**Java Basics**

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.

1. **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.

1. **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.
7. **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.

1. **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.
   1. 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.

**OOPS**

1. **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
5. **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.

1. **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.

1. **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.

1. **19. Can we inherit a Constructor?**

No, Java does not support inheritance of constructor.

1. **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**

1. **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.

1. **Explain the concept of Inheritance?**

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

It is also represented by IS-A relationship.

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

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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**

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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**

1. **What is the other name of Method Overloading?**

Method Overloading is also known as Static Polymorphism.

1. **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
4. **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
4. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **Is it allowed to override an overloaded method?**

Yes. You can override an overloaded method in Java.

1. **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.
5. **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.

1. **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**

1. **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.

1. **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.

1. **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**

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **Can we instantiate an abstract class in Java?**

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

1. **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.

1. **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.

1. **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.

1. **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.

1. **What can we use instead of Marker interface?**

We can use annotations instead of Marker interface.

1. **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.

1. **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.
5. **Does Java allow us to use private and protected modifiers for variables in interfaces?**

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

1. **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.

1. **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.

1. **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.

1. **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.

1. **Is it allowed to declare main method as final?**

Yes, we can mark the main method as final.

**Package**

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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**

1. **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.

1. **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.

**Reflection**

1. **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.

1. **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.

1. **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);

}

}

1. **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**

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.
3. **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.

1. **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
5. **How can we reference an unreferenced object again?**

We can provide implementation in finalize() method to reference and unreferenced object. For an unreferenced 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.

1. **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.

1. **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.
4. **How can we invoke an external process in Java?**

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

1. **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**

1. **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.

1. **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
5. **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.
4. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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**

1. **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.

1. **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”

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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
7. **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.

1. **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**

1. **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.

1. **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 extends

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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.
3. **Can we re-throw an Exception in Java?**

Yes, Java allows to re-throw an Exception.

1. **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.

1. **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.

1. **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).

1. **Explain exception hierarchy in java**

It’s also the basic Exception handling interview question. Freshers must know about this.

**Exception hierarchy in java >** ****

java.lang.**Object** is superclass of all classes in java.

java.lang.**Throwable** is superclass of java.lang.**Exception** and java.lang.**Error**

java.lang.**Exception** is superclass of java.lang.**RuntimeException, IOException, SQLException,** [**BrokenBarrierException**](http://www.javamadesoeasy.com/2015/03/cyclicbarrier-in-java.html)and many more other classes in java.

java.lang.[**RuntimeException**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)is superclass of java.lang.[**NullPointerException**](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html)**, ArithmeticException** and many more other classesin java.

java.lang.[**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html)is superclass of java.lang.**VirtualMachineError, IOError, AssertionError,** [**ThreadDeath**](http://www.javamadesoeasy.com/2015/04/threaddeath-error-calling-stop-method.html)and many more other classesin java.

java.lang.**VirtualMachineError** is superclass of java.lang.**OutOfMemoryError, StackOverflowError** and many more other classesin java.

**What are differences between checked and unchecked exceptions in java?**

**Answer**. This is very important Exception handling interview question in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***checked exception*** | ***unchecked exception*** |
| 1 | Also known as | **checked** exceptions are also known as **compileTime** exceptions in java. | **unchecked** exceptions are also known as **runtime** exceptions in java. |
| 2 | Should be solved at compile or runtime? | Checked exceptions are those which need to be taken care at compile time in java. | Unchecked exceptions are those which need to be taken care at runtime in java. |
| 3 | Benefit/ Advantage | We cannot proceed until we fix compilation issues which are most likely to happen in program, this helps us in avoiding runtime problems upto lot of extent in java. | Whenever runtime exception occurs execution of program is interrupted, but by handling these kind of exception we avoid such interruptions and end up giving some meaningful message to user in java. |
| 4 | Creating custom/own exception | |  | | --- | | **class** UserException **extends Exception** {     UserException(String s) {  **super**(s);     }  } |   By extending java.lang.**Exception**, we can create checked exception. | |  | | --- | | **class** UserException **extends RuntimeException** {     UserException(String s) {  **super**(s);     }  } |   By extending java.lang.**RuntimeException**, we can create unchecked exception. |
| 5 | [Exception propagation](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) | For **propagating checked** exceptions method must throw exception by using **throws** keyword. | **unchecked** exceptions are [**automatically propagated**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)in java. |
| 6 | handling checked and unchecked exception while overriding superclass method | *If superclass method throws/declare* ***checked******exception*** *>*   * overridden method of subclass **can** declare/**throw** **narrower** (subclass of) **checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_93.html)), or * overridden method of subclass **cannot** declare/**throw** **broader** (superclass of) **checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_94.html)), or * overridden method of subclass **can** declare/**throw any unchecked /RuntimeException** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_37.html)) | *If superclass method throws/declare* ***unchecked*** *>*   * overridden method of subclass **can** declare/**throw any unchecked /RuntimeException (superclass or subclass)** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of.html)), or * overridden method of subclass **cannot** declare/**throw** **any checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_6.html)), |
|  | Which classes are which type of exception?  either  **checked or** **unchecked** exception? | The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions in java. | The class **RuntimeException and all its subclasses** are unchecked exceptions.  Likewise,  The class **Error and all its subclasses** are unchecked exceptions in java. |
| 7 | Most frequently faced exceptions | [SQLException](http://www.javamadesoeasy.com/2015/05/sqlexception-in-java.html),  IOException,  ClassNotFoundException | [NullPointerException](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html),  ArithmeticException ArrayIndexOutOfBoundsException. |

**What are differences between Exception and Error in java?**

**Answer**. It is another very important exception interview question to differentiate between Exception and Error in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | [**Exception**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) | [**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html) |
| 1 | serious problem? | Exception does **not indicate any serious problem**. | Error **indicates some serious problems** that our **application should not try to catch.** |
| 2 | divided into  [**checked** and **unchecked**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) | Exception are divided into **checked** and **unchecked exceptions in java**. | Error are **not divided** further into such classifications in java. |
| 3 | Which classes are which type of exception? either  **checked or** **unchecked** exception? | The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions.  The class **RuntimeException and all its subclasses** are unchecked exceptions.  Likewise,  The class **Error and all its subclasses** are unchecked exceptions in java. | Error and its subclasses are regarded as **unchecked** exceptions in java |
| 4 | Most frequently faced exception and errors | **checked exceptions>**  SQLException,  IOException,  ClassNotFoundException  **unchecked exceptions>**  [NullPointerException](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html), ArithmeticException, | **VirtualMachineError, IOError, AssertionError,** [**ThreadDeath**](http://www.javamadesoeasy.com/2015/04/threaddeath-error-calling-stop-method.html),  **OutOfMemoryError, StackOverflowError.** |
| 5 | Why to catch or not to catch? | Application **must catch** the Exception because they does not cause any major threat to application in java. | Application **must not catch** the Error because they does cause any major threat to application.  Example >  Let’s say errors like OutOfMemoryError and StackOverflowError occur and are caught then JVM might not be able to free up memory for rest of application to execute, so it will be better if application don’t catch these errors and is allowed to terminate in java. |

**What is difference between throw and throws in java?**

**Answer**. This is also another important and frequently asked exception handling interview question. To confuse interviewees Interviewers might give you code snippet and ask you to insert throw or throws keyword in java.

|  |  |  |
| --- | --- | --- |
|  | [**throw**](http://www.javamadesoeasy.com/2015/05/throw-exception-in-java.html) | [**throws**](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html) |
| 1 | **throw** [keyword](http://www.javamadesoeasy.com/2015/05/keywords-in-java-language.html) is used to throw an [exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) explicitly in java. | **throws** keyword is used to declare an exception in java. |
| 2 | **throw** is used **inside method**.  Example in java >  **static** **void** m(){  **throw** **new** FileNotFoundException();  } | **throws** is used **in** **method declaration**.  Example in java >  **static** **void** m() throws FileNotFoundException{  } |
| 3 | **throw** is always **followed by** **instanceof** Exception class in java.  Example >  **throw** **new** FileNotFoundException() | **throws** is always **followed by name of Exception class in java**.  Example >  **throws** FileNotFoundException |
| 4 | **throw** can be used to throw **only one exception at time**.  Example >  **throw** **new** FileNotFoundException() | **throws** can be used to throw **multiple exception at time**.  Example >  **throws** FileNotFoundException, NullPointerException  and many more... |
| 5 | **throw** cannot propagate exception to calling method in java. | **throws** can [propagate exception](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) to calling method.  Please see these programs to understand how exception is propagated to calling method.  [**Program 1**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) - Handling Exception by throwing it from m() method (using throws keyword) and handling it in try-catch block from where call to method m() was made.  [**Program 2**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)- Throwing Exception from m() method and then again throwing it from calling method [ i.e. main method] |

**How to create user defined checked and unchecked Exception in java?**

**Answer**. Very important exception handling interview question. Interviewers generally expects interviewees  to write code to create checked and unchecked Exception in java.

*Creating user defined* [***checked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception in java >*

|  |
| --- |
| **class** UserDefinedException **extends Exception** {     UserDefinedException(String s) {  **super**(s);     }  } |

By extending java.lang.**Exception**, we can create checked exception.

*Creating user defined* [***unchecked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception in java >*

|  |
| --- |
| **class** UserDefinedException **extends RuntimeException** {     UserDefinedException(String s) {  **super**(s);     }  } |

By extending java.lang.**RuntimeException**, we can create unchecked exception.

**Explain try-with-resource in java?**

**Answer**. Again experienced java developers must be well versed with this exception interview question. **Before java 7**, we used to write **explicit code for closing file in** [**finally**](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) **block by using** [**try-finally block**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html)like this >

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal [JavaMadeSoEasy.com](http://javamadesoeasy.com/) \*/  **public** **class** TryWithResourseTest {  **public** **static** **void** main(String[] args) **throws** IOException {            InputStream inputStream = **null**;  **try**{                   inputStream = **new** FileInputStream("c:/txtFile.txt");                   //code......            }**finally**{  **if**(inputStream!=**null**)  **inputStream.close();**            }     }  } |

**In java 7**, using **Try-with-resources >**

* we need not to write **explicit code for closing file**.

|  |
| --- |
| **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  /\*\* Copyright (c), AnkitMittal [JavaMadeSoEasy.com](http://javamadesoeasy.com/) \*/  **public** **class** TryWithResourseTest {  **public** **static** **void** main(String[] args) **throws** IOException {  **try (InputStream inputStream = new FileInputStream("c:/txtFile.txt"))** {               //code...           }    }  } |

*Using multiple resources inside* ***Try-with-resources is also allowed in java.***

**What will happen when catch and finally block both return value, also when try and finally both return value in java?**

**Answer**. This is very important exception handling interview question for experienced developers.

When **catch and finally block** both return value, **method will ultimately return value returned by** [**finally**](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) block irrespective of value returned by [catch](http://www.javamadesoeasy.com/2015/05/catch-block-and-automatic-resource.html) block.

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal [JavaMadeSoEasy.com](http://javamadesoeasy.com/) \*/  **public** **class** ExceptionTest {  **public** **static** **void** main(String[] args) {            System.*out*.println("method return -> "+*m*());     }    **static** String m(){  **try**{  **int** i=10/0; //will throw ArithmeticException            }**catch**(ArithmeticException e){  **return** "catch";            }**finally**{  **return** "finally";            }       }  }  /\*OUTPUT  method return -> finally  \*/ |

In above program, i=10/0 will throw ArithmeticException and enter catch block to return "catch", but ultimately control will enter finally block to return "finally".

Likewise, when [**try and finally**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) **block** both return value, **method will ultimately return value returned by finally block** irrespective of value returned by try block. For program [please refer](http://www.javamadesoeasy.com/2015/05/what-will-happen-when-catch-and-finally.html).

**What is exception propagation in java?**

**Answer**.

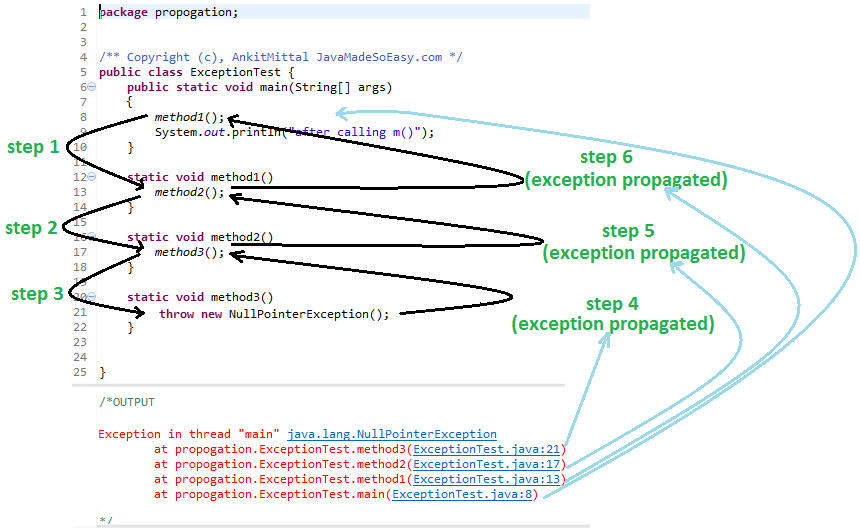
Experienced developers must know in detail about Exception handling interview question in java. Even freshers must try and understand this in depth concept of exception propagation in java.

Whenever methods are called [stack](http://javamadesoeasy.com/2015/01/stacks.html) is formed and an exception is first thrown from the top of the stack and if it is not caught, it starts coming down the stack to previous methods until it is not caught.

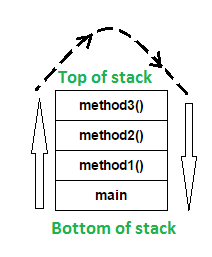
If exception remains uncaught even after reaching bottom of the stack it is propagated to JVM and program is terminated in java.

*Propagating* [***unchecked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception (NullPointerException) >*

**unchecked** exceptions are **automatically propagated** in java.

[](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)

[***stack***](http://javamadesoeasy.com/2015/01/stacks.html)*of methods is formed >*

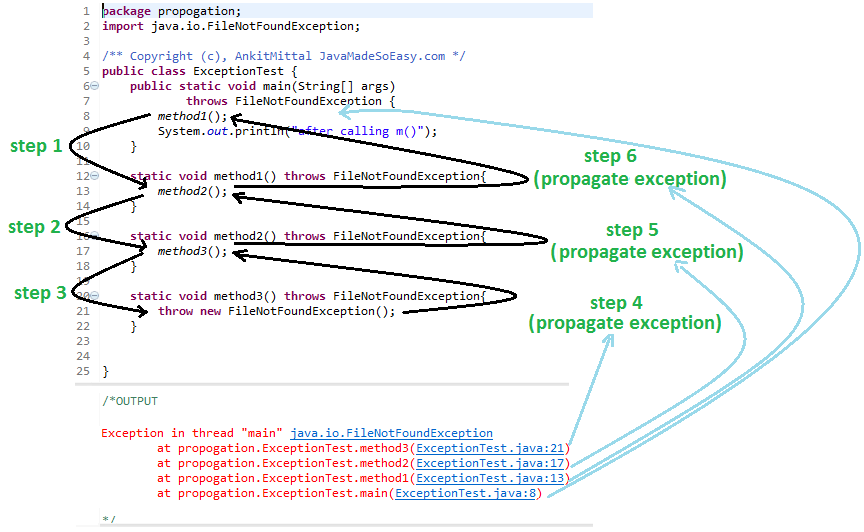


In the above program, stack is formed and an exception is first thrown from the top of the stack [ **method3()** ] and it remains uncaught there, and starts coming down the stack to previous methods to **method2()**,then to **method1()**,than to **main()** and it remains uncaught throughout.

exception remains uncaught even after reaching bottom of the stack [ **main()** ] so it is propagated to JVM and ultimately program is terminated by throwing exception [ as shown in output ] in java.

*Propagating* ***checked*** *exception (FileNotFoundException) using throws keyword >*

For **propagating checked** exceptions method must throw exception by using [**throws**](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html)keyword.



**What is Difference between** [**multiple catch block**](http://www.javamadesoeasy.com/2015/05/multiple-catch-block-in-java.html) **and** [**multi catch syntax**](http://www.javamadesoeasy.com/2015/05/catch-block-and-automatic-resource.html)**?**

**Answer**. Experienced developers must know in detail about this Exception handling interview question in java

|  |  |  |
| --- | --- | --- |
|  | **multiple catch block** | **multi catch syntax** |
| 1 | multiple catch blocks were introduced in prior versions of Java 7 and does not provide any automatic resource management in java. | **multi catch syntax was introduced in** java 7 for improvements in multiple exception handling which helps in **automatic resource management in java.** |
| 2 | Here is the syntax for writing **multiple catch block in java** >   |  | | --- | | **try**{  //code . . . . .  }**catch**(**IOException** ex1){  //code . . . . .  } **catch**(**SQLException** ex2){  //code . . . . .  } | | Here is the **multi catch syntax in java** >   |  | | --- | | **try**{  //code . . . . .  }**catch**(IOException **|** SQLException ex){  //code . . . . .  } |   We could separate different exceptions using **pipe** ( **|** ) |
| 3 | For catching IOException and SQLException we need to write **two catch block** like this > | with the help of multi catch syntax we can catch IOException and SQLException in one catch block using **multi catch syntax** like this > |
| 4 | **When multiple catch blocks** are used , first catch block could be subclass of Exception class handled in following catch blocks like this >  IOException is subclass of Exception in java. | If **Multi catch syntax** is used to catch subclass and its superclass than compilation error will be thrown.  IOException and Exception in **multi catch syntax** will cause compilation error “The exception **IOException** is already caught by the alternative **Exception**”.    **Solution >**  We must use only **Exception** to catch its subclass like this > |
| 5 | Does not provide such features. | *Features of* ***multi catch syntax*** *>*   * Has **improved way of catching multiple** [**exceptions**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html)**.** * This syntax does **not looks clumsy**. * **Reduces developer efforts** of writing multiple catch blocks. * Allows us to **catch more than one exception in one catch block**. * Helps in **automatic resource management.** |

**Difference between Final, Finally and Finalize in java?**

**Answer**. It is another very very important exception interview question to differentiate between final, finally and finalize in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | [***final***](http://www.javamadesoeasy.com/2015/05/final-keyword-in-java-20-salient.html) | [***finally***](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) | [***finalize***](http://www.javamadesoeasy.com/2015/05/finalize-method-in-java-10-salient.html) |
| **1** | [final](http://www.javamadesoeasy.com/2015/05/final-keyword-in-java-20-salient.html) can be applied to **variable**, **method** and **class** in java. | [finally](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) is a block. | [finalize](http://www.javamadesoeasy.com/2015/05/finalize-method-in-java-10-salient.html) is a method. |
| **2** | ***2.1) Final variable***  **final memberVariable**  **final local variable**  **final static variable**  **Final memberVariable** of class must be initialized at time of declaration, once initialized final memberVariable cannot be assigned a new value.  Final variables are called **constants** in java.   |  | | --- | | **class** FinalTest {  **final** **int** x=1; //memberVariable/instanceVariable  } |   If constructor is defined then final memberVariable can be initialized in constructor but  once initialized cannot be assigned a new value.   |  | | --- | | **class** FinalTest {  **final** **int** x; //memberVariable/instanceVariable     FinalTest() {            x = 1; //final memberVariable can be initialized in constructor.     }  } |   **Final local variable** can be left uninitialized at time of declaration and can be initialized later, but once initialized cannot be assigned a new value in java.   |  | | --- | | **class** FinalTest {  **void** method(){  **final** **int** x; //uninitialized at time of declaration        x=1;    }  } |   **Final static variable** of class must be initialized at time of declaration or can be initialized in static block, once initialized final static variable cannot be assigned a new value.  If static block is defined then final static variable can be initialized in static block, once initialized final static variable cannot be assigned a new value.   |  | | --- | | **class** FinalTest {  **final** **static** **int** *x*; //static variable  **static**{ //static block  *x*=1;   }  } |   ***2.2) Final method***  **Final method** cannot be overridden, any attempt to do so will cause compilation error.    Runtime polymorphism is not applicable on final methods because they cannot be inherited.  ***2.3) Final class***  **Final class** cannot be extended, any attempt to do so will cause compilation error. | *try or* [*try-catch*](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) *block can be followed by finally block >*  try**-finally** block, or   |  | | --- | | **try{**  **//Code to be enclosed in try-finally block**  **}finally{**  **}** |   try-catch**-finally** block.   |  | | --- | | **try{**  **//Code to be enclosed in try-catch-finally block**  **}catch(Exception e){**  **}finally{**  **}** |   ***finally block*** *can can only exist if try or try-catch block is there, finally block can’t be used alone in java.*  ***finally block is not executed in following scenarios >***  finally is not executed when **System.exit** is called.  if in case **JVM crashes** because of some java.util.[**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html). | finalize method is called before garbage collection by JVM,  finalize method is called for any cleanup action that may be required before garbage collection.   |  | | --- | | /\*\* Copyright (c), AnkitMittal [JavaMadeSoEasy.com](http://javamadesoeasy.com/) \*/  @Override  **protected void finalize() throws Throwable** {  **try** {        System.*out*.println("in   finalize() method, "                                +   "doing cleanup activity");    } **catch** (Throwable throwable) {  **throw** throwable;  }  } |   finalize() method is defined **in java.lang.Object** |
| **3** | - | finally block can only exist if try or try-catch block is there, finally block can’t be used alone in java. | We can *force early garbage collection in java* by using following methods >  **System.*gc*(); Runtime.*getRuntime*().gc();**  **System.*runFinalization*(); Runtime.*getRuntime*().runFinalization();** |
| **4** | - | finally is always executed irrespective of exception thrown in java. | If any uncaught [exception](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) is thrown inside finalize method -  **exception is ignored,**  **thread is terminated and**  **object is discarded.**  **Note :** Any exception thrown by the finalize method causes the finalization of this object to be halted, but is otherwise ignored. |
| **5** | - | Currently executing thread calls finally method in java. | JVM does not guarantee which [***daemon***](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html)[thread](http://www.javamadesoeasy.com/2015/03/what-is-thread-in-java.html) will invoke the finalize method for an object. |
| **6** | final is a keyword in java. | finally Is a keyword in java. | finalize is not a keyword in java. |

**What are the differences between** [**between ClassNotFoundException and NoClassDefFoundError in java ?**](http://www.javamadesoeasy.com/2015/12/what-is-difference-between.html)

**Answer**.

|  |  |  |
| --- | --- | --- |
|  | ***ClassNotFoundException*** | ***NoClassDefFoundError*** |
| 1 | ClassNotFoundException is [Checked (compile time) **Exception** in java.](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) | NoClassDefFoundError is a [Error](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html) in java**.** Error and its subclasses are regarded as **unchecked** exceptions in java. |
| 2 | Here is the **hierarchy** of java.lang.ClassNotFoundException -  -java.lang.Object  -java.lang.Throwable   -java.lang.[Exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html)    -java.lang.ReflectiveOperationException     -java.lang.ClassNotFoundException | Here is the **hierarchy** of java.lang.NoClassDefFoundError -  -java.lang.Object  -java.lang.Throwable   -java.lang.[Error](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html)    -java.lang.LinkageError     -java.lang.NoClassDefFoundError |
| 3 | **ClassNotFoundException** is thrown when JVM tries to class from classpath but it does not find that class. | **NoClassDefFoundError** is thrown when JVM tries to load class which >   * **was NOT** available at **runtime** but * **was** available at **compile** time. |
|  | **ExceptionInInitializerError** has got nothing to do with **ClassNotFoundException**. | You must ensure that class does not throws **java.lang.ExceptionInInitializerError** because that is likely to be followed by **NoClassDefFoundError**. |

**Java Collection**

1. **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.

1. **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.
5. **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.

1. **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.
4. **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



1. **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.

1. **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);

}

1. **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);

}

1. **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.

1. **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);

1. **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);

1. **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.

1. **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.
3. **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.
4. **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.

1. **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.
4. **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.

1. **In Java, how will you decide when to use a List, Set or a Map collection?**
2. If we want a Collection that does not store duplicate values, then we use a Set based collection.
3. If we want to frequently access elements operations based on an index value then we use a List based collection. E.g. ArrayList
4. If we want to maintain the insertion order of elements in a collection then we use a List based collection.
5. For fast search operation based on a key, value pair, we use a HashMap based collection.
6. If we want to maintain the elements in a sorted order, then we use a TreeSet based collection.
7. **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.

1. **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() , pollLastEntry() , tailMap() , firstKey() , lastKey() etc. that are not provided by a HashMap.
9. **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.
7. **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.

1. **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.

1. **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.

1. **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 behaviour 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.

1. **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.

1. **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);

Option 2: Use Collection.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.

1. **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

**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

1. **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.

1. 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.
2. **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.
6. **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 baseddynamic array. A LinkedList is a Doubly Linked List data structure.
2. **Insertion**: It is easier to insert new elements in aLinkedList, 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 inremoval of elements than ArrayList.
4. **Memory Usage**: LinkedList uses more memory thanArrayList, 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.
6. **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 insertingduplicate 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 nullvalue. 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.
4. **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.

1. **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.

1. **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.

1. **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. AHashMap 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 storingduplicate 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 nullvalue. In a HashMap we can store single null key, but any number of null values.
5. **Element Type**: A HashSet contains only values of objectsas 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.
7. **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 notsynchronized 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 theorder of elements. These are unordered collections.
3. **Internal Implementation**: A HashMap backs up a HashSetinternally. So HashSet uses a HashMap for performing its operations.
4. **Performance**: Both of these collections provide constanttime performance for basic operations such as insertion and removal of elements.
5. **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.

1. **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.

1. **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.

1. **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



1. **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 .

1. **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.

1. **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. **remove() method**: The main difference betweenEnumeration 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 arehasNext(), next(), remove(). Names of methods in

Enumeration interface are hasMoreElements(), nextElement().

1. **Legacy Interface**: Enumeration is considered as a legacyinterface. 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.
2. **Fail-fast vs. Fail-safe**: Iterator is based on fail-fastprinciple. 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.
3. **Safety**: Since Iterator is fail-fast and does not allowmodification of a collection by other threads, it is considered safer than Enumeration.
4. **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.

1. **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.

1. **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);

1. **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);

1. **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.

1. **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 thesize 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 differentperformance for different operations.
3. **add() or get():** Adding an element to or retrieving anelement from an array or ArrayList object has similar performance. These are constant time operations.
4. **resize():** Automatic resize of ArrayList slows down theperformance. 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 aswell as objects. But ArrayList cannot contain primitive data types. It contains only objects.
6. **Iterator**: In an ArrayList we use an Iterator object totraverse the elements. We use for loop for iterating elements in an array.
7. **Type Safety**: Java helps in ensuring Type Safety ofelements 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.

1. **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.
2. **Adding elements**: In an ArrayList we can use add()method to add objects. In an Array assignment operator is used for adding elements.
3. Multi-dimension: An Array can be multi-dimensional. An ArrayList is always of single dimension.
4. **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

* HashMap.
  1. **Retrieve**: We use get() method to retrieve elements from aHashMap.

Value get(Object key)

* 1. **Insert**: We use put() method to insert a key value pair in aHashMap.

Value put(Key k, Value v)

* 1. **Delete**: We use remove() method to delete key-value pairfrom the HashMap.

Value remove(Object key)

1. **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 aConcurrentHashMap is a synchronized object.
2. **Null Key**: A HashMap can have one null key and anynumber of null values. A ConcurrentHashMap cannot have null keys or null values.
3. **Multi-threading**: A ConcurrentHashMap works well in amulti-threading environment.
4. **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.

1. **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.

1. **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 iteratethe objects of the list.
2. **For-each loop**: We can use for-each loop to traverse allthe elements of a list.
3. **What is CopyOnWriteArrayList? How it is 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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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 mustcome from a single enum type when the set is created.
3. **Bit vector**: Internally, EnumSet is represented as bitvector.
4. **Iterator**: The iterator of EnumSet traverses the elements intheir natural order. (It is the order in which the enum constants are declared).
5. **Null**: In an EnumSet, null elements are not permitted. If wetry 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.
8. **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



1. **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)

1. **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).

1. **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.

1. The node table should not considered distinct objects as equal even if they happen to be equal.
2. 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.
3. **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.

1. **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.

1. **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.

1. **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)



1. **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.

1. **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.

1. **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.

1. **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.

1. **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 canuse following code:

myMap = Collections.emptyMap();

1. **Any map**: For all other scenarios, we can use followingcode by using new method:

myMap = new HashMap();

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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());

1. **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.

1. **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.

1. **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.

1. **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.

1. **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 withgeneric 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 toavoid ClassCastException at runtime.
3. **Collections**: It is recommended to use Collections utilityclass for algorithms and various other common methods for Collections.
4. **Right Type**: We have to choose the right type of Javacollection 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.

1. **Initial Size**: In some collection classes we can specify theinitial 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.
2. **Map**: We should use immutable classes provided by Javaas key elements in a Map.
3. **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.

1. **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.

1. **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.

1. **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().

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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 ofDeque. 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 ofDeque interface in Java. It is also not synchronized, so it is not thread-safe. It mainly provides functionality of a doubly linked list.