1. WS has provided as the following component,
2. WSDL
3. Skeleton
4. Stub
5. SOAP
6. HTTP
7. UDDI🡺 Universal Description Discovery and Integration
8. WS is for applications interaction
9. A Simple flow Explanation🡺 see the notes🡺 provider has Skeleton and Consumer has Stub where Stub and Skeleton(IS A SERVLET CLASS) has Same Method Declaration as SEI but Logic is to covert Java req and Res to SOAP req and res
10. We have 4 implementers of WS namely,(again this 4 are specifications only we need implementation)
11. JAX-RPC
12. JAX-M🡺 is for async
13. JAX-WS
14. JAX-RS
15. Eclipse will support only 3 implementations it seems for remaining we need some plugins,
16. Axis1
17. Axis2
18. Apache CXF
19. AXIS-1 example🡺 is the implementer of JAX\_RPC
20. Service endpoint Url syntax🡺http://localhost:8080/webappName/skeletonUrl/unique Name of the Service
21. AXIS-1 uses server-config.wsdd(will be present in web-inf folder and this file also contains class ,method and parameters details of a service) file in order to map unique name to the Service

Syntax🡺

<ns:service name=”Unique name of Service”-------- >

<ns1:parameter name=”**allowedMethods**” value=”add sub multiply”/>

1. WSDL GENERATION TOOL is 🡺org.apache.axis.wsdl.java2wsdl 🡺 this is a class that is acting as a WSDL generator
2. java2wsdl WSDL generator needs following information to generate WSDL
3. Service Class
4. endpoint url
5. wsdl file name
6. targetnamespace
7. web.xml file will have skeleton information
8. Steps to create WS provider and Consumer in STS see the notes
9. Final url to be hit in the Browser🡺 <http://localhost:8080/webappName/skeletonUrl/unique?wsdl>
10. In WS we have 2 types of Clients(a client app can be Standalone or web app)
11. Proxy based Client🡺 see the notes
12. DII client🡺 Dynamic Invocation Interface🡺 here we will not generate stub , ws implementors provide few set of classes that acts as a stub to call ws methods
13. Ms to invoke Service class methods we need its reference type, for getting the ref type we will use Service class of Java and URL class of Java🡺URL will have a wsdl location and Service class will have some sort of reference to SEI.
14. Axis2 Example🡺
15. Here Skeleton used is 🡺org.apache.**axis2**.transport.http.AxisServlet
16. Stub and WSDL generation see the notes

SOAP

1. SOAP🡺 Simple Object access protocol
2. IS a Messaging protocol
3. IN WS errors/exceptions are called as Faults
4. Soap contains 3 parts
5. Soap-envelope
6. Soap-header
7. Soap-Body
8. If req is failed then Soap Message will contain 4th Section called🡺 Soap-fault
9. Syntax🡺

<?xml version=”1.0”?>

<soap: Envelope xmlns: soap=”http:------------“

Soap:encodingstyle=”http:-------“>

<soap:header>

------

</soap:header>

<soap:body>

-------

<soap:fault>

------

</soap:fault>

</soap:body>

</soap: Envelope>

1. Here Servlet class used is (I don’t know the Exactly which implementation is this)

<servlet-class>Com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>

Note🡺 **for this servlet some Listener class is also used**

Rest Service

1. Rest🡺Representational state transfer
2. REST is an architecture
3. REST service don’t have such WSDL specification file thus URL or URI itself is a specification for rest services to access the web service
4. In case of rest client will directly use the uri to get response
5. SOAP is a Protocol and REST is a architecture
6. Soap permits XML data format only but REST allows different formats like HTML, XML, JSON and etc...
7. SOAP supports different types of protocols like http protocol, RMI protocols. But REST supports only Http and https
8. REST do not need client generation, wsdl generation and etc..
9. Jersey is the implementation of JAX-RS
10. Apache CXF also provides implementation for JAX-RS
11. HTTP Methods,
12. Get🡺 to get information from Server
13. Post🡺 to send or save information in the server
14. Put🡺 is use to update data of that URL
15. Delete🡺 to delete data of that URL
16. Metadata🡺 along with req and res we also have header information which is Meta data
17. HTTP Status Code,
18. 2xx🡺 success
19. 1xx🡺informational
20. 3xx🡺redirection
21. 4xx🡺Client error
22. 5xx🡺Server error
23. We can pass the input parameters to operation In 2 ways for a rest service
24. Request params
25. Path parameter
26. Syntax for Path parameter🡺

Baseurl/100🡺 where 100 is employee id this is sent as method input

1. Syntax for Request parameter or Query Parameter,

baseURL**?id**=1

Note🡺 Query parameter is not like a path parameter its additional parameter that gets added to http header

1. As discussed early along with req and res we have header information🡺 note we have req header and response header
2. List of annotations used
3. @Path🡺 both at class level and method level
4. @Produces🡺is used to, what we want to return from WS 🡺 ms is method level @
5. @Consumes🡺 is used to, what we want to send to WS🡺 ms is method level @
6. @PathParam
7. @Get
8. @Post
9. @Put
10. @Delete
11. JSON will store the value in the form of key and value pair
12. JSONLint is used to validate JSON.
13. @Produces(MediaType.APPLICATION\_JSON)
14. Example
15. web.xml🡺 root tag for web.xml is <web-app>

<servlet>

------

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</ servlet-class>

<initiparam>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>com.capgemini.services</param-value>

</initiparam>

<load-on-startup>1</ load-on-startup>

</servlet>

<init-param>🡺 in our app we might have 10 packages among this 10 packages only one package will have all service classes thus we need to tell JAX\_RS from where or which package it should start searching for WS classes

1. Java file

@Path(“/Capgemini”)

Public class NotifyNondlineRestService

{

@GET

@Path(“{a}/{b}”)

Public int add (@PathParam(“a”) int a, @PathParam(“b”) int b)

{

Return a+b;

}

}

1. Output🡺

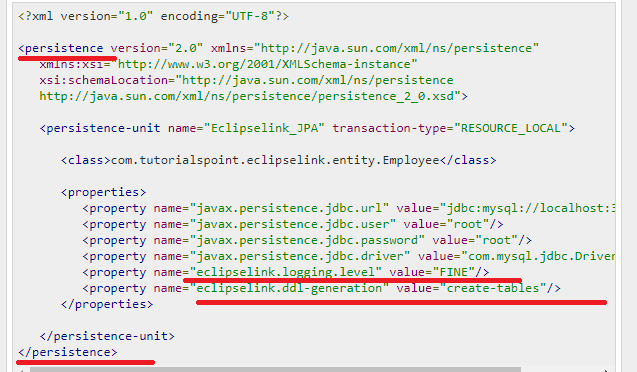
http:..loclahost:8080/applicationContext/url in web.xml file/url as in @path annotation**/10/20**

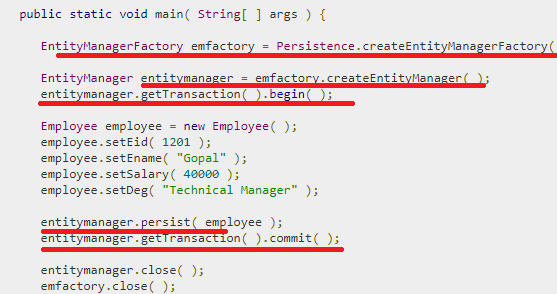
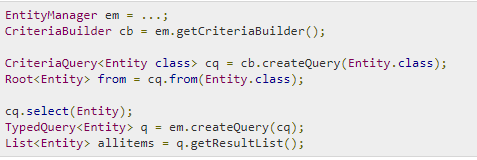
JPA

1. JPA is an api(classes and interface) to persistently store the vast amount of data into DB
2. JPA is by Oracle
3. While Introducing EJB 3.0 the persistence layer was separated and specified as JPA 1.0
4. JPA 2.1 in 2013
5. Has EMF, EM ,E, ETransaction, Query, Persistence
6. Persistence🡺 this class contains static method to obtain EMF
7. EM and ET has one to one relationship
8. ORM takes care of Converting data from Object type to relational type and Vice versa
9. Mapping.xml🡺



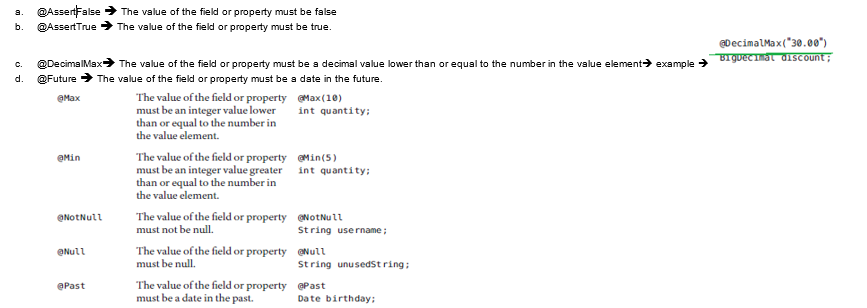
1. List of Annotations
2. @SequenceGenerator
3. @TableGenerator
4. @UniqueConstraint
5. Bean Conversion 🡺should gave getters for non-Boolean Property and must have isEmpty() for boolean properties
6. Persistence.xml



1. EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink\_JPA" );
2. CRUD methods 🡺 ms
3. Persist
4. Find
5. Remove
6. Update
7. 
8. CQ is Type-safe
9. 

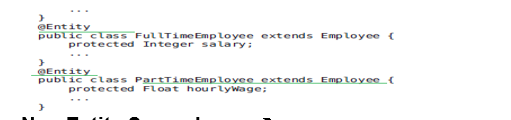
JPA Documentation

1. **Needed an Example on Arrays storage**
2. **Map Was not covered in Collections Section**
3. Bean Validation

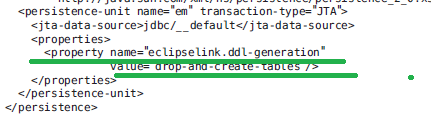


1. @Pattern and @Size🡺
2. **Example on primary key as Date field**
3. Caccade\_type.All/Detach/Merge/persist/refresh/remove
4. Abstract entities🡺 an abstract class annotated with @Entity. Abstract entities are like concrete entities but cannot be instantiated.
5. Mapped superclass🡺 Entities may inherit from super classes that contain persistent state and mapping information but are not entities. That is, the superclass is not decorated with the @Entity annotation and is not mapped as an entity by the Java Persistence provider.

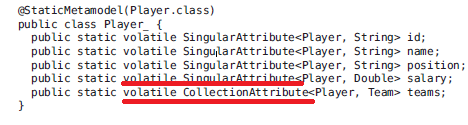
Example🡺 Parent 

Child🡺

1. @SecondaryTable🡺
2. MetaModel🡺 MS🡺CriteriaQuery uses metamodel to create query
3. **Metamodel classes model an entity’s attributes and are used by Criteria queries to navigate to an entity’s attributes**
4. Automatic table generation inJPA



1. An Equivalent Meta model class for every entity gets generated at ,**compile time🡺 in maven I think during maven install on JPA project**
2. Syntax🡺



1. The **EntityManager.createQuery and EntityManager.createNamedQuery** methods are used to query the data store by using Java Persistence query language queries
2. Syntax🡺 same as in hibernate, in case of createQuery(“Sql Query is sent as input)

And in case of createNamedQuery(Name of the Query is sent as input);

1. Same as in hibernate JPA aslo has Positional parameter prefixed with “ ?Position number” and Named parameter which is prefixed with “**:**parameter\_name”
2. Needed more information on
3. MEMBER OF
4. EXISTS
5. NOT EXISTS🡺 Scenarios
6. Set of Functions🡺 see later
7. **NEEDED MORE INFORMATION ON CASE EXAMPLE(AND ALSO IN PLSQL)**
8. **Constructor expressions allow you to return Java instances that store a query result element instead of an Object [].**

**Example🡺**

****

NOTE 🡺 ms 🡺**CriteriaQuery has methods like, select, from, where, having, groupBy and orderBy**

1. **Metamodel classes are typically generated by annotation processors either at development time or at runtime**
2. One of the programmatic way of getting metaModel class is🡺

object\_OF\_Root.getModel:

we can also get from em as shown below

**EntityType<Employee>Employee\_=em.getMetamodel().entity(Employee.class)**

1. For more information On CQ and CB methods First see the Notes
2. Ms🡺CriteriaBuilder methods is returning Predicate
3. **NEEDED MORE INFORMATION ON PREDICATE AND EXPRESSION IN JPA**
4. Methods of CriteriaBuilder,
   1. and🡺 return type is Predicate
   2. asc🡺Order
   3. des🡺Order
   4. avg🡺Expression
   5. sum🡺
   6. Count🡺Expression
   7. concat🡺 Expression
   8. between🡺Predicate
   9. equal🡺Predicate
   10. equals🡺Boolean object
   11. exists🡺P
   12. ge()🡺P
   13. greaterThan🡺P
   14. greaterThanOrEqualTo🡺p
   15. greatest🡺P
   16. gt()🡺P, lt
   17. in
   18. isEmpty()🡺P
   19. isnotEmpty
   20. isFalse🡺P
   21. isMember🡺P
   22. isNotMember
   23. like
   24. min
   25. max
   26. not
   27. notEqual
   28. notLike
   29. length
5. Methods of the Root<T> is🡺 ms actually root refers to “from” part of the SQL query
   1. get(field\_name)
   2. fetch(field\_name)
   3. in()
   4. isNull
   5. isNotNull
   6. join()
   7. joinCollection
   8. joinList
   9. joinMap
   10. joinSet

Enum

1. Example🡺

Public enum Week implements Serializable{

SUNDAY(“sun”, “First Day of the Week),

MONDAY(“mon”, “second Day of the Week),

TUESDAY(“tues”, “third Day of the Week),

WEDNESDAY(“wed”, “fourth Day of the Week),

THURSDAY(“thurs”, “fifth Day of the Week),

FRIDAY(“fri”, “sixth Day of the Week),

SATURDAY(“sat”, “seventh Day of the Week),

Private String code;

Private String description;

Private Week(String cd, String des){

this. code= cd;

this. description= des;

}

Public getcode(){

return code;

}

Public getdescription(){

return description;

}

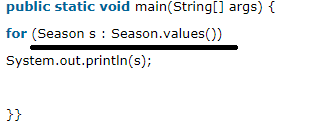
}

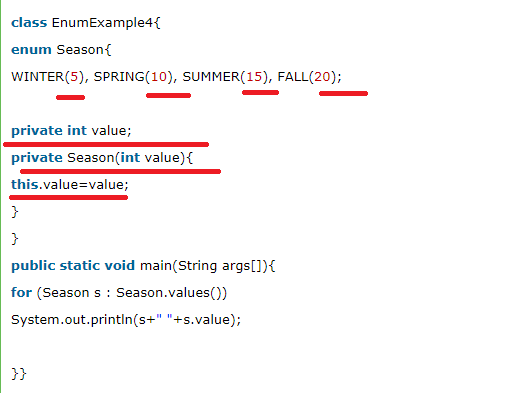
Out the class calling an enum syntax is🡺

String codeFromEnum=Week.SUNDAY.getCode();

String descriptionFromEnum=Week.SUNDAY. getdescription();

1. Enum is used to hold set of predefined values or constant values
2. Implicitly enum constants are static final
3. By default enum will have private constructor
4. Enum can be inside or outside class also.
5. enum improves type safety
6. enum can be easily used in switch
7. enum can have fields, constructors and methods🡺 example see above
8. **enum may implement many interfaces but cannot extend any class because it internally extends Enum class**
9. The java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

Example🡺

1. **we can initialize the specific value to the enum constants by defining fields and constructors🡺 as shown in above example**
2. 

Ms🡺See as shown in the above code🡺 will creating the enum only we are giving its value, and we are not passing any values from outside

1. **Can we create the instance of enum by new keyword? 🡺 No, because it contains private constructors only.**

|  |
| --- |
|  |

Java-Printout

1. **Object-oriented🡺 means we organize our software as a combination of different types of object that has both data and behavior**
2. **OOPS🡺 is a methodology that simplify software development and maintenance by providing some rules**
3. JDK🡺JRE+ debugger+ COMPILER
4. Jre🡺jvm + set of libraries
5. JVM🡺 ms where my app enters and gets executed
6. Data Types🡺 
7. Labelled for loop🡺 needed example on this
8. **Polymorphism🡺 one task is performed in different ways**
9. Encapsulation🡺 Binding or wrapping the code and data together is called encapsulation

JAVA BEAN is fully encapsulated

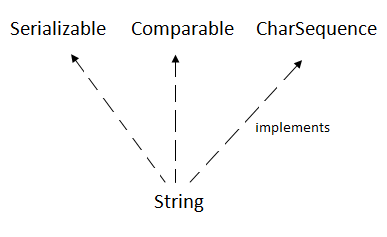
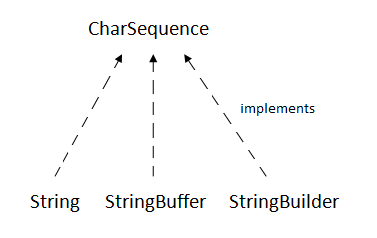
1. Needed example on newInstance() method for object creating in java
2. why the constructor has name Constructor🡺 it constructs the values i.e. provides data for the object
3. Aggregation purpose is reusability
4. **I had observed that Javac places IIB inside each constructor of Child but not in the parent constructor.**
5. Use instanceOf operator to avoid ClassCastException at run time(if both the classes does not have **“is-a”** relationship means we will get class cast exception at run time).
6. Static import🡺advantage🡺 helps in less coding
7. See Qspider notes for more information on System.out.println()
8. Primitive type into its equivalent Wrapper types is called boxing
9. Methods of object class
10. hashCode()
11. equals()
12. clone()🡺throws CloneNotSupportedException
13. toString
14. notify
15. notifyAll
16. wait🡺InterruptedException
17. finalize
18. datatype[] arrayname= new datatype[size]; 🡺or 🡺int[] arrayname ={1,2,3}
19. array has length property
20. In java, array is an object. For array object, an proxy class is created whose name can be obtained by getClass().getName() method on the object.
21. We can copy an array to another by the arraycopy method of System class. 🡺

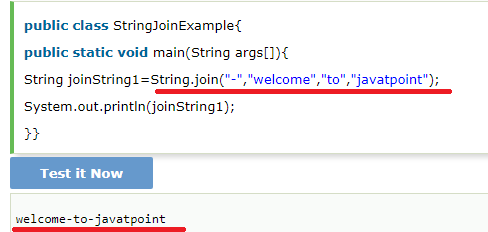
Example

**public** **static** **void** arraycopy(

Object src, **int** srcPosfromwheretocopy,Object dest, **int** destPoswheretoPaste, **int** length

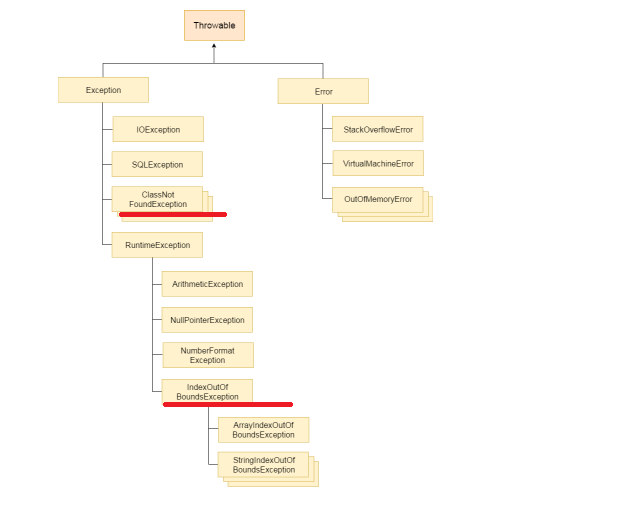
)

1. Wrapper🡺
2. valueOf()🡺 to convert primitive to object type and its static method
3. xxxValue🡺 to convert object type to primitive and its instance method🡺 example intValue()
4.  
5. Methods of String class
6. [char charAt(int index)](https://www.javatpoint.com/java-string-charat)
7. [int length()](https://www.javatpoint.com/java-string-length)
8. [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format)
9. [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring), [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring)
10. [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains)
11. [static String join(CharSequence delimiter, CharSequence... elements](https://www.javatpoint.com/java-string-join))



1. [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals)
2. [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase)
3. [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty)
4. [String concat(String str)](https://www.javatpoint.com/java-string-concat)
5. [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace), [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace)
6. [String[] split(String regex)](https://www.javatpoint.com/java-string-split)
7. [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof), [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof), [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof), [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof)
8. [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase),[String toUpperCase()](https://www.javatpoint.com/java-string-touppercase)
9. [String trim()](https://www.javatpoint.com/java-string-trim)
10. [static String valueOf(int value](https://www.javatpoint.com/java-string-valueof))
11. Needed few information on StringTokenizer
12. Methods of StringBuffer
13. append(String s)🡺Synchronized
14. insert(int offset, String s)🡺 Synchronized
15. replace(int startIndex, int endIndex, String str)🡺 Synchronized
16. delete(int startIndex, int endIndex)🡺 Synchronized
17. reverse()🡺Synchronized
18. and few String methods🡺

* capacity(),
* ensureCapacity(int minCapacity)
* charAt(int index)
* length()
* substring

1. 

Needed more information on ClassNotFoundException and ClassDefException

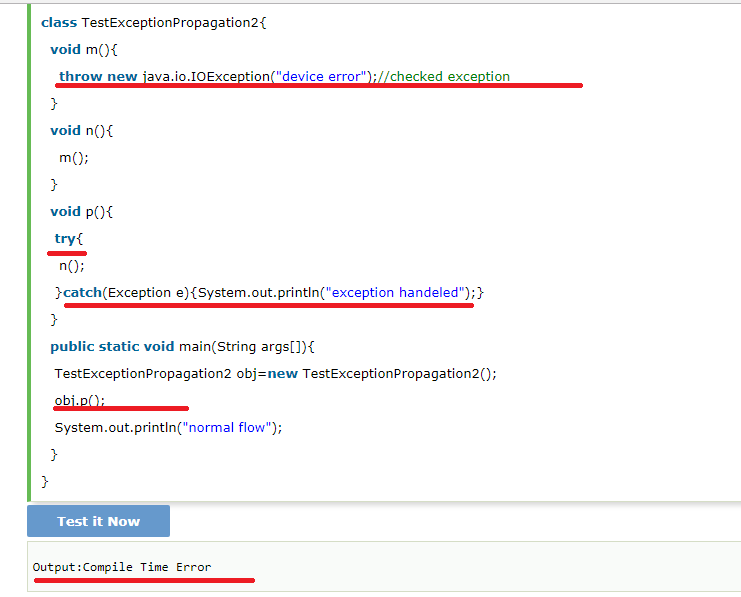
1. CheckedException, unCheckedException and errors
2. Checked Exception🡺 should be handled using try-catch block or throws keyword
3. **Suppose if method does not throw Checked Exception then I cannot simply Throw it either by using Throws or throw🡺 ms 🡺 check this again**
4. I cannot throw Checked exception using throw keyword🡺ms
5. No statements between try-catch-finally🡺 it will become unreachable
6. Throw keyword is mainly used to throw custom Exceptions
7. The Java throw keyword is used to explicitly throw an exception.

# **Java Exception propagation**

#### Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).

#### **Rule: By default, Checked Exceptions are not forwarded in calling chain (propagated).**

Example for above rule is,



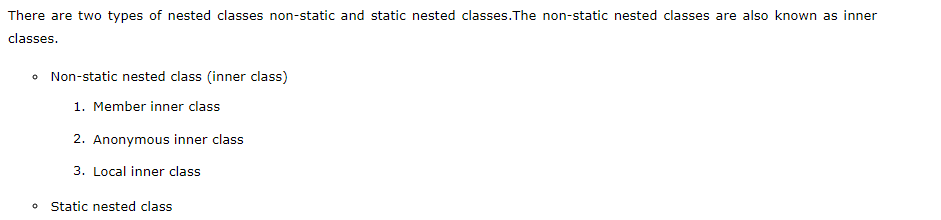
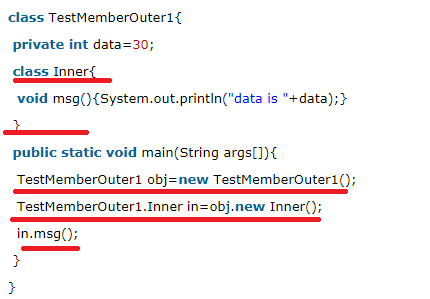
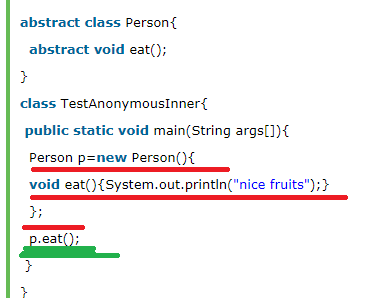
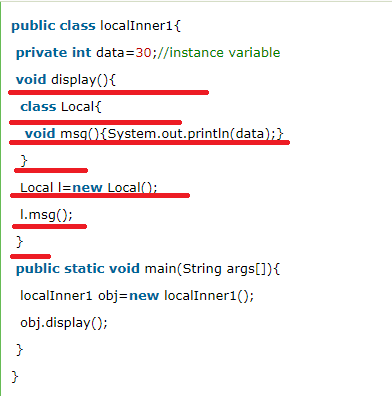
### Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.

1. Lets see the above rule🡺

There are two cases:

1. **Case1:**You caught the exception i.e. handle the exception using try/catch.
2. **Case2:**You declare the exception i.e. specifying throws with the method.

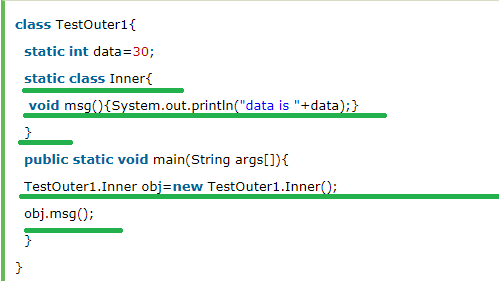
# **Java Inner Classes**

1. 
2. A non-static class that is created inside a class but outside a method is called member inner class. 🡺
3. A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways: 🡺Interface, abstract Class
4. 
5. A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.
6. 
7. Rules for LIC🡺

#### Rule: Local variable can't be private, public or protected.

#### 1) Local inner class cannot be invoked from outside the method.

#### 2) Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in local inner class.

1. Static Nested classes
   1. A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.
   2. 

Explanation🡺

* see static inner class has non static method
* Syntax to call static inner class static method🡺  TestOuter2.Inner.msg();

Java\_PrintOut1

1. Thread is a light weight **sub process**, a smallest unit of processing
2. On calling start() Thread moves from new to runnable state
3. Methods of thread class
   * 1. **run()**
     2. **start**
     3. **sleep(long miliseconds)🡺static**
     4. **join(),join(long miliseconds)🡺** The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

****

**Incase of join(milliseconds)🡺for the specified milliseconds thread will not release the resource🡺 example🡺** when t1 is completes its task for 1500 miliseconds(3 times) then t2 and t3 starts executing.

* + 1. **yield()**
    2. **getPriority(),setPriority()**
    3. **getName(),setName(String name)**
    4. **Thread currentThread()🡺static**
    5. **getId()**
    6. **public Thread.State getState() 🡺 returns the state of the thread**
    7. **isAlive():**
    8. **isDaemon,** **setDaemon**
    9. **interrupt, isInterrupted,** **interrupted**
    10. **suspend, resume** ,**stop🡺 all are deprecated**

1. **Thread scheduler** in java is the part of the JVM that decides which thread should run.
2. There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
3. No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown
4. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on
5. By default each thread has a name i.e. thread-o, thread-1 and so on
6. Make a user thread daemon thread before its started otherwise we will get IllegalThreadStateException
7. Thread Group, Thread Pool, Java Shutdown hook
8. GC is a process of reclaiming runtime unused memory or is a process of destroying the unused obj

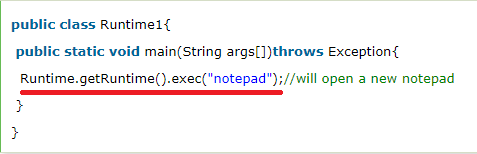
#### Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

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

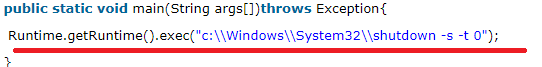
#### Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

#### Note: Neither finalization nor garbage collection is guaranteed.

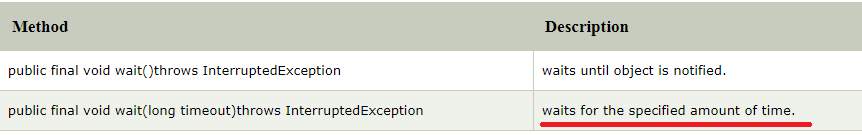
1. **Java Runtime** class is used to interact with java runtime environment
2. **Methods of Runtime,**
   1. static Runtime getRuntime()
   2. exit(int status)
   3. availableProcessors()
   4. freeMemory()
   5. totalMemory()
   6. public Process exec(String command)throws IOException 🡺 executes given command in a separate process.



## **How to shutdown system in Java using exec()**



## **restart 🡺** Runtime.getRuntime().exec("shutdown -r -t 0");

1. 
2. If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException.

JDBC

1. 4 types of driver🡺

* JDBC-ODBC bridge driver
* Native-API driver (partially java driver)
* Network Protocol driver (fully java driver)
* Thin driver (fully java driver)

1. 5 Steps in connecting java app to db
   1. Register the driver
   2. Create connection
   3. Create statement
   4. Execute query
   5. Close connection🡺by closing the connection obj Statement and ResultSet will be closed automatically

Java\_PrintOut2

1. The forName() method of Class class is used to register the driver class🡺 Class.forName("oracle.jdbc.driver.OracleDriver");  🡺
2. Methods of DriverManager class

* registerDriver(Driver driver):
* deregisterDriver(Driver driver):
* getConnection(String url):🡺  static
* static Connection getConnection(String url,String userName,String password):

1. Connection interface Methods,
   * 1. **createStatement**
     2. **setAutoCommit()**
     3. **commit**
     4. **rollback**

By default, connection commits the changes after executing queries

1. Statement methods,
2. **ResultSet executeQuery(String sql)**
3. **int executeUpdate(String sql)**
4. **boolean execute(String sql):**
5. **int[] executeBatch()**
6. Initially ResultSet maintains cursor to first row, methods of ResultSet

#### By default, ResultSet object can be moved forward only and it is not updatable.

1. **boolean next():**
2. **boolean previous():**
3. **boolean first()**
4. **boolean last()**
5. **int getInt(int columnIndex)**
6. **int getInt(String columnName):**
7. **String getString(int columnIndex):**
8. **public String getString(String columnName):**
9. Methods of PreparedStatement
10. setInt(int paramIndex, int value)
11. setString(int paramIndex, String value)
12. setFloat(int paramIndex, float value)
13. setDouble(int paramIndex, double value)
14. int executeUpdate()
15. ResultSet executeQuery()
16. The metadata means data about data i
17. ResultSetMetaData methods,
18. int getColumnCount()throws SQLException
19. String getColumnName(int index)throws SQLException
20. String getColumnTypeName(int index)throws SQLException
21. public String getTableName(int index)throws SQLException
22. DatabaseMetaData interface methods,
23. **String getDriverName()throws SQLException**
24. **String getDriverVersion()throws SQLException:**
25. **String getUserName()throws SQLException:**
26. **String getDatabaseProductName()throws SQLException:**
27. **String getDatabaseProductVersion()throws SQLException:**
28. **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException**

Java\_PrintOur3

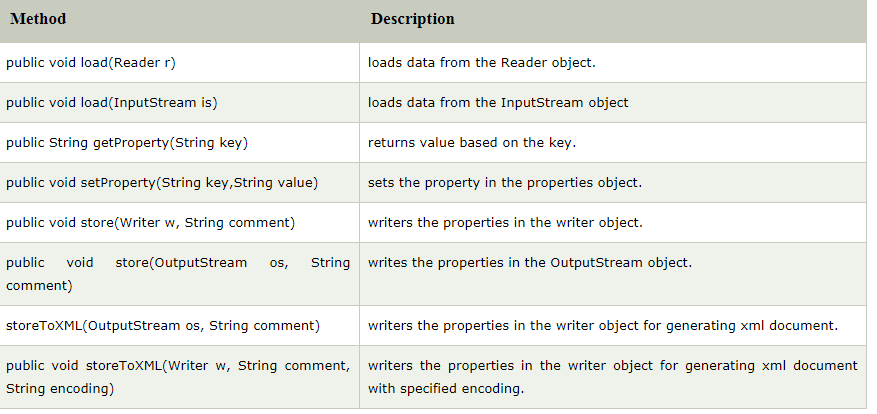
Collection

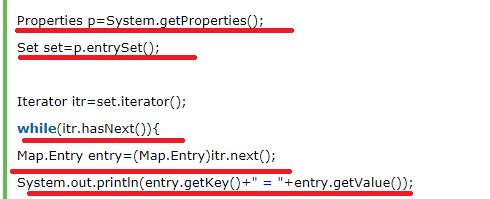
1. hierarchy of collection framework
2. Methods of Collection interface
3. boolean add(Object element)
4. boolean addAll(Collection c)
5. boolean remove(Object element)
6. boolean removeAll(Collection c)
7. boolean retainAll(Collection c)
8. boolean contains(Object element)
9. boolean containsAll(Collection c)
10. int size()
11. clear()
12. Iterator iterator()
13. Object[] toArray()🡺 Needed to do hands on this
14. boolean isEmpty()
15. boolean equals(Object element)
16. int hashCode()
17. Methods of Iterator interface,
18. boolean hasNext()
19. Object next()🡺 in case of ResultSet it’s of boolean type
20. void remove()
21. Methods of Iterator interface
22. Boolean hasPreviuos
23. Object previous
24. above methods
25. Methods of ArrayList
26. **void** add(int index, Object element)🡺 in case of Collection return type was boolean(no no see below method)
27. boolean add(Object o)
28. int indexOf(Object o)
29. int lastIndexOf(Object o)
30. boolean addAll(Collection c)
31. boolean addAll(int index, Collection c)
32. clear()
33. Object[] toArray()
34. **void trimToSize()🡺It is used to trim the capacity of this ArrayList instance to be the list's current size.**
35. Methods of List Interface
36. Add(posi, element)
37. addAll(posi, Collection obj)
38. get(position)
39. set(position, element)
40. remove(position)
41. ListIterator listIterator()
42. Linked list methods,
43. addFirst(Object o)
44. addLast(Object o)
45. Object getFirst()
46. Object getLast()
47. And all above methods of ArrayList +linked List(ms)
48. Methods of HashSet
49. boolean contains(Object o)🡺 It is used to return true if this set contains the specified element.==> Set has this method instead of Get(posi) as Set does not have key or index to get the element based on key or index.
50. Methods of TreeSet,
51. Object first()🡺It is used to return the first (lowest) element currently in this sorted set.
52. Object last() 🡺 It is used to return the last (highest) element currently in this sorted set.
53. Methods of Queue,
54. boolean add(object)🡺 It is used to insert the specified element into this queue and return true upon success.
55. boolean offer(object)🡺 It is used to insert the specified element into this queue.
56. Object remove()🡺It is used to retrieves and removes the head of this queue 🡺 I think this throws some exception
57. Object poll()🡺It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
58. Object element()🡺It is used to retrieves, but does not remove, the head of this queue.
59. Object peek()🡺 It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

NOTE🡺 CLEAR DIFFERENCE BETWEEN THIS METHODS IS GIVEN SEE NOTES OR STACK OVERFLOW

1. [Java Deque & ArrayDeque](https://www.javatpoint.com/java-deque-arraydeque)🡺 LATER
2. Methods of Map
3. Object put(Object key, Object value)
4. void putAll(Map map)
5. Object remove(Object key)
6. Object get(Object key)
7. boolean containsKey(Object key)
8. Set keySet()
9. Set entrySet()
10. Entry is the sub interface of Map. 🡺 methods are 🡺 Object getKey() and Object getValue()
11. HashMap🡺 it may have one null key and multiple null value. It contains only unique elements. it maintains no order.
12. Methods of HashMap🡺
13. boolean containsValue(Object value)
14. boolean isEmpty()
15. Collection values()🡺It is used to return a collection view of the values contained in this map.
16. LinkedHashMap🡺 It may have one null key and multiple null values. It contains only unique elements. **It is same as HashMap instead maintains insertion order.**
17. TreeMap cannot have null key but can have multiple null values.
18. Methods of TreeMap,
19. Object firstKey() 🡺 It is used to return the first (lowest) key currently in this sorted map.
20. Object lastKey()🡺It is used to return the last (highest) key currently in this sorted map.
21. Hashtable🡺 It may have not have any null key or value.
22. Methods of HashTable🡺
23. void rehash()It is used to increase the size of the hash table and rehashes all of its keys.

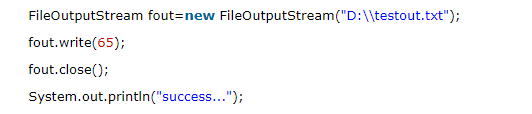
# Collections class🡺 has methods t Synchronize List, Set and Map and has the method to reverse(), and also has method to sort() a list

1. **Sorting a collection🡺 Comparable and Comparator interface🡺 needed to do hands on**
2. The java.util.Properties class is the subclass of Hashtable.
3. The Properties class provides methods to get data from properties file and store data into properties file.
4. Methods of Properties class
   1. 
5. By System.getProperties() method we can get all the properties of system

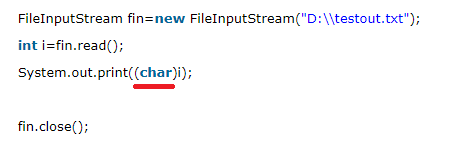


Java\_PrintOut4

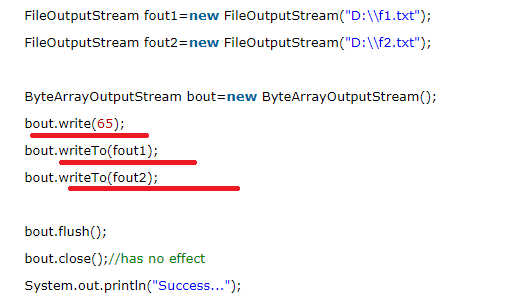
1. Java has Streams for Byte-orientation and Reader and Writer for Character-Orientation
2. OutputStream(is an abstract class and it’s the super class for all output stream class)🡺 for write and InputStream(is an abstract class and it’s the super class for all input stream class)🡺 do to read operation
3. OutputStream methods,
4. public void write(**int**)throws IOException
5. public void write(byte[])throws IOException
6. public void flush()throws IOException
7. public void close()throws IOException
8. Java output stream hierarchy
9. Java input stream hierarchy
10. FileOutputStream example🡺



1. FileIntputStream example🡺

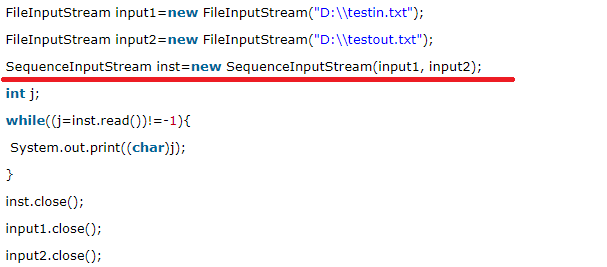


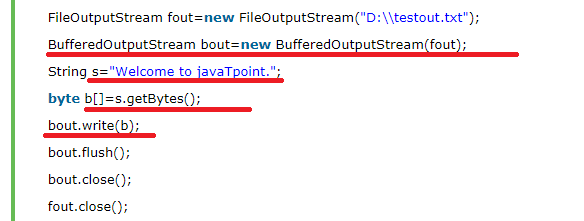
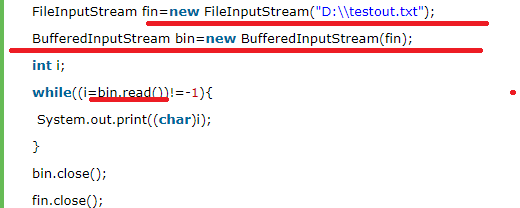
1. Java ByteArrayOutputStream class is used to **write common data** into multiple files, example,

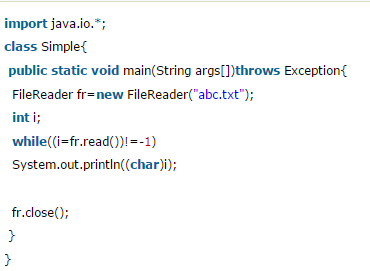


Ms🡺See here first content is written into BAOS by using write() and then writeTo() method is called to write into the files

1. Java SequenceInputStream class is used to read data from multiple streams. It reads data sequentially (one by one). Example🡺



1. **Needed more information on Enumeration**
2. BufferedOutputSTrem🡺  it makes the performance fast.
3. BufferedInputStream🡺
4. Reader and Writer🡺 needed hierarchy structure for this
5. Methods of Writer
6. append(char c), append(CharSequence csq), append(CharSequence csq, int start, int end)
7. write(char[] cbuf), write(int c), write(String str)
8. flush()
9. Reader methods🡺
10. read()
11. close()
12. reset()🡺It resets the stream.
13. skip(long n)🡺 It skips characters.
14. CharArrayWriter class can be used yto write data to multiple files
15. 



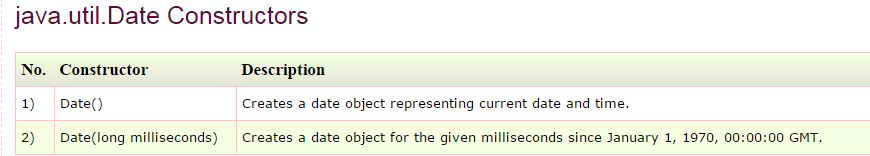
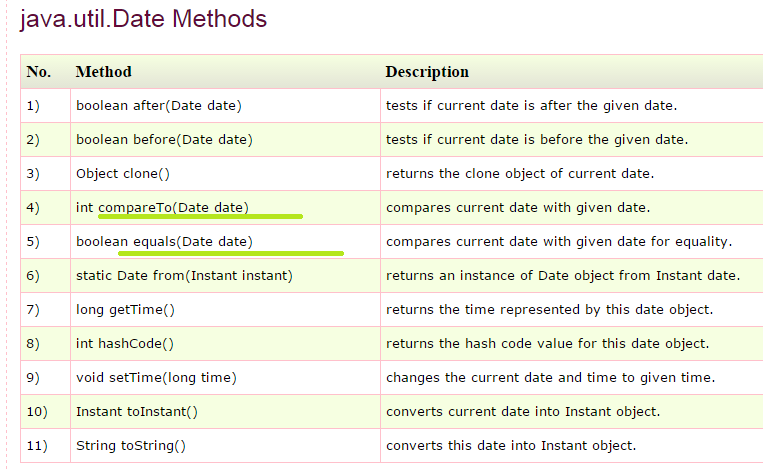
1. Reader example later

NOTE🡺 APART FROM THIS I HAVE OTHER STREAM AND READER CLASSES NEEDED TO COVER THOSE ALSO

Java PrrintOut5

1. Is the difference between Comparable and Comparator interface this we will see 🡺While doing hands on

Java PrintOut 6

1. 
2. 
3. System.currentTimeMillis();
4. java.sql.Date methods🡺
5. java.util.Calendar Java Calendar class is an abstract (Java Calendar class is an abstract) methods,
6. abstract void add(int field, int amount)🡺 It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar's rules.
7. **int get(int field)**
8. static Calendar getInstance()
9. **void set(int field, int value)**
10. void setTime(Date date)🡺 It is used to set this Calendar's time with the given Date.
11. Date getTime()🡺It is used to return a Date object representing this Calendar's time value.
12. Calendar class has lot of Properties/constant fields
13. **NEEDED TO KNOW HOW TO COMPARE DATE ARRAY AS ALREADY DATE CLASS IMPLEMENTS COMPARATOR**
14. **Collect few information on Constructor Chaining and Serialization🡺Qspider**

Core Java Interview Questions(Tpoint )

1. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.
2. delete,next,main,exit or null IS NOT A keyword in java
3. What is the purpose of default constructor 🡺The default constructor provides the default values to the objects. The java compiler creates a default constructor only if there is no constructor in the class
4. What if the static modifier is removed from the signature of the main method🡺 Program compiles. But at runtime throws an error "NoSuchMethodError".
5. **Yes, all functions in Java are virtual by default.**
6. Can you declare the main method as final🡺yes

### What is the base class for Error and Exception? 🡺Throwable.

Can a class have an interface?==> **Yes, it is known as nested interface.**

Can an Interface have a class? 🡺 **Yes, they are static implicitely.**

1. What kind of thread is the Garbage collector thread? 🡺 DAEMON thread

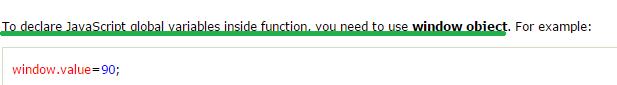
How will you invoke any external process in Java? **🡺** By Runtime.getRuntime().exec(?) method.

1. Serialization is a process of writing the state of an object into a byte stream.It is mainly used to travel object's state on the network.
2. **Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.**
3. If you define any data member as transient,it will not be serialized
4. Reflection is the process of examining or modifying the runtime behaviour of a class at runtime.
5. **The purpose of the System class is to provide access to system resources.**
6. **Singleton Class Program🡺**
7. **The shutdown hook is basically a thread i.e. invoked implicitely before JVM shuts down. So we can use it perform clean up resource**
8. ADVANTAGE OF PROPERTY FILE 🡺If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.
9. **Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.**

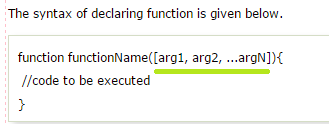
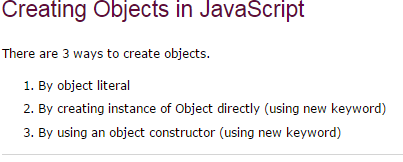
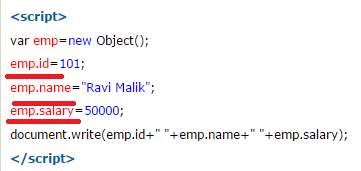
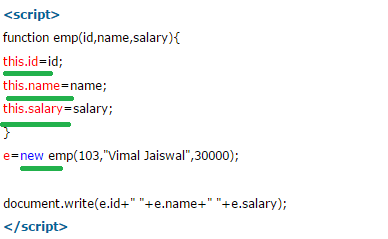
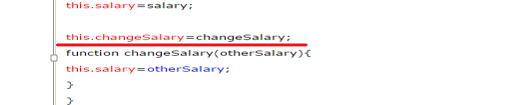


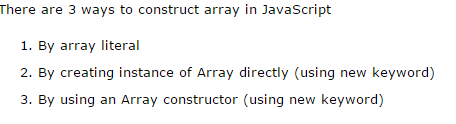
Html and CSS🡺 See the prepared notes that itself is shortly written

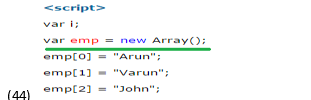
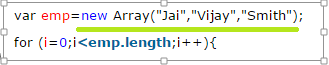
JavaScript

1. 
2. 
3. JS primitive data types🡺string, boolean, number, null and undefined
4. JS non-primitive or advanced data type 🡺Object, array and **RegExp**
5. JS special operators🡺(?:), delete, in, instanceOf, typeOf, void, yield and ,

**Yield🡺 checks what is returned in a generator by the generator's iterator.**

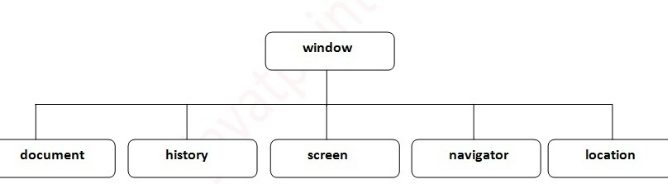
1. Needed more information on “for-in” loop in Javascript
2. 
3. 
4. 
5. By Creating instance of Object
6. By Using the Object Constructor🡺
7. Defining an method in the JS obj🡺
8. In JS Array is created in 3 ways🡺



1. 
2. 
3. 
4. JS string is created in 2 ways,
5. By String literal🡺 var stringname="string value";
6. **By string Obj🡺  var stringname=new String("string literal");**
7. JS String methods,

* charAt(index), concat(str), indexOf(str), lastIndexOf(str), toLowerCase(), toUpperCase(),slice(beginIndex, endIndex), trim()

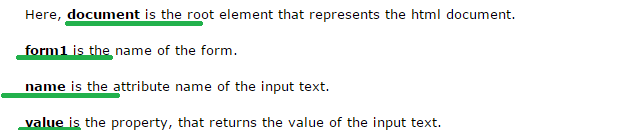
1. BOM🡺Browser object model

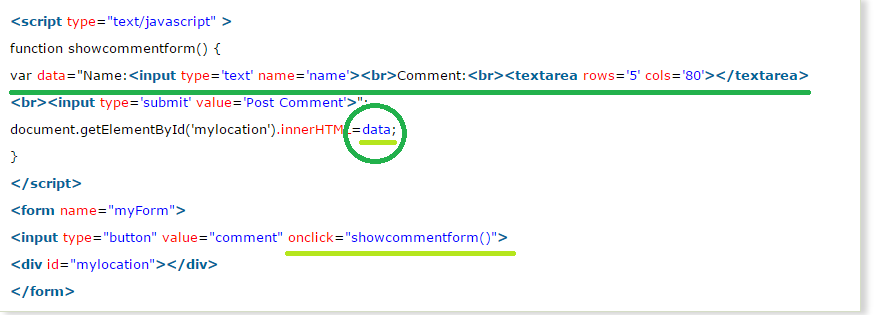


1. **Window is the object of browser, it is not the object of JavaScript**
2. window object method🡺alert()🡺ok button, confirm()🡺 ok and cancel button, prompt()🡺 has input text field and 2 buttons(ok, cancel), open()🡺 new browser window, close()🡺 Closes the current browser window, setTimeout()

setTimeout ()🡺performs action after specified time like calling function, evaluating expressions etc.

1. dom🡺 document object model🡺 ms represents HTML page object
2. dom Methods 🡺write(String), write(String), getElementById(),getElementByName(),getElementByTagName(), getElementByClassName()
3. we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.



1. The **innerHTML** property can be used to write the dynamic html on the html document 🡺
2. The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text
3. JavaScript number validation🡺 **Here, we are using isNaN() function to identify valid number is entered by user or not**
4. SEE THE KOUSHIK NOTES ON JS🡺Prototype and some basic information

JQuery

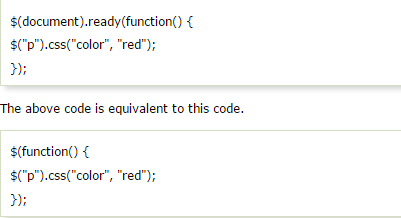
1. JQuery lightweight JavaScript library.
2. JQuery means "write less do more". jQuery simplifies AJAX call and DOM manipulation
3. **jQuery is developed by Google.**
4. **To create the first jQuery example, you need to use JavaScript file for jQuery. You can download the jQuery file from jquery.com or use the absolute URL of jQuery file.**
5. Syntax for absolute URL is shown below

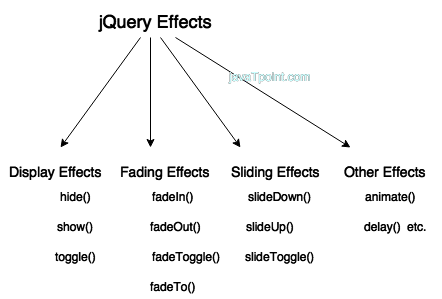
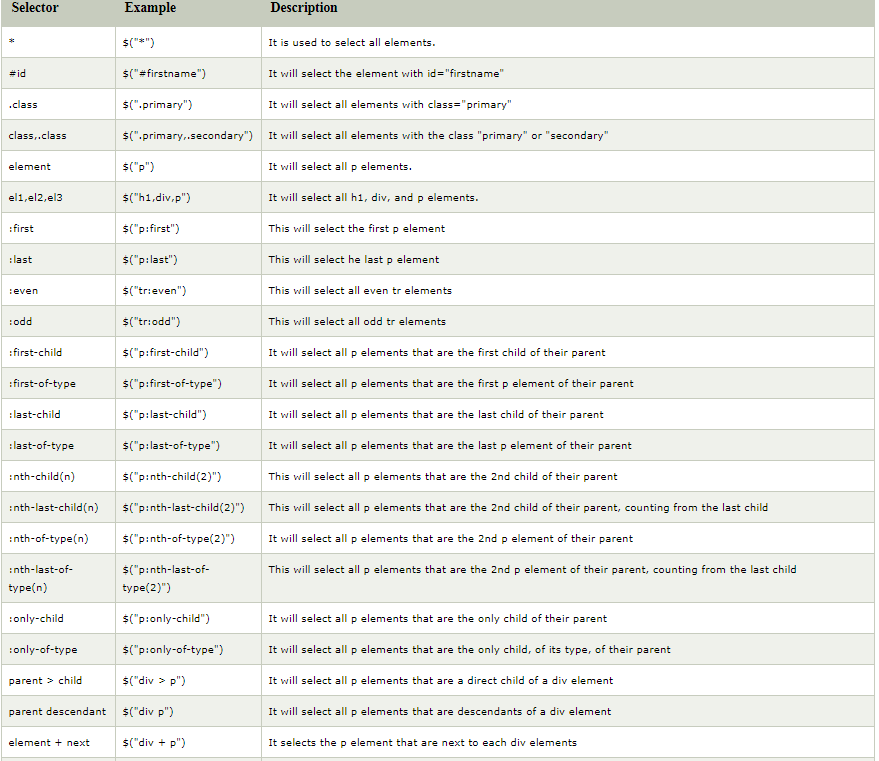
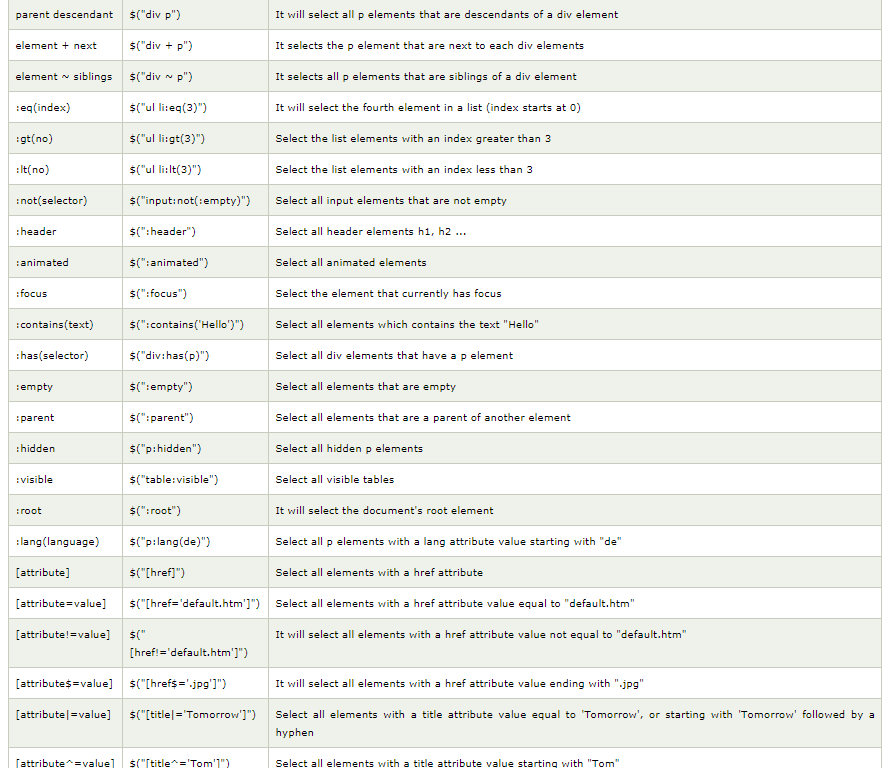


Note: In another Script tag we can write our JavaScript/JQuery Code

1. 
2. In place of $(document).ready(), you can use shorthand notation $() only.,

Example🡺



1. **All jQuery selectors start with a dollar sign and parenthesis e.g. $(). It is known as the factory function.**
2. 
3. Some of the CSS Selectors that was new
   1. 
   2. 
   3. 
4. jQuery Functions🡺
5. Graphic or UI Style related Functions,

clearQueue(),delay(),queue(), dequeue(),fadein(),fadeout(),fadeto(),fadetoggle(),finish(),hide(),show(),toggle(),slidedown(),slidetoggle(),slideup(),stop(),

1. CSS and HTML functions

JMS

1. JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.
2. Messaging is a technique to communicate applications or software components.
3. MS is mainly used to send and receive message from one application to another.

Requirement of JMS

1. Generally, user sends message to application. But, if we want to send message from one application to another, we need to use JMS API.

For Example,

Consider a scenario, one application A is running in INDIA and another application B is running in USA. To send message from A application to B, we need to use JMS.

## Advantage of JMS

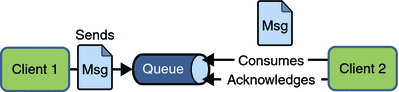
1. **Asynchronous:** To receive the message, client is not required to send request. Message will arrive automatically to the client.
2. **Reliable:** It provides assurance that message is delivered.

## Messaging Domains: There are two types of messaging domains in JMS

1. Point-to-Point Messaging Domain
2. Publisher/Subscriber Messaging Domain

## Point-to-Point (PTP) Messaging Domain

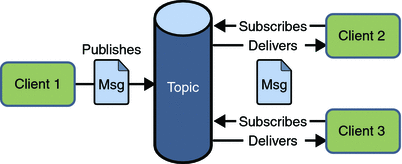
1. In PTP model, one message is **delivered to one receiver** only. Here, **Queue** is used as a message oriented middleware (MOM).The Queue is responsible to hold the message until receiver is ready.
2. In PTP model, there is **no timing dependency** between sender and receiver.



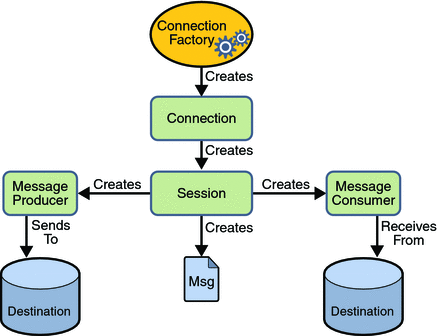
## Publisher/Subscriber (Pub/Sub) Messaging Domain

## In Pub/Sub model, one message is delivered to all the subscribers. It is like broadcasting. Here, Topic is used as a message oriented middleware that is responsible to hold and deliver messages.

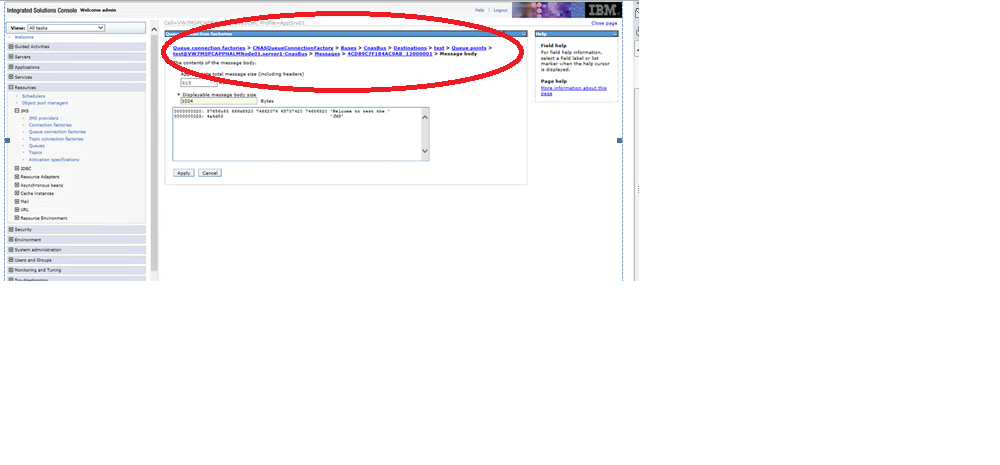
## In PTP model, there is timing dependency between publisher and subscriber.



## JMS Programming Model



## JMS Queue Example

1. Follow the following steps in order to start with JMS
2. **Create connection factory** named **myQueueConnectionFactory**
3. **Create destination resource** named **myQueue**
4. For a detailed explanation on Queue kindly see the below URL
5. 
6. The above URL will tell us, how to create QueueConnectionFactory and Queue
7. AS per my understanding it’s like QueueConnectionFactory has Bus(SIB🡺System Integration Bus) which in turn has the Queue{or Queue Destination}, so first create the Queue and Assign it to a bus, I think by default bus will be assigned to the QCF
8. Now let’s see the steps
9. To create the Bus and assigning the Server to the Bus as a bus member.
10. To create QCF
11. Queue
12. 
13. A JMS Queue is used as a destination for a Point-to-Point Messaging
14. Example code is same as present in the JavaTpoint but to get the InitialContext object we had to add the jars present in the runtime folder which is present in the Websphere7 folder

Notes from packtpub

1. url for WebSphere based JMS🡺 <https://www.packtpub.com/books/content/messaging-websphere-application-server-70-part-1>

<https://www.packtpub.com/books/content/messaging-websphere-application-server-70-part-1>

1. From The Above Url We Will Learn THE FOLLOWING
2. Here we will learn how to create **Queue Connection Factories (QCF)** and **Queue Destinations (QD)**

Note

1. Build is the process in which ear file is generated
2. Staging is the process in which the above generated ear file will be placed in the perspective location and .go file will be created.

**Self-join notes (must see this once)**

**From the Website 🡺http://www.programmerinterview.com/index.php/database-sql/what-is-a-self-join/**

**What is a self join? Explain it with an example and tutorial.**

Let’s illustrate the need for a self join with an example. Suppose we have the following table – that is called employee. The employee table has 2 columns – one for the employee name (called employee\_name), and one for the employee location (called employee\_location):

|  |
| --- |
| employee |
| |  |  | | --- | --- | | **employee\_name** | **employee\_location** | | Joe | New York | | Sunil | India | | Alex | Russia | | Albert | Canada | | Jack | New York | |

Now, suppose we want to find out which employees are from the same location as the employee named Joe. In this example, that location would be New York. Let’s assume – for the sake of our example – that we can ***not*** just directly search the table for people who live in New York with a simple query like this (maybe because we don’t want to hardcode the city name) in the SQL query:

SELECT employee\_name

FROM employee

WHERE employee\_location = "New York"

So, instead of a query like that what we could do is write a nested SQL query (basically a query within another query – which is more commonly called a subquery) like this:

|  |
| --- |
| SELECT employee\_name  FROM employee  WHERE employee\_location in  ( SELECT employee\_location  FROM employee  WHERE employee\_name = "Joe") |

**A subquery is inefficient**

Using a subquery for such a simple question is inefficient. Is there a more efficient and elegant solution to this problem?

It turns out that there is a more efficient solution – we can use something called a ***self join***. A self join is basically when a table is joined to itself. The way you should visualize a self join for a given table is by imagining that a join is performed between ***two identical copies of that table***. And that is exactly why it is called a ***self*** join – because of the fact that it’s just the same table being joined to another ***copy*** of itself rather than being joined with a different table.

**How does a self join work**

Before we come up with a solution for this problem using a self join, we should go over some concepts so that you can fully understand how a self join works. This will also make the SQL in our self join tutorial a lot easier to understand, which you will see further below.

**A self join must have aliases**

|  |
| --- |
|  |

In a self join we are joining the same table to itself by essentially creating two copies of that table. But, how do we distinguish between the two different copies of the table – because there is only one table name after all? Well, when we do a self join, the table names absolutely must use aliases otherwise the column names would be ambiguous. In other words, we would not know which of the two copies of the table’s columns is being referenced without using an alias for each copy of the table. If you don’t already know what an alias is, it’s simply another name given to a table – think of an alias as a nickname – and that nickname is then used in the SQL query to reference the table. Because we need two copies of the employee table, we will just use the aliases e1 and e2 for the employee table when we do a self join.

**Self join predicate**

As with any join there must be a condition upon which a self join is performed – we can not just arbitrarily say “do a self join”, without specifying some condition. That condition will be our join predicate. If you need a refresher on join predicates (or just joins in general) then check this link out: [Inner vs. Outer joins](http://www.programmerinterview.com/sql/inner-vs-outer-joins.php).

Now, let’s come up with a solution to the original problem using a self join instead of a subquery. This will help illustrate how exactly a self join works. The key question that we must ask ourselves is what should our join predicate be in this example? Well, we want to find all the employees who have the same location as Joe.

Because we want to match between our two tables (both of which are the same table – employee – aliased as e1 and e2) on location our join predicate should clearly be “WHERE e1.employee\_location = e2.employee\_location”. But is that enough to give us what we want? No, it’s not, because we also want to filter the rows returned since we only want people who are from the same location as Joe.

So, how can we filter the rows returned so that only people from Joe’s location are returned? Well, what we can do is simply add a condition on one of the tables (e2 in our example) so that it only returns the row where the name is Joe. Then, the other table (e1) will match up all the names that have the same location in e2, because of our join predicate – which is “WHERE e1.employee\_location = e2.employee\_location”. We will then just select the names from e1, and not e2 because e2 will only have Joe’s name. If that’s confusing then keep reading further to understand more about how the query will work.

So, the self join query that we come up with looks like this:

**Self Join SQL Example**

|  |
| --- |
| SELECT e1.employee\_name  FROM employee e1, employee e2  WHERE e1.employee\_location = e2.employee\_location  AND e2.employee\_name="Joe"; |

This query will return the names Joe and Jack – since Jack is the only other person who lives in New York like Joe.

Generally, queries that refer to the same table can be greatly simplified by re-writing the queries as self joins. And, there is definitely a performance benefit for this as well.

**What does a self join look like?**

It will help tremendously to actually visualize the actual results of a self join internally. Remember that a self join is just like any other join, where the two tables are merged into one temporary table. First off, you should visualize that we have two separate copies of the employee table, which are given aliases of e1 and e2. These copies would simply look like this – note that we shortened the column names from employee\_name and employee\_location to just Name and Location for convenience:

|  |  |
| --- | --- |
| e1 | e2 |
| |  |  | | --- | --- | | Name | Location | | Joe | New York | | Sunil | India | | Alex | Russia | | Albert | Canada | | Jack | New York | | |  |  | | --- | --- | | Name | Location | | Joe | New York | | Sunil | India | | Alex | Russia | | Albert | Canada | | Jack | New York | |

And the final results of running the self join query above – the actual joined table – would look like this:

|  |  |  |  |
| --- | --- | --- | --- |
| e1.employee\_name | e1.employee\_location | e2.employee\_name | e2.employee\_location |
| Joe | New York | Joe | New York |
| Jack | New York | Joe | New York |

**Self joins versus inner joins**

|  |
| --- |
|  |

Are self joins and inner joins the same? You might be wondering if all self joins are also inner joins. After all, in our example above our self join uses an inner join because only the rows that match based on the join predicate are returned – non-matching rows are not returned. Well, it turns out that a self join and inner join are completely different concepts. A self join could just as well be an outer join or an inner join – it just depends on how the query is written. We could easily change the query we used above to do a LEFT OUTER JOIN – while the query still remains a self join – but that wouldn’t give us the results we want in our example. So, we use an implied inner join instead because that gives us the correct results. Remember that a query is a self join as long as the two tables being joined are exactly the same table, but whether it’s an inner join or outer join depends on what is specified in the SQL. And, inner/outer joins are separate concepts entirely from a self join.

**Self joins manager employee example**

The most commonly used example for self joins is the classic employee manager table. The table is called Employee, but holds all employees – including their managers. Every employee has an ID, and there is also a column for the manager ID. So, for example, let’s say we have a table that looks like this – and we call it Employee:

|  |  |  |
| --- | --- | --- |
| EmployeeID | Name | ManagerID |
| 1 | Sam | 10 |
| 2 | Harry | 4 |
| 4 | Manager | NULL |
| 10 | AnotherManager | NULL |

Notice that in the table above there are two managers, conveniently named “Manager” and “AnotherManager”. And, those managers don’t have managers of their own – as noted by the NULL value in their Manager column.

Now, given the table above, how can we return results that will show each employee’s name, and his/her manager’s name in nicely arranged results – with the employee in one column and his/her manager’s name in the other column. Well, it turns out we can use a self join to do this. Try to come up with the SQL on your own before reading our answer.

**Self join manager employee answer**

In order to come up with a correct answer for this problem, our goal should be to perform a self join that will have both the employee information and manager information in one row. First off, since we are doing a self join, it helps to visualize the one table as two tables – let’s give them aliases of e1 and e2. Now, with that in mind, we want the employee’s information on one side of the joined table and the manager’s information on the other side of the joined table. So, let’s just say that we want e1 to hold the employee information and e2 to hold the corresponding manager’s information. What should our join predicate be in that case?

Well, the join predicate should look like “ON e1.ManagerID = e2.EmployeeID” – this basically says that we should join the two tables (a self join) based on the condition that the manager ID in e1 is equal to the employee ID in e2. In other words, an employee’s manager in e1 should have the manager’s information in e2. An illustration will help clarify this. Suppose we use that predicate and just select everything after we join the tables. So, our SQL would look like this:

SELECT \*

FROM Employee e1

INNER JOIN Employee e2

ON e1.ManagerID = e2.EmployeeID

The results of running the query above would look like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| e1.EmployeeID | e1.Name | e1.ManagerID | e2.EmployeeID | e2.Name | e2.ManagerID |
| 1 | Sam | 10 | 10 | AnotherManager | NULL |
| 2 | Harry | 4 | 4 | Manager | NULL |

Note that there are only 2 rows returned – this is because an inner join is performed, which means that only when there is a match between employee ID’s and manager ID’s will there be a result returned. And since there are 2 people without managers (who have a manager ID of NULL), they will not be returned as part of table e1, because no employees have a matching ID of NULL.

Now, remember that we only want to return the names of the employee and corresponding manager as a pair. So, we can fine-tune the SQL as follows:

SELECT e1.Name, e2.Name

FROM Employee e1

INNER JOIN Employee e2

ON e1.ManagerID = e2.EmployeeID

Running the SQL above would return:

Sam AnotherManager

Harry Manager

And that is the answer to the employee manager problem using a self join! Feel free to post any comments.

[**Hiring? Job Hunting? Post a JOB or your RESUME on our JOB BOARD >>**](http://www.programmerinterview.com/jobs)

[Subscribe to our newsletter for more free interview questions.](http://www.programmerinterview.com/index.php/database-sql/what-is-a-self-join/)

[Follow @programmerintvw](https://twitter.com/programmerintvw)

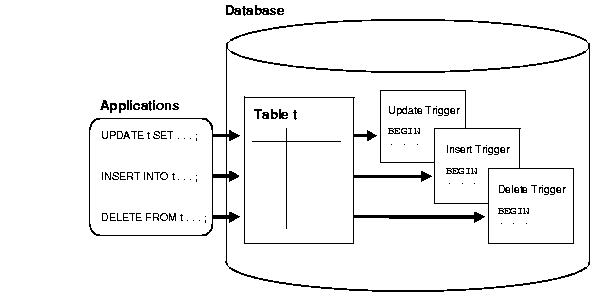
**SQL NOTES**

28/2/2017

This Notes are prepared from various web site and one among them for View is 🡺

<http://sql-plsql.blogspot.in/2009/03/insert-update-delete-views.html>

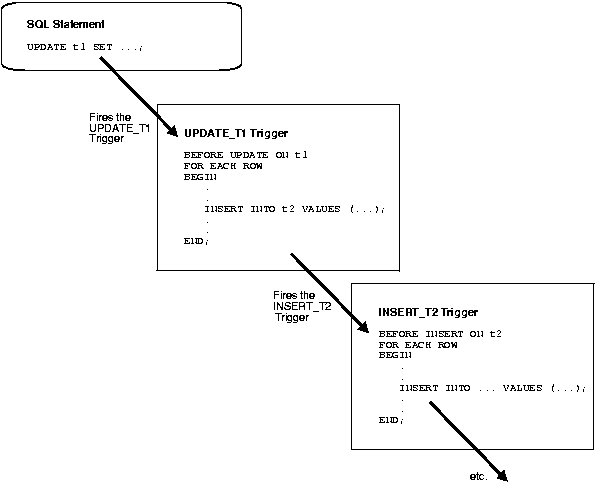
1. Triggers, which are procedures written in PL/SQL, Java, or C that run (fire) implicitly whenever a table or view is modified or when some user actions or database system actions occur.
2. You can write triggers that fire whenever one of the following operations occurs: DML statements on a particular schema object, DDL statements issued within a schema or database, user logon or logoff events, server errors, database startup, or instance shutdown.
3. Oracle lets you define procedures called **triggers** that run implicitly when an INSERT, UPDATE, or DELETE statement is issued against the associated table or, in some cases, against a view, or when database system actions occur. These procedures can be written in PL/SQL **or Java and stored** in the database, or they can be written as C callouts.
4. Graphical representation,



1. Uses of the Triggers,
2. A trigger can restrict DML operations against a table to those issued during regular business hours.
3. Automatically generate derived column values
4. Prevent invalid transactions
5. Modify table data when DML statements are issued against views 🡺as we know that views are mirror image of the tables used for security purpose.

***Cascading Triggers***

Let’s see the graphical image to understand the cascading triggers,



## Parts of a Trigger

1. A trigger has three basic parts:

* A triggering event or statement
* A trigger restriction
* A trigger action

### Trigger Restriction

1. A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. The trigger action is not run if the trigger restriction evaluates to false or **unknown**.

### Trigger Action

A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be run when the following events occur:

* A triggering statement is issued.
* The trigger restriction evaluates to true.

## Types of Triggers

the different types of triggers:

* [Row Triggers and Statement Triggers](https://docs.oracle.com/cd/B10500_01/server.920/a96524/c18trigs.htm#2174)

* [BEFORE and AFTER Triggers](https://docs.oracle.com/cd/B10500_01/server.920/a96524/c18trigs.htm" \l "789)

* [INSTEAD OF Triggers](https://docs.oracle.com/cd/B10500_01/server.920/a96524/c18trigs.htm" \l "10901)

* [Triggers on System Events and User Events](https://docs.oracle.com/cd/B10500_01/server.920/a96524/c18trigs.htm" \l "10004)

### Row Triggers and Statement Triggers

Row Trigger 🡺 Once for every row affected by the triggering statement, such as a trigger fired by an UPDATE statement that updates many rows 🡺 For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.

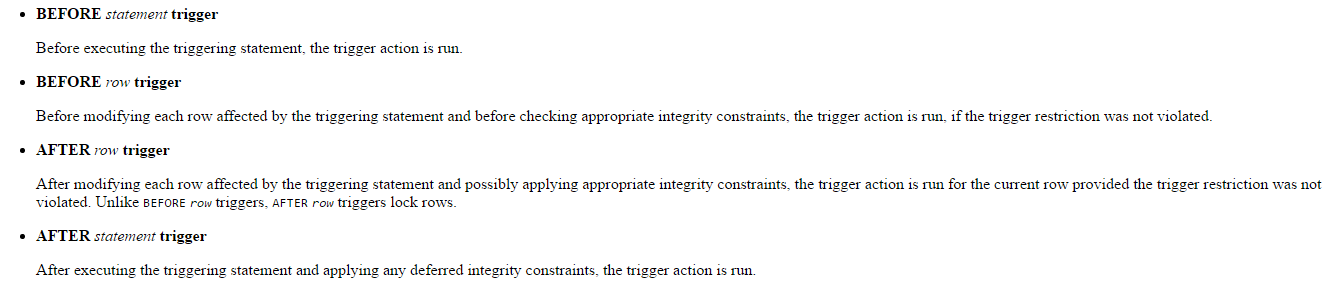
Statement Triggers🡺Once for the triggering statement, no matter how many rows it affects

### BEFORE and AFTER Triggers

When defining a trigger, you can specify the **trigger timing**--whether the trigger action is to be run before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

* BEFORE triggers run the trigger action before the triggering statement is run 🡺 uses is 🡺To derive specific column values before completing a triggering INSERT or UPDATE statement.
* AFTER triggers run the trigger action after the triggering statement is run.

#### Trigger Type Combinations



### INSTEAD OF Triggers

(1) INSTEAD OF triggers provide a transparent way of modifying views that cannot be modified directly through DML statements (INSERT, UPDATE, and DELETE). These triggers are called INSTEAD OF triggers because**,** unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement. 🡺 What does this statement means here?

(2) You can write normal INSERT, UPDATE, and DELETE statements against the view and the INSTEAD OF trigger is fired to update the underlying tables appropriately. INSTEAD OF triggers are activated for each row of the view that gets modified.

But before Continuing with instead of trigger, let’s see what View is and how it works

**Views**

1. **View**, which is a logical table based on one or more tables or views. A view contains no data itself. The tables upon which a view is based are called **base tables**.
2. We have two types of Views, namely Object View and Relational View
3. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.
4. **View is used when we have to give access to limited amount of data. We can give permission to query a view for a table while denied access to the original table. It acts as a security measure.**

#### Modify Views

1. I was able to create a view for my table called usha\_table, the view was having all the records as my table usha\_table. But I was not able to do insert operation on View🡺 TRY THIS LATER
2. Deleting a row in a view could either mean deleting it from the base table or updating some values so that it is no longer selected by the view.
3. Inserting a row in a view could either mean inserting a new row into the base table or updating an existing row so that it is projected by the view.
4. Updating a column in a view that involves joins might change the semantics of other columns that are not projected by the view.

NOTE 🡺 MY UNDERSTANDING IS INSTEAD OF TRIGGER IS USED TO DO OPERATION ON VIEWS WHICH WILL INDIRECTLY MODIFIES THE BASE TABLE.

**Different ways of Inserting a values into database**

1. T

insert into EmpDtl2 (EmpName)

select EmpName from EmpDtl1

**Insert query result into new table**

select \* into EmpCopy from EmpDtl1

1. Below query will insert data into table "EmpDtl1" returned by executing procedure.

insert into EmpDtl1

exec spGetEmpDetails

1. Suppose we want to create a table which is exactly the copy of a given table then we can also use the following Sql query

SELECT \* INTO tbl\_studentscopy FROM tbl\_students

# 

**Can a view be updated/inserted/deleted? If Yes under what conditions?**

**A View can be updated/deleted/inserted if it has only one base table if the view is based on columns from one or more tables then insert, update and delete is not possible.**

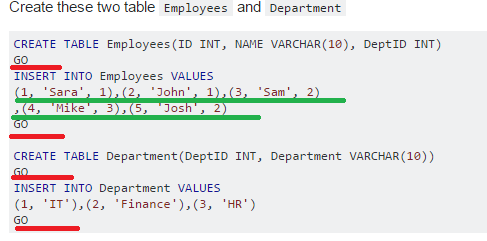
Some other conditions on which DML Operation can be done on view is,

* View can be update/insert/delete if it contain fields of one table, **and 1.primary key should contain in select clause used for view**
* 2. But again if select stmt used in view contains group by clause then we can’t update view.
* 3. **If select clause contains more than one tables then we can use instead of trigger for dml operations**.

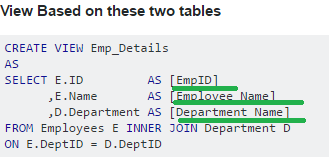
if the DML operation (Update/Delete/Insert) effects only one underlying table of your View the chances are you can do it. if the the operation effects more than one table you need to use Instead of Triggers

**Now let’s see the syntax for doing DML on Views (Created from more than one table) step by step**

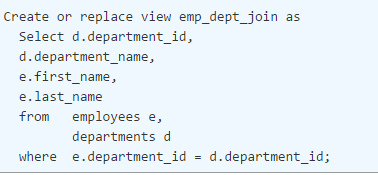
Step 1 🡺Create these two table Employees and Department



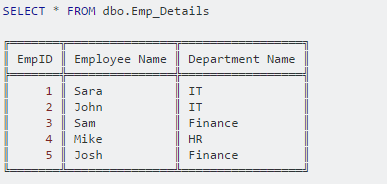
Step 2 🡺 Now let’s create the View



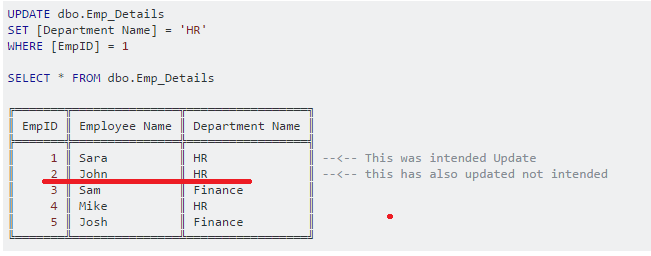
Another Example image I had pasted down for my understanding,



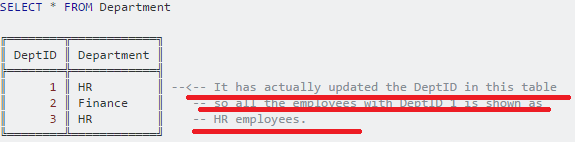
Step 3 🡺now let’s see the **Result Set of View**



Step 4 🡺 **Do an Update on View 🡺**even though UPDATE only effects One underlying table.

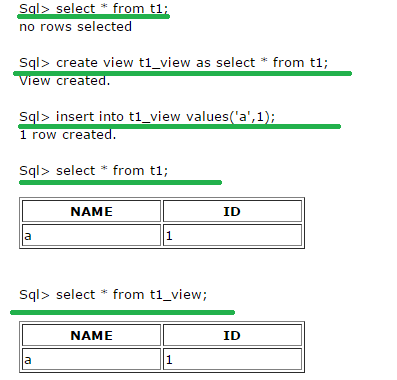


Now let’s see the **Department Table** after executing the above update on view,

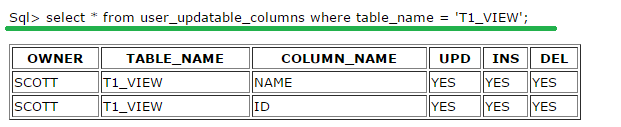


**Finally, I did all this because when you Issue an update statement against multiple Underlying tables it throws and error saying Update is not allowed because it effects multiple tables.**

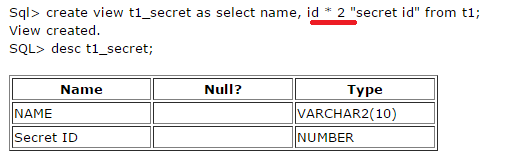
**Now let’s see whether you can check whether the particular columns in a view are updateable or not:**

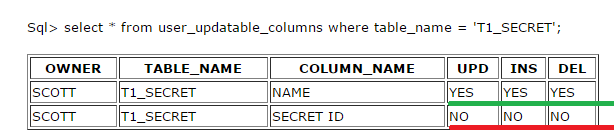


You can check whether the particular columns in a **view are updateable** or not🡺for the t1\_view created above we can check like this



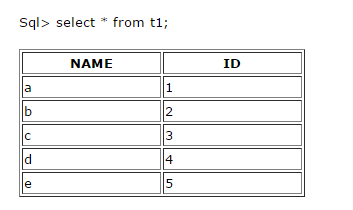
**Now let’s see how to make a View as Unmodifiable**

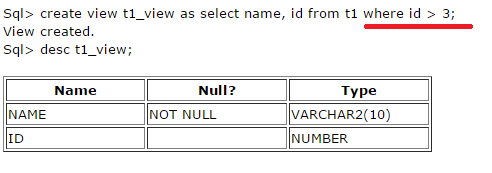




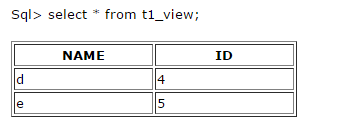
**Now let’s see the Syntax to create view WITH CHECK OPTION**

Step 1 🡺 Consider the following table with the records as shown below,



Step 2 🡺 Now let’s create a view,

Step 3🡺 now let’s see the View data

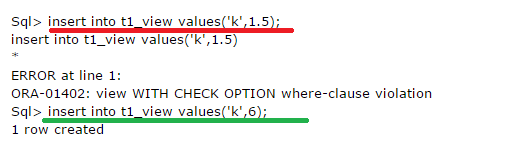


**Note 🡺if we insert any row in the table through view with id less than or equal to 3 it will let us insert that.**

**Now we will use *With Check* option to create that view again**:



NOTE 🡺 Now we will not be able to insert any row less than or equal to 3, only greater than 3 so as to match the view definition.

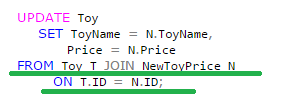


**Different way of updating an value into table**

## Updating Multiple Columns 🡺 Syntax below,

UPDATE Toy SET **ToyName = 'MagicSurfer',  Price = 15.99** WHERE ID = 4;

## Updating a Table Based on Values in a Different Table

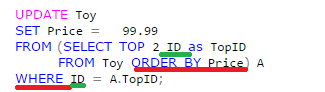


## Limiting the Rows that get Updated Using Top Clause

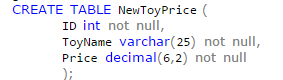
1. ms 🡺We already aware of “WHERE” clause which is used to limit the rows, in addition to where clause we also have TOP clause for limiting the rows.

Syntax 🡺

This code updates two **random** rows in my Toy table because TSQL doesn’t guarantee order unless you have an order by clause.  **If you want to update two rows based on the ORDER BY clause then you will need to use the TOP clause in a sub query in conjunction with a WHERE constraint, like in the following example.**



In order to have a clear understanding of the above query, let us see the Toy table structure first,

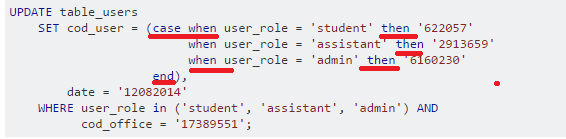


**Now let’s see the Insert query**

1. **Use**INSERT IGNORE**to ignore the duplicate rows**: Syntax 🡺**insert ignore into t1 select \* from t2;**

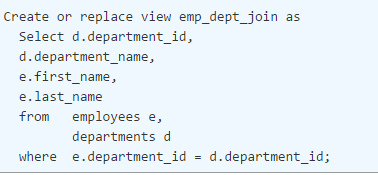
**Note** 🡺 It only ignores rows that violate a unique index. If I have no unique index on a column, IGNORE has no effect.

## Inserting new rows and updating existing rows in a Single Query 🡺 LATER

1. Now let’s see the syntax for using the case in update

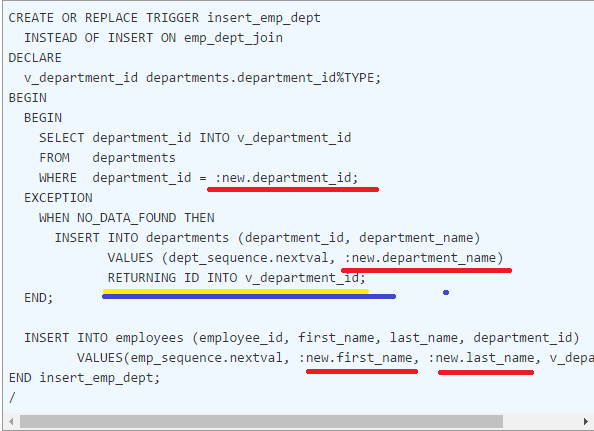
**Now let’s see the Working of Instead of Trigger with Example**

Step 1🡺 First let’s create a view for an Employee and Department table,



As the view consists of two table joins, it is illegal to insert records into this view as the insert requires both the underlying tables to be modified.

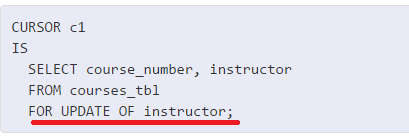
**By creating an instead-of trigger on the view, you can insert the records into both the underlying tables**. An example is shown below:



YELLOW and BLUE color 🡺 **see here its different insert Query is returning the Id column’s value that is the sequence number**

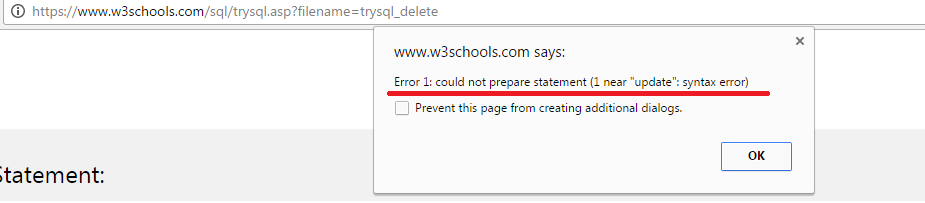
1/3/2017

**Update For**

1. The FOR UPDATE clause is an optional part of a [SELECT statement](http://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj41360.html#rrefsqlj41360).
2. Cursors are read-only by default.
3. The FOR UPDATE clause specifies that the cursor should be updatable, and enforces a check during compilation that the SELECT statement **meets the requirements** for an updatable cursor , below screen shot is an example **
4. **Syntax 🡺**

**SELECT RECEIVED, SOURCE, SUBJECT, NOTE\_TEXT FROM SAMP.IN\_TRAY FOR**

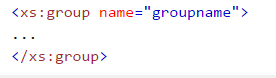
**UPDATE**

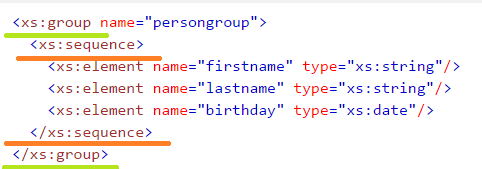
Note 🡺 I tried executing the above query but was not able to run, I am getting the following error message

**XSD/XML extra information which was not present in the xml notes that I have**

### Element Groups:

1. Element groups are defined with the group declaration, like this:



1. You must define an all, choice, or sequence element inside the group declaration.
2. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

* attributeGroup name

1. 

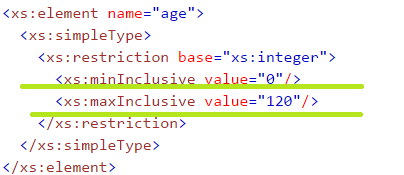
## XSD the <any> Element and The <anyAttribute> Element

# XSD Element Substitution🡺Later

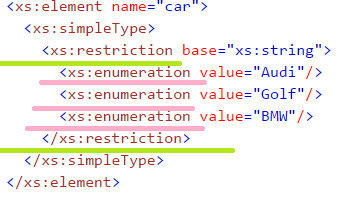
# XSD Restrictions/Facets

1. Restrictions are used to define acceptable values for XML elements or attributes.
2. Restrictions on XML elements are called **facets**.

# Restrictions on Values

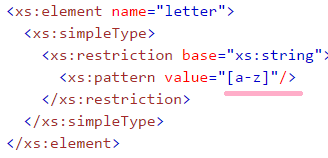
1. 

# Restrictions on a Set of Values

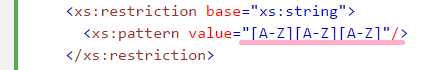
1. To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.
2. 

# Restrictions on a Series of Values

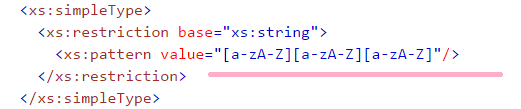
1. To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
2. The example below defines an element called "letter" with a restriction. The only acceptable value is **ONE** of the LOWERCASE letters from a to z:



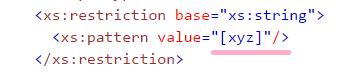
1. Another example:



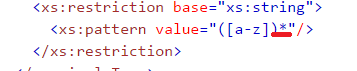
1. Example🡺The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z



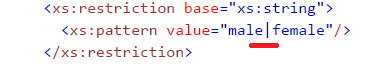
1. Example🡺The only acceptable value is ONE of the following letters: x, y, OR z



1. Example🡺The acceptable value is **zero or more** occurrences of lowercase letters from a to z



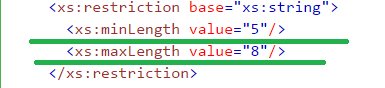
1. Example🡺The only acceptable value is **male OR female**



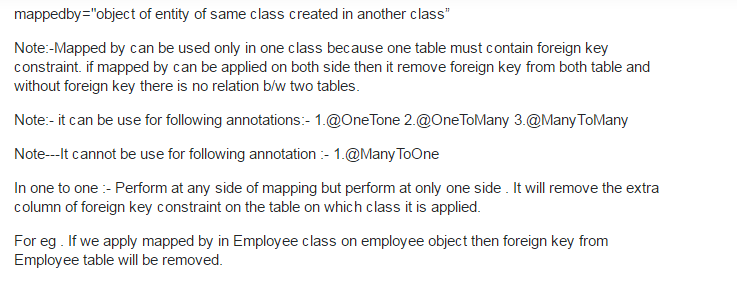
1. And See the tutorial for other examples…..
2. **Restrictions on Length**
3. To limit the length of a value in an element, we would use the **length, maxLength, and minLength** constraints.
4. Example🡺The value must be exactly eight characters



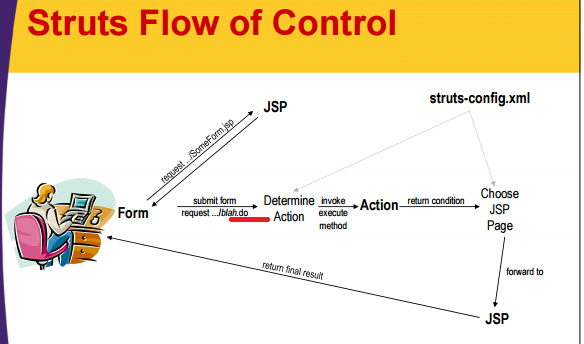
1. The value must be minimum five characters and maximum eight characters,



Few information on Mappedby in hibernate from Stack overflow



**Struts 1.2**

1. Struts provides utility classes to handle many of the most common tasks in Web application development
2. 
3. **Explanation of struts flow is**,

RED color🡺In struts action attribute of the form tag should have .do extension🡺MS

And –That address is mapped by struts-config xml to an Action class and then The execute method of the Action object is invoked

In our case PRHTMLAction extends BaseBondlineInputAction.

BBIAction is an abstract class and it extends BaseBondlineaction

BBAction implements execute() method and it’s an abstract class🡺 here logic is written in such a way that execute() method of BBAction calls performRequest() method which is implemented by its child

PRHTMLAction is the child class that implements performRequest () method.

Now let’s see the syntax for execute () 🡺 Book

Now let’s see the syntax for performRequest ()🡺 Book

The Action object then invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope.

The Action uses **actionMapping.findForward** to return a condition, and the conditions are mapped by struts-config.xml to various JSP pages🡺 Syntax for this is

Final ActionForward forward = actionMapping.findForward(“summary”);

Struts forwards request to the appropriate JSP page Struts forwards request to the appropriate JSP page

1. **The Six Basic Steps in Using Struts**

struts-config.xml 🡺 Use WEB-INF/struts-config.xml to:

Map incoming .do addresses to Action classes

Map return conditions to JSP pages Map return conditions to JSP pages

Declare any form beans that are being used.

Be sure to restart the server after modifying struts-config.xml; the file is read only when the Web application is first loaded.

Define a form bean🡺

This bean is a class extends ActionForm and will represent the data submitted by the user. It is automatically populated when the input form is submitted. Beans are postponed until the next section.

Create results beans🡺

In the MVC architecture, the business-logic and data-access code create the results and the JSP pages present them. To transfer the results from one layer to the other, they are stored in beans. These beans differ from form beans in that they need extend no particular beans differ from form beans in that they need extend no particular class, and they represent the output of the computational process, not the input to the process. Beans will be discussed in the next section.

NOTE 🡺 DIFFERENCE BETWEEN ABOVE POINT “C” AND “D” IS 🡺

The form bean represents the input data – I.e., the data that came from the HTML form.

The results beans represent the output data – I.e., the data created by the business logic to represent the results of the computation or database lookup.

**Note: 🡺 They are stored in request, session, or application scope with the setAttribute method of HttpServletRequest, HttpSession, or ServletContext, respectively.**

Define an Action class to handle requests🡺

The struts-config.xml file designates the Action classes that handle requests for various URLs The Action objects themselves need to requests for various URLs. The Action objects themselves need to do the real work: invoke the appropriate business- and data-accesslogic, store the results in beans, and designate the type of situation (missing data database error success category 1 success category (missing data, database error, success category 1, success category 2, etc.) that is appropriate for the results. The struts-config.xml file then decides which JSP page should apply to that situation.

**Note: Action Class🡺 Instead of reading form data explicitly with request.getParameter, the execute method uses a bean that is automatically q filled in from the request data.**

Create form that invokes blah.do🡺

Create an input form whose ACTION corresponds to one of the .do addresses listed in struts-config.xml.

Rather than using the standard HTML FORM and INPUT tags, we now use **html:form** and **html:text** (and related tags).

Display results in JSP🡺

Since Struts is built around MVC, **these JSP pages should avoid JSP scripting elements whenever possible**. For basic Struts, these pages usually use the bean:write tag, but in JSP 2.0 the JSP 2.0 expression language is a viable alternative.

In most cases the JSP pages only make sense when the request is In most cases, the JSP pages only make sense when the request is funneled through the Action, so the pages go in WEB-INF.

If the JSP pages makes sense independently of the Action (e.g., if they display session data), then the JSP pages should be placed in a regular subdirectory of the Web application, and the forward entries in struts-config.xml should say.

As before, the JSP page uses **bean:write** to output properties of the form and result beans, It may also use **bean:message** to output standard messages and text labels that are defined in a properties file.

struts-config.xml file🡺 Book

**Step 6 (Display results in JSP) in struts-config.xml🡺** In general, there can be several possible JSP pages🡺– Corresponding to the various possible return values of the execute method of the Action execute method of the Action. • In struts-config.xml, each JSP page is declared in a forward entry within the appropriate action.

**Chapter -3**

**Struts-Beans:**

One of the arguments to execute is a form bean that **is automatically created and whose properties are automatically populated based on incoming request parameters of the same name**

**Or**

A **form bean is a Java object that will be automatically filled in based on the incoming form parameters then filled in based on the incoming form parameters, then passed to the execute method.**

Bean must extend ActionForm

Bean must be declared in struts-config.xml with form-beans

For each incoming request parameter, the corresponding setter method and The validate method is called by struts framework itself

Ms🡺uses of the Beans we might see during the Struts tag reading

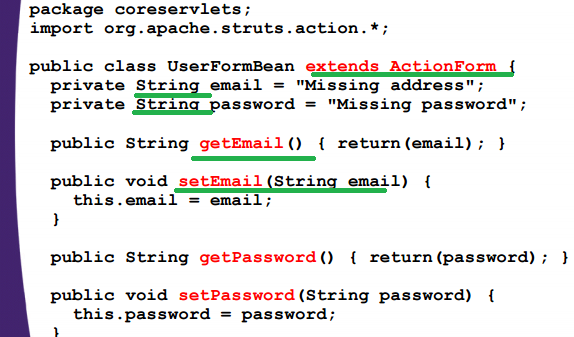
Rules for creating a bean class is,

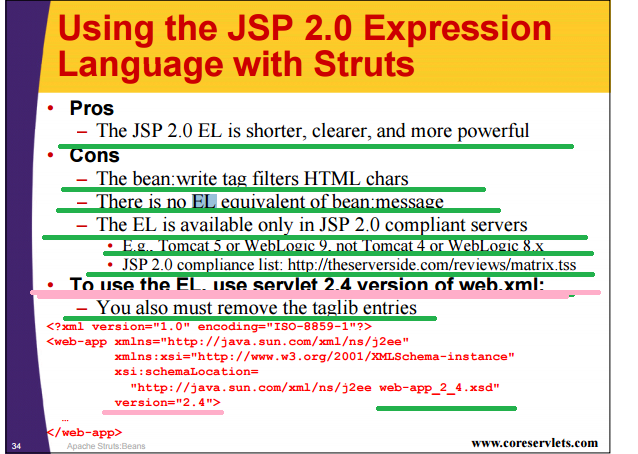
– It must extend ActionForm🡺 The argument to execute is of type ActionForm Cast the value to your The argument to execute is of type ActionForm. Cast the value to your real type, and then each bean property has the value of the request parameter with the matching name.

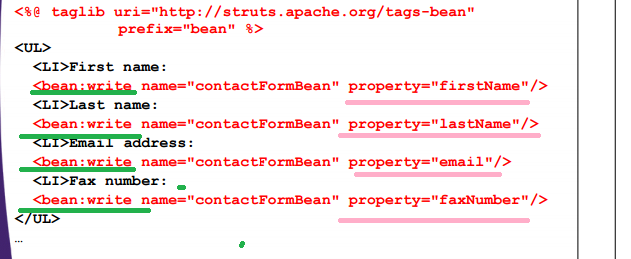
It must have a zero argument constructor🡺 The system will automatically call this default constructor.

It must have settable bean properties that correspond to the request parameter names🡺 That is, it must have a setBlah method corresponding to each incoming request parameter named blah.

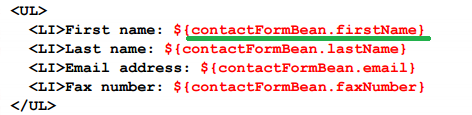
It must have gettable bean properties for each property that you want to output in the JSP page.





Input form example: 🡺

Output JSP example:



**Chapter -4**

**Struts –Forms**

Struts HTML tags

Three Characteristics of html tags:

The textfield names and the bean properties are guaranteed to stay in synch

The textfield values can be prepopulated based on the values in a bean🡺 That is, the initial values of the form elements can be taken from a Java object

--

Rather than using the standard HTML FORM and INPUT tags, we now use html:form and html:text (and related tags).

**The html:form tag associates a bean with the form, and html:text automatically uses bean property names for each textfield NAME and bean property values for each textfield VALUE.**

**Syntax for html:text is 🡺**

**<html:text property="firstName" />**

Explanation for above syntax 🡺Each textfield NAME will be taken from the bean property name, and each textfield VALUE will come from the bean property value.

**In addition, as in the previous section, this form can still use the bean:message tag to output standard messages and text labels.**

**Using the Struts html:form element instead of the standard HTML FORM element yields four results:**

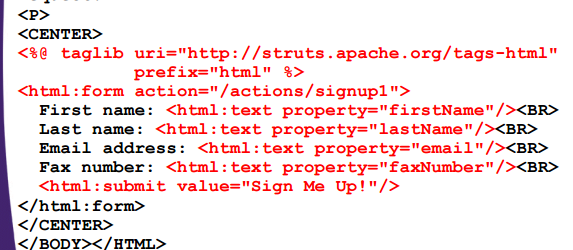
A bean is associated with the form.

The .do suffix is appended automatically

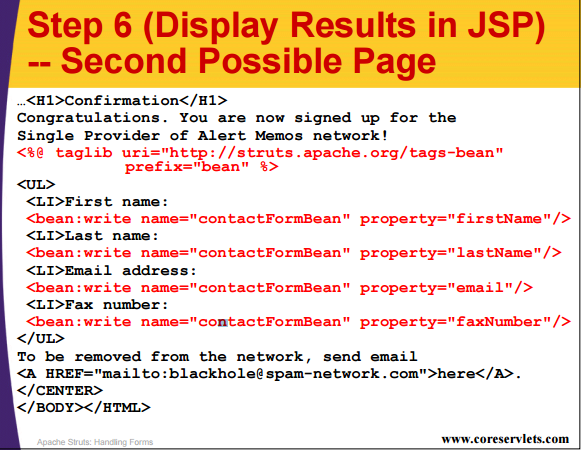
* You say<html:form action = "/actions/blah”> to get <html:form action = ""/webAppPrefix/actions/blah.do”>
* **POST, not GET, is the default METHOD**🡺You say <html:form action = "/actions/blah”> to get <html:form action = ""/webAppPrefix/actions/blah.do” **METHOD="POST"**>

1. Use **html:xxxx** for other input elements 🡺In your input form, use html:button, html:checkbox, html:textarea, etc., to declare submit buttons, checkboxes, text areas, etc
2. Syntax to get inputs from user:

* Top of Form



Output JSP



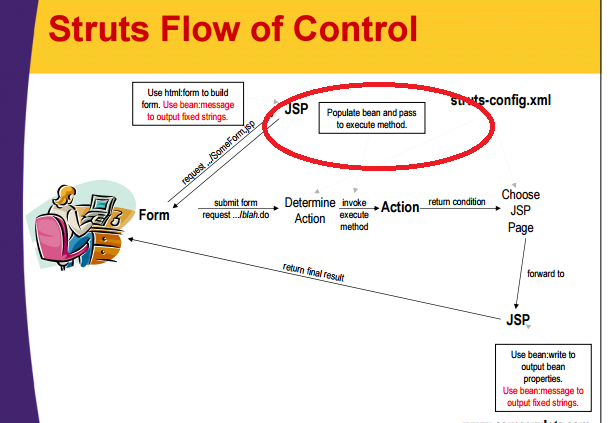
**Chapter 5🡺**

**Struts Messages**

MS 🡺 Difference between **bean:write** and **bean:message**

– Using bean:write to output bean properties

--Using bean:message to output constant strings



**PROPERTIES FILE IN STRUTS**

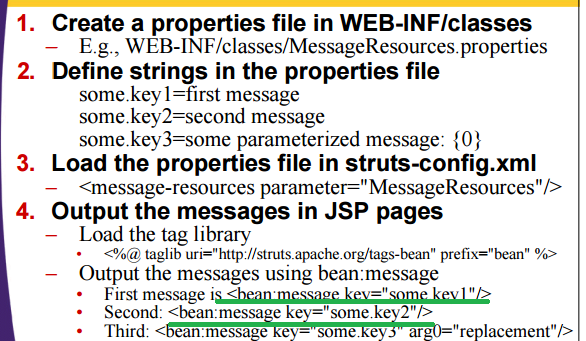
Advantage of using Properties file

Easy Maintenance

Centralized updates🡺If a message is used in several places, it can be updated with a single change.

Supports I18N

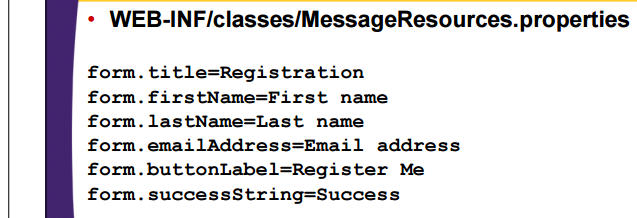
Steps to use the Properties file in struts,



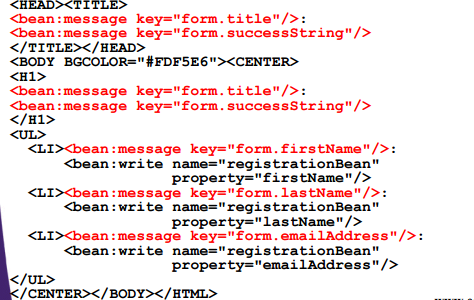
**Syntax for using the Property files,**

**<message-resources parameter="MessageResources" null="false"/>**

Complete struts-config.xml file with the <message-resources> see the Book or tutorials

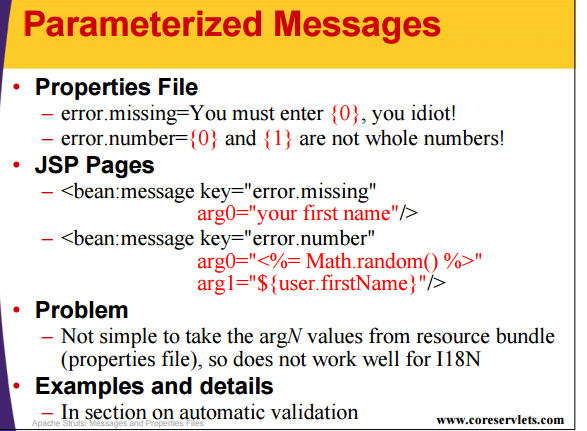
**Syntax for Property files 🡺**

**Example for How we use the content of property files**

**Example for both bean:write and bean:message elements**

Syntax for internationalization 🡺later we wills see

**Parameterized Messages**

1. 

**Chapter -6**

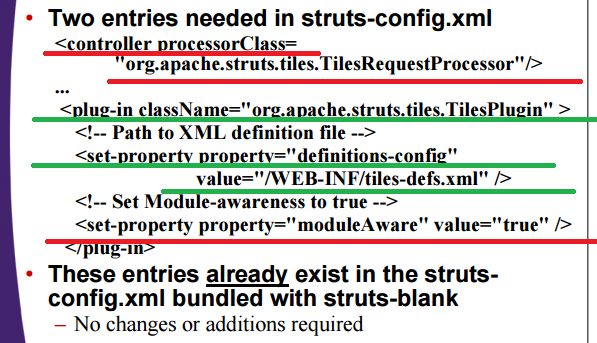
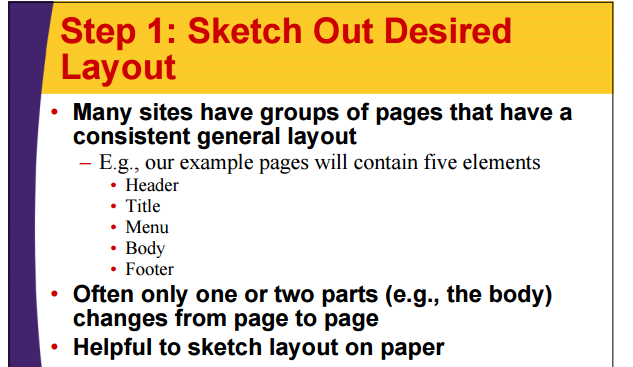
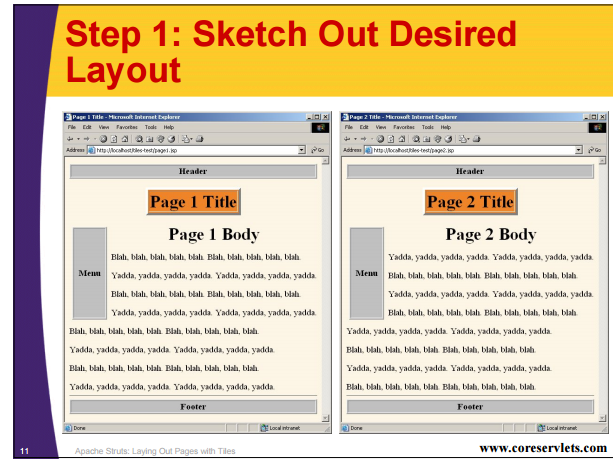
**Advanced \_Actions**

**DispatchAction**

1. In, HTML, one form has one ACTION
2. Needed to see the Syntax and Uses of this class

**Chapter – 10**

**Struts-Tiles**

1. Tiles basics
2. – Sketch out desired layout
3. – Make template file that represents layout
4. – Create JSP pages that define layout pieces
5. – Create JSP pages that populate layout
6. Tiles Motivations
7. **• Reuse (not rewrite) repeated sections of pages**
8. **• Simplify the creation of similar pages**
9. **• Increase flexibility and ease of maintenance compared to .**
10. Tiles Prerequisites
11. 
12. 
13. Example for Layout

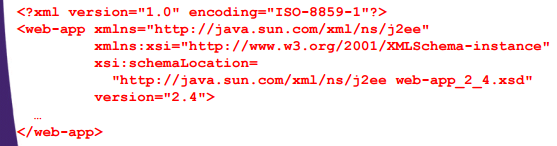
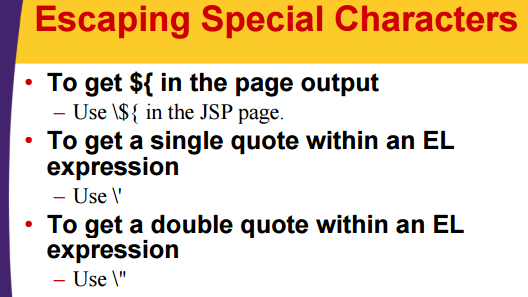
NOTE 🡺IN ORDER TO USE TILES WE HAVE TO ADD <taglib> directive like we do to use html, bean and logic tags of struts framework

Syntax: 🡺

**<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" />**

1. **Our logic in application is different from one discussed here🡺NEEDED TO DISCUSS WITH TEAM**

**Struts –and-JSP2-EL**

1. Available only in servers that support JSP 2.0 (servlets 2.4) 🡺E.g., Tomcat 5, not Tomcat 4
2. You must use the JSP 2.0 web.xml file 🡺 
3. Syntax 🡺 
4. 

**NOTE🡺 TO GET MORE UNDERSTANDING ON THIS TOPIC FIRST SEE THE JSP AND THEN COME BACK AND PREPARE THE NOTES**

**Using the JSP Standard Tag Library (JSTL) with Struts**

1. JSTL is the recommended replacement for the Struts looping and logic tags
2. •JSTL is not part of the JSP 1.2 or 2.0 Specs
3. It is a separate specification that requires a separate download
4. Available only in servers that support servlets 2.3 and JSP 1.2 or later. Cannot be retrofitted into JSP 1.1.
5. **The JSTL expression language is now part of JSP 2.0**
6. Installing JSTL
7. Some servers come with JSTL preinstalled 🡺 – E.g., Caucho Resin
8. JSTL (like JSP) is a specification, not an implementation
9. To install:
10. Download zip file
11. Unzip into directory of your choice (e.g., C:\jakarta-taglibs)
12. Copy install\_dir/standard-1.0.1/lib/jstl.jar and install\_dir/standard-1.0.1/lib/standard.jar to the WEB-INF/lib directory of your Web application

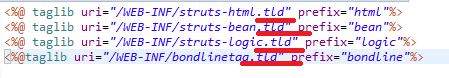
NOTE 🡺 UNDER THIS TOPIC ONLY JSP’S JSTL TAGS ARE EXPLAINED IT DON’T HAVE EXPLAINATION FOR STRUTS TAG

FOLLOWING NOTES ARE PREPARED FROM SOME OTHER TUTORIALS🡺

1. LOGIC TAGS 🡺
2. <logic:iterate>
3. <logic:present>
4. <logic:notPresent>
5. <logic:equal>
6. <logic:notEqual>
7. <logic:empty>
8. <logic:notEmpty>
9. <logic:messagesPresent>
10. <logic:greaterThan>

NOTE🡺IN ORDER TO USE THE STRUTS OR ANY USER DEFINED TAGS WE HAVE TO INCLUDE ITS .tld FILES by using the directive taglib

1. &nbsp🡺 ms 🡺is used to provide the space



1. BEAN TAGS 🡺
2. <bean:write>
3. <bean:message>
4. <bean:parameter>
5. <bean:define>
6. HTML TAGS🡺
7. <html:form>
8. <html:text>
9. <html:textarea>
10. <html:button>
11. <html:select>
12. <html:option>
13. <html:optionsCollection>🡺
14. <html:hidden>
15. <html:radio>

PENDING WORK WITH STRUTS 1.2 🡺

1. Needed to collect more Information🡺Struts tags
2. Needed to collect more Information🡺Tiles 🡺Syntax
3. DispatchAction
4. Error Handling and Validation