https://www.youtube.com/watch?v=mLMvGeb-e-0

**SPRING**

**Learn what you can do with Spring Boot?**

Spring Boot offers a fast way to build applications. It looks at your classpath and at beans you have configured, makes reasonable assumptions about what you’re missing, and adds it. With Spring Boot you can focus more on business features and less on infrastructure.

For example:

* Got Spring MVC? There are several specific beans you almost always need, and Spring Boot adds them automatically. A Spring MVC app also needs a servlet container, so Spring Boot automatically configures embedded Tomcat.
* Got Jetty? If so, you probably do NOT want Tomcat, but instead embedded Jetty. Spring Boot handles that for you.
* Got Thymeleaf? There are a few beans that must always be added to your application context; Spring Boot adds them for you.

These are just a few examples of the automatic configuration Spring Boot provides. At the same time, Spring Boot doesn’t get in your way. For example, if Thymeleaf is on your path, Spring Boot adds a SpringTemplateEngine to your application context automatically. But if you define your own SpringTemplateEngine with your own settings, then Spring Boot won’t add one. This leaves you in control with little effort on your part.

|  |  |
| --- | --- |
|  | Spring Boot doesn’t generate code or make edits to your files. Instead, when you start up your application, Spring Boot dynamically wires up beans and settings and applies them to your application context. |

**What is bean wiring?**

Combining together beans within the spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

**Explain Bean lifecycle in spring framework?**

1. The spring container finds the bean’s definition from the XML file and instantiates the bean.
2. Using the dependency injection, spring populates all of the properties as specified in the bean definition.
3. If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName**() passing the bean’s ID.
4. If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory**(), passing an instance of itself.
5. If there are any BeanPostProcessors associated with the bean, their post- **ProcessBeforeInitialization**() methods will be called.
6. If an init-method is specified for the bean, it will be called.
7. Finally, if there are any BeanPostProcessors associated with the bean, their **postProcessAfterInitialization**() methods will be called.

**What are important ApplicationContext implementations in spring framework?**

**ClassPathXmlApplicationContext** – This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources.

**FileSystemXmlApplicationContext** – This context loads a context definition from an XML file in the filesystem.

**XmlWebApplicationContext** – This context loads the context definitions from an XML file contained within a web application.

**Difference between Dirty Read, Non Repeatable Read and Phantom Read in Database.**

D**irty Read:-**

Dirty read occurs when one transaction is changing the record, and the other transaction can read this record before the first transaction has been committed or rolled back. This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid data.

**Dirty Read Example:-**

Transaction A begins.  
UPDATE EMPLOYEE SET SALARY = 10000 WHERE EMP\_ID= ‘123’;

Transaction B begins.  
SELECT \* FROM EMPLOYEE;  
(Transaction B sees data which is updated by transaction A. But, those updates have not yet been committed.)

**Non-Repeatable Read:-**

Non Repeatable Reads happen when in a same transaction same query yields to a different result. This occurs when one transaction repeatedly retrieves the data, while a difference transaction alters the underlying data. This causes the different or non-repeatable results to be read by the first transaction.

**Non-Repeatable Example:-**

Transaction A begins.  
SELECT \* FROM EMPLOYEE WHERE EMP\_ID= ‘123’;

Transaction B begins.  
UPDATE EMPLOYEE SET SALARY = 20000 WHERE EMP\_ID= ‘123’;  
(Transaction B updates rows viewed by the transaction A before transaction B commits.) If Transaction A issues the same SELECT statement, the results will be different.

**Phantom Read:-**

Phantom read occurs where in a transaction execute same query more than once, and the second transaction result set includes rows that were not visible in the first result set. This is caused by another transaction inserting new rows between the execution of the two queries. This is similar to a non-repeatable read, except that the number of rows is changed either by insertion or by deletion.

**Phantom Read Example:-**

Transaction A begins.  
SELECT \* FROM EMPLOYEE WHERE SALARY > 10000 ;

Transaction B begins.  
INSERT INTO EMPLOYEE (EMP\_ID, FIRST\_NAME, DEPT\_ID, SALARY) VALUES (‘111′, ‘Jamie’, 10, 35000);  
Transaction B inserts a row that would satisfy the query in Transaction A if it were issued again.

**Following are the possible values for isolation level:**

|  |  |
| --- | --- |
| S.N. | Isolation & Description |
| 1 | TransactionDefinition.ISOLATION\_DEFAULT  This is the default isolation level. |
| 2 | TransactionDefinition.ISOLATION\_READ\_COMMITTED  Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur. |
| 3 | TransactionDefinition.ISOLATION\_READ\_UNCOMMITTED  Indicates that dirty reads, non-repeatable reads and phantom reads can occur. |
| 4 | TransactionDefinition.ISOLATION\_REPEATABLE\_READ  Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur. |
| 5 | TransactionDefinition.ISOLATION\_SERIALIZABLE  Indicates that dirty reads, non-repeatable reads and phantom reads are prevented. |

**Following are the possible values for propagation types:**

|  |  |
| --- | --- |
| S.N. | Propagation & Description |
| 1 | TransactionDefinition.PROPAGATION\_MANDATORY  Support a current transaction; throw an exception if no current transaction exists. |
| 2 | TransactionDefinition.PROPAGATION\_NESTED  Execute within a nested transaction if a current transaction exists. |
| 3 | TransactionDefinition.PROPAGATION\_NEVER  Do not support a current transaction; throw an exception if a current transaction exists. |
| 4 | TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED  Do not support a current transaction; rather always execute non-transactionally. |
| 5 | TransactionDefinition.PROPAGATION\_REQUIRED  Support a current transaction; create a new one if none exists. |
| 6 | TransactionDefinition.PROPAGATION\_REQUIRES\_NEW  Create a new transaction, suspending the current transaction if one exists. |
| 7 | TransactionDefinition.PROPAGATION\_SUPPORTS  Support a current transaction; execute non-transactionally if none exists. |
| 8 | TransactionDefinition.TIMEOUT\_DEFAULT  Use the default timeout of the underlying transaction system, or none if timeouts are not supported. |

**What are different modules in SPRING?**

* The Core container module
* Application context module
* AOP module (Aspect Oriented Programming)
* JDBC abstraction and DAO module
* O/R mapping integration module (Object/Relational)
* Web module
* MVC framework module

**What is the Core container module?**

This module provides the fundamental functionality of the spring framework. In this module **BeanFactory** is the heart of any spring-based application. The entire framework was built on the top of this module. This module makes the Spring container.

**Why most users of the Spring Framework choose declarative transaction management?**

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container.

**What is RowCallbackHandler?**

The **RowCallbackHandler** interface extracts values from each row of a ResultSet.

Has one method – processRow (ResultSet) called for each row in ResultSet Typically stateful?

**What is Application context module?**

The Application context module makes spring a framework. This module extends the concept of BeanFactory, providing support for **internationalization** (I18N) **messages, application lifecycle events, and validation**. This module also supplies many enterprise services such **JNDI access, EJB integration, remoting, and scheduling**. It also provides support to other framework.

#### What is AOP module?

The AOP module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between spring and other AOP frameworks. This module also introduces metadata programming to spring. Using spring’s metadata support, we will be able to add annotations to our source code that instruct spring on where and how to apply aspects.

#### What is JDBC abstraction and DAO module?

Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses Spring’s AOP module to provide transaction management services for objects in a spring application.

#### What are object/relational mapping integration module?

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provide support to tie into several popular ORM frameworks, including Hibernate, JDO, and iBATIS SQL Maps. Spring’s transaction management supports each of these ORM frameworks as well as JDBC.

#### What is web module?

Spring comes with a full-featured MVC framework for building web applications. Although spring can easily be integrated with other MVC frameworks, such as Struts, spring’s MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to declaratively bind request parameters to your business objects. It also can take advantage of any of spring’s other services, such as **I18N messaging and validation.**

**Struts-2 integration with spring**

**In Web.xml**

<filter-class>

**org.apache.struts2.dispatcher.ng.filter**.**StrutsPrepareAndExecuteFilter**

</filter-class>

</filter>

<listener-class>

**org.springframework.web.context.ContextLoaderListener**

</listener-class>

**What is Bean Factory, Have you used XMLBean Factory?**

A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application’s configuration and dependencies from the actual application code.

XMBeanFactory is one of the implementation of bean factory **org.springframework.beans.factory.xml.XmlBeanFactory** is used to create bean instance defined in our xml file.

BeanFactory has many implementations in spring. But one of the most useful one is **org.springframework.beans.factory.xml.XmlBeanFactory**, which loads its beans based on the definitions contained in an XML file. To create an **XmlBeanFactory**, pass a **java.io.InputStream** to the constructor. The InputStream will provide the XML to the factory. For example, the following code snippet uses a java.io.FileInputStream to provide a bean definition XML file to XmlBeanFactory.

BeanFactory factory=new XmlBeanFactory(new FileInputStream(“beans.xml””));

(Or)

ClassPathResource resource=new ClassPathResource(“beans.xml”);

XmlBeansFactory factory=new XmlBeansFactory(resource);

**Cache vs. Buffer**

The terms "**buffer**" and "cache" tend to be used interchangeably; note however they represent different things. **A buffer is used traditionally as an intermediate temporary store for data between a fast and a slow entity**. As one party would have to *wait* for the other affecting performance, the buffer alleviates this by allowing entire blocks of data to move at once rather then in small chunks. The data is written and read only once from the buffer. Furthermore, the buffers are *visible* to at least one party which is aware of it.

A **cache** on the other hand by definition is hidden and neither party is aware that caching occurs. It as well improves performance but does that by allowing the same data to be read multiple times in a fast fashion.

**@Cacheable annotation**

As the name implies, @**Cacheable** is used to demarcate methods that are cacheable - that is, methods for whom the result is stored into the cache so on subsequent invocations (with the same arguments), the value in the cache is returned without having to actually execute the method. In its simplest form, the annotation declaration requires the name of the cache associated with the annotated method:

**@Cacheable("books")**

**public Book findBook(ISBN isbn) {...}**

In the snippet above, the method findBook is associated with the cache named books. Each time the method is called, the cache is checked to see whether the invocation has been already executed and does not have to be repeated. While in most cases, only one cache is declared, the annotation allows multiple names to be specified so that more than one cache are being used. In this case, each of the caches will be checked before executing the method - if at least one cache is hit, then the associated value will be returned:

|  |  |
| --- | --- |
|  | All the other caches that do not contain the value will be updated as well even though the cached method was not actually executed. |

@Cacheable({"books", "isbns"})

public Book findBook(ISBN isbn) {...}

**Enable caching annotations**

It is important to note that even though declaring the cache annotations does not automatically trigger their actions - like many things in Spring, the feature has to be declaratively enabled (which means if you ever suspect caching is to blame, you can disable it by removing only one configuration line rather than all the annotations in your code).

To enable caching annotations add the annotation @**EnableCaching** to one of your @Configuration classes:

**@Configuration**

**@EnableCaching**

public class AppConfig {

}

Alternatively for XML configuration use the **cache:annotation-driven** element:

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:cache="http://www.springframework.org/schema/cache"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

**http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.xsd">**

**<cache:annotation-driven />**

</beans>

Both the **cache:annotation-driven** element and **@EnableCaching** annotation allow various options to be specified that influence the way the caching behavior is added to the application through AOP.

The configuration is intentionally similar with that of [@Transactional](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#tx-annotation-driven-settings):

The **tx** tags deal with configuring all of those beans in Spring’s comprehensive support for transactions.

xmlns:tx="http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

The **aop** tags deal with configuring all things AOP in Spring: this includes Spring’s own proxy-based AOP framework and Spring’s integration with the AspectJ AOP framework.

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd"

The **context** tags deal with Application Context configuration that relates to plumbing - that is, not usually beans that are important to an end-user but rather beans that do a lot of grunt work in Spring, such as BeanfactoryPostProcessors.

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd

The **jms** tags deal with configuring JMS-related beans such as Spring’s [MessageListenerContainers](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#jms-mdp).

ttp://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms.xsd

The **lang** tags deal with exposing objects that have been written in a dynamic language such as JRuby or Groovy as beans in the Spring container.

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd

The **jee** tags deal with Java EE (Java Enterprise Edition)-related configuration issues, such as looking up a JNDI object and defining EJB references.

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd

**Defining new Advice types**

Spring AOP is designed to be extensible. While the interception implementation strategy is presently used internally, it is possible to support arbitrary advice types in addition to the out-of-the-box interception around advice, before, throws advice and after returning advice.

The **org.springframework.aop.framework.adapter** package is an SPI package allowing support for new custom advice types to be added without changing the core framework. The only constraint on a custom Advice type is that it must implement the **org.aopalliance.aop.Advice** tag interface.

**How to filter components in auto scanning?**

To include and exclude components based on your requirements. You can filter your components during enabling auto component scanning level, it should be configured at xml based configuration level. Here is an example for including auto scan components:

Here is the xml based configuration, which includes a filter to include components; the filter is specified using "include-filter" tag. Note that below filtering components based on regular expression, and annotation based. Below xml configuration including all Components annotated with @Component annotation and any class containing "Service" String.

 <context:component-scan base-package="com.java2novice">

        <context:include-filter type="regex"

            expression="com.java2novice.service.\*Service.\*" />

        <context:include-filter type="annotation"

            expression="org.springframework.stereotype.Component" />

     </context:component-scan>

**How to read property file in spring 3.0 using java based configuration?**

@Configuration

@PropertySource("classpath:/db.properties")

Public class MyApplicationConfig {

@Autowired

Environment env;

@Bean (name="dbConfig")

Public MyDbConfig getDbConfig (){

**How to read property file in spring using xml based configuration file?**

To load property file from the classpath using xml based configuration. Declare your property file in your xml based configuration file using "context: property-placeholder" tag, and refer property key any where in the xml based configuration file using ${db.host.url} syntax.

In the spring bean configurations, bean attribute called 'scope' defines what kind of object has to created and returned. There are 5 types of bean scopes available, they are:

1) **singleton**: Returns a single bean instance per Spring IoC container.

2) **prototype**: Returns a new bean instance each time when requested.

3) **request**: Returns a single instance for every HTTP request call.

4) **session**: Returns a single instance for every HTTP session.

5) **global session**: global session scope is equal as session scope on portlet-based web applications.

If no bean scope is specified in bean configuration file, then it will be by default 'singleton'.

In spring the beans are managed by Spring IoC container, these are backbone of the application. You can instantiate and manage them in your application using configurations. In xml based spring bean configurations, using <bean> tag, you can manage them. Here we have given complete list of bean tag properties:

**name / id:**

This attribute specifies the bean unique identifier. In XML based configuration metadata, you use the id and/or name attributes to specify the bean identifier.

**class:**

This attribute is mandatory and specify the bean class to be used to create the bean. You should specify fully qualified class name. Include package name.

**scope:**

This attribute specifies the scope of the objects created from a particular bean definition. The scope values can be prototype, singleton, request, session, and global session.

**constructor-arg:**

This is used to inject the dependencies through bean constructor.

**properties:**

This attribute is used to inject the dependencies through setter method.

**autowiring mode:**

This is used to inject the dependencies.

**lazy-init** (lazy-initialization mode):

A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at startup.

**init-method** (initialization method):

A callback to be called just after all necessary properties on the bean have been set by the container. This is part of bean lifecycle.

**destroy-method** (destruction method):

A callback to be used when the container containing the bean is destroyed. This is part of bean lifecycle.

Configure default initialization and destroy methods in all spring beans

  default-init-method="init"

     default-destroy-method="destroy">

Configure spring bean initialization and destroy method calls using java annotations @**PostConstruct** and @**PreDestroy**. These annotations are not belong to spring API, these are part of J2ee library common-annotations.jar file

Spring Annotations: Contents:

|  |  |
| --- | --- |
| **Annotation** | **Package Detail/Import statement** |
| [@Service](http://www.techferry.com/articles/spring-annotations.html#Service) | import org.springframework.stereotype.Service; |
| [@Repository](http://www.techferry.com/articles/spring-annotations.html#Repository) | import org.springframework.stereotype.Repository; |
| [@Component](http://www.techferry.com/articles/spring-annotations.html#Component) | import org.springframework.stereotype.Component; |
| [@Autowired](http://www.techferry.com/articles/spring-annotations.html#Autowired) | import org.springframework.beans.factory.annotation.Autowired; |
| [@Transactional](http://www.techferry.com/articles/spring-annotations.html#Transactional) | import org.springframework.transaction.annotation.Transactional; |
| [@Scope](http://www.techferry.com/articles/spring-annotations.html#Scope) | import org.springframework.context.annotation.Scope; |
| [**Spring MVC Annotations**](http://www.techferry.com/articles/spring-annotations.html#MVC) | |
| [@Controller](http://www.techferry.com/articles/spring-annotations.html#Controller) | import org.springframework.stereotype.Controller; |
| [@RequestMapping](http://www.techferry.com/articles/spring-annotations.html#RequestMapping) | import org.springframework.web.bind.annotation.RequestMapping; |
| [@PathVariable](http://www.techferry.com/articles/spring-annotations.html#PathVariable) | import org.springframework.web.bind.annotation.PathVariable; |
| [@RequestParam](http://www.techferry.com/articles/spring-annotations.html#RequestParam) | import org.springframework.web.bind.annotation.RequestParam; |
| [@ModelAttribute](http://www.techferry.com/articles/spring-annotations.html#ModelAttribute) | import org.springframework.web.bind.annotation.ModelAttribute; |
| [@SessionAttributes](http://www.techferry.com/articles/spring-annotations.html#SessionAttributes) | import org.springframework.web.bind.annotation.SessionAttributes; |
| [**Spring Security Annotations**](http://www.techferry.com/articles/spring-annotations.html#SpringSecurity) | |
| [@PreAuthorize](http://www.techferry.com/articles/spring-annotations.html#PreAuthorize) | import org.springframework.security.access.prepost.PreAuthorize; |

For spring to process annotations, add the following lines in your application-context.xml file.

**<context:annotation-config />**

**<context:component-scan base-package="...specify your package name..." />**

@**Service**

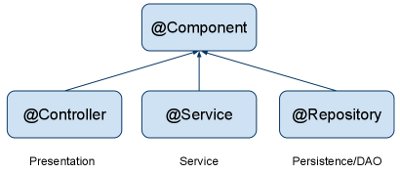
Annotate all your service classes with @**Service**. All your business logic should be in Service classes.

@**Repository**

Annotate all your DAO classes with @**Repository**. All your database access logic should be in DAO classes

@**Component**

Annotate your other components (for example REST resource classes).

@**Component** is a generic stereotype for any Spring-managed component. **@Repository, @Service, and @Controller** are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.   
  
@**Autowired**

Let spring auto-wire other beans into your classes using @Autowired annotation. Spring beans can be wired by name or by type.

@**Autowire** by default is a type driven injection.

@**Qualifier** spring annotation can be used to further fine-tune autowiring.

@**Resource** (javax.annotation.Resource) annotation can be used for wiring by name. Beans that are themselves defined as a **collection** or **map** type cannot be injected through @**Autowired**, because type matching is not properly applicable to them. Use @**Resource** for such beans, referring to the specific collection or map bean by unique name.

@**Transactional**

Configure your transactions with @Transactional spring annotation.

To activate processing of spring’s @Transactional annotation, use the **<tx: annotation-driven**/> element in your spring's configuration file.

The default @Transactional settings are as follows:

Propagation setting is PROPAGATION\_REQUIRED.

Isolation level is ISOLATION\_DEFAULT.

Transaction is read/write.

Transaction timeout defaults to the default timeout of the underlying transaction system or to none if timeouts are not supported.

Any Runtime Exception triggers rollback, and any checked Exception does not.

These default settings can be changed using various properties of the

@**Transactional** spring annotation.

Specifying the @Transactional annotation on the bean class means that it applies to all applicable business methods of the class. Specifying the annotation on a method applies it to that method only. If the annotation is applied at both the class and the method level, the method value overrides if the two disagree

@**Scope**

As with Spring-managed components in general, the default and most common scope for auto detected components is singleton. To change this default behavior, use @Scope spring annotation.

@**Controller**

Annotate your controller classes with @Controller.

@**RequestMapping**

You use the @RequestMapping spring annotation to map URLs onto an entire class or a particular handler method. Typically the class-level annotation maps a specific request path (or path pattern) onto a form controller, with additional method-level annotations narrowing the primary mapping.

@**PathVariable**

You can use the @PathVariable spring annotation on a method argument to bind it to the value of a URI template variable. In our example below, a request path of /company/techferry will bind companyName variable with 'techferry' value.

@**RequestParam**

You can bind request parameters to method variables using spring annotation

@**RequestParam**

Similarly, you can use spring annotation @RequestHeader to bind request headers

@**ModelAttribute**

An @ModelAttribute on a method argument indicates the argument should be retrieved from the model. If not present in the model, the argument should be instantiated first and then added to the model. Once present in the model, the argument's fields should be populated from all request parameters that have matching names. This is known as data binding in Spring MVC, a very useful mechanism that saves you from having to parse each form field individually.

@SessionAttributes

@**SessionAttributes** spring annotation declares session attributes. This will typically list the names of model attributes which should be transparently stored in the session, serving as form-backing beans between subsequent requests.

@**SessionAttribute** works as follows:

is initialized when you put the corresponding attribute into model (either explicitly or using @ModelAttribute-annotated method).

is updated by the data from HTTP parameters when controller method with the corresponding model attribute in its signature is invoked.

@**SessionAttributes** are cleared when you call setComplete () on SessionStatus object passed into controller method as an argument.

@**PreAuthorize**

Using Spring Security @PreAuthorize annotation, you can authorize or deny functionality. In our example below, only a user with Admin role has the access to delete a contact.

**What is JdbcTemplate in spring?**

**org.springframework.jdbc.core.Jdbc. JdbcTemplate**

The **JdbcTemplate** class is the central class in the JDBC core package. It simplifies the use of JDBC since it handles the creation and release of resources. This helps to avoid common errors such as forgetting to always close the connection. It executes the core JDBC workflow like statement creation and execution, leaving application code to provide SQL and extract results. This class executes SQL queries, update statements or stored procedure calls, imitating iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

The **NamedParameterJdbcTemplate** class helps you specify the named parameters instead of classic placeholder('?') argument.

Named parameters improve readability and are easier to maintain.  
  
The **NamedParameterJdbcTemplate** provide better approach than [JdbcTemplate](http://www.dineshonjava.com/2012/12/using-jdbctemplate-in-spring-chapter-33.html#.UMNSg4aeWk9) ,where multiple parameters are in use for an SQL statement. It eliminated need of traditional JDBC "?" and provide named parameters. It is easy to use and provide better documentation. It functionality is similar to [JdbcTemplate](http://www.dineshonjava.com/2012/12/using-jdbctemplate-in-spring-chapter-33.html#.UMNSg4aeWk9) except it incorporate named parameters instead of "?" placeholder.   
  
In [**JdbcTemplate**](http://www.dineshonjava.com/2012/12/using-jdbctemplate-in-spring-chapter-33.html#.UMNSg4aeWk9), SQL parameters are represented by a special placeholder “?” symbol and bind it by position. The problem is whenever the order of parameter is changed, you have to change the parameters bindings as well, it’s error prone and cumbersome to maintain it.

<bean id="dataSource" class="org.springframework.jdbc.datasource.**DriverManagerDataSource**">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:<mysql://localhost:3306/java2novice>" />

<property name="username" value="user\_name" />

<property name="password" value="password" />

</bean>

<bean id="employeeDAO" class="com.java2novice.dao.EmployeeDaoImpl">

<property name="dataSource" ref="dataSource" />

</bean>

**What is Dependency Injection in spring?**

Dependency Injection is an aspect of inversion of control is a general concept, and it can be expressed in many different ways. This concept says that you do not create your object but describe how they should be created. You don’t directly connect your components and services together in code but describe which services are needed by which components in a spring configuration file. A container (the IOC container) is then responsible for hooking it all up.

Java Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application **loosely coupled**, **extendable** and **maintainable**. We can implement dependency injection in java to move the dependency resolution from compile-time to runtime.

To use dependency injection pattern to achieve loose coupling and extendibility in the application

.

Whether to use Constructor based dependency injection or setter based is a design decision and depends on your requirements. For example, if my application can’t work at all without the service class then I would prefer constructor based DI or else I would go for setter method based DI to use it only when it’s really needed.

Dependency Injection in Java is a way to achieve Inversion of control (IoC) in our application by moving objects binding from compile time to runtime. We can achieve IoC through **Factory Pattern, Template Method Design Pattern, Strategy Pattern and Service Locator pattern** too.

Spring Dependency Injection, Google Guice and Java EE CDI frameworks facilitate the process of dependency injection through use of Java Reflection API and java annotations. All we need is to annotate the field, constructor or setter method and configure them in configuration xml files or classes.

**Benefits of Java Dependency Injection?**

Some of the benefits of using Dependency Injection in Java are:

Separation of Concerns Boilerplate Code reduction in application classes because all work to initialize dependencies is handled by the injector component Configurable components makes application easily extendable

Unit testing is easy with mock object.

**Disadvantages of Java Dependency Injection?**

**Java Dependency injection has some disadvantages too:**

If overused, it can lead to maintenance issues because effect of changes is known at runtime. Dependency injection in java hides the service class dependencies that can lead to runtime errors that would have been caught at compile time.

**What are the different types of IOC (dependency injection)?**

**Constructor based DI** is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

**Setter-based dependency injection**: Setter based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

**Aspect Oriented Programming Core Concepts?**

Most of the enterprise applications have some common crosscutting concerns that is applicable for different types of Objects and modules. Some of the common crosscutting concerns are **logging**, **transaction management**, **data validation** etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in Aspect Oriented Programming application modularity is achieved by Aspects and they are configured to cut across different classes.

Spring AOP takes out the direct dependency of crosscutting tasks from classes that we can’t achieve through normal object oriented programming model. For example, we can have a separate class for logging but again the functional classes will have to call these methods to achieve logging across the application.

**Aspect**: An aspect is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management. Aspects can be a normal class configured through Spring XML configuration or we can use Spring **AspectJ** integration to define a class as Aspect using @Aspect annotation.

**<!-- Enable AspectJ style of Spring AOP -->**

**<aop:aspectj-autoproxy />**

**Join Point**: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

**Advice**: Advices are actions taken for a particular join point. In terms of programming, they are methods that gets executed when a certain join point with matching pointcut is reached in the application. You can think of Advices as [Struts2 interceptors](http://www.journaldev.com/2210/struts-2-interceptor-example) or [Servlet Filters](http://www.journaldev.com/1933/java-servlet-filter-example-tutorial).

**Before advice** - @Aspect , @**Before**

**These advices runs before the execution of join point methods. We can use @Before annotation to mark an advice type as Before advice.**

**After advice** - @**After**

**An advice that gets executed after the join point method finishes executing, whether normally or by throwing an exception. We can create after advice using @After annotation.**

**After returning advice** - @**AfterRunning**

**Sometimes we want advice methods to execute only if the join point method executes normally. We can use @AfterReturning annotation to mark a method as after returning advice.**

**After throwing advice** - @**AfterThrowing**

**This advice gets executed only when join point method throws exception, we can use it to rollback the transaction declaratively. We use @AfterThrowing annotation for this type of advice.**

**Around advice - @Around**

**This is the most important and powerful advice. This advice surrounds the join point method and we can also choose whether to execute the join point method or not. We can write advice code that gets executed before and after the execution of the join point method. It is the responsibility of around advice to invoke the join point method and return values if the method is returning something. We use @Around annotation to create around advice methods.**

org.aspectj.lang.annotation.Aspect

org.aspectj.lang.annotation.Before

org.aspectj.lang.annotation.After

org.aspectj.lang.annotation.AfterReturning

org.aspectj.lang.annotation.AfterThrowing

org.aspectj.lang.annotation.Around

**Pointcut**: **Pointcut are expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points and Spring framework uses the AspectJ pointcut expression language.**

**@Before("execution(public String getName())")**

**public void getNameAdvice(){**

**System.out.println("Executing Advice on getName()");**

**}**

**Target Object**: **They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.**

**AOP proxy**: **Spring AOP implementation uses JDK dynamic proxy to create the Proxy classes with target classes and advice invocations; these are called AOP proxy classes. We can also use CGLIB proxy by adding it as the dependency in the Spring AOP project.**

**Weaving**: **It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime**.

**What is the difference between concern and cross-cutting concern in Spring AOP?**

Concern **is behavior which we want to have in a module of an application. Concern may be defined as a functionality we want to implement to solve a specific business problem. E.g. in any ecommerce application different concerns (or modules) may be inventory management, shipping management, user management etc.**

**Cross-cutting concern is a concern which is applicable throughout the application (or more than one module). e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence they are termed as cross-cutting concerns.**

**RESTFUL**

**RestTemplate**

**The** RestTemplate **is the core class for client-side access to RESTful services. It is conceptually similar to other template classes in Spring, such as JdbcTemplate and JmsTemplate and other template classes found in other Spring portfolio projects. RestTemplate’s behavior is customized by providing callback methods and configuring the `HttpMessageConverter used to marshal objects into the HTTP request body and to unmarshal any response back into an object. As it is common to use XML as a message format, Spring provides a** MarshallingHttpMessageConverter **that uses the Object-to-XML framework that is part of the org.springframework.**oxm **package. This gives you a wide range of choices of XML to Object mapping technologies to choose from.**

**Create AppConfig.java file under /src folder. Give appropriate package name to your file. We are using @EnableWebMvc, @ComponentScan and @Configuration annotations. These will bootstrap the spring mvc application and set package to scan controllers and resources.**

**@ComponentScan (basePackages = {"com.javarticles.spring.annotations.packageA",**

**"com.javarticles.spring.annotations.packageB" })**

**What is REST?**

**In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.**

What are Restful web services?

In the web services terms, Representational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web services client uses that URI to access the resource.  
It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.

SOAP vs REST Web Services

There are many differences between SOAP and REST web services. The important differences between SOAP and REST are given below:

|  |  |  |
| --- | --- | --- |
| No. | SOAP | REST |
| 1) | SOAP is a protocol. | REST is an architectural style. |
| 2) | SOAP stands for Simple Object Access Protocol. | REST stands for Representational State Transfer. |
| 3) | SOAP can't use REST because it is a protocol. | REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP uses services interfaces to expose the business logic. | REST uses URI to expose business logic. |
| 5) | JAX-WS is the java API for SOAP web services. | JAX-RS is the java API for RESTful web services. |
| 6) | SOAP defines standards to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP requires more bandwidth and resource than REST. | REST requires less bandwidth and resource than SOAP. |
| 8) | SOAP defines its own security. | Restful web services inherit security measures from the underlying transport. |
| 9) | SOAP permits XML data format only. | REST permits different data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is less preferred than REST. | REST more preferred than SOAP. |
| 11. | JavaScript can call SOAP, but it is difficult to implement | Easy to call from JavaScript |
| 12 | Performance is not great compared to REST. | Performance is much better compared to SOAP less CPU intensive, leaner code etc. |
| 13. | Transfer is over HTTP, also uses other protocols such as SMTP, FTP etc. | Transfer is over HTTP only. |

**What Are Some of the Annotations Defined by JAX-RS?**

Here is a listing of some of the Java programming annotations that are defined by JAX-RS, with a brief description of how each is used.

Table 3–1 Summary of Jersey Annotations

| Annotation | Description |
| --- | --- |
| @Path | The @**Path** annotation's value is a relative URI path indicating where the Java class will be hosted, for example,/helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user, and pass it to the application as a variable in the URI, like this, /helloworld/{username}. |
| @GET | The @**GET** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @POST | The @**POST** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PUT | The @**PUT** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @DELETE | The @**DELETE** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @HEAD | The @**HEAD** annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PathParam | The @**PathParam** annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation. |
| @QueryParam | The @QueryParam annotation is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters. |
| @Consumes | The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client. |
| @Produces | The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client, for example, "text/plain". |
| @Provider | The @Provider annotation is used for anything that is of interest to the JAX-RS runtime, such as **MessageBodyReader** and **MessageBodyWriter**. For HTTP requests, the MessageBodyReader is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body using a **MessageBodyWriter**. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a Response that wraps the entity, and which can be built using Response.**ResponseBuilder**. |
| @MatrixParam | Matrix parameters are a set of “**name=value**” in URI path. The matrix parameter is “**author=mkyong**“, separate by a semi colon “**;**“. |
| @HeaderParam | extracts information from the HTTP headers. |
| @CookieParam | extracts information from the cookies declared in cookie related HTTP headers. |
| @FormParam | to bind HTML form parameters value to a Java method |

**Requirements of a JAX-WS Endpoint?**

JAX-WS endpoints must follow these requirements:

The implementing class must be annotated with either the **javax.jws.WebService or javax.jws.WebServiceProvider** annotation.

The implementing class may explicitly reference an SEI through the endpoint Interface element of the @**WebService** annotation, but is not required to do so. If no endpoint Interface is not specified in @WebService, an SEI is implicitly defined for the implementing class.

The business methods of the implementing class must be public, and must not be declared static or final. Business methods that are exposed to web service clients must be annotated with **javax.jws.WebMethod**. Business methods that are exposed to web service clients must have JAX-B-compatible parameters and return types. See [Default Data Type Bindings](https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/JAXBWorks4.html#wp82947).

The implementing class must not be declared final and must not be abstract.

The implementing class must have a default public constructor.

The implementing class must not define the finalize method.

The implementing class may use the javax**.annotation.PostConstruct** or **javax.annotation.PreDestroy** annotations on its methods for lifecycle event callbacks.

The **@PostConstruct** method is called by the container before the implementing class begins responding to web service clients.

The **@PreDestroy** method is called by the container before the endpoint is removed from operation.

# RESTful Web Services Security Guide

Knowledge of securing RESTful Web Services is as much important as to write them. Mostly RESTful APIs are HTTP protocol based, you any user having internet connection can connect to it, and so hackers as well. It’s very important to write secure APIs to protect the business. But before starting to secure RESTful APIs, let’s understand what are our options as developers? What will be good fit for our usecase?

## 4 Ways to Secure RESTful Web Services

There are multiple ways to secure a RESTful API in Java. Let’s go through 4 most popular choices:

#### BASIC Authentication

It’s simplest of all techniques and probably most used as well. You use login/password forms – it’s basic authentication only. You input your username and password and submit the form to server, and application identify you as a user – you are allowed to use the system – else you get error.

The main problem with this security implementation is that credentials are propagated in a plain way from the client to the server. Credentials are merely encoded with Base64 in transit, but not encrypted or hashed in any way. This way, any sniffer could read the sent packages over the network.

HTTPS is, therefore, typically preferred over or used in conjunction with Basic Authentication which makes the conversation with the web server entirely encrypted. The best part is that nobody can even guess from the outside that Basic Auth is taking place.

#### DIGEST Authentication

This authentication method makes use of a hashing algorithms to encrypt the password (called **password hash**) entered by the user before sending it to the server. This, obviously, makes it much safer than the basic authentication method, in which the user’s password travels in plain text that can be easily read by whoever intercepts it.

**Read More:** [Generate Encrypted Passwords](http://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/)

There are many such hashing algorithms in java also, which can prove really effective for password security such as MD5, SHA, BCrypt, SCrypt and PBKDF2WithHmacSHA1 algorithms.

Please remember that once this password hash is generated and stored in database, you can not convert it back to original password. Each time user login into application, you have to regenerate password hash again, and match with hash stored in database. So, if user forgot his/her password, you will have to send him a temporary password and ask him to change it with his new password. Well, it’s common trend now-a-days.

#### CLIENT CERT Authentication

This is a mechanism in which a trust agreement is established between the server and the client through certificates. They must be signed by an agency established to ensure that the certificate presented for authentication is legitimate, which is known as CA.

Using this technique, when the client attempts to access a protected resource, instead of providing a username or password, it presents the certificate to the server. The certificate contains the user information for authentication including security credentials, besides a unique private-public key pair. The server then determines if the user is legitimate through the CA. Additionally, it must verify whether the user has access to the resource. This mechanism must use HTTPS as the communication protocol as we don’t have a secure channel to prevent anyone from stealing the client’s identity.

You can find a complete tutorial for [generating security certificate in official oracle docs](https://docs.oracle.com/javase/tutorial/security/apisign/index.html).

#### Using API Keys

If you have ever developed applications which interact other with other applications over cloud e.g. facebook integration or twitter authentication etc. then you have already used this. They require you to provide API key and API secret to rightly identify you. These API key and secret are some random encoded string which is impossible to guess.

To understand how it works, let’s assume you are using a Flickr (photo sharing application) and want to post some of your photos using it’s REST API. You build the request as documented in Flickr docs, then send it.

Then, when receiving the request, Flickr authenticates the user by reading the information from the API key with the secret key that belongs to the user. Once these validations are successful, the server delivers the response to the client. Thus, we obtain a response with all the photos that have been recently posted within Flickr.

As you’ll notice, this way, you can easily create applications using the provider’s API. Also, the provider will allow you to authenticate, access public information.

If someone starts disrespecting agreements e.g. sending junk traffic or any policy violation, the provider withdraws the API key and prevent the abusive use of its APIs.

## RESTful Web Services Security Implementations

Apart from above concepts, you will usually need to secure your RESTful APIs in your company using below methods.

#### Using SecurityContext

The javax.ws.rs.core.SecurityContext interface provides access to security-related information for a request and is very similar to javax.servlet.http.HttpServletRequest.

You access the SecurityContext by injecting an instance into a class field, setter method, or method parameter using the javax.ws.rs.core.Context annotation e.g. in below code sc.isUserInRole() is used to check authorization for user.

|  |
| --- |
| @GET  @Produces("text/plain;charset=UTF-8")  @Path("/hello")  public String sayHello(@Context SecurityContext sc) {          if (sc.isUserInRole("admin"))              return "Hello World!";          throw new SecurityException("User is unauthorized.");  } |

#### Method level Authorization Using Annotations

This technique is widely used in enterprise application and used to verify roles and responsibities of an authenticated used – for any certain operation. JAX-RS provides below annotations for this purpose.

* + [@PermitAll](http://docs.oracle.com/javaee/6/api/javax/annotation/security/PermitAll.html)
  + [@DenyAll](http://docs.oracle.com/javaee/6/api/javax/annotation/security/DenyAll.html)
  + [@RolesAllowed](http://docs.oracle.com/javaee/6/api/javax/annotation/security/RolesAllowed.html)

An example ue of annotation can be:

|  |
| --- |
| @RolesAllowed("ADMIN")  @PUT  @Path("/users/{id}")  public Response updateUserById(@PathParam("id") int id)  {      //Update the User resource      UserDatabase.updateUser(id);      return Response.status(200).build();  } |

**Read More :** [RESTEasy Authentication and Authorization Tutorial](http://howtodoinjava.com/resteasy/jax-rs-resteasy-basic-authentication-and-authorization-tutorial/)

## Best Practices

Let’s note down some important points while designing security for your RESTful web services.

1. Use only HTTPS protocol so that your whole communocation is always encrypted.
2. Never send auth credentials or API keys as query param. They appear in URL and can be logged or tracked easily.
3. Use hardest encryption level always. It will help in having more confidence.
4. For resources exposed by RESTful web services, it’s important to make sure any PUT, POST, and DELETE request is protected from Cross Site Request Forgery.
5. Always validate the input data asap it is recieved in server method. Use only primitive data as input parameter as much as possible.
6. Rely on framework provided validation features as they are tested by large community already.

Let me know your thoughts and experiences on securing RESTful web services in your organization.CORE JAVA

# [AtomicInteger in Java](http://www.instanceofjava.com/2015/12/atomic-integer-example-java-interview.html)

* Java.util.concurrent.atomic package provides very useful classes that support lock free and thread safe programming.
* The main use of this class is an int value that may be updated automatically.
* AtomicInteger has some useful methods. Before that let’s see the some points about this class.
* Commonly we will use this AtomicInteger to handle the counter that is accessible by different threads simultaneously.

**Why You Need String Constant Pool? :**

String objects in java are stored in two places in memory. One is **String Constant Pool** and another one is **Heap Memory**. String objects created using string **literals** are stored in String Constant Pool where as string objects created using new operator are stored in **heap memory**.

String objects are most used objects in the development of any kind of applications. Therefore, there has to be a special arrangement to store these objects. String Constant Pool is one such special arrangement. In string constant pool, there will be no two objects with the same content. Heap memory can have any number of objects with same content.

Just imagine creating 1000 string objects with same content in heap memory and one string object with that content in String Constant Pool. Which one saves the memory? Which one will save the time?. Which one will be accessed faster?. It is, of course, String Constant Pool. That’s why you need String Constant Pool.

**What Is String Intern? :**

**String intern** or simply **intern** refers to string object in the String Constant Pool. **Interning** is the process of creating a string object in String Constant Pool which will be exact copy of string object in heap memory.

**What is the use of interning the string?**

**To Save The memory Space:**

Using interned string, you can save the memory space. If you are using lots of string objects with same content in your code, than it is better to create an intern of that string in the pool. Use that intern string whenever you need it instead of creating a new object in the heap. It saves the memory space.

**For Faster Comparison :**

Assume that there are two string objects s1 and s2 in heap memory and you need to perform comparison of these two objects more often in your code. Then using s1.intern() == s2.intern() will be more fast then s1.equals(s2). Because, equals() method performs character by character comparison where as “==” operator just compares references of objects**.**

**STRING:**

## String represents sequence of characters enclosed within the double quotes.  “abc”, “JAVA”, “123”, “A” are some examples of strings. In many languages, strings are treated as character arrays. But In java, strings are treated as objects. To create and manipulate the strings, Java provides three classes.

**1) java.lang.String                  (From JDK 1.0)**

**2) java.lang.StringBuffer            (From JDK 1.5)**

**3) java.lang.StringBuilder           (From JDK 1.5)**

1) All these three classes are members of **java.lang** package and they are final classes. That means you can’t create subclasses to these three classes.

2) All three classes implement **Serializable** and **CharSequence** interface.

3) **java.lang.String** objects are **immutable** in java. That is, once you create String objects, you can’t modify them. Whenever you try to modify the existing String object, a new String object is created with modifications. Existing object is not at all altered. Where as **java.lang.StringBuffer** and **java.lang.StringBuilder** objects are **mutable**. That means, you can perform modifications to existing objects.

4) Only **String** and **StringBuffer** objects are thread safe. **StringBuilder** objects are not thread safe. So whenever you want immutable and thread safe string objects, use *java.lang.String* class and whenever you want mutable as well as thread safe string objects then use *java.lang.StringBuffer* class.

5) In all three classes, **toString()** method is overrided. So. whenever you use reference variables of these three types, they will return contents of the objects not physical address of the objects.

## 6) hashCode() and equals() methods are overrided only in *java.lang.String* class but not in *java.lang.StringBuffer* and*java.lang.StringBuilder* classes.

7) There is no **reverse()** and **delete()** methods in String class. But, StringBuffer and StringBuilder have reverse() and delete() methods.

8) In case of String class, you can create the objects without **new** operator. But in case of StringBuffer and StringBuilder class, you have to use new operator to create the objects.

**Are string objects created using new operator also immutable?**

The answer is yes. String objects created using new operator are also immutable although they are stored in the heap memory. This can be also proved with help of an example.

|  |  |
| --- | --- |
|  | public class StringExamples  {      public static void main(String[] args)      {          String s1 = new String("JAVA");            System.out.println(s1);         //Output : JAVA            s1.concat("J2EE");            System.out.println(s1);         //Output : JAVA      }  } |

In this example, a string object is created with “JAVA” as it’s content using new operator and it’s reference is assigned to s1. I have tried to change the contents of this object using concat() method. But, these changes are not reflected in the object as seen in Line 11. Even after the concatenation, content of the object is same as before. This is because the strings are immutable. Once I tried to concatenate “J2EE” to an existing string “JAVA”, a new string object is created with “JAVAJ2EE” as it’s content. But we don’t have reference to that object in this program.

## Conclusion:

Immutability is the fundamental property of string objects. In whatever way you create the string objects, either using string literals or using new operator, they are immutable.

## Process :

Process is an executing instance of an application. For example, when you double click MS Word icon in your computer, you start a process that will run this MS word application. Processes are heavy weight operations that they require their own separate memory address in operating system. Because of the processes are stored in separate memory, communication between processes (Inter Process Communication) takes time. Context switching from one process to another process is also expensive.

## Thread :

Thread is a smallest executable unit of a process. Thread has it’s own path of execution in a process. For example, when you start MS word, operating system creates a process and start the execution of a primary thread of that process. A process can have multiple threads. Threads of the same process share the memory address of that process. i.e threads are stored inside the memory of a process. As the threads are stored in the same memory space, communication between threads (Inter Thread Communication) is fast. Context switching from one thread to another thread is also less expensive.

**There are two types of Threads in java**.

1**) User Thread**

**2) Daemon Thread**

**1) User Thread:**

User threads are threads which are created by the application or user. They are high priority threads. JVM (Java Virtual Machine) will not exit until all user threads finish their execution. JVM wait for these threads to finish their task. These threads are foreground threads.

2) **Daemon Thread:**

Daemon threads are threads which are mostly created by the JVM. These threads always run in background. These threads are used to perform some background tasks like garbage collection and house-keeping tasks. These threads are less priority threads. JVM will not wait for these threads to finish their execution. JVM will exit as soon as all user threads finish their execution. JVM doesn’t wait for daemon threads to finish their task.

[**How to identify a thread in java**?](http://javaconceptoftheday.com/how-to-identify-a-thread-in-java/)

In a multithreaded application, It is very important to know which thread is currently executing it’s task. But the question is, **How to identify a thread?**. The answer which effortlessly comes to our mind is “through it’s name”. Of course, you can identify a thread by it’s name.  But, more than one threads can have the same name. This makes identifying a thread more difficult. There is a solution for this problem from JDK 1.5 onward. JVM assigns one unique long number for every thread created. This remains unchanged for the whole life term of a thread. This number can be used to identify a thread.

From JDK 1.5 onward, One more method added to **java.lang.Thread** class. That is **getID () method**. This method returns the unique long number associated with a thread. That can be used as an identifier of a thread. Below is the method signature of getID () method.

**What is Memory Leak in Java?**

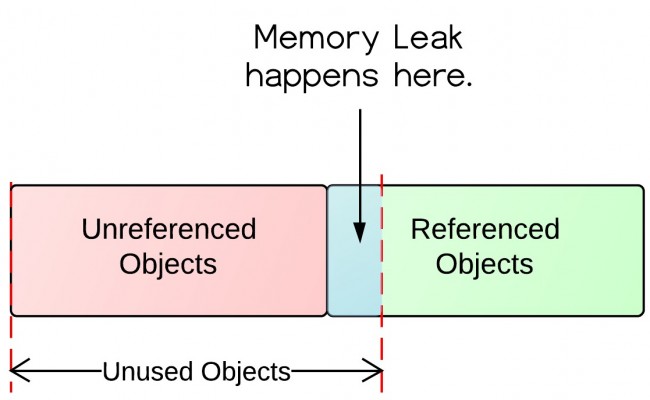
One of the most significant advantages of Java is its memory management. You simply create objects and Java Garbage Collector takes care of allocating and freeing memory. However, the situation is not as simple as that, because memory leaks frequently occur in Java applications.

This process is called garbage collection and the corresponding piece of JVM is called a Garbage Collector or GC. ... Simplifying a bit, we can say that a memory leak in Java is a situation where some objects are not used by the application any more, but GC fails to recognize them as unused.

 What is Memory Leak?

Definition of *Memory Leak*: objects are no longer being used by the application, but Garbage Collector can not remove them because they are being referenced.

To understand this definition, we need to understand objects status in memory. The following diagram illustrates what is unused and what is unreferenced.

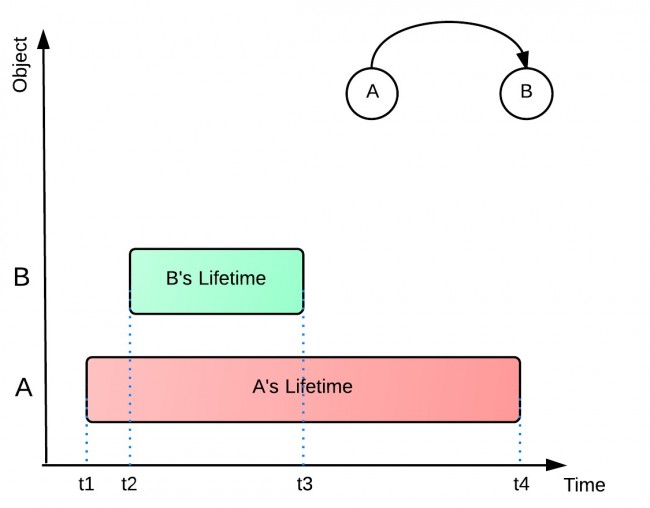


From the diagram, there are *referenced objects* and *unreferenced objects*. Unreferenced objects will be garbage collected, while referenced objects will not be garbage collected. Unreferenced objects are surely unused, because no other objects refer to it. However, unused objects are not all unreferenced. Some of them are being referenced! That's where the memory leaks come from.

Why Memory Leaks Happen?

Let's take a look at the following example and see why memory leaks happen. In the example below, object A refers to object B. A's lifetime (t1 - t4) is much longer than B's (t2 - t3). When B is no longer being used in the application, A still holds a reference to it. In this way, Garbage Collector can not remove B from memory. This would possibly cause out of memory problem, because if A does the same thing for more objects, then there would be a lot of objects that are uncollected and consume memory space.

It is also possible that B hold a bunch of references of other objects. Those objects referenced by B will not get collected either. All those unused objects will consume precious memory space.



**Why substring() method in JDK 6 can cause memory leaks?**

The substring(int beginIndex, int endIndex) method in JDK 6 and JDK 7 are different. Knowing the difference can help you better use them. For simplicity reasons, in the following substring() represent the substring(int beginIndex, int endIndex) method.

**1. What substring() does?**

The substring(int beginIndex, int endIndex) method returns a string that starts with beginIndex and ends with endIndex-1.

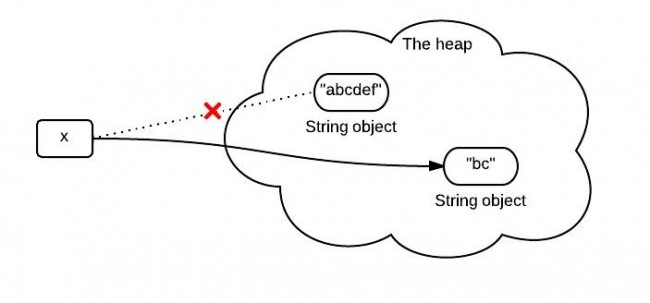
|  |
| --- |
| String x = "abcdef";  x = x.substring(1,3);  System.out.println(x); |

Output:

bc

**2. What happens when substring() is called?**

You may know that because x is immutable, when x is assigned with the result of x.substring(1,3), it points to a totally new string like the following:

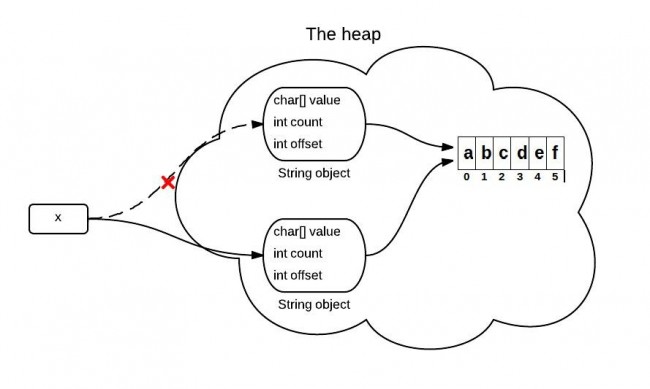


However, this diagram is not exactly right or it represents what really happens in the heap. What really happens when substring() is called is different between JDK 6 and JDK 7.

**3. substring() in JDK 6**

String is supported by a char array. In JDK 6, the String class contains 3 fields: char value[], int offset, int count. They are used to store real character array, the first index of the array, the number of characters in the String.

When the substring() method is called, it creates a new string, but the string's value still points to the same array in the heap. The difference between the two Strings is their count and offset values.



The following code is simplified and only contains the key point for explain this problem.

|  |
| --- |
| *//JDK 6*  String(**int** offset, **int** count, **char** value[]) {  **this**.value = value;  **this**.offset = offset;  **this**.count = count;  }    **public** String substring(**int** beginIndex, **int** endIndex) {  *//check boundary*  **return** **new** String(offset + beginIndex, endIndex - beginIndex, value);  } |

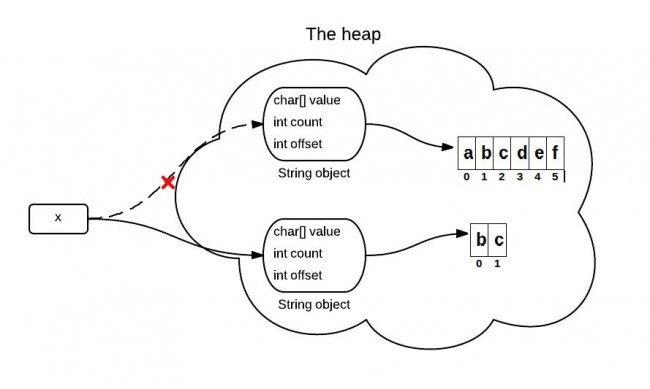
**4. A problem caused by substring() in JDK 6**

If you have a VERY long string, but you only need a small part each time by using substring(). This will cause a performance problem since you need only a small part, you keep the whole thing. For JDK 6, the solution is using the following, which will make it point to a real substring:

|  |
| --- |
| x = x.substring(x, y) + "" |

**5. substring() in JDK 7**

This is improved in JDK 7. In JDK 7, the substring() method actually create a new array in the heap.



|  |
| --- |
| *//JDK 7*  **public** String(**char** value[], **int** offset, **int** count) {  *//check boundary*  **this**.value = Arrays.copyOfRange(value, offset, offset + count);  }    **public** String substring(**int** beginIndex, **int** endIndex) {  *//check boundary*  **int** subLen = endIndex - beginIndex;  **return** **new** String(value, beginIndex, subLen);  } |

**How to Prevent Memory Leaks?**

The following are some quick hands-on tips for preventing memory leaks.

Pay attention to Collection classes, such as HashMap, ArrayList, etc., as they are common places to find memory leaks. When they are declared static, their life time is the same as the life time of the application.

Pay attention to event listeners and callbacks. A memory leak may occur if a listener is registered but not unregistered when the class is not being used any longer.

"If a class manages its own memory, the programer should be alert for memory leaks."[1] Often times member variables of an object that point to other objects need to be null out.

**What is a Thread dump?**

A thread dump is a snapshot of the state of all threads that are part of the process. The state of each thread is presented with a so called stack trace, which shows the contents of a thread's stack. Some of the threads belong to the Java application you are running, while others are JVM internal threads.

**[How will you take thread dump in Java? How will you analyze Thread dump?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "how-will-you-take-thread-dump-in-java--how-will-you-analyze-thread-dump-" \o "How will you take thread dump in Java? How will you analyze Thread dump?)**

A Thread Dump is a complete list of active threads. A java thread dump is a way of finding out what each thread in the JVM is doing at a particular point of time. This is especially useful when your java application seems to have some performance issues. Thread dump will help you to find out which thread is causing this. There are several ways to take thread dumps from a JVM. It is highly recommended to take more than 1 thread dump and analyze the results based on it. Follow below steps to take thread dump of a java processs.

Step 1   
  
On UNIX, Linux and Mac OSX Environment run below command:   
  
 **ps -el | grep java**   
  
On Windows:   
  
Press Ctrl+Shift+Esc to open the task manager and find the PID of the java process 

Step 2:   
  
Use jstack command to print the Java stack traces for a given Java process PID   
  
 **jstack [PID]**   
  
More details of jstack command can be found here: [JSTACK Command Manual](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstack.html)

[How can I trace whether the application has a thread leak?](http://www.fromdev.com/2008/05/java-threading-questions.html#how-can-i-trace-whether-the-application-has-a-thread-leak-)

* If an application has thread leak then with time it will have too many unused threads. Try to find out what type of threads is leaking out. This can be done using following ways, Give unique and descriptive names to the threads created in application. –
* Add log entry in all thread at various entry and exit points in threads. Change debugging config levels (debug, info, error etc) and analyze log messages.
* When you find the class that is leaking out threads check how new threads are instantiated and how they're closed.
* Make sure the thread is guaranteed to close properly by doing following - Handling all Exceptions properly.
* Releasing all resources (e.g. connections, files etc) before it closes.

**How to Avoid Thread Interference or How to Achieve Thread Safeness?**

By declaring the method as synchronized.

By declaring the variables as final.

By declaring the variable as volatile.

By creating the immutable objects.

By using Atomic operations.

By restricting the access to same object by multiple threads.

**[What is thread pool? Why should we use thread pools?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-thread-pool--why-should-we-use-thread-pools-" \o "What is thread pool? Why should we use thread pools?)**

A thread pool is a collection of threads on which task can be scheduled. Instead of creating a new thread for each task, you can have one of the threads from the thread pool pulled out of the pool and assigned to the task. When the thread is finished with the task, it adds itself back to the pool and waits for another assignment. One common type of thread pool is the fixed thread pool. This type of pool always has a specified number of threads running; if a thread is somehow terminated while it is still in use, it is automatically replaced with a new thread. Below are key reasons to use a Thread Pool

Using thread pools minimizes the JVM overhead due to thread creation. Thread objects use a significant amount of memory, and in a large-scale application, allocating and de-allocating many thread objects creates a significant memory management overhead.

You have control over the maximum number of tasks that are being processed in parallel (= number of threads in the pool).

Most of the executor implementations in java.util.concurrent use thread pools, which consist of worker threads. This kind of thread exists separately from the Runnable and Callable tasks it executes and is often used to execute multiple tasks.

**[Can two threads call two different synchronized instance methods of an Object?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "can-two-threads-call-two-different-synchronized-instance-methods-of-an-object-" \o "Can two threads call two different synchronized instance methods of an Object?)**

No. If a object has synchronized instance methods then the Object itself is used a lock object for controlling the synchronization. Therefore all other instance methods need to wait until previous method call is completed. See the below sample code which demonstrate it very clearly. The Class Common has 2 methods called **synchronizedMetohd1**() and **synchronizedMethod2()** MyThread class is calling both the methods.

[**Can we synchronize the run method? If yes then what will be the behavior?**](http://www.fromdev.com/2008/05/java-threading-questions.html#can-we-synchronize-the-run-method--if-yes-then-what-will-be-the-behavior-)

Yes, the run method of a runnable class can be synchronized. If you make run method synchronized then the lock on runnable object will be occupied before executing the run method. In case we start multiple threads using the same runnable object in the constructor of the Thread then it would work. But until the 1st thread ends the 2nd thread cannot start and until the 2nd thread ends the next cannot start as all the threads depend on lock on same object.

**What is Volatile variable in Java?**

volatile variable in Java is a special variable which is used to signal threads, a compiler that this particular variables value are going to be updated by multiple threads inside Java application. By making a variable volatile using the volatile keyword in Java, application programmer ensures that its value should always be read from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html)and thread should not use cached value of that variable from their own stack. With the introduction of Java memory model from Java 5 onwards along with introduction of [**CountDownLatch**](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html), [**CyclicBarrier**](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html), [**Semaphore**](http://javarevisited.blogspot.sg/2012/05/counting-semaphore-example-in-java-5.html)and [**ConcurrentHashMap**](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html), **volatile** variable also guarantees "happens-before" relationship, which means not only another thread has visibility of latest value of volatile variable but also all the variable is seen by the thread which has updated value of volatile variable before these threads sees it. What is volatile variable and when to use it is always a [popular Java threading question](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html).  
  
The volatile keyword can only be applied to a variable, it cannot be applied to class or method. Using volatile keyword along with class and method is a compiler error.

A volatile is also referred as modifier in Java.

**When to use Volatile variable in Java?**

1) Any variable which is shared between multiple threads should be made variable, in order to ensure that all thread must see the latest value of the volatile variable.  
  
2) A signal to compiler and JIT to ensure that compiler does not change ordering or volatile variable and moves them out of synchronized context.  
  
3) You want to save the cost of synchronization as volatile variables are less expensive than synchronization.  
  
**What are the different ways to create an object in Java?**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| There are many ways to create an object in java. They are:  By **new** keyword  By **newInstance**() method  By **clone**() method  By **factory** method etc.  Java String intern() method  A pool of strings, initially empty, is maintained privately by the class String. When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.  public class InternExample{  public static void main(String args[]){  String s1=new String("hello");  String s2="hello";  String s3=s1.intern();//returns string from pool, now it will be same as s2  System.out.println(s1==s2);//false because reference is different  System.out.println(s2==s3);//true because reference is same  }}  **Difference between method overloading and method overriding in java?**   |  |  | | --- | --- | | Method Overloading | Method Overriding | | Definition | When a class has more than one method with same name but with different arguments, then we call it as method overloading. | When a super class method is modified in the sub class, then we call this as method overriding. | | Method Signature | Overloaded methods must have different method signatures.  That means they should differ at least in any one of these three things – Number of arguments, Types of arguments and order of arguments. But, they must have same name. | Overridden methods must have same method signature. I.e. you must not change the method name, types of arguments, number of arguments and order of arguments while overriding a super class method. | | Return Types | Overloaded methods can have same or different return types. | The return type of the overridden method must be compatible with that of super class method. That means if super class method has primitive type as its return type, then it must be overridden with same return type. If super class method has derived type as its return type then it must be overridden with same type or its sub class type. | | Visibility(private, public, protected and default) | Overloaded methods can have same visibility or different visibility. | While overriding a super class method either you can keep the same visibility or you can increase the visibility. But you can’t reduce it. | | Static Context | Overloaded methods can be static or not static. It does not affect the method overloading. | You can’t override a static method. | | Binding | Binding between method call and method definition happens at compile time (Static Binding). | Binding between method call and method definition happens at run time (Dynamic Binding). | | Polymorphism | It shows static polymorphism. | It shows dynamic polymorphism. | | Private methods | Private methods can be overloaded. | Private methods can’t be overridden. | | Final Methods | Final methods can be overloaded. | Final methods can’t be overridden. | | Class Requirement | For method overloading, only one class is required. I.e. Method overloading happens within a class. | For method overriding, two classes are required – super class and sub class. That means method overriding happens between two classes. |   **What are the rules to be followed while overriding a method?**  There are 5 main rules you should kept in mind while overriding a method. They are,  a) Name of the method must be same as that of super class method.  b) Return type of overridden method must be compatible with the method being overridden. i.e if a method has primitive type as it’s return type then it must be overridden with primitive type only and if a method has derived type as it’s return type then it must be overridden with same type or it’s sub class types.  c) You must not reduce the visibility of a method while overriding.  d) You must not change parameter list of a method while overriding.  e) You cannot increase the scope of exceptions while overriding a method with throws clause.  **Abstraction :**  Yes, In the computer science, Abstraction is used to separate ideas from their implementation. Abstraction in java is used to define only ideas in one class so that the idea can be implemented by its sub classes according to their requirements.   1. Abstract classes and abstract methods are declared using ‘**abstract**‘ keyword. We can’t create objects to those classes which are declared as abstract. But, we can create objects to sub classes of abstract class, provided they must implement abstract methods. 2. The methods which are not implemented or which don’t have definitions must be declared with ‘abstract’ keyword and the class which contains it must be also declared as abstract. 3. It is not compulsory that abstract class must have abstract methods. It may or may not have abstract methods. But the class which has at least one abstract method must be declared as abstract. 4. You can’t create objects to abstract class even though it does not contain any abstract methods. 5. Abstract Class can be a combination of concrete and abstract methods. 6. Any class extending an abstract class must implement all abstract methods. If it does not implement, it must be declared as abstract. 7. Inside abstract class, we can keep any number of constructors. If you are not keeping any constructors, then compiler will keep default constructor. 8. Abstract methods cannot be private. Because, abstract methods must be implemented somehow in the sub classes. If you declare them as private, then you can’t use them outside the class. 9. Constructors and fields cannot be declared as abstract. 10. Abstract methods cannot be static.  [Interfaces In Java](http://javaconceptoftheday.com/interfaces-in-java/) Interfaces in java are very much similar to abstract classes but interfaces contain only abstract methods (you can refer to them as only ideas). Abstract classes may contain both abstract methods as well as concrete methods. But interfaces must contain only abstract methods. Concrete methods are not allowed in interfaces. Therefore, Interfaces show 100% abstractness.  Interfaces are declared with keyword ‘**interface**‘and interfaces are implemented by the class using ‘**implements**‘ keyword.  Interfaces should contain only abstract methods. Interfaces should not contain a single concrete method.  Interface can have two types of members.  **1) Fields     2) Abstract Methods.**  By default, Every field of an interface is public, static and final (we will discuss about final keyword Later). You can’t use any other modifiers other than these three for a field of an interface. |
| You can’t change the value of a field once they are initialized. Because they are static and final. Therefore, sometimes fields are called as **Constants**. (We will discuss this feature in detail while covering ‘final’ keyword)  By default, All methods of an interface are public and abstract.  Like classes, for every interface .class file will be generated after compilation  While implementing any interface methods inside a class, that method must be declared as public. Because, according to [method overriding](http://javaconceptoftheday.com/method-overriding-java/) rule, you can’t reduce visibility of super class method. By default, every member of an interface is public and while implementing you should not reduce this visibility.  By default, Interface itself is not public but by default interface itself is abstract like below,  [SIB](http://javaconceptoftheday.com/static-members-java/) – Static Initialization Block and [IIB](http://javaconceptoftheday.com/instance-initialization-block-in-java/) – Instance Initialization Block are not allowed in interfaces.  As we all know that, any class in java cannot extend more than one class. But class can implement more than one interfaces. This is how **multiple inheritance** is implemented in java. |

# Differences Between Static Binding And Dynamic Binding In Java ?

## *The above findings can be summarized like below.*

|  |  |
| --- | --- |
| *Static Binding* | *Dynamic Binding* |
| *It is a binding that happens at compile time.* | *It is a binding that happens at run time.* |
| *Actual object is not used for binding.* | *Actual object is used for binding.* |
| *It is also called early binding because binding happens during compilation.* | *It is also called late binding because binding happens at run time.* |
| *Method overloading is the best example of static binding.* | *Method overriding is the best example of dynamic binding.* |
| *Private, static and final methods show static binding. Because, they can not be overridden.* | *Other than private, static and final methods show dynamic binding. Because, they can be overridden.* |

**Difference between String and StringBuffer?**

*There are many differences between* ***String*** *and* ***StringBuffer****. A list of differences between String and StringBuffer are given below:*

|  |  |  |
| --- | --- | --- |
| No. | String | StringBuffer |
| *1)* | *String class is immutable.* | *StringBuffer class is mutable.* |
| *2)* | *String is slow and consumes more memory when you concat too many strings because every time it creates new instance.* | *StringBuffer is fast and consumes less memory when you cancat strings.* |
| *3)* | *String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.* | *StringBuffer class doesn't override the equals() method of Object class.* |

Difference between StringBuffer and StringBuilder?

*There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:*

|  |  |  |
| --- | --- | --- |
| *No.* | *StringBuffer* | *StringBuilder* |
| *1)* | *StringBuffer is*synchronized*i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.* | *StringBuilder is*non-synchronized*i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.* |
| *2)* | *StringBuffer is*less efficient*than StringBuilder.* | *StringBuilder is*more efficient*than StringBuffer.* |

## Does an interface extend Object class in java.?

You may have come across this question while reading about interfaces in java. You may also know that only classes in java are inherited from **java.lang.Object** class. Interfaces in java don’t inherit from Object class. They don’t have default parent like classes in java. But, following two cases may surprise you.

If an interface does not extend Object class, then why we can call methods of Object class on interface variable like below.

**If an interface does not extend Object class, then why the methods of Object class are visible in interface?**

This is because, for every public method in Object class, there is an implicit abstract and public method declared in every interface which does not have direct super interfaces. This is the standard Java Language Specification which states like this,

“If an interface has no direct super interfaces, then the interface implicitly declares a public abstract member method m with signature s, return type r, and throws clause corresponding to each public instance method m with signature s, return type r, and throws clause t declared in Object, unless a method with the same signature, same return type, and a compatible throws clause is explicitly declared by the interface.”

**NoClassDefFoundError vs ClassNotFoundException?**

In Java, both **ClassNotFoundException** and **NoClassDefFoundError** occur when a particular class is not found at run time. But, they occur at different scenarios. ClassNotFoundException is an exception which occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath. On the other hand, NoClassDefFoundError is an error which occurs when a particular class is present at compile time but it was missing at run time.

## Difference Between ClassNotFoundException Vs NoClassDefFoundError In Java :

|  |  |
| --- | --- |
| ClassNotFoundException | NoClassDefFoundError |
| It is an exception. It is of type java.lang.Exception. | It is an error. It is of type java.lang.Error. |
| It occurs when an application tries to load a class at run time which is not updated in the classpath. | It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at run time. |
| It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass(). | It is thrown by the Java Runtime System. |
| It occurs when classpath is not updated with required JAR files. | It occurs when required class definition is missing at run time. |

**Difference between Exception and Error?**

Both **java.lang.Error** and **java.lang.Exception** classes are sub classes of **java.lang.Throwable** class**,** but there exist some significant differences between them. **java.lang.Error** class represents the errors which are mainly caused by the environment in which application is running. For example, **OutOfMemoryError** occurs when JVM runs out of memory or **StackOverflowError** occurs when stack overflows.

Where as **java.lang.Exception** class represents the exceptions which are mainly caused by the application itself. For example, **NullPointerException** occurs when an application tries to access null object or **ClassCastException**occurs when an application tries to cast incompatible class types. In this article, we will discuss the differences between Error and Exception in java.

`

|  |  |
| --- | --- |
| **Errors** | **Exceptions** |
| Errors in java are of type java.lang.Error. | Exceptions in java are of type java.lang.Exception. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |
| Errors happen at run time. They will not be known to compiler. | Checked exceptions are known to compiler where as unchecked exceptions are not known to compiler because they occur at run time. |
| It is impossible to recover from errors. | You can recover from exceptions by handling them through try-catch blocks. |
| Errors are mostly caused by the environment in which application is running. | Exceptions are mainly caused by the application itself. |
| Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, ClassCastException, NullPointerException |

**What is System.out in Java?**

In System.out, out is an instance of PrintStream. It is a static member variable in System class. This is called standard output stream, connected to console.

**Purpose of garbage collection.**

The garbage collection process is to identify the objects which are no longer referenced or needed by a program so that their resources can be reclaimed and reused. These identified objects will be discarded.

**Java Heap Space**

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space. **Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference**. Any object created in the heap space has global access and can be referenced from anywhere of the application.

**Java Stack Memory**

Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method. Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as method ends, the block becomes unused and become available for next method. Stack memory size is very less compared to Heap memory.

**Difference between Java Heap Space and Stack Memory?**

Based on the above explanations, we can easily conclude following differences between Heap and Stack memory.

Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.

Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into **Young-Generation, Old-Generation** etc, more details at [Java Garbage Collection](http://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java).

Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.

We can use -Xms and -Xmx JVM option to define the startup size and maximum size of heap memory.

We can use -Xss to define the stack memory size.

When stack memory is full, Java runtime throws java**.lang.StackOverFlowError** whereas if heap memory is full, it throws **java.lang.OutOfMemoryError**: Java Heap Space error.

Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

**Abstraction**:

Abstraction is "To represent the essential feature without representing the back ground details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or object by providing relevant information.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

**Encapsulation**:

Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.

    class Bag

    {

        book;

        pen;

        ReadBook();

    }

Encapsulation means hiding the internal details of an object, i.e. how an object does something.

Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.

Encapsulation is a technique used to protect the information in an object from the other object.

Hide the data for security such as making the variables as private, and expose the property to access the private data which would be public.

So, when you access the property you can validate the data and set it.

**Abstraction/Encapsulation**

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| 1. Abstraction solves the problem in the design level. | 1. Encapsulation solves the problem in the implementation level. |
| 2. Abstraction is used for hiding the unwanted data and giving relevant data. | 2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world. |
| 3. Abstraction lets you focus on what the object does instead of how it does it | 3. Encapsulation means hiding the internal details or mechanics of how an object does something. |
| 4. Abstraction- Outer layout, used in terms of design.  For Example:-   Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number. | 4. Encapsulation- Inner layout, used in terms of implementation.  For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits. |

**Q) What is difference between polymorphism and inheritance?**

* Inheritance defines parent-child relationship between two classes, polymorphism take advantage of that relationship to add dynamic behavior in your code.
* Inheritance helps in code reusability by allowing child class to inherit behavior from the parent class. On the other hand Polymorphism allows Child to redefine already defined behavior inside parent class. Without Polymorphism it's not possible for a Child to execute its own behavior while represented by a Parent reference variable, but with Polymorphism he can do that.
* **Java doesn't allow multiple inheritances of classes, but allows**[**multiple inheritance of Interface**](http://java-questions.com/keyConcepts-interview-questions.html#diamond-problem)**, which is actually required to implement Polymorphism**. For example a Class can be Runnable, Comparator and Serializable at same time, because all three are interfaces. This makes them to pass around in code e.g. you can pass instance of this class to a method which accepts Serializable, or to Collections.sort() which accepts a Comparator.
* Both Polymorphism and Inheritance allow Object oriented programs to evolve. For example, by using Inheritance you can define new user types in an Authentication System and by using Polymorphism you can take advantage of already written authentication code. Since, Inheritance guarantees minimum base class behavior, a method depending upon super class or super interface can still accept object of base class and can authenticate it.

**What is Encapsulation?**

The encapsulation is achieved by combining the methods and attribute into a class. The class acts like a container encapsulating the properties. The users are exposed mainly public methods. The idea behind is to hide how thinigs work and just exposing the requests a user can do.

**What is Association?**

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

**What is Aggregation?**

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

**What is Composition ?**

Composition is again specialize form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does 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 cannot belong to two different house if we delete the house, room will automatically delete.

**Shallow Copy Vs Deep Copy In Java?**

Below is the list of differences between shallow copy and deep copy in java.

|  |  |
| --- | --- |
| Shallow Copy | Deep Copy |
| Cloned Object and original object are not 100% disjoint. | Cloned Object and original object are 100% disjoint. |
| Any changes made to cloned object will be reflected in original object or vice versa. | Any changes made to cloned object will not be reflected in original object or vice versa. |
| Default version of clone method creates the shallow copy of an object. | To create the deep copy of an object, you have to override clone method. |
| Shallow copy is preferred if an object has only primitive fields. | Deep copy is preferred if an object has references to other objects as fields. |
| Shallow copy is fast and also less expensive. | Deep copy is slow and very expensive. |

**Why is String immutable in Java?**

1. String Pool  
 When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object. If string is not immutable, changing the string with one reference will lead to the wrong value for the other references.  
  
2. To Cache its Hashcode  
 If string is not immutable, One can change its hashcode and hence not fit to be cached.  
3. Security  
 String is widely used as parameter for many java classes, e.g. network connection, opening files, etc. Making it mutable might possess threats due to interception by the other code segment.

**[What is immutable object in Java? Can you change values of a immutable object?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-immutable-object-in-java--can-you-change-values-of-a-immutable-object-" \o "What is immutable object in Java? Can you change values of a immutable object?)**

A Java object is considered immutable when its state cannot change after it is created. Use of immutable objects is widely accepted as a sound strategy for creating simple, reliable code. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state. s

**java.lang.String and java.lang.Integer classes are the Examples of immutable objects from the Java Development Kit.** Immutable objects simplify your program due to following characteristics:

Immutable objects are simple to use test and construct.

Immutable objects are automatically thread-safe.

Immutable objects do not require a copy constructor.

Immutable objects do not require an implementation of clone.

Immutable objects allow hash Code to use lazy initialization, and to cache its return value.

Immutable objects do not need to be copied defensively when used as a field.

Immutable objects are good Map keys and Set elements (Since state of these objects must not change while stored in a collection).

Immutable objects have their class invariant established once upon construction, and it never needs to be checked again.

Immutable objects always have "failure atomicity" (a term used by Joshua Bloch): if an immutable object throws an exception, it's never left in an undesirable or indeterminate state.

**[How to create a immutable object in Java? Does all property of immutable object needs to be final?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "how-to-create-a-immutable-object-in-java--does-all-property-of-immutable-object-needs-to-be-final-" \o "How to create a immutable object in Java? Does all property of immutable object needs to be final?)**

To create a object immutable You need to make the class final and all its member final so that once objects gets created no one can modify its state. You can achieve same functionality by making member as non final but private and not modifying them except in constructor. Also its NOT necessary to have all the properties final since you can achieve same functionality by making member as non final but private and not modifying them except in constructor.

[**What is difference between String, StringBuffer and StringBuilder? When to use them?**](http://www.fromdev.com/2012/02/java-interview-question-answer.html#what-is-difference-between-string--stringbuffer-and-stringbuilder--when-to-use-them-)

The main difference between the three most commonly used String classes as follows.

->StringBuffer and StringBuilder objects are mutable whereas String class objects are immutable.

->StringBuffer class implementation is synchronized while StringBuilder class is not synchronized.

Concatenation operator "+" is internally implemented by Java using either StringBuffer or StringBuilder.

Criteria to choose among String, StringBuffer and StringBuilder

-> If the Object value will not change in a scenario use String Class because a String object is immutable.

-> If the Object value can change and will only be modified from a single thread, use a StringBuilder because StringBuilder is unsynchronized(means faster).

-> If the Object value may change, and can be modified by multiple threads, use a StringBuffer because StringBuffer is thread safe(synchronized).

[**Why String class is final or immutable?**](http://www.fromdev.com/2012/02/java-interview-question-answer.html#why-string-class-is-final-or-immutable-)

It is very useful to have strings implemented as final or immutable objects. Below are some advantages of String Immutability in Java

Immutable objects are thread-safe.

Two threads can both work on an immutable object at the same time without any possibility of conflict.

**Security**: the system can pass on sensitive bits of read-only information without worrying that it will be altered

You can share duplicates by pointing them to a single instance.

You can create substrings without copying. You just create a pointer into an existing base String guaranteed never to change. Immutability is the secret that makes Java substring implementation very fast.

Immutable objects are good fit for becoming Hashtable keys. If you change the value of any object that is used as a hash table key without removing it and re-adding it you will lose the object mapping.

Since String is immutable, inside each String is a char[] exactly the correct length. Unlike a StringBuilder there is no need for padding to allow for growth.

If String were not final, you could create a subclass and have two strings that look alike when "seen as Strings", but that are actually different.

**How to create an immutable class?**

**To create an immutable class following steps should be followed:**

Create a final class.

Set the values of properties using constructor only.

Make the properties of the class final and private

Do not provide any setters for these properties.

If the instance fields include references to mutable objects, don't allow those objects to be changed:

Don't provide methods that modify the mutable objects.

Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

**What are the advantages of immutability?**

Immutable objects are automatically thread-safe; the overhead caused due to use of synchronization is avoided.

Once created the state of the immutable object cannot be changed so there is no possibility of them getting into an inconsistent state.

The references to the immutable objects can be easily **shared** or **cached** without having to copy or clone them as there state cannot be changed ever after construction.

The best use of the immutable objects is as the keys of a map.

**Prevent Cloning**

To implement cloning, we have to implement java.lang.Cloneable interface and override clone () method from Object class. It is a good idea to prevent cloning in a singleton class. To prevent cloning on singleton object, let us explicitly throw CloneNotSupportedException exception in clone() method.

**Immutable objects are automatically thread-safe –true/false?**

True. Since the state of the immutable objects can not be changed once they are created they are automatically synchronized/thread-safe.

**Use of HashCode and Equals()?**

Object class provides two methods hashcode() and equals() to represent the identity of an object. It is a common convention that if one method is overridden then other should also be implemented.

Before explaining why, lets see what is the contract between these two methods hold. As per the Java API documentation:

Whenever hashcode is invoked on the same object more than once during an execution of a Java application, the hashcode() method must consistently return the same integer, provided no information used in equals() comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

If two objects are equal according to the equals(object) method, then calling the hashCode() method on each of the two objects must produce the same integer result.

It is NOT required that if two objects are unequal according to the equals(Java.lang.Object) method, then calling the hashCode() method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

How to implement equals() method

**Steps that need to be taken into consideration while implementing equals method.**

**Use the == operator to check if the argument is a reference to this object.** If so, return true. This is just a performance optimization, but one that is worth doing if the comparison is potentially expensive.

**Use the instanceof operator to check if the argument has the correct type**.

If not, return false. Typically, the correct type is the class in which the method occurs. Occasionally, it is some interface implemented by this class. Use an interface if the class implements an interface that refines the equals contract to permit comparisons across classes that implement the interface. Collection interfaces such as Set, List, Map, and Map.Entry have this property.

Cast the argument to the correct type. Because this cast was preceded by an instanceof test, it is guaranteed to succeed.

For each significant field in the class, checks if that field of the argument matches the corresponding field of this object.

If all these tests succeed, return true; otherwise, return false

**When you are finished writing your equals method, ask yourself three questions: Is it symmetric? Is it transitive? Is it consistent?**

The correct implementation of equals method for the StringHelper class could be:

**Difference between throw and throws in Java?**

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

|  |  |  |
| --- | --- | --- |
| No. | throw | throws |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| 2) | Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| 3) | Throw is followed by an instance. | Throws is followed by class. |
| 4) | Throw is used within the method. | Throws is used with the method signature. |
| 5) | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |

**Difference Between Checked And Unchecked Exceptions :**

|  |  |
| --- | --- |
| **Checked Exceptions** | **Unchecked Exceptions** |
| They are known at compile time. | They are known at run time. |
| They are checked at compile time. | They are not checked at compile time. Because they occur only at run time. |
| These are compile time exceptions. | These are run time exceptions. |
| If  these exceptions are not handled properly in the application, they give compile time error. | If these exceptions are not handled properly, they don’t give compile time error. But application will be terminated prematurely at run time. |
| All sub classes of java.lang.Exception Class except sub classes of RunTimeException are checked exceptions. | All sub classes of RunTimeException and sub classes of java.lang.Error are unchecked exceptions. |

**Iterator vs Enumeration**

Between Enumeration and Iterator, Enumeration is older and its there from JDK1.0, while iterator was introduced later. Iterator can be used with ArrayList, HashSet and other collection classes.  Another similarity between Iterator and Enumeration in Java is that functionality of Enumeration interface is duplicated by the Iterator interface.

Only major difference between Enumeration and iterator is Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as by using Iterator we can manipulate the objects like adding and removing the objects from collection e.g. **Arraylist**.

Also Iterator is more secure and safe as compared to Enumeration because it does not allow other thread to modify the collection object while some thread is iterating over it and throws ConcurrentModificationException. This is by far most important fact for me for deciding between Iterator vs Enumeration in Java.

In Summary both Enumeration and Iterator will give successive elements, but Iterator is new and improved version where method names are shorter, and has new method called remove. Here is a short comparison:

Enumeration: **hasMoreElement(),nextElement**(),N/A,

Iterator : **hasNext(),next(),remove**()

So Enumeration is used whenever we want to make Collection objects as Read-only.

**Difference between final, finally and finalize**

There are many differences between **final, finally and finalize.** A list of differences between final, finally and finalize are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | final | finally | finalize |
| 1) | 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. | Finally is used to place important code, it will be executed whether exception is handled or not. | Finalize is used to perform clean up processing just before object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method. |

|  |
| --- |
| **Exception Handling with Method Overriding in Java?**  There are many rules if we talk about method overriding with exception handling. The Rules are as follows:  If the super class method does not declare an exception  If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.  If the super class method declares an exception  If the super class method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception. |
| **Advantage of java inner classes?**  There are basically three advantages of inner classes in java. They are as follows:  1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.  2) Nested classes are used to develop more **readable and maintainable** code because it logically group classes and interfaces in one place only.  3) **Code Optimization**: It requires less code to write. |
| |  |  | | --- | --- | | [Member Inner Class](http://www.javatpoint.com/member-inner-class) | A class created within class and outside method. | | [Anonymous Inner Class](http://www.javatpoint.com/anonymous-inner-class) | A class created for implementing interface or extending class. Its name is decided by the java compiler. | | [Local Inner Class](http://www.javatpoint.com/local-inner-class) | A class created within method. | | [Static Nested Class](http://www.javatpoint.com/static-nested-class) | A static class created within class. | | [Nested Interface](http://www.javatpoint.com/nested-interface) | An interface created within class or interface. | |

**There are 2 types of Nested Classes.?**

Static Nested Classes

Non-Static Nested Classes or Inner Classes

## Static Nested Classes In Java

* + If nested class is declared as static, then that nested class is called as **static nested class**
  + Static nested classes can contain both static and non-static members
  + We can access only static members of outer class inside a static nested class. We can’t access non-static members of outer class inside a static nested class.
  + We have seen that static methods can’t be abstract but static nested classes can be abstract.
  + Static nested class can be final.
  + Constructors and methods of nested classes can be overloaded.

## Non-Static Nested Classes In Java :

* + Nested classes which are declared as non-static or nested classes which can be accessed only though instantiating it’s outer class are called non-static nested classes. Non-static nested classes are also called as **Inner Classes.**

They are 3 types of Inner Classes in java.

* + **Member Inner Classes**

Member Inner Classes are non-static nested classes which are declared as non-static members of outer class.

Member inner classes must contain only non-static members. Static members are not allowed inside member inner classes.

You can declare a static field inside a member inner class if the field is final. And such field must be initialized at the time of declaration only. Remember, this rule is only for the fields not for the methods.

Member inner class may contain any number of IIB’s(Instantiation Initialization Block) but should not contain any SIB’s(Static Initialization Block).

We can access both static and non-static members of outer class inside a member inner class.

All members of outer class are accessible inside member inner class and all members of member inner class are accessible inside the outer class irrespective of their visibility.

Member inner classes can be abstract or can be final but not both.

* + **Local Inner Classes**

Local inner class in java is non-static nested class which is declared inside a method or a block.

Local inner classes must be defined inside a method or a block.

Local Inner Classes can’t be static. Because, local inner classes are nothing but local variables and local variables can’t be static.

Local inner classes can’t have static members. Only non-static members are allowed inside local inner classes. But local inner classes can contain static and final field.

To access members of local inner class, you must create an instance of it.

Local inner classes are local to a method or a block in which they are defined. i.e you can’t use local inner classes outside the method or block in which they are defined.

Only final local variables of methods or blocks containing local inner class can be used inside local inner class.

Local inner classes cannot be declared with access modifiers. i.e Local inner classes cannot be private, protected and public. But they can have private, public, protected and default members in them.

Local inner classes cannot be declared with access modifiers. i.e Local inner classes cannot be private, protected and public. But they can have private, public, protected and default members in them.

Local inner classes can be abstract or can be final but not both.

* + **Anonymous Inner classes**

Anonymous inner class, the name itself suggests that it is a class without a name. Anonymous inner class in java is an inner class or non-static nested class without a name.

Anonymous inner classes don’t have name. They are nameless.

You can create only one object to anonymous inner class. If you want to create another object, you have to write the whole class again.

When you are creating an anonymous inner class, you are actually creating a sub class to a class which needs to be modified. This sub class doesn’t have name and it is declared in another class. That’s why it is called Anonymous Inner Class.

While creating an anonymous inner class you are also creating an object to that subclass and it is referenced by super class reference variable. This also shows the polymorphism. Because, Super class reference variable can refer to super class object and also it’s sub class object.

Using anonymous inner class, you can implement both abstract classes and interfaces.

**Enum:**

**Enums** OR **Enum Types** OR **Enumeration types** are special data types which are used to represent similar kind of constants.

Enum types like classes can have **fields**, **constructors** and **methods** along with enum constants.

Enum constructors are **private** by default. Only private constructors are allowed in enum types. That’s why you can’t instantiate enum types using new operator.

Enum constants are created only once for the whole execution. All enum constants are created when you refer any enum constant first time in your code. While creating each enum constant, corresponding constructor is called.

Enum constants must be declared first ahead of fields, constructors and methods (if any).

All enum types extend **java.lang.Enum** class by default. As multiple inheritance is not supported in java, they can’t extend any other classes.

Enum types can implement any number of interfaces.

  Enum constants can have their own body called **Constant Specific Body**. In that body, you can define fields and methods. But, these methods and fields are visible within the Constant Specific Body in which they are defined.

Enum types are **final** by default. They can not be extended by any other types.

For every enum type written in a file, **.class** file will be generated after compilation.

Enum types can have any number of static initialization blocks as well as instance initialization blocks.

As java.lang.Enum class implements Comparable and Serializable interface, all enum types are **Comparable** and**Serializable** by default.

We can compare the enum constants using **“==”** operator.

You can retrieve the enum constants of any enum type using **values()** method. values() method returns an array of enum constants.

 Enums provide **type-safety** during compilation. That means you will get compile time error if you try to assign any other values other than the specified enum constants.

You can define enum types outside a class or inside a class but not inside a method or block.

**ordinal()** method is used get the order of an enum constant in an enum type.

 Enums are mostly used when you want to allow limited set of options which remain constant for whole execution and you know all possible options at compile time itself. For example, **choices on a menu** or **options of a combobox**.

**What if we call run() method directly instead start () method?**

|  |
| --- |
| Each thread starts in a separate call stack.  Invoking the run() method from main thread, the run () method goes onto the current call stack rather than at the beginning of a new call stack. |

**Daemon Thread in Java**

Daemon thread in java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically. There are many java daemon threads running automatically e.g. **gc, finalizer** etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc. Points to remember for Daemon Thread in Java.

It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.

Its life depends on user threads.

It is a low priority thread.

**Why JVM terminates the daemon thread if there is no user thread?**

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread? That is why JVM terminates the daemon thread if there is no user thread.

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

**Java Garbage Collection**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects. To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

**Advantage of Garbage Collection**

It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

**How can an object be unreferenced?**

There are many ways:

By nulling the reference

By assigning a reference to another

By annonymous object etc.

**finalize() method**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

**gc() method**

The gc() method is used to invoke the garbage collector to perform cleanup processing.

The gc() is found in System and Runtime classes.

public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector (GC). This thread calls the finalize () method before object is garbage collected.

**The entire collection framework is divided into four interfaces.**

**1) List**  —> It handles sequential list of objects. **ArrayList**, **Vector** and **LinkedList** classes implement this interface.

**2) Queue**  —> It handles special list of objects in which elements are removed only from the head. **LinkedList** and**PriorityQueue** classes implement this interface.

**3) Set**  —> It handles list of objects which must contain unique element. This interface is implemented by **HashSet**and **LinkedHashSet** classes and extended by **SortedSet** interface which in turn, is implemented by **TreeSet**.

**4) Map**  —> This is the one interface in Collection Framework which is not inherited from Collection interface. It handles group of objects as Key/Value pairs. It is implemented by **HashMap** and **HashTable** classes and extended by **SortedMap** interface which in turn is implemented by **TreeMap**.

Three of above interfaces (List, Queue and Set) inherit from Collection interface. Although, Map is included in collection framework it does not inherit from Collection interface.

1. **List Interface:**

List Interface represents an ordered or sequential collection of objects. This interface has some methods which can be used to store and manipulate the ordered collection of objects. The classes which implement the List interface are called as Lists. **ArrayList**, **Vector** and **LinkedList** are some examples of lists. You have the control over where to insert an element and from where to remove an element in the list.

Here are some properties of lists.

* Elements of the lists are ordered using Zero based index.
* You can access the elements of lists using an integer index.
* Elements can be inserted at a specific position using integer index. Any pre-existing elements at or beyond that position are shifted right.
* Elements can be removed from a specific position. The elements beyond that position are shifted left.
* A list may contain duplicate elements.
* A list may contain multiple null elements.
* List interface extends Collection interface. So, All 15 methods of Collection interface are inherited to List interface. Along with these methods, another 9 methods are included in the List interface to support the properties of lists.

1. [Advantages Of Using ArrayList Over Arrays](http://javaconceptoftheday.com/advantages-of-using-arraylist-over-arrays/)

Array and ArrayList are most used data types while developing any java applications. Both are used to store group of objects.

The drawbacks of arrays.

* + 1. Arrays are of fixed length. You can not change the size of the arrays once they are created.
    2. You can not accommodate an extra element in an array after they are created.
    3. Memory is allocated to an array during it’s creation only, much before the actual elements are added to it.

List down the advantages of using **ArrayList** over **Arrays**.

* + - * 1. You can define ArrayList as **re-sizable array**. Size of the ArrayList is not fixed. ArrayList can grow and shrink dynamically
        2. Elements can be inserted at or deleted from a particular position.
        3. ArrayList class has many methods to manipulate the stored objects.

d) If generics are not used, ArrayList can hold any type of objects.

1. Many are of the assumption that multiple insertion and removal operations on ArrayList will decrease the performance of an application. But, there will be no significant change in the performance of an application if you use ArrayList instead of arrays. Below example shows time taken to add 1000 string elements to ArrayList and array.
2. You can traverse an ArrayList in both the directions – forward and backward using ListIterator.
3. ArrayList can hold multiple null elements.
4. ArrayList can hold duplicate elements.
5. [**Collection Framework – The Queue Interface**](http://javaconceptoftheday.com/collection-framework-queue-interface/)

The Queue Interface extends Collection interface. It defines queue data structure which is normally **First-In-First-Out**. Queue is a data structure in which elements are added from one end and elements are deleted from another end. But, exception being the Priority Queue in which elements are removed from one end, but elements are added according to the order defined by the supplied comparator. Here is the hierarchy diagram of Queue interface.

**How Typical Queue Works?**

Queue is a data structure where elements are added from one end called tail of the queue and elements are removed from another end called head of the queue. Queue is also first-in-first-out type of data structure (except priority queue). That means an element which is inserted first will be the first element to be removed from the queue. You can’t add or get or set elements at an arbitrary position in the queues.

**Properties of Queue:**

1. **Null** elements are not allowed in the queue. If you try to insert null object into the queue, it throws NullPointerException.
2. Queue can have **duplicate** elements.
3. Unlike a normal list, queue is **not random access**. i.e you can’t set or insert or get elements at an arbitrary positions.
4. In most of cases, elements are inserted at one end called **tail** of the queue and elements are removed or retrieved from another end called **head** of the queue.
5. In the Queue Interface, there are two methods to obtain and remove the elements from the head of the queue. They are **poll()** and **remove()**. The difference between them is, poll() returns null if the queue is empty and remove() throws an exception if the queue is empty.
6. There are two methods in the Queue interface to obtain the elements but don’t remove. They are **peek()** and **element()**. peek() returns null if the queue is empty and element() throws an exception if the queue is empty.

## Methods Of Queue Interface:

Here are the methods of Queue interface. Some of the methods throw an exception if operation is not possible and some methods return a value (null or false) if operation is not possible.

|  |  |  |
| --- | --- | --- |
| Operation | Throws An Exception If operation is not possible | Returns null or false if operation is not possible |
| Add an element to the queue. | add() | offer() |
| Retrieve an element from the head of the queue. | element() | peek() |
| Retrieve And Remove an element from the head of the queue. | remove() | poll() |

1. [Collection Framework – The Deque Interface](http://javaconceptoftheday.com/collection-framework-deque-interface/)

The Deque is the short name for “**Double Ended Queue**”. As the name suggest, Deque is a linear collection of objects which supports insertion and removal of elements from both the ends. The Deque interface defines the methods needed to insert, retrieve and remove the elements from both the ends.

## Properties Of Deque:

1. Unlike Queue, Deque can have **null** elements. But, it is recommended not to insert null elements as many methods return null to indicate Deque is empty.
2. Deque can have **duplicate** elements.
3. You can’t set or get or insert the elements at an arbitrary position of Deque. i.e **Random access** is not possible with the Deque.
4. You can use **removeFirstOccurrenec**(E e), **removeLastOccurrence**(E e) and remove(E e) methods to delete the elements from the Deque.
5. **Deque as Queue :**

As Deque interface extends Queue interface, it inherits all methods of Queue interface. So, you can use all those inherited methods to perform Queue operations. Along with them, methods defined in the Deque interface can also be used for Queue operations. Below is the list of Queue methods and their equivalent Deque methods.

|  |  |  |  |
| --- | --- | --- | --- |
| **Queue Methods** | | **Equivalent Deque Methods** | |
| **add()** | | 1. **addLast()** | |
| **offer()** | | 1. **OfferLast()** | |
| **element()** | | 1. **getFirst()** | |
| **peek()** | | 1. **peekFirst()** | |
| **remove()** | | 1. **removeFirst()** | |
| **poll()** | | 1. **pollFirst()** | |

**II. Deque as Stack:**

Deque interface has two more methods – **pop()** and **push()**. These two methods make Deque to function as a stack (Last-In-First-Out). Along with these two methods, you can also use addFirst(), peekFirst() and removeFirst() for stack operations. Below is the list of Stack methods and their equivalent methods of Deque.

|  |  |
| --- | --- |
| Stack Methods | Equivalent Deque Methods |
| **push()** | 1. **addFirst()** |
| **pop()** | 1. **removeFirst()** |
| **peek()** | 1. **peekFirst()** |

1. [**Collection Framework – The Set Interface**](http://javaconceptoftheday.com/collection-framework-set-interface/)**.**

The Set interface defines a set which contains a linear collection of objects with no duplicates. The **Set interface** extends Collection interface. **Set interface does not have its own methods. All its methods are inherited from Collection interface. The only change that has been made to Set interface is that add() method will return false if you try to insert an element which is already present in the set.**

## Properties of Set:

## Set contains only unique elements. It does not allow duplicates.

* + Set can contain only one **null** element.
  + **Random access** of elements is not possible.
  + **Order of elements** in a set is implementation dependent. **HashSet** elements are ordered on hash code of elements. **TreeSet** elements are ordered according to supplied Comparator (If no Comparator is supplied, elements will be placed in ascending order) and **LinkedHashSet** maintains insertion order.
  + Set interface contains only methods inherited from Collection interface. It does not have its own methods. But, applies restriction on methods so that duplicate elements are always avoided.
  + One more good thing about Set interface is that the **stronger contract** between **equals**() and **hashCode**() methods. According to this contract, you can compare two set instances of different implementation types (**HashSet**, **TreeSet** and **LinkedHashSet**).
  + Two set instances, irrespective of their implementation types, are said to be equal if they contain same elements.

1. [**Collection Framework – The SortedSet Interface**](http://javaconceptoftheday.com/collection-framework-sortedset-interface/)

**The SortedSet interface** extends Set interface. SortedSet is a set in which elements are placed according to supplied comparator. This Comparator is supplied while creating a SortedSet. If you don’t supply comparator, elements will be placed in ascending order. SortedSet interface defines 6 more methods along with the inherited methods from Set->Collection interfaces

## Properties of SortedSet Interface:

* + SortedSet cannot have **null** elements. If you try to insert null element, it gives NullPointerException at run time.
  + As SortedSet is a set, **duplicate** elements are not allowed.
  + SortedSet elements are sorted according to supplied **Comparator**. If you don’t mention any Comparator while creating a SortedSet, elements will be placed in ascending order.
  + Inserted elements must be of **Comparable** type and they must be mutually comparable.
  + You can retrieve first element and last elements of the SortedSet. You can’t access SortedSet elements randomly. i.e **Random access** is denied.
  + SortedSets returned by **headSet()**, **tailSet()** and **subSet()** methods are just views of the original set. So, changes in the returned set are reflected in the original set and vice versa.

|  |  |  |
| --- | --- | --- |
| SL NO. | **SortedSet Interface Methods** | **Description** |
| 1 | Comparator<? super E> comparator() | Returns Comparator used to order the elements. If no comparator is supplied, it returns null. |
| 2 | SortedSet<E> **subSet**(E fromElement, E toElement) | Returns a portion of this set whose elements range from ‘fromElement’ (Inclusive) and ‘toElement’ (Exclusive). |
| 3 | SortedSet<E> **headSet**(E toElement) | Returns a SortedSet whose elements are in the range from first element of the set (Inclusive) to ‘toElement’ (exclusive). |
| 4 | SortedSet<E> **tailSet**(E fromElement) | Returns a SortedSet whose elements are in the range from ‘fromElement’ (Inclusive) to last element of the set (exclusive). |
| 5 | E **first**() | Returns first element of the SortedSet. |
| 6 | E **last**() | Returns last element of the SortedSet |

1. [**Collection Framework – The NavigableSet Interface**](http://javaconceptoftheday.com/collection-framework-navigableset-interface/)**.**

The **NavigableSet** is a SortedSet with navigation facilities. The **NavigableSet interface** provides many methods through them you can easily find closest matches of any given element. It has the methods to find out less than, less than or equal to, greater than and greater than or equal of any element in a SortedSet.

**Properties of NavigableSet Interface:**

NavaigableSet can’t have null elements.

NavigableSet doesn’t support duplicate elements.

NavigableSet can be traversed and accessed in either ascending or descending order.

Methods subSet(), headSet() and tailSet() differ from SortedSet interface in taking additional arguments describing whether upper bound and lower bound are inclusive or exclusive.

1. [**Collection Framework – The ArrayList Class**](http://javaconceptoftheday.com/collection-framework-arraylist-class/)**.**

**ArrayList**, in simple terms, can be defined as re-sizable array. ArrayList is same like normal array but it can grow and shrink dynamically to hold any number of elements. ArrayList is a sequential collection of objects which increases or decreases in size as we add or delete the elements.

In ArrayList, elements are positioned according to **Zero-based index**. That means, elements are inserted from index 0. **Default initial capacity** of an ArrayList is 10. This capacity increases automatically as we add more elements to arraylist. You can also specify initial capacity of an ArrayList while creating it.

ArrayList class implements **List interface** and extends **AbstractList**. It also implements 3 marker interfaces –**RandomAccess**, **Cloneable** and **Serializable**.

## Properties Of ArrayList :

* Size of the ArrayList is not fixed. It can increase and decrease dynamically as we add or delete the elements.
* ArrayList can have any number of null elements.
* ArrayList can have duplicate elements.
* As ArrayList implements RandomAccess, you can get, set, insert and remove elements of the ArrayList from  any arbitrary position.
* When you insert an element in the middle of the ArrayList, the elements at the right side of that position are shifted one position right and when you delete an element, they will be shifted one position left. This feature of the ArrayList causes some performance issues as shifting of elements is time consuming if ArrayList has lots of elements.
* Elements are placed according to **Zero-based index**. That means, first element will be placed at index 0 and last element at index n-1, where ‘n’ is the size of the ArrayList.
* ArrayList is not synchronized. That means, multiple threads can use same ArrayList simultaneously.

[**Collection Framework – The Vector Class**](http://javaconceptoftheday.com/collection-framework-vector-class/)**.**

* **The Vector Class** is also dynamically grow-able and shrink-able collection of objects like an [ArrayList](http://javaconceptoftheday.com/collection-framework-arraylist-class/) class. But, the main difference between ArrayList and Vector is that **Vector class is synchronized**. That means, only one thread can enter into vector object at any moment of time.
* Vector class is preferred over ArrayList class when you are developing a multi threaded application. But, precautions need to be taken because vector may reduce the performance of your application as it is thread safety and only one thread is allowed to have object lock at any moment of time and remaining threads have to wait until a thread releases the object lock which is held by it. So, it is always recommended that if you don’t need thread safety environment, it is better to use ArrayList class than the Vector class.
* Vector class has same features as ArrayList. Vector class also extends **AbstractList** class and implements **List interface**. It also implements 3 marker interfaces – **RandomAccess**, **Cloneable** and **Serializable**.

## Properties Of Vector Class:

* The main feature of Vector class is that it is thread safety. All methods of Vector class are synchronized so that only one thread can execute them at any given time. This feature of Vector class is useful when you need thread safety code.
* Thread safety property of Vector class effects the performance of an application as it makes threads to wait for object lock.
* **Capacity Increment:** Capacity increment is an amount by which the capacity of the vector is automatically incremented whenever size of the vector exceeds it’s capacity. You can pass this capacity increment while creating a vector. If you don’t pass, capacity increment will be treated as zero and capacity of the vector will be doubled whenever size exceeds capacity.
* Unlike an ArrayList, you can set the size of the Vector manually. If the new size is greater than the current size, the new slots will be filled with null elements. If the new size is smaller than current size, then the extra elements will be discarded.
* You can traverse the vector using **Enumeration** object. Vector class has a method called **elements()** which returns an Enumeration object consisting of all elements of Vector.
* Vector class has separate methods to retrieve first and last element of vector object. You will not find these methods in ArrayList class. **firstElement()** retrieves first element and **lastElement()** method retrieves last element of the vector.

[**Why Not To Use Vector Class In Your Code?**](http://javaconceptoftheday.com/not-use-vector-class-code/)

Vector class is often considered as obsolete or “Due for Deprecation” by many experienced Java developers. They always recommend and advise not to use Vector class in your code. They prefer using ArrayList over Vector class. In this article, I have tried to list out some points regarding why not to use Vector class in your code.

1) You can achieve Thread Safety without Vector.

Vector class has only one advantage over ArrayList i.e it is thread safety. But, you can achieve thread safe ArrayList by using synchronizedList() method of Collections class. Below is the sample code.

public class MainClass

{

public static void main(String[] args)

{

ArrayList<Integer> list = new ArrayList<Integer>();

Collections.synchronizedList(list);

//It returns Synchronized list backed by original list.

}

}

2) Thread Safeness of Vector class is time consuming.

All methods of Vector class are synchronized. This makes each and every operation on Vector object thread safe. But, it is time consuming. Because, you need to acquire object lock for each operation you want to perform on vector object. Usually, you need set of operations to be synchronized not each and every operation. Isn’t make sense to take the object lock once, perform the operations you want and then release the lock when you are done. Why acquire the lock again and again for each operations?. This is the time consuming process and decreases the performance of your application.

3) Enumeration Vs Iterator

Vector class has a method which return Enumeration over the elements of Vector object. Although, Enumerations are faster than the Iterator, but it is not backed by the original collection. That means, any changes made to original collection does not reflect in Enumeration object. They ignore the modifications done during iteration. This may cause issues.

4) Is Vector class poorly designed?

Vector class combines two features – “Re-sizable Array” and “Synchronization“. This makes poor design. Because, if you need just “Re-sizable Array” and you use Vector class for that, you will get “synchronized Resizable Array” not just re-sizable array. This may reduce the performance of your application. Therefore, instead of using Vector class, always use ArrayList class. You will have re-sizable array and whenever you want to make it synchronized, use Collections.SynchronizedList().

[Java Collection Framework – The **LinkedList** Class](http://javaconceptoftheday.com/java-collection-framework-linkedlist-class/)

In general terms, LinkedList is a data structure where each element consist of three things. First one is the reference to previous element, second one is the actual value of the element and last one is the reference to next element.

The **LinkedList** class in Java is an implementation of doubly linked list which can be used both as a **List** as well as**Queue**. The LinkedList in java can have any type of elements including null and duplicates. Elements can be inserted and can be removed from both the ends and can be retrieved from any arbitrary position.

The LinkedList class extends **AbstractSequentialList** and implements **List** and **Deque** interfaces. It also implements 2 marker interfaces – **Cloneable** and **Serializable**.

## Properties of LinkedList Class In Java

* Elements in the LinkedList are called as **Nodes**. Where each node consist of three parts – Reference To Previous Element, Value Of The Element and Reference To Next Element.
* Reference To Previous Element of first node and Reference To Next Element of last node are null as there will be no elements before the first node and after the last node.
* You can insert the elements at both the ends and also in the middle of the LinkedList. Below is the list of methods for insertion operations
* You can retrieve the elements form the head, from the middle and from the tail of the LinkedList. Below is the list of retrieval methods.
* Insertion and removal operations in LinkedList are faster than the ArrayList. Because in LinkedList, there is no need to shift the elements after each insertion and removal. only references of next and previous elements need to be changed.
* Retrieval of the elements is very slow in LinkedList as compared to ArrayList. Becaues in LinkedList, you have to traverse from beginning or end (whichever is closer to the element) to reach the element.
* The LinkedList can be used as **stack**. It has the methods pop() and push() which make it to function as Stack.
* The LinkedList can also be used as ArrayList, Queue, SIngle linked list and doubly linked list.
* LinkedList can have multiple **null** elements.
* LinkedList can have **duplicate** elements.
* LinkedList class in Java is not of type **Random Access**. i.e the elements can not be accessed randomly. To access the given element, you have to traverse the LinkedList from beginning or end (whichever is closer to the element) to reach the given element.

[Java Collection Framework – The PriorityQueue Class](http://javaconceptoftheday.com/java-collection-framework-priorityqueue-class/)

The PriorityQueue is a queue in which elements are ordered according to specified Comparator. You have to specify this Comparator while creating a PriorityQueue itself. If no Comparator is specified, elements will be placed in their natural order. The PriorityQueue is a special type of queue because it is not a First-In-First-Out (FIFO) as in the normal queues. But, elements are placed according to supplied Comaparator.

The PriorityQueue does not allow null elements. Elements in the PriorityQueue must be of Comparable type, If you insert the elements which are not Comparable, you will get ClassCastException at run time.

PriorityQueue class extends AbstractQueue class which in turn implements Queue interface. PriorityQueue also implements one marker interface – java.io.Serializable interface.

## Properties of PriorityQueue Class:

* Elements in the PriorityQueue are ordered according to supplied **Comparator**. If Comparator is not supplied, elements will be placed in their natural order.
* The PriorityQueue is **unbounded**. That means the capacity of the PriorityQueue increases automatically if the size exceeds capacity. But, how it grows is not specified.
* The PriorityQueue can have **duplicate** elements but cannot have **null** elements.
* All elements of the PriorityQueue must be of **Comparable type**.

Otherwise ClassCastException will be thrown at run time.

* The head element of the PriorityQueue is always the least element and tail element is always the largest element according to specified Comparator.
* The default initial capacity of PriorityQueue is **11**.
* You can retrieve the Comparator used to order the elements of the PriorityQueue using **comparator()** method.
* PriorityQueue is not a thread safe.

[**Java Collection Framework – The HashSet Class**](http://javaconceptoftheday.com/java-collection-framework-hashset-class/)

* The HashSet class in Java is an implementation of Set interface. HashSet is a collection of objects which contains only unique elements. Duplicates are not allowed in HashSet. HashSet gives constant time performance for **insertion, removal and retrieval** operations. It allows only one null element.
* The HashSet internally uses HashMap to store the objects. The elements you insert in HashSet will be stored as keys of that HashMap object and their values will be a constant called PRESENT. This constant is defined as **private static final Object PRESENT = new Object()** in the source code of HashSet class.
* HashSet class extends AbstractSet class and implements Set interface. It also implements Cloneable and Serializable marker interfaces.

## Properties Of HashSet Class In Java

* HashSet class uses **HashMap** internally to store the objects. The keys of that HashMap object will be the elements of HashSet and their values will be a constant.
* HashSet does not allow **duplicate** elements. If you try to insert a duplicate element, older element will be overwritten.
* HashSet can have maximum one **null** element.
* HashSet doesn’t maintain any order. The order of the elements will be largely unpredictable. And it also doesn’t guarantee that order will remain constant over time.
* HashSet offers constant time performance for insertion, removal and retrieval operations
* HashSet class is not synchronized. If you want synchronized HashSet, use **Collections.synchronizedSet()**method.

# [Java Collection Framework – The LinkedHashSet Class](http://javaconceptoftheday.com/java-collection-framework-linkedhashset-class/)

The LinkedHashSet in java is an ordered version of HashSet which internally maintains one doubly linked list running through it’s elements. This doubly linked list is responsible for maintaining the insertion order of the elements. Unlike HashSet which maintains no order, LinkedHashSet maintains insertion order of elements. i.e elements are placed in the order they are inserted. LinkedHashSet is recommended over HashSet if you want a unique collection of objects in an insertion order.

The LinkedHashSet class extends HashSet class and implements Set interface. It also implements Cloneable and Serializable marker interfaces.

## Properties of LinkedHashSet Class In Java:

LinkedHashSet internally uses **LinkedHashMap** to store it’s elements just like HashSet which internally uses HashMap to store it’s elements.

* LinkedHashSet maintains **insertion order**. This is the main difference between LinkedHashSet and HashSet.
* LinkedhashSet also gives **constant time performance** for insertion, removal and retrieval operations. The performance of LinkedHashSet is slightly less than the Hashset as it has to maintain doubly linked list internally to order it’s elements.
* Iterator returned by LinkedHashSet is **fail-fast**. i.e if the LinkedHashSet is modified at any time after the Iterator is created, it throws **ConcurrentModificationException**.
* LinkedHashSet doesn’t allow **duplicate** elements and allows only one **null** element.
* LinkedHashSet is not **synchronized**. To get the synchronized LinkedHashSet, use**Collections.synchronizedSet()** method.

[Java Collection Framework – The TreeSet Class](http://javaconceptoftheday.com/java-collection-framework-treeset-class/)

The **TreeSet** is another popular implementation of Set interface. We have seen other two implementations of Set interface –  [HashSet](http://javaconceptoftheday.com/java-collection-framework-hashset-class/) and [LinkedHashSet](http://javaconceptoftheday.com/java-collection-framework-linkedhashset-class/). HashSet doesn’t maintain any order where as LinkedHashSet maintains insertion order. The main difference between these two implementations and Treeset is, elements in TreeSet are**sorted** according to supplied **Comparator**. You need to supply this Comparator while creating a TreeSet itself. If you don’t pass any Comparator while creating a TreeSet, elements will be placed in their **natural.**

**The TreeSet class** in java is a direct implementation of **NavigableSet** interface which in turn extends **SortedSet** interface (which in turn extends Set interface).

## Properties Of TreeSet Class In Java

The elements in TreeSet are sorted according to specified **Comparator**. If no Comparator is specified, elements will be placed according to their natural ascending order.

Elements inserted in the TreeSet must be of **Comparable** type and elements must be mutually comparable. If the elements are not mutually comparable, you will get **ClassCastException** at run time

TreeSet does not allow even a single **null** element.

TreeSet is not **synchronized**. To get a synchronized TreeSet, use **Collections.synchronizedSortedSet()** method.

TreeSet gives performance of order **log(n)** for insertion, removal and retrieval operations.

Iterator returned by TreeSet is of **fail-fast** nature. That means, If TreeSet is modified after the creation of Iterator object, you will get **ConcurrentModificationException**.

TreeSet internally uses **TreeMap** to store it’s elements just like HashSet and LinkedHashSet which use HashMap and LinkedHashMap respectively to store their elements.

[J**ava Collection Framework – The Map Interface**](http://javaconceptoftheday.com/java-collection-framework-map-interface/)

The **Map interface** in java is one of the four top level interfaces of Java Collection Framework along with [List](http://javaconceptoftheday.com/collection-framework-list-interface/), [Set](http://javaconceptoftheday.com/collection-framework-set-interface/)and [Queue](http://javaconceptoftheday.com/collection-framework-queue-interface/) interfaces. But, unlike others, it doesn’t inherit from [Collection](http://javaconceptoftheday.com/collection-framework-collection-interface/) interface. Instead it starts it’s own interface hierarchy for maintaining the **key-value associations**. Map is an object of key-value pairs where each key is associated with a value. This interface is the replacement for ‘**Dictionary**‘ class which is an abstract class introduced in JDK 1.0.

**HashMap**, **LinkedHashMap** and **TreeMap** are three popular implementations of Map interface.

**Properties of Map Interface In Java:**

1) Map interface is a part of Java Collection Framework, but it doesn’t inherit **Collection Interface**.

2) Map interface stores the data as a **key-value pairs** where each key is associated with a value.

3) A map cannot have duplicate **keys** but can have duplicate **values**.

4) Each key **at most** must be associated with one value.

5) Each key-value pairs of the map are stored as **Map.Entry** objects. Map.Entry is an inner interface of Map interface.

6)The common implementations of Map interface are **HashMap**, **LinkedHashMap** and **TreeMap**.

7) Order of elements in map is implementation dependent. **HashMap** doesn’t maintain any order of elements.**LinkedHashMap** maintains **insertion order** of elements. Where as **TreeMap** places the elements according to supplied **Comparator**.

8) The Map interface provides three methods, which allows map’s contents to be viewed as a **set of keys** (keySet() method), **collection of values** (values() method), or **set of key-value mappings** (entrySet() method).

[What is the difference between Sorting performance of](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15) **[Arrays.sort](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)** [() vs](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15) **[Collections.sort](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)**[() ? Which one is faster? Which one to use and when?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting.   
  
So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.

[Java Collection Framework – The HashMap Class.](http://javaconceptoftheday.com/java-collection-framework-map-interface/)

The *java.util.HashMap* is a popular implementation of *Map* interface which holds the data as key-value pairs. *HashMap* extends *AbstractMap* class and implements *Cloneable* and *Serializable* interfaces. In this article, we will discuss about hierarchy of HashMap, properties of HashMap and some important methods of HashMap in java.

As already said, HashMap extends **AbstractMap** class and implements Cloneable and Serializable interfaces. **AbstractMap** is an abstract class which provides skeletal implementation of Map interface. Below is the hierarchy structure of java.util.HashMap class.

Properties Of HashMap In Java :

1) HashMap holds the data in the form of key-value pairs where each key is associated with one value.

2) HashMap doesn’t allow duplicate keys. But it can have duplicate values.

3) HashMap can have multiple null values and only one null key.

4) HashMap is not synchronized. To get the synchronized HashMap, use Collections.synchronizedMap () method.

5) HashMap maintains no order.

6) HashMap gives constant time performance for the operations like get() and put() methods.

7) Default initial capacity of HashMap is 16.

[**What is Load factor and Rehashing in Hashmap?**](http://javabypatel.blogspot.in/2015/10/what-is-load-factor-and-rehashing-in-hashmap.html)

**Load Factor:**

When the total number of items in hashmap goes on increasing keeping the default initial capacity of hashmap 16, At one point of time, hashmap performance will start degrading and need to increase buckets for improving performance.  
  
 Load Factor is a measure, which decides when exactly to increase the hashmap capacity (buckets) to maintain get and put operation complexity of O(1).

Default load factor of Hashmap is 0.75f (i.e 75% of current map size).

**Rehashing:**

Rehashing is the process of re-calculating the hashcode of already stored entries (Key-Value pairs), to move them to another bigger size hashmap when Load factor threshold is reached.  
  
When the number of items in map, crosses the Load factor limit at that time hashmap doubles its capacity and hashcode is re-calculated of already stored elements for even distribution of key-value pairs across new buckets.  
  
**Why Rehashing is required?**   
  
After doubling the capacity, what to do with the key-value pairs already present in buckets?

If we keep the existing key-value pairs as it is, then doubling the capacity may not help, because O(1) complexity will be achieved only if items are evenly distributed across all buckets.  
  
So for each existing key-value pairs, hashcode is calculated again with increased hashmap capacity as a parameter, which results in either placing the item in same bucket or in different bucket.

When the size of hashmap is changed, the process of re-calculating the hashcode of already placed key-value pair again is known as Rehashing.

Rehashing is done to distribute items across the new length hashmap, so that get and put operation time complexity remains O(1).  
  
NOTE: Hashmap maintain complexity of O(1) while inserting data in and getting data from hashmap, but for 12th key-value pair, put request will no longer be O(1), because as soon as map will realize that 12th element came in, that is 75% of map is filled. It will first double the bucket (array) capacity and then it will go for Rehash.  
 Rehashing requires re-computing hashcode of already placed 11 key-value pairs again and putting them at new index which requires time.

But overall time complexity provided by hashmap, which is O(1) for get and put operations, will amortize Rehashing process over long run.

**How To Convert HashMap To ArrayList In Java?**

As *HashMap* contains key-value pairs, there are three ways you can convert given *HashMap* to *ArrayList*. You can convert *HashMap* keys into *ArrayList* or you can convert *HashMap* values into *ArrayList* or you can convert key-value pairs into *ArrayList*.

a) **Conversion Of HashMap Keys Into ArrayList :**

For this, we use keySet() method of HashMap which returns the Set containing all keys of the HashMap. And then we pass this Set while constructing the ArrayList.

//Creating a HashMap object

HashMap<String, String> map = new HashMap<String, String>();

//Getting Set of keys from HashMap

Set<String> keySet = map.keySet();

//Creating an ArrayList of keys by passing the keySet

ArrayList<String> listOfKeys = new ArrayList<String>(keySet);

**b) Conversion Of HashMap Values Into ArrayList :**

For this, we use values() method of HashMap which returns the Collection containing all values of the HashMap. Then we use this Collection to create the ArrayList of values.

//Creating a HashMap object

HashMap<String, String> map = new HashMap<String, String>();

//Getting Collection of values from HashMap

Collection<String> values = map.values();

**//Creating an ArrayList of values**

ArrayList<String> listOfValues = new ArrayList<String>(values);

c) Conversion Of HashMap’s Key-Value Pairs Into ArrayList :

For this, we use entrySet() method of HashMap which returns the Set of Entry<K, V> objects where each Entry object represents one key-value pair. We pass this Set to create the ArrayList of key-value pairs.

**//Creating a HashMap object**

HashMap<String, String> map = new HashMap<String, String>();

//Getting the Set of entries

Set<Entry<String, String>> entrySet = map.entrySet();

//Creating an ArrayList Of Entry objects

ArrayList<Entry<String, String>> listOfEntry = new ArrayList<Entry<String,String>>(entrySet);

### How HashMap internally works in Java

Hash Map is one of the most used collection, though it will be surprising to know that maps themselves are not collections because they don't implement Collection interface. However collection view of a map can be obtained using entrySet() method. To obtain a collection-view of the keys, keySet() method can be used.

Coming to the internal working of the HashMap, which is also a favourite [Java Collections interview question](http://netjs.blogspot.com/2015/11/java-collections-interview-questions.html), there are four things we should know about before going into the internals of how does **HashMap** work in Java -

* **HashMap** works on the principal of hashing.
* **Map.Entry interface** - This interface gives a map entry (key-value pair). HashMap in Java stores both key and value object, in bucket, as an object of Entry class which implements this nested interface Map.Entry.
* **hashCode()** -HashMap provides put(key, value) for **storing** and get(key) method for**retrieving** Values from HashMap. When put() method is used to store (Key, Value) pair, HashMap implementation **calls hashcode** on Key object to calculate a hash that is used to find a bucket where Entry object will be stored. When get() method is used to retrieve value, again key object is used to calculate a hash which is used then to find a bucket where that particular key is stored.
* **equals()** - equals() method is used to **compare objects for equality**. In case of HashMap key object is used for comparison, also using equals() method Map knows how to handle**hashing collision** (hashing collision means more than one key having the same hash value, thus assigned to the same bucket. In that case objects are stored in a linked list, refer[figure](http://netjs.blogspot.in/2015/05/how-hashmap-internally-works-in-java.html#hashmap) for more clarity.   
  Where hashCode method helps in finding the bucket where that key is stored, equals method helps in finding the right key as there may be more than one key-value pair stored in a single bucket.

**\*\*** Bucket term used here is actually an index of array, that array is called table in HashMap implementation. Thus table[0] is referred as bucket0, table[1] as bucket1 and so on.

* Refer [Overriding hashCode() and equals() method in Java](http://netjs.blogspot.com/2015/06/overriding-hashcode-and-equals-method.html) to know more about hashCode() and equals() method

How important it is to have a proper hash code and equals method can be seen through the help of the following program -

public class HashMapTest {

public static void main(String[] args) {

Map <Key, String> cityMap = new HashMap<Key, String>();

cityMap.put(new Key(1, "NY"),"New York City" );

cityMap.put(new Key(2, "ND"), "New Delhi");

cityMap.put(new Key(3, "NW"), "Newark");

cityMap.put(new Key(4, "NP"), "Newport");

System.out.println("size before iteration " + cityMap.size());

Iterator <Key> itr = cityMap.keySet().iterator();

while (itr.hasNext()){

System.out.println(cityMap.get(itr.next()));

}

System.out.println("size after iteration " + cityMap.size());

}

}

// This class' object is used as key

// in the HashMap

class Key{

int index;

String Name;

Key(int index, String Name){

this.index = index;

this.Name = Name;

}

@Override

// A very bad implementation of hashcode

// done here for illustrative purpose only

public int hashCode(){

return 5;

}

@Override

// A very bad implementation of equals

// done here for illustrative purpose only

public boolean equals(Object obj){

return true;

}

}

**Output**

size before iteration 1

Newport

size after iteration 1

**Understanding the Code**

Lets get through the code to see what is happening, this will also help in understanding how put works internally.

Notice that I am inserting 4 values in the [HashMap](http://netjs.blogspot.com/2015/05/how-to-loop-iterate-hash-map-in-java.html), still in the output it says size is 1 and iterating the map gives me the last inserted entry. **Why is that**?

Answer lies in, how hashCode() and equals() method are implemented for the key Class. Have a look at the hashCode() method of the class Key which **always returns "5"** and the equals() method which is **always returning "true"**.

When a value is put into HashMap **it calculates a hash using key object** and for that it **uses the hashCode()** method of the key object class (or its parent class). Based on the calculated hash value HashMap implementation decides which bucket should store the particular Entry object.

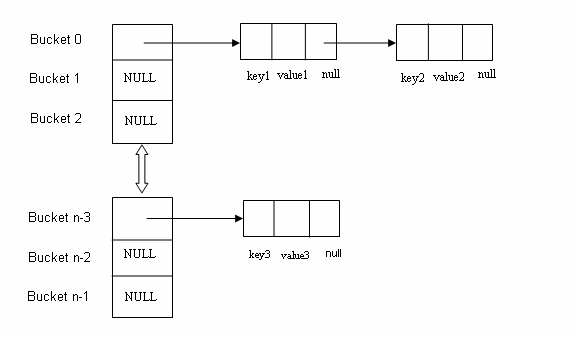
In my code the hashCode() method of the key class always returns "5". This effectively means calculated hash value, is same for all the entries inserted in the HashMap. Thus all the entries are stored in the same bucket.

**Second thing**, a HashMap implementation does is to **use equals() method** to see if the key is equal to any of the already inserted keys (**Recall that there may be more than one entry in the same bucket**). Note that, with in a bucket key-value pair entries (Entry objects) are stored in a linked-list (*Refer figure for more clarity*). In case hash is same, but equals() returns false (which essentially means more than one key having the same hash or hash collision) Entry objects are stored, with in the same bucket, in a linked-list.

In my code, I am always **returning true for equals() method** so the HashMap implementation "*thinks*" that the keys are equal and overwrites the value. So, in a way using hashCode() and equals() I have "*tricked*" HashMap implementation to think that all the keys (even though different) are same, thus overwriting the values.

In a nutshell there are three scenarios in case of put() -

* Using hashCode() method, hash value will be calculated. Using that hash it will be ascertained, in which bucket particular entry will be stored.
* equals() method is used to find if such a key already exists in that bucket, if no then a new node is created with the map entry and stored within the same bucket. A linked-list is used to store those nodes.
* If equals() method returns true, which means that the key already exists in the bucket. In that case, the new value will overwrite the old value for the matched key.

[](http://4.bp.blogspot.com/-x_U5Yjgsg8c/VVNgwvz7WNI/AAAAAAAAAHg/BreaAUMvJpc/s1600/hashMap+internal.png)

**Pictorial representation of how Entry (key-value pair) objects will be stored in table array**

**How get() methods works internally**

As we already know how Entry objects are stored in a bucket and what happens in the case of Hash Collision it is easy to understand what happens when key object is passed in the get method of the HashMap to retrieve a value.

Using the key again hash value will be calculated to determine the bucket where that Entry object is stored, in case there are more than one Entry object with in the same bucket stored as a linked-list equals() method will be used to find out the correct key. As soon as the matching key is found get() method will return the value object stored in the Entry object.

**In case of null Key**

As we know that HashMap also allows null, though there can only be one null key in HashMap. While storing the Entry object HashMap implementation checks if the key is null, in case key is null, it always map to bucket 0 as hash is not calculated for null keys.

**HashMap changes in Java 8**

Though HashMap implementation provides constant time performance O(1) for get() and put() method but that is in the ideal case when the Hash function distributes the objects evenly among the buckets.

But the performance may worsen in the case hashCode() used is not proper and there are lots of hash collisions. As we know now that in case of hash collision entry objects are stored as a node in a linked-list and equals() method is used to compare keys. That comparison to find the correct key with in a linked-list is a linear operation so in a worst case scenario the complexity becomes O(n).

To address this issue in Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached. Which means HashMap starts with storing Entry objects in linked list but after the number of items in a hash becomes larger than a certain threshold, the hash will change from using a linked list to a balanced tree, this will improve the worst case performance from O(n) to O(log n).

**Points to note -**

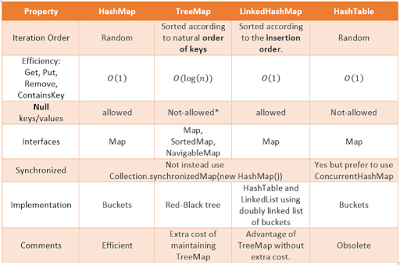
* HashMap works on the principal of hashing.
* HashMap uses the hashCode() method to calculate a hash value. Hash value is calculated using the key object. This hash value is used to find the correct bucket where Entry object will be stored.
* HashMap uses the equals() method to find the correct key whose value is to be retrieved in case of get() and to find if that key already exists or not in case of put().
* Hashing collision means more than one key having the same hash value, in that case Entry objects are stored as a linked-list with in a same bucket.
* With in a bucket values are stored as Entry objects which contain both key and value.
* In Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached while storing values. This improves the worst case performance from O(n) to O(log n).

**Difference between HashMap and Hashtable in Java?**

Both HashMap and Hashtable implements Map interface but there is some significant difference between them which is important to remember before deciding whether to use HashMap or Hashtable in Java. Some of them are thread-safety, synchronization, and speed. here are those differences:

1. The HashMap class is roughly equivalent to Hashtable, except that it is non-synchronized and permits nulls. (HashMap allows null values as key and value whereas [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html)doesn't allow nulls).

2. One of the major differences between HashMap and Hashtable is that HashMap is non-synchronized whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap cannot be shared between multiple threads without proper synchronization. Java 5 introduces [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html)which is an alternative of Hashtable and provides better scalability than Hashtable in Java.

[](https://3.bp.blogspot.com/-iw2pMTU6pHo/VvP3HoW7XrI/AAAAAAAAFSQ/By53EkcfMGkoth857hnsc57XxYy8uwfAA/s1600/Difference+between+HashMap,+TreeMap,+LinkedHashMap+and+hashtable+in+Java.png)

3. Another significant difference between HashMap vs Hashtable is that Iterator in the HashMap is  a fail-fast iterator  while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally  by adding or removing any element except Iterator's own remove() method. But this is not a guaranteed behavior and will be done by JVM on best effort. This is also an important [difference between Enumeration and Iterator in Java](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-enumeration.html).  
  
4. One more notable difference between Hashtable and HashMap is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don't need synchronization and HashMap are only used by one thread, it outperforms Hashtable in Java.  
  
5. HashMap does not guarantee that the order of the map will remain constant over time.

**HashMap and Hashtable : note on Some Important Terms**

1) Synchronized means only one Thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a Hashtable will have to acquire a lock on the object while others will wait for the lock to be released.

2) Fail-safe is relevant from the context of iterators. If an [Iterator or ListIterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

**Difference between ConcurrentHashMap and Hashtable?**

So what is the difference between Hashtable and ConcurrentHashMap, both can be used in the multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for a longer duration.

Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.  In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration. This will be clearer by looking at this diagram which explains the internal working of ConcurrentHashMap in Java.

**The difference between ConcurrentHashMap and Collections.synchronizedMap?**

ConcurrentHashMap is designed for concurrency and improve performance while HashMap which is non-synchronized by nature can be synchronized by applying a wrapper using synchronized Map. Here are some of the common differences between ConcurrentHashMap and synchronized map in Java  
  
ConcurrentHashMap does not allow null keys or null values while synchronized HashMap allows one null key.

**Comparator vs Comparable**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Comparable** | **Comparator** |
| ***Sorting log*ic** | Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects | Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc. |
| **Implementation** | Class whose objects to be sorted must implement this interface.e.g Country class needs to implement comparable to collection of country object by id | Class whose objects to be sorted do not need to implement this interface.Some other class can implement this interface. E.g.-CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id |
| **Sorting method** | int compareTo(Object o1) This method compares this object with o1 object and returns  a integer.Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1 | int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns  a integer.Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1 |
| **Calling method** | Collections.sort(List) Here objects will be sorted on the basis of CompareTo method | Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator |
| **Package** | Java.lang.Comparable | Java.util.Comparator |

**What is Fail-fast in java?**

A fail-fast system is nothing but immediately report any failure that is likely to lead to failure. When a problem occurs, a fail-fast system fails immediately. In Java, we can find this behavior with iterators. In case, you have called iterator on a collection object, and another thread tries to modify the collection object, then concurrent modification exception will be thrown. This is called fail-fast.

**What is Fail-safe iterator?**

Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure. Any structural modification done to the iterator affects the copied data structure. So, original data structure remains structurally unchanged .Hence, no ConcurrentModificationException throws by the fail safe iterator.

Two issues associated with Fail Safe Iterator are :

1. Overhead of maintaining the copied data structure i.e memory.

2. Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.

Recap : Difference between Fail Fast Iterator and Fail Safe Iterator 

|  |  |  |
| --- | --- | --- |
|  | Fail Fast Iterator | Fail Safe Iterator |
| Throw ConcurrentModification Exception | Yes | No |
| Clone object | No | Yes |
| Memory Overhead | No | Yes |
| Examples | HashMap,Vector,ArrayList,HashSet | CopyOnWriteArrayList, ConcurrentHashMap |

**What will happen if you put return statement or System.exit () on try or catch block? Will finally block execute?**

This is a very popular tricky Java question and it's tricky because many programmers think that no matter what, but the [finally block](http://java67.blogspot.com/2016/06/difference-between-final-vs-finally-vs-finalize-in-java.html) will always execute. This question challenge that concept by putting a return statement in the try or catch block or calling System.exit() from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call System.exit() from try or catch block. 

**Synchronization:**

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource. The synchronization is mainly used to

**To prevent thread interference,**

**To prevent consistency problem.**

There are two types of synchronization - Process Synchronization, Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

Mutual Exclusive- Synchronized method, Synchronized block, static synchronization.

Cooperation (Inter-thread communication in java)

**Concept of Lock in Java**

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations

**Static synchronization**

If you make any static method as synchronized, the lock will be on the class not on object.

**What is Dead Lock?**

When two or more threads are waiting for each other to release lock and get stuck for infinite time, situation is called deadlock. It will only happen in case of multitasking.

**[How to find a deadlock has occurred in Java? How to detect a Deadlock in Java?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "how-to-find-a-deadlock-has-occurred-in-java-" \o "How to find a deadlock has occurred in Java?)**

Earlier versions of Java had no mechanism to handle/detect deadlock. Since JDK 1.5 there are some powerful methods added in the java.lang.management package to diagnose and detect deadlocks. The **java.lang.management.ThreadMXBean** interface is management interface for the thread system of the Java virtual machine. It has two methods which can leverage to detect deadlock in a Java application.

**findMonitorDeadlockedThreads**() - This method can be used to detect cycles of threads that are in deadlock waiting to acquire object monitors. It returns an array of thread IDs that are deadlocked waiting on monitor.

**findDeadlockedThreads**() - It returns an array of thread IDs that are deadlocked waiting on monitor or ownable synchronizers.

[**How to Avoid the Deadlock in Java**](http://javaconceptoftheday.com/avoid-the-deadlock-java/)**?**

Deadlock is a dangerous condition, if it happens , it will bring the whole application to complete halt. So, extra care need to be taken to avoid the deadlock. Followings are some tips that can be used to avoid the deadlock in java.

Try to avoid nested synchronized blocks. Nested synchronized blocks makes a thread to acquire another lock while it is already holding one lock. This may create the deadlock if another thread wants the same lock which is currently held by this thread.

**Lock Ordering :**

If you needed nested synchronized blocks at any cost, then make sure that threads acquire the needed locks in some predefined order. For example, If there are three threads t1, t2 and t3 running concurrently and they needed locks A, B and C in thze following manner,

[?](http://javaconceptoftheday.com/avoid-the-deadlock-java/)

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **Thread t1 :**  **Lock A**  **Lock B**  **Thread t2 :**  **Lock A**  **Lock C**  **Thread t3 :**  **Lock A**  **Lock B**  **Lock C** |

In the above scenario, t1 needs A and B locks, t2 needs A and C locks and t3 needs A, B and C locks. If you define an order to acquire the locks like, Lock A must be acquired before Lock B and Lock B must be acquired before Lock c, then deadlock never occurs in the above case.

If you define such lock ordering, then thread t2 never acquire lock C and t3 never acquire lock B and lock C until they got lock A. They will wait for lock A until it is released by t1. After lock A is released by t1, any one of these threads will acquire lock A on the priority basis and finishes their task. Other thread which is waiting for lock A, will never try to acquire remaining locks.

By defining such lock ordering, you can avoid the deadlock.

**Lock Timeout:**

Another deadlock preventive tip is to specify the time for a thread to acquire the lock. If it fails to acquire the specified lock in the given time, then it should give up trying for a lock and retry after some time. Such method of specifying time to acquire the lock is called lock timeout.

Lock the code where it is actually needed. For example, If you want only some part of the method to be thread safety, then lock only that part not the whole method.

**Method vs Block level synchronization**

Scope of lock is reduced by **Synchronized block**. As scope of lock is inversely proportional to performance, that's why it's always better to lock only critical section of the code. The best example of using synchronized block is double checked locking in Singleton pattern where instead of locking whole getInstance() method we only lock critical section of code which is used to create Singleton instance. This improves performance drastically because locking is only required one or two times.

In case of synchronized block, thread acquires lock when they enter synchronized block and release when they leave synchronized block. On the other hand in case of synchronized method, lock is acquired by thread when it enters method and released when it leaves method, either normally or by throwing Exception.

**Difference between "implements Runnable" and "extends Thread" in Java?**

### 1) Multiple Inheritance Limitation

As you know, Java doesn’t support multiple inheritance. A class in java can extend only one class. If you extend Thread class, then your class will not be able to extend any other class. This will limit your class to thread behavior. If you implement Runnable interface, then you will have an option for your class to extend any other class and inherit behaviors from other class also.

### 2) Overhead Of Additional Methods

If you extend Thread class, all methods of Thread class will be inheriting to your class which you may not need. This will cause additional overhead. You can remove this overhead by implementing Runnable interface.

### 3) Logical Separation Of Task From The Runner

### If you implement Runnable interface, it will separate actual task from the runner. Runnable interface represents only the task and you can pass this task to any type of runner, either a thread or any executors.

### 4) Best Object Oriented Design Practice

### In object oriented programming, extending a class means modifying or improving the existing class. If you are not improving the class, then it is not a good practice to extend it. So, implementing Runnable will be the best object oriented design practice.

### 5) Loosely Coupled Vs Tightly coupled

### “Implements Runnable” makes your code loosely coupled. Because it separates the task from the runner. “Extends Thread” will make your code tightly coupled. Because, single class will act as both task container as well as runner.

### 6) Reusability

### Implementing Runnable improves the reusability of your code. Because, Runnable contains only the task and you can use it wherever and whenever you want.

### 7) Specialization Vs Generalization

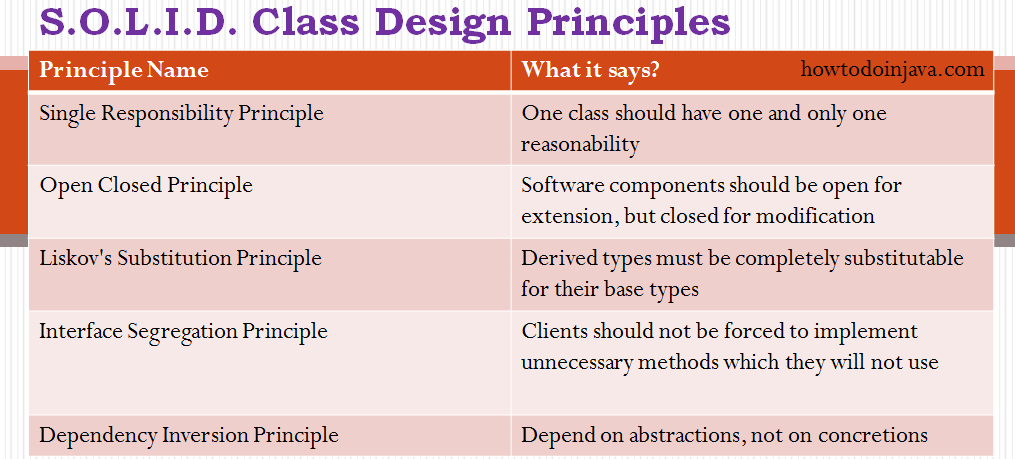
### “Extends Thread” gives more specialized code. Because, it defines the thread specific task. Where as “Implements Runnable” gives more generalized version of the task applicable to many threads.

### 8) Maintenance

### “Implements Runnable” will make your code easily maintainable as it separates the task from the runner. If you want to modify the task at any time, you can do so easily without disturbing the runner.

**Extends Thread Vs Implements Runnable In Java :**

|  |  |
| --- | --- |
| Implements Runnable | Extends Thread |
| You can extend any other class. | You can’t extend any other class. |
| No overhead of additional methods . | Overhead of additional methods from Thread class. |
| Separates the task from the runner. | Doesn’t separate the task from the runner. |
| Best object oriented programming practice. | Not a good object oriented programming practice. |
| Loosely coupled. | Tightly coupled. |
| Improves the reusability of the code. | Doesn’t improve the reusability of the code. |
| More generalized task. | Thread specific task. |
| Maintenance of the code will be easy. | Maintenance of the code will be time consuming. |



**Some of the benefits of collections framework are;**

Reduced development effort by using core collection classes rather than implementing our own collection classes. Code quality is enhanced with the use of well tested collections framework classes. Reduced effort for code maintenance by using collection classes shipped with JDK.

Reusability and Interoperability

**Describe, in general, how java's garbage collector works ?**

The Java runtime environment deletes objects when it determines that they are no longer being used. This process is known as garbage collection. The Java runtime environment supports a garbage collector that periodically frees the memory used by objects that are no longer needed. The Java garbage collector is a mark-sweep garbage collector that scans Java's dynamic memory areas for objects, marking those that are referenced. After all possible paths to objects are investigated, those objects that are not marked (i.e. are not referenced) are known to be garbage and are collected.

**What is synchronization and why is it important in Java?**

|  |
| --- |
| W**hat is synchronization and why is it important? Describe synchronization in respect to multithreading.**  Synchronization is the process of allowing threads to execute one after another.  Synchronization control the access the multiple threads to a shared resources. Without synchronization of threads, one thread can modify a shared variable while another thread can update the same shared variable, which leads to significant errors.  **What is synchronization and why is it important?**  Java supports multiple threads to be executed. This may cause two or more threads to access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in synch. Synchronization avoids memory consistence 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, your thread is blocked until that thread releases the monitor. |

**Join Method in Thread:**

Join method belongs to Thread object and not a static member. When join is invoked an a thread instance, this thread will tell currently executing thread to wait till the Joining thread completes.

Join is used in the situations when a task which should be completed before the current task is going to finish.

Just to give another example of how Join works, say we have a slow train on the track moving and another fast train should take the same track, so, we make way to the fast train and tell the slow train to join once after the fast one is left.

Yield Method in Thread**:** Yield is a Static method.

Yield tells the currently executing thread to give a chance to the threads that have equal priority in the Thread Pool. There is no guarantee that Yield will make the currently executing thread to runnable state immediately.

Remember an important point that yield method does not make the thread to go to wait or Blocked state. It can only make a thread from Running State to Runnable State.

Life cycle of a Thread (Thread States)

[Life cycle of a thread](http://www.javatpoint.com/life-cycle-of-a-thread) : [New](http://www.javatpoint.com/life-cycle-of-a-thread#threadstatenew),[Runnable](http://www.javatpoint.com/life-cycle-of-a-thread#threadstaterunnable),[Running](http://www.javatpoint.com/life-cycle-of-a-thread#threadstaterunning) ,[Non-Runnable (Blocked)](http://www.javatpoint.com/life-cycle-of-a-thread#threadstateblocked) ,[Terminated](http://www.javatpoint.com/life-cycle-of-a-thread#threadstateterminated)

A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

|  |
| --- |
| 1) **New**  The thread is in new state if you create an instance of Thread class but before the invocation of start() method. |

2) **Runnable**

The thread is in runnable state after invocation of start () method, but the thread scheduler has not selected it to be the running thread.

3) **Running**

The thread is in running state if the thread scheduler has selected it.

4) **Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

5) **Terminated**

A thread is in terminated or dead state when its run() method exits.

Some Things-To-Remember About wait(), notify() and notifyAll() :

If a thread calls notify() method and more than one threads are waiting for the object lock, then only one thread will be notified randomly.

When a thread calls notifyAll() method on an object, it notifies all the threads which are waiting for this object lock. But, only one thread will acquire this object lock depending upon priority.

When you call sleep() method on a thread, thread goes to sleep with holding the object lock with it. But, if you call wait() method, thread releases the object lock and goes for sleep. This is the main difference between wait() and sleep() methods.

wait(), notify() and notifyAll() are final methods of **java.lang.Object** class not java.lang.Thread class.

wait(), notify() and notifyAll() – all these three methods throw **IllegalMonitorStateException** if the calling thread does not owns the object lock.

wait() method is overloaded in Object class. There are two more wait() methods available in Object class. They are,

**public final void wait(long timeOut)**  —>  This makes current thread to wait until any other thread calls notify() or notifyAll() on this object or specified time(milli seconds) has elapsed.

**public final void wait(long timeOut, int nanos)**  —>  This makes current thread to wait until any other thread calls notify() or notifyAll() on this object or specified time(milli seconds + nano seconds) has elapsed.

**Interrupt ( non static method)**

Thread interruption in java**is a mechanism in which a thread which is either sleeping or waiting can be made to stop sleeping or waiting.**

The whole thread interruption mechanism depends on an internal flag called **interrupt status**. The initial value of this flag for any thread is false. When you call interrupt() method on a thread, interrupt status of that thread will be set to true. When a thread throws InterruptedException, this status will be set to false again. Remember, InterruptedException is thrown when a thread is interrupted while it is sleeping or waiting. Many methods of Thread class like sleep(), wait(), join() throw InterruptedException.

[7 Things Every Java Programmer Should Know About Threads In Java](http://javaconceptoftheday.com/7-things-every-java-programmer-should-know-about-threads-in-java/)?

1. If you start a thread that is already started, you will get java.lang.**IllegalThreadStateException** at run time. There will be no compilation errors.
2. Exception is thread wise not execution wise. i.e exception effects the thread in which it occurs. Other threads will execute normally. In the below example, exception occurs in thread t1. only this thread will be terminated abruptly. Thread t2 will continue to execute it’s task.
3. As we all know that start() method internally calls run() method. **What happens when you call run() method directly?**. When you call run() method of a thread directly, calling thread will execute the task defined in the run() method. For example, in the below program main thread is calling run() method of thread t. In this case, main thread will execute run() method not thread t.
4. Which one is better way to implement threads in java. Is it using Thread class or using Runnable interface?. It is the most confusing question for a java developer. I am of opinion that when multiple threads need to execute same task, then use Runnable interface. If multiple threads need to execute different tasks, then go for Thread class.
5. Setting the priority to a thread is not effective as we thought. Setting Priority of a thread is just an advice to OS not an instruction. It is up to OS to consider this advice.
6. A thread is a permanent member of a thread group to which it joins during creation. You can’t move a thread to a new group after creating it.

Differences Between wait () and sleep() Methods In Java?

|  |  |
| --- | --- |
| wait() | sleep() |
| The thread which calls wait() method releases the lock it holds. | The thread which calls sleep() method doesn’t release the lock it holds. |
| The thread regains the lock after other threads call either notify() or notifyAll() methods on the same lock. | No question of regaining the lock as thread doesn’t release the lock. |
| wait() method must be called within the synchronized block. | sleep() method can be called within or outside the synchronized block. |
| wait() method is a member of java.lang.Object class. | sleep() method is a member of java.lang.Thread class. |
| wait() method is always called on objects. | sleep() method is always called on threads. |
| wait() is a non-static method of Object class. | sleep() is a static method of Thread class. |
| Waiting threads can be woken up by other threads by calling notify() or notifyAll() methods. | Sleeping threads can not be woken up by other threads. If done so, thread will throw InterruptedException. |
| To call wait() method, thread must have object lock. | To call sleep() method, thread need not to have object lock. |

What is the difference between ArrayList and LinkedList ?

Ans. Underlying data structure for ArrayList is Array whereas LinkedList is the linked list and hence have following differences -  
  
1. ArrayList needs continuous memory locations and hence need to be moved to a bigger space if new elements are to be added to a filled array which is not required for LinkedList.  
  
2. Removal and Insertion at specific place in ArrayList requires moving all elements and hence leads to O(n) insertions and removal whereas its constant O(1) for LinkedList.  
  
3. Random access using index in ArrayList is faster than LinkedList which requires traversing the complete list through references.  
  
4. Though Linear Search takes Similar Time for both, Binary Search using LinkedList requires creating new Model called Binary Search Tree which is slower but offers constant time insertion and deletion.  
  
5. For a set of integers you want to sort using quicksort, it's probably faster to use an array; for a set of large structures you want to sort using selection sort, a linked list will be faster.

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1) ArrayList internally uses dynamic array to store the elements. | LinkedList internally uses doubly linked list to store the elements. |
| 2) Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is faster than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can act as a list only because it implements List only. | LinkedList class can act as a list and queue both because it implements List and Deque interfaces. |
| 4) ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |

**When to use ArrayList or LinkedList ?**

1. Adding new elements is pretty fast for either type of list. Inserting element to nth location in arraylist and to first location in linkedlist takes O(1).
2. For the ArrayList, doing random lookup using "get" is faster O(1), but for LinkedList O(n), it's slow. It's slow because there's no efficient way to index into the middle of a linked list. Linkedlist lookup always start from 1st location.
3. When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But LinkedList is fast, because deletion can be done simply by changing a couple of links.

So an ArrayList works best for cases where you're doing random access on the list and a LinkedList works better if you're doing a lot of editing in the middle of the list.

**HashSet Vs TreeSet Vs LinkedHashSet In Java ?**

Even though, **HashSet**, **LinkedHashSet** and **TreeSet** are all implementations of Set interface, there are some differences exist between them. In this article, I have tried to list out the differences between HashSet, LinkedHashSet and TreeSet in java. They also have some similarities between them. We will also discuss them at the end.

Differences Between HashSet, LinkedHashSet and TreeSet In Java :

|  |  |  |  |
| --- | --- | --- | --- |
|  | **HashSet** | **LinkedHashSet** | **TreeSet** |
| How they work internally? | HashSet uses HashMap internally to store it’s elements. | LinkedHashSet uses  LinkedHashMap internally to store it’s elements. | TreeSet uses TreeMap internally to store it’s elements. |
| Order Of Elements | HashSet doesn’t maintain any order of elements. | LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted. | TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order. |
| Performance | HashSet gives better performance than the LinkedHashSet and TreeSet. | The performance of LinkedHashSet is between HashSet and TreeSet. It’s performance is almost similar to HashSet. But slightly in the slower side as it also maintains **LinkedList** internally to maintain the insertion order of elements. | TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations. |
| Insertion, Removal And Retrieval Operations | HashSet gives performance of order O(1) for insertion, removal and retrieval operations. | LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations. | TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations. |
| How they compare the elements? | HashSet uses equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements. | LinkedHashSet also uses equals() and hashCode() methods to compare the elements. | TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn’t use equals() and hashCode() methods for comparision of elements. |
| Null elements | HashSet allows maximum one null element. | LinkedHashSet also allows maximum one null element. | TreeSet doesn’t allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException. |
| Memory Occupation | HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap internally to store its elements. | LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements. | TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap. |
| When To Use? | Use HashSet if you don’t want to maintain any order of elements. | Use LinkedHashSet if you want to maintain insertion order of elements. | Use TreeSet if you want to sort the elements according to some Comparator. |

**Similarities between HashSet, LinkedHashSet and TreeSet In Java?**

All three doesn’t allow duplicate elements.

All three are not synchronized.

All three are Cloneable and Serializable.

Iterator returned by all three is fail-fast in nature. i.e You will get **ConcurrentModificationException** if they are modified after the creation of Iterator object.

**[What changes are compatible and incompatible to the mechanism of java Serialization?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-changes-are-compatible-and-incompatible-to-the-mechanism-of-java-serialization-" \o "What changes are compatible and incompatible to the mechanism of java Serialization?)**

* This is one of a difficult and tricky questions and answering this correctly would mean you are an expert in Java Serialization concept. In an already serialized object, the most challenging task is to change the structure of a class when a new field is added or removed. As per the specifications of Java Serialization, addition of any method or field is considered to be a compatible change whereas changing of class hierarchy or non-implementation of Serializable interface is considered to be a non-compatible change. You can go through the Java serialization specification for the extensive list of compatible and non-compatible changes. If a serialized object need to be compatible with an older version, it is necessary that the newer version follows some rules for compatible and incompatible changes. A compatible change to the implementing class is one that can be applied to a new version of the class, which still keeps the object stream compatible with older version of same class.

**Some Simple Examples of compatible changes are:**

* Addition of a new field or class will not affect serialization, since any new data in the stream is simply ignored by older versions. the newly added field will be set to its default values when the object of an older version of the class is un marshaled.
* The access modifiers change (like private, public, protected or default) is compatible since they are not reflected in the serialized object stream.
* Changing a transient field to a non-transient field is compatible change since it is similar to adding a field.
* Changing a static field to a non-static field is compatible change since it is also similar to adding a field.

**Some Simple Examples of incompatible changes are:**

* Changing implementation from Serializable to Externalizable interface can not be done since this will result in the creation of an incompatible object stream.
* Deleting a existing Serializable fields will cause a problem.
* Changing a non-transient field to a transient field is incompatible change since it is similar to deleting a field.
* Changing a non-static field to a static field is incompatible change since it is also similar to deleting a field.
* Changing the type of a attribute within a class would be incompatible, since this would cause a failure when attempting to read and convert the original field into the new field.
* Changing the package of class is incompatible. Since the fully-qualified class name is written as part of the object byte stream.

Java serialization is one of the most commonly misunderstood areas. Many developers still think it’s only used for saving objects on the file system.

## [Is it possible to customize the serialization process? How can we customize the Serialization process?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "is-it-possible-to-customize-the-serialization-process--how-can-we-customize-the-serialization-process-" \o "Is it possible to customize the serialization process? How can we customize the Serialization process?)

Yes, the serialization process can be customized. When an object is serialized, objectOutputStream.writeObject (to save this object) is invoked and when an object is read, ObjectInputStream.readObject () is invoked. What most people do not know is that Java Virtual Machine provides you with an option to define these methods as per your needs. Once this is done, these two methods will be invoked by the JVM instead of the application of the default serialization process. Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

private void writeObject(java.io.ObjectOutputStream out)

throws IOException

private void readObject(java.io.ObjectInputStream in)

throws IOException, ClassNotFoundException;

private void readObjectNoData()

throws ObjectStreamException;

## [What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-the-alternatives-to-serialization--if-serialization-is-not-used--is-it-possible-to-persist-or-transfer-an-object-using-any-other-approach-" \o "What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?)

In case, Serialization is not used, Java objects can be serialized by many ways, some of the popular methods are listed below:

* Saving object state to database, this is most common technique used by most applications. You can use ORM tools (e.g. hibernate) to save the objects in a database and read them from the database.
* Xml based data transfer is another popular mechanism, and a lot of XML based web services use this mechanism to transfer data over network. Also a lot of tools save XML files to persist data/configurations.
* JSON Data Transfer - is recently popular data transfer format. A lot of web services are being developed in JSON due to its small footprint and inherent integration with web browser due to JavaScript format.

## [When will you use Serializable or Externalizable interface? and why?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "when-will-you-use-serializable-or-externalizable-interface--and-why-" \o "When will you use Serializable or Externalizable interface? and why?)

Most of the times when you want to do a selective attribute serialization you can use Serializable interface with transient modifier for variables not to be serialized. However, use of Externalizable interface can be really effective in cases when you have to serialize only some dynamically selected attributes of a large object.   
  
Lets take an example, Some times when you have a big Java object with hundreds of attributes and you want to serialize only a dozen dynamically selected attributes to keep the state of the object you should use Externalizable interface writeExternal method to selectively serialize the chosen attributes.  
  
In case you have small objects and you know that most or all attributes are required to be serialized then you should be fine with using Serializable interface and use of transient variable as appropriate.

**Serialization and SerialVersionUID?**

To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object. A Java object is serializable if its class or any of its super classes implements either the java.io.Serializable interface or its sub interface, java.io.Externalizable.

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in an InvalidClassException.

A serializable class can declare its own serialVersionUID explicitly by declaring a field named "**serialVersionUID**" that must be static, final, and of type long:ANY-ACCESS-MODIFIER

**static final long serialVersionUID = 42L;**

If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class, as described in the Java(TM) Object Serialization Specification. However, it is strongly recommended that all serializable classes explicitly declare serialVersionUID values, since the default serialVersionUID computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected **InvalidClassExceptions** during deserialization. Therefore, to guarantee a consistent serialVersionUID value across different java compiler implementations, a serializable class must declare an explicit serialVersionUID value. It is also strongly advised that explicit serialVersionUID declarations use the private modifier where possible, since such declarations apply only to the immediately declaring class--serialVersionUID fields are not useful as inherited members.

**What is the use of hashcode in Java ?**

Ans. Hashcode is used for bucketing in Hash implementations like HashMap, HashTable, HashSet etc. The value received from hashcode() is used as bucket number for storing elements. This bucket number is the address of the element inside the set/map. when you do contains () then it will take the hashcode of the element, then look for the bucket where hashcode points to and if more than 1 element is found in the same bucket (multiple objects can have the same hashcode) then it uses the equals() method to evaluate if object are equal, and then decide if contain() is true or false, or decide if element could be added in the set or not.   
 **Difference between SAX and DOM Parser?**

A DOM (Document Object Model) parser creates a tree structure in memory from an input document whereas A SAX (Simple API for XML) parser does not create any internal structure.  
  
A SAX parser serves the client application always only with pieces of the document at any given time whereas A DOM parser always serves the client application with the entire document no matter how much is actually needed by the client.  
  
A SAX parser, however, is much more space efficient in case of a big input document whereas DOM parser is rich in functionality.  
  
Use a DOM Parser if you need to refer to different document areas before giving back the information. Use SAX is you just need unrelated nuclear information from different areas.  
  
Xerces, Crimson are SAX Parsers whereas XercesDOM, SunDOM, Oracle DOM are DOM parsers.

**[Is Java Pass by Reference or Pass by Value?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "is-java-pass-by-reference-or-pass-by-value-" \o "Is Java Pass by Reference or Pass by Value?)**

The Java Spec says that everything in Java is pass-by-value. There is no such thing as "pass-by-reference" in Java. The difficult thing can be to understand that Java passes "objects as references" passed by value.

[**What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?**](http://www.fromdev.com/2012/02/java-interview-question-answer.html#what-is-outofmemoryerror-in-java--how-to-deal-with-java-lang-outofmemeryerror--error-)

This Error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

Note: Its an Error (extends java.lang.Error) not Exception.

Two important types of OutOfMemoryError are often encountered

java.lang.OutOfMemoryError: Java heap space

The quick solution is to add these flags to JVM command line when Java runtime is started:

**-Xms1024m -Xmx1024m**

java.lang.OutOfMemoryError: PermGen space

The solution is to add these flags to JVM command line when Java runtime is started:

-**XX:+CMSClassUnloadingEnabled-XX:+CMSPermGenSweepingEnabled**

**Long Term Solution**: Increasing the Start/Max Heap size or changing Garbage Collection options may not always be a long term solution for your Out Of Memory Error problem. Best approach is to understand the memory needs of your program and ensure it uses memory wisely and does not have leaks. You can use a Java memory profiler to determine what methods in your program are allocating large number of objects and then determine if there is a way to make sure they are no longer referenced, or to not allocate them in the first place.

**[What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-the-use-of-the-finally-block--is-finally-block-in-java-guaranteed-to-be-called--when-finally-block-is-not-called-" \o "What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?)**

Finally is the block of code that executes always. The code in finally block will execute even if an exception is occurred. Finally block is NOT called in following conditions

If the JVM exits while the try or catch code is being executed, then the finally block may not execute. This may happen due to System.exit() call.

if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.

If a exception is thrown in finally block and not handled then remaining code in finally block may not be executed.

**[Why there are two Date classes; one in java.util package and another in java.sql?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-there-are-two-date-classes--one-in-java-util-package-and-another-in-java-sql-" \o "Why there are two Date classes; one in java.util package and another in java.sql?)**

From the JavaDoc of **java.sql.Date**:

A thin wrapper around a millisecond value that allows JDBC to identify this as an SQL DATE value. A milliseconds value represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT. To conform with the definition of SQL DATE, the millisecond values wrapped inside a java.sql.Date instance must be 'normalized' by setting the hours, minutes, seconds, and milliseconds to zero.

**Explanation**: A java.util.Date represents date and time of day, a java.sql.Date only represents a date (the complement of java.sql.Date is java.sql.Time, which only represents a time of day, but also extends **java.util.Date).**

**[What is Marker interface? How is it used in Java?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-marker-interface--how-is-it-used-in-java-" \o "What is Marker interface? How is it used in Java?)**

**The marker interface is a design pattern**, used with languages that provide run-time type information about objects. It provides a way to associate metadata with a class where the language does not have explicit support for such metadata. **To use this pattern, a class implements a marker interface, and code that interact with instances of that class test for the existence of the interface**. **Whereas a typical interface specifies methods that an implementing class must support, a marker interface does not do so**. The mere presence of such an interface indicates specific behavior on the part of the implementing class. There can be some hybrid interfaces, which act as markers and specify required methods, are possible but may prove confusing if improperly used. Java utilizes this pattern very well and the example interfaces are

**java.io.Serializable** - Serializability of a class is enabled by the class implementing the java.io.Serializable interface. The Java Classes that do not implement Serializable interface will not be able to serialize or deserialize their state. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

**java.rmi.Remote -** The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a "remote interface", an interface that extends java.rmi.Remote are available remotely.

**java.lang.Cloneable** - A class implements the Cloneable interface to indicate to the Object.clone () method that it is legal for that method to make a field-for-field copy of instances of that class. Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception **CloneNotSupportedException** being thrown.

**javax.servlet.SingleThreadModel -** Ensures that servlets handle only one request at a time. This interface has no methods.

**java.util.EvenListener** - A tagging interface that all event listener interfaces must extend. The "**instanceof**" keyword in java can be used to test if an object is of a specified type. So this keyword in combination with Marker interface can be used to take different actions based on type of interface an object implements.

**[Why main() in java is declared as public static void main? What if the main method is declared as private?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-main---in-java-is-declared-as-public-static-void-main-" \o "Why main() in java is declared as public static void main?)**

Public - main method is called by JVM to run the method which is outside the scope of project therefore the access specifier has to be public to permit call from anywhere outside the application static - When the JVM makes are call to the main method there is not object existing for the class being called therefore it has to have static method to allow invocation from class. void - Java is platform independent language therefore if it will return some value then the value may mean different to different platforms so unlike C it can not assume a behavior of returning value to the operating system. If main method is declared as private then - Program will compile properly but at run-time it will give "Main method not public." error.

**Why is it preferred to declare: List<String> list = new ArrayList<String>(); instead of ArrayList<String> = new ArrayList<String>();**

Ans) It is preferred because:

1. If later on code needs to be changed from ArrayList to Vector then only at the declaration place we can do that.
2. The most important one – If a function is declared such that it takes list. E.g void showDetails(List list);  
   When the parameter is declared as List to the function it can be called by passing any subclass of List like ArrayList, Vector, LinkedList making the function more flexible.

**Which data structure HashSet implements ?**

Ans) HashSet implements hashmap internally to store the data. The data passed to hashset is stored as key in hashmap with null as value.

**What is the difference between iterator access and index access?**

Index based access allow access of the element directly on the basis of index. The cursor of the data structure can directly goto the 'n' location and get the element. It does not traverse through n-1 elements.

In Iterator based access, the cursor has to traverse through each element to get the desired element. So to reach the 'n'th element it needs to traverse through n-1 elements.

Insertion,updation or deletion will be faster for iterator based access if the operations are performed on elements present in between the data structure.

Insertion,updation or deletion will be faster for index based access if the operations are performed on elements present at last of the data structure.

Traversal or search in index based data structure is faster.

ArrayList is index access and LinkedList is iterator access.

**How to sort a list in reverse order?**

To sort the elements in the reverse natural order of the strings, get a reverse Comparator from the Collections class with reverse Order(). Then, pass the reverse Comparator to the sort() method.

List list = new ArrayList();

Comparator comp = Collections.reverseOrder();

Collections.sort(list, comp)

**Can a null element be added to a Treeset or HashSet ?**

Ans) A null element can be added only if the set is of size 1 because when a second element is added then as per set defination a check is made to check duplicate value and comparison with null element will throw NullPointerException.  
HashSet is based on hashMap and can contain null element.

**How to sort list of strings - case insensitive?**

using Collections.sort(list, String.CASE\_INSENSITIVE\_ORDER);

**How to make a List (ArrayList, Vector,LinkedList) read only?**

A list implemenation can be made read only using **Collections.unmodifiableList(list)**.

This method returns a new list. If a user tries to perform add operation on the new list; **UnSupportedOperationException** is thrown.

**Which is faster to iterate LinkedHashSet or LinkedList?**

LinkedList.

**Arrange in the order of speed - HashMap, HashTable, Collections.synchronizedMap,concurrentHashmap?**

HashMap is fastest, ConcurrentHashMap,Collections.synchronizedMap,HashTable.

**What is IdentityHashMap?**

The IdentityHashMap uses == for equality checking instead of equals(). This can be used for both performance reasons, if you know that two different elements will never be equals and for preventing spoofing, where an object tries to imitate another.

**What is WeakHashMap?**

A hashtable-based Map implementation with weak keys. An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely, the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector, that is, made finalizable, finalized, and then reclaimed. When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently than other Map implementations.

**Hibernate:**

* 1. **Hibernate vs iBATIS - Comparison of two powerful ORM tools**

Hibernate and iBatis both are open source Object Relational Mapping (ORM) tools available in the industry. Both of these tools are very powerful and use of each of these depends on the context. Below are few criteria we can keep in our mind while choosing one over other.  
  
Within the java persistence there is no one size, fits all solution. Hibernate is most commonly used ORM tool.  
**1. Way of working**

Hibernate maps your Java based POJO objects to corresponding Database tables where as iBatis maps the ResultSet from JDBC API to Java POJO Objets. So in case of iBatis you may need to pass your parameters to the DAO class in a map object which can be passed to the Statement.  
2. **XML mappings**

Hibernate and iBatis both use XML based mapping and are quite intuitive to implement. Tools are available to create and edit these file. e.g. [Abator for iBatis](http://ibatis.apache.org/docs/tools/abator/) and [Hibernate tools](http://www.hibernate.org/255.html).  
3. **Project from Scratch with new Database or Existing Database**

Hibernate works well when you control the data model, iBATIS works well when you need to integrate with an existing database. In the projects where the Project is being developed from scratch and developer has to come up with database design along with the Java object model then Hibernate is a good option as it becomes easy to develop and mappings becomes intuitive. Hibernate also provides ways to generate database from mappings files itself. Which may reduce your deployment script generation efforts.  
 Using iBatis for a project which is developed from scratch is also very simple and does not involve lot of efforts.  
4. **Data centric project**

A project that needs database intensive operations and needs to use lot of existing implementations that are available in Database side then iBatis provides simple ways to leverage it. Hibernate also provides way to handle such situations but it gets complex as your application becomes more and more complex. The hibernate developer need to start understanding lot of complex things to accomplish it.  
**5. Hibernate Query Language (HQL) vs. Structured Query Language (SQL)**

Its another choice which you may want to take in case your application involves lot of complex database intensive operations then in Hibernate you may want to go for Hibernate Query Language (HQL) which will have following aspects Learning curve for developers

You will not be able to see the actual SQL because Hibernate generates it internally, though there is a way to see what is being generated but you lose the direct control over it.

- Abstraction of SQL makes it difficult for the developers visualize what is happening at that layer. Where as in case of iBatis you have same SQL available to you in the XML mapping file and you can play around with it as you wish, which gives you lot of freedom to tune in performance of queries.

6**. Simple Applications**

In case your application is simple with not lot of complex data processing then Hibernate could be a decent option. Also if a developer does not like SQL much then it could be a good option too.

**7. SPRING Integration**

Hibernate and iBatis both have good support from SPRING framework so it should not be a problem to choose one of them.

* 1. **Session.load() vs Session.get()**

**session.load()**

It will always return a “proxy” (Hibernate term) without hitting the database. In Hibernate, proxy is an object with the given identifier value, its properties are not initialized yet, it just look like a temporary fake object.

If no row found, it will throws an ObjectNotFoundException.

**session.get()**

It always hit the database and return the real object, an object that represent the database row, not proxy.

If no row found , it return null.

Hibernate create anything for some reasons, when you do the association, it’s normal to obtain retrieve an object (persistent instance) from database and assign it as a reference to another object, just to maintain the relationship. Let’s go through some examples to understand in what situation you should use **session.load().**

**session.load()**

In above scenario, **session.load()** will be your good solution, let’s see the example,

Stock stock = (Stock)session.load(Stock.class, new Integer(2));

StockTransaction stockTransactions = new StockTransaction();

//set stockTransactions detail

stockTransactions.setStock(stock);

session.save(stockTransactions);

Output

Hibernate:

insert into mkyong.stock\_transaction (...) values (?, ?, ?, ?, ?, ?)

In session.load(), Hibernate will not hit the database (no select statement in output) to retrieve the Stock object, it will return a Stock proxy object – a fake object with given identify value. In this scenario, a proxy object is enough for to save a stock transaction record.

Exception

In exception case, see the examples

**session.load()**

Stock stock = (Stock)session.load(Stock.class, new Integer(100)); //proxy

//initialize proxy, no row for id 100, throw ObjectNotFoundException

System.out.println(stock.getStockCode());

It will always return a proxy object with the given identity value, even the identity value is not exists in database. However, when you try to initialize a proxy by retrieve it’s properties from database, it will hit the database with select statement. If no row is found, a **ObjectNotFoundException** will throw.

org.hibernate.ObjectNotFoundException: No row with the given identifier exists:

[com.mkyong.common.Stock#100]

**session.get()**

//return null if not found

Stock stock = (Stock)session.get(Stock.class, new Integer(100));

System.out.println(stock.getStockCode()); //java.lang.NullPointerException

It will always return null , if the identity value is not found in database

**Hibernate *save*** can be used to save entity to database. We can invoke this method outside a transaction, that’s why I don’t like this method to save data. If we use this without transaction and we have cascading between entities, then only the primary entity gets saved unless we flush the session.

**Hibernate *persist*** is similar to save (with transaction) and it adds the entity object to the persistent context, so any further changes are tracked. If the object properties are changed before the transaction is committed or session is flushed, it will also be saved into database.

Second difference is that we can use persist () method only within the boundary of a transaction, so it’s safe and takes care of any cascaded objects.

**Hibernate *saveOrUpdate*** results into insert or update queries based on the provided data. If the data is present in the database, update query is executed. We can use saveOrUpdate without transaction also, but again you will face the issues with mapped objects not getting saved if session is not flushed. Hibernate saveOrUpdate adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist.

**Hibernate update** should be used where we know that we are only updating the entity information. This operation adds the entity object to persistent context and further changes are tracked and saved when transaction is committed.

**Hibernate merge** can be used to update existing values, however this method create a copy from the passed entity object and return it. The returned object is part of persistent context and tracked for any changes, passed object is not tracked.

**What is Lazy/Select Fetch / Eagar Fetch in Hibernate?**

***Lazy Fetching****:* This is the default Fetch type of **hibernate3**. Select fetch strategy is the lazy fetching of associations. The purpose of lazy strategy is memory optimization. When I say memory optimization that means it saves us from heap error. This is what think. So we can say yes if we are loading too objects in a session we should go for lazy fetching strategy but in terms of time performance it doesn’t provide any benefit.

***Eagar/Join Fetching****: In hibernate 2, this is the default of retrieving object from the database.*

*The purpose* of join fetch strategy is optimization in terms of time. I mean associations are fetched right at the time of fetching parent object. In this case we don’t make database call again and again. So, this will be much faster. Agreed that this will bad if we are fetching too many objects in session because we can get java heap error.

**What is cascading and what are different types of cascading?**

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

import org.hibernate.annotations.Cascade;

@Entity

.

.

@Table(name = "EMPLOYEE")

public class Employee {

s

@OneToOne(mappedBy = "employee")

@Cascade(value = org.hibernate.annotations.CascadeType.ALL)

private Address address;

}

Note that Hibernate CascadeType enum constants are little bit different from JPA javax.persistence.CascadeType, so we need to use the Hibernate CascadeType and Cascade annotations for mappings, as shown in above example.  
Commonly used cascading types as defined in CascadeType enum are:

**None**: No Cascading, it’s not a type but when we don’t define any cascading then no operations in parent affects the child.

ALL: Cascades save, delete, update, evict, lock, replicate, merge, and persist. Basically everything

SAVE\_UPDATE: Cascades save and update, available only in hibernate.

DELETE: Corresponds to the Hibernate native DELETE action, only in hibernate.

DETATCH, MERGE, PERSIST, REFRESH and REMOVE – for similar operations

LOCK: Corresponds to the Hibernate native LOCK action.

REPLICATE: Corresponds to the Hibernate native REPLICATE action.

**Which design patterns are used in Hibernate framework?**

Some of the design patterns used in Hibernate Framework are:

**Domain Model Pattern** – An object model of the domain that incorporates both behavior and data.

**Data Mapper** – A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.

[**Proxy Pattern**](http://www.journaldev.com/1572/proxy-design-pattern) for lazy loading

[**Factory pattern**](http://www.journaldev.com/1392/factory-design-pattern-in-java)in SessionFactory

**What are the Core interfaces of Hibernate Framework?**

Session, SessionFactory, Configuration, Transaction,

Query and Criteria interfaces.

**How Hibernate is database independent explain?**

Only changing the property in XML

<property name=”hibernate.dialect”>org.hibernate.dialect.Oracle9Dialect</property>

<property name=”hibernate.connection.driver\_class”>oracle.jdbc.driver.OracleDriver</property>

**What is the hibernate proxy?**

An object is just a way to avoid retrieving an object until you need it. Hibernate2 does not proxy objects by default.

**How do you create a session factory in hibernate?**

Configuration cfg=new Configuration();

cfg.addResource(“dir/hibernate.hbm.xml”);

cfg.setProperties(System.getProperties());

SessionFactory sessions=cfg.buildSessionFactory();

**What is a Thin Client?**

A thin client is a program interface to the application that does not have any operation like query of databases, execute complex business rules, or connect to legacy applications.

**What are the key benifits of Hibernate?**

There are several benefits of using Hibernate

Powerful object-oriented hibernate query language

Transparent persistence based on POJOs without byte code processing

Descriptive O/R Mapping through mapping file.

Automatic primary key generation

Hibernate cache : Session Level, Query and Second level cache.

Performance: Lazy initialization, Outer join fetching, Batch fetching

**What is hibernate session and session factory? How do you configure sessionfactory in spring configuration file?**

Hibernate Session is the main runtime interface between a Java application and Hibernate. SessionFactory allows applications to create hibernate session by reading hibernate configurations file hibernate.cfg.xml.

    Configuration cfg = new Configuration().configure();

// Initialize the Hibernate environment

    SessionFactory factory = cfg.buildSessionFactory();

// Create the session factory

    Session session = factory.openSession();

 // Obtain the new session object

The call to Configuration().configure() loads the hibernate.cfg.xml configuration file and initializes the Hibernate environment. Once the configuration is initialized, you can make any additional modifications you desire programmatically. However, you must make these modifications prior to creating the SessionFactory instance. An instance of SessionFactory is typically created once and used to create all sessions related to a given context.

The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of three states:

**transient**: never persistent, not associated with any Session

**persistent**: associated with a unique Session

**detached**: previously persistent, not associated with any Session

A Hibernate Session object represents a single unit-of-work for a given data store and is opened by a SessionFactory instance. You must close Sessions when all work for a transaction is completed. The following illustrates a typical Hibernate session:

    Session session = null;

    UserInfo user = null;

    Transaction tx = null;

    try {

       session = factory.openSession();

       tx = session.beginTransaction();

       user = (UserInfo)session.load(UserInfo.class, id);

       tx.commit();

    } catch(Exception e) {

       if (tx != null) {

          try {

             tx.rollback();

          } catch (HibernateException e1) {

             throw new DAOException(e1.toString()); }

       } throw new DAOException(e.toString());

    } finally {

       if (session != null) {

          try {

             session.close();

          } catch (HibernateException e) { }

       }

    }

**What is the difference between hibernate get and load methods?**

The following Hibernate code snippet retrieves a User object from the database:

    User user = (User) session.get(User.class, userID);

The get() method is special because the identifier uniquely identifies a single instance of a class. Hence it’s common for applications to use the identifier as a convenient handle to a persistent object. Retrieval by identifier can use the cache when retrieving an object, avoiding a database hit if the object is already cached. The get() method returns null if the object can’t be found.

Hibernate also provides a load() method:

    User user = (User) session.load(User.class, userID);

If load() can’t find the object in the cache or database, an exception is thrown. The load() method never returns null.  The load() method may return a proxy instead of a real persistent instance. A proxy is a placeholder instance of a runtime-generated subclass (through cglib or Javassist) of a mapped persistent class, it can initialize itself if any method is called that is not the mapped database identifier getter-method.

On the other hand, get() never returns a proxy. Choosing between get() and load() is easy: If you’re certain the persistent object exists, and nonexistence would be considered exceptional, load() is a good option. If you aren’t certain there is a persistent instance with the given identifier, use get() and test the return value to see if it’s null. Using load() has a further implication: The application may retrieve a valid reference (a proxy) to a persistent instance without hitting the database to retrieve its persistent state. So load() might not throw an exception when it doesn’t find the persistent object in the cache or database; the exception would be thrown later, when the proxy is accessed.

**What type of transaction management is supported in hibernate?**

Hibernate communicates with the database via a JDBC Connection; hence it must support both managed and non-managed transactions.

Non-managed in web containers managed in application server using JTA:

    <bean id="transactionManager" class="org.springframework.orm.hibernate.HibernateTransactionManager">

        <property name="sessionFactory">

            <ref local="sessionFactory"/>

        </property>

    </bean>

    <bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager.">

        <property name="sessionFactory">

            <ref local="sessionFactory"/>

        </property>

    </bean>

**What is lazy loading and how do you achieve that in hibernate?**

Lazy setting decides whether to load child objects while loading the Parent Object. You need to specify parent class.Lazy = true in hibernate mapping file. By default the lazy loading of the child objects is true. This make sure that the child objects are not loaded unless they are explicitly invoked in the application by calling getChild() method on parent. In this case hibernate issues a fresh database call to load the child when getChild() is actully called on the Parent object. But in some cases you do need to load the child objects when parent is loaded. Just make the lazy=false and hibernate will load the child when parent is loaded from the database.

Examples: Address child of User class can be made lazy if it is not required frequently. But you may need to load the Author object for Book parent whenever you deal with the book for online bookshop.

Hibernate does not support lazy initialization for detached objects. Access to a lazy association outside of the context of an open Hibernate session will result in an exception.

**What are the different fetching strategy in Hibernate?**

Hibernate3 defines the following fetching strategies:

**Join fetching** - Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.

**Select fetching** - a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

**Subselect fetching** - a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

**Batch fetching** - an optimization strategy for select fetching - Hibernate retrieves a batch of entity instances or collections in a single SELECT, by specifying a list of primary keys or foreign keys.

**What are different types of cache hibernate supports ?**

Caching is widely used for optimizing database applications. Hibernate uses two different caches for objects:

**First-level cache**: First-level cache is associated with the Session objectBy default, Hibernate uses first-level cache on a per-transaction basis. Hibernate uses this cache mainly to reduce the number of SQL queries it needs to generate within a given transaction. For example, if an object is modified several times within the same transaction, Hibernate will generate only one SQL UPDATE statement at the end of the transaction, containing all the modifications.

**Second-level cache**: second-level cache is associated with the Session Factory object. To reduce database traffic, second-level cache keeps loaded objects at the Session Factory level between transactions. These objects are available to the whole application, not just to the user running the query. This way, each time a query returns an object that is already loaded in the cache, one or more database transactions potentially are avoided.

**Query-level cache**: In addition, you can use a query-level cache if you need to cache actual query results, rather than just persistent objects. The query cache should always be used in conjunction with the second-level cache. Hibernate supports the following open-source cache implementations out-of-the-box:

**EHCache** is a fast, lightweight, and easy-to-use in-process cache. It supports read-only and read/write caching, and memory- and disk-based caching. However, it does not support clustering.

**OSCache** is another open-source caching solution. It is part of a larger package, which also provides caching functionalities for JSP pages or arbitrary objects. It is a powerful and flexible package, which, like EHCache, supports read-only and read/write caching, and memory- and disk-based caching. It also provides basic support for clustering via either JavaGroups or JMS.

**SwarmCache** is a simple cluster-based caching solution based on JavaGroups. It supports read-only or nonstrict read/write caching (the next section explains this term). This type of cache is appropriate for applications that typically have many more read operations than write operations.

JBoss TreeCache is a powerful replicated (synchronous or asynchronous) and transactional cache. Use this solution if you really need a true transaction-capable caching architecture.

**Commercial Tangosol Coherence cache**.

**What are the different caching strategies?**

**Read-only**: This strategy is useful for data that is read frequently but never updated. This is by far the simplest and best-performing cache strategy.

**Read/write**: Read/write caches may be appropriate if your data needs to be updated. They carry more overhead than read-only caches. In non-JTA environments, each transaction should be completed when Session.close() or Session.disconnect() is called.

**Nonstrict read/write**: This strategy does not guarantee that two transactions won't simultaneously modify the same data. Therefore, it may be most appropriate for data that is read often but only occasionally modified.

**Transactional**: This is a fully transactional cache that may be used only in a JTA environment.

**How do you configure 2nd level cach in hibernate?**

To activate second-level caching, you need to define the hibernate.cache.provider\_class property in the hibernate.cfg.xml file as follows:

    <hibernate-configuration>

        <session-factory>

            <property name="hibernate.cache.provider\_class">org.hibernate.cache.EHCacheProvider</property>

        </session-factory>

    </hibernate-configuration>

By default, the second-level cache is activated and uses the EHCache provider.

To use the query cache you must first enable it by setting the property hibernate.cache.use\_query\_cache to true in hibernate.properties.

**What is the difference between sorted and ordered collection in hibernate?**

A sorted collection is sorted in-memory using java comparator, while order collection is ordered at the database level using order by clause.

**What are the types of inheritence models and describe how they work like vertical inheritence and horizontal?**

There are three types of inheritance mapping in hibernate:

Example: Let us take the simple example of 3 java classes. Class Manager and Worker are inherited from Employee Abstract class.

**Table per concrete class with unions**: In this case there will be 2 tables. Tables: Manager, Worker [all common attributes will be duplicated]

**Table per class hierarchy:** Single Table can be mapped to a class hierarchy. There will be only one table in database called 'Employee' that will represent all the attributes required for all 3 classes. But it needs some discriminating column to differentiate between Manager and worker;

**Table per subclass**: In this case there will be 3 tables represent Employee, Manager and Worker.

**What is the difference between the session.get() method and the session.load() method?**

Both the session.get(..) and session.load() methods create a persistent object by loading the required object from the database. But if there was not such object in the database then the method session.load(..) throws an exception whereas session.get(&) returns null.

**What is the difference between the session.update() method and the session.lock() method?**

Both of these methods and saveOrUpdate() method are intended for reattaching a detached object. The session.lock() method simply reattaches the object to the session without checking or updating the database on the assumption that the database in sync with the detached object. It is the best practice to use either session.update(..) or session.saveOrUpdate(). Use session.lock() only if you are absolutely sure that the detached object is in sync with your detached object or if it does not matter because you will be overwriting all the columns that would have changed later on within the same transaction.

Note: When you reattach detached objects you need to make sure that the dependent objects are reatched as well.

How would you reatach detached objects to a session when the same object has already been loaded into the session?

You can use the session.merge() method call.

**What are the general considerations or best practices for defining your Hibernate persistent classes?**

You must have a default no-argument constructor for your persistent classes and there should be getXXX() (i.e accessor/getter) and setXXX( i.e. mutator/setter) methods for all your persistable instance variables.

You should implement the equals() and hashCode() methods based on your business key and it is important not to use the id field in your equals() and hashCode() definition if the id field is a surrogate key (i.e. Hibernate managed identifier). This is because the Hibernate only generates and sets the field when saving the object.

It is recommended to implement the Serializable interface. This is potentially useful if you want to migrate around a multi-processor cluster.

The persistent class should not be final because if it is final then lazy loading cannot be used by creating proxy objects.

Use XDoclet tags for generating your \*.hbm.xml files or Annotations (JDK 1.5 onwards), which are less verbose than \*.hbm.xml files.

**How will you configure Hibernate?**

The configuration files hibernate.cfg.xml (or hibernate.properties) and mapping files \*.hbm.xml are used by the Configuration class to create (i.e. configure and bootstrap hibernate) the SessionFactory, which in turn creates the Session instances. Session instances are the primary interface for the persistence service.   
  
" hibernate.cfg.xml (alternatively can use hibernate.properties): These two files are used to configure the hibernate sevice (connection driver class, connection URL, connection username, connection password, dialect etc). If both files are present in the classpath then hibernate.cfg.xml file overrides the settings found in the hibernate.properties file.   
  
" Mapping files (\*.hbm.xml): These files are used to map persistent objects to a relational database. It is the best practice to store each object in an individual mapping file (i.e mapping file per class) because storing large number of persistent classes into one mapping file can be difficult to manage and maintain. The naming convention is to use the same name as the persistent (POJO) class name. For example Account.class will have a mapping file named Account.hbm.xml. Alternatively hibernate annotations can be used as part of your persistent class code instead of the \*.hbm.xml files.   
  
**What is a SessionFactory? Is it a thread-safe object?**  
  
SessionFactory is Hibernates concept of a single datastore and is threadsafe so that many threads can access it concurrently and request for sessions and immutable cache of compiled mappings for a single database. A SessionFactory is usually only built once at startup. SessionFactory should be wrapped in some kind of singleton so that it can be easily accessed in an application code.   
  
SessionFactory sessionFactory = new Configuration().configure().buildSessionfactory();   
  
**What is a Session? Can you share a session object between different theads?**   
  
Session is a light weight and a non-threadsafe object (No, you cannot share it between threads) that represents a single unit-of-work with the database. Sessions are opened by a SessionFactory and then are closed when all work is complete. Session is the primary interface for the persistence service. A session obtains a database connection lazily (i.e. only when required). To avoid creating too many sessions ThreadLocal class can be used as shown below to get the current session no matter how many times you make call to the currentSession() method.   
  
&   
public class HibernateUtil {   
&   
public static final ThreadLocal local = new ThreadLocal();   
  
public static Session currentSession() throws HibernateException {   
Session session = (Session) local.get();   
//open a new session if this thread has no session   
if(session == null) {   
session = sessionFactory.openSession();   
local.set(session);   
}   
return session;   
}   
}   
  
It is also vital that you close your session after your unit of work completes. Note: Keep your Hibernate Session API handy.   
  
**What are the benefits of detached objects?**   
  
 Detached objects can be passed across layers all the way up to the presentation layer without having to use any DTOs (Data Transfer Objects). You can later on re-attach the detached objects to another session.   
  
**What are the pros and cons of detached objects?**   
   
Pros:

" When long transactions are required due to user think-time, it is the best practice to break the long transaction up into two or more transactions. You can use detached objects from the first transaction to carry data all the way up to the presentation layer. These detached objects get modified outside a transaction and later on re-attached to a new transaction via another session.  
Cons   
  
" In general, working with detached objects is quite cumbersome, and better to not clutter up the session with them if possible. It is better to discard them and re-fetch them on subsequent requests. This approach is not only more portable but also more efficient because - the objects hang around in Hibernate's cache anyway.   
  
" Also from pure rich domain driven design perspective it is recommended to use DTOs (DataTransferObjects) and DOs (DomainObjects) to maintain the separation between Service and UI tiers.

STRUTS

**Easy to remember the differences between Struts 1. x and 2. x:**

|  |  |  |
| --- | --- | --- |
|  | **Struts 1.x** | **Struts 2.x** |
| **Configuration file** | Named as struts-config.xml | Named as struts.xml |
| **Tag Library** | Three tag libraries are required:  <%@ taglib uri = "/WEB-INF/struts-html.tld" prefix = "html" %>    <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix = "logic" %>    <%@ taglib uri = "/WEB-INF/struts-bean.tld" prefix = "bean" %> | Only on tag library encapsulates all the features:  <%@ taglib uri = "/struts-tags"  prefix = "s" %> |
|  | Thread-safe methods should be declared. | Thread-safety is not an issue. |
| **Co-ordination of Beans and Action classes** | Java beans and action classes are separately defined | Getter-setter methods can be added in action classes. |
| **Servlet dependency** | Servlet dependent | Servlet independent |
| **Extending and implementing actions** | Action classes need to extend org.apache.struts.action.Action | Action classes can implement Action interface or can also extend ActionSupport class |
| **Execute() method** | Action classes have to provide body of execute() method | Implementing execute() method is not a compulsion. |
| **Front end** | Front end  is controlled by ActionServlet | Front end  is controlled by Filters |
| **Expression language** | JSTL’s EL (Expression language) is used here | For EL, OGNL (Object Graph Notation Language) is used. |
| **Validations** | Validations are to be performed in ActionForm manually | Validations are handled by XWork Validator which provides validations of certain fields automatically. |
| **Life cycle** | All the request handlers follow same life cycle. | interceptors in struts 2.x does not follow same life cycle. |

**What are interceptors and why to use them?**

The main use of interceptors is to pre-process or post-process a module in a web-site. That is, any pre-processing that is required before the action tag is executed from struts.xml and any post-processing required after the action tag is executed from struts.xml is specified by the interceptors.

Interceptors get executed before and after the action elements are called from struts.xml file.

The main purpose of using interceptors is to provide file validation, as we have seen above, validation of fields in our web page, logging, storing, handling exceptions, setting timers (time-out for some event) etc.

In nut-shell: Web-page restrictions and validations from user as well as server side can be provided by interceptors.

Also, a web page can have multiple interceptors. That is, n number of validations (pre as well as post) can be applied to a web form for different fields.

Types of interceptors in Struts 2:

The types of interceptors in Struts 2 are:

In-built

Custom

Inbuilt interceptors are predefined in struts configuration files. You just have to use it in action element when required. The types of inbuilt interceptors are:

fileUpload

execption

alias

checkbox

chain

datetimes

multiselect

i18n

params

Validation

conversionError

workflow

execAndWait

Application server and web server in Java?

 both are used to host Java web application. Though both application server and web server are generic terms, difference between application server and web server is a [famous J2EE interview question](http://javarevisited.blogspot.com/2011/09/servlet-interview-questions-answers.html). On  Java J2EE perspective main difference between web server and application server is support of EJB. In order to run EJB or host enterprise Java application (.ear) file you need an application server like JBoss, WebLogic, WebSphere or Glassfish, while you can still run your servlet and JSP or java web application (.war) file inside any web server like Tomcat or Jetty

1. Application Server supports distributed transaction and EJB. While Web Server only supports Servlets and JSP.

2. Application Server can contain web server in them. most of App server e.g. JBoss or WAS has Servlet and JSP container.

3. Though its not limited to Application Server but they used to provide services like Connection pooling, Transaction management, messaging, clustering, load balancing and persistence. Now Apache tomcat also provides connection pooling.

4. In terms of logical difference between web server and application server. web server is supposed to provide http protocol level service while application server provides support to web service and expose business level service e.g. EJB.

5. Application server are heavier than web server in terms of resource utilization.

Personally I don't like to ask questions like Difference between Application Server and Web Server. But since its been asked in many companies, you got to be familiar with some differences. Some times different interviewer expect different answer but I guess on Java's perspective until you are sure when do you need an application server and when you need a web server, you are good to go.

JDBC Vs Hibernate

JDBC stands for Java Database Connectivity allows developers to connect, query and update a database using the Structured Query Language. JDBC API standard provides Java developers to interact with different RDBMS and access table data through Java application without learning RDBMS details and using Database Specific JDBC Drivers.

Hibernate is an Object-Relational Mapping (ORM) solution for JAVA. It is a powerful, high performance object/relational persistence and query service. It allows us to develop persistent classes following object-oriented idiom – including association, inheritance and polymorphism.

Listed below is the comparison chart of difference beetween JDBC and Hibernate

|  |  |
| --- | --- |
| JDBC | Hibernate |
| With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. | Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this. |
| With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code. | Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. |
| JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task. | Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application. |
| Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object-to-table. | Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties. |
| With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. | Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost. |
| With JDBC, caching is maintained by hand-coding. | Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. |
| In JDBC there is no check that always every user has updated data. This check has to be added by the developer. | Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data. |

JMS

What is the difference between queue and topic?

**Queue vs Topic**

Java message service, or simply JMS, is a medium which sends messages to two or more clients. It permits contact between diverse mechanisms of a distributed application. This message-oriented middleware has two models which are the point-to-point model and publish or subscribe model. These two models have other names, too. The point-to-point model is also known as the queue model, and the publisher or subscriber model is also simply known as the topic model.

The queue or point-to-point model works by having a sender place messages to a queue, and the [receiver](http://www.differencebetween.net/technology/difference-between-amplifier-and-receiver/) will be able to read the messages from the queue. However, publisher or subscriber or the topic model works by disseminating messages by posting messages about a particular topic and having subscribers read them.

There are many differences between these two models which we will simply [call](http://www.differencebetween.net/business/finance-business-2/difference-between-call-and-put/) “queue” and “topic.” In queue, the sender knows where the message will be going. There is a specific sender and a specific receiver, and there is the intention of being acknowledged as such. On the other hand, in topic you only have a publisher and a subscriber or subscribers. There is anonymity in the identity of both the publisher and subscriber.

Another main difference between the two is the number of recipients. In queue, you only have one receiver or consumer; unlike in topic where in you can have your message be disseminated to a number of subscribers. Also, in topic, the publisher has to be continuously active for a subscriber to receive the messages. Otherwise the message will be reallocated. In queue you do not have to worry about timing because the sender will have the luxury to send messages whenever he or she wants to. And the same goes for the receiver; he or she also has the liberty of reading it whenever he or she wants. In queue you will also be assured that as the sender you have successfully sent out your message because you will be notified by the receiver, but the same is not true for a topic system. There is even the risk of not having any subscribers.

SUMMARY:

1.The point-to-point or queue model works by the sender to receiver setup. On the other hand, publisher/subscriber or topic model works by bulletin setup.

2.In the queue model there is acknowledgement of the identity of the receiver and oftentimes the sender. In the topic model there is anonymity in the identities of both the subscriber and publisher.

3.Queue model is only allowed one recipient; topic, on the other hand, can have multiple recipients.

4.In queue model, the sender and receiver do not have to be both active at the same time. In the topic model, timing is very vital.

5.In the queue model, the sender will receive a notification when the message gets to the receiver. The topic model, on the other hand, will not notify you with such, and there is even a risk that you will have no subscribers.

**AJAX**

**1. What is Ajax?**

Ajax is Asynchronous JavaScript and XML a client-side script that communicates with server without refresh the complete page. You can also say "the method of exchanging data with a server, and updating parts of a web page without reloading the entire page". Ajax is used in all web-technology like PHP, ASP, Java and Mobile Technology etc.

**2. What is advantage of AJax?**

Its faster as it load only required content.

More user friendly.

One page application possible due to Ajax.

Reduce the loading of page.

3. **What is XMLHttpRequest?**

XMLHttpRequest is an API available to web browser scripting languages (i.e. JavaScript) It is used to send HTTP/HTTPS requests to a web server and load the server's response into the script.

The XMLHttpRequest Object

The XMLHttpRequest object is used to exchange data with a server behind the scenes.

The XMLHttpRequest object is the developers dream, because you can:

Update a web page without reloading the page

Request data from a server after the page has loaded

Receive data from a server after the page has loaded

Send data to a server in the background

**XMLHttpRequest Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| abort() | Cancels the current request |
| getAllResponseHeaders() | Returns header information |
| getResponseHeader() | Returns specific header information |
| open(method,url,async,uname,pswd) | Specifies the type of request, the URL, if the request should be handled asynchronously or not, and other optional attributes of a request   method: the type of request: GET or POST url: the location of the file on the server async: true (asynchronous) or false (synchronous) |
| send(string) | send(string) Sends the request off to the server.  string: Only used for POST requests |
| setRequestHeader() | Adds a label/value pair to the header to be sent |

**XMLHttpRequest Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| onreadystatechange | Stores a function (or the name of a function) to be called automatically each time the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest. Changes from 0 to 4:  **0: request not initialized  1: server connection established 2: request received  3: processing request  4: request finished and response is ready** |
| responseText | Returns the response data as a string |
| responseXML | Returns the response data as XML data |
| status | Returns the status-number (e.g. "404" for "Not Found" or "200" for "OK") |
| statusText | Returns the status-text (e.g. "Not Found" or "OK") |

**JQuery**

**What is jQuery**

jQuery is a fast, lightweight, and feature-rich JavaScript library that is based on the priciple*"write less, do more"*. It's easy-to-use APIs makes the things like HTML document traversal and manipulation, event handling, adding animation effects to a web page much simpler that works seamlessly across all the major browsers. jQuery also gives you the ability to create an Ajax based application in a quick and simple way.

Companies like Google, Microsoft and IBM are using the jQuery for their applications. So you can easily understand how popular the jQuery is?

**Advantages of jQuery**

The biggest advantage of jQuery comes from its selectors that allow you to traverse the DOM tree of an HTML document's structure in an efficient manner.

Additionally, using the jQuery inbuilt methods you can create animations and effects like sliding transition, showing or hiding an element, etc. with a single line of code.

Here are some more advantages, why one should opt for jQuery:

**Save lots of time** — You can save lots of time and efforts by using the jQuery inbuilt effects and selectors and concentrate on other development work.

**Simplify common JavaScript tasks** — jQuery considerably simplifies the common JavaScript tasks. Now you can easily create feature rich and interactive web pages with fewer lines of codes, a typical example is retrieving the information from a server and updates the page without refreshing.

**Easy to use** — jQuery is very easy to use. Anybody with the basic working knowledge of HTML, CSS and JavaScript can start development with jQuery.

**Compatible with browsers** — jQuery is created with modern browsers in mind and it is compatible with all major modern browsers such as Mozilla Firefox, Google Chrome, Safari, Internet Explorer, and Opera.

**Absolutely Free** — And the best part is, it is completely free to download and use.

**Tip:**With JavaScript, you often need to write many lines of codes to locate an element of an HTML document, but with jQuery robust selector mechanism you can easily get the exact portion of the document to perform any manipulation.

**What is $() in jQuery library?**

The $() function is an alias of jQuery() function, at first it looks weird and makes jQuery code cryptic, but once you get used to it, you will love it's brevity. $() function is used to wrap any object into jQuery object, which then allows you to call various method defined jQuery object. You can even pass a selector string to $() function, and it will return jQuery object containing an array of all matched DOM elements.

You have five <div> element in your page? How do you select them using jQuery?

Another fundamental jQuery question based on selector. jQuery supports different kinds of selector e.g. ID selector, class selector and tag selector. Since in this question nothing has been mentioned about ID and class, you can use tag selector to select all div elements.  jQuery code : $("div"), will return a jQuery object contain all five div tags.

**Difference between ID selector and class selector in jQuery?**

If you have used CSS, then you might know the difference between ID and class selector, It's same with jQuery. ID selector uses ID e.g. #element1 to select element, while class selector uses CSS class to select elements. When you just need to select only one element, use ID selector, while if you want to select a group of element, having same CSS class than use class selector.  
  
$("#LoginTextBox") -- Returns element wrapped as jQuery object with id="LoginTextBox"

 $(".active") – Returns all elements with CSS class active.

**How do you hide an image on a button click using jQuery?**  
  
  
jQuery provides good support for handling events like button click. You can use following code to hide an image, found using Id or class. What you need to know is hide() method and how to setup an even handler for button, to handle clicks, you can use following jQuery code to do that  
  
$(“ButtonToClick”).click(function(){

$(“ImageToHide”).hide();

} );

**What is $(document).ready() function? Why should you use it?**  
  
ready() function is used to execute code when document is ready for manipulation. jQuery allows you to execute code, when DOM is fully loaded i.e. HTML has been parsed and DOM tree has been constructed. Main benefit of $(document).ready() function is that, it works in all browser, jQuery handles cross browser difficulties for you.  
  
6. **Difference between JavaScript window.onload event and jQuery ready function?**

Main difference between JavaScript onload event and jQuery ready function is that former not only waits for DOM to be created but also waits until all external resources are fully loaded including heavy images, audios and videos.  If loading images and media content takes lot of time that user can experience significant delay on execution of code defined in window.onload event. On the other hand jQuery ready() function only wait for DOM tree, and does not wait for images or external resource loading, means faster execution. Another advantage of using jQuery $(document).ready() is that you can use it multiple times in your page, and browser will execute them in the order they appear in HTML page, as opposed to onload technique, which can only be used for a single function. Given this benefits, it's always better to use jQuery ready() function than JavaScript window.onload event.  
  
  
**7. How do you find all selected options of HTML select tag?**  
  
This is one of the tricky jQuery question on Interviews. It's still a basic, but don't expect every jQuery beginners to know about this. You can use following jQuery selector to retrieve all selected options of <select> tag with multiple=true :

$('[name=NameOfSelectedTag] :selected')

This code uses attribute selector in combination of :selected selector, which returns only selected options. You can tweak this and instead of name, you can even use id attribute to retrieve <select> tag  
  
**8. What is each() function in jQuery? How do you use it?**  
  
each() function is like Iterator in Java, it allows you to iterate over a set of elements. You can pass a function to each() method, which will be executed for each element from the jQuery object, on which it has been called. This question sometime asked as follow-up of previous question e.g. how to show all selected options in alert box. We can use above selector code to find all selected option and than further can use each() method to print them in alert box, one by one, as shown below:  
  
text() method returns text for that option.

$(‘[name=NameOfSelectedTag]:selected’).each(function(selected)

{

Alert($(selected).text());

});

**9. How do you add an HTML element in DOM tree?**

You can use jQuery method appendTo() to add an HTML element in DOM tree. This is one of the many DOM manipulation method jQuery provides. You can add an existing element or a new HTML element, appendTo() add that method in the end of a particular DOM element  
  
10 Can you write jQuery code to select all links, which is inside paragraphs?  
 **11. Difference between $(this) and this keyword in jQuery?**  
  
$(this) returns a jQuery object, on which you can call several jQuery methods e.g. text() to retrieve text, val() to retrieve value etc, while this represent current element, and it's one of the JavaScript keyword to denote current DOM element in a context. You can not call jQuery method on this, until it's wrapped using $() function i.e. $(this).  
  
**How do you retrieve attribute of an HTML tag using jQuery e.g. href of links?**  
attr() method is used to retrieve value of an attribute of any HTML element. You first need to select all links or specified links using jQuery selector and than you can apply attr() method to get value of there href attribute. Below code will find all links from a page and return href value :  
  
$(“a”).each(function(){

Alert($(this).attr(‘href’));

**});**How do you set attribute using jQuery?

attr() method is overload like many other methods in JQuery. If you call attr() method with value e.g. attr(name, value), where name is the name of attribute and value is the new value.

**What is difference between detach() and remove() method in jQuery?**  
  
Though both detach() and remove() method is used to remove a DOM element, Main difference between them is that detach() keep track of the last element detached, so that it can be reattached, while remove() method does keep reference of last removed method. You can also take a look on appendTo() for adding element into DOM.

**empty**(): This method removes all the child element of the matched element where remove() method removes set of matched elements from DOM.  
  
.**remove**(): This method takes elements out of the DOM. Use .remove() when you want to remove the element itself, as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed.

.**detach**(): This method is same as remove(), except that detach() keep all jquery data associated with the removed elements. This method is useful when removed elements are reinserted into the DOM at a later time.

**How do you add and remove CSS classes to an element using jQuery?**

By using addClass() and removeClass() jQuery methods. This can be very handy, while dynamically changing class of elements e.g. marking them inactive or active and using class ".active" etc.

**What is main advantage of loading jQuery library using CDN?**  
  
Well, apart from many advantages including reducing server bandwidth and faster download, one of the most important is that, if browser has already downloaded same jQuery version from same CDN, than it won't download it again. Since now days, almost many public websites use jQuery for user interaction and animation, there is very good chance that browser already have jQuery library downloaded.  
  
**What is difference between jQuery.get() and jQuery.ajax() method?**

ajax() method is more powerful and configurable, allows you to specify how long to wait and how to handle error, get() is a specialization to over ajax just to retrieve some data.

**What is method chaining in jQuery? what is benefit of using method chaining?**

Method chaining is calling another method in result of another method, it result in clean and concise code, single search over DOM so better performance.

**What happen if you return false from a jQuery event handler?**

It used to stop the event bubbling up

**Which one is more efficient, document.getElementbyId( "myId") or $("#myId)?**

First one because its direct call to JavaScript engine

**jQuery Get or Set Contents and Values**

Some jQuery methods can be used to either assign or read some value on a selection. A few of these methods are **text(), html(), attr(), and val().**

When these methods are called with no argument, it is referred to as a *getters*, because it gets (or reads) the value of the element. When these methods are called with a value as an argument, it's referred to as a *setter* because it sets (or assigns) that value.

## What is Traversing

The jQuery selectors we've seen so far only allow us to select the elements down the DOM tree. But there are many occasions when you need to select a parent or ancestor element; that is where jQuery's DOM traversal methods come into play. With these traversal methods, we can go up, down, and all around the DOM tree very easily.

DOM traversing is one of the prominent features of the jQuery. To make the most it you need to understand the relationships between the elements in a DOM tree.

## jQuery find() Method

The jQuery find() method is used to get the descendant elements of the selected element.

The find() and children() methods are similar, except that the find() method search through multiple levels down the DOM tree to the last descendant, whereas the children()method only search a single level down the DOM tree. The following example will add a border around all the [<li>](http://www.tutorialrepublic.com/html-reference/html-li-tag.php) elements that are descendants of the [<div>](http://www.tutorialrepublic.com/html-reference/html-div-tag.php) element.

## 

## jQuery noConflict() Method

The **jQuery.noConflict()** method return the control of the $ **identifier** back to other libraries. The jQuery code in the following example (line no-10) will put the jQuery into **no-conflict** mode immediately after it is loaded onto the page and assign a new variable name $**j** to replace the $**alias** in order to avoid conflicts with the prototype framework.

## Steps to implement quick sort algorithm:

Create a array with some elements. Choose pivot element as middle element. We can choose first or last also. After choosing pivot element arrange the elements like all elements which are less than pivot value comes left side and elements greater than equal to pivot come right side of pivot. And then apply same to both sides. until it becomes one. then all elements in array will be sorted.

[**Insertion sort algorithm**](http://www.instanceofjava.com/2016/08/insertion-sort-algorithm-with-example.html)

Sorting means arranging elements of a array or list in ascending or descending order.

Iterative and recursive algorithms.

Insertion sort is iterative type of algorithm.

Insertion sort algorithm is in place algorithm

The basic idea behind insertion sort is to divide o

We have various sorting algorithms in java.

ur list in to two parts , sorted and un sorted.

At each step of algorithm a number is moved from unsorted part to sorted part.

Initially take 1st element as sorted element.

Noe take second element and compare with first element if it less than the 1st element then swap two numbers means place lesser value in first position.

So now have two elements in sorted part. Take third element and compare with second element if it is less than the second element then we need to compare it with 1st element if is less than first element we need keep that in first place. Take an element from unsorted portion and compare with sorted portion and place it in correct position in order to make sorted.

This will be repeated until entire list will be sorted.

**Selection Sort Algorithm**

Selection sort algorithm is to arrange unsorted elements in to sorting order with increasing or decreasing order.

To sort list of unsorted list of elements first it will search for minimum value in the list and place it in first index.

Now first place is fixed with smallest value in the list now it starts from second element and compare next element and if it is big it compares next element if it is small it picks that element and compares with next element.

So again it picks smallest element in the list and place it in second place. This procedure will repeat until it reaches n-1 position.

**DESIGN PATTERNS**

**Design Patterns** are very popular among software developers. A design pattern is a well described solution to a common software problem. I have written extensively on **java design patterns**.

## Java Design Patterns

Some of the benefits of using design patterns are:

Design Patterns are already defined and provides **industry standard approach** to solve a recurring problem, so it saves time if we sensibly use the design pattern. There are many java design patterns that we can use in our java based projects.

Using design patterns promotes **reusability** that leads to more **robust** and highly maintainable code. It helps in reducing total cost of ownership (TCO) of the software product.

Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily.

**Design patterns:**

Pattern means set of guide lines.

Design patterns are solutions to commonly reoccurring problems in software development.

Design patterns are well proven solutions to common software problems.

Design patterns are best practices to use software technologies effectively in application development.

Design patterns used in analysis and requirement phase of  SDLC.

Design patterns can be implemented by using programming language.

**Advantages:**

**Reusable.**

These are already defined solutions to common re occurring problems so it reduces time.

They are already defined so Easy to understand and debug.

**Categorization:**

These are categorized into two parts.

Java SE Design patterns.

Java EE Design patterns.

**Java SE Design patterns:**

 In Java SE there are mainly three types.

**Creational Design patterns:**

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design.

**Factory Pattern:**

Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Ex: Circle, Rectangle, Square implements Shape interface.

ShapeFactory and FactoryPatternDemo (Main method).

**Abstract Factory Pattern:**

Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

Ex: Circle, Rectangle, Square implements Shape interface.

Red, Green, Blue implements Color interface.

ShapeFactory, ColorFactory implements Abstract class ‘AbstractFactory’.

AbstractFactoryPatternDemo creates factory classes through FactoryProducer.

ShapeFactory and FactoryPatternDemo(Main method).

**Singleton Pattern:**

Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

SingleObject, SinglePatternDemo

**Prototype Pattern**

Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

[**Builder Pattern.**](http://www.instanceofjava.com/2016/08/builder-design-pattern-java-example.html)

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

A Builder class builds the final object step by step. This builder is independent of other objects.

Interfaces - Item, Packing

Abstract Class – Burger implements Item

ColdDrink implements item.

Class: Bottle implements Packing, Wrapper implements Packing, Meal, MealBuilder,

Coke extends ColdDrink, Pepsi extends ColdDrink

Class: ChickenBurger extends Burger, VegBurger extends Burger.

**Structural Design patterns:**

In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships between entities. Examples of Structural Patterns include: Adapter pattern: 'adapts' one interface for a class into one that a client expects.

**Adapter Pattern**

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces. A real life example could be a case of card reader which acts as an adapter between memory card and a laptop. You plugin the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.

Interfaces: MediaPlayer, AdvancedMediaPlayer

Classes : VlcPlayer, Mp4Player implements AdvancedMediaPlayer

Class: MediaAdapter implements MediaPlayer

**Bridge Pattern**

Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

Abstract Class – Shape refers DrawAPI interface

GreenCircle, RedCircle implements DrawAPI interface

Circle Class extends Shape abstract class.

**Composite Pattern**

Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern comes under structural pattern as this pattern creates a tree structure of group of objects.

This pattern creates a class that contains group of its own objects. This class provides ways to modify its group of same objects.

Employee, CompositePatternDemo

**Decorator Pattern**

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

**Circle** class implements **Shape** interface

**Rectangle** class implements **Shape** interface

**ShapeDecorator** abstract class implements **Shape** interface.

**RedShapeDecorator** extends **ShapeDecorator** class.

**Facade Pattern**

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

Circle, Square, Rectangle implements **Shape** interface

ShapeMaker uses all those classes.

**Flyweight Pattern**

Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.

Flyweight pattern tries to reuse already existing similar kind objects by storing them and creates new object when no matching object is found. We will demonstrate this pattern by drawing 20 circles of different locations but we will create only 5 objects. Only 5 colors are available so color property is used to check already existing *Circle* objects.

**Circle** implements **Shape** interface

ShapeFactory is a class

FlyweightPatternDemo

**Proxy Pattern**

In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In proxy pattern, we create object having original object to interface its functionality to outer world.

**Filter Pattern**

Filter pattern or Criteria pattern is a design pattern that enables developers to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations. This type of design pattern comes under structural pattern as this pattern combines multiple criteria to obtain single criteria.

Person is a separate class

AndCriteria, OrCriteria, CriteriaFemale, CriteriaMale, CriteriaFemale, CriteriaSingle classes are implementing Criteria interface.

**Behavioral Design patterns:**

Behavioral patterns provide solution for the better interaction between objects and how to provide lose coupling and flexibility to extend easily.

**Chain of Responsibility Pattern**

As the name suggests, the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.

In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

**Command Pattern**

Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

**Interpreter Pattern**

Interpreter pattern provides a way to evaluate language grammar or expression. This type of pattern comes under behavioral pattern. This pattern involves implementing an expression interface which tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine etc.

**Iterator Pattern**

Iterator pattern is very commonly used design pattern in Java and .Net programming environment. This pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.

Iterator pattern falls under behavioral pattern category.

**Mediator Pattern**

Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling. Mediator pattern falls under behavioral pattern category.

**Memento Pattern**

Memento pattern is used to restore state of an object to a previous state. Memento pattern falls under behavioral pattern category.

**Observer Pattern**

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

**State Pattern**

In State pattern a class behavior changes based on its state. This type of design pattern comes under behavior pattern.

In State pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

**Strategy Pattern**

In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.

In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

**Template Pattern**

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behavior pattern category.

**Visitor Pattern**

In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.

**Null Object Pattern**

In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do nothing relationship. Such Null object can also be used to provide default behaviour in case data is not available.

In Null Object pattern, we create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implemention of this class and will be used seemlessly where we need to check null value.

**MVC pattern**

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

View - View represents the visualization of the data that model contains.

Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

**Java EE Design patterns**

**Business Delegate Pattern**

Business Delegate Pattern is used to decouple presentation tier and business tier. It is basically use to reduce communication or remote lookup functionality to business tier code in presentation tier code. In business tier we have following entities.

**Client** - Presentation tier code may be JSP, servlet or UI java code.

**Business Delegate** - A single entry point class for client entities to provide access to Business Service methods.

**LookUp Service** - Lookup service object is responsible to get relative business implementation and provide business object access to business delegate object.

**Business Service** - Business Service interface. Concrete classes implement this business service to provide actual business implementation logic.

**Composite Entry Pattern**

Composite Entity pattern is used in EJB persistence mechanism. A Composite entity is an EJB entity bean which represents a graph of objects. When a composite entity is updated, internally dependent objects beans get updated automatically as being managed by EJB entity bean. Following are the participants in Composite Entity Bean.

**Composite Entity** - It is primary entity bean. It can be coarse grained or can contain a coarse grained object to be used for persistence purpose.

**Coarse-Grained Object** - This object contains dependent objects. It has its own life cycle and also manages life cycle of dependent objects.

**Dependent Object** - Dependent object is an object which depends on coarse grained object for its persistence lifecycle.

**Strategies** - Strategies represents how to implement a Composite Entity.

**Data Access object Pattern**

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

**Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).

**Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.

**Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

**Front Controller Pattern**

The front controller design pattern is used to provide a centralized request handling mechanism so that all requests will be handled by a single handler. This handler can do the authentication/ authorization/ logging or tracking of request and then pass the requests to corresponding handlers. Following are the entities of this type of design pattern.

**Front Controller** - Single handler for all kinds of requests coming to the application (either web based/ desktop based).

**Dispatcher** - Front Controller may use a dispatcher object which can dispatch the request to corresponding specific handler.

**View** - Views are the object for which the requests are made.

**Intercepting Filter Pattern**

The intercepting filter design pattern is used when we want to do some pre-processing / post-processing with request or response of the application. Filters are defined and applied on the request before passing the request to actual target application. Filters can do the authentication/ authorization/ logging or tracking of request and then pass the requests to corresponding handlers. Following are the entities of this type of design pattern.

**Filter** - Filter which will perform certain task prior or after execution of request by request handler.

**Filter Chain** - Filter Chain carries multiple filters and help to execute them in defined order on target.

**Target** - Target object is the request handler

**Filter Manager** - Filter Manager manages the filters and Filter Chain.

**Client** - Client is the object who sends request to the Target object.

**Service Locator Pattern**

The service locator design pattern is used when we want to locate various services using JNDI lookup. Considering high cost of looking up JNDI for a service, Service Locator pattern makes use of caching technique. For the first time a service is required, Service Locator looks up in JNDI and caches the service object. Further lookup or same service via Service Locator is done in its cache which improves the performance of application to great extent. Following are the entities of this type of design pattern.

**Service** - Actual Service which will process the request. Reference of such service is to be looked upon in JNDI server.

**Context / Initial Context** - JNDI Context carries the reference to service used for lookup purpose.

**Service Locator** - Service Locator is a single point of contact to get services by JNDI lookup caching the services.

**Cache**- Cache to store references of services to reuse them

**Client** - Client is the object that invokes the services via ServiceLocator.

**Transfer Object Pattern**

The Transfer Object pattern is used when we want to pass data with multiple attributes in one shot from client to server. Transfer object is also known as Value Object. Transfer Object is a simple POJO class having getter/setter methods and is serializable so that it can be transferred over the network. It does not have any behavior. Server Side business class normally fetches data from the database and fills the POJO and send it to the client or pass it by value. For client, transfer object is read-only. Client can create its own transfer object and pass it to server to update values in database in one shot. Following are the entities of this type of design pattern.

**Business Object** - Business Service fills the Transfer Object with data.

**Transfer Object** - Simple POJO having methods to set/get attributes only.

**Client** - Client either requests or sends the Transfer Object to Business Object.

**Factory Method Pattern**:

**Problem**: Using new keyword we cannot create object with flexibility and by applying restrictions.

**Solution**: Use Factory pattern (or) Factory method.

By defining a abstract class or an interface but let the subclass  decide which class object to instantiate.

A method of a class capable of constructing and returning its own class object or other class object is called "factory method".

There are two types of factory methods.

**Static factory method.**

**Instance factory method.**

**1.Static Factory method:**

A static method defined to construct and return object of same class or different is known as static factory method.

* + - * Some of the pre defined static factory methods are as follows.

**Thread th= Thread.currentThread();**

**Class c=Class.forName();**

**Runtime rt=Runtime.getRuntime();**

**Calendar c=Calendar.getInstance();**

* **Instance Factory method:**

A non static method defined to construct and return object of same class or different is known as instance factory method. Some of the pre defined instance factory methods are as follows.

**String s= new String("instance of");  
 String s1=s.concat("java");**

**StringBuffer sb=new StringBuffer("instance of");  
sb=sb.subString(0,2);**

**Date d= new Date();  
String s=d.toString();**

[**Singleton Design Pattern**](http://www.instanceofjava.com/2015/05/singleton-design-pattern-in-java.html)

**Problem:**

Instead of creating multiple objects of same class having same data and wasting memory, degrading performance it is recommended to create only one object and use it multiple times.

**Solution:**

 Use Singleton Java Class.

Java class that allows us to create one object per JVM is is called as singleton java class.

The logger class of Log 4J API is given as singleton java class.

**Rules:**

It must have only private constructors.

It must have private static reference variable of same class.

It must have public static factory method having the logic of singleton.

Static method should create and return only one object.

All these rules close all the doors of creating objects for java class and opens only one door to create object.

factory method where singleton logic is placed

The method of a class capable of creating and returning same class or other class object is known as factory method.

[**Builder design pattern**](http://www.instanceofjava.com/2016/08/builder-design-pattern-java-example.html)

Design patterns are solutions to software design problems.

Design patterns classified into three types.

Creational, Structural and behavioral design patterns.

Creational patterns helps us to create objects in a manner suitable to the given situation.

Builder design pattern is one of the creational  design pattern in java.

Builder design pattern helps us to create complex class object.

Builder design pattern helps us to separate the construction process of a complex object from its representation so that same object construction process can be created in different representations.

Means it will separate complex construction into two parts initialization of class instance and return class instance.

When a class having more number of fields and constructor of that class take care of assigning initial values.

And when we want to create object of the class we need to pass all parameters and should be in same order which constructor is accepting.

Builder design pattern helps us to create same class object by passing required number of fields by using separate builder class object.

Builder design pattern is useful when object creation is very complex.

**Advantages of builder design pattern:**

Builder design pattern simplifies complex object creation.

Builder design pattern provides separation between instance creation and representation

Reusability

## *Core J2EE patterns are broadly classified in the following three categories based on the layering of the multi tiered systems. A specific patterns applies to a specific layer of the system under development*

## *[Presentation Tier Patterns](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns)*

* [**Intercepting Filter**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/intercepting-filter-1)intercepts incoming requests and outgoing responses and applies a filter. These filters may be added and removed in a declarative manner, allowing them to be applied unobtrusively in a variety of combinations. After this preprocessing and/or post-processing is complete, the final filter in the group vectors control to the original target object. For an incoming request, this is often a Front Controller, but may be a View.
* [**Front Controller**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/front-controller)is a container to hold the common processing logic that occurs within the presentation tier and that may otherwise be erroneously placed in a View. A controller handles requests and manages content retrieval, security, view management, and navigation, delegating to a Dispatcher component to dispatch to a View.
* [**Application Controller**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/application-controller) centralizes control, retrieval, and invocation of view and command processing. While a Front Controller acts as a centralized access point and controller for incoming requests, the Application Controller is responsible for identifying and invoking commands, and for identifying and dispatching to views.
* [**Context Object**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/context-object) encapsulates state in a protocol-independent way to be shared throughout your application. Using Context Object makes testing easier, facilitating a more generic test environment with reduced dependence upon a specific container.
* [**View Helper**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/view-helper)encourages the separation of formatting-related code from other business logic. It suggests using Helper components to encapsulate logic relating to initiating content retrieval, validation, and adapting and formatting the model. The View component is then left to encapsulate the presentation formatting. Helper components typically delegate to the business services via a Business Delegate or an Application Service, while a View may be composed of multiple subcomponents to create its template.
* [**Composite View**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/composite-view) suggests composing a View from numerous atomic pieces. Multiple smaller views, both static and dynamic, are pieced together to create a single template. The Service to Worker and Dispatcher View patterns represent a common combination of other patterns from the catalog. The two patterns share a common structure, consisting of a controller working with a Dispatcher, Views, and Helpers. Service to Worker and Dispatcher View have similar participant roles, but differ in the division of labor among those roles. Unlike Service to Worker, Dispatcher View defers business processing until view processing has been performed.
* [**Service to worker**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/service-to-worker) performs core request handling and invoke business logic before control is passed to the view. It centralizes control and request handling to retrieve a presentation model before turning control over to the view. The view generates a dynamic response based on the presentation model.
* [**Dispatcher View**](http://www.javagyan.com/tutorials/corej2eepatterns/presentation-tier-patterns/dispatcher-view)combines a controller and dispatcher with views and helpers to handle client requests and prepare a dynamic presentation as the response. Controllers do not delegate content retrieval to helpers, because these activities are deferred to the time of view processing. A dispatcher is responsible for view management and navigation and can be encapsulated either within a controller, a view, or a separate component.

## *[Business Tier Patterns](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns)*

* [**Business Delegate**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/business-delegate) reduces coupling between remote tiers and provides an entry point for accessing remote services in the business tier. A Business Delegate might also cache data as necessary to improve performance. A Business Delegate encapsulates a Session Façade and maintains a one-to-one relationship with that Session Façade. An Application Service uses a Business Delegate to invoke a Session Façade.
* [**Service Locator**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/service-locator) encapsulates the implementation mechanisms for looking up business service components. A Business Delegate uses a Service Locator to connect to a Session Façade. Other clients that need to locate and connect to Session Façade, other business-tier services, and web services can use a Service Locator.
* [**Session Façade**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/session-facade)provides coarse-grained services to the clients by hiding the complexities of the business service interactions. A Session Façade might invoke several Application Service implementations or Business Objects. A Session Façade can also encapsulate a Value List Handler.
* [**Application Service**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/application-service) centralizes and aggregates behavior to provide a uniform service layer to the business tier services. An Application Service might interact with other services or Business Objects. An Application Service can invoke other Application Services and thus create a layer of services in your application.
* [**Business Object**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/business-object) implements your conceptual domain model using an object model. Business Objects separate business data and logic into a separate layer in your application. Business Objects typically represent persistent objects and can be transparently persisted using Domain Store.
* [**Composite Entity**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/composite-entity) implements a Business Object using local entity beans and POJOs. When implemented with bean-managed persistence, a Composite Entity uses Data Access Objects to facilitate persistence.
* [**The Transfer Object**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/transfer-object) pattern provides the best techniques and strategies to exchange data across tiers (that is, across system boundaries) to reduce the network overhead by minimizing the number of calls to get data from another tier.
* [**The Transfer Object Assembler**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/transfer-object-assembler) constructs a composite Transfer Object from various sources. These sources could be EJB components, Data Access Objects, or other arbitrary Java objects. This pattern is most useful when the client needs to obtain data for the application model or part of the model.
* [**The Value List Handler**](http://www.javagyan.com/tutorials/corej2eepatterns/business-tier-patterns/value-list-handler) uses the GoF iterator pattern to provide query execution and processing services. The Value List Handler caches the results of the query execution and return subsets of the result to the clients as requested. By using this pattern, it is possible to avoid overheads associated with finding large numbers of entity beans. The Value List Handler uses a Data Access Object to execute a query and fetch the results from a persistent store.

## *[Integration Tier Patterns](http://www.javagyan.com/tutorials/corej2eepatterns/integration-tier-patterns)*

* [**Data Access Object**](http://www.javagyan.com/tutorials/corej2eepatterns/integration-tier-patterns/data-access-object) enables loose coupling between the business and resource tiers. Data Access Object encapsulates all the data access logic to create, retrieve, delete, and update data from a persistent store. Data Access Object uses Transfer Object to send and receive data.
* [**Service Activator**](http://www.javagyan.com/tutorials/corej2eepatterns/integration-tier-patterns/service-activator) enables asynchronous processing in your enterprise applications using JMS. A Service Activator can invoke Application Service, Session Façade or Business Objects. You can also use several Service Activators to provide parallel asynchronous processing for long running tasks.
* [**Domain Store**](http://www.javagyan.com/tutorials/corej2eepatterns/integration-tier-patterns/domain-store) provides a powerful mechanism to implement transparent persistence for your object model. It combines and links several other patterns including Data Access Objects.
* [**Web Service Broker**](http://www.javagyan.com/tutorials/corej2eepatterns/integration-tier-patterns/web-service-broker) exposes and brokers one or more services in your application to external clients as a web service using XML and standard web protocols. A Web Service Broker can interact with Application Service and Session Façade. A Web Service Broker uses one or more Service Activators to perform asynchronous processing of a request.



**STATIC OR DYNAMIC Websites?**

One of the first questions you’ll need to answer when researching deployment options is whether your application is static or dynamic. Each type of website has different needs, and as such not all deployment solutions may work for a given web application. Below we will look at the terms static and dynamic as they pertain to web development, and provide guidelines for determining which category your site falls in to.

A static website, despite the name, does not mean that the website content does not change. In this sense, a static website is any kind of website that can run on top of an unmodified server. In essence, if your client-side application code never calls back to your server (within certain limits), then you can really deploy on any platform that offers hosting. It is the activity on an application’s server that makes it static – the front end code can be dynamic and communicate with any number of services, but so long as it does not perform any complex work on your web server itself then your web app can be considered static. Some examples of this type of web application are a simple personal home page, an online games portal that doesn’t save data to the server on which it is hosted, or an AngularJS app that performs multiple calls to a RESTful API provided by another service.

A dynamic website, on the other hand, refers to those websites that require a server to perform more complex tasks than simply presenting the individual pages to the user. These can take any number of forms, but they most commonly involve both server side and client side code in the application. Most Ruby on Rails websites, for example, fall into this category, as do most data-oriented websites (there are specific cases where an application can operate on top of a simple CRUD API and still be considered static from the standpoint of certain providers, but these are highly unique scenarios dependent upon the hosting provider. In essence, if your website performs any actions whatsoever in response to a specific call from your front-end code, such as applying business logic to data being saved to a database, then your website is considered dynamic. A good rule is if your website consists entirely of front-end code with no server interaction (such as most JavaScript-based websites), then your page is static, while if it uses a complicated API for database and server interaction (such as .NET, Rails, or Django) it is considered dynamic.

AngularJS

AngularJS is a full frontend MVC framework for JavaScript web applications. It was built at Google and provides a way to quickly build large, single-page web applications. Like jQuery, it is included into a page using the tag, and is itself written in JavaScript. Unlike jQuery, it is meant to be a framework upon which an entire web application is built. It actually includes a minimal version of jQuery by default.

If you’re looking to learn AngularJS, I recommend EggHead’s video tutorials. You’ll need to have a very solid understanding of JavaScript first, since writing any Angular requires a deep understanding of prototyping, scope, and various other JavaScript aspects.

The AngularJS website has a page of example projects built with AngularJS, if you’re so inclined.

Node.js

Remember how I told you that JavaScript ran in the browser, but I mentioned that there was one big exception to that? Node.js is that exception. It’s a command-line tool that runs JavaScript on a machine without needing to run in a browser. It does this by using a version of Chrome’s V8 Engine, which is the JavaScript engine that runs inside Google Chrome.

Before Node.js, developers would have to use different languages for the backend and frontend of their application. For example, PHP, Java, ASP.Net would run on the server, and JavaScript would run in the client browser. Now with Node.js, developers can use JavaScript on the server as well as the client, meaning that developers can focus on learning one language. Whether this is a good thing is still up for debate (nsfw language).

Angular is a hip JavaScript framework which is made for building large, single-page web applications.

Node.js allows JavaScript to be run without a browser, and is commonly used to run web servers.

AngularJS is built and maintained by dedicated (and highly talented) Google engineers. It is not a framework made by hobbyists in the open source community. AngularJS is a flexible JavaScript MVC framework for building rich client-side applications. AngularJS is a framework, not a JavaScript library like jQuery. You still may need to have a server-side backend, but most of the user-interactivity logic will be delegated to the client-side. Other frameworks deal with HTML’s shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM. Neither of these address the root problem that HTML was not designed for dynamic views.

**AngularJS has the following features:**

**Two way data binding**

**MVW pattern (MVC-ish)**

**Template**

**Custom-directive (reusable components, custom markup)**

**REST-friendly**

**Deep Linking (set up a link for any dynamic page)**

**Form Validation**

**Server Communication**

**Localization**

**Dependency injection**

**Full testing environment (both unit, e2e)**

If you are developing a heavy web applications, AngularJS is for you. AngularJS can use jQuery if it’s present in your app when the application is being bootstrapped. If jQuery is not present in your script path, AngularJS falls back to its own implementation of the subset of jQuery that we call jQLite.

 jQuery is a free, fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler. jQuery simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

HTML/DOM manipulation

CSS manipulation

HTML event methods

Effects and animations

AJAX

Utilities

SQL/PLSQL

Inner Join

Select all records from Table A and Table B, where the join condition is met.

select first\_name, last\_name, order\_date, order\_amount

from customers c inner join orders o

on c.customer\_id = o.customer\_id

Left Join

Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

select first\_name, last\_name, order\_date, order\_amount

from customers c

left join orders o

on c.customer\_id = o.customer\_id

Right Join

Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

select first\_name, last\_name, order\_date, order\_amount

from customers c

right join orders o

on c.customer\_id = o.customer\_id

Full Join

Select all records from Table A and Table B, regardless of whether the join condition is met or not.

select first\_name, last\_name, order\_date, order\_amount

from customers c

full join orders o

on c.customer\_id = o.customer\_id

SELF Join:

You use a self join when a table references data in itself.

E.g., an Employee table may have a SupervisorID column that points to the employee that is the boss of the current employee.

To query the data and get information for both people in one row, you could self join like this:

select e1.EmployeeID,

e1.FirstName,

e1.LastName,

e1.SupervisorID,

e2.FirstName as SupervisorFirstName,

e2.LastName as SupervisorLastName

from Employee e1

left outer join Employee e2 on e1.SupervisorID = e2.EmployeeID

Well, one classic example is where you wanted to get a list of employees and their immediate managers:

select e.employee as employee, b.employee as boss

from emptable e, emptable b

where e.manager\_id = b.empolyee\_id

order by 1

Primary Key:

CONSTRAINT MYTABLE\_ID PRIMARY KEY (id));

COMPOSITE PRIMARY KEY:

CONSTRAINT MYTABLE\_NAME\_ID\_PK PRIMARY KEY (name,id));

# Foreign key referencing only part of composite primary key.

FOREIGN KEY (M,N) REFERENCES Table1 (A,B)

# Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let’s discuss about anomalies first then we will discuss normal forms with examples.

## Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

**Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp\_id for storing employee’s id, emp\_name for storing employee’s name, emp\_address for storing employee’s address and emp\_dept for storing the department details in which the employee works. At some point of time the table looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_dept |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

The above table is not normalized. We will see the problems that we face when a table is not normalized.

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

## Normalization

Here are the most commonly used normal forms:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce & Codd normal form (BCNF)

## First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

**Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212  9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123  8123450987 |

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF**as the rule says “each attribute of a table must have atomic (single) values”, the emp\_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

## Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

* Table is in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

|  |  |  |
| --- | --- | --- |
| teacher\_id | subject | teacher\_age |
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys**: {teacher\_id, subject}  
**Non prime attribute**: teacher\_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher\_age is dependent on teacher\_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:  
**teacher\_details table:**

|  |  |
| --- | --- |
| teacher\_id | teacher\_age |
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

**teacher\_subject table:**

|  |  |
| --- | --- |
| teacher\_id | subject |
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

Now the tables comply with Second normal form (2NF).

## Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

* Table must be in 2NF
* [**Transitive functional dependency**](http://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [**candidate key**](http://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

* X is a [**super key**](http://beginnersbook.com/2015/04/super-key-in-dbms/) of table
* Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

**Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee\_details that looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_zip | emp\_state | emp\_city | emp\_district |
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

**Super keys**: {emp\_id}, {emp\_id, emp\_name}, {emp\_id, emp\_name, emp\_zip}…so on  
**Candidate Keys**: {emp\_id}  
**Non-prime attributes**: all attributes except emp\_id are non-prime as they are not part of any candidate keys.

Here, emp\_state, emp\_city & emp\_district dependent on emp\_zip. And, emp\_zip is dependent on emp\_id that makes non-prime attributes (emp\_state, emp\_city & emp\_district) transitively dependent on super key (emp\_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

**employee table:**

|  |  |  |
| --- | --- | --- |
| emp\_id | emp\_name | emp\_zip |
| 1001 | John | 282005 |
| 1002 | Ajeet | 222008 |
| 1006 | Lora | 282007 |
| 1101 | Lilly | 292008 |
| 1201 | Steve | 222999 |

**employee\_zip table:**

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_zip | emp\_state | emp\_city | emp\_district |
| 282005 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

## Boyce Codd normal form (BCNF)

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every [**functional dependency**](http://beginnersbook.com/2015/04/functional-dependency-in-dbms/) X->Y, X should be the super key of the table.

**Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| emp\_id | emp\_nationality | emp\_dept | dept\_type | dept\_no\_of\_emp |
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | stores | D001 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | Purchasing department | D134 | 600 |

**Functional dependencies in the table above**:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

**Candidate key**: {emp\_id, emp\_dept}

The table is not in BCNF as neither emp\_id nor emp\_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:  
**emp\_nationality table:**

|  |  |
| --- | --- |
| emp\_id | emp\_nationality |
| 1001 | Austrian |
| 1002 | American |

**emp\_dept table:**

|  |  |  |
| --- | --- | --- |
| emp\_dept | dept\_type | dept\_no\_of\_emp |
| Production and planning | D001 | 200 |
| stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

**emp\_dept\_mapping table:**

|  |  |
| --- | --- |
| emp\_id | emp\_dept |
| 1001 | Production and planning |
| 1001 | stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

**Functional dependencies**:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

**Candidate keys**:  
For first table: emp\_id  
For second table: emp\_dept  
For third table: {emp\_id, emp\_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

**JAVA SCRIPT:**

## typeof Operator

The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

**result = (typeof b == "string" ? "B is String" : "B is Numeric");**

# preventDefault() Event Method

## Definition and Usage

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

Clicking on a "Submit" button, prevent it from submitting a form

Clicking on a link, prevent the link from following the URL

**Note:** Not all events are cancelable. Use the [cancelable](http://www.w3schools.com/jsref/event_cancelable.asp) property to find out if an event is cancelable.

**Note:** The preventDefault() method does not prevent further propagation of an event through the DOM. Use the stopPropagation() method to handle this.

Prevent the default action of a checkbox:

document.getElementById("myCheckbox").addEventListener("click",

function(event){  
    event.preventDefault()  
});

1. The ‘**with’** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to with becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

1. getElementById( id)

Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document.

**Ex − document.getElementById( id)**

1. getElementsByName( name)

Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array.

**Ex − document.getElementsByName( name)**

1. getElementsByTagName( tagname)

Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source.

Ex **− document.getElementsByTagName( tagname)**

Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

* **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
* **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
* **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
* **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects −



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

* [The Legacy DOM](https://www.tutorialspoint.com/javascript/javascript_legacy_dom.htm) − This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
* [The W3C DOM](https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm) − This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
* [The IE4 DOM](https://www.tutorialspoint.com/javascript/javascript_ie4_dom.htm) − This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

## DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example −

if (document.getElementById) {

// If the W3C method exists, use it

}

else if (document.all) {

// If the all[] array exists, use it

}

else {

// Otherwise use the legacy DOM

}

## The onerror() Method

The onerror event handler was the first feature to facilitate error handling in JavaScript. The error event is fired on the window object whenever an exception occurs on the page.

The **onerror** event handler provides three pieces of information to identify the exact nature of the error −

Error message − The same message that the browser would display for the given error

URL − The file in which the error occurred

Line number− The line number in the given URL that caused the error.

window.onerror = function (msg, url, line) {

alert("Message : " + msg );

alert("url : " + url );

alert("Line number : " + line );

}

**Question: How to set a default parameter value for a JavaScript function?**

/\*\* Here email is parameter in which we have set the default value i.e email@domain.com \*\*/

function function1(name, email)

{

email = typeof email !== 'undefined' ? email : 'defaultemail@domain.com';

console.log('name='+name+', Email= '+email);

}

function1('john','myname@gmail.com');

function1('john');

**Queston: How to convert a string to lowercase?**

var str='This is testing String';

str = str.toLowerCase();

console.log(str);

**Question: How to modify the URL of page without reloading the page?**  
use pushState javascript function.   
**For Example:**

window.history.pushState('page2', 'This is page Title', '/newpage.php');

**Question: How to convert JSON Object to String?**

var myobject=['Web','Technology','Experts','Notes']

JSON.stringify(myobject);

**Question: How to convert JSON String to Object?**

var jsonData = '{"name":"web technology","year":2015}';

var myobject = JSON.parse(jsonData);

console.log(myobject);

**Question: How to check an variable is Object OR String OR Array?**  
Use below function to get Data type of javascript variable.

function checkDataType(someVar){

result ='String';

if(someVar instanceof Object){

result ='Object'

}

if($.isArray(someVar)){

result = 'Array';

}

return result;

}

var someVar= new Array("Saab", "Volvo", "BMW");

console.log(result);

**Question: Can i declare a variable as CONSTANT like in PHP?**  
**No,** I think cosntant not exist in javascript.   
But you can follow same type convention to declare constant.

var CONSTANT\_NAME = "constant value";

**Question: How to open URL in new tab in javascript?**  
use javascript, window.open function.

window.open('http://www.web-technology-experts-notes.in/','\_blank');

**Question: What is difference between undefined and object?**  
**undefined**means some variable's value is not defined yet.  
**object**means variables's value is defined that is either function, object OR array.  
  
**With use of below, you can easily determine whether it is object OR NULL.**

console.log(typeof(null)); // object

console.log(typeof(undefined)); // undefined

**Question: How to get current date in JavaScript?**

var today = new Date();

console.log(today);

**Question: How do I declare a namespace in JavaScript?**

var myNamespace = {

function1: function() { },

function2: function() { }

function3: function() { }

};

myNamespace.function3();

**Question: What is the best way to detect a mobile device in jQuery?** 

if( /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(navigator.userAgent) ) {

}

**Question: How to detect mobiles including ipad using navigator.useragent in javascript?**

if(navigator.userAgent.match(/Android/i) || navigator.userAgent.match(/webOS/i) || navigator.userAgent.match(/BlackBerry/i) || navigator.userAgent.match(/iPhone/i)){

console.log('Calling from Mobile');

}else{

console.log('Calling from Web');

}

**Question: How to detect mobiles including ipad using navigator.useragent in javascript?**

if(navigator.userAgent.match(/Android/i) || navigator.userAgent.match(/webOS/i) || navigator.userAgent.match(/BlackBerry/i) || navigator.userAgent.match(/iPhone/i)){

console.log('Calling from Mobile');

}else{

console.log('Calling from Web');

}

**<xsl:key>** tag element specifies a named name-value pair assigned to a specific element in an XML document. This key is used with the key() function in XPath expressions to access the assigned elements in a XML document.

<xsl:key

name = QName

match = Pattern

use = Expression

</xsl:key>

|  |  |
| --- | --- |
| name | Name of the key to be used. |
| match | Patterns used to identify a node that holds this key. |
| use | XPath expression to identified the value of the nodes of xml document. |

EJB

Question1: What do you mean by EJB?

Ans: Most of the time this EJB interview questions is the first questions asked to interviewee, mostly to check how he responds and how its is different than traditional Java beans. Enterprise java bean is a server side component which runs on application server or we call container, developed for the purpose of distributed and enterprise level application .container will provide support for system level services like [Transaction Management](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html), security which make developer task easy and he can focus on business logic.

Question 2: What are the different types of EJB?

This is another beginner level EJB interview questions mostly asked in telephonic interviews and appeared in 2 to 3 years experience level interviews. Mainly three types of EJB, Entity Bean, Session Bean and Message Driven Bean(MDB).

Types of EJB:

Entity Bean: it represents an entity which is mapped with database or we can say it makes OR object Relational mapping with Database. Entity bean typically represent table in RDBMS and each instance represent row in the table.

 Two types of entity bean:

          CMP Entity bean: Container managed entity bean its responsibility of container to manage the bean persistence behavior.

          BMP Entity bean: Programmer manage the bean persistence behavior.

Session bean: Session bean is responsible for developing business logic it makes the client server relationship so session beans exist till there is a session exist between client and server, it doesn’t contain persistent business concept.

 Types of session bean

          Stateless session bean: when there is not need to maintain state of a particular client stateless session bean is used .They alive for short period of time.

       For example if we are validating the credit card we can use stateless session bean.

          Stateful session bean: stateful session bean maintain the conversational state of client over the series of method call before the bean instance goes to passive state conversational state is saved to persistence area like Hard disk and again when same client send a request and bean instance come into the active state it will come out from hard disk to main memory.

      For Example when we do online banking transaction, online reservation we use stateful session bean

Message Driven Beans: these beans are work as a listener for messaging services like JMS.

Question 3: Explain the life cycle method of EJB?

One for EJB interview questions which is asked in Junior level interviews. I have not seen this EJB question appeared on more senior level interviews but its good to remember life cycle as here you have chance to show how much EJB you know in depth.

Ans: Life Cycle of Entity Bean:

 First stage is Does Not Exist Stage then Container creates the instance of EJB and  call SetEntityContext() method  which will set all entity context to bean and now it will become available on pool ,to get a particular identity of an EJB object it has to move from Pooled stage to ready stage which is done by calling the create() method which in turns call ejbCreate() and ejbPostCreate() method .

There is another way by which directly entity bean can move to pooled stage to ready stage that’s is call ejbActivate() method. now we are ready to invoke the business method of entity bean .After completion of business method if we want to move again in pooled stage from ready stage we can call remove() method which in turns call ejbRemove() or directly call ejbPassivate () method. At the end container remove the instance of EJBfrom pool and call unSetEntityContext().

Life Cycle of Stateful Session Bean :

Stateful session beans life cycle starts when  client call create() method.The container create the instance of session bean and call setSessionContext() and ejbCreate() method

Now the stateful session bean is ready to serve the client request after serving the request if it is not used after a long time container can move this bean to passive stage by calling the ejbPassivate() method. Similarly when bean is in passive stage and client invoke the business method the container call ejbActivate() method to move bean from passive stage to active or ready stage.

At the end of life cycle client  call remove() method and container will call ejbRemove() method and bean is ready for garbage collection.

Life Cycle of Stateless session bean :

Stales session bean has short life cycle it can have two stage does not exist and ready stage. ejb container create the instance of stateless session bean and call setSessionContext () and ejbCreate() method.Now the bean is ready to invoke business method on this.it will not maintain the state so remove () method is been called after completion of business method which in turns  call ejbRemove () and now its ready for  garbage collection.

Life cycle of Message Driven bean:

MDBs have same life cycle like stateless session bean. setMessageDrivenContext() method and ejbCreate() method is called by container to create the instance of MDB.now its ready to receive message .and at the end of lifecycle client will call remove () method

which in turns  call ejbRemove () and now its ready for  garbage collection.

Question 4 : Can we have static initializer Block in EJB.

This is one of the tricky EJB interview questions which makes you think and some time left you stunned with feeling like “ Ah , I haven’t thought about it”.

Ans .Purpose of Static initialize block is to initialize some static fields before any execution of constructor or method or we can say creation of object. According to EJB Spec it’s a violation if static field are used and they are non final .

“EJB Spec”

Enterprise bean are not allowed to read or write the non final fields.

But technically, from the point of java its correct but if in EJB we use static initializer block to initialize static field then because EJB components are used in distributed environment mean to say if  single JVM then it will be ok but if run on different JVM then it will be a problem if we change or update the value in on environment then only the instance running on same JVM have new value .that’s why static blocks are avoided and also all static field should be final.

And same thing we can achieve in ejbCreate(), setSessionContext() or setEntityContext() methods.

Question 5: Is threading possible in EJB?

Another tricky EJB interview question, I love to ask this kind of question because it shows real knowledge of EJB which is important to avoid mistakes while writing enterprise Java application.

Ans: No, Not possible because EJBs are created and managed by container and if in ejbs we allow threading containers life cycle methods will be interrupted by us and also the purpose of EJB is to perform business logic not to controlling a system or implementation level functioning so container will manage the thread and developer can concentrate on business logic.

Question 6: What is connection pooling feature of EJB container?

This EJB interview question is neither tough nor easy, as most of Java programmer familiar with concept of connection pooling that makes it easy but how EJB container does that is something you want to hear from interviewee.

Ans: Connection pooling is one of the Advance feature of container which enhanced our application performance .Using connection pooling mechanism client are not required to create every time connection object to interact with database .Here in pool objects are already available Whenever a client request for a database connection then an instance is picked from the connection pool to get an access to database and when user complete with his work instance is returned to connection pool. There is a limit specified by App server administrator for the availability of number of connections and beyond a specified limit a predefined number increases numbers of connection pool instances. When demand goes back to normal then access amount of connection pool instances are removed.

Question 7: what is Session facade?

This is the most popular interview questions asked in EJB in connection of Session Bean. Session facade not only touch concept of EJB but also it checks design patterns knowledge of interviewee.

Ans: Session Façade is one of the design pattern which is used in EJB component in this pattern session beans are used as a wrapper classes for entity bean and using session bean we interact with Entity bean .it will give clean approach towards client access to bean components and also reduce network calls so as to make the whole system of high performance.

Question 8:What are the transaction Attribute ?

This kind of EJB questions mostly asked in telephonic interviews but believe me its not easy to answer if you have not really used transaction management feature of EJB. Its easy to mug all these transaction attribute and spill in interview but once interviewer asked cross question you will most likely cripple.

Ans: Transaction is group of operation or work which should be performed either completely or none to maintain the data integrity. All transaction must have ACID property (atomicity, consistency, integrity,durability) so transaction can be said completed if we commit on successful execution and rollback on unsuccessful execution.

There are two ways of transaction management.

          Declarative Transaction Mang.

          Programmatic Transaction Mang.

Now we see what the transactions Attribute are

Transaction Attribute conveys to the container the intended transactional behavior of the associated EJB component's method.

Required: if required attribute is associated with Method then new Transaction context may or may not be created, means if method is already associated with transaction context then no new transaction context should be created.

Requires New: if Requires New attribute is associated with Method then always new Transaction context may be created.

NotSupported: if Method is Associated with this Attribute then method is  a not a part of transaction.

Supported: if a Method is Associated with this transaction Attribute then method will act as Not supported if calling component is not associated with transaction and if calling component is associated with transaction then act as a required Attribute.

Mandatory: if a method is Associated with this attribute then always be called from calling component transaction context.

Never: if a method is associated with this attribute then never be called from calling component transaction context.

The default transaction value for EJB is **Support.**

**Question 9: Difference Between ejbStore() and ejbLoad()?**

One more popular EJB interview question, not seen though it for some time but still an important questions on EJB to prepare.

Ans: Both of these methods are used in context of entity bean for the purpose of synchronizing the instance variable to corresponding value store in the database.ejbLoad method load or refresh the instance variable from the database and ejbStore method writes variables value back to the database.

**Question 10**:**What is the difference between JNDI context, Initial context, session context and ejb context.**

JNDI questions most likely asked in any EJB interview so be prepare for that as well.

**Ans:**

**JNDI context**: provides mechanism to lookup resources on network.

**Initial Context**: it provides initial context to resources.

**Session Context**: it has all the information available which is required to session bean from container.

**EjbContext**: it contain information which is required to both session and entity bean.

The high-level  interfaces are:

* ConnectionFactory: An administered object that creates a Connection.
* Connection: An active connection to a provider.
* Destination: An administered object that encapsulates the identity of a message destination, such as where messages are sent to or received from.
* Session: A single-threaded context for sending and receiving messages. For reasons of simplicity and because Sessions control transactions, concurrent access by multiple threads is restricted. Multiple Sessions can be used for multithreaded applications.
* MessageProducer: Used for sending messages.
* MessageConsumer: Used for receiving messages.

The following table identifies the domain-specific interfaces inherited from each high-level interface.

|  |  |  |
| --- | --- | --- |
| High-levelinterface | PTP domain | Pub/sub domain |
| ConnectionFactory | QueueConnectionFactory | TopicConnectionFactory |
|  |  |  |
| Connection | QueueConnection | TopicConnection |
|  |  |  |
| Destination | Queue | Topic |
|  |  |  |
| Session | QueueSession | TopicSession |
|  |  |  |
| MessageProducer | QueueSender | TopicPublisher |
|  |  |  |
| MessageConsumer | QueueReceiver, | TopicSubscriber |

A JMS application is composed of the following parts:

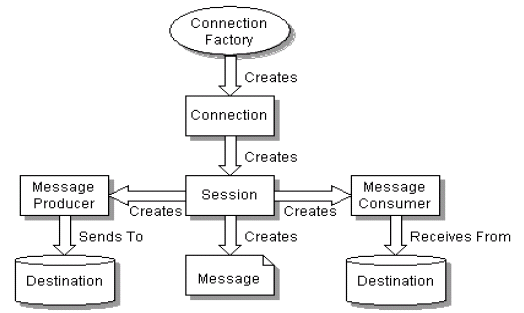
* A JMS provider: A messaging system that implements the JMS specification.
* JMS clients: Java applications that send and receive messages.
* Messages: Objects that are used to communicate information between JMS clients.
* Administered objects: Preconfigured JMS objects that are created by an administrator for the use of JMS clients.

The JMS specification defined six type or classes of messages that a JMS provider must support:

* Message: This represents a message without a message body.
* StreamMessage: A message whose body contains a stream of Java primitive types. It is written and read sequentially.
* MapMessage: A message whose body contains a set of name/value pairs. The order of entries is not defined.
* TextMessage: A message whose body contains a Java string…such as an XML message.
* ObjectMessage: A message whose body contains a serialized Java object.
* BytesMessage: A message whose body contains a stream of uninterpreted bytes.

### Producing and Consuming Messages

Here are the necessary steps to develop clients to produce and consumer messages. Note that there are some common steps that shouldn’t be duplicated if the client is both producing and consuming messages. Figure below depicts the high-level view of the steps

[](http://www.j2eebrain.com/wp-content/uploads/Producing-and-Consuming-Messages.gif)

\* JMSDeliveryMode —   type int

Contains the value DeliveryMode.PERSISTENT or

DeliveryMode.NON\_PERSISTENT. A persistent message is delivered “once and only once”; a non-persistent message is delivered “at most once.” Be aware that “at most once” includes not being delivered at all. A non-persistent message may be lost by a provider during application or system failure. Extra care will be taken to assure that a persistent message is not affected by failures. There is often considerable overhead in sending persistent messages, and the trade-offs between reliability and performance must be carefully considered when deciding the delivery mode of a message.

 JMSPriority —   type int

The priority of the message; set by the provider during the send process. A priority of 0 is the lowest priority; a priority of 9 is the highest priority

A typical JMS program goes through the following steps to begin producing and consuming messages.  
1. Look up a ConnectionFactory through JNDI.  
2. Look up one or more Destinations through JNDI.  
3. Use the ConnectionFactory to create a Connection.  
4. Use the Connection to create one or more Sessions.  
5. Use a Session and a Destination to create the required MessageProducers and MessageConsumers.  
6. Start the Connection.  
At this point, messages can begin to flow and the application can receive, process, and send messages, as required.

Transactions

A JMS transaction groups a set of produced messages and a set of consumed messages into an atomic unit of work. If an error occurs during a transaction, the production and consumption of messages that occurred before the error can be “undone.”

Session objects control transactions and a Session may be denoted as transacted when it is created. A transacted Session always has a current transaction, that is, there is no begin(); commit() and rollback() end one transaction and automatically begin another.

Distributed transactions may be supported by the Java Transaction API (JTA) XAResource API, though this is optional for providers.

Acknowledgement

Acknowledgement is the mechanism whereby a provider is informed that a message has been successfully received.

If the Session receiving the message is transacted, acknowledgement is handled automatically. If the Session is not transacted, then the type of acknowledgement is determined when the Session is created.

There are three types of acknowledgement:

\* Session.DUPS\_OK\_ACKNOWLEDGE: Lazy acknowledgement of message delivery; reduces overhead by minimizing work done to prevent duplicates; should only be used if duplicate messages are expected and can be handled.

\* Session.AUTO\_ACKNOWLEDGE: Message delivery is automatically acknowledged upon completion of the method that receives the message.

\* Session.CLIENT\_ACKNOWLEDGE: Message delivery is explicitly acknowledged by calling the acknowledge() method on the Message.

#### What are the types of messaging?

There are two kinds of Messaging.  
Synchronous Messaging: Synchronous messaging involves a client that waits for the server to respond to a message.  
Asynchronous Messaging: Asynchronous messaging involves a client that does not wait for a message from the server. An event is used to trigger a message from a server.

#### What are the advantages of JMS?

JMS is asynchronous in nature. Thus not all the pieces need to be up all the time for the application to function as a whole. Even if the receiver is down the MOM

will store the messages on it’s behalf and will send them once it comes back up. Thus at least a part of application can still function as there is no blocking.

#### What is the basic difference between Publish Subscribe model and P2P model?

Publish Subscribe model is typically used in one-to-many situation. It is unreliable but very fast. P2P model is used in one-to-one situation. It is highly reliable.

What is the use of BytesMessage?

BytesMessage contains an array of primitive bytes in it’s payload. Thus it can be used for transfer of data between two applications in their native format which may

not be compatible with other Message types. It is also useful where JMS is used purely as a transport between two systems and the message payload is opaque to the JMS

client. Whenever you store any primitive type, it is converted into it’s byte representation and then stored in the payload. There is no boundary line between the

different data types stored. Thus you can even read a long as short. This would result in erroneous data and hence it is advisable that the payload be read in the same

order and using the same type in which it was created by the sender.

What is the use of StreamMessage?

StreamMessage carries a stream of Java primitive types as it’s payload. It contains some conveient methods for reading the data stored in the payload. However

StreamMessage prevents reading a long value as short, something that is allwed in case of BytesMessage. This is so because the StreamMessage also writes the type

information alonwgith the value of the primitive type and enforces a set of strict conversion rules which actually prevents reading of one primitive type as another.

What is the use of TextMessage?

TextMessage contains instance of java.lang.String as it’s payload. Thus it is very useful for exchanging textual data. It can also be used for exchanging complex

character data such as an XML document.

What is the use of ObjectMessage?

ObjectMessage contains a Serializable java object as it’s payload. Thus it allows exchange of Java objects between applications. This in itself mandates that both the

applications be Java applications. The consumer of the message must typecast the object received to it’s appropriate type. Thus the consumer should before hand know

the actual type of the object sent by the sender. Wrong type casting would result in ClassCastException. Moreover the class definition of the object set in the payload

should be available on both the machine, the sender as well as the consumer. If the class definition is not available in the consumer machine, an attempt to type cast

would result in ClassNotFoundException. Some of the MOMs might support dynamic loading of the desired class over the network, but the JMS specification does not

mandate this behavior and would be a value added service if provided by your vendor. And relying on any such vendor specific functionality would hamper the portability

of your application. Most of the time the class need to be put in the classpath of both, the sender and the consumer, manually by the developer.

What is the use of MapMessage?

A MapMessage carries name-value pair as it’s payload. Thus it’s payload is similar to the java.util.Properties object of Java. The values can be Java primitives or

their wrappers.

What is the difference between BytesMessage and StreamMessage?

BytesMessage stores the primitive data types by converting them to their byte representation. Thus the message is one contiguous stream of bytes. While the

StreamMessage maintains a boundary between the different data types stored because it also stores the type information along with the value of the primitive being

stored. BytesMessage allows data to be read using any type. Thus even if your payload contains a long value, you can invoke a method to read a short and it will return

you something. It will not give you a semantically correct data but the call will succeed in reading the first two bytes of data. This is strictly prohibited in the

StreamMessage. It maintains the type information of the data being stored and enforces strict conversion rules on the data being read.

#### What is point-to-point messaging?

With point-to-point message passing the sending application/client establishes a named message queue in the JMS broker/server and sends messages to this queue. The

Receiving client registers with the broker to receive messages posted to this queue. There is a one-to-one relationship between the sending and receiving clients.

#### What is the advantage of persistent message delivery compared to nonpersistent delivery?

If the JMS server experiences a failure, for example, a power outage, any message that it is holding in primary storage potentially could be lost. With persistent storage, the JMS server logs every message to secondary storage. (The logging occurs on the front end, that is, as part of handling the send operation from the message producing client.) The logged message is removed from secondary storage only after it has been successfully delivered to all consuming clients.

AngularJS

1. **What is $injector and $inject?**

$injector is a service which is used to invoke controller functions, service functions, filter functions, and any other function that might need dependencies as parameters. Angular creates only a single $injector object.

$inject is property which is used to inject the dependencies of a function as an array of strings.

MyController['$inject'] = ['$scope', 'greeter']; //inject dependencies as an array of strings

1. **What is Dependency Injection in AngularJS?**

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies. AngularJS comes with a built-in dependency injection mechanism. You can divide your AngularJS app into multiple different types of components which AngularJS can inject into each other.

1. **What is Routing in AngularJS?**

AngularJS Routing helps you to divide your app into multiple views and bind different views to Controllers. The magic of Routing is taken care by an AngularJS service $**routeProvider**. $**routeProvider** service provides method when() and otherwise() to define the routes for your app. Routing has dependency on ngRoute module.

1. **What is difference between $interval and window. setInterval in AngularJS?**

$interval is an Angular service which wraps the browser's window. setInterval() function. It is also used to call a JavaScript function repeatedly within a time interval.

1. **What is difference between $timeout and window.setTimeout in AngularJS?**

$timeout is an Angular service which wraps the browser's window.setTimeout() function into a try/catch block and delegates any exceptions to $exceptionHandler service. It is used to call a JavaScript function after a given time delay. The $timeout service only schedules a single call to the function.

1. **What is Restangular?**

Restangular is an Angular service specifically designed simply to fetch data from the rest of the world. To use Restangular you need to link the restangular.js file and include restangular resource as a dependency within your angular app.

var app = angular.module('myApp', ['restangular']);

app.controller('MainController',function ($scope, Restangular) { //TO DO: });

1. **What is $q service and when to use it?**

$q is a service that helps you to run functions asynchronously, and use their return values when they have done processing.

$q service is said to be inspired by Chris Kowal's Q library which allow users to monitor asynchronous progress by providing a "promise" as a return from a call.

It is good when you need to process a number of asynchronous activities simultaneously. The $q.all() function lets you trigger several callbacks at the same time, and use a single then function to join them all together.

1. **What methods $resource service object support?**

The $resource service object supports the following methods:

1. get() 2. query() 3. save() 4. remove() 5. delete()
2. **How to enable caching in $http service?**

You can enable caching in $http service by setting configuration property cache to true. When cache is enabled, $http service stores the response from the server in local cache. In this way, next time the response will be served from the cache without sending request to server.

$http.get("http://server/myserviceapi",

{ cache:true }).sucess(function()

{ //TO DO: })

1. **What methods $http service support? Ans. The $http service supports the following methods?**

1. $http.get() 2. $http.head() 3. $http.post() 4. $http.put() 5. $http.delete() 6. $http.jsonp() 7. $http.patch()

**11. What is the difference between $http and $resource?**

$http service is a core Angular service which allows you to make AJAX requests by using GET, HEAD, POST, PUT, DELETE, JSONP and PATCH methods. It is very much like the $.ajax() method in jQuery. It can be used with RESTful and Non-RESTful server-side data sources.

$http is good for quick retrieval of server-side data that doesn’t really need any specific structure or complex behaviors.

$resource warps $http and allows you to interact with RESTful server-side data sources. It requires the ngResource module to be installed which exist in angular-resource.js

$http is good for retrieval of RESTful server-side data sources that might need any specific structure or complex behaviors.

1. **What is difference between value and constant?**

Value and Constant are simple objects which are used to share data globally with in a module.

Value - A value can be a number, string, date-time, array or object. You can also register a function as a value. Values are typically used as configuration which is injected into factories, services or controllers.

//define module var app = angular.module('app', []);

//define value app.value("numberValue", 100); app.value("stringValue", "dotnet-tricks.com"); app.value("objectValue", { name: "dotnet-tricks.com", totalUsers: 120000 });

Constant - A constant is like as value. The difference between a value and a constant service is that constant service can be injected into a module configuration function i.e. config() but value service cannot be.

//define module var app = angular.module('app', []);

//define constant app.constant("mynumberValue", 200); app.constant("mystringValue", "webgeekschool.com");

//module configuration function app.config(function (mynumberValue) { //here value objects can't be injected console.log("Before:" + mynumberValue); mynumberValue = "New Angular Constant Service"; console.log("After:" + mynumberValue); });

**What is the difference between Factory, Service and Provider?**

1. **Factory** - A factory is a simple function which allows you to add some logic before creating the object. It returns the created object

When to use: It is just a collection of functions like a class. Hence, it can be instantiated in different controllers when you are using it with constructor function.

1. **Service** - A service is a constructor function which creates the object using new keyword. You can add properties and functions to a service object by using this keyword. Unlike factory, it doesn’t return anything.

When to use: It is a singleton object. Use it when you need to share a single object across the application.

1. **Provider** - A provider is used to create a configurable service object. It returns value by using $get() function.

When to use: When you need to provide module-wise configuration for your service object before making it available.

**What are different ways to create service in AngularJS?**

There are five ways to create a service as given below:

1. Service 2. Factory 3. Provider 4. Value 5. Constant

**What is Service in AngularJS?**

A service is a reusable singleton object which is used to organize and share code across your app. A service can be injected into controllers, filters, directives.

AngularJS offers several built-in services (like $http, $provide, $resource, $window, $parse) which always start with $ sign.

**What are different states of a form in AngularJS?**

The AngularJS form goes to the following states, starting from the form rendering and when the user has finished the filling of form.

State 1: pristine and invalid - When the form is first time rendered and the user has not interacted with the form yet.

State 2: dirty and invalid - User has interacted with the form, but form validity has not been satisfied, yet.

State 3: dirty and valid - User has finished the filling of form and the entire form validations has been satisfied

**What are different Angular form properties?**

Angular provides properties on form which help you to get information about a form or its inputs and to validate them.

$valid - It is a boolean property that tells whether the form or it's inputs are valid or not. If all containing form and controls are valid, then it will be true, otherwise it will be false.

Syntax

formName.$valid formName.inputFieldName.$valid

$invalid - It is a boolean property that tells whether the form or it's inputs are invalid or not. If at least one containing form and control is invalid then it will be true, otherwise it will be false.

Syntax

formName.$invalid formName.inputFieldName.$invalid

$pristine - It is a boolean property that tells whether the form or it's inputs are unmodified by the user or not. If the form or its inputs are unmodified by the user, then it will be true, otherwise it will be false.

Syntax

formName.inputFieldName.$pristine

$dirty - It is a boolean property that is actually just reverse of pristine i.e. it tells whether the form or it's inputs are modified by the user or not. If the form or its inputs are modified by the user, then it will be true, otherwise it will be false.

Syntax

formName.$dirty formName.inputFieldName.$dirty

$error - This is an object hash which contains references to all invalid controls or forms. It has all errors as keys: where keys are validation tokens (such as required, url or email) and values are arrays of controls or forms that are invalid with given error. For a control, if a validation fails then it will be true, otherwise it will be false.

Syntax

formName.$error formName.inputFieldName.$error

**What is View in AngularJS?**

The view is responsible for presenting your models data to end user. Typically it is the HTML markup which exists after AngularJS has parsed and compiled the HTML to include rendered markup and bindings.

**How to apply validation in AngularJS?**

AngularJS provides you built-in validation directives to validate form client side. This makes your life pretty easy to handle client-side form validations without adding a lot of extra effort. AngularJS form validations are based on the HTML5 form validators.

AngularJS directives for form validation. Here is a list of AngularJS directive which can be applied on an input field to validate its value.

**How to share information between controllers in AngularJS?**

**OR**

**What are the ways to communicate between controllers in AngularJS?**

There are various different ways to share data between controllers in an AngularJS app. The most commonly used are **Scope**, **Service**, **Factory** and **Providers.**

**Does AngularJS support MVC?**

AngularJS is a MVC framework. It does not implement MVC in the traditional way, but rather something closer to MVVM Model-View-ViewModel).

**What is Model in AngularJS?**

Models are plain old JavaScript objects that represent data used by your app. Models are also used to represent your app's current state**.**

**What is ViewModel in AngularJS?**

A view model is an object that provides specific data and methods to maintain specific views. Basically, it is a $scope object which lives within your AngularJS app's controller. A view model is associated with a HTML element with the ng-model and **ng-bind** directives.

**What is Controller in AngularJS?**

The controller defines the actual behavior of your app. It contains business logic for the view and connects the model to view with the help of $scope. A controller is associated with a HTML element with the **ng-controller** directive.

AngularJS supports Single Page Application via multiple views on a single page. To do this AngularJS has provided ng-view and ng-template directives and $routeProvider services.

**ng-view**

ng-view tag simply creates a place holder where a corresponding view (html or ng-template view) can be placed based on the configuration.

**ng-template**

ng-template directive is used to create an html view using script tag. It contains "id" attribute which is used by $routeProvider to map a view with a controller.

**Custom directives:**

Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile () method of the custom directive then process the element using link () method of the custom directive based on the scope of the directive. AngularJS provides support to create custom directives for following type of elements.

* **Element** directives − Directive activates when a matching element is encountered.
* **Attribute** − Directive activates when a matching attribute is encountered.
* **CSS** − Directive activates when a matching css style is encountered.
* **Comment** − Directive activates when a matching comment is encountered.

What is the difference between the JRE and the JDK ?

|  |  |
| --- | --- |
| JRE  (Java Runtime environment) | JDK (Java Development Kit) |
| It is an implementation of the Java Virtual Machine\* which actually executes Java programs. | It is a bundle of software that you can use to develop Java based applications. |
| Java Runtime Environment is a plug-in needed for running java programs. | Java Development Kit is needed for developing java applications. |
| The JRE is smaller than the JDK so it needs less Disk space. | The JDK needs more Disk space as it contains the JRE along with various development tools. |
| The JRE can be downloaded/supported freely from [java.com](http://www.java.com/) | The JDK can be downloaded/supported freely from [oracle.com/technetwork/java/javase/downloads/](http://www.oracle.com/technetwork/java/javase/downloads/index.html) |
| It includes the JVM , Core libraries and other additional components to run applications and applets written in Java. | It includes the JRE, set of API classes, Java compiler, Webstart and additional files needed to write Java applets and applications. |

What is the difference between the JRE and the Java SE platform?

|  |  |  |
| --- | --- | --- |
|  | JRE  (Java Runtime Environment) | Java SE (Java Platform, Standard Edition) |
| Who needs it? | Computer users who run applets and applications written using Java technology | Software developers who write applets and applications using Java technology |
| What is it? | An environment required to run applets and applications written using the Java programming language | A software development kit used to write applets and applications using the Java programming language |
| How do you get it? | Distributed freely and is available from:  [java.com](https://www.java.com/en/) | Distributed freely and is available from:  [oracle.com/javase](http://oracle.com/javase) |

**JSP**

**Difference Between include Directive and include Action of JSP?**

|  |  |
| --- | --- |
| **Include Directive** | **Include Action** |
| include directive is processed at the translation time | Include action is processed at the run time. |
| include directive can use relative or absolute path | Include action always use relative path |
| Include directive can only include contents of resource it will not process the dynamic resource | Include action process the dynamic resource and result will be added to calling JSP |
| We can not pass any other parameter | Here we can pass other parameter also using JSP:param |
| We cannot  pass any request or response object to calling jsp to included file or JSP or vice versa | In this case it’s possible. |

**Is it possible for one JSP to extend another java class if yes how?**

Yes it is possible we can extends another JSP using this **<%@ include page extends="classname" %>** it’s a perfectly correct because when JSP is converted to servlet its implements javax.servlet.jsp.HttpJspPage interface, so for jsp page its possible to extend another java class . This question can be tricky if you don’t know some basic fact J, though it's not advisable to write java code in jsp instead it's better to use expression language and tag library.

**scope="page | request | session | application"**

The scope in which the Bean exists and the variable named in id is available. The default value is page. The meanings of the different scopes are shown below:

* **page** – we can use the Bean within the JSP page with the <jsp:useBean> element.
* **request** – we can use the Bean from any JSP page processing the same request, until a JSP page sends a response to the client or forwards the request to another file.
* **session** – we can use the Bean from any JSP page in the same session as the JSP page that created the Bean. The Bean exists across the entire session, and any page that participates in the session can use it..
* **application** – we can use the Bean from any JSP page in the same application as the JSP page that created the Bean. The Bean exists across an entire JSP application, and any page in the application can use the Bean.

**What are the implicit Objects?**

Ans: This is a fact based interview question what it checks is how much coding you do in JSP if you are doing it frequently you definitely know them. Implicit object is the object that is created by web container provides to a developer to access them in their program using JavaBeans and Servlets. These objects are called implicit objects because they are automatically instantiated.they are bydefault available in JSP page.

They are **request, response, pageContext, session, and application, out, config, page, and exception**.

**In JSP page how can we handle runtime exception?**  
  
 We can use the errorPage attribute of the page directive to have uncaught run-time exceptions automatically forwarded to an error processing page.

Example: **<%@ page errorPage="error.jsp" %>**

It will redirect the browser to the JSP page error.jsp if an uncaught exception is encountered during request processing. Within error.jsp, will have to indicate that it is an error-processing page, using the directive: <%@ page isErrorPage="true" %>

**Why is \_jspService() method starting with an '\_' while other life cycle methods do not?**

main JSP life cycle method are **jspInit**() **jspDestroy**() and \_**jspService**() ,by default whatever content we write in our jsp page will go inside the \_jspService() method by the container if again will try to override this method JSP compiler will give error but we can override other two life cycle methods as we have implementing these two in jsp so making this difference container use \_ in jspService() method and shows that we cant override this method.

What is the need of tag library?

Tag library is a collection of custom tags. Custom actions helps recurring tasks will be handled more easily they can be reused across more than one application and increase productivity. JSP tag libraries are used by Web application designers who can focus on presentation issues rather than being concerned with how to access databases and other enterprise services. Some of the popular tag libraries are Apache display tag library and String tag library. You can also check my post on display [tag library example on Spring](http://javarevisited.blogspot.com/2011/09/displaytag-examples-tutorial-jsp-struts.html).

Unix Shell Script:

Assume we create a test.sh script. Note all the scripts would have .sh extension. Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example −

#!/bin/sh

This tells the system that the commands that follow are to be executed by the Bourne shell. It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang.

To create a script containing these commands, you put the shebang line first and then add the commands −

#!/bin/bash

pwd

ls

Oracle

What is Oracle table?

A table is the basic unit of data storage in an oracle database. The tables of a database hold all of the user accessible data. Table data is stored in rows and columns.

What are the Clusters?

Clusters are groups of one or more tables physically stores together to share common columns and are often used together.

What is an Index?

An Index is an optional structure associated with a table to have direct access to rows, which can be created to increase the performance of data retrieval. Index can be created on one or more column of a table.

What are the advantages of views?

Provide an additional level of table security,by restricting access to a predetermined set of rows and columns of a table.

Hide data complexity.

Simplify commands for the user.

Present the data in a different perspective from that of the base table.

Store complex queries.

What are the various types of queries?

The types of queries are

Normal queries

Sub Queries

Co-related queries

Nested queries

Compound queries

What is the difference between clustered and a non-clustered index?

A clustered index is a special type of index that reorders the way records in the table are physically stored. Therefore table can have only one clustered index. The leaf nodes of a clustered index contain the data pages.

A non clustered index is a special type of index in which the logical order of the index does not match the physical stored order or the rows on disk.

What is a tablespace?

A database is divided into logical storage unit called tablespace. A tablespace is used to grouped related logical structures together.

Why use materialized view instead of a table?

Materialized views are basically used to increase query performance since it contains results of a query. They should be used for reporting instead of a table for a faster execution.

What does ROLLBACK do?

ROLLBACK retracts any of the changes resulting from the SQL statements in the transaction.

Compare and contrast TRUNCATE and DELETE for a table?

Both the truncate and delete command have the desired outcome of getting rid of all the rows in a table. The difference between in a table. The difference between the two is that the truncate command is a DDL operation and just moves the high water mark and produces a now roll back. The Delete command, on the other hand, is a DML operation which will produce a rollback and thus make longer to complete.

What is null value?

Null value is neither zero nor it is a blank space. It is some unknown value which occupies 4 bytes of space of memory in SQL.

Define transaction?

A transaction is a sequence of SQL statements that Oracle Database treats as a single unit.

What is the difference between SQL & Oracle?

SQL is Structured Query Language. Oracle is a Database. SQL is used to write queries against Oracle DB.

What are different Oracle database objects?

Tables, Views, Indexes, Synonyms, Sequences, Tablespaces.

What is Hash Cluster?

A row is stored in a hash cluster based on the result of applying a hash function to the row’s cluster key value. All rows with the same hash key value or stores together on disk.

What is a User\_exit?

Calls the user exit named in the user\_exit\_string. Invokes a 3GI program by name which has been properly linked into your current oracle forms executable.

What is schema?

A schema is collection of database objects of a user.

What are Roles?

Roles are named groups of related privileges that are granted to users or other roles.

What are the dictionary tables used to monitor a database spaces?

DBA\_FREE\_SPACE

DBA\_SEGMENTS

DBA\_DATA\_FILES

What is a SNAPSHOT?

Snapshots are read-only occupies of a master table located on a remote node which is periodically refreshed to reflect changes made to the master table.

What is a database instance?

A database instance is a set of memory structure and background processes that access a set of database files. The processes can be shared by all the users.

What are parameters?

Parameters provide a simple mechanism for defining and setting the values of inputs that are required by a form at startup. Form parameters are variables of type char, number, date, that you define at design time.

What are the different file extensions that are created by oracle reports?

Rep file and Rdf file.

What are Clusters?

Clusters are groups of one or more tables physically stores together to share common columns and are often used together.

What is difference between SUBSTR and INSTR?

INSTR function search string for sub-string and retruns an integer indicating the position of the character in string that is the first character of this occurrence of this occurrence. SUBSTR function retruns a portion of string, beginning at character position, substring, substring\_length charcters long. SUBSTR calculates lengths using characters as defined by the input character set.

Define a View?

A view is a virtual table which is based on the one or more physical tables and views.

What is the difference between a view and a synonym?

Synonym is just a second name of table used for multiple link of database. View can be created with many tables, and with virtual columns and with conditions. But synonym can be on view.

SAVEPOINTS

SAVEPOINTS are used to subdivide a transition into smaller parts. It enables rolling back part of a transaction. Maximum of five save points are allowed.

PRECOMPILERS

A precompiler is a tool that allows programmers to embed SQL statements in high-level source programs like C, C++, COBOL, etc. The precompiler accepts the source program as input, translates the embedded SQL statements into standard Oracle runtime library calls, and generates a modified source program that one can compile link, and execute in the usual way.

When do you use WHERE clause and when do you use HAVING clause?

The WHERE condition lets you restrict the rows selected to thosee that satisfy one or more conditions. Use the HAVING clause to restrict the groups of returned rows to those groups for which the specified condition is TRUE.

What are the differences between Stored Procedures and Triggers?

A stored procedure are compiled collection of programs or SQL statements that live the database. A stored procedure can access and modify data present in many tables. Also a stored procedure is not associated with any particular database object. But triggers are event-driven special procedures which are attached to a specific database object.

What must be installed with ODBC on the client in order for it to work with Oracle?

SQLNET and PROTOCOL layers of the transport programs.

What is Trigger?

Triggers are special kind of stored procedures that get executed automatically when an INSERT,UPDATE or DELETE operation takes place on a table.

Explain the difference between a data block, an extent and a segment?

A data block is the smallest unit of logical storage for a database object. As objects grow they take chunks of additional storage that composed of contiguous data blocks. These groupings of contiguous data blocks are called extents. All the extents that an object takes when grouped together are considered the segment of the database object.

What is bind reference and how can it be created?

Bind reference are used to replace the single value in SQL, PL/SQL statements a bind reference can be created using a (:) before a column or a parameter name.

How many LONG columns are allowed in a table? Is it possible to use LONG columns in WHERE clause or ORDER BY?

Only one LONG column is allowed. It is not possible to use LONG column in WHERE or ORDER BY clause.

What are various types of joins?

Equi Joins,

Cartesian Joins

Self Join

Outer Join

What is the maximum number of triggers can apply to a single table?

12 triggers.

What command would you use to create a backup control file?

Alter database backup control file to trace.

What are the different index configurations a table can have?

A table can have one of the following index configurations.

No indexes

A clustered index

A clustered index and many nonclustered indexes

A non clustered index

Many nonclustered indexes.

What is difference between UNIQUE constraint and PRIMARY KEY constraint?

A column defined as UNIQUE can contain Nulls while a column defined as PRIMARY KEY can’t contain Nulls. A table can have only one primary keys.

What is BCP? When does it used?

BulkCopy is a toll used to copy huge amount of data from tables and views. BCP does not copy the structures same as source to destination.

How to know which index a table is using?

SELECT table\_name, index\_name from user\_constraints.

What is SYSTEM tablespace and when is it created?

Every Oracle database contains a tablespace named SYSTEM, which is automatically created when the database is created. The SYSTEM tablespace always contains the data dictionary tables for the entire database.

Which date function returns number value? Which date function returns number value?

months\_between

What is integrity constraints?

Integrity constraints are used to ensure accuracy and consistency of data in a relational database.

What is the difference between Explicit and Implicity Cursors?

An implicit cursor is one created “automatically” for you by Oracle when you execute a query. It is simpler to code An Explicit cursor is one you create yourself. It takes more code, but gives more control – for example, you can just open-fetch-close if you only want the first record and don’t care if there are others DBA\_DATA\_FILES.

What are primary keys?

Primary key are used to uniquely identify each row of the table. A table can have only one primary key.

What are the components of physical database structure of Oracle database?

Oracle database is comprised of three types of files. One or more data files, two are more redo log files, and one or more control files.

What is DECODE function used for?

DECODE is used to decode a CHAR or VARCHAR2 or NUMBER into any of several different character strings or numbers based on value. That is DECODE does a value-by-value substitution.

What is the default return value of a function?

The default return value from a function is int. In other words, unless explicitly specified the default return value by compiler would be integer value from function.

How you will avoid your query from using indexes?

By changing the order of the columns that are used in the index, in the where condition, or by concatenating the columns with some constant values.

What is a cluster key?

The related columns of the tables are called the cluster key. The cluster key is indexed using a cluster index and its value is stored only once for multiple tables in the cluster.

What is SGA?

The System Global Area in an Oracle database is the area in memory to facilitate the transfer or information between users. It holds the most recently requested structural information about the database, the structure is database buffers, dictionary cache, redo log buffer and shared pool area.

What is a data segment?

Data segment are the physical areas within a database block in which the data associated with tables and clusters are stored.

What is ROWID?

ROWID is a pseudo column attached to each row of a table. It is 18 characters long, blockno, row number are the components of ROWID.

PL/SQL is a transaction processing language that offers the following advantages:  
  
Support for SQL - SQL is flexible, powerful and easy to learn..

Support for object-oriented programming.

Better performance - with PL/SQL, an entire block of statements can be sent to Oracle at one time.  
Higher productivity - PL/SQL increases productivity by enabling use of better tools.  
Full portability - PL/SQL are portable to any operating system and platform on which Oracle runs.

Tight integration with Oracle  
 Security PL/SQL can achieve data abstraction, access control.

What is a union, intersect, minus?

UNION  
Union operator is used to return all rows from multiple tables and eliminate duplicate rows. The number of columns and the datatypes of the columns must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.  
  
Syntax

SELECT column1, column2 FROM tablename1  
UNION  
SELECT column1, column2 FROM tablename2;

INTERSECT   
This operator is used to return all rows common to multiple queries. Intersect ignores NULL values  
  
Syntax

SELECT column1, column2 FROM tablename1  
INTERSECT  
SELECT column1, column2 FROM tablename2;

MINUS   
This operator returns all those rows from first query that are not present in the second query.  
  
Syntax

SELECT column1, column2 FROM tablename1  
MINUS  
SELECT column1, column2 FROM tablename2;

How many types of tables supported by Oracle? Explain them.

- Ordinary (heap-organized) table: This is the most basic type of table that stores data as heaps.

- Clustered table: A cluster is a collection of tables sharing the same data block in which the columns are shared as well.  
- Index-organized table: Here, the data is stored in the form of a Btree index structure in which each index entry is storing the non key column value.  
- Partitioned table: Here, the data is broken into partitions that can be managed and operated individually.

What are the types of constraints available in oracle?

Oracle constraints are used to maintain consistent of data and ensure the data is properly maintained. A constraint is more or less a restriction we try to apply on a table.  
  
Types of constraints:  
  
- Check constraints  
- NOT NULL constraint  
- PRIMARY KEY constraint  
- REFERENCES constraint  
- UNIQUE constraint

What is correlated query? Explain its uses.

In a correlated query, the outer query is related to the inner query. This means one or more columns in the outer query are referenced. It’s used when the outer queries value is being used by inner query. For example, we need to find which employee had more perks in the current month than they did in the previous month. The correlated subquery is executed for each row of perks information in the parent query to first determine what the perks were for each employee in the previous month. This data, in turn, is compared to perks for each employee in the current month, and only those employees whose perks in the current month were greater that their previous month's perks are returned.

## WEBLOGIC INTERVIEW QUESTIONS

**Q1. What is Weblogic?**  
WebLogic is a J2EE application server and also an HTTP web server by Oracle , for Unix, Linux, Microsoft Windows, and other platforms. WebLogic supports Oracle, DB2, Microsoft SQL Server, and other JDBC-compliant databases.  
  
**Q2.What is the Basic Components of Weblogic Server?**  
ANSWER : These are the basic weblogic components  
1)Domains  
2)Admin Server  
3) Managed Server  
4) Node Manager  
5) Weblogic Server Cluster  
  
**Q3. What is the Domain in Weblogic server?**  
ANSWER : 1) Domain is a logically related group of Oracle WebLogic Server resources that are managed   
as a single unit   
2) Domain Provides one point of administration   
3) Can logically separate:  
 A) Development, test, and production applications  
 B) Organizational divisions  
  
**Q4. What is the server?**  
ANSWER : A server is an instance of weblogic.Server executing in a Java Virtual Machine (JVM).  
A server:  
1) Runs on a designated Oracle WebLogic Server machine  
2) Has a dedicated amount of RAM  
3) Is multithreaded  
Two types of servers:  
1) Administration Server  
2) Managed Server  
  
**Q5. What is the Admin Server ?**  
ANSWER : Admin Server is the central domain configuration interface which is used to create, delete and   
configure the resources of a domain like managed server, machine, cluster, data source, work managers   
etc.  
  
**Q6. What is the Managed Server?**  
ANSWER : Managed server is a server in a domain that is not the Administration server. It contacts the   
administration server for configuration information. It runs business application in a production   
environment. It is independent of all other Managed servers in a domain (unless they are not in a cluster).   
You can have many managed servers in a domain. Individual managed servers are typically added for   
capacity and application isolation.  
  
**Q7.How Administration server and Managed servers will interact?**  
ANSWER : The Administration Server stores the master copy of the domain configuration, including the   
configuration for all Managed Servers in the domain. Each Managed Server stores a local copy of the   
domain configuration file. When a Managed Server starts, it connects to the Administration Server to   
synchronize the configuration. When the configuration is changed, the Administration Server sends the   
changed configuration to the Managed Servers.  
  
**Q8. What is the MSI mode in Weblogic? How can you enable and disable this option?**  
ANSWER : MSI is nothing but Managed Server Independence. By default, Managed Servers can function   
independently of the Administration Server. A Managed Server instance can start in MSI mode if the   
Administration Server is unavailable. Configure MSI mode from the Administration Console.  
To start a Managed Server in MSI mode, perform the following:  
Ensure that the Managed Server’s root directory contains the config subdirectory. If the config   
subdirectory does not exist, copy it from the Administration Server’s root directory.  
Start the Managed Server at the command line or by using a script.  
**Environment > Servers > Server\_Name > Tuning > Advanced > Managed Server Independence Enabled**   
check box  
  
**Q9. Difference between Weblogic Development and Production Mode?**  
ANSWER : Here are some difference between Weblogic Development Mode and Production Mode:  
Development Mode:  
1) The default JDK for development domain is Sun Hotspot  
2) You can use the demo certificates for SSL  
3) Auto deployment is enabled  
4) Server instances rotate their log files on startup  
5) Admin Server uses an automatically created boot.properties during startup  
6) The default maximum capacity for JDBC Datasource is 15  
  
Production Mode:  
1) The default JDK for production domain is JRockit  
2) If you use the demo certificates for SSL a warning is displayed  
3) Auto deployment is disabled  
4) Server instances rotate their log files when it reaches 5MB  
5) Admin Server prompts for username and password during startup  
6) The default maximum capacity for JDBC Datasource is 25  
  
**Q10. How to change from development mode to production mode in Weblogic 10.3?**  
ANSWER : To change Weblogic start up mode from DEV to production. One way to change it is, by simply   
editing setDomainEnv.cmd which resides in **$root\_domain/bin** folder.  
1. Look for the line that sets the PRODUCTION\_MODE script variable: set PRODUCTION\_MODE  
Add false to the value of the PRODUCTION\_MODE variable to ensure the server starts in development   
Set true for starting in prod mode.  
set PRODUCTION\_MODE=false  
2. Save your changes and exit the text editor.  
  
**Q11. What are the supported installation modes for WebLogic Server?**  
ANSWER : Graphical mode, console mode and silent mode.  
  
**Q12.How to access admin console?**  
ANSWER **: Ssl enabled admin console accessed as** :  
https://hostname or ip address (where admin server running):(port on which admin server is ssl   
enabled admin console accessed as :  
Example - **https://localhost:7002/console**  
mode:  
  
  
**non ssl admin console accessed as** :  
http://hostname or ip address (where admin server running):(port on which admin server is ssl enabled   
admin console accessed as :  
Example - **http://localhost:7001/console**  
  
  
**Q13. What is the default port of Weblogic admin server?**  
ANSWER : By default weblogic admin server configured on port 7001 and SSL is on 7002.  
  
**Q14.How many admin consoles possible in a single domain?**  
ANSWER : Only One   
  
**Q15.What is boot.proerties file in weblogic server?**  
ANSWER : boot.properties is the file used by admin or managed server during startup for username and password. it exist under your domain/servers/server\_name/security folder.  
  
**Q16.What is config.xml?**  
ANSWER: config.xml is the central configuration repository for a domain. every resource you have configured from admin console or by command line or by any other tool registered under this file.WLS 9.x onwards, this file contain references to other xml configuration files also those are under your domain/config folder ( further sub folder under it for respective resources ).  
  
**Q17 How do I provide User credentials for starting weblogic server?**  
ANSWER : When you create a domain, the Configuration Wizard prompts you to provide the username and password for an initial administrative user. If you create the domain in development mode, the wizard saves the username and encrypted password in a boot identity file. A WebLogic Server instance can refer to a boot identity file during its startup process. If a server instance does not find such a file, it prompts you to enter credentials.  
If you create a domain in production mode, or if you want to change user credentials in an existing boot identity file, you can create a new boot identity file.  
  
**Q18.Can I start a Managed Server if the Administration server is unavailable?**  
ANSWER: By default, if a Managed Server is unable to connect to the specified Administration Server during startup, it can retrieve its configuration by reading a configuration file and other files directly. You cannot change the server’s configuration until the Administration Server is available. A Managed Server that starts in this way is running in Managed Server Independence mode.  
  
**Q19. What Is a WebLogic Server Cluster?**  
ANSWER: A WebLogic Server cluster consists of multiple WebLogic Server server instances running simultaneously and working together to provide increased scalability and reliability. A cluster appears to clients to be a single WebLogic Server instance. The server instances that constitute a cluster can run on the same machine, or be located on different machines. You can increase a cluster’s capacity by adding additional server instances to the cluster on an existing machine, or you can add machines to the cluster   
to host the incremental server instances. Each server instance in a cluster must run the same version of WebLogic Server.  
  
**Q20.What is the advantage of clustering?**  
ANSWER: WebLogic clustering offers three important benefits:  
1. **Scalability**:  
 The capacity of an application deployed on a WebLogic Server cluster can be increased dynamically to meet demand. You can add server instances to a cluster without interruption of service — the application continues to run without impact to clients and end users.  
2. **Load balancing**:  
 The ability to distribute requests across all members of the cluster, according to the workload on each server.  
3. **High availability**:  
 A mix of features that ensure applications and services are available even if aserver or machine fails. Clients can continue to work with little or no disruption in a highly available environment. WebLogic achieves high availability using acombination of features: replication, failover, and migratable services.  
  
  
**Q21.What is Mulicast and Unicast in weblogic?**  
ANSWER:

**Mulicast**:  
 Multicast is easier to explain over Unicast. Multicast is a broadcast UDP option for sending a packet/announcement over to a group that is listening on a specific multicast address and port over which the announcement is sent.There is a defined range for valid Multicast address (224.0.0.1 to 239.255.255.255). Everyone listening on the given address hears the announcement just like following a Twitter post. Some limitations with Multicast is the TTL (time to live) across machines/subnets/routers needs to be adjusted and the routers configured to retransmit the multicast packet across subnets.

**Unicast**:  
 Unicast is more of a point to point UDP option to send the packet to a specific member and not everyone. That way, unicast is more of a private conversation between two individuals while multicast is more of a shout to a group or room. Both are UDP based, so there can be losses unlike TCP that handles retransmissions on message loss. But Unicast can span across routers and does not have to worry about TTL without the everyone hearing the announcement. So, Network Admins in general prefer to go with Unicast over Multicast for these reasons  
  
**Q23.What is Horizontal and vertical clustering?**  
ANSWER : There are two method of clustering Horizontal and Vertical.  
**Horizontal clustering** :involves running multiple Java application servers that are run on two or more   
separate physical machines.  
Horizontal scaling is more reliable than vertical scaling, since there are multiple machines involved in the   
cluster environment, as compared to only one machine  
**Vertical clustering:** however, consists of multiple Java application servers on a single physical machine.  
With vertical scaling, the machine's processing power, CPU usage, and JVM heap memory configurations   
are the main factors in deciding how many server instances should be run on one machine.  
  
**Q24. How Does a Cluster Relate to a Domain?**  
ANSWER : A cluster is part of a particular WebLogic Server domain.A domain is an interrelated set of WebLogic Server resources that are managed as a unit. A domain includes one or more WebLogic Server instances, which can be clustered, non-clustered, or a combination of clustered and non-clustered   
instances. A domain can include multiple clusters. A domain also contains the application components   
deployed in the domain, and the resources and services required by those application components and   
the server instances in the domain.  
  
  
**Q25.What is Node Manager?**  
Answer Node Manager is program that is used to control WebLogic Server instances. A single Node   
Manager instance is used to control all of the server instances running on the same physical machine or different machine.    
These instances can reside in different clusters, domains, and such. You must configure each machine in   
your domain to communicate with Node Manager.  
  
**Q.26 How many way to start and stop weblgic servers?**  
ANSWER : There are many ways to start & stop weblogic Admin and managed servers:  
You can start and stop Admin Server using :  
1) Script  
2) "Java weblogic.server "command(The weblogic.Server class is the main class for a WebLogic Server   
instance)  
3) From startup option on windows  
4) WLST with or without node manager  
You can start managed servers using:  
1) scripts  
2) admin console  
3) using WLST  
4) using node manager  
5) java weblogic.server command( The weblogic.Server class is the main class for a WebLogic   
Server instance )  
  
**Q27.How many types you configure managed server in weblogic server?**  
A.Three types  
1)Domain Configuration Wizard  
2)Administration Console  
3)Weblogic Scripting Tool  
  
**Q28. How to know weblogic version?**  
ANSWER : In the WebLogic console, in the left navigation tree, click on "Environment", then "Servers".   
Then click on one of the servers in the list. Then click on the "Monitoring" tab. Look at the "WebLogic   
Version" field.  
  
**Q29How do you set the classpath?**  
ANSWER : WebLogic Server installs the following script that you can use to set the  
classpath that a server requires:  
WL\_HOME\server\bin\setWLSEnv.cmd (on Windows)  
WL\_HOME/server/bin/setWLSEnv.sh (on UNIX)  
  
**Q30.. How does a server know when another server is unavailable?**

WebLogic Server uses two mechanisms to determine if a given server instance is unavailable. Each WebLogic Server instance in a cluster uses multicast to broadcast regular “heartbeat” messages that advertise its availability. By monitoring heartbeat messages, server instances in a cluster determine when a server instance has failed.

The other server instances will drop a server instance from the cluster, if they do not receive three consecutive heartbeats from that server instance. WebLogic Server also monitors socket errors to determine the availability of a server instance. For example, if server instance A has an open socket to server instance B, and the socket unexpectedly closes, server A assumes that server B is offline.  
  
  
**Q31. How do you differentiate between a server hang and server crash issue?**  
Answer : When a Server crashes, the JAVA process no longer exists. When the Server is hung, it stops responding.

We can use the WebLogic .ADMIN utility to ping the server. In case of a hang situation we can take multiple thread dumps and analyze the cause of hang  
  
  
**Q32. How are notifications made when a server is added to a cluster?**  
Answer : The WebLogic Server cluster broadcasts the availability of a new server instance each time a new instance joins the cluster. Cluster-aware stubs also periodically update their list of available server instances.  
  
**Q33 How do I turn the auto-deployment feature off?**  
 The auto-deployment feature checks the applications folder every three seconds to determine whether there are any new applications or any changes to existing applications and then dynamically deploys these changes.  
 The auto-deployment feature is enabled for servers that run in development mode.To disable auto-deployment feature, use one of the following methods to place servers in production mode:  
=>In the Administration Console, click the name of the domain in the left pane, then select the Production Mode checkbox in the right pane.  
=>At the command line, include the following argument when starting the domain’s Administration Server:  
 -Dweblogic.ProductionModeEnabled=true  
Production mode is set for all WebLogic Server instances in a given domain.  
  
**Q34. How managed servers communicate with each other ?**  
ANSWER : Managed servers communicate each other using t3 protocol internally.  
  
**Q35. What is SSL in weblogic?**  
ANSWER : Secure Sockets Layer (SSL) provides secure connections by allowing two applications connecting over a network connection to authenticate the other's identity and by encrypting the data exchanged between the applications. Authentication allows a server and optionally a client to verify the identity of the application on the other end of a network connection. Encryption makes data transmitted over the network intelligible only to the intended recipient  
  
Q36. How you deploy your applications on weblogic server?  
ANSWER : These are the types.  
1) Auto Deployment  
2) Console  
3) Command line – weblogic.deployer  
4) ANT / WLST  
  
Q37. What are stating modes are available in Weblogic Server?

There are three starting modes are available:

1) Stage mode: Administration Server copies the deployment files from their original location on the Administration Server machine to the staging directories of each target server

2) External Stage: target servers deploy using local copies of the deployment files, here the Administration Server does not automatically copy the deployment files to targeted servers in external\_stage mode ; instead, you must copy the files to the staging directory of each target server before deployment.

3) No Stage: The Administration Server does not copy deployment unit files; instead, all servers deploy using the same physical copy of the deployment files, which must be directly accessible by the Administration Server and target servers.

Q38. What is a Thread dump? How will you take in unix/linux and windows?  
A Java thread dump is a way of finding out what every thread in the JVM is doing at a particular point in time. This is especially useful if your Java application sometimes seems to hang when running under load, as an analysis of the dump will show where the threads are stuck.  
1) Linux: kill -3 <ps\_id>  
2) Windows (console mode): crtl+break  
3) Windows (service) : beasvc -dump -svcname: mydomain\_myserver