Object Oriented Concepts: -

1. **What are the principle concepts of OOPS?**

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

a) Abstraction
b) Polymorphism
c) Inheritance
d) Encapsulation

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?**

a) **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.
b) **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
c) **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

5. **What is Inheritance?**

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

a) In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
b) 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.
<u>Note</u>: From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:

a) Method overloading
b) Method overriding through inheritance
c) Method overriding through the Java interface

9. What is runtime polymorphism or dynamic method dispatch?

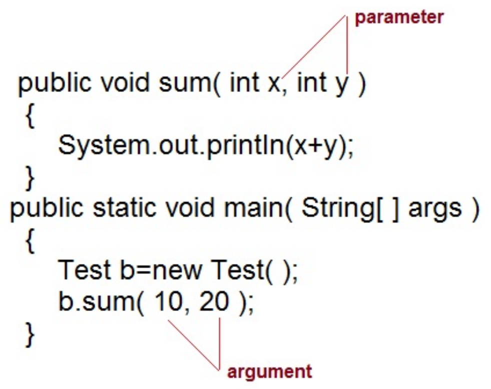
⇨ In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being 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. Parameter Vs. Argument

⇨ While talking about method, it is important to know the difference between two terms parameter and argument. Parameter is variable defined by a method that receives value when the method is called. Parameter are always local to the method they don't have scope outside the method. While argument is a value that is passed to a method when it is called.



```
 public void sum( int x, int y )
  {
     System.out.println(x+y);
  }
public static void main( String[ ] args )
  {
     Test b=new Test( );
     b.sum( 10, 20 );
  }
```

There are two ways to pass an argument to a method

a) **<u>Call-by-value:</u>** In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments.
b) **<u>Call-by-reference:</u>** In this reference of an argument is pass to a method. Any changes made inside the method will affect the argument value.

NOTE: In Java, when you pass a primitive type to a method it is passed by value whereas when you pass an object of any type to a method it is passed as reference.

12. What is method overloading? [Honeywell]

⇨ 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. Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done **by changing number of arguments or by changing the data type of arguments**.
Different ways of Method overloading:
a) Method overloading by changing data type of Arguments
b) Method overloading by changing no. of argument.

Note:

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

13. 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 subclass type.

Note:

a) 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).
b) You cannot override a method marked final
c) You cannot override a method marked static

14. **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 (based on declared argument types) is selected. Happens at compile time. The actual method that's invoked is still a virtual method invocation that happens at runtime, but the compiler will already know the signature of the method to be invoked. So, at runtime the argument match will already have been nailed down, just not the class in which the method lives.	Object type (in other words, the type of the actual instance on the heap) determines which method is selected. Happens at runtime.

15. **Can overloaded methods be override too?**

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

16. **Is it possible to override the main method?**

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

17. 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();
```

18. 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 using 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 using the super keyword.

Note:

a) You can only go back one level.

b) 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.

19. **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
 }
```

20. What is an Interface?

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

Note:

a) You can’t mark an interface as final.

b) Interface variables must be static.

c) An Interface cannot extend anything but another interface.

21. Can we instantiate an interface?

⇨ You can’t instantiate an interface directly but you can instantiate a class that implements an interface.

22. 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 so you must write a class that implements the interface and fulfill all the methods defined in it.

23. 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.

24. **What modifiers are allowed for methods in an Interface?**

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

25. **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 to be serialized and de-serialized.

26. 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.

Note:

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

27. Can we instantiate an abstract class?

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

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

Abstract Class	**Interfaces**
An abstract class can provide complete, default code and/or just the details that should 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 should 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.

29. **When should I use abstract classes and when should I use interfaces?**

Use Interfaces when…

a) You see that something in your design will change frequently.
b) If various implementations only share method signatures, then it is better to use Interfaces.
c) 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…

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

30. **When you declare a method as abstract, can other no abstract methods access it?**
➪ Yes, other no abstract methods can access a method that you declare as abstract.

31. **Can there be an abstract class with no abstract methods in it?**
➪ Yes, there can be an abstract class without abstract methods.

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

   There are two types of Constructor
   a) Default Constructor
   b) Parameterized constructor

```
Car c = new Car()       //Default constructor invoked
Car c = new Car(name); //Parameterized constructor invoked
```

33. **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.

34. **Can constructor be inherited**?
➪ No, constructor cannot be inherited, though a derived class can call the base class constructor.

35. Constructor Overloading
➪ Constructor overloading is done to construct object in different ways.Constructor can also be overloaded. Overloaded constructors are differentiated based on their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

36. **What are the differences between Constructor's 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	Any name except the class.

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
	Constructors can only call once	Methods could be called many times and it can return a value or void.

37. *this* keyword

a) *this* keyword is used to refer to current object.
b) *this* is always a reference to the object on which method was invoked.
c) *this* can be used to invoke current class constructor.
d) *this* can be passed as an argument to another method.
e) The *this* is used to call overloaded constructor in java
f) The *this* is also used to call Method of that class.
g) The *this* is used to return current Object

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

a) Constructors use *this* to refer to another constructor in the same class with a different parameter list.
b) 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.

39. 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.

40. 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 *access specifiers*.

41. **What are Access Specifiers available in Java**?

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

a) **Public**- *public* classes, methods, and fields can be accessed from everywhere.
b) **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.
c) **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.
d) **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

42. **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.
a) ***final* Classes**- A final class cannot have subclasses.
b) ***final* Variables**- A final variable cannot be changed once it is initialized.
c) ***final* Methods**- A final method cannot be overridden by subclasses.

43. What are the uses of final method?
⇨ There are two reasons for marking a method as final:
a) Disallowing subclasses to change the meaning of the method.
b) Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

44. **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.

45. **What are static variables?**
⇨ Variables that have only one copy per class are known as static variables. They are not attached to instance of a class but rather belong to a class. 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.

46. **What is the difference between static and non-static variables?**
⇨ A static variable is associated with the class rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

47. **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 not an instance of the class. Static methods are always invoked without reference to an instance of a class.
Note: The use of a static method suffers from the following restrictions:
a) A static method can only call other static methods.

b) A static method must only access static data.
c) A static method cannot reference to the current object using keywords super or this.

48. Can We Overload main () method?

⇨ Yes, we can overload main () method. A Java class can have any number of main () methods. But to run the java class, class should have main () method with signature as "public static void main(String[] args)". If you do any modification to this signature, compilation will be successful. But, you can't run the java program. You will get run time error as main method not found.

```
public class MainMethod
{
    public static void main(String[] args)
    {
        System.out.println("Execution starts from this method");
    }

    void main(int args)
    {
        System.out.println("Another main method");
    }

    double main(int i, double d)
    {
        System.out.println("Another main method");

        return d;
    }
}
```

49. Can we declare main () method as private or protected or with no access modifier?

No, main () method must be public. You can't define main () method as private or protected or with no access modifier. This is because to make the main () method accessible to JVM. If you define main () method other than public, compilation will be successful but you will get run time error as no main method found.

```
public class MainMethod
{
    private static void main(String[] args)
    {
        //Run time error
    }
}
```

50. Can We Declare main () Method as Non-Static?

No, main () method must be declared as static so that JVM can call main () method without instantiating its class. If you remove 'static' from main () method signature, compilation will be successful but program fails at run time.

```
public class MainMethod
{
    public void main(String[] args)
    {
        System.out.println(1);          //Run time error
    }
}
```

51. Why main () method must be static?

Suppose, if main () is allowed to be non-static, then while calling the main method JVM has to instantiate its class. While instantiating it must call constructor of that class. There will be an ambiguity if constructor of that class takes an argument. For example, In the below program what argument JVM must pass while instantiating class "MainMethod"? That's why main () method must be static.

```java
public class MainMethod
{
    public MainMethod(int i)
    {
        //Constructor taking one argument
    }

    public void main(String[] args)
    {
        //main method as non-static
    }
}
```

52. Can we change return type of main () method?

⇨ No, the return type of main () method must be void only. Any other type is not acceptable.

```java
public class MainMethod
{
    public static int main(String[] args)
    {
        return 1;    //run time error : No main method found
    }
}
```

53. Can main () method take an argument other than string array?

⇨ No, argument of main () method must be string array. But, from the introduction of var args you can pass var args of string type as an argument to main () method. Again, var args are nothing but the arrays.

```java
public class MainMethod
{
    public static void main(String... args)
    {
        //Var args as an argument
    }
}
```

54. Can we run java class without main () method?

⇨ No, you can't run java class without main method. But, there are some scenarios like if super class has main () method, then sub class can be run without defining main () method in it. For example, if you run class B in the below program, you will get 1 as output. Before Java 7, you can run java class by using static initializers. But, from Java 7 it is not possible.

```java
class A
{
    public static void main(String[] args)
    {
        System.out.println(1);
    }
}

public class B extends A
{

}
```

55. Can we make main synchronized in Java?

⇨ Yes, main can be synchronized in Java, synchronized modifier is allowed in the main signature and you can make your main method synchronized in Java.

56. Can we make main final in Java?

⇨ you can make the main method final in Java. JVM has no issue with that. Unlike any final method, you cannot override main in Java

57. What is a class loader? What are the different class loaders used by JVM?

⇨ Part of JVM which is used to load classes and interfaces. Bootstrap, Extension and System are the class loaders used by JVM.

58. Difference between loadClass and Class.forName?

⇨ loadClass only loads the class but doesn't initialize the object whereas Class.forName initialize the object after loading it.

59. What are various types of Class loaders used by JVM?

a) Bootstrap - Loads JDK internal classes, java. * packages.
b) Extensions - Loads jar files from JDK extensions directory - usually lib/ext directory of the JRE
c) System - Loads classes from system classpath.

60. How are classes loaded by JVM?

⇨ Class loaders are hierarchical. The very first class is specially loaded with the help of static main () method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running.

61. Difference between static vs. dynamic class loading?

⇨ static loading - Classes are statically loaded with Java’s “new” operator.
dynamic class loading - Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time.
Class.forName (Test className);

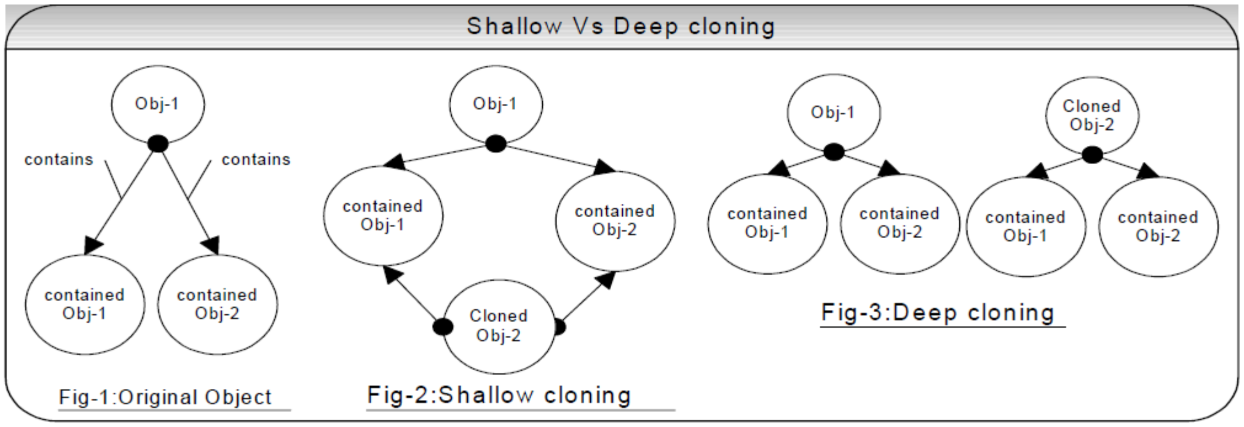
62. What is the difference between final, finally and finalize() in Java?

a) final - constant declaration. Refer Q27 in Java section.
b) finally - handles exception. The finally block is optional and provides a mechanism to clean up regardless of what happens within the try block (except System.exit(0) call). Use the finally block to close files or to release other system resources like database connections, statements etc. (Refer Q45 in Enterprise section)
c) finalize() - method helps in garbage collection. A method that is invoked before an object is discarded by the garbage collector, allowing it to clean up its state. Should not be used to release non-memory resources like file handles, sockets, database connections etc because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these non-memory resources through the finalize() method.

63. What is the main difference between shallow cloning and deep cloning of objects?

⇨ The default behavior of an object’s clone() method automatically yields a shallow copy. So to achieve a deep copy the classes must be edited or adjusted.
Shallow copy: If a shallow copy is performed on obj-1 as shown in fig-2 then it is copied but its contained objects are not. The contained objects Obj-1 and Obj-2 are affected by changes to cloned Obj-2. Java supports shallow cloning of objects by default when a class implements the java.lang.Cloneable interface.
Deep copy: If a deep copy is performed on obj-1 as shown in fig-3 then not only obj-1 has been copied but the objects contained within it have been copied as well. Serialization can be used to achieve deep cloning. Deep

cloning through serialization is faster to develop and easier to maintain but carries a performance overhead.



64. What is Method References in Java 8

[Method references]()

65. How to Implement Stack in Java using Array and Generics

http://javarevisited.blogspot.com/2017/03/how-to-implement-stack-in-java-using-array-example.html#ixzz4bn4PKOGI

66. Java Lambda Expressions Basics

https://dzone.com/articles/java-lambda-expressions-basics
http://viralpatel.net/blogs/lambda-expressions-java-tutorial/

67. Difference between Java 6 and Java 7

Java 6	Java 7
Support for older win9x versions dropped.	Upgrade class-loader architecture: A method that frees the underlying resources, such as open files, held by a URLClassLoader
Scripting lang support: Generic API for integration with scripting languages, & built-in mozilla javascript rhino integration	Concurrency and collections updates: A lightweight fork/join framework, flexible and reusable synchronization barriers, transfer queues, concurrent linked double-ended queues, and thread-local pseudo-random number generators.
Dramatic performance improvements for the core platform, and swing.	Internationalization Upgrade: Upgrade on Unicode 6.0, Locale enhancement and Separate user locale and user-interface locale.
Improved web service support through JAX-WS JDBC 4.0 support	More new I/O APIs for the Java platform (NIO.2), NIO.2 file system provider for zip/jar archives, SCTP, SDP, TLS 1.2 support.
Java compiler API: an API allowing a java program to select and invoke a java compiler programmatically.	Security & Cryptography implemented Elliptic-curve cryptography (ECC)

Upgrade of JAXB to version 2.0: including integration of a stax parser	Upgrade to JDBC 4.1 and Rowset 1.1.
Support for pluggable annotations	XRender pipeline for Java 2D, Create new platform APIs for 6u10 graphics features, Nimbus look-and-feel for Swing, Swing JLayer component, Gervill sound synthesizer.
Many GUI improvements, such as integration of swing worker in the API, table sorting and filtering, and true swing double-buffering (eliminating the gray-area effect).	Upgrade the components of the XML stack to the most recent stable versions: JAXP 1.4, JAXB 2.2a, and JAX-WS 2.2.
	Enhanced MBeans." Support for dynamically-typed languages (InvokeDynamic): Extensions to the JVM, the Java language, and the Java SE API to support the implementation of dynamically-typed languages at performance levels near to that of the Java language itself
	Strict class-file checking: Class files of version 51 (SE 7) or later must be verified with the typechecking verifier; the VM must not fail over to the old inferencing verifier.
	Small language enhancements (Project Coin): A set of small language changes intended to simplify common, day-to-day programming tasks: Strings in switch statements, try-with-resources statements, improved type inference for generic instance creation ("diamond"), simplified var args method invocation, better integral literals, and improved exception handling (multi-catch).

68. **Difference between Java 8 and Java 9**

Java 8	Java 9
JSR 335, JEP 126: Language-level support for lambda expressions.	A lightweight JSON API for consuming and generating JSON documents and data streams.
JSR 223, JEP 174: Project Nashorn, a JavaScript runtime which allows developers to embed JavaScript code within applications.	A HTTP 2 Client that will bring HTTP 2.0 and web sockets, while replacing the legacy HttpURLConnection.
JSR 308, JEP 104: Annotation on Java Types.	Process API Updates to improve controlling and managing operating-system process (developers were often forced to use native code with the current API). Along with several other smaller features, as well as dozens of proposals already being tracked by the JEP Index, Oracle has also promised another trio of performance features

Unsigned Integer Arithmetic.	Improve contended locking, which aims at improving performance when threads compete over access to objects.
JSR 337, JEP 120: Repeating annotations.	Segmented code cache with better performance, shorter sweep times, less fragmentation and further extensions to come.
JSR 310, JEP 150: Date and Time API.	The Smart Java compiler, or sjavac, will be improved to allow default use in the JDK build and general use for building larger projects.
JEP 178: Statically-linked JNI libraries.	
JEP 153: Launch JavaFX applications (direct launching of JavaFX application JARs).	
JEP 122: Remove the permanent generation.	
Java 8 is not supported on Windows XP. But as of JDK 8 update 5, it still can run under Windows XP after forced installation by directly unzipping from the installation executable.	

Java Collections: -

1. **How HashMap works in Java**

⇨ HashMap works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time by providing the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put (key, value) method of HashMap and retrieved by calling get(key) method. When we call put method, hashcode () method on the key object is called so that hash function of the map can find a bucket location to store value object which is an index of the internal array known as the table. HashMap internally stores mapping in the form of Map.Entry object which contains both key and value object. When you want to retrieve the object, you call the get () method and again pass the key object. This time again key object generates same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object, then it is returned and that's your value object which you have stored earlier.

Do you Know how HashMap works in Java or How does get () method of HashMap works in Java?
HashMap works on the principle of hashing we have put (key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object to put () method on Java HashMap. HashMap implementation calls hashCode method on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object. Important point to mention is that HashMap in Java stores both key and value object as Map.Entry in a bucket which is essential to understand the retrieving logic.

What will happen if two different objects have the same hashcode?
About equals () and hashCode () contract that two unequal objects in Java can have same hashcode. Since hashcode is same bucket location would be same and collision will occur in HashMap Since HashMap uses LinkedList to store object, this entry (object of Map.Entry comprise key and value) will be stored in LinkedList.

How will you retrieve Value object if two Keys will have the same hashcode?
We will call get () method and then HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but there are two Value objects are stored in same bucket. HashMap stores both Key and Value in

LinkedList node or as Map.Entry. After finding bucket location, we will call keys.equals () method to identify a correct node in LinkedList and return associated value object for that key in Java HashMap. Using immutable, final object with proper equals () and hashcode() implementation would act as perfect Java HashMap keys and improve the performance of Java HashMap by reducing collision. Immutability also allows caching their hashcode of different keys which makes overall retrieval process very fast and suggest that String and various wrapper classes e.g. Integer very good keys in Java HashMap.

What happens On HashMap in Java if the size of the HashMap exceeds a given threshold defined by load factor?
If the size of the Map exceeds a given threshold defined by load-factor e.g. if the load factor is .75 it will act to re-size the map once it filled 75%. Like other collection classes like ArrayList, Java HashMap re-size itself by creating a new bucket array of size twice of the previous size of HashMap and then start putting every old element into that new bucket array. This process is called rehashing because it also applies the hash function to find new bucket location.

Do you see any problem with resizing of HashMap in Java?
Yes, there is potential race condition exists while resizing HashMap in Java, if two thread at the same time found that now HashMap needs resizing and they both try to resizing. On the process of resizing of HashMap in Java, the element in the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap doesn't append the new element at tail instead it append new element at the head to avoid tail traversing. If race condition happens then you will end up with an infinite loop.
How-hashmap-works

2. **What is the difference between poll() and remove() method of Queue interface?**

⇨ Though both poll() and remove() method from Queue is used to remove the object and returns the head of the queue, there is a subtle difference between them. If Queue is empty then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example, PriorityQueue keeps the lowest element as per Comparator or Comparable at head position.

3. **What is fail safe and fail fast Iterator in Java?**

⇨ Java Collections supports two types of Iterators fail safe and fail fast. The main distinction between a fail-fast and fail-safe Iterator is whether the underlying collection can be modified while it does begin iterated. If you have used Collection like ArrayList then you know that when you iterate over them, no other thread should modify the collection. If Iterator detects any structural change after iteration has begun e.g adding or removing a new element then it throws ConcurrentModificationException, this is known as fail-fast behavior and these iterators are called fail-fast iterator because they fail as soon as they detect any modification. Though it's not necessary that iterator will throw this exception when multiple threads modified it simultaneously. It can happen even with the single thread when you try to remove elements by using ArrayList's remove() method instead of Iterator's remove method.
Most of the Collection classes from Java 1.4 e.g. Vector, ArrayList, HashMap, HashSet has fail-fast iterators. The other type of iterator was introduced in Java 1.5 when a concurrent collection class e.g. ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet was introduced. This iterator uses a view of original collection for doing iteration and that's why they don't throw ConcurrentModificationException even when original collection was modified after iteration has begun. This means you could iterate and work with stale value but this is the cost you need to pay for fail-safe iterator

4. **Difference between Fail Safe and Fail Fast Iterator in Java.**
   a) Fail-fast Iterator throws ConcurrentModfiicationException as soon as they detect any structural change in collection during iteration basically which changes the modCount variable hold by Iterator. While fail-safe iterator doesn't throw CME.

b) Fail-fast iterator traverse over original collection class while fail-safe iterator traverse over a copy or view of original collection. That's why they don't detect any change on original collection classes and this also means that you could operate with stale value.
c) Iterators from Java 1.4 Collection classes e.g. ArrayList, HashSet and Vector are fail-fast while Iterators returned by concurrent collection classes e.g. CopyOnWriteArrayList or CopyOnWriteArraySet are fail-safe.
d) Iterator returned by synchronized Collection is fail-fast while iterator returned by concurrent collections is fail-safe in Java.
e) Fail fast iterator works in live data but become invalid when data is modified while fail-safe iterator are weekly consistent.

<u>When to use fail fast and fail-safe Iterator</u>
Use fail-safe iterator when you are not bothered about Collection to be modified during iteration as fail-fast iterator will not allow that. Unfortunate you can't choose fail safe or fail-fast iterator, it depends on upon which Collection class you are using. Most of the JDK 1.4 Collections e.g. HashSet, Vector, ArrayList has fail-fast Iterator and only Concurrent Collections introduced in JDK 1.5 e.g. CopyOnWriteArrayList and CopyOnWriteArraySet supports fail safe Iteration. Also, if you want to remove elements during iteration please use iterator's remove () method and don't use remove method provided by Collection classes e.g. ArrayList or HashSet because that will result in ConcurrentModificationException.

5. **How do you remove an entry from a Collection? And subsequently what is the difference between the remove() method of Collection and remove() method of Iterator, which one you will use while removing elements during iteration?**

⇨ Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these methods to remove an entry from Collection while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

6. Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

⇨ Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is a significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself which uses synchronized methods to achieve thread-safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. The only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create its synchronized version by using Collections.synchronizedMap() method.
On the other hand, ConcurrentHashMap is specially designed for concurrent use i.e. more than one thread. By default, it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in the internal implementation of ConcurrentHashMap class. Unlike Hashtable and Synchronized Map, it never locks whole Map instead it divides the map into segments and locking is done on those. Though it performs better if number of reader threads are greater than the number of writer threads.

7. Why need ConcurrentHashMap and CopyOnWriteArrayList

⇨ The synchronized collections classes Hashtable and Vector and the synchronized wrapper classes Collections.synchronizedMap() and Collections.synchronizedList() provide a basic conditionally thread-safe implementation of Map and List. However several factors make them unsuitable for use in highly concurrent applications for example, their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent ConcurrentModificationException. ConcurrentHashMap and CopyOnWriteArrayList implementations provide much higher concurrency while preserving thread safety with some minor compromises in their promises to callers. ConcurrentHashMap and CopyOnWriteArrayList are not necessarily useful everywhere you might use HashMap or ArrayList but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.

8. Difference between ConcurrentHashMap and Hashtable

⇨ Both can be used in the multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it must be locked for a longer duration. Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete. In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration.

9. The difference between ConcurrentHashMap and Collections.synchronizedMap

⇨ ConcurrentHashMap is designed for concurrency and improve performance while HashMap which is non-synchronized by nature can be synchronized by applying a wrapper using synchronized Map. ConcurrentHashMap does not allow null keys or null values while synchronized HashMap allows one null key.

10. How HashSet Internally Works in Java

⇨ HashSet is internally implemented using HashMap in Java. HashMap allows duplicate values and this property is exploited while implementing HashSet in Java. Since HashSet implements Set interface it needs to guarantee uniqueness and this is achieved by storing elements as keys with same value always. All you need to look at is how elements are stored in HashSet and how they are retrieved from HashSet. Since HashSet doesn't provide any direct method for retrieving object e.g. get (Key key) from HashMap or get(int index) from List, only way to get object from HashSet is via Iterator. When you create an object of HashSet in Java, it internally creates instance of backup Map with default initial capacity 16 and default load factor 0.75 as shown below:

```
/**

   * Constructs a new, empty set; the backing <tt>HashMap</tt> instance has
   * default initial capacity (16) and load factor (0.75).
   */

public HashSet() {
    map = new HashMap<>();
}
```

<u>**How Object is stored in HashSet**</u>

As you can see below, a call to add(Object) is delegate to put (Key, Value) internally, where Key is the object you have passed and value is another object called PRESENT which is a constant in java.util.HashSet as shown below :
Since PRESENT is a constant for all keys we have same value in backup HashMap called map.

```
private transient HashMap<E,Object> map;

// Dummy value to associate with an Object in the backing Map
private static final Object PRESENT = new Object();

public boolean add(E e) {
   return map.put(e, PRESENT)==null;
}
```

How Object is retrieved from HashSet

Now let's see the code for getting iterator for traversing over HashSet in Java. Iterator () method from java.util.HashSet class returns iterator for backup Map returned by map.keySet().iterator() method.

```
 /**

   * Returns an iterator over the elements in this set.  The elements
   * are returned in no particular order.
   *
   * @return an Iterator over the elements in this set
   * @see ConcurrentModificationException
   */

  public Iterator<E> iterator() {
      return map.keySet().iterator();
  }
```

How to use HashSet

Using HashSet in Java is very simple, don't think it is Map but think more like Collection i.e. add elements by using add method, check its return value to see if object already existed in HashSet or not. Similarly use iterator for retrieving element from HashSet in Java. You can also use contains method to check if any object already exists in HashSet or not. This method is using equals method for comparing object for matching. You can also use remove method to remove object from HashSet. Since element of HashSet is used as key in backup HashMap they must implement equals () and hashCode () method. Immutability is not requirement but if it's immutable then you can assume that object will not be changed during its stay on set.

11. Why you should override equals or hashcode

⇨ From the face, you can guess that equals () is used to check if two objects are equal or not. Now this equality can be defined in two ways, identity equality and logical equality, it's the logical equality, which is taken care by equals method. If you are doing Java programming then you probably know that every class in Java implicitly inherit from java.lang.Object and from there every object inherit equals() and hashcode(). There defaults implementation is in line with == operator i.e. equals () provide identity equality and return true if reference variable pointing to same object. Now if you don't need logical equality, then you don't need to override equals, but the problem is you will need it. All your domain object e.g. Order, Trade, Message can be compared to each other and you need logical comparison. One of the popular examples is java.lang.String class, which needs logical comparison i.e. character based comparison. If two String object contains same characters in same order they are considered equals, which is what you need in many programming tasks. Similarly, all domain object has equality defined, but true need of equals and hashcode arise, when you use them as key in hash based collection e.g. Hashtable or HashMap. These collection classes rely on rules of Java programming around equals and hashcode to work per their specification, popularly known as equals-hashcode contract. Per which, you must override hashcode, if you are overriding equals and

vice-versa. Problem is that this is not enforced by compiler, and if you make such mistake, your program will not work properly.
For example, any object which doesn't follows equals and hashcode contract, if used as key in HashMap, you may not be able to retrieve object again. In short, you need to override equals and hashcode, if you are writing a domain object, or you want to store them in hash based collection.

By default, Object's hashCode() method returns and integer representation of memory address where object is stored.

12. Difference between List and Set

Main difference between Set and List is that List is an ordered Collection which means List preserves the order on which an element is inserted into List. So, if you insert Object A before Object B Then A will be stored at lower index than B. Since Set is an UN-ordered collection it doesn't maintain any inserting order of element, though you can have SortedSet which offers to sort functionality on top of Set interface and you can impose either natural order or Object or any custom order by using Comparator and Comparable while storing object inside Set.
Another *significant difference between List and Set* is that List allows you to store duplicates in the collection while Set doesn't allow any duplicates. This is very significant as it clearly says that if you want a collection of unique object use Set. Duplication of Object is detected using equals () method. So if two objects are equal using equals method, the later object will replace the former in Set if added using add() method due to this reason only one null element is allowed inside Set. It's also worth noting that in the case of SortedSet like TreeSet, compareTo method is used to compare object and decide whether an object is duplicate or not. Two objects will be duplicate if their compareTo() method returns zero and that's why it's said that compareTo should be consistent with equals method in java.

13. Difference between HashMap and HashSet

HashMap	Hash Set
HashMap is an implementation of Map interface	HashSet is an implementation of Set Interface
HashMap Stores data in form of key-value pair	HashSet Store only objects
Put method is used to add element in map	Add method is used to add element in Set
In hash map hash code value is calculated using key object	Here member object is used for calculating hash code value which can be same for two objects so equal () method is used to check for equality if it returns false that means two objects are different.
HashMap is faster than HashSet because unique key is used to access object	HashSet is slower than HashMap

14. What is NavigableMap?

NavigableMap in Java 6 is an extension of SortedMap like TreeMap which provides convenient navigation method like lowerKey, floorKey, ceilingKey and higherKey. NavigableMap is added on Java 1.6 and along with these popular navigation methods it also provides ways to create a Sub Map from existing Map in Java e.g. headMap whose keys are less than specified key, tailMap whose keys are greater than specified key and a subMap which is strictly contains keys which falls between toKey and fromKey. These methods also provide a boolean to include specified key or not. TreeMap and ConcurrentSkipListMap are two concrete implementations of NavigableMap in Java 1.6 API. Though NavigableMap is not as popular as HashMap, ConcurrentHashMap or Hashtable but given that TreeMap implements NavigableMap you already get all good things in a well-known Map implementation.

15. BlockingQueue in Java – ArrayBlockingQueue vs LinkedBlockingQueue

BlockingQueue in Java is added in Java 1.5 along with various other concurrent Utility classes like ConcurrentHashMap, Counting Semaphore, CopyOnWriteArrrayList etc. BlockingQueue is a unique collection type which not only store elements but also supports flow control by introducing blocking if either BlockingQueue is full or empty. take() method of BlockingQueue will block if Queue is empty and put() method of BlockingQueue will block if Queue is full. This property makes BlockingQueue an ideal choice for implementing Producer consumer design pattern where one thread insert element into BlockingQueue and other thread consumes it.

Important properties of BlockingQueue

a) BlockingQueue in Java doesn't allow null elements, various implementation of BlockingQueue like ArrayBlockingQueue, LinkedBlockingQueue throws NullPointerException when you try to add null on queue.

```
BlockingQueue<String> bQueue = new ArrayBlockingQueue<String>(10);
//bQueue.put(null); //NullPointerException - BlockingQueue in Java doesn't allow null

bQueue = new LinkedBlockingQueue<String>();
bQueue.put(null);

Exception in thread "main" java.lang.NullPointerException
        at java.util.concurrent.LinkedBlockingQueue.put(LinkedBlockingQueue.java:288)
```

b) BlockingQueue can be bounded or unbounded. A bounded BlockingQueue is one which is initialized with initial capacity and call to put() will be blocked if BlockingQueue is full and size is equal to capacity. This bounding nature makes it ideal to use a shared queue between multiple threads like in most common Producer consumer solutions in Java. An unbounded Queue is one which is initialized without capacity, by default it initialized with Integer.MAX_VALUE. most common example of BlockingQueue uses bounded BlockingQueue as shown in below example.

```
BlockingQueue<String> bQueue = new ArrayBlockingQueue<String>(2);
bQueue.put("Java");
System.out.println("Item 1 inserted into BlockingQueue");
bQueue.put("JDK");
System.out.println("Item 2 is inserted on BlockingQueue");
bQueue.put("J2SE");
System.out.println("Done");

Output:
Item 1 inserted into BlockingQueue
Item 2 is inserted on BlockingQueue
```

This code will only insert Java and JDK into BlockingQueue and then it will block while inserting 3rd element J2SE because size of BlockingQueue is 2 here.

c) BlockingQueue implementations like ArrayBlockingQueue, LinkedBlockingQueue and PriorityBlockingQueue are thread-safe. All queuing method uses concurrency control and internal locks to perform operation atomically. Since BlockingQueue also extend Collection, bulk Collection operations like addAll(), containsAll() are not performed atomically until any BlockingQueue implementation specifically supports it. So call to addAll() may fail after inserting couple of elements.

d) Common methods of BlockingQueue are put() and take() which are blocking methods in Java and used to insert and retrieve elements from BlockingQueue in Java. put() will block if BlockingQueue is full and take() will block if BlockingQueue is empty, call to take() removes element from head of Queue as shown in following example:

```
BlockingQueue<String> bQueue = new ArrayBlockingQueue<String>(2);
bQueue.put("Java"); //insert object into BlockingQueue
System.out.println("BlockingQueue after put: " + bQueue);
bQueue.take(); //retrieve object from BlockingQueue in Java
System.out.println("BlockingQueue after take: " + bQueue);

Output:
BlockingQueue after put: [Java]
BlockingQueue after take: []
```

e) BlockingQueue interface extends Collection, Queue and Iterable interface which provides it all Collection and Queue related methods like poll(), and peak(), unlike take(), peek() method returns head of the queue without removing it, poll() also retrieves and removes elements from head but can wait till specified time if Queue is empty.

```
BlockingQueue<String> linkedBQueue = new LinkedBlockingQueue<String>(2);
linkedBQueue.put("Java"); //puts object into BlockingQueue
System.out.println("size of BlockingQueue before peek : " + linkedBQueue.size());
System.out.println("example of peek() in BlockingQueue: " + linkedBQueue.peek());
System.out.println("size of BlockingQueue after peek : " + linkedBQueue.size());
System.out.println("calling poll() on BlockingQueue: " + linkedBQueue.poll());
System.out.println("size of BlockingQueue after poll : " + linkedBQueue.size());

Output:
size of BlockingQueue before peek : 1
example of peek() in BlockingQueue: Java
size of BlockingQueue after peek : 1
calling poll() on BlockingQueue: Java
size of BlockingQueue after poll : 0
```

f) Other important methods remainingCapacity() and offer (), former returns number remaining space in BlockingQueue, which can be filled without blocking while later insert object into queue if possible and return true if success and false if fail unlike add() method which throws IllegalStateException if it fails to insert object into BlockingQueue. Use offer () over add () wherever possible.

<u>Usage of BlockingQueue in Java</u>

There can be many creative usages of BlockingQueue in Java given its flow control ability. Two of the most common ways I see programmer uses BlockingQueue is to implement Producer Consumer design pattern and implementing Bounded buffer in Java. It surprisingly made coding and inter thread communication over a shared object very easy.

ArrayBlockingQueue and LinkedBlockingQueue in Java

ArrayBlockingQueue and LinkedBlockingQueue are common implementation of BlockingQueue<E> interface. ArrayBlockingQueue is backed by array and Queue impose orders as FIFO. head of the queue is the oldest element in terms of time and tail of the queue is youngest element. ArrayBlockingQueue is also fixed size bounded buffer on the other hand LinkedBlockingQueue is an optionally bounded queue built on top of Linked nodes. In terms of throughput LinkedBlockingQueue provides higher throughput than ArrayBlockingQueue in Java.

16. **How do you Sort objects on the collection?**

Comparators and comparable in Java are two interfaces which is used to implement sorting in Java. It's often required to sort objects stored in any collection classes like ArrayList, HashSet or in Array at that time we need to use either compare() or compareTo() method defined in java.util.Comparator and java.lang.Comparable.

Comparator vs Comparable in Java

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

a) Comparator in Java is defined in java.util package while Comparable interface in java.lang package, which very much says that Comparator should be used as an utility to sort objects while Comparable should be provided by default.
b) Comparator interface in Java has method public int compare (Object o1, Object o2) which returns a negative integer, zero or a positive integer as the first argument is less than, equal to or greater than the second. While Comparable interface has method public int compareTo (Object o) which returns a negative integer, zero or a positive integer as this object is less than, equal to or greater than the specified object.
c) Logical difference between these two is Comparator in Java compare two objects provided to him, while Comparable interface compares "this" reference with the object specified.
d) Comparable in Java is used to implement natural ordering of object. In Java API String, Date and wrapper classes implements Comparable interface. It's always good practice to override compareTo () for value objects.
e) If any class implement Comparable interface in Java, then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and object will be sorted based on their natural order defined by CompareTo method.
f) Objects which implement Comparable in Java can be used as keys in a SortedMap like TreeMap or elements in a SortedSet for example TreeSet without specifying any Comparator.

How to Compare String in Java

String is immutable in Java and one of the most used value class. For comparing String in Java, we should not be worrying because String implements Comparable interface and provides a **lexicographic** implementation for CompareTo method which compare two strings based on contents of characters or you can say in lexical order. You just need to call String.compareTo(AnotherString) and Java will determine whether specified String is greater than, equal to or less than current object.
See 4 example to compare String in Java for alternatives ways of comparing String.
http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html

How to Compare Dates in Java

Dates are represented by java.util.Date class in Java and like String, Date also implements Comparable in Java so they will be automatically sorted based on their natural ordering if they got stored in any sorted collection like TreeSet or TreeMap. If you explicitly want to compare two dates in Java you can call Date.compareTo(AnotherDate) method in Java and it will tell whether specified date is greater than, equal to or less than current String.
See 3 ways to compare Dates in Java for more alternatives of comparing two dates.
http://javarevisited.blogspot.sg/2012/02/3-example-to-compare-two-dates-in-java.html

When to use Comparator and Comparable in Java

a. If there is a natural or default way of sorting Object already exist during development of Class than use Comparable. This is intuitive and you give the class name people should be able to guess it correctly like Strings are sorted chronically, Employee can be sorted by their Id etc. On the other hand, if an Object can be sorted on multiple ways and client is specifying on which parameter sorting should take place than use Comparator interface. for example, Employee can again be sorted on name, salary or department and client's needs an API to do that. Comparator implementation can sort out this problem.
b. Some time you write code to sort object of a class for which you are not the original author or you don't have access to code. In these cases, you cannot implement Comparable so Comparator is only way to sort those objects.
c. Beware with the fact that How those objects will behave if stored in SorteSet or SortedMap like TreeSet and TreeMap. If an object doesn't implement Comparable than while putting them into SortedMap always provided corresponding Comparator which can provide sorting logic.
d. Order of comparison is very important while implementing Comparable or Comparator interface. for example, if you are sorting object based upon name than you can compare first name or last name on any order so decide it judiciously.
e. Comparator has a distinct advantage of being self-descriptive for example if you are writing Comparator to compare two Employees based upon their salary than name that comparator as SalaryComparator.

17. Difference between Array vs ArrayList

a) First and Major difference between Array and ArrayList in Java is that Array is a fixed length data structure while ArrayList is a variable length Collection class. You cannot change length of Array once created in Java but ArrayList re-size itself when gets full depending upon capacity and load factor. Since ArrayList is internally backed by Array in Java, any resize operation in ArrayList will slow down performance as it involves creating new Array and copying content from old array to new array.
b) Another difference between Array and ArrayList in Java is that you cannot use Generics along with Array, as Array instance knows about what kind of type it can hold and throws ArrayStoreException, if you try to store type which is not convertible into type of Array. ArrayList allows you to use Generics to ensure type-safety.
c) All kinds of Array provide length variable which denotes length of Array while ArrayList provides size () method to calculate size of ArrayList in Java.

d) One more major difference between ArrayList and Array is that, you cannot store primitives in ArrayList, it can only contain Objects. While Array can contain both primitives and Objects in Java. Though Autoboxing of Java 5 may give you an impression of storing primitives in ArrayList, it automatically converts primitives to Object. e.g.

```
ArrayList<Integer> integerList = new ArrayList<Integer>();

integerList.add(1); //here we are not storing primitive in ArrayList, instead autoboxing will

convert int primitive to Integer object
```

e) Java provides add() method to insert element into ArrayList and you can simply use assignment operator to store element into Array e.g. To store Object to specified position use

```
Object[] objArray = new Object[10];

objArray[1] = new Object();
```

f) You can create instance of ArrayList without specifying size, Java will create Array List with default size but its mandatory to provide size of Array while creating either directly or indirectly by initializing Array while creating it. By the way you can also initialize ArrayList while creating it.

18. Can we replace Hashtable with ConcurrentHashMap?

Yes, we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap but you need to be careful with code which relies on locking behavior of Hashtable. Since Hashtable locks whole Map instead of a portion of Map, compound operations like if(Hashtable.get(key) == null) put (key, value) works in Hashtable but not in concurrentHashMap. instead of this use putIfAbsent () method of ConcurrentHashMap

19. What is CopyOnWriteArrayList, how it is different than ArrayList and Vector?

CopyOnWriteArrayList is new List implementation introduced in Java 1.5 which provides better concurrent access than Synchronized List. Better concurrency is achieved by Copying ArrayList over each write and replace with original instead of locking. Also, CopyOnWriteArrayList doesn't throw any ConcurrentModificationException. It's different than ArrayList because its thread-safe and ArrayList is not thread-safe and it's different than Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

20. **What is an Iterator?**

a) The Iterator interface is used to step through the elements of a Collection.
b) Iterators let you process each element of a Collection.
c) Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
d) Iterator is an Interface implemented a different way for every Collection.

21. **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:

a) Obtain an iterator to the start of the collection by calling the collections ***iterator ()*** method.
b) Set up a loop that makes a call to ***hasNext ()***. Have the loop iterate if ***hasNext ()*** returns **true**.
c) Within the loop, obtain each element by calling **next ()**.

22. **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.

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

24. **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

25. **What is the List interface**?
    a) The List interface provides support for ordered collections of objects.
    b) Lists may contain duplicate elements.

26. **What are the main implementations of the List interface**?

⇨ The main implementations of the List interface are as follows:
   a) **ArrayList**: Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
   b) **Vector**: Synchronized resizable-array implementation of the List interface with additional legacy methods.
   c) **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.

27. **What are the advantages of ArrayList over arrays?**

⇨ Some of the advantages ArrayList has over arrays are:
   a) It can grow dynamically
   b) It provides more powerful insertion and search mechanisms than arrays.

28. **Difference between ArrayList and Vector**?

ArrayList	**Vector**
ArrayL ist 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.

29. **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(“A”)

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

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

a) ArrayList internally uses an 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.

b) 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.

31. **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.

32. **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.

33. **What is the Set interface?**
    a) The Set interface provides methods for accessing the elements of a finite mathematical set
    b) Sets do not allow duplicate elements
    c) Contains no methods other than those inherited from Collection
    d) It adds the restriction that duplicate elements are prohibited
    e) Two Set objects are equal if they contain the same elements

34. **What are the main Implementations of the Set interface**?

⇨ The main implementations of the Set interface are as follows:
    a) HashSet
    b) TreeSet
    c) LinkedHashSet
    d) EnumSet

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

36. **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.

37. **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.

38. **Difference between HashSet and TreeSet**?

HashSet	TreeSet
It does not guarantee for either sorted order or sequence order.	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.

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

40. **What are the main Implementations of the Map interface**?

⇨ The main implementations of the Map interface are as follows:
    a) HashMap
    b) HashTable
    c) TreeMap
    d) EnumMap

41. **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.

42. **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.

43. **Difference between HashMap and Hashtable**?

HashMap	**Hashtable**
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

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.

44. **How does a TreeMap 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.

45. **What are the different Collection Views That Maps Provide?**

⇨ Maps provide three collection views.
   a) **Key Set** - allow a map's contents to be viewed as a set of keys.
   b) **Values Collection** - allow a map's contents to be viewed as a set of values.
   c) **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

46. **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.

47. **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.

48. **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.

49. **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).

50. **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)

a) If object1 < object2: The value of i returned will be negative.
b) If object1 > object2: The value of i returned will be positive.
c) If object1 = object2: The value of i returned will be zero.

Generics: -

[Refer generics Tutorial]

Exception: -

1. **What is an exception?**

⇨ An exception is an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

2. **What is error?**

⇨ An Error indicates that a non-recoverable condition has occurred that should not be caught. Error is subclass of Throwable is intended for drastic problems such as OutOfMemoryError which would be reported by the JVM itself.

3. **What is the difference between error and exception in java?**

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory. Whereas exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

4. **Which is superclass of Exception?**

⇨ **"Throwable"** the parent class of all exception related classes.

5. **What are the advantages of using exception handling?**

⇨ Exception handling provides the following advantages over "traditional" error management techniques:

a) Separating Error Handling Code from "Regular" Code.
b) Propagating Errors up the Call Stack.
c) Grouping Error Types and Error Differentiation.

6. **What are the types of Exceptions in Java?**

⇨ There are two types of exceptions in Java, unchecked exceptions and checked exceptions.

a) **Checked exceptions:** A checked exception is some subclass of Exception or Exception itself excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception.

b) **<u>Unchecked exceptions:</u>** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.

7. Why Errors are Not Checked?

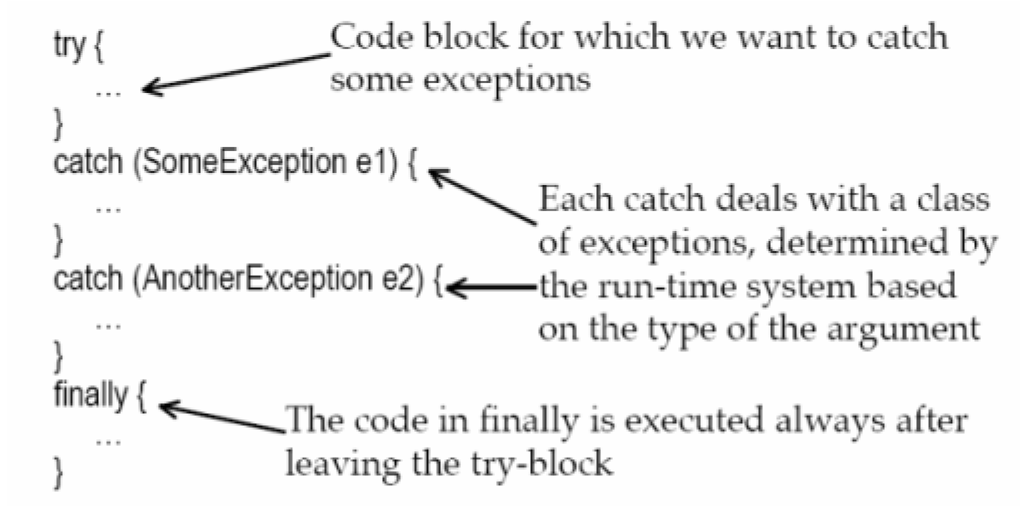
⇨ An unchecked exception classes which are the error classes are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A program declaring such exceptions would be pointlessly.

8. Why Runtime Exceptions are Not Checked?

⇨ The runtime exception classes (RuntimeException and its subclasses) are exempted from compile-time checking because in the judgment of the designers of the Java programming language, having to declare such exceptions would not aid significantly in establishing the correctness of programs. Many of the operations and constructs of the Java programming language can result in runtime exceptions. The information available to a compiler and the level of analysis the compiler performs, are usually not sufficient to establish that such run-time exceptions cannot occur even though this may be obvious to the programmer. Requiring such exception classes to be declared would simply be an irritation to programmers.

9. Explain the significance of try-catch blocks?

⇨ Whenever the exception occurs in Java we need a way to tell the JVM what code to execute. We use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.



```
try {
    ...
}
catch (SomeException e1) {
    ...
}
catch (AnotherException e2) {
    ...
}
finally {
    ...
}
```

Code block for which we want to catch some exceptions
Each catch deals with a class of exceptions, determined by the run-time system based on the type of the argument
The code in finally is executed always after leaving the try-block

10. **What is the use of finally block?**

⇨ The finally block encloses code that is always executed at some point after the try block whether an exception was thrown or not. This is right place to close files, release your network sockets, connections and perform any other cleanup your code requires.

Note: If the try block executes with no exceptions the finally block is executed immediately after the try block completes. If there was an exception thrown the finally block executes immediately after the proper catch block completes.

11. **What if there is a break or return statement in try block followed by finally block**?

⇨ If there is a return statement in the try block the finally block executes right after the return statement encountered and before the return executes.

12. Can we have the try block without catch block?

⇨ Yes, we can have the try block without catch block but finally block should follow the try block.
Note: It is not valid to use a try clause without either a catch clause or a finally clause.

13. What is the difference throw and throws?

⇨ **Throws:** Used in a method's signature if a method can cause an exception that it does not handle so that callers of the method can guard themselves against that exception. If a method is declared as throwing a class of exceptions, then any other method that calls it must either have a try-catch clause to handle that exception or must be declared to throw that exception or its superclass itself.

A method that does not handle an exception it throws must announce this:

```
 public void myfunc(int arg) throws MyException {
      …
   }
```

Throw: Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw a user-defined exception within a block, we use the throw command:

```
throw new MyException("I always wanted to throw an exception!");
```

14. How to create custom exceptions?

⇨ By extending the Exception class or one of its subclasses.
Example:

```
class MyException extends Exception {
 public MyException() { super(); }
 public MyException(String s) { super(s); }
 }
```

15. What are the different ways to handle exceptions?

⇨ There are two ways to handle exceptions:

a) Wrapping the desired code in a try block followed by a catch block to catch the exceptions.
b) List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

16. Difference between Checked and Unchecked Exceptions

a) Unchecked Exception: -

The exceptions that are not checked at compile time are called unchecked exceptions. Classes that extends RuntimeException comes under unchecked exceptions. Examples of some unchecked exceptions are listed below.

i. ArithmeticException

Mathematical operations that are not permitted in normal conditions i.e. dividing a number from ‘0’.

```
package com.beingjavaguys.core;
 public class ExceptionTest {
               public static void main(String[] args) {
              int i = 10/0;
              }
}
```

Console:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:6)
```

ii. ArrayIndexOutOfBoundsException

Trying to access an index that does not exists or inserting values to wrong indexes results to ArrayIndexOutOfBound Exception at runtime.

```
package com.beingjavaguys.core
public class ExceptionTest {

public static void main(String[] args) {
        int arr[] = {'0','1','2'};
        System.out.println(arr[4]);
        }
}
```

```
Console :
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:7)
```

iii. NullPointerException

Trying to access a null object or performing operations on an object having a null value.

```
package com.beingjavaguys.core;
import java.util.ArrayList;

public class ExceptionTest {

        public static void main(String[] args) {
        String string = null;
        System.out.println(string.length());
        }
}
```

```
Console
Exception in thread "main" java.lang.NullPointerException
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:9)
```

Some common Unchecked Exceptions in Java:

Here is a list of some common Unchecked Exceptions in Java language. ArithmeticException ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, SecurityException, SystemException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException

b) CheckedExceptions: -

Exceptions that are checked at compile-time are called checked exceptions. In Exception hierarchy, all classes that extends Exception class except UncheckedException comes under checked exception category. In certain situations, Java Compiler forced the programmer to write Exception handler at compile time, the Exceptions thrown in such situation are called Checked Exception, see the example below

```
try
{
      String input = reader.readLine();
      System.out.println("You typed : "+input); // Exception prone area
 }
 catch (IOException e)
 {
        e.printStackTrace();
}
```

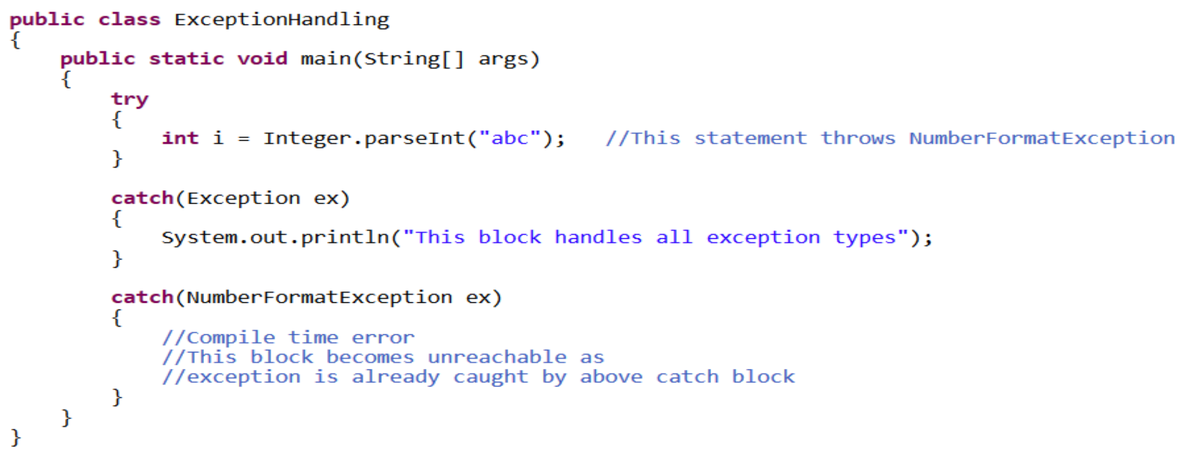
While writing a code to read or write something from files or even from or to console checked Exception i.e. IOException is thrown, these exceptions are checked at compile time and we are forced to write a handler at compile time. That's why we called these exceptions Checked Exceptions.

Some common CheckedExceptions in Java
Here is a list of some common Unchecked Exceptions in Java language. IOException FileNotFoundException, ParseException, ClassNotFoundException, CloneNotSupportedException, InstantiationException, InterruptedException, NoSuchMethodException

17. **There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?**
No. Once a try block throws an exception remaining statements will not be executed. Control comes directly to catch block.

18. **What is unreachable catch block error?**

⇨ When you are keeping, multiple catch blocks the order of catch blocks must be from most specific to most general ones. i.e sub classes of Exception must come first and super classes later. If you keep super classes first and sub classes later compiler will show unreachable catch block error.



```
public class ExceptionHandling
{
    public static void main(String[] args)
    {
        try
        {
            int i = Integer.parseInt("abc");   //This statement throws NumberFormatException
        }

        catch(Exception ex)
        {
            System.out.println("This block handles all exception types");
        }

        catch(NumberFormatException ex)
        {
            //Compile time error
            //This block becomes unreachable as
            //exception is already caught by above catch block
        }
    }
}
```

19. **What is OutOfMemoryError in java?**

⇨ OutOfMemoryError is the sub class of java.lang.Error which occurs when JVM runs out of memory.

20. What is the difference between ClassNotFoundException and NoClassDefFoundError in java?

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

21. Give the list of Java Object class methods.

a) clone() - Creates and returns a copy of this object.
b) equals() - Indicates whether some other object is "equal to" this one.
c) finalize() - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
d) getClass() - Returns the runtime class of an object.
e) hashCode() - Returns a hash code value for the object.
f) notify() - Wakes up a single thread that is waiting on this object's monitor.
g) notifyAll() - Wakes up all threads that are waiting on this object's monitor.
h) toString() - Returns a string representation of the object.
i) wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

22. What is immutable class in Java?

⇨ Immutable classes are those class whose object cannot be modified once created it means any modification on immutable object will result in another immutable object. Best example to understand immutable and mutable objects are String and StringBuffer. Since String is immutable class any change on existing string object will result in another string e.g. replacing a character into String, creating substring from String all result in a new object. While in case of mutable object like StringBuffer any modification is done on object itself and no new objects are created. Sometimes this immutability of String can also cause security hole and that the reason why password should be stored on char array instead of String.

23. How to write immutable class in Java?

⇨ Immutable object still offers several benefits in multi-threaded programming and it's a great choice to achieve thread safety in Java code. Here are few rules which helps to make a class immutable in Java:

a) State of immutable object cannot be modified after construction; any modification should result in new immutable object.
b) All fields of Immutable class should be final.
c) Object must be properly constructed i.e. object reference must not leak during construction process.
d) Object should be final to restrict sub-class for altering immutability of parent class.

By the way, you can still create immutable object by violating few rules, like String has its hashcode in non-final field, but it's always guaranteed to be same. No matter how many times you calculate it, because it's calculated from final fields, which is guaranteed to be same. This required a deep knowledge of Java memory model and can create subtle race conditions if not addressed properly. If your Immutable class has lots of optional and mandatory fields, then you can also use Builder design pattern to make a class Immutable in Java.

24. Benefits of Immutable Classes in Java?

a) Immutable objects are by default thread safe can be shared without synchronization in concurrent environment.
b) Immutable object simplifies development because it's easier to share between multiple threads without external synchronization.
c) Immutable object boost performance of Java application by reducing synchronization in code.
d) Another important benefit of Immutable objects is reusability you can cache Immutable object and reuse them much like String literals and Integers. You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache instead of creating a new one.

Apart from above advantages, immutable object has disadvantage of creating garbage as well. Since immutable object cannot be reused and they are just a use and throw. String being a prime example which can create lot of garbage and can potentially slow down application due to heavy garbage collection but again that's extreme case and if used properly Immutable object adds lot of values.

Serialization: -

1. What is Serialization in Java

⇨ Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running Java virtual machine; the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use their own custom binary format, which is often advised as Serialization best practice, because serialized binary format becomes part of Class's exported API and it can potentially break Encapsulation in Java provided by private and package-private fields.

2. **How to make a Java class Serializable?**

⇨ Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a Class Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is SerialVersionUID. If you don't explicitly declare SerialVersionUID then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than JVM will generate a different SerialVersionUID for new version of class files and when you try to load old object serialized by old version of your program you will get InvalidClassException.

3. **What is the difference between Serializable and Externalizable interface in Java?**

⇨ Externalizable provides us writeExternal() and readExternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can improve performance of application drastically.

4. **How many methods Serializable has? If no method then what is the purpose of Serializable interface?**

⇨ Serializable interface exists in java.io package and forms core of java serialization mechanism. It doesn't have any method and called Marker Interface in Java. When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

5. **What is serialVersionUID? What would happen if you don't define this?**

⇨ SerialVersionUID is an ID which is stamped on object when it get serialized usually hash code of object, you can use tool serialver to see SerialVersionUID of a serialized object. SerialVersionUID is used for version control of object. You can specify serialVersionUID in your class file also. Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for old serialized object will be different. Java serialization process relies on correct serialVersionUID for recovering state of serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch.

6. **While serializing you want some of the members not to serialize? How do you achieve it?**

⇨ If you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

7. **What will happen if one of the members in the class doesn't implement Serializable interface?**

⇨ If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a 'NotSerializableException' will be thrown at runtime and this is why I always put a Serializable Alert (comment section in my code) , one of the code comment best practices, to instruct developer to remember this fact while adding a new field in a Serializable class.

8. **If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?**

⇨ Java serialization process only continues in object hierarchy till the class is Serializable i.e. implements Serializable interface in Java and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. Once the constructor chaining will started it wouldn't be possible to stop that, hence even if classes higher in hierarchy implements Serializable interface, there constructor will be executed.

9. **Can you Customize Serialization process or can you override default Serialization process in Java?**

⇨ The answer is yes you can. We all know that for serializing an object ObjectOutputStream.writeObject (saveThisobject) is invoked and for reading object ObjectInputStream.readObject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods private to avoid being inherited, overridden or overloaded. Since only Java Virtual Machine can call private method integrity of your class will remain and Java Serialization will work as normal.

10. **Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?**

⇨ If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you cannot unimplemented an interface it's not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method.

11. **Which methods are used during Serialization and Deserialization process in java?**

⇨ Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call ObjectOutputStream.writeObject(saveThisobject) and to deserialize that object we call ObjectInputStream.readObject() method. Call to writeObject() method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an Object which needs to be casted on correct type.

12.**Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

⇨ It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it's equal to hashCode of object. by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will throw java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

13. **What are the compatible changes and incompatible changes in Java Serialization Mechanism?**

⇨ The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object***.*** *As per Java Serialization specification adding any field or method comes under* compatible change and changing class hierarchy or UN-implementing Serializable interfaces some under non compatible changes.

14. **Can we transfer a Serialized object vie network?**

⇨ *Yes you can transfer a Serialized object via network* because java serialized object remains in form of bytes which can be transmitted via network. You can also store serialized object in Disk or database as Blob.

15. **Which kind of variables is not serialized during Java Serialization?**

⇨ This question asked sometime differently but the purpose is same whether Java developer knows specifics about static and transient variable or not. Since *static variables belong to the class* and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object's serialized state. After this question, sometime interviewer asks a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those objects?

⍰ *Threads: -*

1. **What is thread pool? Why should use thread pool in Java? [Xoriant + Capgemini]**

⇨ Thread pool is a pool of already created worker thread ready to do the job. The thread pool is one of essential facility any multi-threaded server side Java application requires. If only one thread is used to process client request, then it subsequently limit how many client can access server concurrently. To support large number of clients, you may decide to use one thread per request paradigm in which each request is processed by separate Thread, but this require Thread to be created, when request arrived. Since creation of Thread is time consuming process, it delays request processing.
It also limits number of clients based upon how many thread per JVM is allowed, which is obviously a limited number. Thread pool solves this problem; it creates Thread and manages them. Instead of creating Thread and discarding them once task is done, thread-pool reuses threads in form of worker thread.
Since Thread are usually created and pooled when application starts, your server can immediately start request processing, which can further improve server's response time.

Java Thread Pool - Executor Framework

The core of this thread pool framework is Executor interface which defines an abstraction of task execution with method execute (Runnable task) and ExecutorService which extends Executor to add various life-cycle and thread pool management facilities like shutting down thread pool.

Executor framework also provides a static utility class called Executors (like Collections) which provides several static factory methods to create various type of Thread Pool implementation in Java e.g. fixed size thread pool, cached thread pool and scheduled thread pool. Runnable and Callable interface are used to represent task executed by worker thread managed in these Thread pools.

Interesting point about Executor framework is that, it is based on Producer consumer design pattern where application thread produces task and worker thread consumes or execute those task, so it also suffers with limitation of Producer consumer task like if production speed is substantially higher than consumption than you may run OutOfMemory because of queued task, of course only if your queue is unbounded

How to create fixed size thread pool using Executor framework in Java

Creating fixed size thread pool using Java 5 Executor framework is easy because of static factory methods provided by Executors class. All you need to do is define your task which you want to execute concurrently and then submit that task to ExecutorService from them Thread pool will take care of how to execute that task, it can be executed by any free worker thread and if you are interested in result you can query Future object returned by submit () method. Executor framework also provides different kind of Thread Pool e.g. SingleThreadExecutor which creates just one worker thread or CachedThreadPool which creates worker threads as and when necessary.

Benefits of Thread Pool in Java

a) Use of Thread Pool reduces response time by avoiding thread creation during request or task processing.
b) Use of Thread Pool allows you to change your execution policy as you need. You can go from single thread to multiple threads by just replacing ExecutorService implementation.
c) Thread Pool in Java application increases the stability of the system by creating a configured number of threads decided based on system load and available resource.
d) Thread Pool frees application developer from thread management stuff and allows focusing on business logic.
Thread Pool POC

2. **What is the difference between Stack and Heap in Java?**

⇨ A stack is also a data structure which is used to store elements in LIFO (Last In First out) order and available in Java API as java.util.Stack. In general, both stack and heap are part of memory; a program is allocated and used for different purposes.

Difference between Stack vs Heap in Java

a) The main difference between heap and stack is that stack memory is used to store local variables and function call while heap memory is used to store objects in Java. No matter, where the object is created in code e.g. as a member variable, local variable or class variable, they are always created inside heap space in Java.
b) Each Thread in Java has their own stack which can be specified using -Xss JVM parameter, similarly, you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of the heap and -Xmx is a maximum size of java heap.
c) If there is no memory left in the stack for storing function call or local variable, JVM will throw java. lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space.

d) If you are using Recursion, on which method calls itself, you can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is a lot lesser than the size of heap memory in Java.
e) Variables stored in stacks are only visible to the owner Thread while objects created in the heap are visible to all thread. In other words, stack memory is kind of private memory of Java Threads while heap memory is shared among all threads.

3. Difference between yield and wait method in Java

⇨ Main difference between wait and yield in Java is that wait() is used for flow control and inter thread communication while yield is used just to relinquish CPU to offer an opportunity to another thread for running. key method used to pause threads in Java, Thread.sleep(), Thread.yield() and Object.wait() methods.

a) Wait () is declared in java.lang.Object class while Yield is declared on java.lang.Thread class.
b) wait is overloaded method and has two versions of wait normal and timed wait while yield is not overloaded.
c) wait is an instance method while yield is a static method and work on current thread.
d) When a thread call wait () it releases the monitor.
e) Wait () method must be called from either synchronized block or synchronized method, there is no such requirement for Yield method.

4. Producer Consumer Design Pattern with Blocking Queue? [Sapient + Capgemini]

⇨ Producer Consumer Design pattern is a classic concurrency or threading pattern which reduces coupling between Producer and Consumer by separating Identification of work with Execution of Work. In producer consumer design pattern a shared queue is used to control the flow and this separation allows you to code producer and consumer separately. It also addresses the issue of different timing require to produce item or consuming item. By using producer consumer pattern both Producer and Consumer Thread can work with different speed. Producer consumer pattern is everywhere in real-life and depict coordination and collaboration. Like one person is preparing food (Producer) while other one is serving food (Consumer), both will use shared table for putting food plates and taking food plates. Producer which is the person preparing food will wait if table is full and Consumer (Person who is serving food) will wait if table is empty. table is a shared object here. On Java library Executor framework, itself implement Producer Consumer design pattern be separating responsibility of addition and execution of task.

<u>Benefit of Producer Consumer Pattern</u>

a) Producer Consumer Pattern simple development. you can Code Producer and Consumer independently and Concurrently, they just need to know shared object.
b) Producer doesn't need to know about who is consumer or how many consumers are there. Same is true with Consumer.
c) Producer and Consumer can work with different speed. There is no risk of Consumer consuming half-baked item. In fact, by monitoring consumer speed one can introduce more consumer for better utilization.
d) Separating producer and Consumer functionality result in more clean, readable and manageable code.

<u>Producer Consumer Problem in Multi-threading</u>

Producer-Consumer Problem is also a popular java interview question where interviewer ask to implement producer consumer design pattern so that Producer should wait if Queue or bucket is full and Consumer should wait if queue or
bucket is empty. This problem can be implemented or solved by different ways in Java, classical way is using wait and notify method to communicate between Producer and Consumer thread and blocking each of them on individual condition like full queue and empty queue. With introduction of BlockingQueue data Structure in Java 5 Its now much simpler because BlockingQueue provides this control implicitly by introducing blocking methods put () and

take (). Now you don't require to use wait and notify to communicate between Producer and Consumer. BlockingQueue put () method will block if Queue is full in case of Bounded Queue and take () will block if Queue is empty.

__**Using Blocking Queue to implement Producer Consumer Pattern**__
BlockingQueue amazingly simplifies implementation of Producer-Consumer design pattern by providing out of box support of blocking on put () and take (). Developer doesn't need to write confusing and critical piece of wait-notify code to implement communication. BlockingQueue is an interface and Java 5 provides different implantation like ArrayBlockingQueue and LinkedBlockingQueue, both implement FIFO order or elements, while ArrayLinkedQueue is bounded in nature LinkedBlockingQueue is optionally bounded.

ProducerConsumerPa
ttern.java

5. **How to create thread safe Singleton in Java**

   Read more:
   http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html#ixzz4b2QG Cahg
6. **How to use ConcurrentHashMap in Java**

7. **Difference between preemptive scheduling and time slicing?**

⇨ Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.
Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

8. **Sleep method in Java?**

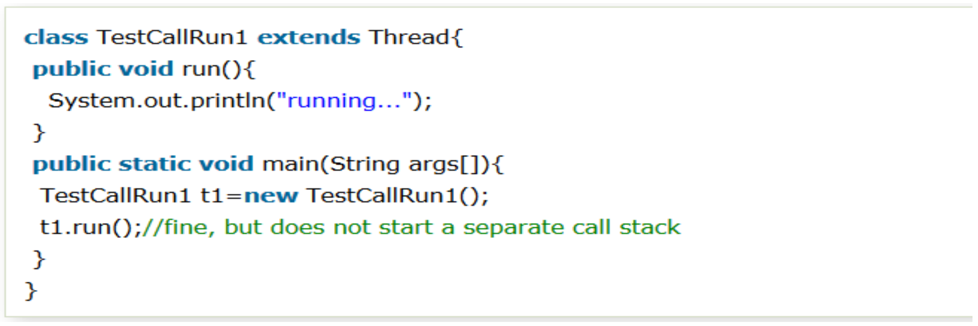
⇨ The sleep () method of Thread class is used to sleep a thread for the specified amount of time. The Thread class provides two methods for sleeping a thread:
   a) public static void sleep (long milliseconds) throws InterruptedException
   b) public static void sleep (long milliseconds, int Nanos) throws InterruptedException

   If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

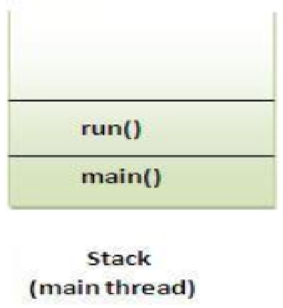
9. **Can we start a Thread twice?**

⇨ No. After starting a thread, it can never be started again. If you do so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

10. **What if we call run () method directly instead start () method?**
    a) Each thread starts in a separate call stack.
    b) Invoking the run () method from main thread, the run () method goes onto the current call stack rather than at the beginning of a new call stack.
    c) Thread will be treated as normal object not thread object.



```
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```



Problem if you direct call run() method

11. **The join () method**

⇨ The non-static join () method of class Thread lets one thread "join onto the end" of another thread. If you have a thread B that can't do its work until another thread A has completed its work, then you want thread B to "join" thread A. This means that thread B will not become runnable until A has finished and entered the dead state.

   a) public void join () throws InterruptedException
   b) public void join (long milliseconds) throws InterruptedException

12. **What is Thread in Java?**

⇨ The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads, you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 thread to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level and it's also one of the strong selling points.

13. **What is the difference between Thread and Process in Java?**

⇨ The thread is a subset of Process, in other words, one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for the different thread and used to store local data to that thread.

14. **How do you implement Thread in Java?**

⇨ At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run () method either by extending Thread class or just implementing Runnable interface.

15. **When to use Runnable vs Thread in Java? which one is better and when to use one?**

⇨ We can implement thread either by extending Thread class or implementing Runnable interface. Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. This means, it’s better to implement Runnable then extends Thread if you also want to extend another class e.g. Canvas or CommandListener.

16. **What is the difference between start () and run () method of Thread class?**

⇨ When program calls start () method a new Thread is created and code inside run () method is executed in new Thread while if you call run () method directly no new Thread is created and code inside run () will execute on current Thread.
⇨ If you want to perform time consuming task than always call start () method otherwise your main thread will have stuck while performing time consuming task if you call run () method directly.
⇨ You cannot call start () method twice on thread object. Once started, second call of start () will throw IllegalStateException in Java while you can call run () method twice.

```
public class StartVsRunCall{

    public static void main(String args[]) {

        //creating two threads for start and run method call
        Thread startThread = new Thread(new Task("start"));
        Thread runThread = new Thread(new Task("run"));

        startThread.start(); //calling start method of Thread - will execute in new Thread
        runThread.run();  //calling run method of Thread - will execute in current Thread

    }

    /*
     * Simple Runnable implementation
     */
    private static class Task implements Runnable{
        private String caller;

        public Task(String caller){
            this.caller = caller;
        }

        @Override
        public void run() {
            System.out.println("Caller: "+ caller + " and code on this Thread is executed by :
" + Thread.currentThread().getName());

        }
    }
}

Output:
Caller: start and code on this Thread is executed by : Thread-0
Caller: run and code on this Thread is executed by : main
```

17. **What is the difference between Runnable and Callable in Java?**

- ⇨ The Runnable interface is older than Callable, there from JDK 1.0, while Callable is added on Java 5.0.
- ⇨ Runnable interface has run() method to define task while Callable interface uses call() method for task definition.
- ⇨ run() method does not return any value, it's return type is void while call method returns value. The Callable interface is a generic parameterized interface and Type of value is provided when an instance of Callable implementation is created.
- ⇨ Another difference on run and call method is that run method cannot throw checked exception while call method can throw checked exception in Java

18. **What is the difference between CyclicBarrier and CountDownLatch in Java?**

⇨ **<u>What is CyclicBarrier:-</u>**

CyclicBarrier in Java is a synchronizer introduced in JDK 5 on java.util.Concurrent package along with other concurrent utility like Counting Semaphore, BlockingQueue, ConcurrentHashMap etc. CyclicBarrier is similar to CountDownLatch allows multiple threads to wait for each other (barrier) before proceeding.

CyclicBarrier is a natural requirement for a concurrent program because it can be used to perform final part of the task once individual tasks are completed. All threads which wait for each other to reach barrier are called parties, CyclicBarrier is initialized with a number of parties to wait and threads wait for each other by calling CyclicBarrier.await() method which is a blocking method in Java and blocks until all Thread or parties call await(). In general calling await() is shout out that Thread is waiting on the barrier. await() is a blocking call but can be timed out or Interrupted by other thread. If you look at CyclicBarrier is also does the same thing but there is different you cannot reuse CountDownLatch once the count reaches zero while you can reuse CyclicBarrier by calling reset () method which resets Barrier to its initial State. What it implies that CountDownLatch is a good for one-time events like application start-up time and CyclicBarrier can be used to in case of the recurrent event e.g. concurrently calculating a solution of the big problem etc.

⇨ **<u>When to use CyclicBarrier:-</u>**

Given the nature of CyclicBarrier it can be very handy to implement map reduce kind of task similar to fork-join framework of Java 7, where a big task is broken down into smaller pieces and to complete the task you need output from individual small task e.g. to count population of India you can have 4 threads which count population from North, South, East, and West and once complete they can wait for each other, When last thread completed their task, Main thread or any other thread can add result from each zone and print total population. You can use CyclicBarrier in Java:

a) To implement multi player game which cannot begin until all player has joined.
b) Perform lengthy calculation by breaking it into smaller individual tasks, In general, to implement Map reduce technique.

<u>Important point of CyclicBarrier:-</u>

a) CyclicBarrier can perform a completion task once all thread reaches to the barrier; this can be provided while creating CyclicBarrier.
b) If CyclicBarrier is initialized with 3 parties means 3 threads needs to call await method to break the barrier.
c) The thread will block on await () until all parties reach to the barrier, another thread interrupt or await timed out.
d) If another thread interrupts the thread which is waiting on barrier it will throw BrokernBarrierException
e) CyclicBarrier.reset() put Barrier on its initial state, other thread which is waiting or not yet reached barrier will terminate with java.util.concurrent.BrokenBarrierException.

19. **What is Java Memory model?**

⇨ Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behavior of concurrent Java programs. For example, happens-before relationship guarantees:

a) Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
b) An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
c) A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
d) A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.
e) A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt (either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
f) The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
g) If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

20. What is volatile variable in Java?

⇨ The volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from main memory. So if you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in an int or a Boolean variable then you can declare them as volatile variable.

21. What is thread-safety? Is Vector a thread-safe class?

⇨ Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two categories, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.

22. What is race condition in Java? Given one example?

Race condition in Java is a type of concurrency bug or issue which is introduced in your program because parallel execution of your program by multiple threads at same time, Since Java is a multi-threaded programming language hence risk of Race condition is higher in Java which demands clear understanding of what causes a race condition and how to avoid that. Anyway Race conditions are just one of hazards or risk presented by use of multi-threading in Java just like deadlock in Java. Race condition occurs when two threads operate on same object without proper synchronization and their operation interleaves on each other. Classical example of Race condition is incrementing a counter since increment is not an atomic operation and can be further divided into three steps like read, update and write. if two threads try to increment count at same time and if they read same value because of interleaving of read operation of one thread to update operation of another thread, one count will be lost when one thread overwrite increment done by other thread. Atomic operations are not subject to race conditions because those operations cannot be interleaved.

Code Example of Race Condition in Java

a) "Check and Act" race condition pattern
Classical example of "check and act" race condition in Java is getInstance() method of Singleton Class. getInstace() method first check for whether instance is null and then initialized the instance and return to caller. Whole purpose of Singleton is that getInstance () should always return same instance of Singleton. if you call getInstance() method from two thread simultaneously it's possible that while one thread is initializing singleton after null check, another thread sees value of _instance reference variable as null (quite possible in java) especially if your object takes longer time to initialize and enters into critical section which eventually results in getInstance() returning two separate instance of Singleton. This may not happen always because a fraction of delay may result in value of _instance updated in main memory. Here is a code example

```
public Singleton getInstance(){
if(_instance == null){   //race condition if two threads sees _instance= null
_instance = new Singleton();
}
}
```

An easy way to fix "check and act" race conditions is to synchronized keyword and enforce locking which will make this operation atomic and guarantees that block or method will only be executed by one thread and result of operation will be visible to all threads once synchronized blocks completed or thread exited form synchronized block.

b) read-modify-update race conditions
This is another code pattern in Java which causes race condition; classical example is the non-thread safe counter. Read-modify-update pattern also comes due to improper synchronization of non-atomic operations or combination of two individual atomic operations which is not atomic together e.g. put if absent scenario. Consider below code

```
if(!hashtable.contains(key)){
hashtable.put(key,value);
}
```

Here we only insert object into hashtable if it's not already there. Point is both contains () and put () are atomic but still this code can result in race condition since both operation together is not atomic. Consider thread T1 checks for conditions and goes inside if block now CPU is switched from T1 to thread T2 which also checks condition and goes inside if block. Now we have two threads inside if block which result in either T1 overwriting T2 value or vice-versa based on which thread has CPU for execution. In order to fix this race condition in Java you need to wrap this code inside synchronized block which makes them atomic together because no thread can go inside synchronized block if one thread is already there.

23. How to stop a thread in Java?

⇨ Earlier there was a stop method exists in Thread Class but Java deprecated that method citing some safety reason. By default, a Thread stops when execution of run() method finish either normally or due to any Exception. Thread can be stopped in Java by using a boolean State variable or flag. Using a flag to stop Thread is a very popular way of stopping the thread and it's also safe because it doesn't do anything special rather than helping run() method to finish itself.

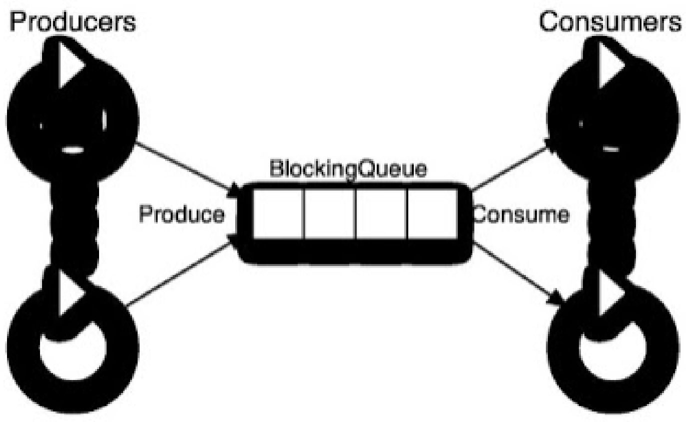
⇨ Since every Thread has its own local memory in Java its good practice to make bExit volatile because we may alter the value of bExit from any thread and make it volatile guarantees that Runner will also see any update done before making bExit.

24. What happens when an Exception occurs in a thread?

⇨ If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.

25. How do you share data between two thread in Java?

⇨ You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.



26. **What is the difference between notify and notifyAll in Java?**

⇨ **<u>Difference between notify and notifyAll in Java</u>**

Java provides two methods notify and notifyAll for waking up threads waiting on some condition and you can use any of them but there is a subtle difference between notify and notifyAll in java. When you call notify only one of waiting for the thread will be woken and it's not guaranteed which thread will be woken, it depends on upon Thread scheduler. While if you call notifyAll method, all threads waiting on that lock will be woken up, but again all woken thread will fight for lock before executing remaining code and that's why wait is called on loop because if multiple threads are woken up, the thread which will get lock will first execute and it may reset waiting for condition, which will force subsequent threads to wait. So key difference between notify and notifyAll is that notify () will cause only one thread to wake up while notifyAll method will make all thread to wake up.

<u>When to use notify and notifyAll in Java</u>

You can use notify over notifyAll if all thread are waiting for the same condition and only one Thread at a time can benefit from condition becoming true. In this case, notify is optimized call over notifyAll because waking up all of them because we know that only one thread will benefit and all other will wait again, so calling notifyAll method is just waste of CPU cycles. Though this looks quite reasonable there is still a caveat that unintended recipient swallowing critical notification. By using notifyAll we ensure that all recipient will get notify.

Which thread will be notified if I use notify ()?
No guaranteed, ThreadScheduler will pick a random thread from a pool of waiting for a thread on that monitor. What is guaranteed is that only one Thread will be notified.

How do I know how many threads are waiting, so that I can use notifyAll() ?
It depends upon your application logic while coding you needs to think whether a piece of code can be run by multiple threads or not. A good example to understand inter-thread communication is implementing the producer-consumer pattern in Java.

How to call notify()?
Wait() and notify() method can only be called from synchronized method or block, you need to call notify method on an object on which other threads are waiting.

What are these thread waiting for being notified etc?
Thread wait on some condition e.g. in the producer-consumer problem, producer thread wait if shared queue is full and consumer thread wait if shared queue is empty. Since multiple threads are working with a shared resource they communicate with each other using wait and notify method.

27. **Why wait, notify and notifyAll are not inside thread class?**

⇨ Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it makes sense to call wait () on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it makes sense to define it on object class because lock belongs to object.

28. **What is the difference between the interrupted () and isInterrupted() method in Java?**

⇨ Main difference between interrupted () and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the static method Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt

29. **Why wait and notify method are called from synchronized block?**

⇨ Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A subtler reason is to avoid the race condition between wait and notify calls.

30. **Why should you check condition for waiting in a loop?**

⇨ It’s possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.

31. **What is the difference between synchronized and concurrent collection in Java?**

⇨ Though both Synchronized and Concurrent Collection classes provide thread-safety, the differences between them come in performance, scalability and how they achieve thread-safety. Synchronized collections like synchronized HashMap, Hashtable, HashSet, Vector, and synchronized ArrayList are much slower than their concurrent counterparts e.g. ConcurrentHashMap, CopyOnWriteArrayList, and CopyOnWriteHashSet. Main reason for this slowness is locking; synchronized collections locks the whole collection e.g. whole Map or List while concurrent collection never locks the whole Map or List. They achieve thread safety by using advanced and sophisticated techniques like lock stripping. For example, the ConcurrentHashMap divides the whole map into several segments and locks only the relevant segments, which allows multiple threads to access other segments of same ConcurrentHashMap without locking.
Similarly, CopyOnWriteArrayList allows multiple reader threads to read without synchronization and when a write happens it copies the whole ArrayList and swap with a newer one.
So if you use concurrent collection classes in their favorable conditions e.g. for more reading and fewer updates, they are much more scalable than synchronized collections.

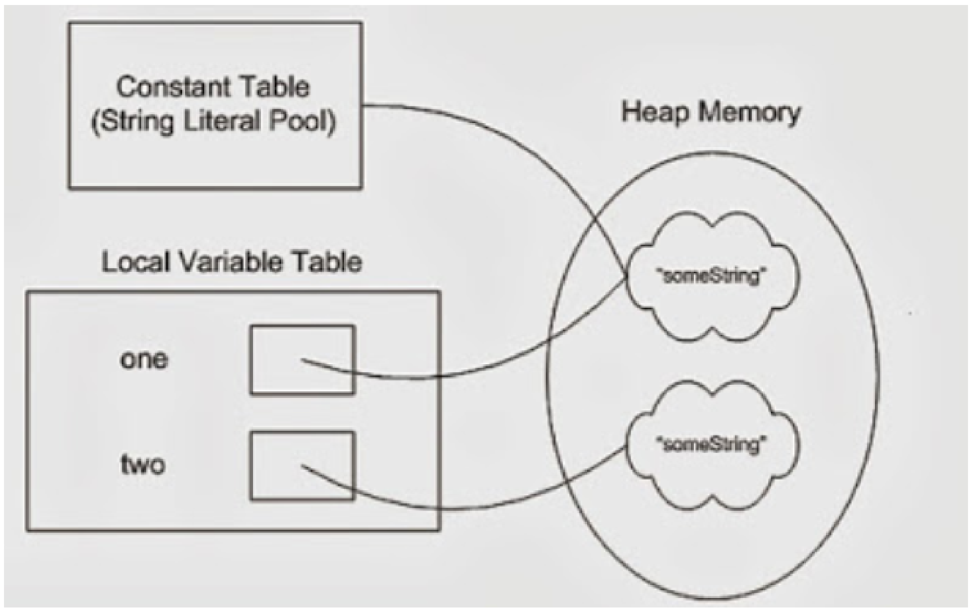
Strings: -

1. **Why String is Immutable or Final in Java**

⇨ The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time String was made final so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since HashMap works in the principle of hashing, which requires same hash value to function properly. Mutable String would produce two different hashcode at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.

<u>Why String is Final in Java</u>

a) Imagine String pool facility without making string immutable, it’s not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many reference variables, so if any one of them change the value others will be automatically gets affected i.e. let’s say
String A = "Test"
String B = "Test"
Now String B called, "Test".toUpperCase() which change the same object into "TEST", so A will also be "TEST" which is not desirable. Here is a nice diagram which shows how String literals are created in heap memory and String literal pool.



b) String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string, you can pass database URL as a string for opening database connection, you can open any file in Java by passing the name of the file as argument to File I/O classes. In case, if String is not immutable, this would lead serious security threat, I mean someone can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access to that file. Because of immutability, you don't need to worry about that kind of threats. This reason also gels with, Why String is final in Java, by making java.lang.String final, Java designer ensured that no one overrides any behavior of String class.

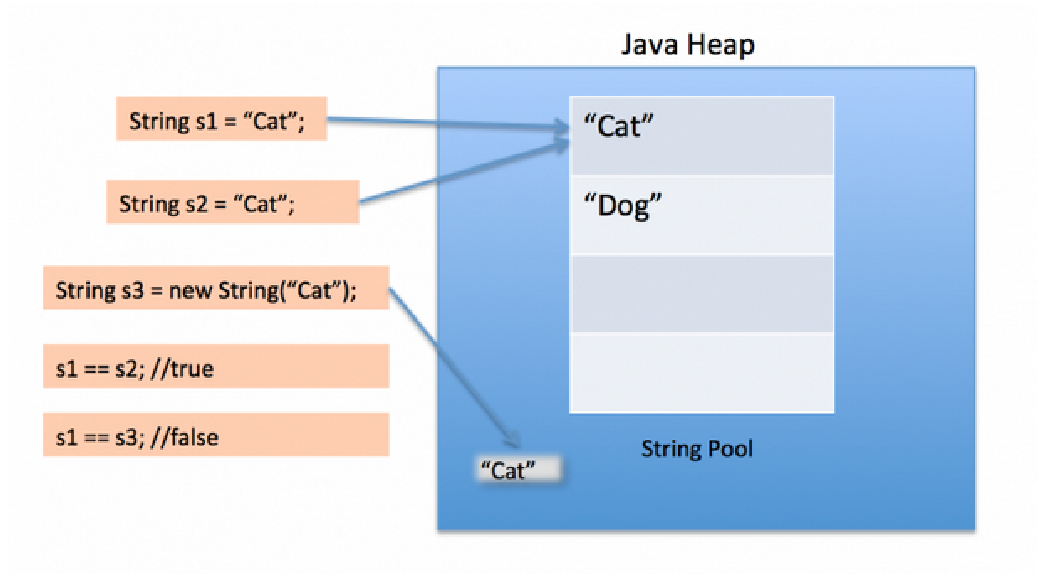
c) Since String is immutable it can safely share between many threads which is very important for multithreaded programming and to avoid any synchronization issues in Java. Immutability also makes String instance thread-safe in Java, means you don't need to synchronize String operation externally. Another important point to note about String is the memory leak caused by SubString, which is not a thread related issues but something to be aware of.

d) Java is to allow String to cache its hashcode, being immutable String, Java caches its hashcode and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in Java. String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocations.

e) String is immutable is that it is used by the class loading mechanism and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"

2. What is Java String Pool?

⇨ String Pool in java is a pool of Strings stored in Java Heap Memory. We know that String is special class in java and we can create String object using new operator as well as providing values in double quotes.
Here is a diagram which clearly explains how String Pool is maintained in java heap space and what happens when we use different ways to create Strings.



String Pool is possible only because String is immutable in Java and its implementation of String interning concept. String pool is also example of Flyweight design pattern. String pool helps in saving a lot of space for Java Runtime although it takes more time to create the String. When we use double quotes to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference. However, using new operator, we force String class to create a new String object in heap space. We can use intern() method to put it into the pool or refer to other String object from string pool having same value.

```
String str = new String("Cat");
```

In above statement, either 1 or 2 string will be created. If there is already a string literal “Cat” in the pool, then only one string “str” will be created in the pool. If there is no string literal “Cat” in the pool, then it will be first created in the pool and then in the heap space, so total 2 string objects will be created. String objects created with the new operator do not refer to objects in the string pool.

3. Which one will you prefer among “==” and equals () method to compare two string objects?

⇨ I prefer equals () method because it compares two string objects based on their content. That provides more logical comparison of two string objects. If you use “==” operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So rather stick to equals () method to compare two string objects.

4. How do you create mutable string objects?

⇨ Using StringBuffer and StringBuilder classes. These classes provide mutable string objects.

5. How many objects will be created in the following code and where they will be stored?

⇨ String s1 = new
String("abc");
String s2 = "abc";

Here, two string objects will be created. Object created using new operator(s1) will be stored in the heap memory. The object created using string literal(s2) is stored in the string constant pool.

6. Where exactly string, constant pool is in the memory?

⇨ Inside the heap memory. JVM reserves some part of the heap memory to store string objects created using string literals.

7. What is string intern?

⇨ String object in the string constant pool is called as String Intern. You can create an exact copy of heap memory string object in string constant pool. This process of creating an exact copy of heap memory string object in the string constant pool is called interning. Intern () method is used for interning.

8. Difference between String and StringBuffer

String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
String class overrides the equals () method of Object class. So, you can compare the contents of two strings by equals () method.	StringBuffer class doesn't override the equals () method of Object class.

Examples:
http://javarevisited.blogspot.com/2015/01/top-20-string-coding-interview-question-programming-interview.html

Wrapper Classes: -

1. What are Wrapper Classes? Describe the wrapper classes in Java?

Java is an object-oriented language and as said everything in java is an object. But what about the primitives? They are sort of left out in the world of objects, that is, they cannot participate in the object activities, such as being returned from a method as an object, and being added to a Collection of objects, etc. As a solution to this problem, Java allows you to include the primitives in the family of objects by using what are called wrapper classes.

Following table lists the primitive types and the corresponding wrapper classes:

Primitive	Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

2. Creating Wrapper Objects with the new Operator?

Before we can instantiate a wrapper class, we need to know its name and the arguments its constructor accepts. The name of the wrapper class corresponding to each primitive data type, along with the arguments its constructor accepts, is listed below:

Primitive datatype–>Wrapper Class–>Constructor arguments

boolean–> Boolean–> boolean or String
byte–> Byte–> byte or String
char–> Character–> char
short–> Short–> short or String
int–>Integer–> int or String
long–> Long–> long or String
float–>Float–> float double or String
double–>Double–> double or String
All the wrapper classes are declared final. That means you cannot derive a subclass from any of them. All the wrapper classes except Boolean and Character are subclasses of an abstract class called Number, whereas Boolean and Character are derived directly from the Object class. All of the wrapper classes except Character provide two constructors: one that takes a primitive of the type being constructed, and one that takes a String representation of the type being constructed

3. **AutoBoxing and UnBoxing in Java 5.0?**
   Boxing/Unboxing
   Java 5 supports automatic conversion of primitive types (int, float, double etc.) to their object equivalents (Integer, Float, Double) in assignments and method and constructor invocations. This conversion is known as autoboxing.

   Java 5 also supports automatic unboxing, where wrapper types are automatically converted into their primitive equivalents if needed for assignments or method or constructor invocations.

4. **What Design Pattern Wrapper Classes implement?**
   Adapter

Garbage Collection: -

Garbage Collection Tutorial

1. **What do you know about the Java garbage collector? When does the garbage collection occur? Explain different types of references in Java?**

⇨ Each time an object is created in Java, it goes into the area of memory known as heap. The Java heap is called the garbage collectable heap. The garbage collection cannot be forced. The garbage collector runs in low memory situations. When it runs, it releases the memory allocated by an unreachable object. The garbage collector runs on a low priority daemon (background) thread. You can nicely ask the garbage collector to collect garbage by calling System.gc() but you can't force it.

What is an unreachable object
An object's life has no meaning unless something has reference to it. If you can't reach it then you can't ask it to do anything. Then the object becomes unreachable and the garbage collector will figure it out. Java automatically collects all the unreachable objects periodically and releases the memory consumed by those unreachable objects to be used by the future reachable objects.

We can use the following options with the Java command to enable tracing for garbage collection events.
-verbose:gc reports on each garbage collection event.

Explain types of references in Java
java.lang.ref package can be used to declare soft, weak and phantom references.

a) Garbage Collector won't remove a strong reference.
b) A soft reference will only get removed if memory is low. So, it is useful for implementing caches while avoiding memory leaks.
c) A weak reference will get removed on the next garbage collection cycle. Can be used for implementing canonical maps. The java.util.WeakHashMap implements a HashMap with keys held by weak references.
d) A phantom reference will be finalized but the memory will not be reclaimed. Can be useful when you want to be notified that an object is about to be collected.

2. How does Java allocate stack and heap memory? Explain re-entrant, recursive and idempotent methods/functions?

⇨ Each time an object is created in Java it goes into the area of memory known as heap. The primitive variables like int and double are allocated in the stack, if they are local method variables and in the heap if they are member variables (i.e. fields of a class). In Java methods, local variables are pushed into stack when a method is invoked and stack pointer is decremented when a method call is completed. In a multi-threaded application, each thread will have its own stack but will share the same heap. Therefore, care should be taken in your code to avoid any concurrent access issues in the heap space. The stack is thread safe (each thread will have its own stack) but the heap is not thread safe unless guarded with synchronization through your code.

A method in stack is **re-entrant** allowing multiple concurrent invocations that do not interfere with each other.
A function is **recursive** if it calls itself. Given enough stack space, recursive method calls are perfectly valid in Java though it is tough to debug. Recursive functions are useful in removing iterations from many sorts of algorithms. All recursive functions are re-entrant but not all re-entrant functions are recursive.
Idempotent methods are methods, which are written in such a way that repeated calls to the same method with the same arguments yield same results.

3. How do you find GC resulted due to calling System.gc()?

⇨ Like major and minor collection, there will be a word "System" included in Garbage collection output.

4. Can we force Garbage collector to run at any time?

⇨ No, you cannot force Garbage collection in Java. Though you can request it by calling Sytem.gc() or its cousin Runtime.getRuntime().gc(). It's not guaranteed that GC will run immediately as result of calling these methods.

5. How to you monitor garbage collection activities?

⇨ You can monitor garbage collection activities either offline or real-time. You can use tools like JConsole and VisualVM VM with its Visual GC plug-in to monitor real time garbage collection activities and memory status of JVM or you can redirect Garbage collection output to a log file for offline analysis by using -XlogGC=&lt;PATH&gt; JVM parameter. Anyway you should always enable GC options like -XX:PrintGCDetails -X:verboseGC and -XX:PrintGCTimeStamps as it doesn't impact application performance much but provide useful states for performance monitoring.

6. Look at below Garbage collection output and answer following question:

```
[GC

        [ParNew: 1512K->64K(1512K), 0.0635032 secs]

        15604K->13569K(600345K), 0.0636056 secs]

        [Times: user=0.03 sys=0.00, real=0.06 secs]
```

a) Is this output of Major Collection or Minor Collection?
It's Minor collection because of "GC" word, In case of Major collection, you would see "Full GC".

b) Which young Generation Garbage collector is used?

This output is of multi-threaded Young Generation Garbage collector "ParNew", which is used along with CMS concurrent Garbage collector.

c) What is size of Young Generation, Old Generation and total Heap Size?
[1512K] which is written in bracket is total size of Young Generation, which include Eden and two survivor space. 1512K on left of arrow is occupancy of Yong Generation before GC and 64K is occupancy after GC. On the next line value if bracket is total heap size which is (600345K). If we subtract size of young generation to total heap size we can calculate size of Old Generation. This line also shows occupancy of heap before and after Garbage collection.

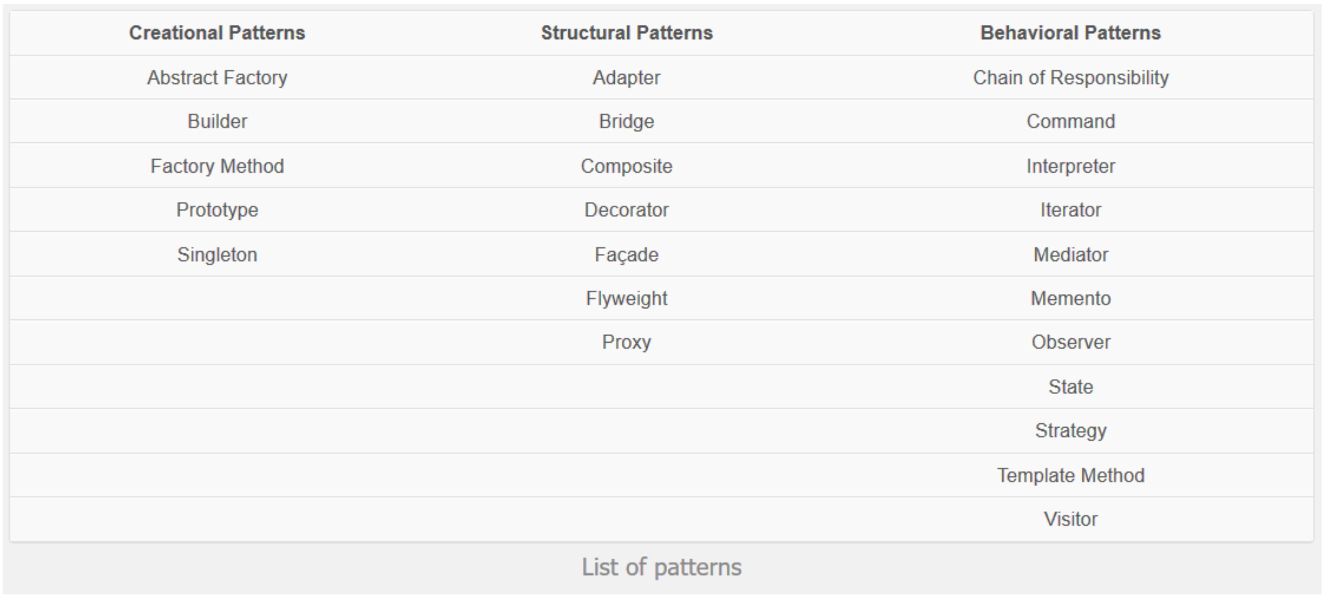
d) How much memory is freed from Garbage collection?
As answered in previous garbage collection interview question, second line shows heap occupancy before and after Garbage collection. If we subtract value of right side 13569K, to value on left side 15604K, we can get total memory freed by GC.

e) How much time is taken for Garbage collection?
0.0636056 secs on second line denotes total time it took to collect dead objects during Garbage collection. It also includes time taken to GC young generation which is shown in first line (0635032 secs).

f) What is current Occupancy of Young Generation?
64K

☐ *Design Patterns: -*

Design Patterns Tutorial



Creational Patterns	Structural Patterns	Behavioral Patterns
Abstract Factory	Adapter	Chain of Responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Façade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

List of patterns

1. When to use Adapter Pattern

⇨ The Adapter pattern should be used when:

a) There is an existing class and its interface does not match the one you need.
b) You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
c) There are several existing subclasses to be use, but it's impractical to adapt their interface by sub classing every one. An object adapter can adapt the interface of its parent class.

2. **What is Singleton pattern**
   Singleton Design Pattern Tutorial
3.

Spring Framework

1. **What is IOC (or Dependency Injection)?**

The basic concept of the Inversion of Control pattern also known as dependency injection is that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A IOC container is then responsible for hooking it all up. Applying IOC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is dependencies are injected into objects. So, IOC means an inversion of responsibility about how an object obtains references to collaborating objects.

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

There are three types of dependency injection:

a) **Constructor Injection:** Dependencies are provided as constructor parameters.
b) **Setter Injection**: Dependencies are assigned through JavaBeans properties (ex: setter methods).
c) **Interface Injection**: Injection is done through an interface.

Note: Spring supports only Constructor and Setter Injection

3. **What are Spring beans?**

The Spring Beans are Java Objects that form the backbone of a Spring application. They are instantiated, assembled, and managed by the Spring IoC container. These beans are created with the configuration metadata that is supplied to the container, for example, in the form of XML <bean/> definitions. Beans defined in spring framework are singleton beans. There is an attribute in bean tag named "singleton" if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default, it is set to true. So, all the beans in spring framework are by default singleton beans.

4. **What does a Spring Bean definition contain?**

A Spring Bean definition contains all configuration metadata which is needed for the container to know how to create a bean, its lifecycle details and its dependencies.

5. How do you provide configuration metadata to the Spring Container?

There are three important methods to provide configuration metadata to the Spring Container:

a) XML based configuration file.
b) Annotation-based configuration
c) Java-based configuration

6. What are the benefits of IOC (Dependency Injection)?

Benefits of IOC (Dependency Injection) are as follows:

a) Minimizes the amount of code in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method with little or no extra configuration.
b) Make your application more testable by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.

c) Loose coupling is promoted with minimal effort and least intrusive mechanism. The factory design pattern is more intrusive because components or services need to be requested explicitly whereas in IOC the dependency is injected into requesting piece of code. Also some containers promote the design to interfaces not to implementations design concept by encouraging managed objects to implement a well-defined service interface of your own.

d) IOC containers support eager instantiation and lazy loading of services. Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management and dependency resolution between managed objects etc.

7. What is spring?

Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

8. What are the advantages of spring framework?

The advantages of spring are as follows:

a) Spring has layered architecture. Use what you need and leave you don't need now.
b) Spring Enables POJO Programming. POJO programming enables continuous integration and testability.
c) Dependency Injection and Inversion of Control Simplifies JDBC
d) Open source and no vendor lock-in.

9. What are features of spring?

a) Lightweight:

Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB and the processing overhead is also very negligible.

b) Inversion of control (IOC):

Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.

c) Aspect oriented (AOP):

Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

d) Container:

Spring contains and manages the life cycle and configuration of application objects.

e) MVC Framework:

Spring comes with MVC web application framework built on core spring functionality. This framework is highly configurable via strategy interfaces and accommodates multiple view technologies like JSP, Velocity, Tiles, iText and POI. But other frameworks can be easily used instead of Spring MVC Framework.

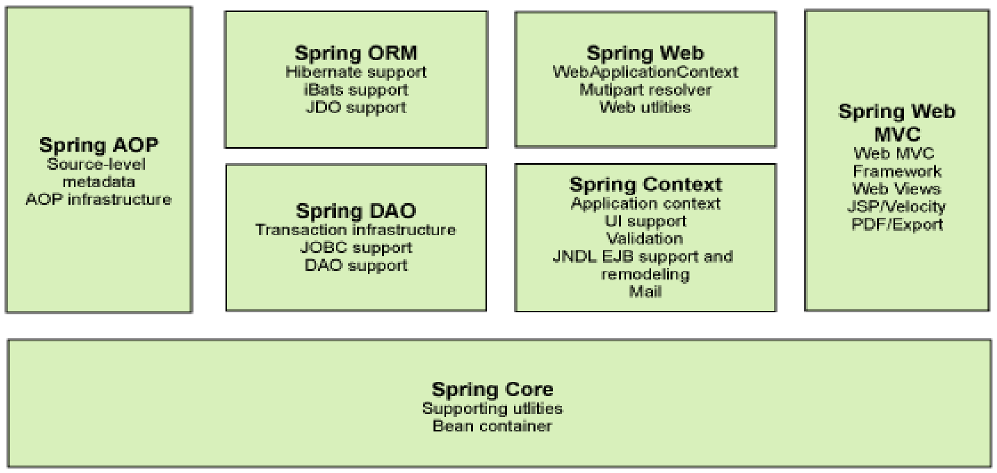
f) Transaction Management:

Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.

g) JDBC Exception Handling:

The JDBC abstraction layer of the spring offers a meaningful exception hierarchy which simplifies the error handling strategy. Integration with Hibernate, JDO and iBATIS. Spring provides best Integration services with Hibernate, JDO and iBATIS.

10. How many modules are there in spring? What are they?



Spring comprises of seven modules. They are.

a) The core container:

The core container provides the essential functionality of the spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the Inversion of Control (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

b) Spring context:

The spring context is a configuration file that provides context information to the spring framework. The spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation and scheduling functionality.

c) Spring AOP:

The Spring AOP module integrates aspect-oriented programming functionality directly into the spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

d) Spring DAO:

The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply with its generic DAO exception hierarchy.

e) Spring ORM:

The spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply with spring's generic transaction and DAO exception hierarchies.

f) Spring Web module:

The Web context module builds on top of the application context module, providing contexts for Web-based applications. As a result, the spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

g) Spring MVC framework:

The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

11. What are the types of Dependency Injection Spring supports?

a) Setter Injection:

Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

b) Constructor Injection:

Constructor-based DI is realized by invoking a constructor with number of arguments each representing a collaborator.

12. What is Bean Factory?

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

a) BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
b) BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

13. What is Application Context?

A bean factory is fine to simple applications but to take advantage of the full power of the spring framework, you may want to move up to springs more advanced container the applicationContext. An applicationContext is same as a bean factory. Both load bean definitions, wire beans together and dispense beans upon request. But it also provides:

a) A means for resolving text messages, including support for internationalization.
b) A generic way to load file resources.
c) Events to beans that are registered as listeners.

14. What is the difference between Bean Factory and Application Context?

On the surface, an application context is same as a bean factory. But application context offers much more.

a) Application contexts provide a means for resolving text messages, including support for i18n of those messages.
b) Application contexts provide a generic way to load file resources such as images.
c) Application contexts can publish events to beans that are registered as listeners.
d) Certain operations on the container or beans in the container which should be handled in a programmatic fashion with a bean factory can be handled declaratively in an application context.
e) ResourceLoader support: Spring's Resource interface use a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader hence provides an application with access to deployment-specific Resource instances.

f) MessageSource support: The application context implements MessageSource an interface used to obtain localized messages with the actual implementation being pluggable.

15. What are the common implementations of the Application Context?
The three commonly used implementation of 'Application Context' are

a) **ClassPathXmlApplicationContext:** It loads context definition from an XML file located in the class path, treating context definitions as class path resources. The application context is loaded from the application's class path by using the code.
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

b) **FileSystemXmlApplicationContext:** It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code.
ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");

c) **XmlWebApplicationContext:** It loads context definition from an XML file contained within a web application.

16. What is the typical Bean life cycle in Spring Bean Factory Container?
Bean life cycle in Spring Bean Factory Container is as follows:

a) The spring container finds the bean’s definition from the XML file and instantiates the bean.
b) Using the dependency injection, spring populates all of the properties as specified in the bean definition

c) If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean’s ID.

d) If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.

e) If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called.

f) If an init-method is specified for the bean, it will be called.

g) Finally, if there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization () methods will be called.

17. What do you mean by Bean wiring?
The act of creating associations between application components (beans) within the spring container is referred to as Bean wiring.

18. What do you mean by Auto Wiring?
The spring container can auto wire relationships between collaborating beans. It is possible to automatically let spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. The auto wiring functionality has five modes.

a) **no:** This is default setting. Explicit bean reference should be used for wiring.
b) **byName:** When autowiring byName, the Spring container looks at the properties of the beans on which autowire attribute is set to byName in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file
c) **byType:** When autowiring by datatype, the Spring container looks at the properties of the beans on which autowire attribute is set to byType in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist a fatal exception is thrown.
d) **constructor:** This mode is like byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container a fatal error is raised.

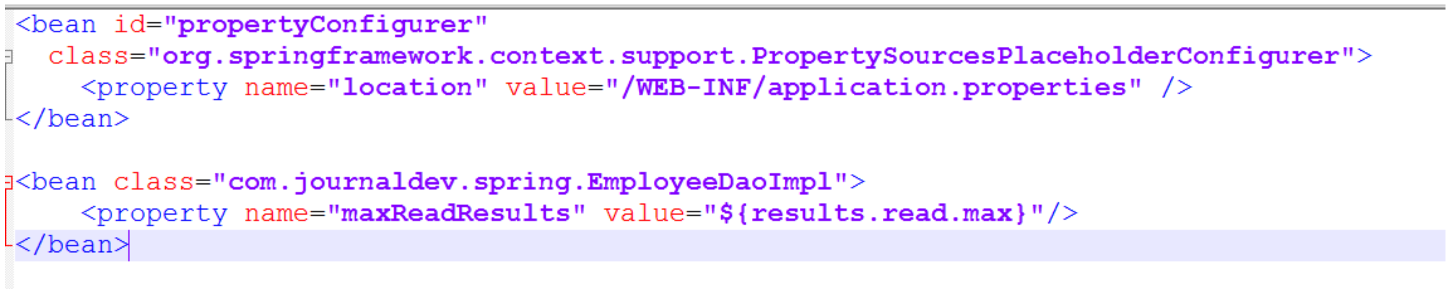
e) **autodetect:** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

19. Are there limitations with autowiring?
Limitations of autowiring are:

a) Overriding: You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
b) Primitive data types: You cannot autowire simple properties such as primitives, Strings and Classes.
c) Confusing nature: Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

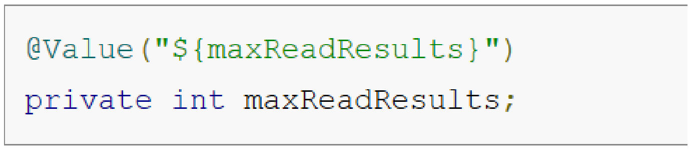
20. Can you inject null and empty string values in spring?
Yes, you can.

21. How to inject a java.util.Properties into a Spring Bean?
We need to define propertyConfigurer bean that will load the properties from the given property file. Then we can use Spring EL support to inject properties into other bean dependencies. For example;



```
<bean id="propertyConfigurer"
  class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">
    <property name="location" value="/WEB-INF/application.properties" />
</bean>

<bean class="com.journaldev.spring.EmployeeDaoImpl">
    <property name="maxReadResults" value="${results.read.max}"/>
</bean>
```

If you are using annotation to configure the spring bean, then you can inject property like below



```
@Value("${maxReadResults}")
private int maxReadResults;
```

22. Name some of the design patterns used in Spring Framework? [L&T]
Spring Framework is using a lot of design patterns some of the common ones are:

a) Singleton Pattern: Creating beans with default scope.
b) Factory Pattern: Bean Factory classes
c) Prototype Pattern: Bean scopes
d) Adapter Pattern: Spring Web and Spring MVC
e) Proxy Pattern: Spring Aspect Oriented Programming support
f) Template Method Pattern: JdbcTemplate, HibernateTemplate
g) Front Controller: Spring MVC DispatcherServlet
h) Data Access Object: Spring DAO support
i) Dependency Injection and Aspect Oriented Programming

23. What are ORM's spring supports?
Spring supports the following ORM's:

a) Hibernate
b) iBatis
c) JPA (Java Persistence API)
d) TopLink

e) JDO (Java Data Objects)
f) OJB

24. What are the ways to access Hibernate using spring?
There are two approaches to Spring's Hibernate integration:
a) Inversion of Control with a HibernateTemplate and Callback
b) Extending HibernateDaoSupport and Applying an AOP Interceptor

25. How to integrate Spring and Hibernate using HibernateDaoSupport?
Spring and Hibernate can integrate using Spring's SessionFactory called LocalSessionFactory. The integration process is
a) Configure the Hibernate SessionFactory
b) Extend your DAO Implementation from HibernateDaoSupport
c) Wire in Transaction Support with AOP

For Example:

```
<!-- Hibernate 4 SessionFactory Bean definition -->
<beans:bean id="hibernate4AnnotatedSessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <beans:property name="dataSource" ref="dataSource"/>
    <beans:property name="annotatedClasses">
        <beans:list>
            <beans:value>com.journaldev.spring.model.Person</beans:value>
        </beans:list>
    </beans:property>
    <beans:property name="hibernateProperties">
        <beans:props>
            <beans:prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</beans:prop>
            <beans:prop key="hbm2ddl.auto">update</beans:prop>
            <beans:prop key="hibernate.show_sql">true</beans:prop>
            <beans:prop key="hibernate.default_schema">ADMIN</beans:prop>
        </beans:props>
    </beans:property>

</beans:bean>
```

26. What are Bean scopes in Spring Framework?
The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext). The scopes supported are listed below:

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is each HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically, only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

27. Is Singleton beans thread safe in Spring Framework? if yes then why, if not then why? [JPMC]
No, singleton beans are not thread-safe in spring framework.

Spring's concept of a singleton bean differs from the Singleton pattern as defined in the Gang of Four (GoF) patterns book. The GoF Singleton hard-codes the scope of an object such that one and only one instance of a class is created per ClassLoader. The scope of the Spring singleton is best described as per container and per bean. This means that if you define one bean for a class in a single Spring container, then the Spring container creates one and only one instance of the class defined by that bean definition. The singleton scope is the default scope in Spring. To define a bean as a singleton in XML

Thread safety is a different context. Singleton spring beans has no relation with thread safety. spring container only manages life-cycle of objects and guaranteed that only one object in spring container. so, If Non-thread safe object is injected then obviously, it is not thread safe. To make it thread safe you must handle it by coding.

28. What are the methods of bean life cycle?
There are two important bean lifecycle methods. The first one is **setup** which is called when the bean is loaded in to the container. The second method is the **teardown** method which is called when the bean is unloaded from the container. The bean tag has two important attributes (init-method and destroy-method) with which you can define your own custom initialization and destroy methods. There are also the correspondent annotations (@PostConstruct and @PreDestroy).

29. How can you inject a Java Collection in Spring?
Spring offers the following types of collection configuration elements:

a) The <list> type is used for injecting a list of values, in the case that duplicates are allowed.
b) The <set> type is used for wiring a set of values but without any duplicates.
c) The <map> type is used to inject a collection of name-value pairs where name and value can be of any type.
d) The <props> type can be used to inject a collection of name-value pairs where the name and value are both Strings.

30. What are the different types of events of Listeners?
Following are the different types of events of listeners:

a) ContextClosedEvent – This event is called when the context is closed.
b) ContextRefreshedEvent – This event is called when context is initialized or refreshed
c) RequestHandledEvent – This event is called when the web context handles request

31. Annotation Based configuration

Starting from Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method or field declaration.

Annotation injection is performed before XML injection; thus the latter configuration will override the former for properties wired through both approaches. Annotation wiring is not turned on in the Spring container by default. So before we can use annotation-based wiring we will need to enable it in our Spring configuration file. So consider to have following configuration file in case you want to use any annotation in your Spring application.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>
   <!-- bean definitions go here -->

</beans>
```

a) **@Required:** The @Required annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time otherwise the container throws a BeanInitializationException.

b) **@Autowired:** The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

   i. **@Autowired on Setter Methods:** You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods it tries to perform byType autowiring on the method.

   ii. **@Autowired on Properties:** You can use @Autowired annotation on properties to get rid of the setter methods. When you will pass values of autowired properties using <property> Spring will automatically assign those properties with the passed values or references

   iii. **@Autowired on Constructors:** You can apply @Autowired to constructors as well. A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean even if no <constructor-arg> elements are used while configuring the bean in XML file

   iv. **@Autowired with (required=false) option:** By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using (required=false) option with @Autowired.

```java
@Autowired(required=false)
public void setAge(Integer age) {
   this.age = age;
}
```

c) **@Qualifier:** There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property in such case you can use @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired. Below is an example to show the use of @Qualifier annotation.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for profile bean -->
   <bean id="profile" class="com.tutorialspoint.Profile">
   </bean>

   <!-- Definition for student1 bean -->
   <bean id="student1" class="com.tutorialspoint.Student">
      <property name="name"  value="Zara" />
      <property name="age"  value="11"/>
   </bean>

   <!-- Definition for student2 bean -->
   <bean id="student2" class="com.tutorialspoint.Student">
      <property name="name"  value="Nuha" />
      <property name="age"  value="2"/>
   </bean>

</beans>
```

```java
public class Profile {
   @Autowired
   @Qualifier("student1")
   private Student student;
```

32. Java based configuration
Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations

@Configuration & @Bean Annotations: Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions. The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

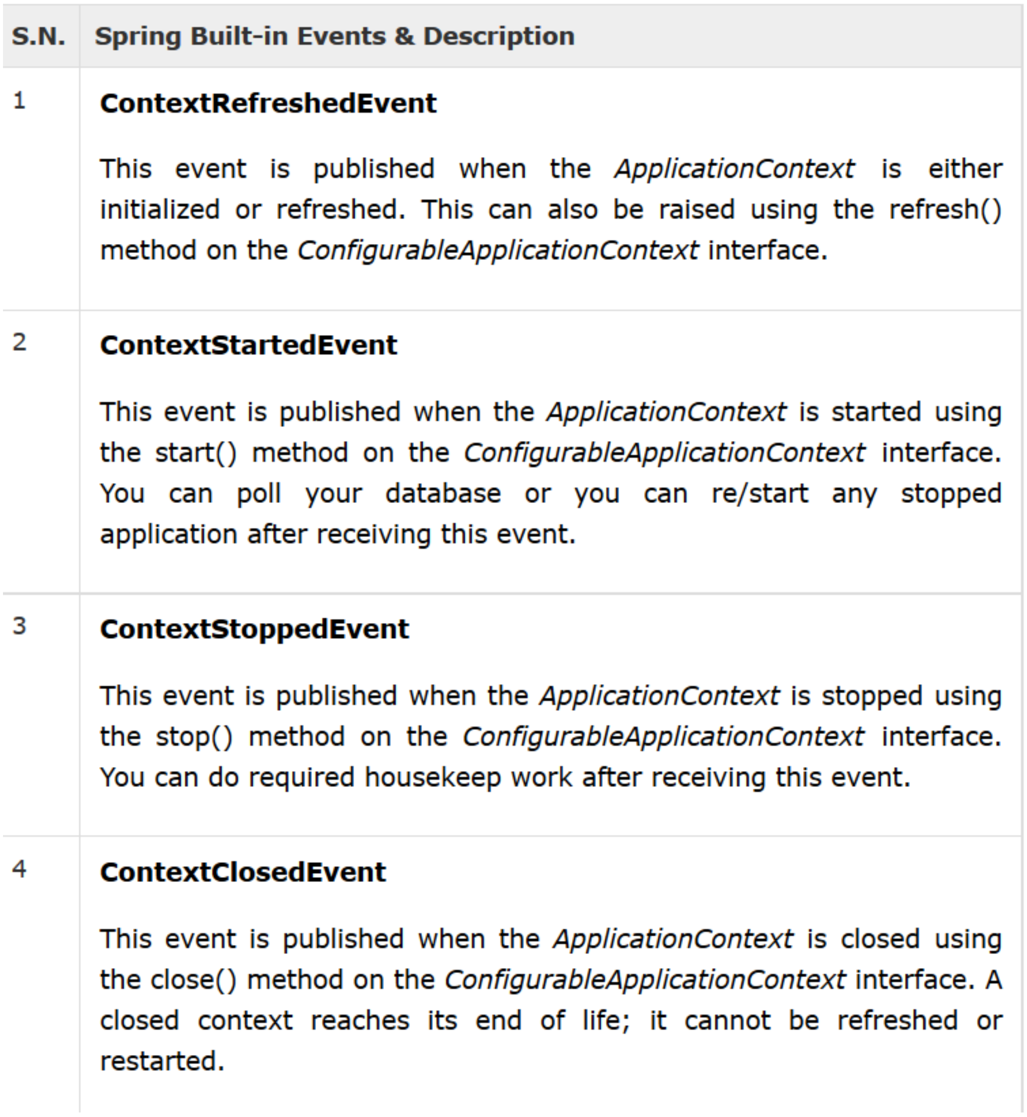
```
package com.tutorialspoint;
import org.springframework.context.annotation.*;

@Configuration
public class HelloWorldConfig {

   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

Here the method name annotated with @Bean works as bean ID and it creates and returns actual bean. Your configuration class can have declaration for more than one @Bean. Once your configuration classes are defined you can load and provide them to Spring container using AnnotationConfigApplicationContext

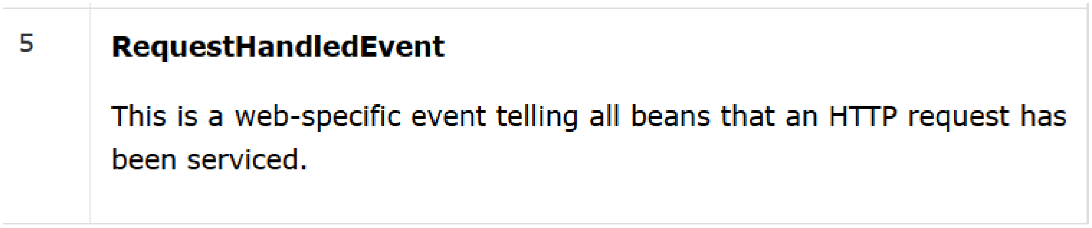
```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(HelloWorldConfig.class);

   HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

   helloWorld.setMessage("Hello World!");
   helloWorld.getMessage();
}
```

33. Event Handling in Spring

The ApplicationContext publishes certain types of events when loading the beans. For example, a ContextStartedEvent is published when the context is started and ContextStoppedEvent is published when the context is stopped.
Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. So if a bean implements the ApplicationListener, then every time an ApplicationEvent gets published to the ApplicationContext, that bean is notified.



S.N.	Spring Built-in Events & Description
1	**ContextRefreshedEvent** This event is published when the *ApplicationContext* is either initialized or refreshed. This can also be raised using the refresh() method on the *ConfigurableApplicationContext* interface.
2	**ContextStartedEvent** This event is published when the *ApplicationContext* is started using the start() method on the *ConfigurableApplicationContext* interface. You can poll your database or you can re/start any stopped application after receiving this event.
3	**ContextStoppedEvent** This event is published when the *ApplicationContext* is stopped using the stop() method on the *ConfigurableApplicationContext* interface. You can do required housekeep work after receiving this event.
4	**ContextClosedEvent** This event is published when the *ApplicationContext* is closed using the close() method on the *ConfigurableApplicationContext* interface. A closed context reaches its end of life; it cannot be refreshed or restarted.



5	**RequestHandledEvent** This is a web-specific event telling all beans that an HTTP request has been serviced.

34. Custom Events in Spring

a) Create an event class, CustomEvent by extending ApplicationEvent. This class must define a default constructor which should inherit constructor from ApplicationEvent class.

b) Once your event class is defined, you can publish it from any class, let us say EventClassPublisher which implements ApplicationEventPublisherAware. You will also need to declare this class in XML configuration file as a bean so that the container can identify the bean as an event publisher because it implements the ApplicationEventPublisherAware interface.

c) A published event can be handled in a class, let us say EventClassHandler which implements ApplicationListener interface and implements onApplicationEvent method for the custom event.

d) Create beans configuration file Beans.xml under the src folder and a MainApp class which will work as spring application.

35. Spring - JDBC Framework

JdbcTemplate Class: The JdbcTemplate class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative exception hierarchy defined in the org.springframework.dao package. Instances of the JdbcTemplate class are thread safe once configured. So, you can configure a single instance of a JdbcTemplate and then safely inject this shared reference into multiple DAOs. A common practice when using the JdbcTemplate class is to configure a DataSource in your Spring configuration file and then dependency-inject that shared DataSource bean into your DAO classes and the JdbcTemplate is created in the setter for the DataSource.

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="sys as sysdba" />
    <property name="password" value="sgr" />
</bean>

<bean id="studentJDBCTemplate" class="com.tutorialspoint.StudentJDBCTemplate">
    <property name="dataSource" ref="dataSource"></property>

</bean>
```

36. What are the types of the transaction management Spring supports?

Spring Framework supports:

a) Programmatic transaction management.
b) Declarative transaction management.

37. What are the benefits of the Spring Framework transaction management?

The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:

a) Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA and JDO.
b) Supports declarative transaction management.
c) Provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
d) Integrates very well with Spring's various data access abstractions.

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

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

39. When to use programmatic and declarative transaction management?

Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic and is not difficult to configure.

40. Explain about the Spring DAO support?

The Data Access Object (DAO) support in Spring is designed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

41. What are the exceptions thrown by the Spring DAO classes?

Spring DAO classes throw exceptions which are subclasses of DataAccessException (org.springframework.dao.DataAccessException). Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

42. What is rowMapper in Spring? [L&T]
we can use RowMapper interface to fetch the records from the database using query () method of JdbcTemplate class. In the execute of we need to pass the instance of RowMapper now.

```
public T query(String sql,RowMapper<T> rm)
```

RowMapper interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. It defines only one method mapRow () that accepts ResultSet instance and int as the parameter list.

```
public T mapRow(ResultSet rs, int rowNumber)throws SQLException
```

43. What is SQLExceptionTranslator?
SQLExceptionTranslator is an interface to be implemented by classes that can translate between SQLException and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

44. What is PreparedStatementCreator?
PreparedStatementCreator:
a) One of the most common used interfaces for writing data to database
b) Has one method – createPreparedStatement(Connection)
c) Responsible for creating a PreparedStatement
d) Does not need to handle SQLExceptions

45. What is SQLProvider?
SQLProvider:
a) Has one method – getSql()
b) Typically implemented by PreparedStatementCreator implementers.
c) Useful for debugging.

46. What is RowCallbackHandler?
The RowCallbackHandler interface extracts values from each row of a ResultSet.
a) Has one method – processRow(ResultSet)
b) Called for each row in ResultSet.
c) Typically stateful.

❖ *Spring MVC*

1. What is Spring MVC framework?
⇨ The spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different

aspects of the application (input logic, business logic and UI logic), while providing a loose coupling between model, view and controller parts of application. Spring framework provides lots of advantages over other MVC frameworks e.g.

a) Clear separation of roles – controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
b) Powerful and straightforward configuration of both framework and application classes as JavaBeans.
c) Reusable business code – no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
d) Customizable binding and validation
e) Customizable handler mapping and view resolution
f) Customizable locale and theme resolution
g) A JSP form tag library, introduced in Spring 2.0, which makes writing forms in JSP pages much easier. etc.

2. What are DispatcherServlet and ContextLoaderListener?
Spring's web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that handles all the HTTP requests and responses. Spring's DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container so it allows you to use every feature that spring has. After receiving an HTTP request, DispatcherServlet consults the HandlerMapping (configuration files) to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods and set model data and then returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pick up the defined view for the request. Once view is finalized, the DispatcherServlet passes the model data to the view which is finally rendered on the browser.

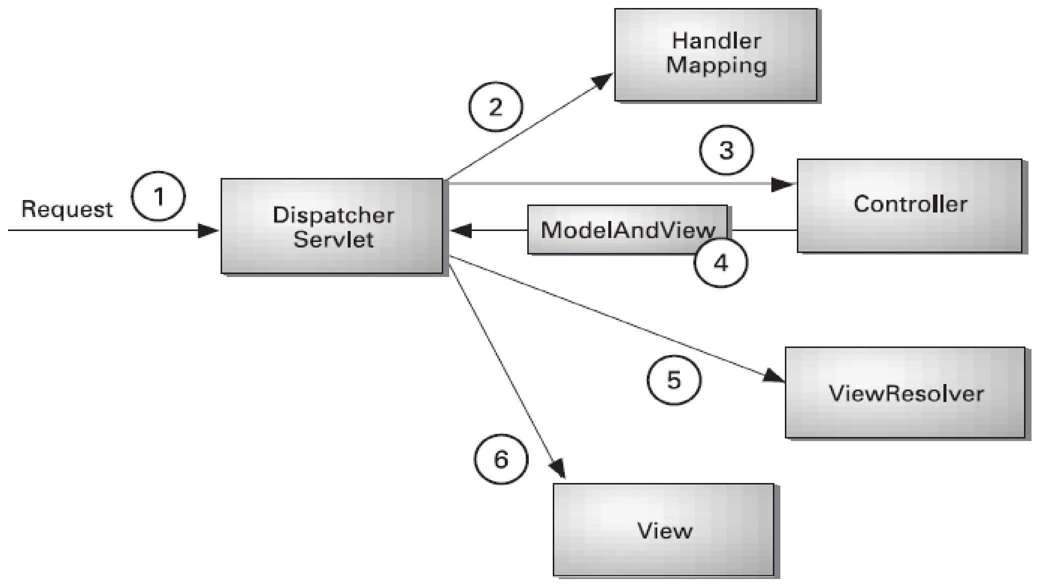


Figure 8.1 The life cycle of a request in Spring MVC

```xml
<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <servlet>
        <servlet-name>spring</servlet-name>
            <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
            </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

By default, DispatcherServlet loads its configuration file using <servlet_name>-servlet.xml. E.g. with above web.xml file, DispatcherServlet will try to find spring-servlet.xml file in class path.

ContextLoaderListener reads the spring configuration file (with value given against "**contextConfigLocation**" in web.xml), parse it and loads the beans defined in that config file.

3. What is the front controller class of Spring MVC?

⇨ A front controller is defined as a controller which handles all requests for a Web Application. **DispatcherServlet is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.** When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring's web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

4. Can we have multiple spring configuration files?

YES. You can have multiple spring context files. There are two ways to make spring read and configure them.

a) Specify all files in web.xml file using **contextConfigLocation** init parameter.

```xml
<servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                WEB-INF/spring-dao-hibernate.xml,
                WEB-INF/spring-services.xml,
                WEB-INF/spring-security.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

b) You can **import them into existing configuration file** you have already configured.

```
<beans>
    <import resource="spring-dao-hibernate.xml"/>
    <import resource="spring-services.xml"/>
    <import resource="spring-security.xml"/>

    ... //Other configuration stuff

</beans>
```

5. Difference between <context:annotation-config> vs <context:component-scan>?
   a) First big difference between both tags is that <context:annotation-config> is used **to activate applied annotations in already registered beans in application context.** Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.
   b) Second difference is driven from first difference itself**. It registers the beans defined in config file into context + it also scans the annotations inside beans and activates them.** So <context:component-scan> does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

6. Difference between @Component, @Controller, @Repository & @Service annotations?
   a) The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:

```
@Component
public class EmployeeDAOImpl implements EmployeeDAO {
    ...
}
```

   b) The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, it also makes the unchecked exceptions eligible for translation into Spring DataAccessException.
   c) The @Service annotation is also a specialization of the component annotation. It doesn't currently provide any additional behavior over the @Component annotation, but it's a good idea to use @Service over @Component in service-layer classes because it specifies intent better.
   d) @Controller annotation marks a class as a Spring Web MVC controller. It is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

7. What does the ViewResolver class?

⇨ ViewResolver is an interface to be implemented by objects that can resolve views by name. There are plenty of ways using which you can resolve view names. These ways are supported by various in-built implementations of this interface. Most commonly used implementation is InternalResourceViewResolver class. It defines prefix and suffix properties to resolve the view component.

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

So with above view resolver configuration, if controller method return "*login*" string, then the "/WEB-INF/views/login.jsp" file will be searched and rendered.

8. What is a MultipartResolver and when it's used?

⇨ Spring comes with MultipartResolver to handle file upload in web application. There are two concrete implementations included in spring:
   a) **CommonsMultipartResolver** for Jakarta Commons FileUpload
   b) **StandardServletMultipartResolver** for Servlet 3.0 Part API

   To define an implementation, create a bean with the id "***multipartResolver***" in a DispatcherServlet's application context. Such a resolver gets applied to all requests handled by that DispatcherServlet.
   If a DispatcherServlet detects a multipart request, it will resolve it via the configured MultipartResolver and pass on a wrapped HttpServletRequest. Controllers can then cast their given request to the MultipartHttpServletRequest interface, which permits access to any MultipartFiles.

9. How does Spring MVC provide validation support?

⇨ Spring supports validations primarily into two ways.
   a) Using **JSR-303 Annotations** and any reference implementation e.g. Hibernate Validator
   b) Using **custom implementation of org.springframework.validation.Validator** interface

10. How to validate form data in Spring Web MVC Framework?

⇨ Spring MVC supports validation by means of a validator object that implements the Validator interface. You need to create a class and implement Validator interface. In this custom validator class, you use utility methods such as rejectIfEmptyOrWhitespace() and rejectIfEmpty() in the ValidationUtils class to validate the required form fields.

```
@Component
public class EmployeeValidator implements Validator
{
    public boolean supports(Class clazz) {
        return EmployeeVO.class.isAssignableFrom(clazz);
    }

    public void validate(Object target, Errors errors)
    {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "error.firstName", "First name is r
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "error.lastName", "Last name is requ
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "error.email", "Email is required.");
    }
}
```

If any of form fields is empty, these methods will create a field error and bind it to the field. The second argument of these methods is the property name, while the third and fourth are the error code and default error message.

To activate this custom validator as a spring managed bean, you need to do one of following things:

1) Add @Component annotation to EmployeeValidator class and activate annotation scanning on the package containing such declarations.

```
<context:component-scan base-package="com.howtodoinjava.demo" />
```

2) Alternatively, you can register the validator class bean directly in context file.

```
<bean id="employeeValidator" class="com.howtodoinjava.demo.validator.EmployeeValidator" />
```

11. What is Spring MVC Interceptor and how to use it?

⇨ HandlerInterceptor interface in your spring mvc application **to pre-handle and post-handle web requests** that are handled by Spring MVC controllers. These handlers are mostly used to manipulate the model attributes returned/submitted they are passed to the views/controllers.

A handler interceptor can be registered for URL mappings, so it only intercepts requests mapped to certain URLs. Each handler interceptor must implement the HandlerInterceptor interface, which contains three callback methods for you to implement: **preHandle(), postHandle() and afterCompletion().**

Problem with HandlerInterceptor interface is that your new class will have to implement all three methods irrespective of whether it is needed or not. To avoid overriding, you can use HandlerInterceptorAdapter class. These classes implements HandlerInterceptor and provide default blank implementations.

```
<bean id="handlerMapping" class="org.springframework.web.servlet.mvc.method.annotation.RequestMa
ppingHandlerMapping">
      <property name="interceptors">
            <list>
                  <ref bean="yourCustomHandlerInterceptor"/>
            </list>
        </property>
</bean>
```

12. How to handle exceptions in Spring MVC Framework?

⇨ In a Spring MVC application, you can register one or more exception resolver beans in the web application context to resolve uncaught exceptions. These beans must implement the HandlerExceptionResolver interface for DispatcherServlet to auto-detect them. Spring MVC comes with a simple exception resolver for you to map each category of exceptions to a view i.e. SimpleMappingExceptionResolver to map each category of exceptions to a view in a configurable way.
Let’s say we have an exception class i.e. AuthException. And we want that everytime this exception is thrown from anywhere into application, we want to show a pre-determined view page /WEB-INF/views/error/authExceptionView.jsp. So the configuration would be.

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="com.howtodoinjava.demo.exception.AuthException">
                error/authExceptionView
            </prop>
        </props>
    </property>
    <property name="defaultErrorView" value="error/genericView"/>
</bean>
```

The “*defaultErrorView*” property can be configured to show a generic message for all other exceptions which are not configured inside “*exceptionMappings*” list.

13. How to achieve localization in Spring MVC applications?
Spring framework is shipped with LocaleResolver to support the internationalization and thus localization as well. To make Spring MVC application supports the internationalization, you will need to register two beans.

1) SessionLocaleResolver : It resolves locales by inspecting a predefined attribute in a user's session. If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header.

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
    <property name="defaultLocale" value="en" />
</bean>
```

2) LocaleChangeInterceptor : This interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the **paramName** property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.

```
<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="lang" />
</bean>

<!-- Enable the interceptor -->
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="localeChangeInterceptor" />
        </list>
    </property>
</bean>
```

Next step is to have each locale specific properties file having texts in that locale specific language e.g. messages.properties and messages_zh_CN.properties etc.

14. How to get ServletContext and ServletConfig object in a Spring Bean?
Simply implement ServletContextAware and ServletConfigAware interfaces and override below methods.

```
@Controller
@RequestMapping(value = "/magic")
public class SimpleController implements ServletContextAware, ServletConfigAware {

    private ServletContext context;
    private ServletConfig config;

    @Override
    public void setServletConfig(final ServletConfig servletConfig) {
        this.config = servletConfig;

    }

    @Override
    public void setServletContext(final ServletContext servletContext) {
        this.context = servletContext;
    }

    //other code
}
```

15. How to use Tomcat JNDI DataSource in Spring Web Application?
    For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it with JdbcTemplate to perform database operations.

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/jdbc/MySQLDB"/>
</bean>
```

16. How do you integrate Spring MVC with tiles?
    Tiles help us to define the layout for a web page. We can integrate Spring MVC with tiles by configuring TilesConfigurer and setting up appropriate view resolver.

```
    <bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles2.TilesConfigurer"
    p:definitions="/WEB-INF/tiles-defs/templates.xml" />
    <bean id="tilesViewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver"
    p:viewClass="org.springframework.web.servlet.view.tiles2.TilesView" />
```

17. *How do you configure Spring MVC web application to use UTF-8 encoding for handling forms?*
    Using org.springframework.web.filter.CharacterEncodingFilter. Shown below.

```
<filter>
    <filter-name>encoding-filter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encoding-filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

18. *How do you enable spring security for a web application?*
    Spring Security is used to implement Authentication and Authorization for a web application. We can enable spring security by configuring an appropriate security filter. Example shown below. We can create a separate security-context.xml to define the authentication and authorization roles and accesses.

```
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Spring Web services

A. SOAP Web services

1. What is a Web Service?

Web Services work on client-server model where client applications can access web services over the network. Web services provide endpoint URLs and expose methods that can be accessed over network through client programs written in java, shell script or any other different technologies. Web services are stateless and don’t maintain user session like web applications.

2. What are the advantages of Web Services?

Some of the advantages of web services are:

a) Interoperability: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language. Web service can be written in java programming and client can be PHP and vice versa.
b) Reusability: One web service can be used by many client applications at the same time.
c) Loose Coupling: Web services client code is totally independent with server code, so we have achieved loose coupling in our application.
d) Easy to deploy and integrate, just like web applications.
e) Multiple service versions can be running at same time.

3. What are different types of Web Services?
There are two types of web services:

a) SOAP Web Services: Runs on SOAP protocol and uses XML technology for sending data.
b) Restful Web Services: It's an architectural style and runs on HTTP/HTTPS protocol almost all the time. REST is a stateless client-server architecture where web services are resources and can be identified by their URIs. Client applications can use HTTP GET/POST methods to invoke Restful web services.

4. What is SOAP?
SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.

5. What are advantages of SOAP Web Services?
SOAP web services have all the advantages that web services has, some of the additional advantages are:

a) WSDL document provides contract and technical details of the web services for client applications without exposing the underlying implementation technologies.
b) SOAP uses XML data for payload as well as contract, so it can be easily read by any technology.
c) SOAP protocol is universally accepted, so it's an industry standard approach with many easily available open source implementations.

6. What are disadvantages of SOAP Web Services?
Some of the disadvantages of SOAP protocol are:

a) Only XML can be used, JSON and other lightweight formats are not supported.
b) SOAP is based on the contract, so there is a tight coupling between client and server applications.
c) SOAP is slow because payload is large for a simple string message, since it uses XML format.
d) Anytime there is change in the server side contract, client stub classes need to be generated again.
e) Can't be tested easily in browser

7. What is WSDL?
WSDL stands for Web Service Description Language. WSDL is an XML based document that provides technical details about the web service. Some of the useful information in WSDL document are: method name, port types, service end point, binding, method parameters etc.

8. What are different components of WSDL?
Some of the different tags in WSDL xml are:

a) xsd:import namespace and schemaLocation: provides WSDL URL and unique namespace for web service.
b) message: for method arguments
c) part: for method argument name and type
d) portType: service name, there can be multiple services in a wsdl document.
e) operation: contains method name
f) soap:address for endpoint URL.

9. What is UDDI?
UDDI is acronym for Universal Description, Discovery and Integration. UDDI is a directory of web services where client applications can lookup for web services. Web Services can register to the UDDI server and make them available to client applications.

10. What is difference between Top Down and Bottom Up approach in SOAP Web Services?
In Top Down approach first WSDL document is created to establish the contract between web service and client and then code is written, it's also termed as contract first approach. This is hard to implement because classes need to be written to confirm the contract established in WSDL. Benefit of this approach is that both client and server code can be written in parallel.

In Bottom Up approach, first web service code is written and then WSDL is generated. It's also termed as contract last approach. This approach is easy to implement because WSDL is generated based on code. In this approach client code has to wait for WSDL from server side to start their work.

11. Compare SOAP and REST web services?

SOAP	REST
SOAP is a standard protocol for creating web services.	REST is an architectural style to create web services.
SOAP is acronym for Simple Object Access Protocol.	REST is acronym for REpresentational State Transfer.
SOAP uses WSDL to expose supported methods and technical details.	REST exposes methods through URIs, there are no technical details.
SOAP web services and client programs are bind with WSDL contract	REST doesn't have any contract defined between server and client
SOAP web services and client are tightly coupled with contract.	REST web services are loosely coupled.
SOAP learning curve is hard, requires us to learn about WSDL generation, client stubs creation etc.	REST learning curve is simple, POJO classes can be generated easily and works on simple HTTP methods.
SOAP supports XML data format only	REST supports any data type such as XML, JSON, image etc.
SOAP web services are hard to maintain, any change in WSDL contract requires us to create client stubs again and then make changes to client code.	REST web services are easy to maintain when compared to SOAP, a new method can be added without any change at client side for existing resources.

SOAP web services can be tested through programs or software such as Soap UI.	REST can be easily tested through CURL command, Browsers and extensions such as Chrome Postman.

12. What are different ways to test web services?
SOAP web services can be tested programmatically by generating client stubs from WSDL or through software such as Soap UI.

REST web services can be tested easily with program, curl commands and through browser extensions. Resources supporting GET method can be tested with browser itself, without any program.

13. Can we maintain user session in web services?
Web services are stateless so we can't maintain user sessions in web services.

14. What is difference between SOA and Web Services?
Service Oriented Architecture (SOA) is an architectural pattern where applications are designed in terms of services that can be accessed through communication protocol over network. SOA is a design pattern and doesn't go into implementation. Web Services can be thought of as Services in SOAP architecture and providing means to implement SOA pattern.

15. How would you choose between SOAP and REST web services?
Web Services work on client-server model and when it comes to choose between SOAP and REST, it all depends on project requirements. Let's look at some of the conditions affecting our choice:

a. Do you know your web service clients beforehand? If Yes, then you can define a contract before implementation and SOAP seems better choice. But if you don't then REST seems better choice because you can provide sample request/response and test cases easily for client applications to use later on.
b. How much time you have? For quick implementation REST is the best choice. You can create web service easily, test it through browser/curl and get ready for your clients.
c. What kind of data format are supported? If only XML then you can go with SOAP but if you think about supporting JSON also in future, then go with REST.

16. What is JAX-WS API?
JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application. It's part of standard Java API, so we don't need to include anything else which working with it.

17. Name some frameworks in Java to implement SOAP web services?
We can create SOAP web services using JAX-WS API, however some of the other frameworks that can be used are Apache Axis and Apache CXF. Note that they are not implementations of JAX-WS API, they are totally different framework that work on Servlet model to expose your business logic classes as SOAP web services.

18. Name important annotations used in JAX-WS API?
Some of the important annotations used in JAX-WS API are:
a) @WebService
b) @SOAPBinding
c) @WebMethod

19. What is use of javax.xml.ws.Endpoint class?

Endpoint class provides useful methods to create endpoint and publish existing implementation as web service. This comes handy in testing web services before making further changes to deploy it on actual server.

20. How to get WSDL file of a SOAP web service?
WSDL document can be accessed by appending? wsdl to the SOAP endoint URL. In above example, we can access it at http://localhost:8888/testWS?wsdl location.

21. What is wsimport utility?
We can use wsimport utility to generate the client stubs. This utility comes with standard installation of JDK

22. What is the use of @XmlRootElement annotation?
XmlRootElement annotation is used by JAXB to transform java object to XML and vice versa. So we have to annotate model classes with this annotation.

B. REST Web services:-

1. What is REST Web Services?
REST is the acronym for REpresentational State Transfer. REST is an architectural style for developing applications that can be accessed over the network. REST architectural style was brought in light by Roy Fielding in his doctoral thesis in 2000. REST is a stateless client-server architecture where web services are resources and can be identified by their URIs. Client applications can use HTTP GET/POST methods to invoke Restful web services. REST doesn't specify any specific protocol to use, but in almost all cases it's used over HTTP/HTTPS. When compared to SOAP web services, these are lightweight and don't follow any standard. We can use XML, JSON, text or any other type of data for request and response.

2. What are advantages of REST web services?
Some of the advantages of REST web services are:
a) Learning curve is easy since it works on HTTP protocol
b) Supports multiple technologies for data transfer such as text, xml, json, image etc.
c) No contract defined between server and client, so loosely coupled implementation.
d) REST is a lightweight protocol
e) REST methods can be tested easily over browser.

3. What are disadvantages of REST web services?
Some of the disadvantages of REST are:
a) Since there is no contract defined between service and client, it has to be communicated through other means such as documentation or emails.
b) Since it works on HTTP, there can't be asynchronous calls.
c) Sessions can't be maintained.

4. What is a Resource in Restful web services
Resource is the fundamental concept of Restful architecture. A resource is an object with a type, relationship with other resources and methods that operate on it. Resources are identified with their URI, HTTP methods they support and request/response data type and format of data.

5. What are different HTTP Methods supported in Restful Web Services?
Restful web services supported HTTP methods are – GET, POST, PUT, DELETE and HEAD.

6. What is the use of Accept and Content-Type Headers in HTTP Request?

These are important headers in Restful web services. Accept headers tells web service what kind of response client is accepting, so if a web service is capable of sending response in XML and JSON format and client sends Accept header as "application/xml" then XML response will be sent. For Accept header "application/json", server will send the JSON response.
Content-Type header is used to tell server what is the format of data being sent in the request. If Content-Type header is "application/xml" then server will try to parse it as XML data. This header is useful in HTTP Post and Put requests.

7. What is JAX-RS API?

Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services. JAX-RS is part of JDK, so you don't need to include anything to use it's annotations.

8. Name some implementations of JAX-RS API?

There are two major implementations of JAX-RS API.

a) Jersey: Jersey is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.
b) RESTEasy: RESTEasy is the JBoss project that provides JAX-RS implementation.

9. How to set different status code in HTTP response?

For setting HTTP status code other than 200, we have to use javax.ws.rs.core.Response class for response. Below are some of the sample return statements showing it's usage.

```
return Response.status(422).entity(exception).build();
return Response.ok(response).build(); //200
```

Hibernate Framework

1. What is hibernate?

   Hibernate: hibernate-core: 5.1 (Latest Version)

Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate and retrieve data from the database. Hibernate is an Object-Relational Mapping (ORM) solution for JAVA and it raised as an open source persistent framework. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application. Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

2. What is ORM?

ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation and data access.

3. What does an ORM solution comprises of?
   a) It should have an API for performing basic CRUD (Create, Read, Update, Delete) operations on objects of persistent classes

b) Should have a language or an API for specifying queries that refer to the classes and the properties of classes
c) An ability for specifying mapping metadata
d) It should have a technique for ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching and other optimization functions.

4. What are the different levels of ORM quality?

There are four levels defined for ORM quality.

a) Pure relational
The entire application, including the user interface, is designed around the relational model and SQL-based relational operations. It needs no extra programming or use of third party API to convert data between incompatible type systems in the object programming languages. It supports virtual object database that could be used from within programming language.

b) Light object mapping
The entities are represented as classes that are mapped manually to the relational tables. The code is hidden from the business logic using specific design patterns. This approach is successful for applications with a less number of entities, or applications with common, metadata-driven data models. This approach is most known to all.

c) Medium object mapping
The application is designed around an object model. The SQL code is generated at build time. And the associations between objects are supported by the persistence mechanism, and queries are specified using an object-oriented expression language. This is best suited for medium-sized applications with some complex transactions. Used when the mapping exceeds 25 different database products at a time.

d) Full object mapping

Full object mapping supports sophisticated object modeling: composition, inheritance, polymorphism and persistence. The persistence layer implements transparent persistence; persistent classes do not inherit any special base class or have to implement a special interface. Efficient fetching strategies and caching strategies are implemented transparently to the application.

5. What are the advantages of ORM over JDBC?

An ORM system has following advantages over plain JDBC

a) Let's business code access objects rather than DB tables.
b) Hide details of SQL queries from OO logic.
c) No need to deal with the database implementation.
d) Entities based on business concepts rather than database structure.
e) Transaction management and automatic key generation.
f) Fast development of application.

6. What are the advantages of using Hibernate?

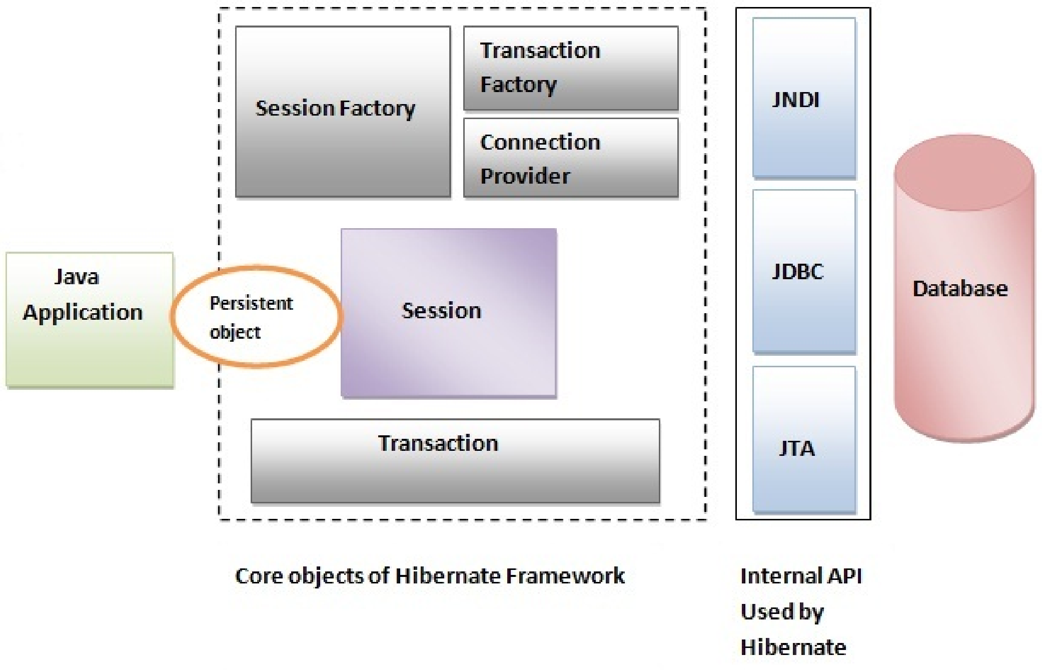
Following are the advantages of using Hibernate.

a) Hibernate takes care of mapping Java classes to database tables using XML files
b) Provides simple APIs for storing and retrieving Java objects directly to and from the database.
c) If there is change in Database or in any table then the only need to change XML file properties.

d) Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
e) Hibernate does not require an application server to operate.
f) Manipulates complex associations of objects of your database.
g) Minimize database access with smart fetching strategies.
h) Provides simple querying of data.

7. Explain hibernate architecture?

Hibernate architecture comprises of many interfaces such as Configuration, SessionFactory, Session, Transaction etc.



Hibernate framework uses many objects session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

Object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

8. What are the core interfaces of Hibernate?

   The core interfaces of Hibernate framework are:

   a) Configuration
   b) SessionFactory
   c) Session
   d) Query
   e) Criteria
   f) Transaction

9. Is SessionFactory a thread-safe object?

   Yes, SessionFactory is a thread-safe object; many threads cannot access it simultaneously.

10. What is Session?

    It maintains a connection between hibernate application and database. It provides methods to store, update, delete or fetch data from the database such as persist (), update (), delete (), load (), get () etc. It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

11. Is Session a thread-safe object?

    No, Session is not a thread-safe object; many threads can access it simultaneously. In other words, you can share it between threads.

12. What is the difference between save() & saveOrUpdate()

No.	**save()**	**saveOrUpdate()**
1)	save() generates a new identifier and INSERT record into database	SaveOrUpdate can either INSERT or UPDATE based upon existence of record.
2)	Returns type is Serializable : public Serializable save(Object o)	Return type is void

13. What is the difference between session.save () and session.persist() method?

No.	**save()**	**persist()**

1)	returns the identifier (Serializable) of the instance.	return nothing because its return type is void.
2)	Syntax: public Serializable save(Object o)	Syntax: public void persist(Object o)

14. What the difference between get and load method?

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

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

The following Hibernate code snippet retrieves a User object from the database:

```
User user = (User) session.get(User.class, userID);
```

The get() method is special because the identifier uniquely identifies a single instance of a class. Hence it's common for applications to use the identifier as a convenient handle to a persistent object. Retrieval by identifier can use the cache when retrieving an object, avoiding a database hit if the object is already cached. The get() method returns null if the object can't be found.

Hibernate also provides a load() method:

```
User user = (User) session.load(User.class, userID);
```

If load() can't find the object in the cache or database, an exception is thrown. The load() method never returns null. The load() method may return a proxy instead of a real persistent instance. A proxy is a placeholder instance of a runtime-generated subclass (through cglib or Javassist) of a mapped persistent class, it can initialize itself if any method is called that is not the mapped database identifier getter-method.

On the other hand, get() never returns a proxy. Choosing between get() and load() is easy: If you're certain the persistent object exists, and nonexistence would be considered exceptional, load() is a good option. If you aren't certain there is a persistent instance with the given identifier, use get() and test the return value to see if it's null. Using load() has a further implication: The application may retrieve a valid reference (a proxy) to a persistent instance without hitting the database to retrieve its persistent state. So load() might not throw an exception when it doesn't find the persistent object in the cache or database; the exception would be thrown later, when the proxy is accessed.

15. How to enable hibernate SQL logging in console

Hibernate has built-in a function to enable the logging of all the generated SQL statements to the console. You can enable it by add a “show_sql” property in the Hibernate configuration file “hibernate.cfg.xml”. This function is good for basic troubleshooting.

```
<property name="show_sql">true</property> //without formatting
 <property name="format_sql">true</property> //formatted sql
<property name="use_sql_comments">true</property> // with comments sql
```

16. What is the difference between update and merge method?

The differences between update() and merge() methods are given below.

No.	update() method	merge() method
1)	Update means to edit something.	Merge means to combine something.
2)	update() should be used if session doesn't contain an already persistent state with same id. It means update should be used inside the session only. After closing the session it will throw error.	merge() should be used if you don't know the state of the session, means you want to make modification at any time.

17. What the states of object are in hibernate?

There are 3 states of object (instance) in hibernate.

a) **Transient**: The object is in transient state if it is just created but has no primary key (identifier) and not associated with session.
b) **Persistent**: The object is in persistent state if session is open, and you just saved the instance in the database or retrieved the instance from the database.
c) **Detached**: The object is in detached state if session is closed. After detached state, object comes to persistent state if you call lock() or update() method.

18. What are the inheritance mapping strategies?

There are 3 ways of inheritance mapping in hibernate.

a) Table per hierarchy
b) Table per concrete class
c) Table per subclass

19. How to make immutable class in hibernate?

If you mark a class as mutable="false", class will be treated as an immutable class. By default, it is mutable="true".

20. What is automatic dirty checking in hibernate?

The automatic dirty checking feature of hibernate, calls update statement automatically on the objects that are modified in a transaction.

a) Hibernate monitors all Persistent objects
b) At the end of a unit of work, it knows which objects have been modified
c) It then calls update statements on all updated objects
d) This process of monitoring and updating only objects that have changed is called automatic dirty checking.

21. How many types of association mapping are possible in hibernate?

There can be 4 types of association mapping in hibernate.

a) One to One
b) One to Many
c) Many to One
d) Many to Many

22. Is it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many

23. What is lazy loading in hibernate?

Loading the object the Lazy loading. For example an individual's on demand is called profile in an online store may have multiple delivery addresses, Which means Addresses are associated to User a **One-to-Many** relationship, so loading the Address of an user when it is required is called the lazy loading. Since Hibernate 3, lazy loading is enabled by default you don't need to do lazy="true".

Also note that @OneToMany and @ManyToMany associations are defaulted to **LAZY loading**; and @OneToOne and @ManyToOne are defaulted to **EAGER loading**. This is important to remember to avoid any pitfall in future.

Lazy loading in hibernate improves the performance. It loads the child objects on demand.

24. What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write query

25. What is the difference between first level cache and second level cache?

No.	First Level Cache	Second Level Cache
1)	First Level Cache is **associated with Session**.	Second Level Cache is associated with **SessionFactory**.
2)	It is **enabled** by default.	It is **not enabled** by default.

26. What the two key components are of hibernate configuration object?

The Configuration object provides two keys components:

a) <u>**Database Connection:**</u> This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.

b) <u>**Class Mapping Setup:**</u> This component creates the connection between the Java classes and database tables.

27. What is the purpose of Session.beginTransaction() method?

Session.beginTransaction method begins a unit of work and returns the associated Transaction object.

28. Which method is used to add a criteria to a query?

Session.createCriteria creates a new Criteria instance, for the given entity class, or a superclass of an entity class.

29. **Which method is used to create a HQL query?**
Session.createQuery creates a new instance of Query for the given HQL query string

30. **Which method is used to create a SQL query?**
Session.createSQLQuery creates a new instance of SQLQuery for the given SQL query string.

31. **Which method is used to remove a persistent instance from the datastore?**
Session.delete removes a persistent instance from the datastore.

32. **Which method is used to get a persistent instance from the datastore?**
Session.get returns the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.

33. **What is named SQL query in Hibernate?**
Named queries are SQL queries which are defined in mapping document using <sql-query> tag and called using Session.getNamedQuery() method. Named query allows you to refer a particular query by the name you provided, by the way, you can define named query in hibernate either by using annotations or XML mapping file, as I said above. @NameQuery is used to define single named query and @NameQueries is used to define multiple named query in hibernate

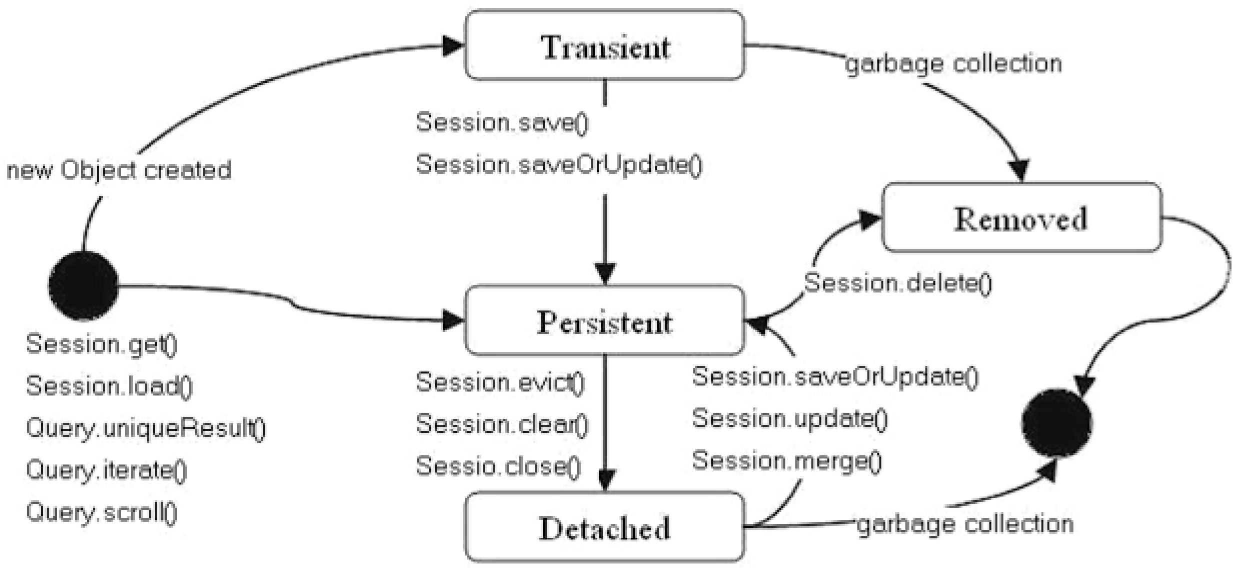
34. **What is SessionFactory in Hibernate? Is SessionFactory thread-safe?**
SessionFactory, as the name suggest, is a factory to hibernate Session objects. SessionFactory is often built during start-up and used by application code to get session object. It acts as a single data store and it's also thread-safe so that multiple threads can use the same SessionFactory. Usually, a Java JEE application has just one SessionFactory, and individual threads, which are servicing client’s request obtain hibernate Session instances from this factory, that’s why any implementation of SessionFactory interface must be thread-safe. Also, the internal state of SessionFactory, which contains all metadata about Object/Relational mapping is Immutable and cannot be changed once created.

35. **What is Session in Hibernate? Can we share single Session among multiple threads in Hibernate?**
After SessionFactory its time for Session. Session represent a small unit of work in Hibernate, they maintain a connection with the database and they are not thread-safe, it means you cannot share Hibernate Session between multiple threads. Though Session obtains database connection lazily it's good to close session as soon as you are done with it.

36. **What is the difference between sorted and ordered collection in hibernate?**
A sorted collection is sorted in memory by using Java Comparator while an ordered collection uses database's order by clause for ordering. For large data set it's better to use ordered collection to avoid any OutOfMemoryError in Java, by trying to sort them in memory.

37. **What is the difference between transient, a persistent and detached object in Hibernate?**
In Hibernate, Object can remain in three state transient, persistent or detached. An object which is associated with Hibernate session is called persistent object. Any change in this object will reflect in the database based on your flush strategy i.e. automatic flush whenever any property of object change or

explicit flushing by calling Session.flush() method. On the other hand, if an object which is earlier associated with Session, but currently not associated with it are called detached object.



38. **What does Session lock() method do in Hibernate OR what is difference between Session's lock() and update() method?**
Session's lock() method reattach object without synchronizing or updating with the database. So you need to be very careful while using lock() method. By the way, you can always use Session's update() method to sync with the database during reattachment.

39. **What is Second-level Cache in Hibernate?**
Second-level Cache is maintained at SessionFactory level and can improve performance by saving few database round trip. Another worth noting point is that second level cache is available to the whole application rather than any particular session.

40. **Why it's important to provide no argument constructor in Hibernate Entities?**
Every Hibernate Entity class must contain a no argument constructor, because Hibernate framework creates an instance of them using Reflection API, by calling Class.newInstance() method. This method will throw InstantiationException if it doesn't found any argument constructor inside Entity class.

41. **What is query cache in Hibernate?**
QueryCache actually stores the result of SQL query for future calls. Query cache can be used along with second level cache for improved performance. Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

42. **Can we make a Hibernate Entity Class final?**
Yes, you can make a Hibernate Entity class final, but that's not a good practice. Since Hibernate uses a proxy pattern for performance improvement in the case of the lazy association, by making an entity final, Hibernate will no longer be able to use a proxy, because Java doesn't allow extension of the final class, thus limiting your performance improvement options. Though, you can avoid this penalty if your

persistent class is an implementation of an interface, which declares all public methods defined in the Entity class.

43. Java Annotation (is the part of JAVA 5) – replaces the mapping XML file

Some of the example Annotation are below:

```
@Entity
@Table(name = "EMPLOYEE")

@Id @GeneratedValue
@Column(name = "id")
```

Package: javax.persistence

44. What jars are to be needed for Hibernate Annotation?

a) hibernate-commons-annotations.jar
b) ejb3-persistence.jar
c) hibernate-annotations.jar

45. What are the core annotation used in hibernate?

a) @Entity is used to mark the class as an Entity Bean. Class should have at least a package scope no-argument constructor.
b) Table is used to specify the table to persist the data. name attribute refers table name.
c) If @Table is not defined then Hibernate will by default use class name as table name.
d) @Id is the identifier property of Entity Bean. It determine the default access strategy that hibernate will use for mapping. If it is placed over field - field access will use.
e) If it is placed over setter method – property access will use.
f) @GeneratedValue is used to specify primary key generation strategy. By default Auto is used.
g) @Column is used to specify a column to which a field / property will mapped. If not defined by default property name will be used as column name.

46. When to use Annotation?

a. If we have flexibility to use Java 5 Environment
b. If we know which production database will use. In order to support multiple database it's better to have mapping information into separate xml files rather than using Annotations. We can have multiple xml files each specific to particular database.

47. What is HQL?

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries.

48. What are the Named Paramters?

Hibernate supports named parameters in its HQL queries. This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks. Following is the simple syntax of using named parameters:

For Ex:

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
```

```
query.setParameter("employee_id",10);
List results = query.list();
```

49. Pagination using Query

There are two methods of the Query interface for pagination.

a) **Query setFirstResult(int startPosition)**
   This method takes an integer that represents the first row in your result set, starting with row 0.

b) **Query setMaxResults(int maxResult)**
   This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

50. Criteria

What is Criteria?

Criteria is used to retrieve/update the data based on some criteria.

1. We can add Restriction to the criteria using add() method
```
Criteria cr = session.createCriteria(Employee.class);
cr.add(Restrictions.eq("salary", 2000));
List results = cr.list();
```
2. You can create AND or OR conditions using LogicalExpression restrictions as follows:
3. Pagination using Criteria
4. Sorting the Results
```
// To sort records in descening order
crit.addOrder(Order.desc("salary"));
```

51. Caching

What is Caching?

Caching is all about application performance optimization and it sits between your application and the database to avoid the number of database hits as many as possible to give a better performance for performance critical applications.

a) First-level cache:

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass.
The Session object keeps an object under its own power before committing it to the database.
If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.

b) Second-level cache:

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.
The Second Level Cache: Can be enabled by placing property in hbm (hibernate mapping file) as below, to use second level we need to decide based on some concurrency strategy

<cache usage="read-write"/>

Any third-party cache can be used with Hibernate. An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

c) **Query-level cache:**
Hibernate also implements a cache for query result sets that integrates closely with the second-level cache. To use query level cache we need to set the following tag in the **hibernate.cfg.xml file**

```
use_query_cache="true"
```

Or in java code as

```
Session session = SessionFactory.openSession();
Query query = session.createQuery("FROM EMPLOYEE");
query.setCacheable(true);
query.setCacheRegion("employee");
List users = query.list();
SessionFactory.closeSession();
```

52. Batch Processing

Consider a situation when you need to upload a large number of records into your database using Hibernate. Following is the code snippet to achieve this using Hibernate:

Because by default, Hibernate will cache all the persisted objects in the session-level cache and ultimately your application would fall over with an OutOfMemoryException somewhere around the 50,000th row. You can resolve this problem if you are using batch processing with Hibernate.

To enable batch processing in configuration xml file as below: in **hibernate.cfg.xml**

```
<property name="hibernate.jdbc.batch_size">
    50
</property>
```

To use the batch processing feature, first set **hibernate.jdbc.batch_size** as batch size to a number either at 20 or 50 depending on object size. This will tell the hibernate container that every X rows to be inserted as batch. To implement this in your code we would need to do little modification as follows:

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
   Employee employee = new Employee(.....);
   session.save(employee);
        if( i % 50 == 0 ) { // Same as the JDBC batch size
      //flush a batch of inserts and release memory:
      session.flush();
      session.clear();
   }
}
```

```
tx.commit();
session.close();
```

flush() and **clear()** methods available with Session object so that Hibernate keep writing these records into the database instead of caching them in the memory.

53. Hibernate call store procedure

In Hibernate, there are three approaches to call a database store procedure.

1. Native SQL – createSQLQuery
```
Query query = session.createSQLQuery(
        "CALL GetStocks(:stockCode)")
        .addEntity(Stock.class)
        .setParameter("stockCode", "7277");

List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

2. NamedNativeQuery in annotation
```
//Stock.java
...
@NamedNativeQueries({
        @NamedNativeQuery(
        name = "callStockStoreProcedure",
        query = "CALL GetStocks(:stockCode)",
        resultClass = Stock.class
        )
})

@Entity
@Table(name = "stock")
public class Stock implements java.io.Serializable {
```
Call it with **getNamedQuery()**.
```
Query query = session.getNamedQuery("callStockStoreProcedure")        .setParameter("stockCode",
"7277");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```
3. sql-query in XML mapping file

Declare your store procedure inside the "sql-query" tag.
```
<hibernate-mapping>
    <class name="com.mkyong.common.Stock" table="stock" ...>
        <id name="stockId" type="java.lang.Integer">
            <column name="STOCK_ID" />
```

```
                <generator class="identity" />
            </id>
            <property name="stockCode" type="string">
                <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
            </property>
            ...
        </class>

        <sql-query name="callStockStoreProcedure">
             <return alias="stock" class="com.mkyong.common.Stock"/>
             <![CDATA[CALL GetStocks(:stockCode)]]>
        </sql-query>

    </hibernate-mapping>
```

Call it with getNamedQuery().

```
Query query = session.getNamedQuery("callStockStoreProcedure")      .setParameter("stockCode",
"7277");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

Servlet

1. What is the Servlet?

a) Servlet is a technology i.e. used to create web application.
b) Servlet is an API that provides many interfaces and classes including documentations.
c) Servlet is an interface that must be implemented for creating any servlet.
d) Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
e) Servlet is a web component that is deployed on the server to create dynamic web page.

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

Following are the changes introduced in Servlet 2.5:

a) A new dependency on J2SE 5.0
b) Support for annotations
c) Loading the class
d) Several web.xml conveniences
e) A handful of removed restrictions
f) Some edge case clarifications

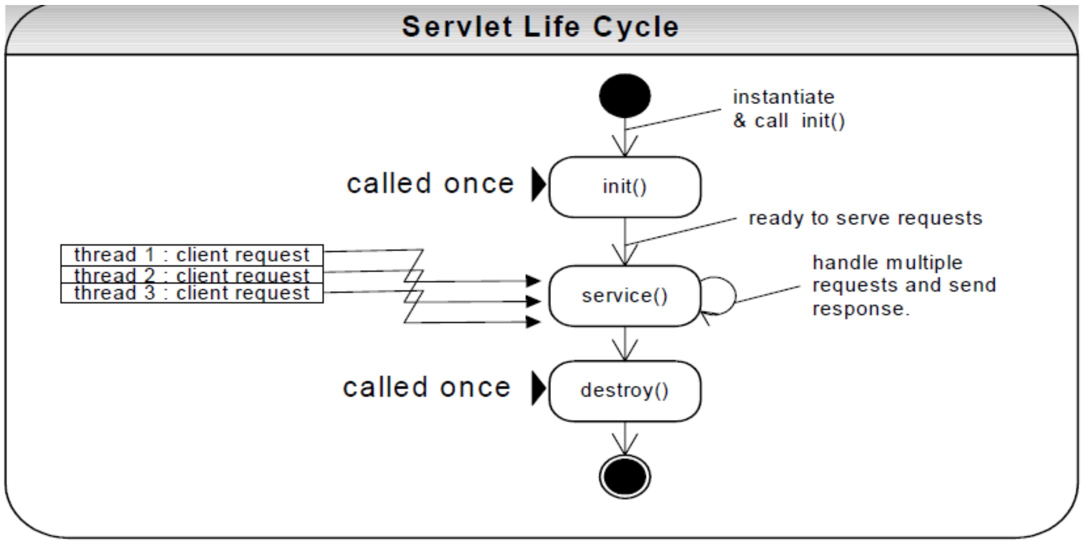
3. *What are the uses of Servlet?*

Typical uses for HTTP Servlets include:

a) Processing and/or storing data submitted by an HTML form.
b) Providing dynamic content, e.g. returning the results of a database query to the client.
c) A Servlet can handle multiple request concurrently and be used to develop high performance system
d) 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 phases of the servlet life cycle?*

The Web container is responsible for managing the servlet's life cycle. The Web container creates an instance of the servlet and then the container calls the init() method. At the completion of the init() method the servlet is in ready state to service requests from clients. The container calls the servlet's service() method for handling each request by spawning a new thread for each request from the Web container's thread pool. Before destroying the instance the container will call the destroy() method. After destroy() the servlet becomes the potential candidate for garbage collection.


Servlet Life Cycle
instantiate & call init()
called once
init()
ready to serve requests
thread 1 : client request
thread 2 : client request
thread 3 : client request
service()
handle multiple requests and send response.
called once
destroy()

5. *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. Just like a normal POJO 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.

6. *How the servlet is loaded?*

A servlet can be loaded when:

a) First request is made.

b) Server starts up (auto-load).

c) Administrator manually loads.

7. *How a Servlet is unloaded?*

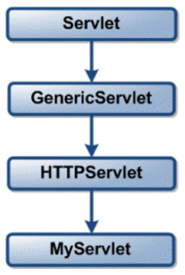
A servlet is unloaded when:

a) Server shuts down.

b) Administrator manually unloads.

8. *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.

9. *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.

10. *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 a 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.

11. Why 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.

12. Can servlet have a constructor?

One can definitely have constructor in servlet. Even you can use the constructor in servlet for initialization purpose, but this type of approach 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.

13. 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.

14. 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 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 change anything on the server. (doGet() should be idempotent)	doPost() provides information 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.

15. 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:

a) If data is sensitive
b) Data is greater than 1024 characters
c) If your application don't need bookmarks.

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

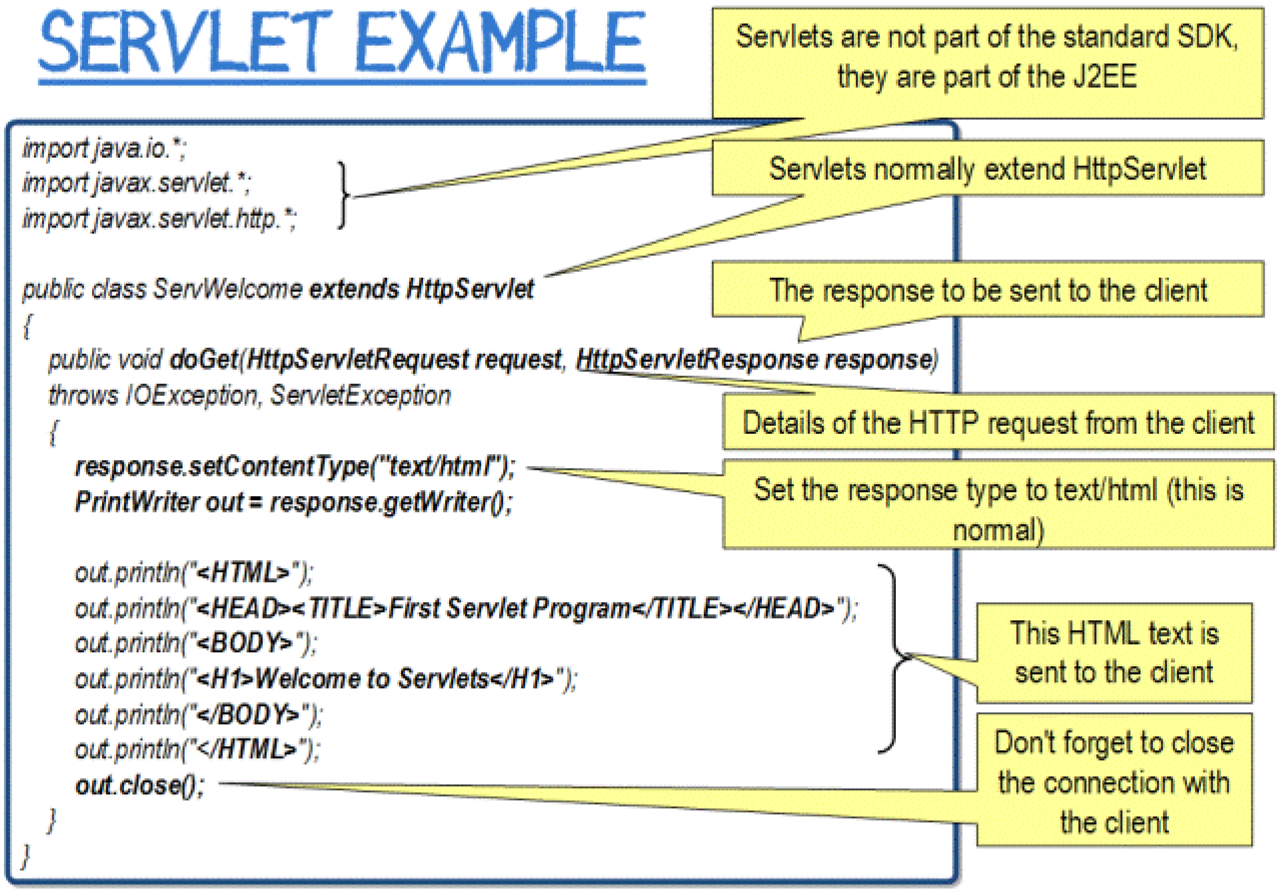
The easy way is, just support POST, then have your doGet method call your doPost method:

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

17. 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 methods 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.

18. How the typical servlet codes 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();
  }
}
```

19. 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.

20. *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.

21. *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 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	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.

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

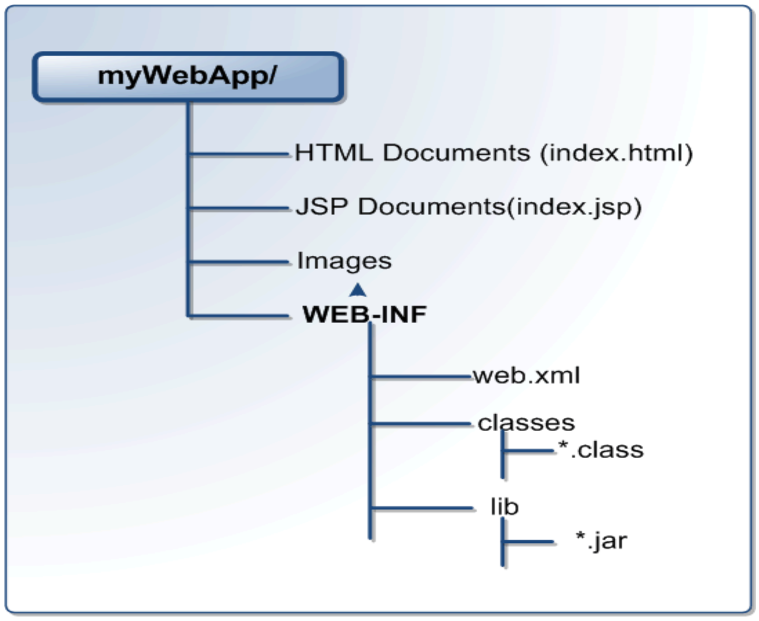
include()	forward()

The `RequestDispatcher include` method inserts 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 response status code or HTTP headers, any such request will be ignored.	The forwarded resource may be another servlet, JSP or static HTML document, but the 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.

23. *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 behavior is to pass all method calls directly to the underlying objects.

24. *What is the directory structure of a WAR file?*



25. *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.

26. *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.

27. What is pre-initialization 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 pre-initializing a servlet**.

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

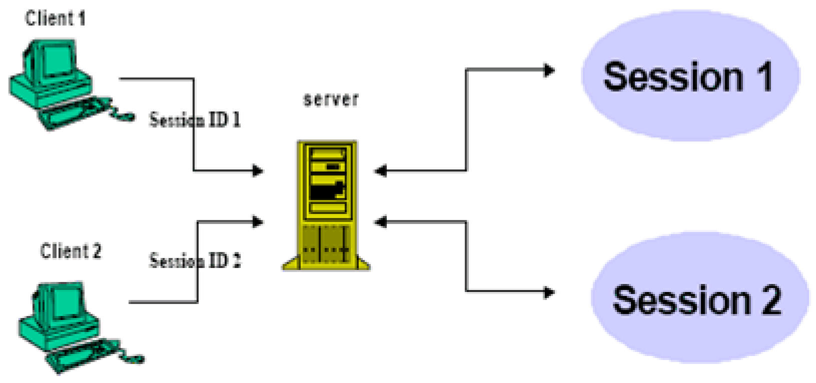
The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

```
<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**.**

29. 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.



30. What is Session Tracking?

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

31. What is the need of Session tracking in web application?

HTTP is a stateless protocol that is 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 referred 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.

32. 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:

a) **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.

b) **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.

c) **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.

d) **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.

33. 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.

a) **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.

b) To get all of the cookies associated with a single HTTP request, use the getCookies() method of class HttpServletRequest

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

a) 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.

b) For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

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

a) Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.

b) 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.

c) Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, and 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.

d) Browser instances share cookies, so users cannot have multiple simultaneous sessions.

e) 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.

36. 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**

a) URL rewriting works just about everywhere, especially when cookies are turned off.
b) 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.
c) 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**

a) 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.
b) URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
c) URL rewriting does not work well with JSP technology.
d) If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

37. How can an existing session be invalidated?

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

a) **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.

b) **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 as negative (-1), makes the container stop tracking session, i.e, sesssion never expires.**

38. How can session in Servlet can be destroyed?

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

a) Programmatically: Using `session.invalidate()` method, which makes the container abandon the session on which the method is called.
b) When the server itself is shutdown.

39. 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.

40. What is servlet lazy loading?

a) 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.

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

41. 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.

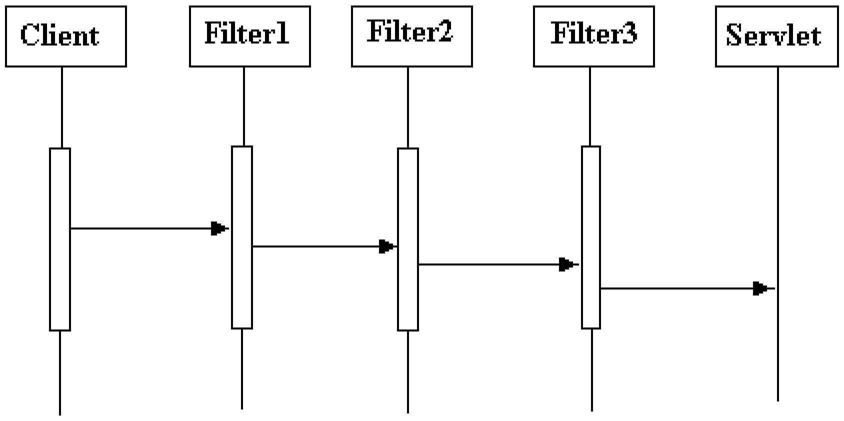
42. How are filters?

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:
a) Security checks
b) Modifying the request or response
c) Data compression
d) Logging and auditing
e) 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.

43. What are the functions of an intercepting filter?

The functions of an intercepting filter are as follows:
a) It intercepts the request from a client before it reaches the servlet and modifies the request if required.
b) It intercepts the response from the servlet back to the client and modifies the response if required.
c) 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.



44. What are the functions of the Servlet container?

The functions of the Servlet container are as follows:
a) **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.
b) **Communication support**: It handles the communication between the servlet and the Web server.
c) **Multithreading support**: It automatically creates a new thread for every servlet request received. When the Servlet service () method completes, the thread dies.
d) **Declarative security**: It manages the security inside the XML deployment descriptor file.
e) **JSP support**: The container is responsible for converting JSPs to servlets and for maintaining them.

JSP

1. What are the advantages of JSP over Servlet?

JSP is a server side 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:

a) Translation
b) Compilation
c) Loading the class
d) Instantiating the class
e) jspInit() invocation
f) _jspService() invocation
g) jspDestroy() invocation

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?

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

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.

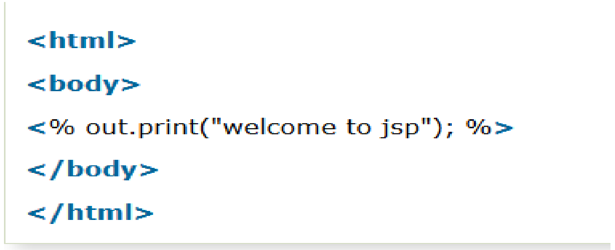
6. What JSP lifecycle methods can I override?

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. JSP Scriptlet tag (Scripting elements)

a) JSP Scriptlet tag

A scriptlet tag is used to execute java source code in JSP



```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

b) expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

c) Declaration tag

The **JSP declaration tag** is used *to declare fields and methods*. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

8. Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

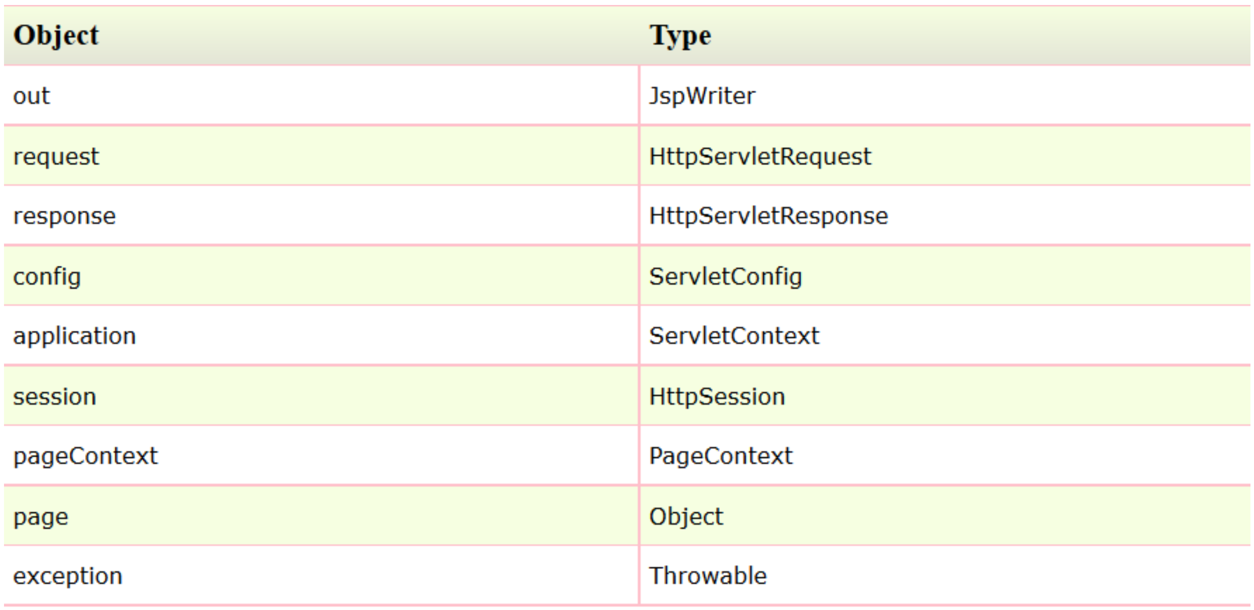
7. How can I override the jspInit() and jspDestroy() methods within a JSP page?

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:



Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

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.

a) **Request:** The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc. It can also be used to set, get and remove attributes from the jsp request scope.

b) **Response:** In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request. It can be used to add or manipulate response such as redirect response to another resource, send error etc.

c) **Config**: In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page. Generally, it is used to get initialization parameter from the web.xml file.

d) **Application**: In JSP, application is an implicit object of type *ServletContext*. The instance of ServletContext is created only once by the web container when application or project is deployed on the server. This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope. This initialization parameter can be used by all jsp pages.

e) **Session**: In JSP, session is an implicit object of type HttpSession. The Java developer can use this object to set, get or remove attribute or to get session information.

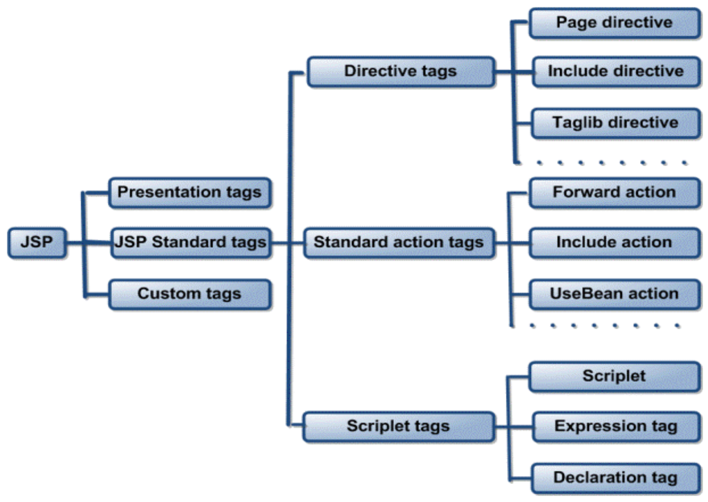
f) **pageContext**: In JSP, pageContext is an implicit object of type PageContext class. The pageContext object can be used to set, get or remove attribute from one of the page, request, session, application. In JSP, page scope is the default scope

g) **page**: In JSP, page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class. It is written as: Object page=this;

h) **Exception**: In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

9. What are the different types of JSP tags?

The different types of JSP tags are as follows:



10. What are JSP directives?

a) 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
b) They are used to set global values such as a class declaration, method implementation, output content type, etc.
c) They do not produce any output to the client.
d) Directives are always enclosed within <%@ ….. %> tag.
e) Ex: page directive, include directive, etc.

11. What is page directive?

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

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

13. What is the include directive?

a) The include directive is used to **statically** insert the contents of a resource into the current JSP.
b) This enables a user to reuse the code without duplicating it, and includes the contents of the specified file at the translation time.
c) The syntax of the include directive is as follows:
<%@ include file = "FileName" %>
d) 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:

a) <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.
b) <jsp:forward>: It forwards a response from a servlet or a JSP page to another page.
c) <jsp:useBean>: It makes a JavaBean available to a page and instantiates the bean.
d) <jsp:setProperty>: It sets the properties for a JavaBean.
e) <jsp:getProperty>: It gets the value of a property from a JavaBean component and adds it to the response.
f) <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.
g) <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:
a) **page scope:** It specifies that the object will be available for the entire JSP page but not outside the page.
b) **request scope**: It specifies that the object will be associated with a particular request and exist as long as the request exists.
c) **application scope**: It specifies that the object will be available throughout the entire Web application but not outside the application.
d) **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.

a) 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.

b) 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:setProperty> action. The container converts the property to a String as follows:

a) If it is an object, it uses the toString () method to convert it to a String.
b) If it is a primitive, it converts it directly to a String using the valueOf () method of the corresponding Wrapper class.
c) 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:

a) Expressions of the form **<%= expression %>** that are evaluated and inserted into the output,
b) Scriptlets of the form **<% code %>** that are inserted into the servlet's service method,
c) 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?
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 sub element 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>
```

JDBC

1. What is the JDBC?
Java Database Connectivity (**JDBC**) is a standard Java API to interact with relational databases from Java. **JDBC** has set of classes and interfaces which can use from Java application and talk to database without learning RDBMS details and using Database Specific JDBC Drivers.

2. What are the new features added to JDBC 4.0?
The major features added in JDBC 4.0 include:
a) Auto-loading of JDBC driver class
b) Connection management enhancements
c) Support for RowId SQL type
d) DataSet implementation of SQL using Annotations
e) SQL exception handling enhancements
f) SQL XML support

3. Explain Basic Steps in writing a Java program using JDBC?

JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database:

a) Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).
b) Open the connection to database which is then used to send SQL statements and get results back.
c) Create JDBC Statement object. This object contains SQL query.
d) Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
e) Process the result set.
f) Close the connection.

4. Explain the JDBC Architecture.

The JDBC Architecture consists of two layers:

a) The JDBC API, which provides the **application-to-JDBC Manager** connection.
b) The JDBC Driver API, which supports the **JDBC Manager-to-Driver** Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure 1.

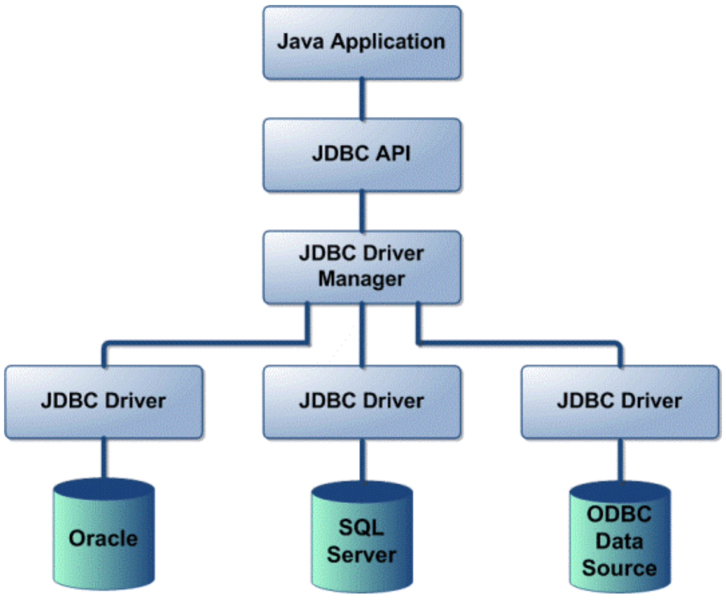


Figure 1: JDBC Architecture

5. What are the main components of JDBC?

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

a) **DriverManager**: Manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.
b) **Driver**: The database communications link, handling all communication with the database. Normally, once the driver is loaded, the developer need not call it explicitly.
c) **Connection:** Interface with all methods for contacting a database. The connection object represents communication context, i.e. all communication with database is through connection object only.
d) **Statement**: Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.

e) **ResultSet**: The ResultSet represents set of rows retrieved due to query execution.

6. How the JDBC application works?

A JDBC application can be logically divided into two layers:

a) Driver layer

b) Application layer

I. Driver layer consists of DriverManager class and the available JDBC drivers.

II. The application begins with requesting the DriverManager for the connection.

III. An appropriate driver is chosen and is used for establishing the connection. This connection is given to the application which falls under the application layer.

IV. The application uses this connection to create Statement kind of objects, through which SQL commands are sent to backend and obtain the results.

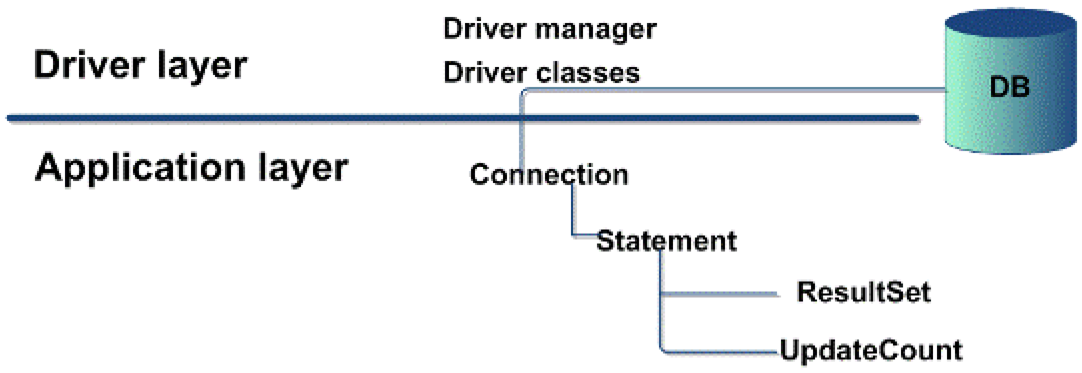


Figure 2: JDBC Application

7. How do I load a database driver with JDBC 4.0 / Java 6?

Provided the JAR file containing the driver is properly configured, just place the JAR file in the classpath. Java developers no longer need to explicitly load JDBC drivers using code like Class.forName () to register a JDBC driver. The DriverManager class takes care of this by automatically locating a suitable driver when the DriverManager.getConnection () method is called. This feature is backward-compatible, so no changes are needed to the existing JDBC code.

8. What is JDBC Driver interface?

The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendor driver must provide implementations of the java.sql.Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

9. What does the connection object represents?

The connection object represents communication context, i.e., all communication with database is through connection object only.

10. What is Statement?

Statement acts like a vehicle through which SQL commands can be sent through the connection object we create statement kind of objects.

```
Statement stmt = conn.createStatement();
```

This method returns object which implements statement interface.

11. What is PreparedStatement?

A prepared statement is an SQL statement that is precompiled by the database. Through precompilation, prepared statements improve the performance of SQL commands that are executed multiple times. Once compiled, prepared statements can be customized prior to each execution by altering predefined SQL parameters.

```
PreparedStatement pstmt = conn.prepareStatement ("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");
                pstmt.setBigDecimal (1, 153833.00);
                pstmt.setInt (2, 110592);
```

Here: ***conn*** *is an instance of the Connection class and "****?****" represents parameters. These parameters must be specified before execution.*

12. What is the difference between a Statement and a PreparedStatement?

Statement	PreparedStatement
A standard Statement is used to create a Java representation of a literal SQL statement and execute it on the database.	A PreparedStatement is a precompiled statement. This means that when the PreparedStatement is executed, the RDBMS can just run the PreparedStatement SQL statement without having to compile it first.
Statement has to verify its metadata against the database every time.	While a prepared statement has to verify its metadata against the database only once.
If you want to execute the SQL statement once go for STATEMENT	If you want to execute a single SQL statement multiple number of times, then go for PREPAREDSTATEMENT. PreparedStatement objects can be reused with passing different values to the queries

13. What are callable statements?
Callable statements are used from JDBC application to invoke stored procedures and functions.

14. How to call a stored procedure from JDBC?
PL/SQL stored procedures are called from within JDBC programs by means of the prepareCall () method of the Connection object created. A call to this method takes variable bind parameters as input parameters as well as output variables and creates an object instance of the CallableStatement class.

The following line of code illustrates this:

```
        CallableStatement stproc_stmt = conn.prepareCall("{call procname(?,?,?)}");
```

Here conn is an instance of the Connection class.

15. What are types of JDBC drivers?
There are four types of drivers defined by JDBC as follows:

a) **Type 1: JDBC/ODBC**— These require an ODBC (Open Database Connectivity) driver for the database to be installed. This type of driver works by translating the submitted queries into equivalent ODBC queries and forwards them via native API calls directly to the ODBC driver. It provides no host redirection capability.
b) **Type2: Native API (partly-Java driver)** — This type of driver uses a vendor-specific driver or database API to interact with the database. An example of such an API is Oracle OCI (Oracle Call Interface). It also provides no host redirection.

c) **Type 3: Open Protocol-Net**— This is not vendor specific and works by forwarding database requests to a remote database source using a net server component. How the net server component accesses the database is transparent to the client. The client driver communicates with the net server using a database-independent protocol and the net server translates this protocol into database calls. This type of driver can access any database.

d) **Type 4: Proprietary Protocol-Net(pure Java driver)**—This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database. Again this is all transparent to the client.

Note: *Type 4 JDBC driver is most preferred kind of approach in JDBC.*

16. Which type of JDBC driver is the fastest one?

JDBC Net pure Java driver (Type IV) is the fastest driver because it converts the JDBC calls into vendor specific protocol calls and it directly interacts with the database.

17. Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?

No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.

18. Which is the right type of driver to use and when?

a) Type I driver is handy for prototyping
b) Type III driver adds security, caching, and connection control
c) Type III and Type IV drivers need no pre-installation

19. What are the standard isolation levels defined by JDBC?

The values are defined in the class java.sql.Connection and are:

a) TRANSACTION_NONE
b) TRANSACTION_READ_COMMITTED
c) TRANSACTION_READ_UNCOMMITTED
d) TRANSACTION_REPEATABLE_READ
e) TRANSACTION_SERIALIZABLE

Any given database may not support all of these levels.

20. What is resultset?

The ResultSet represents set of rows retrieved due to query execution.

```
ResultSet rs = stmt.executeQuery(sqlQuery);
```

21. What are the types of resultsets?

The values are defined in the class java.sql.Connection and are:

a) **TYPE_FORWARD_ONLY** specifies that a resultset is not scrollable, that is, rows within it can be advanced only in the forward direction.

b) **TYPE_SCROLL_INSENSITIVE** specifies that a resultset is scrollable in either direction but is insensitive to changes committed by other transactions or other statements in the same transaction.

c) **TYPE_SCROLL_SENSITIVE** specifies that a resultset is scrollable in either direction and is affected by changes committed by other transactions or statements within the same transaction.

Note*: A TYPE_FORWARD_ONLY resultset is always insensitive.*

22. What's the difference between TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE?

TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE

An insensitive resultset is like the snapshot of the data in the database when query was executed.	A sensitive resultset does NOT represent a snapshot of data, rather it contains points to those rows which satisfy the query condition.
After we get the resultset the changes made to data are not visible through the resultset, and hence they are known as insensitive.	After we obtain the resultset if the data is modified then such modifications are visible through resultset.
Performance not effected with insensitive.	Since a trip is made for every **'get'** operation, the performance drastically get affected.

22. What is rowset?
A RowSet is an object that encapsulates a set of rows from either Java Database Connectivity (JDBC) result sets or tabular data sources like a file or spreadsheet. RowSets support component-based development models like JavaBeans, with a standard set of properties and an event notification mechanism.

24. What are the different types of RowSet?
There are two types of RowSet are there. They are:

a) **Connected** - A connected RowSet object connects to the database once and remains connected until the application terminates.
b) **Disconnected -** A disconnected RowSet object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected RowSet while it is disconnected. Modified data can be updated in the database after a disconnected RowSet reestablishes the connection with the database.

25. What is the need of BatchUpdates?
The BatchUpdates feature allows us to group SQL statements together and send to database server in one single trip.

26. What is a DataSource?
A DataSource object is the representation of a data source in the Java programming language. In basic terms,

a) A DataSource is a facility for storing data.
b) DataSource can be referenced by JNDI.
c) Data Source may point to RDBMS, file System, any DBMS etc.

27. What are the advantages of DataSource?
The few advantages of data source are:

a) An application does not need to hardcode driver information, as it does with the DriverManager.
b) The DataSource implementations can easily change the properties of data sources. *For example*: There is no need to modify the application code when making changes to the database details.
c) The DataSource facility allows developers to implement a DataSource class to take advantage of features like connection pooling and distributed transactions.

28. What is connection pooling? What is the main advantage of using connection pooling?
A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested.

29. Explain differences among java.util.Date, java.sql.Date, java.sql.Time, and java.sql.Timestamp?
As shown below all the sql Date classes extend the util Date class.

a) java.util.Date - class supports both the Date (i.e. year/month/date etc) and the Time (hour, minute, second, and millisecond) components.

b) java.sql.Date - class supports only the Date (i.e. year/month/date etc) component. The hours, minutes, seconds and milliseconds of the Time component will be set to zero in the particular time zone with which the instance is associated.
c) java.sql.Time - class supports only Time (i.e. hour, minute, second, and millisecond) component. The date components should be set to the "zero epoch" value of January 1, 1970 and should not be accessed.
d) java.sql.TimeStamp : class supports both Date (i.e. year/month/date etc) and the Time (hour, minute, second, millisecond and nanosecond) components.

30. Difference between DataSource vs DriverManager

A DataSource object is the representation of a data source in the Java programming language. In basic terms, a data source is a facility for storing data. It can be as sophisticated as a complex database for a large corporation or as simple as a file with rows and columns. A data source can reside on a remote server, or it can be on a local desktop machine. Applications access a data source using a connection, and a DataSource object can be thought of as a factory for connections to the particular data source that the DataSource instance represents. The DataSource interface provides two methods for establishing a connection with a data source.

Using a DataSource object is the preferred alternative to using the DriverManager for establishing a connection to a data source. They are similar to the extent that the DriverManager class and DataSource interface both have methods for creating a connection, methods for getting and setting a timeout limit for making a connection, and methods for getting and setting a stream for logging.

Their differences are more significant than their similarities, however. Unlike the DriverManager, a DataSource object has properties that identify and describe the data source it represents. Also, DataSource object works with a Java Naming and Directory Interface (JNDI) naming service and is created, deployed, and managed separately from the applications that use it. A driver vendor will provide a class that is a basic implementation of the DataSource interface as part of its JDBC 2.0 or 3.0 driver products. What a system administrator does to register a DataSource object with a JNDI naming service and what an application does to get a connection to a data source using a DataSource object registered with a JNDI naming service are described later in this chapter.

Being registered with a JNDI naming service gives a DataSource object two major advantages over the DriverManager. First, an application does not need to hardcode driver information, as it does with the DriverManager. A programmer can choose a logical name for the data source and register the logical name with a JNDI naming service. The application uses the logical name, and the JNDI naming service will supply the DataSource object associated with the logical name. The DataSource object can then be used to create a connection to the data source it represents.

The second major advantage is that the DataSource facility allows developers to implement a DataSource class to take advantage of features like connection pooling and distributed transactions. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested. The ability to use distributed transactions enables an application to do the heavy duty database work of large enterprises.

Although an application may use either the DriverManager or a DataSource object to get a connection, using a DataSource object offers significant advantages and is the recommended way to establish a connection.

Delloite: -

1. Design Pattern

2. Singleton Program
3. How you will write and call custom Exception class in your web component
4. How you will ensure that your custom exceptions that you have written is a compile time exception or run time exception
5. Throw and Throws
6. What are the JDBC Drivers
7. Steps to create JDBC connection
8. If you have frequent data changes operation then what you will use in terms of List, Set and Map and why?
9. Why you use Interface? And why you use Abstraction
10. How many levels you can go in Abstraction to define Abstract method in it.
11. What are the advantages of using Abstraction and Interface other than common Behavior and common contract?
12. JSP Include Action and Include directive
13. What will happen if you define a new class in your JSP what the compiler will do?
14. Synchronized Thread.
15. Given a List [a,b,c,d,a,b,d,r] write the efficient way to find the unique values in it.
16. What is a Session and how you will create a Session.
17. Inner Join and Self Join, explain with Example.
18. Have you work on Struts/Spring/
19. What is the DOM and SAX
20. What is the diff between DOM and SAX parser
21. Web Service.
22. What are the Inner classes and Anonymous class?
23. Reflection
24. JSP Life cycle
25. What do you mean by a Transition in JSP lifecycle?

When to use Interface and when Abstraction

1. If I have only the requirement, and we don't know anything about implementation then go for the Interface: Example **Servlet** itself interface. Which means we have only the specification.
2. If we know the partial implementation, then we should go with the Abstract Ex. Generic Servlet && HttpServlet

Honeywell: -

1) If you have a static variable in your class. You have created new class and create an instance of that class is null
Then what will be the value of the variable.
e.g: if you have Employee class and Static Id variable. You have created new class Address then you have instantiated Employee class which is equal to null then what will be the value of Id variable?

2) How do you instantiate a class using fully qualified class name as a string?

3) Create Employee and Address class, Using Map add elements in it and iterate through it.

4) Method overloading and overriding example.
    Suppose testOveridingMethod(int id, string Name) and testOverride(int age, String MName);

When I call it with testOveridingMethod(1, "abc") which method will be called?

5) Can we declare Protected variable in Interface?
6) Can we create an instance of the Abstract class?
==>> An abstract class can never be instantiated. Its sole purpose is to be extended (sub classed).

7) What are the differences between Interface and Abstract class?

8) Exception Handling: In try block you are having an arithmetic operation such as divide by zero and you have two catch blocks, 1st catch block declare java exception and 2nd catch block declare custom exception which catch block will be executed?

9) How to make a Java class Serializable practical implementation for example Employee class?

10) Suppose Employee is parent class and 3rd party class you cannot modified it then how will you make variables in parent class as transient?

JPMC

Mphasis

Sapient

LnT