

String s =”hello” this String litteral store into String constant pool

And String s =new String ( Heap memory we save the data and copy of data story oin SCP as well)

Immutable objects are objects which once declared elements can’t be modified after it.

A diagram of a hello world

Description automatically generated

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

### Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

**Security:**

**Thread Safe:**

In Java, "@Repository" is an annotation used to indicate that a class is a repository, which is a component responsible for accessing and manipulating data from a database.

### ****What is the purpose of repository in Spring boot?****

In Spring Boot, a repository is used to manage data persistence and retrieval in a database. It provides an abstraction layer over the database and simplifies the process of interacting with it, allowing developers to focus on business logic rather than database operations

By using **@Autowired**, you don't need to manually create an instance of **UserRepository** in

**UserService**. Spring handles the dependency injection, making your code more modular, flexible, and easier to maintain.

### Abstract class in Java

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### **Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)
3. **abstract** **class** Bike{
4. Bike(){System.out.println("bike is created");}
5. **abstract** **void** run();
6. **void** changeGear(){System.out.println("gear changed");}
7. }
8. //Creating a Child class which inherits Abstract class
9. **class** Honda **extends** Bike{
10. **void** run(){System.out.println("running safely..");}
11. }
12. //Creating a Test class which calls abstract and non-abstract methods
13. **class** TestAbstraction2{
14. **public** **static** **void** main(String args[]){
15. Bike obj = **new** Honda();
16. obj.run();
17. obj.changeGear();
18. }
19. }

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method bod

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.
* **interface** Printable{
* **void** print();
* }
* **interface** Showable{
* **void** show();
* }
* **class** A7 **implements** Printable,Showable{
* **public** **void** print(){System.out.println("Hello");}
* **public** **void** show(){System.out.println("Welcome");}
* **public** **static** **void** main(String args[]){
* A7 obj = **new** A7();
* obj.print();
* obj.show();
* }
* }

|  |  |
| --- | --- |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |