Wrapper class

**String Basics ---String s=new String(“Remya”)**

string is basically an object that represents sequence of char values. An [array](https://www.tpointtech.com/array-in-java) of characters works as a string in Java. For example:

**char**[] ch={'p','o','i','n','t'};

String s=**new** String(ch);

**Java String** class provides a lot of methods to perform operations on strings such as **compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.**

The **java.lang.String** class implements ***Serializable*, *Comparable* and *CharSequence***[**interfaces**](https://www.tpointtech.com/interface-in-java)**.**



How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

String s1="Welcome";   object

String s2="Welcome";*//It doesn't create a new instance*

String s3=new String(“Remya”)



*String objects are stored in a special memory area known as the "string constant pool".*

StringBuffer

StringBuffer represents a mutable sequence of characters that ensures thread safety, making it suitable for scenarios involving multiple threads that modify a character sequence.

StringBuilder

StringBuilder shares similarities with StringBuffer by being a mutable character sequence. The crucial distinction lies in **StringBuilder not being synchronized**, rendering it not suitable for thread-safe operations. This absence of synchronization, though, contributes to StringBuilder offering superior performance in environments that are single-threaded or confined to a specific thread.

1. String s1 = "Hello"; *// String literal*
2. String s2 = "Hello"; *// Points to the same "Hello" in the Pool as s1*

String s3 = **new** String("Hello"); *// A new string object is created in the heap*

*In Java, the Java Virtual Machine (JVM) divides memory into different regions, with Stack and Heap being two primary areas, each serving distinct purposes:*

*Stack Memory:*

* ***Purpose:***

*The Stack is used for storing local variables (primitive types like int, char, boolean), method call information (like return addresses), and references to objects that reside in the Heap.*

* ***Structure:***

*It follows a Last-In-First-Out (LIFO) order, meaning the last item pushed onto the stack is the first one to be popped off.*

* ***Management:***

*Stack memory is automatically managed by the JVM. When a method is called, a new stack frame is created; when the method finishes execution, its stack frame is popped off, and the memory is automatically freed.*

* ***Characteristics:***

*It is generally faster for allocation and deallocation due to its LIFO nature and close CPU management. It is typically smaller in size compared to the Heap. A StackOverflowError can occur if the stack runs out of space, often due to excessive recursion.*

*Heap Memory:*

* ***Purpose:***

*The Heap is used for storing objects and their instance variables (fields). Any object created using the new keyword resides in the Heap.*

* ***Structure:***

*It does not follow a strict order like the Stack; memory allocation and deallocation can happen dynamically and in a less predictable manner.*

* ***Management:***

*Heap memory is managed by the Java Garbage Collector. Unreferenced objects in the Heap are automatically cleaned up by the Garbage Collector to reclaim memory.*

* ***Characteristics:***

*It is generally slower for allocation and deallocation compared to the Stack due to its dynamic nature and garbage collection overhead. It is typically larger in size and shared across the entire application. An OutOfMemoryError can occur if the Heap runs out of space, indicating insufficient memory for object creation.*

***1. Value Types***

*These store the* ***actual value*** *in the variable itself.  
They are not objects and are stored* ***directly in memory (stack)****.*

***🔹 Examples:***

*int x = 10;*

*double y = 20.5;*

*char c = 'A';*

*boolean flag = true;*

*Here, the variables x, y, c, and flag hold their actual data (10, 20.5, 'A', true).*

***🔹 Characteristics:***

* *Stored in* ***stack memory****.*
* *Hold* ***actual value****.*
* *Are* ***faster*** *and* ***memory efficient****.*
* *Default values (when class fields):*
  + *int → 0*
  + *double → 0.0*
  + *boolean → false*
  + *char → '\u0000'*

***🔹 Example:***

*int a = 10;*

*int b = a; // copy value of a into b*

*b = 20;*

*System.out.println(a); // 10*

*System.out.println(b); // 20*

*a is unaffected because the value is copied —* ***each has its own copy****.*

***2. Reference Types***

*These store* ***references (memory addresses)*** *to objects in the heap.  
They don’t hold the actual data, but a* ***reference (pointer)*** *to the object.*

***Examples:***

*String name = "Remya";*

*int[] numbers = {1, 2, 3};*

*MyClass obj = new MyClass();*

*Here:*

* *name, numbers, and obj store* ***references*** *(addresses) to objects in the* ***heap memory****.*

***Characteristics:***

* *Stored in* ***heap memory****, but reference variables live in* ***stack****.*
* *Hold a* ***reference (address)*** *to the object, not the actual data.*
* *Can be* ***null****.*
* *Are* ***objects*** *(can have methods, attributes).*

***Example:***

*int[] a = {1, 2, 3};*

*int[] b = a; // b references the same array as a*

*b[0] = 10;*

*System.out.println(a[0]); // 10*

*System.out.println(b[0]); // 10*

*Both a and b point to the* ***same array*** *in memory — changing one affects the other.*

OOPs (Object-Oriented Programming)

Object-oriented programming aims to implement real-world entities, for example, objects, classes, abstraction, inheritance, polymorphism, etc.

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-oriented programming** is a methodology or paradigm for designing a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.tpointtech.com/object-in-java)
* [Class](https://www.tpointtech.com/class-definition-in-java)
* [Inheritance](https://www.tpointtech.com/inheritance-in-java)
* [Polymorphism](https://www.tpointtech.com/polymorphism-in-java)
* [Abstraction](https://www.tpointtech.com/abstraction-in-java)
* [Encapsulation](https://www.tpointtech.com/encapsulation-in-java)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class.

Class

Class is a logical entity. A class can also be defined as a blueprint from which we can create an individual object. Class does not consume any space.

**Why OOP?** Improves modularity, reusability, and maintainability.

**Classes vs Object**

* **Class:** A blueprint or template describing properties (fields) and behaviors (methods).
* **Object (Instance):** A concrete realization of a class with actual values.

public class Person {

String name;

int age;

void display()

{

}

}

// Using the class

Person p = new Person(); // p is an object/instance

p.name = "Alice";

p.age = 30;

p.display();

**Classes - Members (Fields, Methods, Constructors)**

**Members of a class** include:

* **Fields (instance variables)** — store state.
* **Methods** — behavior/actions.
* **Constructors** — special methods for object initialization.
* **Nested types** (inner classes, enums), static initializers.

public class Rectangle {

private double width; // field

private double height; // field

// Constructor

public Rectangle(double w, double h) {

this.width = w;

this.height = h;

}

// Method

public double area() {

return width \* height;

}

// Overloaded constructor

public Rectangle(double side) { this(side, side); }

}

**Static Members**

**Static field/methods** belong to the class rather than any instance.

* Shared across all instances.
* Useful for constants and utility methods.

Single inheritance -----one class inherits from another class

Multilevel ---a class inherits from another class,which itself inherits from another

Hierrarchial ---multiple clasees inherit from the same parent class

Multiple inheritance ----not possible in java

One class inherits from two or more classes

Hybrid----combination of two or more inheritance types

Object class--------super class of all classes

Default its is extending our object class……

Methods toString()---- object to human reader

Hashcode()

Non Access specifier

-provide additional information about your class ,method or variable

Final String name=”revature”

Class ---final ,abstract

Methods -----final,static,synchronied,native

Native

Software1 --------C++ /C

Software2-------Java

----need to use some methods from software1 –diffferent

Native void add(){}-----declares amethod which is implemented in another language

Strictfp---ensure floating point calculation precision

Variables ---final ,static ,transient and volatile

Serialization------object over the network byte stream

So many values ---ipadd,ifcongi,pwd-----

Pwd ---transient

Volatile ----

Web project ----jsp /servlet ,EJB

Framework -----struts1 struts 2 ------

Framework spring MVC -----download jar files folder ---add to it ---hectic task

Dependency in our doenload the needed jar file its corresponding version

Tomcat server ----built in our ide -----

Create set up configuration ----war file ---put in our server

**What is Maven?**

Maven is a **build automation and project management tool** used primarily for **Java projects**.  
It simplifies compiling, packaging, testing, and deploying Java applications.

**Key Features**

* **Build Automation:** Handles compilation, packaging (JAR/WAR), and deployment automatically.
* **Dependency Management:** Automatically downloads required JAR files from repositories.
* **Standard Project Structure:** Follows a predefined directory layout (src/main/java, src/test/java).
* **Integration with IDEs:** Works smoothly with Eclipse, IntelliJ, VS Code, etc.
* **Lifecycle Management:** Handles build phases like clean, compile, test, package, install, deploy.

**Maven Lifecycle Phases**

| **Phase** | **Description** |
| --- | --- |
| **validate** | Checks project structure and configurations |
| **compile** | Compiles source code |
| **test** | Runs unit tests |
| **package** | Packages code into JAR/WAR |
| **install** | Installs artifact in local repository |
| **deploy** | Uploads to remote repository (production/distribution) |

**Advantages**

* Automates repetitive build tasks.
* Reduces dependency conflicts.
* Supports multi-module projects.
* Ensures consistent builds across environments.

**🧩 2️⃣ Central Repository**

**What is it?**

The **Central Repository** (also called **Maven Central**) is an **online database of open-source libraries** maintained by the Maven community.

When you declare dependencies in pom.xml, Maven downloads them automatically from **Central Repository** and stores them in your **local repository** (on your computer).

**Types of Repositories**

| **Type** | **Description** |
| --- | --- |
| **Local Repository** | Stored on your system (usually in .m2/repository). |
| **Central Repository** | Main public repository managed by Apache Maven. |
| **Remote Repository** | Company-specific or internal repository (e.g., Nexus, Artifactory). |

**What is POM?**

POM stands for **Project Object Model** — the **core configuration file** for every Maven project.  
It’s an **XML file** named pom.xml located at the root of your project.

**Purpose**

* Defines project information (name, version, packaging type).
* Lists dependencies (external JARs).
* Specifies plugins, build settings, repositories, and modules.

**Basic Structure of pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId> <!-- Project group or company -->

<artifactId>myapp</artifactId> <!-- Project name -->

<version>1.0.0</version> <!-- Version -->

<packaging>jar</packaging> <!-- Output type: jar/war -->

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Important Elements**

| **Tag** | **Description** |
| --- | --- |
| **groupId** | Unique ID of your organization/project |
| **artifactId** | Name of your project/module |
| **Version** | Version of the project |
| **packaging** | Type of artifact (jar, war, pom) |
| **dependencies** | Libraries your project depends on |
| **build/plugins** | Custom build tools and tasks |