# **Class & Objects in java**

**🔹 What is a Class?**

A **class** is like a **blueprint** or **template** for creating **objects**.

📌 Think of a **class** as a **car design**—it defines what the car should look like (color, brand, speed), but it’s not an actual car.

**🔹 What is an Object?**

An **object** is a **real-world instance** of a class—like your own car created from that car design.

**🚗 Real-Life Example: Car**

Let’s model a car in Java:

class Car {

// Instance variables

String brand;

int speed;

// Static variable

static int numberOfCars = 0;

// Constructor

Car(String brand, int speed) {

this.brand = brand;

this.speed = speed;

numberOfCars++; // Every time a new car is created

}

// Method with local variable

void displayInfo() {

String message = "Details: "; // local variable

System.out.println(message + brand + " is moving at " + speed + " km/h");

}

static void showTotalCars() {

System.out.println("Total cars made: " + numberOfCars);

}

}  
  
✅ Using the class in a program:  
public class Main {

public static void main(String[] args) {

Car car1 = new Car("Toyota", 120);

Car car2 = new Car("Honda", 140);

car1.displayInfo(); // Output: Details: Toyota is moving at 120 km/h

car2.displayInfo(); // Output: Details: Honda is moving at 140 km/h

Car.showTotalCars(); // Output: Total cars made: 2

}

}  
  
🔸 **1. Student & School System  
  
✅ Class: Student**

Each student has their own **name**, **roll number**, and **marks**, but the **school name** is common.

class Student {

String name; // instance variable

int rollNumber;

double marks;

static String school = "Greenwood High"; // static variable

void display() {

System.out.println(name + " (" + rollNumber + ") got " + marks + " marks. School: " + school);

}

}  
  
Usage:  
Student s1 = new Student();

s1.name = "Asha";

s1.rollNumber = 101;

s1.marks = 92.5;

Student s2 = new Student();

s2.name = "Ravi";

s2.rollNumber = 102;

s2.marks = 88.0;

s1.display();

s2.display();

**🔸 2. Bank Account**

**✅ Class: BankAccount**

Every customer has their own **account number** and **balance**, but the **bank name** is shared.

class BankAccount {

String accountHolder; // instance variable

int accountNumber;

double balance;

static String bankName = "State Bank of Java";

void deposit(double amount) {

balance += amount;

System.out.println(amount + " deposited. New balance: " + balance);

}

void display() {

System.out.println(accountHolder + "'s balance at " + bankName + ": " + balance);

}

}  
  
🔸 **3. Book & Library  
✅ Class: Book**

Each book has a **title**, **author**, and **ISBN**, but all books belong to the **same library**.

class Book {

String title;

String author;

String isbn;

static String libraryName = "City Central Library";

void displayDetails() {

System.out.println(title + " by " + author + " (ISBN: " + isbn + ") - " + libraryName);

}

}  
  
**🔸 4. Employee & Company**

**✅ Class: Employee**

Each employee has a **name**, **ID**, and **salary**. All employees work for the **same company**.

class Employee {

String name;

int empId;

double salary;

static String companyName = "TechNova Inc.";

void showInfo() {

System.out.println(name + " (ID: " + empId + ") earns $" + salary + " at " + companyName);

}

}  
  
🔸 **5. Product & Shopping Cart  
✅ Class: Product**

Each product has a **name**, **price**, and **quantity**, but all products come from the **same store**.

class Product {

String name;

double price;

int quantity;

static String storeName = "JavaMart";

double totalCost() {

return price \* quantity;

}

void printReceipt() {

System.out.println(quantity + " x " + name + " @ $" + price + " = $" + totalCost() + " [" + storeName + "]");

}

}  
  
🧠 Key Takeaways

| **Example** | **Class** | **Object (Real Instance)** |
| --- | --- | --- |
| School | Student | Asha, Ravi, etc. |
| Banking | BankAccount | Your personal bank account |
| Library | Book | A specific book like *Harry Potter* |
| Company | Employee | An employee named Raj with ID 101 |
| Shopping Cart | Product | A shopping item like 2 bags of rice |

**🔷 Types of Variables in Java (with Real Examples)**

| **Type** | **Declared inside** | **Associated with** | **Lives until** | **Accessed using** |
| --- | --- | --- | --- | --- |
| **Local** | Method / block | Method only | Method ends | Just name |
| **Instance** | Inside class, outside methods | One per object | Object exists | this.variable |
| **Static** | Inside class, with static | Shared by all objects | Program ends | ClassName.variable |

**🟡 1. Local Variable**

* Declared inside a method.
* Cannot be used outside the method.

**🧠 Real-World Analogy:**

Think of a **temporary shopping receipt** — it's valid only for that purchase. Once you leave the store, it's discarded.

**🧾 Example: Inside a payment method**class Payment {

void makePayment(double amount) {

String receiptId = "TXN123"; // local variable

System.out.println("Paid $" + amount + " | Receipt: " + receiptId);

}

}  
**✅ Key Points:**

* receiptId exists **only inside** makePayment()
* You **can’t use it** in another method

**🔵 2. Instance Variable**

* Declared inside class, but outside methods.
* Each object gets its own copy.

**🧠 Real-World Analogy:**

Think of a **bank account** — every customer has their **own balance**, **own name**, and **own account number**.

class BankAccount {

String accountHolder; // instance variable

double balance; // instance variable

void showDetails() {

System.out.println(accountHolder + "'s Balance: $" + balance);

}

}  
**✅ Key Points:**

* Each object (account) has **its own balance and accountHolder**
* Changing one **doesn’t affect** the others

**🔴 3. Static Variable**

* Shared across all instances.
* Declared using static keyword.

**🧠 Real-World Analogy:**

Imagine a **common bank name** or **interest rate** shared by all accounts — it's **not specific** to any one customer.  
💰 Example: Shared Bank Name  
class BankAccount {

String accountHolder;

double balance;

static String bankName = "State Bank of Java"; // static variable

void showDetails() {

System.out.println(accountHolder + " | " + bankName + " | Balance: $" + balance);

}

}

🎓 Real-World Summary

| **Type** | **Scope** | **Lifetime** | **Real-World Example** | **Keyword Used** |
| --- | --- | --- | --- | --- |
| **Local** | Inside methods only | Until method ends | Shopping receipt, trip timer | None |
| **Instance** | Per object | Until object is alive | Your name, account balance | None |
| **Static** | Shared across all objects | Program ends | Bank name, school name, interest rate | static |

🔁 Quick Comparison in Action  
 class Student {

String name; // instance variable

static String school = "Greenwood High"; // static variable

void printDetails() {

int rollNumber = 101; // local variable

System.out.println(name + " | Roll: " + rollNumber + " | School: " + school);

}

}  
  
🧠 **Quiz: Types of Variables in Java  
🔹 Q1. Identify the Variable Type**class Car {

String color; // Line A

static int wheels = 4; // Line B

void drive() {

int speed = 60; // Line C

System.out.println(color + " car is going at " + speed + " km/h");

}

}  
  
🔹 **Q2. What is shared among all objects of a class?**A. Local variables  
B. Instance variables  
C. Static variables  
D. Constructors  
  
**🔹 Q3. Fill in the blanks**

**Local variables** are created when the method is \_\_\_ and destroyed when the method \_\_\_.

A. called, returns  
B. compiled, executes  
C. loaded, ends  
D. executed, terminates

**🔹 Q4. Real-World Analogy**

Which of the following is best represented by a **static variable**?

A. Each student’s name  
B. A bank’s interest rate  
C. Customer’s phone number  
D. A trip timer in a cab ride

🧠 **Quiz: Java Class and objects**

**1. What is a class in Java?**

A. A variable  
B. A template or blueprint for creating objects  
C. An object  
D. A data type

✅ **Answer:** B. A template or blueprint for creating objects

**2. What is the correct syntax to create an object in Java?**

A. ClassName object = ClassName();  
B. ClassName object = new ClassName();  
C. object = new ClassName();  
D. new object = ClassName();

✅ **Answer:** B. ClassName object = new ClassName();

**3. What is the keyword used to create an object?**

A. class  
B. void  
C. new  
D. object

✅ **Answer:** C. new

**4. How many objects can be created from a single class in Java?**

A. Only one  
B. As many as needed  
C. Five  
D. None

✅ **Answer:** B. As many as needed

**5. Which of the following is true about constructors?**

A. They must have a return type  
B. They initialize an object  
C. They must be abstract  
D. They can only be private

✅ **Answer:** B. They initialize an object

**6. What is the default access modifier of a class member in Java?**

A. public  
B. private  
C. protected  
D. package-private (no modifier)

✅ **Answer:** D. package-private (no modifier)

**7. Which of the following statements is true about objects?**

A. An object holds methods only  
B. An object is an instance of a class  
C. An object is the class itself  
D. Objects can't hold data

✅ **Answer:** B. An object is an instance of a class

**8. Which method is called automatically when an object is created?**

A. main()  
B. initialize()  
C. Constructor  
D. object()

✅ **Answer:** C. Constructor

**9. What will the following code print?**

class Demo {

int x = 10;

}

public class Test {

public static void main(String[] args) {

Demo d = new Demo();

System.out.println(d.x);

}

}

A. x  
B. 0  
C. 10  
D. Compilation error

✅ **Answer:** C. 10

**10. Which of the following best describes object-oriented programming?**

A. Writing functions only  
B. Dividing the program into methods  
C. Organizing code using classes and objects  
D. Using only arrays

✅ **Answer:** C. Organizing code using classes and objects

## 🧠 What is a Method in Java?

A **method** is a **block of code** that performs a specific task.  
Think of a method like a **machine** in a factory — it takes input (if needed), does some work, and may return a result.

**🔹 Method Declaration Syntax**

returnType methodName(parameterList) {

// body of method

}  
  
int add(int a, int b) {

return a + b;

}  
  
🚦 **Types of Methods in Java**

| **Type** | **Description** | **Real-World Analogy** |
| --- | --- | --- |
| **No return, no parameter** | Doesn’t take input, doesn’t return anything | Ringing a doorbell |
| **No return, with parameter** | Takes input, but doesn’t return anything | Giving instructions to someone |
| **With return, no parameter** | Doesn’t take input, but returns something | Thermometer giving current temperature |
| **With return, with parameter** | Takes input, and returns something | Calculator returning the sum of two numbers |

✅ **1. No Return Type, No Parameter**void greet() {

System.out.println("Hello, welcome!");

}  
🎯 *Real-life:* Press a button to get a greeting. No input, no result expected.

✅ **2. No Return Type, With Parameter**

void greetUser(String name) {

System.out.println("Hello " + name + ", welcome!");

}  
🎯 *Real-life:* You give your name; it greets you — no response expected.  
  
✅ **3. With Return Type, No Parameter**int getDiscount() {

return 10; // fixed 10% discount

}  
🎯 *Real-life:* You ask, “What’s the discount?” — it gives you the number.  
  
✅ **4. With Return Type, With Parameter**int add(int a, int b) {

return a + b;

}  
🎯 Real-life: You enter two numbers into a calculator; it gives the result back.  
  
🛒 **Real-World Example: ShoppingCart**  
  
class ShoppingCart {

int totalItems = 0;

// No return, no parameter

void startShopping() {

System.out.println("Shopping started!");

}

// No return, with parameter

void addItem(String itemName) {

totalItems++;

System.out.println(itemName + " added to cart.");

}

// Return type, no parameter

int getTotalItems() {

return totalItems;

}

// Return type, with parameter

double calculateBill(double pricePerItem) {

return totalItems \* pricePerItem;

}

}  
  
✅ Using the Methods:  
public class Main {

public static void main(String[] args) {

ShoppingCart cart = new ShoppingCart();

cart.startShopping(); // No input/output

cart.addItem("Book"); // Input only

cart.addItem("Pen");

System.out.println("Total items: " + cart.getTotalItems()); // Output only

double totalBill = cart.calculateBill(50); // Input + Output

System.out.println("Total bill: $" + totalBill);

}

}  
  
🧠 **Interactive Quiz: Java Methods  
🔹 Q1. Which of the following is a valid method declaration?**

A. method int add(a, b)  
B. int add(int a, int b)  
C. add int(int a, int b)  
D. int add = (int a, int b)

**🔹 Q2. What will this method return?**

int multiply(int x, int y) {

return x \* y;

}

If we call multiply(4, 5);

A. 9  
B. 20  
C. Error  
D. void  
  
🔹 **Q3. What type of method is this?**void printHello() {

System.out.println("Hello!");

}

A. No return, no parameter  
B. With return, no parameter  
C. With return, with parameter  
D. No return, with parameter  
  
**🔹 Q4. Fill in the Blanks**

A method with a return type must use the \_\_\_\_\_\_ keyword to return a value.  
A method with no return type uses the keyword \_\_\_\_\_\_.

**🧱 What is a Constructor in Java?**

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

It is used to initialize objects.  
  
**🔑 Key Features of Constructors**

* Name is **same as the class name**
* **No return type**, not even void
* Called automatically when you create an object with new

🏗️ **Types of Constructors**

| **Type** | **Description** | **Example Use Case** |
| --- | --- | --- |
| Default Constructor | Automatically provided by Java if no other constructor is defined | Create objects with default values |
| No-Arg Constructor | User-defined constructor without parameters | Show a welcome message, or set default states |
| Parameterized Constructor | Constructor with parameters to initialize object | Set name, age, ID of a student on creation |

**💡 1. Default Constructor**

Java provides this **only when you don’t create any constructor**.

class Student {

String name;

// No constructor defined

void show() {

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

}

}  
public class Main {

public static void main(String[] args) {

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

s.show(); // Outputs: Name: null

}

}  
  
💡 **2. No-Argument Constructor (User-defined)**class Student {

Student() {

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

}

}  
public class Main {

public static void main(String[] args) {

Student s1 = new Student(); // Output: Student object created!

}

}  
  
💡 **3. Parameterized Constructor**class Student {

String name;

int age;

// Parameterized constructor

Student(String n, int a) {

name = n;

age = a;

}

void display() {

System.out.println(name + " is " + age + " years old.");

}

}  
public class Main {

public static void main(String[] args) {

Student s1 = new Student("Amit", 17);

s1.display(); // Output: Amit is 17 years old.

}

}  
🎯 **Real-World Analogy**

| **Real-Life Scenario** | **Constructor Type** |
| --- | --- |
| A student fills in a form with name and age | Parameterized constructor |
| A default user is created with empty values | Default constructor |
| A user logs in and sees "Welcome!" | No-arg constructor |

💻 **Hands-On Practical – Car Example**class Car {

String brand;

int year;

// Constructor with parameters

Car(String b, int y) {

brand = b;

year = y;

}

void showDetails() {

System.out.println("Brand: " + brand + " | Year: " + year);

}

}  
public class Main {

public static void main(String[] args) {

Car c1 = new Car("Toyota", 2022);

Car c2 = new Car("Honda", 2023);

c1.showDetails();

c2.showDetails();

}

}  
  
🧠 **Interactive Quiz – Constructors in Java  
🔹 Q1. What is the purpose of a constructor?**

A. Perform calculations  
B. Print values  
C. Initialize an object  
D. Destroy an object

**🔹 Q2. Which statement is true about constructors?**

A. Constructors can return values  
B. Constructors must have a return type  
C. Constructors must be public  
D. Constructors have the same name as the class

**🔹 Q3. What happens if you don’t write any constructor in your class?**

A. Program crashes  
B. Java provides a default constructor  
C. Error occurs  
D. Constructor must always be defined

🔹 **Q4. Identify the Constructor Type**class Book {

Book() {

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

}

}  
A. Default Constructor  
B. No-Arg Constructor  
C. Parameterized Constructor  
D. Static Constructor  
  
🔹 **Q5. Guess the Output**class Employee {

String name;

int id;

Employee(String n, int i) {

name = n;

id = i;

}

void print() {

System.out.println(name + " | ID: " + id);

}

}  
What will be the output of the following?

Employee e1 = new Employee("Ravi", 101);

e1.print();  
A. Ravi | ID: 101  
B. name | id  
C. Compilation Error  
D. null | 0  
  
**🔹 Q6. Fill in the Blanks**

1. A constructor has the same \_\_\_\_\_\_\_ as the class name.
2. A constructor does not have any \_\_\_\_\_\_\_ type.
3. A parameterized constructor accepts \_\_\_\_\_\_\_.

📝 Wrap-Up Summary

| **Type** | **Has Parameters?** | **Defined by You?** | **Common Use** |
| --- | --- | --- | --- |
| Default Constructor | ❌ | ❌ (by Java) | Auto-create objects with nulls/0 |
| No-Arg Constructor | ❌ | ✅ | Show welcome or default setup |
| Parameterized | ✅ | ✅ | Initialize with actual values |

## **Assignments for this week:**

📝 **Assignment 1: Student Report Card  
  
🔹 Objective:**

Use **class, instance/static variables, and parameterized constructors**.

**🔹 Requirements:**

* Create a class Student with instance variables: name, rollNo, marks.
* Use a **static variable** to store the schoolName.
* Use a **parameterized constructor** to initialize student details.
* Use a method displayDetails() to print student details.

🔹 **Input:**

Student s1 = new Student("Amit", 101, 85);

Student s2 = new Student("Priya", 102, 92);  
🔹 **Output:**

School: ABC International

Name: Amit, Roll No: 101, Marks: 85

School: ABC International

Name: Priya, Roll No: 102, Marks: 92

**📝 Assignment 2: Bank Account Management**

**🔹 Objective:**

Use **constructors**, **method types**, and variable types.

**🔹 Requirements:**

* Create a class BankAccount.
* Instance variables: accountHolder, balance.
* Static variable: bankName.
* Constructors: one **no-arg constructor** and one **parameterized constructor**.
* Method deposit(double amount) (void, with parameter).
* Method getBalance() (return type, no parameter).

🔹 **Input:**BankAccount acc1 = new BankAccount("Riya", 5000);

acc1.deposit(2000);

System.out.println("Balance: " + acc1.getBalance());  
🔹 Output:

Balance: 7000.0

**📝 Assignment 3: Movie Ticket Booking**

**🔹 Objective:**

Practice method declaration types, static and local variables.

**🔹 Requirements:**

* Create class MovieTicket.
* Instance variables: movieName, seatNo, price.
* Static variable: theaterName.
* Constructor to initialize ticket details.
* Method printTicket() to display details.
* Method calculateTotal(int quantity) — use local variable to calculate total cost.

**🔹 Input:**

MovieTicket t1 = new MovieTicket("Jawan", "A5", 200);

t1.printTicket();

System.out.println("Total for 3 tickets: " + t1.calculateTotal(3));

🔹 **Output:**

Theater: PVR Cinemas

Movie: Jawan | Seat: A5 | Price: 200

Total for 3 tickets: 600  
  
📝 **Assignment 4: Library System**

**🔹 Objective:**

Use class, constructors, and all types of methods.

**🔹 Requirements:**

* Create class LibraryBook.
* Instance variables: title, author, isAvailable.
* Static variable: libraryName.
* No-arg constructor: sets book as available.
* Parameterized constructor: sets title and author.
* Method borrowBook() sets availability to false.
* Method returnBook() sets availability to true.
* Method checkAvailability() returns boolean.

🔹 Input:  
LibraryBook book1 = new LibraryBook("The Alchemist", "Paulo Coelho");

book1.borrowBook();

System.out.println("Available: " + book1.checkAvailability());

book1.returnBook();

System.out.println("Available: " + book1.checkAvailability());  
🔹 **Output:**

Available: false

Available: true