String Immutability
StringBuilder
StringBufferr
Packages

Java String Kya Hoti Hai?

- Characters ka sequence (text data)
- Immutable (modify nahi ho sakti)
- `java.lang.String` class ka object
- Double quotes me likhi jati hai:

```
String name = "Divyansh";
```

PART–1 : STRING IMMUTABILITY (VERY IMPORTANT TOPIC)

Java me **String immutable hoti hai**.
Matlab: Ek baar string ban gayi → uski value **change nahi** ho sakti.

1) WHY String is Immutable?

Iske 5 major reasons hain (interview favorite):

Memory Optimization (String Pool)

Java ke paas **String Constant Pool (SCP)** hota hai.
Yaha ek hi value wali string **sirf ek baar** store hoti hai.
Example:

```
String s1 = "Hello";
String s2 = "Hello";
```

Dono **same object ko refer** karte hain.
Agar strings mutable hoti, to koi bhi string ko modify karke **dusre reference ki value bhi badal** deta—
Memory corruption ho jati.

Security (Most Important)

String ka use hota hai:

- Password
- Username
- File paths
- Network URLs
- Database connections

Agar Strings mutable hoti to koi object badal kar:
`url = "jdbc:mysql://production-db"`
`→ change → "jdbc:mysql://hacker-db"`

Security issue ho jata.

Thread Safety

String immutable hone se Java unnecessary synchronization se bach jata.
String **by default thread-safe** hoti hai.

Hashcode Caching

String ka hashcode **cached** hota hai (ek baar compute hota hai).
Isliye Strings HashMap, HashSet me fast search dete hain.

Agar mutate hoti to har modification ke baad **hashcode badalta**, jo performance degrade karta.

Performance

Immutable string ka optimization JVM ke liye easy hota hai.

Summary (Interview 1-Line)

"String immutable hai because of security, memory optimization (SCP), thread-safety and hashcode caching."

2) String modification actually kaise hoti hai?

Jab tum String change karte ho:
```
String s = "Hello";
s = s + "World";
```
Java ek **naya String object** banata hai:
```
"Hello"  → old object
"HelloWorld" → new object
```
Purani string unchanged.

Example Proof (Memory Reference Change)**

```
String s = "Java";
System.out.println(System.identityHashCode(s));
s = s + " Rocks";
System.out.println(System.identityHashCode(s));
```

Output me dono hashcode different honge

Proof: **String immutable hai**

Real life analogy:

Tumhare Aadhaar card ka number immutable hota hai –
badal nahi sakta.

Important String Methods (sabse zyada used)

Java String ke **most commonly used methods**

1. length()

String ke total characters batata hai.

```
String s = "Hello";
s.length();  // 5
```

2. charAt(index)

Kisi index par kaun sa character hai.

```
s.charAt(1);  // 'e'
```

3. substring(start) / substring(start, end)

String ka part nikalne ke liye.

```
s.substring(1);      // "ello"
s.substring(1, 4);   // "ell"
```

4. equals() & equalsIgnoreCase()

Do strings ko compare karta hai.

```
"Hello".equals("Hello");          // true
"hello".equalsIgnoreCase("HELLO"); // true
```

5. compareTo()

Lexicographically compare karta hai.

```
"a".compareTo("b"); // -1
```

6. toUpperCase(), toLowerCase()

Case change karne ke liye.

```
"hello".toUpperCase(); // "HELLO"
```

7. trim()

Aage/peeche ke spaces hataata hai.

```
"  hello  ".trim(); // "hello"
```

8. contains()

Check karta hai ki substring exist karti hai ya nahi.

```
"hello world".contains("world"); // true
```

9. startsWith() & endsWith()

```
"hello".startsWith("he"); // true
"hello".endsWith("lo");   // true
```

10. indexOf() & lastIndexOf()

Character ya substring ka index.

```
"banana".indexOf("a");       // 1
"banana".lastIndexOf("a");   // 5
```

11. replace() & replaceAll()

```
"hello".replace('l', 'p'); // "heppo"
"hello123".replaceAll("\\d", ""); // "hello"
```

12. split()

String ko parts me todta hai.

```
"apple,banana,mango".split(",");
```

13. isEmpty() & isBlank()

```
"".isEmpty();    // true
"   ".isBlank(); // true
```

14. join()

Values ko ek delimiter ke saath jodta hai.

```
String.join("-", "a", "b", "c"); // "a-b-c"
```

15. format()

Formatted string banane ke liye.

```
String.format("Name: %s, Age: %d", "Amit", 20);
```

PART–2 : STRINGBUILDER (Mutable String, Fastest)

Ye topic **String immutability ka continuation** hai.

1) What is StringBuilder?

StringBuilder mutable hota hai

Matlab: Ek baar object ban gaya → tum **uski value change kar sakte ho** bina naya object banaye.

Ye **String ke opposite** hota hai.

2) When to use StringBuilder?

- Jab **bahut zyada modifications** karne ho
- Loop me string concatenate karni ho
- Performance important ho
- Single-threaded program ho

3) Why is StringBuilder faster than String?

Example:

```
String s = "";
for(int i = 0; i < 5; i++) {
    s = s + i;
}
```
Yaha har iteration me **new String object** banta hai.

Performance↓, Memory waste ↑

But StringBuilder:
```
StringBuilder sb = new StringBuilder("");
for(int i = 0; i < 5; i++) {
    sb.append(i);
}
```

Same object modify hota hai
No new object creation
Fastest

4) Common Methods in StringBuilder

Method	Description
`append()`	Add text at end
`insert()`	Insert at specific index
`replace()`	Replace characters
`delete()`	Remove characters
`reverse()`	Reverse string
`capacity()`	Internal buffer size
`toString()`	Return final String

5) Example Program (Very Important)

```
public class BuilderDemo {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hello");

        sb.append(" World");
        sb.insert(6, "Java ");
        sb.replace(0, 5, "Hi");
        sb.delete(3, 8);
```

```
        sb.reverse();

        System.out.println(sb);
    }
}
```

6) Proof of Mutability

```
StringBuilder sb = new StringBuilder("A");
System.out.println(System.identityHashCode(sb));
sb.append("B");
System.out.println(System.identityHashCode(sb));
```

Dono hashcode **same** honge
Proof: **StringBuilder mutable hai**

7) StringBuilder is NOT thread-safe

- Isme **synchronization nahi hota**
- Multi-thread environment me unsafe
- But isi ke wajah se **fastest** hota hai

Summary (Interview Answer)

“StringBuilder mutable, non-synchronized, and fastest class hai jo heavily string modification ke liye use hota hai.”

PART–3 : STRINGBUFFER (Mutable + Thread-Safe String Class)

Ab hum **StringBuffer** ko detail me samjhenge.
Ye **StringBuilder ka twin brother** jaisa hota hai, but ek major difference ke saath.

1) What is StringBuffer?

StringBuffer mutable hota hai (just like StringBuilder)
Thread-safe hota hai (StringBuilder ke opposite)

Matlab multi-thread environment me safe.

2) Why is StringBuffer THREAD-SAFE?

Because **sab methods synchronized hote hain**.

Example:

```
public synchronized StringBuffer append(String str)
```

- synchronized → ek time me ek hi thread access karega
- situation:

Imagine do threads ek hi message ko update kar rahe hain —
StringBuffer ensure karega ki dono race condition create na kare.

3) Is StringBuffer fast or slow?

StringBuilder se **slow**
String se **fast**
Reason:

- Synchronized method → extra locking
- Extra overhead → slow performance

4) When to use StringBuffer?

Use karo jab:

✓ Multiple threads string pe kaam kar rahe ho
✓ Banking apps
✓ Ticket booking
✓ Payment processing
✓ Logging system

5) Common StringBuffer Methods

Same as StringBuilder:

Method	Work
append()	Add text
insert()	Insert at index

replace()	Replace range
delete()	Remove characters
reverse()	Reverse string
capacity()	Internal buffer size

6) Example (Real-Code)

```
public class BufferDemo {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Hello");

        sb.append(" World");
        sb.insert(0, "Java ");
        sb.replace(5, 10, "Beautiful");
        sb.reverse();

        System.out.println(sb);
    }
}
```

7) Proof: StringBuffer mutable

```
StringBuffer sb = new StringBuffer("A");

System.out.println(System.identityHashCode(sb));
sb.append("B");
System.out.println(System.identityHashCode(sb));
```

Same hashcode → **mutable**

8) StringBuilder vs StringBuffer — Quick Practical Difference

Feature	StringBuilder	StringBuffer
Thread Safety	❌ No	✓ Yes
Performance	Fastest	Slower
Use Case	Single-thread	Multi-thread
Introduced	Java 5	Java 1.0

Interview One-Liner

“StringBuffer mutable + synchronized hota hai isliye thread-safe hai,
but StringBuilder mutable + non-synchronized hota hai isliye faster hai.”

PART–4 : PACKAGES IN JAVA

Java me **packages** ek bahut important topic hai —
Interview me bhi poocha jata hai, aur projects me *daily use* hota hai.

1) What is a Package?

Package = Folder in Java

Java me package ek tarah ka **directory / folder** hota hai jisme hum **classes, interfaces, enums** ko logically group karte hain.

Simple definition:

“Package is a way to organize related classes into folders.”

2) Why Packages Are Required? (Interview Reasons)

1. Code Organization

Jaise tum mobile ke folders use karte ho—
Photos, Videos, Documents—
Waise hi Java me classes ko organize karne ke liye packages hote hain.

Example:

- models
- services
- controllers
- Utils

2. Avoiding Class Name Conflicts

Imagine 10 developers same project me "Employee" class bana dete—
Name collision ho jayega.

Packages solve this:

```
company.hr.Employee
company.payroll.Employee
```

Dono ka name same, package different → No conflict.

3. Access Control

Access modifiers (`public`, `protected`, `default`, `private`) ka behavior **package based** hota hai.

Example:

- `default` — same package me accessible
- `protected` — different package me only through inheritance

4. Reusability

Ek package ko import karke kahi bhi use kar sakte ho.

3) Types of Packages

1. Built-in packages (Java ke ready-made)

Examples:

- `java.util`
- `java.io`
- `java.lang`
- `java.sql`
- `java.time`

2. User-defined packages (Hum banate hain)

Example:

```
package myapp.models;
```

4) How to create a package?

Java file ke **top** me likho:

```
package myapp.models;
```

This means class belongs to:

```
myapp/models/
```

5) How to use a class from another package?

Using **import**:

```
import myapp.models.Employee;
```

6) BEST EXAMPLE (Most important)

Tumne already Q30 kiya — main yahan **super-detailed version** de raha hoon.

FOLDER STRUCTURE

```
src/
 ├── myapp/
 │     ├── models/
 │     │       └── Employee.java
 │     └── MainApp.java
```

Employee.java (In models package)

```
package myapp.models;

public class Employee {

    private int id;
    private String name;
    private double salary;

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public void display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
```

```
    }
}
```

MainApp.java (Different package)

```
package myapp;

import myapp.models.Employee;

public class MainApp {
    public static void main(String[] args) {
        Employee emp = new Employee(101, "Rahul", 60000);
        emp.display();
    }
}
```

7) Package = Directory Structure Rule

```
package a.b.c;
```

Means folder structure:

```
a/b/c/
```

8) Default Package

Agar tum package nahi likhte, to Java tumhari file ko **default package** me rakh deta hai.

But production code me “default package” **avoid** karte hain.

9) Sub-Packages

Java me dotted notation ka har part ek folder hota hai:
Example:
`com.company.project.module.utils`

Means:

```
com/
   company/
```

```
project/
      module/
            utils/
```

10) Interview One-Liners

What is a package?

“Package is a way to group related classes to avoid name conflicts and improve code organization.”

Why package?

“Better organization, security, reusability, and conflict-free development.”

Can we access default members outside the package?

No.

Can we import sub-packages automatically?

No.
`import java.util.*;` does NOT import `java.util.concurrent.*`.