**What is the difference between procedural and object-oriented programs? -** a) In procedural program, programming logic follows certain procedures and the instructions are executed one after another. In OOP program, unit of program is object, which is nothing but combination of data and code. b) In procedural program, data is exposed to the whole program whereas in OOPs program, it is accessible within the object and which in turn assures the security of the code.

**What are Encapsulation, Inheritance and Polymorphism?** - Encapsulation is the mechanism that binds together code and data it manipulates and keeps both safe from outside interference and misuse. Inheritance is the process by which one object acquires the properties of another object. Polymorphism is the feature that allows one interface to be used for general class actions.

**What is the difference between Assignment and Initialization?** - Assignment can be done as many times as desired whereas initialization can be done only once.

**What is OOPs?** - Object oriented programming organizes a program around its data, i. e. , objects and a set of well defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code.

**What are Class, Constructor and Primitive data types?** - Class is a template for multiple objects with similar features and it is a blue print for objects. It defines a type of object according to the data the object can hold and the operations the object can perform. Constructor is a special kind of method that determines how an object is initialized when created. Primitive data types are 8 types and they are: byte, short, int, long, float, double, boolean, char.

**What is an Object and how do you allocate memory to it?** - Object is an instance of a class and it is a software unit that combines a structured set of data with a set of operations for inspecting and manipulating that data. When an object is created using new operator, memory is allocated to it.

**What is the difference between constructor and method?** - Constructor will be automatically invoked when an object is created whereas method has to be called explicitly.

**What are methods and how are they defined?** - Methods are functions that operate on instances of classes in which they are defined. Objects can communicate with each other using methods and can call methods in other classes. Method definition has four parts. They are name of the method, type of object or primitive type the method returns, a list of parameters and the body of the method. A method’s signature is a combination of the first three parts mentioned above.

**What is the use of bin and lib in JDK?** - Bin contains all tools such as javac, appletviewer, awt tool, etc., whereas lib contains API and all packages.

**What is casting?** - Casting is used to convert the value of one type to another.

**How many ways can an argument be passed to a subroutine and explain them?** - An argument can be passed in two ways. They are passing by value and passing by reference. Passing by value: This method copies the value of an argument into the formal parameter of the subroutine. Passing by reference: In this method, a reference to an argument (not the value of the argument) is passed to the parameter.

**What is the difference between a parameter and an argument?** - While defining method, variables passed in the method are called parameters. While using those methods, values passed to those variables are called arguments.

**What is final, finalize() and finally?**

final: final keyword can be used for class, method and variables. A final class cannot be subclassed and it prevents other programmers from subclassing a secure class to invoke insecure methods. A final method can’t be overridden. A final variable can’t change from its initialized value.

finalize() : finalize() method is used just before an object is destroyed and can be called just prior to garbage collection.

finally: finally, a key word used in exception handling, creates a block of code that will be executed after a try/catch block has completed and before the code following the try/catch block. The finally block will execute whether or not an exception is thrown. For example, if a method opens a file upon exit, then you will not want the code that closes the file to be bypassed by the exception-handling mechanism. This finally keyword is designed to address this contingency.

**What is UNICODE?** - Unicode is used for internal representation of characters and strings and it uses 16 bits to represent each other.

**What is Garbage Collection and how to call it explicitly?** - When an object is no longer referred to by any variable, java automatically reclaims memory used by that object. This is known as garbage collection. System. gc() method may be used to call it explicitly.

**What is finalize() method?** - finalize () method is used just before an object is destroyed and can be called just prior to garbage collection.

**What are Transient and Volatile Modifiers?** - Transient: The transient modifier applies to variables only and it is not stored as part of its object’s Persistent state. Transient variables are not serialized. Volatile: Volatile modifier applies to variables only and it tells the compiler that the variable modified by volatile can be changed unexpectedly by other parts of the program.

**What is method overloading and method overriding?** - Method overloading: When a method in a class having the same method name with different arguments is said to be method overloading. Method overriding: When a method in a class having the same method name with same arguments is said to be method overriding.

**What is difference between overloading and overriding? -** a) In overloading, there is a relationship between methods available in the same class whereas in overriding, there is relationship between a superclass method and subclass method. b) Overloading does not block inheritance from the superclass whereas overriding blocks inheritance from the superclass. c) In overloading, separate methods share the same name whereas in overriding, subclass method replaces the superclass. d) Overloading must have different method signatures whereas overriding must have same signature.

**What is meant by Inheritance and what are its advantages? -** Inheritance is the process of inheriting all the features from a class. The advantages of inheritance are reusability of code and accessibility of variables and methods of the super class by subclasses.

**What is the difference between this() and super()? -** this() can be used to invoke a constructor of the same class whereas super() can be used to invoke a super class constructor.

**What is the difference between superclass and subclass? -** A super class is a class that is inherited whereas sub class is a class that does the inheriting.

**What modifiers may be used with top-level class? -** public, abstract and final can be used for top-level class.

**What are inner class and anonymous class? -** Inner class : classes defined in other classes, including those defined in methods are called inner classes. An inner class can have any accessibility including private. Anonymous class : Anonymous class is a class defined inside a method without a name and is instantiated and declared in the same place and cannot have explicit constructors.

**What is a package? -** A package is a collection of classes and interfaces that provides a high-level layer of access protection and name space management.

**What is a reflection package? -** java. lang. reflect package has the ability to analyze itself in runtime.

**What is interface and its use? -** Interface is similar to a class which may contain method’s signature only but not bodies and it is a formal set of method and constant declarations that must be defined by the class that implements it. Interfaces are useful for: a)Declaring methods that one or more classes are expected to implement b)Capturing similarities between unrelated classes without forcing a class relationship. c)Determining an object’s programming interface without revealing the actual body of the class.

**What is an abstract class? -** An abstract class is a class designed with implementation gaps for subclasses to fill in and is deliberately incomplete.

**What is the difference between Integer and int? -** a) Integer is a class defined in the java. lang package, whereas int is a primitive data type defined in the Java language itself. Java does not automatically convert from one to the other. b) Integer can be used as an argument for a method that requires an object, whereas int can be used for calculations.

**What is a cloneable interface and how many methods does it contain? -** It is not having any method because it is a TAGGED or MARKER interface.

**What is the difference between abstract class and interface? -** a) All the methods declared inside an interface are abstract whereas abstract class must have at least one abstract method and others may be concrete or abstract. b) In abstract class, key word abstract must be used for the methods whereas interface we need not use that keyword for the methods. c) Abstract class must have subclasses whereas interface can’t have subclasses.

**Can you have an inner class inside a method and what variables can you access? -** Yes, we can have an inner class inside a method and final variables can be accessed.

**What is the difference between String and String Buffer? -** a) String objects are constants and immutable whereas StringBuffer objects are not. b) String class supports constant strings whereas StringBuffer class supports growable and modifiable strings.

**What is the difference between Array and vector? -** Array is a set of related data type and static whereas vector is a growable array of objects and dynamic.

**What is the difference between exception and error? -** The exception class defines mild error conditions that your program encounters. Exceptions can occur when trying to open the file, which does not exist, the network connection is disrupted, operands being manipulated are out of prescribed ranges, the class file you are interested in loading is missing. The error class defines serious error conditions that you should not attempt to recover from. In most cases it is advisable to let the program terminate when such an error is encountered.

**What is the difference between process and thread? -** Process is a program in execution whereas thread is a separate path of execution in a program.

**What is multithreading and what are the methods for inter-thread communication and what is the class in which these methods are defined? -** Multithreading is the mechanism in which more than one thread run independent of each other within the process. wait (), notify () and notifyAll() methods can be used for inter-thread communication and these methods are in Object class. wait() : When a thread executes a call to wait() method, it surrenders the object lock and enters into a waiting state. notify() or notifyAll() : To remove a thread from the waiting state, some other thread must make a call to notify() or notifyAll() method on the same object.

**What is the class and interface in java to create thread and which is the most advantageous method? -** Thread class and Runnable interface can be used to create threads and using Runnable interface is the most advantageous method to create threads because we need not extend thread class here.

**What are the states associated in the thread? -** Thread contains ready, running, waiting and dead states.

**What is synchronization? -** Synchronization is the mechanism that ensures that only one thread is accessed the resources at a time.

**When you will synchronize a piece of your code? -** When you expect your code will be accessed by different threads and these threads may change a particular data causing data corruption.

**What is deadlock? -** When two threads are waiting each other and can’t precede the program is said to be deadlock.

**What is daemon thread and which method is used to create the daemon thread? -** Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation for the java runtime system. setDaemon method is used to create a daemon thread.

**Are there any global variables in Java, which can be accessed by other part of your program? -** No, it is not the main method in which you define variables. Global variables is not possible because concept of encapsulation is eliminated here.

**What is an applet? -** Applet is a dynamic and interactive program that runs inside a web page displayed by a java capable browser.

**What is the difference between applications and applets? -** a)Application must be run on local machine whereas applet needs no explicit installation on local machine. b)Application must be run explicitly within a java-compatible virtual machine whereas applet loads and runs itself automatically in a java-enabled browser. d)Application starts execution with its main method whereas applet starts execution with its init method. e)Application can run with or without graphical user interface whereas applet must run within a graphical user interface.

**How does applet recognize the height and width? -** Using getParameters() method.

**When do you use codebase in applet? -** When the applet class file is not in the same directory, codebase is used.

**What is the lifecycle of an applet? -** init() method - Can be called when an applet is first loaded start() method - Can be called each time an applet is started. paint() method - Can be called when the applet is minimized or maximized. stop() method - Can be used when the browser moves off the applet’s page. destroy() method - Can be called when the browser is finished with the applet.

**How do you set security in applets? -** using setSecurityManager() method

**What is an event and what are the models available for event handling? -** An event is an event object that describes a state of change in a source. In other words, event occurs when an action is generated, like pressing button, clicking mouse, selecting a list, etc. There are two types of models for handling events and they are: a) event-inheritance model and b) event-delegation model

**What are the advantages of the model over the event-inheritance model? -** The event-delegation model has two advantages over the event-inheritance model. They are: a)It enables event handling by objects other than the ones that generate the events. This allows a clean separation between a component’s design and its use. b)It performs much better in applications where many events are generated. This performance improvement is due to the fact that the event-delegation model does not have to be repeatedly process unhandled events as is the case of the event-inheritance.

**What is source and listener? -** source : A source is an object that generates an event. This occurs when the internal state of that object changes in some way. listener : A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.

**What is adapter class? -** An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act listener by extending one of the adapter classes and implementing only those events in which you are interested. For example, the MouseMotionAdapter class has two methods, mouseDragged()and mouseMoved(). The signatures of these empty are exactly as defined in the MouseMotionListener interface. If you are interested in only mouse drag events, then you could simply extend MouseMotionAdapter and implement mouseDragged() .

**What is meant by controls and what are different types of controls in AWT? -** Controls are components that allow a user to interact with your application and the AWT supports the following types of controls: Labels, Push Buttons, Check Boxes, Choice Lists, Lists, Scrollbars, Text Components. These controls are subclasses of Component.

**What is the difference between choice and list? -** A Choice is displayed in a compact form that requires you to pull it down to see the list of available choices and only one item may be selected from a choice. A List may be displayed in such a way that several list items are visible and it supports the selection of one or more list items.

**What is the difference between scrollbar and scrollpane? -** A Scrollbar is a Component, but not a Container whereas Scrollpane is a Conatiner and handles its own events and perform its own scrolling.

**What is a layout manager and what are different types of layout managers available in java AWT? -** A layout manager is an object that is used to organize components in a container. The different layouts are available are FlowLayout, BorderLayout, CardLayout, GridLayout and GridBagLayout.

**How are the elements of different layouts organized? -** FlowLayout: The elements of a FlowLayout are organized in a top to bottom, left to right fashion. BorderLayout: The elements of a BorderLayout are organized at the borders (North, South, East and West) and the center of a container. CardLayout: The elements of a CardLayout are stacked, on top of the other, like a deck of cards. GridLayout: The elements of a GridLayout are of equal size and are laid out using the square of a grid. GridBagLayout: The elements of a GridBagLayout are organized according to a grid. However, the elements are of different size and may occupy more than one row or column of the grid. In addition, the rows and columns may have different sizes.

**Which containers use a Border layout as their default layout? -** Window, Frame and Dialog classes use a BorderLayout as their layout.

**Which containers use a Flow layout as their default layout? -** Panel and Applet classes use the FlowLayout as their default layout.

**What are wrapper classes?** - Wrapper classes are classes that allow primitive types to be accessed as objects.

**What are Vector, Hashtable, LinkedList and Enumeration?** - Vector : The Vector class provides the capability to implement a growable array of objects. Hashtable : The Hashtable class implements a Hashtable data structure. A Hashtable indexes and stores objects in a dictionary using hash codes as the object’s keys. Hash codes are integer values that identify objects. LinkedList: Removing or inserting elements in the middle of an array can be done using LinkedList. A LinkedList stores each object in a separate link whereas an array stores object references in consecutive locations. Enumeration: An object that implements the Enumeration interface generates a series of elements, one at a time. It has two methods, namely hasMoreElements() and nextElement(). HasMoreElemnts() tests if this enumeration has more elements and nextElement method returns successive elements of the series.

**What is the difference between set and list?** - Set stores elements in an unordered way but does not contain duplicate elements, whereas list stores elements in an ordered way but may contain duplicate elements.

**What is a stream and what are the types of Streams and classes of the Streams?** - A Stream is an abstraction that either produces or consumes information. There are two types of Streams and they are: Byte Streams: Provide a convenient means for handling input and output of bytes. Character Streams: Provide a convenient means for handling input & output of characters. Byte Streams classes: Are defined by using two abstract classes, namely InputStream and OutputStream. Character Streams classes: Are defined by using two abstract classes, namely Reader and Writer.

**What is the difference between Reader/Writer and InputStream/Output Stream?** - The Reader/Writer class is character-oriented and the InputStream/OutputStream class is byte-oriented.

**What is an I/O filter? -** 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.

**What is serialization and deserialization?** - Serialization is the process of writing the state of an object to a byte stream. Deserialization is the process of restoring these objects.

**What is JDBC?** - JDBC is a set of Java API for executing SQL statements. This API consists of a set of classes and interfaces to enable programs to write pure Java Database applications.

**What are drivers available?** - a) JDBC-ODBC Bridge driver b) Native API Partly-Java driver c) JDBC-Net Pure Java driver d) Native-Protocol Pure Java driver

**What is the difference between JDBC and ODBC? -** a) OBDC is for Microsoft and JDBC is for Java applications. b) ODBC can’t be directly used with Java because it uses a C interface. c) ODBC makes use of pointers which have been removed totally from Java. d) ODBC mixes simple and advanced features together and has complex options for simple queries. But JDBC is designed to keep things simple while allowing advanced capabilities when required. e) ODBC requires manual installation of the ODBC driver manager and driver on all client machines. JDBC drivers are written in Java and JDBC code is automatically installable, secure, and portable on all platforms. f) JDBC API is a natural Java interface and is built on ODBC. JDBC retains some of the basic features of ODBC.

**What are the types of JDBC Driver Models and explain them? -** There are two types of JDBC Driver Models and they are: a) Two tier model and b) Three tier model Two tier model: In this model, Java applications interact directly with the database. A JDBC driver is required to communicate with the particular database management system that is being accessed. SQL statements are sent to the database and the results are given to user. This model is referred to as client/server configuration where user is the client and the machine that has the database is called as the server. Three tier model: A middle tier is introduced in this model. The functions of this model are: a) Collection of SQL statements from the client and handing it over to the database, b) Receiving results from database to the client and c) Maintaining control over accessing and updating of the above.

**What are the steps involved for making a connection with a database or how do you connect to a database?**a) Loading the driver : To load the driver, Class. forName() method is used. Class. forName(”sun. jdbc. odbc. JdbcOdbcDriver”); When the driver is loaded, it registers itself with the java. sql. DriverManager class as an available database driver. b) Making a connection with database: To open a connection to a given database, DriverManager. getConnection() method is used. Connection con = DriverManager. getConnection (”jdbc:odbc:somedb”, “user”, “password”); c) Executing SQL statements : To execute a SQL query, java. sql. statements class is used. createStatement() method of Connection to obtain a new Statement object. Statement stmt = con. createStatement(); A query that returns data can be executed using the executeQuery() method of Statement. This method executes the statement and returns a java. sql. ResultSet that encapsulates the retrieved data: ResultSet rs = stmt. executeQuery(”SELECT \* FROM some table”); d) Process the results : ResultSet returns one row at a time. Next() method of ResultSet object can be called to move to the next row. The getString() and getObject() methods are used for retrieving column values: while(rs. next()) { String event = rs. getString(”event”); Object count = (Integer) rs. getObject(”count”);

**What type of driver did you use in project? -** JDBC-ODBC Bridge driver (is a driver that uses native(C language) libraries and makes calls to an existing ODBC driver to access a database engine).

**What are the types of statements in JDBC? -** Statement: to be used createStatement() method for executing single SQL statement PreparedStatement — To be used preparedStatement() method for executing same SQL statement over and over. CallableStatement — To be used prepareCall() method for multiple SQL statements over and over.

**What is stored procedure? -** 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.

**How to create and call stored procedures? -** To create stored procedures: Create procedure procedurename (specify in, out and in out parameters) BEGIN Any multiple SQL statement; END; To call stored procedures: CallableStatement csmt = con. prepareCall(”{call procedure name(?,?)}”); csmt. registerOutParameter(column no. , data type); csmt. setInt(column no. , column name) csmt. execute();

**What is servlet? -** Servlets are modules that extend request/response-oriented servers, such as java-enabled web servers. For example, a servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company’s order database.

**What are the classes and interfaces for servlets? -** There are two packages in servlets and they are javax. servlet and

**What is the difference between an applet and a servlet? -** a) Servlets are to servers what applets are to browsers. b) Applets must have graphical user interfaces whereas servlets have no graphical user interfaces.

**What is the difference between doPost and doGet methods? -** a) doGet() method is used to get information, while doPost() method is used for posting information. b) doGet() requests can’t send large amount of information and is limited to 240-255 characters. However, doPost()requests passes all of its data, of unlimited length. c) A doGet() request is appended to the request URL in a query string and this allows the exchange is visible to the client, whereas a doPost() request passes directly over the socket connection as part of its HTTP request body and the exchange are invisible to the client.

**What is the life cycle of a servlet? -** Each Servlet has the same life cycle: a) A server loads and initializes the servlet by init () method. b) The servlet handles zero or more client’s requests through service() method. c) The server removes the servlet through destroy() method.

**Who is loading the init() method of servlet? -** Web server

**What are the different servers available for developing and deploying Servlets? -** a) Java Web Server b) JRun g) Apache Server h) Netscape Information Server i) Web Logic

**How many ways can we track client and what are they? -** The servlet API provides two ways to track client state and they are: a) Using Session tracking and b) Using Cookies.

**What is session tracking and how do you track a user session in servlets? -** Session tracking is a mechanism that servlets use to maintain state about a series requests from the same user across some period of time. The methods used for session tracking are: a) User Authentication - occurs when a web server restricts access to some of its resources to only those clients that log in using a recognized username and password. b) Hidden form fields - fields are added to an HTML form that are not displayed in the client’s browser. When the form containing the fields is submitted, the fields are sent back to the server. c) URL rewriting - every URL that the user clicks on is dynamically modified or rewritten to include extra information. The extra information can be in the form of extra path information, added parameters or some custom, server-specific URL change. d) Cookies - a bit of information that is sent by a web server to a browser and which can later be read back from that browser. e) HttpSession- places a limit on the number of sessions that can exist in memory. This limit is set in the session. maxresidents property.

**What is Server-Side Includes (SSI)? -** Server-Side Includes allows embedding servlets within HTML pages using a special servlet tag. In many servlets that support servlets, a page can be processed by the server to include output from servlets at certain points inside the HTML page. This is accomplished using a special internal SSINCLUDE, which processes the servlet tags. SSINCLUDE servlet will be invoked whenever a file with an. shtml extension is requested. So HTML files that include server-side includes must be stored with an . shtml extension.

**What are cookies and how will you use them? -** Cookies are a mechanism that a servlet uses to have a client hold a small amount of state-information associated with the user. a) Create a cookie with the Cookie constructor: public Cookie(String name, String value) b) A servlet can send a cookie to the client by passing a Cookie object to the addCookie() method of HttpServletResponse: public void HttpServletResponse. addCookie(Cookie cookie) c) A servlet retrieves cookies by calling the getCookies() method of HttpServletRequest: public Cookie[ ] HttpServletRequest. getCookie().

**Is it possible to communicate from an applet to servlet and how many ways and how? -** Yes, there are three ways to communicate from an applet to servlet and they are: a) HTTP Communication(Text-based and object-based) b) Socket Communication c) RMI Communication

**What is connection pooling? -** With servlets, opening a database connection is a major bottleneck because we are creating and tearing down a new connection for every page request and the time taken to create connection will be more. Creating a connection pool is an ideal approach for a complicated servlet. With a connection pool, we can duplicate only the resources we need to duplicate rather than the entire servlet. A connection pool can also intelligently manage the size of the pool and make sure each connection remains valid. A number of connection pool packages are currently available. Some like DbConnectionBroker are freely available from Java Exchange Works by creating an object that dispenses connections and connection Ids on request. The ConnectionPool class maintains a Hastable, using Connection objects as keys and Boolean values as stored values. The Boolean value indicates whether a connection is in use or not. A program calls getConnection() method of the ConnectionPool for getting Connection object it can use; it calls returnConnection() to give the connection back to the pool.

**Why should we go for interservlet communication? -** Servlets running together in the same server communicate with each other in several ways. The three major reasons to use interservlet communication are: a) Direct servlet manipulation - allows to gain access to the other currently loaded servlets and perform certain tasks (through the ServletContext object) b) Servlet reuse - allows the servlet to reuse the public methods of another servlet. c) Servlet collaboration - requires to communicate with each other by sharing specific information (through method invocation)

**Is it possible to call servlet with parameters in the URL? -** Yes. You can call a servlet with parameters in the syntax as (?Param1 = xxx || m2 = yyy).

**What is Servlet chaining? -** Servlet chaining is a technique in which two or more servlets can cooperate in servicing a single request. In servlet chaining, one servlet’s output is piped to the next servlet’s input. This process continues until the last servlet is reached. Its output is then sent back to the client.

**How do servlets handle multiple simultaneous requests? -** The server has multiple threads that are available to handle requests. When a request comes in, it is assigned to a thread, which calls a service method (for example: doGet(), doPost() and service()) of the servlet. For this reason, a single servlet object can have its service methods called by many threads at once.

**What is the difference between TCP/IP and UDP? -** TCP/IP is a two-way communication between the client and the server and it is a reliable and there is a confirmation regarding reaching the message to the destination. It is like a phone call. UDP is a one-way communication only between the client and the server and it is not a reliable and there is no confirmation regarding reaching the message to the destination. It is like a postal mail.

**What is Inet address? -** Every computer connected to a network has an IP address. An IP address is a number that uniquely identifies each computer on the Net. An IP address is a 32-bit number.

**What is Domain Naming Service(DNS)? -** It is very difficult to remember a set of numbers(IP address) to connect to the Internet. The Domain Naming Service(DNS) is used to overcome this problem. It maps one particular IP address to a string of characters. For example, www. mascom. com implies com is the domain name reserved for US commercial sites, moscom is the name of the company and www is the name of the specific computer, which is mascom’s server.

**What is URL? -** URL stands for Uniform Resource Locator and it points to resource files on the Internet. URL has four components: http://www. address. com:80/index.html, where http - protocol name, address - IP address or host name, 80 - port number and index.html - file path.

**What is RMI and steps involved in developing an RMI object? -** Remote Method Invocation (RMI) allows java object that executes on one machine and to invoke the method of a Java object to execute on another machine. The steps involved in developing an RMI object are: a) Define the interfaces b) Implementing these interfaces c) Compile the interfaces and their implementations with the java compiler d) Compile the server implementation with RMI compiler e) Run the RMI registry f) Run the application

**What is RMI architecture? -** RMI architecture consists of four layers and each layer performs specific functions: a) Application layer - contains the actual object definition. b) Proxy layer - consists of stub and skeleton. c) Remote Reference layer - gets the stream of bytes from the transport layer and sends it to the proxy layer. d) Transportation layer - responsible for handling the actual machine-to-machine communication.

**what is UnicastRemoteObject? -** All remote objects must extend UnicastRemoteObject, which provides functionality that is needed to make objects available from remote machines.

**Explain the methods, rebind() and lookup() in Naming class? -** rebind() of the Naming class(found in java. rmi) is used to update the RMI registry on the server machine. Naming. rebind(”AddSever”, AddServerImpl); lookup() of the Naming class accepts one argument, the rmi URL and returns a reference to an object of type AddServerImpl.

**What is a Java Bean? -** A Java Bean is a software component that has been designed to be reusable in a variety of different environments.

**What is a Jar file? -** Jar file allows to efficiently deploying a set of classes and their associated resources. The elements in a jar file are compressed, which makes downloading a Jar file much faster than separately downloading several uncompressed files. The package java. util. zip contains classes that read and write jar files.

**What is BDK? -** BDK, Bean Development Kit is a tool that enables to create, configure and connect a set of set of Beans and it can be used to test Beans without writing a code.

**What is JSP? -** JSP is a dynamic scripting capability for web pages that allows Java as well as a few special tags to be embedded into a web file (HTML/XML, etc). The suffix traditionally ends with .jsp to indicate to the web server that the file is a JSP files. JSP is a server side technology - you can’t do any client side validation with it. The advantages are: a) The JSP assists in making the HTML more functional. Servlets on the other hand allow outputting of HTML but it is a tedious process. b) It is easy to make a change and then let the JSP capability of the web server you are using deal with compiling it into a servlet and running it.

**What are JSP scripting elements? -** JSP scripting elements lets to insert Java code into the servlet that will be generated from the current JSP page. There are three forms: a) Expressions of the form <%= expression %> that are evaluated and inserted into the output, b) Scriptlets of the form<% code %>that are inserted into the servlet’s service method, and c) Declarations of the form <%! Code %>that are inserted into the body of the servlet class, outside of any existing methods.

**What are JSP Directives? -** A JSP directive affects the overall structure of the servlet class. It usually has the following form:<%@ directive attribute=”value” %> However, you can also combine multiple attribute settings for a single directive, as follows:<%@ directive attribute1=”value1″ attribute 2=”value2″ . . . attributeN =”valueN” %> There are two main types of directive: page, which lets to do things like import classes, customize the servlet superclass, and the like; and include, which lets to insert a file into the servlet class at the time the JSP file is translated into a servlet

**What are Predefined variables or implicit objects? -** To simplify code in JSP expressions and scriptlets, we can use eight automatically defined variables, sometimes called implicit objects. They are request, response, out, session, application, config, pageContext, and page.

**What are JSP ACTIONS? -** JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin. Available actions include: jsp:include - Include a file at the time the page is requested. jsp:useBean - Find or instantiate a JavaBean. jsp:setProperty - Set the property of a JavaBean. jsp:getProperty - Insert the property of a JavaBean into the output. jsp:forward - Forward the requester to a newpage. Jsp: plugin - Generate browser-specific code that makes an OBJECT or EMBED

**How do you pass data (including JavaBeans) to a JSP from a servlet? -** (1) Request Lifetime: Using this technique to pass beans, a request dispatcher (using either “include” or forward”) can be called. This bean will disappear after processing this request has been completed. Servlet: request. setAttribute(”theBean”, myBean); RequestDispatcher rd = getServletContext(). getRequestDispatcher(”thepage. jsp”); rd. forward(request, response); JSP PAGE:<jsp: useBean id=”theBean” scope=”request” class=”. . . . . ” />(2) Session Lifetime: Using this technique to pass beans that are relevant to a particular session (such as in individual user login) over a number of requests. This bean will disappear when the session is invalidated or it times out, or when you remove it. Servlet: HttpSession session = request. getSession(true); session. putValue(”theBean”, myBean); /\* You can do a request dispatcher here, or just let the bean be visible on the next request \*/ JSP Page:<jsp:useBean id=”theBean” scope=”session” class=”. . . ” /> 3) Application Lifetime: Using this technique to pass beans that are relevant to all servlets and JSP pages in a particular app, for all users. For example, I use this to make a JDBC connection pool object available to the various servlets and JSP pages in my apps. This bean will disappear when the servlet engine is shut down, or when you remove it. Servlet: GetServletContext(). setAttribute(”theBean”, myBean); JSP PAGE:<jsp:useBean id=”theBean” scope=”application” class=”. . . ” />

**How can I set a cookie in JSP? -** response. setHeader(”Set-Cookie”, “cookie string”); To give the response-object to a bean, write a method setResponse (HttpServletResponse response) - to the bean, and in jsp-file:<% bean. setResponse (response); %>

**How can I delete a cookie with JSP? -** Say that I have a cookie called “foo, ” that I set a while ago & I want it to go away. I simply: <% Cookie killCookie = new Cookie(”foo”, null); KillCookie. setPath(”/”); killCookie. setMaxAge(0); response. addCookie(killCookie); %>

**How are Servlets and JSP Pages related? -** JSP pages are focused around HTML (or XML) with Java codes and JSP tags inside them. When a web server that has JSP support is asked for a JSP page, it checks to see if it has already compiled the page into a servlet. Thus, JSP pages become servlets and are transformed into pure Java and then compiled, loaded into the server and executed.

**What is garbage collection? What is the process that is responsible for doing that in java?** - Reclaiming the unused memory by the invalid objects. Garbage collector is responsible for this process

**What kind of thread is the Garbage collector thread?** - It is a daemon thread.

**What is a daemon thread?** - These are the threads which can run without user intervention. The JVM can exit when there are daemon thread by killing them abruptly.

**How will you invoke any external process in Java?** - Runtime.getRuntime().exec(….)

**What is the finalize method do?** - Before the invalid objects get garbage collected, the JVM give the user a chance to clean up some resources before it got garbage collected.

**What is mutable object and immutable object?** - If a object value is changeable then we can call it as Mutable object. (Ex., StringBuffer, …) If you are not allowed to change the value of an object, it is immutable object. (Ex., String, Integer, Float, …)

**What is the basic difference between string and stringbuffer object?** - String is an immutable object. StringBuffer is a mutable object.

**What is the purpose of Void class?** - The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the primitive Java type void.

**What is reflection?** - Reflection allows programmatic access to information about the fields, methods and constructors of loaded classes, and the use reflected fields, methods, and constructors to operate on their underlying counterparts on objects, within security restrictions.

**What is the base class for Error and Exception?** - Throwable

**What is the byte range?** -128 to 127

**What is the implementation of destroy method in java.. is it native or java code?** - This method is not implemented.

**What is a package?** - To group set of classes into a single unit is known as packaging. Packages provides wide namespace ability.

**What are the approaches that you will follow for making a program very efficient?** - By avoiding too much of static methods avoiding the excessive and unnecessary use of synchronized methods Selection of related classes based on the application (meaning synchronized classes for multiuser and non-synchronized classes for single user) Usage of appropriate design patterns Using cache methodologies for remote invocations Avoiding creation of variables within a loop and lot more.

**What is a DatabaseMetaData?** - Comprehensive information about the database as a whole.

**What is Locale?** - A Locale object represents a specific geographical, political, or cultural region

**How will you load a specific locale?** - Using ResourceBundle.getBundle(…);

**What is JIT and its use?** - Really, just a very fast compiler… In this incarnation, pretty much a one-pass compiler — no offline computations. So you can’t look at the whole method, rank the expressions according to which ones are re-used the most, and then generate code. In theory terms, it’s an on-line problem.

**Is JVM a compiler or an interpreter?** - Interpreter

**When you think about optimization, what is the best way to findout the time/memory consuming process?** - Using profiler

**What is the purpose of assert keyword used in JDK1.4.x?** - In order to validate certain expressions. It effectively replaces the if block and automatically throws the AssertionError on failure. This keyword should be used for the critical arguments. Meaning, without that the method does nothing.

**How will you get the platform dependent values like line separator, path separator, etc., ?** - Using Sytem.getProperty(…) (line.separator, path.separator, …)

**What is skeleton and stub? what is the purpose of those?** - Stub is a client side representation of the server, which takes care of communicating with the remote server. Skeleton is the server side representation. But that is no more in use… it is deprecated long before in JDK.

**What is the final keyword denotes?** - final keyword denotes that it is the final implementation for that method or variable or class. You can’t override that method/variable/class any more.

**What is the significance of ListIterator?** - You can iterate back and forth.

**What is the major difference between LinkedList and ArrayList?** - LinkedList are meant for sequential accessing. ArrayList are meant for random accessing.

**What is nested class?** - If all the methods of a inner class is static then it is a nested class.

**What is inner class?** - If the methods of the inner class can only be accessed via the instance of the inner class, then it is called inner class.

**What is composition?** - Holding the reference of the other class within some other class is known as composition.

**What is aggregation?** - It is a special type of composition. If you expose all the methods of a composite class and route the method call to the composite method through its reference, then it is called aggregation.

**What are the methods in Object?** - clone, equals, wait, finalize, getClass, hashCode, notify, notifyAll, toString

**Can you instantiate the Math class?** - You can’t instantiate the math class. All the methods in this class are static. And the constructor is not public.

**What is singleton?** - It is one of the design pattern. This falls in the creational pattern of the design pattern. There will be only one instance for that entire JVM. You can achieve this by having the private constructor in the class. For eg., public class Singleton { private static final Singleton s = new Singleton(); private Singleton() { } public static Singleton getInstance() { return s; } // all non static methods … }

**Describe what happens when an object is created in Java?** - Several things happen in a particular order to ensure the object is constructed properly:   
1. Memory is allocated from heap to hold all instance variables and implementation-specific data of the object and its superclasses. Implementation-specific data includes pointers to class and method data.   
2. The instance variables of the objects are initialized to their default values.   
3. The constructor for the most derived class is invoked. The first thing a constructor does is call the constructor for its uppercase. This process continues until the constructor for java.lang.Object is called, as java.lang.Object is the base class for all objects in java.   
4. Before the body of the constructor is executed, all instance variable initializers and initialization blocks are executed. Then the body of the constructor is executed. Thus, the constructor for the base class completes first and constructor for the most derived class completes last.

**In Java, you can create a String object as below : String str = "abc"; & String str = new String("abc");  Why cant a button object be created as : Button bt = "abc"? Why is it compulsory to create a button object as: Button bt = new Button("abc"); Why this is not compulsory in String’s case?**

Button bt1= "abc"; It is because "abc" is a literal string (something slightly different than a String object, by-the-way) and bt1 is a Button object. That simple. The only object in Java that can be assigned a literal String is java.lang.String. Important to not that you are NOT calling a java.lang.String constuctor when you type String s = "abc"; For example String x = "abc"; String y = "abc"; refer to the same object. While String x1 = new String("abc");   
String x2 = new String("abc"); refer to two different objects.

**What is the advantage of OOP?** - You will get varying answers to this question depending on whom you ask. Major advantages of OOP are:   
1. Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear;   
2. Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system;   
3. Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods;   
4. Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones;   
5. Maintainability: objects can be maintained separately, making locating and fixing problems easier;   
6. Re-usability: objects can be reused in different programs

**What are the main differences between Java and C++?** - Everything is an object in Java( Single root hierarchy as everything gets derived from java.lang.Object). Java does not have all the complicated aspects of C++ ( For ex: Pointers, templates, unions, operator overloading, structures etc..)  The Java language promoters initially said "No pointers!", but when many programmers questioned how you can work without pointers, the promoters began saying "Restricted pointers." You can make up your mind whether it’s really a pointer or not. In any event, there’s no pointer arithmetic. There are no destructors in Java. (automatic garbage collection),  Java does not support conditional compile (#ifdef/#ifndef type). Thread support is built into java but not in C++. Java does not support default arguments. There’s no scope resolution operator :: in Java. Java uses the dot for everything, but can get away with it since you can define elements only within a class. Even the method definitions must always occur within a class, so there is no need for scope resolution there either. There’s no "goto " statement in Java. Java doesn’t provide multiple inheritance (MI), at least not in the same sense that C++ does. Exception handling in Java is different because there are no destructors. Java has method overloading, but no operator overloading. The String class does use the + and += operators to concatenate strings and String expressions use automatic type conversion, but that’s a special built-in case. Java is interpreted for the most part and hence platform independent

**What are interfaces?** - Interfaces provide more sophisticated ways to organize and control the objects in your system.   
The interface keyword takes the abstract concept one step further. You could think of it as a “pure” abstract class. It allows the creator to establish the form for a class: method names, argument lists, and return types, but no method bodies. An interface can also contain fields, but The interface keyword takes the abstract concept one step further. You could think of it as a “pure” abstract class. It allows the creator to establish the form for a class: method names, argument lists, and return types, but no method bodies. An interface can also contain fields, but an interface says: “This is what all classes that implement this particular interface will look like.” Thus, any code that uses a particular interface knows what methods might be called for that interface, and that’s all. So the interface is used to establish a “protocol” between classes. (Some object-oriented programming languages have a keyword called protocol to do the same thing.)  Typical example from "Thinking in Java":

import java.util.\*;   
interface Instrument {   
int i = 5; // static & final   
// Cannot have method definitions:   
void play(); // Automatically public   
String what();   
void adjust();   
}   
class Wind implements Instrument {   
public void play() {   
System.out.println("Wind.play()");   
public String what() { return "Wind"; }   
public void adjust() {}   
}

**How can you achieve Multiple Inheritance in Java?** - Java’s interface mechanism can be used to implement multiple inheritance, with one important difference from c++ way of doing MI: the inherited interfaces must be abstract. This obviates the need to choose between different implementations, as with interfaces there are no implementations.

interface CanFight {   
void fight();   
interface CanSwim {   
void swim();   
interface CanFly {   
void fly();   
class ActionCharacter {   
public void fight() {}   
class Hero extends ActionCharacter implements CanFight, CanSwim, CanFly {   
public void swim() {}   
public void fly() {}   
}

You can even achieve a form of multiple inheritance where you can use the \*functionality\* of classes rather than just the interface:

interface A {   
void methodA();   
}   
class AImpl implements A {   
void methodA() { //do stuff }   
}   
interface B {   
void methodB();   
}   
class BImpl implements B {   
void methodB() { //do stuff }   
}   
class Multiple implements A, B {   
private A a = new A();   
private B b = new B();   
void methodA() { a.methodA(); }   
void methodB() { b.methodB(); }   
}   
This completely solves the traditional problems of multiple inheritance in C++ where name clashes occur between multiple base classes. The coder of the derived class will have to explicitly resolve any clashes. Don’t you hate people who point out minor typos? Everything in the previous example is correct, except you need to instantiate an AImpl and BImpl. So class Multiple would look like this:   
class Multiple implements A, B {   
private A a = new AImpl();   
private B b = new BImpl();   
void methodA() { a.methodA(); }   
void methodB() { b.methodB(); }   
}

**What is the difference between StringBuffer and String class?** - A string buffer implements a mutable sequence of characters. A string buffer is like a String, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls. The String class represents character strings. All string literals in Java programs, such as "abc" are constant and implemented as instances of this class; their values cannot be changed after they are created. Strings in Java are known to be immutable. What it means is that every time you need to make a change to a String variable, behind the scene, a "new" String is actually being created by the JVM. For an example: if you change your String variable 2 times, then you end up with 3 Strings: one current and 2 that are ready for garbage collection. The garbage collection cycle is quite unpredictable and these additional unwanted Strings will take up memory until that cycle occurs. For better performance, use StringBuffers for string-type data that will be reused or changed frequently. There is more overhead per class than using String, but you will end up with less overall classes and consequently consume less memory. Describe, in general, how java’s garbage collector works? The Java runtime environment deletes objects when it determines that they are no longer being used. This process is known as garbage collection. The Java runtime environment supports a garbage collector that periodically frees the memory used by objects that are no longer needed. The Java garbage collector is a mark-sweep garbage collector that scans Java’s dynamic memory areas for objects, marking those that are referenced. After all possible paths to objects are investigated, those objects that are not marked (i.e. are not referenced) are known to be garbage and are collected. (A more complete description of our garbage collection algorithm might be "A compacting, mark-sweep collector with some conservative scanning".) The garbage collector runs synchronously when the system runs out of memory, or in response to a request from a Java program. Your Java program can ask the garbage collector to run at any time by calling System.gc(). The garbage collector requires about 20 milliseconds to complete its task so, your program should only run the garbage collector when there will be no performance impact and the program anticipates an idle period long enough for the garbage collector to finish its job. Note: Asking the garbage collection to run does not guarantee that your objects will be garbage collected. The Java garbage collector runs asynchronously when the system is idle on systems that allow the Java runtime to note when a thread has begun and to interrupt another thread (such as Windows 95). As soon as another thread becomes active, the garbage collector is asked to get to a consistent state and then terminate.

**What’s the difference between == and equals method?** - equals checks for the content of the string objects while == checks for the fact that the two String objects point to same memory location ie they are same references.

**What are abstract classes, abstract methods?** - Simply speaking a class or a method qualified with "abstract" keyword is an abstract class or abstract method. You create an abstract class when you want to manipulate a set of classes through a common interface. All derived-class methods that match the signature of the base-class declaration will be called using the dynamic binding mechanism. If you have an abstract class, objects of that class almost always have no meaning. That is, abstract class is meant to express only the interface and sometimes some default method implementations, and not a particular implementation, so creating an abstract class object makes no sense and are not allowed ( compile will give you an error message if you try to create one). An abstract method is an incomplete method. It has only a declaration and no method body. Here is the syntax for an abstract method declaration: abstract void f(); If a class contains one or more abstract methods, the class must be qualified an abstract. (Otherwise, the compiler gives you an error message.). It’s possible to create a class as abstract without including any abstract methods. This is useful when you’ve got a class in which it doesn’t make sense to have any abstract methods, and yet you want to prevent any instances of that class. Abstract classes and methods are created because they make the abstractness of a class explicit, and tell both the user and the compiler how it was intended to be used.   
For example:   
abstract class Instrument {   
int i; // storage allocated for each   
public abstract void play();   
public String what() {   
return "Instrument";   
public abstract void adjust();   
}   
class Wind extends Instrument {   
public void play() {   
System.out.println("Wind.play()");   
}   
public String what() { return "Wind"; }   
public void adjust() {}   
Abstract classes are classes for which there can be no instances at run time. i.e. the implementation of the abstract classes are not complete. Abstract methods are methods which have no defintion. i.e. abstract methods have to be implemented in one of the sub classes or else that class will also become Abstract.

**What is the difference between an Applet and an Application?** - A Java application is made up of a main() method declared as public static void that accepts a string array argument, along with any other classes that main() calls. It lives in the environment that the host OS provides. A Java applet is made up of at least one public class that has to be subclassed from java.awt.Applet. The applet is confined to living in the user’s Web browser, and the browser’s security rules, (or Sun’s appletviewer, which has fewer restrictions).  The differences between an applet and an application are as follows:   
1. Applets can be embedded in HTML pages and downloaded over the Internet whereas Applications have no special support in HTML for embedding or downloading.   
2. Applets can only be executed inside a java compatible container, such as a browser or appletviewer whereas Applications are executed at command line by java.exe or jview.exe.   
3. Applets execute under strict security limitations that disallow certain operations(sandbox model security) whereas Applications have no inherent security restrictions.   
4. Applets don’t have the main() method as in applications. Instead they operate on an entirely different mechanism where they are initialized by init(),started by start(),stopped by stop() or destroyed by destroy().

**Java says "write once, run anywhere". What are some ways this isn’t quite true?** - As long as all implementations of java are certified by sun as 100% pure java this promise of "Write once, Run everywhere" will hold true. But as soon as various java core implementations start digressing from each other, this won’t be true anymore. A recent example of a questionable business tactic is the surreptitious behavior and interface modification of some of Java’s core classes in their own implementation of Java. Programmers who do not recognize these undocumented changes can build their applications expecting them to run anywhere that Java can be found, only to discover that their code works only on Microsoft’s own Virtual Machine, which is only available on Microsoft’s own operating systems.

**What is the difference between a Vector and an Array. Discuss the advantages and disadvantages of both?**

Vector can contain objects of different types whereas array can contain objects only of a single type.   
- Vector can expand at run-time, while array length is fixed.   
- Vector methods are synchronized while Array methods are not

**What are java beans?** - JavaBeans is a portable, platform-independent component model written in the Java programming language, developed in collaboration with industry leaders. It enables developers to write reusable components once and run them anywhere — benefiting from the platform-independent power of Java technology. JavaBeans acts as a Bridge between proprietary component models and provides a seamless and powerful means for developers to build components that run in ActiveX container applications. JavaBeans are usual Java classes which adhere to certain coding conventions:   
1. Implements java.io.Serializable interface   
2. Provides no argument constructor   
3. Provides getter and setter methods for accessing it’s properties

**What is RMI?** - RMI stands for Remote Method Invocation. Traditional approaches to executing code on other machines across a network have been confusing as well as tedious and error-prone to implement. The nicest way to think about this problem is that some object happens to live on another machine, and that you can send a message to the remote object and get a result as if the object lived on your local machine. This simplification is exactly what Java Remote Method Invocation (RMI) allows you to do. Above excerpt is from "Thinking in java". For more information refer to any book on Java.

**What does the keyword "synchronize" mean in java. When do you use it? What are the disadvantages of synchronization?** - Synchronize is used when you want to make your methods thread safe. The disadvantage of synchronize is it will end up in slowing down the program. Also if not handled properly it will end up in dead lock.

**What gives java it’s "write once and run anywhere" nature?** - Java is compiled to be a byte code which is the intermediate language between source code and machine code. This byte code is not platorm specific and hence can be fed to any platform. After being fed to the JVM, which is specific to a particular operating system, the code platform specific machine code is generated thus making java platform independent.

**What are native methods? How do you use them?** - Native methods are methods written in other languages like C, C++, or even assembly language. You can call native methods from Java using JNI. Native methods are used when the implementation of a particular method is present in language other than Java say C, C++. To use the native methods in java we use the keyword native   
public native method\_a(). This native keyword is signal to the java compiler that the implementation of this method is in a language other than java. Native methods are used when we realize that it would take up a lot of rework to write that piece of already existing code in other language to Java.

**What does the "final" keyword mean in front of a variable? A method? A class?**

FINAL for a variable: value is constant   
FINAL for a method: cannot be overridden   
FINAL for a class: cannot be derived   
A final variable cannot be reassigned,   
but it is not constant. For instance,   
final StringBuffer x = new StringBuffer()   
x.append("hello");   
is valid. X cannot have a new value in it,but nothing stops operations on the object   
that it refers, including destructive operations. Also, a final method cannot be overridden   
or hidden by new access specifications.This means that the compiler can choose   
to in-line the invocation of such a method.(I don’t know if any compiler actually does   
this, but it’s true in theory.) The best example of a final class is   
String, which defines a class that cannot be derived.

**What synchronization constructs does Java provide? How do they work?** - The two common features that are used are:   
1. Synchronized keyword - Used to synchronize a method or a block of code. When you synchronize a method, you are in effect synchronizing the code within the method using the monitor of the current object for the lock.   
The following have the same effect.   
synchronized void foo() {   
}   
and   
void foo() {   
synchronized(this) {   
}   
If you synchronize a static method, then you are synchronizing across all objects of the same class, i.e. the monitor you are using for the lock is one per class, not one per object.   
2. wait/notify. wait() needs to be called from within a synchronized block. It will first release the lock acquired from the synchronization and then wait for a signal. In Posix C, this part is equivalent to the pthread\_cond\_wait method, which waits for an OS signal to continue. When somebody calls notify() on the object, this will signal the code which has been waiting, and the code will continue from that point. If there are several sections of code that are in the wait state, you can call notifyAll() which will notify all threads that are waiting on the monitor for the current object. Remember that both wait() and notify() have to be called from blocks of code that are synchronized on the monitor for the current object.

**Does Java have multiple inheritance?** - Java does not support multiple inheritence directly but it does thru the concept of interfaces.   
We can make a class implement a number of interfaces if we want to achieve multiple inheritence type of functionality of C++.

**How does exception handling work in Java?**

1. It separates the working/functional code from the error-handling code by way of try-catch clauses.   
2. It allows a clean path for error propagation. If the called method encounters a situation it can’t manage, it can throw an exception and let the calling method deal with it.   
3. By enlisting the compiler to ensure that "exceptional" situations are anticipated and accounted for, it enforces powerful coding.   
4. Exceptions are of two types: Compiler-enforced exceptions or checked exceptions. Runtime exceptions or unchecked exceptions. Compiler-enforced (checked) exceptions are instances of the Exception class or one of its subclasses — excluding the RuntimeException branch. The compiler expects all checked exceptions to be appropriately handled. Checked exceptions must be declared in the throws clause of the method throwing them — assuming, of course, they’re not being caught within that same method. The calling method must take care of these exceptions by either catching or declaring them in its throws clause. Thus, making an exception checked forces the us to pay heed to the possibility of it being thrown. An example of a checked exception is java.io.IOException. As the name suggests, it throws whenever an input/output operation is abnormally terminated.

**Does Java have destructors?** - Garbage collector does the job working in the background   
Java does not have destructors; but it has finalizers that does a similar job.   
the syntax is   
public void finalize(){   
}   
if an object has a finalizer, the method is invoked before the system garbage collects the object

**What does the "abstract" keyword mean in front of a method? A class?** - Abstract keyword declares either a method or a class.   
If a method has a abstract keyword in front of it,it is called abstract method.Abstract method hs no body.It has only arguments and return type.Abstract methods act as placeholder methods that are implemented in the subclasses.   
Abstract classes can’t be instantiated.If a class is declared as abstract,no objects of that class can be created.If a class contains any abstract method it must be declared as abstract

**Are Java constructors inherited ? If not, why not?** - You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of it’s superclasses. One of the main reasons is because you probably don’t want to overide the superclasses constructor, which would be possible if they were inherited. By giving the developer the ability to override a superclasses constructor you would erode the encapsulation abilities of the language.

**Do I need to use synchronized on setValue(int)?** - It depends whether the method affects method local variables, class static or instance variables. If only method local variables are changed, the value is said to be confined by the method and is not prone to threading issues.

**What is the SwingUtilities.invokeLater(Runnable) method for?** - The static utility method invokeLater(Runnable) is intended to execute a new runnable thread from a Swing application without disturbing the normal sequence of event dispatching from the Graphical User Interface (GUI). The method places the runnable object in the queue of Abstract Windowing Toolkit (AWT) events that are due to be processed and returns immediately. The runnable objectÃ¢â‚¬â„¢s run() method is only called when it reaches the front of the queue. The deferred effect of the invokeLater(Runnable) method ensures that any necessary updates to the user interface can occur immediately, and the runnable work will begin as soon as those high priority events are dealt with. The invoke later method might be used to start work in response to a button click that also requires a significant change to the user interface, perhaps to restrict other activities, while the runnable thread executes.

**What is the volatile modifier for?** - The volatile modifier is used to identify variables whose values should not be optimized by the Java Virtual Machine, by caching the value for example. The volatile modifier is typically used for variables that may be accessed or modified by numerous independent threads and signifies that the value may change without synchronization.

**Which class is the wait() method defined in?** - The wait() method is defined in the Object class, which is the ultimate superclass of all others. So the Thread class and any Runnable implementation inherit this method from Object. The wait() method is normally called on an object in a multi-threaded program to allow other threads to run. The method should should only be called by a thread that has ownership of the objectÃ¢â‚¬â„¢s monitor, which usually means it is in a synchronized method or statement block.

**Which class is the wait() method defined in? I get incompatible return type for my threadÃ¢â‚¬â„¢s getState( ) method!** - It sounds like your application was built for a Java software development kit before Java 1.5. The Java API Thread class method getState() was introduced in version 1.5. Your thread method has the same name but different return type. The compiler assumes your application code is attempting to override the API method with a different return type, which is not allowed, hence the compilation error.

**What is a working thread?** - A working thread, more commonly known as a worker thread is the key part of a design pattern that allocates one thread to execute one task. When the task is complete, the thread may return to a thread pool for later use. In this scheme a thread may execute arbitrary tasks, which are passed in the form of a Runnable method argument, typically execute(Runnable). The runnable tasks are usually stored in a queue until a thread host is available to run them. The worker thread design pattern is usually used to handle many concurrent tasks where it is not important which finishes first and no single task needs to be coordinated with another. The task queue controls how many threads run concurrently to improve the overall performance of the system. However, a worker thread framework requires relatively complex programming to set up, so should not be used where simpler threading techniques can achieve similar results.

**What is a green thread?** - A green thread refers to a mode of operation for the Java Virtual Machine (JVM) in which all code is executed in a single operating system thread. If the Java program has any concurrent threads, the JVM manages multi-threading internally rather than using other operating system threads. There is a significant processing overhead for the JVM to keep track of thread states and swap between them, so green thread mode has been deprecated and removed from more recent Java implementations. Current JVM implementations make more efficient use of native operating system threads.

**What are native operating system threads?** - Native operating system threads are those provided by the computer operating system that plays host to a Java application, be it Windows, Mac or GNU/Linux. Operating system threads enable computers to run many programs simultaneously on the same central processing unit (CPU) without clashing over the use of system resources or spending lots of time running one program at the expense of another. Operating system thread management is usually optimised to specific microprocessor architecture and features so that it operates much faster than Java green thread processing.

**How could Java classes direct program messages to the system console, but error messages, say to a file?** - The class System has a variable out that represents the standard output, and the variable err that represents the standard error device. By default, they both point at the system console. This how the standard output could be re-directed:

Stream st = new Stream (new FileOutputStream ("techinterviews\_com.txt"));

System.setErr(st);

System.setOut(st);

**What’s the difference between an interface and an abstract class?** - An abstract class may contain code in method bodies, which is not allowed in an interface. With abstract classes, you have to inherit your class from it and Java does not allow multiple inheritance. On the other hand, you can implement multiple interfaces in your class.

**Why would you use a synchronized block vs. synchronized method?** - Synchronized blocks place locks for shorter periods than synchronized methods.

**Explain the usage of the keyword transient?** - This keyword indicates that the value of this member variable does not have to be serialized with the object. When the class will be de-serialized, this variable will be initialized with a default value of its data type (i.e. zero for integers).

**How can you force garbage collection?** - You can’t force GC, but could request it by calling System.gc(). JVM does not guarantee that GC will be started immediately.

**How do you know if an explicit object casting is needed?** - If you assign a superclass object to a variable of a subclass’s data type, you need to do explicit casting. For example:

Object a;Customer b; b = (Customer) a;

When you assign a subclass to a variable having a supeclass type, the casting is performed automatically.

**What’s the difference between the methods sleep() and wait()?** - The code sleep(1000); puts thread aside for exactly one second. The code wait(1000), causes a wait of up to one second. A thread could stop waiting earlier if it receives the notify() or notifyAll() call. The method wait() is defined in the class Object and the method sleep() is defined in the class Thread.

**Can you write a Java class that could be used both as an applet as well as an application?** - Yes. Add a main() method to the applet.

**What’s the difference between constructors and other methods?** - Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times.

**Can you call one constructor from another if a class has multiple constructors?** - Yes. Use this() syntax.

**Explain the usage of Java packages.** - This is a way to organize files when a project consists of multiple modules. It also helps resolve naming conflicts when different packages have classes with the same names. Packages access level also allows you to protect data from being used by the non-authorized classes.

**If a class is located in a package, what do you need to change in the OS environment to be able to use it?** - You need to add a directory or a jar file that contains the package directories to the CLASSPATH environment variable. LetÃ¢â‚¬â„¢s say a class Employee belongs to a package com.xyz.hr; and is located in the file c:/dev/com.xyz.hr.Employee.java. In this case, you’d need to add c:/dev to the variable CLASSPATH. If this class contains the method main(), you could test it from a command prompt window as follows:  
c:\>java com.xyz.hr.Employee

**What’s the difference between J2SDK 1.5 and J2SDK 5.0?** - There’s no difference, Sun Microsystems just re-branded this version.

**What would you use to compare two String variables - the operator == or the method equals()?** - I’d use the method equals() to compare the values of the Strings and the = = to check if two variables point at the same instance of a String object.

**Does it matter in what order catch statements for FileNotFoundException and IOExceptipon are written?** - Yes, it does. The FileNoFoundException is inherited from the IOException. Exception’s subclasses have to be caught first.

**Can an inner class declared inside of a method, access local variables of this method?** - It’s possible if these variables are final.

**What can go wrong if you replace && with & in the following code:**

String a=null;

if (a!=null && a.length()>10){...}

A single ampersand here would lead to a NullPointerException.

**What’s the main difference between a Vector and an ArrayList?** - Java Vector class is internally synchronized and ArrayList is not.

**When should the method invokeLater()be used?** - This method is used to ensure that Swing components are updated through the event-dispatching thread.

**How can a subclass call a method or a constructor defined in a superclass?** - Use the following syntax: super.myMethod(); To call a constructor of the superclass, just write super(); in the first line of the subclass’s constructor.

**What’s the difference between a queue and a stack?** - Stacks works by last-in-first-out rule (LIFO), while queues use the FIFO rule.

**You can create an abstract class that contains only abstract methods. On the other hand, you can create an interface that declares the same methods. So can you use abstract classes instead of interfaces?** - Sometimes. But your class may be a descendent of another class and in this case the interface is your only option.

**What comes to mind when you hear about a young generation in Java?** - Garbage collection.

**What comes to mind when someone mentions a shallow copy in Java?** - Object cloning.

**If you’re overriding the method equals() of an object, which other method you might also consider?** - hashCode()

**You are planning to do an indexed search in a list of objects. Which of the two Java collections should you use: ArrayList or LinkedList?** - ArrayList

**How would you make a copy of an entire Java object with its state?** - Have this class implement Cloneable interface and call its method clone().

**How can you minimize the need of garbage collection and make the memory use more effective?** - Use object pooling and weak object references.

**There are two classes: A and B. The class B need to inform a class A when some important event has happened. What Java technique would you use to implement it?** - If these classes are threads I’d consider notify() or notifyAll(). For regular classes you can use the Observer interface.

**What access level do you need to specify in the class declaration to ensure that only classes from the same directory can access it?** - You do not need to specify any access level, and Java will use a default package access level.

**What makes J2EE suitable for distributed multitiered Applications?** - The J2EE platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. The J2EE application parts are:

* + Client-tier components run on the client machine.
  + Web-tier components run on the J2EE server.
  + Business-tier components run on the J2EE server.
  + Enterprise information system (EIS)-tier software runs on the EIS server.

**What is J2EE? -** J2EE is an environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, web-based applications.

**What are the components of J2EE application? -** A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and communicates with other components. The J2EE specification defines the following J2EE components:

* Application clients and applets are client components.
* Java Servlet and JavaServer Pages technology components are web components.
* Enterprise JavaBeans components (enterprise beans) are business components.
* Resource adapter components provided by EIS and tool vendors.

**What do Enterprise JavaBeans components contain?** - Enterprise JavaBeans components contains Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. All the business code is contained inside an Enterprise Bean which receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.

**Is J2EE application only a web-based?** - No, It depends on type of application that client wants. A J2EE application can be web-based or non-web-based. if an application client executes on the client machine, it is a non-web-based J2EE application. The J2EE application can provide a way for users to handle tasks such as J2EE system or application administration. It typically has a graphical user interface created from Swing or AWT APIs, or a command-line interface. When user request, it can open an HTTP connection to establish communication with a servlet running in the web tier.

**Are JavaBeans J2EE components?** - No. JavaBeans components are not considered J2EE components by the J2EE specification. They are written to manage the data flow between an application client or applet and components running on the J2EE server or between server components and a database. JavaBeans components written for the J2EE platform have instance variables and get and set methods for accessing the data in the instance variables. JavaBeans components used in this way are typically simple in design and implementation, but should conform to the naming and design conventions outlined in the JavaBeans component architecture.

**Is HTML page a web component?** - No. Static HTML pages and applets are bundled with web components during application assembly, but are not considered web components by the J2EE specification. Even the server-side utility classes are not considered web components, either.

**What can be considered as a web component?** - J2EE Web components can be either servlets or JSP pages. Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that execute as servlets but allow a more natural approach to creating static content.

**What is the container?** - Containers are the interface between a component and the low-level platform specific functionality that supports the component. Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container.

**What are container services?** - A container is a runtime support of a system-level entity. Containers provide components with services such as lifecycle management, security, deployment, and threading.

**What is the web container?** - Servlet and JSP containers are collectively referred to as Web containers. It manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

**What is Enterprise JavaBeans (EJB) container?** - It manages the execution of enterprise beans for J2EE applications.  
Enterprise beans and their container run on the J2EE server.

**What is Applet container?** - IManages the execution of applets. Consists of a Web browser and Java Plugin running on the client together.

**How do we package J2EE components?** - J2EE components are packaged separately and bundled into a J2EE application for deployment. Each component, its related files such as GIF and HTML files or server-side utility classes, and a deployment descriptor are assembled into a module and added to the J2EE application. A J2EE application is composed of one or more enterprise bean,Web, or application client component modules. The final enterprise solution can use one J2EE application or be made up of two or more J2EE applications, depending on design requirements. A J2EE application and each of its modules has its own deployment descriptor. A deployment descriptor is an XML document with an .xml extension that describes a component’s deployment settings.

**What is a thin client?** - A thin client is a lightweight interface to the application that does not have such operations like query databases, execute complex business rules, or connect to legacy applications.

**What are types of J2EE clients?** - Following are the types of J2EE clients:

* + Applets
  + Application clients
  + Java Web Start-enabled rich clients, powered by Java Web Start technology.
  + Wireless clients, based on Mobile Information Device Profile (MIDP) technology.

**What is deployment descriptor?** - A deployment descriptor is an Extensible Markup Language (XML) text-based file with an .xml extension that describes a component’s deployment settings. A J2EE application and each of its modules has its own deployment descriptor. For example, an enterprise bean module deployment descriptor declares transaction attributes and security authorizations  
for an enterprise bean. Because deployment descriptor information is declarative, it can be changed without modifying the bean source code. At run time, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

**What is the EAR file?** - An EAR file is a standard JAR file with an .ear extension, named from Enterprise ARchive file. A J2EE application with all of its modules is delivered in EAR file.

**What is JTA and JTS?** - JTA is the abbreviation for the Java Transaction API. JTS is the abbreviation for the Jave Transaction Service. JTA provides a standard interface and allows you to demarcate transactions in a manner that is independent of the transaction manager implementation. The J2EE SDK implements the transaction manager with JTS. But your code doesn’t call the JTS methods directly. Instead, it invokes the JTA methods, which then call the lower-level JTS routines. Therefore, JTA is a high level transaction interface that your application uses to control transaction. and JTS is a low level transaction interface and ejb uses behind the scenes (client code doesn’t directly interact with JTS. It is based on object transaction service(OTS) which is part of CORBA.

**What is JAXP?** - JAXP stands for Java API for XML. XML is a language for representing and describing text-based data which can be read and handled by any program or tool that uses XML APIs. It provides standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and create the appropriate JavaBeans component to perform those operations.

**What is J2EE Connector?** - The J2EE Connector API is used by J2EE tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged into any J2EE product. Each type of database or EIS has a different resource adapter. Note: A resource adapter is a software component that allows J2EE application components to access and interact with the underlying resource manager. Because a resource adapter is specific to its resource manager, there is typically a different resource adapter for each type of database or enterprise information system.

**What is JAAP?** - The Java Authentication and Authorization Service (JAAS) provides a way for a J2EE application to authenticate and authorize a specific user or group of users to run it. It is a standard Pluggable Authentication Module (PAM) framework that extends the Java 2 platform security architecture to support user-based authorization.

**What is Java Naming and Directory Service?** - The JNDI provides naming and directory functionality. It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Using JNDI, a J2EE application can store and retrieve any type of named Java object. Because JNDI is independent of any specific implementations, applications can use JNDI to access multiple naming and directory services, including existing naming and  
directory services such as LDAP, NDS, DNS, and NIS.

**What is Struts?** - A Web page development framework. Struts combines Java Servlets, Java Server Pages, custom tags, and message resources into a unified framework. It is a cooperative, synergistic platform, suitable for development teams, independent developers, and everyone between.

**How is the MVC design pattern used in Struts framework?** - In the MVC design pattern, application flow is mediated by a central Controller. The Controller delegates requests to an appropriate handler. The handlers are tied to a Model, and each handler acts as an adapter between the request and the Model. The Model represents, or encapsulates, an application’s business logic or state. Control is usually then forwarded back through the Controller to the appropriate View. The forwarding can be determined by consulting a set of mappings, usually loaded from a database or configuration file. This provides a loose coupling between the View and Model, which can make an application significantly easier to create and maintain. Controller: Servlet controller which supplied by Struts itself; View: what you can see on the screen, a JSP page and presentation components; Model: System state and a business logic JavaBeans.

**How to retrieve a key and Value from a map?**

Map<String, Object> map = new HashMap<String, Object>();   
  
for (String key : map.keySet()) {   
Object value = map.get(key);   
// ...   
}   
  
OR   
  
class Test{   
  
public static void main(String[] args) {   
Map map = new HashMap();   
map.put("man", "john");   
map.put("woman", "jane");   
System.out.println("Key for john is: " +   
getKey(map, "john"));   
}   
  
private static String getKey(Map map, String value) {   
for (Iterator i = map.keySet().iterator(); i.hasNext() {   
String key = (String)i.next();   
if (map.get(key).equals(value)) {   
return key;   
}   
}   
return "No value found";   
}   
}

**Why can an interface supertype call methods that belong to Object class?** - The members of an interface are:   
1) Those members declared in the interface.   
2) Those members inherited from direct superinterfaces.   
3) If an interface has no direct superinterfaces, then the interface implicitly declares a public abstract member method m with signature s, return type r, and throws clause t corresponding to each public instance method m with signature s, return type r, and throws clause t declared in Object, unless a method with the same signature, same return type, and a compatible throws clause is explicitly declared by the interface. It is a compile-time error if the interface explicitly declares such a method m in the case where m is declared to be final in Object.

**What is difference between shallow copy and deep copy ?** - Only instances of classes that implement the Cloneable interface can be cloned. Trying to clone an object that does not implement the Cloneable interface throws a CloneNotSupportedException. Both shallow copy and deep copy object nedd to implement Cloneable Interface.   
  
Shallow copy :   
  
The java.lang.Object root superclass defines a clone() method.   
default behavior of clone() is to return a shallow copy of the object. This means that the values of all of the original object?s fields are copied to the fields of the new object.   
If a shallow copy is performed on obj1, then it is copied but its contained objects are not.   
For Example:   
Public class Emp   
{   
Private Address address;   
}   
Emp emp1 = new Emp();   
Address add = new Address();   
emp1. address= add.   
  
If we clone   
Emp emp2 = emp1.clone();   
Then emp2.addess reference to the same Address object which emp1 refer.   
  
Deep Copy :   
  
A deep copy occurs when an object is copied along with the objects to which it refers.   
A deep copy makes a distinct copy of each of the object?s fields, recursing through the entire graph of other objects referenced by the object being copied. The Java API provides no deep-copy equivalent to Object.clone(). One solution is to simply implement your own custom method (e.g., deepCopy()) that returns a deep copy of an instance of one of your classes.   
Your Object Class should implements Serializable and Cloneable interface. .   
public static Object copy(Object orig) {   
  
Object obj = null;   
  
try {   
  
// Write the object out to a byte array   
  
ByteArrayOutputStream bo = new ByteArrayOutputStream();   
  
ObjectOutputStream out = new ObjectOutputStream(bo);   
  
out.writeObject(orig);   
  
out.flush();   
  
out.close();   
  
// Make an input stream from the byte array and read   
  
// a copy of the object back in.   
  
ObjectInputStream in = new ObjectInputStream(   
  
new ByteArrayInputStream(bos.toByteArray()));   
  
obj = in.readObject();   
  
}   
  
catch(IOException e) {   
  
e.printStackTrace();   
  
}   
  
catch(ClassNotFoundException cnfe) {   
  
cnfe.printStackTrace();   
  
}   
  
return obj;   
  
}

**Why to override equals() and hashCode()? and How i can implement both equals() and hashCode() for Set ?** - If you are implementing HashSet to store unique object then you need to implement equals() and hashcode() method.   
  
if two objects are equal according to the equals() method, they must have the same hashCode() value (although the reverse is not generally true).   
  
Two scenarios   
  
Case 1) : If you don't implement equals() and hashcode() method :   
  
When you are adding objects to HashSet , HashSet checks for uniqueness using   
equals() and hashcode() method the class ( ex. Emp class). If there is no equals() and hashcode() method the Emp class then HashSet checks default Object classes   
equals() and hashcode() method.   
  
In the Object class , equals method is   
public boolean equals(Object obj) {   
    return (this == obj);   
}   
Under this default implementation, two references are equal only if they refer to the exact same object. Similarly, the default implementation of hashCode() provided by Object is derived by mapping the memory address of the object to an integer value.   
  
This will fail to check if two Emp object with same employee name .   
For Example :   
Emp emp1 = new Emp();   
emp1.setName("sat");   
Emp emp2 = new Emp();   
emp2.setName("sat");   
Both the objects are same but based on the default above method both objects are dirrefent because references and hashcode are different .   
So in the HashSet you can find the duplicate emp objects.   
  
To overcome the issue equals() and hashcode() method need to override.   
Case 2) : If you override equals() and hashcode() method   
  
Example : implement equals and hashcode   
public class Emp {   
    private long empId;   
    String name = "";   
public boolean equals(Object o) {   
        if (o == this)   
        return true;   
        if (!(o instanceof Emp))   
        return false;   
        Emp emp = (Emp)o;   
        return emp. empId == empId &&   
        emp. name == name;   
           
        }   
       
    public int hashCode(){   
        int result = 10;   
        result = result \* new Integer(String.valueOf(empId)).intValue();   
        return result;   
    }   
}   
In the equals() , it check for name is same or not. This way you can find out the objects are equals or not.   
In the hashCode() also it return some unique value for each object. In this way if two Emp object has same empId then it will say both are same object.   
Now HashSet store only unique objects.   
If you do   
Emp emp1 = new Emp();   
emp1.setName("sat");   
Emp emp2 = new Emp();   
emp2.setName("sat");   
  
if(emp1.equals(emp2)){   
            System.out.println("equal");   
        }   
This will print : equal

**How to sort list of objects ( User Defined) using comparator?** - You can use Collections.sort(List,Comparator) to sort objects.   
Example : You have User Bean , you want to sort based on username filed.   
User Class :   
public class User {   
String userName = "";   
String city = "";   
String state = "";   
/\*\*     \* @return Returns the userName.   
     \*/   
    public String getUserName() {   
        return userName;   
    }   
    /\*\*   
     \* @param userName The userName to set.   
     \*/   
    public void setUserName(String userName) {   
        this.userName = userName;   
    }   
}   
  
Then you have to write a comparator class.   
public class UserNameComparator implements Comparator {   
     public int compare(Object user, Object anotherUser) {   
       
     String firstName1 = ((User) user).getUserName().toUpperCase();   
       
     String firstName2 = ((User) anotherUser).getUserName().toUpperCase();   
       
     return firstName1.compareTo(firstName2);   
       
     }   
    }   
  
Now sorting code   
User user1 = new User();   
        user1.setUserName("das");   
        User user2 = new User();   
        user2.setUserName("nick");   
        User user3 = new User();   
        user3.setUserName("ram");   
        User user4 = new User();   
        user4.setUserName("jadu");   
           
        ArrayList list = new ArrayList();   
        list.add(user1);   
        list.add(user2);   
        list.add(user3);   
        list.add(user4);   
           
        Collections.sort(list,new UserNameComparator()); // sort ascending order.   
  
// Descending order   
Collections.reverse(list);   
           
        for(int i=0;i<list.size();i++){   
            User usr = (User)list.get(i);   
            System.out.println(usr.getUserName());   
        }

**What is the difference between int and Interger?** - int is primitive type and Integer is Wrapper Class.   
For Example :   
int i=2;   
Integer i = new Integer (2);   
  
You can have question like why we need Wrapper Class(Integer )?   
Answer is, if you are using Collection objects like Hashtable etc. you can't use int as a key. You have to use object so you can use Integer class.   
  
Hashtable ht = new Hashtable ();   
ht.put(2, "USA") ;// You can not do this   
  
ht.put(new Integer(2), "USA") ;// you have to do like this.

**Singleton Double-checked locking in Java?**

public static Singleton getInstance()   
{   
if (instance == null)   
{   
synchronized(Singleton.class) { //1   
if (instance == null) //2   
instance = new Singleton(); //3   
}   
}   
return instance;   
}   
Thread 1 enters the getInstance() method.   
Thread 1 enters the synchronized block at //1 because instance is null.   
Thread 1 is preempted by thread 2.   
Thread 2 enters the getInstance() method.   
Thread 2 attempts to acquire the lock at //1 because instance is still null. However, because thread 1 holds the lock, thread 2 blocks at //1.   
Thread 2 is preempted by thread 1.   
Thread 1 executes and because instance is still null at //2, creates a Singleton object and assigns its reference to instance.   
Thread 1 exits the synchronized block and returns instance from the getInstance() method.   
Thread 1 is preempted by thread 2.   
Thread 2 acquires the lock at //1 and checks to see if instance is null.   
Because instance is non-null, a second Singleton object is not created and the one created by thread 1 is returned.

**What are the parameters to follow Creating and Destroying Objects in Java?** - Item 1: Consider providing static factory methods instead of   
constructors   
public static Boolean valueOf(boolean b) {   
return (b ? Boolean.TRUE : Boolean.FALSE);   
}   
advantage of static factory methods is that, unlike constructors, they are not   
required to create a new object each time they're invoked.   
This allows immutable classes   
(Item 13) to use preconstructed instances or to cache instances as they're constructed and to   
dispense these instances repeatedly so as to avoid creating unnecessary duplicate objects.   
The Boolean.valueOf(boolean) method illustrates this technique: It never creates an object.   
This technique can greatly improve performance if equivalent objects are requested   
frequently, especially if these objects are expensive to create.   
it allows an immutable class to ensure that no two equal instances exist:   
a.equals(b) if and only if a==b. If a class makes this guarantee, then its clients can use   
the == operator instead of the equals(Object) method, which may result in a substantial   
performance improvement   
implements this   
optimization, and the String.intern method implements it in a limited form.   
  
advantage of static factory methods is that, unlike constructors, they can return   
an object of any subtype of their return type. This gives you great flexibility in choosing   
the class of the returned object.   
One application of this flexibility is that an API can return objects without making their   
classes public. Hiding implementation classes in this fashion can lead to a very compact API.   
  
Item 2: Enforce the singleton property with a private constructor   
  
Item 4: Avoid creating duplicate objects   
It is often appropriate to reuse a single object instead of creating a new functionally equivalent   
object each time it is needed. Reuse can be both faster and more stylish. An object can always   
be reused if it is immutable   
As an extreme example of what not to do, consider this statement:   
String s = new String("silly"); // DON'T DO THIS!   
The statement creates a new String instance each time it is executed, and none of those   
object creations is necessary. The argument to the String constructor ("silly") is itself a   
String instance, functionally identical to all of the objects created by the constructor. If this   
usage occurs in a loop or in a frequently invoked method, millions of String instances can be   
created needlessly.   
The improved version is simply the following:   
String s = "No longer silly";   
Item 5: Eliminate obsolete object references   
So where is the memory leak? If a stack grows and then shrinks, the objects that were popped   
off the stack will not be garbage collected, even if the program using the stack has no more   
references to them. This is because the stack maintains obsolete references to these objects.   
An obsolete reference is simply a reference that will never be dereferenced again. In this case,   
any references outside of the ?active portion? of the element array are obsolete. The active   
portion consists of the elements whose index is less than size.   
  
public Object pop() {   
if (size == 0)   
throw new EmptyStackException();   
return elements[--size];   
}   
The fix for this sort of problem is simple: Merely null out references once they become   
obsolete. In the case of our Stack class, the reference to an item becomes obsolete as soon as   
it's popped off the stack. The corrected version of the pop method looks like this:   
public Object pop() {   
if (size==0)   
throw new EmptyStackException();   
Object result = elements[--size];   
elements[size] = null; // Eliminate obsolete reference   
return result;   
}   
Item 6: Avoid finalizers   
There is no guarantee that finalizers will be executed promptly. It can take   
arbitrarily long between the time that an object becomes unreachable and the time that its   
finalizer is executed. This means that nothing time-critical should ever be done by a   
finalizer. For example, it is a grave error to depend on a finalizer to close open files because   
open file descriptors are a limited resource. If many files are left open because the JVM is   
tardy in executing finalizers, a program may fail because it can no longer open files.

**What are the different scopes for Java variables?** - The scope of a Java variable is determined by the context in which the variable is declared. Thus a java variable can have one of the three scopes at any given point in time.   
1. Instance: - These are typical object level variables, they are initialized to default values at the time of creation of object, and remain accessible as long as the object accessible.   
2. Local: - These are the variables that are defined within a method. They remain accessible only during the course of method execution. When the method finishes execution, these variables fall out of scope.   
3. Static: - These are the class level variables. They are initialized when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance.

**What method must be implemented by all threads?** - All tasks must implement the run() method, whether they are a subclass of Thread or implement the Runnable interface.

**Can an unreachable object become reachable again?** - An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects**.**

What are the steps in the JDBC connection? - 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\" );   
Class.forName("com.mysql.jdbc.Driver");   
Step 2 : Now create a database connection using :   
Connection con = DriverManager.getConnection(url,username,password);   
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/art",art,art);   
Step 3: Now Create a query using :   
Statement stmt = Connection.Statement(\"select \* from EMP\");   
Step 4 : Exceute the query :   
ResultSet rs = stmt.exceuteQuery();   
while(rs.next()){   
System.out.println(rs.getString(1));   
}

**What is the difference between static and non-static variables?** - A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.   
For Example :   
public class Test{   
static int i=5;   
int j=7;   
}   
Test t1= new Test ();   
Test t2= new Test ();   
t1.j=10;   
t2.j=15;   
but for the case of static i ,   
t1.i=10;   
t2.i=15;   
System.out.println(t2.i) will give you 15 not 10 . because i is sharable and related to class not instance.

### What is the purpose of finalization? - The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.

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

### What is synchronization and why is it important? - With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors

### What is daemon thread and which method is used to create the daemon thread? - Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation for the java runtime system. setDaemon method is used to create a daemon thread.

### What are synchronized methods and synchronized statements? - Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement

### If I write System.exit (0); at the end of the try block, will the finally block still execute? - No, in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

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

**What is the basic difference between the 2 approaches to exception handling.**1> try catch block and   
2> specifying the candidate exceptions in the throws clause?   
When should you use which approach?

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

**If I write return at the end of the try block, will the finally block still execute?** - Yes even if you write return as the last statement in the try block and no exception occurs, the finally block will execute. The finally block will execute and then the control return

**What are the different ways to handle exceptions?** - There are two ways to handle exceptions,   
1. By wrapping the desired code in a try block followed by a catch block to catch the exceptions. and   
2. List the desired exceptions in the throws clause of the method and let the caller of the method hadles those exceptions.

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

**What is wrapper class? Explain with example?** - Java provides specialized classes corresponding to each of the primitive data types. These are called wrapper classes.

They are e.g. Integer, Character, Double etc.   
All wrapper classes are final. You can't extend the class.

What is serialization? Explain with example? - Serialization involves saving the current state of an object to a stream, and restoring an equivalent object from that stream.   
  
For an object to be serialized, it must be an instance of a class that implements either the Serializable or Externalizable interface. Both interfaces only permit the saving of data associated with an object's variables.   
They depend on the class definition being available to the Java Virtual Machine at reconstruction time in order to construct the object.   
The Serializable interface relies on the Java runtime default mechanism to save an object's state.   
Writing an object is done via the writeObject() method in the ObjectOutputStream class (or the ObjectOutput interface).   
Writing a primitive value may be done through the appropriate write<datatype>() method.   
Reading the serialized object is accomplished using the readObject() method of the ObjectInputStream class, and primitives may be read using the various read<datatype>() methods.   
  
The Externalizable interface specifies that the implementing class will handle the serialization on its own, instead of relying on the default runtime mechanism.   
This includes which fields get written (and read), and in what order.   
The class must define a writeExternal() method to write out the stream, and a corresponding readExternal() method to read the stream.   
Inside of these methods the class calls ObjectOutputStream writeObject(), ObjectInputStream readObject(), and any necessary write<datatype>() and read<datatype>() methods, for the desired fields.   
  
class DataOrder implements Serializable{   
String apples, peaches, pears, cardnum, custID;   
double icost;   
int itotal;   
}   
  
Saving the current state of an object to a stream   
DataOrder orders = new DataOrder();   
FileOutputStream fos =   
new FileOutputStream(orders);   
ObjectOutputStream oos =   
new ObjectOutputStream(fos);   
oos.writeObject(order);   
  
restoring an equivalent object from that stream   
FileInputStream fis =   
new FileInputStream(orders);   
ObjectInputStream ois =   
new ObjectInputStream(fis);   
order = (DataOrder)ois.readObject();

**What one should take care of while serializing the object?** - make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

### When you serialize an object, what happens to the object references included in the object? - The serialization mechanism generates an object graph for serialization. Thus it determines whether the included object references are serializable or not. This is a recursive process. Thus when an object is serialized, all the included objects are also serialized alongwith the original obect.

**What happens to the static fields of a class during serialization?** - There are three exceptions in which serialization doesnot necessarily read and write to the stream. These are   
1. Serialization ignores static fields, because they are not part of ay particular state state.   
2. Base class fields are only hendled if the base class itself is serializable.   
3. Transient fields.

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

**Primitive data types are passed by reference or pass by value?** - Primitive data types are passed by value.

**What type of parameter passing does Java support?** - Java only supports pass by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object.

**Can a top level class be private or protected?** - No. top level class can be public, abstract and final.

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

**What is the difference between declaring a variable and defining a variable?** - In declaration we just mention the type of the variable and it's name. We do not initialize it. But defining means declaration + initialization.   
e.g String s; is just a declaration while String s = new String ("abcd"); Or String s = "abcd"; are both definitions.

**What are different types of inner classes?** - Nested top-level classes, Member classes, Local classes, Anonymous classes   
  
Nested top-level classes- If you declare a class within a class and specify the static modifier, the compiler treats the class just like any other top-level class.   
Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package. eg, outer.inner. Top-level inner classes implicitly have access only to static variables.There can also be inner interfaces. All of these are of the nested top-level variety.   
  
Member classes - Member inner classes are just like other member methods and member variables and access to the member class is restricted, just like methods and variables. This means a public member class acts similarly to a nested top-level class. The primary difference between member classes and nested top-level classes is that member classes have access to the specific instance of the enclosing class.   
  
Local classes - Local classes are like local variables, specific to a block of code. Their visibility is only within the block of their declaration. In order for the class to be useful beyond the declaration block, it would need to implement a   
more publicly available interface.Because local classes are not members, the modifiers public, protected, private, and static are not usable.   
  
Anonymous classes - Anonymous inner classes extend local inner classes one level further. As anonymous classes have no name, you cannot provide a constructor.

**What is Overriding?** - When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass.   
When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass.   
Rules to folow :   
Methods may be overridden to be more public, not more private.   
Methods may be overridden to be thows the same exception or subclass of the exception thrown by the method of superclass.

**What are Checked and UnChecked Exception?** - A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses.   
Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown. eg, IOException thrown by java.io.FileInputStream's read() method·   
Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the   
exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. eg, StringIndexOutOfBoundsException thrown by String's charAt() method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be

**What is final?** - A final class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant).

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

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

**What are the modifiers in Java ?**

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

**Difference between HashMap and HashTable?**

a)The key difference between the two is that access to the Hashtable is synchronized while access to the HashMap is not synchronized.   
b)HashMap permits null values as Key, while Hashtable doesn't.   
c)iterator in a HashMap is fail-safe(Fail-safe 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) while the enumerator of HashTable is not.

### Difference between Vector and ArrayList?

Vectors are synchronized. Any method that touches the Vector's contents is thread safe.   
ArrayList, on the other hand, is unsynchronized, making them, therefore, not thread safe.   
With that difference in mind, using synchronization will incur a performance hit.   
So if you don't need a thread-safe collection, use the ArrayList don't use Vector.   
A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.   
Depending on how you use these classes, you could end up taking a large performance hit while adding new elements.   
If you don't know how much data you'll have, but you do know the rate at which it grows, Vector does possess a slight advantage since you can set the increment value.   
Both the ArrayList and Vector are good for retrieving elements from a specific position in the container or for adding and removing elements from the end of the container.   
All of these operations can be performed in constant time -- O(1).   
However, adding and removing elements from any other position proves more expensive -- linear to be exact: O(n-i), where n is the number of elements and i is the index of the element added or removed.   
For Example : ArrayList Has 5 elements and you want to add an element on 2nd position.   
Then arrayList add the element on 2nd position and shifted existing 2nd position element to 3rd , 3rd position element to 4th etc..   
For the case of remove just opposite.   
These operations are more expensive because you have to shift all elements at index i and higher over by one element

**Difference between ArrayList and LinkedList?**

ArrayList is good for adding and removing elements from the end of the container. All of these operations can be performed in constant time -- O(1).   
adding and removing elements from any other position proves more expensive -- linear to be exact: O(n-i), where n is the number of elements and i is the index of the element added or removed.   
For Example : ArrayList Has 5 elements and you want to add an element on 2nd position.   
Then arrayList add the element on 2nd position and shifted existing 2nd position element to 3rd , 3rd position element to 4th etc..   
  
LinkedList can add or remove an element at any position in constant time -- O(1). Indexing an element is a bit slower -- O(i) where i is the index of the element in case of LinkedList.   
  
Traversing an ArrayList is also easier since you can simply use an index instead of having to create an iterator.   
The LinkedList creates an internal object for each element inserted. So you have to be aware of the extra garbage being created.   
  
An ArrayList is a List implementation backed by a Java array.   
With a LinkedList, the List implementation is backed by a doubly linked list data structure

**What are pass by reference and passby value in Java?**

Pass By Reference means the passing the address itself rather than passing the value.   
Passby Value means passing a copy of the value to be passed.   
Java only supports pass by value. With objects, the object reference itself is passed by value and so both the original reference and parameter copy both refer to the same object.

**When we go for Abstract and Interface in Java?** - When the sub-types behaviour is totally different then you use an interface, when the sub-types behaviour is partially common and different with respect to the supertype an abstract class is used.   
In an abstract class the partially common behaviour is given a concrete implementation.   
Since there is no common behaviour between an interface and a sub-type an interface does not have an implementation for any of its behaviour.   
For Example - Abstract Class   
abstract public class Animal {   
public eatChicken(){   
System.out.println("Eat Chicken");   
}   
}   
  
public Lion extends Animal{   
}   
public Tiger extends Animal{   
}   
  
Both Lion and Tiger class has common behaviour eatChicken() and all the implementaion put into eatChicken() method of super calss.   
so we go for abstract class.   
  
Example- Interface   
  
Interface Animal{   
public eat();   
}   
public Lion implements Animal{   
public eat(){   
System.out.println("Lion eat Non Veg");   
}   
}   
public Cow implements Animal{   
public eat(){   
System.out.println("Cow eat Veg");   
}   
}   
Both Lion and Cow class has different behavior for eat() and they have different implementations. so we go for Interface.

**What is the difference between interface and abstract class?**

\* interface contains methods that must be abstract; abstract class may contain concrete methods.   
\* interface contains variables that must be static and final; abstract class may contain non-final and final variables.   
\* members in an interface are public by default, abstract class may contain non-public members.   
\* Interface is used to "implements"; whereas abstract class is used to "extends".   
\* interface can be used to achieve multiple inheritance; abstract class can be used as a single inheritance.   
\* interface can "extends" another interface, abstract class can "extends" another class and "implements" multiple interfaces.   
\* interface is absolutely abstract; abstract class can be invoked if a main() exists.   
\* interface is more flexible than abstract class because one class can only "extends" one super class, but "implements" multiple interfaces.   
\* If given a choice, use interface instead of abstract class.

**What are the Garbage collection algorithms in Java?** - Any garbage collection algorithm must do two basic things.   
First, it must detect garbage objects.   
Second, it must reclaim the heap space used by the garbage objects and make it available to the program.   
Garbage detection is ordinarily accomplished by defining a set of roots and determining reachability from the roots.   
An object is reachable if there is some path of references from the roots by which the executing program can access the object.   
The roots are always accessible to the program. Any objects that are reachable from the roots are considered live.   
Objects that are not reachable are considered garbage, because they can no longer affect the future course of program execution.   
  
    a. Reference counting collectors   
Reference counting was an early garbage collection strategy.   
here a reference count is maintained for each object.   
When an object is first created its reference count is set to one.   
When any other object or root is assigned a reference to that object, the object's count is incremented.   
When a reference to an object goes out of scope or is assigned a new value, the object's count is decremented.   
Any object with a reference count of zero can be garbage collected.   
When an object is garbage collected, any objects that it refers to has their reference counts decremented.   
In this way the garbage collection of one object may lead to the subsequent garbage collection of other objects.   
  
b. Tracing collectors   
Tracing garbage collectors trace out the graph of object references starting with the root nodes.   
Objects that are encountered during the trace are marked in some way.   
Marking is generally done by either setting flags in the objects themselves or by setting flags in a separate bitmap.   
After the trace is complete, unmarked objects are known to be unreachable and can be garbage collected.   
The basic tracing algorithm is called mark and sweep.   
This name refers to the two phases of the garbage collection process.   
In the mark phase, the garbage collector traverses the tree of references and marks each object it encounters.   
In the sweep phase unmarked objects are freed, and the resulting memory is made available to the executing program.   
In the JVM the sweep phase must include finalization of objects.   
  
    c. Compacting collectors   
Garbage collectors of JVMs will likely have a strategy to combat heap fragmentation.   
Two strategies commonly used by mark and sweep collectors are compacting and copying.   
Both of these approaches move objects on the fly to reduce heap fragmentation.   
Compacting collectors slide live objects over free memory space toward one end of the heap.   
In the process the other end of the heap becomes one large contiguous free area.   
All references to the moved objects are updated to refer to the new location.   
  
    d. Copying collectors   
Copying garbage collectors move all live objects to a new area.   
As the objects are moved to the new area, they are placed side by side,   
thus eliminating any free spaces that may have separated them in the old area.   
The old area is then known to be all free space.   
The advantage of this approach is that objects can be copied as they are discovered by the traversal from the root nodes.   
There are no separate mark and sweep phases.   
Objects are copied to the new area on the fly, and forwarding pointers are left in their old locations.   
The forwarding pointers allow objects encountered later in the traversal that refer to already copied objects to know the new location of the copied objects.

**What is garbage collection and the purpose of garbage collection in Java?** - The JVM's heap stores all objects created by an executing Java program.   
Objects are created by Java's "new" operator, and memory for new objects is allocated on the heap at run time.   
Garbage collection is the process of automatically freeing objects that are no longer referenced by the program.   
This frees the programmer from having to keep track of when to free allocated memory,   
thereby preventing many potential bugs and headaches.   
The name "garbage collection" implies that objects that are no longer needed by the program are "garbage" and can be thrown away.   
Other word we can say memory recycling.   
When an object is no longer referenced by the program, the heap space it occupies must be recycled so that the space is available for subsequent new objects.   
The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects.   
In the process of freeing unreferenced objects, the garbage collector must run any finalizers of objects being freed.

**What is the difference between an Interface and an Abstract class?** - An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

**What are the steps involved in establishing a JDBC connection?**  - This action involves two steps: loading the JDBC driver and making the connection.

**How can you load the drivers?** - Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:

*Class.forName(”sun.jdbc.odbc.JdbcOdbcDriver”);*

Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ, you would load the driver with the following line of code:

*Class.forName(”jdbc.DriverXYZ”);*

**What will Class.forName do while loading drivers?**  - It is used to create an instance of a driver and register it with the DriverManager. When you have loaded a driver, it is available for making a connection with a DBMS.

**How can you make the connection?**  - To establish a connection you need to have the appropriate driver connect to the DBMS.  
The following line of code illustrates the general idea:

*String url = “jdbc:odbc:Fred”;  
Connection con = DriverManager.getConnection(url, “Fernanda”, “J8?);*

**How can you create JDBC statements and what are they?** - A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate. It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object

*Statement stmt = con.createStatement();*

**How can you retrieve data from the ResultSet?** - JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs.

*ResultSet rs = stmt.executeQuery(”SELECT COF\_NAME, PRICE FROM COFFEES”);  
String s = rs.getString(”COF\_NAME”);*

The method getString is invoked on the ResultSet object rs, so getString() will retrieve (get) the value stored in the column COF\_NAME in the current row of rs.

**What are the different types of Statements?** - Regular statement (use createStatement method), prepared statement (use prepareStatement method) and callable statement (use prepareCall)

**How can you use PreparedStatement?**  - This special type of statement is derived from class Statement.If you need a Statement object to execute many times, it will normally make sense to use a PreparedStatement object instead. The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement’s SQL statement without having to compile it first.

PreparedStatement updateSales =con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF\_NAME LIKE ?");

**What does setAutoCommit do?** - When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed. The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode:

*con.setAutoCommit(false);*

Once auto-commit mode is disabled, no SQL statements will be committed until you call the method commit explicitly.

con.setAutoCommit(false);

PreparedStatement updateSales =

con.prepareStatement( "UPDATE COFFEES SET SALES = ? WHERE COF\_NAME LIKE ?");

updateSales.setInt(1, 50); updateSales.setString(2, "Colombian");

updateSales.executeUpdate();

PreparedStatement updateTotal =

con.prepareStatement("UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF\_NAME LIKE ?");

updateTotal.setInt(1, 50);

updateTotal.setString(2, "Colombian");

updateTotal.executeUpdate();

con.commit();

con.setAutoCommit(true);

**How do you call a stored procedure from JDBC?** - The first step is to create a CallableStatement object. As with Statement an and PreparedStatement objects, this is done with an open  
Connection object. A CallableStatement object contains a call to a stored procedure.

CallableStatement cs = con.prepareCall("{call SHOW\_SUPPLIERS}");

ResultSet rs = cs.executeQuery();

**How do I retrieve warnings?** - SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an  
application, as exceptions do; they simply alert the user that something did not happen as planned. A warning can be reported on a  
Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these  
classes has a getWarnings method, which you must invoke in order to see the first warning reported on the calling object:

SQLWarning warning = stmt.getWarnings();

if (warning != null)

{

System.out.println("n---Warning---n");

while (warning != null)

{

System.out.println("Message: " + warning.getMessage());

System.out.println("SQLState: " + warning.getSQLState());

System.out.print("Vendor error code: ");

System.out.println(warning.getErrorCode());

System.out.println("");

warning = warning.getNextWarning();

}

}

**How can you move the cursor in scrollable result sets?** - One of the new features in the JDBC 2.0 API is the ability to move a result set’s cursor backward as well as forward. There are also methods that let you 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 COF\_NAME, PRICE FROM COFFEES”);*

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. The point to remember here is that if you specify a type, you must also specify whether it is read-only or updatable. Also, you must specify the type first, and because both parameters are of type int , the compiler will not complain if you switch the order. 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.

**What’s the difference between TYPE\_SCROLL\_INSENSITIVE , and TYPE\_SCROLL\_SENSITIVE?** - 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 COF\_NAME, PRICE FROM COFFEES");

srs.afterLast();

while (srs.previous())

{

String name = srs.getString("COF\_NAME");

float price = srs.getFloat("PRICE");

System.out.println(name + " " + price);

}

**How to Make Updates to Updatable Result Sets?** - Another new feature in the JDBC 2.0 API is the ability to update rows in a result set using methods in the Java programming language rather than having to send an SQL command. But before you can take advantage of this capability, you need to create a ResultSet object that is updatable. In order to do this, you supply the ResultSet constant CONCUR\_UPDATABLE to the createStatement method.

Connection con =DriverManager.getConnection("jdbc:mySubprotocol:mySubName");

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

ResultSet uprs =stmt.executeQuery("SELECT COF\_NAME, PRICE FROM COFFEES");

**What is the JDBC?** - Java Database Connectivity (**JDBC**) is a standard Java API to interact with relational databases form Java. **JDBC** has set of classes and interfaces which can use from Java application and talk to database without learning RDBMS details and using Database Specific JDBC Drivers.

**What are the new features added to JDBC 4.0?** - The major features added in JDBC 4.0 include:

* Auto-loading of JDBC driver class
* Connection management enhancements
* Support for RowId SQL type
* DataSet implementation of SQL using Annotations
* SQL exception handling enhancements
* SQL XML support

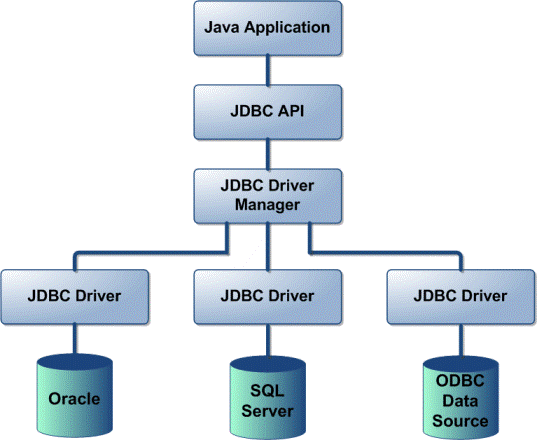
**Explain Basic Steps in writing a Java program using JDBC?** - JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database :

* Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).
* Open the connection to database which is then used to send SQL statements and get results back.
* Create JDBC Statement object. This object contains SQL query.
* Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
* Process the result set.
* Close the connection.

**Exaplain the JDBC Architecture.** - The JDBC Architecture consists of two layers:

* The JDBC API, which provides the **application-to-JDBC Manager** connection.
* The JDBC Driver API, which supports the **JDBC Manager-to-Driver** Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure 1.



**Figure 1: JDBC Architecture**

**What are the main components of JDBC ?** - The life cycle of a servlet consists of the following phases:

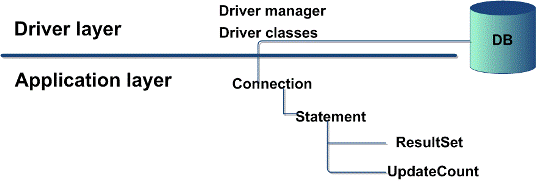
* **DriverManager**: Manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* **Driver**: The database communications link, handling all communication with the database. Normally, once the driver is loaded, the developer need not call it explicitly.
* **Connection**: Interface with all methods for contacting a database.The connection object represents communication context, i.e., all communication with database is through connection object only.
* **Statement** : Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.
* **ResultSet**: The ResultSet represents set of rows retrieved due to query execution.

**How the JDBC application works?** - A JDBC application can be logically divided into two layers:

1. **Driver layer**

2. **Application layer**

* Driver layer consists of DriverManager class and the available JDBC drivers.
* The application begins with requesting the DriverManager for the connection.
* An appropriate driver is choosen and is used for establishing the connection. This connection is given to the application which falls under the application layer.
* The application uses this connection to create Statement kind of objects, through which SQL commands are sent to backend and obtain the results.



**Figure 2: JDBC Application**

**How do I load a database driver with JDBC 4.0 / Java 6?** - Provided the JAR file containing the driver is properly configured, just place the JAR file in the classpath. Java developers **NO** longer need to explicitly load JDBC drivers using code like Class.forName() to register a JDBC driver.The DriverManager class takes care of this by automatically locating a suitable driver when the DriverManager.getConnection() method is called. This feature is backward-compatible, so no changes are needed to the existing JDBC code.

**What is JDBC Driver interface?** - The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendor driver must provide implementations of the java.sql.Connection,Statement,PreparedStatement, CallableStatement, ResultSet and Driver.

**What does the connection object represents?** - The connection object represents communication context, i.e., all communication with database is through connection object only.

**What is Statement ?** - Statement acts like a vehicle through which SQL commands can be sent. Through the connection object we create statement kind of objects.  
Through the connection object we create statement kind of objects.

Statement stmt = conn.createStatement();

This method returns object which implements statement interface.

**What is PreparedStatement?** - A prepared statement is an SQL statement that is precompiled by the database. Through precompilation, prepared statements improve the performance of SQL commands that are executed multiple times (given that the database supports prepared statements). Once compiled, prepared statements can be customized prior to each execution by altering predefined SQL parameters.

PreparedStatement pstmt = conn.prepareStatement("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");

pstmt.setBigDecimal(1, 153833.00);

pstmt.setInt(2, 110592);

Here:connis an instance of the Connection class and"**?**"represents parameters. These parameters must be specified before execution.

**What is the difference between a Statement and a PreparedStatement?**

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| A standard Statement is used to create a Java representation of a literal SQL statement and execute it on the database. | A PreparedStatement is a precompiled  statement. This means that when the PreparedStatement is executed, the RDBMS can just run the PreparedStatement SQL statement without having to compile it first. |
| Statement has to verify its metadata against the database every time. | While a prepared statement has to verify its metadata against the database only once. |
| If you want to execute the SQL statement once go for STATEMENT | If you want to execute a single SQL statement multiple number of times, then go for PREPAREDSTATEMENT. PreparedStatement objects can be reused with passing different values to the queries |

**What are callable statements ?** - Callable statements are used from JDBC application to invoke stored procedures and functions.

**How to call a stored procedure from JDBC ?** - PL/SQL stored procedures are called from within JDBC programs by means of the prepareCall() method of the Connection object created. A call to this method takes variable bind parameters as input parameters as well as output variables and creates an object instance of the CallableStatement class.

The following line of code illustrates this:

CallableStatement stproc\_stmt = conn.prepareCall("{call procname(?,?,?)}");

Here conn is an instance of the Connection class.

**Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?** - No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.

**Which is the right type of driver to use and when?**

* Type I driver is handy for prototyping
* Type III driver adds security, caching, and connection control
* Type III and Type IV drivers need no pre-installation

**What are the standard isolation levels defined by JDBC?** - The values are defined in the class java.sql.Connection and are:

* TRANSACTION\_NONE
* TRANSACTION\_READ\_COMMITTED
* TRANSACTION\_READ\_UNCOMMITTED
* TRANSACTION\_REPEATABLE\_READ
* TRANSACTION\_SERIALIZABLE

Any given database may not support all of these levels.

**What is resultset ?** - The ResultSet represents set of rows retrieved due to query execution.

ResultSet rs = stmt.executeQuery(sqlQuery);

**What are the types of resultsets?** - The values are defined in the class java.sql.Connection and are:

* TYPE\_FORWARD\_ONLY specifies that a resultset is not scrollable, that is, rows within it can be advanced only in the forward direction.
* TYPE\_SCROLL\_INSENSITIVE specifies that a resultset is scrollable in either direction but is insensitive to changes committed by other transactions or other statements in the same transaction.
* TYPE\_SCROLL\_SENSITIVE specifies that a resultset is scrollable in either direction and is affected by changes committed by other transactions or statements within the same transaction.

***Note****:*A TYPE\_FORWARD\_ONLY*resultset is always insensitive.*

**What’s the difference between TYPE\_SCROLL\_INSENSITIVE and TYPE\_SCROLL\_SENSITIVE?**

|  |  |
| --- | --- |
| **TYPE\_SCROLL\_INSENSITIVE** | **TYPE\_SCROLL\_SENSITIVE** |
| An insensitive  resultset is like the snapshot of the data in the database when query was executed. | A sensitive resultset does NOT represent a snapshot of data, rather it contains points to those rows which satisfy the query condition. |
| After we get the resultset the changes made to data are not visible through the resultset, and hence they are known as insensitive. | After we obtain the resultset if the data is modified then such modifications are visible through resultset. |
| Performance not effected with insensitive. | Since a trip is made for every ‘**get’**operation, the performance drastically get affected. |

**What is rowset?** - A RowSet is an object that encapsulates a set of rows from either Java Database Connectivity (JDBC) result sets or tabular data sources like a file or spreadsheet. RowSets support component-based development models like JavaBeans, with a standard set of properties and an event notification mechanism.

**What are the different types  of RowSet ?** – There are two types of RowSet are there. They are:

* **Connected**- A connected RowSet object connects to the database once and remains connected until the application terminates.
* **Disconnected -** A disconnected RowSet object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected RowSet while it is disconnected. Modified data can be updated in the database after a disconnected RowSet reestablishes the connection with the database.

**What is the need of BatchUpdates?** - The BatchUpdates feature allows us to group SQL statements together and send to database server in one single trip.

**What is a DataSource?** - A DataSource object is the representation of a data source in the Java programming language. In basic terms,

* A DataSource is a facility for storing data.
* DataSource can be referenced by JNDI.
* Data Source may point to RDBMS, file System , any DBMS etc..

**What are the advantages of DataSource?** - The few advantages of data source are:

* An application does not need to hardcode driver information, as it does with the DriverManager.
* The DataSource implementations can easily change the properties of data sources. For example: There is no need to modify the application code when making changes to the database details.
* The DataSource facility allows developers to implement aDataSource class to take advantage of features like connection pooling and distributed transactions.

**What is connection pooling? What is the main advantage of using connection pooling?** - A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested..

**What is JDBC?** - JDBC technology is an API (included in both J2SE and J2EE releases) that provides cross-DBMS connectivity to a wide range of SQL databases and [access](http://click.adbrite.com/mb/click.php?sid=417387&banner_id=11424124&variation_id=504733&uts=1201570213&cpc=302e30313032&keyword_id=22613&inline=y&zk_id=22702689&ab=168362148&sscup=8061fcb34e010df2557fcc2a609fa4bb&sscra=7460e13d996801ce685741f21db0eac1&ub=2057309322&rs=&r=) to other tabular datasources, such as spreadsheets or flat files. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.

**What are stored procedures?** - A stored procedure is a set of statements/commands which reside in the database. The stored procedure is precompiled. Each Database has it's own stored procedure language.

**What is JDBC Driver ?** - A The JDBC Driver provides vendor-specific implementations of the abstract classes provided by the JDBC API. This driver is used to connect to the database.

**What are the steps required to execute a query in JDBC?** - A First we need to create an instance of a JDBC driver or load JDBC drivers, then we need to register this driver with DriverManager class. Then we can open a connection. By using this connection , we can create a statement object and this object will help us to execute the query.

**What is DriverManager ?** - A DriverManager is a class in java.sql package. It is the basic service for managing a set of JDBC drivers.

**What is a ResultSet ?** - A table of data representing a database result set, which is usually generated by executing a statement that queries the database.  
  
A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

**What is Connection?** - A Connection class represents a connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.  
A Connection object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the getMetaData method.

**What does Class.forName return?** - A class as loaded by the classloader.

**What is Connection pooling?** - A Connection pooling is a technique used fo sharing server resources among requesting clients. Connection pooling increases the performance of Web applications by reusing active database connections instead of creating a new connection with every request. Connection pool manager maintains a pool of open database connections.

**What are the different JDB drivers available**? - There are mainly four type of JDBC drivers available. They are:  
  
Type 1 : JDBC-ODBC Bridge Driver - 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.

Type 2: Native API Partly Java Driver - A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or otherDBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.  
  
Type 3: Network protocol Driver - 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.  
  
Type 4: JDBC Net pure Java Driver - 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 is the fastest type of JDBC driver?** - Type 4 (JDBC Net pure Java Driver) is the fastest JDBC driver. Type 1 and Type 3 drivers will be slower than Type 2 drivers (the database calls are make at least three translations versus two), and Type 4 drivers are the fastest (only one translation).

**Is the JDBC-ODBC Bridge multi-threaded?** - No. The JDBC-ODBC Bridge does not support multi threading. The JDBC-ODBC Bridge uses synchronized methods to serialize all of the calls that it makes to ODBC. Multi-threaded Java programs may use the Bridge, but they won't get the advantages of multi-threading.

**What is cold backup, hot backup, warm backup recovery?** - Cold backup means all these files must be backed up at the same time, before the database is restarted. Hot backup (official name is 'online backup’) is a backup taken of each tablespace while the database is running and is being accessed by the users

**What is the advantage of denormalization?** - Data denormalization is reverse procedure, carried out purely for reasons of improving performance. It maybe efficient for a high-throughput system to replicate data for certain data.

**How do you handle your own transaction ?** - Connection Object has a method called setAutocommit( boolean flag). For handling our own transaction we can set the parameter to false and begin your transaction . Finally commit the transaction by calling the commit method.

**I have the choice of manipulating database data using a byte[] or a java.sql.Blob. Which has best performance?** - java.sql.Blob, 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 insertioninto the database, you should use a byte[] since data has not been uploaded to the database yet. Thus, use the Blob class only for extraction.

**Conclusion**: use the java.sql.Blob class for extraction whenever you can.

**I have the choice of manipulating database data using a String or a java.sql.Clob. Which has best performance?** - java.sql.Clob, since it does not extract any data from the database until you explicitly ask it to. The Java platform 2 type Clob wraps a database locator (which is essentially a pointer to char). 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 Clob content. For insertioninto the database, you should use a String since data need not been downloaded from the database. Thus, use the Clob class only for extraction.

**Conclusion:** Unless you always intend to extract the full textual data stored in the particular table cell, use the java.sql.Clob class for extraction whenever you can.

**Do I need to commit after an INSERT call in JDBC or does JDBC do it automatically in the DB?** - If your autoCommit flag (managed by the Connection.setAutoCommit method) is false, you are required to call the commit() method - and vice versa.

**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 so on ?**  - 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.

**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?** - 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 procedurethat 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.

**What are SQL3 data types?** - 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.

You can retrieve, store and update SQL3 types using the corresponding getXXX(), setXXX(), and updateXXX() methods defined in ResultSet interface

**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.

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 underbar.

**What is SQLJ and why would I want to use it instead of JDBC?** - SQL/J is a technology, originally developed by Oracle Corporation that enables you to embed SQL statements in Java. The purpose of the SQLJ API is to simplify the development requirements of the JDBC API while doing the same thing. Some major databases (Oracle, Sybase) support SQLJ, but others do not. Currently, SQLJ has not been accepted as a standard, so if you have to learn one of the two technologies, I recommend JDBC.

**How do I insert an image file (or other raw data) into a database?** - All raw data types (including binary documents or images) should be read and uploaded to the database as an array of bytes, byte[]. Originating from a binary file,

1. Read all data from the file using a FileInputStream.
2. Create a byte array from the read data.
3. Use method setBytes(int index, byte[] data); of java.sql.PreparedStatement to upload the data.

**How can I pool my database connections so I don't have to keep reconnecting to the database?**

* you gets a reference to the pool
* you gets a free connection from the pool
* you performs your different tasks
* you frees the connection to the pool

Since your application retrieves a pooled connection, you don't consume your time to connect / disconnect from your data source

**Will a call to PreparedStatement.executeQuery() always close the ResultSet from the previous executeQuery()?** - 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.

**How do I upload SQL3 BLOB & CLOB data to a database?** - Although one may simply extract BLOB & CLOB data from the database using the methods of the java.sql.CLOB and java.sql.BLOB, one must upload the data as normal java data types. The example below inserts a BLOB in the form of a byte[] and a CLOB in the form of a String into the database

|  |
| --- |
| Inserting SQL3 type data [BLOB & CLOB] |
| private void runInsert() {  try {  // Log  this.log("Inserting values ... ");    // Open a new Statement  PreparedStatement stmnt = conn.prepareStatement(  "insert Lobtest (image, name) values (?, ?)");    // Create a timestamp to measure the insert time  Date before = new java.util.Date();    for(int i = 0; i <>  // Set parameters  stmnt.setBytes(1, blobData);  stmnt.setString(2, "i: " + i + ";" + clobData);    // Perform insert  int rowsAffected = stmnt.executeUpdate();  }    // Get another timestamp to complete the time measurement  Date after = new java.util.Date();  this.log(" ... Done!");  log("Total run time: " + (  after.getTime() - before.getTime()));    // Close database resources  stmnt.close();  } catch(SQLException ex) {  this.log("Hmm... " + ex);  }  } |

**What is the difference between client and server database cursors?** - What you see on the client side is the current row of the cursor which called a Result (ODBC) or ResultSet (JDBC). The cursor is a server-side entity only and remains on the[server side](http://faqs.javabeat.net/jdbc/jdbc-interview-questions-faqs-1.php).

**Are prepared statements faster because they are compiled? if so, where and when are they compiled?** - Prepared Statements aren't actually compiled, but they are bound by theJDBC 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. For additional information on Prepared Statement performance and binding see JDBC Performance Tips on IBM'swebsite.

**Is it possible to connect to multiple databases simultaneously? Can one extract/update data from multiple databases with a single statement?** - 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

**Why do I get an UnsupportedOperationException?** - 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.

**What advantage is there to using prepared statements if I am using connection pooling or closing the connection frequently to avoid resource/connection/cursor limitations?** - The ability to choose the 'best' efficiency (or evaluate tradeoffs, if you prefer, ) is, at times, the most important piece of a mature developer's skill set. This is YAA (Yet Another Area) where that maxim applies. Apparently there is an effort to allow prepared statements to work 'better' with connection pools in JDBC 3.0, but for now, one loses most of the original benefit of prepared statements when the connection is closed. A prepared statement obviously fits best when a statement differing only in variable criteria is executed over and over without closing the statement.

However, depending on the DB engine, the SQL may be cached and reused even for a different prepared statement and most of the work is done by the DB engine rather than the driver. In addition, prepared statements deal with data conversions that can be error prone in straight ahead, built on the fly SQL; handling quotes and dates in a manner transparent to the developer, for example.

**What is JDBC, anyhow?** - JDBC is Java's means of dynamically accessing tabular data, and primarily data in relational databases, in a generic manner, normally using standard SQL statements.

**Can I reuse a Statement or must I create a new one for each query?** - When using a JDBC compliant driver, you can use the same Statement for any number of queries. However, some older drivers did not always "respect the spec." Also note that a Statement SHOULD automatically close the current ResultSet before executing a new query, so be sure you are done with it before re-querying using the same Statement.

**What is a three-tier architecture?** - A three-tier architecture is any system which enforces a general separation between the following three parts:

1. Client Tier or user interface
2. Middle Tier or business logic
3. Data Storage Tier

Applied to web applications and distributed programming, the three logical tiers usually correspond to the physical separation between three types of devices or hosts:

**What separates one tier from another in the context of n-tiered architecture?** - It depends on the application.

In a web application, for example, where tier 1 is a web-server, it may communicate with a tier 2 Application Server using RMI over IIOP, and subsequently tier 2 may communicate with tier 3 (data storage) using JDBC, etc.

Each of these tiers may be on separate physical machines or they may share the same box.

The important thing is the functionality at each tier.

* Tier 1 - Presentation - should be concerned mainly with display of user interfaces and/or data to the client browser or client [system](http://faqs.javabeat.net/jdbc/jdbc-interview-questions-faqs-4.php).
* Tier 2 - Application - should be concerned with business logic

Tier 3+ - Storage/Enterprise Systems - should be focused on data persistence and/or communication with other Enterprise Systems.

**What areas should I focus on for the best performance in a JDBC application?** - 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 Tablexyz*.
* Always close Statement and ResultSet objects as soon as possible.
* Write modular classes to handle database interaction specifics.
* Work with DatabaseMetaData to get information about database functionality.
* Softcode database specific parameters with, for example, properties files.
* Always catch AND handle database warnings and exceptions. Be sure to check for additional pending exceptions.
* Test your code with debug statements to determine the time it takes to execute your query and so on to help in tuning your code. Also use query plan functionality if available.
* Use proper ( and a single standard if possible ) formats, especially for dates.
* Use proper data types for specific kind of data. For example, store birthdate as a date type rather than, say, varchar.

**How can I insert multiple rows into a database in a single transaction?**

//turn off the implicit commit

Connection.setAutoCommit(false);

//..your insert/update/delete goes here

Connection.Commit();

a new transaction is implicitly started.

**How do I convert a java.sql.Timestamp to a java.util.Date?** - While Timesteamp extends Date, it stores the fractional part of the time within itself instead of within the Date superclass. If you need the partial seconds, you have to add them back in.

Date date = new Date(ts.getTime() + (ts.getNanos() / 1000000 ));

**What is SQL?** - SQL is a standardized language used to create, manipulate, examine, and manage relational databases.

**Is Class.forName(Drivername) the only way to load a driver? Can I instantiate the Driver and use the object of the driver?** - 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.

**What's new in JDBC 3.0?** - Probably the new features of most interest are:

* Savepoint support
* Reuse of prepared statements by connection pools
* Retrieval of auto-generated keys
* Ability to have multiple open ResultSet objects
* Ability to make internal updates to the data in Blob and Clob objects
* Ability to Update columns containing BLOB, CLOB, ARRAY and REF types
* Both java.sql and javax.sql ( JDBC 2.0 Optional Package ) are expected to be included with J2SE 1.4.

**Why do I get the message "No Suitable Driver"?** - Often the answer is given that the correct driver is not loaded. This may be the case, but more typically, the JDBC database URL passed is not properly constructed. When a Connection request is issued, the DriverManager asks each loaded driver if it understands the URL sent. If no driver responds that it understands the URL, then the "No Suitable Driver" message is returned.

**When I create multiple Statements on my Connection, only the current Statement appears to be executed. What's the problem?** - All JDBC objects are required to be threadsafe. Some drivers, unfortunately, implement this requirement by processing Statements serially. This means that additional Statements are not executed until the preceding Statement is completed.

**Can a single thread open up multiple connections simultaneously for the same database and for same table?** - The general answer to this is yes. If that were not true, connection pools, for example, would not be possible. As always, however, this is completely dependent on the JDBC driver.

You can find out the theoretical maximum number of active Connections that your driver can obtain via the DatabaseMetaData.getMaxConnections method.

**Can I ensure that my app has the latest data?** - 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.

**What does *normalization* mean for java.sql.Date and java.sql.Time?** - These classes are thin wrappers extending java.util.Date, which has both date and time components. java.sql.Date should carry only date information and a normalized instance has the time information set to zeros. java.sql.Time should carry only time information and a normalized instance has the date set to the Java epoch ( January 1, 1970 ) and the milliseconds portion set to zero.

**What's the best way, in terms of performance, to do multiple insert/update statements, a PreparedStatement or Batch Updates?** - 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.

**What is JDO?** - JDO provides for the transparent persistence of data in a data store agnostic manner, supporting object, hierarchical, as well as relational stores.

**What is the difference between setMaxRows(int) and SetFetchSize(int)? Can either reduce processing time?** - 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 number 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.

**What is DML? -** DML is an abbreviation for Data Manipulation Language. This portion of the SQL standard is concerned with manipulating the data in a database as opposed to the structure of a database. The core verbs for DML are SELECT, INSERT, DELETE, UPDATE, COMMIT and ROLLBACK.

**What is DDL?** - DDL is an abbreviation for Data Definition Language. This portion of the SQL standard is concerned with the creation, deletion and modification of database objects like tables, indexes and views. The core verbs for DDL are CREATE, ALTER and DROP. While most DBMS engines allow DDL to be used dynamically (and available to JDBC), it is often not supported in transactions.

**How can I get information about foreign keys used in a table?** - DatabaseMetaData.getImportedKeys() returns a ResultSet with data about foreign key columns, tables, sequence and update and delete rules.

**What isolation level is used by the DBMS when inserting, updating and selecting rows from a database? -** 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).

**1. Which of the following are valid definitions of an application’s main( ) method?**  
a) public static void main();  
b) public static void main( String args );  
c) public static void main( String args[] );  
d) public static void main( Graphics g );  
e) public static boolean main( String args[] );

**2. If MyProg.java were compiled as an application and then run from the command line as:**

java MyProg I like tests

**what would be the value of args[ 1 ] inside the main( ) method?**  
a) MyProg  
b) "I"  
c) "like"  
d) 3  
e) 4  
f) null until a value is assigned  
  
**3. Which of the following are Java keywords?**  
a) array  
b) boolean  
c) Integer  
d) protect  
e) super

**4. After the declaration:**

char[] c = new char[100];

what is the value of c[50]?  
a) 50  
b) 49  
c) ‘\u0000′  
d) ‘\u0020′  
e) " "  
f) cannot be determined  
g) always null until a value is assigned

**5. After the declaration:**

int x;

**the range of x is:**  
a) -231 to 231-1  
b) -216 to 216 -1   
c) -232 to 232   
d) -216 to 216  
e) cannot be determined; it depends on the machine

**6. Which identifiers are valid?**  
a) \_xpoints  
b) r2d2  
c) bBb$  
d) set-flow  
e) thisisCrazy

**7. Represent the number 6 as a hexadecimal literal.**  
  
8. Which of the following statements assigns "Hello Java" to the String variable s?  
a) String s = "Hello Java";  
b) String s[] = "Hello Java";  
c) new String s = "Hello Java";  
d) String s = new String("Hello Java");

**9. An integer, x has a binary value (using 1 byte) of 10011100. What is the binary value of z after these statements:**

int y = 1 << 7;  
int z = x & y;

a) 1000 0001  
b) 1000 0000  
c) 0000 0001  
d) 1001 1101  
e) 1001 1100

**10. Which statements are accurate:**  
a) >> performs signed shift while >>> performs an unsigned shift.  
b) >>> performs a signed shift while >> performs an unsigned shift.  
c) << performs a signed shift while <<< performs an unsigned shift.  
d) <<< performs a signed shift while << performs an unsigned shift.

**11. The statement …**

String s = "Hello" + "Java";

yields the same value for s as …

String s = "Hello";  
String s2= "Java";  
s.concat( s2 );

True  
False

**12. If you compile and execute an application with the following code in its main() method:**

String s = new String( "Computer" );

if( s == "Computer" )

System.out.println( "Equal A" );

if( s.equals( "Computer" ) )

System.out.println( "Equal B" );

a) It will not compile because the String class does not support the = = operator.  
b) It will compile and run, but nothing is printed.  
c) "Equal A" is the only thing that is printed.  
d) "Equal B" is the only thing that is printed.  
e) Both "Equal A" and "Equal B" are printed.

**13. Consider the two statements:**

1. boolean passingScore = false && grade == 70;

2. boolean passingScore = false & grade == 70;

The expression

grade == 70

is evaluated:  
a) in both 1 and 2  
b) in neither 1 nor 2  
c) in 1 but not 2  
d) in 2 but not 1  
e) invalid because false should be FALSE

**14. Given the variable declarations below:**

byte myByte;  
int myInt;  
long myLong;  
char myChar;  
float myFloat;  
double myDouble;

**Which one of the following assignments would need an explicit cast?**  
a) myInt = myByte;  
b) myInt = myLong;  
c) myByte = 3;  
d) myInt = myChar;  
e) myFloat = myDouble;  
f) myFloat = 3;  
g) myDouble = 3.0;

**15. Consider this class example:**

class MyPoint

{ void myMethod()

{ int x, y;

x = 5; y = 3;

System.out.print( " ( " + x + ", " + y + " ) " );

switchCoords( x, y );

System.out.print( " ( " + x + ", " + y + " ) " );

}

void switchCoords( int x, int y )

{ int temp;

temp = x;

x = y;

y = temp;

System.out.print( " ( " + x + ", " + y + " ) " );

}

}

**What is printed to standard output if myMethod() is executed?**  
a) (5, 3) (5, 3) (5, 3)  
b) (5, 3) (3, 5) (3, 5)  
c) (5, 3) (3, 5) (5, 3)

**16. To declare an array of 31 floating point numbers representing snowfall for each day of March in Gnome, Alaska, which declarations would be valid?**  
a) double snow[] = new double[31];  
b) double snow[31] = new array[31];  
c) double snow[31] = new array;  
d) double[] snow = new double[31];

**17. If arr[] contains only positive integer values, what does this function do?**

public int guessWhat( int arr[] )

{ int x= 0;

for( int i = 0; i < arr.length; i++ )

x = x < arr[i] ? arr[i] : x;

return x;

}

a) Returns the index of the highest element in the array  
b) Returns true/false if there are any elements that repeat in the array  
c) Returns how many even numbers are in the array  
d) Returns the highest element in the array  
e) Returns the number of question marks in the array

**18. Consider the code below:**

arr[0] = new int[4];  
arr[1] = new int[3];  
arr[2] = new int[2];  
arr[3] = new int[1];  
for( int n = 0; n < 4; n++ )  
System.out.println( /\* what goes here? \*/ );

**Which statement below, when inserted as the body of the for loop, would print the number of values in each row?**  
a) arr[n].length();  
b) arr.size;  
c) arr.size-1;  
d) arr[n][size];  
e) arr[n].length;

**19. If size = 4, triArray looks like:**

int[][] makeArray( int size)

{ int[][] triArray = new int[size] [];

int val=1;

for( int i = 0; i < triArray.length; i++ )

{ triArray[i] = new int[i+1];

for( int j=0; j < triArray[i].length; j++ )

{ triArray[i][j] = val++;

}

}

return triArray;

}

a)   
1 2 3 4  
5 6 7  
8 9  
10

b)   
1 4 9 16

c)   
1 2 3 4

d)   
1 2 3 4  
5 6 7 8  
9 10 11 12  
13 14 15 16

e)   
1  
2 3   
4 5 6   
7 8 9 10

**20. Which of the following are legal declarations of a two-dimensional array of integers?**  
a) int[5][5]a = new int[][];  
b) int a = new int[5,5];   
c) int[]a[] = new int[5][5];  
d) int[][]a = new[5]int[5];

**21. Which of the following are correct methods for initializing the array "dayhigh" with 7 values?**  
a) int dayhigh = { 24, 23, 24, 25, 25, 23, 21 };  
b) int dayhigh[] = { 24, 23, 24, 25, 25, 23, 21 };  
c) int[] dayhigh = { 24, 23, 24, 25, 25, 23, 21 };  
d) int dayhigh [] = new int[24, 23, 24, 25, 25, 23, 21];  
e) int dayhigh = new[24, 23, 24, 25, 25, 23, 21];

**22. If you want subclasses to access, but not to override a superclass member method, what keyword should precede the name of the superclass method?**  
  
**23. If you want a member variable to not be accessible outside the current class at all, what keyword should precede the name of the variable when declaring it?**

**24. Consider the code below:**

public static void main( String args[] )

{ int a = 5;

System.out.println( cube( a ) );

}

int cube( int theNum )

{

return theNum \* theNum \* theNum;

}

**What will happen when you attempt to compile and run this code?**  
a) It will not compile because cube is already defined in the java.lang.Math class.  
b) It will not compile because cube is not static.  
c) It will compile, but throw an arithmetic exception.  
d) It will run perfectly and print "125" to standard output.

**25. Given the variables defined below:**

int one = 1;  
int two = 2;  
char initial = ‘2′;  
boolean flag = true;

**Which of the following are valid?**  
a) if(one ){}  
b) if(one = two ){}  
c) if(one == two ){}  
d) if(flag ){}  
e) switch(one ){}  
f) switch(flag ){}  
g) switch(initial ){}

**26. If val = 1 in the code below:**

switch( val )

{ case 1: System.out.print( "P" );

case 2:

case 3: System.out.print( "Q" );

break;

case 4: System.out.print( "R" );

default: System.out.print( "S" );

}

**Which values would be printed?**  
a) P  
b) Q  
c) R  
d) S

**27. Assume that val has been defined as an int for the code below:**

if( val > 4 )

{ System.out.println( "Test A" );

}

else if( val > 9 )

{ System.out.println( "Test B" );

}

else System.out.println( "Test C" );

**Which values of val will result in "Test C" being printed:**

a) val < 0   
b) val between 0 and 4   
c) val between 4 and 9   
d) val > 9   
e) val = 0   
f) no values for val will be satisfactory

**28. What exception might a wait() method throw?**  
  
**29. For the code:**

m = 0;

while( m++ < 2 )

System.out.println( m );

**Which of the following are printed to standard output?**  
a) 0  
b) 1  
c) 2  
d) 3  
e) Nothing and an exception is thrown

**30. Consider the code fragment below:**

 outer: for( int i = 1; i <3; i++ )

{ inner: for( j = 1; j < 3; j++ )

{ if( j==2 )

continue outer;

System.out.println( "i = " +i ", j = " + j );

}

}

**Which of the following would be printed to standard output?**  
a) i = 1, j = 1  
b) i = 1, j = 2  
c) i = 1, j = 3  
d) i = 2, j = 1  
e) i = 2, j = 2  
f) i = 2, j = 3  
g) i = 3, j = 1  
h) i = 3, j = 2

**31. Consider the code below:**

void myMethod()

{ try

{

fragile();

}

catch( NullPointerException npex )

{

System.out.println( "NullPointerException thrown " );

}

catch( Exception ex )

{

System.out.println( "Exception thrown " );

}

finally

{

System.out.println( "Done with exceptions " );

}

System.out.println( "myMethod is done" );

}

**What is printed to standard output if fragile() throws an IllegalArgumentException?**  
a) "NullPointerException thrown"  
b) "Exception thrown"  
c) "Done with exceptions"  
d) "myMethod is done"  
e) Nothing is printed

**32. Consider the following code sample:**

class Tree{}

class Pine extends Tree{}

class Oak extends Tree{}

public class Forest

{ public static void main( String[] args )

{ Tree tree = new Pine();

if( tree instanceof Pine )

System.out.println( "Pine" );

if( tree instanceof Tree )

System.out.println( "Tree" );

if( tree instanceof Oak )

System.out.println( "Oak" );

else System.out.println( "Oops" );

}

}

**Select all choices that will be printed:**  
a) Pine  
b) Tree  
c) Forest  
d) Oops  
e) (nothing printed)

**33. Consider the classes defined below:**

import java.io.\*;

class Super

{

int methodOne( int a, long b ) throws IOException

{ // code that performs some calculations

}

float methodTwo( char a, int b )

{ // code that performs other calculations

}

}

public class Sub extends Super

{

}

**Which of the following are legal method declarations to add to the class Sub? Assume that each method is the only one being added.**

a) public static void main( String args[] ){}  
b) float methodTwo(){}  
c) long methodOne( int c, long d ){}  
d) int methodOne( int c, long d ) throws ArithmeticException{}  
e) int methodOne( int c, long d ) throws FileNotFoundException{}

**34. Assume that Sub1 and Sub2 are both subclasses of class Super.  
Given the declarations:**  
Super super = new Super();  
Sub1 sub1 = new Sub1();  
Sub2 sub2 = new Sub2();   
**Which statement best describes the result of attempting to compile and execute the following statement:  
super = sub1;**  
a) Compiles and definitely legal at runtime  
b) Does not compile  
c) Compiles and may be illegal at runtime

**35. For the following code:**

class Super

{ int index = 5;

public void printVal()

{ System.out.println( "Super" );

}

}

class Sub extends Super

{ int index = 2;

public void printVal()

{ System.out.println( "Sub" );

}

}

public class Runner

{ public static void main( String argv[] )

{ Super sup = new Sub();

System.out.print( sup.index + "," );

sup.printVal();

}

}

**What will be printed to standard output?**  
a) The code will not compile.  
b) The code compiles and "5, Super" is printed to standard output.  
c) The code compiles and "5, Sub" is printed to standard output.  
d) The code compiles and "2, Super" is printed to standard output.  
e) The code compiles and "2, Sub" is printed to standard output.  
f) The code compiles, but throws an exception.

**36. How many objects are eligible for garbage collection once execution has reached the line labeled Line A?**  
String name;  
String newName = "Nick";  
newName = "Jason";  
name = "Frieda";  
String newestName = name;  
name = null;  
//Line A  
a) 0  
b) 1  
c) 2  
d) 3  
e) 4

**37. Which of the following statements about Java’s garbage collection are true?**  
a) The garbage collector can be invoked explicitly using a Runtime object.  
b) The finalize method is always called before an object is garbage collected.  
c) Any class that includes a finalize method should invoke its superclass’ finalize method.  
d) Garbage collection behavior is very predictable.

**38. What line of code would begin execution of a thread named myThread?**  
  
**39. Which methods are required to implement the interface Runnable.**  
a) wait()  
b) run()  
c) stop()  
d) update()  
e) resume()

**40. What class defines the wait() method?**  
  
**41. For what reasons might a thread stop execution?**  
a) A thread with higher priority began execution.  
b) The thread’s wait() method was invoked.  
c) The thread invoked its yield() method.  
d) The thread’s pause() method was invoked.  
e) The thread’s sleep() method was invoked.

**42. Which method below can change a String object, s ?**  
a) equals( s )  
b) substring(s)  
c) concat(s)  
d) toUpperCase(s)  
e) none of the above will change s

**43. If s1 is declared as:**   
String s1 = "phenobarbital";  
What will be the value of s2 after the following line of code:

String s2 = s1.substring( 3, 5 );

a) null  
b) "eno"  
c) "enoba"  
d) "no"

**44. What method(s) from the java.lang.Math class might method() be if the statement**

method( -4.4 ) == -4;

**is true.**

a) round()  
b) min()  
c) trunc()  
d) abs()  
e) floor()  
f) ceil()

**45. Which methods does java.lang.Math include for trigonometric computations?**

a) sin()  
b) cos()  
c) tan()  
d) aSin()  
e) aCos()  
f) aTan()  
g) toDegree()

**46. This piece of code:**

TextArea ta = new TextArea( 10, 3 );

Produces (select all correct statements):

a) a TextArea with 10 rows and up to 3 columns  
b) a TextArea with a variable number of columns not less than 10 and 3 rows  
c) a TextArea that may not contain more than 30 characters  
d) a TextArea that can be edited

**47. In the list below, which subclass(es) of Component cannot be directly instantiated:**

a) Panel  
b) Dialog  
c) Container  
d) Frame

**48. Of the five Component methods listed below, only one is also a method of the class MenuItem. Which one?**

a) setVisible(boolean b )  
b) setEnabled( boolean b )  
c) getSize()  
d) setForeground( Color c )  
e) setBackground( Color c )

**49. If a font with variable width is used to construct the string text for a column, the  
initial size of the column is:**

a) determined by the number of characters in the string, multiplied by the width of a character in this font  
b) determined by the number of characters in the string, multiplied by the average width of a character in this font  
c) exclusively determined by the number of characters in the string  
d) undetermined

**50. Which of the following methods from the java.awt.Graphics class would be used to draw the outline of a rectangle with a single method call?**

a) fillRect()  
b) drawRect()  
c) fillPolygon()  
d) drawPolygon()  
e) drawLine()  
  
**51. The Container methods add( Component comp ) and add( String name, Component comp) will throw an IllegalArgumentException if comp is a:**  
  
a) button  
b) list  
c) window  
d) textarea  
e) container that contains this container

**52. Of the following AWT classes, which one(s) are responsible for implementing the components layout?**  
  
a) LayoutManager  
b) GridBagLayout  
c) ActionListener  
d) WindowAdapter  
e) FlowLayout

**53. A component that should resize vertically but not horizontally should be placed in a:**

a) BorderLayout in the North or South location  
b) FlowLayout as the first component  
c) BorderLayout in the East or West location  
d) BorderLayout in the Center location  
e) GridLayout

**54. What type of object is the parameter for all methods of the MouseListener interface?**

**55. Which of the following statements about event handling in JDK 1.1 and  
later are true?**

a) A class can implement multiple listener interfaces  
b) If a class implements a listener  
interface, it only has to overload the methods it uses  
c) All of the MouseMotionAdapter class  
methods have a void return type.

**56. Which of the following describe the sequence of method calls that result in a component  
being redrawn?**

a) invoke paint() directly  
b) invoke update which calls paint()  
c) invoke repaint() which invokes update(), which in turn invokes paint()  
d) invoke repaint() which invokes paint directly

**57. Which of these is a correct fragment within the web-app element of deployment descriptor. Select the two correct answers.**

* 1. <error-page>  
     <error-code>404</error-code>  
     <location>/error.jsp</location> </error-page>
  2. <error-page>  
     <exception-type>mypackage.MyException</exception-type>  
     <error-code>404</error-code>  
     <location>/error.jsp</location> </error-page>
  3. <error-page> <exception-type>mypackage.MyException</exception-type>  
     <error-code>404</error-code> </error-page>
  4. <error-page>  
     <exception-type>mypackage.MyException</exception-type>  
     <location>/error.jsp</location> </error-page>

**58. A bean with a property color is loaded using the following statement**  
<jsp:usebean id="fruit" class="Fruit"/>   
**Which of the following statements may be used to set the of color property of the bean. Select the one correct answer.**

1. <jsp:setColor id="fruit" property="color" value="white"/>

2. <jsp:setColor name="fruit" property="color" value="white"/>

3.    <jsp:setValue name="fruit" property="color" value="white"/>

4.    <jsp:setProperty name="fruit" property="color" value="white">

5.    <jsp:setProperty name="fruit" property="color" value="white"/>

6.     <jsp:setProperty id="fruit" property="color" value="white">

**59. What gets printed when the following JSP code is invoked in a browser. Select the one correct answer.**  
<%=  
if(Math.random() < 0.5){ %>  
  hello  
<%=  
} else { %>  
  hi  
<%=  
} %>

a.      The browser will print either hello or hi based upon the return value of random.

b.     The string hello will always get printed.

c.      The string hi will always get printed.

d.     The JSP file will not compile.

**60. Given the following web application deployment descriptor:**

<servlet-class>MyServlet</servlet-class>

...

</servlet>

<servlet-mapping>

<servlet-name>myServlet</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

**Which statements are true?**

1) servlet-mapping element should be inside servlet element

2) url-pattern can’t be defined that way.

3) if you make the http call: href="http://host/servlet/Hello.do">http://host/Hello.jsp the servlet container will execute MyServlet.

4) It would work with any extension excepting jsp,html,htm

**61.Name the class that includes the getSession method that is used to get the HttpSession object.**

* 1. HttpServletRequest
  2. HttpServletResponse
  3. SessionContext
  4. SessionConfig

**62. What will be the result of running the following jsp file taking into account that the Web server has just been started and this is the first page loaded by the server?**

<html><body>

<%=request.getSession(false).getId() %>

</body></html>

1)It won’t compile

2)It will print the session id.

3)It will produce a NullPointerException as the getSession(false) method’s call returns null, cause no session had been created.

4)It will produce an empty page.

**63. The page directive is used to convey information about the page to JSP container.  
Which of these are legal syntax of page directive. Select the two correct statements.**

* 1. <% page info="test page" %>
  2. <%@ page info="test page" session="false"%>
  3. <%@ page session="true" %>
  4. <%@ page isErrorPage="errorPage.jsp" %>
  5. <%@ page isThreadSafe=true %>

**64. Which of the following are methods of the Cookie Class?**

1) setComment

2) getVersion

3) setMaxAge

4) getSecure