1. **Defintion of Inheritance?**

- Inheritance is a mechanism wherein a new class is derived from an existing class
- Inheritance is implemented using extends keyword in Java.When one Class extends another Class it inherit all non private members including fields and methods

2. **Definition of Abstract Class?**

- Abstract class is a class
- Used to sharing the common algorithm across multiple implementation.
- Abstract class can't be directly instantiated(object).
- Ex:Sample s= new Sample1();
- We can use concrete method (Declare and implemention)in abstract class
- If a class includes abstract methods, the class itself *must* be declared abstract.

3.Definition of Interface?

It is an contract or promise made by an object.

All methods declared in an interface are implicitly public

4.Types of Interface?

Marker interface

5.What is String class?

String Class is immutable, i.e. **Strings in java,** once created and initialized, cannot be changed on the same reference.

A **java.lang.String class** is final which implies no class and extend it

String s="ar";

s=s.concat("rahman");

s.o.p(s);

//it will print only→ar

6.Difference between string literal and new operator (while the string creation)

Each time your code creates a string literal, the JVM checks the string literal pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string does not exist in the pool, a new String object is created and placed in the pool. JVM keeps at most one object of any String in this pool. String literals always refer to an object in the string pool.

7.Difference between String Buffer and String Builder?

StringBuffer	StringBuilder
Synchronized	Not synchronized
Slower than StringBuilder	Faster bcoz of sync.
Mutable	Mutable
Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.	Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.

8.What is substring in String class?

public String substring(int startIndex)

This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

public String substring(int startIndex,int endIndex)

This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

Here, startIndex is inclusive while endIndex is exclusive.

Example:

```
String name="Hello World";

/* This will print the substring starting from index 6 */

System.out.println(name.substring(6));->World

/*  This will print the substring starting from index 0 upto 4 not 5.
IMPORTANT : Here startIndex is inclusive while endIndex is exclusive.  */

System.out.println(name.substring(0,5)); ->Hello
```

9.Index in String?

int indexOf(int ch)

int indexOf(int ch, int begin)

int indexOf(String ch)

int indexOf(String ch, int begin)

All of them preform the same overall action of returning an integer, which represents the **FIRST OCCURRANCE of a character or String** contained in that string.

int lastIndexOf(int ch)

int lastIndexOf(int ch, int begin)

int lastIndexOf(String ch)

int lastIndexOf(String ch, int begin)

10.String to charArray,byteArray and viceversa?

String to charArray:

```
String str = "Abcdefg";
char[] cArray = str.toCharArray();
```

String to bytearray:

```
String stringToConvert = "This String is 76";

byte[] theByteArray = stringToConvert.getBytes();
```

11.what is an Encapsulation? Why it is an java?

Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

Example: Pojo class

12.use of static keyword?

Static block:

Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

Static method:

- It is a method which belongs to the class and not to the object(instance)
- A static method can access only static data. It can not access non-static data (instance variables)
- A static method can call only other static methods and can not call a non-static method from it.
- A static method can be accessed directly by the class name and doesn't need any object
- Syntax : <class-name>.<method-name>
- A static method cannot refer to "this" or "super" keywords in anyway

Static variable:

- It is a variable which **belongs to the class** and **not** to **object**(instance)
- Static variables are **initialized only once** , at the start of the execution . These variables will be initialized first, before the initialization of any instance variables
- A **single copy** to be shared by all instances of the class
- A static variable can be **accessed directly** by the **class name** and doesn't need any object
- Syntax : <***class-name>.<variable-name>***

13.final keyword and where we can uses?

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

- ***final* Classes**- A final class cannot have subclasses.
- ***final* Variables**- A final variable cannot be changed once it is initialized.
- ***final* Methods**- A final method cannot be overridden by subclasses.

14.synchronized keyword uses and where we can uses?

synchronization is the capability to control the access of multiple threads to any shared resources.

Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread.

Types of synchronization:

Synchronizd method

Synchronizd block

Static Synchronizion

```
public void synchronized method(){}
public void synchronized staticmethod(){}
public void myMethod(){
```

```
synchronized (this){  // synchronized keyword on block of  code
    }

}
```

15.How to access super class methods in sub class?

```
class Animal
{
public void move()
{
System.out.println("Animals can move");
}

class Dog extends Animal
{
     public void move()
 {
super.move();//call super class methods
  System.out.println("Dogs can walk and run");
                }
}
```

16.what is overload and overriding?

Method overloading

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.

Note:

- *Overloaded methods MUST change the argument list*
- *Overloaded methods CAN change the return type*
- *Overloaded methods CAN change the access modifier*
- *Overloaded methods CAN declare new or broader checked exceptions*
- *A method can be overloaded in the same class or in a subclass*

Method overriding:

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that's specific to a particular subclass type.

Note:

- *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can't override a method marked public and make it protected).*
- *You cannot override a method marked final*
- *You cannot override a method marked static*

17.use of this and super keyword?

this refers to a reference of the **current** class.
super refers to the **parent** of the current class (which called the super keyword).

18.Difference between abstract class and interface?

Interface	Abstract class
interface is a interface	abstractclass is a class
can not create non abstract method , every method in interface is by default abstract	can create non abstract method in abstract class
SLOW	FASTER
interface are better suited for Type declaration	abstract class is more suited for code reuse
If a class includes abstract methods, the class itself *must* be declared abstract	
Methods only declare not implemention.We must give implem. For all methods in subclasses	Both we can declare or implementation abstract void addNames(); Or Public void addNames() { }
Interface-Interface =extends Class-class=extends Interface-class=implements	

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code at all,just the signature
In case of abstract class, a class may extend only one abstract class.	A Class may implement several interfaces.
An abstract class can have non-abstract methods.	All methods of an Interface are abstract.
An abstract class can have instance variables.	An Interface cannot have instance variables.
An abstract class can have any visibility: public, private, protected.	An Interface visibility must be public (or) none.
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.	If we add a new method to an Interface then we have to track down all the implementations of the interfa and define implementation for the new method.
An abstract class can contain constructors .	An Interface cannot contain constructors .
Abstract classes are fast.	Interfaces are slow as it requires extra indirection to find corresponding method in the actual class.

When should I use abstract classes and when should I use interfaces?

Use Interfaces when…

- You see that something in your design will change frequently.
- If various implementations only share method signatures then it is better to use Interfaces.
- you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

Use Abstract Class when…

- If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
- When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
- Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

19.Enhanced for loop?

It was introduced in java 5 mainly used to traverse array or collection elements.

Syntax:

For(datatype variable:collection or array)

20.static import?

Used to access any static member of a class directly.

import static java.lang.System.*;

class Test

{

out.println("HI");

}

21.Describe java collection framework?

collection framework provides an architecture to store and manipulate the group of objects.

It has inbuilt interfaces List,Set,Map,Queue

22.List,set methods?

List Methods:

void	**add**(int index, Object element) Inserts the specified element at the specified position in this list (optional operation).
boolean	**add**(Object o) Appends the specified element to the end of this list (optional operation).
boolean	**addAll**(Collection c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	**addAll**(int index, Collection c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	**clear**() Removes all of the elements from this list (optional operation).
boolean	**contains**(Object o) Returns true if this list contains the specified element.

boolean	**containsAll**(Collection c) Returns true if this list contains all of the elements of the specified collection.
boolean	**equals**(Object o) Compares the specified object with this list for equality.
Object	**get**(int index) Returns the element at the specified position in this list.
int	**hashCode**() Returns the hash code value for this list.
int	**indexOf**(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(Object o) Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
ListIterator	**listIterator**() Returns a list iterator of the elements in this list (in proper sequence).
ListIterator	**listIterator**(int index) Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.
Object	**remove**(int index) Removes the element at the specified position in this list (optional operation).
boolean	**remove**(Object o) Removes the first occurrence in this list of the specified element (optional operation).
boolean	**removeAll**(Collection c) Removes from this list all the elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(Collection c) Retains only the elements in this list that are contained in the specified collection (optional operation).
Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	**size**() Returns the number of elements in this list.

List	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.

Set Methods:

Boolean	**add** (Object o) Ensures that this collection contains the specified element (optional operation).
Boolean	**addAll**(Collection c) Adds all of the elements in the specified collection to this collection (optional operation).
Void	**clear**() Removes all of the elements from this collection (optional operation).
Boolean	**contains**(Object o) Returns true if this collection contains the specified element.
Boolean	**containsAll**(Collection c) Returns true if this collection contains all of the elements in the specified collection.
Boolean	**equals**(Object o) Compares the specified object with this collection for equality.
Int	**hashCode**() Returns the hash code value for this collection.
Boolean	**isEmpty**() Returns true if this collection contains no elements.
Iterator	**iterator**() Returns an iterator over the elements in this collection.
Boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
Boolean	**removeAll**(Collection c) Removes all this collection's elements that are also contained in the specified collection (optional operation).

Boolean	**retainAll**(Collection c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
Int	**size**() Returns the number of elements in this collection.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

23.what is an iterator?

- The Iterator interface is used to step through the elements of a Collection.
- Iterators let you process each element of a Collection.
- Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
- Iterator is an Interface ,iterating elements only **forward direction** only
- Obtain an iterator to the start of the collection by calling the collectionâ€™s ***iterator()*** method.
- Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns **true**.
- Within the loop, obtain each element by calling **next()**.

Example:

```
Iterator itr=al.iterator();

While(itr.hasNext())

        {

        Sop(itr.next);}
```

24.what is ListIterator?

ListIterator is available **only for Lists** which is a special type of collection.

ListIterator is just like Iterator, except it allows us to access the collection in either the forward or backward direction .

ListIterator allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. A ListIterator has no current element; its cursor position always lies

between the element that would be returned by a call to previous() and the element that would be returned by a call to next().

25.Difference between ListIterator and iterator?

Iterator	ListIterator
• Using iterator you can visit the elements of collection in the forward direction only	In list iterator you can visit the elements of collection in the forward and backward direction
Using iterator you cannot add elements to collection	• Using list Iterator you can add elements to collection.
• Using iterator you can not replace existing object with new object.	Using iterator you can replace existing object with new object.
Using iterator you can not get index of elements	Using list Iterator you get index of elements list

26.Difference between list and set?

- Set – Stored elements in unordered or shuffles way, and does not allow duplicate values.
- List – Stored elements in ordered way, and allow duplicate values.

27.Implements of set ? what is the differences?

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.

1) HashSet

- A HashSet is an unsorted, unordered Set.
- It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you'll get).
- Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.

2) TreeSet

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

3) LinkedHashSet

It orders its elements on the basis of order in which they were inserted in the set.

28.difference between LinkedList and List?

29.Map methods?

Void	**clear**() Removes all mappings from this map (optional operation).
Boolean	**containsKey**(Object key) Returns true if this map contains a mapping for the specified key.
Boolean	**containsValue**(Object value) Returns true if this map maps one or more keys to the specified value.
Set	**entrySet**() Returns a set view of the mappings contained in this map.
Boolean	**equals**(Object o) Compares the specified object with this map for equality.
Object	**get**(Object key) Returns the value to which this map maps the specified key.
Int	**hashCode**() Returns the hash code value for this map.
Boolean	**isEmpty**() Returns true if this map contains no key-value mappings.

Set	**keySet**() Returns a set view of the keys contained in this map.
Object	**put**(Object key, Object value) Associates the specified value with the specified key in this map (optional operation).
Void	**putAll**(Map t) Copies all of the mappings from the specified map to this map (optional operation).
Object	**remove**(Object key) Removes the mapping for this key from this map if it is present (optional operation).
Int	**size**() Returns the number of key-value mappings in this map.
Collection	**values**() Returns a collection view of the values contained in this map.

30. Is Map collection?

The Map interface is not an extension of Collection interface. Instead the interface starts of it's own interface hierarchy, for maintaining key-value associations. The interface describes a mapping from keys to values, without duplicate keys, by definition.

31.what is retainAll() map?

Retains only the elements in this list that are contained in the specified collection

32.How to find empty in map,list and set?

List and Set ,Map- isEmpty()

33.what is index(),capacity()?

34.use of retainAll()?

35.convert list to array ,array to list?

List to array convertion:

```
ArrayList<String> al = new ArrayList<String>();
```

```
String[] convertArray=new String[al.size()];

al.toArray(convertArray);

for(String mine:convertArray)

{

System.out.println(mine);

}
```

Array to List convertion:

```
List al=Arrays.asList(convertArray);
```

36.how to sort objects based on the specific order?

37.How to implement comparable and comparator?

38.How to equals and override method?

39.Difference Hashmap and Hashtable?

HashMap	**Hashtable**
HashMap lets you have null values as well as one null key.	HashTable does not allows null values as key and value.
The iterator in the HashMap is fail-safe (If you change the map while iterating, you'll know).	The enumerator for the Hashtable is not fail-safe
HashMap is unsynchronized.	Hashtable is synchronized.

Note: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

Note: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

41.Load properties file in java?

```
Properties prop = new Properties();
        try {
            //load a properties file
                prop.load(new FileInputStream("config.properties"));
            //get the property value and print it out
             System.out.println(prop.getProperty("database"));
                System.out.println(prop.getProperty("dbuser"));
                System.out.println(prop.getProperty("dbpassword"));

         } catch (IOException ex) {
                ex.printStackTrace();
       }
```

Difference between synchronized collection and concurrent collection?

42.Purpose of synchronized collection?

43.serizable interface

44.how many obj create serizable in singleton class?

45.static variable,if static variable serizable what will happened?

46.return types in try , finally ,and outer so where it will reach?

47.how to connect jdbc?

48.statement ,preparedStatement,callablestatement

49.if connection is not closed what will happened in jdbc?

50.boxing and unboxing string to int, and int to string

51.what is servlet

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

52.what are all the methods in servlet?

GenericServlet

HTTPServlet

53.difference between get and post?

#	doGet()	doPost()
1	In doGet() the parameters are appended to the URL and sent along with header information.	In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar.
2	The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.	You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects!
3	doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent)	doPost() provides information (such as placing an order for merchandise) that the server is expected to remember
4	Parameters are not encrypted	Parameters are encrypted
5	doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance	doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length
6	doGet() should be idempotent. i.e. doget should be able to be repeated safely many times	This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable.

7	doGet() should be safe without any side effects for which user is held responsible	This method does not need to be either safe
8	It allows bookmarks.	It disallows bookmarks

54. Difference between printwriter and ServletOutputStream ?

ServletOutputStream: ServletResponse.getOutputStream() returns a ServletOutputStream
suitable for writing binary data in the response. The servlet container does not encode the binary data, it sends the raw data as it is.

PrintWriter: ServletResponse.getWriter() returns PrintWriter object which sends character text to the client. The PrintWriter uses the character encoding returned by getCharacterEncoding(). If the response's character encoding has not been specified then it does default character encoding.

55.How to send binary data in response?

56.servlet life cycle?

- **Servlet class loading** : For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.
- **Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.
- **Initialization (call the init method)** : After instantiation, the container initializes a servlet before it is ready to handle client requests. The container initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.
- **Request handling (call the service method)** : After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a

different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.

- **Removal from service (call the destroy method)** : A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

57.What is servlet initparam?

```
<servlet>
        <servlet-name>ServletName</servlet-name>
        <servletclass>com.mkyong.ServletDemo</servletclas>
            <init-param>
                <param-name>email</param-name>
                <param-value>email.com</paramvalue>
            </init-param>
        </servlet>
        <servlet-mapping>
            <servlet-name>ServletName</servlet-name>
            <url-pattern>/Demo</url-pattern>
        </servlet-mapping>
```

```
getServletConfig().getInitParameter("email"));
```

58.what is contextparam?

```
<web-app>

  <context-param>
     <param-name> n1 </param-name>
     <param-value> 100 </param-value>
  </context-param>
```

```
ServletConfig conf = getServletConfig();
ServletContext context = conf.getServletContext();
```

59.web.xml?

A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.
The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

```
<web-app>

  <servlet>

      <servlet-name>onServletContext</servlet-name>
      <servlet-class>java4s.OnServletContext</servlet-class>
   </servlet>

   <servlet-mapping>
         <servlet-name>onServletContext</servlet-name>
         <url-pattern>/onContext</url-pattern>
   </servlet-mapping>
<welcome-file-list>
      <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

60.difference between servlet and jsp?

61.jsp lifecycle?

When a request is mapped to a JSP page for the first time, it translates the JSP page into a servlet class and compiles the class. It is this servlet that services the client requests.
A JSP page has seven phases in its lifecycle, as listed below in the sequence of occurrence:

- Translation
- Compilation
- Loading the class
- Instantiating the class
- jspInit() invocation

- _jspService() invocation
- jspDestroy() invocation

1. JSP Page Translation:

A java servlet file is generated from the JSP source file. This is the first step in its tedious multiple phase life cycle. In the translation phase, the container validates the syntactic correctness of the JSP pages and tag files. The container interprets the standard directives and actions, and the custom actions referencing tag libraries used in the page.

2. JSP Page Compilation:

The generated java servlet file is compiled into a java servlet class.

Note: The translation of a JSP source page into its implementation class can happen at any time between initial deployment of the JSP page into the JSP container and the receipt and processing of a client request for the target JSP page.

3. Class Loading:

The java servlet class that was compiled from the JSP source is loaded into the container.

4. Execution phase:

In the execution phase the container manages one or more instances of this class in response to requests and other events.
The interface JspPage contains jspInit() and jspDestroy(). The JSP specification has provided a special interface HttpJspPage for JSP pages serving HTTP requests and this interface contains _jspService().

5. Initialization:

The jspInit() method of the javax.servlet.jsp.JspPage interface is similar to the init() method of servlets. This method is invoked by the container only once when a JSP page is initialized. It can be overridden by a page author to initialize resources such as database and network connections, and to allow a JSP page to read persistent configuration data.

6. _jspService() execution:

The _jspService() method of the javax.servlet.jsp.HttpJspPage interface is invoked every time a new request comes to a JSP page. This method takes the HttpServletRequest and HttpServletResponse objects as its arguments. A page author cannot override this method, as its implementation is provided by the container.

7. jspDestroy() execution:

The jspDestroy() method of the javax.servlet.jsp.JspPage interface is invoked by the container when a JSP page is about to be destroyed. This method is similar to the destroy() method of servlets. It can be overridden by a page author to perform any cleanup operation such as closing a database connection.

jspInit(), _jspService() and jspDestroy() are called the life cycle methods of the JSP.

62.scriptlet tags in jsp?

scriptlet tag allow you write java code inside JSP Page .

Scriptlets are executed at request time, when the JSP engine processes the client request. If the scriptlet produces output, the output is stored in the out object, from which you can display it.

<% java code%>

<%int count=0;%>

63.difference between sendRedirect and RequestDispatcher?

RequestDispatcher	**sendRedirect**
A forward is performed internally by the servlet.	A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original.
The browser is completely unaware that it has taken place, so its original URL remains intact.	The browser, in this case, is doing the work and knows that it's making a new request.
Any browser reload of the resulting page will simple repeat the original request, with the original URL	A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL.
Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable)	This method can be used to redirect users to resources that are not part of the current context, or even in the same domain.
Since both resources are part of same	Because this involves a new request,

context, the original request context is retained	the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object).
Forward is marginally faster than redirect.	redirect is marginally slower than a forward, since it requires two browser requests, not one.

64.what is EL?How to use in jsp?

65.jstl?

66.use of c:out. c:foreach

67.web server and appserver difference

Application server is used to serve web based applications and enterprise based applications(i.e sevlets, jsps and ejbs...). because application server contains web server internally.ex weblogic websphereA server that exposes business logic to client applications through various protocols including HTTP.

Web server is used to serve web based applications.(i.e servlets and jsps)ex.tomcat server. A server that handles HTTP protocol.

68.Http error codes?

100 - Continue.
101 - Switching protocols.

2xx - Success

This class of status codes indicates that the server successfully accepted the client request.

200 - OK. The client request has succeeded.
201 - Created.
202 - Accepted.
203 - Non-authoritative information.
204 - No content.
205 - Reset content.
206 - Partial content.
207 - Multi-Status (WebDay).

3xx - Redirection

The client browser must take more action to fulfill the request. For example, the browser may have to request a different page on the server or repeat the request by using a proxy server.

301 - Moved Permanently
302 - Object moved.
304 - Not modified.
307 - Temporary redirect.

4xx - Client Error

An error occurs, and the client appears to be at fault. For example, the client may request a page that does not exist, or the client may not provide valid authentication information.

* 400 - Bad request.
* **401 - Access denied. IIS defines several different 401 errors that indicate a more specific cause of the error. These specific error codes are displayed in the browser but are not displayed in the IIS log:**
o 401.1 - Logon failed.
o 401.2 - Logon failed due to server configuration.
o 401.3 - Unauthorized due to ACL on resource.
o 401.4 - Authorization failed by filter.
o 401.5 - Authorization failed by ISAPI/CGI application.
o 401.7 – Access denied by URL authorization policy on the Web server. This error code is specific to IIS 6.0.

*** 403 - Forbidden. IIS defines several different 403 errors that indicate a more specific cause of the error:**

o 403.1 - Execute access forbidden.
o 403.2 - Read access forbidden.
o 403.3 - Write access forbidden.
o 403.4 - SSL required.
o 403.5 - SSL 128 required.
o 403.6 - IP address rejected.
o 403.7 - Client certificate required.
o 403.8 - Site access denied.
o 403.9 - Too many users.
o 403.10 - Invalid configuration.

o 403.11 - Password change.
o 403.12 - Mapper denied access.
o 403.13 - Client certificate revoked.
o 403.14 - Directory listing denied.
o 403.15 - Client Access Licenses exceeded.
o 403.16 - Client certificate is untrusted or invalid.
o 403.17 - Client certificate has expired or is not yet valid.
o 403.18 - Cannot execute requested URL in the current application pool. This error code is specific to IIS 6.0.
o 403.19 - Cannot execute CGIs for the client in this application pool. This error code is specific to IIS 6.0.
o 403.20 - Passport logon failed. This error code is specific to IIS 6.0.

*** 404 - Not found.**

o 404.0 - (None) – File or directory not found.
o 404.1 - Web site not accessible on the requested port.
o 404.2 - Web service extension lockdown policy prevents this request.
o 404.3 - MIME map policy prevents this request.
* 405 - HTTP verb used to access this page is not allowed (method not allowed.)
* 406 - Client browser does not accept the MIME type of the requested page.
* 407 - Proxy authentication required.
* 412 - Precondition failed.
* 413 – Request entity too large.
* 414 - Request-URI too long.
* 415 – Unsupported media type.
* 416 – Requested range not satisfiable.
* 417 – Execution failed.
* 423 – Locked error.

5xx - Server Error

The server cannot complete the request because it encounters an error.

*** 500 - Internal server error.**

o 500.12 - Application is busy restarting on the Web server.
o 500.13 - Web server is too busy.
o 500.15 - Direct requests for Global.asa are not allowed.
o 500.16 – UNC authorization credentials incorrect. This error code is specific to IIS 6.0.
o 500.18 – URL authorization store cannot be opened. This error code is specific to

IIS 6.0.
o 500.19 - Data for this file is configured improperly in the metabase.
o 500.100 - Internal ASP error.

* 501 - Header values specify a configuration that is not implemented.
* 502 - Web server received an invalid response while acting as a gateway or proxy.
o 502.1 - CGI application timeout.
o 502.2 - Error in CGI application.
* 503 - Service unavailable. This error code is specific to IIS 6.0.
* 504 - Gateway timeout.
* 505 - HTTP version not supported.

69.difference and use of c:out, and other EL?

70.jsp action

Standard Actions are actions that are defined in JSP itself, it need not be declared with a "taglib" directive. All the Standard Actions in JSP has a default prefix "jsp".

The following is the list of Standard Actions in JSP.

Action Element	Description
<jsp:useBean>	Enables JavaBeans components in that page
<jsp:getProperty>	Get a property value from JavaBeans component and adds to the response
<jsp:setProperty>	Sets the JavaBeans property value
<jsp:include>	Includes the response from the jsp page while processing.
<jsp:forward>	Forwards the processing of a request to ajsp page
<jsp:param>	Adds a parameter value to a request handed off using the <jsp:include or <jsp:forward>
<jsp:plugin>	Generates a HTML elements appropriate to a browser to execute an applet using Java Plugin.
<jsp:attribute>	Sets the value of the action element based on the body of this element.
<jsp:body>	Sets action element body based on the body of this element.

<jsp:element>	Dynamically generates XML element.
<jsp:text>	Encapsulates template text, needed in JSP pages written in XML.

71.if method is not declared by default ?get or post

72.jsp:param

73.jsp:include and include tag difference

Include directive in JSP is used to import a static file or dynamic file e.g. JSP inside another JSP. Most common example of include directive is including header and footer in JSP. Include directive is also called **file include** because resource to be included is specified using file attribute as shown in below example of include directive:

<%@ include file="loan.jsp" %>

The code written in loan.jsp will be included as it is in the jsp file which included it during JSP translation time and than merged code is get compiled to generate Servlet which is later used to server request.

Include action in JSP is another way of including a jsp file inside another jsp file. Included resource is specified using page attribute as shown in below example :

<jsp:include page="loan.jsp" %>

here contents of loan.jsp will be included during request time instead of translation time and any change in loan.jsp will reflect immediately. Due to this nature include action is also called dynamic include in JSP. It also referred as **page include** because of **page attribute** used to specify included resource.

74.scriptlet <%@import>

75.iterate and display in ui using jsp?

76.use of jsp:param

77.what are the type of attributes ,scopes and listener

78.how to add listener in web app

```
<web-app ...>
        <listener>
<listener-class>com.mkyong.AppServletContextListener</listener-class>
        </listener>
</web-app>
public class AppServletContextListener implements ServletContextListener{
        @Override
        public void contextDestroyed(ServletContextEvent arg0) {
                System.out.println("ServletContextListener destroyed");
        }
        @Override
        public void contextInitialized(ServletContextEvent arg0) {
                System.out.println("ServletContextListener started");
        }
}
```

79.how to create session? .how will you use?

Cookies,Hidden Form Field,URL Rewriting,HttpSession

80.jsp page scope?

81.cookies and how to expire

82.get() and find() in jsp

83.what is filter?

A filter is a reusable piece of code that can transform the content of HTTP requests, responses, and header information. Filters do not generally create a response or respond to a request as servlets do, rather they modify or adapt the requests for a resource, and modify or adapt responses from a resource.

84.How to write filter

WAY 1:

We can create a filter by writing a new class which implements javax.servlet.Filter. We need to specify the filters in the deployment descriptor so that the container can create a filter chain to process the request/response.

The filters in deployment descriptor:

```
<web-app>
	<filter>
		<filter-name>MyFilter</filter-name>
		<filter-class>subin.rnd.enterprise.filter.MyFilter</filter-class>
	</filter>
	<filter-mapping>
		<filter-name>MyFilter</filter-name>
		<url-pattern>/*</url-pattern>
	</filter-mapping>
	<filter-mapping>
		<filter-name>MyFilter</filter-name>
		<servlet-name>MyServlet</servlet-name>
		<dispatcher>REQUEST</dispatcher>
	</filter-mapping>
</web-app>
```

We can have a filter (or a chain of filters) processing the request/response. In the above descriptor, we are linking a filter for a URL pattern as well as for a specific servlet. So in the above case, for MyServlet, the filter will get executed twice as we have mapped the filter MyFilter to all URLs and for

MyServlet. So the order of execution will be MyFilter.doFilter() > MyFilter.doFilter() > MyServlet.doGet().

WAY 2:

1. Implement `javax.servlet.Filter`.
2. In `doFilter()` method, cast the incoming `ServletRequest` to `HttpServletRequest`.
3. Use `HttpServletRequest#getRequestURI()` to grab the path.
4. Use straightforward `java.lang.String` methods like `substring()`, `split()`, `concat()` and so on to extract the part of interest and compose the new path.
5. Use either `ServletRequest#getRequestDispatcher()` and then`RequestDispatcher#forward()` to forward the request/response to the new URL (server-side redirect, not reflected in browser address bar), **or** cast the incoming `ServletResponse` to`HttpServletResponse` and then `HttpServletResponse#sendRedirect()` to redirect the response to the new URL (client side redirect, reflected in browser address bar).
6. Register the filter in `web.xml` on an `url-pattern` of `/*` or `/Check_License/*`, depending on the context path, or if you're on Servlet 3.0 already, use the `@WebFilter` annotation for that instead.

Don't forget to add a check in the code if the URL *needs* to be changed and if *not*, then just call`FilterChain#doFilter()`, else it will call itself in an infinite loop.