1. **[What is immutable object in Java? Can you change values of a immutable object?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-immutable-object-in-java--can-you-change-values-of-a-immutable-object-" \o "What is immutable object in Java? Can you change values of a immutable object?)**

A Java object is considered immutable when its state cannot change after it is created. Use of immutable objects is widely accepted as a sound strategy for creating simple, reliable code. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state. java.lang.String and java.lang.Integer classes are the Examples of immutable objects from the Java Development Kit. Immutable objects simplify your program, since they :

* + are simple to use test and construct.
  + are automatically thread-safe.
  + do not require a copy constructor.
  + do not require an implementation of clone.
  + allow hashCode to use lazy initialization, and to cache its return value.
  + do not need to be copied defensively when used as a field.
  + are good Map keys and Set elements (these objects must not change state while stored in the collection).
  + have their class invariant established once upon construction, and it never needs to be checked again.
  + always have "failure atomicity" (a term used by Joshua Bloch) : if an immutable object throws an exception, it's never left in an undesirable or indeterminate state.

1. **How to create a immutable object in Java? Does all property of immutable object needs to be final?**

To create a object immutable You need to make the class final and all its member final so that once objects gets crated no one can modify its state. You can achieve same functionality by making member as non final but private and not modifying them except in constructor. Also its NOT necessary to have all the properties final since you can achieve same functionality by making member as non final but private and not modifying them except in constructor.

1. **[What is difference between String, StringBuffer and StringBuilder? When to use them?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-difference-between-string--stringbuffer-and-stringbuilder--when-to-use-them-" \o "What is difference between String, StringBuffer and StringBuilder? When to use them?)**

Below is the main difference between these three most commonly used classes.

* + String class objects are immutable whereas StringBuffer and StringBuilder objects are mutable.
  + StringBuffer is synchronized while StringBuilder is not synchronized.
  + Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.

Criteria to choose among String, StringBuffer and StringBuilder

* + If the Object value is not going to change use String Class because a String object is immutable.
  + If the Object value can change and will only be accessed from a single thread, use a StringBuilder because StringBuilder is unsynchronized.
  + In case the Object value can change, and will be modified by multiple threads, use a StringBuffer because StringBuffer is synchronized.

1. **[Why String class is final or immutable?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-string-class-is-final-or-immutable-" \o "Why String class is final or immutable?)**

It is very useful to have strings implemented as final or immutable objects. Below are some advantages of String Immutability in Java

* + Immutable objects are thread-safe. Two threads can both work on an immutable object at the same time without any possibility of conflict.
  + Security: the system can pass on sensitive bits of read-only information without worrying that it will be altered
  + You can share duplicates by pointing them to a single instance.
  + You can create substrings without copying. You just create a pointer into an existing base String guaranteed never to change. Immutability is the secret behind Java’s very fast substring implementation.
  + Immutable objects are much better suited to be Hashtable keys. If you change the value of an object that is used as a hash table key without removing it and re-adding it you lose the mapping.
  + Since String is immutable, inside each String is a char[] exactly the correct length. Unlike a StringBuilder there is no need for padding to allow for growth.
  + If String were not final, you could create a subclass and have two strings that look alike when "seen as Strings", but that are actually different.

1. **[Is Java Pass by Reference or Pass by Value?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "is-java-pass-by-reference-or-pass-by-value-" \o "Is Java Pass by Reference or Pass by Value?)**

The Java Spec says that everything in Java is pass-by-value. There is no such thing as "pass-by-reference" in Java. The difficult thing can be to understand that Java passes "objects as references" passed by value. This can certainly get confusing and I would recommend reading this article from an expert: <http://javadude.com/articles/passbyvalue.htm> Also read this interesting thread with example on StackOverflow : [Java Pass By Ref or Value](http://stackoverflow.com/questions/40480/is-java-pass-by-reference)

1. **[What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-outofmemoryerror-in-java--how-to-deal-with-java-lang-outofmemeryerror--error-" \o "What is OutOfMemoryError in java? How to deal with java.lang.OutOfMemeryError error?)**

This Error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector. **Note:** Its an Error (extends java.lang.Error) not Exception. Two important types of OutOfMemoryError are often encountered

* + **java.lang.OutOfMemoryError: Java heap space**

The quick solution is to add these flags to JVM command line when Java runtime is started:

[view plainprint?](http://www.fromdev.com/2012/02/java-interview-question-answer.html)

* + 1. -Xms1024m -Xmx1024m
  + **java.lang.OutOfMemoryError: PermGen space**

The solution is to add these flags to JVM command line when Java runtime is started:

[view plainprint?](http://www.fromdev.com/2012/02/java-interview-question-answer.html)

* + 1. -XX:+CMSClassUnloadingEnabled-XX:+CMSPermGenSweepingEnabled

**Long Term Solution**: Increasing the Start/Max Heap size or changing Garbage Collection options may not always be a long term solution for your Out Of Memory Error problem. Best approach is to understand the memory needs of your program and ensure it uses memory wisely and does not have leaks. You can use a Java memory profiler to determine what methods in your program are allocating large number of objects and then determine if there is a way to make sure they are no longer referenced, or to not allocate them in the first place.

1. **[What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-the-use-of-the-finally-block--is-finally-block-in-java-guaranteed-to-be-called--when-finally-block-is-not-called-" \o "What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?)**

Finally is the block of code that executes always. The code in finally block will execute even if an exception is occurred. Finally block is NOT called in following conditions

* + If the JVM exits while the try or catch code is being executed, then the finally block may not execute. This may happen due to System.exit() call.
  + if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.
  + If a exception is thrown in finally block and not handled then remaining code in finally block may not be executed.

1. **[Why there are two Date classes; one in java.util package and another in java.sql?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-there-are-two-date-classes--one-in-java-util-package-and-another-in-java-sql-" \o "Why there are two Date classes; one in java.util package and another in java.sql?)**

From the JavaDoc of java.sql.Date:

A thin wrapper around a millisecond value that allows JDBC to identify this as an SQL DATE value. A milliseconds value represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT. To conform with the definition of SQL DATE, the millisecond values wrapped by a java.sql.Date instance must be 'normalized' by setting the hours, minutes, seconds, and milliseconds to zero in the particular time zone with which the instance is associated.

**Explanation**: A java.util.Date represents date and time of day, a java.sql.Date only represents a date (the complement of java.sql.Date is java.sql.Time, which only represents a time of day, but also extends java.util.Date).

1. **[What is Marker interface? How is it used in Java?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-marker-interface--how-is-it-used-in-java-" \o "What is Marker interface? How is it used in Java?)**

The marker interface is a design pattern, used with languages that provide run-time type information about objects. It provides a way to associate metadata with a class where the language does not have explicit support for such metadata. To use this pattern, a class implements a marker interface, and code that interact with instances of that class test for the existence of the interface. Whereas a typical interface specifies functionality (in the form of method declarations) that an implementing class must support, a marker interface need not do so. The mere presence of such an interface indicates specific behavior on the part of the implementing class. There can be some hybrid interfaces, which both act as markers and specify required methods, are possible but may prove confusing if improperly used. Java utilizes this pattern very well and the example interfaces are

* + java.io.Serializable - Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.
  + java.rmi.Remote - The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a "remote interface", an interface that extends java.rmi.Remote are available remotely.
  + java.lang.Cloneable - A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class. Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.
  + javax.servlet.SingleThreadModel - Ensures that servlets handle only one request at a time. This interface has no methods.
  + java.util.EvenListener - A tagging interface that all event listener interfaces must extend.

The "instanceof" keyword in java can be used to test if an object is of a specified type. So this keyword in combination with Marker interface can be used to take different actions based on type of interface an object implements.

1. **[Why main() in java is declared as public static void main? What if the main method is declared as private?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "why-main---in-java-is-declared-as-public-static-void-main-" \o "Why main() in java is declared as public static void main?)**

Public - main method is called by JVM to run the method which is outside the scope of project therefore the access specifier has to be public to permit call from anywhere outside the application static - When the JVM makes are call to the main method there is not object existing for the class being called therefore it has to have static method to allow invocation from class. void - Java is platform independent language therefore if it will return some value then the value may mean different to different platforms so unlike C it can not assume a behavior of returning value to the operating system. If main method is declared as private then - Program will compile properly but at run-time it will give "Main method not public." error.

The only difference between experienced and inexperienced software developers is that the experienced ones realize when they're making a mistake.

Can you think of a questions which is not part of this post? Please don't forget to share it with me in comments section & I will try to include it in the list.

## What are available drivers in JDBC?

JDBC technology drivers fit into one of four categories:

* 1. A *JDBC-ODBC bridge* provides JDBC API access via one or more ODBC drivers. Note that some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver. Hence, this kind of driver is generally most appropriate when automatic installation and downloading of a Java technology application is not important. For information on the JDBC-ODBC bridge driver provided by Sun, see [JDBC-ODBC Bridge Driver](http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/bridge.doc.html#996747).
  2. A *native-API partly Java technology-enabled driver* converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.
  3. A *net-protocol fully Java technology-enabled driver* translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all of its Java technology-based clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC API alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use. In order for these products to also support Internet access they must handle the additional requirements for security, access through firewalls, etc., that the Web imposes. Several vendors are adding JDBC technology-based drivers to their existing database middleware products.
  4. A *native-protocol fully Java technology-enabled driver* converts JDBC technology calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary the database vendors themselves will be the primary source for this style of driver. Several database vendors have these in progress.

## What are the types of statements in JDBC?

the JDBC API has 3 Interfaces, (1. Statement, 2. PreparedStatement, 3. CallableStatement ). The key features of these are as follows: **Statement**

* 1. This interface is used for executing a static SQL statement and returning the results it produces.
  2. The object of Statement class can be created using Connection.createStatement() method.

**PreparedStatement**

* 1. A SQL statement is pre-compiled and stored in a PreparedStatement object.
  2. This object can then be used to efficiently execute this statement multiple times.
  3. The object of PreparedStatement class can be created using Connection.prepareStatement() method. This extends Statement interface.

**CallableStatement**

* 1. This interface is used to execute SQL stored procedures.
  2. This extends PreparedStatement interface.
  3. The object of CallableStatement class can be created using Connection.prepareCall() method.

## What is a stored procedure? How to call stored procedure using JDBC API?

Stored procedure is a group of SQL statements that forms a logical unit and performs a particular task. Stored Procedures are used to encapsulate a set of operations or queries to execute on database. Stored procedures can be compiled and executed with different parameters and results and may have any combination of input/output parameters. Stored procedures can be called using CallableStatement class in JDBC API. Below code snippet shows how this can be achieved.

[view plainprint?](http://www.fromdev.com/2010/08/10-jdbc-questions-for-java-beginners.html)

* 1. CallableStatement cs = con.prepareCall("{call MY\_STORED\_PROC\_NAME}");
  2. ResultSet rs = cs.executeQuery();

## What is Connection pooling? What are the advantages of using a connection pool?

Connection Pooling is a technique used for sharing the server resources among requested clients. It was pioneered by database vendors to allow multiple clients to share a cached set of connection objects that provides access to a database.

Getting connection and disconnecting are costly operation, which affects the application performance, so we should avoid creating multiple connection during multiple database interactions. A pool contains set of Database connections which are already connected, and any client who wants to use it can take it from pool and when done with using it can be returned back to the pool.

Apart from performance this also saves you resources as there may be limited database connections available for your application.

### How to do database connection using JDBC thin driver ?

This is one of the most commonly asked questions from JDBC fundamentals, and knowing all the steps of JDBC connection is important.

* 1. import java.sql.\*;
  2. class JDBCTest {
  3. public static void main (String args []) throws Exception
  4. {
  5. //Load driver class
  6. Class.forName ("oracle.jdbc.driver.OracleDriver");
  7. //Create connection
  8. Connection conn = DriverManager.getConnection
  9. ("jdbc:oracle:thin:@hostname:1526:testdb", "scott", "tiger");
  10. // @machineName:port:SID,   userid,  password
  12. Statement stmt = conn.createStatement();
  13. ResultSet rs = stmt.executeQuery("select 'Hi' from dual");
  14. while (rs.next())
  15. System.out.println (rs.getString(1));   // Print col 1 => Hi
  16. stmt.close();
  17. }
  18. }

## What does Class.forName() method do?

Method forName() is a static method of java.lang.Class. This can be used to dynamically load a class at run-time. Class.forName() loads the class if its not already loaded. It also executes the static block of loaded class. Then this method returns an instance of the loaded class. So a call to Class.forName('MyClass') is going to do following

- Load the class MyClass.

- Execute any static block code of MyClass.

- Return an instance of MyClass.

JDBC Driver loading using Class.forName is a good example of best use of this method. The driver loading is done like this

* 1. Class.forName("org.mysql.Driver");

All JDBC Drivers have a static block that registers itself with DriverManager and DriverManager has static initializer method registerDriver() which can be called in a static blocks of Driver class. A MySQL JDBC Driver has a static initializer which looks like this:

* 1. static {
  2. try {
  3. java.sql.DriverManager.registerDriver(new Driver());
  4. } catch (SQLException E) {
  5. throw new RuntimeException("Can't register driver!");
  6. }
  7. }

Class.forName() loads driver class and executes the static block and the Driver registers itself with the DriverManager.

## Which one will you use Statement or PreparedStatement? Or Which one to use when (Statement/PreparedStatement)? Compare PreparedStatement vs Statement.

By Java API definitions: **Statement** is a object used for executing a static SQL statement and returning the results it produces. **PreparedStatement** is a SQL statement which is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times. There are few advantages of using PreparedStatements over Statements

* 1. Since its pre-compiled, Executing the same query multiple times in loop, binding different parameter values each time is faster. (What does pre-compiled statement means? The prepared statement(pre-compiled) concept is not specific to Java, it is a database concept. Statement precompiling means: when you execute a SQL query, database server will prepare a execution plan before executing the actual query, this execution plan will be cached at database server for further execution.)
  2. In PreparedStatement the setDate()/setString() methods can be used to escape dates and strings properly, in a database-independent way.
  3. SQL injection attacks on a system are virtually impossible when using PreparedStatements.

## What does setAutoCommit(false) do?

A JDBC connection is created in auto-commit mode by default. This means that each individual SQL statement is treated as a transaction and will be automatically committed as soon as it is executed. If you require two or more statements to be grouped into a transaction then you need to disable auto-commit mode using below command

* 1. con.setAutoCommit(false);

Once auto-commit mode is disabled, no SQL statements will be committed until you explicitly call the commit method. A Simple transaction with use of autocommit flag is demonstrated below.

* 1. con.setAutoCommit(false);
  2. PreparedStatement updateStmt =
  3. con.prepareStatement( "UPDATE EMPLOYEE SET SALARY = ? WHERE EMP\_NAME LIKE ?");
  4. updateStmt.setInt(1, 5000); updateSales.setString(2, "Jack");
  5. updateStmt.executeUpdate();
  6. updateStmt.setInt(1, 6000); updateSales.setString(2, "Tom");
  7. updateStmt.executeUpdate();
  8. con.commit();
  9. con.setAutoCommit(true);

## What are database warnings and How can I handle database warnings in JDBC?

Warnings are issued by database to notify user of a problem which may not be very severe. Database warnings do not stop the execution of SQL statements. In JDBC SQLWarning is an exception that provides information on database access warnings. Warnings are silently chained to the object whose method caused it to be reported. Warnings may be retrieved from Connection, Statement, and ResultSet objects. Handling SQLWarning from connection object

* 1. //Retrieving warning from connection object
  2. SQLWarning warning = conn.getWarnings();
  4. //Retrieving next warning from warning object itself
  5. SQLWarning nextWarning = warning.getNextWarning();
  7. //Clear all warnings reported for this Connection object.
  8. conn.clearWarnings();

Handling SQLWarning from Statement object

* 1. //Retrieving warning from statement object
  2. stmt.getWarnings();
  4. //Retrieving next warning from warning object itself
  5. SQLWarning nextWarning = warning.getNextWarning();
  7. //Clear all warnings reported for this Statement object.
  8. stmt.clearWarnings();

Handling SQLWarning from ResultSet object

* 1. //Retrieving warning from resultset object
  2. rs.getWarnings();
  4. //Retrieving next warning from warning object itself
  5. SQLWarning nextWarning = warning.getNextWarning();
  7. //Clear all warnings reported for this resultset object.
  8. rs.clearWarnings();

The call to **getWarnings()** method in any of above way retrieves the first warning reported by calls on this object. If there is more than one warning, subsequent warnings will be chained to the first one and can be retrieved by calling the method **SQLWarning.getNextWarning** on the warning that was retrieved previously. A call to **clearWarnings()** method clears all warnings reported for this object. After a call to this method, the method getWarnings returns null until a new warning is reported for this object. Trying to call **getWarning()** on a connection after it has been closed will cause an SQLException to be thrown. Similarly, trying to retrieve a warning on a statement after it has been closed or on a result set after it has been closed will cause an SQLException to be thrown. Note that closing a statement also closes a result set that it might have produced.

## What is Metadata and why should I use it?

JDBC API has 2 Metadata interfaces DatabaseMetaData & ResultSetMetaData. The DatabaseMetaData provides Comprehensive information about the database as a whole. This interface is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the driver based on JDBC technology ("JDBC driver") that is used with it. Below is a sample code which demonstrates how we can use the DatabaseMetaData

* 1. DatabaseMetaData md = conn.getMetaData();
  2. System.out.println("Database Name: " + md.getDatabaseProductName());
  3. System.out.println("Database Version: " + md.getDatabaseProductVersion());
  4. System.out.println("Driver Name: " + md.getDriverName());
  5. System.out.println("Driver Version: " + md.getDriverVersion());

The ResultSetMetaData is an object that can be used to get information about the types and properties of the columns in a ResultSet object. Use DatabaseMetaData to find information about your database, such as its capabilities and structure. Use ResultSetMetaData to find information about the results of an SQL query, such as size and types of columns. Below a sample code which demonstrates how we can use the ResultSetMetaData

* 1. ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM TABLE2");
  2. ResultSetMetaData rsmd = rs.getMetaData();
  3. int numberOfColumns = rsmd.getColumnCount();
  4. boolean b = rsmd.isSearchable(1);

## What is RowSet? or What is the difference between RowSet and ResultSet? or Why do we need RowSet? or What are the advantages of using RowSet over ResultSet?

RowSet is a interface that adds support to the JDBC API for the JavaBeans component model. A rowset, which can be used as a JavaBeans component in a visual Bean development environment, can be created and configured at design time and executed at run time. The RowSet interface provides a set of JavaBeans properties that allow a RowSet instance to be configured to connect to a JDBC data source and read some data from the data source. A group of setter methods (setInt, setBytes, setString, and so on) provide a way to pass input parameters to a rowset's command property. This command is the SQL query the rowset uses when it gets its data from a relational database, which is generally the case. Rowsets are easy to use since the RowSet interface extends the standard java.sql.ResultSet interface so it has all the methods of ResultSet. There are two clear advantages of using RowSet over ResultSet

* 1. RowSet makes it possible to use the ResultSet object as a JavaBeans component. As a consequence, a result set can, for example, be a component in a Swing application.
  2. RowSet be used to make a ResultSet object scrollable and updatable. All RowSet objects are by default scrollable and updatable. If the driver and database being used do not support scrolling and/or updating of result sets, an application can populate a RowSet object implementation (e.g. JdbcRowSet) with the data of a ResultSet object and then operate on the RowSet object as if it were the ResultSet object.

## What is a connected RowSet? or What is the difference between connected RowSet and disconnected RowSet? or Connected vs Disconnected RowSet, which one should I use and when?

## Connected RowSet

A RowSet object may make a connection with a data source and maintain that connection throughout its life cycle, in which case it is called a connected rowset. A rowset may also make a connection with a data source, get data from it, and then close the connection. Such a rowset is called a disconnected rowset. A disconnected rowset may make changes to its data while it is disconnected and then send the changes back to the original source of the data, but it must reestablish a connection to do so. **Example of Connected RowSet:** A JdbcRowSet object is a example of connected RowSet, which means it continually maintains its connection to a database using a JDBC technology-enabled driver.

## Disconnected RowSet

A disconnected rowset may have a reader (a RowSetReader object) and a writer (a RowSetWriter object) associated with it. The reader may be implemented in many different ways to populate a rowset with data, including getting data from a non-relational data source. The writer can also be implemented in many different ways to propagate changes made to the rowset's data back to the underlying data source. **Example of Disconnected RowSet:** A CachedRowSet object is a example of disconnected rowset, which means that it makes use of a connection to its data source only briefly. It connects to its data source while it is reading data to populate itself with rows and again while it is propagating changes back to its underlying data source. The rest of the time, a CachedRowSet object is disconnected, including while its data is being modified. Being disconnected makes a RowSet object much leaner and therefore much easier to pass to another component. For example, a disconnected RowSet object can be serialized and passed over the wire to a thin client such as a personal digital assistant (PDA).

## What is the benefit of having JdbcRowSet implementation? Why do we need a JdbcRowSet like wrapper around ResultSet?

The JdbcRowSet implementation is a wrapper around a ResultSet object that has following advantages over ResultSet

* 1. This implementation makes it possible to use the ResultSet object as a JavaBeans component. A JdbcRowSet can be used as a JavaBeans component in a visual Bean development environment, can be created and configured at design time and executed at run time.
  2. It can be used to make a ResultSet object scrollable and updatable. All RowSet objects are by default scrollable and updatable. If the driver and database being used do not support scrolling and/or updating of result sets, an application can populate a JdbcRowSet object with the data of a ResultSet object and then operate on the JdbcRowSet object as if it were the ResultSet object.

Can you think of a questions which is not part of this post? Please don't forget to share it with me in comments section & I will try to include it in the list.

1. **[What is Java Collections API?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Java-Collections-API)**

Java Collections framework API is a unified architecture for representing and manipulating collections. The API contains Interfaces, Implementations & Algorithm to help java programmer in everyday programming. In nutshell, this API does 6 things at high level

* + Reduces programming efforts. - Increases program speed and quality.
  + Allows interoperability among unrelated APIs.
  + Reduces effort to learn and to use new APIs.
  + Reduces effort to design new APIs.
  + Encourages & Fosters software reuse.

To be specific, There are six collection java interfaces. The most basic interface is Collection. Three interfaces extend Collection: Set, List, and SortedSet. The other two collection interfaces, Map and SortedMap, do not extend Collection, as they represent mappings rather than true collections.

1. **[What is an Iterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-is-Iterator)**

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.

1. **[What is the difference between java.util.Iterator and java.util.ListIterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Iterator-vs-ListIterator)**

Iterator : Enables you to traverse through a collection in the forward direction only, for obtaining or removing elements ListIterator : extends Iterator, and allows bidirectional traversal of list and also allows the modification of elements.

1. **[What is HashMap and Map?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-is-HashMap)**

Map is Interface which is part of Java collections framework. This is to store Key Value pair, and Hashmap is class that implements that using hashing technique.

1. **[Difference between HashMap and HashTable? Compare Hashtable vs HashMap?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "HashMap-vs-HashTable)**

Both Hashtable & HashMap provide key-value access to data. The Hashtable is one of the original collection classes in Java (also called as legacy classes). HashMap is part of the new Collections Framework, added with Java 2, v1.2. There are several differences between HashMap and Hashtable in Java as listed below

* + The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesn’t allow nulls).
  + HashMap does not guarantee that the order of the map will remain constant over time. But one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.
  + HashMap is non synchronized whereas Hashtable is synchronized.
  + Iterator in the HashMap is fail-fast while the enumerator for the Hashtable isn't. So this could be a design consideration.

1. **What does synchronized means in Hashtable context?**

Synchronized means only one thread can modify a hash table at one point of time. Any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

1. **What is fail-fast property?**

At high level - Fail-fast is a property of a system or software with respect to its response to failures. A fail-fast system is designed to immediately report any failure or condition that is likely to lead to failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly-flawed process. When a problem occurs, a fail-fast system fails immediately and visibly. Failing fast is a non-intuitive technique: "failing immediately and visibly" sounds like it would make your software more fragile, but it actually makes it more robust. Bugs are easier to find and fix, so fewer go into production. In Java, Fail-fast term can be related to context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. **Why doesn't Collection extend Cloneable and Serializable?**

From Sun FAQ Page: Many Collection implementations (including all of the ones provided by the JDK) will have a public clone method, but it would be mistake to require it of all Collections. For example, what does it mean to clone a Collection that's backed by a terabyte SQL database? Should the method call cause the company to requisition a new disk farm? Similar arguments hold for serializable. If the client doesn't know the actual type of a Collection, it's much more flexible and less error prone to have the client decide what type of Collection is desired, create an empty Collection of this type, and use the addAll method to copy the elements of the original collection into the new one. Note on Some Important Terms

* + Synchronized means only one thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.
  + Fail-fast is relevant from the context of iterators. If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally”, a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn’t modify the collection "structurally”. However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

1. **How can we make Hashmap synchronized?**

HashMap can be synchronized by *Map m = Collections.synchronizedMap(hashMap);*

1. **Where will you use Hashtable and where will you use HashMap?**

There are multiple aspects to this decision: 1. The basic difference between a Hashtable and an HashMap is that, Hashtable is synchronized while HashMap is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Hashtable. While if not multiple threads are going to access the same instance then use HashMap. Non synchronized data structure will give better performance than the synchronized one. 2. If there is a possibility in future that - there can be a scenario when you may require to retain the order of objects in the Collection with key-value pair then HashMap can be a good choice. As one of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you can easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable. Also if you have multiple thread accessing you HashMap then Collections.synchronizedMap() method can be leveraged. Overall HashMap gives you more flexibility in terms of possible future changes.

1. **Difference between Vector and ArrayList? What is the Vector class?**

Vector & ArrayList both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. ArrayList and Vector class both implement the List interface. Both the classes are member of Java collection framework, therefore from an API perspective, these two classes are very similar. However, there are still some major differences between the two. Below are some key differences

* + Vector is a legacy class which has been retrofitted to implement the List interface since Java 2 platform v1.2
  + Vector is synchronized whereas ArrayList is not. Even though Vector class is synchronized, still when you want programs to run in multithreading environment using ArrayList with Collections.synchronizedList() is recommended over Vector.
  + ArrayList has no default size while vector has a default size of 10.
  + The Enumerations returned by Vector's elements method are not fail-fast. Whereas ArraayList does not have any method returning Enumerations.

**What is the Difference between Enumeration and Iterator interface?**

Enumeration and Iterator are the interface available in java.util package. The functionality of Enumeration interface is duplicated by the Iterator interface. New implementations should consider using Iterator in preference to Enumeration. Iterators differ from enumerations in following ways:

* + Enumeration contains 2 methods namely hasMoreElements() & nextElement() whereas Iterator contains three methods namely hasNext(), next(),remove().
  + Iterator adds an optional remove operation, and has shorter method names. Using remove() we can delete the objects but Enumeration interface does not support this feature.
  + Enumeration interface is used by legacy classes. Vector.elements() & Hashtable.elements() method returns Enumeration. Iterator is returned by all Java Collections Framework classes. java.util.Collection.iterator() method returns an instance of Iterator.

1. **[Why Java Vector class is considered obsolete or unofficially deprecated? or Why should I always use ArrayList over Vector?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Why-vector-class-deprecated)**

You should use ArrayList over Vector because you should default to non-synchronized access. Vector synchronizes each individual method. That's almost never what you want to do. Generally you want to synchronize a whole sequence of operations. Synchronizing individual operations is both less safe (if you iterate over a Vector, for instance, you still need to take out a lock to avoid anyone else changing the collection at the same time) but also slower (why take out a lock repeatedly when once will be enough)? Of course, it also has the overhead of locking even when you don't need to. It's a very flawed approach to have synchronized access as default. You can always decorate a collection using Collections.synchronizedList - the fact that Vector combines both the "resized array" collection implementation with the "synchronize every operation" bit is another example of poor design; the decoration approach gives cleaner separation of concerns. Vector also has a few legacy methods around enumeration and element retrieval which are different than the List interface, and developers (especially those who learned Java before 1.2) can tend to use them if they are in the code. Although Enumerations are faster, they don't check if the collection was modified during iteration, which can cause issues, and given that Vector might be chosen for its syncronization - with the attendant access from multiple threads, this makes it a particularly pernicious problem. Usage of these methods also couples a lot of code to Vector, such that it won't be easy to replace it with a different List implementation. Despite all above reasons Sun may never officially deprecate Vector class. (Read details [Deprecate Hashtable and Vector](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6201870))

1. **[What is an enumeration?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "What-enumeration)**

An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection.

1. **[What is the difference between Enumeration and Iterator?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Enumeration-vs-Iterator)**

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects. So Enumeration is used when ever we want to make Collection objects as Read-only.

1. **[Where will you use Vector and where will you use ArrayList?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "13)**

The basic difference between a Vector and an ArrayList is that, vector is synchronized while ArrayList is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Vector. While if not multiple threads are going to access the same instance then use ArrayList. Non synchronized data structure will give better performance than the synchronized one.

1. **What is the importance of hashCode() and equals() methods? How they are used in Java?**

The java.lang.Object has two methods defined in it. They are - public boolean equals(Object obj) public int hashCode(). These two methods are used heavily when objects are stored in collections. There is a contract between these two methods which should be kept in mind while overriding any of these methods. The Java API documentation describes it in detail. The hashCode() method returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable or java.util.HashMap. The general contract of hashCode is: Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application. If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables. As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. The equals(Object obj) method indicates whether some other object is "equal to" this one. The equals method implements an equivalence relation on non-null object references: It is reflexive: for any non-null reference value x, x.equals(x) should return true. It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true. It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true. It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified. For any non-null reference value x, x.equals(null) should return false. The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true). Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes. **A practical Example of hashcode() & equals():** This can be applied to classes that need to be stored in Set collections. Sets use equals() to enforce non-duplicates, and HashSet uses hashCode() as a first-cut test for equality. Technically hashCode() isn't necessary then since equals() will always be used in the end, but providing a meaningful hashCode() will improve performance for very large sets or objects that take a long time to compare using equals().

1. **[What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)**

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting. So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.

1. **[What is java.util.concurrent BlockingQueue? How it can be used?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "16)**

Java has implementation of BlockingQueue available since Java 1.5. Blocking Queue interface extends collection interface, which provides you power of collections inside a queue. Blocking Queue is a type of Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. A typical usage example would be based on a producer-consumer scenario. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers. An ArrayBlockingQueue is a implementation of blocking queue with an array used to store the queued objects. The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. ArrayBlockingQueue requires you to specify the capacity of queue at the object construction time itself. Once created, the capacity cannot be increased. This is a classic "bounded buffer" (fixed size buffer), in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Attempts to put an element to a full queue will result in the put operation blocking; attempts to retrieve an element from an empty queue will be blocked.

1. **[Set & List interface extend Collection, so Why doesn't Map interface extend Collection?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "17)**

Though the Map interface is part of collections framework, it does not extend collection interface. This is by design, and the answer to this questions is best described in Sun's FAQ Page: This was by design. We feel that mappings are not collections and collections are not mappings. Thus, it makes little sense for Map to extend the Collection interface (or vice versa). If a Map is a Collection, what are the elements? The only reasonable answer is "Key-value pairs", but this provides a very limited (and not particularly useful) Map abstraction. You can't ask what value a given key maps to, nor can you delete the entry for a given key without knowing what value it maps to. Collection could be made to extend Map, but this raises the question: what are the keys? There's no really satisfactory answer, and forcing one leads to an unnatural interface. Maps can be viewed as Collections (of keys, values, or pairs), and this fact is reflected in the three "Collection view operations" on Maps (keySet, entrySet, and values). While it is, in principle, possible to view a List as a Map mapping indices to elements, this has the nasty property that deleting an element from the List changes the Key associated with every element before the deleted element. That's why we don't have a map view operation on Lists.

1. **[Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "18)**

a. Vector b. ArrayList c. LinkedList ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

1. **[What is the difference between ArrayList and LinkedList? (ArrayList vs LinkedList.)](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "19)**

java.util.ArrayList and java.util.LinkedList are two Collections classes used for storing lists of object references **Here are some key differences:**

* + ArrayList uses primitive object array for storing objects whereas LinkedList is made up of a chain of nodes. Each node stores an element and the pointer to the next node. A singly linked list only has pointers to next. A doubly linked list has a pointer to the next and the previous element. This makes walking the list backward easier.
  + ArrayList implements the RandomAccess interface, and LinkedList does not. The commonly used ArrayList implementation uses primitive Object array for internal storage. Therefore an ArrayList is much faster than a LinkedList for random access, that is, when accessing arbitrary list elements using the get method. Note that the get method is implemented for LinkedLists, but it requires a sequential scan from the front or back of the list. This scan is very slow. For a LinkedList, there's no fast way to access the Nth element of the list.
  + Adding and deleting at the start and middle of the ArrayList is slow, because all the later elements have to be copied forward or backward. (Using System.arrayCopy()) Whereas Linked lists are faster for inserts and deletes anywhere in the list, since all you do is update a few next and previous pointers of a node.
  + Each element of a linked list (especially a doubly linked list) uses a bit more memory than its equivalent in array list, due to the need for next and previous pointers.
  + ArrayList may also have a performance issue when the internal array fills up. The arrayList has to create a new array and copy all the elements there. The ArrayList has a growth algorithm of (n\*3)/2+1, meaning that each time the buffer is too small it will create a new one of size (n\*3)/2+1 where n is the number of elements of the current buffer. Hence if we can guess the number of elements that we are going to have, then it makes sense to create a arraylist with that capacity during object creation (using construtor new ArrayList(capacity)). Whereas LinkedLists should not have such capacity issues.

1. **[Where will you use ArrayList and Where will you use LinkedList? Or Which one to use when (ArrayList / LinkedList).](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "20)**

Below is a snippet from SUN's site. The Java SDK contains 2 implementations of the List interface - ArrayList and LinkedList. If you frequently add elements to the beginning of the List or iterate over the List to delete elements from its interior, you should consider using LinkedList. These operations require constant-time in a LinkedList and linear-time in an ArrayList. But you pay a big price in performance. Positional access requires linear-time in a LinkedList and constant-time in an ArrayList.

1. **[What is performance of various Java collection implementations/algorithms? What is Big 'O' notation for each of them ?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "BigONotationJavaCollections)**

Each java collection implementation class have different performance for different methods, which makes them suitable for different programming needs.

## Performance of Map interface implementations

## Hashtable

An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. Note that the hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. The initial capacity and load factor parameters are merely hints to the implementation. The exact details as to when and whether the rehash method is invoked are implementation-dependent.

## HashMap

This implementation provides constant-time [ Big O Notation is O(1) ] performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## TreeMap

The TreeMap implementation provides guaranteed log(n) [ Big O Notation is O(log N) ] time cost for the containsKey, get, put and remove operations.

## LinkedHashMap

A linked hash map has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashMap. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashMap, as iteration times for this class are unaffected by capacity.

## Performance of Set interface implementations

## HashSet

The HashSet class offers constant-time [ Big O Notation is O(1) ] performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets. Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

## TreeSet

The TreeSet implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).

## LinkedHashSet

A linked hash set has two parameters that affect its performance: initial capacity and load factor. They are defined precisely as for HashSet. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for HashSet, as iteration times for this class are unaffected by capacity.

## Performance of List interface implementations

## LinkedList

- Performance of get and remove methods is linear time [ Big O Notation is O(n) ] - Performance of add and Iterator.remove methods is constant-time [ Big O Notation is O(1) ]

## ArrayList

- The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. [ Big O Notation is O(1) ] - The add operation runs in amortized constant time [ Big O Notation is O(1) ] , but in worst case (since the array must be resized and copied) adding n elements requires linear time [ Big O Notation is O(n) ] - Performance of remove method is linear time [ Big O Notation is O(n) ] - All of the other operations run in linear time [ Big O Notation is O(n) ]. The constant factor is low compared to that for the LinkedList implementation.

Can you think of a questions which is not part of this post? Please don't forget to share it with me in comments section & I will try to include it in the list.

## What is synchronization in respect to multi-threading in Java?

With respect to multi-threading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one Java thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to erroneous behavior or program.

## Explain different way of using thread?

A Java thread could be implemented by using Runnable interface or by extending the Thread class. The Runnable is more advantageous, when you are going for multiple inheritance.

## [What is the difference between Thread.start() & Thread.run() method?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-the-difference-between-thread-start-----thread-run---method-" \o "What is the difference between Thread.start() & Thread.run() method?)

Thread.start() method (native method) of Thread class actually does the job of running the Thread.run() method in a thread. If we directly call Thread.run() method it will executed in same thread, so does not solve the purpose of creating a new thread.

## [Why do we need run() & start() method both. Can we achieve it with only run method?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "why-do-we-need-run-----start---method-both--can-we-achieve-it-with-only-run-method-" \o "Why do we need run() & start() method both. Can we achieve it with only run method?)

We need run() & start() method both because JVM needs to create a separate thread which can not be differentiated from a normal method call. So this job is done by start method native implementation which has to be explicitly called. Another advantage of having these two methods is we can have any object run as a thread if it implements Runnable interface. This is to avoid Java’s multiple inheritance problems which will make it difficult to inherit another class with Thread.

## [What is ThreadLocal class? How can it be used?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-threadlocal-class--how-can-it-be-used-" \o "What is ThreadLocal class? How can it be used?)

Below are some key points about ThreadLocal variables

* + A thread-local variable effectively provides a separate copy of its value for each thread that uses it.
  + ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread
  + In case when multiple threads access a ThreadLocal instance, separate copy of Threadlocal variable is maintained for each thread.
  + Common use is seen in DAO pattern where the DAO class can be singleton but the Database connection can be maintained separately for each thread. (Per Thread Singleton)

ThreadLocal variable are difficult to understand and I have found below reference links very useful in getting better understanding on them

* + [Good article on ThreadLocal on IBM DeveloperWorks](http://www-128.ibm.com/developerworks/java/library/j-threads3.html)
  + [Managing data : Good example](http://javaboutique.internet.com/tutorials/localdata)
  + [Refer Java API Docs](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/ThreadLocal.html)

## [When InvalidMonitorStateException is thrown? Why?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "when-invalidmonitorstateexception-is-thrown--why-" \o "When InvalidMonitorStateException is thrown? Why?)

This exception is thrown when you try to call wait()/notify()/notifyAll() any of these methods for an Object from a point in your program where u are NOT having a lock on that object.(i.e. u r not executing any synchronized block/method of that object and still trying to call wait()/notify()/notifyAll()) wait(), notify() and notifyAll() all throw IllegalMonitorStateException. since This exception is a subclass of RuntimeException so we r not bound to catch it (although u may if u want to). and being a RuntimeException this exception is not mentioned in the signature of wait(), notify(), notifyAll() methods.

## [What is the difference between sleep(), suspend() and wait() ?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-the-difference-between-sleep----suspend---and-wait----" \o "What is the difference between sleep(), suspend() and wait() ?)

Thread.sleep() sends the current thread into the "Not Runnable" state for some amount of time. The thread keeps the monitors it has aquired -- i.e. if the thread is currently in a synchronized block or method no other thread can enter this block or method. If another thread calls t.interrupt() it will wake up the sleeping thread. Note that sleep is a static method, which means that it always affects the current thread (the one that is executing the sleep method). A common mistake is to call t.sleep() where t is a different thread; even then, it is the current thread that will sleep, not the t thread. t.suspend() is deprecated. Using it is possible to halt a thread other than the current thread. A suspended thread keeps all its monitors and since this state is not interruptable it is deadlock prone. object.wait() sends the current thread into the "Not Runnable" state, like sleep(), but with a twist. Wait is called on a object, not a thread; we call this object the "lock object." Before lock.wait() is called, the current thread must synchronize on the lock object; wait() then releases this lock, and adds the thread to the "wait list" associated with the lock. Later, another thread can synchronize on the same lock object and call lock.notify(). This wakes up the original, waiting thread. Basically, wait()/notify() is like sleep()/interrupt(), only the active thread does not need a direct pointer to the sleeping thread, but only to the shared lock object.

## [What happens when I make a static method as synchronized?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-happens-when-i-make-a-static-method-as-synchronized-" \o "What happens when I make a static method as synchronized?)

Synchronized static methods have a lock on the class "Class", so when a thread enters a synchronized static method, the class itself gets locked by the thread monitor and no other thread can enter any static synchronized methods on that class. This is unlike instance methods, as multiple threads can access "same synchronized instance methods" at same time for different instances.

## [Can a thread call a non-synchronized instance method of an Object when a synchronized method is being executed ?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "can-a-thread-call-a-non-synchronized-instance-method-of-an-object-when-a-synchronized-method-is-being-executed--" \o "Can a thread call a non-synchronized instance method of an Object when a synchronized method is being executed ?)

Yes, a Non synchronized method can always be called without any problem. In fact Java does not do any check for a non-synchronized method. The Lock object check is performed only for synchronized methods/blocks. In case the method is not declared synchronized Jave will call even if you are playing with shared data. So you have to be careful while doing such thing. The decision of declaring a method as synchronized has to be based on critical section access. If your method does not access a critical section (shared resource or data structure) it need not be declared synchronized. Below is the example which demonstrates this, The Common class has two methods synchronizedMethod1() and method1() MyThread class is calling both the methods in separate threads,

[view plainprint?](http://www.fromdev.com/2008/05/java-threading-questions.html)

* + public class Common {
  + public synchronized void synchronizedMethod1() {
  + System.out.println("synchronizedMethod1 called");
  + try {
  + Thread.sleep(1000);
  + } catch (InterruptedException e) {
  + e.printStackTrace();
  + }
  + System.out.println("synchronizedMethod1 done");
  + }
  + public void method1() {
  + System.out.println("Method 1 called");
  + try {
  + Thread.sleep(1000);
  + } catch (InterruptedException e) {
  + e.printStackTrace();
  + }
  + System.out.println("Method 1 done");
  + }
  + }

[view plainprint?](http://www.fromdev.com/2008/05/java-threading-questions.html)

* + public class MyThread extends Thread {
  + private int id = 0;
  + private Common common;
  + public MyThread(String name, int no, Common object) {
  + super(name);
  + common = object;
  + id = no;
  + }
  + public void run() {
  + System.out.println("Running Thread" + this.getName());
  + try {
  + if (id == 0) {
  + common.synchronizedMethod1();
  + } else {
  + common.method1();
  + }
  + } catch (Exception e) {
  + e.printStackTrace();
  + }
  + }
  + public static void main(String[] args) {
  + Common c = new Common();
  + MyThread t1 = new MyThread("MyThread-1", 0, c);
  + MyThread t2 = new MyThread("MyThread-2", 1, c);
  + t1.start();
  + t2.start();
  + }
  + }

Here is the output of the program

[view plainprint?](http://www.fromdev.com/2008/05/java-threading-questions.html)

* + Running ThreadMyThread-1
  + synchronizedMethod1 called
  + Running ThreadMyThread-2
  + Method 1 called
  + synchronizedMethod1 done
  + Method 1 done

This shows that method1() - is called even though the synchronizedMethod1() was in execution.

## [Can two threads call two different synchronized instance methods of an Object?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "can-two-threads-call-two-different-synchronized-instance-methods-of-an-object-" \o "Can two threads call two different synchronized instance methods of an Object?)

No. If a object has synchronized instance methods then the Object itself is used a lock object for controlling the synchronization. Therefore all other instance methods need to wait until previous method call is completed. See the below sample code which demonstrate it very clearly. The Class Common has 2 methods called synchronizedMethod1() and synchronizedMethod2() MyThread class is calling both the methods

[view plainprint?](http://www.fromdev.com/2008/05/java-threading-questions.html)

* + public class Common {
  + public synchronized void synchronizedMethod1() {
  + System.out.println("synchronizedMethod1 called");
  + try {
  + Thread.sleep(1000);
  + } catch (InterruptedException e) {
  + e.printStackTrace();
  + }
  + System.out.println("synchronizedMethod1 done");
  + }
  + public synchronized void synchronizedMethod2() {
  + System.out.println("synchronizedMethod2 called");
  + try {
  + Thread.sleep(1000);
  + } catch (InterruptedException e) {
  + e.printStackTrace();
  + }
  + System.out.println("synchronizedMethod2 done");
  + }
  + }

[view plainprint?](http://www.fromdev.com/2008/05/java-threading-questions.html)

* + public class MyThread extends Thread {
  + private int id = 0;
  + private Common common;
  + public MyThread(String name, int no, Common object) {
  + super(name);
  + common = object;
  + id = no;
  + }
  + public void run() {
  + System.out.println("Running Thread" + this.getName());
  + try {
  + if (id == 0) {
  + common.synchronizedMethod1();
  + } else {
  + common.synchronizedMethod2();
  + }
  + } catch (Exception e) {
  + e.printStackTrace();
  + }
  + }
  + public static void main(String[] args) {
  + Common c = new Common();
  + MyThread t1 = new MyThread("MyThread-1", 0, c);
  + MyThread t2 = new MyThread("MyThread-2", 1, c);
  + t1.start();
  + t2.start();
  + }
  + }

## [What is a deadlock?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-a-deadlock-" \o "What is a deadlock?)

Deadlock is a situation where two or more threads are blocked forever, waiting for each other. This may occur when two threads, each having a lock on one resource, attempt to acquire a lock on the other's resource. Each thread would wait indefinitely for the other to release the lock, unless one of the user processes is terminated. In terms of Java API, thread deadlock can occur in following conditions:

* + When two threads call Thread.join() on each other.
  + When two threads use nested synchronized blocks to lock two objects and the blocks lock the same objects in different order.

## [What is Starvation? and What is a Livelock?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-starvation--and-what-is-a-livelock-" \o "What is Starvation? and What is a Livelock?)

Starvation and livelock are much less common a problem than deadlock, but are still problems that every designer of concurrent software is likely to encounter.

### LiveLock

Livelock occurs when all threads are blocked, or are otherwise unable to proceed due to unavailability of required resources, and the non-existence of any unblocked thread to make those resources available. In terms of Java API, thread livelock can occur in following conditions:

* + When all the threads in a program execute Object.wait(0) on an object with zero parameter. The program is live-locked and cannot proceed until one or more threads call Object.notify() or Object.notifyAll() on the relevant objects. Because all the threads are blocked, neither call can be made.
  + When all the threads in a program are stuck in infinite loops.

### Starvation

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked. Starvation occurs when one thread cannot access the CPU because one or more other threads are monopolizing the CPU. In Java, thread starvation can be caused by setting thread priorities inappropriately. A lower-priority thread can be starved by higher-priority threads if the higher-priority threads do not yield control of the CPU from time to time.

## [How to find a deadlock has occurred in Java? How to detect a Deadlock in Java?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "how-to-find-a-deadlock-has-occurred-in-java-" \o "How to find a deadlock has occurred in Java?)

Earlier versions of Java had no mechanism to handle/detect deadlock. Since JDK 1.5 there are some powerful methods added in the java.lang.management package to diagnose and detect deadlocks. The java.lang.management.ThreadMXBean interface is management interface for the thread system of the Java virtual machine. It has two methods which can leveraged to detect deadlock in a Java application.

* + findMonitorDeadlockedThreads() - This method can be used to detect cycles of threads that are in deadlock waiting to acquire object monitors. It returns an array of thread IDs that are deadlocked waiting on monitor.
  + findDeadlockedThreads() - It returns an array of thread IDs that are deadlocked waiting on monitor or ownable synchronizers.

## [What is immutable object? How does it help in writing concurrent application?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-immutable-object--how-does-it-help-on-writing-concurrent-application-" \o "What is immutable object? How does it help on writing concurrent application?)

An object is considered immutable if its state cannot change after it is constructed. Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state. Examples of immutable objects from the JDK include String and Integer. Immutable objects greatly simplify your multi threaded program, since they are

* + Simple to construct, test, and use.
  + Automatically thread-safe and have no synchronization issues.

To create a object immutable You need to make the class final and all its member final so that once objects gets crated no one can modify its state. You can achieve same functionality by making member as non final but private and not modifying them except in constructor.

## [How will you take thread dump in Java? How will you analyze Thread dump?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "how-will-you-take-thread-dump-in-java--how-will-you-analyze-thread-dump-" \o "How will you take thread dump in Java? How will you analyze Thread dump?)

A Thread Dump is a complete list of active threads. A java thread dump is a way of finding out what each thread in the JVM is doing at a particular point of time. This is especially useful when your java application seems to have some performance issues. Thread dump will help you to find out which thread is causing this. There are several ways to take thread dumps from a JVM. It is highly recommended to take more than 1 thread dump and analyze the results based on it. Follow below steps to take thread dump of a java process

* + Step 1   
      
    On UNIX, Linux and Mac OSX Environment run below command:   
      
    ps -el | grep java   
      
    On Windows:   
      
    Press Ctrl+Shift+Esc to open the task manager and find the PID of the java process
  + Step 2:   
      
    Use jstack command to print the Java stack traces for a given Java process PID   
      
    jstack [PID]   
      
    More details of jstack command can be found here : [JSTACK Command Manual](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstack.html)

## What is a thread leak? What does it mean in Java?

Thread leak is when a application does not release references to a thread object properly. Due to this some Threads do not get garbage collected and the number of unused threads grow with time. Thread leak can often cause serious issues on a Java application since over a period of time too many threads will be created but not released and may cause applications to respond slow or hang.

## [How can I trace whether the application has a thread leak?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "how-can-i-trace-whether-the-application-has-a-thread-leak-" \o "How can I trace whether the application has a thread leak?)

If an application has thread leak then with time it will have too many unused threads. Try to find out what type of threads is leaking out. This can be done using following ways

* + Give unique and descriptive names to the threads created in application. - Add log entry in all thread at various entry and exit points in threads.
  + Change debugging config levels (debug, info, error etc) and analyze log messages.
  + When you find the class that is leaking out threads check how new threads are instantiated and how they're closed.
  + Make sure the thread is Guaranteed to close properly by doing following - Handling all Exceptions properly.
  + Make sure the thread is Guaranteed to close properly by doing following
    - Handling all Exceptions properly.
    - releasing all resources (e.g. connections, files etc) before it closes.

## [What is thread pool? Why should we use thread pools?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-thread-pool--why-should-we-use-thread-pools-" \o "What is thread pool? Why should we use thread pools?)

A thread pool is a collection of threads on which task can be scheduled. Instead of creating a new thread for each task, you can have one of the threads from the thread pool pulled out of the pool and assigned to the task. When the thread is finished with the task, it adds itself back to the pool and waits for another assignment. One common type of thread pool is the fixed thread pool. This type of pool always has a specified number of threads running; if a thread is somehow terminated while it is still in use, it is automatically replaced with a new thread. Below are key reasons to use a Thread Pool

* + Using thread pools minimizes the JVM overhead due to thread creation. Thread objects use a significant amount of memory, and in a large-scale application, allocating and de-allocating many thread objects creates a significant memory management overhead.
  + You have control over the maximum number of tasks that are being processed in parallel (= number of threads in the pool).

Most of the executor implementations in java.util.concurrent use thread pools, which consist of worker threads. This kind of thread exists separately from the Runnable and Callable tasks it executes and is often used to execute multiple tasks.

## [Can we synchronize the run method? If yes then what will be the behavior?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "can-we-synchronize-the-run-method--if-yes-then-what-will-be-the-behavior-" \o "Can we synchronize the run method? If yes then what will be the behavior?)

Yes, the run method of a runnable class can be synchronized. If you make run method synchronized then the lock on runnable object will be occupied before executing the run method. In case we start multiple threads using the same runnable object in the constructor of the Thread then it would work. But until the 1st thread ends the 2nd thread cannot start and until the 2nd thread ends the next cannot start as all the threads depend on lock on same object

## [Can we synchronize the constructor of a Java Class?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "can-we-synchronize-the-constructor-of-a-java-class-" \o "Can we synchronize the constructor of a Java Class?)

As per Java Language Specification, constructors cannot be synchronized because other threads cannot see the object being created before the thread creating it has finished it. There is no practical need of a Java Objects constructor to be synchronized, since it would lock the object being constructed, which is normally not available to other threads until all constructors of the object finish

## [Define Serialization? What do you mean by Serialization in Java?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "define-serialization--what-do-you-mean-by-serialization-in-java-" \o "Define Serialization? What do you mean by Serialization in Java?)

Serialization is a mechanism by which you can save or transfer the state of an object by converting it to a byte stream. This can be done in java by implementing Serialiazable interface. Serializable is defined as a marker interface which needs to be implemented for transferring an object over a network or persistence of its state to a file. Since its a marker interface, it does not contain any methods. Implementation of this interface enables the conversion of object into byte stream and thus can be transferred. The object conversion is done by the JVM using its default serialization mechanism.

## [Why is Serialization required? What is the need to Serialize?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "why-is-serialization-required--what-is-the-need-to-serialize-" \o "Why is Serialization required? What is the need to Serialize?)

Serialization is required for a variety of reasons. It is required to send across the state of an object over a network by means of a socket. One can also store an object’s state in a file. Additionally, manipulation of the state of an object as streams of bytes is required. The core of Java Serialization is the Serializable interface. When Serializable interface is implemented by your class it provides an indication to the compiler that java Serialization mechanism needs to be used to serialize the object.

## [What is the Difference between Externalizable and Serializable Interfaces?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-is-the-difference-between-externalizable-and-serializable-interfaces-" \o "What is the Difference between Externalizable and Serializable Interfaces?)

This is one of top serialization questions that is asked in many big companies to test your in-depth understanding of serialization. Serializable is a marker interface therefore you are not forced to implement any methods, however Externalizable contains two methods readExternal() and writeExternal() which must be implemented. Serializable interface provides a inbuilt serialization mechanism to you which can be in-efficient at times. However Externilizable interface is designed to give you greater control over the serialization mechanism. The two methods provide you immense opportunity to enhance the performance of specific object serialization based on application needs. Serializable interface provides a default serialization mechanism, on the other hand, Externalizable interface instead of relying on default Java Serialization provides flexibility to control this mechanism. One can drastically improve the application performance by implementing the Externalizable interface correctly. However there is also a chance that you may not write the best implementation, so if you are not really sure about the best way to serialize, I would suggest your stick to the default implementation using Serializable interface.

## [When will you use Serializable or Externalizable interface? and why?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "when-will-you-use-serializable-or-externalizable-interface--and-why-" \o "When will you use Serializable or Externalizable interface? and why?)

Most of the times when you want to do a selective attribute serialization you can use Serializable interface with transient modifier for variables not to be serialized. However, use of Externalizable interface can be really effective in cases when you have to serialize only some dynamically selected attributes of a large object. Lets take an example, Some times when you have a big Java object with hundreds of attributes and you want to serialize only a dozen dynamically selected attributes to keep the state of the object you should use Externalizable interface writeObject method to selectively serialize the chosen attributes. In case you have small objects and you know that most or all attributes are required to be serialized then you should be fine with using Serializable interface and use of transient variable as appropriate.

## [What are the ways to speed up Object Serialization? How to improve Serialization performance?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-the-ways-to-speed-up-object-serialization--how-to-improve-serialization-performance-" \o "What are the ways to speed up Object Serialization? How to improve Serialization performance?)

The default Java Serialization mechanism is really useful, however it can have a really bad performance based on your application and business requirements. The serialization process performance heavily depends on the number and size of attributes you are going to serialize for an object. Below are some tips you can use for speeding up the marshaling and un-marshaling of objects during Java serialization process.

* + Mark the unwanted or non Serializable attributes as transient. This is a straight forward benefit since your attributes for serialization are clearly marked and can be easily achieved using Serialzable interface itself.
  + Save only the state of the object, not the derived attributes. Some times we keep the derived attributes as part of the object however serializing them can be costly. Therefore consider calcualting them during de-serialization process.
  + Serialize attributes only with NON-default values. For examples, serializing a int variable with value zero is just going to take extra space however, choosing not to serialize it would save you a lot of performance. This approach can avoid some types of attributes taking unwanted space. This will require use of Externalizable interface since attribute serialization is determined at runtime based on the value of each attribute.
  + Use Externalizable interface and implement the readObject and writeObject methods to dynamically identify the attributes to be serialized. Some times there can be a custom logic used for serialization of various attributes.

## [What is a Serial Version UID (serialVersionUID) and why should I use it? How to generate one?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-is-a-serialversionuid-and-why-should-i-use-it--how-to-generate-one-" \o "What is a serialVersionUID and why should I use it? How to generate one?)

The serialVersionUID represents your class version, and you should change it if the current version of your class is not backwards compatible with its earlier versions. This is extract from Java API Documentation

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

Most of the times, we probably do not use serialization directly. In such cases, I would suggest to generate a default serializable uid by clicking the quick fix option in eclipse.

## [What would happen if the SerialVersionUID of an object is not defined?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-would-happen-if-the-serialversionuid-of-an-object-is-not-defined-" \o "What would happen if the SerialVersionUID of an object is not defined?)

If you don't define serialVersionUID in your serilizable class, Java compiler will make one by creating a hash code using most of your class attributes and features. When an object gets serialized, this hash code is stamped on the object which is known as the SerialVersionUID of that object. This ID is required for the version control of an object. SerialVersionUID can be specified in the class file also. In case, this ID is not specified by you, then Java compiler will regenerate a SerialVersionUID based on updated class and it will not be possible for the already serialized class to recover when a class field is added or modified. Its recommended that you always declare a serialVersionUID in your Serializable classes.

## [Does setting the serialVersionUID class field improve Java serialization performance?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "does-setting-the-serialversionuid-class-field-improve-java-serialization-performance-" \o "Does setting the serialVersionUID class field improve Java serialization performance?)

Declaring an explicit serialVersionUID field in your classes saves some CPU time only the first time the JVM process serializes a given Class. However the gain is not significant, In case when you have not declared the serialVersionUID its value is computed by JVM once and subsequently kept in a soft cache for future use.

## [What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-the-alternatives-to-serialization--if-serialization-is-not-used--is-it-possible-to-persist-or-transfer-an-object-using-any-other-approach-" \o "What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?)

In case, Serialization is not used, Java objects can be serialized by many ways, some of the popular methods are listed below:

* + Saving object state to database, this is most common technique used by most applications. You can use ORM tools (e.g. hibernate) to save the objects in a database and read them from the database.
  + Xml based data transfer is another popular mechanism, and a lot of XML based web services use this mechanism to transfer data over network. Also a lot of tools save XML files to persist data/configurations.
  + JSON Data Transfer - is recently popular data transfer format. A lot of web services are being developed in JSON due to its small footprint and inherent integration with web browser due to JavaScript format.

## [What are transient variables? What role do they play in Serialization process?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-are-transient-variables--what-role-do-they-play-in-serialization-process-" \o "What are transient variables? What role do they play in Serialization process?)

The transient keyword in Java is used to indicate that a field should not be serialized. Once the process of de-serialization is carried out, the transient variables do not undergo a change and retain their default value. Marking unwanted fields as transient can help you boost the serialization performance. Below is a simple example where you can see the use of transient keyword.

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + class MyVideo implements Serializable
  + {
  + private Video video;
  + private transient Image thumbnailVideo;
  + private void generateThumbnail()
  + {
  + // Generate thumbnail.
  + }
  + private void readObject(ObjectInputStream inputStream)
  + throws IOException, ClassNotFoundException
  + {
  + inputStream.defaultReadObject();
  + generateThumbnail();
  + }
  + }

## [Why does serialization NOT save the value of static class attributes? Why static variables are not serialized?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "why-does-serialization-not-save-the-value-of-static-class-attributes--why-static-variables-are-not-serialized-" \o "Why does serialization NOT save the value of static class attributes? Why static variables are not serialized?)

The Java variables declared as static are not considered part of the state of an object since they are shared by all instances of that class. Saving static variables with each serialized object would have following problems

* + It will make redundant copy of same variable in multiple objects which makes it in-efficient.
  + The static variable can be modified by any object and a serialized copy would be stale or not in sync with current value.

## [How to Serialize a collection in java? How to serialize a ArrayList, Hashmap or Hashset object in Java?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "how-to-serialize-a-collection-in-java--how-to-serialize-a-arraylist--hashmap-or-hashset-object-in-java-" \o "How to Serialize a collection in java? How to serialize a ArrayList, Hashmap or Hashset object in Java?)

All standard implementations of collections List, Set and Map interface already implement java.io.Serializable. All the commonly used collection classes like java.util.ArrayList, java.util.Vector, java.util.Hashmap, java.util.Hashtable, java.util.HashSet, java.util.TreeSet do implement Serializable. This means you do not really need to write anything specific to serialize collection objects. However you should keep following things in mind before you serialize a collection object - Make sure all the objects added in collection are Serializable. - Serializing the collection can be costly therefore make sure you serialize only required data isntead of serializing the whole collection. - In case you are using a custom implementation of Collection interface then you may need to implement serialization for it.

## [Is it possible to customize the serialization process? How can we customize the Serialization process?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "is-it-possible-to-customize-the-serialization-process--how-can-we-customize-the-serialization-process-" \o "Is it possible to customize the serialization process? How can we customize the Serialization process?)

Yes, the serialization process can be customized. When an object is serialized, objectOutputStream.writeObject (to save this object) is invoked and when an object is read, ObjectInputStream.readObject () is invoked. What most people do not know is that Java Virtual Machine provides you with an option to define these methods as per your needs. Once this is done, these two methods will be invoked by the JVM instead of the application of the default serialization process. Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + private void writeObject(java.io.ObjectOutputStream out)
  + throws IOException
  + private void readObject(java.io.ObjectInputStream in)
  + throws IOException, ClassNotFoundException;
  + private void readObjectNoData()
  + throws ObjectStreamException;

## [How can a sub-class of Serializable super class avoid serialization? If serializable interface is implemented by the super class of a class, how can the serialization of the class be avoided?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "how-can-a-sub-class-of-serializable-super-class-avoid-serialization--if-serializable-interface-is-implemented-by-the-super-class-of-a-class--how-can-the-serialization-of-the-class-be-avoided-" \o "How can a sub-class of Serializable super class avoid serialization? If serializable interface is implemented by the super class of a class, how can the serialization of the class be avoided?)

In Java, if the super class of a class is implementing Serializable interface, it means that it is already serializable. Since, an interface cannot be unimplemented, it is not possible to make a class non-serializable. However, the serialization of a new class can be avoided. For this, writeObject () and readObject() methods should be implemented in your class so that a Not Serializable Exception can be thrown by these methods. And, this can be done by customizing the Java Serialization process. Below the code that demonstrates it

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + class MySubClass extends SomeSerializableSuperClass {
  + private void writeObject(java.io.ObjectOutputStream out)
  + throws IOException {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + private void readObject(java.io.ObjectInputStream in)
  + throws IOException, ClassNotFoundException {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + private void readObjectNoData()
  + throws ObjectStreamException; {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + }

## [What changes are compatible and incompatible to the mechanism of java Serialization?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "what-changes-are-compatible-and-incompatible-to-the-mechanism-of-java-serialization-" \o "What changes are compatible and incompatible to the mechanism of java Serialization?)

This is one of a difficult tricky questions and answering this correctly would mean you are an expert in Java Serialization concept.   
  
In an already serialized object, the most challenging task is to change the structure of a class when a new field is added or removed. As per the specifications of Java Serialization, addition of any method or field is considered to be a compatible change whereas changing of class hierarchy or non-implementation of Serializable interface is considered to be a non-compatible change. You can go through the Java serialization specification for the extensive list of compatible and non-compatible changes. If a serialized object need to be compatible with an older version, it is necessary that the newer version follows some rules for compatible and incompatible changes. A compatible change to the implementing class is one that can be applied to a new version of the class, which still keeps the object stream compatible with older version of same class. **Some Simple Examples of compatible changes are:**

* + Addition of a new field or class will not affect serialization, since any new data in the stream is simply ignored by older versions. the newly added field will be set to its default values when the object of an older version of the class is un marshaled.
  + The access modifiers change (like private, public, protected or default) is compatible since they are not reflected in the serialized object stream.
  + Changing a transient field to a non-transient field is compatible change since it is similar to adding a field.
  + Changing a static field to a non-static field is compatible change since it is also similar to adding a field.

**Some Simple Examples of incompatible changes are:**

* + Changing implementation from Serializable to Externalizable interface can not be done since this will result in the creation of an incompatible object stream.
  + Deleting a existing Serializable fields will cause a problem.
  + Changing a non-transient field to a transient field is incompatible change since it is similar to deleting a field.
  + Changing a non-static field to a static field is incompatible change since it is also similar to deleting a field.
  + Changing the type of a attribute within a class would be incompatible, since this would cause a failure when attempting to read and convert the original field into the new field.
  + Changing the package of class is incompatible. Since the fully-qualified class name is written as part of the object byte stream.

Java serialization is one of the most commonly misunderstood areas. Many developers still think its only used for saving objects on the file system.

## [Define Serialization? What do you mean by Serialization in Java?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#define-serialization--what-do-you-mean-by-serialization-in-java-)

Serialization is a mechanism by which you can save or transfer the state of an object by converting it to a byte stream. This can be done in java by implementing Serialiazable interface. Serializable is defined as a marker interface which needs to be implemented for transferring an object over a network or persistence of its state to a file. Since its a marker interface, it does not contain any methods. Implementation of this interface enables the conversion of object into byte stream and thus can be transferred. The object conversion is done by the JVM using its default serialization mechanism.

## [Why is Serialization required? What is the need to Serialize?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#why-is-serialization-required--what-is-the-need-to-serialize-)

Serialization is required for a variety of reasons. It is required to send across the state of an object over a network by means of a socket. One can also store an object’s state in a file. Additionally, manipulation of the state of an object as streams of bytes is required. The core of Java Serialization is the Serializable interface. When Serializable interface is implemented by your class it provides an indication to the compiler that java Serialization mechanism needs to be used to serialize the object.

## [What is the Difference between Externalizable and Serializable Interfaces?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-is-the-difference-between-externalizable-and-serializable-interfaces-)

This is one of top serialization questions that is asked in many big companies to test your in-depth understanding of serialization. Serializable is a marker interface therefore you are not forced to implement any methods, however Externalizable contains two methods readExternal() and writeExternal() which must be implemented. Serializable interface provides a inbuilt serialization mechanism to you which can be in-efficient at times. However Externilizable interface is designed to give you greater control over the serialization mechanism. The two methods provide you immense opportunity to enhance the performance of specific object serialization based on application needs. Serializable interface provides a default serialization mechanism, on the other hand, Externalizable interface instead of relying on default Java Serialization provides flexibility to control this mechanism. One can drastically improve the application performance by implementing the Externalizable interface correctly. However there is also a chance that you may not write the best implementation, so if you are not really sure about the best way to serialize, I would suggest your stick to the default implementation using Serializable interface.

## [When will you use Serializable or Externalizable interface? and why?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#when-will-you-use-serializable-or-externalizable-interface--and-why-)

Most of the times when you want to do a selective attribute serialization you can use Serializable interface with transient modifier for variables not to be serialized. However, use of Externalizable interface can be really effective in cases when you have to serialize only some dynamically selected attributes of a large object. Lets take an example, Some times when you have a big Java object with hundreds of attributes and you want to serialize only a dozen dynamically selected attributes to keep the state of the object you should use Externalizable interface writeObject method to selectively serialize the chosen attributes. In case you have small objects and you know that most or all attributes are required to be serialized then you should be fine with using Serializable interface and use of transient variable as appropriate.

## [What are the ways to speed up Object Serialization? How to improve Serialization performance?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-are-the-ways-to-speed-up-object-serialization--how-to-improve-serialization-performance-)

The default Java Serialization mechanism is really useful, however it can have a really bad performance based on your application and business requirements. The serialization process performance heavily depends on the number and size of attributes you are going to serialize for an object. Below are some tips you can use for speeding up the marshaling and un-marshaling of objects during Java serialization process.

* + Mark the unwanted or non Serializable attributes as transient. This is a straight forward benefit since your attributes for serialization are clearly marked and can be easily achieved using Serialzable interface itself.
  + Save only the state of the object, not the derived attributes. Some times we keep the derived attributes as part of the object however serializing them can be costly. Therefore consider calcualting them during de-serialization process.
  + Serialize attributes only with NON-default values. For examples, serializing a int variable with value zero is just going to take extra space however, choosing not to serialize it would save you a lot of performance. This approach can avoid some types of attributes taking unwanted space. This will require use of Externalizable interface since attribute serialization is determined at runtime based on the value of each attribute.
  + Use Externalizable interface and implement the readObject and writeObject methods to dynamically identify the attributes to be serialized. Some times there can be a custom logic used for serialization of various attributes.

## [What is a Serial Version UID (serialVersionUID) and why should I use it? How to generate one?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-is-a-serialversionuid-and-why-should-i-use-it--how-to-generate-one-)

The serialVersionUID represents your class version, and you should change it if the current version of your class is not backwards compatible with its earlier versions. This is extract from Java API Documentation

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

Most of the times, we probably do not use serialization directly. In such cases, I would suggest to generate a default serializable uid by clicking the quick fix option in eclipse.

## [What would happen if the SerialVersionUID of an object is not defined?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-would-happen-if-the-serialversionuid-of-an-object-is-not-defined-)

If you don't define serialVersionUID in your serilizable class, Java compiler will make one by creating a hash code using most of your class attributes and features. When an object gets serialized, this hash code is stamped on the object which is known as the SerialVersionUID of that object. This ID is required for the version control of an object. SerialVersionUID can be specified in the class file also. In case, this ID is not specified by you, then Java compiler will regenerate a SerialVersionUID based on updated class and it will not be possible for the already serialized class to recover when a class field is added or modified. Its recommended that you always declare a serialVersionUID in your Serializable classes.

## [Does setting the serialVersionUID class field improve Java serialization performance?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#does-setting-the-serialversionuid-class-field-improve-java-serialization-performance-)

Declaring an explicit serialVersionUID field in your classes saves some CPU time only the first time the JVM process serializes a given Class. However the gain is not significant, In case when you have not declared the serialVersionUID its value is computed by JVM once and subsequently kept in a soft cache for future use.

## [What are the alternatives to Serialization? If Serialization is not used, is it possible to persist or transfer an object using any other approach?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-are-the-alternatives-to-serialization--if-serialization-is-not-used--is-it-possible-to-persist-or-transfer-an-object-using-any-other-approach-)

In case, Serialization is not used, Java objects can be serialized by many ways, some of the popular methods are listed below:

* + Saving object state to database, this is most common technique used by most applications. You can use ORM tools (e.g. hibernate) to save the objects in a database and read them from the database.
  + Xml based data transfer is another popular mechanism, and a lot of XML based web services use this mechanism to transfer data over network. Also a lot of tools save XML files to persist data/configurations.
  + JSON Data Transfer - is recently popular data transfer format. A lot of web services are being developed in JSON due to its small footprint and inherent integration with web browser due to JavaScript format.

## [What are transient variables? What role do they play in Serialization process?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-are-transient-variables--what-role-do-they-play-in-serialization-process-)

The transient keyword in Java is used to indicate that a field should not be serialized. Once the process of de-serialization is carried out, the transient variables do not undergo a change and retain their default value. Marking unwanted fields as transient can help you boost the serialization performance. Below is a simple example where you can see the use of transient keyword.

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + class MyVideo implements Serializable
  + {
  + private Video video;
  + private transient Image thumbnailVideo;
  + private void generateThumbnail()
  + {
  + // Generate thumbnail.
  + }
  + private void readObject(ObjectInputStream inputStream)
  + throws IOException, ClassNotFoundException
  + {
  + inputStream.defaultReadObject();
  + generateThumbnail();
  + }
  + }

## [Why does serialization NOT save the value of static class attributes? Why static variables are not serialized?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#why-does-serialization-not-save-the-value-of-static-class-attributes--why-static-variables-are-not-serialized-)

The Java variables declared as static are not considered part of the state of an object since they are shared by all instances of that class. Saving static variables with each serialized object would have following problems

* + It will make redundant copy of same variable in multiple objects which makes it in-efficient.
  + The static variable can be modified by any object and a serialized copy would be stale or not in sync with current value.

## [How to Serialize a collection in java? How to serialize a ArrayList, Hashmap or Hashset object in Java?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#how-to-serialize-a-collection-in-java--how-to-serialize-a-arraylist--hashmap-or-hashset-object-in-java-)

All standard implementations of collections List, Set and Map interface already implement java.io.Serializable. All the commonly used collection classes like java.util.ArrayList, java.util.Vector, java.util.Hashmap, java.util.Hashtable, java.util.HashSet, java.util.TreeSet do implement Serializable. This means you do not really need to write anything specific to serialize collection objects. However you should keep following things in mind before you serialize a collection object - Make sure all the objects added in collection are Serializable. - Serializing the collection can be costly therefore make sure you serialize only required data isntead of serializing the whole collection. - In case you are using a custom implementation of Collection interface then you may need to implement serialization for it.

## [Is it possible to customize the serialization process? How can we customize the Serialization process?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#is-it-possible-to-customize-the-serialization-process--how-can-we-customize-the-serialization-process-)

Yes, the serialization process can be customized. When an object is serialized, objectOutputStream.writeObject (to save this object) is invoked and when an object is read, ObjectInputStream.readObject () is invoked. What most people do not know is that Java Virtual Machine provides you with an option to define these methods as per your needs. Once this is done, these two methods will be invoked by the JVM instead of the application of the default serialization process. Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + private void writeObject(java.io.ObjectOutputStream out)
  + throws IOException
  + private void readObject(java.io.ObjectInputStream in)
  + throws IOException, ClassNotFoundException;
  + private void readObjectNoData()
  + throws ObjectStreamException;

## [How can a sub-class of Serializable super class avoid serialization? If serializable interface is implemented by the super class of a class, how can the serialization of the class be avoided?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#how-can-a-sub-class-of-serializable-super-class-avoid-serialization--if-serializable-interface-is-implemented-by-the-super-class-of-a-class--how-can-the-serialization-of-the-class-be-avoided-)

In Java, if the super class of a class is implementing Serializable interface, it means that it is already serializable. Since, an interface cannot be unimplemented, it is not possible to make a class non-serializable. However, the serialization of a new class can be avoided. For this, writeObject () and readObject() methods should be implemented in your class so that a Not Serializable Exception can be thrown by these methods. And, this can be done by customizing the Java Serialization process. Below the code that demonstrates it

[view plainprint?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html)

* + class MySubClass extends SomeSerializableSuperClass {
  + private void writeObject(java.io.ObjectOutputStream out)
  + throws IOException {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + private void readObject(java.io.ObjectInputStream in)
  + throws IOException, ClassNotFoundException {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + private void readObjectNoData()
  + throws ObjectStreamException; {
  + throw new NotSerializableException(“Can not serialize this class”);
  + }
  + }

## [What changes are compatible and incompatible to the mechanism of java Serialization?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html#what-changes-are-compatible-and-incompatible-to-the-mechanism-of-java-serialization-)

This is one of a difficult tricky questions and answering this correctly would mean you are an expert in Java Serialization concept.   
  
In an already serialized object, the most challenging task is to change the structure of a class when a new field is added or removed. As per the specifications of Java Serialization, addition of any method or field is considered to be a compatible change whereas changing of class hierarchy or non-implementation of Serializable interface is considered to be a non-compatible change. You can go through the Java serialization specification for the extensive list of compatible and non-compatible changes. If a serialized object need to be compatible with an older version, it is necessary that the newer version follows some rules for compatible and incompatible changes. A compatible change to the implementing class is one that can be applied to a new version of the class, which still keeps the object stream compatible with older version of same class. **Some Simple Examples of compatible changes are:**

* + Addition of a new field or class will not affect serialization, since any new data in the stream is simply ignored by older versions. the newly added field will be set to its default values when the object of an older version of the class is un marshaled.
  + The access modifiers change (like private, public, protected or default) is compatible since they are not reflected in the serialized object stream.
  + Changing a transient field to a non-transient field is compatible change since it is similar to adding a field.
  + Changing a static field to a non-static field is compatible change since it is also similar to adding a field.

**Some Simple Examples of incompatible changes are:**

* + Changing implementation from Serializable to Externalizable interface can not be done since this will result in the creation of an incompatible object stream.
  + Deleting a existing Serializable fields will cause a problem.
  + Changing a non-transient field to a transient field is incompatible change since it is similar to deleting a field.
  + Changing a non-static field to a static field is incompatible change since it is also similar to deleting a field.
  + Changing the type of a attribute within a class would be incompatible, since this would cause a failure when attempting to read and convert the original field into the new field.
  + Changing the package of class is incompatible. Since the fully-qualified class name is written as part of the object byte stream.

Java serialization is one of the most commonly misunderstood areas. Many developers still think its only used for saving objects on the file system.