Java Interview
Questions & Answers

JournalDev Pankaj Kumar

Table of Content

A. Core Java

1. What are the important features of Java 8 release?

Java 8 has been released in March 2014, so it's one of the hot topic in java interview questions. If you answer this question clearly, it will show that you like to keep yourself up-to-date with the latest technologies.

Java 8 has been one of the biggest release after Java 5 annotations and generics. Some of the important features of Java 8 are:

1. Interface changes with default and static methods
2. Functional interfaces and Lambda Expressions
3. Java Stream API for collection classes
4. Java Date Time API

I strongly recommend to go through above links to get proper understanding of each one of them, also read Java 8 Features.

2. 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.

3. 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.

4. 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.

5. 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.

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. 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.

9. 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.

10. What is the importance of main method in Java?

main() method is the entry point of any standalone java application. The syntax of main method is public static void main(String args[]).

main method is public and static so that java can access it without initializing the class. The input parameter is an array of String through which we can pass runtime arguments to the java program. Check this post to learn how to compile and run java program.

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. 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[]).

13. 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.

14. 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.

[sociallocker id="2713″]

15. 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. A java class can only have public or default access modifier. Read Java Access Modifiers to learn more about these in detail.

16. 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.

17. 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.

Read more in detail at java static keyword.

18. 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.

19. 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.

Read more about inner classes at java inner class.

20. 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.

21. 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. Read more at Java Automatic Resource Management.

22. 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.

Read more at Java multi catch block.

23. 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.

24. What is an interface?

Interfaces are core part of java programming language and used a lot not only in JDK but also java design patterns, most of the frameworks and tools. Interfaces provide a way to achieve abstraction in java and used 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. Read more at java interface.

25. 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 an 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.

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

abstract keyword is used to create abstract class whereas interface is the keyword for interfaces.

Abstract classes can have method implementations whereas interfaces can't.

A class can extend only one abstract class but it can implement multiple interfaces.

We can run abstract class if it has main() method whereas we can't run an interface.

Some more differences in detail are at Difference between Abstract Class and Interface.

27. 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.

28. 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.

29. 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.

Read more at Wrapper classes in Java.

30. 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.

31. 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. Read more at java annotations.

32. 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.

Read Java Reflection Tutorial to get in-depth knowledge of reflection api.

33. 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. Read more with example at Java Composition example.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. What are different types of classloaders?

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

1. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
2. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA_HOME/lib/ext directory.
3. 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.

40. 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.

41. 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");
    }
}
```

42. 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.

43. 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;
}
```

44. 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.

45. Can we have try without catch block?

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

46. 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 method System.gc(). For a detailed analysis of Heap Memory and Garbage Collection, please read Java Garbage Collection.

47. 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.

48. How to run a JAR file through command prompt?

We can run a jar file using java command but it requires Main-Class entry in jar manifest file. Main-Class is the entry point of the jar and used by java command to execute the class. Learn more at java jar file.

49. 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.

50. 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.

51. 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.

52. 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.

53. What is difference between Heap and Stack Memory?

Major difference between Heap and Stack memory are as follows:

- Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
- 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.
- 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.

54. Java Compiler is stored in JDK, JRE or JVM?

The task of java compiler is to convert java program into bytecode, we have javac executable for that. So it must be stored in JDK, we don't need it in JRE and JVM is just the specs.

55. What will be the output of following programs?

1. **static method in class**

```
package com.journaldev.util;

public class Test {

    public static String toString(){
        System.out.println("Test toString called");
        return "";
    }
```

```
    public static void main(String args[]){
        System.out.println(toString());
    }
}
```

Answer: The code won't compile because we can't have an Object class method with static keyword. You will get compile time error as "This static method cannot hide the instance method from Object". The reason is that static method belongs to class and since every class base is Object, we can't have same method in instance as well as in class.

2. static method invocation

```
package com.journaldev.util;

public class Test {

    public static String foo(){
        System.out.println("Test foo called");
        return "";
    }

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

Answer: Well this is a strange situation. We all have seen NullPointerException when we invoke a method on object that is NULL. The compiler will give warning as "The static method foo() from the type Test should be accessed in a static way" but when executing it will work and prints "Test foo called".

Ideally Java API should have given error when a static method is called from an object rather than giving warning, but I think it's too late now to impose this. And most strange of all is that even though obj is null here, when invoking static method it works fine. I think it's working fine because Java runtime figures out that foo() is a static method and calls it on the class loaded into the memory and doesn't use the object at all, so no NullPointerException.

B. String Interview Questions

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

2. 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.

3. Write a method to check if input String is Palindrome?

A String is said to be Palindrome if its 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 isPalindrome(String str) {
    if (str == null)
        return false;
    StringBuilder strBuilder = new StringBuilder(str);
    strBuilder.reverse();
    return strBuilder.toString().equals(str);
}
```

4. 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), "");
}
```

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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
```

18. 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.

19. 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.

20. Does String is thread-safe in Java?

Strings are immutable, so we can't change its 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.

21. 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 its processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

1. String Programming Questions

1. What is the output of below program?

```
package com.journaldev.strings;

public class StringTest {

    public static void main(String[] args) {
        String s1 = new String("pankaj");
        String s2 = new String("PANKAJ");
        System.out.println(s1 = s2);
    }

}
```

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.

2. What is the output of below program?

```
package com.journaldev.strings;

public class Test {

     public void foo(String s) {
     System.out.println("String");
     }

     public void foo(StringBuffer sb){
     System.out.println("StringBuffer");
     }

     public static void main(String[] args) {
        new Test().foo(null);
    }

}
```

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.

3. What is the output of below code snippet?

```
String s1 = new String("abc");
String s2 = new String("abc");
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.

4. What will be output of below code snippet?

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

It will print false because s2 is not of type String. If you will look at the equal's 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.

5. What will be output of below program?

```
String s1 = "abc";
String s2 = new String("abc");
s2.intern();
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.

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

```
String s1 = new String("Hello");
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.

C. Collections Interview Questions

1. What are Collection related features in Java 8?

Java 8 has brought major changes in the Collection API. Some of the changes are:

1. Java Stream API for collection classes for supporting sequential as well as parallel processing
2. Iterable interface is extended with forEach() default method that we can use to iterate over a collection. It is very helpful when used with lambda expressions because it's argument Consumer is a function interface.
3. Miscellaneous Collection API improvements such as forEachRemaining(Consumer action) method in Iterator interface, Map replaceAll(), compute(), merge() methods.

2. What is Java Collections Framework? List out some benefits of Collections framework?

Collections are used in every programming language and initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **Array**. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.
Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package.
Some of the benefits of collections framework are;

- Reduced development effort by using core collection classes rather than implementing our own collection classes.
- Code quality is enhanced with the use of well tested collections framework classes.
- Reduced effort for code maintenance by using collection classes shipped with JDK.
- Reusability and Interoperability

3. What is the benefit of Generics in Collections Framework?

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error. This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don't need to use casting and *instanceof* operator.

I would highly recommend to go through **Java Generic Tutorial** to understand generics in a better way.

4. What are the basic interfaces of Java Collections Framework?

Collection is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn't provide any direct implementations of this interface.

Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

List is an ordered collection and can contain duplicate elements. You can access any element from its index. List is more like array with dynamic length.

A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are Queue, Dequeue, Iterator, SortedSet, SortedMap and ListIterator.

5. Why Collection doesn't extend Cloneable and Serializable interfaces?

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don't.

A lot of the Collection implementations have a public clone method. However, it doesn't really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation.
The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.
So mandating cloning and serialization in all implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

6. Why Map interface doesn't extend Collection interface?

Although Map interface and its implementations are part of Collections Framework, Map are not collections and collections are not Map. Hence it doesn't make sense for Map to extend Collection or vice versa.

If Map extends Collection interface, then where are the elements? Map contains key-value

pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn't fit into the "group of elements" paradigm.

7. What is an Iterator?

Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using *iterator()* method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Java Collection iterator provides a generic way for traversal through the elements of a collection and implements **Iterator Design Pattern**.

8. What is difference between Enumeration and Iterator interface?

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.

Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make its functionality clear.

9. Why there is not method like Iterator.add() to add elements to the collection?

The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.

10. Why Iterator don't have a method to get next element directly without moving the cursor?

It can be implemented on top of current Iterator interface but since its use will be rare, it doesn't make sense to include it in the interface that everyone has to implement.

11. What is difference between Iterator and ListIterator?

- We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
- Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
- ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

12. What are different ways to iterate over a list?

We can iterate over a list in two different ways – using iterator and using for-each loop.

```
List&lt;String&gt; strList = new ArrayList&lt;&gt;();
//using for-each loop
for(String obj : strList){
    System.out.println(obj);
}
//using iterator
Iterator&lt;String&gt; it = strList.iterator();
while(it.hasNext()){
    String obj = it.next();
    System.out.println(obj);
}
```

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw ConcurrentModificationException.

13. What do you understand by iterator fail-fast property?

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

14. What is difference between fail-fast and fail-safe?

Iterator fail-safe property work with the clone of underlying collection, hence it's not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe.

Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException.
Check this post for CopyOnWriteArrayList Example.

15.How to avoid ConcurrentModificationException while iterating a collection?

We can use concurrent collection classes to avoid ConcurrentModificationException while iterating over a collection, for example CopyOnWriteArrayList instead of ArrayList. Check this post for ConcurrentHashMap Example.

16. Why there are no concrete implementations of Iterator interface?

Iterator interface declare methods for iterating a collection but its implementation is responsibility of the Collection implementation classes. Every collection class that returns an iterator for traversing has its own Iterator implementation nested class. This allows collection classes to choose whether iterator is fail-fast or fail-safe. For example ArrayList iterator is fail-fast whereas CopyOnWriteArrayList iterator is fail-safe.

17. What is UnsupportedOperationException?

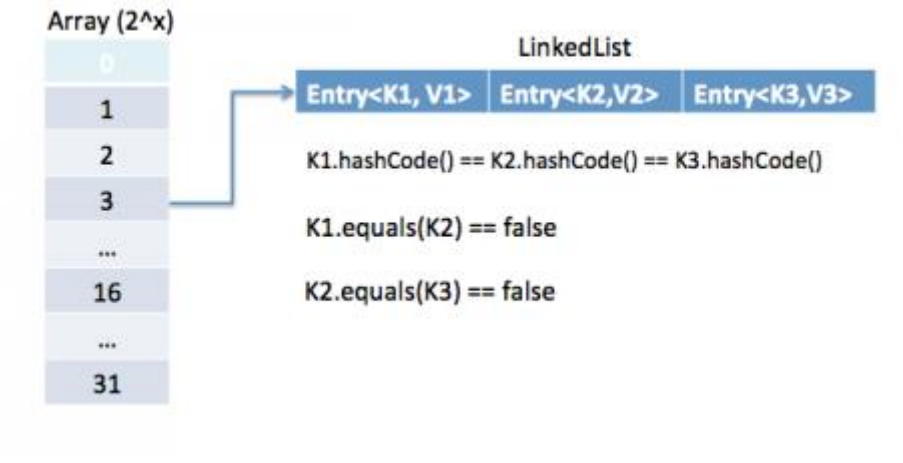
UnsupportedOperationException is the exception used to indicate that the operation is not supported. It's used extensively in JDK classes, in collections framework java.util.Collections.UnmodifiableCollection throws this exception for all add and remove operations.

18. How HashMap works in Java?

HashMap stores key-value pair in Map.Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.

When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.

When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it's value. Below image will explain these detail clearly.



The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is **16** and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry, if map size is

greater than threshold, HashMap rehashes the contents of map into a new array with a larger capacity. The capacity is always power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it's good idea to initialize the HashMap with correct capacity and load factor.

19. What is the importance of hashCode() and equals() methods?

HashMap uses Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key's might produce same hashCode() and equals() output and in that case rather than storing it at different location, HashMap will consider them same and overwrite them.

Similarly all the collection classes that doesn't store duplicate data use hashCode() and equals() to find duplicates, so it's very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.

```
  If o1.equals(o2), then o1.hashCode() == o2.hashCode()should always be true.
    If o1.hashCode() == o2.hashCode is true, it does not mean that o1.equals(o2) will be true.
```

20. Can we use any class as Map key?

We can use any class as Map Key, however following points should be considered before using them.

- If the class overrides equals() method, it should also override hashCode() method.
- The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.
- If a class field is not used in equals(), you should not use it in hashCode() method.
- Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.
  For example, let's say I have a class MyKey that I am using for HashMap key.

```
//MyKey name argument passed is used for equals() and hashCode()
MyKey key = new MyKey("Pankaj"); //assume hashCode=1234
myHashMap.put(key, "Value");

// Below code will change the key hashCode() and equals()
// but it's location is not changed.
key.setName("Amit"); //assume new hashCode=7890

//below will return null, because HashMap will try to look for key
```

```
//in the same index as it was stored but since key is mutated,
//there will be no match and it will return null.
myHashMap.get(new MyKey("Pankaj"));
```

This is the reason why String and Integer are mostly used as HashMap keys.

21. What are different Collection views provided by Map interface?

Map interface provides three collection views:

1. **Set keySet()**: Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.
2. **Collection values()**: Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator's own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.
3. **Set<Map.Entry<K, V>> entrySet()**: Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

[sociallocker id="2713"]

22. What is difference between HashMap and Hashtable?

HashMap and Hashtable both implements Map interface and looks similar, however there are following difference between HashMap and Hashtable.

1. HashMap allows null key and values whereas Hashtable doesn't allow null key and values.

2. Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.
3. LinkedHashMap was introduced in Java 1.4 as a subclass of HashMap, so incase you want iteration order, you can easily switch from HashMap to LinkedHashMap but that is not the case with Hashtable whose iteration order is unpredictable.
4. HashMap provides Set of keys to iterate and hence it's fail-fast but Hashtable provides Enumeration of keys that doesn't support this feature.
5. Hashtable is considered to be legacy class and if you are looking for modifications of Map while iterating, you should use ConcurrentHashMap.

23. How to decide between HashMap and TreeMap?

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

24. What are similarities and difference between ArrayList and Vector?

ArrayList and Vector are similar classes in many ways.

1. Both are index based and backed up by an array internally.
2. Both maintains the order of insertion and we can get the elements in the order of insertion.
3. The iterator implementations of ArrayList and Vector both are fail-fast by design.
4. ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector.

1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
2. ArrayList is faster than Vector because it doesn't have any overhead because of synchronization.
3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

25. What is difference between Array and ArrayList? When will you use Array over ArrayList?

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.
Arrays are fixed size whereas ArrayList size is dynamic.
Arrays doesn't provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc.

Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

- If the size of list is fixed and mostly used to store and traverse them.
- For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.
- If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

26. What is difference between ArrayList and LinkedList?

ArrayList and LinkedList both implement List interface but there are some differences between them.

1. ArrayList is an index based data structure backed by Array, so it provides random access to its elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to its previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.
2. Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.
3. LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

27. Which collection classes provide random access of its elements?

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to its elements. Download java collections pdf for more information.

28. What is EnumSet?

java.util.EnumSet is Set implementation to use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. EnumSet is not synchronized and null elements are not allowed. It also provides some useful methods like copyOf(Collection c), of(E first, E… rest) and complementOf(EnumSet s).

Check this post for java enum tutorial.

29. Which collection classes are thread-safe?

Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment. Java 1.5 Concurrent API included some

collection classes that allows modification of collection while iteration because they work on the clone of the collection, so they are safe to use in multi-threaded environment.

30. What are concurrent Collection Classes?

Java 1.5 Concurrent package (java.util.concurrent) contains thread-safe collection classes that allow collections to be modified while iterating. By design Iterator implementation in java.util packages are fail-fast and throws ConcurrentModificationException. But Iterator implementation in java.util.concurrent packages are fail-safe and we can modify the collection while iterating. Some of these classes are CopyOnWriteArrayList, ConcurrentHashMap, CopyOnWriteArraySet.

Read these posts to learn about them in more detail.

- Avoid ConcurrentModificationException
- CopyOnWriteArrayList Example
- HashMap vs ConcurrentHashMap

31. What is BlockingQueue?

java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue interface is part of java collections framework and it's primarily used for implementing producer consumer problem. We don't need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue as it's handled by implementation classes of BlockingQueue.

Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc.
Check this post for use of BlockingQueue for producer-consumer problem.

32. What is Queue and Stack, list their differences?

Both Queue and Stack are used to store data before processing them. java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it's not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.
Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.

Stack is a class that extends Vector whereas Queue is an interface.

33. What is Collections Class?

java.util.Collections is a utility class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

This class contains methods for collection framework algorithms, such as binary search, sorting, shuffling, reverse etc.

34. What is Comparable and Comparator interface?

Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if "this" object is less than, equal to, or greater than the object passed as argument.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can't chose the field on which we want to sort the Object.

Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

Check this post for use of Comparable and Comparator interface to sort objects.

35. What is difference between Comparable and Comparator interface?

Comparable and Comparator interfaces are used to sort collection or array of objects.

Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic.

Comparator interface is used to provide different algorithms for sorting and we can chose the comparator we want to use to sort the given collection of objects.

36. How can we sort a list of Objects?

If we need to sort an array of Objects, we can use Arrays.sort(). If we need to sort a list of objects, we can use Collections.sort(). Both these classes have overloaded sort() methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator). Collections internally uses Arrays sorting method, so both of them have same performance except that Collections take sometime to convert list to array.

37. While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?

We can create a read-only collection using Collections.unmodifiableCollection(Collection c) method before passing it as argument, this will make sure that any operation to change the collection will throw UnsupportedOperationException.

38. How can we create a synchronized collection from given collection?

We can use Collections.synchronizedCollection(Collection c) to get a synchronized (thread-safe) collection backed by the specified collection.

39. What are common algorithms implemented in Collections Framework?

Java Collections Framework provides algorithm implementations that are commonly used such as sorting and searching. Collections class contain these method implementations. Most of these algorithms work on List but some of them are applicable for all kinds of collections.
Some of them are sorting, searching, shuffling, min-max values.

40. What is Big-O notation? Give some examples?

The Big-O notation describes the performance of an algorithm in terms of number of elements in a data structure. Since Collection classes are actually data structures, we usually tend to use Big-O notation to choose the collection implementation to use based on time, memory and performance.

Example 1: ArrayList get(index i) is a constant-time operation and doesn't depend on the number of elements in the list. So it's performance in Big-O notation is O(1).
Example 2: A linear search on array or list performance is O(n) because we need to search through entire list of elements to find the element.

41. What are best practices related to Java Collections Framework?

- Chosing the right type of collection based on the need, for example if size is fixed, we might want to use Array over ArrayList. If we have to iterate over the Map in order of insertion, we need to use TreeMap. If we don't want duplicates, we should use Set.
- Some collection classes allows to specify the initial capacity, so if we have an estimate of number of elements we will store, we can use it to avoid rehashing or resizing.

- Write program in terms of interfaces not implementations, it allows us to change the implementation easily at later point of time.
- Always use Generics for type-safety and avoid ClassCastException at runtime.
- Use immutable classes provided by JDK as key in Map to avoid implementation of hashCode() and equals() for our custom class.
- Use Collections utility class as much as possible for algorithms or to get read-only, synchronized or empty collections rather than writing own implementation. It will enhance code-reuse with greater stability and low maintainability.

42. What is Java Priority Queue?

PriorityQueue is an unbounded queue based on a priority heap and the elements are ordered in their natural order or we can provide Comparator for ordering at the time of creation. PriorityQueue doesn't allow null values and we can't add any object that doesn't provide natural ordering or we don't have any comparator for them for ordering. Java PriorityQueue is not thread-safe and provided O(log(n)) time for enqueing and dequeing operations. Check this post for java priority queue example.

43. Why can't we write code as List<Number> numbers = new ArrayList<Integer>();?

Generics doesn't support sub-typing because it will cause issues in achieving type safety. That's why List<T> is not considered as a subtype of List<S> where S is the super-type of T. To understanding why it's not allowed, let's see what could have happened if it has been supported.

```
List<Long> listLong = new ArrayList<Long>();
listLong.add(Long.valueOf(10));
List<Number> listNumbers = listLong; // compiler error
listNumbers.add(Double.valueOf(1.23));
```

As you can see from above code that IF generics would have been supporting sub-typing, we could have easily add a Double to the list of Long that would have caused ClassCastException at runtime while traversing the list of Long.

44. Why can't we create generic array? Or write code as List<Integer>[] array = new ArrayList<Integer>[10];

We are not allowed to create generic arrays because array carry type information of it's elements at runtime. This information is used at runtime to throw ArrayStoreException if elements type doesn't match to the defined type. Since generics type information gets erased at runtime by Type Erasure, the array store check would have been passed where it should have failed. Let's understand this with a simple example code.

```
List<Integer>[] intList = new List<Integer>[5]; // compile error
Object[] objArray = intList;
List<Double> doubleList = new ArrayList<Double>();
```

```
doubleList.add(Double.valueOf(1.23));
objArray[0] = doubleList; // this should fail but it would pass because
at runtime intList and doubleList both are just List
```

Arrays are covariant by nature i.e S[] is a subtype of T[] whenever S is a subtype of T but generics doesn't support covariance or sub-typing as we saw in last question. So if we would have been allowed to create generic arrays, because of type erasure we would not get array store exception even though both types are not related.

To know more about Generics, read **Java Generics Tutorial**.

D. Java Multi-Threading and Concurrency

1. What is the difference between Process and Thread?

A process is a self-contained execution environment and it can be seen as a program or application whereas Thread is a single task of execution within the process. Java runtime environment runs as a single process which contains different classes and programs as processes. Thread can be called lightweight process. Thread requires less resources to create and exists in the process, thread shares the process resources.

2. What are the benefits of multi-threaded programming?

In Multi-Threaded programming, multiple threads are executing concurrently that improves the performance because CPU is not idle in case some thread is waiting to get some resources. Multiple threads share the heap memory, so it's good to create multiple threads to execute some task rather than creating multiple processes. For example, Servlets are better in performance than CGI because Servlet support multi-threading but CGI doesn't.

3. What is difference between user Thread and daemon Thread?

When we create a Thread in java program, it's known as user thread. A daemon thread runs in background and doesn't prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

4. How can we create a Thread in Java?

There are two ways to create Thread in Java – first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class. Read this post to learn more about creating threads in java.

5. What are different states in lifecycle of Thread?

When we create a Thread in java program, its state is New. Then we start the thread that change its state to Runnable. Thread Scheduler is responsible to

allocate CPU to threads in Runnable thread pool and change their state to Running. Other Thread states are Waiting, Blocked and Dead. Read this post to learn more about life cycle of thread.

[sociallocker id="2713"]

6. Can we call run() method of a Thread class?

Yes, we can call run() method of a Thread class but then it will behave like a normal method. To actually execute it in a Thread, we need to start it using **Thread.start()** method.

7. How can we pause the execution of a Thread for specific time?

We can use Thread class sleep() method to pause the execution of Thread for certain time. Note that this will not stop the processing of thread for specific time, once the thread awake from sleep, its state gets changed to runnable and based on thread scheduling, and it gets executed.

8. What do you understand about Thread Priority?

Every thread has a priority, usually higher priority thread gets precedence in execution but it depends on Thread Scheduler implementation that is OS dependent. We can specify the priority of thread but it doesn't guarantee that higher priority thread will get executed before lower priority thread. Thread priority is an *int* whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

9. What is Thread Scheduler and Time Slicing?

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads. Once we create and start a thread, its execution depends on the implementation of Thread Scheduler. Time Slicing is the process to divide the available CPU time to the available runnable threads. Allocation of CPU time to threads can be based on thread priority or the thread waiting for longer time will get more priority in getting CPU time. Thread scheduling can't be controlled by java, so it's always better to control it from application itself.

10. What is context-switching in multi-threading?

Context Switching is the process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time. Context Switching is the essential feature for multitasking operating system and support for multi-threaded environment.

11. How can we make sure main() is the last thread to finish in Java Program?

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function. Here is an article about Thread join method.

12. How does thread communicate with each other?

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource. Check this post to learn more about thread wait, notify and notifyAll.

13. Why thread communication methods wait(), notify() and notifyAll() are in Object class?

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that's why it's part of Object class so that every class in java has these essential methods for inter thread communication.

14. Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

15. Why Thread sleep() and yield() methods are static?

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That's why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

16. How can we achieve thread safety in Java?

There are several ways to achieve thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes. Learn more at thread safety tutorial.

17. What is volatile keyword in Java?

When we use volatile keyword with a variable, all the threads read its value directly from the memory and don't cache it. This makes sure that the value read is the same as in the memory.

18. Which is more preferred – Synchronized method or synchronized block?

Synchronized block is more preferred way because it doesn't lock the Object, synchronized methods lock the Object and if there are multiple synchronization blocks in the class, even though they are not related, it will stop them from execution and put them in wait state to get the lock on Object.

19. How to create daemon thread in Java?

Thread class setDaemon(true) can be used to create daemon thread in java. We need to call this method before calling start() method else it will throw IllegalThreadStateException.

20. What is ThreadLocal?

Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it's variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables.

Every thread has it's own ThreadLocal variable and they can use it's get() and set() methods to get the default value or change its value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread. Check this post for small example program showing ThreadLocal Example.

21. What is Thread Group? Why it's advised not to use it?

ThreadGroup is a class which was intended to provide information about a thread group. ThreadGroup API is weak and it doesn't have any functionality that is not provided by Thread. Two of the major feature it had are to get the list of active threads in a thread group and to set the uncaught exception handler for the thread. But Java 1.5 has added *setUncaughtExceptionHandler(UncaughtExceptionHandler eh)* method using which we can add uncaught exception handler to the thread. So ThreadGroup is obsolete and hence not advised to use anymore.

```
t1.setUncaughtExceptionHandler(new UncaughtExceptionHandler(){

    @Override
    public void uncaughtException(Thread t, Throwable e) {
        System.out.println("exception occured:"+e.getMessage());
    }

});
```

22. What is Java Thread Dump, How can we get Java Thread dump of a Program?

Thread dump is list of all the threads active in the JVM, thread dumps are very helpful in analyzing bottlenecks in the application and analyzing deadlock situations. There are many ways using which we can generate Thread dump – Using Profiler, Kill -3 command, jstack tool etc. I prefer jstack tool to generate thread dump of a program because it's easy to use and comes with JDK installation. Since it's a terminal based tool, we can create script to generate thread dump at regular intervals to analyze it later on. Read this post to know more about generating thread dump in java.

23. What is Deadlock? How to analyze and avoid deadlock situation?

Deadlock is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

To analyze a deadlock, we need to look at the java thread dump of the application, we need to look out for the threads with state as BLOCKED and then the resources it's waiting to lock, every resource has a unique ID using which we can find which thread is already holding the lock on the object.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation, read this post to learn how to analyze deadlock in java with sample program.

24. What is Java Timer Class? How to schedule a task to run after specific interval?

java.util.Timer is a utility class that can be used to schedule a thread to be executed at certain time in future. Java Timer class can be used to schedule a task to be run one-time or to be run at regular intervals.

java.util.TimerTask is an **abstract class** that implements Runnable interface and we need to extend this class to create our own TimerTask that can be scheduled using java Timer class.

Check this post for java Timer example.

25. What is Thread Pool? How can we create Thread Pool in Java?

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.

A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.

java.util.concurrent.Executors provide implementation of java.util.concurrent.Executor interface to create the thread pool in java. Thread Pool Example program shows how to create and use Thread Pool in java. Or read ScheduledThreadPoolExecutor Example to know how to schedule tasks after certain delay.

26. What will happen if we don't override Thread class run() method?

Thread class run() method code is as shown below.

```
public void run() {
    if (target != null) {
        target.run();
    }
}
```

Above target set in the init() method of Thread class and if we create an instance of Thread class as new TestThread(), it's set to null. So nothing will happen if we don't override the run() method. Below is a simple example demonstrating this.

```
public class TestThread extends Thread {

    //not overriding Thread.run() method

    //main method, can be in other class too
    public static void main(String args[]){
        Thread t = new TestThread();
        System.out.println("Before starting thread");
        t.start();
        System.out.println("After starting thread");
    }
}
```

It will print only below output and terminate.

```
Before starting thread
After starting thread
```

[/sociallocker]

1. Java Concurrency Interview

1. What is atomic operation? What are atomic classes in Java Concurrency API?

Atomic operations are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.

int++ is not an atomic operation. So by the time one threads read its value and increment it by one, other thread has read the older value leading to wrong result.

To solve this issue, we will have to make sure that increment operation on count is atomic, we can do that using Synchronization but Java 5 java.util.concurrent.atomic provides wrapper classes for int and long that can be used to achieve this atomically without usage of Synchronization. Go to this article to learn more about atomic concurrent classes.

2. What is Lock interface in Java Concurrency API? What are it's benefits over synchronization?

Lock interface provide more extensive locking operations than can be obtained using synchronized methods and statements. They allow more flexible structuring, may have quite different properties, and may support multiple associated Condition objects.

The advantages of a lock are

- it's possible to make them fair
- It's possible to make a thread responsive to interruption while waiting on a Lock object.
- it's possible to try to acquire the lock, but return immediately or after a timeout if the lock can't be acquired
- it's possible to acquire and release locks in different scopes, and in different orders

  Read more at **Java Lock Example**.

3. What is Executors Framework?

In Java 5, Executor framework was introduced with the java.util.concurrent.Executor interface.

The Executor framework is a framework for standardizing invocation, scheduling, execution, and control of asynchronous tasks according to a set of execution policies.

Creating a lot many threads with no bounds to the maximum threshold can cause application to run out of heap memory. So, creating a ThreadPool is a better solution as a finite number of threads can be pooled and reused. Executors framework facilitate process of creating Thread pools in java. Check out this post to learn with example code to create thread pool using Executors framework.

4. What is BlockingQueue? How can we implement Producer-Consumer problem using Blocking Queue?

java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue doesn't accept null values and throw NullPointerException if you try to store null value in the queue.

BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.

BlockingQueue interface is part of java collections framework and it's primarily used for implementing producer consumer problem. Check this post for producer-consumer problem implementation using BlockingQueue.

5. What is Callable and Future?

Java 5 introduced java.util.concurrent.Callable interface in concurrency package that is similar to Runnable interface but it can return any Object and able to throw Exception.

Callable interface use Generic to define the return type of Object. Executors class provide useful methods to execute Callable in a thread pool. Since callable tasks run in parallel, we have to wait for the returned Object. Callable tasks return java.util.concurrent.Future object. Using Future we can find out the status of the Callable task and get the returned Object. It provides get() method that can wait for the Callable to finish and then return the result. Check this post for Callable Future Example.

6. What is FutureTask Class?

FutureTask is the base implementation class of Future interface and we can use it with Executors for asynchronous processing. Most of the time we don't need to use FutureTask class but it comes real handy if we want to override some of the methods of Future interface and want to keep most of the base implementation. We can just extend this class and override the methods

according to our requirements. Check out **Java FutureTask Example** post to learn how to use it and what are different methods it has.

7. What are Concurrent Collection Classes?

Java Collection classes are fail-fast which means that if the Collection will be changed while some thread is traversing over it using iterator, the iterator.next() will throw ConcurrentModificationException.

Concurrent Collection classes support full concurrency of retrievals and adjustable expected concurrency for updates. Major classes are ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet, check this post to learn how to avoid ConcurrentModificationException when using iterator.

8. What is Executors Class?

Executors class provide utility methods for Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes.

Executors class can be used to easily create Thread Pool in java, also this is the only class supporting execution of Callable implementations.

9. What are some of the improvements in Concurrency API in Java 8?

Some important concurrent API enhancements are:

- ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.
- CompletableFuture that may be explicitly completed (setting its value and status).
- Executors newWorkStealingPool() method to create a work-stealing thread pool using all available processors as its target parallelism level.

Recommended Read: Java 8 Features

E. Java Exception Handling

1. What is Exception in Java?

Exception is an error event that can happen during the execution of a program and disrupts its 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.

Java Exception handling framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

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 its 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.

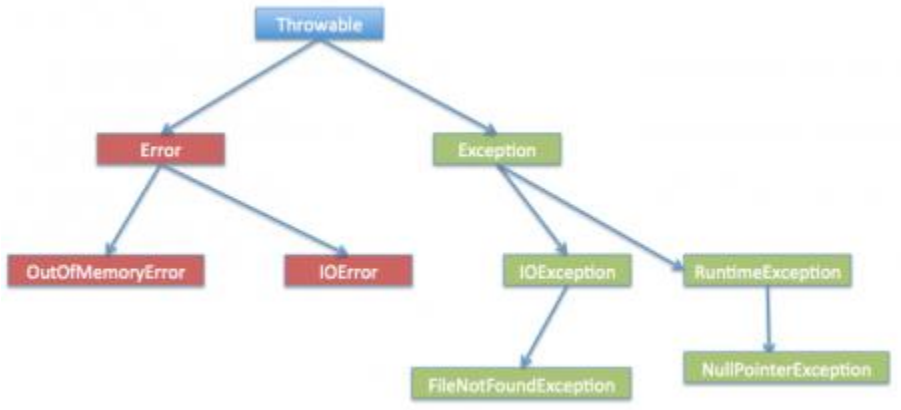
3. Explain Java Exception Hierarchy?

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 its 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());
}
```

Most of the time, we use finally block just to close the resources and sometimes we forget to close them and get runtime exceptions when the resources are exhausted. These exceptions are hard to debug and we might need to look into each place where we are using that type of resource to make sure we are closing it. So java 7 one of the improvement was **try-with-**

resources where we can create a resource in the try statement itself and use it inside the try-catch block. When the execution comes out of try-catch block, runtime environment automatically close these resources. Sample of try-catch block with this improvement is:

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

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 its subclasses to create our custom exception class. The custom exception class can have its 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.

a. What is the problem with below program?

```
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.

1. **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.

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

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException2 {

    public static void main(String[] args) {
        try {
            foo();
        } catch (IOException e) {
            e.printStackTrace();
        }catch(JAXBException e){
            e.printStackTrace();
        }catch(NullPointerException e){
            e.printStackTrace();
```

```
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public static void foo() throws IOException{

    }
}
```

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.

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

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException2 {

    public static void main(String[] args) {
        try {
            foo();
        } catch (IOException e) {
            e.printStackTrace();
        }catch(JAXBException e){
            e.printStackTrace();
        }catch(NullPointerException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

```
    public static void foo() throws IOException{

    }
}
```

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.

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

```
package com.journaldev.exceptions;

public class TestException3 {

    public static void main(String[] args) {
        try{
        bar();
        }catch(NullPointerException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }

        foo();
    }

    public static void bar(){

    }

    public static void foo() throws NullPointerException{

    }
}
```

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 signature in subclass to be exact same as superclass or we can remove throws clause from subclass method as shown below.

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

```
package com.journaldev.exceptions;
```

```
import java.io.IOException;

public class TestException4 {

    public void start() throws IOException{
    }

    public void foo() throws NullPointerException{

    }
}

class TestException5 extends TestException4{

    public void start() throws Exception{
    }

    public void foo() throws RuntimeException{

    }
}
```

The above program won't compile because exception object in multi-catch block is final and we can't change its 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.

Read more at Java 7 multi-catch block.

F. Java Programming Questions

1. What is the output of the below statements?

```
String s1 = "abc";
String s2 = "abc";
System.out.println("s1 == s2 is:" + s1 == s2);
```

2. What is the output of the below statements?

```
String s3 = "JournalDev";
int start = 1;
char end = 5;
System.out.println(start + end);
System.out.println(s3.substring(start, end));
```

3. What is the output of the below statements?

```
HashSet shortSet = new HashSet();
for (short i = 0; i < 100; i++) {
    shortSet.add(i);
    shortSet.remove(i - 1);
}
System.out.println(shortSet.size());
```

4. What will be the boolean “flag” value to reach the finally block?

```
try {
    if (flag) {
        while (true) {
        }
    } else {
        System.exit(1);
    }
} finally {
    System.out.println("In Finally");
}
```

5. What will be the output of the below statements?

```
String str = null;
String str1="abc";
System.out.println(str1.equals("abc") | str.equals(null));
```

Java Programming Test Question 1 Answer and Explanation

The given statements output will be “false” because in java + operator precedence is more than == operator. So the given expression will be evaluated to “s1 == s2 is:abc” == “abc” i.e. false.

Java Programming Test Question 2 Answer and Explanation

The given statements output will be “ourn”. First character will be automatically type caste to int. After that since in java first character index is 0, so it will start from ‘o’ and print till ‘n’. Note that in String *substring* function it leaves the end index.

Java Programming Test Question 3 Answer and Explanation

The size of the shortSet will be 100. Java Autoboxing feature has been introduced in JDK 5, so while adding the short to HashSet<Short> it will automatically convert it to Short object. Now “i-1” will be converted to int while evaluation and after that it will autoboxed to Integer object but there are no Integer object in the HashSet, so it will not remove anything from the HashSet and finally its size will be 100.

Java Programming Test Question 4 Answer and Explanation

The finally block will never be reached here. If flag will be TRUE, it will go into an infinite loop and if it’s false it’s exiting the JVM. So finally block will never be reached here.

Java Programming Test Question 5 Answer and Explanation

The given print statement will throw java.lang.NullPointerException because while evaluating the OR logical operator it will first evaluate both the literals and since str is null, .equals() method will throw exception. It’s always advisable to use short circuit logical operators i.e. “||” and “&&” which evaluates the literals values from left and since the first literal will return true, it will skip the second literal evaluation.

I hope that the above scenarios will help a bit in understanding some of the java concepts. Please try the questions before going to the solution and comment to let me know your score.

Copyright Notice

References

1. http://www.journaldev.com/2366/core-java-interview-questions-and-answers
2. http://www.journaldev.com/1321/java-string-interview-questions-and-answers
3. http://www.journaldev.com/1330/java-collections-interview-questions-and-answers
4. http://www.journaldev.com/1162/java-multi-threading-concurrency-interview-questions-with-answers
5. http://www.journaldev.com/2167/java-exception-interview-questions-and-answers
6. http://www.journaldev.com/370/5-java-programming-test-questions-for-interview