**Database constraints:**

Not Null

Foreign Key

Primary key

unique

check

default

**Unique constraint:**

Helps to uniquely identify the rows in the column.

Also checks for sub query returns duplicate tuples or not. Below are details

* Evaluates to true on an empty sub query.
* Returns true only if there are unique tuples present as the output of the sub query (two tuples are unique if the value of any attribute of the two tuples differ).
* Returns true if the sub query has two duplicate rows with at least one attribute as NULL**.**

**Check Constraint:**

Helps to restrict values to the column

Can use **alter table** to create, name modify constraint, remove or drop on column

SQL Server

alter table TABLE\_NAME drop constraint CHECK\_CONSTRAINT\_NAME;

MySql

alter table TABLE\_NAME drop check CHECK\_CONSTRAINT\_NAME;

**Checking Existing Constraints on a Table using Data Dictionaries:**

1. Yes, we can check in SQL server using Data dictionaries like USER\_CONSTRAINTS, USER\_CONS\_COLUMNS
2. PostgreSQL: **pg\_catalog.pg\_constraint –**can query this to get details
3. MySQL: using SHOW command or information\_schema.KEY\_COLUMN\_USAGE

**Nth Highest salary:**

Select \* from (select name, sal, dense\_rank() over(order by sal DESC) r from emp) where r=&n;

Dense rank gives same rank to same values of column. It value starts from 1.

Or

Select \* from emp order by sal desc LIMIT 5, 1; here looking for 6th highest.

Or

Select \* from (Select \* from emp order by sal DESC LIMIT 6) AS t order by sal ASC limit 1;

**case manipulation functions available in SQL: 3**

Upper,Lower,INITCAP-initioal letter cap and other lower

**DDL:** Data definition lang: Create, alter, drop. These defines data.

**DML:** insert, update, retrieve, delete

**Clustered vs Non-Clustered index:**

1. Clustered can only one for table, like PK, PK itself creates clustered index. Non-Clustered can any for one table
2. Clustered index has no separate storage for index, for non-clustered(like text book) extra storage will be there
3. Clustered index is slower than Non\_Clustered index.
4. Note that defining a column as a primary key makes that column the Clustered Index of that table. To make any other column, clustered index first we have to remove the previous one.

//Drop index

drop index table\_name.index\_name

//Create Clustered index index

create Clustered index IX\_table\_name\_column\_name

on table\_name (column\_name ASC)

**ARRAY:**

Size variable should be int.

In Java Array is abject. And dynamically allocated – this is by using new key word, memory for the array is dynamically allocated.

It implements Cleanable and java.io. Serializable

We can choose array in the case of frequent accessing of elements. For frequent insertion and deletion it takes o(n) in worst cases.

Memebers: length, all members from Object except clone(),

Clone of array:

single dimension array: performs deep copy, array1 not equal to array1.clone()

Multi dimension array: shallow copy

**// Java program to demonstrate**

**// cloning of multi-dimensional arrays**

class Test

{

public static void main(String args[])

{

int intArray[][] = {{1,2,3},{4,5}};

int cloneArray[][] = intArray.clone();

// will print false

System.out.println(intArray == cloneArray);

// will print true as shallow copy is created

// i.e. sub-arrays are shared

System.out.println(intArray[0] == cloneArray[0]);

System.out.println(intArray[1] == cloneArray[1]);

}

}

**Output**:

false

true

true

**Time complexity:** Instead of measuring actual time required in executing each statement in the code, we consider how many times each statement execute.

**O(n2):** You go and ask the first person of the class, if he has the pen. Also, you ask this person about other 99 people in the classroom if they have that pen and so on,  
This is what we call O(n2).

**O(n):** Going and asking each student individually is O(N).

**O (log n):** Now I divide the class into two groups, then ask: “Is it on the left side, or the right side of the classroom?” Then I take that group and divide it into two and ask again, and so on. Repeat the process till you are left with one student who has your pen. This is what you mean by O (log n).

**Sorting Algos:**

Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.

2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Here swapping will happen with smallest and element in that place

Bubble sort:

Compares adjacent elements and bring largest or smallest to the end of array at each n-i number of iterations

Insertion sort:

It is incremental approach, searches for the number which is less than that number and places it in next to less number.  
use: when no of elements are less/ array is almost in sorting order

Merge sort:

Merge Sort is useful for sorting linked lists in O(nLogn) time.In the case of linked lists, the case is different mainly due to the difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike an array, in the linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

Quick Sort:

Like [Merge Sort](http://quiz.geeksforgeeks.org/merge-sort/), Quicksort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quicksort is partition (). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Used in case of array sorting. And large size of data.

**Searching Algo’s:**

Linear search:O(n)

Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to Linear search.

Binary Search:O(logn)

**Trigger:** A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger\_name]

[before | after]

{insert | update | delete}

on [table\_name]

[for each row]

[trigger\_body]

**String:**

Immutable, will create another string when we modify the string.

It will store value in Heap and String constant pool.

When we create new string with already created string, new String will point to SCP value.

String comaparision ways:

* String.equals()-compares data/content of the string variable
* String.equalsIgnoreCase
* Custome method-we can write method to compare each character in string
* Object.equals()
* String.compareTo()

“**==**” checks for reference equality (whether two are pints to same location or not)

StringTokenizer:

* StringTokenizer(String str)
* StringTokenizer(String str, char delim)
* StringTokenizer(String str, char delim, boolean flag)

If flag is true, consider delim as one of the tokens

hasMoreTokens()

nextToken()

StrringJoiner:

Add ()

Merge ()

Length ()

setEmptyValue ()

length ()

**Constructors:**

Base class no argument constructor will be called first before derived

If we need to call base parameterized constructor, need to use Super () as the first statement in derived class constructor

There are no “return value” statements in constructor, but constructor returns current class instance. We can write ‘return’ inside a constructor.

Initialization block({statements;}) will execute before constructors

We cannot say whenever the class constructor called means object created. Example below:

**Base class:**

Class Fruit {

public Fruit(){

sop(this.hashCode());

}

}

**Derived Class:**

Class Apple extends Fruit {

public Apple(){

sop(this.hashCode());

}

}

**Driver Class:**

Class Test{

Public static void main(String args[]){

Apple ap=new Apple();

}

}

Above program print same hash code bcoz here only one object (Apple is created)

**Shallow Vs Deep Copy:**

**Shallow**

While copying object type, copies the reference, so both fields will point to same location, Clone () of Cloneable interface (CloneNotSupportedException).

It is cheap and simple

**Deep**

Need to do it manually

**Arrays.equals():**

Shallow comparison-compares object references in case of array of arrays, not content

**Arrays.deepEquals():**

Deep comparison i.e. content of array of arrays will be compared

**Object: java.lang package**

Methods:

toString()

hashCode()-not return address of the object-it coverts internal address of the object to integer using algorithem

equals()-if want to override this, should override hashCode(). If two objects are equal only when hash code is equal

getClass()-Final method, cannot override. Returns class object of “this” object. Used to get metadata of class.

finalize():called just before an object is garbage collected. For example, before destroying Servlet objects web container, always called finalize method to perform clean-up activities of the session.

Clone()

Wait,notify,notifyAll():-final methods of object class

Must be used in Synchronized class.

**Garbage collector:**

It will Free heap memory by destroying unreachable objects/island of isolation objects.

Ways to make object eligible for garbage collection:

1. Nullify reference variable
2. Re assign reference variable
3. Object created inside method
4. Island of Isolation
5. Anonymous object

New objct();

**Ways for requesting JVM to run Garbage Collector:**

Make it eligible for GC

Request JVM to invoke GC:

System.gc();

Runtime.getRunTime().gc();

**Date comparison:**

Date.comareTo()

Date.befire(),after(),equals()

Calender.before(),after(),equals()

Calendar c= Calendar.getInstance();

c.SetTime((Date)sdf.parse(date))

Jave8 isBefore(),isAfter(),isEquals()

**Shadowing of static functions in Java**

the variable type decides the method being invoked, not the assigned object type in overriding of static functions.

compiler uses special run-time polymorphism mechanism only for methods.

It is possible to access child data members using parent pointer with typecasting. If a parent reference variable is holding the reference of the child class and we have the “value” variable in both the parent and child class, it will refer to the parent class “value” variable, whether it is holding child class object reference. The reference holding the child class object reference will not be able to access the members (functions or variables) of the child class. It is because compiler uses special run-time polymorphism mechanism only for methods.

**Reflection:**

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

* The required classes for reflection are provided under java.lang.reflect package.
* Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
* Through reflection we can invoke methods at runtime irrespective of the access specifier used with them

**Thread life cycle:**

1. New
2. Runnable
3. Blocked
4. Waiting
5. Timed Waiting
6. Terminated

**Exceptions:**

**Throwable** is root class of **Error** (used by JVM to indicate that errors having to do with JRE ex:StachOverFlow error, Virtual Machine error) and **Exception** (unexpected event accrues during the execution of prog, can catch using try/Catch)

**Unchecked exception**: run time (Arithmetic, null pointer)

**Checked Exception**: compile time (IO or compile time)

**Throw**: to manually throw an exception

**Throws**:to handle exception without try catch block, specifies the exception that method can throw to the caller method and does not handle itself

-For each try block there can be zero or more catch blocks but should be only one finally block

Chained exceptions: in case of exception caused due to the cause of another exception, we can use Throwable class methods.

getCauase()

initCause(Trowable cause)

Two ways to write catch for multiple occurrences of exceptions:

Multiple catch blocks

One catch with if conditions using **instanceof**

**Instanceof vs isInstance():**

Both are for checking class of the object, I f we want to dynamically check class then we have to use isInstace() method

instanceof operator throws compile time error(Incompatible conditional operand types) if we check object with other classes which it doesn’t instantiate.

|  |
| --- |
| public class Test  {  public static void main(String[] args)      {          Integer i = new Integer(5);          // Below line causes compile time error:-          // Incompatible conditional operand types          // Integer and String          System.out.println(i instanceof String);      }  } |
|  |

**Output**:

13: error: incompatible types: Integer cannot be converted to String

System.out.println(i instanceof String);

^

**New vs newInstance Method ();**

In general, new operator is used to create objects, but if we want to decide type of object to be created at runtime, there is no way we can use new operator. In this case, we have to use newInstance() method.

Object obj = Class.forName(c).newInstance();

**User defined exceptions:**

The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:

class MyException extends Exception

We can write a default constructor in his own exception class.

MyException(){}

We can also create a parameterized constructor with a string as a parameter.

We can use this to store exception details. We can call super class(Exception) constructor from this and send the string there.

MyException(String str)

{

super(str);

}

To raise exception of user-defined type, we need to create an object to his exception class and throw it using throw clause, as:

MyException me = new MyException(“Exception details”);

throw me;

**Exception with overriding of methods with throws:**

1. If SuperClass does not declare an exception, then the SubClass can only declare unchecked exceptions, but not the checked exceptions.
2. If SuperClass declares an exception, then the SubClass can only declare the child exceptions of the exception declared by the SuperClass, but not any other exception.
3. If SuperClass declares an exception, then the Subclass can declare without exception.

**Static:**

We can create static variables at class-level only.

static block and static variables are executed in order they are present in a program.

Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to this or super in any way.

**Static blocks:**

If you need to do computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.

**Static variables and methods:**

Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

-We cannot override static methods because of early binding or compile time binding

- static methods cannot be Abstract

-can overload static methods  
-interface can have static method

**Enum:**

Enums are used when we know all possible values at compile time

Basically used to create our own data types.

Every enum constant is always implicitly public static final. Since it is static, we can access it by using enum Name. Since it is final, we can’t create child enums.

We can declare main() method inside enum.

All enums implicitly extend java.lang.Enum class. As a class can only extend one parent in Java, so an enum cannot extend anything else.

toString() method is overridden in java.lang.Enum class,which returns enum constant name.

enum can implement many interfaces.

**values(), ordinal() and valueOf() methods :**

These methods are present inside java.lang.Enum.

values() method can be used to return all values present inside enum.

Order is important in enums.By using ordinal() method, each enum constant index can be found, just like array index.

valueOf() method returns the enum constant of the specified string value, if exists.

-can iterate enum

* by converting enum to list(Arrays.asList())/set(EnumSet.allOf())
* foreach
* Stream.of()- **java.util.Stream**

**Iterators:**

1. Enumeration: Added in java 1.1 version

Enumeration e=v.elements();

Can be used only for old data types like Vector, HashTable. Not universal iterator

Remove operations can’t be performed using Enumeration.

Only forward direction iterating is possible.

Methods: hasMoreElements(),nextElement()

1. Iterator: Advanced to Enumeration with remove functionality

Universal iterator can be sued for all collections

Methods:hasNext(), next(), remove()

Iterator i=c.iterator();

Disadvantages: forward direction & replacement/addition of element is not supported with this.

1. ListIterator : It is the most powerful iterator but it is only applicable for List implemented classes, so it is not a universal iterator.
2. Methods: including listteratir methods it has nestIndex, previous …..also add(), set()

Any iterator reference will point to the index just before the index of first element in a collection.

We don’t create objects of Enumeration, Iterator, ListIterator because they are interfaces. We use methods like elements(), iterator(), listIterator() to create objects.

**Collections Framework:**

Collection(Set,List),Map

Set(I):add(),retainAll()-Interception, addAll(),removeAll()-defference()….

Concurrent collections:

ConcurrentHashMap, Concurrent Map,CopyOnWriteArrayList……

**string vs stringbuilder vs stringbuffer:**

* If a string is going to remain constant throughout the program, then use String class object because a String object is immutable.
* If a string can change (example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.
* If a string can change, and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous so you have thread-safety.

**Java8 features:**

Default methods and static methods in Interfaces

Lambda expressions

Stream API-functional programming

Foreach()

Method references – call by method (passes method as variable into other method)

Functional interfaces

**Design patterns**

**Factory design:** Factory design pattern is used when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-class. This pattern takes out the responsibility of instantiation of a class from client program to the factory class.

-Static method in a class

**Singleton:**

The singleton pattern is a design pattern that restricts the instantiation of a class to one object.

* Singleton classes can have only one instance and that instance should be globally accessible.
* java.lang.Runtime and java.awt.Desktop are 2 singleton classes provided by JVM.
* Singleton Design pattern is a type of creational design pattern.
* Outer classes should be prevented to create instance of singleton class.

**Builder Design Pattern:**

“Separate the construction of a complex object from its representation so that the same construction process can create different representations.” It is used to construct a complex object step by step and the final step will return the object. The process of constructing an object should be generic so that it can be used to create different representations of the same object.

**Decorator:**

The decorator pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

**Externalizable interface:**

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream.

Externalizable interface present in java.io, is used for Externalization which extends Serializable interface. It consist of two methods which we have to override to write/read object into/from stream which are-

// to read object from stream

void readExternal(ObjectInput in)

// to write object into stream

void writeExternal(ObjectOutput out)

**Spring:**

DI:

Bean factory:xmlBeanFactory(new FileSystemResource())

ApplicationContect:ClassPathXMLApplicationContext();

Setter Injection(property)/Constructor Injection (constructor-arg)-by type/index/name---(value,ref)

-inside <property> cab add <bean> for inner beans

<idRef>

autowire=”byname/byType/constructor”-Constructor is for constructor injuction like byType is setter injection

Scope of bean:

Singleton- only once per Spring Container – default

Prototype-Creates new bean for each request/reference

Web-aware Context Bean scopes:

Request-new bean per servlet request

Session-new bean per session

Global Session-per global HTTP session

Aware interfaces:

BeanNameAware Interface

ApplicationContextAware - all have corresponding setter methods

Bean definition Inheritance:

Child bean have parant=”parantID”

Merge=” true” to merge values to the existing parent collection in collection tag

abstract=true for not initializing bean by spring

**lifecycle Callback methods:**

InitializingBean(afterPropertiesSet()), DisposableBean(destroy())-interfaces

Or

Init-method, destroy-method in<bean>

Default-Init-method, Default-destroy-method in <beans>

BeanPostProcessor: called after initializing the bean factory

A separate class implements this Interface having methods- postProcessBeforeInitiolization, postProcessAfterInitilization(Object o, string name) –should return bean.

Ex: RequiredAnnotationBeanPostProcessor

AutowiredAnnotationBeanPostProcessor

<context:annotation-config/>

BeanFactoryPostProcessor: called before initializing the bean factory –method postProcessBeanFactory(ConfigurableListBeanFactory factory)

Ex: org.sf.beans.factory.config.PropertiesPlaceholderConfigurer

org.sf.beans.factory.config.PropertiesPlaceholderConfigurer

Annotations:

@Required – for mandatory dependencies

@Autowired: first byType then byname

@Qualifier -nees to provide <qualifier> tag in xml for bean

@Component- declares class as a bean -<context:component-scan base-package=””/>

@Service--<context:component-scan base-package=””/>

@Repository --<context:component-scan base-package=””/>

@Controller --<context:component-scan base-package=””/> - for all stereotypes

JSR 250 Annotations:

@Resource(name=””) –like auto wired

@PostConstruct

@PreDestroy

ResourceBunldeMessageSource:

Messaging and internationli]zation support by application contxt not BF

HandleInterceotprAdaptor

localeChaneInterceptor

CockieLocaleResolver

ThemechangeInterceptor

**Java.util package?**

**Sequences and Synonyms in sql?**

**view?**

**Scheduler?**

**clonable, serializable**

**Regex in java?**

**Java 8 features?**

**Comparator in java**

**JMS-ActiveMQ**

**OAuth**

**Design patterns**

**Xcel data import**

**File operators**

**Bean life cycle**

**beanFactory vs apllicationcontext**

**RESTTemplate**

**Stored proc execution**