OOPS Interview Questions

1.What are the principle concepts of OOPS?

There are four principle concepts upon which object oriented design and programming rest. They are:

- Abstraction
- Polymorphism
- Inheritance
- Encapsulation

  (i.e. easily remembered as A-PIE).

2.What is Abstraction?

Abstraction refers to the act of representing essential features without including the background details or explanations.

3.What is Encapsulation?

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.

4.What is the difference between abstraction and encapsulation?

- **Abstraction** focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it's inside view, where the behavior of the abstraction is implemented.
- **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
- **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

5.What is Inheritance?

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- A class that is inherited is called a superclass.
- The class that does the inheriting is called a subclass.
- Inheritance is done by using the keyword extends.
- The two most common reasons to use inheritance are:
  - To promote code reuse
  - To use polymorphism

6.What is Polymorphism?

Polymorphism is briefly described as "one interface, many implementations." Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

7.How does Java implement polymorphism?

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java). Polymorphism manifests itself in Java in the form of multiple methods having the same name.

- In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
- In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

8.Explain the different forms of Polymorphism.

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is done using inheritance and interface.
Note: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

- *Method overloading*
- *Method overriding through inheritance*
- *Method overriding through the Java interface*

9.What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

10.What is Dynamic Binding?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

11.What is 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*

12.What is 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*

13.What are the differences between method overloading and method overriding?

	Overloaded Method	**Overridden Method**
Arguments	Must change	Must not change
Return type	Can change	Can't change except for covariant returns
Exceptions	Can change	Can reduce or eliminate. Must not throw new or broader checked exceptions
Access	Can change	Must not make more restrictive (can be less restrictive)
Invocation	Reference type determines which overloaded version is selected. Happens at compile time.	Object type determines which method is selected. Happens at runtime.

14.Can overloaded methods be override too?

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

15.Is it possible to override the main method?

NO, because main is a static method. A static method can't be overridden in Java.

16.How to invoke a superclass version of an Overridden method?

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

```
// From subclass
     super.overriddenMethod();
```

17.What is super?

`super` is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.
Note:

- *You can only go back one level.*
- *In the constructor, if you use super(), it must be the very first code, and you cannot access any `this.xxx` variables or methods to compute its parameters.*

18.How do you prevent a method from being overridden?

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

```
      public final void exampleMethod() {
//   Method statements
}
```

19.What is an Interface?

An interface is a description of a set of methods that conforming implementing classes must have.

<u>Note</u>:

- *You can't mark an interface as final.*
- *Interface variables must be static.*
- *An Interface cannot extend anything but another interfaces.*

20.Can we instantiate an interface?

You can't instantiate an interface directly, but you can instantiate a class that implements an interface.

21.Can we create an object for an interface?

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

22.Do interfaces have member variables?

Interfaces may have member variables, but these are implicitly `public, static,` and `final`- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

23.What modifiers are allowed for methods in an Interface?

Only `public` and `abstract` modifiers are allowed for methods in interfaces.

24.What is a marker interface?

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. The `java.io.Serializable` interface and `Cloneable` are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

25.What is an abstract class?

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.
<u>Note</u>:

- *If even a single method is abstract, the whole class must be declared abstract.*
- *Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.*
- *You can't mark a class as both abstract and final.*

26.Can we instantiate an abstract class?

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

27.What are the differences between Interface and Abstract class?

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 interface 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.

28.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.

29.When you declare a method as abstract, can other nonabstract methods access it?

Yes, other nonabstract methods can access a method that you declare as abstract.

30.Can there be an abstract class with no abstract methods in it?

Yes, there can be an abstract class without abstract methods.

31.What is Constructor?

- A constructor is a special method whose task is to initialize the object of its class.
- It is special because its name is the **same as the class name**.
- They do not have return types, not even **void** and therefore they cannot return values.
- They **cannot be inherited**, though a derived class can call the base class constructor.
- Constructor is invoked whenever an object of its associated class is created.

32.How does the Java default constructor be provided?

If a class defined by the code does **not** have any constructor, compiler will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

33.Can constructor be inherited?

No, constructor cannot be inherited, though a derived class can call the base class constructor.

34.What are the differences between Contructors and Methods?

	Constructors	**Methods**
Purpose	Create an instance of a class	Group Java statements
Modifiers	Cannot be *abstract, final, native, static*, or *synchronized*	Can be *abstract, final, native, static*, or *synchronized*
Return Type	No return type, not even void	void or a valid return type
Name	Same name as the class (first letter is	Any name except the class. Method

	capitalized by convention) -- usually a noun	names begin with a lowercase letter by convention -- usually the name of an action
this	Refers to another constructor in the same class. If used, it must be the first line of the constructor	Refers to an instance of the owning class. Cannot be used by static methods.
super	Calls the constructor of the parent class. If used, must be the first line of the constructor	Calls an overridden method in the parent class
Inheritance	Constructors are not inherited	Methods are inherited

35.How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses *super*, it must use it in the first line; otherwise, the compiler will complain.

36.What are the differences between Class Methods and Instance Methods?

Class Methods	Instance Methods
Class methods are methods which are declared as static. The method can be called without creating an instance of the class	Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the new keyword. Instance methods operate on specific instances of classes.
Class methods can only operate on class members and not on instance members as class methods are unaware of instance members.	Instance methods of the class can also not be called from within a class method unless they are being called on an instance of that class.
Class methods are methods which are declared as static. The method can be called without creating an instance of the class.	Instance methods are not declared as static.

37.How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

38.What are Access Specifiers?

One of the techniques in object-oriented programming is *encapsulation*. It concerns the hiding of data in a class and making this class available only through methods. Java allows you to control access to classes, methods, and fields via so-called *access specifiers*..

39.What are Access Specifiers available in Java?

Java offers four access specifiers, listed below in decreasing accessibility:

- **Public**- *public* classes, methods, and fields can be accessed from everywhere.
- **Protected**- *protected* methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
- **Default(no specifier)-** If you do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
- **Private**- *private* methods and fields can only be accessed within the same class to which the methods and fields belong. *private* methods and fields are not visible within subclasses and are not inherited by subclasses.

Situation	**`public`**	**`protected`**	**default**	**`private`**
Accessible to class from same package?	yes	yes	yes	no
Accessible to class from different package?	yes	no, *unless it is a subclass*	no	no

40.What is final modifier?

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.

41.What are the uses of final method?

There are two reasons for marking a method as final:

- Disallowing subclasses to change the meaning of the method.

- Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

42.What is 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.

43.What are static variables?

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. They are declared by using the static keyword as a modifier.

```
static type  varIdentifier;
```

where, the name of the variable is varIdentifier and its data type is specified by type.
Note: Static variables that are not explicitly initialized in the code are automatically initialized with a default value. The default value depends on the data type of the variables.

44.What is the difference between static and non-static variables?

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

45.What are static methods?

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.
Note:The use of a static method suffers from the following restrictions:

- *A static method can only call other static methods.*
- *A static method must only access static data.*
- *A static method* ***cannot*** *reference to the current object using keywords super or this.*

Java Collections Interview Questions

46.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 implemented a different way for every Collection.

47.How do you traverse through a collection using its Iterator?

To use an iterator to traverse through the contents of a collection, follow these steps:

- 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()**.

48.How do you remove elements during Iteration?

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

49.What is the difference between Enumeration and Iterator?

Enumeration	Iterator
Enumeration doesn't have a remove() method	Iterator has a remove() method
Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects	Can be *abstract, final, native, static*, or *synchronized*

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

50.How is ListIterator?

ListIterator is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us modify an element

51.What is the List interface?

- The List interface provides support for ordered collections of objects.
- Lists may contain duplicate elements.

52.What are the main implementations of the List interface ?

The main implementations of the List interface are as follows :

- **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
- **Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
- **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

53.What are the advantages of ArrayList over arrays ?

Some of the advantages ArrayList has over arrays are:

- It can grow dynamically
- It provides more powerful insertion and search mechanisms than arrays.

54.Difference between ArrayList and Vector ?

ArrayList	Vector
ArrayList is **NOT** synchronized by default.	Vector List is synchronized by default.
ArrayList can use only Iterator to access the elements.	Vector list can use Iterator and Enumeration Interface to access the elements.
The ArrayList increases its array size by 50 percent if it runs out of room.	A Vector defaults to doubling the size of its array if it runs out of room
ArrayList has no default size.	While vector has a default size of 10.

55.How to obtain Array from an ArrayList ?

Array can be obtained from an ArrayList using ***toArray()*** method on ArrayList.

```
List arrayList = new ArrayList();
arrayList.add(â€¦

ObjectÂ  a[] = arrayList.toArray();
```

56.Why insertion and deletion in ArrayList is slow compared to LinkedList ?

- **ArrayList** internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
- During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

57.Why are Iterators returned by ArrayList called Fail Fast ?

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

58.How do you decide when to use ArrayList and When to use LinkedList?

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

59.What is the Set interface ?

- The Set interface provides methods for accessing the elements of a finite mathematical set
- Sets do not allow duplicate elements
- Contains no methods other than those inherited from Collection
- It adds the restriction that duplicate elements are prohibited
- Two Set objects are equal if they contain the same elements

60.What are the main Implementations of the Set interface ?

The main implementations of the List interface are as follows:

- HashSet
- TreeSet
- LinkedHashSet
- EnumSet

61.What is a 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.

62.What is a 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.

63.What is an EnumSet ?

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

64.Difference between HashSet and TreeSet ?

HashSet	TreeSet
HashSet is under set interface i.e. it does not guarantee for either sorted order or sequence order.	TreeSet is under set i.e. it provides elements in a sorted order (acceding order).
We can add any type of elements to hash set.	We can add only similar types of elements to tree set.

65.What is a Map ?

- A map is an object that stores associations between keys and values (key/value pairs).
- Given a key, you can find its value. Both keys and values are objects.
- The keys must be unique, but the values may be duplicated.
- Some maps can accept a null key and null values, others cannot.

66.What are the main Implementations of the Map interface ?

The main implementations of the List interface are as follows:

- HashMap
- HashTable
- TreeMap
- EnumMap

67.What is a TreeMap ?

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

68.How do you decide when to use HashMap and when to use TreeMap ?

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

69.Difference between 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.

70.How does a Hashtable internally maintain the key-value pairs?

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

71.What Are the different Collection Views That Maps Provide?

Maps Provide Three Collection Views.

- **Key Set** - allow a map's contents to be viewed as a set of keys.
- **Values Collection** - allow a map's contents to be viewed as a set of values.
- **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

72.What is a KeySet View ?

KeySet is a set returned by the ***keySet()*** method of the Map Interface, It is a set that contains all the keys present in the Map.

73.What is a Values Collection View ?

Values Collection View is a collection returned by the ***values()*** method of the Map Interface, It contains all the objects present as values in the map.

74.What is an EntrySet View ?

Entry Set view is a set that is returned by the ***entrySet()*** method in the map and contains Objects of type Map. Entry each of which has both Key and Value.

75.How do you sort an ArrayList (or any list) of user-defined objects ?

Create an implementation of the *java.lang.Comparable* interface that knows how to order your objects and pass it to *java.util.Collections.sort*(List, Comparator).

76.What is the Comparable interface ?

The Comparable interface is used to sort collections and arrays of objects using the *`Collections.sort()`* and *`java.utils.Arrays.sort()`* methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

```
        interface Comparable<T>
```

where T is the name of the type parameter.

All classes implementing the Comparable interface must implement the `compareTo()` method that has the return type as an integer. The signature of the `compareTo()` method is as follows:

```
      int i = object1.compareTo(object2)
```

- If object1 < object2: The value of i returned will be negative.
- If object1 > object2: The value of i returned will be positive.
- If object1 = object2: The value of i returned will be zero.

77.What are the differences between the Comparable and Comparator interfaces ?

Comparable	**Comparator**
It uses the *compareTo()* method. *int objectOne.compareTo(objectTwo).*	it uses the *compare()* method. *int compare(ObjOne, ObjTwo)*
It is necessary to modify the class whose instance is going to be sorted.	A separate class can be created in order to sort the instances.
Only one sort sequence can be created.	Many sort sequences can be created.
It is frequently used by the API classes.	It used by third-party classes to sort instances.

Difference between HashMap and HashTable? Can we make hashmap synchronized?

1. The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
2. HashMap does not guarantee that the order of the map will remain constant over time.
3. HashMap is non synchronized whereas Hashtable is synchronized.
4. Iterator in the HashMap is fail-safe while the enumerator for the Hashtable isn't.

Note on Some Important Terms

1)Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
2)Fail-safe is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn’t modify the collection "structurally”. However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown. HashMap can be synchronized by
Map m = Collections.synchronizeMap(hashMap);

What is the difference between set and list?

A Set stores elements in an unordered way and does not contain duplicate elements, whereas a list stores elements in an ordered way but may contain duplicate elements.

Servlets Questions

1.What is the 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.

2.What are the new features added to Servlet 2.5?

Following are the changes introduced in Servlet 2.5:

- A new dependency on J2SE 5.0
- Support for annotations
- Loading the class
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

Learn more about Servlets 2.5 features

3.What are the uses of Servlet?

Typical uses for HTTP Servlets include:

- Processing and/or storing data submitted by an HTML form.
- Providing dynamic content, e.g. returning the results of a database query to the client.
- A Servlet can handle multiple request concurrently and be used to develop high performance system
- Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

4.What are the advantages of Servlet over CGI?

Servlets have several advantages over CGI:

- A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
- A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

5.What are the phases of the servlet life cycle?

The life cycle of a servlet consists of the following phases:

- **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.

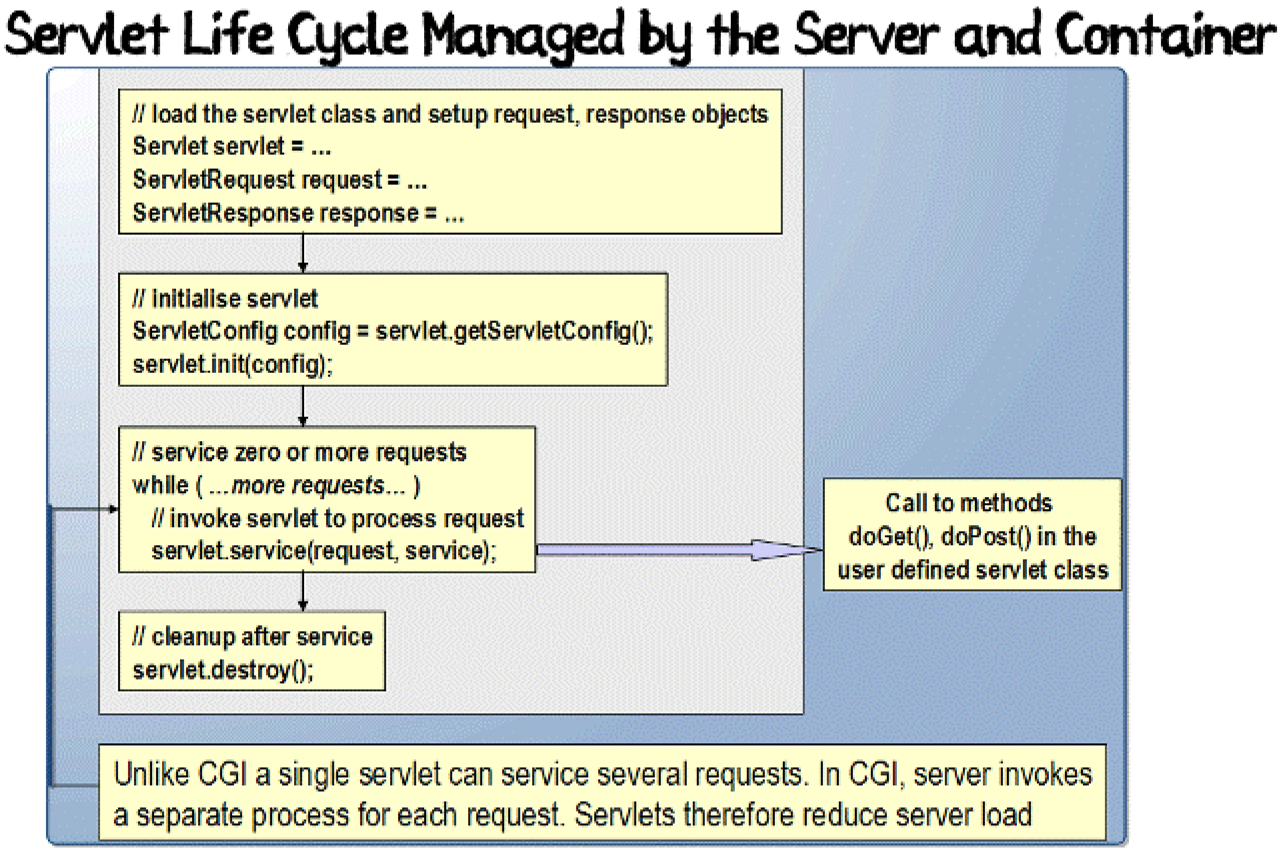
- **Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.

- **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

  The life cycle of a servlet is controlled by the container in which the servlet has been deployed.


Servlet Life Cycle Managed by the Server and Container
// load the servlet class and setup request, response objects
Servlet servlet = ...
ServletRequest request = ...
ServletResponse response = ...
// initialise servlet
ServletConfig config = servlet.getServletConfig();
servlet.init(config);
// service zero or more requests
while (...more requests...)
// invoke servlet to process request
servlet.service(request, service);
// cleanup after service
servlet.destroy();
Call to methods doGet(), doPost() in the user defined servlet class
Unlike CGI a single servlet can service several requests. In CGI, server invokes a separate process for each request. Servlets therefore reduce server load

6.Why do we need a constructor in a servlet if we use the init method?

Even though there is an init method in a servlet which gets called to initialize it, a constructor is still required to instantiate the servlet. Even though you as the developer would never need to explicitly call the servlet's constructor, it is still being used by the container (the container still uses the constructor to create an instance of the servlet). Just like a normal POJO (plain old java object) that might have an init method, it is no use calling the init method if you haven't constructed an object to call it on yet.

7.How the servlet is loaded?

A servlet can be loaded when:

- First request is made.
- Server starts up (auto-load).
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
- Administrator manually loads.

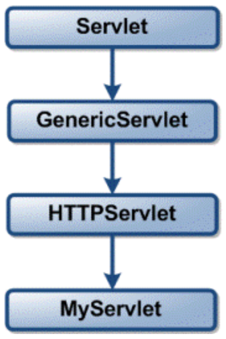
8.How a Servlet is unloaded?

A servlet is unloaded when:

- Server shuts down.
- Administrator manually unloads.

9.What is Servlet interface?

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or , more commonly by extending a class that implements it.



Note: Most Servlets, however, extend one of the standard implementations of that interface, namely `javax.servlet.GenericServlet` and `javax.servlet.http.HttpServlet`.

10.What is the GenericServlet class?

GenericServlet is an abstract class that implements the Servlet interface and the ServletConfig interface. In addition to the methods declared in these two interfaces, this class also provides simple versions of the lifecycle methods init and destroy, and implements the log method declared in the ServletContext interface.
Note: This class is known as generic servlet, since it is not specific to any protocol.

11.What's the difference between GenericServlet and HttpServlet?

GenericServlet	**HttpServlet**
The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods.	An abstract class that simplifies writing HTTP servlets. It extends the GenericServlet base class and provides an framework for handling the HTTP protocol.
The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers.	The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method.

GenericServlet is not specific to any protocol.	HttpServlet only supports HTTP and HTTPS protocol.

12.Why is HttpServlet declared abstract?

The HttpServlet class is declared abstract because the default implementations of the main service methods do nothing and must be overridden. This is a convenience implementation of the Servlet interface, which means that developers do not need to implement all service methods. If your servlet is required to handle `doGet()` requests for example, there is no need to write a `doPost()` method too.

13.Can servlet have a constructor ?

One can definitely have constructor in servlet.Even you can use the constrctor in servlet for initialization purpose,but this type of approch is not so common. You can perform common operations with the constructor as you normally do.The only thing is that you cannot call that constructor explicitly by the new keyword as we normally do.In the case of servlet, servlet container is responsible for instantiating the servlet, so the constructor is also called by servlet container only.

14.What are the types of protocols supported by HttpServlet ?

It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. So, HttpServlet only supports HTTP and HTTPS protocol.

15.What is the difference between doGet() and doPost()?

#	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.

16.When to use doGet() and when doPost()?

Always prefer to use GET (As because GET is faster than POST), except mentioned in the following reason:

- If data is sensitive
- Data is greater than 1024 characters
- If your application don't need bookmarks.

17.How do I support both GET and POST from the same Servlet?

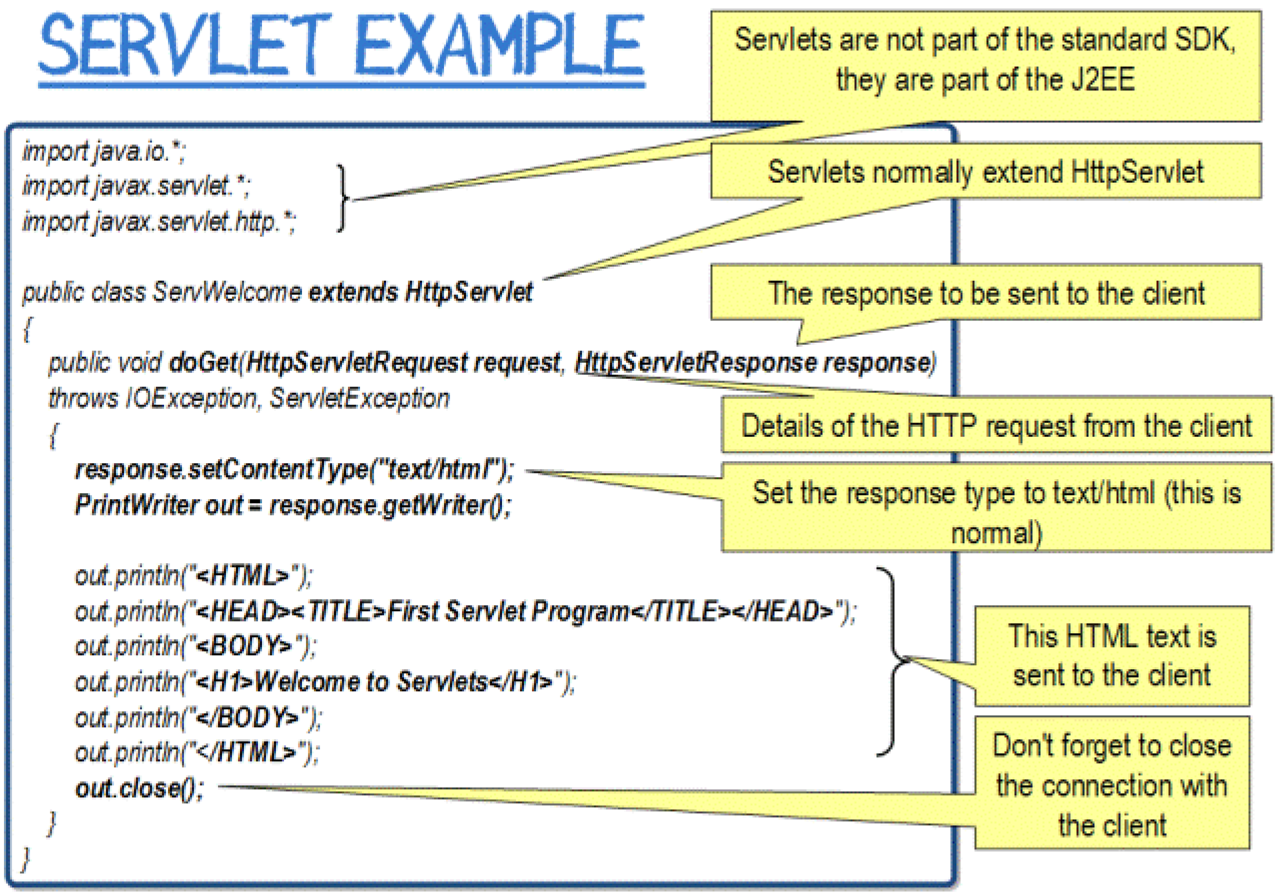
The easy way is, just support POST, then have your doGet method call your doPost method:

```
 public void doGet(HttpServletRequest request, HttpServletResponse response)
                             throws ServletException, IOException
{
    doPost(request, response);
}
```

18.Should I override the service() method?

We never override the service method, since the HTTP Servlets have already taken care of it . The default service function invokes the doXXX() method corresponding to the method of the HTTP request.For example, if the HTTP request method is GET, doGet() method is called by default. A servlet should override the doXXX() method for the HTTP methods that servlet supports. Because HTTP service method check the request method and calls the appropriate handler method, it is not necessary to override the service method itself. Only override the appropriate doXXX() method.

19.How the typical servlet code look like ?


SERVLET EXAMPLE
Servlets are not part of the standard SDK, they are part of the J2EE
Servlets normally extend HttpServlet
The response to be sent to the client
Details of the HTTP request from the client
Set the response type to text/html (this is normal)
This HTML text is sent to the client
Don't forget to close the connection with the client

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServWelcome extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      out.println("<HTML>");
      out.println("<HEAD><TITLE>First Servlet Program</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("<H1>Welcome to Servlets</H1>");
      out.println("</BODY>");
      out.println("</HTML>");
      out.close();
   }
}
```

20.What is a servlet context object?

A servlet context object contains the information about the Web application of which the servlet is a part. It also provides access to the resources common to all the servlets in the application. Each Web application in a container has a single servlet context associated with it.

21.What are the differences between the ServletConfig interface and the ServletContext interface?

ServletConfig	ServletContext
The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.	A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container.
There is one ServletConfig parameter per servlet.	There is one ServletContext for the entire webapp and all the servlets in a webapp share it.

The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file	The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file.

22.What's the difference between forward() and sendRedirect() methods?

forward()	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 context, the original request context is retained	Because this involves a new request, 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.

23.What is the difference between the include() and forward() methods?

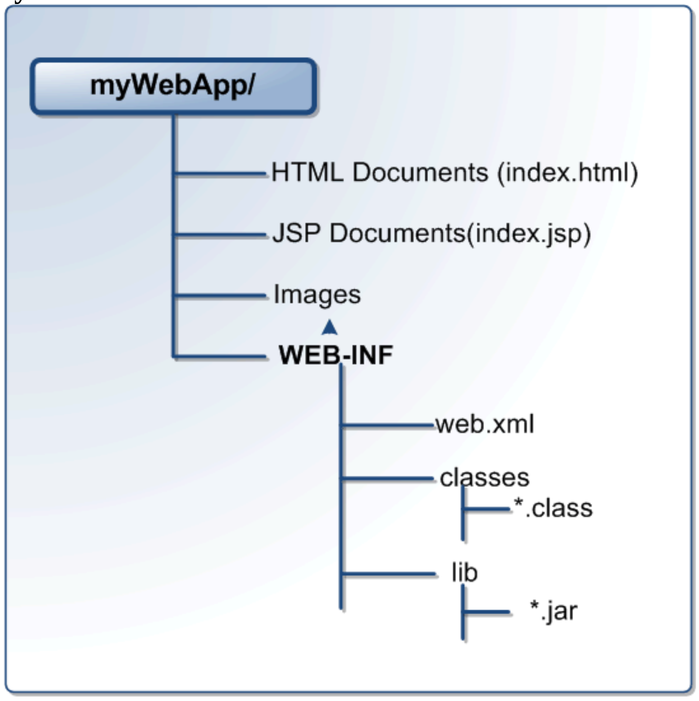
include()	forward()
The `RequestDispatcher include()` method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet.	The `RequestDispatcher forward()` method is used to show a different resource in place of the servlet that was originally called.
If you include a servlet or JSP document, the included resource must not attempt to change	The forwarded resource may be another servlet, JSP or static HTML document, but the

the response status code or HTTP headers, any such request will be ignored.	response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection.
The `include()` method is often used to include common "boilerplate" text or template markup that may be included by many servlets.	The `forward()` method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page.

24.What's the use of the servlet wrapper classes??

The `HttpServletRequestWrapper` and `HttpServletResponseWrapper` classes are designed to make it easy for developers to create custom implementations of the servlet request and response types. The classes are constructed with the standard `HttpServletRequest` and `HttpServletResponse` instances respectively and their default behaviour is to pass all method calls directly to the underlying objects.

25.What is the directory structure of a WAR file?



26.What is a deployment descriptor?

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.

27.What is the difference between the getRequestDispatcher(String path) method of javax.servlet.ServletRequest interface and javax.servlet.ServletContext interface?

ServletRequest.getRequestDispatcher(String path)	**ServletContext.getRequestDispatcher(String path)**
The `getRequestDispatcher(String path)` method of `javax.servlet.ServletRequest` interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a “/” it is interpreted as relative to the current context root.	The `getRequestDispatcher(String path)` method of `javax.servlet.ServletContext` interface cannot accept relative paths. All path must start with a “/” and are interpreted as relative to current context root.

28.What is preinitialization of a servlet?

A container does not initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading. The servlet specification defines the element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up. The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

29.What is the <load-on-startup> element?

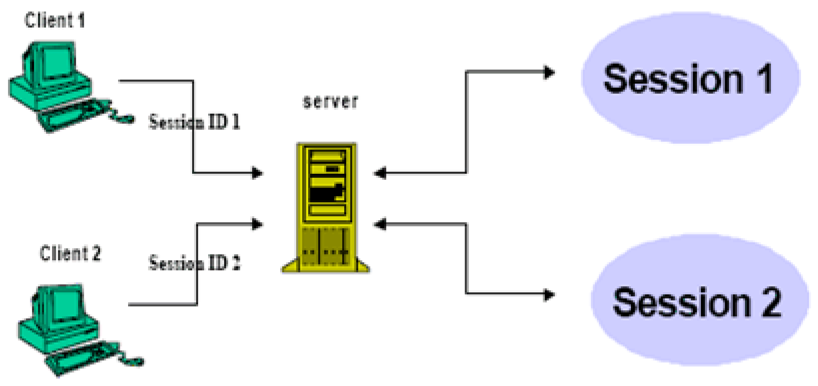
The `<load-on-startup>` element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

```
<servlet>
   <servlet-name>ServletName</servlet-name>
   <servlet-class>ClassName</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
```

Note: The container loads the servlets in the order specified in the <load-on-startup> element.

30.What is session?

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.



31.What is Session Tracking?

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

32.What is the need of Session Tracking in web application?

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is reffered as "state". To keep track of this state we need session tracking.

Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

33.What are the types of Session Tracking ?

Sessions need to work with all web browsers and take into account the users security preferences. Therefore there are a variety of ways to send and receive the identifier:

- **URL rewriting :** URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.

- **Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.

- **Cookies :** Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.

- **Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

Learn more about Session Tracking

34.How do I use cookies to store session state on the client?

In a servlet, the HttpServletResponse and HttpServletRequest objects passed to method HttpServlet.service() can be used to create cookies on the client and use cookie information transmitted during client requests. JSPs can also use cookies, in scriptlet code or, preferably, from within custom tag code.

- To set a cookie on the client, use the addCookie() method in class HttpServletResponse. Multiple cookies may be set for the same request, and a single cookie name may have multiple values.
- To get all of the cookies associated with a single HTTP request, use the getCookies() method of class HttpServletRequest

35.What are some advantages of storing session state in cookies?

- Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.
- For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

36.What are some disadvantages of storing session state in cookies?

- Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.
- All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.
- Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.
- Browser instances share cookies, so users cannot have multiple simultaneous sessions.

- Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package `javax.servlet.http` supports session management (via class `HttpSession`), package `javax.servlet` has no such support.

37.What is URL rewriting?

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method `HttpServletResponse`.encodeURL(). Each time a URL is output, the servlet passes the URL to encodeURL(), which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off. E.g., http://abc/path/index.jsp;jsessionid=123465hfhs

Advantages

- URL rewriting works just about everywhere, especially when cookies are turned off.
- Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and confuse the server by interacting with the same session through two instances.
- Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

DisAdvantages

- Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.
- URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
- URL rewriting does not work well with JSP technology.
- If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

38.How can an existing session be invalidated?

An existing session can be invalidated in the following two ways:

- Setting timeout in the deployment descriptor: This can be done by specifying timeout between the `<session-timeout>`tags as follows:

  ```
  <session-config>
         <session-timeout>10</session-timeout>
  </session-config>
  ```

  This will set the time for session timeout to be ten minutes.

- Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

  ```
  public void setMaxInactiveInterval(int interval)
  ```
  The `setMaxInactiveInterval()` method sets the maximum time in seconds before a session becomes invalid.
  Note :Setting the inactive period a*s negative(-1), makes the container stop tracking session, i.e, session never expires.*

39.How can the session in Servlet can be destroyed?

An existing session can be destroyed in the following two ways:

- Programatically : Using `session.invalidate()` method, which makes the container abonden the session on which the method is called.
- When the server itself is shutdown.

40.A client sends requests to two different web components. Both of the components access the session. Will they end up using the same session object or different session ?

Creates only one session i.e., they end up with using same session .

Sessions is specific to the client but not the web components. And there is a 1-1 mapping between client and a session.

41.What is servlet lazy loading?

- A container doesnot initialize the servlets ass soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.
- The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.

- The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

42.What is Servlet Chaining?

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

43.How are filters?

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

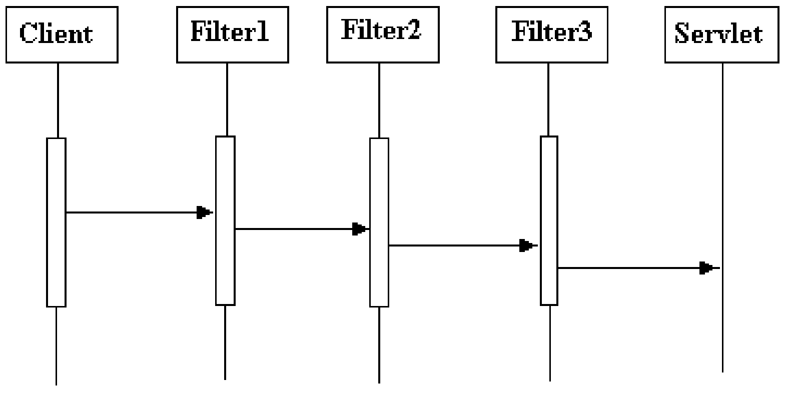
- Security checks
- Modifying the request or response
- Data compression
- Logging and auditing
- Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

44.What are the functions of an intercepting filter?

The functions of an intercepting filter are as follows:

- It intercepts the request from a client before it reaches the servlet and modifies the request if required.
- It intercepts the response from the servlet back to the client and modifies the request if required.
- There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



45.What are the functions of the Servlet container?

The functions of the Servlet container are as follows:

- **Lifecycle management** : It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
- **Communication support** : It handles the communication between the servlet and the Web server.
- **Multithreading support** : It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.
- **Declarative security** : It manages the security inside the XML deployment descriptor file.
- **JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them.

1.What are the advantages of JSP over Servlet?

JSP is a serverside technology to make content generation a simple appear.The advantage of JSP is that they are document-centric. Servlets, on the other hand, look and act like programs. A Java Server Page can contain Java program fragments that instantiate and execute Java classes, but these occur inside an HTML template file and are primarily used to generate dynamic content. Some of the JSP functionality can be achieved on the client, using JavaScript. The power of JSP is that it is server-based and provides a framework for Web application development.

2.What is the life-cycle of JSP?

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

  More about JSP Life cycle

3.What is the jspInit() method?

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.

4.What is the _jspService() method?

SThe _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.

5.What is the jspDestroy() method?

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.

6.What JSP lifecycle methods can I override?

You cannot override the _jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

7.How can I override the jspInit() and jspDestroy() methods within a JSP page?

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

```
<%!
        public void jspInit() {
                . . .
        }
%>
<%!
        public void jspDestroy() {
                . . .
        }
%>
```

8.What are implicit objects in JSP?

Implicit objects in JSP are the Java objects that the JSP Container makes available to developers in each page. These objects need not be declared or instantiated by the JSP author. They are automatically instantiated by the container and are accessed using standard variables; hence, they are called implicit objects.The implicit objects available in JSP are as follows:

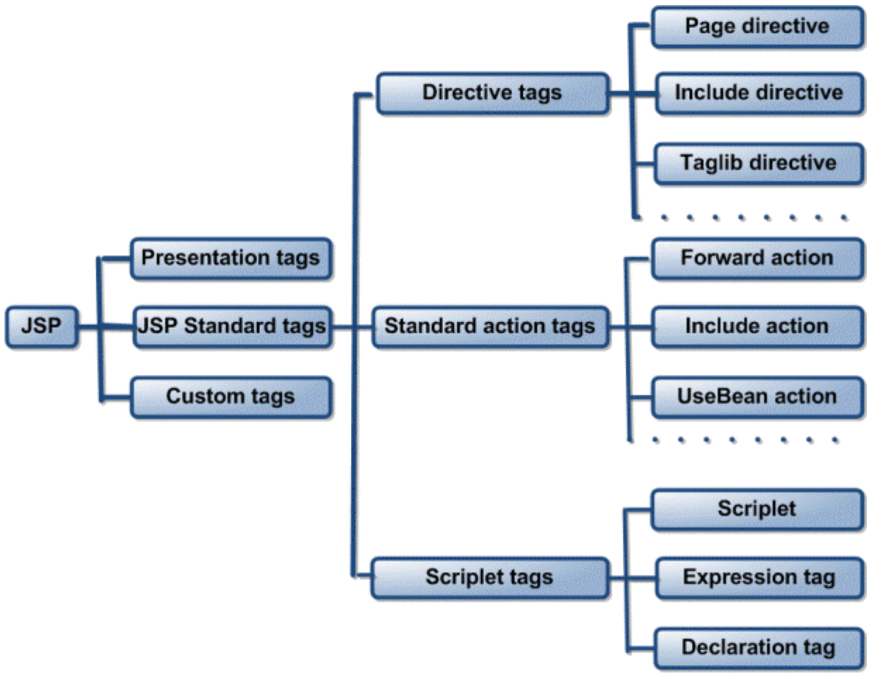
- request
- response
- pageContext
- session
- application
- out
- config
- page
- exception

The implicit objects are parsed by the container and inserted into the generated servlet code. They are available only within the jspService method and not in any declaration.

Check more about implicit objects

9.What are the different types of JSP tags?

The different types of JSP tags are as follows:



10.What are JSP directives?

- JSP directives are messages for the JSP engine. i.e., JSP directives serve as a message from a JSP page to the JSP container and control the processing of the entire page
- They are used to set global values such as a class declaration, method implementation, output content type, etc.
- They do not produce any output to the client.
- Directives are always enclosed within <%@ ….. %> tag.
- Ex: page directive, include directive, etc.

11.What is page directive?

- A page directive is to inform the JSP engine about the headers or facilities that page should get from the environment.
- Typically, the page directive is found at the top of almost all of our JSP pages.
- There can be any number of page directives within a JSP page (although the attribute – value pair must be unique).
- The syntax of the include directive is: <%@ page attribute="value">
- Example:`<%@ include file="header.jsp" %>`

12.What are the attributes of page directive?

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

- **import**: It specifies the packages that are to be imported.
- **session**: It specifies whether a session data is available to the JSP page.
- **contentType**: It allows a user to set the content-type for a page.
- **isELIgnored**: It specifies whether the EL expressions are ignored when a JSP is translated to a servlet.

13.What is the include directive?

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

- The include directive is used to statically insert the contents of a resource into the current JSP.
- This enables a user to reuse the code without duplicating it, and includes the contents of the specified file at the translation time.
- The syntax of the include directive is as follows:
  `<%@ include file = "FileName" %>`
- This directive has only one attribute called `file` that specifies the name of the file to be included.

14.What are the JSP standard actions?

- The JSP standard actions affect the overall runtime behavior of a JSP page and also the response sent back to the client.
- They can be used to include a file at the request time, to find or instantiate a JavaBean, to forward a request to a new page, to generate a browser-specific code, etc.
- Ex: `include, forward, useBean,etc.` object

15.What are the standard actions available in JSP?

The standard actions available in JSP are as follows:

- **<jsp:include>**: It includes a response from a servlet or a JSP page into the current page. It differs from an include directive in that it includes a resource at request processing time, whereas the include directive includes a resource at translation time.
- **<jsp:forward>**: It forwards a response from a servlet or a JSP page to another page.
- **<jsp:useBean>**: It makes a JavaBean available to a page and instantiates the bean.
- **<jsp:setProperty>**: It sets the properties for a JavaBean.
- **<jsp:getProperty>**: It gets the value of a property from a JavaBean component and adds it to the response.
- **<jsp:param>**: It is used in conjunction with <jsp:forward>;, <jsp:, or plugin>; to add a parameter to a request. These parameters are provided using the name-value pairs.
- **<jsp:plugin>**: It is used to include a Java applet or a JavaBean in the current JSP page.

16.What is the <jsp:useBean> standard action?

The `<jsp:useBean>` standard action is used to locate an existing JavaBean or to create a JavaBean if it does not exist. It has attributes to identify the object instance, to specify the lifetime of the bean, and to specify the fully qualified classpath and type.

17.What are the scopes available in <jsp:useBean>?

The scopes available in <jsp:useBean> are as follows:

- **page scope:**: It specifies that the object will be available for the entire JSP page but not outside the page.
- **request scope**: It specifies that the object will be associated with a particular request and exist as long as the request exists.
- **application scope**: It specifies that the object will be available throughout the entire Web application but not outside the application.
- **session scope**: It specifies that the object will be available throughout the session with a particular client.

18.What is the <jsp:forward> standard action?

- The `<jsp:forward>` standard action forwards a response from a servlet or a JSP page to another page.
- The execution of the current page is stopped and control is transferred to the forwarded page.
- The syntax of the <jsp:forward> standard action is :
  ```
  <jsp:forward page="/targetPage" />
  ```
  Here, targetPage can be a JSP page, an HTML page, or a servlet within the same context.

- If anything is written to the output stream that is not buffered before `<jsp:forward>`, an IllegalStateException will be thrown.

Note : Whenever we intend to use <jsp:forward> or <jsp:include> in a page, buffering should be enabled. By default buffer is enabled.

19.What is the <jsp:include> standard action?

The <jsp:include> standard action enables the current JSP page to include a static or a dynamic resource at runtime. In contrast to the include directive, the include action is used for resources that change frequently. The resource to be included must be in the same context.The syntax of the <jsp:include> standard action is as follows:
```
<jsp:include page="targetPage" flush="true"/>
```
Here, targetPage is the page to be included in the current JSP.

20.What is the difference between include directive and include action?

Include directive	Include action
The *include* directive, includes the content of the specified file during the translation phase–when the page is converted to a servlet.	The *include* action, includes the response generated by executing the specified page (a JSP page or a servlet) during the request processing phase–when the page is requested by a user.
The include directive is used to statically insert the contents of a resource into the current JSP.	The include standard action enables the current JSP page to include a static or a dynamic resource at runtime.
Use the include directive if the file changes rarely. It's the fastest mechanism.	Use the include action only for content that changes often, and if which page to include cannot be decided until the main page is requested.

21.Differentiate between pageContext.include and jsp:include?

The `<jsp:include>` standard action and the `pageContext.include()` method are both used to include resources at runtime. However, the `pageContext.include()` method always flushes the output of the current page before including the other components, whereas `<jsp:include>` flushes the output of the current page only if the value of flush is explicitly set to true as follows:

```
<jsp:include page="/index.jsp" flush="true"/>
```

22.What is the jsp:setProperty action?

You use `jsp:setProperty` to give values to properties of beans that have been referenced earlier. You can do this in two contexts. First, you can use `jsp:setProperty` after, but outside of, a `jsp:useBean` element, as below:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="myProperty" ... />
```

In this case, the jsp:setProperty is executed regardless of whether a new bean was instantiated or an existing bean was found.

A second context in which `jsp:setProperty` can appear is inside the body of a `jsp:useBean` element, as below:

```
<jsp:useBean id="myName" ... >
  ...
  <jsp:setProperty name="myName"
                   property="someProperty" ... />
</jsp:useBean>
```

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

23.What is the jsp:getProperty action?

The <jsp:getProperty> action is used to access the properties of a bean that was set using the <jsp:getProperty> action. The container converts the property to a String as follows:

- If it is an object, it uses the toString() method to convert it to a String.
- If it is a primitive, it converts it directly to a String using the valueOf() method of the corresponding Wrapper class.
- The syntax of the <jsp:getProperty> method is: `<jsp:getProperty name="Name" property="Property" />`

Here, name is the id of the bean from which the property was set. The property attribute is the property to get. A user must create or locate a bean using the <jsp:useBean> action before using the <jsp:getProperty> action.

24.What is the <jsp:param> standard action?

The <jsp:param> standard action is used with <jsp:include> or <jsp:forward> to pass parameter names and values to the target resource. The syntax of the <jsp:param> standard action is as follows:

```
<jsp:param name="paramName" value="paramValue"/>
```

25.What is the jsp:plugin action ?

This action lets you insert the browser-specific `OBJECT` or `EMBED` element needed to specify that the browser run an applet using the Java plugin.

26.What are scripting elements?

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

1. **Expressions** of the form `<%= expression %>` that are evaluated and inserted into the output,
2. **Scriptlets** of the form `<% code %>` that are inserted into the servlet's service method,
3. **Declarations** of the form `<%! code %>` that are inserted into the body of the servlet class, outside of any existing methods.

27.What is a scriptlet?

A scriptlet contains Java code that is executed every time a JSP is invoked. When a JSP is translated to a servlet, the scriptlet code goes into the `service()` method. Hence, methods and variables written in scriptlets are local to the `service()` method. A scriptlet is written between the **<% and %>** tags and is executed by the container at request processing time.

28.What are JSP declarations?

As the name implies, JSP declarations are used to declare class variables and methods in a JSP page. They are initialized when the class is initialized. Anything defined in a declaration is available for the whole JSP page. A declaration block is enclosed between the **<%! and %>** tags. A declaration is not included in the `service()` method when a JSP is translated to a servlet.

29.What is a JSP expression?

A JSP expression is used to write an output without using the `out.print statement`. It can be said as a shorthand representation for scriptlets. An expression is written between the **<%= and %>** tags. It is not required to end the expression with a semicolon, as it implicitly adds a semicolon to all the expressions within the expression tags.

30.How is scripting disabled?

Scripting is disabled by setting the scripting-invalid element of the deployment descriptor to true. It is a subelement of jsp-property-group. Its valid values are true and false. The syntax for disabling scripting is as follows:

```
<jsp-property-group>
   <url-pattern>*.jsp</url-pattern>
   <scripting-invalid>true</scripting-invalid>
</jsp-property-group>
```

STRUTS Questions

1.What is MVC?

Model-View-Controller (MVC) is a design pattern put together to help control change. MVC decouples interface from business logic and data.

- **Model :** The model contains the core of the application's functionality. The model encapsulates the state of the application. Sometimes the only functionality it contains is state. It knows nothing about the view or controller.

- **View:** The view provides the presentation of the model. It is the *look* of the application. The view can access the model getters, but it has no knowledge of the setters. In addition, it knows nothing about the controller. The view should be notified when changes to the model occur.

- **Controller:**The controller reacts to the user input. It creates and sets the model.

More about Model-View-Controller Architecture >>

2.What is a framework?

A framework is made up of the set of classes which allow us to use a library in a best possible way for a specific requirement.

3.What is Struts framework?

Struts framework is an open-source framework for developing the web applications in Java EE, based on MVC-2 architecture. It uses and extends the Java Servlet API. Struts is robust architecture and can be used for the development of application of any size. Struts framework makes it much easier to design scalable, reliable Web applications with Java.

4.What are the components of Struts?

Struts components can be categorize into Model, View and Controller:

- **Model:** Components like business logic /business processes and data are the part of model.
- **View:** HTML, JSP are the view components.

- **Controller:** Action Servlet of Struts is part of Controller components which works as front controller to handle all the requests.

5.What are the core classes of the Struts Framework?

Struts is a set of cooperating classes, servlets, and JSP tags that make up a reusable MVC 2 design.

- JavaBeans components for managing application state and behavior.
- Event-driven development (via listeners as in traditional GUI development).
- Pages that represent MVC-style views; pages reference view roots via the JSF component tree.

6.What is ActionServlet?

ActionServlet is a simple servlet which is the backbone of all Struts applications. It is the main Controller component that handles client requests and determines which Action will process each received request. It serves as an Action factory – creating specific Action classes based on user's request.

7.What is role of ActionServlet?

ActionServlet performs the role of Controller:

- Process user requests
- Determine what the user is trying to achieve according to the request
- Pull data from the model (if necessary) to be given to the appropriate view,
- Select the proper view to respond to the user
- Delegates most of this grunt work to Action classes
- Is responsible for initialization and clean-up of resources

8.What is the ActionForm?

ActionForm is javabean which represents the form inputs containing the request parameters from the View referencing the Action bean.

9.What are the important methods of ActionForm?

The important methods of ActionForm are : `validate()` & reset().

10.Describe validate() and reset() methods ?

validate() : Used to validate properties after they have been populated; Called before FormBean is handed to Action. Returns a collection of `ActionError` as `ActionErrors`.

Following is the method signature for the `validate()` method.

```
public ActionErrors validate(ActionMapping mapping,HttpServletRequest
request)
```

reset(): `reset()` method is called by Struts Framework with each request that uses the defined ActionForm. The purpose of this method is to reset all of the ActionForm's data members prior to the new request values being set.

```
public void reset() {}
```

11.What is ActionMapping?

Action mapping contains all the deployment information for a particular Action bean. This class is to determine where the results of the Action will be sent once its processing is complete.

12.How is the Action Mapping specified ?

We can specify the action mapping in the configuration file called `struts-config.xml`. Struts framework creates `ActionMapping` object from `<ActionMapping>` configuration element of `struts-config.xml` file

```
<action-mappings>
 <action path="/submit"
        type="submit.SubmitAction"
         name="submitForm"
         input="/submit.jsp"
         scope="request"
         validate="true">
  <forward name="success" path="/success.jsp"/>
  <forward name="failure" path="/error.jsp"/>
 </action>
</action-mappings>
```

13.What is role of Action Class?

An Action Class performs a role of an adapter between the contents of an incoming HTTP request and the corresponding business logic that should be executed to process this request.

14.In which method of Action class the business logic is executed ?

In the `execute()` method of Action class the business logic is executed.

```
public ActionForward execute(
            ActionMapping mapping,
              ActionForm form,
              HttpServletRequest request,
              HttpServletResponse response)
          throws Exception ;
```

`execute()` method of Action class:

- Perform the processing required to deal with this request
- Update the server-side objects (Scope variables) that will be used to create the next page of the user interface
- Return an appropriate `ActionForward` object

15.What design patterns are used in Struts?

Struts is based on model 2 MVC (Model-View-Controller) architecture. Struts controller uses the *command design pattern* and the action classes use the *adapter design pattern*. The `process()` method of the RequestProcessor uses the *template method design pattern*. Struts also implement the following J2EE design patterns.

- Service to Worker
- Dispatcher View
- Composite View (Struts Tiles)
- Front Controller
- View Helper
- Synchronizer Token

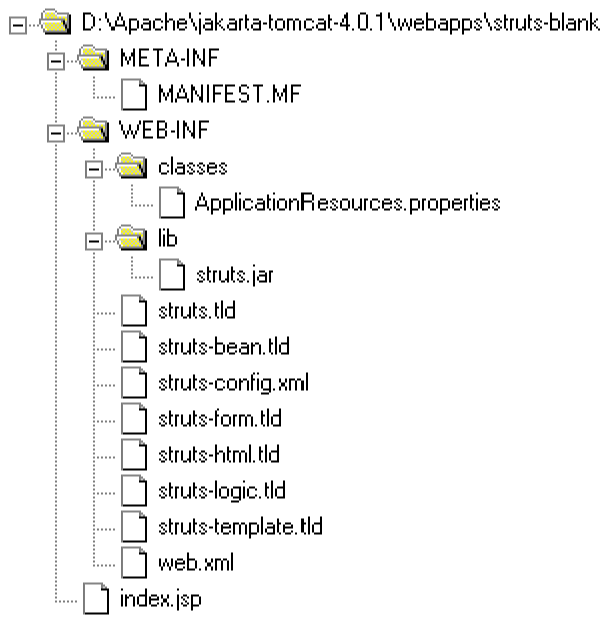
16.Can we have more than one struts-config.xml file for a single Struts application?

Yes, we can have more than one struts-config.xml for a single Struts application. They can be configured as follows:

```
<servlet>
<servlet-name>action</servlet-name>
  <servlet-class>
        org.apache.struts.action.ActionServlet
  </servlet-class>
<init-param>
  <param-name>config</param-name>
  <param-value>
     /WEB-INF/struts-config.xml,
     /WEB-INF/struts-admin.xml,
     /WEB-INF/struts-config-forms.xml
  </param-value>
</init-param>
.....
<servlet>
```

17.What is the directory structure of Struts application?

The directory structure of Struts application :



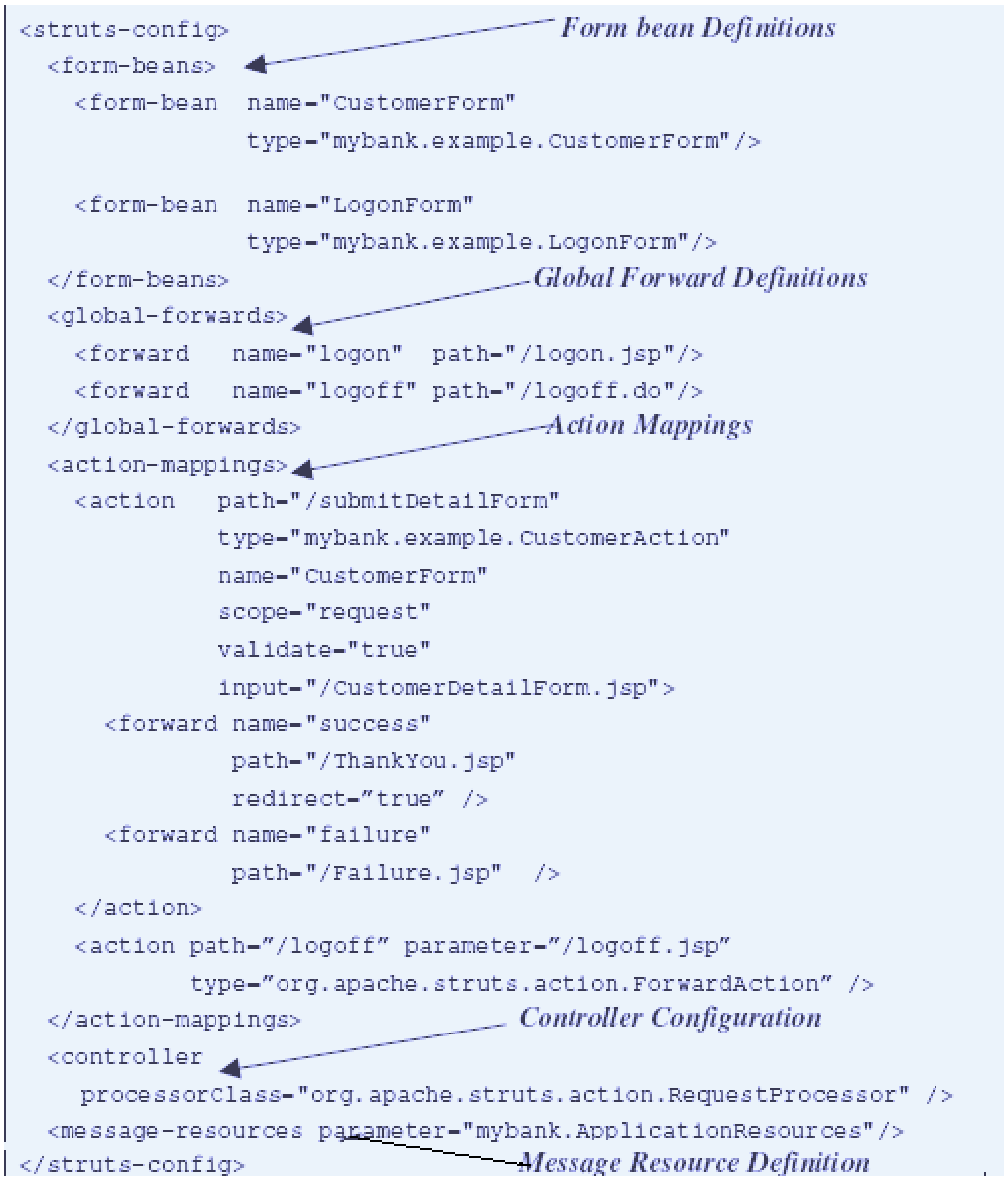
```
D:\Apache\jakarta-tomcat-4.0.1\webapps\struts-blank
    META-INF
        MANIFEST.MF
    WEB-INF
        classes
            ApplicationResources.properties
        lib
            struts.jar
        struts.tld
        struts-bean.tld
        struts-config.xml
        struts-form.tld
        struts-html.tld
        struts-logic.tld
        struts-template.tld
        web.xml
    index.jsp
```

18.What is the difference between session scope and request scope when saving formbean ?

when the scope is *request*,the values of formbean would be available for the current request.
when the scope is *session*,the values of formbean would be available throughout the session.

19.What are the important tags of struts-config.xml ?

The five important sections are:



```
<struts-config>
  <form-beans>
    <form-bean  name="CustomerForm"
                type="mybank.example.CustomerForm"/>

    <form-bean  name="LogonForm"
                type="mybank.example.LogonForm"/>
  </form-beans>
  <global-forwards>
    <forward   name="logon"  path="/logon.jsp"/>
    <forward   name="logoff" path="/logoff.do"/>
  </global-forwards>
  <action-mappings>
    <action   path="/submitDetailForm"
              type="mybank.example.CustomerAction"
              name="CustomerForm"
              scope="request"
              validate="true"
              input="/CustomerDetailForm.jsp">
      <forward name="success"
               path="/ThankYou.jsp"
               redirect="true" />
      <forward name="failure"
               path="/Failure.jsp"  />
    </action>
    <action path="/logoff" parameter="/logoff.jsp"
            type="org.apache.struts.action.ForwardAction" />
  </action-mappings>
  <controller
     processorClass="org.apache.struts.action.RequestProcessor" />
  <message-resources parameter="mybank.ApplicationResources"/>
</struts-config>
```

20.What are the different kinds of actions in Struts?

The different kinds of actions in Struts are:

- ForwardAction
- IncludeAction
- DispatchAction
- LookupDispatchAction
- SwitchAction

21.What is DispatchAction?

The DispatchAction class is used to group related actions into one class. Using this class, you can have a method for each logical action compared than a single execute method. The DispatchAction dispatches to one of the logical actions represented by the methods. It picks a method to invoke based on an incoming request parameter. The value of the incoming parameter is the name of the method that the DispatchAction will invoke.

22.How to use DispatchAction?

To use the DispatchAction, follow these steps :

- Create a class that extends DispatchAction (instead of Action)
- In a new class, add a method for every function you need to perform on the service – The method has the same signature as the `execute()` method of an Action class.
- Do not override `execute()` method – Because DispatchAction class itself provides `execute()` method.
- Add an entry to struts-config.xml

DispatchAction Example »

23.What is the use of ForwardAction?

The `ForwardAction` class is useful when you're trying to integrate Struts into an existing application that uses Servlets to perform business logic functions. You can use this class to take advantage of the Struts controller and its functionality, without having to rewrite the existing Servlets. Use `ForwardAction` to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page. By using this predefined action, you don't have to write your own Action class. You just have to set up the `struts-config` file properly to use `ForwardAction`.

24.What is IncludeAction?

The `IncludeAction` class is useful when you want to integrate Struts into an application that uses Servlets. Use the IncludeAction class to include another resource in the response to the request being processed.

25.What is the difference between ForwardAction and IncludeAction?

The difference is that you need to use the `IncludeAction` only if the action is going to be included by another action or jsp. Use `ForwardAction` to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page.

26.What is LookupDispatchAction?

The `LookupDispatchAction` is a subclass of `DispatchAction`. It does a reverse lookup on the resource bundle to get the key and then gets the method whose name is associated with the key into the Resource Bundle.

27.What is the use of LookupDispatchAction?

LookupDispatchAction is useful if the method name in the Action is not driven by its name in the front end, but by the Locale independent key into the resource bundle. Since the key is always the same, the LookupDispatchAction shields your application from the side effects of I18N.

28.What is difference between LookupDispatchAction and DispatchAction?

The difference between LookupDispatchAction and DispatchAction is that the actual method that gets called in LookupDispatchAction is based on a lookup of a key value instead of specifying the method name directly.

29.What is SwitchAction?

The SwitchAction class provides a means to switch from a resource in one module to another resource in a different module. SwitchAction is useful only if you have multiple modules in your Struts application. The SwitchAction class can be used as is, without extending.

30.What if `<action>` element has `<forward>` declaration with same name as global forward?

In this case the global forward is not used. Instead the `<action>` element's `<forward>` takes precendence.

31.What is DynaActionForm?

A specialized subclass of `ActionForm` that allows the creation of form beans with dynamic sets of properties (configured in configuration file), without requiring the developer to create a Java class for each type of form bean.

32.What are the steps need to use DynaActionForm?

Using a `DynaActionForm` instead of a custom subclass of ActionForm is relatively straightforward. You need to make changes in two places:

- In struts-config.xml: change your `<form-bean>` to be an `org.apache.struts.action.DynaActionForm` instead of some subclass of `ActionForm`

```
<form-bean name="loginForm"type="org.apache.struts.action.DynaActionForm" >
    <form-property name="userName" type="java.lang.String"/>
    <form-property name="password" type="java.lang.String" />
</form-bean>
```

- In your `Action` subclass that uses your form bean:
  - o import `org.apache.struts.action.DynaActionForm`
  - o downcast the `ActionForm` parameter in `execute()` to a `DynaActionForm`
  - o access the form fields with `get(field)` rather than `getField()`

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;
import org.apache.struts.action.DynaActionForm;

public class DynaActionFormExample extends Action {
 public ActionForward execute(ActionMapping mapping, ActionForm form,
   HttpServletRequest request, HttpServletResponse response)
            throws Exception {
  DynaActionForm loginForm = (DynaActionForm) form;
                ActionMessages errors = new ActionMessages();
        if (((String) loginForm.get("userName")).equals("")) {
            errors.add("userName", new ActionMessage(
                            "error.userName.required"));
        }
        if (((String) loginForm.get("password")).equals("")) {
            errors.add("password", new ActionMessage(
                            "error.password.required"));
        }
        ...........
```

33.How to display validation errors on jsp page?

`<html:errors/>` tag displays all the errors. `<html:errors/>` iterates over ActionErrors request attribute.

34.What are the various Struts tag libraries?

The various Struts tag libraries are:

- HTML Tags
- Bean Tags
- Logic Tags
- Template Tags
- Nested Tags
- Tiles Tags

35.What is the use of <logic:iterate>?

`<logic:iterate>` repeats the nested body content of this tag over a specified collection.

```
<table border=1>
  <logic:iterate id="customer" name="customers">
    <tr>
      <td><bean:write name="customer" property="firstName"/></td>
      <td><bean:write name="customer" property="lastName"/></td>
      <td><bean:write name="customer" property="address"/></td>
   </tr>
  </logic:iterate>
</table>
```

36.What are differences between <bean:message> and <bean:write>

<bean:message>: is used to retrive keyed values from resource bundle. It also supports the ability to include parameters that can be substituted for defined placeholders in the retrieved string.

```
<bean:message key="prompt.customer.firstname"/>
```

<bean:write>: is used to retrieve and print the value of the bean property. <bean:write> has no body.

```
<bean:write name="customer" property="firstName"/>
```

37.How the exceptions are handled in struts?

Exceptions in Struts are handled in two ways:

- **Programmatic exception handling** :

  Explicit try/catch blocks in any code that can throw exception. It works well when custom value (i.e., of variable) needed when error occurs.

- **Declarative exception handling** :You can either define `<global-exceptions>` handling tags in your `struts-config.xml` or define the exception handling tags within `<action></action>` tag. It works well when custom page needed when error occurs. This approach applies only to exceptions thrown by Actions.

```
<global-exceptions>
 <exception key="some.key"
            type="java.lang.NullPointerException"
            path="/WEB-INF/errors/null.jsp"/>
</global-exceptions>
```

(OR)

```
<exception key="some.key"
           type="package.SomeException"
           path="/WEB-INF/somepage.jsp"/>
```

38.What is difference between ActionForm and DynaActionForm?

- An `ActionForm` represents an HTML form that the user interacts with over one or more pages. You will provide properties to hold the state of the form with getters and setters to access them. Whereas, using `DynaActionForm` there is no need of providing properties to hold the state. Instead these properties and their type are declared in the `struts-config.xml`
- The `DynaActionForm` bloats up the Struts config file with the xml based definition. This gets annoying as the Struts Config file grow larger.
- The `DynaActionForm` is not strongly typed as the ActionForm. This means there is no compile time checking for the form fields. Detecting them at runtime is painful and makes you go through redeployment.
- ActionForm can be cleanly organized in packages as against the flat organization in the Struts Config file.
- ActionForm were designed to act as a Firewall between HTTP and the Action classes, i.e. isolate and encapsulate the HTTP request parameters from direct use in Actions. With `DynaActionForm`, the property access is no different than using request.getParameter(..).
- `DynaActionForm` construction at runtime requires a lot of Java Reflection (Introspection) machinery that can be avoided.

39.How can we make message resources definitions file available to the Struts framework environment?

We can make message resources definitions file (properties file) available to Struts framework environment by adding this file to `struts-config.xml`.

```
<message-resources parameter="com.login.struts.ApplicationResources"/>
```

40.What is the life cycle of ActionForm?

The lifecycle of `ActionForm` invoked by the `RequestProcessor` is as follows:

- Retrieve or Create Form Bean associated with `Action`
- "Store" FormBean in appropriate scope (`request` or `session`)
- Reset the properties of the FormBean
- Populate the properties of the FormBean
- Validate the properties of the FormBean
- Pass FormBean to `Action`

Hibernate Questions

1.What is ORM ?

ORM stands for object/relational mapping. ORM is the automated persistence of objects in a Java application to the tables in a relational database.

2.What does ORM consists of ?

An ORM solution consists of the followig four pieces:

- API for performing basic CRUD operations
- API to express queries refering to classes
- Facilities to specify metadata
- Optimization facilities : dirty checking,lazy associations fetching

3.What are the ORM levels ?

The ORM levels are:

- Pure relational (stored procedure.)
- Light objects mapping (JDBC)
- Medium object mapping
- Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

4.What is Hibernate?

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files.Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

5.Why do you need ORM tools like hibernate?

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

- **Improved productivity**
    - High-level object-oriented API
    - Less Java code to write
    - No SQL to write
- **Improved performance**
    - Sophisticated caching
    - Lazy loading
    - Eager loading
- **Improved maintainability**

    - A lot less code to write
- **Improved portability**
    - ORM framework generates database-specific SQL for you

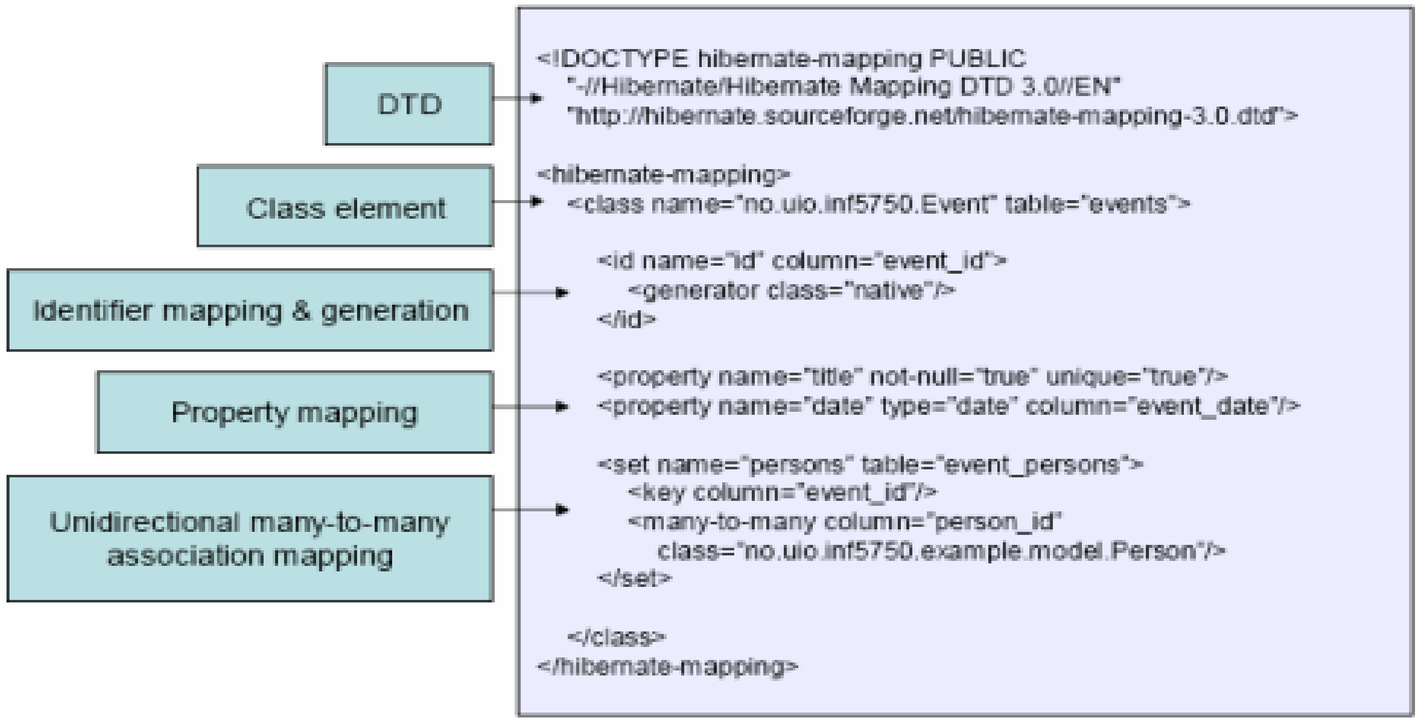
6.What Does Hibernate Simplify?

Hibernate simplifies:

- Saving and retrieving your domain objects
- Making database column and table name changes
- Centralizing pre save and post retrieve logic
- Complex joins for retrieving related items
- Schema creation from object model

7.What is the need for Hibernate xml mapping file?

Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:


DTD
Class element
Identifier mapping & generation
Property mapping
Unidirectional many-to-many association mapping

```
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="no.uio.inf5750.Event" table="events">

      <id name="id" column="event_id">
         <generator class="native"/>
      </id>

      <property name="title" not-null="true" unique="true"/>
      <property name="date" type="date" column="event_date"/>

      <set name="persons" table="event_persons">
         <key column="event_id"/>
         <many-to-many column="person_id"
            class="no.uio.inf5750.example.model.Person"/>
      </set>

   </class>
</hibernate-mapping>
```

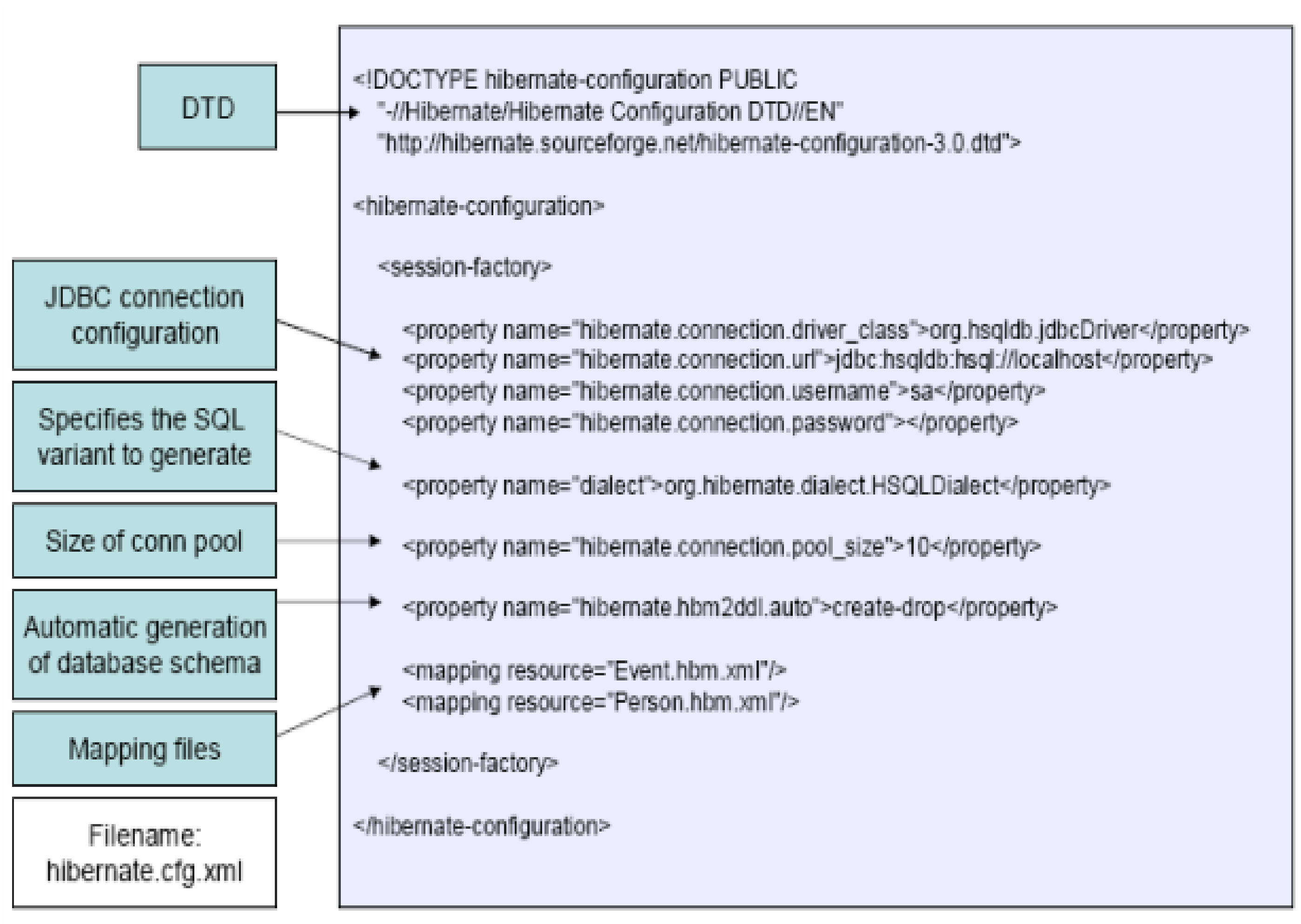
8.What are the most common methods of Hibernate configuration?

The most common methods of Hibernate configuration are:

- Programmatic configuration
- XML configuration (`hibernate.cfg.xml`)

9.What are the important tags of hibernate.cfg.xml?

Following are the important tags of hibernate.cfg.xml:


DTD
JDBC connection configuration
Specifies the SQL variant to generate
Size of conn pool
Automatic generation of database schema
Mapping files
Filename: hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <property name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="hibernate.connection.url">jdbc:hsqldb:hsql://localhost</property>
    <property name="hibernate.connection.username">sa</property>
    <property name="hibernate.connection.password"></property>

    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <property name="hibernate.connection.pool_size">10</property>

    <property name="hibernate.hbm2ddl.auto">create-drop</property>

    <mapping resource="Event.hbm.xml"/>
    <mapping resource="Person.hbm.xml"/>

  </session-factory>

</hibernate-configuration>
```

10.What are the Core interfaces are of Hibernate framework?

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

- Session interface
- SessionFactory interface
- Configuration interface
- Transaction interface
- Query and Criteria interfaces

11.What role does the Session interface play in Hibernate?

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.

```
Session session = sessionFactory.openSession();
```

Session interface role:

- Wraps a JDBC connection
- Factory for Transaction
- Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

12.What role does the SessionFactory interface play in Hibernate?

The application obtains Session instances from a SessionFactory. There is typically a single SessionFactory for the whole application—created during application initialization. The SessionFactory caches generate SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

13.What is the general flow of Hibernate communication with RDBMS?

The general flow of Hibernate communication with RDBMS is :

- Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
- Create session factory from configuration object
- Get one session from this session factory
- Create HQL Query
- Execute query to get list containing Java objects

14.What is Hibernate Query Language (HQL)?

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

15.How do you map Java Objects with Database tables?

- First we need to write Java domain objects (beans with setter and getter).
- Write hbm.xml, where we map java class to table and database columns to Java class variables.

Example :

```
<hibernate-mapping>
  <class name="com.test.User"  table="user">
   <property  column="USER_NAME" length="255"
      name="userName" not-null="true"  type="java.lang.String"/>
   <property  column="USER_PASSWORD" length="255"
     name="userPassword" not-null="true"  type="java.lang.String"/>
 </class>
</hibernate-mapping>
```

16.What's the difference between load() and get()?

load() vs. get() :-

load()	**get()**
Only use the `load()` method if you are sure that the object exists.	If you are not sure that the object exists, then use one of the `get()` methods.
`load()` method will throw an exception if the unique id is not found in the database.	`get()` method will return null if the unique id is not found in the database.
`load()` just returns a proxy by default and database won't be hit until the proxy is first invoked.	`get()` will hit the database immediately.

17.What is the difference between and merge and update ?

Use `update()` if you are sure that the session does not contain an already persistent instance with the same identifier, and `merge()` if you want to merge your modifications at any time without consideration of the state of the session.

18.How do you define sequence generated primary key in hibernate?

Using <generator> tag.
Example:-

```
<id column="USER_ID" name="id" type="java.lang.Long">
   <generator class="sequence">
     <param name="table">SEQUENCE_NAME</param>
   <generator>
</id>
```

19.Define cascade and inverse option in one-many mapping?

cascade - enable operations to cascade to child entities.
cascade="all|none|save-update|delete|all-delete-orphan"

inverse - mark this collection as the "inverse" end of a bidirectional association.
inverse="true|false"
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

20.What do you mean by Named – SQL query?

Named SQL queries are defined in the mapping xml document and called wherever required.
Example:

```
<sql-query name = "empdetails">
   <return alias="emp" class="com.test.Employee"/>
      SELECT emp.EMP_ID AS {emp.empid},
                 emp.EMP_ADDRESS AS {emp.address},
                 emp.EMP_NAME AS {emp.name}
      FROM Employee EMP WHERE emp.NAME LIKE :name
</sql-query>
```

Invoke Named Query :

```
List people = session.getNamedQuery("empdetails")
                    .setString("TomBrady", name)
                    .setMaxResults(50)
                    .list();
```

21.How do you invoke Stored Procedures?

```
<sql-query name="selectAllEmployees_SP" callable="true">
 <return alias="emp" class="employee">
   <return-property name="empid" column="EMP_ID"/>

   <return-property name="name" column="EMP_NAME"/>
   <return-property name="address" column="EMP_ADDRESS"/>
    { ? = call selectAllEmployees() }
 </return>
</sql-query>
```

22.Explain Criteria API

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.
Example :

```
List employees = session.createCriteria(Employee.class)
                        .add(Restrictions.like("name", "a%") )
                        .add(Restrictions.like("address", "Boston"))
                        .addOrder(Order.asc("name") )
                        .list();
```

23.Define HibernateTemplate?

`org.springframework.orm.hibernate.HibernateTemplate` is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

24.What are the benefits does HibernateTemplate provide?

The benefits of HibernateTemplate are :

- `HibernateTemplate`, a Spring Template class simplifies interactions with Hibernate Session.
- Common functions are simplified to single method calls.
- Sessions are automatically closed.
- Exceptions are automatically caught and converted to runtime exceptions.

25.How do you switch between relational databases without code changes?

Using Hibernate SQL Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

26.If you want to see the Hibernate generated SQL statements on console, what should we do?

In Hibernate configuration file set as follows:
```
<property name="show_sql">true</property>
```

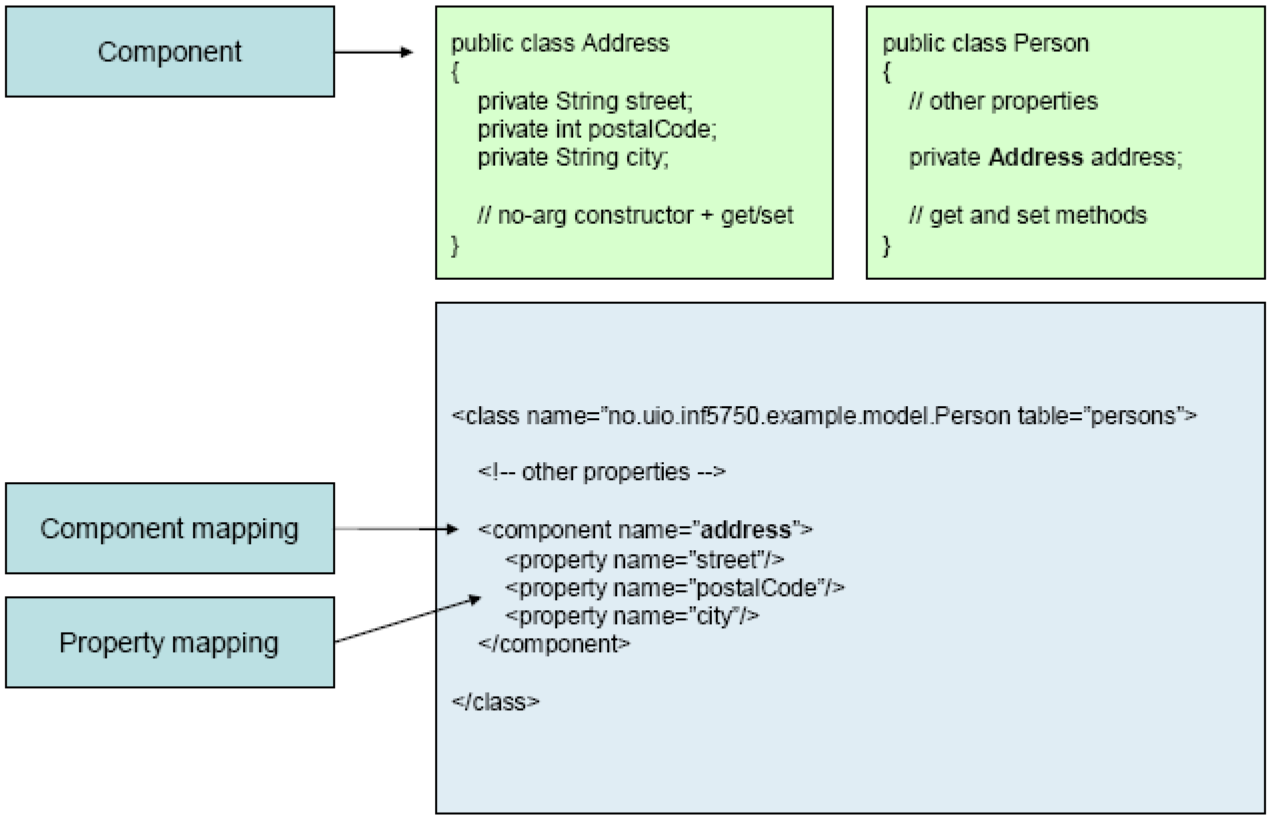
27.What are derived properties?

The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

28.What is component mapping in Hibernate?

- A component is an object saved as a value, not as a reference
- A component can be saved directly without needing to declare interfaces or identifier properties
- Required to define an empty constructor
- Shared references not supported

Example:



```
public class Address
{
    private String street;
    private int postalCode;
    private String city;

    // no-arg constructor + get/set
}
```

```
public class Person
{
    // other properties

    private Address address;

    // get and set methods
}
```

```
<class name="no.uio.inf5750.example.model.Person table="persons">

    <!-- other properties -->

    <component name="address">
        <property name="street"/>
        <property name="postalCode"/>
        <property name="city"/>
    </component>

</class>
```

29.What is the difference between sorted and ordered collection in hibernate?

sorted collection vs. order collection :-

sorted collection	**order collection**
A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator.	Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval.
If your collection is not large, it will be more efficient way to sort it.	If your collection is very large, it will be more efficient way to sort it .

31.What is the advantage of Hibernate over jdbc?

Hibernate Vs. JDBC :-

JDBC	Hibernate
With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema.	Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code.	Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS.
JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task.	Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application.
Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object-to-table.	Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties.
With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually.	Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost.
With JDBC, caching is maintained by hand-coding.	Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code.

In JDBC there is no check that always every user has updated data. This check has to be added by the developer.	Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data.

32.What are the Collection types in Hibernate ?

- Bag
- Set
- List
- Array
- Map

33.What are the ways to express joins in HQL?

HQL provides four ways of expressing (inner and outer) joins:-

- An *implicit* association join
- An ordinary join in the FROM clause
- A fetch join in the FROM clause.
- A *theta-style* join in the WHERE clause.

34.Define cascade and inverse option in one-many mapping?

cascade - enable operations to cascade to child entities.
cascade="all|none|save-update|delete|all-delete-orphan"

inverse - mark this collection as the "inverse" end of a bidirectional association.
inverse="true|false"
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

35.What is Hibernate proxy?

The `proxy` attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

36.How can Hibernate be configured to access an instance variable directly and not through a setter method ?

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

37.How can a whole class be mapped as immutable?

Mark the class as mutable="false" (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

38.What is the use of dynamic-insert and dynamic-update attributes in a class mapping?

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

- `dynamic-update` (defaults to `false`): Specifies that `UPDATE` SQL should be generated at runtime and contain only those columns whose values have changed
- `dynamic-insert` (defaults to `false`): Specifies that `INSERT` SQL should be generated at runtime and contain only the columns whose values are not null.

39.What do you mean by fetching strategy ?

A *fetching strategy* is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association. Fetch strategies may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or `Criteria` query.

40.What is automatic dirty checking?

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

41.What is transactional write-behind?

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations but is still sufficiently predictable to the user. This feature is called transactional write-behind.

42.What are Callback interfaces?

Callback interfaces allow the application to receive a notification when something interesting happens to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.

43.What are the types of Hibernate instance states ?

Three types of instance states:

- Transient -The instance is not associated with any persistence context
- Persistent -The instance is associated with a persistence context
- Detached -The instance was associated with a persistence context which has been closed – currently not associated

44.What are the differences between EJB 3.0 & Hibernate

Hibernate Vs EJB 3.0 :-

Hibernate	**EJB 3.0**
Session–Cache or collection of loaded objects relating to a single unit of work	**Persistence Context**-Set of entities that can be managed by a given EntityManager is defined by a persistence unit
XDoclet Annotations used to support Attribute Oriented Programming	**Java 5.0 Annotations** used to support Attribute Oriented Programming
Defines HQL for expressing queries to the database	**Defines EJB QL** for expressing queries
Supports Entity Relationships through mapping files and annotations in JavaDoc	**Support Entity Relationships** through Java 5.0 annotations
Provides a Persistence Manager API exposed via the Session, Query, Criteria, and Transaction API	**Provides and Entity Manager Interface** for managing CRUD operations for an Entity
Provides callback support through lifecycle, interceptor, and validatable interfaces	**Provides callback support** through Entity Listener and Callback methods
Entity Relationships are unidirectional. Bidirectional relationships are implemented by two unidirectional relationships	**Entity Relationships are bidirectional or unidirectional**

45.What are the types of inheritance models in Hibernate?

There are three types of inheritance models in Hibernate:

- Table per class hierarchy
- Table per subclass
- Table per concrete class