1. What is the difference between JDK and JRE?

JDK stands for Java Development Kit. It contains the tools and libraries for development of Java programs. It also contains compilers and debuggers needed to compile Java program,

JRE stands for Java Runtime Environment. This is included in JDK. JRE provides libraries and JVM that is required to run a Java program.

2. What is Java Virtual Machine (JVM)?

Java Virtual Machine (JVM) is an abstract machine that executes Java Bytecode. There are different JVM for different hardware and software platforms. So JVM is platform dependent. JVM is responsible for loading, verifying and executing the Bytecode on a platform.

3. What are the different types of memory areas allocated by JVM?

In java, JVM allocates memory to different processes, methods and objects. Some of the memory areas allocated by JVM are:

1. ClassLoader: It is a component of JVM used to load class files.

2. Class (Method) Area: It stores per-class structures such as the runtime constant pool, field and method data, and the code for methods.

3. Heap: Heap is created a runtime and it contains the runtime data area in which objects are allocated.

4. Stack: Stack stores local variables and partial results at runtime. It also helps in method invocation and return value. Each thread creates a private JVM stack at the time of thread creation.

5. Program Counter Register: This memory area contains the address of the Java virtual machine instruction that is currently being executed.

6. Native Method Stack: This area is reserved for all the native methods used in the application.

4. What is JIT compiler?

Just In Time compiler also known as JIT compiler is used for performance improvement in Java. It is enabled by default. It is compilation done at execution time rather earlier.
Java has popularized the use of JIT compiler by including it in JVM.

5. How Java platform is different from other platforms?

Java is a platform independent language. Java compiler converts Java code in to byte code that can be interpreted by JVM. There are JVM written for almost all the popular platforms in the world.

Java byte code can run on any supported platform in same way. Where as other languages require libraries compiled for a specific platform to run.

6. Why people say that Java is 'write once and run anywhere' language?

You can write Java code on Windows and compile it in Windows platform. The class and jar files that you get from Windows platform can run as it is on Unix environment. So it is a truly platform independent language.

Behind all this portability is Java byte code. Byte code generated by Java compiler can be interpreted by any JVM. So it becomes much easier to write programs in Java and expect those to run on any platform.

Java compiler javac compiles java code and JVM java runs that code.

7. How does ClassLoader work in Java?

In Java, ClassLoader is a class that is used to load files in JVM. ClassLoader loads files from their physical file locations e.g. Filesystem, Network location etc.

There are three main types of ClassLoaders in Java.

1. Bootstrap ClassLoader: This is the first ClassLoader. It loads classes from rt.jar file.

2. Extension ClassLoader: It loads class files from jre/lib/ext location.

3. Application ClassLoader: This ClassLoader depends on CLASSPATH to find the location of class files. If you specify your jars in CLASSPATH, then this ClassLoader will load them.

8. Do you think ‘main’ used for main method is a keyword in Java?

No, main is just a name of method. There can be multiple methods with same name main in a class file. It is not a keyword in Java.

9. Can we write main method as public void static instead of public static void?

No, you cannot write it like this. Any method has to first specify the modifiers and then the return value. The order of modifiers can change.

We can write static public void main() instead of public static void main().

10.In Java, if we do not specify any value for local variables, then what will be the default value of the local variables?

Java does not initialize local variables with any default value. So these variables will be just null by default.

11. Let say, we run a java class without passing any arguments. What will be the value of String array of arguments in Main method?

By default, the value of String array of arguments is empty in Java. It is not null.

12.What is the difference between byte and char data types in Java?

Both byte and char are numeric data types in Java. They are used to represent numbers in a specific range.

Major difference between them is that a byte can store raw binary data where as a char stores characters or text data.

Usage of char is E.g. char ch = ‘x’; Byte values range from -128 to 127.

A byte is made of 8 bits. But a char is made of 16 bits. So it is equivalent to 2 bytes.

OOPS

13.What are the main principles of Object Oriented Programming?

Main principles of Object Oriented Programming (OOPS) are: 1. Abstraction
2. Encapsulation
3. Inheritance

4. Polymorphism

14.What is the difference between Object Oriented Programming language and Object Based Programming language?

Object Oriented Programming languages like Java and C++ follow concepts of OOPS like- Encapsulation, Abstraction, Polymorphism and Inheritance etc.

Object Based Programming languages follow some features of OOPS but they do not provide support for Polymorphism and Inheritance. Egg. JavaScript, VBScript etc.

Object Based Programming languages provide support for Objects and you can build objects from constructor. They languages also support Encapsulation. These are also known as Prototype-oriented languages.

15.In Java what is the default value of an object reference defined as an instance variable in an Object?

All the instance variable object references in Java are null.

16.Why do we need constructor in Java?

Java is an object-oriented language, in which we create and use objects. A constructor is a piece of code similar to a method. It is used to create an object and set the initial state of the object.

A constructor is a special function that has same name as class name.

Without a constructor, there is no other way to create an object.

By default, Java provides a default constructor for every object. If we overload a constructor then we have to implement default constructor.

17.Why do we need default constructor in Java classes?

Default constructor is the no-argument constructor that is automatically generated by Java if no other constructor is defined.

Java specification says that it will provide a default constructor if there is no overloaded constructor in a class. But it does not say anything about the scenario in which we write an overloaded constructor in a class.

We need at least one constructor to create an object, that’s why Java provides a default constructor.

When we have overloaded constructor, then Java assumes that we want some custom treatment in our code. Due to which it does not provide default constructor. But it needs default constructor as per the specification. So it gives error.

18.What is the value returned by Constructor in Java?

When we call a constructor in Java, it returns the object created by it. That is how we create new objects in Java.

19.Can we inherit a Constructor?

No, Java does not support inheritance of constructor.

20. Why constructors cannot be final, static, or abstract in Java?

If we set a method as final it means we do not want any class to override it. But the constructor (as per Java Language Specification) cannot be overridden. So there is no use of marking it final.

If we set a method as abstract it means that it has no body and it should be implemented in a child class. But the constructor is called implicitly when the new keyword is used. Therefore it needs a body.

If we set a method as static it means that it belongs to the class, but not a particular object. The constructor is always called to initialize an object. Therefore, there is no use of marking constructor static.

Inheritance

21.What is the purpose of ‘this’ keyword in java?

In Java, ‘this’ keyword refers to current instance of the object.

It is useful for differentiating between instance variables and local variables.

It can be used to call constructors. Or it can be used to refer to the instance.

In case of method overriding, this is used for falling the method of current class.

22. Explain the concept of Inheritance?

Inheritance is an important concept in Object Oriented Programming. Some objects share certain characteristics and behavior. By using Inheritance, we can put the common behavior and characteristics in a base class which also known as super class. And then all the objects with common behavior inherit from this base class.

It is also represented by IS-A relationship.

Inheritance promotes, code reuse, method overriding and poly- morphism.

23. Which class in Java is superclass of every other class?

Java is an object oriented programming language. In Java, Object class is the superclass of every other class.

24. Why Java does not support multiple inheritance?

Multiple Inheritance means that a class can inherit behavior from two or more parent classes.

The issue with Multiple Inheritance is that both the parent classes may have different implementation for the same method. So they have different ways of doing the same thing. Now which implementation should the child class choose?

This leads to ambiguity in Multiple Inheritance. This is the main reason for Java not supporting Multiple Inheritance in implementation.

Lets say you have a class TV and another class AtomBomb. Both have method switchOn() but only TV has switchOff() method. If your class inherits from both these classes then you have an issue that you can switchOn() both parents, but switchOff will only switchOff() TV.

But you can implement multiple interfaces in Java.

25. In OOPS, what is meant by composition?

Composition is also known as “has-a” relationship. In composition, “has-a” relation relates two classes. E.g. Class Car has a steering wheel.

If a class holds the instance of another class, then it is called composition.

26. How aggregation and composition are different concepts?

In OOPS, Aggregation and Composition are the types of association relations. A composition is a strong relationship. If the composite object is destroyed, then all its parts are destroyed. E.g. A Car has a Steering Wheel. If Car object is destroyed, then there is no meaning of Steering Wheel.

In Aggregation, the relationship is weaker than Composition.

E.g. A Library has students. If a Library is destroyed, Students still exist. So Library and Student are related by Aggregation. A Library has Books. If Library is destroyed, the Books are also destroyed. Books of a Library cannot exist without the Library. So Book and Library are related by Composition.

27. Why there are no pointers in Java?

In Java there are references instead of pointers. These references point to objects in memory. But there is no direct access to these memory locations. JVM is free to move the objects within VM memory.

The absence of pointers helps Java in managing memory and garbage collection effectively. Also it provides developers with convenience of not getting worried about memory allocation and de- allocation.

28. If there are no pointers in Java, then why do we get NullPointerException?

In Java, the pointer equivalent is Object reference. When we use a . it points to object reference. So JVM uses pointers but programmers only see object references.

In case an object reference points to null object, and we try to access a method or member variable on it, then we get NullPointerException.

29. What is the purpose of ‘super’ keyword in java?

‘super’ keyword is used in the methods or constructor of a child class. It refers to immediate parent class of an object.

By using ‘super’ we can call a method of parent class from the method of a child class.

We can also call the constructor of a parent class from the constructor of a child class by using 'super' keyword.

30. Is it possible to use this() and super() both in same constructor?

No, Java does not allow using both super() and this() in same constructor. As per Java specification, super() or this() must be the first statement in a constructor.

31.What is the meaning of object cloning in Java?

Object.clone() method is used for creating an exact copy of the object in Java. It acts like a copy constructor. It creates and returns a copy of the object, with the same class and with all the fields having same values as of the original object.

One disadvantage of cloning is that the return type is an Object. It has to be explicitly cast to actual type.

Static

32. In Java, why do we use static variable?

Whenever we want to have a common property for all objects of a class, we use a class level variable i.e. a static variable.

This variable is loaded in memory only once at the time of class loading. So it saves memory, since it is not defined per object in Java.

33. Why it is not a good practice to create static variables in Java?

Static variables are common to all the objects of a class. If a new object is created, there is no need to test the value of static variable. Any code that uses static variable can be in any state. It can be within a new object or at a class level. So the scope of static variable is open ended in a Java class.

If we want tighter control on scope, then variables should be created at the object creation level.

Also defining static variables is not a good practice because they go against the principles of Object Oriented Programming.

34. What is the purpose of static method in Java?

Java provides the feature of static method to create behavior at the class level. The static method is common to all the objects of a class. We do not need to create any object of a class to call a static method. So it provides convenience of not creating an object for calling it.

Also a static method can access and modify static data members. This also helps in keeping the behavior as well as state at the class level.

35. Why do we mark main method as static in Java?

The main method in Java is marked as static, so that JVM can call it to start the program. If main method is not static, then which constructor will be called by Java process?

As such it is a known as convention to mark main method static in Java. But if we remove the static, then there will be ambiguity. Java process may not know which method of a class to call to start the program.

So this convention helps in Java process to identify the starting code for a program in class that is passed as an argument to java process.

36. In what scenario do we use a static block?

At times, there is a class that has static member variables. These variables need some complicated initialization. At this time static block helps as a tool to initialize complex static member variable initialization.

The static block is executed even before the execution of main.

Sometimes, we can also replace static block with a static method of class.

37. Is it possible to execute a program without defining a main() method?

No, with Java 7 onwards, you need a main() method to execute a program. In earlier versions of Java, there was a workaround available to use static blocks for execution. But now this gap has been closed.

38. What happens when static modifier is not mentioned in the signature of main method?

As per Java specification, main method has to be marked as static. It needs only one argument that is an array of String.

A program can compile with a non-static method. But on execution it will give NoSuchMethodError.

39. What is the difference between static method and instance method in Java?

Often, there is a need to define a behavior for a class that is not dependent on member variables of an object. Such behavior is captured in a static method. If there is a behavior dependent upon the member variables of an object, then we do not mark it static, it remains as instance method.

To call as static method, we do not need to create an object. We just call it with class name. But to call an instance method, we need to create/get an object first.

Instance member variables cannot be accessed by a static method. But an instance method can call both instance variables and static variables.

Method Overloading and Overriding

40. What is the other name of Method Overloading?

Method Overloading is also known as Static Polymorphism.

41.How will you implement method overloading in Java?

In Java, a class can have multiple methods with same name but different arguments. It is called Method Overloading. To implement method overloading we have to create two methods with same name in a class and do one/more of the following:

1. Different number of parameters

2. Different data type of parameters

3. Different sequence of data type of parameters

42. What kinds of argument variations are allowed in Method Overloading?

Method Overloading allows two methods with same name to differ in:

1. Number of parameters

2. Data type of parameters

3. Sequence of data type of parameters

43. Why it is not possible to do method overloading by changing return type of method in java?

If we change the return type of overloaded methods then it will lead to ambiguous behavior. How will clients know which method will return what type. Due to this different return type are not allowed in overloaded methods.

44. Is it allowed to overload main() method in Java?

Yes, Java allows users to create many methods with same name ‘main’. But only public static void main(String[] args) method is used for execution.

45. How do we implement method overriding in Java?

To override a method, we just provide a new implementation of a method with same name in subclass. So there will be at least two implementations of the method with same name. One implementation is in parent class. And another implementation is in child class.

46. Are we allowed to override a static method in Java?

No. Java does not allow overriding a static method. If you create a static method with same name in subclass, then it is a new method, not an overridden method.

47. Why Java does not allow overriding a static method?

To override a method, you need an instance of a class. Static method is not associated with any instance of the class. So the concept of overriding does not apply here.

Therefore, Java does not allow overriding a static method.

48. Is it allowed to override an overloaded method?

Yes. You can override an overloaded method in Java.

49. What is the difference between method overloading and method overriding in Java?

Differences between method overloading and overriding are:

1. Method overloading is static polymorphism. Method overriding is runtime polymorphism.
2. Method overloading occurs within the same class. Method overriding happens in two classes with hierarchy relationship.
3. Parameters must be different in method overloading. Parameters must be same in method overriding.
4. Method overloading is a compile time concept. Method overriding is a runtime concept.

50. Does Java allow virtual functions?

Yes. All instance methods in Java are virtual functions by default. Only class methods and private instance methods are not virtual methods in Java.

51.What is meant by covariant return type in Java?

A covariant return type of a method is one that can be replaced by a "narrower" type when the method is overridden in a subclass.

Let say class B is child of class A. There is a get() method in class A as well as class B. get() method of class A can return an instance of A, and get() method of class B return an instance of B. Here class B overrides get() method, but the return type is different.

Before Java 5, any method that overrides the method of parent class would have same return type.

From Java 5 onwards, a child class can override a method of parent class and the child class method can return an object that is child of object return by parent class method.

Polymorphism

52. What is Runtime Polymorphism?

Runtime Polymorphism or Dynamic Polymorphism is the polymorphism that exists at runtime. In case of method overriding it is not known which method will be called at runtime. Based on the type of object, JVM decides the exact method that should be called.

So at compile time it is not known which method will be called at run time.

53. Is it possible to achieve Runtime Polymorphism by data members in Java?

No. We need to create Runtime Polymorphism by implementing methods at two levels of inheritance in Java.

54. Explain the difference between static and dynamic binding?

In Static binding references are resolved at compile time. In Dynamic binding references are resolved at Run time.

E.g.
```
Person p = new Person();
p.walk(); // Java compiler resolves this binding at compile time.
```

```
public void walk(Object o){
((Person) o).walk(); // this is dynamic binding. }
```

Abstraction

55. What is Abstraction in Object Oriented programming?

Abstraction is the process of hiding certain implementation details of an object and showing only essential features of the object to outside world.

It is different from Abstract class in Java.

Abstraction process identifies commonalities and hides the complexity of implementation. It helps us in focusing on the interface that we share with the outside world.

56. How is Abstraction different from Encapsulation?

Abstraction happens at class level design. It results in hiding the implementation details. Encapsulation is also known as "Information Hiding". An example of encapsulation is marking the member variables private and providing getter and setter for these member variables.

57. What is an abstract class in Java?

An abstract class in Java has one or more abstract methods. An abstract method is just declared in the abstract class, but it is not implemented.

An abstract class has to be extended in Java and its abstract methods have to be implemented by a child class. Also Java does not allow new instance of Abstract class.

58. Is it allowed to mark a method abstract method without marking the class abstract?

No. Java specification says that if there is at least one abstract method in a class, the class has to be marked abstract.

59. Is it allowed to mark a method abstract as well as final?

No. It will be contradictory statement to mark a method abstract as well as final.

An abstract method has to be overridden by a child class. And a final method cannot be overridden. Therefore a method can be either abstract or final in Java.

60. Can we instantiate an abstract class in Java?

No. We cannot create an instance of an abstract class in Java.

61.What is an interface in Java?

An Interface in Java is an abstract type blueprint of a class. It contains the methods that a class must implement. It is like a protocol.

It has method signatures and constant declarations.

62. Is it allowed to mark an interface method as static?

Yes, from Java 8 onwards, we can define static and default methods in an interface. Prior to Java 8, it was not allowed.

63. Why an Interface cannot be marked as final in Java?

A final method cannot be overridden. But an interface method has to be implemented by another class. So the interface method cannot be marked as final.

64. What is a marker interface?

There are interfaces that do not have any data member or methods. These interfaces are called Marker interface.
E.g. Serializable, Cloneable, Remote etc.

65. What can we use instead of Marker interface?

We can use annotations instead of Marker interface.

66. How Annotations are better than Marker Interfaces?

Annotations serve the purpose of conveying metadata about the class to its consumers without creating a separate type for it.

Annotations are more powerful than a Marker interface. They allow programmers to pass more sophisticated information to classes that "consume" it.

67. What is the difference between abstract class and interface in Java?

Differences between Abstract class and Interface are as follows:

1. An abstract class can have implemented methods with body (non-abstract methods). Interface has only abstract methods. From Java 8 onwards, interface can have static/default methods in implemented form.

2. An abstract class can have instance member variables. An interface cannot have instance variables. It can only have constants.

3. An abstract class can have a constructor. Interface cannot have constructor. It has to be implemented by another class.

4. A class can extend only one abstract class. A class can implement more than one interface.

68. Does Java allow us to use private and protected modifiers for variables in interfaces?

No. All the variables in an interface are implicitly public.

69. How can we cast to an object reference to an interface reference?

An Object that implements an Interface can be cast to the same Interface. Since An Object implementing an Interface already provides implementation for the methods of that Interface, it is allowed to do so as per the rules of Inheritance.

Final

70. How can you change the value of a final variable in Java?

Java does not allow changing the value of a final variable. Once the value is set, it cannot be changed.

71.Can a class be marked final in Java?

Yes a class can be marked final in Java. Once a class is marked final, it cannot be extended.

72. How can we create a final method in Java?

To mark a method, add modifier final to that method. A final method can not be overridden by a child class.

73. How can we prohibit inheritance in Java?

If you mark a class final, it cannot be extended. This will prohibit the inheritance of that class in Java.

74. Why Integer class in final in Java?

Integer class is a wrapper for int. If it is not marked final, then any other class can extend it and modify the behavior of Integer operations. To avoid this Integer wrapper class is marked as final.

75. What is a blank final variable in Java?

When we declare a final variable without giving any initial value, then it is called blank final variable.

76. How can we initialize a blank final variable?

A blank final instance variable can be initialized in a constructor.

A blank final static variable can be initialized in the static block of class.

77. Is it allowed to declare main method as final?

Yes, we can mark the main method as final.

Package

78. What is the purpose of package in Java?

A package is used to encapsulate a group of classes, interfaces and sub-packages. Often, it is a hierarchical structure of storing information. It is easier to organize the related classes and sub- packages in this manner.

A Package also provides access protection for classes and interfaces. A package also helps in removing naming collision.

79. What is java.lang package?

In Java, java.lang package contains the classes that are fundamental to the design of Java programming language. The most important class in this package is Object class.

It also contains wrapper classes like- Integer, Boolean, Character etc. It provides Math class for mathematical operations.

80. Which is the most important class in Java?

It is an open-ended question with many answers. In my view, Object class is the most important class of Java programming language. It is the root of all the classes in Java. It provides some very important and fundamental methods.

81.Is it mandatory to import java.lang package every time?

No. By default, JVM loads it internally.

82. Can you import same package or class twice in your class?

If we import same package multiple times in a class, compiler includes it only once. So neither JVM nor Compiler gives any error/warning on including a package multiple times.

If you have two classes with same name, then you may get name collision on importing the class erroneously.

JVM internally loads the class only one time.

83. What is a static import in Java?

Static import is similar to normal import declaration. Normal import allows us to import classes from packages without using package qualifier. Static import allows us to import static members from a class without using class qualifier.

84. What is the difference between import static com.test.Fooclass and import com.test.Fooclass?

First import is a static import and the second import is normal import of a class. First import allows us to import static members of class.

Internationalization

85. What is Locale in Java?

A Locale object represents a specific geographical, political, or cultural region. It is used to locale-sensitive operations in Java.

It helps is following the local conventions of a country, native or region. These conventions can be for formatting the dates, money, numbers etc.

86. How will you use a specific Locale in Java?

To use a specific Locale, we need to load that Locale. We can use ResourceBundle.getBundle("Locale.UK") method to load a Locale.

Serialization

87. What is the serialization?

Serialization is a process converting an object into a byte array. This byte array represents the class, version and internal state of the object. JVM can use this byte array to transmit/read the object over a network.

88. What is the purpose of serialization?

Some of the uses of serialization are:

1. 2. 3.

4.

Communication: It is used for transmitting an object over network between two machines.
Persistence: We can store the object's state in a database and retrieve it from database later on.

Caching: Serialization can be used for caching to improve performance. We may need 10 minutes to build an object, but it may take just 10 seconds to de-serialize the object. Cross JVM Synchronization: It can be used in same way across multiple JVM that follow different architecture.

89. What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse process of serialization.

90. What is Serialization and Deserialization conceptually?

Serialization is to convert Object data into a stream of bytes

Deserialization is to convert a stream of bytes back into a copy of the original object.

91.Why do we mark a data member transient?

Member variables of an object are marked transient to indicate that they should not be serialized.

During serialization process the transient variables are not considered part of the persistent state of an object.

92. Is it allowed to mark a method as transient?

No, Java does not allow marking a method as transient. The transient keyword is valid only for member variables.

93. How does marking a field as transient makes it possible to serialize an object?

Let say we have a class ABC that implements Serializable interface, but it contains a member variable object of class XYZ that does not implement Serializable interface. Due to this it is not possible to Serialize the class ABC.

To solve this issue, we can mark the member variable XYZ as Transient in class ABC. This will allow us to serialize the class ABC.

94. What is Externalizable interface in Java?

Externalizable interface extends Serializable interface in Java. It is used for giving the Class control over saving and restoring the contents of its instances.

A class implements methods writeExternal() and readExternal() to store and restore the object.

95. What is the difference between Serializable and Externalizable interface?

Serializable is a marker interface but Externalizable is not a marker interface.

When we implement Serializable interface, the class is serialized automatically by default. We can override writeObject() and readObject()methods to control more complex object Serialization process.

In case of Externalizable, we use readExternal() and writeExternal() methods to give control to class for class's serialization process.

Serializable interface is based on recursive algorithm.

Serializable gives you two options. One option is to provide custom way of serialization, the other default way. In Externalizable, you have to always implement readExternal() and writeExternal() methods.

A public no-arg constructor is needed while using Externalizable interface.

In Serialization, we need to define serialVersionUID. If it is not explicitly defined it will be generated automatically based on all the fields, methods of the class.

Reflection

96. What is Reflection in Java?

Reflection is Java language's ability to inspect and dynamically call classes, methods, attributes etc. at Runtime. It helps in examining or modifying the Runtime behavior of a class at Runtime.

97. What are the uses of Reflection in Java?

Reflection is often used in Testing, Debugging and in Integrated Development Environment (IDE).

Reflection allows you to write programs that do not have to "know" everything at compile time. It makes programs more dynamic, since they can be tied together at runtime.

Many modern frameworks like Spring etc. use Reflection. Some modern languages like Python etc. also use Reflection.

JAVA API for XML Parsing (JAXP) also uses Reflection.

98. How can we access private method of a class from outside the class?

We can use Reflection to access private method of a class from outside the class. IN Java, we use getDeclaredMethod() to get instance of a private method. Then we mark this method accessible and finally invoke it.

In following sample code, we are accessing private method message() of class Foo by Reflection.

```
FileName: Foo.java public class Foo {

private void message(){System.out.println("hello java"); } }
```

```
FileName: FooMethodCall.java
import java.lang.reflect.Method;
public class FooMethodCall{
public static void main(String[] args)throws Exception{

Class c = Class.forName("Foo");
Object o= c.newInstance();
Method m =c.getDeclaredMethod("message", null); m.setAccessible(true);
m.invoke(o, null);

} }
```

99. How can we create an Object dynamically at Runtime in Java?

We can use Reflection to create an Object dynamically at Runtime in Java. We can use Class.newInstance() or Constructor.newInstance() methods for creating such Objects.

Garbage Collection

100.What is Garbage Collection in Java?

Java has an internal mechanism called Garbage collection to reclaim the memory of unused projects at run time.

Garbage collection is also known as automatic memory management.

101. Why Java provides Garbage Collector?

In Java, there are no pointers. Memory management and allocation is done by JVM. Since memory allocation is automated, after some time JVM may go low on memory. At that time, JVM has to free memory from unused objects. To help with the process of reclaiming memory, Java provides an automated process called Garbage Collector.

102.What is the purpose of gc() in Java?

Java provides two methods System.gc() and Runtime.gc() to request the JVM to run the garbage collection. By using these methods, programmers can explicitly send request for Garbage Collection. But JVM process can reject this request and wait for some time before running the GC.

103.How does Garbage Collection work in Java?

Java has an automated process called Garbage Collector for Memory Management. It is a daemon in JVM that monitors the memory usage and performs memory cleanup. Once JVM is low on memory, GC process finds the unused objects that are not referenced by other objects. These unused objects are cleaned up by Garbage Collector daemon in JVM.

104. When does an object become eligible for Garbage Collection in Java?

An object can be Garbage Collected by JVM, if it is not reachable. There are two cases for deciding eligibility of objects for Garbage Collection:

1. An Object/instance that cannot be reached by a live thread.

2. A set of circularly referenced instances that cannot be reached by any other instance outside that set.

105.Why do we use finalize() method in Java?

Java provides finalize() method to perform any cleanup before Garbage Collection. This method is in Object class, and it is invoked by JVM internally. Developers are free to implement this method for any custom cleanup in case of Garbage Collection.

If an Object is not Garbage Collected, then this method may not be called.

This method is never invoked more than once by JVM.

106.What are the different types of References in Java?

In Java, there are four types of references:

1. Strong Reference

2. Soft Reference

3. Weak Reference

4. Phantom Reference

107.How can we reference an unreferenced object again?

We can provide implementation in finalize() method to reference and unreferenced object. For an un-referenced object, finalize() method is called at the time of Garbage Collection. At this time, Object can pass its reference 'this' to finalize() method and revive itself.

108.What kind of process is the Garbage collector thread?

Garbage Collection is a Daemon process in JVM. It is an internal process that keep checking Memory usage and cleans up the memory.

109.What is the purpose of the Runtime class?

The purpose of the Runtime class is to provide access to the Java Runtime system. This class provides certain important methods like:

1. Runtime.freeMemory() – This method returns the value of free memory in JVM

2. Runtime.maxMemory() - This method returns the value of maximum memory that JVM can use.

3. Runtime.gc() – This method can invoke garbage collection.

110. How can we invoke an external process in Java?

Java provides the method Runtime.getRuntime().exec() to invoke an external process from JVM.

111. What are the uses of Runtime class?

Runtime class in Java provides following benefits:

1. It allows to read data via key board
2. It can use system properties and environment variables
3. It helps in running non-java programs from within a java

application.

Inner Classes

112.What is a Nested class?

In Java, a Nested class is a class declared inside another class. We can have more than one class declared inside a file.

113. How many types of Nested classes are in Java?

Java provides four types of Nested classes:

1. Member inner class

2. Local inner class

3. Anonymous inner class

4. Static nested class

114.Why do we use Nested Classes?

There are following reasons for using nested classes:

1. Logical Grouping: We can logically group classes in one place. If one class is useful to only one other class, then we put smaller class within the larger class and keep them in one file. This kind of nesting "helper classes" in a top- level class makes the package more streamlined.

2. Encapsulation: Nested classes increase encapsulation. Let say there are two top-level classes, Foo and Bar. Bar needs access to private members of Foo. We can hide class Bar within class Foo. In this way, private members of Foo can be accessed by class Bar. So class Foo remains encapsulated. Also, class Bar remains hidden from the outside world.

3. Code Clarity: Nested classed make the code more readable and well organized. Only Top-level classes are exposed. The helper classes are kept hidden and closer the code where it is used by a Top-level class.

115.What is the difference between a Nested class and an Inner class in Java?

An Inner class in Java is non-static class. It is a type of Nested class that is defined in another class but not qualified with a Static modifier. A Nested class is also a class can be Static Nested class or a non-Static Inner class.

An Inner class has access to other members of the enclosing class, even if they are declared private. A Static Nested class can not access the other members of the enclosing class.

116. What is a Nested interface?

A Nested interface is declared inside another interface or a top- level class. By default it is static.

A Nested interface is also known as Static interface.

117. How can we access the non-final local variable, inside a Local Inner class?

Java allows a Local Inner class to access only Constant local members. So we have to make the non-final local variable as final constant to access it inside a Local Inner class.

118.Can an Interface be defined in a Class?

Yes, we can define a Static Nested interface within a class. Only the enclosing class can access it.

119.Do we have to explicitly mark a Nested Interface public static?

A Nested Interface is implicitly public static. So the modifiers public and static are redundant in declaration.

120.Why do we use Static Nested interface in Java?

Only the enclosing class can access a Static Nested interface. Consider following code in which interface Xyz is enclosed in class Abc.

```
public class Abc {

public interface Xyz { void callback();

}

public static void registerCallback(Xyz xyz) {...} }

// Client Code Abc.registerCallback(new Abc.Xyz() {

public void callback() {...} });
```

Any code that cannot access Abc can not access interface Xyz also.

So the purpose of declaring an Inner interface is to restrict its access from outside world.

String

121.What is the meaning of Immutable in the context of String class in Java?

An Immutable object cannot be modified or changed in Java. String is an Immutable class in Java.

Once a String object is created, it cannot be changed. When we assign the String to a new value, a new object is created.

122.Why a String object is considered immutable in java?

Java language uses String for a variety of purposes. For this it has marked String Immutable.

There is a concept of String literal in Java.

Let say there are 2 String variables A and B that reference to a String object "TestData". All these variables refer to same String literal. If one reference variable A changes the value of the String literal from "TestData" to "RealData", then it will affect the other variable as well. Due to which String is considered Immutable. In this case, if one variable A changes the value to "RealData", then a new String literal with "RealData" is created and A will point to new String literal. While B will keep pointing to "TestData"

123.How many objects does following code create?

Code:
String s1="HelloWorld"; String s2=" HelloWorld "; String s3=" HelloWorld ";

The above code creates only one object. Since there is only one String Literal "HelloWorld" created, all the references point to same object.

124. How many ways are there in Java to create a String object?

Java provides two ways to create a String object. One is by using String Literal, the other is by using new operator.

125.How many objects does following code create?

Code:
String s = new String("HelloWorld");

The above code creates two objects. One object is created in String constant pool and the other is created on the heap in non-pool area.

126.What is String interning?

String interning refers to the concept of using only one copy of a distinct String value that is Immutable.

It provides the advantage of making String processing efficient in Time as well as Space complexity. But it introduces extra time in creation of String.

127.Why Java uses String literal concept?

Java uses String literal concept to make Java more efficient in memory. If same String already exists in String constant pool, it can be reused. This saves memory usage.

128. What is the basic difference between a String and StringBuffer object?

String is an immutable object. Its value cannot change after creation. StringBuffer is a mutable object. We can keep appending or modifying the contents of a StringBuffer in Java.

129.How will you create an immutable class in Java?

In Java, we can declare a class final to make it immutable. There are following detailed steps to make it Immutable:

1. Add final modifier to class to prevent it from getting extended

2. Add private modifier to all the fields to prevent direct access

3. Do not provide any setter methods for member variables

4. Add final modifier to all the mutable fields to assign value only once

5. Use Deep Copy to initialize all the fields by a constructor

6. In clone method, return a copy of object instead of the actual object reference

130.What is the use of toString() method in java ?

In Java, Object class has toString() method. This method can be used to return the String representation of an Object. When we print an object, Java implicitly calls toString() method.

Java provides a default implementation for toString() method. But we can override this method to return the format that we want to print.

131. Arrange the three classes String, StringBuffer and StringBuilder in the order of efficiency for String processing operations?

StringBuilder is the most efficient class. It does not have the overhead of Synchronization. StringBuffer is a Synchronized class. It has better performance than String but it is slower than StringBuilder. String is the slowest for any String processing operations, since it is leads to creation of new String literal with each modification.

So the decreasing order of efficiency is: StringBuilder, StringBuffer, String

Exception Handling

132.What is Exception Handling in Java?

Java provides Exception Handling mechanism to handle Runtime errors that occur in JVM. There are checked exceptions in a program that we expect to occur in certain situations.

Exception handling mechanism catches these checked exceptions and takes relevant actions.

133.In Java, what are the differences between a Checked and Unchecked?

Checked Exceptions extend Throwable class, but they do not extend RuntimeException or Error classes. UncheckedException extend RuntimeException class.

Checked Exceptions are checked at compile time in Java. Unchecked Exceptions happen at Runtime, so they are not checked at compile time.

IOException, SQLException etc. are examples of Checked Exceptions. NullPointerException, ArithmeticException etc. are examples of Unchecked Exceptions.

134. What is the base class for Error and Exception classes in Java?

Error as well as Exception class is derived from Throwable class in Java.

135.What is a finally block in Java?

Java provides a finally block with a try block. This is an optional block. But finally block is always executed after the execution of try block.

136.What is the use of finally block in Java?

As per Java specification, a finally block is always executed, whether an error occurs or not, whether an exception is handled or not. It helps in doing the cleanup like- Rollback Transaction, Close Connection, Close a file etc.

137.Can we create a finally block without creating a catch block?

Yes. A finally block can follow a try block or catch block. So we can defined a finally block just after a try block.

138.Do we have to always put a catch block after a try block?

Java does not enforce the rule to put a catch block after try block. We can write catch block or finally block after a try block.

Any exception that we want to catch is mentioned in catch block.

139.In what scenarios, a finally block will not be executed?

There are two main scenarios in which finally block is not executed:

1. Program exits by calling system.exit() call.

2. A fatal error causes JVM to crash.

140. Can we re-throw an Exception in Java?

Yes, Java allows to re-throw an Exception.

141.What is the difference between throw and throws in Java?

Java provides throw keyword to throw an exception from a method or a static block. Java provides throws keyword to mention the probable exception thrown by a method in its declaration.

We use throw to explicitly throw an exception. We used throws to declare an exception in method definition.

We cannot propagate checked exceptions with throw only. But checked exceptions can be propagated with throws keyword.

A throw call is followed by an instance. Class or Exception follows a throws keyword.

Call to throw occurs within a method. throws is just used with method signature.

We can throw only one exception at a time. But we can mention as many exceptions in throws clause.

142. What is the concept of Exception Propagation?

In Exception Propagation, uncaught exceptions are propagated in the call stack until stack becomes empty. This propagation is called Exception Propagation.

Let say an exception propagates from one method to another method. A() calls B(), which calls C(), which calls D(). And if D() throws an exception, the exception will propagate from D to C to B to A, unless one of the methods catches the exception.

143. When we override a method in a Child class, can we throw an additional Exception that is not thrown by the Parent class method?

Yes, Java allows us to throw additional Exception in a child class, but the additional exception should be an unchecked exception (RuntimeException).

Java Collection

144. What is the difference between Collection and Collections Framework in Java?

In Java, a Collection is an object that contains multiple elements of same type in a single unit. These multiple elements can be accessed through one Collection object.

In Java Collections Framework is a library that provides common architecture for creating, updating and accessing different types of collections. In Collections framework there are common methods that are frequently used by developers for working on a Collection object.

145. What are the main benefits of Collections Framework in Java?

Main benefits of Collections Framework in Java are as follows:

1. Reusability: Java Collections Framework provides common classes and utility methods than can be used with different types of collections. This promotes the reusability of the code. A developer does not have to re-invent the wheel by writing the same method again.

2. Quality: Using Java Collection Framework improves the program quality, since the code is already tested and used by thousands of developers.

3. Speed: Most of programmers report that their development speed increased since they can focus on core logic and use the generic collections provided by Java framework.

4. Maintenance: Since most of the Java Collections framework code is open source and API documents is widely available, it is easy to maintain the code written with the help of Java Collections framework. One developer can easily pick the code of previous developer.

146. What is the root interface of Collection hierarchy in Java?

The root interface of Collection hierarchy in Java is Collection interface.

But the Collection interface extends Iterable interface. Due to this some people consider Iterable interface as the root interface.

Iterable interface is present in java.lang package but Collection interface is present in java.util package. Oracle Java API docs mention that Collection interface is a member of the Java Collections framework.

Whereas, Iterable interface is not stated as a part of Java Collections framework in Java docs.

Due to this Collection interface is the root of Collections Framework.

147. What are the main differences between Collection and Collections?

Main differences between Collection and Collections are as follows:

1. Collection is an interface in Java. But Collections is a class in Java.

2. Collection is a base interface. Collections is a utility class in Java.

3. Collection defines methods that are used for data structures that contain the objects. Collections defines the methods that are used for operations like access, find etc. on a Collection.

148. What are the Thread-safe classes in Java Collections framework?

The Thread-safe classes in Java Collections framework are:

Stack
Properties
Vector
Hashtable
BlockingQueue ConcurrentMap ConcurrentNavigableMap

149. How will you efficiently remove elements while iterating a Collection?

The right way to remove elements from a collection while iterating is by using ListIterator.remove() method.

E.g.

```
ListIterator<Integer> iter = myList.iterator(); while(iter.hasNext()) {
itr.remove();
}
```

Some developers use following code to remove an element which is incorrect:

```
Iterator<Integer> iter = myList.iterator(); while(iter.hasNext()) {
itr.remove();
}
```

By doing so we get ConcurrentModificationException.

An iterator is first created to traverse the list. But at the same time the list is changed by remove() method.

In Java, it is not allowed for a thread to modify a collection while another thread is iterating it. ListIterator provides the capability of removing an object during traversal.

150.How will you convert a List into an array of integers like- int[]?

We can use ArrayUtils class in Apache Commons Lang library.

Sample code is:
```
int[]intArray = ArrayUtils.toPrimitive(myList.toArray(new Integer[0]));
```

If we use List.toArray(), it will convert List to Integer[]. Another option is:

```
int[] intArray = new int[myList.size()]; for (int i=0; i < myList.size(); i++) {

intArray [i] = myList.get(i); }
```

151.How will you convert an array of primitive integers int[] to a List collection?

We can use ArrayUtils in Apache Commons Lang library for this purpose.

Sample code is:
```
List intList = Arrays.asList(ArrayUtils.toObject(intArray));
```

The other option would be to use a for loop and explicitly adding integers to a List.

Sample code is:

```
int[]intArray = {10,20,30};
List<Integer> intList = new ArrayList<Integer>(); for (int i: intArray) {

intList.add(i); }
```

152.How will you run a filter on a Collection?

We can use CollectionUtils of Apache for this purpose. We will have to create a Predicate that will define the condition for our filter. Then we can apply this Predicate in filter() method.

Sample code is:

In this example we filter any names that are less than 5 characters long.

List<String> namesList = asList("Red", "Blue", "Green");

List<String> shortNamesList = new ArrayList<String>(); shortNamesList.addAll(namesList);

CollectionUtils.filter(shortNamesList, new Predicate(){ public boolean evaluate(Object input) {

return ((String) input).length() < 5; }

});

We can also use Google Guava library for this.

In Java 8, we can use Predicate to filter a Collection through Stream.

153.How will you convert a List to a Set?

There are two ways to convert a List to a Set in Java.

Option 1: Use HashSet

Set<Integer> mySet = new HashSet<Integer>(myList);

In this case we put a list into a HashSet. Internally hashCode() method is used to identify duplicate elements.

Option 2: Use TreeSet
In this case we use our own comparator to find duplicate objects.

Set<Integer> mySet = new TreeSet<Integer>(myComparator); mySet.addAll(myList);

154. How will you remove duplicate elements from an ArrayList?

The trick in this question is to use a collection that does not allow duplicate elements. So we use a Set for this purpose.

Option 1: Use Set
If ordering of elements is not important then we just put the elements of ArrayList in a HashSet and then add them back to the ArrayList.

Sample Code is:
ArrayList myList = // ArrayList with duplicate elements Set<Integer> mySet = new

HashSet<Integer>(myList); myList.clear();
myList.addAll(mySet);

Option 2: Use LinkedHashSet
If ordering of elements is important then we put the elements of ArrayList in a LinkedHashSet and then add them back to the ArrayList.

Sample Code is:
ArrayList myList = // ArrayList with duplicate elements Set<Integer> mySet = new LinkedHashSet<Integer>(myList); myList.clear();
myList.addAll(mySet);

155.How can you maintain a Collection with elements in Sorted order?

In Java, there are many ways to maintain a Collection with elements in sorted order.

Some collections like TreeSet store elements in the natural ordering. In case of natural ordering we have to implement Comparable interface for comparing the elements.

We can also maintain custom ordering by providing a custom Comparator to a Collection.

Another option is to use the utility method Collections.sort() to sort a List. This sorting gives nlog(n) order of performance. But if we have to use this method multiple times then it will be costly on performance.

Another option is to use a PriorityQueue that provides an ordered queue. The main difference between PriorityQueue and Collections.sort() is that PriorityQueue maintains a queue in Order all the time, but we can only retrieve head element from queue. We cannot access the elements of PriorityQueue in Random order.

We can use TreeSet to maintain sorted order of elements in collection if there are no duplicate elements in collection.

156.What are the differences between the two data structures: a Vector and an ArrayList?

An ArrayList is a newer class than a Vector. A Vector is considered a legacy class in Java. The differences are:

1. Synchronization: Vector is synchronized, but the ArrayList is not synchronized. So an ArrayList has faster operations than a Vector.

2. Data Growth: Internally both an ArrayList and Vector use an array to store data. When an ArrayList is almost full it increases its size by 50% of the array size. Whereas a Vector increases it by doubling the underlying array size.

157.What are the differences between Collection and Collections in Java?

Main differences between Collection and Collections are:

1. Type: Collection is an interface in Java. Collections is a class.

2. Features: Collection interface provides basic features of data structure to List, Set and Queue interfaces. Collections is a utility class to sort and synchronize collection elements. It has polymorphic algorithms to operate on collections.

3. Method Type: Most of the methods in Collection are at instance level. Collections class has mainly static methods that can work on an instance of Collection.

158. In which scenario, LinkedList is better than ArrayList in Java?

ArrayList is more popular than LinkedList in Java due to its ease of use and random access to elements feature.

But LinkedList is better in the scenario when we do not need random access to elements or there are a lot of insertion, deletion of elements.

159.What are the differences between a List and Set collection in Java?

Main differences between a List and a Set are:

1. Order: List collection is an ordered sequence of elements. A Set is just a distinct collection of elements that is unordered.

2. Positional Access: When we use a List, we can specify where exactly we want to insert an element. In a Set there is no order, so we can insert element anywhere without worrying about order.

3. Duplicate: In a List we can store duplicate elements. A Set can hold only unique elements.

160.What are the differences between a HashSet and TreeSet collection in Java?

Main differences between a HashSet and TreeSet are:

1. Ordering: In a HashSet elements are stored in a random order. In a TreeSet, elements are stored according to natural ordering.

2. Null Value Element: We can store null value object in a HashSet. A TreeSet does not allow to add a null value object.

3. Performance: HashSet performs basic operations like add(), remove(), contains(), size() etc in a constant size time. A TreeSet performs these operations at the order of log(n) time.

4. Speed: A HashSet is better than a TreeSet in performance for most of operations like add(), remove(), contains(), size() etc .

5. Internal Structure: a HashMap in Java internally backs a HashSet. A NavigableMap backs a TreeSet internally.

6. Features: A TreeSet has more features compared to a HashSet. It has methods like pollFirst(), pollLast(), first(), last(), ceiling(), lower() etc.

7. Element Comparison: A HashSet uses equals() method for comparison. A TreeSet uses compareTo() method for

comparison to maintain ordering of elements.

161. In Java, how will you decide when to use a List, Set or a Map collection?

1. If we want a Collection that does not store duplicate values, then we use a Set based collection.

2. If we want to frequently access elements operations based on an index value then we use a List based collection. E.g. ArrayList

3. If we want to maintain the insertion order of elements in a collection then we use a List based collection.

4. For fast search operation based on a key, value pair, we use a HashMap based collection.

5. If we want to maintain the elements in a sorted order, then we use a TreeSet based collection.

162.What are the differences between a HashMap and a Hashtable in Java?

Main differences between a HashMap and a Hashtable are:

1. Synchronization: HashMap is not a synchronized collection. If it is used in multi-thread environment, it may not provide thread safety. A Hashtable is a synchronized collection. Not more than one thread can access a Hashtable at a given moment of time. The thread that works on Hashtable acquires a lock on it and it makes other threads wait till its work is completed.

2. Null values: A HashMap allows only one null key and any number of null values. A Hashtable does not allow null keys and null values.

3. Ordering: A HashMap implementation by LinkedHashMap maintains the insertion order of elements. A TreeMap sorts the mappings based on the ascending order of keys. On the other hand, a Hashtable does not provide guarantee of any kind of order of elements. It does not maintain the mappings of key values in any specific order.

4. Legacy: Hashtable was not the initial part of collection framework in Java. It has been made a collection framework member, after being retrofitted to implement the Map interface. A HashMap implements Map interface and is a part of collection framework since the beginning.

5. Iterator: The Iterator of HashMap is a fail-fast and it throws ConcurrentModificationException if any other Thread modifies the map by inserting or removing any element except iterator’s own remove() method.

Enumerator of the Hashtable is not fail-fast.

163.What are the differences between a HashMap and a TreeMap?

Main differences between a HashMap and a TreeMap in Java are:

1. Order: A HashMap does not maintain any order of its keys. In a HashMap there is no guarantee that the element inserted first will be retrieved first.

2. In a TreeMap elements are stored according to natural ordering of elements. A TreeMap uses compareTo() method to store elements in a natural order.

3. Internal Implementation: A HashMap uses Hashing internally. A TreeMap internally uses Red-Black tree implementation.

4. Parent Interfaces: A HashMap implements Map interface. TreeMap implements NavigableMap interface.

5. Null values: A HashMap can store one null key and multiple null values. A TreeMap can not contain null key but it may contain multiple null values.

6. Performance: A HashMap gives constant time performance for operations like get() and put(). A TreeMap gives order of log(n) time performance for get() and put() methods.

7. Comparison: A HashMap uses equals() method to compare keys. A TreeMap uses compareTo() method for maintaining natural ordering.

8. Features: A TreeMap has more features than a HashMap. It has methods like pollFirstEntry() , poll-LastEntry() , tailMap() , firstKey() , lastKey() etc. that are not provided by a HashMap.

164. What are the differences between Comparable and Comparator?

Main differences between Comparable and Comparator are:

1. Type: Comparable<T> is an interface in Java where T is the type of objects that this object may be compared to.

2. Comparator<T> is also an interface where T is the type of objects that may be compared by this comparator.

3. Sorting: In Comparable, we can only create one sort sequence. In Comparator we can create multiple sort sequences.

4. Method Used: Comparator<T> interface in Java has method public int compare (Object o1, Object o2) that returns a negative integer, zero, or a positive integer when the object o1 is less than, equal to, or greater than the object o2. A Comparable<T> interface has method public int compareTo(Object o) that returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the object o.

5. Objects for Comparison: The Comparator compares two objects given to it as input. Comparable interface compares "this" reference with the object given as input.

6. Package location: Comparable interface in Java is defined in java.lang package. Comparator interface in Java is defined in java.util package.

165.In Java, what is the purpose of Properties file?

A Properties file in Java is a list of key-value pairs that can be parsed by java.util.Properties class.

Generally a Properties file has extension .properties e.g. myapp.properties.

Properties files are used for many purposes in all kinds of Java applications. Some of the uses are to store configuration, initial data, application options etc.
When we change the value of a key in a properties file, there is no need to recompile the Java application. So it provides benefit of changing values at runtime.

166.What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

167.How does hashCode() method work in Java?

Object class in Java has hashCode() method. This method returns a hash code value, which is an integer.

The hashCode() is a native method and its implementation is not pure Java.

Java doesn't generate hashCode(). However, Object generates a HashCode based on the memory address of the instance of the object.

If two objects are same then their hashCode() is also same.

168.Is it a good idea to use Generics in collections?

Yes. A collection is a group of elements put together in an order or based on a property. Often the type of element can vary. But the properties and behavior of a Collection remains same. Therefore it is good to create a Collection with Generics so that it is type-safe and it can be used with wide variety of elements.

169.What is the difference between Collections.emptyList() and creating new instance of Collection?

In both the approaches, we get an empty list. But Collections.emptyList() returns an Immutable list. We cannot add new elements to an Immutable empty list.

Collections.emptyList() works like Singleton pattern. It does not create a new instance of List. It reuses an existing empty list instance.

Therefore, Collections.emptylist() gives better performance if we need to get an emptyList multiple times.

170.How will you copy elements from a Source List to another list?

There are two options to copy a Source List to another list.
Option 1: Use ArrayList constructor
ArrayList<Integer> newList = new ArrayList<Integer>(sourceList);

Option2: UseCollection.copy()
To use Collections.copy() destination list should be of same or larger size than source list.

ArrayList<Integer> newList = new ArrayList<Integer> (sourceList.size());
Collections.copy(newList, sourceList);

Collections.copy() does not reallocate the capacity of destination List if it does not have enough space to contain all elements of source List. It throws IndexOutOfBoundsException.

The benefit of Collection.copy() is that it guarantees that the copy will happen in linear time. It is also good for the scenario when we want to reuse an array instead of allocating more memory in the constructor of ArrayList.

One limitation of Collections.copy() is that it can accept only List as source and destination parameters.

171. What are the Java Collection classes that implement List interface?

Java classes that implement List interface are:

AbstractList AbstractSequentialList ArrayList
AttributeList CopyOnWriteArrayList LinkedList
RoleList RoleUnresolvedList Stack
Vector

172.What are the Java Collection classes that implement Set interface?

Java classes that implement Set interface are:

AbstractSet ConcurrentSkipListSet CopyOnWriteArraySet EnumSet

HashSet JobStateReasons LinkedHashSet TreeSet

173.What is the difference between an Iterator and ListIterator in Java?

Iterator and ListIterator are two interfaces in Java to traverse data structures. The differences between these two are:

1. ListIterator can be used to traverse only a List. But Iterator can be used to traverse List, Set, and Queue etc.

2. An Iterator traverses the elements in one direction only. It just goes. ListIterator can traverse the elements in two directions i.e. backward as well as forward directions.

3. Iterator cannot provide us index of an element in the Data Structure. ListIterator provides us methods like nextIndex() and previousIndex() to get the index of an element during traversal.

4. Iterator does not allow us to add an element to collection while traversing it. It throws ConcurrentModificationException. ListIterator allows use to add an element at any point of time while traversing a list.

5. An existing element's value cannot be replaced by using Iterator. ListIterator provides the method set(e) to replace the value of last element returned by next() or previous() methods.

174. What is the difference between Iterator and Enumeration?

Both Iterator and Enumeration are interfaces in Java to access Data Structures. The main differences between these are:

1. Enumeration is an older interface. Iterator is a newer interface.

2. Enumeration can only traverse legacy collections. Iterator can traverse both legacy as well as newer collections.

3. Enumeration does not provide remove() method. So we cannot remove any element during traversal. Iterator provides remove() method.

4. Iterator is a fail-fast interface, it gives ConcurrentModificationException if any thread tries to modify an element in the collection being iterated. Enumeration is not fail-fast.

5. Method names in Iterator are shorter than in an Enumeration.

175.What is the difference between an ArrayList and a LinkedList data structure?

Main differences between ArrayList and LinkedList data structures are:

1. **Data Structure** : An ArrayList is an indexed based dynamic array. A LinkedList is a Doubly Linked List data structure.

2. **Inse rtion**: It is easier to insert new elements in a LinkedList, since there is no need to resize an array. Insertion in ArrayList is O(n), since it may require resizing of array and copying its contents to new array.

3. **Remove elements**: LinkedList has better performance in removal of elements than ArrayList.

4. **Me mory Usage** : LinkedList uses more memory than ArrayList, since it has to maintain links for next and previous nodes as well.

5. **Access**: LinkedList is slower in accessing an element, since we have to traverse the list one by one to access the right location.

176.What is the difference between a Set and a Map in Java?

Main differences between a Set and a Map in Java are:

1. **Duplicate Elements**: A Set does not allow inserting duplicate elements. A Map does not allow using duplicate keys, but it allows inserting duplicate values for unique keys.

2. **Null values**: A Set allows inserting maximum one null value. In a Map we can have single null key at most and any number of null values.

3. **Ordering**: A Set does not maintain any order of elements. Some of sub-classes of a Set can sort the elements in an order like LinkedHashSet. A Map does not maintain any order of its elements. Some of its sub-classes like TreeMap store elements of the map in ascending order of keys.

177.What is the use of a Dictionary class?

The Dictionary class in Java is used to store key-value pairs. Any non-null object can be used for key or value. But we cannot insert a null key or null object in Dictionary.

Dictionary class is deprecated now. So it should not be used in newer implementations.

178.What is the default size of load factor in a HashMap collection in Java?

Default value of load factor in a HashMap is 0.75.

179.What is the significance of load factor in a HashMap in Java?

A HashMap in Java has default initial capacity 16 and the load factor is 0.75f (i.e. 75% of current map size). The load factor of a HashMap is the level at which its capacity should be doubled.

For example, in a HashMap of capacity 16 and load factor .75. The capacity will become 32 when the HashMap is 75% full. Therefore, after storing the 12th key– value pair (16 * .75 = 12) into HashMap, its capacity becomes 32.

180.What are the major differences between a HashSet and a HashMap?

The main difference between a HashSet and a HashMap are:

1. **Base class**: A HashSet class implements the Set interface. Whereas a HashMap class implements the Map interface.

2. **Storage**: A HashSet is used to store distinct objects. A HashMap is used for storing key & value pairs, so that these can be retrieved by key later on.

3. **Duplicate Elements**: A HashSet does not allow storing duplicate elements. A HashMap also does not allow duplicate keys. But we can store duplicate values in a HashMap.

4. **Null Elements**: In a HashSet we can store a single null value. In a HashMap we can store single null key, but any number of null values.

5. **Element Type**: A HashSet contains only values of objects as its elements. Whereas a HashMap contains entries(key value pairs).

6. **Iteration**: By using an Iterator we can iterate a HashSet. But a HashMap has to be converted into Set for iteration.

181.What are the similarities between a HashSet and a HashMap in Java?

As the name suggests, HashSet and HashMap are Hashing based collections. Similarities between HashSet and HashMap are:

1. **Thread Safety**: Both HashMap and HashSet are not synchronized collections. Therefore they are not good for thread-safe operations. To make these thread-safe we need to explicitly use synchronized versions.

2. **Order of Elements**: None of these classes guarantee the order of elements. These are unordered collections.

3. **Internal Implementation**: A HashMap backs up a HashSet internally. So HashSet uses a HashMap for performing its operations.

4. **Performance**: Both of these collections provide constant time performance for basic operations such as insertion and removal of elements.

182. What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

183.How can we synchronize the elements of a List, a Set or a Map?

Sometimes we need to make collections Thread-safe for use in Multi-threading environment. In Java, Collections class provides useful static methods to make a List, Set or Map as synchronized collections. Some of these methods are:

static <T> Collection<T> synchronizedCollection(Collection<T> c)
Returns a synchronized (thread-safe) collection backed by the specified collection.

static <T> List<T> synchronizedList(List<T> list)
Returns a synchronized (thread-safe) list backed by the specified list.

static <K,V> Map<K,V>synchronizedMap(Map<K,V> m)
Returns a synchronized (thread-safe) map backed by the specified map.

static <T> Set<T> synchronizedSet(Set<T> s)
Returns a synchronized (thread-safe) set backed by the specified set.

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m) Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.

static<T>SortedSet<T> synchronizedSortedSet(SortedSet<T>s) Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

184. What is Hash Collision? How Java handles hash-collision in HashMap?

In a Hashing scenario, at times two different objects may have same HashCode but they may not be equal. Therefore, Java will face issue while storing the two different objects with same HashCode in a HashMap. This kind of situation is Hash Collision.

There are different techniques of resolving or avoiding Hash Collision. But in HashMap, Java simply replaces the Object at old Key with new Object in case of Hash Collision.

185. What are the Hash Collision resolution techniques?

To resolve a Hash Collision we can use one of the following techniques:

Separate Chaining with Linked List Separate Chaining with List Head Cells Open Addressing with Coalesced Hashing Open Addressing with Cuckoo Hashing Hopscotch Hashing
Robinhood Hashing

186.What is the difference between Queue and Stack data structures?

Queue is a FIFO data structure. FIFO stands for First In First Out. It means the element added first will be removed first from the queue. A real world example of Queue is a line for buying tickets at a station. The person entering first in the Queue is served first.

Stack is a LIFO data structure. LIFO stands for Last In First Out. The element that is added last is removed first from the collection. In a Stack elements are added or removed from the top of stack.

A real world example of Stack is back button in browser. We can go back one by one only and it works in the reverse order of adding webpages to history .

187.What is an Iterator in Java?

Iterator is an interface in Java to access the elements in a collection. It is in java.util package.
It provides methods to iterate over a Collection class in Java.

Iterator interface in Java is based on Iterator design pattern. By using an Iterator one can traverse a container of objects and can also access the objects in the container. A container of objects is a Collection class in Java.

188. What is the difference between Iterator and Enumeration in Java?

Main differences between Iterator and Enumeration in Java are:

1. **Version**: Enumeration interface is in Java since JDK 1.0. Iterator interface was introduced in Java 1.2.

2. **re move () me thod**: The main difference between Enumeration and Iterator interface is remove() method. Enumeration can just traverse a Collection object. If we use Enumeration, we cannot do any modifications to a Collection while traversing the collection. Iterator interface provides remove() method to remove an element while traversing the Collection. There is not remove() method in Enumeration interface.

3. **Method names**: Names of methods in Iterator interface are hasNext(), next(), remove(). Names of methods in Enumeration interface are hasMoreElements(), nextElement().

4. **Legacy Interface**: Enumeration is considered as a legacy interface. It is used to traverse legacy classes like Vector, Stack and HashTable. Iterator is a newer interface that is used to traverse almost all of the classes in Java Collections framework.

5. **Fail-fast vs. Fail-safe** : Iterator is based on fail-fast principle. It throws ConcurrentModificationException if a collection is modified during iteration over that collection. An Enumeration is based on fail-safe principle. It doesn't throw any exception if a collection is modified during traversal.

6. **Safety**: Since Iterator is fail-fast and does not allow modification of a collection by other threads, it is considered safer than Enumeration.

189. What is the design pattern used in the implementation of Enumeration in Java?

Enumeration is based on Iterator design pattern. Iterator design pattern provides a common interface with methods to traverse the collection of objects. It hides the underlying implementation details of the collection.

190.Which methods do we need to override to use an object as key in a HashMap?

If we want to use an object as a key in a HashMap in Java, then we have to make sure that it has the implementation of equals() and hashCode() methods.

191.How will you reverse a List in Java?

In Collections class, Java provides a method reverse(List list) that can be used to reverse a List.

E.g. Collections.reverse(myList);

192.How will you convert an array of String objects into a List?

Java provides Arrays class in java.util package. Arrays class has a method asList() that accepts an Array as input and returns a List as output.

public static <T> List<T> asList(T... a)

String[] myArray = {"George" , "Jack" , "Ryan"}; List myList = Arrays.asList(myArray);

193.What is the difference between peek(), poll() and remove() methods of Queue interface in java?

In a Java Queue, poll() and remove() methods can be used for removing the head object of Queue. The main difference arises in the case when Queue is empty().

If Queue is empty then poll() method returns null value. If Queue is empty then remove() method throws NoSuchElementException.

In a Java Queue, peek() method retrieves the head of Queue but it does not remove it. If queue is empty then peek() method returns null value.

194. What is the difference between Array and ArrayList in Java?

The main differences between Array and ArrayList in Java are:

1. **Size**: Array in Java is fixed in size. We cannot change the size of array after creating it. ArrayList is dynamic in size. When we add elements to an ArrayList, its capacity increases automatically.

2. **Performance**: In Java Array and ArrayList give different performance for different operations.

3. **add() or get():** Adding an element to or retrieving an element from an array or ArrayList object has similar performance. These are constant time operations.

4. **resize():** Automatic resize of ArrayList slows down the performance. ArrayList is internally backed by an Array. In resize() a temporary array is used to copy elements from old array to new array.

5. **Primitives**: Array can contain both primitive data types as well as objects. But ArrayList cannot contain primitive data types. It contains only objects.

6. **Iterator**: In an ArrayList we use an Iterator object to traverse the elements. We use for loop for iterating elements in an array.

7. **Type Safety**: Java helps in ensuring Type Safety of elements in an ArrayList by using Generics. An Array can

contain objects of same type of class. If we try to store a different data type object in an Array then it throws ArrayStoreException.

8. **Length**: Size of ArrayList can be obtained by using size() method. Every array object has length variable that is same as the length/size of the array.

9. **Adding elements**: In an ArrayList we can use add() method to add objects. In an Array assignment operator is used for adding elements.

10. Multi-dimension: An Array can be multi-dimensional. An ArrayList is always of single dimension.

195.How will you insert, delete and retrieve elements from a HashMap collection in Java?

We use following methods to insert, delete and retrieve elements in a HashMap.

1. **Retrieve**: We use get() method to retrieve elements from a HashMap.
   Value get(Object key)

2. **Insert**: We use put() method to insert a key value pair in a HashMap.
   Value put(Key k, Value v)

3. **Delete**: We use remove() method to delete key-value pair from the HashMap.
   Value remove(Object key)

196.What are the main differences between HashMap and ConcurrentHashMap in Java?

Main differences between HashMap and ConcurrentHashMap are:

1. **Synchronization**: A HashMap is not synchronized. But a ConcurrentHashMap is a synchronized object.

2. **Null Key**: A HashMap can have one null key and any number of null values. A ConcurrentHashMap cannot have null keys or null values.

3. **Multi-threading**: A ConcurrentHashMap works well in a multi-threading environment.

197.What is the increasing order of performance for following collection classes in Java?

The increasing order of performance is:

Hashtable Collections.SynchronizedMap ConcurrentHashMap HashMap

Hashtable has the worst performance and HashMap has the best performance.

198. Why does Map interface not extend Collection interface in Java?

A Map is a collection objects. But Map interface is not compatible with Collection interface in Java.

A Map requires key as well as a value. So it requires two parameters to add an element to a HashMap.

But Collection interface provides add(Object o) method with only one parameter.

Map collection has to provide methods like valueSet, keySet etc. These methods are specific to Map collection. Where as methods in Collection interface can be reused by a List, Set, Queue etc.

199.What are the different ways to iterate elements of a list in Java?

There are mainly two ways to iterate the elements of list in Java:

1. **Iterator**: We can get an Iterator for list and use it to iterate the objects of the list.

2. **For-each loop**: We can use for-each loop to traverse all the elements of a list.

200. What is CopyOnWriteArrayList? Howitis different from ArrayList in Java?

CopyOnWriteArrayList was introduced in Java 5 version. It is a thread-safe collection. It is similar to an ArrayList.

In CopyOnWriteArrayList, all mutative operations (add, set etc.) are implemented by making a fresh copy of the underlying array.

Iterator of CopyOnWriteArrayList is guaranteed to not throw ConcurrentModificationException. But Iterator also does not reflect any additions, removals that happened to list after the Iterator was created.

All elements including null are permitted in CopyOnWriteArrayList.

201.How remove() method is implemented in a HashMap?

Remove() method in HashMap uses logic similar to the one used in get() method. First we locate the correct bucket in HashMap for an entry. Then within that bucket we remove the element e. It is similar to removing a node from a single-linked list.

If e is the first element in the bucket we set the corresponding element of Hash to e.next. Else we set the next field of the element just before e to e.next.

202. What is BlockingQueue in Java Collections?

BlockingQueue was introduced in Java 1.5. It extends Queue interface in Java.

BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element. Also it supports the operations that wait for space to become available in the queue while storing an element.

Some of the features of BlockingQueue are:

It does not accept null elements.
Its main use is in producer-consumer problems. BlockingQueue implementation is thread-safe.
It can be used in inter-thread communications.
It does not support any kind of "close" or "shutdown" operation to indicate that no more items will be added.

203. How is TreeMap class implemented in Java?

Internally, a TreeMap class in Java uses Red-Black tree.

It is a NavigableMap. The map sorts the keys in natural order or it can use a Comparator supplied at the creation time.

The implementation of TreeMap is not synchronized in Java.

204. What is the difference between Fail-fast and Fail-safe iterator in Java?

Differences between Fail-fast and Fail-safe iterators are as follows:

Fail-fast iterator throws ConcurrentModificationException. But Fail-safe iterator does not throw this exception.

Fail-fast iterator does not clone the original collection. Fail-safe iterator creates a copy of the original collection of objects.

A Fail-fast iterator tries to immediately throw Exception when it encounters failure. A Fail-safe Iterator works on a copy of collection instead of original collection.

205. How does ConcurrentHashMap work in Java?

ConcurrentHashMap extends AbstractMap in Java. It was introduced in Java 1.5. It provides concurrency in a collection based on a HashMap.

All methods are thread-safe in ConcurrentHashMap.

Internally there is a Hashtable backing a ConcurrentHashMap. This Hashtable supports the concurrent methods for retrieval of data as well as updates on ConcurrentHashMap.

It has same functional specification as a Hashtable.

It also supports a set of sequential and bulk operations. These operations accept parallelismThreshold argument.

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

In a HashMap collection it is very important for a key object to implement hashCode() method and equals() method. If hashCode() method returns same hashcode for all key objects then the hash collision will be high in HashMap. Also with same hashcode, we will get same equals method that will make our HashMap inefficient.

The problem arises when HashMap treats both outputs same instead of different. It will overwrite the most recent key-value pair with the previous key-value pair.

So it is important to implement hashCode() and equals() methods correctly for an efficient HashMap collection.

207. What is the contract of hashCode() and equals() methods in Java?

Contract of hashCode() and equals() methods is as follows in Java:

If object1.equals(object2), then object1.hashCode() == object2.hashCode() should always be true. It means if two objects are equal then their hashCode should be same.

If object1.hashCode() == object2.hashCode() is true, it does not guarantee that object1.equals(object2). It means if two objects have same hashCode, then can still have different values so that may not be equal objects.

208. What is an EnumSet in Java?

Set: EnumSet is a specialized implementation of Set.

1. **Use**: It is mainly used with enum types.

2. **Single enum type**: All the elements in an EnumSet must come from a single enum type when the set is created.

3. **Bit vector**: Internally, EnumSet is represented as bit vector.

4. **Iterator**: The iterator of EnumSet traverses the elements in their natural order. (It is the order in which the enum constants are declared).

5. **Null**: In an EnumSet, null elements are not permitted. If we try to insert a null element it throws NullPointerException.

6. **Thread-safe**: EnumSet is not a synchronized collection. For use in multi-threading scenarios, EnumSet should be synchronized.

7. **Bit flags**: EnumSet is a very good alternative to int based "bit flags" implementation.

209. What are the main Concurrent Collection classes in Java?

Java 1.5 has provided new package java.util.concurrent. This package contains thread-safe collection classed. These collection classes can be modified while iterating. The iterator of these classes is fail-safe.

Main Concurrent Collection classes in Java 8 are:

ArrayBlockingQueue CopyOnWriteArrayList CopyOnWriteArraySet ConcurrentHashMap ConcurrentLinkedDeque ConcurrentLinkedQueue LinkedBlockingQueue LinkedBlockingDeque PriorityBlockingQueue

210.How will you convert a Collection to SynchronizedCollection in Java?

Java provides an easy method in java.utils.Collections class to create a ThreadSafe collection from a regular collection.

We can use the method synchronizedCollection() for this purpose. For any class of type T we can use following method:

static <T> Collection<T> synchronizedCollection(Collection<T> c)

211.How IdentityHashMap is different from a regular Map in Java?

IndentityHashMap in Java implements Map interface. But it is not a general purpose implementation. It violates the general contract of Map interface by a different implementation of equals() method.

In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2). (In a normal Map implementation (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).)

It implements the Map interface with a hash table, using reference- equality in place of object-equality when comparing keys (and values).

212.What is the main use of IdentityHashMap?

Main uses of IdentityHashMap are:

1. Topology Preservation: The typical use of IdentityHashMap class is topology-preserving object graph transformations, such as serialization or deep- copying. In such a scenario, a program must maintain a "node table" to keep track of all the object references that have already been processed.

2. The node table should not considered distinct objects as equal even if they happen to be equal.

3. Proxy objects: Another use of this class is to maintain proxy objects. A debugging program has to maintain a proxy object for each object in the program being debugged.

213.How can we improve the performance of IdentityHashMap?

IdentityHashMap class has one tuning parameter for performance improvement: expectedMaxSize.

This parameter is the maximum number of key-value mappings that the map is expected to hold.

We can use this parameter is used to determine the number of buckets initially in the hash table. The precise relationship between the expected maximum size and the number of buckets is unspecified.

If the number of key-value mappings exceeds the expected maximum size, the number of buckets is increased.

Increasing the number of buckets is also known as rehashing. Rehashing may be fairly expensive. So it is better to create identity hash maps with a sufficiently large expected maximum size.

But iteration over a Map collection requires time proportional to the number of buckets in the hash table. So iteration may take extra time due to large number of buckets.

Therefore the value of expectedMaxSize should be set in consideration with both of these aspects.

214. Is IdentityHashMap thread- safe?

The implementation of IdentityHashMap is not thread-safe, since its methods are not synchronized.

The iterators returned by the iterator method of IdentityHashMap are fail-fast. But the fail-fast behavior of an iterator cannot be guaranteed.

Since the Iterator is fail-fast, it throws ConcurrentModificationException.

215.What is a WeakHashMap in Java?

WeakHashMap is a class similar to IdentityHashMap. Internally, it is represented by a Hashtable.

It is not a synchronized class. We can make a WeakHashMap thread safe by using Collections.synchronizedMap() method.

An entry in WeakHashMap is automatically removed when it is no longer in ordinary use.

The presence of a mapping for a given key does not prevent the key from being discarded by the garbage collector.

WeakHashMap also permits null keys and null values.

216.How can you make a Collection class read Only in Java?

In Java, there are useful methods to make a Collection class read Only. We can make the Collection read Only by using one of the following methods:

Collections.unmodifiableMap(Map m) Collections.unmodifiableList(List l) Collections.unmodifiableSet(Set s) Collections.unmodifiableCollection(Collection c)

217.When is UnsupportedOperationException thrown in Java?

In a Java collection UnsupportedOperationException is thrown when the requested operation is not supported by the collection.

It is an unchecked exception that is thrown on optional operations.

If there is an optional add() or remove() methods in a read only collection, then this exception can be thrown.

218. Let say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of Customer class?

There are two ways to handle this scenario. We can use these options:
Comparable: Implement the Comparable interface for Customer class and compare customer objects by firstName attribute.

Comparator: Implement Comparator for comparing two Customer objects on the basis of firstName attribute. Then use this comparator object in sort method of Collections class.

219.What is the difference between Synchronized Collection and Concurrent Collection?

In Java 1.5 many Concurrent collection classes were added in SDK. These are ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc.

Java also provides utility methods to get a synchronized copy of collection like ArrayList, HashMap etc. by using Collections.synchronizedList(), Collections.synchronizedMap() methods.

The main difference is in performance. Concurrent collection classes have better performance than synchronized collection classes because they lock only a portion of the class to achieve concurrency and thread-safety.

220. What is the scenario to use ConcurrentHashMap in Java?

ConcurrentHashMap is more suited for scenarios where we have multiple reader threads and one writer thread. In this case map is locked only during the write operation.

If we have an equal number of reader and writer threads then ConcurrentHashMap performance is similar to a Hashtable or a synchronized HashMap.

221.How will you create an empty Map in Java?

There are two ways to create an empty Map in Java.

1. **Immutable**: If we want an immutable empty Map, we can use following code:
   myMap = Collections.emptyMap();

2. **Any map**: For all other scenarios, we can use following code by using new method:

myMap = new HashMap();

222. What is the difference between remove() method of Collection and remove() method of Iterator?

In Collection interface remove(Object o) method is used to remove objects from a Collection.

List interface also provides remove(int index) method to remove an object at a specific index.

These methods are used to remove an entry from Collection, while no thread is iterating over it.

When we are iterating over a Collection, then we have to remove() method of Iterator. This method removes current element from Iterator's point of view. If we use remove(0 method of Collection or List, then we will get ConcurrentModificationException.

Therefore, it is recommended to use remove() method of Iterator during the traversal of a Collection by an Iterator.

223. Between an Array and ArrayList, which one is the preferred collection for storing objects?

An ArrayList is backed up by array internally. There are many usability advantages of using an ArrayList over an array in Java.

Array has a fixed length at the time of creation. Once it is created we cannot change its length.

ArrayList is dynamic in size. Once it reaches a threshold, it automatically allocates a new array and copies contents of old array to new array.

Also ArrayList provides support of Generics. But Array does not support Generics.

E.g. If we store an Integer object in a String array at Runtime it will throw ArrayStoreException. Whereas, if we use ArrayList then as compile time we will get the error. This helps in preventing errors from happening at runtime.

If we know the size in advance and do not need re-sizing the collection then Array should be used in place of an ArrayList.

224. Is it possible to replace Hashtable with ConcurrentHashMap in Java?

Yes, a ConcurrentHashMap can be replaced with Hashtable in Java.

But it requires careful observation, since locking behavior of Hashtable is different than that of ConcurrentHashmap.

A Hashtable locks whole Map instead of a portion of Map. Compound operations like if(Hashtable.get(key) == null) put(key, value) work in Hashtable but not in ConcurrentHashMap.

In a ConcurrentHashMap we use putIfAbsent() method for such a scenario.

225. How CopyOnWriteArrayList class is different from ArrayList and Vector classes?

CopyOnWriteArrayList was introduced in Java 1.5. It implements List interface.

It provides better concurrent access methods than a Synchronized List.

In CopyOnWriteList, concurrency is achieved by copying ArrayList over each write and replace with original instead of locking.

CopyOnWriteArrayList also does not throw any ConcurrentModification Exception during Iteration.

It is a thread-safe list.

It is different from a Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

226. Why ListIterator has add() method but Iterator does not have?

ListIterator can iterate in the both directions of a Collection. It maintains two pointer for previous and next element. In ListIterator we can use add() method to add an element into the list immediately before the element returned by next() method.

So a subsequent call to next() method will not be affected. And the call to previous() method will return the newly added element.

In Iterator we can only traverse in one direction. So there is no purpose of add() method there.

227. Why do we sometime get ConcurrentModificationException during iteration?

When we remove an object by using remove() method of a Collection or List while an Iterator thread is traversing it, we get ConcurrentModificationException. If an Iterator detects any structural change in Collection it can throw ConcurrentModificationException.

228. How will you convert a Map to a List in Java?

In Java, a Map has three collection sets:

key set
value set key-value set

Each of these Sets can be converted to List by using a constructor. Sample code is as follows:

List keyList = new ArrayList(map.keySet()); List valueList = new ArrayList(map.values()); List entryList = new ArrayList(map.entrySet());

229. How can we create a Map with reverse view and lookup in Java?

In a Map we can lookup for a value by using a distinct key. In a Map with reverse view and lookup, even the values are distinct. So there is one to one mapping between keys and values and vice version.

If we enable this constraint on a Map then we can look up a key by its value. Such data structure is called bi-directional map.

There is no built data structure similar to reverse lookup Map in JDK.

But Apache Common Collections and Guava libraries provide implementation of bidirectional map. It is called BidiMap and BiMap. Both of these data structure enforce the constraint of one to one mapping between keys and values.

230. How will you create a shallow copy of a Map?

In Java, most implementations of Map interface provide a constructor to create copy of another map. But the copy method is not synchronized.

Therefore, when a thread is copying the map, another thread can modify it.

To prevent such a scenario, we should use Collections.synchronizedMap() method to first create a thread-safe map.

Another way of to create a shallow copy is by using clone() method. But it is not considered as a recommended approach.

231.Why we cannot create a generic array in Java?

Java does not allow creation of array with generics as elements.

In Java an array has to know the type information of its elements at runtime.

This information is used at runtime to throw ArrayStoreException if data type of an element to be inserted does not match the type of Array.

In case of Generics, the type information of a collection is erased at runtime by Type Erasure. Due to this array cannot use generics as elements.

232. What is a PriorityQueue in Java?

A PriorityQueue is data structure based on Queue. Unlike Queue, the elements on PriorityQueue are not returned in FIFO order.

A PriorityQueue maintains the natural order of its elements or it uses a Comparator provided at initialization.

It is an unbounded queue based on a priority heap.

PriorityQueue does not allow null values. We cannot add any object that does not provide natural ordering to PriorityQueue.

PriorityQueue in Java is not thread-safe.
It gives O(log n) time for enqueing and dequeing operations.

233. What are the important points to remember while using Java Collections Framework?

Some of the important points to remember while using Java Collections Framework are:

1. **Interfaces**: For Collections, we should write code with generic interfaces instead of concrete implementation. Due to this we maintain the flexibility of changing the implementation at a

later point of time.

2. **Generics**: We should use Generics for type-safety and to avoid ClassCastException at runtime.

3. **Collections**: It is recommended to use Collections utility class for algorithms and various other common methods for Collections.

4. **Right Type**: We have to choose the right type of Java collection based on our need. If size is fixed, we can use Array over ArrayList. If we do not want duplicate elements we use Set. If we need the ability to iterate the elements of a Map in the order of insertion then we use a TreeMap.

5. **Initial Size**: In some collection classes we can specify the initial size/capacity. Therefore we should have an estimate of number of elements in a Collection before deciding the right collection type. We can use it to avoid rehashing or resizing.

6. **Map**: We should use immutable classes provided by Java as key elements in a Map.

234. How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

To ensure that a method is not able to modify a Collection passed as an argument, we have to make the Collection read only.

We can make a read only collection by using Collections.unmodifiableCollection(Collection c) method.

This will make sure that any operation to change the collection will throw UnsupportedOperationException.

235. Can you explain how HashMap works in Java?

In Java, a HashMap works on the concept of hashing.

A HashMap in Java stores both key and value objects, in a bucket. It is stored as an Entry object that implements Map.Entry interface.

The key object used in a HashMap has to provide implementation for hashCode() and equals() methods.

When put() method is used to store a key-value pair, the HashMap implementation calls hashCode() method on Key object to calculate a hash that is used to find a bucket where Entry object will be stored.

When get() method is used to retrieve a value stored against a key object, we first calculate a hash of Key object. Then we use this hash to find the bucket in which that particular key is stored.

Once Key object's location is found, it may happen that more than one Key is stored in same location. So now we use equals() method to find the exact Key object. Once the exact Key object is found we use it to get Value object.

236. Can you explain how HashSet is implemented in Java?

Internally, a HashSet uses a HashMap to store the elements and to maintain the uniqueness of elements.

When we create a HashSet object, a corresponding HashMap object is also created.

When we insert an element in HashSet, it inserts it into corresponding HashMap.

237. What is a NavigableMap in Java?

As the name suggests, NavigableMap provides the capability to navigate the keys of a Map in Java. A NavigableMap extends SortedMap interface.

Some of the interesting methods of a NavigableMap are descendingKeySet(), descendingMap(), headMap() and tailMap().

238. What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

The descendingKeySet() method of NavigableMap returns a NavigableSet in which the elements are stored in reversed order as compared to the original key set.

The returned view is internally represented by the original KeySet of NavigableMap. Therefore any changes to the descending set also get reflected in the original set.

But it is not recommended to remove elements directly from the key set. We should use the Map.remove() method.

The descendingMap() method of NavigableMap returns a NavigableMap which is an inverse view of the original Map. The order of the elements in this view are in reverse order of the elements in original map. Any changes to this view are also reflected in the original map.

239. What is the advantage of NavigableMap over Map?

The main advantage of NavigableMap over Map is the Navigation capability.

It provides the capabilities of a Map, SortedMap and navigation in one collection.

It even returns the closest matches for given search targets.

Methods like lowerEntry, floorEntry, ceilingEntry, and higherEntry return Map.Entry objects associated with keys respectively less than, less than or equal, greater than or equal, and greater than a given key.

Methods like lowerKey, floorKey, ceilingKey, and higherKey return only the associated keys. All of these methods are designed for locating, not traversing entries.

240. What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

The headMap() method returns a view of the original NavigableMap that contains the elements that are less than a given element.

NavigableMap original = new TreeMap(); original.put("1", "1");
original.put("2", "2");
original.put("3", "3");

//this headmap1 will contain elements "1" and "2" SortedMap headmap1 = original.headMap("3");

//this headmap2 will contain elements "1", "2", and "3" because "inclusive"=true
NavigableMap headmap2 = original.headMap("3", true);

The tailMap() method works similar to headMap() method, but it returns all elements that are higher than the given input element.

The subMap() method accepts two parameters demarcating the boundaries of the view map to return.

All the three methods return a subset of the original map in a view form.

241. How will you sort objects by Natural order in a Java List?

We can use Collections.sort method to sort the elements of a List in natural order. To use this method, we have to make sure that element objects implement compareTo() method.

We can also use a Comparator to define the natural ordering for elements of a List. Then we can use this Custom Comparator in sort method of Collections class.

242. How can we get a Stream from a List in Java?

From Java 8 onwards it is a very easy to get a Stream from a List. We can just use stream() method to get a stream from a list of elements.

243. Can we get a Map from a Stream in Java?

Yes, we can create a Map from the elements of a Stream. We can use map() method to get a Map.

E.g. items.stream()
.map(item -> item.toLowerCase())

In this example we are creating a map with each item object mapped to its LowerCase equivalent.

This is also used in Map-Reduce implementation on a Stream.

244. What are the popular implementations of Deque in Java?

The two most popular implementation of Deque interface in Java are:

1. **ArrayDeque**: It is a resizable array implementation of Deque. The capacity of ArrayDeque can increase based on the need of the program. It is not thread safe implementation. Also the iterator on ArrayDeque is fail- fast.

2. **LinkedList**: This is another popular implementation of Deque interface in Java. It is also not synchronized, so it is not thread-safe. It mainly provides functionality of a doubly linked list.

Multi-threading

245. What is a Thread in Java?

A thread in Java is a lightweight process that runs within another process or thread.

It is an independent path of execution in an application. JVM gives each thread its own method-call stack.

When we start JVM, Java starts one thread. This thread calls the main method of the class passed in argument to java call.

246. What is the priority of a Thread and how it is used in scheduling?

In Java, every Thread has a priority. This priority is specified as a number between 1 to 10.

Scheduler in Java schedules different threads based on the priority of a thread. It is also known as preemptive scheduling.

The thread with higher priority gets preference in execution over a thread with lower priority.

247. What is the default priority of a thread in Java?

In Java, a new thread gets the same priority as the priority of the parent thread that creates it.

Default priority of a thread is 5 (NORM_PRIORITY).

248. What are the three different priorities that can be set on a Thread in Java?

We can set following three priorities on a Thread object in Java:

1. MIN_PRIORITY: This is the minimum priority that a thread can have.

2. NORM_PRIORITY: This is the default priority that is assigned to a thread.

3. MAX_PRIORITY: This is the maximum priority that a thread can have.

Default priority of a thread is 5 NORM_PRIORITY. The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

249. What is the purpose of join() method in Thread class?

In Java, Thread Scheduler controls thread scheduling. But we can use join() method on a thread to make current thread to wait for another thread to finish.

When we use join(), the current thread stops executing. It wait for the thread on which join() is called to finish.

This makes sure that current thread will continue only after the thread it joined finished running. Consider following example:

```
Public class ThreadJoin {
Thread importantThread = new Thread(

new Runnable() { public void run () {

//do something }

} );

Thread currentThread = new Thread( new Runnable() {

public void run () { //do something

} }

);
importantThread.start(); // Line 1 importantThread.join(); // Line 2 currentThread.start(); // Line 3

}
```

In the above example, main thread is executing. On Line 1, a new thread called importantThread is ready to run. But at Line 2, main thread joins the importantThread. Now it lets importantTread to finish and then it moves to Line 3. So currentThread at Line 3 will not start till the importantThread has finished.

250. What is the fundamental difference between wait() and sleep() methods?

The main difference between wait() and sleep() is that wait is an Object level method, whereas sleep() is a static method in Thread class. A waiting thread can be woken up by another thread by calling notify() on the monitor which is being waited on. But a sleeping thread cannot be woken up.

A wait() and notify() has to happen within the same block that is synchronized on the monitor object.

When we call wait() the current thread releases the monitor and goes to waiting state. Then another thread calls notify() to wake it up.

In case of sleep() current thread does not release the monitor or locks. It just sleeps for some pre-defined time period.

251.Is it possible to call run() method instead of start() on a thread in Java?

Yes. We can call run() method of a thread. But it does not work as a separate thread. It will just work as a normal object in main thread and there will not be context switching between the threads.

252. How Multi-threading works in Java?

Java provides support for Multithreading. In a Multithreading environment, one process can execute multiple threads in parallel at the same time.

In Java, you can create process and then create multiple threads from that process. Each process can execute in parallel to perform independent tasks.

Java provides methods like- start(), notify(), wait(), sleep() etc. to maintain a multi-threading environment.

253. What are the advantages of Multithreading?

Main advantages of Multithreading are:

1. Improved performance: We can improve performance of a job by Multi-threading.

2. Simultaneous access to Multiple Applications: We can access multiple applications from a process by doing multithreading

3. Reduced number of Servers required: With Multi- threading we need lesser number of servers, since one process can spawn multiple threads.

4. Simplified Coding: In certain scenarios, it is easier to code multiple threads than managing it from same thread.

254. What are the disadvantages of Multithreading?

There are certain downsides to Multithreading. These are:

1. Difficult to Debug: Multithreading code is difficult to debug in case of an issue.

2. Difficult to manage concurrency: Due to multiple threads, we may experience different kinds of issues.

3. Difficulty of porting code: It is difficult to convert existing single threaded code into multi-threading code.

4. Deadlocks: In case of multi-threading we can experience deadlocks in threads that are waiting for same resource.

255. What is a Thread in Java?

In Java, a thread is a lightweight process that runs within another process or thread. It is an independent path of execution in an application. Each thread runs in a separate stack frame.

By default Java starts one thread when the main method of a class is called.

256. What is a Thread's priority and how it is used in scheduling?

In Java, every Thread has a priority. This priority is specified as an integer value. The priority value is used in scheduling to pick up the thread with higher priority for execution. The threads with higher priority get more preference in execution than the threads with lower priority.

The task scheduler schedules the higher priority threads first, followed by the lower priority threads.

257. What are the differences between Pre-emptive Scheduling Scheduler and Time Slicing Scheduler?

In Pre-emptive scheduling, the highest priority task will keep getting time to execute until it goes to waiting state or dead state or a task with higher priority comes into queue for scheduling.

In Time slicing scheduling, every task gets a predefined slice of time for execution, and then it goes to the pool of tasks ready for execution. The scheduler picks up the next task for execution, based on priority and various other factors.

258. Is it possible to call run() method instead of start() on a thread in Java ?

Yes. We can call run() method of a thread. But it does not work as a separate thread. It will just work as a normal object in main thread and there will not be context-switching between the threads.

259. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread to daemon thread once it has already started.

If we do it by calling setDaemon(), it will throw IllegalThreadStateException

260. Can we start a thread two times in Java?

No. We can call start() method only once on a thread in Java. If we call it twice, it will give us exception.

261.In what scenarios can we interrupt a thread?

We can interrupt a thread if we want to wake it up from the sleep or wait state.

262. In Java, is it possible to lock an object for exclusive use by a thread?

Yes. We can use synchronized block to lock an object. The locked object is inaccessible to any other thread. Only the thread that has locked it can access it.

263. How notify() method is different from notifyAll() method?

In Java, notify() method is used to unblock a specific thread that is in waiting stated. Whereas, notifyAll() method is used to unblock all the threads that are in waiting state.

264. What is a daemon thread in Java?

A daemon thread in Java is a low priority thread that does not prevent the JVM from exiting when the program finishes. The thread keeps running. Garbage Collection is an example of daemon thread.

265. How can we make a regular thread Daemon thread in Java?

We can call setDaemon(boolean) method to change a thread to daemon thread before the thread starts.

266. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread to daemon thread once it has already started. If we do it by calling setDaemon(), it will throw IllegalThreadStateException

267. Can we start a thread two times in Java?

No. We can call start() method only once on a thread in Java. If we call it twice, it will give us exception.

268. What is a Shutdown hook in Java?

The shutdown hook is a thread that is invoked implicitly by JVM just before the shut down. It can be used to clean up unused resources etc.

We can use java.lang.Runtime.addShutdownHook(Thread hook) method to register a new virtual-machine shutdown hook.

269. What is synchronization in Java?

The concept of Synchronization in Java is used in Multi-threading programming.

It is a feature in Java that helps in controlling the access of multiple threads to a shared resource.

It is used to prevent Deadlock between multiple threads.

270. What is the purpose of Synchronized block in Java?

Synchronized block has many uses in Java multi-threading environment. Some of the uses are:

It can prevent thread interference
It is also used to avoid memory inconsistency issues

In general, scope of synchronized block is smaller than the scope of a method.

271.What is static synchronization?

We can make a static method as synchronized in Java. Adding synchronized keyword to a static method can do this.

In static synchronization, the lock is on class not on object.

272. What is a Deadlock situation?

A Deadlock is a situation in which two or more threads are waiting on each other to release a resource. Each thread is waiting for a resource that is held by the other waiting thread.

At times there is a circular wait when more than two threads are waiting on each other's resources.

273. What is the meaning of concurrency?

Concurrency is the ability of a program to execute several programs simultaneously. This is achieved by distributing computations over multiple CPU cores of a machine or even over different machines within the same network.

It can increase the speed of execution of the overall program in multi-processor or multi-core system.

274. What is the main difference between process and thread?

As such both process and thread are independent sequences of execution.

The main difference is that a thread runs in a shared memory space, where as a process runs in its own memory space.

A process runs the execution in an environment provided by the operating system. A process has its own set of private resources (e.g. memory, open files, etc.).

A thread lives within a process and shares the resources like- memory, open files etc. with the other threads of the same process.

This ability to share resources between different threads makes thread more suitable for tasks where performance is a significant factor.

275. What is a process and thread in the context of Java?

In Java, a process refers to the running of Java Virtual Machine (JVM). But a thread lives within a JVM and it can be created or stopped by the Java application at runtime.

276. What is a Scheduler?

A scheduler is a program that is the implementation of a scheduling algorithm to manage access of processes and threads to limited resource like CPU or an I/O channel.

The goal of most scheduling algorithms is to provide load balancing for the available processes/threads and to guarantee that each process/thread will get a reasonable time frame to access the requested resource exclusively.

277. What is the minimum number of Threads in a Java program?

In a JVM, each Java program is executed within the main process that starts with java.exe. Therefore each Java application has at least one thread.

278. What are the properties of a Java thread?

Each Java thread has following properties:

1. **Identifier**: An identifier of type long that is unique within the JVM

2. **Name**: A name of type String

3. **Priority**: Priority of type int

4. **State**: A state of type java.lang.Thread.State

5. **Group**: A thread group the thread belongs to

279. What are the different states of a Thread in Java?

Following are the different states of a Thread in Java:

1. **New**: In the New state the thread has not yet.

2. **Runnable**: A thread executing in the JVM is in Runnable state.

3. **Blocked**: A thread waiting for a monitor lock is in Blocked state.

4. **Waiting**: A thread waiting indefinitely for another thread to perform a particular action is in Waiting state.

5. **Timed_waiting**: A thread waiting for another thread to perform an action for up to a specified waiting time is in Timed_waiting state.

6. **Terminated**: A thread that has exited is in Terminated state.

280. How will you set the priority of a thread in Java?

The priority of a thread in Java can be set by using setPriority(int priority) method.

We can use constant Thread.MAX_PRIORITY to set the maximum priority of a thread.

We can use constant Thread.MIN_PRIORITY to set the minimum priority of a thread.

Or we can use constant Thread.NORM_PRIORITY to set the default priority of a thread.

281. What is the purpose of Thread Groups in Java?

In Java, every thread belongs to a group of threads.

The JDK class java.lang.ThreadGroup provides methods to handle a whole group of Threads.

With the help of these methods we can interrupt all threads of a group or set the maximum priority of all threads of a group.

So a thread group is used for taking collective actions on a group of threads.

282. Why we should not stop a thread by calling its stop() method?

The stop() method in Thread class is a deprecated method. Its use is not recommended.

When we call stop() method, the thread unlocks all monitors that it has acquired. If any locked object was in an inconsistent state, this state gets visible to all other threads.

It can cause unexpected behavior when other threads work on this inconsistent object.

So calling stop() method to stop a thread is not advisable.

283. How will you create a Thread in Java?

There are two main ways to create a thread in Java.

1. **Extend Thread class**: We can extend java.lang.Thread class and implement run() method. On calling start() method it will start a new thread.

2. **Imple me nt Runnable inte rface** : We can implement java.lang.Runnable interface and pass the implemented object to the constructor of java.lang.Thread class. On calling start() it will start a new thread.

284. How can we stop a thread in the middle of execution in Java?

We can use a volatile variable as an indicator to stop the thread.

We can create a volatile reference pointing to the current thread. This reference can be set to null by other threads to flag that the current thread should stop execution.

In following example threadStopper is the volatile reference that can be set as null in stopThread() method by other threads.

Sample code is as follows:
public static class MyThread extends Thread {

private volatile Thread threadStopper;

```
public void start() {
threadStopper = new Thread(this); threadStopper.start();

}

public void stopThread() { threadStopper = null;

}

public void run() {
Thread currThread = Thread.currentThread(); while(currThread == threadStopper) {

try{ Thread.sleep(100);

} catch (InterruptedException e) {

} }

} }
```

285. How do you access the current thread in a Java program?

We can access the current thread in Java by calling the static method currentThread() of java.lang.Thread class.

Sample code is as follows: public class MyThread {

```
public static void main(String[] args) { // Get ID of Current Thread

long id = Thread.currentThread().getId();

// Get Name of Current Thread
String name = Thread.currentThread().getName();

} }
```

286. What is Busy waiting in Multi- threading?

Busy waiting is also known as busy-looping or spinning. It is a multi-threading technique in which a process repeatedly checks if a condition is true.

For example, a process can keep checking if any keyboard input is available.

In general, busy waiting is considered as Anti-pattern that wastes processor time, so it should be avoided.

Sample code for busy waiting is as follows:

```
Thread thread = new Thread(new Runnable() { @Override
public void run() {

long timeToStop = System.currentTimeMillis() + 1000; long currentTime =
System.currentTimeMillis();

// Busy waiting
while (timeToStop > currentTime) { currentTime = System.currentTimeMillis(); }

} });
```

287. How can we prevent busy

waiting in Java?

There is a simple way to prevent busy-waiting in Java. We can just put the current thread to sleep for given amount of time.

It can be done by calling sleep() method of java.lang.Thread class. We can pass the number of milliseconds to sleep() method as an argument.

288. Can we use Thread.sleep() method for real-time processing in Java?

Java does not guarantee that Thread.sleep() will cause the thread to sleep for exactly N number of milliseconds. Sometime the thread can sleep for than N number of milliseconds.

In real-time processing we need precise time period for which a thread should run or sleep.

Therefore the invocation of Thread.sleep() method is not recommended for use in real-time processing.

289. Can we wake up a thread that has been put to sleep by using Thread.sleep() method?

We can use interrupt() method of java.lang.Thread class to interrupt a thread that is in sleep state. It will get InterruptedException to wake up from the sleep.
Sample code is as follows:

```
public class ThreadInterrupt implements Runnable { public void run() {
try{

Thread.sleep(Long.MAX_VALUE); } catch (InterruptedException e) {

SOP("Interrupted by exception!"); }

}
public static void main(String[] args) throws InterruptedException

{
Thread myThread = new Thread(new ThreadInterrupt(),
```

“myThread");
myThread.start();
SOP(“Sleeping in main thread for 10 seconds”);

Thread.sleep(10000); SOP(“Interrupting myThread");

myThread.interrupt(); }

}

290. What are the two ways to check if a Thread has been interrupted?

These are the two ways to check for thread interruption:

1. In Java, a Thread can call Thread.interrupted() method to check if it has been interrupted or not.

2. The other option is to call isInterrupted() method of Thread class to check if it has been interrupted or not.

291.How can we make sure that Parent thread waits for termination of Child thread?

We can use join() method for this purpose. On calling join() method, current thread waits for the child thread to which it joins to finish.

Sample code is as follows:

Thread myThread = new Thread(new Runnable() { public void run() {

} });

myThread.start();
// Join on myThread myThread.join();

292. How will you handle InterruptedException in Java?

In Java we can get InterruptedException from sleep() or join() methods. Throwing InterruptedException is way to inform that another thread has interrupted this thread.

In general, the purpose of Interrupt is to ask current thread to stop its current execution and finish unexpectedly.

Therefore ignoring this exception by catching it and only logging it to the console or some log file is not the recommended approach.

The run() method of the Runnable interface does not allow that throwing any exceptions. So we cannot re-throw InterruptedException.

Therefore the correct way to handle this exception is that run() method should check and handle this exception by itself and take appropriate action.

293. Which intrinsic lock is acquired by a synchronized method in Java?

When we mark a method as synchronized and then call this method, then this method will first acquire the intrinsic lock of the object in which that method is mentioned.

Once the synchronized method returns, it releases the lock.

In case the synchronized method throws an exception, the intrinsic lock will be released.

Sample code equivalent to a synchronized method is:

```
public void myMethod() { synchronized(this) {
}

}
```

294. Can we mark a constructor as synchronized in Java?

No. We cannot mark a constructor as synchronized. This will lead to compiler error.

The reasoning behind this is that, in this case, only the constructing thread would have access to the object being constructed.

295. Can we use primitive values for intrinsic locks?

No. Java does not allow primitive values to be used for intrinsic locks.

296. Do we have re-entrant property in intrinsic locks?

Yes. An intrinsic lock can be accessed by the same thread multiple times. So an Intrinsic lock is re-entrant.

If it is not allowed then the code that acquires a lock would have to avoid acquiring the lock that it has already acquired.

297. What is an atomic operation?

An atomic operation is an operation that completes in a single step relative to other threads.

An Atomic operation is either executed completely or not at all. There is no halfway mark in Atomic operation.

298. Can we consider the statement i++ as an atomic operation in Java?

No. The statement i++ is not an Atomic operation. It has more than one operation.

First JVM loads the current value of i in memory. Then it increments it. Finally it stores the new value back into variable i.

The current thread that executes this operation may be interrupted between any of the above-mentioned three steps. Therefore it is not an atomic operation.

299. What are the Atomic operations in Java?

Java language provides some basic Atomic operations. These operations can be used to make sure that concurrent threads always see the same value.

Some of these Atomic operations are:

1. 2.

3. 4.

Read operations on reference variables and primitive variables (except long and double)

Write operations on reference variables and primitive variables (except long and double)

Read operations on all variables declared as volatile Write operations on all variables declared as volatile

300. Can you check if following code is thread-safe?

```
public class SingletonDoubleCheck {
private SingletonDoubleCheck instance = null;

public SingletonDoubleCheck getInstance() {
if (instance == null) {
synchronized (SingletonDoubleCheck.class) { if (instance == null) {

instance = new SingletonDoubleCheck(); }
}
}

return instance; }

}
```

The above-mentioned code is for creating a Singleton class. But this code is not thread-safe.

In this we check the value of instance second time in the synchronized block. But the JIT compiler can rearrange the Bytecode in such a way that the reference to SingletonDoubleCheck instance will be set before the execution of constructor.

Due to this the method getInstance() will return an object that may not have been initialized properly.

We can use the keyword volatile for instance to make this thread- safe code.

Any variables that is marked as volatile will be visible to other threads only after the completion of the constructor of the object.

301.What are the minimum requirements for a Deadlock situation in a program?

For a deadlock to occur following are the minimum requirements:

1. **Mutual exclusion**: There has to be a resource that can be accessed by only one thread at any point of time.

2. **Resource holding**: One thread locks one resource and holds it, and at the same time it tries to acquire lock on another mutually exclusive resource.

3. **No preemption**: There is no pre-emption mechanism by which resource held by a thread can be freed after a specific period of time.

4. **Circular wait**: There can be a scenario in which two or more threads lock one resource each and they wait for each other's resource to get free. This causes circular wait among threads for same set of resources.

302. How can we prevent a Deadlock?

To prevent a Deadlock from occurring at least one requirement for a deadlock has to be removed:

1. **Mutual exclusion**: We can use optimistic locking to prevent mutual exclusion among resources.

2. **Resource holding**: A thread has to release all its exclusive locks if it does not succeed in acquiring all exclusive locks for resources required.

3. **No pre e mption**: We can use timeout period for an exclusive lock to get free after a given amount of time.

4. **Circular wait**: We can check and ensure that circular wait does not occur, when all exclusive locks have been acquired by all the threads in the same sequence.

303. How can we detect a Deadlock situation?

We can use ThreadMXBean.findDeadlockedThreads() method to detect deadlocks in Java program. This bean comes with JDK:

Sample code is as follows:

```
ThreadMXBean bean = ManagementFactory.getThreadMXBean(); long[] threadIds = bean.findDead-
lockedThreads(); // It will return null for no deadlock
if (threadIds != null) {

ThreadInfo[] infos = bean.getThreadInfo(threadIds);

for (ThreadInfo info : infos) {
StackTraceElement[] stack = info.getStackTrace(); // Log or store stack trace information.

} }
```

304. What is a Livelock?

Livelock is a scenario in which two or more block each other by responding to an action caused by another thread.

In a deadlock situation two or more threads wait in one specific state.

In a Livelock scenario, two more threads change their state in such a way that it prevents progress on their regular work.

E.g. Consider scenario in which two threads try to acquire two locks. They release a lock that they have acquired, when they cannot acquire the second lock.

In a Livelock situation, both threads concurrently try to acquire the locks. Only one thread would succeed, the second thread may succeed in acquiring the second lock.

Now both threads hold two different locks. And both threads want to have both locks. So they release their lock and try again from the beginning. This situation keeps repeating multiple times..

305. What is Thread starvation?

In a priority based scheduling, Threads with lower priority get lesser time for execution than higher priority threads.

If a lower priority thread performs a long running computation, it may happen that this thread does not get enough time to finish its computations just in time. In such a scenario, the tread with lower priority would starve. It will remain away from the threads with higher priority.

306. How can a synchronized block cause Thread starvation in Java?

It is not defined for synchronization that which thread will enter a synchronized block. It may happen that if many threads are waiting for the entry to a synchronized block, some threads may have to wait longer than other threads.

Hence these threads with lower priority will not get enough time to finish their work in time.

307. What is a Race condition?

A race condition is an unwanted situation in which a program attempts to perform two or more operations at the same time, but because of the logic of the program, the operations have to be performed in proper sequence to run the program correctly.

Since it is an undesirable behavior, it is considered as a bug in code.

Most of the time race condition occurs in “check then act” scenario. Both threads check and act on same value. But one of the threads acts in between check and act. See this example to understand race condition.

if (x == 3) // Check {

y = x * 5; // Act

// If another thread changes x
// between "if (x == 3)” and "y = x * 5”, // then y will not be equal to 15.

}

308. What is a Fair lock in multi- threading?

In Java there is a class ReentrantLock that is used for implementing Fair lock. This class accepts an optional parameter fairness. When fairness is set to true, the RenentrantLock will give access to the longest waiting thread.

The most popular use of Fair lock is in avoiding thread starvation. Since longest waiting threads are always given priority in case of contention, no thread can starve.

Downside of Fair lock is the low throughput of the program. Since low priority or slow threads are getting locks multiple time, it leads to slower execution of a program.

The only exception to a Fair lock is tryLock() method of ReentrantLock. This method does not honor the value of fairness parameter.

309. Which two methods of Object class can be used to implement a Producer Consumer scenario?

In a Producer Consumer scenario, one thread is a Producer and another thread is a Consumer.

For this scenario to start working, a Consumer has to know when the Producer has produced. In Object class, there is a wait() method. A Consumer calls wait method to wait on Producer. The Producer used notify() method of Object class to inform Consumer that it has produced.

In this way the processor time between produce and consume operations is freed due to the use of wait() and notify() methods.

310.How JVM determines which thread should wake up on notify()?

If multiple threads are waiting on an object's monitor, JVM awakens one of them. As per Java specification the choice of this thread is arbitrary and it is at the discretion of the implementation. So there is no guarantee of rule that a specific thread will be awakened by JVM on notify() method call.

311. Check if following code is thread- safe for retrieving an integer value from a Queue?

```
public class QueueCheck { Queue queue;

public Integer getNextInt() { Integer retVal = null; synchronized (queue) {

try{
while (queue.isEmpty()) { queue.wait();
}
} catch (InterruptedException e) { e.printStackTrace();
}

}
synchronized (queue) {

retVal = queue.poll();
if (retVal == null) {
System.err.println("retVal is null");
throw new IllegalStateException(); }

}

return retVal; }

}
```

In the above code Queue is used as object monitor to handle concurrency issues. But it may not behave correctly in a multi- threading scenario.

There are two separate synchronized blocks in above code. In case two threads are woken up simultaneously by another thread, both

threads will enter one after in the second synchronized block.

Only one of the two threads will get new value from the queue and make it empty. The second thread will poll on an empty queue and it will not get any non-null return value.

312.How can we check if a thread has a monitor lock on a given object?

In Java, Thread class has a static method holdsLock(Object objToCheck) to check whether thread has a lock on objToLock object.

This method will return true if current thread holds the lock on the objToLock object that was passed as an argument to this method.

313.What is the use of yield() method in Thread class?

The yield() method of Thread class is used to give a hint to scheduler that the current thread wants to free the processor.

The scheduler can either use this hint or just ignore this hint. Since the scheduler behavior is not guaranteed, it may happen that the current thread again gets the processor time.

It can be used for debugging or testing purposes. But there is rarely any concrete use of this method.

314. What is an important point to consider while passing an object from one thread to another thread?

This is a multi-threading scenario. In a multi-threading scenario, the most important point is to check whether two threads can update same object at the same time.

If it is possible for two threads to update the same object at the same time, it can cause issues like race condition.

So it is recommended to make the object Immutable. This will help in avoiding any concurrency issues on this object.

315.What are the rules for creating Immutable Objects?

As per Java specification, following are the rules for creating an Immutable object:

Do not provide "setter" methods that modify fields or objects referred to by fields.

Make all fields final and private.

Do not allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.

If the instance fields include references to mutable objects, do not allow those objects to be changed.

Do not provide methods that modify the mutable objects.

Do not share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

316.What is the use of ThreadLocal class?

ThreadLocal class provides thread-local variables. Each thread accesses only its own local variables. It has its own copy of the variable.

By using ThreadLocal, if thread X stores a variable with value x and another thread Y stores same variable with the value y, then X gets x from its ThreadLocal instance and Y gets y from its ThreadLocal instance.

Typically, ThreadLocal instances are private static fields that are associated with the state of a thread.

317.What are the scenarios suitable for using ThreadLocal class?

We can use instance of ThreadLocal class to transport information within an application.

One use case is to transport security or login information within an instance of ThreadLocal so that every method can access it.

Another use case is to transport transaction information across an application, without using the method-to-method communication.

318.How will you improve the performance of an application by multi-threading?

In an environment with more than one CPU, we can parallelize the computation tasks on multiple CPUs. This leads to parallel processing of a bigger task that takes lesser time due to multiple threads dividing the work among themselves.

One example is that if we have to process 1000 data files and calculate the sum of numbers in each file. If each file takes 5 minutes, then 1000 files will take 5000 minutes for processing.

But by using multi-threading we can process these files in 10 parallel threads. So each thread will take 100 files each. Since now work is happening in 10 parallel threads, the time taken will be around 500 minutes.

319.What is scalability in a Software program?

Scalability is the capability of a program to handle growing amount of work or its potential to be enlarged in order to accommodate growth.

A program is considered scalable, if it is suitable to handle a large amount of input data or a large number of users or a large number of nodes.

When we say a program does not scale, it means that program fails on increasing the size of task.

320. How will you calculate the maximum speed up of an application by using multiple processors?

Amdahl’s law gives the theoretical speedup in latency of the execution of a task at fixed workload.

It gives the formula to compute the theoretical maximum speed up that can be achieved by providing multiple processors to an application.

If S is the theoretical speedup then the formula is: $S(n) = 1 / (B + (1-B)/n)$

where n is the number of processors
B is the fraction of the program that cannot be executed in parallel.

When n converges against infinity, the term (1-B)/n converges against zero. Therefore, the formula can be reduced in this special case to 1/B.

In general, the theoretical maximum speedup behaves in inverse proportion to the fraction that has to be executed serially. This means the lower this fraction is, the more theoretical speedup can be achieved.

321.What is Lock contention in multi- threading?

Lock contention is the situation when one thread is waiting for a lock/object that being held by another thread. The waiting thread cannot use this object until the other thread releases the lock on that object.

It is also known as Thread contention.

Ideally locks reduce the thread contention. Without locks, multiple threads can operate on same object and cause undesirable behavior. If locking is implemented correctly it reduces the occurrence of contention between multiple threads.

322. What are the techniques to reduce Lock contention?

There are following main techniques to reduce Lock contention:

1. Reduce the scope of lock.
2. Reduce object pooling.
3. Reduce the number of times a certain lock can be acquired.
4. Avoid synchronization at unnecessary places.
5. Implement hardware supported Optimistic locking in place of synchronization.

323. What technique can be used in following code to reduce Lock contention?

```
synchronized (map) {
Random r = new Random();
Integer value = Integer.valueOf(42); String key = r.nextString(5); map.put(key, value);

}
```

The code uses Random() to get a random string and it also used Integer to convert 42 in an object. Since these lines of code are specific to this thread, these can be moved out of Synchronization block.

```
Random r = new Random();
Integer value = Integer.valueOf(42); String key = r.nextString(5);

synchronized (map) { map.put(key, value);

}
```

324. What is Lock splitting technique?

Lock splitting is a technique to reduce Lock contention in multi- threading. It is applicable in scenario when one lock is used to synchronize access to different aspects of the same application.

Sometimes we put one lock to protect the whole array. There can be multiple threads trying to get the lock for same array. This single lock on array can cause Lock contention among threads. To resolve this we can give one lock to each element of the array. Or we can use modulus function to assign different locks to a small group of array elements. In this way we can reduced the chance of Lock contention. This is Lock splitting technique.

325. Which technique is used in ReadWriteLock class for reducing Lock contention?

ReadWriteLock uses two locks. One lock for read-only operations, another lock for write operations.

Its implementation is based on the premise that concurrent threads do not need a lock when they want to read a value while no other thread is trying to write.

In this implementation, read-only lock can be obtained by multiple threads. And the implementation guarantees that all read operation will see only the latest updated value as soon as the write lock is released.

326. What is Lock striping?

In Lock splitting we use different locks for different parts of the application. In Lock striping we use multiple locks to protect different parts of the same data structure.

ConcurrentHashMap class of Java internally uses different buckets to store its values. Each bucket is chosen based on the value of key. ConcurrentHashMap uses different locks to guard different buckets. When one thread that tries to access a hash bucket, it can acquire the lock for that bucket. While another thread can simultaneously acquire lock for another bucket and access it. In a synchronized version of HashMap, the whole map is has one lock.

Lock striping technique gives better performance than Synchronizing the whole data structure.

327. What is a CAS operation?

CAS is also known a Compare-And-Swap operation.

In a CAS operation, the processor provides a separate instruction that can update the value of a register only if the provided value is equal to the current value.

CAS operation can be used as an alternate to synchronization.

Let say thread T1 can update a value by passing its current value and the new value to be updated to the CAS operation. In case another thread T2 has updated the current value of previous thread, the previous thread T1's current value is not equal to the current value of T2. Hence the update operation fails.

In this case, thread T1 will read the current value again and try to update it.

This is an example of optimistic locking.

328. Which Java classes use CAS operation?

Java classes like AtomicInteger or AtomicBoolean internally use CAS operations to support multi-threading.

These classes are in package java.util.concurrent.atomic.

329. Is it always possible to improve performance by object pooling in a multi-threading application?

By using Object pools in an application we limit the number of new objects to be created for a class. In a single thread operation, it can improve the performance by reusing an already created object from a pool.

In a multi-threading application an object pool has to provide synchronized access to multiple threads. Due to this only one thread can access the pool at a time. Also there is additional cost due to Lock contention on pool. These additional costs can outweigh the cost saved by reuse of an object from the pool.

Therefore using an Object pool may not always improve the performance in a multi-threading application.

330. How can techniques used for performance improvement in a single thread application may degrade the performance in a multi-threading application?

In a single thread applications we can use Object pool for performance optimization. Where as in multi-threading environment, it may not be a good idea to use an Object pool. Increased overhead of synchronization and lock contention can degrade the performance gained by using Object pool in a multi-threading application.

Another example is the implementation in which a List keeps a separate variable to hold the number of elements. This technique is useful in single thread application where size() method can return the value from this variable, without the need to count all the elements of list.

But in a multi-threading application, this separate variable can rather degrade the performance. This variable has to be access controlled by a lock since multiple concurrent threads can insert an element in a list. The additional cost of lock on this variable can outweigh the benefit gained by it in a multi-threading application.

331.What is the relation between Executor and ExecutorService interface?

Executor interface has only execute(Runnable) method. The implementing class of this interface has to execute the given Runnable instance passed to execute() method at some time in the future.

ExecutorService interface extends Executor interface. It provides additional methods like-invokeAny(), invokeAll(), shutdown(), awaitTermination(). These method provide the ability to shut-down the thread so that further requests can be rejected. Also it provides ability to invoke a collection of Callable tasks.

332. What will happen on calling submit() method of an ExecutorService instance whose queue is already full?

The implementation of ExecutorService will throw RejectedExecutionException, when its queue is already full and a new task is submitted by calling submit() method.

333. What is a ScheduledExecutorService?

ScheduledExecutorService interface extends the interface ExecutorService. It provides various schedule() methods that can be used to submit new tasks to be executed at a given point of time.

One of the schedule() method provides the ability to schedule a one- shot task that can be executed after given delay.

Another version of schedule() method provides the ability to execute ScheduleFuture after a given amount of delay.

In addition there are scheduleAtFixedRate() and scheduleWithFixedDelay() methods that can execute an action at a periodic interval of time.

334. How will you create a Thread pool in Java?

In Java, Executors framework provides a method newFixedThreadPool(int nThreads) that can be used to create a Thread pool with a fixed number of threads.

Sample code is as follows:

```
public static void main(String[] args) throws InterruptedException, ExecutionException
{
```

```
ExecutorService myService = Executors.newFixedThreadPool(5); Future<Integer>[] futureList = new
Future[5];
for (int i = 0; i < futureList.length; i++) {

futureList[i] = myService.submit(new MyCallable()); }

for (int i = 0; i < futureList.length; i++) { Integer retVal = futureList[i].get(); println(retVal);

}

myService.shutdown(); }
```

335. What is the main difference between Runnable and Callable interface?

Runnable interface defines run() method that does not return any value.

Callable interface allows call() method to return a value to its caller. A Callable interface can also throw an exception in case of an error. Also Callable is a newer addition to Java since version 1.5.

336. What are the uses of Future interface in Java?

We can use Future interface to represent the result of an asynchronous computation.

These are the operations whose result is not immediately available.

Therefore Future interface provides isDone() method to check if the asynchronous computation has finished or not.

We can also check if the task was cancelled by calling isCancelled() method.

Future also provides cancel() method to attempt the cancellation of a task.

337. What is the difference in concurrency in HashMap and in Hashtable?

In a Hashtable class all methods are synchronized.
In a HashMap implementation all the methods are not synchronized.

Therefore Hashtable is a thread-safe collection. HashMap is not a thread-safe collection.

In a multi-threading it is not advisable to use regular HashMap. We can use ConcurrentHashMap class in multi-threading applications.

338. How will you create synchronized instance of List or Map Collection?

In Java, Collections class provides methods to synchronize any collection.

It also provides synchronizedList(List) and synchronizedMap(Map) methods that can be used to convert a List or Map to a synchronized instance.

339. What is a Semaphore in Java?

Semaphore class in Java is used to implement a counting semaphore. It is used to restrict the number of threads that can access a physical or logical resource.

A Semaphore maintains a set of permits that should be acquired by competing threads.

We can also use it to control how many threads can access the critical section of a program or a resource concurrently.

The first argument in Semaphore constructor is the total number of permits available. Each invocation of acquire() method tries to obtain one of the available permits.

The acquire() method is used to acquire a permit from the semaphore. If we pass number of permits required to acquire() method, then it blocks the thread until that number of permits are available.

Once a thread has finished its work, we can use release() method to release the permits.

340. What is a CountDownLatch in Java?

CountDownLatch class helps in implementing synchronization in Java. It is used to implement the scenarios in which one or more threads have to wait until other threads have reached the same state such that all thread can start.

There is a synchronized counter that is decremented until it reaches the value zero. Once it reaches zero, it means that all waiting threads can proceed now.

It is a versatile tool that can be used for other Synchronization scenarios as well. It can also work as on/off latch or gate. All threads invoking await() method wait at the gate until it is opened by a thread invoking countdown() method.

341. What is the difference between CountDownLatch and CyclicBarrier?

CyclicBarrier takes an optional Runnable task that is run once the common barrier condition is achieved.

CountDownLatch is used in simple use cases where a simple start stop is required. A CyclicBarrier is useful in complex scenarios where more coordination is required. E.g. MapReduce algorithm implementation.

CyclicBarrier resets the internal value to the initial value once the value reaches zero. CyclicBarrier can be used to implement the scenarios in which threads have to wait for each other multiple times.

342. What are the scenarios suitable for using Fork/Join framework?

ForkJoinPool class is in the center of Fork/Join framework. It is a thread pool that can execute instances of ForkJoinTask.

ForkJoinTask class provides the fork() and join() methods. The fork() method is used to start the asynchronous execution of a task. The join() method is used to await the result of the computation.

Therefore, divide-and-conquer algorithms can be easily implemented with Fork/Join framework.

343. What is the difference between RecursiveTask and RecursiveAction class?

RecursiveAction class has compute() method that does not have to return a value.

RecursiveAction can be used when the action has to directly operate on a Data structure. It does not need to return any computed value.

In RecursiveTask class has compute() method that always returns a value.

Both RecursiveTask and RecursiveAction classes are used in ForkJoinTask implementations.

344. In Java 8, can we process stream operations with a Thread pool?

In Java 8, Collections provide parallelStream() method to create a stream that can be processed by a Thread pool.

We can also call the intermediate method parallel() on a given stream to convert it into a sequential stream of parallel tasks.

345. What are the scenarios to use parallel stream in Java 8?

A parallel stream in Java 8 has a much higher overhead compared to a sequential one.

It takes a significant amount of time to coordinate the threads. We can use parallel stream in following scenarios:

When there are a large number of items to process and the processing of each item takes time and is parallelizable.
When there is a performance problem in the sequential processing. When current implementation is not already running in a multi- thread environment. If there is already a multi-threading environment, adding parallel stream can degrade the performance.

346. How Stack and Heap work in Java multi-threading environment?

In Java, Stack and heap are memory areas available to an application. Every thread has its own stack. It is used to store local variables, method parameters and call stack.

Local variables stored in Stack of one Thread are not visible to another Thread.

Where as, Heap is a common memory area in JVM. Heap is shared by all threads. All objects are created inside heap.

To improve performance thread can cache the values from heap into their stack. This can create problem if the same variable is modified by more than one thread.

In such a scenario we should used volatile keyword to mark a variable volatile. For a volatile variable the thread always reads the value from main memory.

347. How can we take Thread dump in Java?

The steps to take Thread dump of Java process depends on the operating system.

On taking Thread dump, Java writes the state of all threads in log files or standard error console.

We can press Ctrl + Break key together to take thread dump in Windows.
We can execute kill -3 command for taking Thread dump on Linux. Another option to take Thread dump is jstack tool. We can pass process id of java process to this tool for taking Thread dump.

This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this list of JVM options to learn more about this parameter.

348. Which parameter can be used to control stack size of a thread in Java?

We use –Xss parameter to control the stack size of a thread in Java. If we set it as 1 MB, then every thread will get 1MB of stack size.

349. There are two threads T1 and T2? How will you ensure that these threads run in sequence T1, T2 in Java?

In Java there are multiple ways to execute threads in a sequence.

One of the simplest way for sequencing is join() method of Thread class.

We can call join() method to start a thread when another thread has finished.

We start with the last thread to execute first. And make this thread join on the next thread.

In this case we start thread T2 first. And then call T1.join() so that thread T2 waits for thread T1 to finish execution.

Once T1 completes execution, T2 thread starts executing.

Java 8

350. What are the new features released in Java 8?

The new features released in Java 8 are:

1. Lambda Expression

2. Stream API

3. Date and Time API

4. Functional Interface

5. Interface Default and Static Methods

6. Optional

7. Base64 Encoding and Decoding

8. Nashorn JavaScript Engine

9. Collections API Enhancements

10. Concurrency Enhancements

11. Fork/Join Framework Enhancements

12. Spliterator

13. Internal Iteration

14. Type Annotations and Repeatable Annotations

15. Method Parameter Reflection

16. JVM Parameter Changes

351.What are the main benefits of new features introduced in Java 8?

The main benefits of Java 8 features are:

1. Support for functional programming by Lambda and Streams

2. Ease of high volume data processing by Streams

3. Ease of use by getting Parameter names through Reflection

4. Reusable code with enhanced Collection APIs

5. Smart exception handling with Optional

6. Control on JVM with new Parameters

7. Enhanced encryption support with Base 64

8. Faster execution with Nashorn JavaScript engine support

352. What is a Lambda expression in Java 8?

Lambda expression is an anonymous function. It is like a method that does not need any access modifiers, name or return value declaration. It accepts a set of input parameters and returns result.

Lambda expression can be passed as a parameter in a method. So we can treat code in Lambda expression as data. This piece of code can be passed to other objects and methods.

353. What are the three main parts of a Lambda expression in Java?

Three main parts of a Lambda expression are:

1. Parameter list: A Lambda expression can have zero or more parameters. Parameter list is optional to Lambda.

2. Lambda arrow operator: “->” is known as Lambda arrow operator. It separates the list of parameters and the body of Lambda.

3. Lambda expression body: The piece of code that we want to execute is written in Lambda expression body.

E.g. In following example:
Arrays.asList("a", "b", "d").forEach(e -> System.out.println(e));

Parameter list = e
Arrow = ->
Body = System.out.println(e)

354. What is the data type of a Lambda expression?

A Lambda expression fulfills the purpose of passing code as data. The data type of a Lambda expression is a Functional interface. In most of the cases this is java.lang.Runnable interface.

355. What is the meaning of following lambda expression?

(e -> System.out.println(e));

This Lambda expression takes a parameter e and prints it via System.out.

356. Why did Oracle release a new version of Java like Java 8?

The main theme of Java 8 is support for functional programming. With increase in Database size and growth of multi-code CPU servers, there is need for Java to support such large-scale systems.

With new features of Java 8, it is possible to create functional programs to interact efficiently with Big Data systems. Support for Streams is very helpful in this regard.

Lambda expressions are very useful for cloud computing where we can pass code as data and run the same code on multiple servers.

Optional is a best practice that is borrowed from Google Guava library for handling the exceptional cases. This has made programs more robust with support for edge cases.

357. What are the advantages of a lambda expression?

We can pass a lambda expression as an object to a method. This reduces the overhead involved in passing an anonymous class.

We can also pass a method as a parameter to another method using lambda expressions.

358. What is a Functional interface in Java 8?

A Functional interface in Java is an interface that has exactly one abstract method.

It can have default methods with implementation. A default method is not abstract.

In Java 8, java.lang.Runnable and java.util.concurrent.Callable are two very popular Functional interfaces.

359. What is a Single Abstract Method (SAM) interface in Java 8?

A Functional interface is also known as Single Abstract Method Interface, since it has exactly one abstract method.

360. How can we define a Functional interface in Java 8?

To define a Functional interface in Java 8, we can create an Interface with exactly one abstract method.

Another way is to mark an Interface with annotation @FunctionalInterface. Even with the annotation we have to follow the rule of exactly one abstract method.

The only exception to this rule is that if we override java.lang.Object class's method as an abstract method, then it does not count as an abstract method.

361.Why do we need Functional interface in Java?

Functional Interfaces are mainly used in Lambda expressions, Method reference and constructor references.

In functional programming, code can be treated as data. For this purpose Lambda expressions are introduced. They can be used to pass a block of code to another method or object.

Functional Interface serves as a data type for Lambda expressions. Since a Functional interface contains only one abstract method, the implementation of that method becomes the code that gets passed as an argument to another method.

362. Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

No, it is not mandatory to mark a Functional interface with @FunctionalInterface annotation.

Java does not impose this rule.

But, if we mark an interface with @FunctionalInterface annotation then Java Compiler will give us error in case we define more than one abstract method inside that interface.

363. What are the differences between Collection and Stream API in Java 8?

Main differences between Collection and Stream API in Java 8 are:

1. Version: Collection API is in use since Java 1.2. Stream API is recent addition to Java in version 8.

2. Usage: Collection API is used for storing data in different kinds of data structures. Stream API is used for computation of data on a large set of Objects.

3. Finite: With Collection API we can store a finite number of elements in a data structure. With Stream API, we can handle streams of data that can contain infinite number of elements.

4. Eager vs. Lazy: Collection API constructs objects in an eager manner. Stream API creates objects in a lazy manner.

5. Multiple consumption: Most of the Collection APIs support iteration and consumption of elements multiple times. With Stream API we can consume or iterate elements only once.

364. What are the main uses of Stream API in Java 8?

Main uses of Stream API in Java 8 are:

1. It helps in using data in a declarative way. We can make use of Database functions like Max, Min etc., without running a full iteration.

2. It makes good use of multi-core architectures without worrying about multi-threading code.

3. We can create a pipeline of data operations with Java Stream that can run in a sequence or in parallel.

4. It provides support for group by, order by etc. operations.

5. It supports writing for code in Functional programming style.

6. It provides parallel processing of data.

365. What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

Main differences between Intermediate and Terminal Stream operations are as follows:

1. Evaluation: Intermediate operations are not evaluated until we chain it with a Terminal Operation of Stream. Terminal Operations can be independently evaluated.

2. Output: The output of Intermediate Operations is another Stream. The output of Terminal Operations is not a Stream.

3. Lazy: Intermediate Operations are evaluated in lazy manner. Terminal Operations are evaluated in eager manner.

4. Chaining: We can chain multiple Intermediate Operations in a Stream. Terminal Operations cannot be chained multiple times.

5. Multiple: There can be multiple Intermediate operations in a Stream operation. There can be only one Terminal operation in Stream processing statement.

366. What is a Spliterator in Java 8?

A Spliterator is a special type of Iterator to traverse and partition the elements of a source in Java. A source can be a collection, an IO channel or a generator function.

A Spliterator may traverse elements individually or sequentially in bulk.

367. What are the differences between Iterator and Spliterator in Java 8?

Main differences between Iterator and Spliterator are as follows:

1. Spliterator can be used with Streams in Java 8. Where as, Iterator is just used with Collection.

2. Spliterator uses Internal Iteration to iterate Streams. Iterator uses External Iteration to iterate Collections.

3. Spliterator can iterate Streams in Parallel as well as Sequential manner. Iterator only iterates in Sequential manner.

4. Spliterator can traverse elements individually as well as in bulk. Iterator only iterates elements individually.

368. What is Type Inference in Java 8?

A Java compiler can see each method's invocation and it declaration to determine what are type arguments required for invocation.

By Type Inference, Java can determine the types of the arguments as well as the type of the result being returned.

Type inference algorithm also tries to find the most specific type that can work with all types of arguments.

369. Does Java 7 support Type Inference?

Yes, Java 7 supports Type Inference. In Java 8, Oracle has enhanced the Type Inference concept. Now it can be used to define Lambda expressions, functions and Method references.

370. How does Internal Iteration work in Java 8?

In an Iterator, the fundamental question is that which party controls the iteration. Is it Iterator or the Collection on which iterator runs.

When a Collection controls the iterator, then it is called External Iteration. When the Iterator controls the iteration then it is called Internal Iteration.

In case of Internal Iteration, the client hands over an operation to Iterator and the Iterator applies the operation to all the elements in aggregate.

Internal Iteration is easier to implement, since the Iterator does not have to store the state of the collection.

371.What are the main differences between Internal and External Iterator?

Main differences between Internal and External Iterator are as follows:

1. An Internal Iterator controls the iteration itself. In an External Iterator collection controls the iteration.

2. Internal Iterator can iterate elements in individually as well as in

3. Bulk (like forEach). External iterator iterates element one by one.

4. Internal Iterator does not have to iterate elements only sequentially. External Iterator always iterates sequentially.

5. Internal Iterator supports declarative programming style that goes well with functional programming. External Iterator follows imperative style OOPS programming.

6. Some people consider Internal Iterator code more readable than that of External Iterator.

372. What are the main advantages of Internal Iterator over External Iterator in Java 8?

Some of the main advantages of Internal Iterator are:

1. 2.

3. 4.

Internal Iterator is based on Functional programming, therefore it can work on declarative style code. There is no need to sequentially iterate elements in Internal Iterator.

Code is more readable and concise in Internal Iterator. Internal Iterator supports concurrency and parallel processing.

373. What are the applications in which we should use Internal Iteration?

We need Internal Iterator in applications that require high performance, parallel processing, fast iteration and bulk operations support.

Also in Internal Iteration applications, we do not have much control over iteration. The other features like parallel processing etc. become more important.

374. What is the main disadvantage of Internal Iteration over External Iteration?

Internal Iteration has many advantages over External Iteration. But it has one big disadvantage. Since Java API is responsible for iterating in Internal iterator, developer does not get any control over iteration.

375. Can we provide implementation of a method in a Java Interface?

Before Java 8, it was not allowed to provide implementation of a method in an Interface.

Java 8 has introduced the flexibility of providing implementation of a method in an interface. There are two options for that:

1. Default Method: We can give default implementation of a method.

2. Static Method: We can create a static method in an interface and provide implementation.

376. What is a Default Method in an Interface?

In Java 8, we can provide implementation of a method in an Interface and mark this method with Default keyword.

In this way, this implementation of the method becomes default behavior for any class implementing the interface.

377. Why do we need Default method in a Java 8 Interface?

Default methods in an Interface provide backward compatibility feature in Java 8.

Let say there is an interface Car that is implemented by BMW, Chevrolet and Toyota classes. Now a Car needs to add capability for flying. It will require change in Car interface. Some of the car classes that do not have flying capability may fail. Therefore a Default Implementation of flying methods is added in Car interface so that cars with no flying capability can continue to implement the original Car interface.

378. What is the purpose of a Static method in an Interface in Java 8?

A Static method in an Interface is utility or helper method. This is not an object level instance method. Some of the uses of Static method in an Interface are:

1. Single Class: There is no need to create a separate Utils class for storing utility or helper methods. We can keep these methods in same interface.

2. Encapsulation: With Static methods, complete behavior of a Class is encapsulated in same class. There is no need to maintain multiple classes.

3. Extension: It is easier to extend a Class/API. If we extend a collection ArrayList, we get all the methods. We need not extend Collections class also.

379. What are the core ideas behind the Date/Time API of Java 8?

There are three core ideas behind the Date/Time API of Java 8:

1. Immutable-value classes: The new API avoids thread- safety and concurrency issues by ensuring that all the core classes are immutable and represent well-defined values.

2. Domain-driven design: The new API is modeled on precise domain with classes that represent different use cases for Date and Time.

3. The emphasis on domain-driven design offers benefits like clarity and understandability.

4. Separation of chronologies: The new API allows people to work with different calendar systems. It supports the needs of users in different areas of the world likes Japan or Thailand that don’t follow ISO-8601.

380. What are the advantages of new Date and Time API in Java 8 over old Date API?

Some of the advantages of Java 8 Date Time API over existing Date API are:

Concurrency: Existing Date Time classes (such as java.util.Date and SimpleDateFormatter) are not thread-safe. This does not work well in concurrent applications. In new Date Time API, developer does not have to deal with concurrency issues while writing date- handling code.

Better Design: Date/Time classes prior to Java 8 have poor API design. For example, years in java.u-til.Date start at 1900, months start at 1, and days start at 0. It is not very intuitive. Java 8 Date Time API handles it very well.

No need for 3rd Party Libraries: With the popularity of third-party Date/Time libraries like Joda Time, Java has to make its native Date/Time API comparable. Now we can use the Java API instead of using 3rd party libraries.

381.What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?

Main difference between legacy Date/Time API and Java 8 Date/Time API are:

1. Old API is not Thread safe. Java 8 API is Thread safe.

2. Old API has many mutable objects. New Java 8 API is based on Immutable objects.

3. Performance of old API is not good. New Java 8 Date/Time API gives better performance.

4. Old API is less readable and maintainable. New Java 8 API is very well designed and is more readable.

5. Old API has month values from 0 to 11. New API has months from 1 to 12.

382. How can we get duration between two dates or time in Java 8?

In Java8, we have a new class Duration that provides the utility of computing duration between two dates.

We can call the static method Duration.between(date1, date2) to get the time period in hours, mins, days etc. between date1 and date2.

383. What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?

Java 8 has enhanced the Arrays class with methods that can run efficiently on multi core machines.

These methods start with keyword parallel.
Egg. Arrays.parallelSetAll(), Arrays.parallelSort() etc.

This parallel set of methods provides parallel processing of Arrays that can run Java code very fast on a multi core machine.

384. How does Java 8 solve Diamond problem of Multiple Inheritance?

In Multiple Inheritance if a class extends more than one classes with two different implementation of same method then it causes Diamond problem.

Consider following example to see problem and solution for Diamond problem in Java 8:

```
public interface BaseInterface{
default void display() { //code goes here }

}
public interface BaseOne extends BaseInterface { }
public interface BaseTwo extends BaseInterface { }
public class ChildClass implements BaseOne, BaseTwo { }
```

In the above code, class ChildClass gives compile time error. Java Compiler cannot decide which display method should it invoke in ChildClass.

To solve this problem, Java SE 8 has given the following remedy:

```
public interface A{
default void display() { //code goes here }

}
public interface B extends A{ } public interface C extends A{ } public class D implements B,C{

default void display() { B.super.display();

} }

public interface BaseInterface{
default void display() { //code goes here }

}
public interface BaseOne extends BaseInterface { } public interface BaseTwo extends BaseInterface
{ } public class ChildClass implements BaseOne, BaseTwo {

default void display(){

BaseOne.super.display(); }

}
```

The method invocation at BaseOne.super.display(); solves the Diamond problem as it resolves the confusion for compiler.

385. What are the differences between Predicate, Supplier and Consumer in Java 8?

The subtle difference between Predicate, Supplier and Consumer in Java 8 is as follows:

Predicate is an anonymous function that accepts one argument and returns a result.

Supplier is an anonymous function that accepts no argument and returns a result.

Consumer is an anonymous function that accepts one argument and returns no result.

386. Is it possible to have default method definition in an interface without marking it with default keyword?

No, we have to always mark a default method in interface with default keyword.

If we create a method with implementation in an interface, but do not mark it as default, then we will get compile time error.

387. Can we create a class that implements two Interfaces with default methods of same name and signature?

No, it is not allowed to create a class that implements interfaces with same name default methods.

It will give us compile time error for duplicate default methods.

388. How Java 8 supports Multiple Inheritance?

In Multiple Inheritance a class can inherit behavior from more than one parent classes.
Prior to Java 8, a class can implement multiple interfaces but extend only one class.

In Java 8, we can have method implementation within an interface. So an interface behaves like an Abstract class.

Now if we implement more than one interface with method implementation in a class, it means we are inheriting behavior from multiple abstract classes. That is how we get Multiple Inheritance in Java 8.

389. In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?

In such a scenario, JVM will pick the definition in base class.

390. If we create same method and define it in a class , in its parent class and in an interface implemented by the class, then definition will be invoked if we access it using the reference of Interface and the object of class?

In all the cases, method defined in the class will be invoked.

391.Can we access a static method of an interface by using reference of the interface?

No, a static method of interface has to be invoked by using the name of the interface.

392. How can you get the name of Parameter in Java by using reflection?

Java 8 has introduced a method Parameter.getName() to get the name of a parameter by using reflection.

Before using this feature, we need to turn on this feature in Java compiler.

To turn on this feature, just run javac with –parameters argument.

To verify the availability of this feature, we can use Parameter. isNamePresent() method.

393. What is Optional in Java 8?

Optional is a container object that may have a null or non-null value. If it has a value then isPresent() method returns true.

It a value is present, we can call get() method to get the value. Else we will get nothing.

It is very useful in handling data that has null values.

394. What are the uses of Optional?

Some of the uses of Optional in Java are:

We can use Optional to avoid NullPointerException in an application.
Optional performs Null check at compile time, so we do not get run time exception for a null value.

Optional reduces the codebase pollution by removing unnecessary null checks.
Optional can also be used to handle default case for data when a value is null.

395. Which method in Optional provides the fallback mechanism in case of null value?

In case, an Optional has null value, we can use orElseGet() method as fallback mechanism. If we implement orElseGet() method, it will be invoked when the value of Optional is null.

396. How can we get current time by using Date/Time API of Java 8?

In Java 8 we can use Clock class to get the current time. Instead of using old method System.currentTimeMillis(), we can create a Clock object and call millis() method to get the current time in milliseconds.

We can also call instant() method on Clock object to get the current time in a readable format.

397. Is it possible to define a static method in an Interface?

Yes, from Java 8, an Interface can also has a static method.

398. How can we analyze the dependencies in Java classes and packages?

Java 8 comes with a new command line tool jdeps that can help in analyzing the package-level and class-level dependencies.

We can pass a jar file name or a class name as an argument to this tool. It will list all the dependencies of that jar or class.

399. What are the new JVM arguments introduced by Java 8?

In Java 8, PermGen space of ClassLoader is removed. It has been replaced with MetaSpace.

Now we can set the initial and maximum size of MetaSpace.

The JVM options -XX:PermSize and –XX:MaxPermSize are replaced by -XX:MetaSpaceSize and -XX:MaxMetaspaceSize respectively in Java 8.

400. What are the popular annotations introduced in Java 8?

Some of the popular annotations introduced in Java 8 are:

@FunctionalInterface: This annotation is used to mark an interface as Functional Interface. As mentioned earlier, A FunctionalInterface can be used for lambda expressions.

@Repeatable: This annotation is used for marking another annotation. It indicates that the marked annotation can be applied multiple times on a type.

401. What is a StringJoiner in Java 8?

StringJoiner is a new class in Java 8 that can be used to create a String. It can construct a sequence of characters separated by a delimiter. It can also optionally add a prefix and suffix to this sequence. We can use this sequence to get a String.

E.g.
The String "[One:Two:Three]" may be constructed as follows:

```
StringJoiner sj = new StringJoiner(":", "[", "]"); sj.add("One").add("Two").add("Three");
String desiredString = sj.toString();
```

402. What is the type of a Lambda expression in Java 8?

The type of a lambda expression depends on the context it is being used.

A lambda is like a method reference. It does not have a type of its own.

Generally, a Lambda is an instance of a Functional Interface.

403. What is the target type of a lambda expression ?

The target type of a lambda expression represents a type to which the expression can be converted.

The target type for a lambda expression is a functional interface.

The lambda expression must have same parameter type as the parameter in the function of the interface. It must also return a type compatible with the return type of function.

404. What are the main differences between an interface with default method and an abstract class in Java 8?

An interface with a default method appears same as an Abstract class in Java. But there are subtle differences between two.

1. Instance variable: An interface cannot have instance variables. An abstract class can have instance variables.

2. Constructor: An interface cannot have a constructor. An abstract class can have constructor.

3. Concrete Method: An interface cannot have concrete methods other than default method. An abstract class is allowed to define concrete methods with implementation.

4. Lambda: An interface with exactly one default method can be used for lambda expression. An abstract class cannot be used for lambda expression.

Java Tricky Questions

405. Is there any difference between a = a + b and a += b expressions?

When we add two integral variables e.g. variables of type byte, short, or int in Java, then they are first promoted to int type, and then addition happens.

The += operator implicitly casts the result of addition into the type of variable used to hold the result.
What happens when you put return statement or System.exit () on try or catch block? Will finally block execute?

It is a popular tricky Java interview question. Most of the programmers think that no matter what the finally block will always execute. This question challenges that concept by putting a return statement in the try or catch block or calling System.exit() from try or catch block.

You can answer by saying that finally block executes even if we put a return statement in the try block or catch block. But finally block does not execute if you call System.exit() from try or catch block.

406. What does the expression 1.0 / 0.0 return? Will there be any compilation error?

Double class is the source of many tricky interview questions. You may know about the double primitive type and Double class. But while doing floating point arithmetic some people don't pay enough attention to Double.INFINITY, NaN, and -0.0. There are rules that govern the floating point arithmetic calculations involving Double.

The answer to this question is that 1.0 / 0.0 will compile successfully. And it will not throw ArithmeticException. It will just return Double.INFINITY.

.

407. Can we use multiple main methods in multiple classes?

Yes. When we start an application in Java, we just mention the class name to be run to java command. The JVM looks for the main method only in the class whose name is passed to java command. Therefore, there is no conflict amongst the multiple classes having main method.

408. Does Java allow you to override a private or static method?

The question is tricky but the answer is very simple. You cannot override a private or static method in Java. If we create a similar method with same return type and same method arguments in child class, then it will hide the superclass method. This is known as method hiding.

Also, you cannot override a private method in sub class because Private method is not visible even in a subclass. Therefore, what you can do is to create another private method with the same name in the child class.

So in both the cases, it is not method overriding. It is either method hiding or a new method.

409. What happens when you put a key object in a HashMap that is already present?

In a HashMap there are buckets in which objects are stored. Key objects with same HashCode go to same bucket.

If you put the same key again in a HashMap, then it will replace the old mapping because HashMap doesn't allow duplicate keys. The same key will have same HashCode as previous key object. Due to same HashCode, it will be stored at the same position in the bucket.

410. How can you make sure that N threads can access N resources without deadlock?

This question checks your knowledge of writing multi-threading code. If you have experience with deadlock and race conditions, you can easily answer this.

The answer is that by resource ordering you can prevent deadlock. If in our program we always acquire resources in a particular order and release resources in the reverse order, then we can prevent the deadlock.

So a thread waiting for same resource can not get into deadlock while the other thread is trying to get it and holding the resource required by first thread. If both of them release the resources in right order, one of them can acquire it to finish the work.

411.How can you determine if JVM is 32-bit or 64-bit from Java Program?

We can find JVM bit size 32 bit or 64 bit by running java command from the command prompt.

Or we can get it from Java program.

Sun has a Java System property to determine the bit size of the JVM: 32 or 64:

sun.arch.data.model=32 // 32 bit JVM sun.arch.data.model=64 // 64 bit JVM

We can use System.getProperty("sun.arch.data.model") to determine if it is 32/64 bit from Java program.

412. What is the right data type to represent Money (like Dollar/Pound) in Java?

To represent money you need decimal points in the numbers like $1.99.

BigDecimal class provides good methods to represent Money. Using BigDecimal, we can do the calculation with decimal points and correct rounding. But using BigDecimal is a little bit high on memory usage.

We can also use double with predefined precision. But calculation on double can give erroneous results.

413. How can you do multiple inheritances in Java?

This is a question to trick people coming from C++ and Scala background to Java. There are many Object Oriented languages that support multiple inheritances. But Java is not one of them.

Answer of this question can be that, Java does support multiple inheritances of by allowing an interface to extend other interfaces. You can implement more than one interface. But you cannot extend multiple classes. So Java doesn't support multiple inheritances of implementation.

But in Java 8, the default method breaks the rule of multiple inheritances behavior.

414. Is ++ operation thread-safe in Java?

No, ++ operator is not a thread safe operation. It involves multiple instructions like- reading a value, incrementing it and storing it back into memory. These instructions can overlap between multiple threads. So it can cause issues in multi-threading.

415. How can you access a non- static variable from the static context?

We cannot access a non-static variable from the static context in Java. If you write a code like that, then you will get compile time error. It is one of the most common problems for beginner Java programmers, when they try to access instance variable inside the main method in a class.

Since main method is static in Java, and instance variables are non- static, we cannot access instance variable inside main. The solution is to create an instance of the object and then access the instance variables.

416. Let say there is a method that throws NullPointerException in the superclass. Can we override it with a method that throws RuntimeException?

This question is checking your understanding of the concepts of method overloading and overriding in Java.

We can throw superclass of RuntimeException in an overridden method, but we cannot do the same if it is a checked Exception.

417. How can you mark an array volatile in Java?

If you know multi-threading well then you can easily answer it.

We can mark an array volatile in Java. But it makes only the reference to array volatile, not the whole array.

If one thread changes the reference variable to point to another array, then it will provide a volatile guarantee. But if multiple threads are changing individual array elements, they won't be having same reference due to the reference itself being volatile.

418. What is a thread local variable in Java?

Thread-local variable is a variable restricted to a specific thread. It is like thread's own copy of variable that is not shared among multiple threads.
Java provides ThreadLocal class to support thread-local variables. To achieve thread-safety, you can use it. To avoid any memory leak, it is always good to remove a thread-local variable, once its work is done.

419. What is the difference between sleep() and wait() methods in Java?

In Java, we use these methods to pause currently running thread. There is a simple difference between these.

sleep() is actually meant for short pause because it doesn't release lock.

wait() is meant for conditional wait and it can release a lock that can be acquired by another thread to change the condition on which it is waiting.

420. Can you create an Immutable object that contains a mutable object?

In Java, it is possible to create an Immutable object that contains a mutable object.

We should not share the reference of the mutable object, since it is inside an immutable object. Instead, we can return a copy of it to other methods.

421. How can you convert an Array of bytes to String?

You can convert an Array of bytes to String object by using the String constructor that accepts byte[]. We need to make sure that right character encoding is used. Else we may get different results after conversion.

422. What is difference between CyclicBarrier and CountDownLatch class?

CyclicBarrier and CountDownLatch classes were introduced from Java 5.

We can reuse CyclicBarrier even if it is broken, but we cannot reuse CountDownLatch in Java.

423. What is the difference between StringBuffer and StringBuilder?

StringBuilder was introduced in Java 5. The main difference between both of them is that StringBuffer methods e.g. length(), capacity(), append() are synchronized. But corresponding methods in StringBuilder are not synchronized.

Due to this difference, concatenation of String using StringBuilder is faster than StringBuffer. Now it is considered bad practice to use StringBuffer, because, in most of the scenarios, we perform string concatenation in the same thread.

424. Which class contains clone method? Cloneable or Object class?

It is a very basic trick question. clone() method is defined in Object class. Cloneable is a marker interface that doesn't contain any method.

425. How will you take thread dump in Java?

There are platform specific commands to take thread dump in Java.

In Linux/Unix, just use kill -3 PID, where PID is the process id of Java process. It will give the thread dump of Java process.

In Windows, press Ctrl + Break. This will instruct JVM to print thread dump in standard out or err. It can also go to console or log file depending upon your application configuration.

426. Can you cast an int variable into a byte variable? What happens if the value of int is larger than byte?

An int is 32 bit in Java. But a byte is just 8 bit in Java. We can cast an int to byte. But we will lose higher 24 bits of int while casting. Because a byte can hold only first 8 bits of int. Remaining 24 bits (32-8 = 24) will be lost.

427. In Java, can we store a double value in a long variable without explicit casting?

No, we cannot store a double value into a long variable without casting it to long. The range of double is more than that of long. So we need to type cast.

To answer this question, just remember which one is bigger between double and long in Java.

428. What will this return 5*0.1 == 0.5? true or false?

The answer is false because floating point numbers can not be represented exactly in Java, so 5*0.1 is not same as 0.5.

429. Out of an int and Integer, which one takes more memory?

An Integer object takes more memory than an int in Java. An Integer is an object and it stores meta-data overhead about the object. An int is a primitive type so its takes less memory and there is no meta-data overhead.

430. Can we use String in the switch case statement in Java?

Yes. From Java 7 onwards, String can be used in switch case statement. This gives convenience to programmer. But internally hash code of String is used for the switch statement.

431. Can we use multiple main methods in same class?

Yes. You can have multiple methods with name main in the same class. But there should be only one main method with the signature public static void main(String[] args). JVM looks for main with this signature only. Other methods with name main in same class are just ignored.

432. When creating an abstract class, is it a good idea to call abstract methods inside its constructor?

No, we should avoid calling abstract methods in the constructor of an abstract class. Because, it can restrict how these abstract methods can be implemented by child classes.

Many IDE give “Overridable method call in constructor” warning for such implementation.

This is a problem of object initialization order. The superclass constructor will run before the child class constructor. It means child class is not yet initialized. But due to presence of overridden method in superclass, the overridden method of subclass is called when the subclass is not fully initialized.

433. How can you do constructor chaining in Java?

When we call one constructor from another constructor of the same class, then it is known as constructor chaining in Java. When you have multiple overloaded constructors in a class, you can do constructor chaining.

434. How can we find the memory usage of JVM from Java code?

We can use memory management related methods provided in java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.

By using these methods, you can find out how much of the heap is used and how much heap space still remains.

Runtime.freeMemory() returns amount of free memory in bytes. Runtime.totalMemory() returns total memory in bytes. Runtime.maxMemory() returns maximum memory in bytes.

435. What is the difference between x == y and x.equals(y) expressions in Java?

The x == y expression does object reference matching if both a and b are an object and only returns true if both are pointing to the same object in the heap space.

The x.equals(y) expression is used for logical mapping and it is expected from an object to override this method to provide logical equality.

Eg. A Book object may be logically equal to another copy of same Book, but it is a different object which will be false while doing x == y.

436. How can you guarantee that the garbage collection takes place?

No. We cannot guarantee the garbage collection in Java. Java documentation explicitly says that GarbageCollection is not guaranteed.

You can call System.gc() to request garbage collection, however, that's what it is - a request. It is upto GC's discretion to run.

437. What is the relation between x.hashCode() method and x.equals(y) method of Object class?

x.hashCode() method returns an int hash value corresponding to an object instance.

It is used in hashCode based collection classes like Hashtable, HashMap, LinkedHashMap etc.

hashCode() method is also related to equals() method.

As per Java specification, two objects which are equal to each other using equals() method must have same hash code.

Therefore, two objects with same hashCode may or may not be equal to each other. But two equal objects should have same hash code.

438. What is a compile time constant in Java?

A compile time constant is public static final variable. The public modifier is optional here. At compile time, they are replaced with actual values because compiler knows their value up-front and it also knows that it cannot be changed during run-time. So they are constants.

439. Explain the difference between fail-fast and fail-safe iterators?

The main difference between fail-fast and fail-safe iterators is whether or not the collection can be modified while it is being iterated.

Fail-safe iterators allow modification of collection in an iteration task. But fail-fast iterators do not allow any modification to collection during iteration.

During iteration, fail-fast iterators fail as soon as they realize that the collection has been modified. Modification can be addition, removal or update of a member. And it will throw a ConcurrentModificationException.

Eg. ArrayList, HashSet, and HashMap are fail-fast.

Fail-safe iterators operate on a copy of the collection. Therefore they do not throw an exception if the collection is modified during iteration.
Eg. ConcurrentHashMap, CopyOnWriteArrayList are fail-safe.

440. You have a character array and a String. Which one is more secure to store sensitive data (like password, date of birth, etc.)?

Short answer is, it is safe to store sensitive information in character array.

In Java, String is immutable and it is stored in the String pool. Once a String is created, it stays in the pool in memory until it is garbage collected. You have no control on garbage collection. Therefore, anyone having access to a memory dump can potentially extract the sensitive data and use it.

Whereas, if you use a mutable object like a character array, to store the value, you can set it to blank once you are done with it. Once it is made blank it cannot be used by anyone else.

441. Why do you use volatile keyword in Java?

The volatile keyword guarantees global ordering on reads and writes to a variable. This implies that every thread accessing a volatile field will read the variable's current value instead of using a cached value.

By marking the variable volatile, the value of a variable is never cached thread-locally. All reads and writes will go straight to main memory of Java.

442. What is the difference between poll() and remove() methods of Queue in Java?

It is a basic question to know the understanding of Queue data structure. Both poll() and remove() methods remove and return the head of the Queue.

When Queue is empty, poll() method fails and it returns null, but remove() method fails and throws Exception.

443. Can you catch an exception thrown by another thread in Java?

Yes, it can be done by using Thread.UncaughtExceptionHandler.

Java Documentation says “When a thread is about to terminate due
to an uncaught exception the Java Virtual Machine will query the
thread for
its UncaughtExceptionHandler usingThread.getUncaughtExceptionHandler() will invoke the handler's uncaughtException method, passing the

thread and the exception as arguments.”

444. How do you decide which type of Inner Class – Static or Non-Static to use in Java?

An inner class has full access to the fields and methods of the enclosing class. This is convenient for event handlers, but comes at a cost. Every instance of an inner class retains and requires a reference to its enclosing class.

Due to this cost, there are many situations where static nested classes are preferred over inner classes. When instances of the nested class outlive instances of the enclosing class, the nested class should be static to prevent memory leaks.

At times, due to their “hidden” reference to enclosing class, Inner classes are harder to construct via reflection.

445. What are the different types of Classloaders in Java?

Java Classloader is the part of the Java Runtime Environment (JRE) that loads classes on demand into Java Virtual Machine (JVM).

When the JVM is started, three types of class loaders are used:

1. Bootstrap Classloader: It loads core java API file rt.jar classes from folder.
2. Extension Classloader: It loads jar files from lib/ext folder.
3. System/Application Classloader: It loads jar files from path specified in the CLASSPATH environment variable.

Classes may be loaded from the local file system, a remote file system, or even the web.

446. What are the situations in which you choose HashSet or TreeSet?

HashSet is better than TressSet in almost every way. It gives O(1) for add(), remove() and contains() operations. Whereas, TressSet gives O(log(N)) for these operations.

Still, TreeSet is useful when you wish to maintain order over the inserted elements or query for a range of elements within the set.

We should use TreeSet when we want to maintain order. Or when there are enough read operations to offset the increased cost of write operations.

447. What is the use of method references in Java?

Java 8 has introduced Method references. It allows constructors and methods to be used as lambdas.

The main uses of Method reference are to improve code organization, clarity and terseness.

448. Do you think Java Enums are more powerful than integer constants?

Yes. Java Enums provide many features that integer constants cannot. Enums can be considered as final classes with a fixed number of instances. Enums can implement interfaces but cannot extend another class.

While implementing the strategy pattern, we can use this feature of Enums. Especially, when the number of strategies is fixed.

You can also attach meta-data to enum values in Java. Also enum values are typesafe, where as integer constants are not.

You can also define custom behavior in enum values.

449. Why do we use static initializers in Java?

In Java, a static initializer can run code during the initial loading of a class and it guarantees that this code will only run once. Also the static code will finish running before a class can be accessed in any way.

Initializing static members from constructors is more work. You have to make sure that every constructor does this. You need to maintain a flag to mark the static work when it is done. You may have to think about synchronization or races conditions for work in static block not initialized from static context.

450. Your client is complaining that your code is throwing NoClassDefFoundError or NoSuchMethodError, even though you are able to compile your code without error and method exists in your code. What could be the reason behind this?

Sometimes we upgrade our libraries even with same method name. But we forget to let the client know about the new version. Due this different in version, we get NoClassDefFoundError or NoSuchMethodError at runtime when one library was not compatible with such an upgrade.

Java build tools and IDEs can also produce dependency reports that tell you which libraries depend on that JAR. Mostly, identifying and upgrading the library that depends on the older JAR resolve the issue.

451. How can you check if a String is a number by using regular expression?

Regex is a powerful tool for matching patterns and searching patterns.

A numeric String can only contain digits i.e. 0 to 9. It can also contain + and - sign at start of the String. We can create a regular expression for these two rules. One simple example is as follows:

Pattern pattern = Pattern.compile(".*\\D.*");

452. What is the difference between the expressions String s = "Temporary" and String s = new String("Temporary ")? Which one is better and more efficient?

In general, String s = " Temporary " is more efficient to use than String s = new String("Temporary ").

In case of String s = " Temporary ", a String with the value “Temporary” is created in String pool. If another String with the same value is created (e.g., String s2 = " Temporary "), it will reference the same object in the String pool.

But, when you use String s = new String("Temporary "), Java creates a String with the value “Temporary” in the String pool. Also, that String object is then passed to the constructor of the String Object i.e. new String("Temporary "). And this call creates another String object (not in the String pool) with that value.

Therefore, each such call creates an additional String object. E.g. String s2 = new String("Temporary ") creates an extra String object, rather than just reusing the same String object from the String pool.

So String s = “Temporary” is always an efficient way.

453. In Java, can two equal objects have the different hash code?

No. It is not possible for two equal objects to have different hashcode. But two objects with same hashcode may or may not be equal.

454. How can we print an Array in Java?

We can print an array by using methods of Arrays class. We can either use Arrays.toString() method or we can use Arrays.deepToString() method.

Since array doesn't implement toString() method by itself, just passing an array to System.out.println() will not print its contents. But we can use Arrays.toString() to print each element of an array.

455. Is it ok to use random numbers in the implementation of hashcode() method in Java?

No. The hashcode of an object should be always same. If you use random number in hashcode() method, then you may get a different value of hashcode for same object. This will break the hashcode contract.

456. Between two types of dependency injections, constructor injection and setter dependency injection, which one is better?

Constructor injection guarantees that a class will be initialized with all its dependencies during creation. But setter injection provides flexibility to set an optional dependency.

If we are using an XML file to describe dependencies, the setter injection is more readable.

In general, it is a good practice to use constructor injection for mandatory dependencies and use setter injection for optional dependencies.

457. What is the difference between DOM and SAX parser in Java?

In Java, Document Object Model (DOM) parser loads the whole XML into memory and creates a tree based on DOM model. This helps it in quickly locating the nodes, and making a change in the structure of XML.

On the other hand, Simple API for XML (SAX) parser is an event based parser. It doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and is not suitable to parse large XML files.

458. Between Enumeration and Iterator, which one has better performance in Java?

Enumeration interface is a read-only interface. It has better performance than Iterator. It is almost twice as fast as compared to an Iterator. It also uses very less memory. Also Enumeration does not have remove() method.

On the other hand, Iterator interface is safer than Enumeration, since it can check whether a collection is modified or not during iteration. If a collection is altered while an Iterator is iterating, then it throws ConcurrentModificationException.

459. What is the difference between pass by reference and pass by value?

Whenever an object is passed by value, it means that a copy of the object is passed. Even if changes are made to that object, it doesn’t affect the original value.

Whenever an object is passed by reference, it means that the actual object is not passed, rather a reference of the object is passed. Therefore, any changes made by an external method, are also reflected in the actual object and its reference.

460. What are the different ways to sort a collection in Java?

The most popular way to sort a collection in Java is by calling Collections.sort() method. You can provide your custom Comparator to sort() method for sorting the data in your custom way.

The other way is to use a Sorted collection like TreeSet or TreeMap that stores the information in a sorted order and then you can convert it to a List.

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

Collection interface just specifies groups of objects known as elements. Each concrete implementation of a Collection can choose its own way of how to maintain and order its elements.

Some collections may allow duplicate keys, while other collections may not.

A lot of collection implementations have clone method. But many do not. It is not worthwhile to include it in all, since Collection is an abstract representation. What matters is the concrete implementation.

Cloning and serialization come into picture while doing concrete implementation. Therefore, the concrete implementations of collections should decide how they can be cloned or serialized.

462. What is the difference between a process and a thread in Java?

A process is simply an execution of a program.
A Thread is a single execution sequence within a process.

A process may contain multiple threads. A Thread is also called as a lightweight process.

463. What are the benefits of using an unordered array over an ordered array?

In an ordered array the search time has time complexity of O(log n). Whereas, in an unordered array, search time complexity is O (n).

In an ordered array, the insert operation has a time complexity of O(n). Whereas, the insertion operation for an unordered array takes constant time of O(1).

Therefore, when we have more writes than reads, it is preferable to use an unordered array.

464. Between HashSet and TreeSet collections in Java, which one is better?

A HashSet is Implemented using a HashTable. Therefore, its elements are stored in a random order. The add(), remove(), and contains() methods of a HashSet have constant time complexity O(1).

A TreeSet is implemented using a tree data structure. The elements in a TreeSet are sorted in a natural order. Therefore, add(), remove(), and contains() methods have time complexity of O(logn).

So from performance perspective, HashSet has better performance than TreeSet. But if you want to store elements in a natural sorting order, then TreeSet is a better collection.

465. When does JVM call the finalize() method?

JVM instructs the Garbage Collector to call the finalize method, just before releasing an object from the memory. A programmer can implement finalize() method to explicitly release the resources held by the object. This will help in better memory management and avoid any memory leaks.

466. When would you use Serial Garabage collector or Throughput Garbage collector in Java?

The Serial Garbage collector is used for small applications that require heap memory upto 100 MB.

The Throughput Garbage collector is used in medium to large size Java applications.

467. In Java, if you set an object reference to null, will the Garbage Collector immediately free the memory held by that object?

No. JVM decides to run the Garbage Collector whenever it is low on memory. When Garbage Collector runs, it looks for objects that are available for garbage collection and then frees the memory associated with this object.

So just setting an Object reference null makes it eligible for Garbage Collection, but it does not immediately free the memory.

468. How can you make an Object eligible for Garbage collection in Java?

To make an Object eligible for Garbage collection, just make sure that it is unreachable to the program in which it is currently defined / created / used. You can set the object reference to null and make sure no other object refers it. Once the object cannot be reached, Garbage Collection can clean it during the next run.

469. When do you use Exception or

Error in Java? What is the difference

between these two?

Throwable class is the superclass of Exception and Error classes in Java.

When you want to catch the exceptional conditions that your program can create or encounter, then use the Exception class or subclass of Exception.

When you come across situations that are unexpected then use Error class in Java. Also recovering from Error is not possible in most of cases. So it is better to terminate the program.

470. What is the advantage of PreparedStatement over Statement class in Java?

PreparedStatements are precompiled statements for database queries. Due to this their performance is much better. Also, we can reuse PreparedStatement objects with different input values to the same query.

Where as, Statement class does not provide these features.

471. In Java, what is the difference between throw and throws keywords?

When we want to raise an exception in our code, we use the throw keyword with the name of the exception to be raised.

Where as, throws keyword is used in method declaration. Throws keyword tells us the Exception that can be thrown by this method. Any caller of this method should be prepared to expect this Exception.

Another minor difference is that throw is used only with one exception, but throws can be used with comma-separated list of multiple exceptions.

472. What happens to the Exception object after the exception handling is done?

Once the exception handling is complete, the Exception object is not reachable. Then it is garbage collected in the next run of Garbage Collector.

473. How do you find which client machine is sending request to your servlet in Java?

We can use the ServletRequest class to find the IP address or host name of the client machine.

There are methods getRemoteAddr() to get the IP address of the client machine and getRemoteHost() to get the host name of the client machine.

474. What is the difference between a Cookie and a Session object in Java?

Both Cookie and Session are used during communication between Client and Server. The Client can disable a Cookie. Due to which the Web server cannot send a cookie. But a client cannot disable a session. So a Session always works irrespective of any setting at the client side.

Also a Session can store any Java object. But the Cookie can only store small information in a String object.

475. Which protocol does Browser and Servlet use to communicate with each other?

HTTP protocol. The Browser and Servlet communicate with each other by using the HTTP protocol.

476. What is HTTP Tunneling?

There are many network communication protocols on the Internet. But HTTP is the most popular among them. HTTP Tunneling is a technique in which HTTP or HTTPS protocol encapsulated the communication done by any other type of protocol. The masking of other protocol requests as HTTP requests is known as HTTP Tunneling.

477. Why do we use JSP instead of Servlet in Java?

Since JSP pages are dynamically compiled into servlets, the programmers can easily make updates to the presentation layer code.

For better performance, JSP pages can be pre-compiled.

Also JSP pages provide flexibility to combine static templates like HTML or XML snippets.

In addition, programmers can make logic changes at the class level, without editing the JSP pages that use the class logic.

478. Is empty '.java' file name a valid source file name in Java?

Yes. You can create a class and store it in a file with name .java. You can try it yourself, by creating, compiling and running such a file. It will run correctly.

479. How do you implement Servlet Chaining in Java?

To implement, Servlet Chaining, there has to be more than one servlet. The output of one servlet has to be sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. In this way, a chain of servlets is formed to complete a task.

The last servlet in the chain will be responsible for sending final response to client.

480. Can you instantiate this class?

public class A {

A a = new A(); }

No, this class cannot be instantiated, since it will result in recursively calling its constructor.

481. Why Java does not support operator overloading?

Java supports Method overloading but does not support operator overloading. It would make the design more complex by adding operator loading. Also it will make more complex compiler.

One more reason is that, it will reduce the performance of JVM by operator overloading, since JCM has to do extra work to find the real meaning of overloaded operators at run time.

482. Why String class is Immutable or Final in Java?

Since String objects are cached in a String pool, it makes sense to make the String immutable. The cached String literals are shared between multiple clients. And there is a possibility that one client's action may affect another client's access to String pool.

String is also used as a parameter in many Java classes. Eg. You can pass hostname, port number as String while opening a network connection. If any one can modify your copy of the String, it can change the hostname. Due to this reason, it makes sense to make String final as soon as it is created.

483. What is the difference between sendRedirect and forward methods?

When you use sendRedirect method, it creates a new request. When you use the forward method, it just forwards a request to a new target.

In case of sendRedirect, the previous request scope objects are not available, because it creates a new request.

In case of forward method, the previous request scope objects are available after forwarding.

Also the sendRedirect method is considered slower than the forward method.

484. How do you fix your Serializable class, if it contains a member that is not serializable?

If you want to make a class Serializable, but find that this class contains members that are not Serializable, then you have to mark those members as transient. This will ensure that this member is not persisted to a stream of bytes during Serialization.

Therefore, Transient keyword of Java comes to help in this scenario.

485. What is the use of run time polymorphism in Java?

During the run time the behavior of an Object can change based on its run time state. Due to this run time polymorphism is introduced in Java. If you override a method in a child class, then you are providing run time polymorphism. Nothing will happen at the compile time. But at the run time, JVM decides which method will be called based on the class of the Object.

486. What are the rules of method overloading and method overriding in Java?

When we want to overload a method, we need to make sure that the method name remains same. But method signature can vary in the number or datatype of arguments or in the order of arguments.

When we want to override a method, we ensure that the method is not throwing checked exceptions that are new or higher than those declared by the overridden method. Also we make sure that the method name, arguments and return type remain the same.

Also we cannot override Static and Final methods in Java.

487. What is the difference between a class and an object in Java?

A Class is a template or a blue print of an Object to be created. An Object is an instance of a Class. A Class defines the methods and member variables. But an Object populates the values of the member variables.

Therefore a class is a blueprint that you use to create objects. An object is an instance of a class – it is a concrete 'thing' that you made using a specific class.

Most of the OOPS concepts are valid only when an Object is created.

488. Can we create an abstract class that extends another abstract class?

Yes. An abstract class can extend another abstract class. It does not need to define the methods of parent abstract class. Only the last non-abstract class has to define the abstract methods of a parent abstract class.

489. Why do you use Upcasting or Downcasting in Java ?

When we want to cast a Sub class to Super class, we use Upcasting. It is also known as widening. Upcasting is always allowed in Java.

When we want to cast a Super class to Sub class, we use Downcasting. It is also known as narrowing.

At times, Downcasting can throw the ClassCastException if it fails the type check.

490. What is the reason to organize classes and interfaces in a package in Java?

As the name suggests, a package contains a collection of classes. It helps in setting the category of a file. Like- whether it is a Data Access Object (DAO) or an API.

It helps in preventing the collision of Name space.

Also we can introduce access restriction by using package and the right modifiers on a class and its methods.

491. What is information hiding in Java?

Information hiding is OOPS concept. In Java you can use encapsulation to do Information hiding. An object can use the access modifiers like-public, private, protected to hide its internal details from another object. This helps in decoupling the internal logic of an object from outside world.

By using Information hiding, an object can change its internal implementation without impacting the outside calling client’s code.

492. Why does Java provide default constructor?

In Java all the interaction takes place between Object instances. To create an Object instance, JVM needs a constructor. Java does not enforce the rule on a programmer to define a default constructor for every class.

Whenever an object has to be created and programmer has not provided a constructor, Java uses default constructor to create the object. Default constructor also initializes member variables with their default values.

493. What is the difference between super and this keywords in Java?

We use super keyword to access the methods of the super class from child class.

We use this keyword to access methods of the same class.

494. What is the advantage of using Unicode characters in Java?

Unicode characters have much larger number of characters in the specification.

They also contain Asian and non-western European characters.

Most of the modern technologies, websites and browsers support these Unicode characters.

495. Can you override an overloaded method in Java?

Yes. Java allows to override an overloaded method, if that method is not a static or final method.

496. How can we change the heap size of a JVM?

Java provides the command line parameters to set the heap size for JVM.
You can specify the values in –Xms and –Xmx parameters. These parameters stand for initial and maximum heap size of JVM.

497. Why should you define a default constructor in Java?

In general, Java provides a default constructor with each class. But there are certain cases when we want to define our own version of default constructor.

When we want to construct an object with default values, we create our default constructor.

At times, we can mark the default constructor private. So that any other class cannot create an instance of our class. This technique is generally used in Singleton design pattern.

498. How will you make an Object Immutable in Java?

To make an object immutable follow these two rules. One, do not use any setter methods that can change the fields of your class. Two, make the fields final. By following these rules, the member variables cannot be changed after initialization. This will ensure that member variables of an Object do not change. And thus the Object will be considered Immutable.

499. How can you prevent SQL Injection in Java Code?

In Java, you can use PreparedStatement to prevent SQL injection. In a PreparedStatement you can pass the precompiled SQL queries with pre-defined parameters. This helps in checking the type of parameters to SQL queries. So it protects your code from SQL injection attacks.

500. Which two methods should be always implemented by HashMap key Object?

Any object that we want to use as key for HashMap or in any other hash based collection data structure e.g. Hashtable, or ConcurrentHashMap must implement equals() and hashCode() method.

501.Why an Object used as Key in HashMap should be Immutable?

The Key object should be immutable so that hashCode() method always return the same value for that object.

The Hashcode returned by hashCode() method depends on values of member variables of an object. If an object is mutable, then the member variables can change. Once the member variables change, the Hashcode changes. If the same object returns different hash code at different times, then it is not reliable to be used in the HashMap.

Let say, when you insert the object, the Hashcode is X, the HashMap will store it in bucket X. But when you search for it the Hashcode is Y, then HashMap will look for the object in bucket Y. So you are not getting what you stored.

To solve this, a key object should be immutable.

Although, the compiler does not enforce this rule, a good programmer always remembers this rule.

502. How can we share an object between multiple threads?

There are many ways to share same object between multiple threads. You can use a BlockingQueue to pass an object from one thread to another thread.

You can also use Exchanger class for this purpose. An Exchanger is a bidirectional form of a SynchronousQueue in Java. You can use it to swap the objects as well.

503. How can you determine if your program has a deadlock?

If we suspect that our application is stuck due to a Deadlock, then we just take a thread dump by using the command specific to environment in which your application is running. Eg. In Linux you can use command kill -3.

In case of deadlock, you will see in thread dump the current status and stack trace of threads in the JVM, and one or more of them will be stuck with message deadlock.

Also you can do this programmatically by using the ThreadMXBean class that ships with the JDK.

If you don't need programmatic detection you can do this via JConsole. On the thread tab there is a "detect deadlock" button.