleton Instance Created

(Even though getInstance() is called twice, only one instance is created.)

**10. Constructor in Inheritance**

When a subclass object is created, the **constructor of the parent class is called first**.

**Example**

class Parent {

Parent() {

System.out.println("Parent Constructor Called");

}

}

class Child extends Parent {

Child() {

System.out.println("Child Constructor Called");

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

}

}

**Output**

Parent Constructor Called

Child Constructor Called