TABLE OF CONTENTS

JDBC Interview Questions and Answers 82

1. What do you mean by platform independence of Java?

Platform independence means that you can run the same Java Program in any Operating System. For example, you can write java program in Windows and run it in Mac OS.

2. What is JVM and is it platform independent?

Java Virtual Machine (JVM) is the heart of java programming language. JVM is responsible for converting byte code into machine readable code. JVM is not platform independent, thats why you have different JVM for different operating systems. We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It's called virtual because it provides an interface that doesn't depend on the underlying OS.

3. What is the difference between JDK and JVM?

Java Development Kit (JDK) is for development purpose and JVM is a part of it to execute the java programs.

JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence.

4. What is the difference between JVM and JRE?

Java Runtime Environment (JRE) is the implementation of JVM. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn't contain any development tools like java compiler, debugger etc. If you want to execute any java program, you should have JRE installed.

5. How are Java objects stored in memory?

In Java, all objects are dynamically allocated on Heap. In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on heap

6. Which class is the superclass of all classes?

java.lang.Object is the root class for all the java classes and we don't need to extend it.

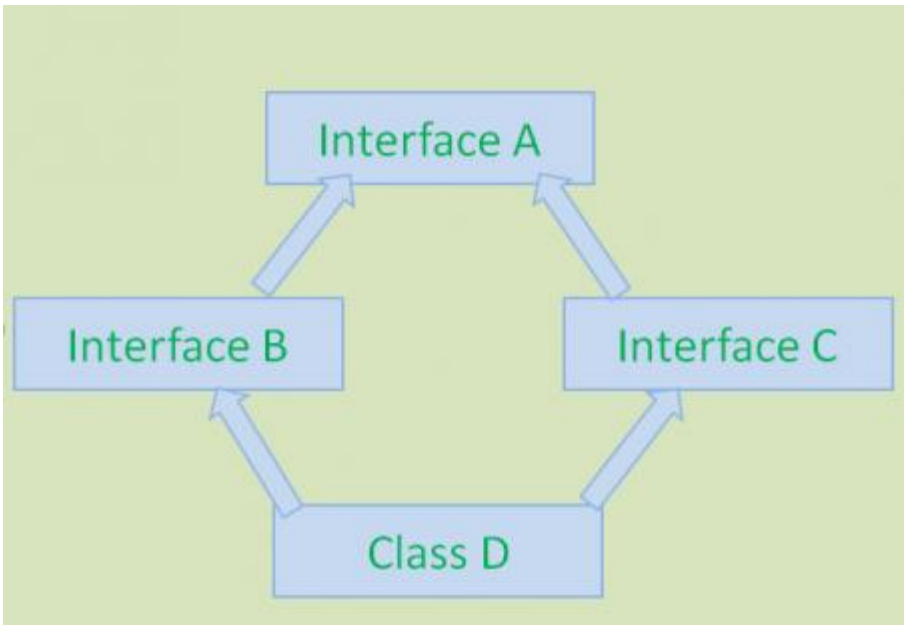
7. Why Java doesn't support multiple inheritance?

Java doesn't support multiple inheritance in classes because of "Diamond Problem". To know more about diamond problem with example, read Multiple Inheritance in Java.

However multiple inheritance is supported in interfaces. An interface can extend multiple interfaces because they just declare the methods and implementation will be present in the implementing class. So there is no issue of diamond problem with interfaces.

8. What is Diamond Problem in Inheritance? How Java 8 Solves this problem?

A Diamond Problem is a Multiple Inheritance problem. In Java, It occurs when a Class extends more than one Interface which have same method implementation (Default method).



This above diagram shows Diamond Problem. To avoid this problem, Java 7 and Earlier versions does not support methods implementation in interface and also doesn't support Multiple Inheritance. Java 8 has introduced new feature: Default methods to support Multiple Inheritance with some limitations.

Sample Java SE 8 Code to show this Diamond Problem:

```
public interface A{
  default void display() { //code goes here }
}
public interface B extends A{ }
public interface C extends A{ }
public class D implements B,C{ }
```

In the above code snippet, class D gives compiltime errors because Java Compiler will get bit confusion about which display() has to provide in class D. Class D inherits display() method from both interfaces B and C. To solve this problem, Java SE 8 has given the following remedy:

```
1    public interface A{
```

```
2       default void display() { //code goes here }
3     }
4     public interface B extends A{ }
5     public interface C extends A{ }
6     public class D implements B,C{
7       default void display() {
8         B.super.display();
9       }
10    }
```

This **B.super.display();** will solve this Diamond Problem.

When to use StringBuilder in Java

If you use String concatenation in a loop, something like this,

```
String s = "";
for (int i = 0; i < 100; i++) {
   s += ", " + i;
}
```

then you should use a StringBuilder (not StringBuffer) instead of a String, because it is much faster and consumes less memory.
If you have a single statement,

```
String s = "1, " + "2, " + "3, " + "4, " ...;
```

then you can use Strings, because the compiler will use StringBuilder automatically.

9. Why Java is not pure Object Oriented language?

Java is not said to be pure object oriented because it support primitive types such as int, byte, short, long etc. I believe it brings simplicity to the language while writing our code. Obviously java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.

As we know, for all the primitive types we have wrapper classes such as Integer, Long etc that provides some additional methods.

10. What is difference between path and classpath variables?

PATH is an environment variable used by operating system to locate the executables. That's why when we install Java or want any executable to be found by OS, we need to add the directory location in the PATH variable. If you work on Windows OS, read this post to learn how to setup PATH variable on Windows.

Classpath is specific to java and used by java executables to locate class files. We can provide the classpath location while running java application and it can be a directory, ZIP files, JAR files etc.

11. What is overloading and overriding in java?

When we have more than one method with same name in a single class but the arguments are different, then it is called as method overloading.

Overriding concept comes in picture with inheritance when we have two methods with same signature, one in parent class and another in child class. We can use @Override annotation in the child class overridden method to make sure if parent class method is changed, so as child class.

12. Method Overriding Rules

- The overriding method in subclass should have exactly same method signature as that of overridden method in superclass. Method signature includes - method name, type and number of parameters and the order of arguments. If they don't then it will result in overloading rather than overriding. Method signature does not comprise the final modifier of the parameter
- The return type of overriding method must be a covariant type (same class or sub-class), i.e. you can narrow down the return type.

- The overriding method cannot narrow the accessibility of the method, but it can widen it.
- You are free to throw any kind of Runtime Exception in the overriding method (But this is not applicable for checked exception)
- You shall only throw same or narrowed checked exception (same or sub-class) or none at all.
- methods marked private or final can not be overridden in subclass
- static methods can not be overridden since they are not instance methods of superclass.
- If a method is not inherited, then it can not be overridden. For example, private methods in superclass are not inherited.

13. Can we overload main method?

Yes, we can have multiple methods with name “main” in a single class. However if we run the class, java runtime environment will look for main method with syntax as `public static void main(String args[])`.

14. Can we have multiple public classes in a java source file?

We can’t have more than one public class in a single java source file. A single source file can have multiple classes that are not public.

15. What is Java Package and which package is imported by default?

Java package is the mechanism to organize the java classes by grouping them. The grouping logic can be based on functionality or modules based. A java class fully classified name contains package and class name. For example, `java.lang.Object` is the fully classified name of `Object` class that is part of`java.lang` package.

java.lang package is imported by default and we don't need to import any class from this package explicitly.

16. What are access modifiers?

Java provides access control through public, private and protected access modifier keywords. When none of these are used, it's called default access modifier.

Most Restrictive → Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Same rules apply for inner classes too, they are also treated as outer class properties

17. What is final keyword?

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can't extend it.

We can use final keyword with methods to make sure child classes can't override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

Java interface variables are by default final and static.

18. What is static keyword?

static keyword can be used with class level variables to make it global i.e all the objects will share the same variable.

static keyword can be used with methods also. A static method can access only static variables of class and invoke only static methods of the class.

19. What is finally and finalize in java?

finally block is used with try-catch to put the code that you want to get executed always, even if any exception is thrown by the try-catch block. finally block is mostly used to release resources created in the try block.

finalize() is a special method in Object class that we can override in our classes. This method get's called by garbage collector when the object is getting garbage collected. This method is usually overridden to release system resources when object is garbage collected.

20. What is memory leak?

A memory leak is where an unreferenced object that will never be used again still hangs around in memory and doesn't get garbage collected.

21. Can we declare a class as static?

We can't declare a top-level class as static however an inner class can be declared as static. If inner class is declared as static, it's called static nested class. Static nested class is same as any other top-level class and is nested for only packaging convenience.

22. What is static import?

If we have to use any static variable or method from other class, usually we import the class and then use the method/variable with class name.

```
import java.lang.Math;
//inside class
double test = Math.PI * 5;
```

We can do the same thing by importing the static method or variable only and then use it in the class as if it belongs to it.

```
import static java.lang.Math.PI;
//no need to refer class now
double test = PI * 5;
```

Use of static import can cause confusion, so it's better to avoid it. Overuse of static import can make your program unreadable and unmaintainable.

23. What is try-with-resources in java?

One of the Java 7 features is try-with-resources statement for automatic resource management. Before Java 7, there was no auto resource management and we should explicitly close the resource. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks when we forgot to close the resource.

From Java 7, we can create resources inside try block and use it. Java takes care of closing it as soon as try-catch block gets finished.

```
try (MyResource mr = new MyResource()) {
       System.out.println("MyResource created in try-with-resources");
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
```

24. What is multi-catch block in java?

Java 7 one of the improvement was multi-catch block where we can catch multiple exceptions in a single catch block. This makes are code shorter and cleaner when every catch block has similar code.

If a catch block handles multiple exception, you can separate them using a pipe (|) and in this case exception parameter (ex) is final, so you can't change it.

```
catch(IOException | SQLException | Exception ex){
     logger.error(ex);
     throw new MyException(ex.getMessage());
}
```

25. What is static block?

Java static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader. It is used to initialize static variables of the class. Mostly it's used to create static resources when class is loaded.

Unlike C++, Java supports a special block, called static block (also called static clause) which can be used for static initializations of a class. This code inside static block is executed only once: the first time you make an object of that class or the first time you access a static member of that class (even if you never make an object of that class).

26. Execution Order of Static/Intializer/contructor

```
public class JavaPocs {
      {
            System.out.println("Intializer block called ");
      }
```

```
    static {
            System.out.println("static block called ");
  }
    JavaPocs(){
             System.out.println("Constructor block called ");
    }

    public static void main(String[] args) {
          JavaPocs javaPocs = new JavaPocs();
          /*
           * output :
           *
           * static block called
           * Intializer block called
           * Constructor block called */
    }
}
```

27. Why static methods cannot access non static variables or methods?

A static method cannot access non static variables or methods because static methods can be accessed without instantiating the class, so if the class is not instantiated the variables are not intialized and thus cannot be accessed from a static method.

28. What is static class ?

A class cannot be declared static. But a class can be said a static class if all the variables and methods of the class are static and the constructor is private. Making the constructor private will prevent the class to be instantiated. So the only possibility to access is using Class name only

29. The Initializer Block in Java

Initializer block contains the code that is always executed whenever an instance is created. It is used to declare/initialize the common part of various constructors of a class.

The order of initialization constructors and initializer block doesn't matter, initializer block is always executed before constructor.

30. What is the difference between equals() and ==?

Ans) == operator is used to compare the references of the objects.
public bollean equals(Object o) is the method provided by the Object class. The default implementation uses == operator to compare two objects. But since the method can be overriden like for String class. equals() method can be used to compare the values of two objects.

```
String str1 = "MyName";
String str2 = new String("MyName");
String str3 = str2;

if (str1 == str2) {
          System.out.println("Objects are equal");
} else {
          System.out.println("Objects are not equal");
}
if (str1.equals(str2)) {
          System.out.println("Objects are equal");
} else {
          System.out.println("Objects are not equal");
}

Output:
Objects are not equal
Objects are equal
String str2 = "MyName";
String str3 = str2;
if (str2 == str3) {
          System.out.println("Objects are equal");
} else {
          System.out.println("Objects are not equal");
}
if (str3.equals(str2)) {
          System.out.println("Objects are equal");
} else {
          System.out.println("Objects are not equal");
}
Output:
Objects are equal
Objects are equal
```

31. What is use of synchronized keyword?

This keyword is used to prevent concurrency. Synchronized keyword can be applied to static/non-static methods or a block of code. Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread. Synchronized keyword provides a lock on the object and thus prevents race condition.

This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

E.g.

```
public void synchronized simpleMethod() {}
  public void synchronized staticMethod() {}
  public void myMethod() {
    synchronized (this) { // synchronized keyword on block  of code
    }
  }
```

32. What is volatile keyword?

In general each thread has its own copy of variable, such that one thread is not concerned with the value of same variable in the other thread. But sometime this may not be the case. Consider a scenario in which the count variable is holding the number of times a method is called for a given class irrespective of any thread calling, in this case irrespective of thread access the count has to be increased. In this case the count variable is declared as volatile. The copy of volatile variable is stored in the main memory, so every time a thread access the variable even for reading purpose the local copy is updated each time from the main memory. The volatile variable also have performance issues.

33. What is a transient variable?

If some of the properties of a class are not required to be serialized then the varaibles are marked as transient. When an object is deserialized the transient variables retains the default value depending on the type of variable declared and hence lost its original value.

34. What is a strictfp modifier?

Strictfp is used with variable only . It is used to restrict floating point calculations (fp) to ensure portability (platform Independent). When this modifier is specified, the JVM adheres to the Java specifications (IEEE-754 floating-point specification) and returns the consistent value independent of the platform. That is, if you want the answers from

your code (which uses floating point values) to be consistent in all platforms, then you need to specify the strictfp modifier.

35. System.exit() in Java

The java.lang.System.exit() method exits current program by terminating running Java virtual machine. This method takes a status code. A non-zero value of status code is generally used to indicate abnormal termination. This is similar exit in C/C++.
Following is the declaration for java.lang.System.exit() method:

```
public static void exit(int status)
```

exit(0) : Generally used to indicate successful termination.
exit(1) or exit(-1) or any other non-zero value – Generally indicates unsuccessful termination.
Finally block will not execute for system.exit()

36. Consider a scenario in which the admin want to sure that if some one has written System.exit() at some part of application then before system shutdown all the resources should be released. How is it possible?

Ans) This is possible using **Runtime.getRuntime().addShutdownHook(Thread hook)**.
Straight from Java Spec:
This method registers a new virtual-machine shutdown hook.
The Java virtual machine shuts down in response to two kinds of events:

- The program exits normally, when the last non-daemon thread exits or when the exit (equivalently, System.exit) method is invoked.
- The virtual machine is terminated in response to a user interrupt, such as typing ^C, or a system-wide event, such as user logoff or system shutdown.

A shutdown hook is simply an initialized but unstarted thread. When the virtual machine begins its shutdown sequence it will start all registered shutdown hooks in some unspecified order and let them run concurrently. When all the hooks have finished it will then run all uninvoked finalizers if finalization-on-exit has been enabled. Finally, the virtual machine will halt. Note that daemon threads will continue to run during the

shutdown sequence, as will non-daemon threads if shutdown was initiated by invoking the exit method.
Once the shutdown sequence has begun it can be stopped only by invoking the halt method, which forcibly terminates the virtual machine.

37. What is an interface?

Interfaces are to define the contract for the subclasses to implement.

Interfaces are good for starting point to define Type and create top level hierarchy in our code. Since a java class can implements multiple interfaces, it's better to use interfaces as super class in most of the cases.

38. What is an abstract class?

Abstract classes are used in java to create a class with some default method implementation for subclasses. An abstract class can have abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class. Abstract classes can't be instantiated and mostly used to provide base for sub-classes to extend and implement the abstract methods and override or use the implemented methods in abstract class. Read important points about abstract classes at java abstract class.

39. What is the difference between abstract class and interface?

1. `abstract` keyword is used to create an abstract class and it can be used with methods also whereas `interface` keyword is used to create interface and it can't be used with methods.
2. Subclasses use `extends` keyword to extend an abstract class and they need to provide implementation of all the declared methods in the abstract

class unless the subclass is also an abstract class whereas subclasses use `implements` keyword to implement interfaces and should provide implementation for all the methods declared in the interface.

3. Abstract classes can have methods with implementation whereas interface provides absolute abstraction and can't have any method implementations.
4. Abstract classes can have constructors but interfaces can't have constructors.
5. Abstract class have all the features of a normal java class except that we can't instantiate it. We can use `abstract` keyword to make a class abstract but interfaces are a completely different type and can have only public static final constants and method declarations.
6. Abstract classes methods can have access modifiers as public, private, protected, static but interface methods are implicitly public and abstract, we can't use any other access modifiers with interface methods.
7. A subclass can extend only one abstract class but it can implement multiple interfaces.
8. Abstract classes can extend other class and implement interfaces but interface can only extend other interfaces.
9. We can run an abstract class if it has `main()` method but we can't run an interface because they can't have main method implementation.
10. Interfaces are used to define contract for the subclasses whereas abstract class also define contract but it can provide other methods implementations for subclasses to use.

40. Can an abstract class have a static method?

Yes an abstract class have a static method and it can be accessed by any other class(even not a concrete class).

41. Can an abstract class have a constructor?

Yes an abstract class have a default and parameterized constructors.

42. Can an interface implement or extend another interface?

Interfaces don’t implement another interface, they extend it. Since interfaces can’t have method implementations, there is no issue of diamond problem. That’s why we have multiple inheritance in interfaces i.e an interface can extend multiple interfaces.

43. Explain with example to describe when to use abstract class and interface?

Consider a scenario where all Cars will have 4 tyres and other features can be different. In this case any subclass of Car has to have 4 tyres. This is a case where abstract class will be used and a default implementaion for tyres will be provided.

```
public abstract class Car {
          private final static TOTAL_TYRES = 4;
          public abstract String getCarName();
          public final int getNoOfTyres() {
                    return TOTAL TYRES;
          }
}
```

Consider a scenario where Cars can have any number of tyres and other features can also be different. In this case interface will be created.

```
public interface Car {
          public abstract String getCarName();
          public abstract int getNoOfTyres();
}
```

44. Can this keyword be assigned null value?

Ans) No

45. What is Marker interface?

A marker interface is an empty interface without any method but used to force some functionality in implementing classes by Java. Some of the well known marker interfaces are Serializable and Cloneable.

46. What are Wrapper classes?

Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing

and unboxing allows easy conversion between primitive types and their corresponding wrapper classes.

47. What is Enum in Java?

Enum was introduced in Java 1.5 as a new type whose fields consists of fixed set of constants. For example, in Java we can create Direction as enum with fixed fields as EAST, WEST, NORTH, SOUTH.

enum is the keyword to create an enum type and similar to class. Enum constants are implicitly static and final. Read more in detail at java enum.

48. What is Java Annotations?

Java Annotations provide information about the code and they have no direct effect on the code they annotate. Annotations are introduced in Java 5. Annotation is metadata about the program embedded in the program itself. It can be parsed by the annotation parsing tool or by compiler. We can also specify annotation availability to either compile time only or till runtime also. Java Built-in annotations are @Override, @Deprecated and @SuppressWarnings

49. What is Java Reflection API? Why it's so important to have?

Java Reflection API provides ability to inspect and modify the runtime behavior of java application. We can inspect a java class, interface, enum and get their methods and field details. Reflection API is an advanced topic and we should avoid it in normal programming. Reflection API usage can break the design pattern such as Singleton pattern by invoking the private constructor i.e violating the rules of access modifiers.

Even though we don't use Reflection API in normal programming, it's very important to have. We can't have any frameworks such as Spring, Hibernate or servers such as Tomcat, JBoss without Reflection API. They invoke the

appropriate methods and instantiate classes through reflection API and use it a lot for other processing.

50. What is composition in java?

Composition is the design technique to implement has-a relationship in classes. We can use Object composition for code reuse.

Java composition is achieved by using instance variables that refers to other objects. Benefit of using composition is that we can control the visibility of other object to client classes and reuse only what we need

51. What is the benefit of Composition over Inheritance?

One of the best practices of java programming is to “favor composition over inheritance”. Some of the possible reasons are:

- Any change in the superclass might affect subclass even though we might not be using the superclass methods. For example, if we have a method test() in subclass and suddenly somebody introduces a method test() in superclass, we will get compilation errors in subclass. Composition will never face this issue because we are using only what methods we need.
- Inheritance exposes all the super class methods and variables to client and if we have no control in designing superclass, it can lead to security holes. Composition allows us to provide restricted access to the methods and hence more secure.
- We can get runtime binding in composition where inheritance binds the classes at compile time. So composition provides flexibility in invocation of methods.

You can read more about above benefits of composition over inheritance at java composition vs inheritance.

52. How to sort a collection of custom Objects in Java?

We need to implement Comparable interface to support sorting of custom objects in a collection. Comparable interface has compareTo(T obj) method which is used by sorting methods and by providing this method implementation, we can provide default way to sort custom objects collection.

However, if you want to sort based on different criteria, such as sorting an Employees collection based on salary or age, then we can create Comparator instances and pass it as sorting methodology. For more details read Java Comparable and Comparator.

53. What is inner class in java?

We can define a class inside a class and they are called nested classes. Any non-static nested class is known as inner class. Inner classes are associated with the object of the class and they can access all the variables and methods of the outer class. Since inner classes are associated with instance, we can't have any static variables in them.

We can have local inner class or anonymous inner class inside a class. For more details read java inner class.

54. What is anonymous inner class?

A local inner class without name is known as anonymous inner class. An anonymous class is defined and instantiated in a single statement. Anonymous inner class always extend a class or implement an interface.

Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class. Anonymous inner classes are accessible only at the point where it is defined.

55. What is Classloader in Java?

Java Classloader is the program that loads byte code program into memory when we want to access any class. We can create our own classloader by extending ClassLoader class and overriding loadClass(String name) method. Learn more at java classloader.

56. What are different types of classloaders?

There are three types of built-in Class Loaders in Java:

0. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
1. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA_HOME/lib/ext directory.
2. System Class Loader – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

57. What is ternary operator in java?

Java ternary operator is the only conditional operator that takes three operands. It's a one liner replacement for if-then-else statement and used a lot in java programming. We can use ternary operator if-else conditions or even switch conditions using nested ternary operators. An example can be found at java ternary operator.

58. What does super keyword do?

super keyword can be used to access super class method when you have overridden the method in the child class.

We can use super keyword to invoke super class constructor in child class constructor but in this case it should be the first statement in the constructor method.

```
package com.journaldev.access;
public class SuperClass {
	public SuperClass(){
	}
	public SuperClass(int i){}
	public void test(){
		System.out.println("super class test method");
	}
}
```

Use of super keyword can be seen in below child class implementation.

```
package com.journaldev.access;
public class ChildClass extends SuperClass {

	public ChildClass(String str){
		//access super class constructor with super keyword
		super();
		//access child class method
		test();
		//use super to access super class method
		super.test();
	}
	@Override
	public void test(){
		System.out.println("child class test method");
	}
}
```

59. What is break and continue statement?

We can use break statement to terminate for, while, or do-while loop. We can use break statement in switch statement to exit the switch case. You can see the example of break statement at java break. We can use break with label to terminate the nested loops.

The continue statement skips the current iteration of a for, while or do-while loop. We can use continue statement with label to skip the current iteration of outermost loop.

60. What is this keyword?

this keyword provides reference to the current object and it's mostly used to make sure that object variables are used, not the local variables having same name.

```
//constructor
public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

We can also use this keyword to invoke other constructors from a constructor.

```
public Rectangle() {
    this(0, 0, 0, 0);
}
public Rectangle(int width, int height) {
    this(0, 0, width, height);
}
public Rectangle(int x, int y, int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
```

61. What is default constructor?

No argument constructor of a class is known as default constructor. When we don't define any constructor for the class, java compiler automatically creates the

default no-args constructor for the class. If there are other constructors defined, then compiler won't create default constructor for us.

62. Can we have try without catch block?

Yes, we can have try-finally statement and hence avoiding catch block.

63. What is Garbage Collection?

Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, process of deallocating memory is handled automatically by the garbage collector.

We can run the garbage collector with code Runtime.getRuntime().gc() or use utility methodSystem.gc(). For a detailed analysis of Heap Memory and Garbage Collection, please read Java Garbage Collection.

64. What is Serialization and Deserialization?

We can convert a Java object to an Stream that is called Serialization. Once an object is converted to Stream, it can be saved to file or send over the network or used in socket connections.

The object should implement Serializable interface and we can use java.io.ObjectOutputStream to write object to file or to any OutputStream object. Read more at Java Serialization.

The process of converting stream data created through serialization to Object is called deserialization. Read more at Java Deserialization.

Difference between java.util.Date and java.sql.Date

in Java from site. It is well explained.
Here are few differences on java.sql.Date and java.util.Date in Java in point format, if you any other difference between them which is worth noting then please post in comments :

1. As per Javadoc java.sql.Date is a thin wrapper around millisecond value which is used by JDBC to identify an SQL DATE type.
2. java.sql.Date just represent DATE without time information while java.util.Date represent both Date and Time information. This is the major differences why java.util.Date can not directly map to java.sql.Date.
3. In order to suppress time information and to confirm with definition of ANSI SQL DATE type, the millisecond values used in java.sql.Date instance must be "normalized by setting the hours, minutes, seconds and milliseconds to zero in the timezone with with DATE instance is associated. In other words all time related information is removed from java.sql.Date class.

65. What is the use of System class?

Java System Class is one of the core classes. One of the easiest way to log information for debugging is System.out.print() method.

System class is final so that we can't subclass and override it's behavior through inheritance. System class doesn't provide any public constructors, so we can't instantiate this class and that's why all of it's methods are static.

Some of the utility methods of System class are for array copy, get current time, reading environment variables. Read more at Java System Class.

66. What is instanceof keyword?

We can use instanceof keyword to check if an object belongs to a class or not. We should avoid it's usage as much as possible. Sample usage is:

```
public static void main(String args[]){
   Object str = new String("abc");

   if(str instanceof String){
            System.out.println("String value:"+str);
   }

   if(str instanceof Integer){
```

```
            System.out.println("Integer value:"+str);
    }
}
```

Since str is of type String at runtime, first if statement evaluates to true and second one to false.

67. What is difference between instanceof and isInstance(Object obj)?

Ans) Differences are as follows:

1) instanceof is a reserved word of Java, but isInstance(Object obj) is a method of java.lang.Class.

2) instanceof is used of identify whether the object is type of a particular class or its subclass but isInstance(obj) is used to identify object of a particular class.

68. Can we use String with switch case?

One of the Java 7 feature was improvement of switch case of allow Strings. So if you are using Java 7 or higher version, you can use String in switch-case statements. Read more at Java switch-case String example.

69. Java is Pass by Value or Pass by Reference?

This is a very confusing question, we know that object variables contain reference to the Objects in heap space. When we invoke any method, a copy of these variables is passed and gets stored in the stack memory of the method. We can test any language whether it's pass by reference or pass by value through a simple generic swap method, to learn more read Java is Pass by Value and Not Pass by Reference.

70. What is difference between Heap and Stack Memory?

Major difference between Heap and Stack memory are as follows:

0. Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
1. Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.
2. Memory management in stack is done in LIFO manner whereas it's more complex in Heap memory because it's used globally.

For a detailed explanation with a sample program, read Java Heap vs Stack Memory.

71. How to change the heap size of a JVM?

Ans) The old generation's default heap size can be overridden by using the -Xms and -Xmx switches to specify the initial and maximum sizes respectively:
java -Xms <initial size> -Xmx <maximum size> program
For example:
java -Xms64m -Xmx128m program

72. How does Java allocate stack and heap memory?

Ans) Each time an object is created in Java it goes into the part 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. This is why care should be taken in your code to avoid any concurrent access issues in the heap space. The stack is threadsafe (each thread will have its own

stack) but the heap is not threadsafe unless guarded with synchronisation through your code.

73. String vs StringBuilder vs StringBuffer in Java

- Objects of String are immutable, and objects of StringBuffer and StringBuilder are mutable.
- StringBuffer and StringBuilder are similar, but StringBuilder is faster and preferred over StringBuffer for single threaded program. If thread safety is needed, then StringBuffer is used.

74. Java Scanner and nextChar()

- Scanner class in Java supports nextInt(), nextLong(), nextDouble() etc. But there is no nextChar() (See this for examples)
- To read a char, we use **next().charAt(0)**. next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

75. Using underscore in Numeric Literals

A new feature was introduced by JDK 7 which allows to write numeric literals using the underscore character. Numeric literals are broken to enhance the readability.

This feature enables us to separate groups of digits in numeric literals, which improves readability of code. For instance, if our code contains numbers with many digits, we can use an underscore character to separate digits in groups of three, similar to how we would use a punctuation mark like a comma, or a space, as a separator.

The following example shows different ways we can use underscore in numeric literals:

```
// Java program to demonstrate that we can use underscore
// in numeric literals
class Test
{
     public static void main (String[] args)
               throws java.lang.Exception
     {
         int inum = 1_00_00_000;
         System.out.println("inum:" + inum);

         long lnum = 1 00 00 000;
         System.out.println("lnum:" + lnum);

         float fnum = 2.10_001F;
         System.out.println("fnum:" + fnum);
```

```
        double dnum = 2.10_12_001;
        System.out.println("dnum:" + dnum);
    }
}
```

Output:

```
inum: 10000000
lnum: 10000000
fnum: 2.10001
dnum: 2.1012001
```

76. BigInteger Class in Java

BigInteger class is used for mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

77. What is the maximum size of the array in Java?

In Java, arrays internally use integers (int not Integer) for index, the max size is limited by the max size of integers. So theoretically it is 2^31-1 = 2147483647, which is Integer.MAX_VALUE.

But in recent HotSpot JVM it has been observed that the max size of array can be Integer.MAX_VALUE - 5.

78. What will happen if you put System.exit(0) on try or catch block? Will finally block execute?

By Calling System.exit(0) in try or catch block, we can skip the finally block.

System.exit(int) method can throw a SecurityException. If Sysytem.exit(0) exits the JVM without throwing that exception then finally block will not execute. But, if System.exit(0) does throw security exception then finally block will be executed.

79. Label- Java

java supports label. The only place where a label is useful in Java is right before nested loop statements. We can specify label name with break to break out a specific outer loop. Similarly, label name can be specified with continue.
See following program for example.

```
// file name: Main.java
public class Main {
  public static void main(String[] args) {
    outer: //label for outer loop
    for (int i = 0; i < 10; i++) {
      for (int j = 0; j < 10; j++) {
        if (j == 1)
          break outer;
        System.out.println(" value of j = " + j);
      }
    } //end of outer loop
  } // end of main()
} //end of class Main
```

80. Pair Class in Java

In C++, we have std::pair in the utility library which is of immense use if we want to keep a pair of values together. We were looking for an equivalent class for pair in Java but Pair class did not come into existence till Java 7. JavaFX 2.2 has
the javafx.util.Pair class which can be used to store a pair. We need to store the values into Pair using the parameterized constructor provided by the javafx.util.Pair class.

Note : Note that the <Key, Value> pair used in HashMap/TreeMap. Here, <Key, Value> simply refers to a pair of values that are stored together.

Methods provided by the javafx.util.Pair class

- Pair (K key, V value) : Creates a new pair

- boolean equals() : It is used to compare two pair objects. It does a deep comparison, i.e., it compares on the basic of the values (<Key, Value>) which are stored in the pair objects.

Example:

```
Pair p1 = new Pair(3,4);
Pair p2 = new Pair(3,4);
Pair p3 = new Pair(4,4);
System.out.println(p1.equals(p2) + " " + p2.equals(p3));
```

Output:
true false

81. What is throw keyword?

Throw keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application.The exception thrown should be subclass of Throwable.

82. What is use of throws keyword?

Throws clause is used to throw the exception from a method to the calling method which could decide to handle exception or throw to its calling method in a class.

83. Difference between Scanner and BufferReader Class in Java

java.util.Scanner class is a simple text scanner which can parse primitive types and strings. It internally uses regular expressions to read different types.

Java.io.BufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of sequence of characters

Following are differences between above two.

- -BufferedReader is synchronous while Scanner is not. BufferedReader should be used if we are working with multiple threads.
- -BufferedReader has significantly larger buffer memory than Scanner.
- -The Scanner has a little buffer (1KB char buffer) as opposed to the BufferedReader (8KB byte buffer), but it's more than enough.

- -BufferedReader is a bit faster as compared to scanner because scanner does parsing of input data and BufferedReader simply reads sequence of characters

84. Explain re-entrant, recursive and idempotent methods/functions?

Ans) 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. Allrecursive 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. For example clustered EJBs, which are written with idempotent methods, can automatically recover from a server failure as long as it can reach another server.

85. What are the different types of references in java?

Ans) Java has a more expressive system of reference than most other garbage-collected programming languages, which allows for special behavior for garbage collection. A normal reference in Java is known as a strong reference. The java.lang.ref package defines three other types of references—soft, weak and phantom references. Each type of reference is designed for a specific use.

A **SoftReference** can be used to implement a cache. An object that is not reachable by a strong reference (that is, not strongly reachable) but is referenced by a soft reference is called softly reachable. A softly reachable object may be garbage collected at the discretion of the garbage collector. This generally means that softly reachable objects will only be garbage collected when free memory is low, but again, it is at the discretion of the garbage collector. Semantically, a soft reference means "keep this object unless the memory is needed."

A **WeakReference** is used to implement weak maps. An object that is not strongly or softly reachable, but is referenced by a weak reference is called weakly reachable. A weakly reachable object will be garbage collected during the next collection cycle. This behavior is used in the class java.util.WeakHashMap. A weak map allows the

programmer to put key/value pairs in the map and not worry about the objects taking up memory when the key is no longer reachable anywhere else. Another possible application of weak references is the string intern pool. Semantically, a weak reference means "get rid of this object when nothing else references it."

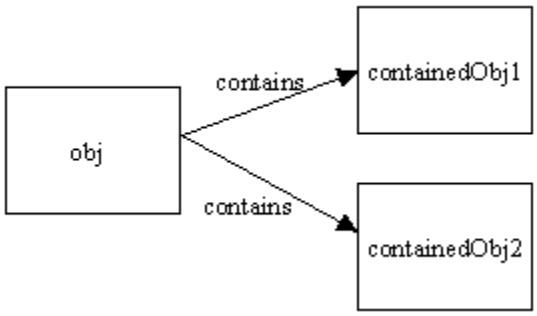
A **PhantomReference** is used to reference objects that have been marked for garbage collection and have been finalized, but have not yet been reclaimed. An object that is not strongly, softly or weakly reachable, but is referenced by a phantom reference is calledphantom reachable. This allows for more flexible cleanup than is possible with the finalization mechanism alone. Semantically, a phantom reference means "this object is no longer needed and has been finalized in preparation for being collected."

86. What are different type of cloning in Java?

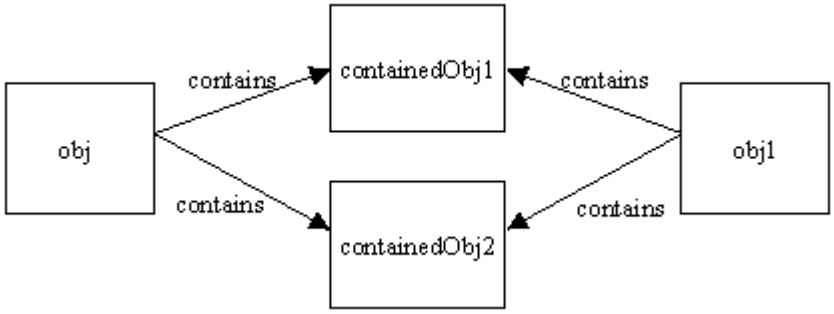
- Ans) Java supports two type of cloning: - **Deep and shallow cloning.** By default shallow clone is used in Java. Object class has a method clone() which does shallow cloning.

87. What is Shallow copy?

- Ans) Shallow clone is a copying the reference pointer to the object, which mean the new object is pointing to the same memory reference of the old object. The memory usage is lower.



- Figure 1: Original java object obj
- The shallow copy is done for obj and new object obj1 is created but contained objects of obj are not copied.



- Figure 2: Shallow copy object obj1
- It can be seen that no new objects are created for obj1 and it is referring to the same old contained objects. If either of the containedObj contain any other object no new reference is created.

88. What is deep copy and how it can be acheived?

Ans) In deep copy is the copy of object itself. A new memory is allocated for the object and contents are copied.

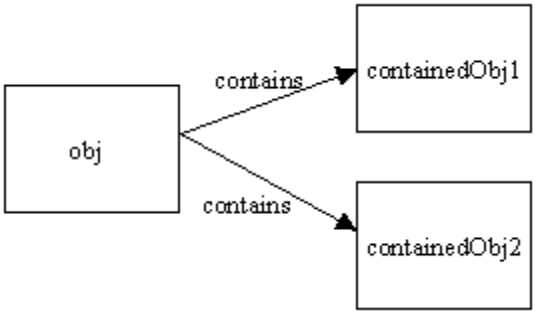


Figure 3 : Original Object obj

When a deep copy of the object is done new references are created.

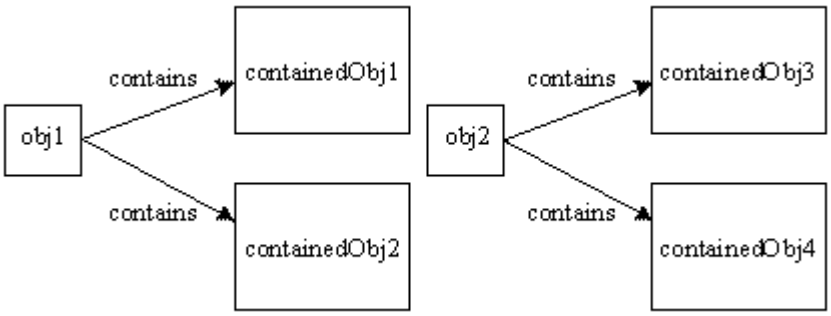


Figure 4: obj2 is deep copy of obj1

One solution is to simply implement your own custom method (e.g., deepCopy()) that returns a deep copy of an instance of one of your classes. This may be the best solution if you need a complex mixture of deep and shallow copies for different fields, but has a few significant drawbacks:

- You must be able to modify the class (i.e., have the source code) or implement a subclass. If you have a third-party class for which you do not have the source and which is marked final, you are out of luck.
- You must be able to access all of the fields of the classe's superclasses. If significant parts of the object's state are contained in private fields of a superclass, you will not be able to access them.
- You must have a way to make copies of instances of all of the other kinds of objects that the object references. This is particularly problematic if the exact classes of referenced objects cannot be known until runtime.
- Custom deep copy methods are tedious to implement, easy to get wrong, and difficult to maintain. The method must be revisited any time a change is made to the class or to any of its superclasses.

Shallow Copy	**Deep Copy**
Cloned Object and original object are not 100% disjoint.	Cloned Object and original object are 100% disjoint.
Any changes made to cloned object will be reflected in original object or vice versa.	Any changes made to cloned object will not be reflected in original object or vice versa.
Default version of clone method creates the shallow copy of an object.	To create the deep copy of an object, you have to override clone method.
Shallow copy is preferred if an object has only primitive fields.	Deep copy is preferred if an object has references to other objects as fields.
Shallow copy is fast and also less expensive.	Deep copy is slow and very expensive.

Other common solution to the deep copy problem is to use **Java Object Serialization**(JOS). The idea is simple: Write the object to an array using **ObjectOutputStream** and then use **ObjectInputStream** to reconsistute a copy of the object. The result will be a completely distinct object, with completely distinct referenced objects. JOS takes care of all of the details: superclass fields, following object graphs, and handling repeated references to the same object within the graph.

- It will only work when the object being copied, as well as all of the other objects references directly or indirectly by the object, are serializable. (In other words, they must implement java.io.Serializable.) Fortunately it is often sufficient to simply declare that a given class implements java.io.Serializable and let Java's default serialization mechanisms do their thing. Java Object Serialization is slow, and using it to make a deep copy requires both serializing and deserializing.

There are ways to speed it up (e.g., by pre-computing serial version ids and defining custom readObject() and writeObject() methods), but this will usually be the primary bottleneck. The byte array stream implementations included in the java.io package are designed to be general enough to perform reasonable well for data of different sizes and to be safe to use in a multi-threaded environment. These characteristics, however, slow down ByteArrayOutputStream and (to a lesser extent) ByteArrayInputStream .

89. What is difference between deep and shallow cloning?

Ans) The differences are as follows:

Consider a class:

```
public class MyData{
String id;
Map myData;
}
```

The shallow copying of this object will be pointing to the same memory reference as the original object. So a change in myData by either original or cloned object will be reflected in other also. But in deep copying there will memory allocated and values assigned to the property will be same. Any change in object will not be reflected in other.

Shallow copying is default cloning in Java which can be achieved using **Object.clone()**method of Object class. For deep copying override the clone method to create new object and copy its values.

90. What are disadvantages of deep cloning ?

Ans) Disadvantages of using Serialization to achieve deep cloning –

- Serialization is more expensive than using object.clone().
- Not all objects are serializable.
- Serialization is not simple to implement for deep cloned object..

91. What is Association?

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

What is Aggregation?

Aggregation is a specialize form of Association where all object have their own lifecycle but there is ownership and child object cannot belongs to another parent object. Let's take an example of Department and teacher. A single teacher cannot belongs to multiple departments, but if we delete the department teacher object will not destroy. We can think about "has-a" relationship.

What is Composition?

Composition is again specialize form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different house if we delete the house room will automatically delete.

String

What is String in Java? String is a data type?

String is a Class in java and defined in java.lang package. It's not a primitive data type like int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String in immutable and final in Java and JVM uses String Pool to store all the String objects.
String pool is also example of Flyweight **design pattern**

Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of "+" operator for concatenation.

What are different ways to create String Object?

We can create String object using `new` operator like any normal java class or we can use double quotes to create a String object. There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.

```
String str = new String("abc");
String str1 = "abc";
```

When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.

When we use new operator, JVM creates the String object but don't store it into the String Pool. We can use`intern()` method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.

Write a method to check if input String is Palindrome?

A String is said to be Palindrome if it's value is same when reversed. For example "aba" is a Palindrome String.
String class doesn't provide any method to reverse the String
but `StringBuffer` and `StringBuilder` class has reverse method that we can use to check if String is palindrome or not.

```
 private static boolean isPalindrome(String str) {
    if (str == null)
        return false;
    StringBuilder strBuilder = new StringBuilder(str);
    strBuilder.reverse();
    return strBuilder.toString().equals(str);
}
```

Sometimes interviewer asks not to use any other class to check this, in that case we can compare characters in the String from both ends to find out if it's palindrome or not.

```
private static boolean isPalindromeString(String str) {
    if (str == null)
        return false;
    int length = str.length();
    System.out.println(length / 2);
    for (int i = 0; i < length / 2; i++) {

        if (str.charAt(i) != str.charAt(length - i - 1))
            return false;
```

```
        }
        return true;
}
```

Write a method that will remove given character from the String?

We can use `replaceAll` method to replace all the occurance of a String with another String. The important point to note is that it accepts String as argument, so we will use `Character` class to create String and use it to replace all the characters with empty String.

```
    private static String removeChar(String str, char c) {
        if (str == null)
            return null;
        return str.replaceAll(Character.toString(c), "");
    }
```

How can we make String upper case or lower case?

We can use String class `toUpperCase` and `toLowerCase` methods to get the String in all upper case or lower case. These methods have a variant that accepts Locale argument and use that locale rules to convert String to upper or lower case.

What is String subSequence method?

Java 1.4 introduced CharSequence interface and String implements this interface, this is the only reason for the implementation of subSequence method in String class. Internally it invokes the String substring method.
Check this post for String subSequence example.

How to compare two Strings in java program?

Java String implements `Comparable` interface and it has two variants of `compareTo()` methods.

`compareTo(String anotherString)` method compares the String object with the String argument passed lexicographically. If String object precedes the argument passed, it returns negative integer and if String object follows the argument String passed, it returns positive integer. It returns zero when both the String have same value, in this case `equals(String str)` method will also return true.

compareToIgnoreCase(String str): This method is similar to the first one, except that it ignores the case. It uses String CASE_INSENSITIVE_ORDER Comparator for case insensitive comparison. If the value is zero then `equalsIgnoreCase(String str)` will also return true.
Check this post for String compareTo example.

How to convert String to char and vice versa?

This is a tricky question because String is a sequence of characters, so we can't convert it to a single character. We can use use `charAt` method to get the character at given index or we can use `toCharArray()` method to convert String to character array.
Check this post for sample program on converting String to character array to String.

How to convert String to byte array and vice versa?

We can use String `getBytes()` method to convert String to byte array and we can use String constructor `new String(byte[] arr)` to convert byte array to String.
Check this post for String to byte array example.

Can we use String in switch case?

This is a tricky question used to check your knowledge of current Java developments. Java 7 extended the capability of switch case to use Strings also, earlier java versions doesn't support this.

If you are implementing conditional flow for Strings, you can use if-else conditions and you can use switch case if you are using Java 7 or higher versions.

Check this post for Java Switch Case String example.

Write a program to print all permutations of String?

This is a tricky question and we need to use recursion to find all the permutations of a String, for example "AAB" permutations will be "AAB", "ABA" and "BAA".
We also need to use Set to make sure there are no duplicate values.
Check this post for complete program to find all permutations of String.

Write a function to find out longest palindrome in a given string?

A String can contain palindrome strings in it and to find longest palindrome in given String is a programming question.
Check this post for complete program to find longest palindrome in a String.

Difference between String, StringBuffer and StringBuilder?

String is immutable and final in java, so whenever we do String manipulation, it creates a new String. String manipulations are resource consuming, so java provides two utility classes for String manipulations - StringBuffer and StringBuilder.
StringBuffer and StringBuilder are mutable classes. StringBuffer operations are thread-safe and synchronized where StringBuilder operations are not thread-safe. So when multiple threads are working on same String, we should use StringBuffer but in single threaded environment we should use StringBuilder.
StringBuilder performance is fast than StringBuffer because of no overhead of synchronization.

Check this post for extensive details about String vs StringBuffer vs StringBuilder.
Read this post for benchmarking of StringBuffer vs StringBuilder.

Why String is immutable or final in Java

There are several benefits of String because it's immutable and final.

- String Pool is possible because String is immutable in java.
- It increases security because any hacker can't change its value and it's used for storing sensitive information such as database username, password etc.
- Since String is immutable, it's safe to use in multi-threading and we don't need any synchronization.
- Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.

Check this post to get more details why String is immutable in java.

How to Split String in java?

We can use `split(String regex)` to split the String into String array based on the provided regular expression.
Learn more at java String split.

Why Char array is preferred over String for storing password?

String is immutable in java and stored in String pool. Once it's created it stays in the pool until unless garbage collected, so even though we are done with password it's available in memory for longer duration and there is no way to avoid it. It's a security risk because anyone having access to memory dump can find the password as clear text.
If we use char array to store password, we can set it to blank once we are done with it. So we can control for how long it's available in memory that avoids the security threat with String.

How do you check if two Strings are equal in Java?

There are two ways to check if two Strings are equal or not - using "==" operator or using `equals` method. When we use "==" operator, it checks for value of String as well as reference but in our programming, most of the time we are checking equality of String for value only. So we should use equals method to check if two Strings are equal or not.
There is another function `equalsIgnoreCase` that we can use to ignore case.

```
        String s1 = "abc";

        String s2 = "abc";

        String s3= new String("abc");

        System.out.println("s1 == s2 ? "+(s1==s2)); //true

        System.out.println("s1 == s3 ? "+(s1==s3)); //false

        System.out.println("s1 equals s3 ? "+(s1.equals(s3)));
//true
```

What is String Pool?

As the name suggests, String Pool 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.
Check this post for more details about String Pool.

What does String intern() method do?

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.
This method always return a String that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

Does String is thread-safe in Java?

Strings are immutable, so we can't change it's value in program. Hence it's thread-safe and can be safely used in multi-threaded environment.
Check this post for Thread Safety in Java.

Why String is popular HashMap key in Java?

Since String is immutable, its hashcode is cached at the time of creation and it doesn't need to be calculated again. This makes it a great candidate for key in a Map and it's processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

String Programming Questions

1. What is the output of below program?

```
2. package com.journaldev.strings;
3.
4. public class StringTest {
5.
6.   public static void main(String[] args) {
7.      String s1 = new String("pankaj");
8.      String s2 = new String("PANKAJ");
9.      System.out.println(s1 = s2);
10.     }
11.  }
```

It's a simple yet tricky program, it will print "PANKAJ" because we are assigning s2 String to s1. Don't get confused with == comparison operator.

12. What is the output of below program?

```
13.  package com.journaldev.strings;
```

```
14.
15.  public class Test {
16.
17.      public void foo(String s) {
18.      System.out.println("String");
19.      }
20.
21.      public void foo(StringBuffer sb){
22.      System.out.println("StringBuffer");
23.      }
24.
25.      public static void main(String[] args) {
26.          new Test().foo(null);
27.     }
```

```
28.
```

The above program will not compile with error as "The method foo(String) is ambiguous for the type Test". For complete clarification read Understanding the method X is ambiguous for the type Y error.

29. What is the output of below code snippet?

```
30.  String s1 = new String("abc");
31.  String s2 = new String("abc");
32.  System.out.println(s1 == s2);
```

It will print **false** because we are using *new* operator to create String, so it will be created in the heap memory and both s1, s2 will have different reference. If we create them using double quotes, then they will be part of string pool and it will print true.

33. What will be output of below code snippet?

```
34.  String s1 = "abc";
35.  StringBuffer s2 = new StringBuffer(s1);
36.  System.out.println(s1.equals(s2));
```

It will print false because s2 is not of type String. If you will look at the equals method implementation in the String class, you will find a check using **instanceof** operator to check if the type of passed object is String? If not, then return false.

37. What will be output of below program?

```
38.  String s1 = "abc";
39.  String s2 = new String("abc");
40.  s2.intern();
41.  System.out.println(s1 ==s2);
```

It's a tricky question and output will be **false**. We know that intern() method will return the String object reference from the string pool, but since we didn't assigned it back to s2, there is no change in s2 and hence both s1 and s2 are having different reference. If we change the code in line 3 to `s2 = s2.intern();` then output will be true.

42. How many String objects got created in below code snippet?

```
43.  String s1 = new String("Hello");
44.  String s2 = new String("Hello");
```

Answer is 3.
First - line 1, "Hello" object in the string pool.
Second - line 1, new String with value "Hello" in the heap memory.
Third - line 2, new String with value "Hello" in the heap memory. Here "Hello" string from string pool is reused.

Exception Handling.

1. What is Exception in Java?

Exception is an error event that can happen during the execution of a program and disrupts it's normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then**JRE** tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as **catching the exception**. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

2. What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

1. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. **throw** keyword is used to throw exception to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it's caller method using `throws` keyword. We can provide multiple exceptions in the throws clause and it can be used with **main()** method also.

3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurrs or not.

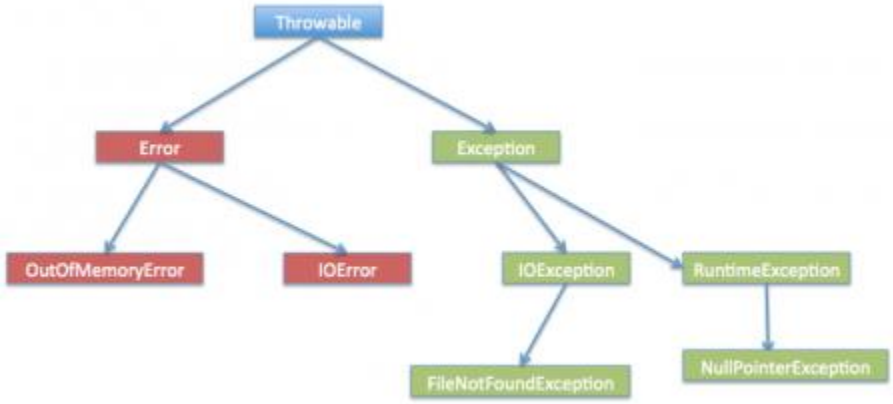
3. Explain Java Exception Hierarchy?

Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. `Throwable` is the parent class of Java Exceptions Hierarchy and it has two child objects – `Error` and `Exception`. Exceptions are further divided into checked exceptions and runtime exception.

Errors are exceptional scenarios that are out of scope of application and it's not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error.

Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. `Exception` is the parent class of all Checked Exceptions.

Runtime Exceptions are caused by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw `ArrayIndexOutOfBoundException` at runtime. `RuntimeException` is the parent class of all runtime exceptions.



4. What are important methods of Java Exception Class?

Exception and all of it's subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable.

1. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it's constructor.
2. **String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use `getMessage()` method to return the exception message.
3. **synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

5. Explain Java 7 ARM Feature and multi-catch block?

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the feature was multi-catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:

```
catch(IOException | SQLException | Exception ex){
     logger.error(ex);
     throw new MyException(ex.getMessage());
}
```

Read more about this at **Java 7 ARM**.

6. What is difference between Checked and Unchecked Exception in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else main() method should use throws keyword to let JRE know about these exception that might be thrown from the program. Unchecked Exceptions are not required to be handled in the program or to mention them in throws clause.
2. `Exception` is the super class of all checked exceptions whereas `RuntimeException` is the super class of all unchecked exceptions.
3. Checked exceptions are error scenarios that are not caused by program, for example FileNotFoundException in reading a file that is not present, whereas Unchecked exceptions are mostly caused by poor programming, for example NullPointerException when invoking a method on an object reference without making sure that it's not null.

7. What is difference between throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of program and handing over the exception object to runtime to handle it.

8. How to write custom exception in Java?

We can extend `Exception` class or any of it's subclasses to create our custom exception class. The custom exception class can have it's own variables and methods that we can use to pass error codes or other exception related information to the exception handler.

A simple example of custom exception is shown below.

```
package com.journaldev.exceptions;
import java.io.IOException;

public class MyException extends IOException {

  private static final long serialVersionUID = 4664456874499611218L;

  private String errorCode="Unknown_Exception";

  public MyException(String message, String errorCode){
          super(message);
          this.errorCode=errorCode;
  }

  public String getErrorCode(){
          return this.errorCode;
  }


}
```

9. What is OutOfMemoryError in Java?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and it's thrown by JVM when it ran out of heap memory. We can fix this error by providing more memory to run the java application through java options.

```
$>java MyProgram -Xms1024m -Xmx1024m -XX:PermSize=64M -XX:MaxPermSize=256m
```

10. What are different scenarios causing "Exception in thread main"?

Some of the common main thread exception scenarios are:

- **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.
- **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.
- **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn't have main method.
- **Exception in thread "main" java.lang.ArithmeticException**: Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

Read more about these at Java Exception in Thread main.

11. What is difference between final, finally and finalize in Java?

final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can't be reassigned, with class to avoid extending by classes and with methods to avoid overriding by

subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it's great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

12. What happens when exception is thrown by main method?

When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

13. Can we have an empty catch block?

We can have an empty catch block but it's the example of worst programming. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it wil be a nightmare to debug it. There should be at least a logging statement to log the exception details in console or log files.

14. Provide some Java Exception Handling Best Practices?

Some of the best practices related to Java Exception Handling are:

- Use Specific Exceptions for ease of debugging.
- Throw Exceptions Early (Fail-Fast) in the program.
- Catch Exceptions late in the program, let the caller handle the exception.
- Use Java 7 ARM feature to make sure resources are closed or use finally block to close them properly.
- Always log exception messages for debugging purposes.
- Use multi-catch block for cleaner close.

- Use custom exceptions to throw single type of exception from your application API.
- Follow naming convention, always end with Exception.
- Document the Exceptions Thrown by a method using @throws in javadoc.
- Exceptions are costly, so throw it only when it makes sense. Else you can catch them and provide null or empty response.

Read more about them in detail at Java Exception Handling Best Practices.

15. What is the problem with below programs and how do we fix it?

In this section, we will look into some programming questions related to java exceptions.

```
package com.journaldev.exceptions;
import java.io.FileNotFoundException;
import java.io.IOException;
public class TestException {

  public static void main(String[] args) {
          try {
                  testExceptions();
          } catch (FileNotFoundException | IOException e) {
                  e.printStackTrace();
          }
  }
  public static void testExceptions() throws IOException,
FileNotFoundException{

  }
}
```

Above program won't compile and you will get error message as "The exception FileNotFoundException is already caught by the alternative

IOException". This is because FileNotFoundException is subclass of IOException, there are two ways to solve this problem.

First way is to use single catch block for both the exceptions.

```
        try {
                testExceptions();
        }catch(FileNotFoundException e){
                e.printStackTrace();
        }catch (IOException  e) {
                e.printStackTrace();
        }
```

Another way is to remove the FileNotFoundException from multi-catch block.

```
        try {
                testExceptions();
        }catch (IOException  e) {
                e.printStackTrace();
        }
```

You can chose any of these approach based on your catch block code.

0. **What is the problem with below program?**

```
package com.journaldev.exceptions;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.xml.bind.JAXBException;
public class TestException1 {
  public static void main(String[] args) {
                try {
                        go();
                } catch (IOException e) {
                        e.printStackTrace();
```

```
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (JAXBException e) {
                        e.printStackTrace();
                }
  }
  public static void go() throws IOException, JAXBException,
FileNotFoundException{

  }
}
```

The program won't compile because FileNotFoundException is subclass of IOException, so the catch block of FileNotFoundException is unreachable and you will get error message as "Unreachable catch block for FileNotFoundException. It is already handled by the catch block for IOException".

You need to fix the catch block order to solve this issue.

```
                try {
                        go();
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (JAXBException e) {
                        e.printStackTrace();
                }
```

Notice that JAXBException is not related to IOException or FileNotFoundException and can be put anywhere in above catch block hierarchy.

1. **What is the problem with below program?**

```
2. package com.journaldev.exceptions;
3. import java.io.IOException;
4. import javax.xml.bind.JAXBException;
5.
6. public class TestException2 {
7.     public static void main(String[] args) {
8.             try {
9.                     foo();
10.                    } catch (IOException e) {
11.                     e.printStackTrace();
12.                    }catch(JAXBException e){
13.                     e.printStackTrace();
14.                    }catch(NullPointerException e){
15.                     e.printStackTrace();
16.                    }catch(Exception e){
17.                     e.printStackTrace();
18.                    }
19.            }
20.            public static void foo() throws IOException{
21.
22.            }
23.    }
```

The program won't compile because JAXBException is a checked exception and foo() method should throw this exception to catch in the calling method. You will get error message as "Unreachable catch block for JAXBException. This exception is never thrown from the try statement body".

To solve this issue, you will have to remove the catch block of JAXBException.

Notice that catching NullPointerException is valid because it's an unchecked exception.

24. **What is the problem with below program?**

```
25.     package com.journaldev.exceptions;
26.
27.     public class TestException3 {
28.
29.             public static void main(String[] args) {
30.                     try{
31.                     bar();
32.                     }catch(NullPointerException e){
33.                      e.printStackTrace();
34.                     }catch(Exception e){
35.                      e.printStackTrace();
36.                     }
37.                      foo();
38.             }
39.             public static void bar(){
40.     }
41.     public static void foo() throws NullPointerException{
42.             }
43.     }
```

This is a trick question, there is no problem with the code and it will compile successfully. We can always catch Exception or any unchecked exception even if it's not in the throws clause of the method.

Similarly if a method (foo) declares unchecked exception in throws clause, it is not mandatory to handle that in the program.

44. **What is the problem with below program?**

```
45.     package com.journaldev.exceptions;
46.     import java.io.IOException;
47.
48.     public class TestException4 {
49.             public void start() throws IOException{
50.             }
```

```
51.                     public void foo() throws NullPointerException{
52.             }
53.     }
54.     class TestException5 extends TestException4{
55.             public void start() throws Exception{
56.             }
57.             public void foo() throws RuntimeException{
58.
59.             }
60.     }
```

The above program won't compile because start() method signature is not same in subclass. To fix this issue, we can either change the method singnature in subclass to be exact same as superclass or we can remove throws clause from subclass method as shown below.

```
61.     @Override
62.             public void start(){
63.             }
```

64. **What is the problem with below program?**

```
65.     package com.journaldev.exceptions;
66.     import java.io.IOException;
67.     import javax.xml.bind.JAXBException;
68.     public class TestException6 {
69.             public static void main(String[] args) {
70.                     try {
71.                      foo();
72.                     } catch (IOException | JAXBException e) {
73.                      e = new Exception("");
74.                      e.printStackTrace();
75.                     }catch(Exception e){
76.                      e = new Exception("");
77.                      e.printStackTrace();
78.                     }
```

```
79.             }
80.             public static void foo() throws IOException,
   JAXBException{
81.             }
82.     }
```

The above program won't compile because exception object in multi-catch block is final and we can't change it's value. You will get compile time error as "The parameter e of a multi-catch block cannot be assigned".

We have to remove the assignment of "e" to new exception object to solve this error.

JAVA 8:

Why do we need change to Java again?

Oracle Corporation has introduced a lot of new concepts in Java SE 8 to introduce the following benefits:

- **To Utilize Current Multi-Core CPUs Efficiently**

  Recently, we can observe drastic changes in Hardware. Now-a-days, all systems are using Multi-Core CPUs(2,4,8,16-Core etc.) to deploy and run their Applications. We need new Programming Constructs in Java to utilize these Multi-Core Processors efficiently to develop Highly Concurrently and Highly Scalable applications.

- **To Utilize FP Features**

  Oracle Corporation has introduced a lot of FP(Functional Programming) concepts as part of Java SE 8 to utilize the advantages of FP.

Java SE 8 New Features?

- Lambda Expressions
- Functional Interfaces
- Stream API
- Date and Time API
- Interface Default Methods and Static Methods
- Spliterator
- Method and Constructor References
- Collections API Enhancements
- Concurrency Utils Enhancements
- Fork/Join Framework Enhancements
- Internal Iteration
- Parallel Array and Parallel Collection Operations
- Optional
- Type Annotations and Repeatable Annotations
- Method Parameter Reflection
- Base64 Encoding and Decoding
- IO and NIO2 Enhancements
- Nashorn JavaScript Engine
- javac Enhancements
- JVM Changes
- Java 8 Compact Profiles: compact1,compact2,compact3
- JDBC 4.2
- JAXP 1.6
- Java DB 10.10
- Networking
- Security Changes

Advantages of Java SE 8 New Features?

We can get the following benefits from Java SE 8 New Features:

- More Concise and Readable code

- More Reusable code
- More Testable and Maintainable Code
- Highly Concurrent and Highly Scalable Code
- Write Parallel Code
- Write Database Like Operations
- Better Performance Applications
- More Productive code

What is Lambda Expression?

Lambda Expression is an anonymous function which accepts a set of input parameters and returns results.

Lambda Expression is a block of code without any name, with or without parameters and with or without results. This block of code is executed on demand.

What are the three parts of a Lambda Expression? What is the type of Lambda Expression?

A Lambda Expression contains 3 parts:

- Parameter List

  A Lambda Expression can contain zero or one or more parameters. It is optional.

- Lambda Arrow Operator

  “->” is known as Lambda Arrow operator. It separates parameters list and body.

- Lambda Expression Body

The type of “Journal Dev” is java.lang.String. The type of “true” is Boolean. In the same way, what is the type of a Lambda Expression?

The Type of a Lambda Expression is a Functional Interface.

Example:- What is the type of the following Lambda Expression?

```
1    () -> System.out.println("Hello World");
```

This Lambda Expression does not have parameters and does return any results. So it's type is "java.lang.Runnable" Functional Interface.

What is a Functional Interface? What is SAM Interface?

A Functional Interface is an interface, which contains one and only one abstract method. Functional Interface is also know as SAM Interface because it contains only one abstract method.

SAM Interface stands for Single Abstract Method Interface. Java SE 8 API has defined many Functional Interfaces.

Is is possible to define our own Functional Interface? What is @FunctionalInterface? What are the rules to define a Functional Interface?

Yes, it is possible to define our own Functional Interfaces. We use Java SE 8's @FunctionalInterface annotation to mark an interface as Functional Interface.

We need to follow these rules to define a Functional Interface:

- Define an interface with one and only one abstract method.
- We cannot define more than one abstract method.
- Use @FunctionalInterface annotation in interface definition.
- We can define any number of other methods like Default methods, Static methods.
- If we override java.lang.Object class's method as an abstract method, which does not count as an abstract method.

Is @FunctionalInterface annotation mandatory to define a Functional Interface? What is the use of @FunctionalInterface annotation? Why do we need Functional Interfaces in Java?

It is not mandatory to define a Functional Interface with @FunctionalInterface annotation. If we don't want, We can omit this annotation. However, if we use it in

Functional Interface definition, Java Compiler forces to use one and only one abstract method inside that interface.

Why do we need Functional Interfaces? The type of a Java SE 8's Lambda Expression is a Functional Interface. Whereever we use Lambda Expressions that means we are using Functional Interfaces.

When do we go for Java 8 Stream API? Why do we need to use Java 8 Stream API in our projects?

When our Java project wants to perform the following operations, it's better to use Java 8 Stream API to get lot of benefits:

- When we want perform Database like Operations. For instance, we want perform groupby operation, orderby operation etc.
- When want to Perform operations Lazily.
- When we want to write Functional Style programming.
- When we want to perform Parallel Operations.
- When want to use Internal Iteration
- When we want to perform Pipelining operations.
- When we want to achieve better performance.

Explain Differences between Collection API and Stream API?

S.NO.	COLLECTION API	STREAM API
1.	It's available since Java 1.2	It is introduced in Java SE8
2.	It is used to store Data(A set of Objects).	It is used to compute data(Computation on a set of Objects).
3.	We can use both Spliterator and Iterator to iterate elements.	We can use both Spliterator and Iterator to iterate elements.
4.	It is used to store limited number of	It is used to store either Limited or Infinite Number of

	Elements.	Elements.
5.	Typically, it uses Internal Iteration concept to iterate Elements.	It uses External Iteration to iterate Elements.
6.	Collection Object is constructed Eagerly.	Stream Object is constructed Lazily.
7.	We add elements to Collection object only after it is computed completely.	We can add elements to Stream Object without any prior computation. That means Stream objects are computed on-demand.
8.	We can iterate and consume elements from a Collection Object at any number of times.	We can iterate and consume elements from a Stream Object only once.

What is Spliterator in Java SE 8?Differences between Iterator and Spliterator in Java SE 8?

Spliterator stands for Splitable Iterator. It is newly introduced by Oracle Corporation as part Java SE 8.
Like Iterator and ListIterator, It is also one of the Iterator interface.

S.NO.	SPLITERATOR	ITERATOR
1.	It is introduced in Java SE 8.	It is available since Java 1.2.
2.	Splitable Iterator	Non-Splitable Iterator
3.	It is used in Stream API.	It is used for Collection API.
4.	It uses Internal Iteration concept to iterate Streams.	It uses External Iteration concept to iterate Collections.
5.	We can use Spliterator to iterate Streams in Parallel and Sequential order.	We can use Spliterator to iterate Collections only in Sequential order.
6.	We can get Spliterator by calling spliterator() method on Stream Object.	We can get Iterator by calling iterator() method on Collection Object.
7.	Important Method: tryAdvance()	Important Methods: next(), hasNext()

What is Optional in Java 8? What is the use of Optional?Advantages of Java 8 Optional?

Optional:

Optional is a final Class introduced as part of Java SE 8. It is defined in java.util package.

It is used to represent optional values that is either exist or not exist. It can contain either one value or zero value. If it contains a value, we can get it. Otherwise, we get nothing.

It is a bounded collection that is it contains at most one element only. It is an alternative to “null” value.

Main Advantage of Optional is:

- It is used to avoid null checks.
- It is used to avoid “NullPointerException”.

What is Type Inference? Is Type Inference available in older versions like Java 7 and Before 7 or it is available only in Java SE 8?

Type Inference means determining the Type by compiler at compile-time.

It is not new feature in Java SE 8. It is available in Java 7 and before Java 7 too.

Before Java 7:-

Let us explore Java arrays. Define a String of Array with values as shown below:

```
1    String str[] = { "Java 7", "Java 8", "Java 9" };
```

Here we have assigned some String values at right side, but not defined it’s type. Java Compiler automatically infers it’s type and creates a String of Array.

Java 7:-

Oracle Corporation has introduced “Diamond Operator” new feature in Java SE 7 to avoid unnecessary Type definition in Generics.

```
1    Map<String,List<Customer>> customerInfoByCity = new HashMap<>();
```

Here we have not defined Type information at right side, simply defined Java SE 7’s Diamond Operator “”.

Java SE 8:-

Oracle Corporation has enhanced this Type Inference concept a lot in Java SE 8. We use this concept to define Lambda Expressions, Functions, Method References etc.

```
1    Integer add = (a,b) -> a + b;
```

Here Java Compiler observes the type definition available at left-side and determines the type of Lambda Expression parameters a and b is Integer.

What is Internal Iteration in Java SE 8?

Before Java 8, We don’t Internal Iteration concept. Java 8 has introduced a new feature known as “Internal Iteration”. Before Java 8, Java Language has only External Iteration to iterate elements of an Aggregated Object like Collections, Arrays etc.

Internal Iteration means “Iterating an Aggregated Object elements one by one internally by Java API”. Instead of Java Application do iteration externally, We ask Java API to do this job internally.

Differences between External Iteration and Internal Iteration?

S.NO.	EXTERNAL ITERATION	INTERNAL ITERATION
1.	Available before Java 8 too.	It is introduced in Java SE 8
2.	Iterating an Aggregated Object elements externally.	Iterating an Aggregated Object elements internally (background).

3.	Iterate elements by using for-each loop and Iterators like Enumeration, Iterator, ListIterator.	Iterate elements by using Java API like “forEach” method.
4.	Iterating elements in Sequential and In-Order only.	Not required to iterate elements in Sequential order.
5.	It follows OOP approach that is Imperative Style.	It follows Functional Programming approach that is Declarative Style.
6.	It does NOT separate responsibilities properly that is, it defines both “What is to be done” and “How it is to be done”.	It defines only “What is to be done”. No need to worry about “How it is to be done”. Java API takes care about “How to do”.
7.	Less Readable Code.	More Readable code.

What are the major drawbacks of External Iteration?

External Iteration has the following drawbacks:

- We need to write code in Imperative Style.
- There is no clear separation of Responsibilities. Tightly-Coupling between “What is to be done” and “How it is to be done” code.
- Less Readable Code.
- More Verbose and Boilerplate code.
- We have to iterate elements in Sequential order only.
- It does not support Concurrency and Parallelism properly.

What are the major advantages of Internal Iteration over External Iteration?

Compare to External Iteration, Internal Iteration has the following advantages:

- As it follows Functional Programming style, we can write Declarative Code.
- More Readable and concise code.
- Avoids writing Verbose and Boilerplate code
- No need to iterate elements in Sequential order.
- It supports Concurrency and Parallelism properly.
- We can write Parallel code to improve application performance.

- Clear separation of Responsibilities. Loosely-Coupling between “What is to be done” and “How it is to be done” code.
- We need to write code only about “What is to be done” and Java API takes care about “How it is to be done” code.

What is the major drawback of Internal Iteration over External Iteration?

Compare to External Iteration, Internal Iteration has one major drawback:

- In Internal Iteration, as Java API takes care about Iterating elements internally, we do NOT have control over Iteration.

What is the major advantage of External Iteration over Internal Iteration?

Compare to Internal Iteration, External Iteration has one major advantage:

- In External Iteration, as Java API does NOT take care about Iterating elements, we have much control over Iteration.

When do we need to use Internal Iteration? When do we need to use External Iteration?

We need to understand the situations to use either Internal Iteration or External Iteration.

- When we need more control over Iteration, we can use External Iteration.
- When we do NOT need more control over Iteration, we can use Internal Iteration.
- When we need to develop Highly Concurrency and Parallel applications and we , we should use Internal Iteration.

Differences between Intermediate Operations and Terminal Operations of Java 8’s Stream API?

S.NO.	STREAM INTERMEDIATE OPERATIONS	STREAM TERMINAL OPERATIONS
1.	Stream Intermediate operations are not evaluated until we chain it with Stream Terminal Operation.	Stream Terminal Operations are evaluated on it's own. No need other operations help.
2.	The output of Intermediate Operations is another Stream.	The output of Intermediate Operations is Not a Stream. Something else other than a Stream.
3.	Intermediate Operations are evaluated Lazily.	Terminal Operations are evaluated Eagerly.
4.	We can chain any number of Stream Intermediate Operations.	We can NOT chain Stream Terminal Operations.
5.	We can use any number of Stream Intermediate Operations per Statement.	We can use only one Stream Terminal Operation per Statement.

Is it possible to provide method implementations in Java Interfaces? If possible, how do we provide them?

In Java 7 or earlier, It is not possible to provide method implementations in Interfaces. Java 8 on-wards, it is possible.

In Java SE 8, We can provide method implementations in Interfaces by using the following two new concepts:

- Default Methods
- Static Methods

What is a Default Method? Why do we need Default methods in Java 8 Interfaces?

A Default Method is a method which is implemented in an interface with "default" keyword. It's new featured introduced in Java SE 8.

We need Default Methods because of the following reasons:

- It allow us to provide method's implementation in Interfaces.

- To add new Functionality to Interface without breaking the Classes which implement that Interface.
- To provide elegant Backwards Compatibility Feature.
- To ease of extend the existing Functionality.
- To ease of Maintain the existing Functionality.

What is a Static Method? Why do we need Static methods in Java 8 Interfaces?

A Static Method is an Utility method or Helper method, which is associated to a class (or interface). It is not associated to any object.

We need Static Methods because of the following reasons:

- We can keep Helper or Utility methods specific to an interface in the same interface rather than in a separate Utility class.
- We do not need separate Utility Classes like Collections, Arrays etc to keep Utility methods.
- Clear separation of Responsibilities. That is we do not need one Utility class to keep all Utility methods of Collection API like Collections etc.
- Easy to extend the API.
- Easy to Maintain the API.

Differences between Functional Programming and Object-Oriented Programming?

FUNCTIONAL PROGRAMMING	OOP
Does not exist State	Exists State
Uses Immutable data	Uses Mutable data
It follows Declarative Programming Model	It follows Imperative Programming Model
Stateless Programming Model	Stateful Programming Model

Main Fcous on: “What you are doing”	Main focus on “How you are doing”
Good for Parallel (Concurrency) Programming	Poor for Parallel (Concurrency) Programming
Good for BigData processing and analysis	NOT Good for BigData processing and analysis
Supports pure Encapsulation	It breaks Encapsulation concept
Functions with No-Side Effects	Methods with Side Effects
Functions are first-class citizens	Objects are first-class citizens
Primary Manipulation Unit is “Function”	Primary Manipulation Unit is Objects(Instances of Classes)
Flow Controls: Function calls, Function Calls with Recursion	Flow Controls: Loops, Conditional Statements
It uses “Recursion” concept to iterate Collection Data.	It uses “Loop” concept to iterate Collection Data. For example:-For-each loop in Java
Order of execution is less importance.	Order of execution is must and very important.
Supports both “Abstraction over Data” and “Abstraction over Behavior”.	Supports only “Abstraction over Data”.
We use FP when we have few Things with more operations.	We use OOP when we have few Operations with more Things. For example: Things are classes and Operations are Methods in Java.

NOTE:- For more information about FP, IP and OOP comparisons, Please go through my previous post at: “Compare FP, OOP(IP)”

Explain issues of Old Java Date API? What are the advantages of Java 8’s Date and Time API over Old Date API and Joda Time API?

Java’s OLD Java Date API means Date API available before Java SE 8 that is Date, Calendar, SimpleDateFormat etc.

Java's Old Date API has the following Issues or Drawbacks compare to Java 8's Date and Time API and Joda Time API.

- Most of the API is deprecated.
- Less Readability.
- java.util.Date is Mutable and not Thread-Safe.
- java.text.SimpleDateFormat is not Thread-Safe.
- Less Performance.

Java SE 8's Date and Time API has the following Advantages compare to Java's OLD Date API.

- Very simple to use.
- Human Readable Syntax that is More Readability.
- All API is Thread-Safe.
- Better Performance.

Why do we need new Date and Time API in Java SE 8?Explain how Java SE 8 Data and Time API solves issues of Old Java Date API?

We need Java 8's Date and Time API to develop Highly Performance, Thread-Safe and Highly Scalable Java Applications.

Java 8's Date and Time API solves all Java's Old Date API issues by following Immutability and Thread-Safety principles.

What are the Differences between Java's OLD Java Date API and Java 8's Date and Time API?

Differences between Java's OLD Java Date API and Java 8's Date and Time API:

S.NO.	JAVA'S OLD JAVA DATE API	JAVA 8'S DATE AND TIME API

1.	Available before Java 8 too.	It is introduced in Java SE 8
2.	Not Thread Safe.	Thread Safe.
3.	Mutable API.	Immutable API.
4.	Less Performance.	Better Performance.
5.	Less Readability.	More Readability.
6.	It's not recommended to use as its deprecated.	It's always recommended to use.
7.	Not Extendable.	Easy to Extend.
8.	It defines months values from 0 to 11, that is January = 0.	It defines months values from 1 to 12, that is January = 1.
9.	It's an old API.	It's a new API.

What is Multiple Inheritance? How Java 8 supports Multiple Inheritance?

Multiple Inheritance means a class can inherit or extend characteristics and features from more than one parent class.

In Java 7 or Earlier, Multiple Inheritance is not possible because Java follows "A class should extend one and only one class or abstract class" Rule. However, it's possible to provide Multiple Implementation Inheritance using Interface because Java follows "A class can extend any number of Interfaces" Rule.

However, Java 8 supports "Implementing Methods in Interfaces" by introducing new features: Default methods in Interface. Because of this feature, Java 8 supports Multiple Inheritance with some limitations.

JDBC Interview Questions and Answers

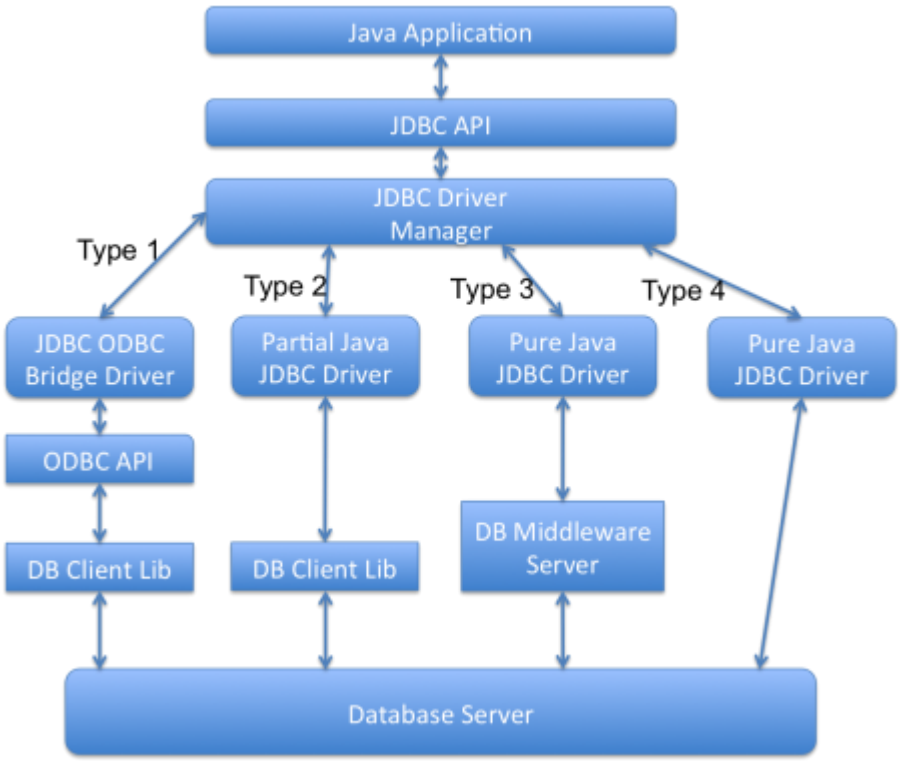
1. What is JDBC API and when do we use it?

Java DataBase Connectivity API allows us to work with relational databases. JDBC API interfaces and classes are part of `java.sql` and `javax.sql` package. We can use JDBC API to get the database connection, run SQL queries and stored procedures in the database server and process the results.

JDBC API is written in a way to allow loose coupling between our Java program and actual JDBC drivers that makes our life easier in switching from one database to another database servers easily.

2. What are different types of JDBC Drivers?

There are four types of JDBC drivers. Any java program that works with database has two parts, first part is the JDBC API and second part is the driver that does the actual work.



1. **JDBC-ODBC Bridge plus ODBC Driver (Type 1)**: It uses ODBC driver to connect to database. We should have ODBC drivers installed to connect to database, that's why this driver is almost obsolete.
2. **Native API partly Java technology-enabled driver (Type 2)**: This driver converts JDBC class to the client API for the database servers. We should have database client API installed. Because of extra dependency on database client API drivers, this is also not preferred driver.
3. **Pure Java Driver for Database Middleware (Type 3)**: This driver sends the JDBC calls to a middleware server that can connect to different type of databases. We should have a middleware server installed to work with this driver. This adds to extra network calls and slow performance and thats why not widely used JDBC driver.
4. **Direct-to-Database Pure Java Driver (Type 4)**: This driver converts the JDBC calls to the network protocol understood by the database server. This solution is simple and suitable for database connectivity over the network. However for this solution, we should use database specific drivers, for example OJDBC jars by Oracle for Oracle DB and MySQL Connector/J for MySQL databases.

3. How does JDBC API helps us in achieving loose coupling between Java Program and JDBC Drivers API?

JDBC API uses Java Reflection API to achieve loose coupling between java programs and JDBC Drivers. If you look at a simple JDBC example, you will notice that all the programming is done in terms of JDBC API and Driver comes in picture only when it's loaded through reflection using `Class.forName()` method.

I think this is one of the best example of using Reflection in core java classes to make sure that our application doesn't work directly with Drivers API and that makes it very easy to move from one database to another. Please read more at JDBC Example.

4. What is JDBC Connection? Explain steps to get Database connection in a simple java program.

JDBC Connection is like a Session created with the database server. You can also think Connection is like a Socket connection from the database server.

Creating a JDBC Connection is very easy and requires two steps:

1. Register and Load the Driver: Using `Class.forName()`, Driver class is registered to the DriverManager and loaded in the memory.
2. Use DriverManager to get the Connection object: We get connection object from `DriverManager.getConnection()` by passing Database URL String, username and password as argument.

```
Connection con = null;
try{
  // load the Driver Class
  Class.forName("com.mysql.jdbc.Driver");

  // create the connection now
```

```
  con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/UserDB",
                                    "pankaj",
                                    "pankaj123");
  }catch (SQLException e) {
                  System.out.println("Check database is UP and configs
are correct");
                  e.printStackTrace();
  }catch (ClassNotFoundException e) {
                  System.out.println("Please include JDBC MySQL jar in
classpath");
                  e.printStackTrace();
  }
```

5. What is the use of JDBC DriverManager class?

JDBC `DriverManager` is the factory class through which we get the Database Connection object. When we load the JDBC Driver class, it registers itself to the DriverManager, you can look up the JDBC Driver classes source code to check this.

Then when we call `DriverManager.getConnection()` method by passing the database configuration details, DriverManager uses the registered drivers to get the Connection and return it to the caller program.

6. How to get the Database server details in java program?

We can use `DatabaseMetaData` object to get the database server details. When the database connection is created successfully, we can get the meta data object by calling *getMetaData()* method. There are so many methods in DatabaseMetaData that we can use to get the database product name, it's version and configuration details.

```
DatabaseMetaData metaData = con.getMetaData();
String dbProduct = metaData.getDatabaseProductName();
```

7. What is JDBC Statement?

JDBC API `Statement` is used to execute SQL queries in the database. We can create the Statement object by calling Connection *getStatement()* method. We can use Statement to execute static SQL queries by passing query through different execute methods such as execute(), executeQuery(), executeUpdate() etc.

Since the query is generated in the java program, if the user input is not properly validated it can lead to SQL injection issue, more details can be found at SQL Injection Example.

By default, only one ResultSet object per Statement object can be open at the same time. Therefore, if we want to work with multiple ResultSet objects, then each must have been generated by different Statement objects. All execute() methods in the Statement interface implicitly close a statment's current ResultSet object if an open one exists.

8. What is the difference between execute, executeQuery, executeUpdate?

Statement *execute(String query)* is used to execute any SQL query and it returns TRUE if the result is an ResultSet such as running Select queries. The output is FALSE when there is no ResultSet object such as running Insert or Update queries. We can use *getResultSet()* to get the ResultSet and*getUpdateCount()* method to retrieve the update count.

Statement *executeQuery(String query)* is used to execute Select queries and returns the ResultSet. ResultSet returned is never null even if there are no records matching the query. When executing select queries we should use executeQuery method so that if someone tries to execute insert/update statement it will throw

java.sql.SQLException with message “executeQuery method can not be used for update”.

Statement executeUpdate(String query) is used to execute Insert/Update/Delete (DML) statements or DDL statements that returns nothing. The output is int and equals to the row count for SQL Data Manipulation Language (DML) statements. For DDL statements, the output is 0.

You should use execute() method only when you are not sure about the type of statement else use executeQuery or executeUpdate method.

9. What is JDBC PreparedStatement?

JDBC `PreparedStatement` object represents a precompiled SQL statement. We can use it’s setter method to set the variables for the query.

Since PreparedStatement is precompiled, it can then be used to efficiently execute this statement multiple times. PreparedStatement is better choice that Statement because it automatically escapes the special characters and avoid SQL injection attacks.

10. How to set NULL values in JDBC PreparedStatement?

We can use PreparedStatement setNull() method to bind the null variable to a parameter. The setNull method takes index and SQL Types as argument, for example `ps.setNull(10, java.sql.Types.INTEGER);`.

11. What is the use of getGeneratedKeys() method in Statement?

Sometimes a table can have auto generated keys used to insert the unique column value for primary key. We can use

Statement `getGeneratedKeys()` method to get the value of this auto generated key.

12. What are the benefits of PreparedStatement over Statement?

Some of the benefits of PreparedStatement over Statement are:

- PreparedStatement helps us in preventing SQL injection attacks because it automatically escapes the special characters.
- PreparedStatement allows us to execute dynamic queries with parameter inputs.
- PreparedStatement is faster than Statement. It becomes more visible when we reuse the PreparedStatement or use it's batch processing methods for executing multiple queries.
- PreparedStatement helps us in writing object Oriented code with setter methods whereas with Statement we have to use String Concatenation to create the query. If there are multiple parameters to set, writing Query using String concatenation looks very ugly and error prone.

13. What is the limitation of PreparedStatement and how to overcome it?

One of the limitation of PreparedStatement is that we can't use it directly with IN clause statements. Some of the alternative approaches to use PreparedStatement with IN clause are;

0. **Execute Single Queries** – very slow performance and not recommended
1. **Using Stored Procedure** – Database specific and hence not suitable for multiple database applications.
2. **Creating PreparedStatement Query dynamically** – Good approach but looses the benefit of cached PreparedStatement.
3. **Using NULL in PreparedStatement Query** – A good approach when you know the maximum number of variables inputs, can be extended to allow unlimited parameters by executing in parts.

A more detailed analysis can be found at JDBC PreparedStatement IN clause alternatives.

14. What is JDBC ResultSet?

JDBC `ResultSet` is like a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next() method moves the cursor to the next row. If there are no more rows, next() method returns false and it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce ResultSet objects that are scrollable and/or updatable using below syntax.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
```

A ResultSet object is automatically closed when the Statement object that generated it is closed, re-executed, or used to retrieve the next result from a sequence of multiple results.

We can use ResultSet getter method with column name or index number starting from 1 to retrieve the column data.

15. What are different types of ResultSet?

There are different types of ResultSet objects that we can get based on the user input while creating the Statement. If you will look into the Connection methods,

you will see that createStatement() and prepareStatement() method are overloaded to provide ResultSet type and concurrency as input argument.

There are three types of ResultSet object.

0. **ResultSet.TYPE_FORWARD_ONLY**: This is the default type and cursor can only move forward in the result set.
1. **ResultSet.TYPE_SCROLL_INSENSITIVE**: The cursor can move forward and backward, and the result set is not sensitive to changes made by others to the database after the result set was created.
2. **ResultSet.TYPE_SCROLL_SENSITIVE**: The cursor can move forward and backward, and the result set is sensitive to changes made by others to the database after the result set was created.

Based on the concurrency there are two types of ResultSet object.

3. **ResultSet.CONCUR_READ_ONLY**: The result set is read only, this is the default concurrency type.
4. **ResultSet.CONCUR_UPDATABLE**: We can use ResultSet update method to update the rows data.

16. What is the use of setFetchSize() and setMaxRows() methods in Statement?

We can use `setMaxRows(int i)` method to limit the number of rows that the database returns from the query. You can achieve the same thing using SQL query itself. For example in MySQL we can use LIMIT clause to set the max rows that will be returned by the query.

Understanding **fetchSize** can be tricky, for that you should know how Statement and ResultSet works. When we execute a query in the database, the result is obtained and maintained in the database cache and ResultSet is returned. ResultSet is the cursor that has the reference to the result in the database.

Let's say we have a query that returns 100 rows and we have set fetchSize to 10, so in every database trip JDBC driver will fetch only 10 rows and hence there will be 10 trips to fetch all the rows. Setting optimal fetchSize is helpful when you need a lot of processing time for each row and number of rows in the result is huge.

We can set fetchSize through Statement object but it can be overridden through ResultSet object setFetchSize() method.

17. How to use JDBC API to call Stored Procedures?

Stored Procedures are group of SQL queries that are compiled in the database and can be executed from JDBC API. JDBC `CallableStatement` can be used to execute stored procedures in the database. The syntax to initialize CallableStatement is;

```
CallableStatement stmt = con.prepareCall("{call
insertEmployee(?,?,?,?,?,?)}");
stmt.setInt(1, id);
stmt.setString(2, name);
stmt.setString(3, role);
stmt.setString(4, city);
stmt.setString(5, country);

//register the OUT parameter before calling the stored procedure
stmt.registerOutParameter(6, java.sql.Types.VARCHAR);

stmt.executeUpdate();
```

We need to register the OUT parameters before executing the CallableStatement. More details about this can be found at JDBC CallableStatement Example.

18. What is JDBC Batch Processing and what are it's benefits?

Sometimes we need to run bulk queries of similar kind for a database, for example loading data from CSV files to relational database tables. As we know that we have option to use Statement or PreparedStatement to execute queries. Apart from that JDBC API provides Batch Processing feature through which we can execute bulk of queries in one go for a database.

JDBC API supports batch processing through Statement and PreparedStatement `addBatch()` and`executeBatch()` methods.

Batch Processing is faster than executing one statement at a time because the number of database calls are less, read more at JDBC Batch Processing Example.

19. What is JDBC Transaction Management and why do we need it?

By default when we create a database connection, it runs in auto-commit mode. It means that whenever we execute a query and it's completed, the commit is fired automatically. So every SQL query we fire is a transaction and if we are running some DML or DDL queries, the changes are getting saved into database after every SQL statement finishes.

Sometimes we want a group of SQL queries to be part of a transaction so that we can commit them when all the queries runs fine and if we get any exception, we have a choice of rollback all the queries executed as part of the transaction.

JDBC API provide method `setAutoCommit(boolean flag)` through which we can disable the auto commit feature of the connection. We should disable auto commit only when it's required because the transaction will not be committed unless we call the commit() method on connection. Database servers uses table locks to achieve transaction management and it's resource intensive process. So we should commit the transaction as soon as we are done with it. Read more with example program atJDBC Transaction Management Example.

20. How to rollback a JDBC transaction?

We can use Connection object `rollback()` method to rollback the transaction. It will rollback all the changes made by the transaction and release any database locks currently held by this Connection object.

21. What is JDBC Savepoint? How to use it?

Sometimes a transaction can be group of multiple statements and we would like to rollback to a particular point in the transaction. JDBC Savepoint helps us in creating checkpoints in a transaction and we can rollback to that particular checkpoint.

Any savepoint created for a transaction is automatically released and become invalid when the transaction is committed, or when the entire transaction is rolled back. Rolling a transaction back to a savepoint automatically releases and makes invalid any other savepoints that were created after the savepoint in question. Read more at JDBC Savepoint Example.

22. What is JDBC DataSource and what are it's benefits?

JDBC DataSource is the interface defined in `javax.sql` package and it is more powerful that DriverManager for database connections. We can use DataSource to create the database connection and Driver implementation classes does the actual work for getting connection. Apart from getting Database connection, DataSource provides some additional features such as:

- Caching of PreparedStatement for faster processing
- Connection timeout settings
- Logging features
- ResultSet maximum size threshold
- Connection Pooling in servlet container using JNDI support

Read more about DataSource at JDBC DataSource Example.

23. How to achieve JDBC Connection Pooling using JDBC DataSource and JNDI in Apache Tomcat Server?

For web applications deployed in a servlet container, creating JDBC connection pool is very easy and involve only few steps.

0. Creating JDBC JNDI resource in the container configuration files, usually server.xml or context.xml. For example

`server.xml`

```
<Resource name="jdbc/MyDB"
      global="jdbc/MyDB"
      auth="Container"
      type="javax.sql.DataSource"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/UserDB"
      username="pankaj"
      password="pankaj123"

      maxActive="100"
      maxIdle="20"
      minIdle="5"
      maxWait="10000"/>
```

`context.xml`

```
<ResourceLink name="jdbc/MyLocalDB"
                global="jdbc/MyDB"
                auth="Container"
                type="javax.sql.DataSource" />
```

1. In web application, using InitialContext to look up the JNDI resource configured in the first step and then get the connection.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)
ctx.lookup("java:/comp/env/jdbc/MyLocalDB");
```

For a complete example, read Tomcat DataSource JNDI Example.

24. What is Apache DBCP API?

If you use `DataSource` to get the Database connection, usually the code to get the connection is tightly coupled with the Driver specific DataSource implementation. Also most of the code is boiler-plate code except the choice of the DataSource implementation class.

Apache DBCP helps us in getting rid of these issues by providing DataSource implementation that works as an abstraction layer between our program and different JDBC drivers. Apache DBCP library depends on Commons Pool library, so make sure they both are in the build path.

For a complete example, read Apache DBCP Example.

25. What is JDBC Connection isolation levels?

When we use JDBC Transactions for data integrity, DBMS uses locks to block access by others to the data being accessed by the transaction. DBMS uses locks to prevent Dirty Read, Non-Repeatable Reads and Phantom-Read issue.

JDBC transaction isolation level is used by DBMS to use the locking mechanism, we can get the isolation level information through Connection getTransactionIsolation() method and set it with setTransactionIsolation() method.

ISOLATION LEVEL	TRANSACTION	DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
TRANSACTION_NONE	Not Supported	Not Applicable	Not Applicable	Not Applicable
TRANSACTION_READ_COMMITTED	Supported	Prevented	Allowed	Allowed
TRANSACTION_READ_UNCOMMITTED	Supported	Allowed	Allowed	Allowed
TRANSACTION_REPEATABLE_READ	Supported	Prevented	Prevented	Allowed
TRANSACTION_SERIALIZABLE	Supported	Prevented	Prevented	Prevented

26. What is JDBC RowSet? What are different types of RowSet?

JDBC `RowSet` holds tabular data in more flexible ways that ResultSet. All RowSet objects are derived from ResultSet, so they have all the capabilities of ResultSet with some additional features. RowSet interface is defined in `javax.sql` package.

Some additional features provided by RowSet are:

- Functions as Java Beans with properties and their getter-setter methods. RowSet uses JavaBeans event model and they can send notifications to any registered component for events such as cursor movement, update/insert/delete of a row and change to RowSet contents.
- RowSet objects are scrollable and updatable by default, so if DBMS doesn't support scrollable or updatable ResultSet, we can use RowSet to get these features.

RowSet are broadly divided into two types:

2. **Connected RowSet Objects** – These objects are connected to database and are most similar to ResultSet object. JDBC API provides only one connected RowSet

object `javax.sql.rowset.JdbcRowSet` and it's standard implementation class is `com.sun.rowset.JdbcRowSetImpl`

3. **Disconnected RowSet Objects** – These RowSet objects are not required to connected to a database, so they are more lightweight and serializable. They are suitable for sending data over a network. There are four types of disconnected RowSet implementations.
    - CachedRowSet – They can get the connection and execute a query and read the ResultSet data to populate the RowSet data. We can manipulate and update data while it is disconnected and reconnect to database and write the changes.
    - WebRowSet derived from CachedRowSet – They can read and write XML document.
    - JoinRowSet derived from WebRowSet – They can form SQL JOIN without having to connect to a data source.
    - FilteredRowSet derived from WebRowSet – We can apply filtering criteria so that only selected data is visible.

27. What is the different between ResultSet and RowSet?

RowSet objects are derived from ResultSet, so they have all the features of ResultSet with some additional features. One of the huge benefit of RowSet is that they can be disconnected and that makes it lightweight and easy to transfer over a network.

Whether to use ResultSet or RowSet depends on your requirements but if you are planning to use ResultSet for longer duration, then a disconnected RowSet is better choice to free database resources.

28. What are common JDBC Exceptions?

Some of the common JDBC Exceptions are:

0. java.sql.SQLException – This is the base exception class for JDBC exceptions.

1. java.sql.BatchUpdateException – This exception is thrown when Batch operation fails, but it depends on the JDBC driver whether they throw this exception or the base SQLException.
2. java.sql.SQLWarning – For warning messages in SQL operations.
3. java.sql.DataTruncation – when a data values is unexpectedly truncated for reasons other than its having exceeded MaxFieldSize.

29. What is CLOB and BLOB datatypes in JDBC?

Character Large OBjects (CLOBs) are character string made up of single-byte characters with an associated code page. This data type is appropriate for storing text-oriented information where the amount of information can grow beyond the limits of a regular VARCHAR data type (upper limit of 32K bytes).

Binary Large OBjects (BLOBs) are binary string made up of bytes with no associated code page. This data type can store binary data larger than VARBINARY (32K limit). This data type is good for storing image, voice, graphical, and other types of business or application-specific data.

30. What is “dirty read” in JDBC? Which isolation level prevents dirty read?

When we work with transactions, there is a chance that a row is updated and at the same time other query can read the updated value. This results in a dirty read because the updated value is not permanent yet, the transaction that has updated the row can rollback to previous value resulting in invalid data.

Dirty Read is prevented by isolation levels TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ and TRANSACTION_SERIALIZABLE.

31. What is 2 phase commit?

When we work in distributed systems where multiple databases are involved, we are required to use 2 phase commit protocol. 2 phase commit protocol is an atomic commitment protocol for distributed systems. In the first phase, transaction

manager sends commit-request to all the transaction resources. If all the transaction resources are OK, then transaction manager commits the transaction changes for all the resources. If any of the transaction resource responds as Abort, then the transaction manager can rollback all the transaction changes.

32. What are the different types of locking in JDBC?

On a broad level, there are two types of locking mechanism to prevent data corruption because of more than one user working with the same data.

0. Optimistic Locking – Locking the record only when update is taking place
1. Pessimistic Locking – Locking the record from the select to read, update and commit phase.

Apart from that some DBMS systems provide locking mechanism to lock single row, table or database.

33. What do you understand by DDL and DML statements?

Data Definition Language (DDL) statements are used to define the database schema. Create, Alter, Drop, Truncate, Rename statements comes under DDL statements and usually they don't return any result.

Data Manipulation Language (DML) statements are used to manipulate data in the database schema. Select, Insert, Update, Delete, Call etc are example of DML statements.

34. What is difference between java.util.Date and java.sql.Date?

java.util.Date contains information about the date and time whereas java.sql.Date contains information only about the date, it doesn't have time information. So if

you have to keep time information in the database, it is advisable to use Timestamp or DateTime fields.

35. How to insert an image or raw data into database?

We can use BLOB to insert image or raw binary data into database.

36. What is phantom read and which isolation level prevents it?

A phantom read is the situation where a transaction executes a query multiple times and get different data. Suppose a transaction is executing a query to get data based on a condition and then another transaction inserts a row that matches the condition. Now when same transaction will execute the query again, a new row will be part of the result set. This new row is referred as Phantom Row and this situation is termed as Phantom Read.

Phantom read can be prevented only with TRANSACTION_SERIALIZABLE isolation level.

37. What is SQL Warning? How to retrieve SQL warnings in the JDBC program?

SQLWarning is the subclass of SQLException and we can retrieve it by calling getWarnings() method on Connection, Statement, and ResultSet objects. SQL Warnings doesn’t stops the execution of the script but alerts the user about the warning.

38. How to invoke Oracle Stored Procedure with Database Objects as IN/OUT?

If Oracle Stored Procedure has IN/OUT parameters as DB Objects then we need to create an Object array of the same size in the program and then use it to create

Oracle STRUCT object. Then we can set this STRUCT object for the database object by calling setSTRUCT() method and work with it.

39. When do we get java.sql.SQLException: No suitable driver found?

You get No suitable driver found exception when the SQL URL String is not properly formatted. You can get this exception in both simple java application using DriverManager or with JNDI resource using DataSource. The exception stack trace looks like below.

```
org.apache.tomcat.dbcp.dbcp.SQLNestedException: Cannot create JDBC
driver of class 'com.mysql.jdbc.Driver' for connect URL
''jdbc:mysql://localhost:3306/UserDB'
  at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.createConnectionFactory(Bas
icDataSource.java:1452)
  at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.createDataSource(BasicDataS
ource.java:1371)
  at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.getConnection(BasicDataSour
ce.java:1044)


java.sql.SQLException: No suitable driver found for
'jdbc:mysql://localhost:3306/UserDB
  at java.sql.DriverManager.getConnection(DriverManager.java:604)
  at java.sql.DriverManager.getConnection(DriverManager.java:221)
  at
com.journaldev.jdbc.DBConnection.getConnection(DBConnection.java:24)
  at
com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:15)
Exception in thread "main" java.lang.NullPointerException
  at
com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:16)
```

While debugging this exception, just check the URL getting printed in the logs, as in above logs the URL String is ‘jdbc:mysql://localhost:3306/UserDB whereas it should be jdbc:mysql://localhost:3306/UserDB.

40. What are JDBC Best Practices?

Some of the JDBC Best Practices are:

- Database resources are heavy, so make sure you close it as soon as you are done with it. Connection, Statement, ResultSet and all other JDBC objects have close() method defined to close them.
- Always close the result set, statement and connection explicitly in the code, because if you are working in connection pooling environment, the connection might be returned to the pool leaving open result sets and statement objects resulting in resource leak.
- Close the resources in the finally block to make sure they are closed even in case of exception scenarios.
- Use batch processing for bulk operations of similar kind.
- Always use PreparedStatement over Statement to avoid SQL Injection and get pre-compilation and caching benefits of PreparedStatement.
- If you are retrieving bulk data into result set, setting an optimal value for fetchSize helps in getting good performance.
- The database server might not support all isolation levels, so check it before assuming.
- More strict isolation levels result in slow performance, so make sure you have optimal isolation level set for your database connections.
- If you are creating database connections in a web application, try to use JDBC DataSource resources using JNDI context for re-using the connections.
- Try to use disconnected RowSet when you need to work with ResultSet for a long time.

Rules for writing Constructor:

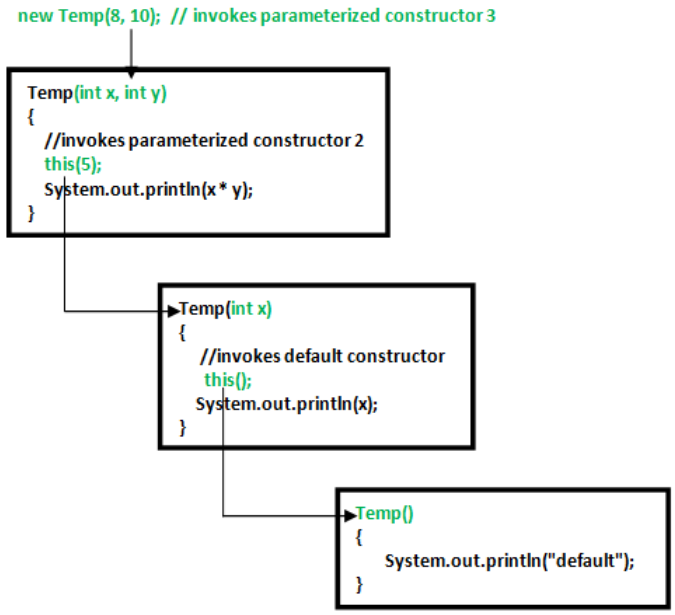
- Constructor(s) of a class must has same name as the class name in which it resides.
- A constructor in Java can not be abstract, final, static and Synchronized.
- Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

Does constructor return any value?

- There are no “return value” statements in constructor, but constructor returns current class instance. We can write ‘return’ inside a constructor.

Why do we need constructor chaining ?

This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.



toString

You're not explicitly calling `toString()`, but implicitly you are:
See:

```
System.out.println(foo); // foo is a non primitive variable
```

System is a class, with a static field out, of type PrintStream. So you're calling the println(Object) method of a PrintStream.
It is implemented like this:

```
public void println(Object x) {
    String s = String.valueOf(x);
    synchronized (this) {
        print(s);
```

```
        newLine();
    }
}
```

As we see, it's calling the String.valueOf(Object) method.
This is implemented as follows:

```
public static String valueOf(Object obj) {
    return (obj == null) ? "null" : obj.toString();
}
```

And here you see, that toString() is called.

finalize() method In Java:

finalize() method is a protected and non-static method of **java.lang.Object** class. This method will be available in all objects you create in java. This method is used to perform some final operations or clean up operations on an object before it is removed from the memory. you can override the finalize() method to keep those operations you want to perform before an object is destroyed. Here is the general form of finalize() method.

```
protected void finalize() throws Throwable
{
    //Keep some resource closing operations here
}
```

Garbage Collection In Java :

Whenever you run a java program, JVM creates three threads. 1) main thread 2) Thread Scheduler 3) Garbage Collector Thread. In these three threads, main thread is a user thread and remaining two are daemon threads which run in background.

The task of main thread is to execute the main() method. The task of thread scheduler is to schedule the threads. The task of garbage collector thread is to sweep out abandoned objects from the heap memory. Abandoned objects or dead objects are those objects which does not have live references. Garbage collector thread before sweeping out an abandoned object, it calls finalize() method of that object. After finalize() method is executed, object is destroyed from the memory. That means clean up operations which you have kept in the finalize() method are executed before an object is destroyed from the memory.

Garbage collector thread does not come to heap memory whenever an object becomes abandoned. It comes once in a while to the heap memory and at that time if it sees any abandoned objects, it sweeps out those objects after calling finalize() method on them. Garbage collector thread calls finalize() method only once for one object.

Let's discuss some interesting points about garbage collection and finalize() method.

Some Interesting Points About Garbage Collection And finalize() method In Java :

1) In some scenarios, finalize() method is not at all called by the garbage collector thread. For example, When I executed the below program in my system, finalize() method of Class A is not at all executed.

```
class A
{
    int i = 50;

    @Override
    protected void finalize() throws Throwable
    {
        System.out.println("From Finalize Method");
    }
}

public class Test
{
   public static void main(String[] args)
   {
      //Creating two instances of class A

      A a1 = new A();

      A a2 = new A();

      //Assigning a2 to a1

      a1 = a2;

      //Now both a1 and a2 will be pointing to same object

      //An object earlier referred by a1 will become abandoned

      System.out.println("done");
   }
}
```

2) You can make finalize() method to be executed forcefully using either **Runtime.getRuntime().runFinalization()** OR **Runtime.runFinalizersOnExit(true)**. But, both the methods have disadvantages. Runtime.getRuntime().runFinalization() makes the just best effort to execute finalize() method. It is not gauranteed that it will execute finalize() method. Runtime.runFinalizersOnExit(true) is deprecated in JDK because some times it runs finalize() method on live objects also.

```
class A
{
    int i = 50;

    @Override
    protected void finalize() throws Throwable
    {
        System.out.println("From Finalize Method");
    }
}

public class Test
{
   public static void main(String[] args)
   {
      //Creating two instances of class A

      A a1 = new A();

      A a2 = new A();

      //Assigning a2 to a1

      a1 = a2;

      //Making finalize() method to execute forcefully

      Runtime.getRuntime().runFinalization();
```

```
      System.out.println("done");
   }
}
```

3) you can call garbage collector explicitly using **System.gc()** or **RunTime.getRunTime().gc()**. Again it is just a request to garbage collector not a command. It is up to garbage collector to honour this request.

```
class A
{
    int i;

    public A(int i)
    {
        this.i = i;
    }

    @Override
    protected void finalize() throws Throwable
    {
        System.out.println("From Finalize Method, i = "+i);
    }
}

public class Test
{
   public static void main(String[] args)
   {
       //Creating two instances of class A

       A a1 = new A(10);

       A a2 = new A(20);

       //Assigning a2 to a1

       a1 = a2;

       //Now both a1 and a2 will be pointing same object

       //An object earlier referred by a1 will become abandoned

           //Calling garbage collector thread explicitly

       System.gc();              //OR call Runtime.getRuntime().gc();

       System.out.println("done");
   }
}
```

4) finalize() methods are not chained like constructors.i.e there is no calling statement to super class finalize() method inside the finalize() method of sub class. You need to explicitly call super class finalize() method.

```
1     protected void finalize() throws Throwable
2     {
3         System.out.println("From Finalize Method");
4
5         //Calling super class finalize() method explicitly
6
7         super.finalize();
8     }
```

5) Exceptions occurred in finalize() method are not propagated. They are ignored by the garbage collector.

6) You can call finalize() method explicitly on an object before it is abandoned. When you call, only operations kept in finalize() method are performed on an object. Object will not be destroyed from the memory.

```
1       class A
2       {
3           int i;
4
5           public A(int i)
6           {
7               this.i = i;
8           }
9
10          @Override
11          protected void finalize() throws Throwable
12          {
13              System.out.println("From Finalize Method, i = "+i);
14
15              //Calling super class finalize() method explicitly
16
17              super.finalize();
18          }
19      }
20
21      public class Test
22      {
23         public static void main(String[] args)
24         {
25              //Creating two instances of class A
26
27              A a1 = new A(10);
28
29              A a2 = new A(20);
30
31              //Calling finalize() method of a1 before it is abandoned
32              try
33              {
34                  a1.finalize();
35              }
36              catch (Throwable e)
37              {
38                  e.printStackTrace();
39              }
40
41              //Assigning a2 to a1
42
43              a1 = a2;
44
45              //Now both a1 and a2 will be pointing same object
46
47              //An object earlier referred by a1 will become abandoned
48
49              System.out.println("done");
50         }
51      }
```

7) finalize() method on an abandoned object is called only once by the garbage collector thread. GC ignores finalize() method called on an object by the developer.