Module 1: Introduction to Java Fundamentals

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

February 9, 2026

Contents

Why Not Always Use String?

Strings in Java are **immutable**.

Why Not Always Use String?

Strings in Java are **immutable**.

- Cannot change existing content
- Every change creates a new object

Why Not Always Use String?

Strings in Java are **immutable**.

- Cannot change existing content
- Every change creates a new object

Problem:

- Waste of memory
- Slower for repeated changes

Solution — Mutable Strings

Java provides:

Solution — Mutable Strings

Java provides:

- **StringBuffer**
- **StringBuilder**

Solution — Mutable Strings

Java provides:

- **StringBuffer**
- **StringBuilder**

They are:

- Mutable
- Faster for modifications

Creating StringBuffer and StringBuilder

```
StringBuffer sb1 = new StringBuffer("Hello");
StringBuilder sb2 = new StringBuilder("Hello");
```

append() Method

```
StringBuilder sb = new StringBuilder("Java");
sb.append(" Programming");

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

StringBuffer vs StringBuilder

- StringBuffer $\rightarrow$ safer

StringBuffer vs StringBuilder

- StringBuffer $\rightarrow$ safer
- StringBuilder $\rightarrow$ faster

StringBuffer vs StringBuilder

- StringBuffer $\rightarrow$ safer
- StringBuilder $\rightarrow$ faster

Rule:

- Single-user program $\rightarrow$ StringBuilder

Tricky Understanding

- String $\rightarrow$ creates new object

Tricky Understanding

- String $\rightarrow$ creates new object
- Buffer/Builder $\rightarrow$ modifies same object

Tricky Understanding

- String $\rightarrow$ creates new object
- Buffer/Builder $\rightarrow$ modifies same object

Think: Paper vs Whiteboard

Understanding

- Create a `StringBuilder` with `"Hello"`, append `" World"`, and print it.
- Create a `StringBuffer` with `"Java"`, append `" Programming"`, and print its length.
- Reverse the string `"MADAM"` using `StringBuilder`.
- Insert the word `"Programming "` into `"I Love Java"` using `StringBuilder`.
- Remove numeric characters from `"Java123"` using `StringBuffer`.

Understanding

1. Predict the output:

```
String s = "Hello";
s.concat(" World");
System.out.println(s);
```

2. Predict the output and explain why the result occurs.

Understanding

1. Predict the output:

```
StringBuilder sb1 = new StringBuilder("Hi");
StringBuilder sb2 = sb1;
sb2.append(" All");
System.out.println(sb1);
```

2. Which is faster for repeated string modification: `String`, `StringBuffer`, or `StringBuilder`? Why?

Math Class — What is it?

The **Math class** provides ready-made mathematical functions.

Math Class — What is it?

The **Math class** provides ready-made mathematical functions.

- Square root
- Power
- Rounding
- Maximum / Minimum
- Random numbers

Math Class — Important Rule

You do **NOT** create an object of Math class.

Math Class — Important Rule

You do **NOT** create an object of Math class.
Correct:

- `Math.sqrt(25)`

Math Class — Important Rule

You do **NOT** create an object of Math class.

Correct:

- `Math.sqrt(25)`

Reason:

- All Math methods are **static**

Math Class — Common Methods

- `Math.abs(x)`

Math Class — Common Methods

- `Math.abs(x)`
- `Math.max(a, b)`

Math Class — Common Methods

- `Math.abs(x)`
- `Math.max(a, b)`
- `Math.min(a, b)`

Math Class — Common Methods

- `Math.abs(x)`
- `Math.max(a, b)`
- `Math.min(a, b)`
- `Math.sqrt(x)`

Math Class — Common Methods

- `Math.abs(x)`
- `Math.max(a, b)`
- `Math.min(a, b)`
- `Math.sqrt(x)`
- `Math.pow(a, b)`

Math Class — Rounding Methods

- `Math.round(4.6)` $\rightarrow$ 5

Math Class — Rounding Methods

- `Math.round(4.6)` $\rightarrow$ 5
- `Math.ceil(4.1)` $\rightarrow$ 5.0

Math Class — Rounding Methods

- `Math.round(4.6)` $\rightarrow$ 5
- `Math.ceil(4.1)` $\rightarrow$ 5.0
- `Math.floor(4.9)` $\rightarrow$ 4.0

Math Class — Rounding Methods

- `Math.round(4.6)` $\rightarrow$ 5
- `Math.ceil(4.1)` $\rightarrow$ 5.0
- `Math.floor(4.9)` $\rightarrow$ 4.0

Tip: Ceil = ceiling, Floor = floor

Math Class — Random Numbers

Method:

- `Math.random()`

Math Class — Random Numbers

Method:

- `Math.random()`

Range:

- $0.0 \leq \text{value} < 1.0$

Math Class — Random Numbers

Method:

- `Math.random()`

Range:

- $0.0 \leq \text{value} < 1.0$

Example:

- `(int)(Math.random() * 10)` $\rightarrow$ 0 to 9

Math Class — Example Program

```
public class MathExample {
 public static void main(String[] args) {

  System.out.println(Math.abs(-20));
  System.out.println(Math.sqrt(36));
  System.out.println(Math.pow(3, 2));
  System.out.println(Math.round(5.6));

  int r = (int)(Math.random() * 50);
  System.out.println(r);
 }
}
```

Wrapper Classes — Introduction

A **Wrapper class** converts a **primitive data type** into an **object**.

Wrapper Classes — Introduction

A **Wrapper class** converts a **primitive data type** into an **object**.
In simple words:

- Wraps primitive values into objects

Wrapper Classes — Introduction

A **Wrapper class** converts a **primitive data type** into an **object**.
In simple words:

- Wraps primitive values into objects

Example:

- `int` → `Integer`
- `double` → `Double`

Wrapper Classes — Why Needed?

Java is an **object-oriented language**.

Wrapper Classes — Why Needed?

Java is an **object-oriented language**.
But:

- Primitive types are NOT objects

Wrapper Classes — Why Needed?

Java is an **object-oriented language**.
But:

- Primitive types are NOT objects

Wrapper classes help when:

- Working with Collections
- Using Generics
- Converting data types

Primitive Types vs Wrapper Classes

Primitive	**Wrapper Class**
int	Integer
double	Double
char	Character
boolean	Boolean
float	Float
long	Long
byte	Byte
short	Short

Creating Wrapper Objects

```
Integer a = new Integer(10);   // old style
Integer b = Integer.valueOf(20);
```

Creating Wrapper Objects

```
Integer a = new Integer(10);   // old style
Integer b = Integer.valueOf(20);
```

Modern Java

```
Integer c = 30;   // autoboxing
```

Autoboxing

Autoboxing:

- Automatic conversion of primitive $\rightarrow$ object

Autoboxing

Autoboxing:

- Automatic conversion of primitive → object

Example:

- `Integer x = 10;`

Autoboxing

Autoboxing:

- Automatic conversion of primitive → object

Example:

- `Integer x = 10;`

Done by JVM automatically

Unboxing

Unboxing:

- Automatic conversion of object $\rightarrow$ primitive

Unboxing

Unboxing:

- Automatic conversion of object $\rightarrow$ primitive

Example:

- `int y = x;`

Wrapper Classes — Useful Methods

- `parseInt()`

Wrapper Classes — Useful Methods

- `parseInt()`
- `parseDouble()`

Wrapper Classes — Useful Methods

- `parseInt()`
- `parseDouble()`
- `valueOf()`

Wrapper Classes — Useful Methods

- `parseInt()`
- `parseDouble()`
- `valueOf()`
- `toString()`

String to Primitive Conversion

```
String s = "123";

int num = Integer.parseInt(s);
double d = Double.parseDouble("45.6");

System.out.println(num);
System.out.println(d);
```

Wrapper Classes — Tricky Understandings

- Wrapper classes are immutable

Wrapper Classes — Tricky Understandings

- Wrapper classes are immutable
- Autoboxing can cause `NullPointerException`

Wrapper Classes — Tricky Understandings

- Wrapper classes are immutable
- Autoboxing can cause `NullPointerException`
- `parseInt()` throws exception for invalid input

Wrapper Classes

```
public class WrapperDemo {
 public static void main(String[] args) {

  int a = 10;
  Integer obj = a;         // autoboxing

  int b = obj;             // unboxing

  String s = "100";
  int num = Integer.parseInt(s);

  System.out.println(obj);
  System.out.println(b);
  System.out.println(num);
 }
}
```

Wrapper Classes — Real-Life Usage

- Collections like ArrayList

Wrapper Classes — Real-Life Usage

- Collections like ArrayList
- Parsing user input

Wrapper Classes — Real-Life Usage

- Collections like ArrayList
- Parsing user input
- Generics and frameworks

Output

```
Integer a = 20;
Integer b = a;

b = 30;

System.out.println(a);
```

Output

```
String s = "50";
int x = Integer.parseInt(s);
System.out.println(x * 2);
```

Thank You!

Stay Connected

Premanand S

Email:	premanand.s@vit.ac.in
Phone:	+91-7358679961
LinkedIn:	linkedin.com/in/premsanand
Instagram:	instagram.com/premsanand
WhatsApp Channel:	anandsDataX
Google Scholar:	Google Scholar Profile
GitHub:	github.com/anandprems