1. Local variables
   1. Only final modifier (No Access modifier) is available to the local variables. Also they are not initialized by default.
2. Externalizable and Serializable interface.
   1. Serializable: It’s a marker interface. Only instances of classes implementing this interface can be serialized/deserialized using ObjectOutputStream/ ObjectInputStream. We can write our own serialization implementation by defining writeObject (ObjectOutputStream) and readObject (ObjectInputStream) methods in the object to be serialized. For using default implementation of serialization process the following methods of object stream can be used along with the writeDefaultObject (ObjectOutputStream) and readDefaultObject (ObjectInputStream).

Externalizable: Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances. This can be done by using writeExternal(ObjectOutput) and readExternal(ObjectInput). If the object supports Externalizable, the writeExternal method is called. If the object does not support Externalizable and does implement Serializable, the object is saved using ObjectOutputStream. When an Externalizable object is reconstructed, an instance is created using the public no-arg constructor, and then the readExternal method is called. Serializable objects are restored by reading them from an ObjectInputStream.

1. Transient and Volatile.
   1. Transient modifier implies that this field is not to be serialized, and thus is excluded from the serialization and deserialization process.

Volatile modifier is used in case of multithreaded access to variable. It makes the accessor thread to synchronize the thread local copy of variable with main copy upon each access. Also the order of synchronization is the same as requested by the thread.

1. What are all the methods used for Inter Thread communication and what is the class in which these methods are defined?
   1. The methods are wait, notify, notifyAll defined in class Object.

|  |  |
| --- | --- |
| Method | Description |
| wait( ) | It indicates the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls method notify() or notifyAll(). |
| notify( ) | It wakes up the first thread that called wait() on the same object. |
| notifyAll( ) | Wakes up (Unlock) all the threads that called wait( ) on the same object. The highest priority thread will run first. |

1. Thread.interrupt() and Thread.join()
   1. Thread.interrupt() will cause a change in the interrupt status of the thread by setting it to true, if the thread is in a blocked state (sleep, wait, notify, notifyAll) then it will throw an interrupted exception and come out of its current state, else invoking thread.isInterrupted() will give the interruption status.

Also calling Thread.interrupted() will reset the interrupt status and return the interrupt status.

Thread.join() , when called on a thread instance waits for the execution of that thread to complete before proceeding. Also we can specify the (no. of milliseconds, no. of nanaoseconds) to wait before proceeding. The actual waiting time is dependent on the underlying OS and is not guaranteed to be exactly the same as specified in the argument.

1. Synchronization with reference to stack, heap, static variables.
   1. Method variables: Since each thread execution has its own stack thus there is no need for synchronization of the local variables/ method level variables as they are not shared.

Instance variables: These are shared resources and hence require synchronization. For proper synchronization the executing thread needs to take a lock on the shared object(s).

Static variables: For accessing the static variables the thread need to take a lock on the class object.

1. Why wait and notify method in Object class?
   1. wait()/ notify() is invoked on an object during a thread execution under synchronized block. This implies that the current thread of execution wants to (take a lock/ release a lock) on the particular object, and since every class inherits Object class in java it makes all the more sense to have these methods in Object class.
2. In Java objects can be created in 4 ways.  
   1. Using new operator  
   2. First get class reference either using Class.forName() or class loader. On this class, create object using 'newInstance' method.  
   3. Clone  
   4. Deserialization
3. Null is a literal in Java. It cannot be used as an identifier.
4. What is Difference between Event-Based Framework and Request-Based Framework?
   1. In event based framework an event is triggered from within the system when a particular condition is met/ arises, while in request based framework a request is received from client which is then processed by the system.
5. Shallow copy and Deep Copy
   1. Shallow copy clone only parent object fields and copies references of the child objects, while in Deep copy clones the child objects too.
6. What is the difference between Static and Dynamic Polymorphisim?
   1. Static Polymorphism is shown by method overloading, Dynamic polymorphism is shown by method overriding.
7. Why Java is secured compare with other languages? Explain with example
   1. Absence of pointers to directly aceess the memory, Authentication mechanisms using Public key encryption, Providing isolation between executing components in different namespaces, Array bounds checking, The JVM’s class file verifier examines classes for basic class file structure upon loading. Note that, by default, the only trusted classes are the base classes. All other classes, including those loaded from the application classpath are considered untrusted and must be verified.
8. What is JIT Engine in Java?
   1. In a bytecode-compiled system, [source code](http://en.wikipedia.org/wiki/Source_code) is translated to an intermediate representation known as [bytecode](http://en.wikipedia.org/wiki/Bytecode). Bytecode is not the machine code for any particular computer, and may be portable among computer architectures. The bytecode may then be interpreted by, or run on, a [virtual machine](http://en.wikipedia.org/wiki/Virtual_machine). A just-in-time compiler can be used as a way to speed up execution of bytecode. At the time the bytecode is run, the just-in-time compiler will compile some or all of it to native machine code for better performance. This can be done per-file, per-function or even on any arbitrary code fragment; the code can be compiled when it is about to be executed (hence the name "just-in-time").
9. How to kill all threads at once in Java?

Using System.exit();

1. What is ClassDefNotFoundException and NoClassDefFoundError and explain differences between these two?
   1. ClassDefNotFoundException is thrown when an application tries to load in a class through its string name using:
      1. The forName method in class Class.
      2. The findSystemClass method in class ClassLoader.
      3. The loadClass method in class ClassLoader.

but no definition for the class with the specified name could be found.

NoClassDefFoundError is thrown if the Java Virtual Machine or a ClassLoader instance tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the new expression) and no definition of the class could be found.

The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

1. Directing output to a file can be done by passing an outputstream to the print methods.
2. Connection pooling design in multithreaded environment.
   1. Allocate one connection to individual thread. Implement locking/ synchronization on the connection objects, when a therad performs any database access. Find out if a connection object is in use/ locked by a thread otherwise return it ot the pool.
3. Soft Reference, Weak Reference and Phantom Reference.
   1. Phantom reachability indicates that an object is ready for reclamation. When the garbage collector determines that the referent of a phantom reference object is phantom reachable, it appends the phantom reference object to its associated reference queue. (Unlike soft and weak reference objects, which can optionally be created without associating them with a reference queue, phantom reference objects cannot be instantiated without associating the reference object with a reference queue.) You can use the arrival of a phantom reference in a reference queue to trigger some action that you wish to take at the end of an object's lifetime. Because you can't get a strong reference to a phantom reachable object (the get() method always returns null), you won't be able to take any action that requires you to have access to the instance variables of the target. Once you have finished the pre-mortem cleanup actions for a phantom reachable object, you must invoke clear() on the phantom reference objects that refer to it. Invoking clear() on a phantom reference object is the coup de gras for its referent, sending the referent from the phantom reachable state to its final resting place: unreachability.

Virtual machine implementations are required to clear soft references before throwing OutOfMemoryError, but are otherwise free to decide when or whether to clear them. Implementations are encouraged, however, to clear soft references only when the programs demand for memory exceeds the supply, to clear older soft references before newer ones, and to clear soft references that haven't been used recently before soft references that have been used recently. Soft references enable you to cache in memory data that you can more slowly retrieve from an external source, such as a file, database, or network.

Weak references are similar to soft references, except that whereas the garbage collector is free to decide whether or not to clear soft references to softly reachable objects, it must clear weak references to weakly reachable objects as soon as it determines the objects are weakly reachable.

1. Calling multiple applications using simple command in java
   1. Use Runtime.getRuntime.exec(String[] )Pass a String array containing the cmd.exe command as the first element and rest of the elements represent the names of applications to run.
2. Increasing JVM Memory.
   1. java -Xm256m -Xmx256m
3. Why return type is not part of method signature?
   1. I think you all have the experience that you can make a function call with checking the return or discarding the return. To have this flexibility, method and paramters of the method combination has to be unique.
4. What is the use of marker interface?
   1. In java language programming, interfaces with no methods are known as marker interfaces. Marker interfaces are Serializable, Clonable, SingleThreadModel, Event listener. Marker Interfaces are implemented by the classes or their super classes in order to add some functionality.e.g.  Suppose you want to persist (save) the state of an object then you have to implement the Serializable interface otherwise the compiler will throw an error. To make more clearly understand the concept of marker interface you should go through one more example. Suppose the interface Clonable is neither implemented by a class named Myclass nor it's any super class, then a call to the method clone() on Myclass's object will give an error. This means, to add this functionality one should implement the Clonable interface. While the Clonable is an empty interface but it provides an important functionality.
5. ArrayList vs Vector

|  |  |
| --- | --- |
| Array List  Unsynchronized, not thread safe  No default size  Increase by 50% of its size  Array list doesn’t need iterator to display it contents   Array list doesn’t defines any increment  size | Vector  Synchronized , thread safe  Default size 10  Doubling the size of array  Needs iterator to display its content  Vector does |

1. Why no pointers in java?
   1. Java does not have pointers. But Java does have references. A reference is an abstract identifier for an object. It is not a pointer. A reference tags a particular object with a name in the Java virtual machine so that the programmer may refer to it. How exactly the virtual machine implements references at the level of machine code is VM-dependent and completely hidden from the programmer in any case. Most VMs including Sun’s use handles, not pointers. A handle is a pointer to a pointer. At the level of machine code in the CPU a reference is an address in memory where the address of the object is stored. This way the objects can be moved around in memory and only the master pointer needs to be updated rather than all references to the object. This is completely hidden from the Java programmer, though. Only the implementer of the virtual machine needs to worry about it. Indeed, this is not the only way references can be implemented. Microsoft’s VM actually does use pointers rather than handles. Other schemes are possible.

Most studies agree that pointers are one of the primary features that enable programmers to inject bugs into their code. Given that structures are gone, and arrays and strings are objects, the need for pointers to these constructs goes away. Thus, Java has no pointer data types. Any task that would require arrays, structures, and pointers in C can be more easily and reliably performed by declaring objects and arrays of objects. Instead of complex pointer manipulation on array pointers, you access arrays by their arithmetic indices. The Java run-time system checks all array indexing to ensure indices are within the bounds of the array. You no longer have dangling pointers and trashing of memory because of incorrect pointers, because there are no pointers in Java.

1. Alternative to Inheritance.
   1. Delegation is an alternative to inheritance. Delegation means that you include an instance of another class as an instance variable, and forward messages to the instance. It is often safer than inheritance because it forces you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense. On the other hand, it makes you write more code, and it is harder to re-use
2. What is the order of initialization of variables in Java?
   1. i. Static initializer blocks are executed.

ii. Instance initialize blocks are executed.

iii. Constructors in the order of inherited classes from parent to child. If a constructor has a this() call then it has to be the first statement else super() is default if not expilicitly specified. The super class variables are initialized first and then process goes down to the child classes.

1. Interpretation and compilation.
   1. With Java the source first gets compiled into bytecode in class files. Then at runtime the bytecode is interpreted, and may optionally be compiled to native machine code. When I say "optionally", what that means is that almost always it will get compiled. Using only interpretation makes for much slower programs.
2. Memory Leak avoidance in Java.
   1. Memory Leak Types include:

“Traditional” memory leaks

Heap keeps growing....

OutOfMemoryError

“Temporary” memory leaks

Heap usage is temporarily very high, then it decrease

Bursts of frequent GCs

Memory Leak Sources

Object in the wrong scope/ Static fields.

Lapsed listeners

Exceptions change control flow

Instances of inner classes

Metadata mismanagement

Use of finalizers/reference objects

Monitor VM’s heap usage with tools, get reachability analysis.

1. *Iterator: ---* Iterator takes the place of Enumeration in the Java collections framework. One can traverse through the collection with the help of iterator in forward direction only and Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics

*ListIterator*:--An iterator for lists that allows one to traverse the list in either direction.modify the list during iteration and obtain the iterator's current position in the list. A *ListIterator* has no current element. its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next(). In a list of length n there are n+1 valid index values from 0 to n inclusive.

1. Ways to prevent a class from being inherited.
   1. Make the class final. Define the class with default access so it will not be extended outside the package.
2. Class Path and Path
   1. Set the path variable if you want to be able to run the executables (javac, java, javadoc, and so on) from any directory without having to type the full path of the command.

The CLASSPATH variable is one way to tell applications, including the JDK tools, where to look for user classes. Classes that are part of the JRE, JDK platform, and extensions should be defined through other means, such as the bootstrap class path or the extensions directory. The preferred way to specify the class path is by using the -cp command line switch. This allows the CLASSPATH to be set individually for each application without affecting other applications. Setting the CLASSPATH can be tricky and should be performed with care. The default value of the class path is ".", meaning that only the current directory is searched. Specifying either the CLASSPATH variable or the -cp command line switch overrides this value. A class path entry that contains \* will not match class files. To match both classes and JAR files in a single directory foo, use either foo;foo/\* or foo/\*;foo. The order chosen determines whether the classes and resources in foo are loaded before JAR files in foo, or vice versa. Subdirectories are not searched recursively. For example, foo/\* looks for JAR files only in foo, not in foo/bar, foo/baz, etc.

1. Daemon Threads.
   1. These threads continue to run even after all the application threads have finished execution. To avoid daemon threads continue execution which might be using resources we use the setDaemon(true) method to set the thread to true which will enable the JVM to kill it after all the application threads have run to completion. Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation for the java runtime system.
2. Native with example.
   1. Write java code.

Compile Java Code.

Create C header file.

Create C stubs file.

Write C code.

Create shared code library.

Run application.

For further reference http://www.javaworld.com/javaworld/javatips/jw-javatip23.html

1. Types of JDBC drivers
   1. Type 1: JDBC-ODBC Bridge plus ODBC Driver: The first type of JDBC driver is the JDBC-ODBC Bridge. It is a driver that provides JDBC access to databases through ODBC drivers. The ODBC driver must be configured on the client for the bridge to work. This driver type is commonly used for prototyping or when there is no JDBC driver available for a particular DBMS.  
      Type 2: Native-API partly-Java Driver: The Native to API driver converts JDBC commands to DBMS-specific native calls. This is much like the restriction of Type 1 drivers. The client must have some binary code loaded on its machine. These drivers do have an advantage over Type 1 drivers because they interface directly with the database.  
      Type 3: JDBC-Net Pure Java Driver: The JDBC-Net drivers are a three-tier solution. This type of driver translates JDBC calls into a database independent network protocol that is sent to a middleware server. This server then translates this DBMS independent protocol into a DBMS-specific protocol, which is sent to a particular database. The results are then routed back through the middleware server and sent back to the client. This type of solution makes it possible to implement a pure Java client. It also makes it possible to swap databases without affecting the client.  
      Type 4: Native-Protocol Pure Java Driver: These are pure Java drivers that communicate directly with the vendor's database. They do this by converting JDBC commands directly into the database engine's native protocol. This driver has no additional translation or middleware layer, which improves performance tremendously.
2. What are the steps in the JDBC connection?
   1. While making a JDBC connection we go through the following steps :   
       Step 1 : Register the database driver by using :

Class.forName(\" driver classs for that specific database\" );

Step 2: Now create a database connection using:

Connection con = DriverManager.getConnection(url,username,password);

Step 3: Now create a query using:

Statement stmt = Connection.Statement(\"select \* from TABLE NAME\");

Step 4: Execute the query:

stmt.exceuteUpdate();

1. Map and HashTable
2. If you've used Hashtable, you're already familiar with the general basics of Map. (Of course, Map is an interface, while Hashtable is a concrete implementation.) The following are the major differences:
   * Map provides Collection views instead of direct support for iteration via Enumeration objects. Collection views greatly enhance the expressiveness of the interface, as discussed later in this section.
   * Map allows you to iterate over keys, values, or key-value pairs; Hashtable does not provide the third option.
   * Map provides a safe way to remove entries in the midst of iteration; Hashtable did not.
   * Finally, Map fixes a minor deficiency in the Hashtable interface. Hashtable has a method called contains, which returns true if the Hashtable contains a given value. Given its name, you'd expect this method to return true if the Hashtable contained a given key, because the key is the primary access mechanism for a Hashtable. The Map interface eliminates this source of confusion by renaming the method containsValue. Also, this improves the interface's consistency — containsValue parallels containsKey.
3. WeakHashMap vs IdentityHashMap
   1. WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage-collected when its key is no longer referenced outside of the WeakHashMap. This class provides the easiest way to harness the power of weak references. It is useful for implementing "registry-like" data structures, where the utility of an entry vanishes when its key is no longer reachable by any thread.

IdentityHashMap is an identity-based Map implementation based on a hash table. This class is useful for topology-preserving object graph transformations, such as serialization or deep-copying. To perform such transformations, you need to maintain an identity-based "node table" that keeps track of which objects have already been seen. Identity-based maps are also used to maintain object-to-meta-information mappings in dynamic debuggers and similar systems. Finally, identity-based maps are useful in thwarting "spoof attacks" that are a result of intentionally perverse equals methods because IdentityHashMap never invokes the equals method on its keys. An added benefit of this implementation is that it is fast.

1. Equals() and hashCode()

a. **The equals() Contract**

Pulled straight from the Java docs, the equals() contract says

1. It is **reflexive**. For any reference value x, x.equals(x) should return true.
2. It is **symmetric**. For any reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
3. It is **transitive**. For any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) must return true.
4. It is **consistent**. For any 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 object is modified.
5. For any non-null reference value x, x.equals(null) should return false.

**The hashCode() Contract**

Now coming to you straight from the fabulous Java API documentation for class Object, may we present (drum roll) the hashCode() contract:

1. 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.
2. 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.
3. 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.
4. If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?
   1. It is empty. But not null.
5. What environment variables do I need to set on my machine in order to be able to run Java programs?
   1. CLASSPATH and PATH are the two variables.
6. Can I import same package/class twice? Will the JVM load the package twice at runtime?
   1. One can import the same package or same class multiple times. Neither compiler nor JVM complains about it. And the JVM will internally load the class only once no matter how many times you import the same class.
7. There are three exceptions in which serialization does not necessarily read and write to the stream. These are:   
   1. Serialization ignores static fields, because they are not part of any particular state.  
   2. Base class fields are only handled if the base class itself is serializable.  
   3. Transient fields.
8. What is the difference between error and an exception?
   1. An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you cannot repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.).
9. Can main method be declared final?
   1. Yes, the main method can be declared final, in addition to being public static.
10. Shift operators can be applied only to integer or long types
11. What happens to the bits that fall off after shifting? What values of the bits are shifted in after the shift
    1. They are discarded. In case of signed left shift >> the new bits are set to zero, but in case of signed right shift it takes the value of most significant bit before the shift, that is if the most significant bit before shift is 0 it will introduce 0, else if it is 1, it will introduce 1
12. What are the rules for primitive arithmetic promotion conversion?
    1. For Unary operators :

If operand is byte, short or a char it is converted to an int  
if it is any other type it is not converted

For binary operands:

If one of the operands is double, the other operand is converted to double  
Else If one of the operands is float, the other operand is converted to float  
Else If one of the operands is long, the other operand is converted to long  
Else both the operands are converted to int

1. Can RMI and CORBA based applications interact?  
   a. Yes they can. RMI is available with IIOP as the transport protocol instead of JRMP
2. What is the difference between RMI & CORBA?  
   a. Remote method invocation: java only. Uses JRMP to communication between java objects  
   Common object request broker architecture: platform independent. Uses IIOP to communicate between objects
3. Why java is not a 100% oops?
   1. Multiple inheritance, primitive types
4. What is a resource bundle?
   1. Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale.
5. How are Observer and Observable used?
   1. Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.
6. How does Java handle integer overflows and underflows?
   1. All the lower order bits are used.
7. What is the difference between the >> and >>> operators?
   1. Signed Right shift and Unsigned Right shift
8. How many bits are used to represent Unicode, ASCII, UTF-16, and UTF-8 characters?
   1. Unicode requires 16 bits and ASCII require 7 bits. Although the ASCII character set uses only 7 bits, it is usually represented as 8 bits. UTF-8 represents characters using 8, 16 and 18 bit patterns. UTF-16 uses 16-bit and larger bit patterns.
9. What is the purpose of the Runtime class?
   1. Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime method.
10. Which Java operator is right associative?
    1. The ternary operator, assignment operators (12 in all).
11. How is rounding performed under integer division?
    1. The fractional part of the result is truncated. This is known as rounding toward zero
12. What is the purpose of the System class?
    1. The System class contains several useful class fields and methods. It cannot be instantiated. Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.
13. What is an I/O filter?
    1. An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.
14. What are three ways in which a thread can enter the waiting state?
    1. A Thread can enter the waiting state by invoking it's sleep() method. By blocking on i/o by unsuccessfully attempting to acquire an object's lock or by invoking an object's wait() method.
15. How to Make Updates to Updatable Result Sets?
    1. Using JDBC 2.0 API we have the ability to update rows in a result set.

For this we need to create a ResultSet object that is updatable.

For this, we pass the ResultSet constant CONCUR\_UPDATABLE to the createStatement method.

Connection con = DriverManager.getConnection(url, “myLogin”, “myPassword”);

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSet uprs = stmt.executeQuery(”SELECT NAME, SALARY FROM EMPLOYEES”);

1. How can you move the cursor in scrollable result sets?
   1. In JDBC 2.0 API we have the ability to move a result set cursor backward as well as forward. We can also move the cursor to a particular row and check the position of the cursor.

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

ResultSet srs = stmt.executeQuery(SELECT NAME, SALARY FROM EMPLOYEES);

The first argument is one of three constants added to the ResultSet API to indicate the type of a ResultSet object: TYPE\_FORWARD\_ONLY, TYPE\_SCROLL\_INSENSITIVE , and TYPE\_SCROLL\_SENSITIVE. The second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable: CONCUR\_READ\_ONLY and CONCUR\_UPDATABLE.

Make sure that when you specify a type, you must also specify whether it is read-only or updatable. Specifying the constant TYPE\_FORWARD\_ONLY creates a nonscrollable result set, that is, one in which the cursor moves only forward. If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE\_FORWARD\_ONLY and CONCUR\_READ\_ONLY.

1. What is the query used to display all tables names in SQL Server (Query analyzer)?
   1. select \* from information\_schema.tables
2. I made my class Cloneable but I still get can’t access protected method clone. Why?
   1. Some of the Java books imply that all you have to do in order to have your class support clone() is implement the Cloneable interface. Not so. Perhaps that was the intent at some point, but that is not the way it works currently. As it stands, you have to implement your own public clone() method, even if it doesn’t do anything special and just calls super.clone().
3. What is the difference between TYPE\_SCROLL\_INSENSITIVE, and TYPE\_SCROLL\_SENSITIVE?
   1. You will get a scrollable ResultSet object if you specify one of these ResultSet constants.The difference between the two has to do with whether a result set reflects changes that are made to it while it is open and whether certain methods can be called to detect these changes. Generally speaking, a result set that is TYPE\_SCROLL\_INSENSITIVE does not reflect changes made while it is still open and one that is TYPE\_SCROLL\_SENSITIVE does. All three types of result sets will make changes visible if they are closed and then reopened:

Statement stmt =

con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

ResultSet srs =

stmt.executeQuery(”SELECT NAME, SALARY FROM PERSON”);

srs.afterLast();

while (srs.previous())

{

String name = srs.getString(”NAME”);

float salary = srs.getFloat(”SALARY”);

System.out.println(name + ” ” + salary);

}

1. How do you insert images in Database using JDBC?
   1. We can store images in the database using the BLOB datatype where in the image is stored as a byte stream
2. What is Metadata?

It is information about one of two things: Database information (java.sql.DatabaseMetaData), or Information about a specific ResultSet (java.sql.ResultSetMetaData). 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.

A usage of metadata is as follows: (similarly we can retrieve other DB objects e.g. Stored Procedures, Functions etc.)

Connection connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);

DatabaseMetaData metadata = connection.getMetaData();

ResultSet resultSet;

resultSet = metadata.getTables(null, null, "tablename", null);

1. I have the choice of manipulating database data using a byte[]/String or a java.sql.Blob /Clob. Which has best performance?
   1. java.sql.Blob/Clob, since it does not extract any data from the database until you explicitly ask it to. The Java platform 2 type Blob wraps a database locator (which is essentially a pointer to byte). That pointer is a rather large number (between 32 and 256 bits in size) – but the effort to extract it from the database is insignificant next to extracting the full blob content. For insertion into the database, you should use a byte[]/String since data has not been uploaded to the database yet. Thus, use the Blob/Clob class only for extraction.
2. How can I retrieve only the first n rows, second n rows of a database using a particular WHERE clause? For example, if a SELECT typically returns a 1000 rows, how do first retrieve the 100 rows, then go back and retrieve the next 100 rows and soon?
   1. Use the Statement.setFetchSize method to indicate the size of each database fetch. Note that this method is only available in the Java 2 platform. For Jdk 1.1.X and Jdk 1.0.X, no standardized way of setting the fetch size exists. Please consult the Db driver manual.
3. What does ResultSet actually contain? Is it the actual data of the result or some links to databases? If it is the actual data then why can’t we access it after connection is closed?
   1. A ResultSet is an interface. Its implementation depends on the driver and hence, what it “contains” depends partially on the driver and what the query returns. For example with the Odbc Bridge what the underlying implementation layer contains is an ODBC result set. A Type 4 driver executing a stored procedure that returns a cursor – on an oracle database it actually returns a cursor in the database. The oracle cursor can however be processed like a ResultSet would be from the client.

Closing a connection closes all interaction with the database and releases any locks that might have been obtained in the process.

1. What are SQL3 data types?
   1. The next version of the ANSI/ISO SQL standard defines some new datatypes, commonly referred to as the SQL3 types. The primary SQL3 types are:

STRUCT: This is the default mapping for any SQL structured type, and is manifest by the java.sql.Struct type.

REF: Serves as a reference to SQL data within the database. Can be passed as a parameter to a SQL statement. Mapped to the java.sql.Ref type.

BLOB: Holds binary large objects. Mapped to the java.sql.Blob type.

CLOB: Contains character large objects. Mapped to the java.sql.Clob type.

ARRAY: Can store values of a specified type. Mapped to the java.sql.Array type.

1. How can I manage special characters (for example:” \_ ‘%) when I execute an INSERT query? If I don’t filter the quoting marks or the apostrophe, for example, the SQL string will cause an error.
   1. The characters “%” and “\_” have special meaning in SQL LIKE clauses (to match zero or more characters, or exactly one character, respectively). In order to interpret them literally, they can be preceded with a special escape character in strings, e.g. “\”. In order to specify the escape character used to quote these characters, include the following syntax on the end of the query:

{escape 'escape-character'}

For example, the query

SELECT NAME FROM IDENTIFIERS WHERE ID LIKE ‘\\_%’ {escape ‘\’} finds identifier names that begin with an underscore.

1. Will a call to PreparedStatement.executeQuery() always close the ResultSet from the previous executeQuery()?
   1. A ResultSet is automatically closed by the Statement that generated it when that Statement is closed, re-executed, or is used to retrieve the next result from a sequence of multiple results.
2. Are prepared statements faster because they are compiled? If so, where and when are they compiled?
   1. Prepared Statements aren’t actually compiled, but they are bound by the JDBC driver. Depending on the driver, Prepared Statements can be a lot faster – if you re-use them. Some drivers bind the columns you request in the SQL statement. When you execute Connection.prepareStatement(), all the columns bindings take place, so the binding overhead does not occur each time you run the Prepared Statement.
3. Is it possible to connect to multiple databases simultaneously? Can one extract/update data from multiple databases with a single statement?
   1. In general, subject, as usual, to the capabilities of the specific driver implementation, one can connect to multiple databases at the same time. At least one driver (and probably others) will also handle commits across multiple connections. Obviously one should check the driver documentation rather than assuming these capabilities.

As to the second part of the question, one needs special middleware to deal with multiple databases in a single statement or to effectively treat them as one database. DRDA (Distributed Relational Database Architecture — I, at least, make it rhyme with “Gerta”) is probably most commonly used to accomplish this.

Oracle has a product called Oracle Transparent Gateway for IBM DRDA and IBM has a product called DataJoiner that make multiple databases appear as one to your application. No doubt there are other products available. XOpen also has papers available regarding DRDA.

1. Why do I get an UnsupportedOperationException?
   1. JDBC 2.0, introduced with the 1.2 version of Java, added several capabilities to JDBC. Instead of completely invalidating all the older JDBC 1.x drivers, when you try to perform a 2.0 task with a 1.x driver, an UnsupportedOperationException will be thrown. You need to update your driver if you wish to use the new capabilities.
2. What areas should I focus on for the best performance in a JDBC application?
   1. These are few points to consider:

Use a connection pool mechanism whenever possible.

Use prepared statements. These can be beneficial, for example with DB specific escaping, even when used only once.

Use stored procedures when they can be created in a standard manner. Do watch out for DB specific SP definitions that can cause migration headaches.

Even though the jdbc promotes portability, true portability comes from NOT depending on any database specific data types, functions and so on.

Select only required columns rather than using select \* from Table xyz.

Always close Statement and ResultSet objects as soon as possible.

Write modular classes to handle database interaction specifics.

Always catch and handle database warnings and exceptions. Be sure to check for additional pending exceptions.

1. Is Class.forName(Drivername) the only way to load a driver? Can I instantiate the Driver and use the object of the driver?
   1. Yes, you can use the driver directly. Create an instance of the driver and use the connect method from the Driver interface. Note that there may actually be two instances created, due to the expected standard behavior of drivers when the class is loaded.
2. Can I ensure that my app has the latest data?
   1. Typically an application retrieves multiple rows of data, providing a snapshot at an instant of time. Before a particular row is operated upon, the actual data may have been modified by another program. When it is essential that the most recent data is provided, a JDBC 2.0 driver provides the ResultSet.refreshRow method.
3. What is the difference between setMaxRows(int) and SetFetchSize(int)? Can either reduce processing time?
   1. setFetchSize(int) defines the number of rows that will be read from the database when the ResultSet needs more rows. The method in the java.sql.Statement interface will set the ‘default’ value for all the ResultSet derived from that Statement; the method in the java.sql.ResultSet interface will override that value for a specific ResultSet. Since database fetches can be expensive in a networked environment, fetch size has an impact on performance.

setMaxRows(int) sets the limit of the maximum nuber of rows in a ResultSet object. If this limit is exceeded, the excess rows are “silently dropped”. That’s all the API says, so the setMaxRows method may not help performance at all other than to decrease memory usage. A value of 0 (default) means no limit.

1. What’s the best way, in terms of performance, to do multiple insert/update statements, a PreparedStatement or Batch Updates?
   1. Because PreparedStatement objects are precompiled, their execution can be faster than that of Statement objects. Consequently, an SQL statement that is executed many times is often created as a PreparedStatement object to increase efficiency.

A CallableStatement object provides a way to call stored procedures in a standard manner for all DBMSes. Their execution can be faster than that of PreparedStatement object.

Batch updates are used when you want to execute multiple statements together. Actually, there is no conflict here. While it depends on the driver/DBMS engine as to whether or not you will get an actual performance benefit from batch updates, Statement, PreparedStatement, and CallableStatement can all execute the addBatch() method.

1. How can I get information about foreign keys used in a table?
   1. DatabaseMetaData.getImportedKeys() returns a ResultSet with data about foreign key columns, tables, sequence and update and delete rules.
2. What isolation level is used by the DBMS when inserting, updating and selecting rows from a database?
   1. The answer depends on both your code and the DBMS. If the program does not explicitly set the isolation level, the DBMS default is used. You can determine the default using DatabaseMetaData.getDefaultTransactionIsolation() and the level for the current Connection with Connection.getTransactionIsolation(). If the default is not appropriate for your transaction, change it with Connection.setTransactionIsolation(int level).
3. What is JDO?
   1. The majority of applications need to persist (or store) data during their lifecycle. There are many ways of doing this with an application written in Java.

If your datastore is RDBMS you can handle the persistence (and retrieval) of data yourself using JDBC. Obviously with this route you have the burden of having to write the persistence layer yourself. This gives much control, but also creates significant work, both in writing the code but also in testing and maintenance.

You can use JDO, a standardized persistence API. With JDO you can develop plain old java objects (POJOs) and persist them as they are transparently. This requires very little work from the developer. It allows persistence to any type of datastore in principle, being designed with flexibility and datastore agnositicity in mind. This has been a standard since 2002 (JDO1), being upgraded in 2006 (JDO2) and is in the process of being developed further (JDO2.1) by Apache JDO

You can use JPA, a standardised persistence API, and part of the EJB3 specification. This also allows you to develop plain old Java objects (POJOs) and persist them using a standardised API. Its specification is not as mature or as feature rich as the JDO API, nor does it provide the flexibility of using any type of datastore. This was released in 2006 (JPA1) to supercede EJB2. It really only allows persistence to RDBMS datastores. If you want to persist to other datastores you should consider JDO.

If you are stuck with using an EJB2.\* architecture you could use Entity Beans. This means that you hand off your objects to the EJB part of the J2EE server. This simplifies things for the developer in some respect but places major restrictions in that your objects have to be Entity Beans.

You can also use a proprietary persistence API (e.g Hibernates own API, TopLinks own API, iBatis, Castor etc). The disadvantages of going this route are that you cannot easily swap to an alternative implementation of the API if you hit problems with your software choice.

1. What properties should I supply to a database driver in order to connect to a database?
   1. Driver Class Name, URL (HOST, PORT, Username, Password
2. At a glance, how does the Java Database Connectivity (JDBC) work?
   1. The main objects of the JDBC API include:

A DataSource object is used to establish connections. Although the Driver Manager can also be used to establish a connection, connecting through a DataSource object is the preferred method.

A Connection object controls the connection to the database. An application can alter the behavior of a connection by invoking the methods associated with this object. An application uses the connection object to create statements.

Statement, PreparedStatement, and CallableStatement objects are used for executing SQL statements. A PreparedStatement object is used when an application plans to reuse a statement multiple times. The application prepares the SQL it plans to use. Once prepared, the application can specify values for parameters in the prepared SQL statement. The statement can be executed multiple times with different parameter values specified for each execution. A CallableStatement is used to call stored procedures that return values. The CallableStatement has methods for retrieving the return values of the stored procedure.

A ResultSet object contains the results of a query. A ResultSet is returned to an application when a SQL query is executed by a statement object. The ResultSet object provides methods for iterating through the results of the query.

1. What are the pros and cons of annotations over XML based deployment descriptors?

Service related attributes in your application can be configured through a XML based deployment descriptor files or annotations. XML based deployment descriptor files are processed separately from the code, often at runtime, while annotations are compiled with your source code and checked by the compiler.

XML

More verbose because has to duplicate a lot of information like class names and method names from your code.

Less robust due to duplication of information which introduces multiple points for failure. If you misspell a method name then the application will fail.

More flexible since processed separately from the code. Since it is not hard-coded can be changed later. Your deployment team has a greater flexibility to inspect and modify the configuration.

XML files can express complex relationships and hierarchical structures at the expense of being verbose.

Annotations

Less verbose since class names and method names are part of yourcode.

More robust because annotations are processed with your code and checked by the compiler for any discrepancies and inaccuracies.

Less flexible since annotations are embedded in Java comment style within your code. For example, to define a stateless session EJB 3.0 with annotations, which can serve both local and remote clients:

@Stateless

@Local ({LocalCounter.class})

@Remote ({RemoteCounter.class})

public class CounterBean implements LocalCounter, RemoteCounter {

...

}

Annotations can hold only a small amount of configuration information and most of the plumbing has to be done in the framework.

Which one to use?

Annotations are suitable for most application needs. XML files are more complex and can be used to address more advanced issues. XML files can be used to override default annotation values. Annotations cannot be used if you do not have access to source-code. The decision to go with annotation or XML depends upon the architecture behind the framework. For example Spring is primarily based on XML and EJB 3.0 is primarily based on annotations, but both support annotations and XML to some degree. EJB 3.0 uses XML configuration files as an optional overriding mechanism and Spring uses annotations to configure some Spring services.

1. JAXB usage with Map types

For generating Map type out of an XSD, file must be created that targets appropriate elements in the schema and overrides their baseType with a fully-qualified custom Map subclass. Custom java files must be hand-created for the Map subclass, and also an XmlAdapter subclass which contains the logic to unmarshal/marshal to/from the Map subclass.

1. Unit testing enterprise components

Unit testing of Java EE 6 applications is no different than testing Java Platform, Standard Edition (Java SE). Java EE 6 components are just annotated classes. You should not treat them in a special way; instead, focus on the verification of the business logic.

Arquillian seeks to minimize the burden on the developer to carry out integration testing by handling all aspects of test execution, including:

managing the lifecycle of the container (start/stop),

bundling the test class with dependent classes and resources into a deployable archive,

enhancing the test class (e.g., resolving @Inject, @EJB and @Resource injections),

deploying the archive to test (deploy/undeploy) and

capturing results and failures.

1. Multiple applications in a container

Two ideas:

1. Multiple war in a ear - user APP-INF/ lib, specify the path in application.xml

2. Multiple ear - Put the libs on system classpath e.g. in JBOSS\_HOME/server/default/lib, deployed as a shared library in Weblogic

1. Spring shared context between multiple webapp

Assuming that spring configuration consists of beans specific to the web application concern (validators, controllers), to have bunch of beans share a single spring context, we use the 'locatorFactorySelector' and 'parentContextKey'. We simply add the following into our web.xml(s)

<context-param>

<param-name>locatorFactorySelector</param-name>

<param-value>classpath:common-beans.xml</param-value>

</context-param>

The above would mean that you would have a file called common-beans.xml in the classpath for the web application, which has the following bean configured;

<bean id="commonContext" class="org.springframework.context.support.ClassPathXmlApplicationContext">

<constructor-arg>

<list>

<value>classpath:service-beans.xml</value>

</list>

</constructor-arg>

</bean>

1. PowerMockito v/s Mockito

Szczepan Faber is the founder of the Mockito project. PowerMock was founded by Johan Haleby and Jan Kronquist.

Mockito is available under MIT License. PowerMock is available under Apache license 2.0.

Mockito does not include specific language characteristics like constructors or static methods for mocking.

PowerMock offers constructors and static methods to Mockito and other frameworks, through its individual classloader and bytecode management.

Mockito does not require ‘@RunWith’ annotation and base test class, while performing tests in suite.

PowerMock requires both ‘@RunWith’ annotation and a base test class for testing a suite.

Mockito does not support mocking of constructors.

PowerMock supports mocking of constructors and also supports mocking of (i) final (ii)

static (iii) native and (iv) private methods.

Mockito does not support mocking of ‘new’-ed objects. PowerMock supports mocking of ‘new’-ed objects.

Mockito contains a jar file in the classpath for supporting mocking APIs.

PowerMock is a Mockito API.

Mockito does not require any codes to be executed before a test.

PowerMock includes ‘preparation for test’ codes.

Mockito does not support mocking of enum data types. PowerMock supports mocking of enum data types.

1. Varargs vs Overloading

According to (JLS 15.2.2), there are 3 phases used in overload resolution: First phase performs overload resolution without permitting boxing or unboxing conversion, Second phase performs overload resolution while allowing boxing and unboxing and Third phase allows overloading to be combined with variable arity methods, boxing, and unboxing. If no applicable method is found during these phases, then ambiguity occurs.

1. Array[Integer] - toString()

Arrays.toString(intarray) prints elements.toString comma separated.

1. Overriding static method

Overriding depends on having an instance of a class. The point of polymorphism is that you can subclass a class and the objects implementing those subclasses will have different behaviors for the same methods defined in the superclass (and overridden in the subclasses). A static method is not associated with any instance of a class so the concept is not applicable. Hence no overriding happens for static method

1. Handling Multiple Exception types in catch block

In releases prior to Java SE 7, it is difficult to create a common method to eliminate the duplicated code because the variable ex has different types. The following example, which is valid in Java SE 7 and later, eliminates the duplicated code:

catch (IOException|SQLException ex) {

logger.log(ex);

throw ex;

}

The catch clause specifies the types of exceptions that the block can handle, and each exception type is separated with a vertical bar (|).

Note: If a catch block handles more than one exception type, then the catch parameter is implicitly final. In this example, the catch parameter ex is final and therefore you cannot assign any values to it within the catch block.

1. Order of finally and catch block

Finally block can only appear after the catch block if a catch block is present for the corresponding try block.

1. Exception broadening

It means that if a method declares to throw a given exception, the overriding method in a subclass can only declare to throw that exception or its subclass. For example:

class A {

public void foo() throws IOException {..}

}

class B extends A {

@Override

public void foo() throws SocketException {..} // allowed

@Override

public void foo() throws SQLException {..} // NOT allowed

}

SocketException extends IOException, but SQLException does not.

This is because of polymorphism:

A a = new B();

try {

a.foo();

} catch (IOException ex) {

// forced to catch this by the compiler

}

If B had decided to throw SQLException, then the compiler could not force you to catch it, because you are referring to the instance of B by its superclass - A. On the other hand, any subclass of IOException will be handled by clauses (catch or throws) that handle IOException

The rule that you need to be able to refer to objects by their superclass is the Liskov Substitution Principle.

Since unchecked exceptions can be thrown anywhere then they are not subject to this rule. You can add an unchecked exception to the throws clause as a form of documentation if you want, but the compiler doesn't enforce anything about it.

1. Overloading v/s Overriding - Child Class Example

Java Tutorials: Overloading is compile-time binding

Most beginners in Java get confused between Overloading and Overriding. One should understand that overloading is compile-time binding whereas overriding is runtime binding.

Have a look at the following example. There are three classes - Base, Derived and Test. As the name indicates class Derived extends class Base. The class Test has two overloaded methods with name methodA, with parameters Base and Derived respectively.

class Base{

}

class Derived extends Base{

}

class Test{

public void methodA(Base b){

System.out.println("Test.methodA(Base)");

}

public void methodA(Derived b){

System.out.println("Test.methodA(Derived)");

}

public static void main(String []args){

Test t = new Test();

Base b = new Base();

Base d = new Derived();

t.methodA(b);

t.methodA(d);

}

}

What is the output?

If your answer is

Test.methodA(Base)

Test.methodA(Derived)

This is wrong

For your surprise the answer is wrong. The actual output is

Test.methodA(Base)

Test.methodA(Base)

Surprised?

This is because overloading is compile-time binding. When the compiler sees the line t.methodA(d);

It checks the data type of 'd', which is declared as 'Base'. So it looks for the method, methodA(Base) and binds the call to this method and hence the result.

Let us look at another common problem. When the programmers think to override the 'equals', but endup really overloading the method, there by creating some unforeseen problems.

Have a look at the following code, and say whether the equals method and the hashCode are implemented in the correct way?

public class EqualsOverloadTest {

String id;

public EqualsOverloadTest(String id){

this.id = id;

}

public boolean equals(EqualsOverloadTest other){

return (other!=null) && this.id.equals(other.id);

}

public int hashCode() {

return id.hashCode();

}

}

In the first go, anyone will say the equals method is implemented correctly.

It follows all the constraints for the 'equals' method, and also implements the 'hashCode()' method following the same contract. But, if you look closer, you fill notice that, the 'equals' method really overloads the Object.equals(Object) method, instead of over loading it.

To prove that this won't work, let me give a simple program. In the main method, we are creating two EqualsOverloadTest objects with the same id. The two objects are added into the Set. Then we are printing the size of the set.

public static void main(String[] args) {

EqualsOverloadTest first = new EqualsOverloadTest("123");

EqualsOverloadTest second =

new EqualsOverloadTest(new String("123"));

System.out.println(first.equals(second));

Set set = new HashSet();

set.add(first);

set.add(second);

System.out.println(set.size());

}

We will expect the size of the Set to be '1' since the two Objects are equal. But it will print as '2'.

This is because we didn't override the 'equals' method. Whereas the first check with the equals method returned true, because we called the method as equals(EqualsOverloadTest), hence the proper method was called. But withing the set, it called the method equals(Object), which is not implemented, so uses the Object.equals(Object), which really checks whether they both are same instance or not. Hence we get an unexpected behaviour.

Summary:

Overloading is a static or compile-time binding and Overriding is dynamic or run-time binding.

1. How to define immutable object

Don't provide "setter" methods — methods that modify fields or objects referred to by fields.

Make all fields final and private.

Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.

If the instance fields include references to mutable objects, don't allow those objects to be changed:

Don't provide methods that modify the mutable objects.

Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

1. How to know about the .class file

1. class file in java is generated when you compile .java file using any Java compiler like Sun's javac which comes along JDK installation and can be found in JAVA\_HOME/bin directory.

2. class file contains byte codes. byte codes are special platform independent instruction for Java virtual machine. bytecode is not a machine language and mere instruction to JVM. since every machine or processor e.g. INTEL or AMD processor may have different instruction for doing same thing, its left on JVM to translate bytecode into machine instruction and by this way java achieves platform independence.

3. class file of inner class contains $ in there name. So if a Java source file contains two classes, one of them is inner class than java compiler will generate two class file. separate class file for top level and inner class. you can distinguish them by looking at there name. Suppose you have top level class as "Hello" and inner class as "GoodBye" then java compiler will generate two class file:

Hello.class

Hello$GoodBye.class

Hello$GoodBye.class is a class file for inner class. whose name is in format of top-class$inner-class.

4. You can look bytecode of class file using javap command. javap command can also display method and field information from .class file. see my post how to decompile .class file in Java for more details.

5. class file format is subject to change and its changed to support new feature introduced in Java 1.5. In general every java compiler and JRE comes with supported version of .class file format and you can not run a .class file which is in higher version of those supported by JRE. this often result in java.lang.UnsupportedClassVersion. class file has two version major and minor which is included inside class file. See my post Bad version number in .class file for more details.

6. .class file in java is identified by a magic number in header which is a 4 byte CA FE BA BE ( in hex). which is the first element in .class file, followed by major and minor versions of class file

The types u1, u2, and u4 represent an unsigned one-, two-, or four-byte quantity, respectively. In the Java SE platform, these types may be read by methods such as readUnsignedByte, readUnsignedShort, and readInt of the interface java.io.DataInput.

ClassFile {

u4 magic;

u2 minor\_version;

u2 major\_version;

u2 constant\_pool\_count;

cp\_info constant\_pool[constant\_pool\_count-1];

u2 access\_flags;

u2 this\_class;

u2 super\_class;

u2 interfaces\_count;

u2 interfaces[interfaces\_count];

u2 fields\_count;

field\_info fields[fields\_count];

u2 methods\_count;

method\_info methods[methods\_count];

u2 attributes\_count;

attribute\_info attributes[attributes\_count];

}