**Spring**

The spring framework is a powerful and highly flexible framework focused on building web application in java. Spring makes programming in java quicker, easier for everybody. Its focus on speed, simplicity and productivity made it the world’s most popular java framework. First version of spring was released in 2002.

Spring core is the heart of spring framework which contains spring base classes, principles and mechanism.

It contains some important components:

* IOC (Inversion of Control)
* DI (Dependency Injection)
* Beans
* Context
* SpEl (Spring Expression Language)
* IOC Container

**Inversion of Control and Dependency Injection**

* Inversion of control (IOC) is a software design principle, independent of language, which does not create the object but describe the way in which object is being created.
* IOC is the principle where flow of the program is inverted instead of programmer controlling the flow of the program the framework or service takes control of the program flow.
* Dependency Injection is a pattern through which inversion of control achieved.
* Through dependency injection the responsibility of the object creation is shifted from the application to the spring IOC container. It reduces coupling between multiple objects as it is dynamically injected by the framework.

**Advantages of IOC and DI:**

* Loose coupling between the components.
* Increases system maintainability and module reusability.
* Minimize the amount of code in your application
* Allow the concurrent and independent development.
* Makes unit testing easy with different mocks.
* Replacing module has not side-effect on other modules.

**Spring Bean, Context, SpEl:**

* Any normal java class that is instantiated, assembled and managed by a Spring IOC container is called Spring Bean.

Spring Framework

POJO BEAN

* These beans are created with the configuration metadata that you supply to the container in the form of XML configurations and annotations.
* Spring IOC container manages the life cycle of spring Bean scope and injecting any required dependencies in the bean.
* Context is like a memory location in your application in which we add all the objects instances that we want the framework to manage. By default spring does not know any of the objects you define in your application. To enable spring to see your objects, you need to add them to the context.
* The SPEL provides a powerful expression language for querying and manipulating and object graph at runtime like setting and getting property values, property assignment, method invocation etc.

**IOC container is responsible for:**

* To initiating the application class.
* To configure the objects.
* To assemble the dependencies between the objects.

There are two types of IOC container.

1. Org.springframework.beans.factory.Beanfactory.
2. Org.springframework.context.ApplicationContext.

**Annotations**

1. **@Component:** This is one of the most commonly used stereotype annotation. Using this we can easily create and add a bean to the spring context by writing less code compared to the @Bean option. @component is used on the top of any java class it act as the base for other annotations.
2. **@Service:** This annotation is used on top of the class inside the service layer especially where we write the business logic and make external API calls.
3. **@Repository:** This annotation is used on top of the class where we handle the code related to database access and other operations like create, update and delete etc.
4. **@Controller:** This annotation is used on top of the class in controller layer of MVC application where we write code related to endpoint mapping with the corresponding methods.
5. **@PostConstruct:** This annotation is used on top of the method which instructs spring to execute that method after it finishes creation of bean.
6. **@PreDestroy:** This annotation is used on top of the method which instruct spring to execute that method just before clearing or destroying the context. This is used to close any IO resources or database connection.
7. **@Autowired:** This annotation is used on top of the field, setter, method and constructor. It is used to auto wire beans/object at runtime by spring dependency injection mechanism.
8. : This annotation is used to map either query parameters or form data.
9. **@PathVariable**: This annotation is used to extract the value from the URL.
10. **@ExceptionHandler** is used to handle the exception if this annotation is used in the controller which is annotated only @Controller annotation then it will handle only those exceptions which occurred within that specific class.
11. **@ControllerAdvice** is used on top of the class so that @ExceptionHandler annotated method can handle the exceptions globally.

**Autowiring with multiple beans of same type:**

* By default spring tries auto wire with class type but this approach fails if the multiple beans have same class type then it will try to auto-wire based on the parameter name/field name that we use while configuring the auto-wiring annotation or it will look for the @Primary annotation for auto-wiring.

A Circular dependency will happen if two beans are waiting for each other to create inside the spring context in order to do auto-wiring. UnsatisfiedDependencyException occurred due to circular dependency

**Scope of the Beans**

1. **Singleton:** The instance of the object is created only once. By default bean scope is singleton in spring. Same instance is provided when we auto-wire inside my application (Eager Instantiation).
2. **Prototype:** Every time new instance of the object is being created and provided when we auto-wire or we request a reference of a bean.
3. **Request:**
4. **Session:**
5. **Application:**

**Aspect-Oriented Programming**

An Aspect is a piece of code executed by spring when we call specific method inside application.

**www (3w) Rule:**

**WHAT – Aspect**

**WHEN - Advice**

**WHICH – Pointcut**

**WHAT** code or logic the spring to execute when we call a specific method. This is called Aspect.

**WHEN** the spring needs to execute the given aspect for example it can be before or after the method call. This is called Advice.

**WHICH** method inside the application that framework needs to intercept and execute the given aspect. This is called pointcut.

**JoinPoint**: It defines the event that triggers the execution of Aspect.

**Target-Object**: It is the bean that declares the method/pointcut which is intercepted by an aspect.

Aspect

Pointcut

Developer want some logic to be executed before each execution of method playMusic() present inside the VechicleServices class.

Target Object

Joinpoint

**Type of Advice:**

* @After
* @Before
* @AfterReturning
* @AfterThrowing

**SpringBoot**

* Springboot was developed in April 2014 to reduce some of the burdens while developing a Java web application.
* Before SpringBoot developer needs to configure a servlet container (Web server : Jboss), establish link between Tomcat and Dispatcher servlet deploy into server, define a lot of dependencies.
* Springboot takes care of all the configuration. Developers don’t need to worry about it.
* Springboot embed a web server so that we do not require an external application server.
* Springboot provides several useful production ready features out of the box to monitor and manage the application.

**Lombok:**  This is a java library which provides many annotations to reduce the boilerplate code in the application like @Getter, @Setter, @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor, @ToString, @Data and @EqualsAndHashCode

**@Data:** This is the most commonly used annotation that combines the feature of below annotations together.

* @ToString
* @Getter, @Setter
* @EqualsAndHashCode
* @RequiredArgsConstructor

**Monolithic Architecture:** Monolithic application has single code base with multiple modules.

* **Disadvantages:**
* As project scale it becomes difficult to manage.
* For a single change redeployment of whole application needed.
* Difficult to adopt new technology for single functionality.
* Single bug may down the whole application.
* **Advantages:**
* Simple to develop.
* Simple to build and deploy.
* Problem of network latency are relatively less.

**Microservices Architecture:** Microservices are small, loosely coupled distributed services. These smaller services communicate with each-other directly using light weight protocols like HTTP.

* **Advantages:**
* It is possible to change or upgrade each service individually rather than the entire application.
* One service may down without impacting to others.
* It allows to use different technologies for building different microservices.
* Less dependency, loosely coupled.

**Core Java**

**Oop’s Concept:** Object oriented programing is a methodology to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts.

There are 6 pillars of object oriented programming.

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation
7. **Object:** Any entity that has state and behavior is called object.
8. **Class:** Collection of objects is called class. It is a blueprint from which we can create an individual object.
9. **Inheritance:** It is a mechanism In which one object acquires all the properties and behaviors of a parent object.

* Inheritance is used for method overriding and code reusability.

**Class A**

**Class A**

**Class A**

**Class C**

**Class B**

**Class B**

**Class C**

**Class B**

1. **Single 2. Multilevel 3. Hierarchical**

**Class A**

**Class D**

**Class C**

**Class B**

**Class C**

**Class B**

**Class A**

**4. Multiple**

**5. Hybrid**

**Note:** Multiple inheritance is not supported in java due method ambiguity.

1. **Polymorphism:** If one task is performed in different ways it is known as polymorphism. We use method overloading and method overriding to achieve polymorphism.

* **Method Overloading:** If a class has multiple methods having same name but different in parameters it is known as method overloading. There are two ways to overload the method.

1. By changing the number of arguments.
2. By changing the data type.

**Note:** Method overloading is not possible by changing the return type of method only because of ambiguity.

Q. Can we overload java main method ?

Ans: Yes, But JVM calls main() method which receives string arrays as arguments only.

**Type promotion in method overloading:** One type is promoted to another implicitly if no matching data type is found.

**byte**

**short**

**char**

**float**

**int**

**double**

**long**

* **Method Overriding:** If child class has the same method as declared in the parent class. It is known as method overriding.

Usage of Java method overriding:

1. To provide the specific implementation of a method which is already provided by its superclass.
2. Method overloading is used for runtime polymorphism.

Q. Can we override static method?

Ans: No, A static method cannot be overridden because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area and an instance belongs to the heap area.

Q. Can we override the java main method?

Ans: No, because the main is a static method.

1. **Abstraction:** Hiding internal details and showing only functionality is known as abstraction. Abstraction can be achieved through abstract class and interface.

**Abstract Class:** 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.

* An abstract class must have declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructor and static method also.
* It can have final methods which will force the subclass not to change the body of the methods.

**Note**:

1. If there is any abstract method in the class that class must be abstract.
2. If you are extending an abstract class that has abstract method. You must either provide the implementation of the method or make this class abstract.

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

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance because it resolve the method ambiguity problem as implementation is provided by the implementation class.
* It can be used to achieve loose coupling.

**Note:**

* Since Java 8, we can have method body in interface, but we need to make it default method.
* Since java 8, we can have static method in interface.

1. **Encapsulation:** Encapsulation in java is a process of wrapping up code and data together into single unit for example a capsule which is mixed of several medicines. The java bean class is the example of fully encapsulated class.

**Advantages of encapsulation:**

* We can make the class read-only or write-only.
* It provides you the control over the data.
* It is a way to achieve data-hiding.
* The encapsulated class is easy to test.
* The standard IDEs are providing the facility to generate the getters and setters so that it is easy and fast to create an encapsulated class in java.
* **Static Keyword:-** The static keyword is used in java for memory management mainly we can apply static keyword with variables, methods, blocks and nested classes.

**Java Static variables:**

* The static variables can be used to refer the common property of all objects.
* The static variable gets memory only once in the class area at the time of class loading.
* It makes program memory efficient.

**Java Static methods:**

* The static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of the class.
* A static method can access static data member and change the value of it.

Q. Why is java main() method is static ?

Ans: It is because the object is not required to call a static method. If it were non-static method, JVM created an object first then call main() method that will lead the problem of extra memory overhead.

**Java Static blocks:**

* It is used to initialize the static data member.
* It is executed before the main method at the time of class loading.

Q. Can we execute a program without main() method ?

Ans: No, One of the ways was the static block but it was possible till JDK 1.6 Since JDK 1.7, It is not possible to execute the java class without the main method.

* **‘this’ keyword:-** In java, “this” is a reference variable that refers to the current object.
* ‘this’ can be used to refer current class instance variable.
* ‘this’ can be used to invoke current class method.
* ‘this()’ can be used to invoke current class constructor call.
* ‘this’ can be passed as argument in the constructor call.
* ‘this’ can be used to return the current class instance from the method.

**Note:** Call to this() must be the first statement in constructor.

* **super keyword:-** The super keyword in java is a reference variable which is used to refer immediate parent class object.
* ‘super’ can be used to refer immediate parent class instance variable.
* ‘super’ can be used to invoke parent class method.
* super() can be used to invoke immediate parent class constructor.

**Note:**

1. super() is added in each class constructor automatically by compiler if there is no super() or this().

* **final keyword:-** The final keyword is used to restrict the user. It can be used in many context.

Final can be used for below cases:

1. **Variable:** The value of final variable cannot be changed. It will be constant.
2. **Method:** Final method cannot be overridden.
3. **Class:** Java final class cannot be extended.

**Q:** Is final method inherited ?

**Ans:** Yes, final method is inherited but you cannot override it.

**Q:** Can we initialize blank final variable?

**Ans:** Yes, but only in constructor.

**Note:** A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

**Q:** Can we declare a constructor final ?

**Ans:** No, because constructor is never inherited.

**Exception Handling**

**Errors:** Errors are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash. Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:

* **OutOfMemoryError:** Thrown when the Java Virtual Machine (JVM) runs out of memory.
* **StackOverflowError:** Thrown when the call stack overflows due to too many method invocations.
* **NoClassDefFoundError:** Thrown when a required class cannot be found.

**Exception:** In java, an exception is an event that disrupts the normal flow of the program. It is an object which is known at runtime.

**Exception Handling:** Exception handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException etc. The core advantage of exception handling is to maintain the normal flow of the application.

**Unchecked Ex**

**Checked Exception**

**Note:**

1. Checked exception are checked at compile time.
2. Unchecked exception are not checked at compile-time, but they are checked at run time.

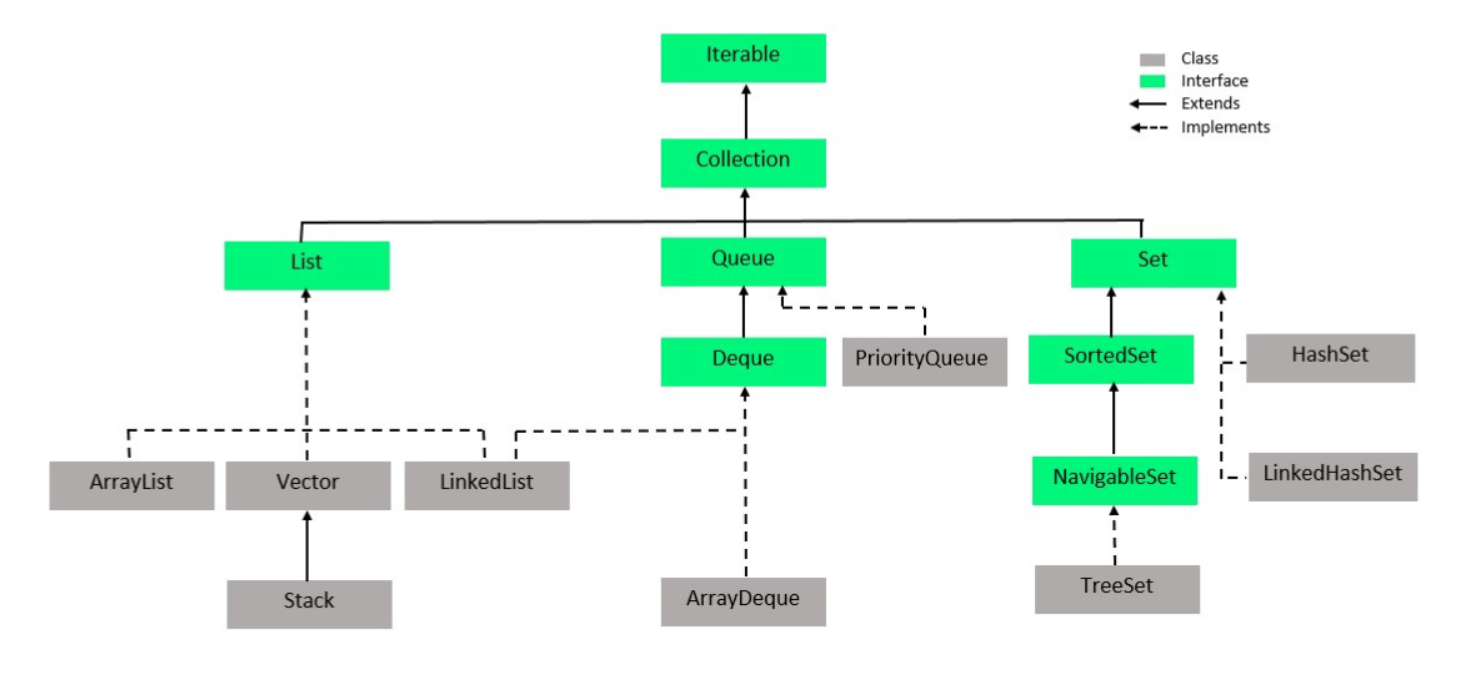
Java provides 5 keywords that are used to handle the exception.

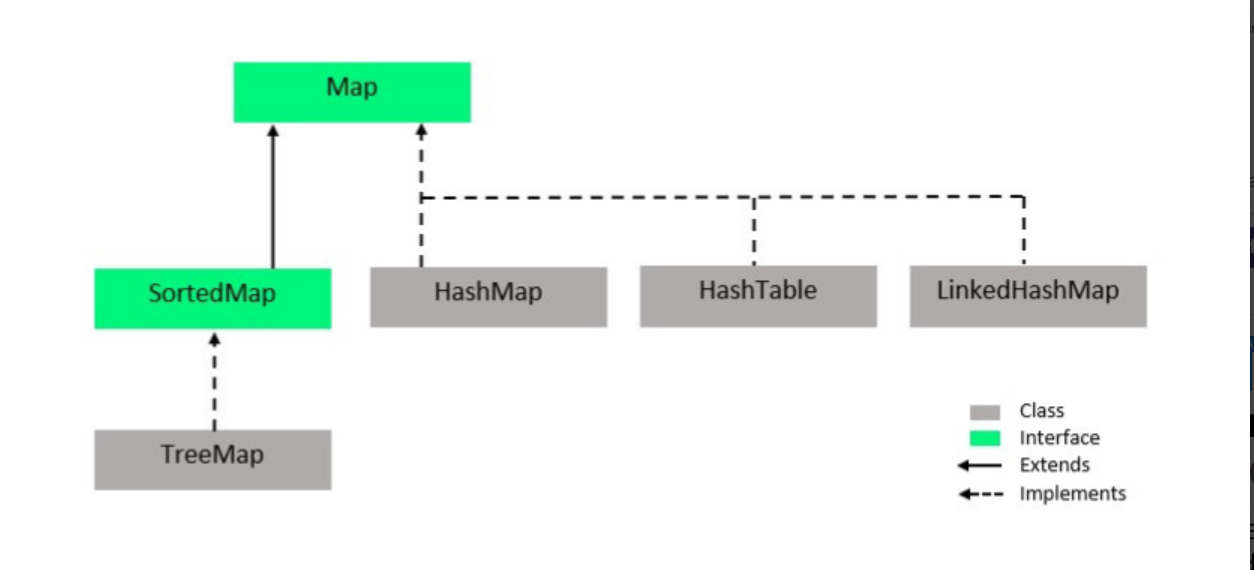
1. **Try:** The “try” keyword is used to specify a block where we should place an exception code.
2. **Catch:** The “catch” block is used to handle the exception. It must be preceded by try block which means we can’t use catch block alone. It can be followed by finally block later.
3. **Finally:** The “finally” block is used to execute the necessary code of the program. It is executed whether an exception occurred or not.
4. **Throw:** The “throw” keyword is used to throw an exception.
5. **Throws:** The “throws” keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It does not throw an exception. It is always used with method signature.

**Java Collection**

**Collection:** Collection is a [interface](https://www.geeksforgeeks.org/interfaces-in-java/) present in java.util package. It is used to represent a group of individual objects as a single unit.The collection is considered as the root interface of the collection framework. It provides several classes and interfaces to represent a group of individual objects as a single unit.

**Collections:** Collections is a utility class present in java.util package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods. These methods provide much-needed convenience to developers, allowing them to effectively work with [Collection Framework](https://www.geeksforgeeks.org/collections-in-java-2/). For example, It has a method sort() to sort the collection elements according to default sorting order, and it has a method min(), and max() to find the minimum and maximum value respectively in the collection elements.

****

****

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | **Map** |
| The list interface allows duplicate elements | Set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | Set do not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in set almost only one null value. | The map allows a single null key at most and any number of null values. |
| List implementation classes are [Array List](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/). | Set implementation classes are [HashSet](https://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-in-java-with-examples/), and [TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/). | Map implementation classes are [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/), [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/), [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/), and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/). |
| It is indexed based collection | It is not an indexed based collection | It is also not an indexed based collection |
| It is suitable when data will fetched through index | It is suitable when unique elements need to be stored. | It is suitable when data needs to be stored in key, value form. |
| To traverse the list elements by using Listlterator. Ex:  Iterator<String>itr = l.iterator(); | Iterator can be used traverse the set elements | Through keyset, value, and entry set. |

**LinkedHashSet:** Order is preserved, Duplicate not allowed.

**LinkedHashMap:** Order of key will be preserved, Duplicate not allowed.

**TreeSet:** Data is stored in ascending order.

**TreeMap:** Data is stored in key-wise ascending order.

|  |  |
| --- | --- |
| **HashMap** | **HashTable** |
| It inherits the abstract map class | It inherits the Dictionary class |
| It is non-synchronized non thread safe class | It is synchronized thread safe class |
| It was introduced in java 1.2 | It is a legacy class. |
| It allows one null key and multiple null values | It does not allow any null value and null key. |
| It is fast | It is slow |

**Multithreading**

**Thread** is a light-weight sub-process. It is a smallest unit of process.

**Multithreading** is a process of executing multiple threads simultaneously.

**Daemon Thread** in java is a service provider thread which provides the services to the user thread. It’s life depends on the mercy of the user’s thread when all the user’s thread die JVM terminates this thread automatically. Ex – garbage collector.

Thread can be created through 2 ways.

1. By extending Thread class.
2. By implementing Runnable interface.

**class** Multi **extends** Thread{

**public** **void** run(){

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[]){

Multi t1=**new** Multi();

t1.start();

 }

}

**class** Multi3 **implements** Runnable{

**public** **void** run(){

System.out.println("thread is running...");

}

**public** **static** **void** main(String args[]){

Multi3 m1=**new** Multi3();

Thread t1 =**new** Thread(m1);   // Using the constructor Thread(Runnable r)

t1.start();

t1.setDaemon(true); // To create daemon thread.

 }

}

**Testing Framework**

There are two types of testing

1. **Manual Testing**: if we execute test cases without any tool support this is called manual testing. It is time consuming and less reliable.
2. **Automated Testing**: If we execute test cases by tool support. It is known as automated testing. It is fast and more reliable.

**JUNIT** is an open-source testing framework for java programmers. We can create test cases and test our own code. It is one of the unit testing framework.

In JUNIT 5.x below annotations are used.

**@Test** annotation specifies that method is the test method.

**@BeforeAll** annotated method will be invoked only once before starting all the tests.

**@AfterAll** annotated method will be invoked only once after finishing all the test tests.

**@BeforeEach** annotated method will be invoked before each test.

**@AfterEach** annotated method will be invoked after each test.

**SQL Index:**

* SQl index can search the information from the large database quickly.
* Each index table contains only two columns the first column is row\_id and second column is indexed column.
* When indexes are used with smaller tables the performance may not be recongnized.

**Drawback:**

* SQL indexing slow down the execution of insert and update statement.
* SQL indexing should not be used on those cases when the column of table contains large number of NULL values.

**Hibernate**

Hibernate is a light weight open source ORM tool that is used to store, manipulate and retrieve data from database.

**ORM** is acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database.

It simplifies data creation, data manipulation and data access.

**Key components of Hibernate:**

* Configuration
* Session
* Sessionfactory
* Criteria
* Query
* Transaction

**Sessionfactory** provides the instance of the session. It is a factory of session. It is a thread-safe object.

**Session** maintains a connection between the hibernate application and database. It provide method to store, update, delete and fetch the data from database such as persist(), update(), delete(), load(), get(). It is not thread-safe object.

There are three states of the object (instance) in hibernate.

1. **Transient**: The object is in transient state if it is just created but has no primary keyand not associated with a session.
2. **Persistent**: The object is in persistent state if a session is open, and you just saved the instance of the database.
3. **Detached**: The object is in detached state if a session is closed.

**Advantage of ORM tools:**

* It increases the Application development speed.
* It simplifies the management of transaction.
* It hide the details of sql queries.

For JPA, **Repository** or **CRUDRepository** or **JPARepository** or **PagingAndSortingRepository** Interface needs to be implemented to our custom class.

**Http Methods**

The Hypertext transfer protocol is designed to enable communication between clients and servers.

Http methods are:

1. **GET (Read):** GET request is used to request the data from a specified resource.

* GET request can be cached.
* GET request remain in the browser’s history.
* GET request can be bookmarked.
* GET request should never be used while dealing with sensitive data.
* GET request have length restrictions.
* GET request are used only to request data not modify.

1. **POST (Create):** Post is used to send data to the server to create/update a resource. The data sent to the server with POST is stored in the request body of the Http request.

* POST request are never cached.
* POST request do not remain in the browser’s history.
* POST request cannot be bookmarked.
* POST request has no restriction on data length.

1. **PUT (Update):** PUT is used to send data to the server to create/update a resource. The difference between POST and PUT is that PUT requests are idempotent. It means calling the same PUT request multiple times will always produce the same result.
2. **DELETE:** DELETE method is used to delete the specified user. It is also idempotent method.

POST is neither safe nor idempotent

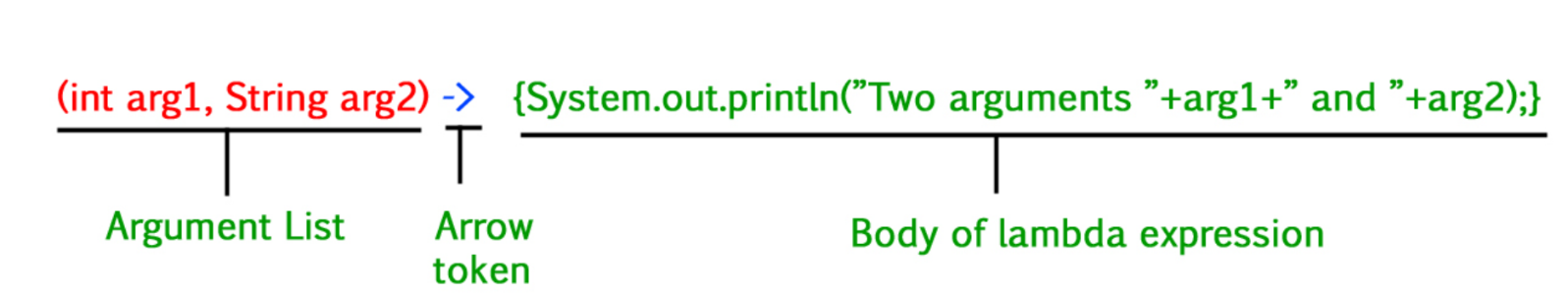
**Java 8 Features**

Java 8 is the most awaited release of Java programming language development because, in the entire history of Java, it never release that many major features.

Below are the features added in java 8

* Lambda Expression
* Functional Interface
* Method Reference
* Stream
* Comparable and Comparator
* Optional Class
* Date\Time Api
* Optional Class

**Lambda Expression:** In Java, Lambda expressions basically express instances of functional interfaces (*An interface with a single abstract method is called a functional interface*).



**Functional Interface:** An interface that contains only one abstract method is known as a functional interface, but there is no restriction, you can have **n**number of default and static methods inside a functional interface.

Some built-in functional interfaces are : Runnable, Comparable, ActionListner, Callable

We can create functional interface manually using @FunctionalInterface.

Q. **How many types of functional interface?**

**Stream**: **Java 8** introduces Stream, which is a new abstract layer, and some new additional packages in Java 8 called java.util.stream. A Stream is a sequence of components that can be processed sequentially. These packages include classes, interfaces, and enum to allow functional-style operations on the elements.

Syntax:

Stream s = c.stream(); //”c” refers to the collections

***Note:***

* *If we want to represent a group of objects as a single entity then we should go for*[***collection***](https://www.geeksforgeeks.org/collections-in-java-2/)*.*
* *But if we want to process objects from the collection then we should go for streams.*

**Features of Stream:**

* A stream is not a data structure instead it takes input
* , Arrays**,** or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
* Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Examples:

List number = Arrays.asList(2,3,4,5);

List square = number.stream().map(x->x\*x).collect(Collectors.toList());

List evenNumber = number.stream().filter(x->x%2==0).collect(Collectors.toList());

Optional<Integer> thirdLargest = number.stream().sorted(Comparator.reverseOrder()).skip(2).findFirst();

int thirdLargestNumber = thirdLargest.get();

Map<Character,Long> mp = number.stream().collect(Collectors.groupingBy(x -> x, Collectors.counting()));

**Intermediate Operations:**

1. filter()
2. map()
3. sorted()

**Terminal Operations:**

1. collect()
2. forEach()
3. reduce()

***Note:****Intermediate Operations are running based on the concept of Lazy Evaluation, which ensures that every method returns a fixed value(Terminal operation) before moving to the next method.*

**Optional Class:**

Java introduced a new class Optional in Java 8. It is a public final class which is used to deal with NullPointerException in Java application. We must import java.util package to use this class. It provides methods to check the presence of value for particular variable.

**Transaction Management:** @Transactional annotation is used for transaction management. It will roll back all the DB operations if there is any exception occurred inside the function.

**Synchronization:** In multithreaded environment when a shared resource is being used by multiple threads simultaneously then to avoid the data inconsistency @Synchronized annotation is used. It means the shared resource will not be used by other thread until the first thread completes its work with the specified resource.

Q. How many types of bindings ?

Q. What is the internal functionality of HashMap?

Q. What is the default capacity of HahsMap?

16

**Status Code**

|  |  |
| --- | --- |
| **Status code** | **Description** |
| **200, OK** | Indicates that the request has succeeded. |
| **201, Created** | Indicates that the request has succeeded and a new resource has been created as a result. |
| **400, Bad Request** | The request could not be understood by the server due to incorrect syntax. The client SHOULD NOT repeat the request without modifications. |
| **401, Unauthorized** | Indicates that the request requires user authentication information. The client MAY repeat the request with a suitable Authorization header field |
| **500, Internal Server Error** | The server encountered an unexpected condition that prevented it from fulfilling the request. |
| **403, Forbidden** | Unauthorized request. The client does not have access rights to the content. Unlike 401, the client’s identity is known to the server. |
| **404, Not found** | The server can not find the requested resource. |
| **405, Method Not Allowed** | The request HTTP method is known by the server but has been disabled and cannot be used for that resource. |

**Interview Questions**

1. **How to make a Collection read only?**
2. **How to create REST Services?**
3. **How to use Response entity?**
4. **How to disable auto configuration in springboot?**
5. **What is the use case of abstract class?**
6. **What is JWT token how to user it?**
7. **How to get data from the header?**
8. **How to create and use daemon thread?**
9. **How to call garbage Collector manually in java?**
10. **How to create custom annotation in java?**
11. **What is server side and client side exceptions?**
12. **How to enable caching?**
13. **How equals function work in String in java?**
14. **What is the difference between SQL and NoSQL?**
15. **What is SQL indexing?**
16. **What is Synchronization in java?**
17. **What is Serialization in java?**
18. **What is reflection in java?**
19. **What is Object Cloning in Java?**
20. **What is Deep copy and Shallow Copy in java?**
21. **How to get current state of thread?**
22. **How to create custom exceptions in java?**
23. **Why String is immutable in java?**
24. **Custom sorting in java using Comparator and Comparable?**
25. **What is the difference between HashMap and HashTable why we use HashMap instead of hashTable?**
26. **How to configure Swagger?**
27. **How to map exceptions to the methods?**
28. **How to configure multiple database in springboot?**
29. **What is agile methodology?**
30. **What is design pattern in software Engineering?**
31. **How can we achieve singleton design pattern ?**
32. **Why we use jboss server if tomcat server is already embedded in springboot applications?**

Tomcat is ideal for small projects where there is no need for high availability or scalability. JBoss is a enterprise application server based on the Java platform. It offers a wide range of features such as clustering, load balancing, security, management tools, and more.

1. **Switch case in mysql.**
2. **How to use upcast in objects**
3. **Mapping and joins in springboot**
4. **What is commit and flush in mysql.**
5. **How to get unique characters from a string through stream api?**