Q: What is the difference between an Interface and an Abstract class?

A: An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

Q: What is the purpose of garbage collection in Java, and when is it used?

A: The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

Q: Describe synchronization in respect to multithreading.

A: With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

Q: Explain different way of using thread?

A: The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, 'cause when you are going for multiple inheritance the only interface can help.

Q: What are pass by reference and passby value?

A: Pass By Reference means the passing the address itself rather than passing the value. Passby Value means passing a copy of the value to be passed.

Q: What is HashMap and Map?

A: Map is Interface and Hashmap is class that implements that.

Q: Difference between HashMap and HashTable?

A: The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesnt allow). HashMap does not guarantee that the order of the map will remain constant over time. HashMap is unsynchronized and Hashtable is synchronized.

Q: Difference between Vector and ArrayList?

A: Vector is synchronized whereas arraylist is not.

Q: Difference between Swing and Awt?

A: AWT are heavy-weight componenets. Swings are light-weight components. Hence swing works faster than AWT.

Q: What is the difference between a constructor and a method?

A: A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the new operator. A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator.

Q: What is an Iterator?

A: Some of the collection classes provide traversal of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn. Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

Q: State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.

A: ***public :*** Public class is visible in other packages, field is visible everywhere (class must be public too)

private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.

protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.
default :What you get by default ie, without any access modifier (ie, public private or protected).It means that it is visible to all within a particular package.

Q: What is an abstract class?

A: Abstract class must be extended/subclassed (to be useful). It serves as a template. A class that is abstract may not be instantiated (ie, you may not call its constructor), abstract class may contain static data. Any class with an abstract method is automatically abstract itself, and must be declared as such. A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.

Q: What is static in java?

A: Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class. Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object. A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a non-static method. In other words, you can't change a static method into an instance method in a subclass.

Q: What is final?

A: A final class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant)

Q: What if the main method is declared as private?

A: The program compiles properly but at runtime it will give "Main method not public." message.

Q: What if the static modifier is removed from the signature of the main method?

A: Program compiles. But at runtime throws an error "NoSuchMethodError".

Q: What if I write static public void instead of public static void?

A: Program compiles and runs properly.

Q: What if I do not provide the String array as the argument to the method?

A: Program compiles but throws a runtime error "NoSuchMethodError".

Q: What is the first argument of the String array in main method?

A: The String array is empty. It does not have any element. This is unlike C/C++ where the first element by default is the program name.

Q: If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?

A: It is empty. But not null.

Q: How can one prove that the array is not null but empty using one line of code?

A: Print args.length. It will print 0. That means it is empty. But if it would have been null then it would have thrown a NullPointerException on attempting to print args.length.

Q: What environment variables do I need to set on my machine in order to be able to run Java programs?

A: CLASSPATH and PATH are the two variables.

Q: Can an application have multiple classes having main method?

A: Yes it is possible. While starting the application we mention the class name to be run. The JVM will look for the Main method only in the class whose name you have mentioned. Hence there is not conflict amongst the multiple classes having main method.

Q: Can I have multiple main methods in the same class?

A: No the program fails to compile. The compiler says that the main method is already defined in the class.

Q: Do I need to import java.lang package any time? Why ?

A: No. It is by default loaded internally by the JVM.

Q: Can I import same package/class twice? Will the JVM load the package twice at runtime?

A: One can import the same package or same class multiple times. Neither compiler nor JVM complains abt it. And the JVM will internally load the class only once no matter how many times you import the same class.

Q: What are Checked and UnChecked Exception?

A: A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown. eg, IOException thrown by java.io.FileInputStream's read() method· Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. eg, StringIndexOutOfBoundsException thrown by String's charAt() method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be.

Q: What is Overriding?

A: When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass. When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass. Methods may be overridden to be more public, not more private.

Q: What are different types of inner classes?

A: *Nested top-level classes*, *Member classes, Local classes, Anonymous classes*

Nested top-level classes- If you declare a class within a class and specify the static modifier, the compiler treats the class just like any other top-level class. Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package. eg, outer.inner. Top-level inner classes implicitly have access only to static variables.There can also be inner interfaces. All of these are of the nested top-level variety.

Member classes - Member inner classes are just like other member methods and member variables and access to the member class is restricted, just like methods and variables. This means a public member class acts similarly to a nested top-level class. The primary difference between member classes and nested top-level classes is that member classes have access to the specific instance of the enclosing class.

Local classes - Local classes are like local variables, specific to a block of code. Their visibility is only within the block of their declaration. In order for the class to be useful beyond the declaration block, it would need to implement a more publicly available interface.Because local classes are not members, the modifiers public, protected, private, and static are not usable.

Anonymous classes - Anonymous inner classes extend local inner classes one level further. As anonymous classes have no name, you cannot provide a constructor.

Q: **Are the imports checked for validity at compile time? e.g. will the code containing an import such as java.lang.ABCD compile?**
A: Yes the imports are checked for the semantic validity at compile time. The code containing above line of import will not compile. It will throw an error saying,can not resolve symbol
symbol : class ABCD
location: package io
import java.io.ABCD;

Q: **Does importing a package imports the subpackages as well? e.g. Does importing com.MyTest.* also import com.MyTest.UnitTests.*?**
A: No you will have to import the subpackages explicitly. Importing com.MyTest.* will import classes in the package MyTest only. It will not import any class in any of it's subpackage.

Q: **What is the difference between declaring a variable and defining a variable?**
A: In declaration we just mention the type of the variable and it's name. We do not initialize it. But defining means declaration + initialization.
e.g String s; is just a declaration while String s = new String ("abcd"); Or String s = "abcd"; are both definitions.

Q: **What is the default value of an object reference declared as an instance variable?**
A: null unless we define it explicitly.
Q: **Can a top level class be private or protected?**
A: No. A top level class can not be private or protected. It can have either "public" or no modifier. If it does not have a modifier it is supposed to have a default access.If a top level

class is declared as private the compiler will complain that the "modifier private is not allowed here". This means that a top level class can not be private. Same is the case with protected.

Q: **What type of parameter passing does Java support?**
A: In Java the arguments are always passed by value .

Q: **Primitive data types are passed by reference or pass by value?**
A: Primitive data types are passed by value.

Q: **Objects are passed by value or by reference?**
A: Java only supports pass by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object .
Q: **What is serialization?**
A: Serialization is a mechanism by which you can save the state of an object by converting it to a byte stream.

Q: **How do I serialize an object to a file?**
A: The class whose instances are to be serialized should implement an interface Serializable. Then you pass the instance to the ObjectOutputStream which is connected to a fileoutputstream. This will save the object to a file.

Q: **Which methods of Serializable interface should I implement?**
A: The serializable interface is an empty interface, it does not contain any methods. So we do not implement any methods.

Q: **How can I customize the seralization process? i.e. how can one have a control over the serialization process?**
A: Yes it is possible to have control over serialization process. The class should implement Externalizable interface. This interface contains two methods namely readExternal and writeExternal. You should implement these methods and write the logic for customizing the serialization process.

Q: **What is the common usage of serialization?**
A: Whenever an object is to be sent over the network, objects need to be serialized. Moreover if the state of an object is to be saved, objects need to be serilazed.
Q: **What is Externalizable interface?**
A: Externalizable is an interface which contains two methods readExternal and writeExternal. These methods give you a control over the serialization mechanism. Thus if your class implements this interface, you can customize the serialization process by implementing these

methods.

Q: When you serialize an object, what happens to the object references included in the object?

A: The serialization mechanism generates an object graph for serialization. Thus it determines whether the included object references are serializable or not. This is a recursive process. Thus when an object is serialized, all the included objects are also serialized alongwith the original obect.

Q: What one should take care of while serializing the object?

A: One should make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

Q: What happens to the static fields of a class during serialization?

A: There are three exceptions in which serialization doesnot necessarily read and write to the stream. These are

1. Serialization ignores static fields, because they are not part of ay particular state state.
2. Base class fields are only hendled if the base class itself is serializable.
3. Transient fields.

Q: Does Java provide any construct to find out the size of an object?

A: No there is not sizeof operator in Java. So there is not direct way to determine the size of an object directly in Java.

Q: Give a simplest way to find out the time a method takes for execution without using any profiling tool?

A: Read the system time just before the method is invoked and immediately after method returns. Take the time difference, which will give you the time taken by a method for execution.To put it in code...

```
long start = System.currentTimeMillis ();
method ();
long end = System.currentTimeMillis ();
System.out.println ("Time taken for execution is " + (end - start));
```

Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amout of processing.

Q: What are wrapper classes?

A: Java provides specialized classes corresponding to each of the primitive data types. These are called wrapper classes. They are e.g. Integer, Character, Double etc.

Q: Why do we need wrapper classes?

A: It is sometimes easier to deal with primitives as objects. Moreover most of the collection classes store objects and not primitive data types. And also the wrapper classes provide many utility methods also. Because of these resons we need wrapper classes. And since we create instances of these classes we can store them in any of the collection classes and pass them around as a collection. Also we can pass them around as method parameters where a method expects an object.

Q: What are checked exceptions?

A: Checked exception are those which the Java compiler forces you to catch. e.g. IOException are checked Exceptions.

Q: What are runtime exceptions?

A: Runtime exceptions are those exceptions that are thrown at runtime because of either wrong input data or because of wrong business logic etc. These are not checked by the compiler at compile time.

Q: What is the difference between error and an exception?

A: An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you can not repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.).

Q: How to create custom exceptions?

A: Your class should extend class Exception, or some more specific type thereof.

Q: If I want an object of my class to be thrown as an exception object, what should I do?

A: The class should extend from Exception class. Or you can extend your class from some more precise exception type also.

***Q: If my class already extends from some other class what should I do if I want an instance of my class to be thrown as an exception object?**

A: One can't do anything in this scenario. Because Java does not allow multiple inheritance and does not provide any exception interface as well.

Q: How does an exception permeate through the code?

A: An unhandled exception moves up the method stack in search of a matching When an exception is thrown from a code which is wrapped in a try block followed by one or more catch blocks, a search is made for matching catch block. If a matching type is found then that block will be invoked. If a matching type is not found then the exception moves up the method stack and reaches the caller method. Same procedure is repeated if the caller method is included in a try catch block. This process continues until a catch block handling the appropriate type of exception is found. If it does not find such a block then finally the program terminates.

Q: What are the different ways to handle exceptions?

A: There are two ways to handle exceptions,

1. By wrapping the desired code in a try block followed by a catch block to catch the exceptions. And
2. List the desired exceptions in the throws clause of the method and let the caller of the method hadle those exceptions.

Q: What is the basic difference between the 2 approaches to exception handling. 1> try catch block and

2> specifying the candidate exceptions in the throws clause? When should you use which approach?

A: In the first approach as a programmer of the method, you urself are dealing with the exception. This is fine if you are in a best position to decide should be done in case of an exception. Whereas if it is not the responsibility of the method to deal with it's own exceptions, then do not use this approach. In this case use the second approach. In the second approach we are forcing the caller of the method to catch the exceptions, that the method is likely to throw. This is often the approach library creators use. They list the exception in the throws clause and we must catch them. You will find the same approach throughout the java libraries we use.

Q: Is it necessary that each try block must be followed by a catch block?

A: It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

Q: If I write return at the end of the try block, will the finally block still execute?

A: Yes even if you write return as the last statement in the try block and no exception occurs, the finally block will execute. The finally block will execute and then the control return.

Q: If I write System.exit (0); at the end of the try block, will the finally block still execute?

A: No in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

Q: How are Observer and Observable used?

A: Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.

Q: What is synchronization and why is it important?

A: With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

Q: How does Java handle integer overflows and underflows?

A: It uses those low order bytes of the result that can fit into the size of the type allowed by the operation.

Q: Does garbage collection guarantee that a program will not run out of memory?

A: Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection
.

Q: What is the difference between preemptive scheduling and time slicing?

A: 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.

Q: When a thread is created and started, what is its initial state?

A: A thread is in the ready state after it has been created and started.

Q: What is the purpose of finalization?

A: The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.

Q: What is the Locale class?

A: The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region.

Q: What is the difference between a while statement and a do statement?

A: A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.

Q: What is the difference between static and non-static variables?

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

Q: How are this() and super() used with constructors?

A: This() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.

Q: What are synchronized methods and synchronized statements?

A: Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

Q: What is daemon thread and which method is used to create the daemon thread?

A: Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation for the java runtime system. setDaemon method is used to create a daemon thread.

Q: Can applets communicate with each other?

A: At this point in time applets may communicate with other applets running in the same virtual machine. If the applets are of the same class, they can communicate via shared static variables. If the applets are of different classes, then each will need a reference to the same class with static variables. In any case the basic idea is to pass the information back and forth through a static variable. An applet can also get references to all other applets on the same page using the getApplets() method of java.applet.AppletContext. Once you get the reference to an applet, you can communicate with it by using its public members.

It is conceivable to have applets in different virtual machines that talk to a server somewhere on the Internet and store any data that needs to be serialized there. Then, when another applet needs this data, it could connect to this same server. Implementing this is non-trivial.

Q: What are the steps in the JDBC connection?

A: While making a JDBC connection we go through the following steps :

Step 1 : Register the database driver by using :
Class.forName(\" driver classs for that specific database\");
Step 2 : Now create a database connection using :
Connection con = DriverManager.getConnection(url,username,password);
Step 3: Now Create a query using :
Statement stmt = Connection.Statement(\"select * from TABLE NAME\");
Step 4 : Exceute the query :
stmt.exceuteUpdate();

Q: How does a try statement determine which catch clause should be used to handle an exception?

A: When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exceptionis executed. The remaining catch clauses are ignored.

Q: Can an unreachable object become reachable again?

A: An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects.

Q: What method must be implemented by all threads?

A: All tasks must implement the run() method, whether they are a subclass of Thread or implement the Runnable interface.

Q: What are synchronized methods and synchronized statements?

A: Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

Q: What is Externalizable?

A: Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, writeExternal(ObjectOuput out) and readExternal(ObjectInput in)

Q: What modifiers are allowed for methods in an Interface?

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

Q: What are some alternatives to inheritance?

A: Delegation is an alternative to inheritance. Delegation means that you include an instance of another class as an instance variable, and forward messages to the instance. It is often safer than inheritance because it forces you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense. On the other hand, it makes you write more code, and it is harder to re-use (because it is not a subclass).

Q: What does it mean that a method or field is "static"?

A: Static variables and methods are instantiated only once per class. In other words they are class variables, not instance variables. If you change the value of a static variable in a particular object, the value of that variable changes for all instances of that class.

Static methods can be referenced with the name of the class rather than the name of a particular object of the class (though that works too). That's how library methods like System.out.println() work out is a static field in the java.lang.System class.

Q: What is the difference between preemptive scheduling and time slicing?

A: 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.

Q: What is the catch or declare rule for method declarations?

A: If a checked exception may be thrown within the body of a method, the method must either catch the exception or declare it in its throws clause.

Q: Is Empty .java file a valid source file?

A: Yes, an empty .java file is a perfectly valid source file.

Q: Can a .java file contain more than one java classes?

A: Yes, a .java file contain more than one java classes, provided at the most one of them is a public class.

Q: Is String a primitive data type in Java?

A: No String is not a primitive data type in Java, even though it is one of the most extensively used object. Strings in Java are instances of String class defined in java.lang package.

Q: Is main a keyword in Java?

A: No, main is not a keyword in Java.

Q: Is next a keyword in Java?

A: No, next is not a keyword.

Q: Is delete a keyword in Java?

A: No, delete is not a keyword in Java. Java does not make use of explicit destructors the way C++ does.

Q: Is exit a keyword in Java?

A: No. To exit a program explicitly you use exit method in System object.

Q: What happens if you dont initialize an instance variable of any of the primitive types in Java?

A: Java by default initializes it to the default value for that primitive type. Thus an int will be initialized to 0, a boolean will be initialized to false.

Q: What will be the initial value of an object reference which is defined as an instance variable?

A: The object references are all initialized to null in Java. However in order to do anything useful with these references, you must set them to a valid object, else you will get NullPointerExceptions everywhere you try to use such default initialized references.

Q: What are the different scopes for Java variables?

A: The scope of a Java variable is determined by the context in which the variable is declared. Thus a java variable can have one of the three scopes at any given point in time.

1. Instance : - These are typical object level variables, they are initialized to default values at the time of creation of object, and remain accessible as long as the object accessible.

2. Local : - These are the variables that are defined within a method. They remain accessbile only during the course of method excecution. When the method finishes execution, these variables fall out of scope.

3. Static: - These are the class level variables. They are initialized when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance.

Q: What is the default value of the local variables?

A: The local variables are not initialized to any default value, neither primitives nor object references. If you try to use these variables without initializing them explicitly, the java compiler will not compile the code. It will complain abt the local varaible not being initilized..

Q: How many objects are created in the following piece of code?

MyClass c1, c2, c3;
c1 = new MyClass ();
c3 = new MyClass ();

A: Only 2 objects are created, c1 and c3. The reference c2 is only declared and not initialized.

Q: Can a public class MyClass be defined in a source file named YourClass.java?

A: No the source file name, if it contains a public class, must be the same as the public class name itself with a .java extension.

Q: Can main method be declared final?

A: Yes, the main method can be declared final, in addition to being public static.

Q: What will be the output of the following statement? System.out.println ("1" + 3);

A: It will print 13.

Q: What will be the default values of all the elements of an array defined as an instance variable?

A: If the array is an array of primitive types, then all the elements of the array will be initialized to the default value corresponding to that primitive type. e.g. All the elements of an array of int will be initialized to 0, while that of boolean type will be initialized to false. Whereas if the array is an array of references (of any type), all the elements will be initialized to null.

Q: What is reflection in Java?

A: Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection. Inspecting classes is often the first thing you do when using Reflection. From the classes you can obtain information about **Class Name**, **Class Modifies (public, private, synchronized etc.)** , **Package Info**, **Superclass** , **Implemented Interfaces** , **Constructors** , **Methods** , **Fields** , **Annotations** plus lot more information regarding java classes.

Reflection in Java is very powerful feature and allows you to **access private method and fields** which is not possible by any other means in Java and because of this feature of reflection many code coverage tool, static analysis tool and Java IDE like Eclipse and Netbeans has been so helpful.

Q: Does Java Support Operator Overloading?

A: No, Java doesn't support user-defined operator overloading. The only aspect of Java which comes close to "custom" operator overloading is the handling of + for strings, which either results in compile-time concatenation of constants or execution-time concatenation using StringBuilder/StringBuffer. You can't define your own operators which act in the same way though.

1) Simplicity and Cleanliness
Simple and clear design is one of the goals of java designer they just don't want to replicate the language but wanted to have a clear, truly object oriented language. Adding Operator overloading will definitely make design more complex than without it and it also slows the JVM because it needs to do extra work to identify the actual meaning of operators and reduce the opportunity to optimize the language by guarantee behavior of operators in Java.

2) Avoid programming errors
Java doesn't allow user defined operator overloading because if you allow programmer to do operator overloading they will come up with multiple meanings for same operator which will make the learning curve of any developer hard and things more confusing and messing. Its been observed that there is increase in programming errors when language supports operator overloading which in turn increase the development and delivery time and since Java and JVM has taken most of developers responsibility in memory management by proving garbage collector it doesn't really make sense to left this feature to pollute the code and a loop hole for programming errors.

3) JVM complexity
Form JVM perspective supporting operator overloading is more difficult and if the same thing can be achieved by using method overloading in more intuitive and clean way it does make sense to not support operator overloading in java. a complex JVM will result in slower JVM than a relatively simpler JVM and reduce the opportunity of optimization by taking out guaranteed behavior of operators in java.

4) Easy development of tools
This you can think of an additional benefit of not supporting operator overloading in Java. Omission of operator overloading has kept the language easier to handle and process which in turn makes to developed the tools that process the language e.g. IDE or re-factoring tool. Re-factoring tools are far better than C++

In conclusion many things which can be achieved via operator overloading can be achieved using method overloading using more intuitive and easy way and that might be the reason java designer thought that supporting operator overloading will not be a big benefit for language, but in fact only java designer can answer real motivation of why Java doesn't support operator overloading, like some other questions as Why Java doesn't support multiple inheritance or Why String is immutable in Java.

Q: What is Java Collections Framework?

A: A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.

- **Implementations i.e. Classes:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.

- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

Collection framework defines several interfaces.

The collection Interface: Enables you to work with groups of objects; it is at the top of collections hierarchy

The List Interface: Entends collection & an instance of List store an ordered collection of elements

The Set: This extends Collection to handle sets, which must contain unique elements

The SortedSet: extends Set to handle sorted sets.

The Map: maps unique keys to the values.

The Map Entry:Describes an element (key/value pair) in a map. It’s an inner class of Map

The SortedMap: This extends Map so that the keys are maintained in ascending order.
The Enumeration: is legacy interface and defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator.

Standard Collection Classes

AbstractCollection: Implements most of the Collection interface

AbstractList: Extends AbstractCollection and implements most of the List interface

AbstractSequentialList: Extends AbstractList for use by a collection that uses sequential rather than random access of its elements

LinkedList: Implements a linked list by extending AbstractSequentialList

ArrayList: Implements a dynamic array by extending AbstractList

AbstractSet: Extends AbstractCollection and implements most of the Set interface.

HashSet: Extends AbstractSet for use with a hash table.

LinkedHashSet: Extends HashSet to allow insertion- order iterations.

TreeSet: Implements set stored in a tree. Extends AbstractSet

AbstractMap: Implements most of the Map interface.

HashMap: Extends AbstractMap to use a hash table

TreeMap: Extends AbstractMap to use a tree

WeakHashMap: Extends AbstractMap to use a hash table with weak keys.

LinkedHashMap: Extends HashMap to allow insertion-order iterations.

IdentityHashMap: Extends AbstractMap and uses reference equality when comparing documents

"**What is deadlock?**"

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

How do you detect deadlock in Java ?

Though this could have many answers, my version is first I would look the code if I see nested synchronized block or calling one synchronized method from other or trying to get lock on different object then there is good chance of deadlock if developer is not very careful.

other way is to find it when you actually get locked while running the application , try to take thread dump , in Linux you can do this by command "kill -3", this will print status of all the thread in application log file and you can see which thread is locked on which object.

other way is to use jconsole , jconsole will show you exactly which threads are get locked and on which object.

Write code which will result in deadlock ?

here is one of my version

```
public void method1(){
synchronized(String.class){
System.out.println("Aquired lock on String.class object");

synchronized (Integer.class) {
System.out.println("Aquired lock on Integer.class object");
}   }   }

public void method2(){
synchronized(Integer.class){
System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {
System.out.println("Aquired lock on String.class object");
}    }   }
```

If method1() and method2() both will be called by two or many threads , there is a good chance of deadlock because if thead 1 aquires lock on Sting object while executing method1() and thread 2 acquires lock on Integer object while executing method2() both will be waiting for each other to release lock on Integer and String to proceed further which will never happen.

How to fix deadlock ? or How to avoid deadlock in Java ?

If you have looked above code carefully you may have figured out that real reason for deadlock is not multiple threads but the way they access lock , if you provide an ordered access then problem will be resolved , here is the fixed version.

```
public void method1(){
synchronized(Integer.class){
System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {
System.out.println("Aquired lock on String.class object");
} } }

public void method2(){
synchronized(Integer.class){
System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {
System.out.println("Aquired lock on String.class object");
}  }  }
```

Now there would not be any deadlock because both method is accessing lock on Integer and String object in same order. so if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock , same way thread A will not be blocked even if thread B holds String lock because now thread B will not expect thread A to release Integer lock to proceed further.

1) You have thread T1, T2 and T3, how will you ensure that thread T2 run after T1 and thread T3 run after T2?

This thread interview questions is mostly asked in first round or phone screening round of interview and purpose of this multi-threading question is to check whether candidate is familiar with concept of *"join"* method or not. Answer of this multi-threading questions is simple it can be achieved by using **join** method of Thread class.

2) What is the advantage of new Lock interface over synchronized block in Java? You need to implement a high performance cache which allows multiple reader but single writer to keep the integrity how will you implement it?

The major advantage of lock interfaces on multi-threaded and concurrent programming is they provide two separate lock for reading and writing which enables you to write high performance data structure likeConcurrentHashMap and conditional blocking. This java threads interview question is getting increasingly popular and more and more follow-up questions come based upon answer of interviewee. I would strongly suggest reading **Locks** before appearing for any *java multi-threading interview* because now days Its heavily used to build cache for electronic trading system on client and exchange connectivity space.

3) What are differences between wait and sleep method in java?

Another frequently asked thread interview question in Java mostly appear in phone interview. Only major difference is wait release the lock or monitor while sleep doesn't release any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution. See my post wait vs sleep in Java for more differences

4) Write code to implement blocking queue in Java?

This is relatively tough java multi-threading interview question which servers many purpose, it checks whether candidate can actually write Java code using thread or not, it sees how good candidate is on understanding concurrent scenarios and you can ask lot of follow-up question based upon his code. If he uses wait() and notify() method to implement blocking queue, Once interviewee successfully writes it you can ask him to write it again using new java 5 concurrent classes etc.

5) Write code to solve the Produce consumer problem in Java?

Similar to above questions on thread but more classic in nature, sometime interviewer ask follow up questions How do you solve producer consumer problem in Java, well it can be solved in multiple way, I have shared one way to solve producer consumer problem using BlockingQueue in Java , so be prepare for surprises. Some time they even ask to implement solution of dining philosopher problem as well.

6) Write a program which will result in deadlock? How will you fix deadlock in Java?

This is my favorite java thread interview question because even though deadlock is quite common while writing multi-threaded concurrent program many candidates not able to write deadlock free code and they simply struggle. Just ask them you have n resources and n thread and to complete an operation you require all resources. Here n can be replace with 2 for simplest case and higher number to make question more intimidating. see How to avoid deadlock in java for more information on deadlock in Java.

7) What is atomic operation? What are atomic operations in Java?

Simple java thread interview questions, another follow-up is do you need to synchronized an atomic operation? :) You can read more about java synchronization here.

8) What is volatile keyword in Java? How to use it? How is it different from synchronized method in Java?

Thread questions based on volatile keyword in Javahas become more popular after changes made on it on Java 5 and Java memory model. It’s good to prepare well about how volatile variables ensures visibility, ordering and consistency in concurrent environment.

9) What is race condition? How will you find and solve race condition?

Another multi-threading question in Java which appear mostly on senior level interviews. Most interviewer grill on recent race condition you have faced and how did you solve it and some time they will write sample code and ask you detect race condition. See my post on Race condition in Java for more information. In my opinion this is one of the best java thread interview question and can really test the candidate's experience on solving race condition or writing code which is free of data race or any other race condition. Best book to get mastery of this topic is "Concurrency practices in Java'".

10) How will you take thread dump in Java? How will you analyze Thread dump?

In UNIX you can use **kill -3** and then thread dump will print on log on windows you can use **"CTRL+Break".** Rather simple and focus thread interview question but can get tricky if he ask how you analyze it. Thread dump can be useful to analyze deadlock situations as well.

11) Why we call start() method which in turns calls run() method, why not we directly call run() method ?

Another classic java multi-threading interview question This was my original doubt when I started programming in thread. Now days mostly asked in phone interview or first round of interview at mid and junior level java interviews. Answer to this question is that, when you call start() method it creates new Thread and execute code declared in run() while directly calling run() method doesn't create any new thread and execute code on same calling thread. Read my post Difference between start and run method in Thread for more details.

12) How will you awake a blocked thread in java?

This is tricky question on threading, blocking can result on many ways, if thread is blocked on IO then I don't think there is a way to interrupt the thread, let me know if there is any, on the other hand if thread is blocked due to result of calling wait(), sleep() or join() method you can interrupt the thread and it will awake by throwing InterruptedException. See my post How to deal with blocking methods in Java for more information on handling blocked thread.

13) What is difference between CyclicBarriar and CountdownLatch in Java ?

New java thread interview questions mostly to check familiarity with JDK 5 concurrent packages. One difference is that you can reuse CyclicBarrier once barrier is broken but you can't reuse ContdownLatch.

14) What is immutable object? How does it help on writing concurrent application?

Another classic interview questions on multi-threading, not directly related to thread but indirectly helps a lot. This java interview question can become more tricky if ask you to write an immutable class or ask you Why String is immutable in Java as follow-up.

15) What are some common problems you have faced in multi-threading environment? How did you resolve it?

Memory-interference, race conditions, deadlock, live lock and starvation are example of some problems comes in multi-threading and concurrent programming. There is no end of problem if you get it wrong and they will be hard to detect and debug. This is mostly experienced based interview question on java thread instead of fact based.

1. What is design patterns? Have you used any design pattern in your code ?

Design patterns are tried and tested way to solve particular design issues by various programmers in the world. Design patterns are extension of code reuse.

2. Can you name few design patterns used in standard JDK library?

Decorator design pattern which is used in various Java IO classes, Singleton pattern which is used in Runtime , Calendar and various other classes, Factory pattern which is used along with various Immutable classes likes Boolean e.g. Boolean.valueOf and Observer pattern which is used in Swing and many event listener frameworks.

3. What is Singleton design pattern in Java? write code for thread-safe singleton in Java

Singleton pattern focus on sharing of expensive object in whole system. Only one instance of a particular class is maintained in whole application which is shared by all modules. Java.lang.Runtime is a classical example of Singleton design pattern.

4. What is main benefit of using factory pattern? Where do you use it?

Factory pattern's main benefit is increased level of encapsulation while creating objects. If you use Factory to create object you can later replace original implementation of Products or classes with more advanced and high performance implementation without any change on client layer. See my post on Factory pattern for more detailed explanation and benefits.

5. What is observer design pattern in Java?

Observer design pattern is based on communicating changes in state of object to observers so that they can take their action. Simple example is a weather system where change in weather must be reflected in Views to show to public. Here weather object is Subject while different views are Observers.

6. Give an example of decorator design pattern in Java? Does it operate on object or class level?

Decorator pattern enhances capability of individual object. Java IO uses decorator pattern extensively and classical example is Buffered classes like BufferedReader and BufferedWriter which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance.

7. What is Singleton class? Have you used Singleton before?

Singleton is a class which has only one instance in whole application and provides a getInstance() method to access the singleton instance. There are many classes in JDK which is implemented using Singleton pattern like java.lang.Runtime which provides getRuntime() method to get access of it and used to get free memory and total memory in Java.

8) Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?
Here they will check whether candidate has enough experience on usage of singleton or not. Does he is familiar of advantage/disadvantage or alternatives available for singleton in Java or not.

Answer: Any class which you want to be available to whole application and whole only one instance is viable is candidate of becoming Singleton. One example of this is Runtime class , since on whole java application only one runtime environment can be possible making Runtime Singleton is right decision. Another example is a utility classes like Popup in GUI application, if you want to show popup with message you can have one PopUp class on whole GUI application and anytime just get its instance, and call show() with message.

9) Can you write code for getInstance() method of a Singleton class in Java?

Most of the java programmer fail here if they have mugged up the singleton code because you can ask lots of follow-up question based upon the code they have written. I have seen many programmer write Singleton getInstance() method with double checked locking but they are not really familiar with the caveat associated with double checking of singleton prior to Java 5.

Answer: Until asked don't write code using double checked locking as it is more complex and chances of errors are more but if you have deep knowledge of double checked locking, volatile variable and lazy loading than this is your chance to shine. I have shared code examples of writing singleton classes using enum, using static factory and with double checked locking in my recent post Why Enum Singletons are better in Java, please see there.

10) Is it better to make whole getInstance() method synchronized or just critical section is enough? Which one you will prefer?

This is really nice question and I mostly asked to just quickly check whether candidate is aware of performance trade off of unnecessary locking or not. Since locking only make sense when we need to create instance and rest of the time its just read only access so locking of critical section is always better option.

Answer: This is again related to double checked locking pattern, well synchronization is costly and when you apply this on whole method than call to getInstance() will be synchronized and contented. Since synchronization is only needed during initialization on singleton instance, to prevent creating another instance of Singleton, It's better to only synchronize critical section and not whole method. Singleton pattern is also closely related to factory design pattern where getInstance() serves as static factory method.

11) What is lazy and early loading of Singleton and how will you implement it?

This is another great Singleton interview question in terms of understanding of concept of loading and cost associated with class loading in Java. Many of which I have interviewed not really familiar with this but its good to know concept.

Answer: As there are many ways to implement Singleton like using **double checked locking** or Singleton class with staticfinal instance initialized during class loading. Former is called lazy loading because Singleton instance is created only when client calls getInstance() method while later is called early loading because Singleton instance is created when class is loaded into memory.

12) Example of Singleton in standard Java Development Kit?

This is open question to all, please share which classes are Singleton in JDK. Answer to this question is java.lang.Runtime
Answer: There are many classes in Java Development Kit which is written using singleton pattern, here are few of them:

- Java.lang.Runtime with getRuntime() method
- Java.awt.Toolkit with getDefaultToolkit()
- Java.awt.Desktop with getDesktop()

13) What is double checked locking in Singleton?

One of the most hyped question on Singleton pattern and really demands complete understanding to get it right because of Java Memory model caveat prior to Java 5. If a guy comes up with a solution of using volatile keyword with Singleton instance and explains it then it really shows it has in depth knowledge of Java memory model and he is constantly updating his Java knowledge.

Answer: Double checked locking is a technique to prevent creating another instance of Singleton when call to getInstance() method is made in multi-threading environment. In Double checked locking pattern as shown in below example, singleton instance is checked two times before initialization.

```
public Singleton getInstance(){
if(_INSTANCE == null){
synchronized(Singleton.class){
//double checked locking - because second check of Singleton instance with lock
if(_INSTANCE == null){
_INSTANCE = new Singleton();
} }  }
return _INSTANCE;
}
```

Double checked locking should only be used when you have requirement for lazy initialization otherwise use Enum to implement singleton or simple static final variable.

14) How do you prevent for creating another instance of Singleton using clone() method? This type of questions generally comes some time by asking how to break singleton or when Singleton is not Singleton in Java.

Answer: Preferred way is not to implement Clonnable interface as why should one wants to create clone() of Singleto and if you do just throw Exception from clone() method as "Can not create clone of Singleton class".

15) How do you prevent for creating another instance of Singleton using reflection?

Open to all. In my opinion throwing exception from constructor is an option.

Answer: This is similar to previous interview question. Since constructor of Singleton class is supposed to be private it prevents creating instance of Singleton from outside but Reflection can access private fields and methods, which opens a threat of another instance. This can be avoided by throwing Exception from constructor as "Singleton already initialized"

16) How do you prevent for creating another instance of Singleton during serialization? Another great question which requires knowledge of Serialization in Java and how to use it for persisting Singleton classes. This is open to you all but in my opinion use of readResolve() method can sort this out for you.

Answer: You can prevent this by using readResolve() method, since during serialization readObject() is used to create instance and it return new instance every time but by using readResolve you can replace it with original Singleton instance. I have shared code on how to do it in my post Enum as Singleton in Java. This is also one of the reason I have said that use Enum to create Singleton because serialization of enum is taken care by JVM and it provides guaranted of that.

17) When is Singleton not a Singleton in Java?

There is a very good article present in Sun's Java site which discusses various scenarios when a Singleton is not really remains Singleton and multiple instance of Singleton is possible. Here is the link of that article http://java.sun.com/developer/technicalArticles/Programming/singletons/

18) Why you should avoid the singleton anti-pattern at all and replace it with DI?

Answer: Singleton Dependency Injection: every class that needs access to a singleton gets the object through its constructors or with a DI-container.

Singleton Anti-Pattern: with more and more classes calling getInstance the code gets more and more tightly coupled, monolithic, not testable and hard to change and hard to reuse because of not configurable, hidden dependencies. Also, there would be no need for this clumsy double checked locking if you call getInstance less often (i.e. once).

19) How many ways you can write Singleton Class in Java?

Answer: I know atleast four ways to implement Singleton pattern in Java

1) Singleton by synchronizing getInstance() method 2) Singleton with public static final field initialized during class loading. 3) Singleton generated by static nested class, also referred as Singleton holder pattern. 4) From Java 5 on-wards using Enums

20) How to write thread-safe Singleton in Java?

Answer: Thread-Saffe Singleton usually refers to write thread safe code which creates one and only one instance of Singleton if called by multiple thread at same time. There are many ways to achieve this like by using double checked locking technique as shown above and by using Enum or Singleton initialized by classloader.

21) Difference between Singleton and Static Class in java?

The big difference between a singleton and a bunch of static methods is that singletons can implement interfaces (or derive from useful base classes, although that's less common IME), so you can pass around the singleton as if it were "just another" implementation.

A singleton allows access to a single created instance - that instance (or rather, a reference to that instance) can be passed as a parameter to other methods, and treated as a normal object.

A static class allows only static methods.

22) Advantage of Singleton over Static Class?

The Singleton pattern has several advantages over static classes. First, a singleton can extend classes and implement interfaces, while a static class cannot (it can extend classes, but it does not inherit their instance members). A singleton can be initialized lazily or asynchronously while a static class is generally initialized when it is first loaded, leading to potential class loader issues. However the most important advantage, though, is that singletons can be handled polymorphically without forcing their users to assume that there is only one instance.

23) When to choose Singleton over Static Class?

static classes should not do anything need state, it is useful for putting bunch of functions together i.e Math (or Utils in projects). So the class name just give us a clue where we can find the functions and there's nothing more.

Singleton is my favorite pattern and use it to manage something at a single point. It's more flexible than static classes and can maintain state. It can implement interfaces, inherit from other classes and allow inheritance.

My rule for choosing between static and singleton: If there are bunch of functions should be kept together, then static is the choice. Anything else which needs single access to some resources, could be implemented singleton.

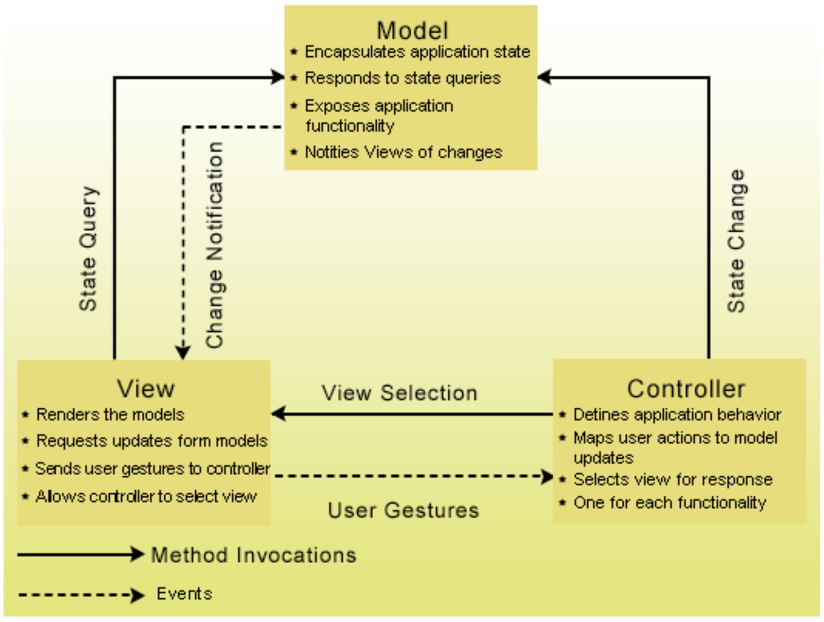
24) What is MVC design pattern? Give one example of MVC design pattern?

Model View controller is a classical design pattern used in applications who needs a clean separation between their business logic and view who represents data. MVC design pattern isolates the application logic from the user interface and permitted the individual development, testing and maintenance for each components. This design pattern is divided into three parts.

1. Model- This component manages the information and notify the observers when the information changes. It represents the data when on which the application operates. The model provides the persistent storage of data, which manipulated by the controller.

2. View- The view displays the data, and also takes input from user. It renders the model data into a form to display to the user. there can be several view associated with a single model. It is actually representation of model data.

3. Controller- The controller handles all request coming from the view or user interface. The data flow to whole application is controlled by controller. It forwarded the request to the appropriate handeler. Only the controller is responsible for accessing model and rendering it into various UIs.


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies Views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates form models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

25) What is FrontController design pattern in Java ? Give an example of front controller pattern?

When the user access the view directly without going thought any centralized mechanism each view is required to provide its; own system services, often resulting in duplication of code and view navigations is left to the views, this may result in commingled view content and view navigation.

A centralized point of contact for handling a request may be useful, for example, to control and log a user's progress through the site.

Introducing a controller as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

The controller provides a centralized entry point that controls and manages Web request handling. By centralizing decision points and controls, the controller also helps reduce the amount of Java code, called scriptlets, embedded in the JavaServer Pages (JSP) page.

Centralizing control in the controller and reducing business logic in the view promotes code reuse across requests. It is a preferable approach to the alternative-embedding code in multiple views-because that approach may lead to a more error-prone, reuse-by-copy- and-paste environment.

Typically, a controller coordinates with a dispatcher component. Dispatchers are responsible for view management and navigation. Thus, a dispatcher chooses the next view for the user and vectors control to the resource. Dispatchers may be encapsulated within the controller directly or can be extracted into a separate component.

The responsibilities of the components participating in this patterns are :

Controller : The controller is the initial contact point for handling all requests in the system. The controller may delegate to a helper to complete authentication and authorization of a user or to initiate contact retrieval.

Dispatcher : A dispatcher is responsible for view management and navigation, managing the choice of the next view to present to the user, and providing the mechanism for vectoring control to this resource. A dispatcher can be encapsulated within a controller or can be a separate component working in coordination. The dispatcher provides either a static dispatching to the view or a more sophisticated dynamic dispatching mechanism. The dispatcher uses the RequestDispatcher object (supported in the servlet specification) and encapsulates some additional processing.

Helper : A helper is responsible for helping a view or controller complete its processing. Thus, helpers have numerous responsibilities, including gathering data required by the view and storing this intermediate model, in which case the helper is sometimes referred to as a value bean. Additionally, helpers may adapt this data model for use by the view. Helpers can service requests for data from the view by simply providing access to the raw data or by formatting the data as Web content. A view may work with any number of helpers, which are typically implemented as JavaBeans components (JSP 1.0+) and custom tags (JSP 1.1+). Additionally, a helper may represent a Command object, a delegate, or an XSL Transformer, which is used in combination with a stylesheet to adapt and convert the model into the appropriate form.

View: A view represents and displays information to the client. The view retrieves information from a model. Helpers support views by encapsulating and adapting the underlying data model for use in the display.
Front Controller centralizes control. A controller provides a central place to handle system services and

business logic across multiple requests. A controller manages business logic processing and request handling. Centralized access to an application means that requests are easily tracked and logged. Keep in mind, though, that as control centralizes, it is possible to introduce a single point of failure. In practice, this rarely is a problem, though, since multiple controllers typically exist, either within a single server or in a cluster.

Front Controller improves manageability of security. A controller centralizes control, providing a choke point for illicit access attempts into the Web application. In addition, auditing a single entrance into the application requires fewer resources than distributing security checks across all pages.

Front Controller improves reusability. A controller promotes cleaner application partitioning and encourages reuse, as code that is common among components moves into a controller or is managed by a controller.

26. What is Chain of Responsibility design pattern?

Decoupling is one of the prominent mantras in software engineering. Chain of responsibility helps to decouple sender of a request and receiver of the request with some trade-offs. Chain of responsibility is a design pattern where a sender sends a request to a chain of objects, where the objects in the chain decide themselves who to honor the request. If an object in the chain decides not to serve the request, it forwards the request to the next object in the chain.

Responsibility is outsourced. In a chain of objects, the responsibility of deciding who to serve the request is left to the objects participating in the chains. It is similar to 'passing the question in a quiz scenario'. When the quiz master asks a question to a person, if he doesn't knows the answer, he passes the question to next person and so on. When one person answers the question, the passing flow stops. Sometimes, the passing might reach the last person and still nobody gives the answer.

Highlights of Chain of Responsibility

- Sender will not know which object in the chain will serve its request.
- Every node in chain will have the responsibility to decide, if they can serve the request.
- If node decides to forward the request, it should be capable of choosing the next node & forward it.
- There is a possibility where none of the node may serve the request.

27) What is Adapter design pattern ? Give examples of adapter design pattern in Java?

The adapter pattern is adapting between classes and objects. Like any adapter in the real world it is used to be an interface, a bridge between two objects. In real world we have adapters for power supplies, adapters for camera memory cards, and so on. Probably everyone have seen some adapters for memory cards. If you can't plug in the camera memory in your laptop you can use and adapter. You plug the camera memory in the adapter and the adapter in to laptop slot. That's it, it's really simple.

What about software development? It's the same. Can you imagine an situation when you have some class expecting some type of object and you have an object offering the same features, but exposing a different interface? Of course, you want to use both of them so you don't to implement again one of them, and you don't want to change existing classes, so why not create an adapter...

Intent

• Convert the interface of a class into another interface clients expect.
• Adapter lets classes work together, that could not otherwise because of incompatible interfaces.

A Simple Example of Model view Controller design pattern is given below

MainClaz.java

```
package roseindia.net;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class MainClaz {
	public static void main(String[] arguments) {
		// Creating Contact model
		StudentModel studentModel = new StudentModel();
		// Creating Contact view
		StudentView view = new StudentView(studentModel);
		studentModel.addContactView(view);
		createGui(view, "Enter Student deatail");
		// Contact View
		StudentDisplayView displayView = new StudentDisplayView();
		studentModel.addContactView(displayView);
		createGui(displayView, "Student Details");
	}
	private static void createGui(JPanel contents, String windowName) {
		JFrame frame = new JFrame(windowName);
		frame.getContentPane().add(contents);
		frame.addWindowListener(new WindowCloseManager());
		frame.pack();
		frame.setVisible(true);
	}
	private static class WindowCloseManager extends WindowAdapter {
		public void windowClosing(WindowEvent evt) {
			System.exit(0);
		}		}	}
```

StudentBean.java

```
package roseindia.net;
public class StudentBean {
	private String firstName;
	private String lastName;
	private String course;
	private String address;
	private StudentRefInterface view;

	public StudentBean(StudentRefInterface v) {

		firstName = "";
		lastName = "";
		course = "";
		address = "";
		view = v;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public String getCourse() {
		return course;
	}
	public void setCourse(String course) {
		this.course = course;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public StudentRefInterface getView() {
		return view;
	}
	public void setView(StudentRefInterface view) {
		this.view = view;
	}
}
```

StudentController.java

```
package roseindia.net;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StudentController implements ActionListener {
	private StudentModel model;
	private StudentView view;

	public StudentController(StudentModel m, StudentView v) {
		this.model = m;
		this.view = v;
	}

	public void actionPerformed(ActionEvent evt) {
		Object source = evt.getSource();
		if (source == view.getUpdateRef()) {
```

```
                updateModel();
            }
      }
      private void updateModel() {
            String firstName = null;
            String lastName = null;

            firstName = view.getFirstName();
            lastName = view.getLastName();
            model.updateModel(firstName, lastName, view.getCourse(),
view
                        .getAddress());
      }
}
```

StudentDisplayView.java

```
package roseindia.net;
import java.awt.BorderLayout;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class StudentDisplayView extends JPanel implements
StudentRefInterface {
      private JTextArea studentDetail;

      public StudentDisplayView() {
            createGui();
      }

      public void createGui() {
            setLayout(new BorderLayout());
            studentDetail = new JTextArea(10, 40);
            studentDetail.setEditable(false);
            JScrollPane scrollDisplay = new JScrollPane(studentDetail);
            this.add(scrollDisplay, BorderLayout.CENTER);
      }

      public void refresh(String newFirstName, String newLastName,
                String newCourse, String newAddress) {
            studentDetail.setText("\n\tStudent Details\n\t" + "\tName:
"
                        + newFirstName + "\n\t\tLast Name: " +
newLastName + "\n"
                        + "\t\tCourse: " + newCourse + "\n" +
"\t\tAddress: "
                        + newAddress);
      }
}
```

StudentModel.java

```java
package roseindia.net;

import java.util.ArrayList;
import java.util.Iterator;

public class StudentModel {
	private String firstName;
	private String lastName;
	private String course;
	private String address;
	private ArrayList studentViews = new ArrayList();

	public StudentModel() {
		this(null);
	}

	public StudentModel(StudentRefInterface view) {
		firstName = "";
		lastName = "";
		course = "";
		address = "";
		if (view != null) {
			studentViews.add(view);
		}
	}
	public void addContactView(StudentRefInterface view) {
		if (!studentViews.contains(view)) {
			studentViews.add(view);
		}
	}
	public void removeContactView(StudentRefInterface view) {
		studentViews.remove(view);
	}
	public String getFirstName() {
		return firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public String getTitle() {
		return course;
	}
	public String getOrganization() {
		return address;
	}
	public void setFirstName(String newFirstName) {
		firstName = newFirstName;
	}
	public void setLastName(String newLastName) {
		lastName = newLastName;
	}
	public void setCourse(String newCourse) {
		course = newCourse;
	}
	public void setAddress(String newAddress) {
		address = newAddress;
	}
	public void updateModel(String newFirstName, String newLastName,
			String newTitle, String newOrganization) {
		if (!isEmptyString(newFirstName)) {
			setFirstName(newFirstName);
		}
		if (!isEmptyString(newLastName)) {
			setLastName(newLastName);
		}
		if (!isEmptyString(newTitle)) {
			setCourse(newTitle);
		}
		if (!isEmptyString(newOrganization)) {
			setAddress(newOrganization);
		}
		updateView();
	}
	private boolean isEmptyString(String input) {
		return ((input == null) || input.equals(""));
	}
	private void updateView() {
		Iterator notifyViews = studentViews.iterator();
		while (notifyViews.hasNext()) {
			((StudentRefInterface)
notifyViews.next()).refresh(firstName,
					lastName, course, address);
		}
	}
}
```

StudentRefInterface.java

```
package roseindia.net;
public interface StudentRefInterface {
	public void refresh(String firstName, String lastName, String
course,
			String address);
}
```

StudentView.java

```
package roseindia.net;
```

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

```
public class StudentView extends JPanel implements StudentRefInterface
{
      private static final long serialVersionUID = 1L;
      private static final String SHOW_BUTTON = "Show";
      private static final String EXIT_BUTTON = "Exit";
      private static final String STUDENT_FIRST_NAME = "First Name  ";
      private static final String STUDENT_LAST_NAME = "Last Name  ";
      private static final String STUDENT_COURSE = "Course  ";
      private static final String STUDENT_ADDRESS = "Address  ";
      private static final int FNAME_COL_WIDTH = 30;
      private static final int LNAME_COL_WIDTH = 50;
      private static final int COURSE_COL_WIDTH = 35;
      private static final int ADDRESS_COL_WIDTH = 50;
      private StudentController controller;
      private JLabel firstNameLabel, lastNameLabel, courseLabel,
addressLabel;
      private JTextField firstName, lastName, course, address;
      private JButton display, exit;

      public StudentView(StudentModel model) {
            controller = new StudentController(model, this);
            createGui();
      }

      public StudentView(StudentModel model, StudentController
newController) {
            controller = newController;
            createGui();
      }

      public void createGui() {
            display = new JButton(SHOW_BUTTON);
            exit = new JButton(EXIT_BUTTON);

            firstNameLabel = new JLabel(STUDENT_FIRST_NAME);
            lastNameLabel = new JLabel(STUDENT_LAST_NAME);
            courseLabel = new JLabel(STUDENT_COURSE);
            addressLabel = new JLabel(STUDENT_ADDRESS);

            firstName = new JTextField(FNAME_COL_WIDTH);
            lastName = new JTextField(LNAME_COL_WIDTH);
            course = new JTextField(COURSE_COL_WIDTH);
            address = new JTextField(ADDRESS_COL_WIDTH);
            JPanel editPanel = new JPanel();
            editPanel.setLayout(new BoxLayout(editPanel,
BoxLayout.X_AXIS));

            JPanel labelPanel = new JPanel();
            labelPanel.setLayout(new GridLayout(0, 1));

            labelPanel.add(firstNameLabel);
            labelPanel.add(lastNameLabel);
            labelPanel.add(courseLabel);
            labelPanel.add(addressLabel);

            editPanel.add(labelPanel);

            JPanel fieldPanel = new JPanel();
            fieldPanel.setLayout(new GridLayout(0, 1));

            fieldPanel.add(firstName);
            fieldPanel.add(lastName);
            fieldPanel.add(course);
            fieldPanel.add(address);

            editPanel.add(fieldPanel);

            JPanel controlPanel = new JPanel();
            controlPanel.add(display);
            controlPanel.add(exit);
            display.addActionListener(controller);
            exit.addActionListener(new ExitHandler());

            setLayout(new BorderLayout());
            add(editPanel, BorderLayout.CENTER);
            add(controlPanel, BorderLayout.SOUTH);
      }
      public Object getUpdateRef() {
            return display;
      }
      public String getFirstName() {
            return firstName.getText();
      }
      public String getLastName() {
            return lastName.getText();
      }
      public String getCourse() {
            return course.getText();
      }
      public String getAddress() {
            return address.getText();
      }
      public void refresh(String newFirstName, String newLastName,
                  String newTitle, String newOrganization) {
            firstName.setText(newFirstName);
            lastName.setText(newLastName);
            course.setText(newTitle);
            address.setText(newOrganization);
      }
      private class ExitHandler implements ActionListener {
            public void actionPerformed(ActionEvent event) {
                  System.exit(0);
            }
      }
}
```