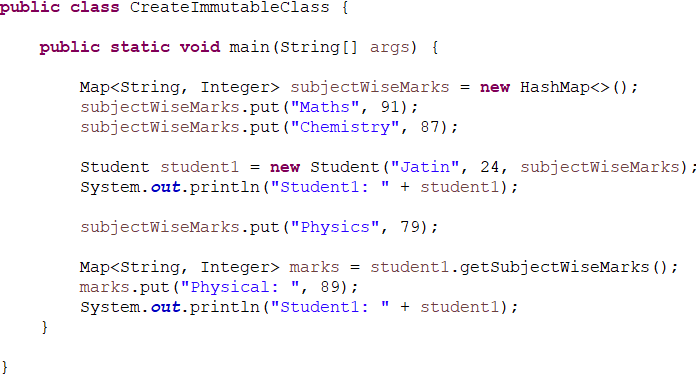
**How to create Immutable class in java?**

Immutable class in java means that once an object is created, we cannot change its content. In Java, all the [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well. Prior to going ahead do go through characteristics of immutability in order to have a good understanding while implementing the same. Following are the requirements:

* The class must be declared as final so that child classes can’t be created.
* Data members in the class must be declared private so that direct access is not allowed.
* Data members in the class must be declared as final so that we can’t change the value of it after object creation.
* A parameterized constructor should initialize all the fields performing a deep copy so that data members can’t be modified with an object reference.
* Deep Copy of objects should be performed in the getter methods to return a copy rather than returning the actual object reference).

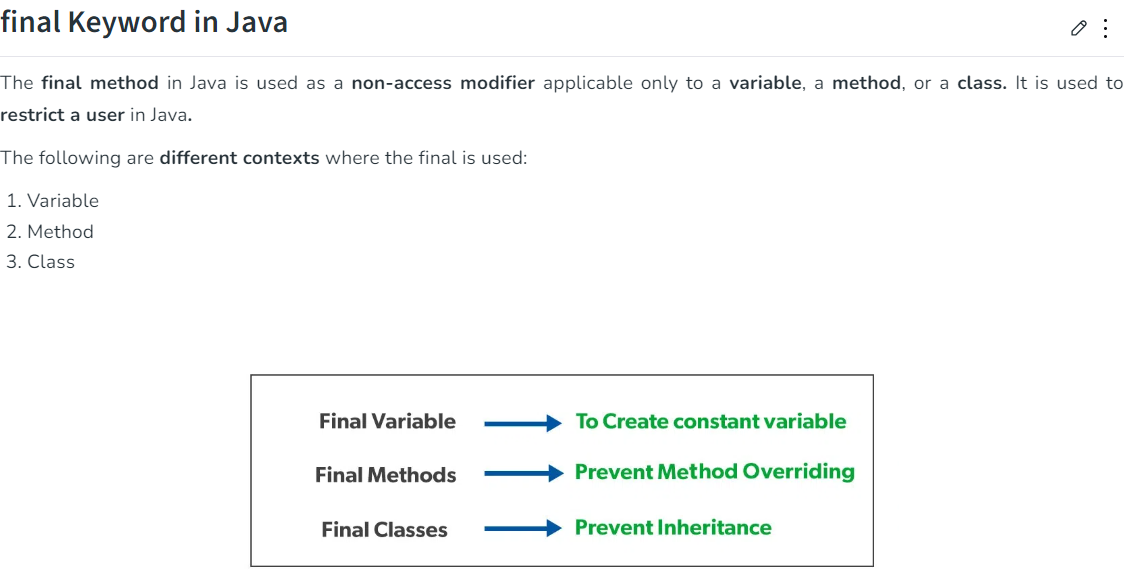
**Note:** There should be no setters or in simpler terms, there should be no option to change the value of the instance variable.





Output:



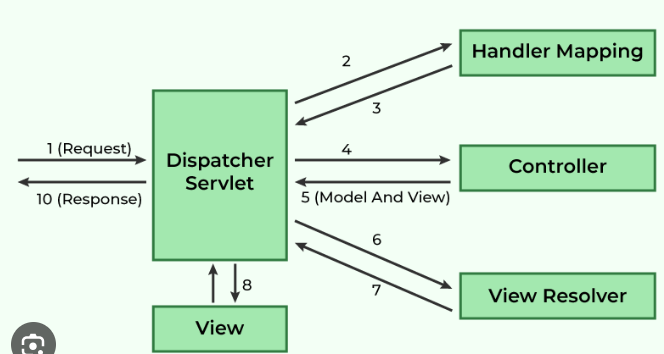


**Map and collection different:**

Collection- A collection represents a group of objects, known as its elements.   
Map-  A map cannot contain duplicate keys. Each key can map to at most one value.

**Front Controller:**

**DispatcherServlet** is the front controller in Spring Web MVC. Incoming requests for the HTML file are forwarded to the DispatcherServlet.

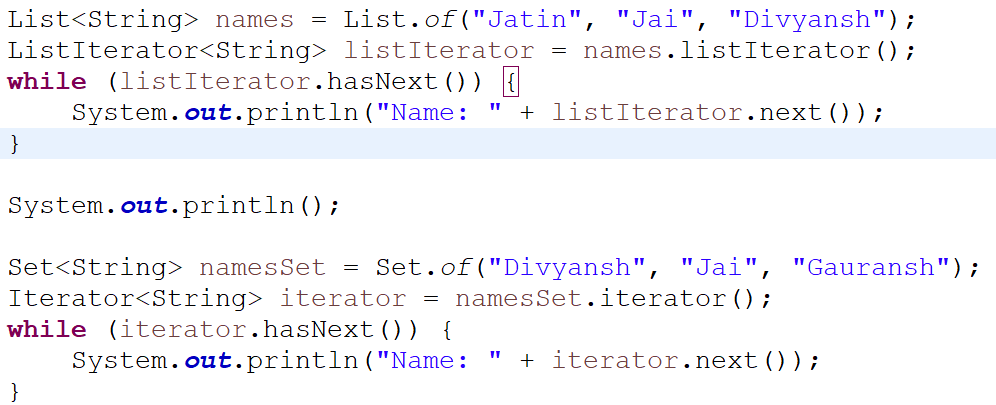


# Difference between an Iterator and ListIterator in Java:

**Iterators**are used in Collection framework in Java to retrieve elements one by one. It can be applied to any Collection object.

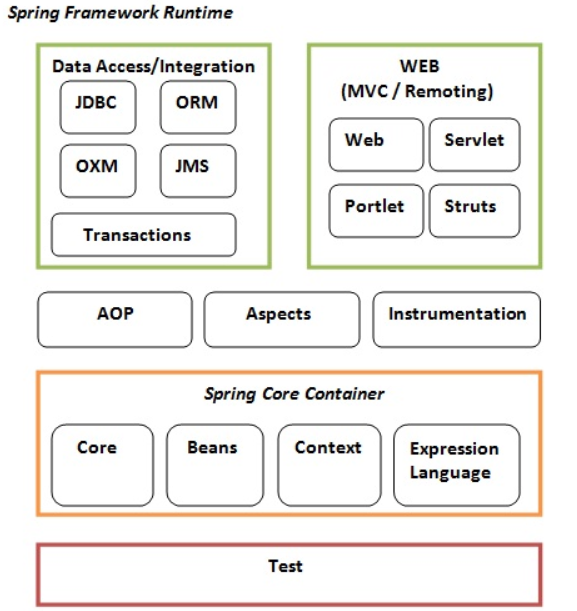
**ListIterator** It is only applicable for List collection implemented classes like [arraylist](https://www.geeksforgeeks.org/arraylist-in-java/), [linkedlist](https://www.geeksforgeeks.org/linked-list-in-java/) etc.

By the help of ListIterator you can iterator in reverse order too

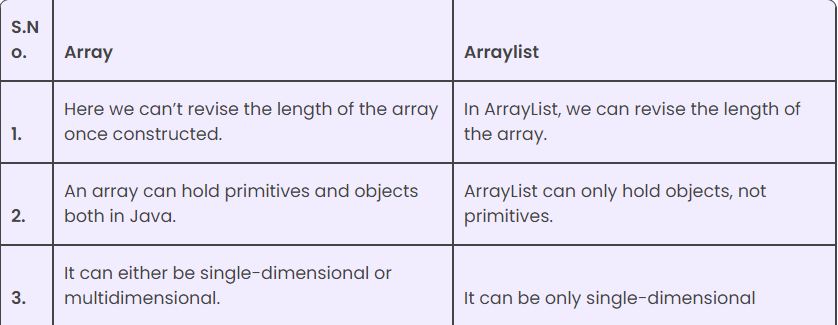


**Spring Modules:**

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web as displayed in the following diagram.



# Array vs ArrayList in Java:



**Benefits in Generic:**

Java is a popular programming language known for its flexibility, reliability, and safety features. One of the key features that make Java a versatile language is its support for generics. Generics in Java provide a way to create type-safe classes, methods, and interfaces, which can work with any data type.

**Type safety, Code reuse, Improved performance, Greater flexibility**

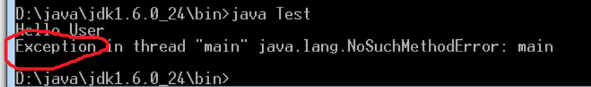
# Is main method compulsory in Java?

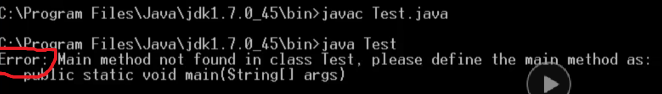
The answer to this question depends on the version of java you are using. Prior to JDK 7, the main method was not mandatory in a java program.

* You could write your full code under [static block](https://www.geeksforgeeks.org/g-fact-79/) and it ran normally.
* The static block is first executed as soon as the class is loaded before the main(); the method is invoked and therefore before the main() is called. main is usually declared as static method and hence [Java doesn’t need an object to call the main method.](https://www.geeksforgeeks.org/jvm-create-object-main-class-class-contains-main/)
* When you will give the run command(i.e java Test in the below-mentioned program in notepad), so compiler presumes Test is that class in which main() is there and since compiler load, the main() method, static blocks are ready to get executed. So here, it will run static block first and then it will see no main() is there. Therefore it will give **“exception”**, as exception comes while execution. However, if we don’t want an exception, we can terminate the program by  
  System.exit(0);

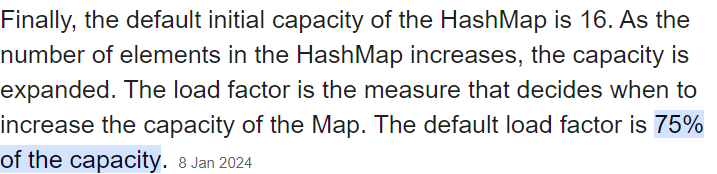
However, from JDK7 main method is mandatory. The compiler will verify first, whether main() is present or not. If your program doesn’t contain the main method, then you will get an **error** “main method not found in the class”. It will give an error (byte code verification error because in it’s byte code, main is not there) not an exception because the program has not run yet.

Before Java 7:



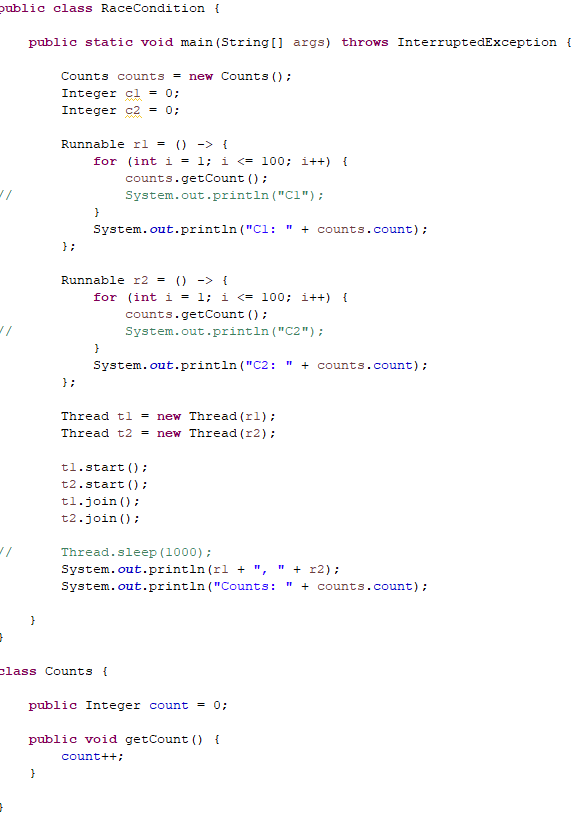
Java7,8,.:  


**Default value of hashmap capacity, load factor:**



**Race condition:**

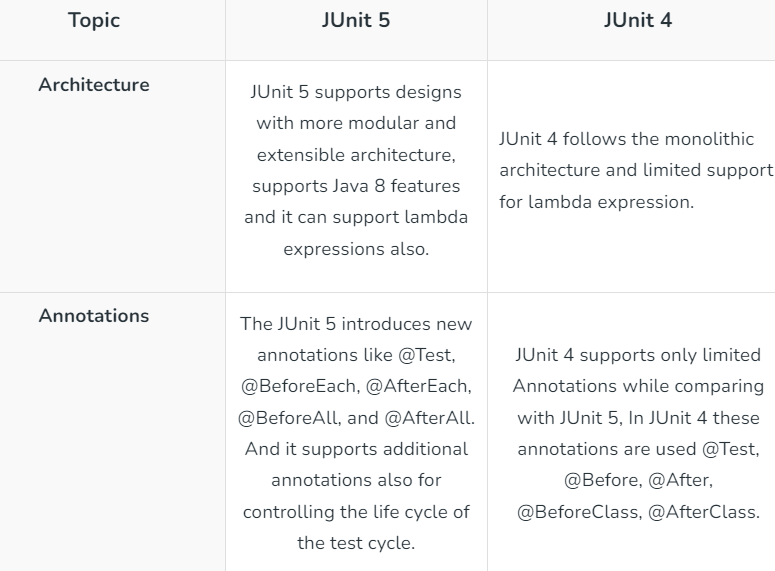
A race condition in Java emerges when two or more threads concurrently access shared data, and the final outcome hinges on the timing or order of their execution.



**Atomic operations:**

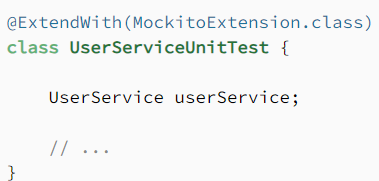
Atomic operations are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency. Mostly work for integer and Long,..

JUnit5 VS JUnit4



The @ExtendWith annotation aims to serve a different purpose compared to the @RunWith annotation. Instead of changing the test runner, @ExtendWith allows developers to register extensions, which can intercept the test execution lifecycle to add additional behavior or services.

@ExtendWith is a [repeatable](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Repeatable.html) annotation that is used to register [extensions](https://junit.org/junit5/docs/5.8.0/api/org.junit.jupiter.api/org/junit/jupiter/api/extension/Extension.html) for the annotated test class, test interface, test method, parameter, or field.



Here you registered “MockitoExtension.class” in your test class, so mockito extension will work in current class.

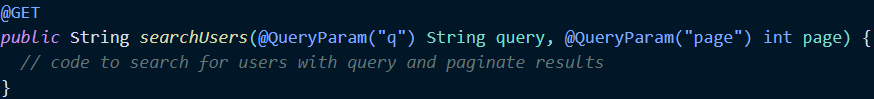
**@PathParam, @PathVariable, @QueryParam, @RequestParam**

@QueryParam:

The @QueryParam annotation is used to bind a query parameter to a method parameter.

eg:

**https://example.com/search?q=java&page=1**, the **q** and **page** parameters are query parameters.

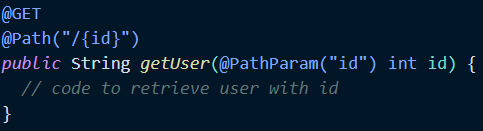


@PathParam

The @PathParam annotation is used to bind a path parameter to a method parameter.

eg:

**https://example.com/users/123**, the **123** parameter is a path parameter



@QueryParam is same as @RequestParam

@PathParam is same as @PathVariable

But the difference is that:

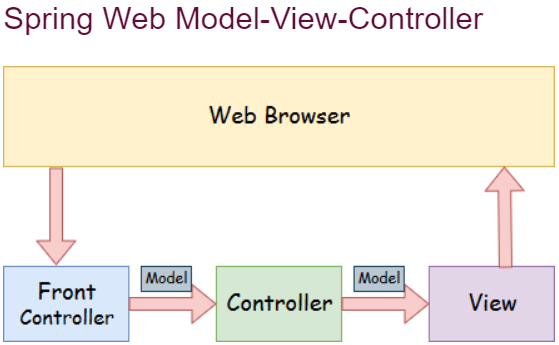
@RequestParam and @PathVariable is from spring

**import** org.springframework.web.bind.annotation.\*;

@QueryParam and @PathParam is from jakarta

**import** jakarta.websocket.server.\*;

**Spring MVC:**



IOC: **Inversion of control** is the core concept in spring framework. We give the object creation control to framework to create/instantiate object, configure object and inject object to other classes.

There are two types of IOC containers.

BeanFactory

ApplicationContext

ApplicationContext is built on top of BeanFactory interface to provide some additional functionality. It is an extension of BeanFactory.

Implementation classes of ApplicationContext(I)

a. FileSystemXmlApplicationContext(standalone)

b. ClassPathXmlApplicationContext(standalone)

c. XmlWebApplicationContext(SpringMVC apps)

d. AnnotationConfigApplicationContext(Standaloneapp's)

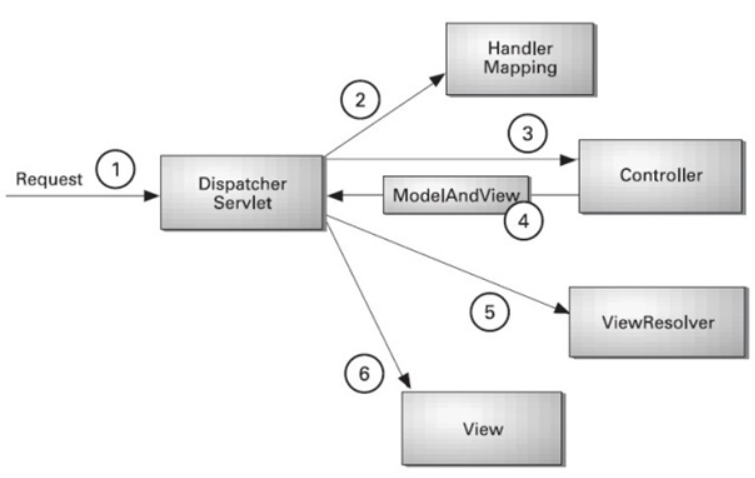
e. AnnotationConfigWebApplicationContext(SpringMVC apps)

* **Model** - A model contains the data of the application. A **data can be a single objec**t or a collection of objects.
* **Controller** - A controller contains the **business logic** of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.
* **ViewResolver:** It is used for mapping the view name with actual view. It contains prefix amd suffix

prefix is store the folder path(/WEB-INF/view/) and suffix is the file extension(.jsp, .html).

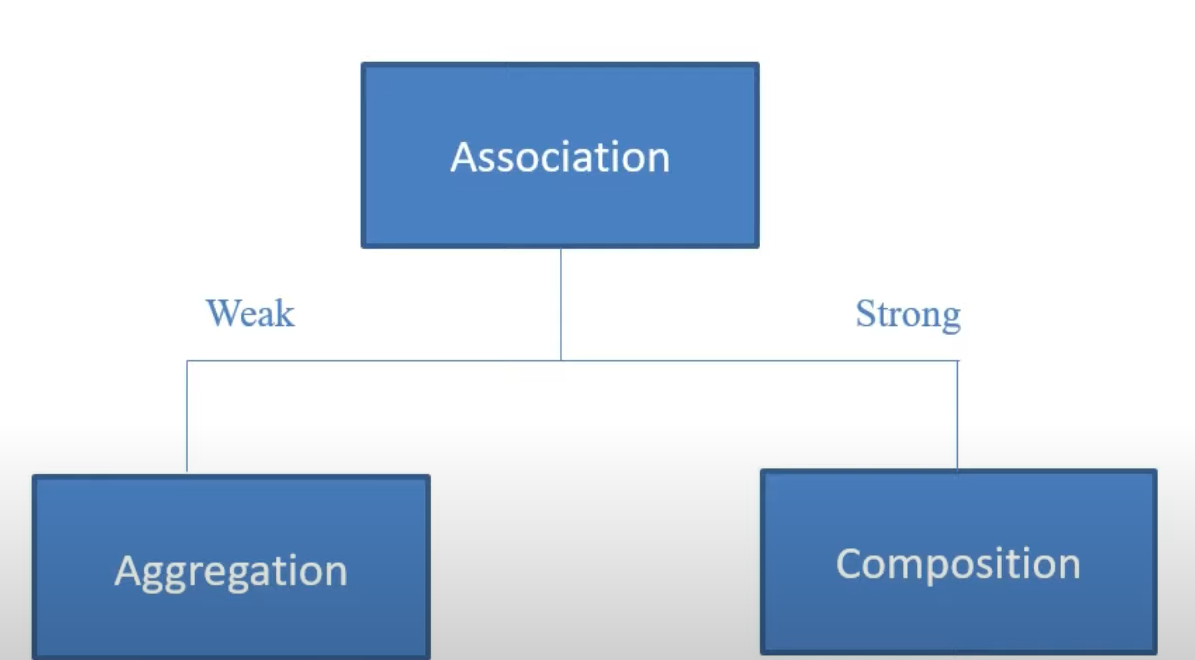
**Working:**

When we hit request to our service it first goes to front controller which is basically a dispatcher servlet which transfer the request to required controller based on urlPattern and in controller we are doing business logic and use model to store any data which can be shown on page using JSP+JSTL. It returns string which is basically a webpage name(view) eg:(home). Then ModelAndView data is send to dispatcher servlet and it send to viewResolver which used to map the return view(string) name to the actual view present in prefix folder and return the web page(home.jsp) to user.



**HandlerMapping:** Use to map the url to correct controller.

**Association, aggregation and composition:**



Association is a relation between two separate classes which is established through their Objects. Composition and Aggregation are the two forms of association.

**Association are a HAS A relationship. In inheritance we have IS A relationship.**

**Weak(Aggergation):** Here weak connection means class can exist without other class and other class can also exist without another class. **It is a HAS A relationship.**

Eg:

public class Driver{

private Car car;

}

Driver can exist without car and car can also exist without driver.

**Strong(Composition):** When a class cannot exist with another class. **It is a PART OF relationship.**

Eg:

public class Car{

private Engine engine;

}

Engine cannot exist without a car and car can aso not exist with engine.

If we damage car then engine will also damage due to strong connection. But if we damage car can it will not damage driver as driver is not a part of car.

**Why field injection is not recommended?**

* **Null-Safety:** Field injection creates a risk of *NullPointerException* if dependencies aren’t correctly initialized.

## Immutability: Using the field injection, we are unable to create immutable classes. Field injection creates a risk of *NullPointerException* if dependencies aren’t correctly initialized.

## Design Problems

### Single Responsibility Violation: We can easily add more dependencies than necessary and create a class that’s doing more than one job.

### Circular Dependencies: Since the dependencies are injected when needed and not on the context load, Spring won’t throw *BeanCurrentlyInCreationException*.

**Design Pattern:**

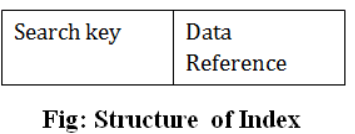
Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

How Indexing works?

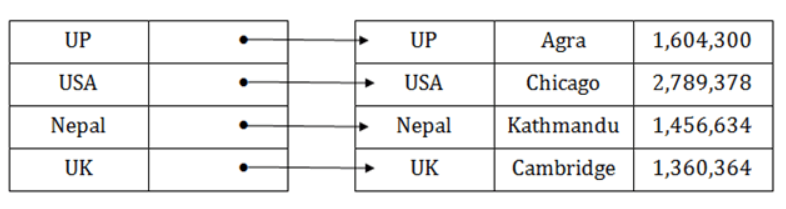


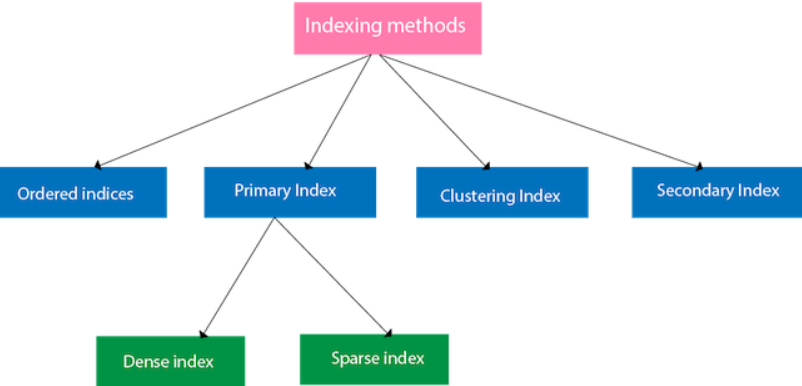
* Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
* The index is a type of data structure. It is used to locate and access the data in a database table quickly.

**Structure:**



* The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
* The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.





<https://www.javatpoint.com/indexing-in-dbms>

# How do you choose between SQL and NoSQL databases for your projects?

SQL databases are suitable for certain types of projects due to their advantages. These include a clear and predefined schema that ensures data integrity and consistency, as well as support for ACID (Atomicity, Consistency, Isolation, Durability) properties that guarantee reliable and secure transactions. Additionally, SQL databases allow complex and analytical queries that can join multiple tables and perform aggregations and calculations. Furthermore, they have a mature and standardized language that is widely used and supported by many tools and frameworks.

NoSQL databases offer several advantages over SQL databases, such as a schemaless or dynamic schema, BASE (Basically Available, Soft state, Eventual consistency) properties, simple and fast queries, and a variety of data models. This makes them more flexible and scalable, allowing them to store and handle different types of data and structures, scale horizontally and distribute across multiple servers, and access and manipulate data without joining multiple tables or performing complex operations.

[https://www.linkedin.com/advice/1/how-do-you-choose-between-sql-nosql-databases?src=go-pa&trk=sem-ga\_campid.20316911727\_asid.154319842041\_crid.663989285736\_kw.\_d.c\_tid.dsa-2089354945817\_n.g\_mt.\_geo.1007765&mcid=7080236969011671041&cid=&gad\_source=1&gclid=EAIaIQobChMIo\_aSv4rshQMVlKRmAh2gRwfeEAAYASAAEgL40PD\_BwE&gclsrc=aw.ds](https://www.linkedin.com/advice/1/how-do-you-choose-between-sql-nosql-databases?src=go-pa&trk=sem-ga_campid.20316911727_asid.154319842041_crid.663989285736_kw._d.c_tid.dsa-2089354945817_n.g_mt._geo.1007765&mcid=7080236969011671041&cid=&gad_source=1&gclid)

**Singleton class**

**Use Cases and Benefits of Singleton Class in Java**

Singleton classes in Java are useful when you need only one class instance to control shared resources or ensure consistent behavior. They manage database connections, configuration settings, and thread pools. Singleton can simplify access, reduce memory consumption, and maintain data integrity if you want a global point of control in your program. It prevents multiple instances that could cause conflicts.

**When To Use Singleton Class And When To Avoid It?**

Using a singleton class in Java is suitable when you require exactly one instance of a class throughout your program's lifecycle. This is particularly helpful for managing resources such as database connections, logging systems, or configuration settings. Singleton ensures that these resources are easily accessible and consistent across the application.

However, it's important to avoid using singletons when they're unnecessary or could hinder your code's flexibility. Avoid them if they lead to a global state that's difficult to manage or if they complicate unit testing due to tightly coupled dependencies.

# How does singleton bean serve multiple requests at the same time in Spring?

It’s possible for Spring to use the same bean instance in multiple threads, firstly because for each thread, Java creates a private [stack memory](https://www.baeldung.com/java-stack-heap#stack-memory-in-java).

**The stack memory is responsible for storing the states of the local variables used inside methods during thread execution.** This way, Java makes sure that threads executing in parallel do not overwrite each other’s variables.

Secondly, because  bean sets no restrictions or locks at the heap level, **the**[**program counter**](https://www.baeldung.com/cs/process-control-block#2-program-counter)**of each thread is able to point to the same reference of the bean instance in the heap memory.**

[https://www.baeldung.com/spring-singleton-concurrent-requests#:~:text=The%20Java%20heap%2C%20as%20we,to%20the%20same%20bean%20instance](https://www.baeldung.com/spring-singleton-concurrent-requests#:~:text=The%20Java%20heap,%20as%20we,to%20the%20same%20bean%20instance).

<https://medium.com/@hasanli.vusala.73/how-does-singleton-bean-serve-multiple-requests-at-the-same-time-in-spring-f4c9d797dec9>

**URI and URL difference**

<https://medium.com/@abhirup.acharya009/uri-vs-urn-vs-url-key-distinctions-explained-dec8e02ebd18>

**RESTful APIs**

RESTful APIs (**Representational State Transfer APIs**) are a type of **web service API** that adhere to the principles of REST (Representational State Transfer) architecture. **REST is an architectural style** that is designed to take advantage of the existing protocols, most commonly HTTP, to **create scalable and easy-to-use web services**.

### Key Principles of RESTful APIs

* **Stateless**: Each request from a client to the server must contain all the information the server needs to fulfill that request. The server does not store any client context between requests.
* **Client-Server Architecture**: The client and server are separate entities that interact over a network. The client makes requests, and the server processes those requests and returns the appropriate responses.
* **Uniform Interface**: RESTful APIs have a standardized way of interacting with resources using HTTP methods:
* **GET**: Retrieve a resource.
* **POST**: Create a new resource.
* **PUT**: Update an existing resource.
* **DELETE**: Remove a resource.
* **Resources and URIs**: Resources are identified by URIs (Uniform Resource Identifiers). Each resource is represented by a specific URL.
* **Representation**: Resources can be represented in various formats such as JSON, XML, HTML, etc. The most common format is JSON.
* **Stateless Interactions**: All interactions are stateless. Each request from a client to a server must contain all the information needed to understand and process the request.

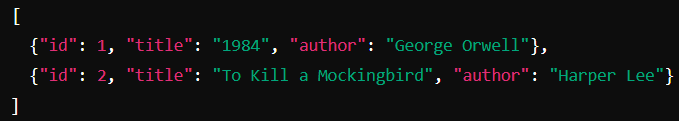
### Versioning: Use versioning to manage changes to the API without breaking existing clients.Common approaches include URL versioning (e.g., /v1/users) and header versioning.

**Example:**

Assume you are working with a RESTful API for a library system. Here are some example endpoints and their corresponding actions:

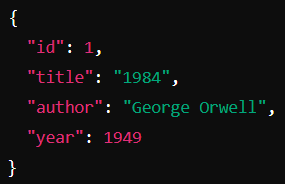
* **GET /books**

1. Retrieves a list of books.
2. Example response:



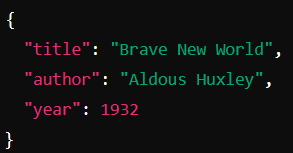
* **GET /books/1**

1. Retrieves details of a specific book by its ID.
2. Example response:

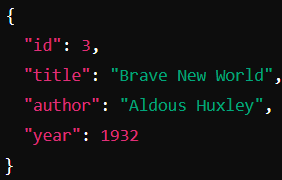


* **POST /books**

1. Creates a new book.
2. Request body:

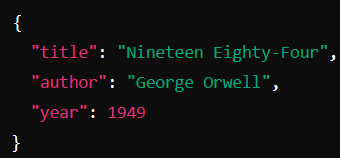


1. Example response:

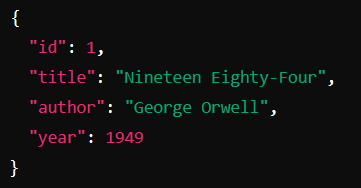


* **PUT /books/1**

1. Updates an existing book.
2. Request body:

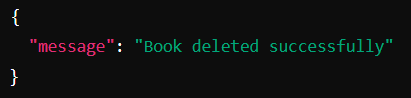


1. Example response:



* **DELETE /books/1**

1. Deletes a book by its ID.
2. Example response:



### Advantages of RESTful APIs

* **Scalability**: RESTful APIs are stateless, which makes them highly scalable.
* **Flexibility**: They can handle multiple types of calls, return different data formats, and even change structurally with the correct implementation of hypermedia.
* **Performance**: The stateless nature of REST can help in reducing server load and improving performance.

**HTTP status codes**:

**2xx: Successful:**

* **200** OK for successful requests.
* **201** Created for successful resource creation.
* **202** Accepted, The request is accepted for processing, but the processing is not complete.
* **204** No Content for successful deletions.

**3xx: Redirection:**

* **300** request has multiple possible responses and the user should choose one.

**4xx: Client Error:**

* **400** Bad Request for invalid requests.
* **401** Unauthorized for authentication errors.
* **402** client must make a payment to access the requested resource
* **403** Forbidden, Access is forbidden to the requested page
* **404** Not Found for non-existent resources.
* **405** Method not found(Incorrect http method)
* **408** Request Timeout, The request took longer than the server was prepared to wait

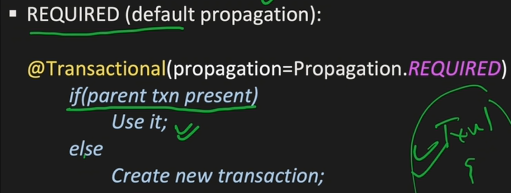
**5xx: Server Error:**

* **500** Internal Server Error for server errors.
* **502** Bad Gateway
* **503** Service Unavailable. The server is temporarily overloading or down.

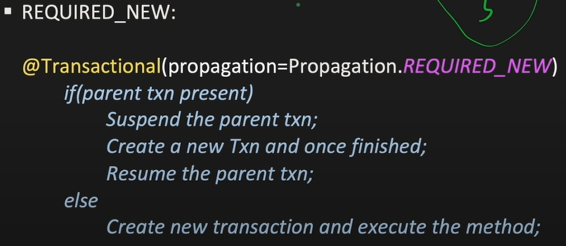
**Transaction:**

Propagation:

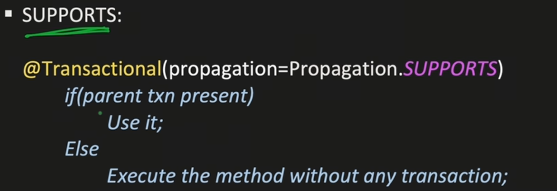
REQUIRED:



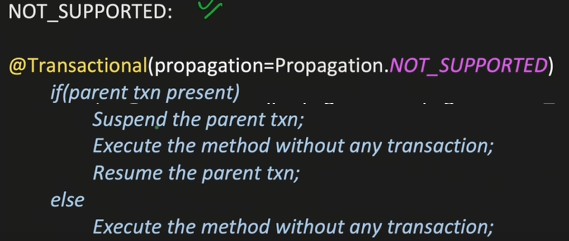
REQUIRED\_NEW:



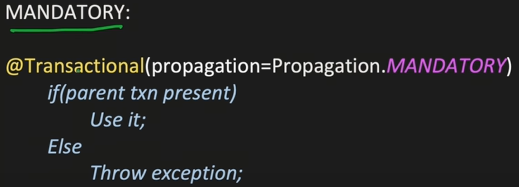
SUPPORTS:



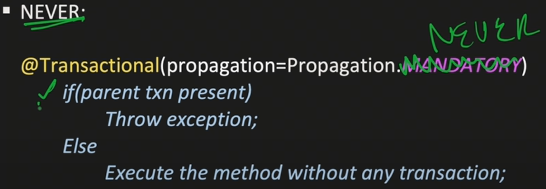
NOT\_SUPPORTED:



MANDATORY:



NEVER:



**ClassNotFoundException and NoClassDefFoundError**

**new Vs newInstance( )**

1. new is an operator to create an objects , if we know class name at the beginning then we can create an object by using new operator .

2. newInstance( ) is a method presenting class " Class " , which can be used to create object.

3. If we don't know the class name at the beginning and its available dynamically Runtime then we should go for

newInstance() method.

eg#1.

public class Test {

public static void main(String[] args) throws Exception {

//Take the input of the classname for which object has to be created at the runtime

String className = args[0];

//Load the class file explicitly

Class c = Class.forName(className);

//for the loaded class object is created using zero param constructor only

Object obj=c.newInstance();

//Perform type casting to get Student Object

Student std = (Student)obj;

System.out.println(std);

}

}

If dynamically provide class name is not available then we will get **ClassNotFoundException**. To use newInstance( ) method, compulsory corresponding class should contains no argument constructor, otherwise we will get "**InstantiationException**".

If the argument constructor is private then it would result in "**IllegalAccessException**".

**Note**: During typecasting, if there is no relationship b/w 2 classes as parent to child then it would result in "**ClassCastException**".

**Difference between new and newInstance( ):**

**new()**

new is an operator , which can be used to create an object.We can use new operator if we know the class name at the beginning.

**Test t= new Test( );**

If the corresponding .class file not available at Runtime then we will get RuntimeException saying **NoClassDefFoundError**,It is unchecked. To used new operator the corresponding class not required to contain no argument constructor.

**newInstance( )**

newInstance( ) is a method , present in class Class , which can be used to create an object . We can use the newInstance( ) method , If we don't class name at the beginning and available dynamically Runtime.

**Object o=Class.forName(arg[0]).newInstance( );**

If the corresponding .class file not available at Runtime then we will get RuntimeException saying **ClassNotFoundException** , It is checked. To used newInstance( ) method the corresponding class should compulsory contain no argument constructor , Other wise we will get RuntimeException saying **InstantiationException**.

**Difference between ClassNotFoundException & NoClassDefFoundError:**

1. For hard coded class names at Runtime in the corresponding .class files not available we will get **NoClassDefFoundError** , which is **unchecked**

**Test t = new Test( );**

In Runtime Test.class file is not available then we will get **NoClassDefFoundError**

2. For Dynamically provided class names at Runtime , If the corresponding .class files is not available then we will get the RuntimeException saying "**ClassNotFoundException**".

**Ex : Object o=Class.forname("Test").newInstance( );**

At Runtime if Test.class file not available then we will get the ClassNotFoundException , which is checked exception.

Note:

new will create a memory on the heap area

Student => JVM will search for Student.class file in Current Working Directory if found load the .class file data into MethodArea

**During the loading of .class file**

a. static variables will get memory set with default vaalue

b. static block gets executed

In the heap area, for the required object memory for instance variables is given by jvm, will set the default values.

a. Execute the instance block if available

b. call the constructor to set the meaningful values to the instance varaibles.

**JVM will give the address of the object to hashing algorithm which generates the hashCode for the object and that hashCode will be returned as the reference to the programmer**

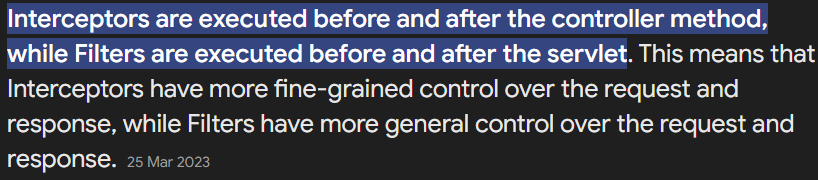
**new**() => requried class details known to compiler but not available at jvm then it would result in "NoClassDefFoundError"

**newInstance**() => requried class details not available at jvm then it would result in "ClassNotFoundException"

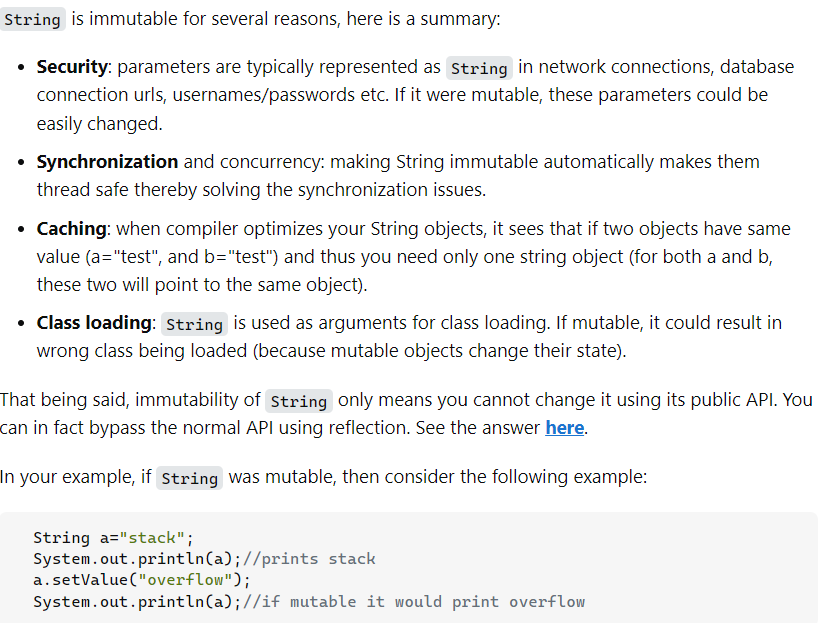
**How to make object unreferenced for garbage collections?**

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

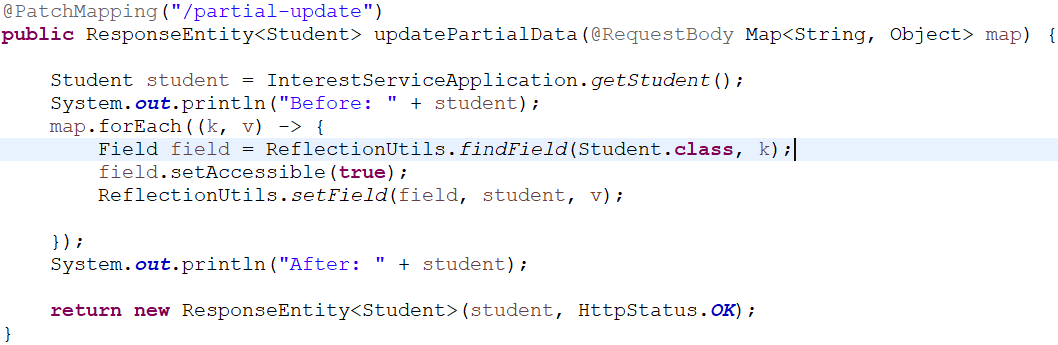
Intereptor:



**Why String is immutable?**



How to partially update data using patch mapping where field can we change?



Here We can use ReflectionApi to update fields present in Map as key.

### 1. Association

* Both classes are independent of each other. They can exist and function without each other.
* **Example**: A Teacher and a Student. A teacher can teach many students, and a student can have multiple teachers.

### 2. Aggregation

* In aggregation, one class (the whole) can contain references to objects of another class (the parts), but the parts can exist independently of the whole. So, while there is some dependency (the whole needs parts), the parts do not depend on the whole's lifecycle.
* **Example**: A Library and Book. A library can have multiple books, but books can exist independently of the library.

### 3. Composition

* Composition indicates a strong dependency where the part’s lifecycle is tied to the whole. If the whole is destroyed, the parts are also destroyed. However, it doesn’t mean that both classes are dependent on each other in the same way—only that the part cannot exist without the whole.
* **Example**: A House and Room. A house is composed of rooms, and if the house is destroyed, the rooms no longer exist.

### Corrected Definitions

* **Association**: Both classes are independent and can exist without each other.
* **Aggregation**: One class (the whole) may contain references to another class (the part), but the part can exist independently.
* **Composition**: The part cannot exist without the whole, indicating a strong lifecycle dependency.