

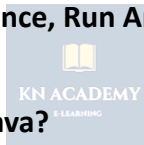
## Contents

Java Basics .....	1
Methods .....	6
Memory Management .....	9
Arrays & Strings .....	13
Exception Handling .....	16
Multithreading .....	19

## Java Basics

### 1. What is Java?

- Java is a **high-level, object-oriented, platform-independent** programming language.
- It follows **WORA (Write Once, Run Anywhere)** due to the JVM (Java Virtual Machine).



### 2. What are the main features of Java?

- **Object-Oriented, Platform-Independent, Secure, Robust, Multithreading Support, Garbage Collection.**

### 3. What is JDK, JRE, and JVM?

- **JDK (Java Development Kit)** → Includes JRE + compiler + tools.
- **JRE (Java Runtime Environment)** → Includes JVM + libraries.
- **JVM (Java Virtual Machine)** → Converts Java code into **bytecode** and runs it.

### 4. Write a simple "Hello World" Java program.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

✓ **Output:**

Hello, World!

#### 5. What are data types in Java?

- **Primitive Types:** int, double, char, boolean, float, long, byte, short.
- **Non-Primitive Types:** String, Array, Class, Interface.

#### 6. What is type casting in Java?

- **Implicit (Widening):** Smaller → Larger (automatic).
- **Explicit (Narrowing):** Larger → Smaller (manual).

```
double d = 5; // Implicit
```

```
int x = (int) 5.99; // Explicit
```

#### 7. What is the difference between == and .equals()?

- == compares **memory references**.
- .equals() compares **actual values**.

```
String s1 = new String("Java");
```

```
String s2 = new String("Java");
```

```
System.out.println(s1 == s2); // false
```

```
System.out.println(s1.equals(s2)); // true
```



#### 8. What are operators in Java?

- **Arithmetic** (+, -, \*, /, %)
- **Relational** (==, !=, >, <, >=, <=)
- **Logical** (&&, ||, !)

---

#### 9. What are control flow statements in Java?

- **Conditional Statements:** if, if-else, switch.
- **Looping Statements:** for, while, do-while.
- **Branching Statements:** break, continue, return.

#### 10. Write a program to check if a number is even or odd using if-else.

```

public class EvenOdd {
    public static void main(String[] args) {
        int num = 7;
        if (num % 2 == 0) {
            System.out.println("Even");
        } else {
            System.out.println("Odd");
        }
    }
}

```

✅ **Output:**

Odd

### 11. How does the switch statement work in Java?

```

public class SwitchExample {
    public static void main(String[] args) {
        int day = 3;
        switch (day) {
            case 1: System.out.println("Monday"); break;
            case 2: System.out.println("Tuesday"); break;
            case 3: System.out.println("Wednesday"); break;
            default: System.out.println("Invalid day");
        }
    }
}

```



### 12. What are loops in Java?

- **For Loop** (fixed iterations).
- **While Loop** (condition-based).
- **Do-While Loop** (executes at least once).

### 13. Write a program to print numbers from 1 to 5 using a for loop.

```

public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.print(i + " ");
        }
    }
}

```

✅ **Output:**

**14. What is the difference between while and do-while loops?**

```

int i = 5;
do {
    System.out.println(i);
    i--;
} while (i > 0);

```



**15. What does the break statement do?**

```

for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    System.out.print(i + " "); // Output: 1 2
}

```

**16. What does the continue statement do?**

```

for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    System.out.print(i + " "); // Output: 1 2 4 5
}

```

**17. How to find the sum of numbers from 1 to N using a loop?**

```

public class SumNumbers {
    public static void main(String[] args) {
        int n = 10, sum = 0;
        for (int i = 1; i <= n; i++) {
            sum += i;
        }
        System.out.println("Sum: " + sum);
    }
}

```

✅ **Output:**

Sum: 55

**18. What is an infinite loop? Give an example.**



```

while (true) {
    System.out.println("This will run forever!");
}

```

**19. How do you generate random numbers in Java?**

```

import java.util.Random;

public class RandomExample {
    public static void main(String[] args) {
        Random rand = new Random();
        System.out.println(rand.nextInt(100)); // Random number (0-99)
    }
}

```

**20. Write a Java program to check if a number is prime.**

```

public class PrimeCheck {

```

```
public static void main(String[] args) {  
    int num = 7, count = 0;  
    for (int i = 1; i <= num; i++) {  
        if (num % i == 0) count++;  
    }  
    System.out.println(count == 2 ? "Prime" : "Not Prime");  
}  
}
```

✅ **Output:**

Prime

## Methods



### 1. What is a method in Java?

A method is a block of code that performs a specific task.

#### Syntax:



```
returnType methodName(parameters) {  
    // Method body  
    return value;  
}
```

#### Example:

```
public int add(int a, int b) {  
    return a + b;  
}
```

---

### 2. What is the difference between static and instance methods?

Feature	Static Method	Instance Method
Belongs To	Class	Object
Called Using	ClassName.methodName()	object.methodName()
Uses this?	 No	 Yes

### Example:

```
class Example {
    static void staticMethod() {
        System.out.println("Static method");
    }

    void instanceMethod() {
        System.out.println("Instance method");
    }
}
```



```
// Calling
Example.staticMethod();
Example obj = new Example();
obj.instanceMethod();
```

---

### 3. What is method overloading?

Method overloading allows multiple methods with the same name but different parameters.

### Example:

```
class MathOperations {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
}
```

```
public class Test {  
    public static void main(String[] args) {  
        MathOperations obj = new MathOperations();  
        System.out.println(obj.add(5, 10)); // Calls int method  
        System.out.println(obj.add(5.5, 2.2)); // Calls double method  
    }  
}
```

---

#### 4. What is method overriding?

Method overriding allows a subclass to redefine a method from the parent class.

##### Example:

```
class Parent {  
    void show() { System.out.println("Parent method"); }  
}
```



```
class Child extends Parent {  
    @Override  
    void show() { System.out.println("Child method"); }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Parent obj = new Child();  
        obj.show(); // Output: Child method  
    }  
}
```

---

#### 5. What is call by value in Java?

Java always passes arguments by value.



### Example (Primitive Type - No change in original value):

```
class Example {  
    void modify(int num) { num = num + 10; }  
  
    public static void main(String[] args) {  
        Example obj = new Example();  
        int x = 5;  
        obj.modify(x);  
        System.out.println(x); // Output: 5 (unchanged)  
    }  
}
```

### Example (Object Reference - Value inside object changes):

```
class Example {  
    int value = 10;  
  
    void modify(Example obj) { obj.value += 10; }  
  
    public static void main(String[] args) {  
        Example obj = new Example();  
        obj.modify(obj);  
        System.out.println(obj.value); // Output: 20 (modified)  
    }  
}
```



## Memory Management

### 1. What is memory management in Java?

Memory management in Java is the process of allocating and deallocating memory automatically using **Garbage Collection (GC)**. Java uses **Heap and Stack memory** for efficient memory management.

---

## 2. What is the difference between Heap and Stack memory?

Feature	Heap Memory	Stack Memory
Stores	Objects & Class Instances	Method calls & Local Variables
Access Speed	Slower	Faster
Lifetime	Exists till GC removes it	Exists till method execution finishes
Example	Objects created using new Method variables, function calls	

### Example:

```
class Example {  
    int x = 10; // Stored in Heap  
  
    void method() {  
        int y = 20; // Stored in Stack  
    }  
}
```



---

## 3. What is Garbage Collection in Java?

Garbage Collection (GC) automatically removes unused objects from memory to **prevent memory leaks**.

- Uses **Mark and Sweep Algorithm** to find unreachable objects.
- **No free() or delete()** like C/C++.
- Java provides the `System.gc()` method to request garbage collection (but it is not guaranteed to run immediately).

### Example:

```
class Example {  
    protected void finalize() { // Called before GC destroys object  
        System.out.println("Object destroyed");  
    }  
}
```

```

public class Test {

    public static void main(String[] args) {

        Example obj = new Example();

        obj = null; // Making object eligible for GC

        System.gc();

    }

}

```

---

#### 4. What are strong, weak, soft, and phantom references?

In Java, references determine how objects are handled by the Garbage Collector.

Type	Description
<b>Strong Reference</b>	Normal object references (not garbage collected)
<b>Weak Reference</b>	Garbage collected if no strong reference exists
<b>Soft Reference</b>	Collected only when memory is low
<b>Phantom Reference</b>	Used for final cleanup before object removal

#### Example (Weak Reference):

```

import java.lang.ref.WeakReference;

class Example {

    void show() { System.out.println("Hello"); }

}

public class Test {

    public static void main(String[] args) {

        Example obj = new Example();

        WeakReference<Example> weakRef = new WeakReference<>(obj);

        obj = null; // Making eligible for GC
    }

}

```

```
        System.gc();

        System.out.println(weakRef.get()); // Might print null if GC has run
    }
}
```

---

## 5. What are memory leaks in Java and how to prevent them?

A **memory leak** occurs when objects are not garbage collected due to **unintended strong references**, leading to **high memory usage**.

### Causes of Memory Leaks:

- **Unclosed resources** (e.g., Streams, Connections)
- **Static references** holding objects
- **Inner class references to outer classes**

### Prevention Methods:

- Use **WeakReferences** for cache-like data.
- Always **close I/O resources** (try-with-resources).
- Use **profiling tools** like **VisualVM**, **JProfiler** to detect memory leaks.

### Example (Avoiding Memory Leak in Streams):

```
import java.io.*;

public class Test {

    public static void main(String[] args) {

        try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

            System.out.println(br.readLine());

        } catch (IOException e) {

            e.printStackTrace();

        } // No need to manually close, handled by try-with-resources

    }

}
```

## Arrays & Strings

### 1. What is an array in Java?

- An array is a **fixed-size, homogeneous** collection of elements stored in **contiguous memory locations**.

### 2. How do you declare and initialize an array in Java?

```
int[] arr = {1, 2, 3, 4, 5}; // Declaration and Initialization
```

```
int[] arr2 = new int[5]; // Declaration with size
```

### 3. Can we change the size of an array in Java?

- No, arrays have a **fixed size**. Use **ArrayList** if you need a dynamic array.

### 4. What are the different ways to traverse an array?

- For Loop
- Enhanced For Loop (for-each)
- While Loop
- Streams API

### 5. How to copy an array in Java?

- Using `System.arraycopy()`
- Using `Arrays.copyOf()`
- Using `clone()`
- Using a loop



Example:

```
int[] original = {1, 2, 3};
```

```
int[] copy = Arrays.copyOf(original, original.length);
```

### 6. What is the difference between deep copy and shallow copy?

- **Shallow Copy** → Copies only references, changes reflect in both arrays.
- **Deep Copy** → Creates a new independent copy.

### 7. How do you sort an array in Java?

- Using `Arrays.sort()` (Ascending order by default).
- Using `Arrays.sort(arr, Collections.reverseOrder())` (For descending).

Example:

```
int[] arr = {3, 1, 4, 2};  
Arrays.sort(arr); // [1, 2, 3, 4]
```

### 8. What is a jagged array?

- A **jagged array** is an array of arrays where the sub-arrays can have different sizes.

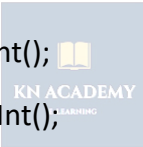
```
int[][] jaggedArray = { {1, 2}, {3, 4, 5}, {6} };
```

### 9. What is the difference between a 1D and 2D array?

- **1D Array** → `int[] arr = new int[5];` (Single row).
- **2D Array** → `int[][] arr = new int[3][3];` (Rows & Columns).

### 10. How to find the largest and smallest element in an array?

```
int[] arr = {5, 1, 9, 3};  
int min = Arrays.stream(arr).min().getAsInt();  
int max = Arrays.stream(arr).max().getAsInt();
```



---

## 🔴 Java Strings Questions

### 11. What is a String in Java?

- A **String** is an **immutable** sequence of characters stored in a **char array**.

### 12. How do you create a String in Java?

```
String str1 = "Hello"; // String literal (Stored in String Pool)  
String str2 = new String("Hello"); // String object (Stored in Heap)
```

### 13. What is the difference between == and .equals() in Strings?

- `==` → Compares **memory references**.
- `.equals()` → Compares **actual content**.

```
String s1 = new String("Java");  
String s2 = new String("Java");  
System.out.println(s1 == s2); // false
```

```
System.out.println(s1.equals(s2)); // true
```

#### 14. Why are Strings immutable in Java?

- **Security:** Prevents modification.
- **Caching:** Can be stored in String Pool.
- **Thread Safety:** No need for synchronization.

#### 15. How to convert a String to a character array?

```
String str = "Hello";
```

```
char[] charArray = str.toCharArray();
```

#### 16. How to reverse a String in Java?

- **Using StringBuilder**

```
String str = "Java";
```

```
String reversed = new StringBuilder(str).reverse().toString();
```

#### 17. What is StringBuilder and StringBuffer?

Feature	StringBuilder	StringBuffer
Mutability	Mutable	Mutable
Thread Safety	✗ No	✓ Yes
Performance	✓ Faster	✗ Slower



Example:

```
StringBuilder sb = new StringBuilder("Hello");
```

```
sb.append(" World"); // Mutates the original string
```

#### 18. How to split a String in Java?

```
String str = "Java,Python,C++";
```

```
String[] words = str.split(",");
```

#### 19. How to check if a String contains a particular word?

```
String str = "I love Java";  
boolean contains = str.contains("Java"); // true
```

## 20. How to remove white spaces from a String?

```
String str = " Hello World ";  
String trimmed = str.trim(); // "Hello World"  
String removedSpaces = str.replaceAll("\\s", ""); // "HelloWorld"
```

# Exception Handling

## 1. What is Exception Handling in Java?

- Exception handling is a way to **handle runtime errors** and prevent program crashes.
- Java provides **try, catch, finally, throw, and throws** to handle exceptions.

---

## 2. What are the types of exceptions in Java?

- **Checked Exceptions** → Must be handled using try-catch or throws.
  - Example: IOException, SQLException.
- **Unchecked Exceptions** → Occur at runtime and don't require handling.
  - Example: NullPointerException, ArithmeticException.

---

## 3. What is the difference between throw and throws?

- **throw** → Used inside a method to **explicitly throw an exception**.

```
throw new ArithmeticException("Division by zero");
```

- **throws** → Used in the **method signature** to indicate that a method **may throw an exception**.

```
void myMethod() throws IOException { }
```

---



#### 4. What is the difference between try, catch, and finally?

- try → Contains code that might cause an exception.
- catch → Handles the exception.
- finally → Always executes, whether an exception occurs or not.

Example:

```
try {  
    int result = 10 / 0;  
} catch (ArithmeticException e) {  
    System.out.println("Cannot divide by zero");  
} finally {  
    System.out.println("This block always executes");  
}
```

---

#### 5. What is the difference between Checked and Unchecked Exceptions?

Feature	Checked Exception	Unchecked Exception
When it occurs	Compile-time	Runtime
Requires Handling	Yes (must be handled)	No, but can be handled
Examples	IOException, SQLException NullPointerException, ArithmeticException	

---

#### 6. Can we have multiple catch blocks in Java?

- Yes, we can have multiple catch blocks to handle different exceptions.

```
try {  
    int arr[] = new int[5];  
    arr[10] = 50; // ArrayIndexOutOfBoundsException  
} catch (ArithmeticException e) {  
    System.out.println("Arithmetic Exception");  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Array Index Out of Bounds Exception");  
}
```

```
}
```

---

## 7. What happens if an exception is not handled?

- If an exception is not handled, the program **terminates abruptly** and shows an error message.

Example:

```
int a = 10 / 0; // Causes ArithmeticException  
System.out.println("This will not execute");
```

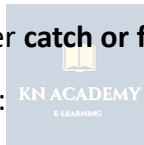
✓ **Output:**

Exception in thread "main" java.lang.ArithmeticException: / by zero

---

## 8. Can we write only try block without catch or finally?

- No, try must be followed by either **catch or finally**.
- This will give a **compilation error**:



```
try {  
    System.out.println("Hello");  
}  
// No catch or finally → Error
```

---

## 9. What is the use of finally block?

- The finally block **always executes**, even if an exception occurs.
- Used to **close resources** (like files, database connections).

Example:

```
try {  
    int num = 10 / 0;
```

```
} catch (ArithmeticException e) {  
    System.out.println("Exception caught");  
}  
} finally {  
    System.out.println("Finally block always executes");  
}
```

✅ **Output:**

Exception caught

Finally block always executes

---

## 10. What is the difference between throw and try-catch?

- **throw** is used to manually **raise an exception**.
- **try-catch** is used to **handle** exceptions.

Example of throw:

```
void checkAge(int age) {  
    if (age < 18) {  
        throw new IllegalArgumentException("Not eligible to vote");  
    }  
}
```



# Multithreading

## 1. What is multithreading in Java?

- Multithreading is the process of executing **multiple threads simultaneously** to improve performance.
  - Each thread runs **independently**, sharing the same memory space.
- 

## 2. What is the difference between a process and a thread?

Feature	Process	Thread
Definition	A separate executing program.	A lightweight sub-task within a process.
Memory	Has its own memory.	Shares memory with other threads.
Communication	Uses <b>Inter-Process Communication (IPC)</b> .	Communicates easily via shared memory.
Creation Overhead	High	Low

---

### 3. How to create a thread in Java?

- **Using Thread class** (by extending Thread class).
- **Using Runnable interface** (by implementing Runnable).

Example using Thread class:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start(); // Start the thread
    }
}
```



Example using Runnable interface:

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable thread is running");
    }
}
```

```

    }
}

public class Main {

    public static void main(String[] args) {

        Thread t1 = new Thread(new MyRunnable());

        t1.start();

    }

}

```

---

#### 4. What is the difference between start() and run() method in threads?

Method	start()	run()
--------	---------	-------

Purpose	Starts a new thread	Runs in the same thread (no new thread created)
---------	---------------------	---

Execution	Calls run() internally	Just executes like a normal method
-----------	------------------------	------------------------------------

Example	t1.start();	t1.run();
---------	-------------	-----------

Example:

```
Thread t = new Thread(() -> System.out.println("Running"));
```

```
t.run(); // Runs in the main thread (no new thread)
```

```
t.start(); // Runs in a separate thread
```

---

#### 5. What are different thread states in Java?

A thread in Java can be in one of the following states:

1. **NEW** → Thread is created but not started.
2. **RUNNABLE** → Thread is ready to run but waiting for CPU.
3. **RUNNING** → Thread is executing.
4. **BLOCKED** → Thread is waiting for a resource.
5. **WAITING** → Thread is waiting indefinitely.
6. **TIMED\_WAITING** → Thread is waiting for a specific time.
7. **TERMINATED** → Thread has completed execution.

Example:



```
Thread.State state = t1.getState();  
System.out.println(state); // Prints the current state
```

---

## 6. What is synchronization in multithreading?

- **Synchronization** ensures that **only one thread** can access a **critical section** at a time.
- Prevents **race conditions** and ensures **thread safety**.

Example using synchronized keyword:

```
class Counter {  
    private int count = 0;  
  
    public synchronized void increment() {  
        count++;  
    }  
  
    public int getCount() {  
        return count;  
    }  
}
```



---

## 7. What is the difference between synchronized method and synchronized block?

Feature	synchronized Method	synchronized Block
Lock Scope	Entire method is locked	Only specific block is locked
Efficiency	Slower (locks entire method)	Faster (locks only critical section)
Example	synchronized void method()	synchronized (this) {}

Example of **synchronized block**:

```
public void method() {  
    synchronized (this) {  
        // Critical section  
    }  
}
```

```
}
```

---

## 8. What is the difference between wait(), notify(), and notifyAll()?

Method	Description
--------	-------------

wait()	Makes the current thread wait until another thread calls notify().
--------	--

notify()	Wakes up <b>one</b> waiting thread.
----------	-------------------------------------

notifyAll()	Wakes up <b>all</b> waiting threads.
-------------	--------------------------------------

Example:

```
synchronized(obj) {  
    obj.wait(); // Thread waits  
    obj.notify(); // Wakes up one waiting thread  
}
```

---

## 9. What is the difference between volatile and synchronized?

Feature	volatile	synchronized
Purpose	Ensures visibility of variables across threads	Ensures atomicity and prevents race conditions
Locking	No lock	Uses locks
Performance	Faster	Slower
Example	volatile int count;	synchronized void method() {}

Example using volatile:

```
class Shared {  
    volatile boolean flag = false;  
}
```

---

## 10. What is a deadlock in multithreading? How to prevent it?

- **Deadlock** occurs when **two or more threads** are waiting for each other's **locked resources**, causing an infinite wait.

Example of deadlock:

```

class Deadlock {

    static final Object resource1 = new Object();
    static final Object resource2 = new Object();

    public static void main(String[] args) {

        Thread t1 = new Thread(() -> {
            synchronized (resource1) {
                System.out.println("Thread 1: Locked resource 1");
                synchronized (resource2) {
                    System.out.println("Thread 1: Locked resource 2");
                }
            }
        });

        Thread t2 = new Thread(() -> {
            synchronized (resource2) {
                System.out.println("Thread 2: Locked resource 2");
                synchronized (resource1) {
                    System.out.println("Thread 2: Locked resource 1");
                }
            }
        });

        t1.start();
        t2.start();
    }
}

```



#### 💡 How to prevent deadlock?

- Always **acquire locks in the same order**.



- Use **timeout mechanisms**.
- Avoid nested locks.

