- What is the purpose of the method `public static void main` in a Java program?
  Where, **public** because it has to be invoked from outside (parameters are received by the string array called args), and **static** because we could start off with no objects created. **main** is **void** because it does not return a value.

- What is the difference between `System.out.println` and `System.err.println`?
  System.out is "*standard output*" (stdout) and *System.err* is "*error output*" (stderr).
  Along with *System.in* (stdin), these are the three standard I/O streams in the Unix model. Most modern programming environments supports this model.

- What is an `interface` in Java?
  It is a collection of *abstract methods*. A class **implements** an interface, thereby **inheriting** *the abstract methods of the interface*. (Along with abstract methods),
  an interface may also contain *constants*, *default methods*, *static methods*, *and nested types*. **Method bodies** exist only for ***default methods*** and ***static methods***.

- When would you use an `abstract` class instead of an `interface`?
  In shorts, abstract classes are either *partially implemented* or *not implemented at all*.
  You can have functionality in your abstract class—the **methods** in an abstract class can be *both* ***abstract*** *and* ***concrete***. An abstract class can have **constructors**—this is one major difference
  between an abstract class and an interface. You can take advantage of abstract classes to design components and specify some level of common functionality that must be implemented by derived classes.

- What are the differences between a `public` method and a `protected` one?

Modifier of x \ xCanBeSeenBy	highest precedence <---------> lowest precedence this class nonsubbed	any class in same package	this subclass in another package	any class
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
package-private (no modifier)	✓	✓	✗	✗
private	✓	✗	✗	✗

- What is a `static` variable?
  The **static keyword** is used for memory management mainly and belongs to the class than an instance of the class. So, we can access the static methods or varibles without creating an instance of a class.

- What is an `Exception` in Java?
  an **exception** is an object that wraps *an error event* that occurred *within a method* and contains: Information about the error including its type. So, we can catch it and recover from the error condition.

- Is it a good practice to catch a `RuntimeException`?
  Unless you can correct a RuntimeException, you don't want to catch it...
  RuntimeException is intended to be used for programmer errors. As such it should never be caught.

- What is the keyword to use in a method signature to allow non-catching of an exception in this method?
  throws keyword is used to ***declare the exception*** that might raise during program execution.
  This ***forces or tells*** the caller method to handle that exception.

- What is the purpose of a garbage collector?

  to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused.

  A **Java** object is subject to **garbage collection** when it becomes unreachable to the program in which it is used.

- What is the difference between a `HashSet` and a `TreeSet`?

  HashSet is faster than **TreeSet** and should be preferred choice if sorting of element is not required.
  HashSet is Implemented using a hash table. Elements are not ordered. The `add, remove,` and ***contains*** methods have constant time **complexity O(1)**.
  TreeSet is implemented using a tree structure(red-black tree in algorithm book). The elements in a set are sorted, but the *add, remove*, and *contains* methods has time **complexity O(log (n))**. It offers several methods to deal with the ordered set like `first(), last(), headSet(), tailSet()`, etc.

- Which `Thread` method is called when a thread starts?

  The **start**() **method** of **thread** class is used to **begin** the execution of **thread**. The result of this **method** is two **threads** that are running concurrently**:** *the current* ***thread*** (which returns from the call to the **start method**) and *the other* ***thread*** (which executes its **run** method).

- Is it possible to update a `String` object (without using introspection)?

- What is the contract between the methods `equals` and `hashcode`?

  The Java super class `java.lang.Object` defines two important methods:

```
public boolean equals(Object obj)
public int hashCode()
```

  The contract between **equals()** and **hashCode()** is:
  1) If two objects are equal, then they must have the same hash code.
  2) If two objects have the same hash code, they may or may not be equal.

- Who is James Gosling?

  Since 1984, Gosling has been with Sun Microsystems, and is generally known best as the founder of the Java programming language.

Q1. Explain JDK, JRE and JVM?

It is used to provide runtime environment.It is the implementation of **JVM**.
It physically exists.It contains set of libraries + other files that **JVM** uses at runtime.

Q2. Explain public static void main(String args[]).

- **public** : Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
- **static** : It is a keyword in java which identifies it is class based i.e it can be accessed without creating the instance of a Class.
- **void** : It is the return type of the method. Void defines the method which will not return any value.
- **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
- **String args[]** : It is the parameter passed to the main method.

Q3. Why Java is platform independent?

Platform independent practically means “write once run anywhere”. Java is called so because of its byte codes which can run on any system irrespective of its underlying operating system.

Q4. Why java is not 100% Object-oriented?
Java is not 100% Object-oriented because it makes use of eight primitive datatypes such as boolean, byte, char, int, float, double, long, short which are **not objects.**

Q5. What are wrapper classes?
Wrapper classes converts the java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they "wrap" the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

Premitive Data Types

Data Type	Size / default	Description
byte	1 byte / 0	-128 to 127
short	2 bytes / 0	-32,768 to 32,767
int	4 bytes / 0	-2,147,483,648 to 2,147,483,647
long	8 bytes / 0L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
float	4 bytes / 0.0f	3.4e−038 to 3.4e+038. Enough for 6 to 7 decimal digits
double	8 bytes / 0.0d	1.7e−308 to 1.7e+038. Enough for 15 decimal digits
boolean	1 byte / false	true or false values
char	2 bytes / \u0000	a single character/letter

Q6. What are constructors in Java?

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.
There are two types of constructors:

1. Default constructor
2. Parameterized constructor

Q7. What is singleton class and how can we make a class singleton?

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

Q8. What is the difference between ArrayList and LinkedList?
`LinkedList`implements it with a doubly-linked list.

- `get(int index)` is *O(n)* (with *n/4* steps on average)
- `add(E element)` is *O(1)*
- `add(int index, E element)` is *O(n)* (with *n/4* steps on average), but *O(1)* when `index = 0` <--- main benefit of `LinkedList<E>`
- `remove(int index)` is *O(n)* (with *n/4* steps on average)
- `Iterator.remove()` is *O(1)*. <--- main benefit of `LinkedList<E>`
- `ListIterator.add(E element)` is *O(1)* This is one of the main benefits of `LinkedList<E>`

`ArrayList` implements it with a dynamically re-sizing array.

- `get(int index)` is *O(1)* <--- main benefit of `ArrayList<E>`
- `add(E element)` is *O(1)* amortized, but *O(n)* worst-case since the array must be resized and copied
- `add(int index, E element)` is *O(n)* (with *n/2* steps on average)
- `remove(int index)` is *O(n)* (with *n/2* steps on average)
- `Iterator.remove()` is *O(n)* (with *n/2* steps on average)
- `ListIterator.add(E element)` is *O(n)* (with *n/2* steps on average)

Also, if you have **large lists**, keep in mind that **memory usage** is also different. Each element of a `LinkedList` has more overhead since pointers to the next and previous elements are also stored. `ArrayList` don't have this overhead. However, `ArrayLists` take up as much memory as is allocated for the capacity, regardless of whether elements have actually been added or not.
The default initial capacity of an `ArrayList` is pretty small (**10** from Java 1.4 - 1.8). But since the underlying implementation is an array, the array must be resized if you add a lot of elements. To avoid the high cost of resizing when you know you're going to add a lot of elements, construct the `ArrayList` with a **higher initial capacity**.

Q9. What is the difference between **equals()** and **==** ?

equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic. "**==**" or **equality operator** in Java is a binary operator provided by Java programming language and used to compare primitives and objects. *public boolean equals(Object o)* is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

Q10. What are the differences between **Heap** and **Stack Memory?**

Features	Stack	Heap
Memory	Stack memory is used only by one thread of execution.	Heap memory is used by all the parts of the application.
Access	Stack memory can't be accessed by other threads.	Objects stored in the heap are globally accessible.
Memory Management	Follows LIFO manner to free memory.	Memory management is based on generation associated to each object.
Lifetime	Exists until the end of execution of the thread.	Heap memory lives from the start till the end of application execution.
Usage	Stack memory only contains local primitive and reference variables to objects in heap space.	Whenever an object is created, it's always stored in the Heap space.

OOPS Java Interview Questions:

Q1. What is Polymorphism?
Polymorphism is briefly described as "one interface, many implementations". Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have **more than one form**. There are two types of polymorphism:

1. Compile time polymorphism (overloading)
2. Run time polymorphism (overriding)

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

Q2. What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let's take a look at the example below to understand it better.

```
class Car {
   void run()
   {
      System.out.println("car is running");
   }
}

class Audi extends Car {
   void run()
   {
     System.out.prinltn("Audi is running safely with 100km");
   }
   public static void main(String args[])
   {
     Car b= new Audi();    //upcasting
     b.run();
   }
}
```

Q3. What is the difference between ??

Q4. What is method overloading and method overriding?

Method Overloading :

- In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.
- Method Overloading is to "add" or "extend" more to method's behavior.
- It is a compile time polymorphism.
- The methods must have different signature.
- It may or may not need inheritance in Method Overloading.

Let's take a look at the example below to understand it better.

```
class  Adder {
   static int add(int a, int b)
   {
     return a+b;
   }

   static double add( double a, double b)
   {
     return a+b;
   }

   public static void main(String args[])
   {
     System.out.println(Adder.add(11,11));
     System.out.println(Adder.add(12.3,12.6));
   }
}
```

Method Overriding:

- In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.
- Method Overriding is to "Change" existing behavior of method.
- It is a run time polymorphism.
- The methods must have same signature.
- It always requires inheritance in Method Overriding.

Let’s take a look at the example below to understand it better.

```
class Car {
   void run(){
     System.out.println(“car is running”);
   }
}

class Audi extends Car{
   void run()
   {
     System.out.prinltn(“Audi is running safely with 100km”);
   }

   public static void main( String args[])
   {
     Car b=new Audi();
     b.run();
   }
}
```

Q5. Can you override a private or static method in Java?

You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class, then it will hide the super class method; this is known as method hiding. Similarly, you cannot override a private method in sub class because it’s not accessible there. What you can do is create another private method with the same name in the child class. Let’s take a look at the example below to understand it better.

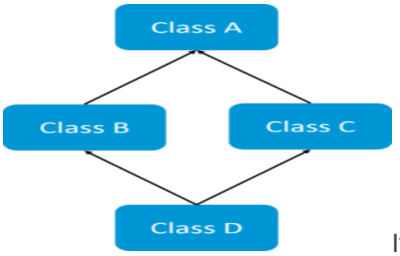
```
class Base {
   private static void display() {
     System.out.println("Static or class method from Base");
   }

   public void print() {
     System.out.println("Non-static or instance method from Base");
   }
}
class Derived extends Base {
   private static void display() {
     System.out.println("Static or class method from Derived");
   }

   public void print() {
     System.out.println("Non-static or instance method from Derived");
   }
}

public class test {
   public static void main(String args[])
   {
     Base obj= new Derived();
     obj1.display();
     obj1.print();
   }
}
```

Q6. What is multiple inheritance? Is it supported by Java?



If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.
The problem with multiple inheritance is that if multiple parent classes have a same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.
Therefore, Java doesn't support multiple inheritance. The problem is commonly referred as Diamond Problem.

Q7. What is association?

Association is a relationship where all object has their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationships can be one to one, One to many, many to one and many to many.

Q8. What do you mean by aggregation?

Aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object can not belong to another parent object. Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department teacher object will not destroy.

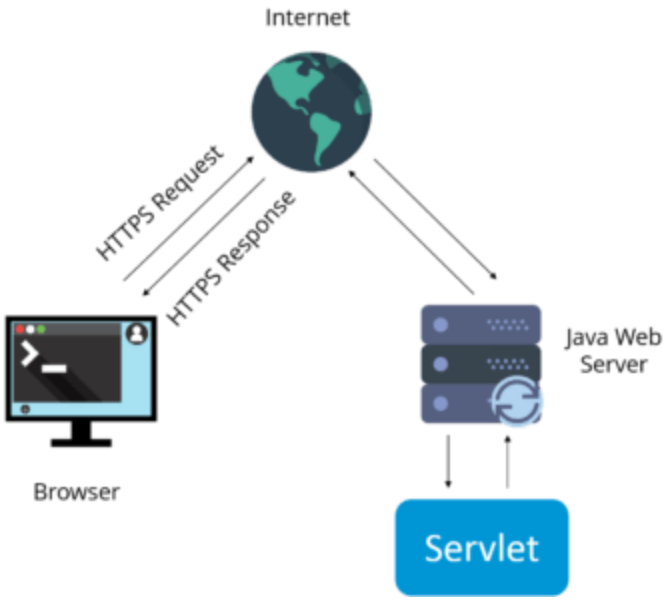
Q9. What is composition in Java?

Composition is again specialized form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belong to two different houses if we delete the house room will automatically delete.
In case you are facing any challenges with these java interview questions, please comment your problems in the section below. Apart from this Java Interview Questions Blog, if you want to get trained from professionals on this technology, you can opt for a structured training from edureka!

Servlets Interview Questions

Q1. What is a servlet?

- Java Servlet is server-side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.
- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing our own servlets.
- All servlets must implement the javax.servlet.Servlet interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet() and doPost(), for handling HTTP-specific services.
- Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend HttpServlet class. Servlet API hierarchy is shown in below image.

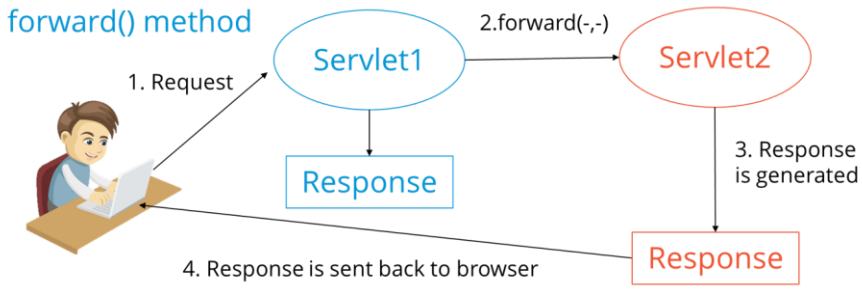


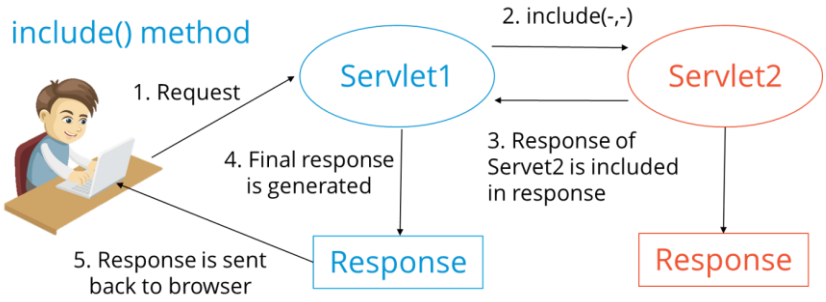
Q2. What are the differences between Get and Post methods?

Get	Post
Limited amount of data can be sent because data is sent in header.	Large amount of data can be sent because data is sent in body.
Not Secured because data is exposed in URL bar.	Secured because data is not exposed in URL bar.
Can be bookmarked	Cannot be bookmarked
Idempotent	Non-Idempotent
It is more efficient and used than Post	It is less efficient and used

Q3. What is Request Dispatcher?

RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in same application. We can also use this to include the content of another resource to the response.
There are two methods defined in this interface:
1.void forward()
2.void include()

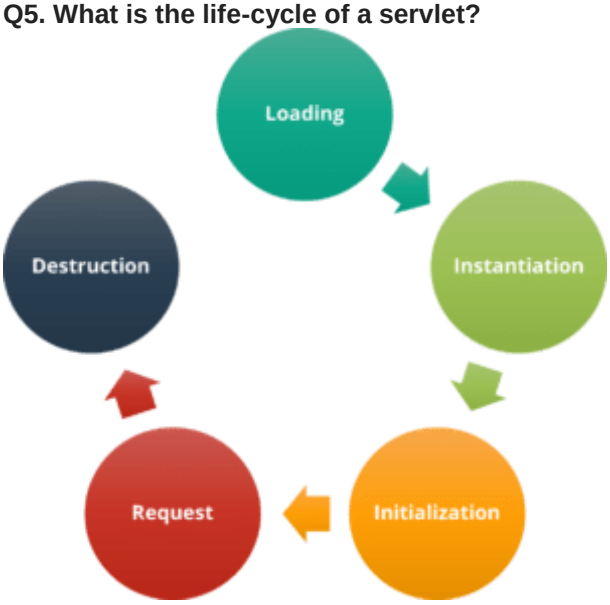




Q4. What are the differences between forward() method and sendRedirect() methods?

Forward() method	SendRedirect() method
forward() sends the same request to another resource.	sendRedirect() method sends new request always because it uses the URL bar of the browser.
forward() method works at server side.	sendRedirect() method works at client side.
forward() method works within the server only.	sendRedirect() method works within and outside the server.

Q5. What is the life-cycle of a servlet?



There are 5 stages in the lifecycle of a servlet:

1. Servlet is loaded
2. Servlet is instantiated
3. Servlet is initialized
4. Service the request
5. Servlet is destroyed

Q6. How does cookies work in Servlets?

- Cookies are text data sent by server to the client and it gets saved at the client local machine.
- Servlet API provides cookies support through javax.servlet.http.Cookie class that implements Serializable and Cloneable interfaces.
- **HttpServletRequest** getCookies() method is provided to get the array of Cookies from request, since there is no point of adding Cookie to request, there are no methods to set or add cookie to request.
- Similarly, **HttpServletResponse** addCookie(Cookie c) method is provided to attach cookie in response header, there are no getter methods for cookie.

Q7. What are the differences between ServletContext vs ServletConfig?

The difference between ServletContext and ServletConfig in Servlets JSP is in below tabular format.

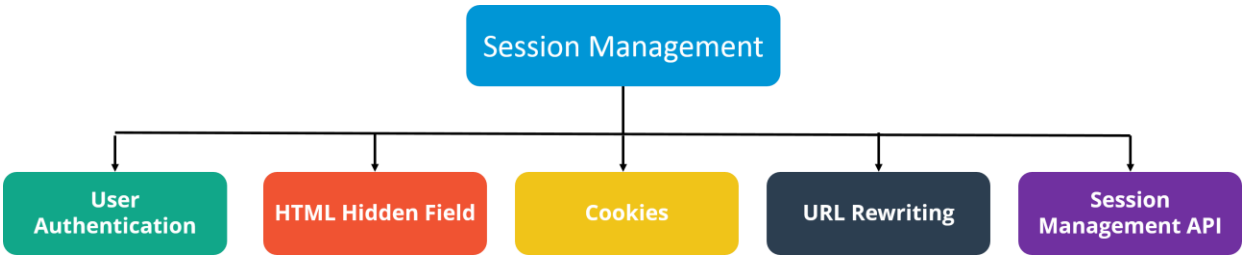
ServletConfig	ServletContext
Servlet config object represent single servlet	It represents whole web application running on particular JVM and common for all the servlet
Its like local parameter associated with particular servlet	Its like global parameter associated with whole application
It's a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope	ServletContext has application wide scope so define outside of servlet tag in web.xml file.
getServletConfig() method is used to get the config object	getServletContext() method is used to get the context object.
for example, shopping cart of a user is a specific to particular user so here we can use servlet config	To get the MIME type of a file or application session related information is stored using servlet context object.

Q8. What are the different methods of session management in servlets?

Session is a conversational state between client and server and it can consist of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

Some of the common ways of session management in servlets are:

1. User Authentication
2. HTML Hidden Field
3. Cookies
4. URL Rewriting
5. Session Management API

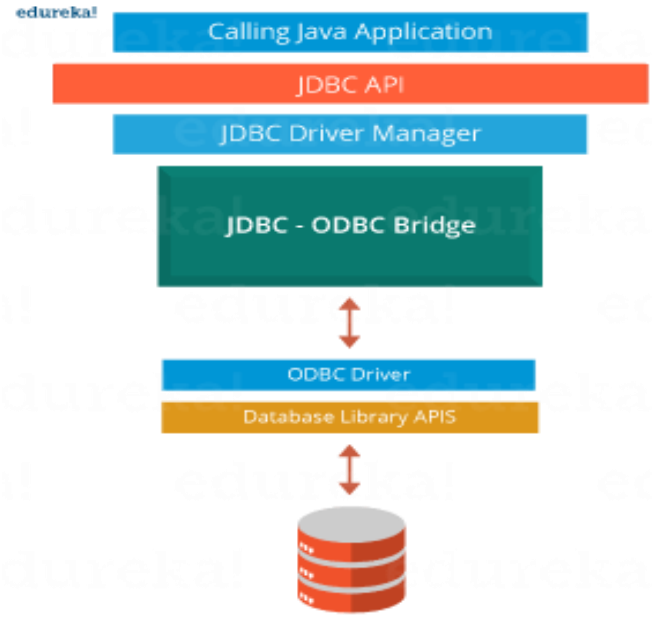


JDBC Interview Questions

Q1. What is JDBC Driver?

JDBC Driver is a software component that enables java application to interact with the database.
There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)



Q2. What are the steps to connect to a database in java?

- Registering the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

Q3. What are the JDBC API components?

The java.sql package contains interfaces and classes for JDBC API.
Interfaces:

- Connection
- Statement
- PreparedStatement
- ResultSet
- ResultSetMetaData
- DatabaseMetaData
- CallableStatement etc.

Classes:

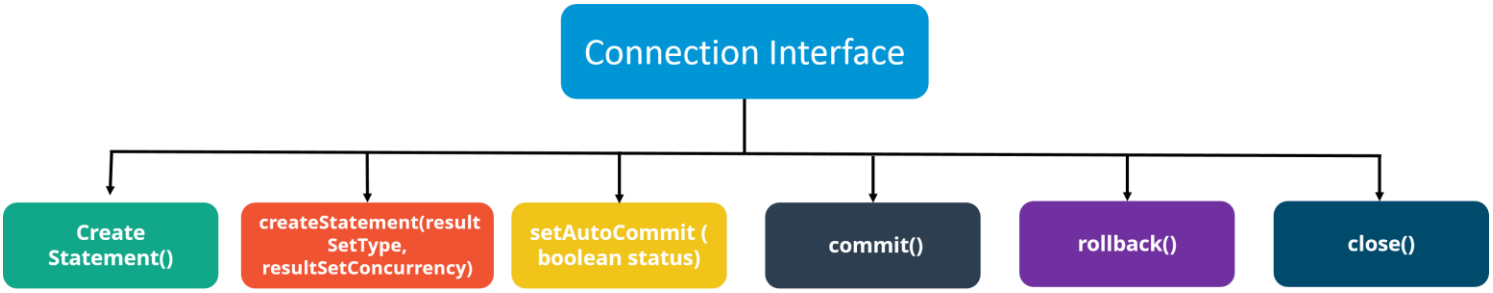
- DriverManager
- Blob
- Clob
- Types
- SQLException etc.

Q4. What is the role of JDBC DriverManager class?

The DriverManager *class* manages the registered drivers. It can be used to register and unregister drivers. It provides factory method that returns the instance of Connection.

Q5. What is JDBC Connection interface?

The Connection interface maintains a session with the database. It can be used for transaction management. It provides factory methods that returns the instance of Statement, PreparedStatement, CallableStatement and DatabaseMetaData.



Q6. What is the purpose of JDBC ResultSet interface?

The ResultSet object represents a row of a table. It can be used to change the cursor pointer and get the information from the database.

Q7. What is JDBC ResultSetMetaData interface?

The ResultSetMetaData interface returns the information of table such as total number of columns, column name, column type etc.

Q8. What is JDBC DatabaseMetaData interface?

The DatabaseMetaData interface returns the information of the database such as username, driver name, driver version, number of tables, number of views etc.

Q9. What do you mean by batch processing in JDBC?

Batch processing helps you to group related SQL statements into a batch and execute them instead of executing a single query. By using batch processing technique in JDBC, you can execute multiple queries which makes the performance faster.

Q10. What is the difference between execute, executeQuery, executeUpdate?

Statement ***execute(String query)*** is used to execute any SQL query and it returns TRUE if the result is a ResultSet such as running Select queries. The output is FALSE when there is no ResultSet object such as running Insert or Update queries. We can use *getResultSet()* to get the ResultSet and *getUpdateCount()* method to retrieve the update count.
Statement ***executeQuery(String query)*** is used to execute Select queries and returns the ResultSet. ResultSet returned is never null even if there are no records matching the query. When executing select queries we should use executeQuery method so that if someone tries to execute insert/update statement it will throw java.sql.SQLException with message “executeQuery method can not be used for update”.
Statement **executeUpdate(String query**) is used to execute Insert/Update/Delete (DML) statements or DDL statements that returns nothing. The output is int and equals to the row count for SQL Data Manipulation Language (DML) statements. For DDL statements, the output is 0.
You should use execute() method only when you are not sure about the type of statement else use executeQuery or executeUpdate method.

Spring Interview Questions

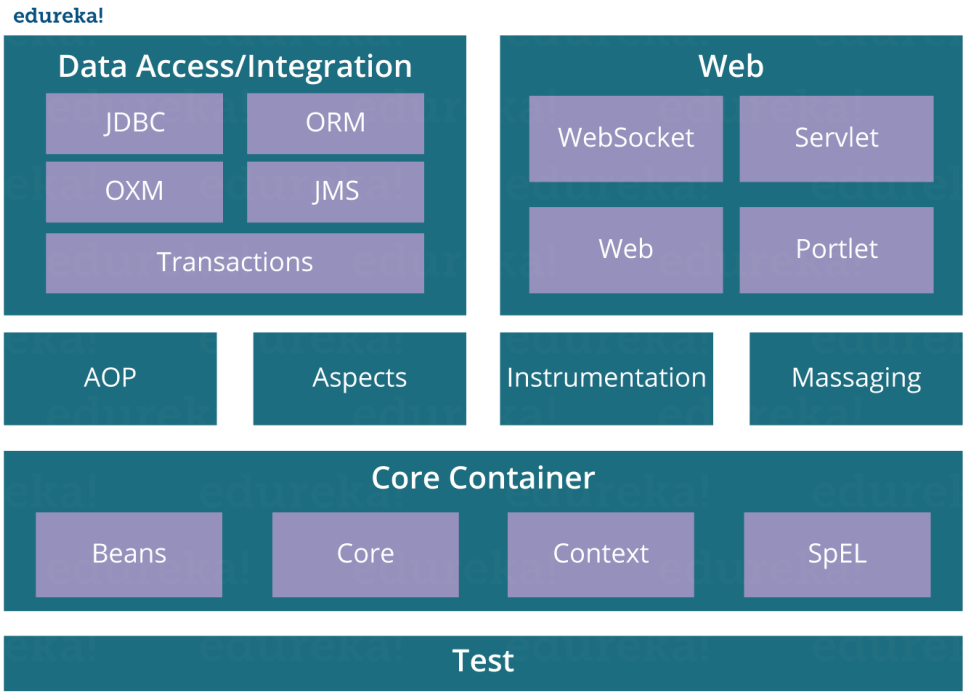
Q1. What is a Spring?

Wikipedia defines the Spring framework as “an application framework and inversion of control container for the Java platform. The framework’s core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform.” Spring is essentially a lightweight, integrated framework that can be used for developing enterprise applications in java.

Q2. Name the different modules of the Spring framework.

Some of the important Spring Framework modules are:

- Spring Context – for dependency injection.
- Spring AOP – for aspect-oriented programming.
- Spring DAO – for database operations using DAO pattern
- Spring JDBC – for JDBC and DataSource support.
- Spring ORM – for ORM tools support such as Hibernate
- Spring Web Module – for creating web applications.
- Spring MVC – Model-View-Controller implementation for creating web applications, web services etc.



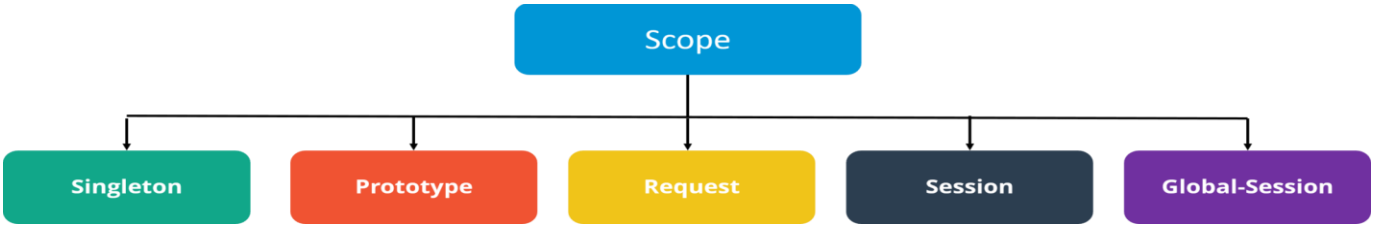
Q3. List some of the important annotations in annotation-based Spring configuration.

The important annotations are:

- @Required
- @Autowired
- @Qualifier
- @Resource
- @PostConstruct
- @PreDestroy

Q4. Explain Bean in Spring and List the different Scopes of Spring bean.

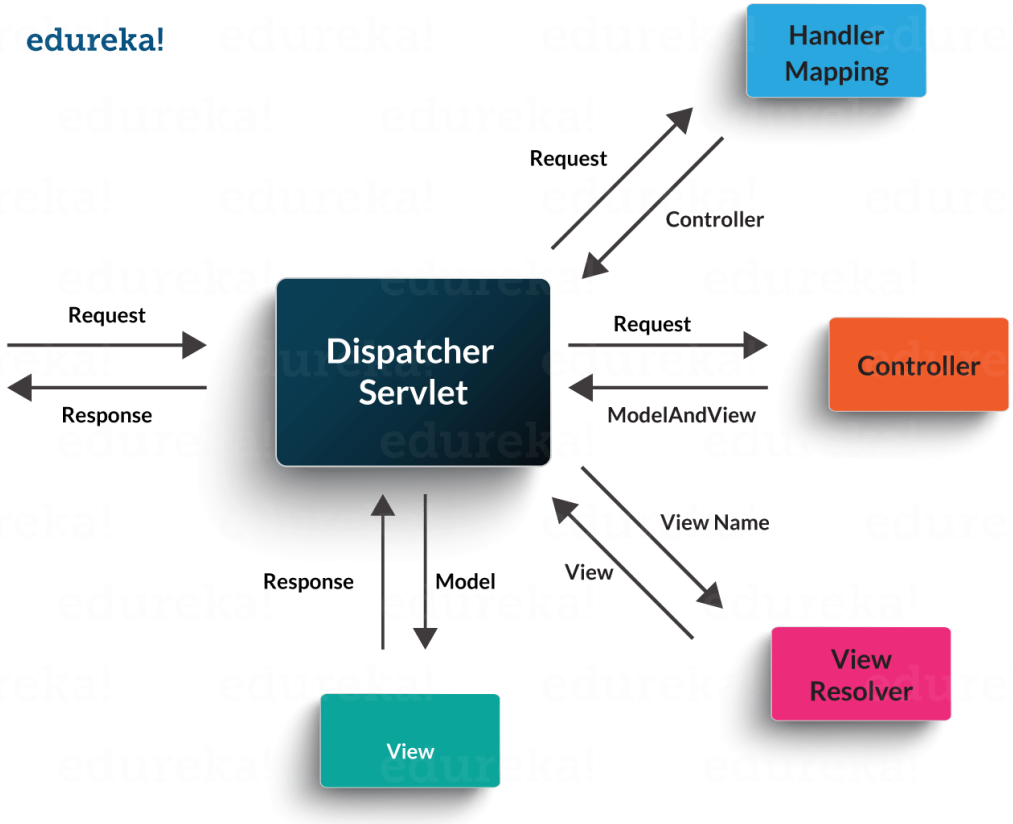
Beans are objects that form the backbone of a Spring application. They are managed by the Spring IoC container. In other words, a bean is an object that is instantiated, assembled, and managed by a Spring IoC container.
There are five Scopes defined in Spring beans.



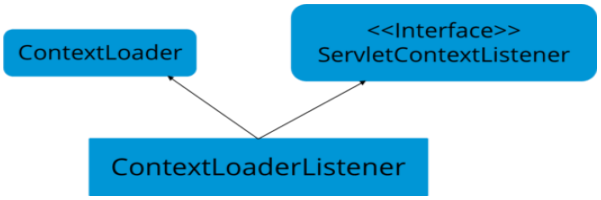
- **Singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues because it's not thread-safe.
- **Prototype**: A new instance will be created every time the bean is requested.
- **Request**: This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
- **Session**: A new bean will be created for each HTTP session by the container.
- **Global-session**: This is used to create global session beans for Portlet applications.

Q5. Explain the role of DispatcherServlet and ContextLoaderListener.

DispatcherServlet is basically the front controller in the Spring MVC application as it loads the spring bean configuration file and initializes all the beans that have been configured. If annotations are enabled, it also scans the packages to configure any bean annotated with @Component, @Controller, @Repository or @Service annotations.



ContextLoaderListener, on the other hand, is the listener to start up and shut down the WebApplicationContext in Spring root. Some of its important functions includes tying up the lifecycle of Application Context to the lifecycle of the ServletContext and automating the creation of ApplicationContext.



Q6. What are the differences between constructor injection and setter injection?

No.	Constructor Injection	Setter Injection
1)	No Partial Injection	Partial Injection
2)	Desn't override the setter property	Overrides the constructor property if both are defined.
3)	Creates new instance if any modification occurs	Doesn't create new instance if you change the property value
4)	Better for too many properties	Better for few properties.

Q7. What is autowiring in Spring? What are the autowiring modes?

Autowiring enables the programmer to inject the bean automatically. We don't need to write explicit injection logic. Let's see the code to inject bean using dependency injection.

1. <bean id="emp" class="com.javatpoint.Employee" autowire="byName" />

The autowiring modes are given below:

No.	Mode	Description
1)	no	this is the default mode, it means autowiring is not enabled.
2)	byName	Injects the bean based on the property name. It uses setter method.
3)	byType	Injects the bean based on the property type. It uses setter method.
4)	constructor	It injects the bean using constructor

Q8. How to handle exceptions in Spring MVC Framework?

Spring MVC Framework provides the following ways to help us achieving robust exception handling.

Controller Based:
We can define exception handler methods in our controller classes. All we need is to annotate these methods with @ExceptionHandler annotation.

Global Exception Handler:
Exception Handling is a cross-cutting concern and Spring provides @ControllerAdvice annotation that we can use with any class to define our global exception handler.

HandlerExceptionResolver implementation:
For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

Q9. What are some of the important Spring annotations which you have used?

Some of the Spring annotations that I have used in my project are:
@Controller – for controller classes in Spring MVC project.
@RequestMapping – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through Spring MVC RequestMapping Annotation Examples

@ResponseBody – for sending Object as response, usually for sending XML or JSON data as response.
@PathVariable – for mapping dynamic values from the URI to handler method arguments.
@Autowired – for autowiring dependencies in spring beans.
@Qualifier – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.
@Service – for service classes.
@Scope – for configuring scope of the spring bean.
@Configuration, @ComponentScan and @Bean – for java-based configurations.
AspectJ annotations for configuring aspects and advices, @Aspect, @Before, @After, @Around, @Pointcut etc.

Q10. How to integrate Spring and Hibernate Frameworks?

We can use Spring ORM module to integrate Spring and Hibernate frameworks, if you are using Hibernate 3+ where SessionFactory provides current session, then you should avoid using HibernateTemplate or HibernateDaoSupport classes and better to use DAO pattern with dependency injection for the integration.
Also, Spring ORM provides support for using Spring declarative transaction management, so you should utilize that rather than going for hibernate boiler-plate code for transaction management.

Hibernate Interview Questions

Q1. What is Hibernate Framework?

Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.
Hibernate provides reference implementation of Java Persistence API, that makes it a great choice as ORM tool with benefits of loose coupling. We can use Hibernate persistence API for CRUD operations. Hibernate framework provide option to map plain old java objects to traditional database tables with the use of JPA annotations as well as XML based configuration.
Similarly hibernate configurations are flexible and can be done from XML configuration file as well as programmatically.

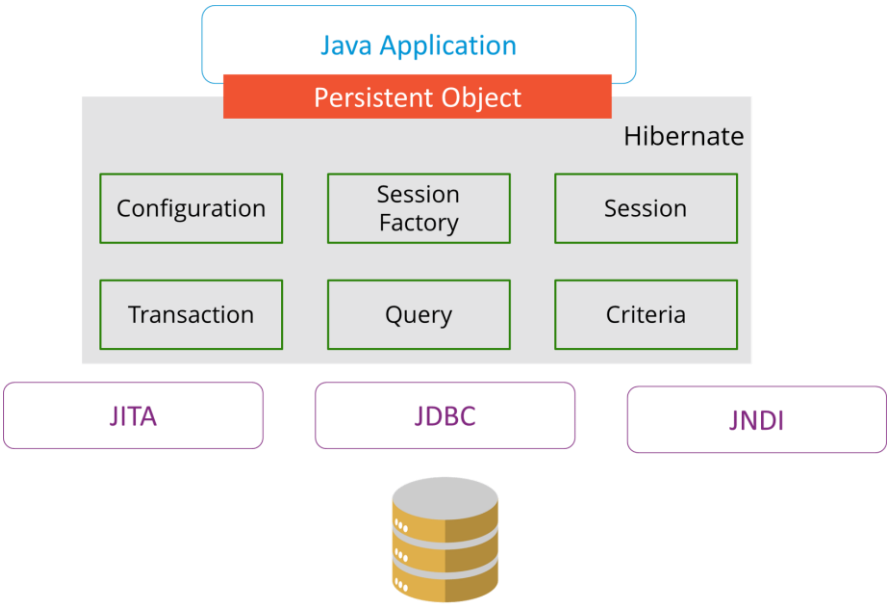
Q2. What are the important benefits of using Hibernate Framework?

Some of the important benefits of using hibernate framework are:

1. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.
2. Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.
3. Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.
4. Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.
5. Hibernate is easy to integrate with other Java EE frameworks, it's so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.
6. Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it's required.
7. Hibernate cache helps us in getting better performance.
8. For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Overall hibernate is the best choice in current market for ORM tool, it contains all the features that you will ever need in an ORM tool.

Q3. Explain Hibernate architecture?



Q4 What are the differences between get and load methods?

The differences between get() and load() methods are given below.

No.	get()	load()
1)	Returns null if object is not found.	Throws ObjectNotFoundException if object is not found.
2)	get() method always hit the database.	load() method doesn't hit the database.
3)	It returns real object not proxy.	It returns proxy object.
4)	It should be used if you are not sure about the existence of instance.	It should be used if you are sure that instance exists.

Q5 What are the advantages of Hibernate over JDBC?

Some of the important advantages of Hibernate framework over JDBC are:

1. Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks cleaner and more readable.
2. Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.
3. Hibernate implicitly provides transaction management, in fact most of the queries can't be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback.
4. JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it's redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException orHibernateException un-checked exception, so we don't need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.
5. Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.
6. Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.
7. Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.
8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.
1. Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

Java Interview Questions: JSP

Q1. What is the life-cycle methods for a jsp?

Method	Description
public void jspInit()	It is invoked only once, same as init method of servlet.
public void _jspService(ServletRequest request,ServletResponse)throws ServletException,IOException	It is invoked at each request, same as service() method of servlet.
public void jspDestroy()	It is invoked only once, same as destroy() method of servlet.

Q2. What are the JSP implicit objects?

JSP provides 9 implicit objects by default. They are as follows:

Object	Type
1) out	JspWriter
2) request	HttpServletRequest
3) response	HttpServletResponse
4) config	ServletConfig
5) session	HttpSession
6) application	ServletContext
7) pageContext	PageContext
8) page	Object
9) exception	Throwable

Q3. What are the differences between include directive and include action?

include directive	include action
The include directive includes the content at page translation time.	The include action includes the content at request time.
The include directive includes the original content of the page so page size increases at runtime.	The include action doesn't include the original content rather invokes the include() method of Vendor provided class.
It's better for static pages.	It's better for dynamic pages.

Q4. How to disable caching on back button of the browser?

```
<%
response.setHeader("Cache-Control","no-store");
response.setHeader("Pragma","no-cache");
response.setHeader ("Expires", "0");          //prevents caching at the proxy server
%>
```

Q5. What are the different tags provided in JSTL?

There are 5 type of JSTL tags.

1. core tags
2. sql tags
3. xml tags
4. internationalization tags
5. functions tags

Q6. How to disable session in JSP?

1. **<**%@ page session="false" %**>**

Q7. How to delete a Cookie in a JSP?

The following code explain how to delete a Cookie in a JSP :

```
1     Cookie mycook = new Cookie("name1","value1");
2
3     response.addCookie(mycook1);
4
5     Cookie killmycook = new Cookie("mycook1","value1");
6
7     killmycook . set MaxAge ( 0 );
8
9     killmycook . set Path ("/");
10
11    killmycook . addCookie ( killmycook 1 );
```

Q8. Explain the jspDestroy() method.

jspDestry() method is invoked from javax.servlet.jsp.JspPage interface whenever a JSP page is about to be destroyed. Servlets destroy methods can be easily overridden to perform cleanup, like when closing a database connection.

Q9. How is JSP better than Servlet technology?

JSP is a technology on the server's side to make content generation simple. They are document centric, whereas servlets are programs. A Java server page can contain fragments of Java program, which execute and instantiate Java classes. However, they occur inside HTML template file. It provides the framework for development of a Web Application.

Q10. Why should we not configure JSP standard tags in web.xml?

We don't need to configure JSP standard tags in web.xml because when container loads the web application and find TLD files, it automatically configures them to be used directly in the application JSP pages. We just need to include it in the JSP page using taglib directive.

Exception and Thread Java Interview Questions

Q1. What is difference between Error and Exception?

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you can not repair them at runtime.Though error can be caught in catch-block but the execution of application will come to a halt and is not recoverable.
While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.
Q2. How can you handle Java exceptions?

There are five keywords used to handle exceptions in java:

1. try
2. catch
3. finally
4. throw
5. throws

Q3. What are the differences between Checked Exception and Unchecked Exception?

Checked Exception

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
- Checked exceptions are checked at compile-time.
- Example: IOException, SQLException etc.

Unchecked Exception

- The classes that extend RuntimeException are known as unchecked exceptions.
- Unchecked exceptions are not checked at compile-time.
- Example: ArithmeticException, NullPointerException etc.

Q4. What purpose does the keywords final, finally, and finalize fulfill?

Final:
Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden, and final variable value can't be changed. Let's take a look at the example below to understand it better.

```
1
2     class FinalVarExample {
3           public static void main(String args[]) {
4                 final int a = 10; // Final variable
5                 a = 50; // Error as value can't be changed
6           }
```

Finally
Finally is used to place important code, it will be executed whether exception is handled or not. Let's take a look at the example below to understand it better.

```
1
2     class FinallyExample {
3           public static void main(String args[]) {
4                 try {
5                       int x = 100;
6                 } catch (Exception e) {
7                       System.out.println(e);
8                 } finally {
9                       System.out.println("finally block is executing");
10                }
11          }
12    }
```

Finalize
Finalize is used to perform clean up processing just before object is garbage collected. Let's take a look at the example below to understand it better.

```
1     class FinalizeExample {
2           public void finalize() {
3                 System.out.println("Finalize is called");
4           }
5
6           public static void main(String args[]) {
7                 FinalizeExample f1 = new FinalizeExample();
8                 FinalizeExample f2 = new FinalizeExample();
9                 f1 = NULL;
```

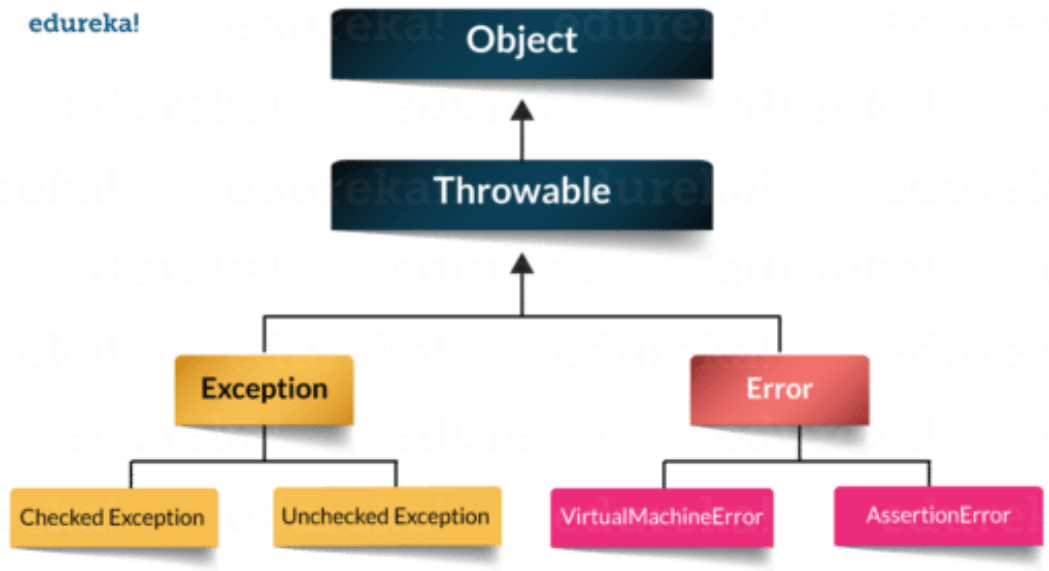
```
10                  f2 = NULL;
11                  System.gc();
12              }
13      }
```

Q5. What are the differences between throw and throws?

throw keyword	throws keyword
Throw is used to explicitly throw an exception.	Throws is used to declare an exception.
Checked exceptions can not be propagated with throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

Q6. What is exception hierarchy in java?

The hierarchy is as follows:
Throwable is a parent class of all Exception classes. There are two types of Exceptions: Checked exceptions and UncheckedExceptions or RunTimeExceptions. Both type of exceptions extends Exception class whereas errors are further classified into Virtual Machine error and Assertion error.



Q7. How to create a custom Exception?

To create you own exception extend the Exception class or any of its subclasses.

- class New1Exception extends Exception { } // this will create Checked Exception
- class NewException extends IOException { } // this will create Checked exception
- class NewException extends NullPonterExcpetion { } // this will create UnChecked exception

Q8. What are the important methods of Java Exception Class?

Exception and all of its subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable.

1. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it's constructor.
2. **String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

Q9. What are the differences between processes and threads?

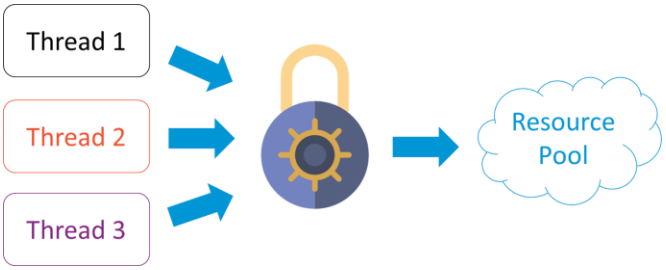
	Process	Thread
Definition	An executing instance of a program is called a process.	A thread is a subset of the process.
Communication	Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with other threads of its process.
Control	Processes can only exercise control over child processes.	Threads can exercise considerable control over threads of the same process.
Changes	Any change in the parent process does not affect child processes.	Any change in the main thread may affect the behavior of the other threads of the process.
Memory	Run in separate memory spaces.	Run in shared memory spaces.
Controlled by	Process is controlled by the operating system.	Threads are controlled by programmer in a program.
Dependence	Processes are independent.	Threads are dependent.

Q10. What is a finally block? Is there a case when finally will not execute?

Finally block is a block which always executes a set of statements. It is always associated with a try block regardless of any exception that occurs or not.
Yes, finally will not be executed if the program exits either by calling System.exit() or by causing a fatal error that causes the process to abort.

Q11. What is synchronization?

Synchronization refers to multi-threading. A synchronized block of code can be executed by only one thread at a time. As Java supports execution of multiple threads, two or more threads may access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in sync. Synchronization avoids memory consistency errors caused due to inconsistent view of shared memory. When a method is declared as synchronized the thread holds the monitor for that method's object. If another thread is executing the synchronized method the thread is blocked until that thread releases the monitor.



Q12. Can we write multiple catch blocks under single try block?

Yes, we can have multiple catch blocks under single try block but the approach should be from specific to general. Let's understand this with a programmatic example.

```
1
2
3
4     public class Example {
5           public static void main(String args[]) {
6                 try {
7                       int a[] = new int[10];
8                       a[10] = 10 / 0;
9                 } catch (ArithmeticException e) {
10                      System.out.println("Arithmetic exception in first catch block");
11                } catch (ArrayIndexOutOfBoundsException e) {
12                      System.out.println("Array index out of bounds in second catch block");
13                } catch (Exception e) {
14                      System.out.println("Any exception in third catch block");
15                }
16          }
17
18
19
```

Q13. What are the important methods of Java Exception Class?

Methods are defined in the base class Throwable. Some of the important methods of Java exception class are stated below.

1. **String getMessage()** – This method returns the message String about the exception . The message can be provided through its constructor.
2. **public StackTraceElement[] getStackTrace() –** This method returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack whereas the last element in the array represents the method at the bottom of the call stack.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id as represented by a Throwable object.
4. **String toString()** – This method returns the information in String format. The returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream.