1. **What is advantage of using Java language?**

Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.

Java is object-oriented. This allows you to create modular programs and reusable code. Java is platform-independent. One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Java does not used pointers there by makes java secure. i.e if we use pointers we can get the exact location of the data and can modify the data and in java that is not possible.

1. **JVM, JDK, JRE**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

Loads code

Verifies code

Executes code

Provides runtime environment

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

1. **Java versions and featured (latest java 8)**

|  |  |  |  |
| --- | --- | --- | --- |
| Java 5 (Tiger) | Java 6 (Mustang) | Java 7 (dolphin) | Java 8 (spider) |
| Generics are used, hence provide compile time type safety for collections and there by eliminates the need of typecasting | Improved web service support | Use of string in switch statements  switch (dayOfWeekArg) {  case "Monday": typeOfDay = "Start of work week";  break;  case "Tuesday":  case "Wednesday": | Default and static methods can be added to the existing interface without breaking the old code and can implement the default and static methods in interface |
| Autoboxing/unboxing: converting primitive to their respective wrapper classes (int to INTEGER) and viceverca | GUI improvements such as integration of swing worker in api | Allowing \_ in numeric literals  long creditCardNumber = 1234\_5678\_9012\_3456L;  long socialSecurityNumber = 999\_99\_9999L;  You can place underscores only between digits; you cannot place underscores in the following places:   * At the beginning or end of a number * Adjacent to a decimal point in a floating point literal * Prior to an F or L suffix * In positions where a string of digits is expected | Use of lambda expressions |
| Varargs, Variable arguments, in a method we can pass the last parameter with …(3 dots), which indicates that we can pass 0-n number of same type data to method,  Ex add(string name, int … value) here we can pass n number of int values  Vararg should be only to a method and that should be the last parameter of method | Scripting language support | Catching multiple exception types , in a single catch we can have multiple exceptions  Catch (exception1 | exception2 ex) |  |
| Enhanced for each loop | Support for pluggable annotations | Try with resource statement  Like try(any sql connection or file opening etc) |  |
| Enumerations |  |  |  |
| Static imports, access to any of  members (fields and methods) defined in a class as public static to be used in Java code; without specifying the class in which the field is defined  Import static package; |  |  |  |
| Covariant returntype :  Before Java 5.0, when you override a method, both parameters and return type must match exactly. In Java 5.0, it introduces a new facility called covariant return type. You can override a method with the same signature but returns a subclass of the object returned |  |  |  |

1. **OOP features**

Inheritance

Polymorphism

Abstraction

Encapsulation

1. **Abstraction**

It is a mechanism in which only the essential features of an object is shown to the uses and hides the internal implementation. Ex : working of email, here user is given with only the functionality and user not worried about the internal functioning.

abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface)

Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user

1. **Encapsulation**

Encapsulation is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse.

We declare data member as private and getters and setters for the data member to access that data member

If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. ... That's **why encapsulation** is known as data hiding

1. **Inheritance**

It is a mechanism in which one class acquire the properties and behavior of another class, so that both classes can have specific behavior

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also

Class which acquire properties and behavior of another class is child(Sub) and the class which provide the properties and behavior is parent (Super)class

A extends B

A – child class, B parent class

1. **Polymorphism**

It a mechanism in which same operation/task can be performed in different ways.

2 types of polymorphism

1. Static or Compile time polymorphism (Method overloading)

In a class if we have multiple methods with same name but with different no of parameters/ diff in datatypes. During compile time, the jvm will know which method should be invoked, hence is is called as compile time or static polymorphism

Add(int,int)

Add(int,int,int)

Add(float,float)

In java, Method Overloading is not possible by changing the return type of the method only

1. Dynamic or Run time polymorphism (Method overriding)

If parent class and child class have same method name with same no of parameters

So that the child and parent can have different implementation.

Here jvm will not have any idea which method to be invoke during comple time. So at run time the respective method will be invoked based in the object created , so it is called as runtime or dynamic poly

**class** Vehicle{

**void** run(){System.out.println("Vehicle is running");}

}

**class** Bike2 **extends** Vehicle{

**void** run(){System.out.println("Bike is running safely");}

**public** **static** **void** main(String args[]){

Bike2 obj = **new** Bike2();

obj.run();

}

\*\* runtime polymorphism cannot be achieved on data members

class Vehicle{

int value =100;

void run(){System.out.println("Vehicle is running");}

}

class Bike2 extends Vehicle{

int value =150;

void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){

Vehicle obj = new Bike2();

Bike2 obj1 = new Bike2();

Syso(obj.value)// op 100, Syso(obj1.value)// op 150

}

Can we override static method?

No, static method cannot be overridden. It can be proved by runtime polymorphism

Why we cannot override static method?

because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Can we override java main method?

Yes we can override but the parent class mail method will be hidden, and it is called as method hiding not overriding

<https://www.youtube.com/watch?v=L0SdH2Dtrnw> .

**\*\*\*\***

It may differ in the return types of the method overriding, but their are some limitations. Before **Java** 5.0, when you **override**a method, both parameters and **return type** must match exactly. In **Java** 5.0, it introduces a new facility called covariant **return type**. You can **override** a method with the same signature but **returns** a subclass of the object returned.

1. **Object & Class**

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical. Object is an instance of a class.

Class is a Collection of objects. It is a logical entity. Class is a template or blueprint from which objects are created. A class is a group of objects which have common properties

1. **Variables**

It is a location in memory where the data can be stored

Local variables: Created inside a method and scope is inside the method

Global/Class/Instance variables, created inside the class and scope is throughout the class. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.

Static variables : These deals with memory management, If we have a common property of all the objects, then instead of creating the same property all times, we can create the variable once using static and can be used for all objects

Ex: company name of employees

static String name = “abc”;

the static variables are stored in a class area

|  |  |
| --- | --- |
|  | **Stack** and **heap** are the memories allocated by the OS to the JVM that runs in the system.**Stack** is a memory place where the methods and the local variables are stored. (variable references either **primitive** or **object** references are also stored in the stack). **Heap** is a memory place where the objects and its instance variable are stored.  So to sum it up:   * Class objects, including method code and static fields: heap. * Objects, including instance fields: heap. * Local variables and calls to methods, references : stack * **static fields** are stored in heap. On the other hand, **local variables** are stored in stack. * Only method invocation and partial results will be stored in stack not the method itself.   \*\* local variables does not have default values when undefined and class variables have default values when undefined.  **Virtual method**: In object-oriented programming, a virtual function or virtual method is a function or method whose behaviour can be overridden within an inheriting class by a function with the same signature to provide the polymorphic behavior.   1. **Constructor**   Constructor is a special method in java which has same name as of class and no explicit return type and used to initialize the object.  Constructor is invoked at time of object creation  By default, we will have default constructor in java class even if we not specify constructor in class.  Default constructor set the default values to the object in class   * Constructor will return the current class instance  1. **This key word**   This is a reference variable used to refer the current clas  Used to refer the current class variables or to call the current class method or current class constructor  This.variable for variable  This.method() for method  This() --constructor  Call to this() must be the first statement in constructor. And this used to return current class instance   1. **Super**   Super is used to refer the immediate parent class  Using super we can access the parent class variables or parent class methods or parent class constructor  super() is added in each class constructor automatically by compiler if there is no super() or this()  super should be the 1st statement same like this   1. **Inheritance (IS-A)**   Mechanism thru which one object acquires the properties of other  Types   1. Single inheritance   Class A extends B   1. Multilevel   Class A extends B  Class C extends A   1. Hierarchical   Class B  Class A extends B  Class C extends B  Java does not support multiple inheritance. Because, |

Suppose we have a method add in class A & B, if Class C is extending A,B (multiple) and calling Add method in C, then JVM will not understand which method to invoke (from a or b), in order to overcome this we use inheritance

Imp:

Class A

Class B extends A

Main class

B b = new B()

b.anymethods (Invokes B methods and if method is not available in B looks for method in A)

A a = New A();

a.methods (Can invokes only A methods and cannot invoke B methods)

A a=new B();

a.methods—can only parent methods

B b = new A();

b.methods – compile time error

1. **Aggregation (HAS-A relatioship)**

If one class has a relation with other class, then it is aggregation, like dependency and tight coupling

Employee has a relation with address

1. **Covariant return type**

The covariant return type specifies that the return type may vary in the same direction as the subclass.

it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

**class** A{

A get(){**return** **this**;}

}

**class** B1 **extends** A{

B1 get(){**return** **this**;}

**void** message(){System.out.println("welcome to covariant return type");}

**public** **static** **void** main(String args[]){

**new** B1().get().message(); // out:  welcome to covariant return type

}

}

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B class is B. Both methods have different return type but it is method overriding. This is known as covariant return type.

1. **Instance Initializer Block (IIB)**

Instance Initialization Blocks or IIB are used to initialize instance variables. IIBs are executed before constructors. They run each time when object of the class is created.

These are used if we want to set some of the values even before creating the objects.

Initialization blocks are executed whenever the class is initialized and before constructors are invoked.

They are typically placed above the constructors within braces.

It is not at all necessary to include them in your classes.

Instance Initialization Blocks run every time a new instance is created.

Initialization Blocks run in the order they appear in the program

1. **Final keyword**

Final is mainly used to restrict.

If we have any value that should be changed further the we can declare is as final

Ex pan card number of person

Final variables

Variables which cannot be changed further

Final methods:

The method whose implementation is final and we cannot override the final methods

Final class;

Final class cannot be extended

Final means we cannot override and by default we cannot override constructor so it is not needed to final constructor. Constructor cannot be declared as final because it is never inhereted(if we want to create singleton then we can).

1. **Binding**

Connection of Method call to method body

**Static binding (Early binding):**

Binding is done during compile time itself, jvm will come to know which method/variable should to bind at compile time. As the binding is happening at compile time, even before runtime so called as early binding

**Dynamic binding (Late binding)**

Here binding will happen at run time as the jvm will not know which method/ variable should bind, hence at run time the respective one will be invoked. Called as Late binding as at run time the binding is taking place.

1. **Abstract**

Just declaration and no implementation

Abstract method: An abstract method which is just declared but not implemented

abstract void run();

If a class has abstract method then the class must be declared as abstract

An abstract class can have abstract methods and also non abstract methods

It needs to be extended and its method implemented. It cannot be instantiated.

if you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

Abstract class can have constructors and these can be called from the child class

We can use any access modifier for methods in abstract except private because private methods cannot be overridden in child classes and the normal methods in abstract class can have private but it does not make any sense because we are implementing there and also if needed we need to invoke the same from other methods of same abstract.

1. **Interface**

Interface is used to achieve complete abstraction in java and interface is blue print of static final constants and abstract methods

Helps in loose coupling bw classes

Can achieve multiple inheritance using interface

By default, the variables are public static final and methods are abstract

If only some of the methods of interface are implemented in one class, then the implemented class should be declared as abstract.

**Marker or Tagged Interface**

An interface that have no member is known as marker or tagged interface. For example: Serializable, Clone able, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

public interface Serializable {

}

**Abstract vs Interface**

|  |  |
| --- | --- |
| Abstract | Interface |
| Can have abstract and non-abstract methods | By default all methods are abstract |
| Can have public, private static non-static variables | By default all variable are public static and final |
| Can provide implementation for some methods or for interface | Cannot provide implementation |
| 0-100 abstraction | 100% abstraction |

1. **Access modifiers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

Constructor cannot be private because we cannot create object from outside of class if it is private

1. **Wrapper class**

It is the mechanism to convert the primitive to object class and vice verca

Int to INTEGER

boolean Boolean

char Character

byte Byte

short Short

int Integer

long Long

float Float

double Double

int a=20;

Integer i=Integer.valueOf(a);//converting int into Integer

Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

Integer a=new Integer(3);

int i=a.intValue();//converting Integer to int

int j=a;//unboxing, now compiler will write a.intValue() internally

1. **Strictfp**

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

The strictfp keyword can be applied on methods, classes and interfaces.

The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

1. **Static**

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static method cannot use non static data member or call non-static method directly.

this and super cannot be used in static context.

1. **Strings**

sequence of characters. In Java programming language, strings are treated as objects.

The Java platform provides the String class to create and manipulate strings.

String literals or by new key word

String s =”name”;

String s1= new String(“abc”)

If we try to create same string using string literal, then the jvm initially search the string pool to check whether the string is already created or not, if created then the newly created reference points to old string, if not created earlier then creates the new string

String s1= “india”;

String s2=”india”;

Here s1 & s2 points to same string, it will not create new string as india is already available

S1==s2 true compares the references

S1.equals(s2) true

String s3= new String(“India”);

Now new string is created and new reference s3 points to New value

S1==s3 // false compares the references

S1.equals(s3)// true compares the content

Strings are immutable

An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper classes objects are immutable.

String str1 = “abc”;

Str1 = “xyz”;

Syso(str1)// xyx as the reference is changed

Str1.concat(“ind”);

Syso(str1)// xyz as strings are immutable

String str2 =str1.concat(“ind”);

Syso(str2)// xyzind here new string is created

**Concat vs Append**

Concat will create new string object and append will not create new object , it will append the value to old string

**String buffer and string builder**

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

1. **How to create Immutable class?**

The instance variable of the class is final i.e. we cannot change the value of it after creating an object.

The class is final so we cannot create the subclass.

There is no setter methods i.e. we have no option to change the value of the instance variable.

1. **Exception Handing**

Checked and unchecked exceptions

Checked are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

Ex: file not found, io exceptions, sql exceptions

Unchecked are the exceptions that are not checked at compiled time and these arises at runtime

In C++, all exceptions are unchecked

**Try** block

Used to place the code that might result in exception

And try must immediately followed by either catch or finally

**Catch** block is used to catch any exception occurred In try block and at a time only one exception can occur and all the catch blocks should be in specific to general order

i.e Exception (general) cannot be the 1st and all other cannot be next

all other child exception 1st then the general exception

**Finally** is used to perform some actions even exception arises or not, ex closing of file or sql connection

Finally does not execute is system stops or exits suddenly , system.exit();

**Throw** is used to throw the exception explicitly, mainly used to throw custom exceptions

We can throw either checked or uncheked exception in java by throw keyword

Exception propagation : An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.This is called exception propagation.

By default Unchecked Exceptions are forwarded in calling chain (propagated).

By default, Checked Exceptions are not forwarded in calling chain (propagated). It will result in compile time error

**Throws**: The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Which exception should be declared

Ans) checked exception only, because:

unchecked Exception: under your control so correct your code.

error: beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

There are many rules if we talk about methodoverriding with exception handling. The Rules are as follows:

* **If the superclass method does not declare an exception**
  + If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
* **If the superclass method declares an exception**
  + If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

1. **Multithreading**

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Thread can be created by extending thread class or by implementing runnable interface

Implementing runnable is good because java does not support multiple inheritance, if we use implements runnable we will have a chance to extend some other class and suppose if we use extends thread, then we cannot extend any other class further.

At a time only one thread can be executed

Life cycle

1) **New**

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) **Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) **Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

5) **Terminated**

A thread is in terminated or dead state when its run() method exits.

Run() method used to perform some action for thread

Runnable interface has only 1 method that is run()

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

**start() method** of Thread class is used to start a newly created thread

Out of three methods which can be used to pause a thread in Java, sleep() and yield() methods are defined in thread class while wait() is defined in the Object class

sleep () : when a thread calls the sleep() method, it never releases the monitor even if it is holding. And it will pause the execution for specified period of time

wait(): Inter thread communication, thread calls the wait() method, it releases the monitor or lock it was holding on that object

Yield(): it just releases the CPU hold by Thread to give another thread an opportunity to run though it's not guaranteed who will get the CPU. It totally depends upon thread scheduler and it's even possible that the thread which calls the yield() method gets the CPU again. Hence, it's not reliable to depend upon yield() method, it's just on best effort basis.

Notify() : It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource. It tells a waiting thread that that thread can wake up. However, the lock is not actually given up until the notifier’s synchronized block has completed. So, if a notifier calls notify() on a resource but the notifier still needs to perform 10 seconds of actions on the resource within its synchronized block, the thread that had been waiting will need to wait at least another additional 10 seconds for the notifier to release the lock on the object, even though notify() had been called

Notifyall() ; It wakes up all the threads that called wait() on the same object. The highest priority thread will run first in most of the situation, though not guaranteed. Other things are same as notify() method above

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

We cannot start the thread twice, 1st started will execute and when it comes to 2nd one it throws exception

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically. There are many java daemon threads running automatically e.g. gc, finalizer etc.

**Synchronization** in java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

1. **Synchronized block vs synchronized method.**

One significant difference between synchronized method and block is that, Synchronized block generally reduce scope of lock. As scope of lock is inversely proportional to performance, its always better to lock only critical section of code. One of the best example of using synchronized block is [double checked locking in Singleton pattern](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html) where instead of locking whole getInstance() method we only lock critical section of code which is used to create Singleton instance. This improves performance drastically because locking is only required one or two times.

Synchronized block provide granular control over lock, as you can use arbitrary any lock to provide mutual exclusion to critical section code. On the other hand synchronized method always lock either on current object represented by [this keyword](http://javarevisited.blogspot.com/2012/01/this-keyword-java-example-tutorial.html)  or class level lock, if its static synchronized method.

Synchronized block can throw throw [java.lang.NullPointerException](http://java67.blogspot.sg/2012/09/what-is-nullpointerexception-in-java.html) if expression provided to block as parameter evaluates to null, which is not the case with synchronized methods.

In case of synchronized method, lock is acquired by thread when it enter method and released when it leaves method, either normally or by throwing Exception. On the other hand in case of synchronized block, thread acquires lock when they enter synchronized block and release when they leave synchronized block

1. **Can we synchronize run method?**

If you synchronize the run method, there are two outcomes:

1) Synchronize does not help, if the threads are created using different runnable instances.

2) There is no multi-threading, if all threads are created using the same runnable. Its as good as directly calling the run method from the same thread of execution

1. **Static synchronization**

If you make any static method as synchronized, the lock will be on the class not on object.

If we have 2 classes and created two objects and each is trying to access the same resources then the problem of deadlock arises , so inorder to deal in such cases we use static synchronization, so that the lock will be on class instead of on object

1. **Class level and object level lock**

What is class level lock vs object level lock?  (synchronized on static method will apply class level lock, synchronized on non-static method will apply object level lock)

class Program {  
 public **synchronized** void f() {  
 }  
}

class Program {  
 public void f() {  
 **synchronized(this){  
 ...  
 }** }}

1. **Deadlock**

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

To avoid this case, inter thread communication is used, like mutual exclusion

If one thread is using the shared resource the other has to wait and after completion invoke notify so that the waiting thread will be notified and performs its action

1. **Race condition**

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data

1. **Java Shutdown Hook**

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

When does the JVM shut down?

The JVM shuts down when:

user presses ctrl+c on the command prompt

System.exit(int) method is invoked

user logoff

user shutdown etc.

The addShutdownHook(Thread hook) method

The addShutdownHook() method of Runtime class is used to register the thread with the Virtual Machine. Syntax:

public void addShutdownHook(Thread hook){}

The object of Runtime class can be obtained by calling the static factory method getRuntime(). For example:

Runtime r = Runtime.getRuntime();

1. **Serialization**

Must implement serializable interface

Serialization in java is a mechanism of writing the state of an object into a byte stream.

And this is used to exchange the state of object over network and also called as Marshelling

ObjectOutputStream out=**new** ObjectOutputStream(f);

out.writeObject(s1); // serialize

De-serialization: vice versa of derialization: process of reconstructing the object from the serialized state

Student s=(Student)in.readObject();// de-serialize

If we don’t want to save the state of any variable then we declare is as Transient

In de-serialized case the value will be the default for transient.

1. **Volatile:**

Is used for a variables in multithreading env, if a variable is shared among many resources and each thread is trying to modify the value, inorder to have the updated value we use the variable as volatile

1. **Single ton**

We can make constructor as private. So that We can not create an object outside of the class.

This property is useful to create singleton class in java.

Singleton pattern helps us to keep only one instance of a class at any time.

The purpose of singleton is to control object creation by keeping private constructor.

1. **Array List vs Linked List**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

1. **Hashmap vs Hash Table**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

1. **Comparable vs Comparator**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| 2) Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method. |

1. **Array list vs Vector**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity. | Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object. |
| 5) ArrayList uses **Iterator** interface to traverse the elements. | Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also. |

1. **Methods to override**

To store a object in hashmap we have to override gethashcode, equals methods in class

1. **Concurrent hash maps**

If we are iterating on collection and tries to update or delete or add any extra entry into collection it results in concurrent modification exception which is also called as fail fast iterator. This happens because before iteration we will get the iterator and if we try to modify the collection then the collection size will get altered and the iterator associated to the collection also changes, hence results in exception.

List, map etc

To avoid this, we will use concurrent collections which creates the clone of existing collection and if we perform any modification operation it will not affect the iterator, so called as fail safe iterator.

Concurrenthashmap, CopyOnWriteArrayList

1. **List vs Map vs Set**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **list** | **Map** | **set** |
| **Duplicate Objects** | Can allow to have duplicate elements | Holds as key value pair and key are unique and the values can be duplicate | Does not allow duplicate elements |
| **Order** | Order is order of insertion | Random order as based on the hash code | Sorted order and you get no guarantee on which order element will be stored |
| **Null elements** | List allows null elements and you can have many null objects in a List, because it also allowed duplicates | Map you can have null values and at most one null key | Set just allow one null element as there is no duplicate permitted |

1)If you need to access elements frequently by using index, than List is a way to go. Its implementation e.g. [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) provides faster access if you know index.

2) If you want to store elements and want them to maintain an **order** on which they are inserted into collection then go for List again, as List is an ordered collection and maintain insertion order.

3) If you want to create collection of unique elements and **don't want any duplicate** than choose any Set implementation e.g. HashSet, LinkedHashSet or TreeSet. All Set implementation follow there general contract e.g. uniqueness but also add addition feature e.g. TreeSet is a SortedSet and elements stored on TreeSet can be sorted by using [Comparator or Comparable in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html). LinkedHashSet also maintains insertion order.

4) If you store data in form of key and value than Map is the way to go. You can choose from Hashtable, HashMap, TreeMap based upon your subsequent need.

1. **Iterators in collections**

|  |  |  |
| --- | --- | --- |
| **Enumeration** | **Iterator** | **ListIterator** |
| Using Enumeration interface, we can enumerate only legacy classes like Hashtable or Vector or Properties | Iterator interface is applicable for every collection classes like ArrayList, HashSet or Hashtable | ListIterator interface is applicable only for List objects like ArrayList or LinkedList or Vector |
| We can enumerate legacy collection items only in FORWARD direction | Here, too we can iterate through collection items only in FORWARD direction | But with ListIterator, we can iterate through list items either in FORWARD or BACKWARD directions |
| That is, it is unidirectional or single directional cursor | That is, it is unidirectional or single directional cursor | That is, it is bi-directional cursor |
| Using Enumeration interface, we can enumerate to read or get element/object from legacy collection | Using Iterator interface, we can read as well as remove collection items, while iterating | Addition or replacement of new objects is possible alongside read and remove operation in ListIterator interface |
| To get an Enumeration object, we can use elements() method of any legacy collection class  For example,  Vector v = new Vector();  Enumeration e = v.elements(); | To get an Iterator object, we can use iterator() method of any collection class  For example,  Iterator itr = col.iterator();  Where col = any collection class | To get a ListIterator object, we can use listIterator() method of any List classes  For example,  ListIterator ltr = list.listIterator();  Where list = any List objects |
| Enumeration interface has 2 important methods to enumerate through legacy collection objects  boolean hasMoreElements();  Object nextElement(); | Iterator interface has 3 important methods to iterate through any collection objects  boolean hasNext();  Object next();  void remove(); | ListIterator interface has 9 important methods to iterate through any List objects  add(E e)  hasNext()  hasPrevious()  next(), returns E  nextIndex()  previous(), returns E  previousIndex()  remove()  set(E e) |
| Fail safe as no option to modify on iteration | Fail fast as there is remove operation which can cause concurrent modification exception | Fail fast |

1. **Hashing internal functionality**

Hashing is a process of converting an object into integer form by using the method hashCode().

HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time O(1) provided we know the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method. When we call put method, hashcode() method of the key object is called so that hash function of the map can find a bucket location to store value object, which is actually an index of the internal array, known as the table. HashMap internally stores mapping in the form of Map.Entry object which contains both key and value object. When you want to retrieve the object, you call the get() method and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object then it is returned and that's your value object which you have stored earlier. Things get little tricky when collisions occur. It's easy to answer this question if you have read good books on data structure and algorithms like this one. If you know how hash table data structure works then this is a piece of cake.

two equal objects must produce same hash code consistently. But if the hash code is same for two objects, then it does not mean that both objects are same

What will happen if two different objects have the same hashcode

Since hashcode is same, bucket location would be same and collision will occur in HashMap Since HashMap uses LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html)

How will you retrieve Value object  if two Keys will have the same hashcode?

Interviewee will say we will call get() method and then HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html)until we find the value object , then you ask *how do you identify value object because you don't  have value object to compare* ,Until they know that HashMap  stores both Key and Value in LinkedList node or as Map.Entry they won't be able to resolve this issue and will try and fail.

But those bunch of people who remember this key information will say that after finding bucket location, we will **call keys.equals() method** to identify a correct node in LinkedList and return associated value object for that key in Java HashMap. Perfect this is the correct answer.

1. **Constructor chaining**

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

Constructor chaining can be done in two ways:

Within same class: It can be done using this() keyword for constructors in same class

From base class: by using super() keyword to call constructor from the base class.

Constructor chaining occurs through inheritance. A sub class constructor’s task is to call super class’s constructor first. This ensures that creation of sub class’s object starts with the initialization of the data members of the super class. There could be any numbers of classes in inheritance chain. Every constructor calls up the chain till class at the top is reached.

Why do we need constructor chaining ?

This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

1. **Can a method in abstract class be both abstract and static?**

No, abstract and static cannot are illegal combination, because we cannot override the static method and abstract methods has to be overridden.

1. **Can it be both abstract and final?**

No, because we cannot override the final methods and abstract methods has to be overridden in order to have implementation.

1. **Can it be both static and final?**

Yes, normal methods can be static and final in abstract class as we are going to implement the normal method in abstract class itself so it will not cause any issue

1. **What access modifiers are allowed in interface for method and class level variables**

In interface, the variables should be public static final, so the access modifier can be same (public static and final)

If we want to use other access modifiers, then it’s better to use abstract instead of interface

1. **Deep copy vs Shallow copy**

Cloning is a process of creating an exact copy of an existing object in the memory. In java, clone() method of java.lang.Object class is used for cloning process. This method creates an exact copy of an object on which it is called through field-by-field assignment and returns the reference of that object. Not all the objects in java are eligible for cloning process. The objects which implement Cloneable interface are only eligible for cloning process. Cloneable interface is a marker interface which is used to provide the marker to cloning process. Click here to see more info on clone() method in java.

Both shallow copy and deep copy are related to this cloning process. The default version of clone() method creates the shallow copy of an object. To create the deep copy of an object, you have to override the clone() method.

Shallow copies duplicate as little as possible. A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

1. **Types Of References In Java : Strong, Soft, Weak And Phantom**

**Strong**

These type of references we use daily while writing the code. Any object in the memory which has active **strong reference** is not eligible for garbage collection. For example, in the below program, reference variable **‘a’** is a strong reference which is pointing to class A-type object. At this point of time, this object can’t be garbage collected as it has strong reference.

**Soft**

The objects which are softly referenced will not be garbage collected (even though they are available for garbage collection) until JVM badly needs memory. These objects will be cleared from the memory only if JVM runs out of memory. You can create a soft reference to an existing object by using  **java.lang.ref.SoftReference** class. Below is the code example on how to create a soft reference

A a = new A(); //Strong Reference

//Creating Soft Reference to A-type object to which 'a' is also pointing

SoftReference<A> softA = new SoftReference<A>(a);

a = null; //Now, A-type object to which 'a' is pointing earlier is eligible for garbage collection. But, it will be garbage collected only when JVM needs memory.

a = softA.get();

**Weak**

VM ignores the **weak references**. That means objects which has only week references are eligible for garbage collection. They are likely to be garbage collected when JVM runs garbage collector thread. JVM doesn’t show any regard for weak references.

**Phantom**

The objects which are being referenced by phantom references are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called ‘reference queue’ . They are put in a reference queue after calling finalize() method on them. You can’t retrieve back the objects which are being phantom referenced. That means calling get() method on phantom reference always returns null

# **Database questions**

Database: it is a storage of large volume of organized data in order to perform some operations (retrieving, updating, inserting etc,.) on it.

The main purpose of database is to operate large amount of information by storing, retrieving and managing

SQL is a structured query language used to perform operations on the data stored on database(mysql, oracle, mongodb etc)

DBMS vs RDBMS

The key difference is that RDBMS (relational database management system) applications store data in a tabular form, while DBMS applications store data as files.

There can be, but there will be no “relation” between the tables, like in a RDBMS. In DBMS, data is generally stored in either a hierarchical form or a navigational form. This means that a single data unit will have one parent node and zero, one or more children nodes. It may even be stored in a graph form, which can be seen in the network model.

In a RDBMS, the tables will have an identifier called primary key. Data values will be stored in the form of tables. The relationships between these data values will be stored in the form of a table as well. Every value stored in the relational database is accessible. This value can be updated by the system. The data in this system is also physically and logically independent.

Data is represented in terms of tuples (rows) in RDBMS.

Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

Due to a collection of organized set of tables, data can be accessed easily in RDBMS

**Entity integrity**: It specifies that there should be no duplicate rows in a table.

**Domain integrity**: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

**Referential integrity**: It specifies that rows cannot be deleted, which are used by other records.

**User-defined integrity**: It enforces some specific business rules that are defined by users. These rules are different from entity, domain or referential integrity.

Char vs Varchar

Char is used for storing fixed length of strings and varchar for variable length

Char will add blank/spaces at end if the length is remained and var char cant add

Char(5)—can accept string of 5 char and if we have only 3 char then remaining 2 will be blank space

Varchar2(10) can store up to 10 char and if we have only 3 char then remaining 7 will not be blank space

VARCHAR is used to support distinction between NULL and empty string in future, as ANSI standard prescribes.

VARCHAR2 does not distinguish between a NULL and empty string, and never will

Drop: is used to delete the table definition along with all the data present in table

DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.

DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.

DELETE is executed using a row lock, each row in the table is locked for deletion.

We can use where clause with DELETE to filter & delete specific records.

The DELETE command is used to remove rows from a table based on WHERE condition.

It maintain the log, so it slower than TRUNCATE.

The DELETE statement

TRUNCATE is executed using a table lock and whole table is locked for remove all records.

We cannot use Where clause with TRUNCATE.

TRUNCATE removes all rows from a table.

There is a slight difference b/w delete and truncate statement. The DELETE statement only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The TRUNCATE statement: it is used to delete all the rows from the table and free the containing space.

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

Table structure will be dropped

Relationship will be dropped

Integrity constraints will be dropped

Access privileges will also be dropped

On the other hand when we TRUNCATE a table, the table structure remains the same, so you will not face any of the above problems.

Joins: Join is used to join 2 or more tables on specific condition to form single table

If you want to access more than one table through a select statement.

If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.

The joining of two or more tables is based on common field between them.

SQL INNER JOIN also known as simple join is the most common type of join

Inner join or normal join or equijoin



Select \* from a,b where a.id=b.id

Outer joins:  outer JOIN is used to join all the content of the both tables are integrated together either they are matched or not

**1.Left outer join** (also known as left join): this join returns all the rows from left table combine with the matching rows of the right table. If you get no matching in the right table it returns NULL values.



**2.Right outer join** (also known as right join): this join returns all the rows from right table are combined with the matching rows of left table .If you get no column matching in the left table .it returns null value.



**3.Full outer join**

full join is the result of combination of both left and right outer join and the join tables have all the records from both tables. It puts NULL on the place of matches not found.

SQL full outer join and SQL join are same. generally it is known as SQL FULL JOIN



**4. Self-join**

A self JOIN is a regular join, but the table is joined with itself.

**Union vs Union all**: Union selects the distinct values and union all selects the duplicate values also

**Group by**: The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

**Having**: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions (post condition).

**Where** (Precondition)

**Exists**: The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records.

**NOT NULL** - Ensures that a column cannot have a NULL value

**UNIQUE** - Ensures that all values in a column are different

**PRIMARY KEY -** A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

**FOREIGN KEY** - Uniquely identifies a row/record in another table

**CHECK** - Ensures that all values in a column satisfies a specific condition

**DEFAULT** - Sets a default value for a column when no value is specified

**INDEX** - Used to create and retrieve data from the database very quickly

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

**Views**: view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

* **Functions** − These subprograms return a single value; mainly used to compute and return a value.
* **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

**Procedure:** is a sequence of statements which performs some action, we can pass params to procedures

1. **IN parameters:**The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:**The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:**The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

**Syntax:**

create or replace procedure "INSERTUSER"

(id IN NUMBER,

name IN VARCHAR2)

is

begin

insert into user values(id,name);

end;

/

Executing

BEGIN

insertuser(101,'Rahul');

dbms\_output.put\_line('record inserted successfully');

END;

/

Stored procedures are the procedures which are executed and stored for further references.

**Functions**

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

create or replace function adder(n1 in number, n2 in number)

return number

is

n3 number(8);

begin

n3 :=n1+n2;

return n3;

end;

/

**Cursor :**

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

Implicit Cursors

Explicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don?t use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

**CURSOR** cursor\_name **IS** select\_statement;;

**Triggers:**

Triggers are stored programs that are executes when an event occurs

Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

A database definition (DDL) statement (CREATE, ALTER, or DROP).

A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

**JDBC connectivity**

Java Database connectivity, jdbc is used to establish connection to DB from our java code and perform all operations like insert, delete, update etc from java.

Before JDBC, odbc is used to establish connection but is Platform dependent and written in C, java has its own API for JDBC

API: application program interface: a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

**JDBC Driver:** It is a software component that enables the user to interact with the database using java

1. **JDBC-ODBC bridge driver**

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

1. **Native-API driver (partially java driver)**

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

1. **Network Protocol driver (fully java driver)**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

1. **Thin driver (fully java driver)**

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

**Steps to connect to DB from Java**

Register the driver class

Creating connection

Creating statement

Executing queries

Closing connection

**Class.forName("oracle.jdbc.driver.OracleDriver"); //class.forname is used to register the driver class**

**Establish connection:**

Connection is established using getConnection() of the drivermanager class

Connection con=DriverManager.getConnection("host","username","pass");

**Statement:**

The statement is created by using the createStatement() of connection interface and the statement is used for executing the queries

Statement st = con.createStatemet();

**Executing the query**

ExecuteQuery() of statement interface is used to execute the query using the statement obj and This method returns the object of ResultSet that can be used to get all the records of a table.

ResultSet rs=stmt.executeQuery("select \* from emp");

**Closing the connection:**

After executing the query the connection of the db has to be closed. If we close the connection, the statement and result will be closed automatically

Con.close();

Register for driver can be done in 2 ways:

Class.forname(“”); or

DriverManager.registerDriver(DriverObj);

Ex:

Driver myDriver = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver( myDriver );

**Establishing connection:**

After registering for Driver, we need to establish connection to Db to perform operations, this can be done by using getConnection() of DriverManager class

**Types**

getConnection(String url) // url will be host with user & pass

jdbc:oracle:thin:username/password@amrood:1521:EMP

getConnection(String url, Properties prop) // properties will have username and pass

Properties info = new Properties( );

info.put( "user", "username" );

info.put( "password", "password" )

getConnection(String url, String user, String password)

here we will pass the username and password

**Statement:**

After establishing connection, we need to create statement which is used for querying to DB

CreateStatement() of Connection interface is used to create statement.

**Types:**

**Statement:** Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

**PreparedStatement:** Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

This statement gives you the flexibility of supplying arguments dynamically.

String SQL = "Update Employees SET age = ? WHERE id = ?";

pstmt = conn.prepareStatement(SQL);

You must supply values for every parameter before executing the SQL statement.

The **setXXX()** methods bind values to the parameters, where **XXX** represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive an SQLException.

Prepared statement can take only **in** parameter

String sql = "UPDATE Employees set age=? WHERE id=?";

stmt = conn.prepareStatement(sql);

//Bind values into the parameters.

stmt.setInt(1, 35); // This would set age 1 means 1st ?

stmt.setInt(2, 102); // This would set ID

stmt.executeUpdate();

**CallableStatement:**

the CallableStatement object, which would be used to execute a call to a database stored procedure.

The CallableStatement object can use all the three parameters (IN, OUT, INOUT).

String SQL = "{call getEmpName (?, ?)}";

cstmt = conn.prepareCall (SQL);

For IN it acts same as PreparedStatment setXXX()

When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return.

String sql = "{call getEmpName (?, ?)}";

stmt = conn.prepareCall(sql);

//Bind IN parameter first, then bind OUT parameter

int empID = 102;

stmt.setInt(1, empID); // This would set ID as 102

// Because second parameter is OUT so register it

stmt.registerOutParameter(2, java.sql.Types.VARCHAR);

//Use execute method to run stored procedure.

System.out.println("Executing stored procedure..." );

stmt.execute();

**Resultsets:** After executing the statement we will be getting some output which is the result of the query and the result is stored in resultset

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

## **Type of ResultSet**

The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is TYPE\_FORWARD\_ONLY.

|  |  |
| --- | --- |
| **Type** | **Description** |
| ResultSet.TYPE\_FORWARD\_ONLY | The cursor can only move forward in the result set. |
| ResultSet.TYPE\_SCROLL\_INSENSITIVE | The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created. |
| ResultSet.TYPE\_SCROLL\_SENSITIVE. | The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created. |

Rs is a cursor points to current record

rs.next() points to next

rs.getInt(“col name”);

rs.getString(“col name”);

**Batch processing:**

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.

* The **addBatch()** method of *Statement, PreparedStatement,* and *CallableStatement* is used to add individual statements to the batch. The **executeBatch()** is used to start the execution of all the statements grouped together.
* The **executeBatch()** returns an array of integers, and each element of the array represents the update count for the respective update statement.
* Just as you can add statements to a batch for processing, you can remove them with the **clearBatch()** method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.

steps

* Create a Statement object using either *createStatement()* methods.
* Set auto-commit to false using *setAutoCommit()*.
* Add as many as SQL statements you like into batch using *addBatch()* method on created statement object.
* Execute all the SQL statements using *executeBatch()* method on created statement object.
* Finally, commit all the changes using *commit()* method.

**Exmaples see in tutorialpoint:**

**Sample**

Statement stmt = conn.createStatement();

// Set auto-commit to false

conn.setAutoCommit(false);

// Create SQL statement

String SQL = "INSERT INTO Employees (id, first, last, age) " +"VALUES(200,'Zia', 'Ali', 30)";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create one more SQL statement

String SQL = "INSERT INTO Employees (id, first, last, age) " + "VALUES(201,'Raj', 'Kumar', 35)";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create one more SQL statement

String SQL = "UPDATE Employees SET age = 35 " + "WHERE id = 100";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create an int[] to hold returned values

int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes

conn.commit();

**Struts Questions**

**Interceptors**

Struts Interceptors are like Servlet Filters that executes before and after the request is being processed. They are used to perform common operations for different actions. For example logging, session validation, adding common headers to response etc.

**Advantage of interceptors**

Pluggable If we need to remove any concern such as validation, exception handling, logging etc. from the application, we don't need to redeploy the application. We only need to remove the entry from the struts.xml file.

In struts 2, we can create the custom interceptor by implementing the Interceptor interface in a class and overriding its three life cycle method.

For creating the custom interceptor, **Interceptor** interface must be implemented. It has three methods:

1. **public void init()** It is invoked only once and used to initialize the interceptor.
2. **public String intercept(ActionInvocation ai)** It is invoked at each request, it is used to define the request processing logic. If it returns string, result page will be invoked, if it returns invoke() method of ActionInvocation interface, next interceptor or action will be invoked.
3. **public void destroy()** It is invoked only once and used to destroy the interceptor.

we need to follow **2 steps** to create custom interceptor

1. **Create an interceptor (must implement Interceptor interface)**
2. **Define the entry of interceptor in the struts.xml file**

Plug in config

<interceptors>

<interceptor name="upper" class="com.MyInterceptor"></interceptor>

</interceptors>

**ValueStack**

is the storage area where the application data is stored by Struts 2 for processing a client request. The data is stored in ActionContext objects that use **[ThreadLocal](https://www.journaldev.com/1076/java-threadlocal-example)** to have values specific to the particular request thread.

**Object-Graph Navigation Language**

(OGNL) is a powerful Expression Language that is used to manipulate data stored on the ValueStack. both interceptors and result pages can access data stored on ValueStack using OGNL.