Java Collection Latest Interview Question Updated Regularly

Last Updated On: October 27, 2019 By Softwaretestingo Admin

Java Collection Interview Question

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

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

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

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

Q: Describe synchronisation with respect to multithreading.

Ans: On multithreading, synchronisation is the capability to control the access of multiple threads to shared resources. Without synchronisation, one thread can modify a shared variable while another thread is in the process of using or updating the same shared variable. This usually leads to significant errors.

Q: Explain the different way of using thread?

Ans: The thread could be implemented by using runnable interface or by inheriting from the Thread class. Runnable Interface is more advantageous because when you are going for multiple inheritances, the only interface can help.

Q: What are pass by reference and pass by value?
Ans: Pass By Reference means the passing the address itself rather than passing the value. Pass by Value means passing a copy of the value to be passed.

Q: What are HashMap and Map?
Ans: Map is Interface and Hashmap is a class that implements that.

Q: Difference between HashMap and HashTable?
Ans: The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronised and permits nulls. (HashMap allows null values as a key, and value whereas Hashtable doesn't allow). HashMap does not guarantee that the order of the map will remain constant over time. HashMap is unsynchronised, and Hashtable is synchronised.

Q: Difference between Vector and ArrayList?
Ans: Vector is synchronised whereas ArrayList is not.

Q: Difference between Swing and AWT?
Ans: AWT is heavyweight components. Swings are lightweight components. Hence swing works faster than AWT.

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

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

Q: State the significance of public, private, protected, default modifiers both singly and in combination and state

The effect of package relationships on declared items qualified by these modifiers.

- Public: Public class is visible in other packages, a field is visible everywhere (class must be public too)
- Private: Private variables or methods may be used only by an instance of the same class that declares the variable or method. A private feature may only be accessed by the class that owns the feature.
- Protected: Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature. This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.
- Default: What you get by default, i.e., without any access modifier (i.e., public-private or protected).It means that it is visible to all within a particular package.

Q: What is an abstract class?

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

Q: What is static in java?

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

Q: What is the Final?

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

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

Ans: The program compiles properly, but at runtime, it will give

"Main method not public." message.

Q: What if the static modifier is removed from the signature of the main method?
Ans: Program compiles. But at runtime throws an error "NoSuchMethodError."

Q: What if I write static public void instead of the public static void?
Ans: Program compiles and runs properly.

Q: What if I do not provide the String array as the argument to the method?
Ans: Program compiles but throws a runtime error "NoSuchMethodError."

Q: What is the first argument of the String array in the main method?
Ans: The String array is empty. It does not have any element. This is unlike C/C++, where the first element by default is the program name.

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

Q: How can one prove that the array is not null but empty using one line of code?
Ans: Print args.length. It will print 0. That means it is empty. But if it would have been null, then it would have thrown a NullPointerException on attempting to print args.length.

Q: What environment variables do I need to set on my machine to be able to run Java programs?
Ans: CLASSPATH and PATH are the two variables.

Q: Can an application have multiple classes having the main method?
Ans: Yes, it is possible. While starting the application, we mention the class name to be run. The JVM will look for the Main method only in the class whose name you have mentioned. Hence there is no conflict amongst the multiple classes having the main method.

Q: Can I have multiple main methods in the same class?
Ans: No, the program fails to compile. The compiler says that the main method is already defined in the class.

Q: Do I need to import java.lang package any time? Why?
Ans: No. It is by default loaded internally by the JVM.

Read Also: Final VS Finally VS Finalize

Q: Can I import the same package/class twice? Will the JVM load the package twice at runtime?
Ans: One can import the same package or the same class multiple times. Neither compiler nor JVM complains abt it. And the JVM will internally load the class only once no matter how many times you import the same class.

Q: What are Checked and Unchecked Exception?
Ans: A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown. e.g., IOException has thrown by java.io.FileInputStream's read() method•

Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the exception or declare it in a throws clause. Client programmers may not even know that the exception could be thrown. E.g., StringIndexOutOfBoundsException thrown by String's charAt() method• Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be.

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

Q: What are the different types of inner classes?

Ans: Nested top-level classes, Member classes, Local classes, Anonymous classes

Nested top-level classes, you declare a class within a class and specify the static modifier, the compiler treats the class just like any other top-level class.

Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package. e.g., outer.inner. Top-level inner classes implicitly have access only to static variables. There can also be inner interfaces. All of these are of the nested top-level variety.

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

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

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

Q: Are the imports checked for validity at compile time? e.g. will the code containing an import such asjava.lang.ABCD compile?

Ans: Yes, the imports are checked for the semantic validity at compile time. The code containing the above line of
import will not compile. It will throw an error saying, can not resolve symbol: class ABCD

location: package io
import java.io.ABCD;

Q: Does importing a package imports the subpackages as well? e.g. Does import com.MyTest.* also import com.MyTest.UnitTests.*?
Ans: No, you will have to import the sub-packages explicitly. It is importing com.MyTest.* will import classes in the package MyTest only. It will not import any class in any of its subpackages.

Q: What is the difference between declaring a variable and defining a variable?
Ans: In the declaration, we mention the type of the variable and its name. We do not initialise it. But defining means declaration + initialisation.
e.g. String s; is just a declaration while String s = new String (“abcd”); Or String s = “abcd”; are both definitions.

Q: What is the default value of an object reference declared as an instance variable?
Ans: null unless we define it explicitly.

Q: Can a top-level class be private or protected?
Ans: No. A top-level class can not be private or protected. It can have either “public” or no modifier. If it does not have a modifier, it is supposed to have a default access. If a top-level class is declared as private, the compiler will complain that the “modifier private is not allowed here”. This means that a top-level class can not be private.
Same is the case with protected.

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

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

Q: Objects are passed by value or by reference?
Ans: Java only supports pass by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object.

Q: What is serialisation?

Ans: Serialization is a mechanism by which you can save the state of an object by converting it to a byte stream.

Q: How do I serialise an object to a file?

Ans: The class whose instances are to be serialised should implement an interface Serializable. Then you pass the instance to the ObjectOutputStream which is connected to a fileoutputstream. This will save the object to a file.

Q: Which methods of the Serializable interface should I implement?

Ans: The serializable interface is an empty interface; it does not contain any methods. So we do not implement any methods.

Q: How can I customise the serialisation process? i.e. how can one have control over the serialisation process?

Ans: Yes, it is possible to have control over the serialisation process. The class should implement the Externalizable interface. This interface contains two methods, namely readExternal and writeExternal. You should implement these methods and write the logic for customising the serialisation process.

Q: What is the common usage of serialisation?

Ans: Whenever an object is to be sent over the network, objects need to be serialised. Moreover, if the state of an object is to be saved, objects need to be serialised.

Q: What is the Externalizable interface?

Ans: Externalizable is an interface which contains two methods readExternal and writeExternal. These methods give you control over the serialisation mechanism. Thus if your class implements this interface, you can customise the serialisation process by implementing these methods.

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

Ans: The serialisation mechanism generates an object graph for serialisation. Thus it determines whether the included object references are serializable or not. This is a recursive process. Thus when an object is serialised, all the included objects are also serialised along with the original object.

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

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

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

Ans: There are three exceptions in which serialisation does not necessarily read and write to the stream. These are

- Serialisation ignores static fields because they are not part of ay particular state.
- Base class fields are only handled if the base class itself is serializable.
- Transient fields.

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

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

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

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

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

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

Q: What are wrapper classes?

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

Q: Why do we need wrapper classes?

Ans: It is sometimes easier to deal with primitives as objects. Moreover, most of the collection classes store objects and not

primitive data types. And also the wrapper classes provide many utility methods. Because of these reasons we need wrapper classes. And since we create instances of these classes, we can store them in any of
the collection classes and pass them around as a collection. Also, we can pass them around as method parameters where a method expects an object.

Q: What are the checked exceptions?
Ans: Checked exception is those which the Java compiler forces you to catch. e.g. IOException has checked Exceptions.

Q: What are runtime exceptions?
Ans: Runtime exceptions are those exceptions that are thrown at runtime because of either wrong input data or because of wrong business logic etc. The compiler does not check these at compile time.

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

Q: How to create custom exceptions?
Ans: Your class should extend class Exception or some more specific type thereof.

Q: If I want an object of my class to be thrown as an exception object, what should I do?
Ans: The class should extend from Exception class. Or you can extend your class from some more precise exception type also.

Q: If my class already extends from some other class, what should I do if I want an instance of my class to be thrown as an exception object?
Ans: One can not do anything in this scenario. Because Java does

not allow multiple inheritances and does not provide any exception interface as well.

Q: How does an exception permeate through the code?

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

Q: What are the different ways to handle exceptions?

Ans: There are two ways to handle exceptions,

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

Q: What is the basic difference between the 2 approaches to exception handling.

- Try catch block and
- specifying the candidate exceptions in the throws clause?

Q: When should you use which approach?

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

Q: Is it necessary that a catch block must follow each try block?

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

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

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

Q: How are the Observer and Observable used?

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

Q: What is synchronisation, and why is it important?

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

Q: How does Java handle integer overflows and underflows?

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

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

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

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

Ans: Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

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

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

Q: What is the purpose of finalisation?

Ans: The purpose of finalisation is to allow an unreachable object to perform any cleanup processing before the object is garbage collected.

Q: What is the Locale class?

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

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

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

Q: What is the difference between static and nonstatic variables?

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

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

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

Q: What are synchronised methods and synchronised statements?

Ans: Synchronized methods are methods that are used to control

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

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

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

Q: What are the steps in the JDBC connection?

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

Step 1: Register the database driver by using :
Class.forName(\" driver class for that specific database\");

Step 2: Now create a database connection using :
Connection con = DriverManager.getConnection(url,username,password);

Step 3: Now Create a query using :
Statement stmt = Connection.Statement(\" select * from TABLE NAME\");

Step 4: Execute the query :
stmt.exceuteUpdate();

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

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

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

Ans: No, in this case, the finally block will not execute because

when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

Q: Is Empty .java file a valid source file?
Ans: Yes, an empty .java file is a perfectly valid source file.

Q: Can a .java file contain more than one java classes?
Ans: Yes, a .java file contains more than one java classes, provided at the most one of them is a public class.

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

Q: Is the main a keyword in Java?
Ans: No, the main is not a keyword in Java.

Q: Is next to a keyword in Java?
Ans: No, next is not a keyword.

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

Q: Is exit a keyword in Java?
Ans: No. To exit a program explicitly, you use exit method in the System object

Q: What happens if you don't initialise an instance variable of any of the primitive types in Java?
Ans: Java by default, initialises it to the default value for that primitive type. Thus an int will be initialised to 0, and a boolean will be initialised to false.

Q: What will be the initial value of an object reference which is defined as an instance variable?
Ans: The object references are all initialised to null in Java. However, to do anything useful with these references, you must set them to a valid object, else you will get NullPointerExceptions everywhere you try to use such default initialised references.

Q: What are the different scopes for Java variables?

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

- Instance: – These are typical object level variables; they are initialised to default values at the time of the creation of an object. They remain accessible as long as the object is accessible.
- Local: – These are the variables that are defined within a method. They remain accessible only during method execution. When the method finishes execution, these variables fall out of scope.
- Static: – These are the class level variables. They are initialised when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance.

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

Ans: The local variables are not initialised to any default value, neither primitives nor object references. If you try to use these variables without initialising them explicitly, the java compiler will not compile the code. It will complain abt the local variable not being initialised.

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

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

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

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

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

Q: Can the main method be declared final?

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

Q: What will be the output of the following statement?

System.out.println (“1″ + 3);

Ans: It will print 13.

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

Ans: If the array is an array of primitive types, then all the elements of the array will be initialised to the default value

corresponding to that primitive type. e.g. All the elements of an array of int will be initialised to 0, while that of boolean type will be initialised to false. Whereas if the array is an array of references (of any type), all the elements will be initialised to null.