**Crack Technical Interview:**

**1.Strings:**

**Why is char[] preferred over String for passwords?**

Since Strings are immutable in Java, if you store the password as plain text it will be available in memory until the Garbage collector clears it, and since String is used in the String pool for reusability there is a pretty high chance that it will remain in memory for a long duration, which poses a security threat.

### **What are different ways to create a String Object?**

We can create String object using new operator like any normal java class or we can use double quotes to create a String object.

When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with the same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.

When we use the new operator, JVM creates the String object but don’t store it into the String Pool. We can use intern() method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.

**Difference between String, StringBuffer and StringBuilder?**

The string is immutable and final in Java, so whenever we do String manipulation, it creates a new String. String manipulations are resource consuming, so java provides two utility classes for String manipulations – StringBuffer and StringBuilder.

StringBuffer and StringBuilder are mutable classes. StringBuffer operations are thread-safe and synchronized where StringBuilder operations are not thread-safe. So in a multi-threaded environment, we should use StringBuffer but in the single-threaded environment, we should use StringBuilder.

StringBuilder performance is fast than StringBuffer because of no overhead of synchronization.

### **Why String is immutable or final in Java**

* It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as database username, password etc.
* Since String is immutable, it’s safe to use in multi-threading and we don’t need any synchronization.

### **How do you check if two Strings are equal in Java?**

There are two ways to check if two Strings are equal or not – using “==” operator or using equals method. When we use “==” operator, it checks for the value of String as well as the reference but in our programming, most of the time we are checking equality of String for value only. So we should use the equals method to check if two Strings are equal or not.

### **What does String intern() method do?**

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

### **Does String is thread-safe in Java?**

Strings are immutable, so we can’t change it’s value in program. Hence it’s thread-safe and can be safely used in multi-threaded environment.

### **Why String is popular HashMap key in Java?**

Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for the key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

**What is Race condition?**

**How do you detect them?**

**How do you handle them?**

**Finally, how do you prevent them from occurring?**

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.

Problems often occur when one thread does a "check-then-act" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g:

if (x == 5) // The "Check"

{ y = x \* 2; // The "Act"

// If another thread changed x in between "if (x == 5)" and "y = x \* 2" above, // y will not be equal to 10.

}

The point being, y could be 10, or it could be anything, depending on whether another thread changed x in between the check and act. You have no real way of knowing.

In order to prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time. This would mean something like this:

// Obtain lock for x if (x == 5) {

y = x \* 2; // Now, nothing can change x until the lock is released.

// Therefore y = 10

} // release lock for x

**What does the other thread do when it encounters the lock? Does it wait? Error?**

Yes, the other thread will have to wait until the lock is released before it can proceed. This makes it very important that the lock is released by the holding thread when it is finished with it. If it never releases it, then the other thread will wait indefinitely.

**2.Exception Handing:**

### **What is Exception in Java?**

Exception is an error event that can happen during the execution of a program and disrupts it’s normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then [JRE](https://www.journaldev.com/546/difference-jdk-vs-jre-vs-jvm) tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as catching the exception. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.Java Exception handling framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

### **What are the Exception Handling Keywords in Java?**

1. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. **throw** keyword is used to throw exception to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it’s caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with **main()** method also.
3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurrs or not.

### **Explain Java Exception Hierarchy?**

ava Exceptions are hierarchical and [inheritance](https://www.journaldev.com/644/inheritance-java-example) is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exception.

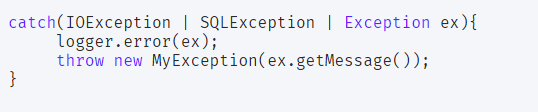
**Errors** are exceptional scenarios that are out of scope of application and it’s not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error.

**Checked Exceptions** are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. Exception is the parent class of all Checked Exceptions.

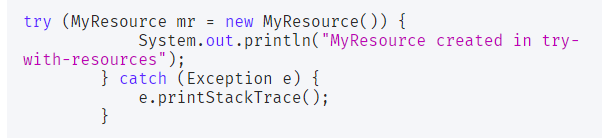
**Runtime Exceptions** are caused by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions.

### **Explain Java 7 ARM Feature and multi-catch block?**

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the feature was multi-catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:



Most of the time, we use finally block just to close the resources and sometimes we forget to close them and get runtime exceptions when the resources are exhausted. These exceptions are hard to debug and we might need to look into each place where we are using that type of resource to make sure we are closing it. So java 7 one of the improvement was **try-with-resources** where we can create a resource in the try statement itself and use it inside the try-catch block. When the execution comes out of try-catch block, runtime environment automatically close these resources. Sample of try-catch block with this improvement is:



### **What is difference between final, finally and finalize in Java?**

final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can’t be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it’s great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

**What is the difference between error and exception in java?**

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory. Where as exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

**What is the difference between ClassNotFoundException and NoClassDefFoundError in java?**

In Java, both ClassNotFoundException and NoClassDefFoundError occur when a particular class is not found at run time. But, they occur at different scenarios. ClassNotFoundException is an exception which occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath. On the other hand, NoClassDefFoundError is an error which occurs when a particular class is present at compile time but it was missing at run time.

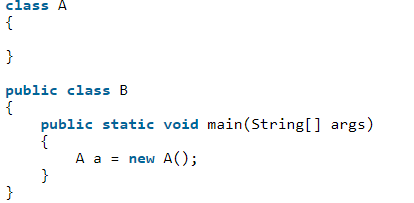
**ClassNotFoundException In Java :**

ClassNotFoundException is a run time exception which is thrown when an application tries to load a class at run time using **Class.forName()** or **loadClass()** or **findSystemClass()** methods and the class with specified name are not found in the classpath.

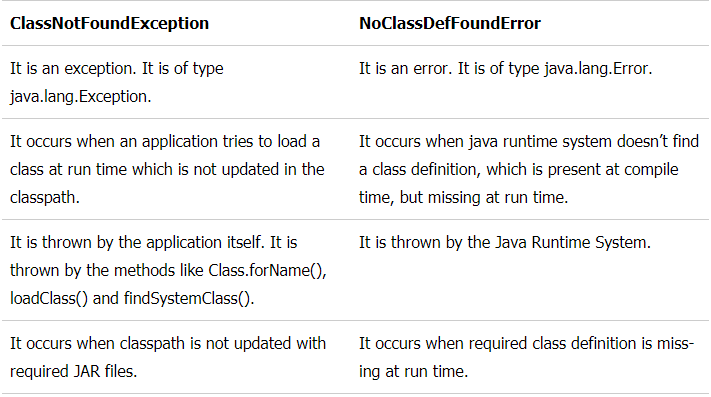
program will throw ClassNotFoundException if the mentioned class **“oracle.jdbc.driver.OracleDriver”** is not found in the classpath.

## **NoClassDefFoundError In Java :**

NoClassDefFoundError is an error which is thrown when Java Runtime System tries to load the definition of a class and class definition is no longer available. The required class definition was present at compile time but it was missing at run time.



When you compile the above program, two .class files will be generated. One is **A.class** and another one is **B.class**. If you remove the **A.class** file and run the **B.class** file, Java Runtime System will throw NoClassDefFoundError

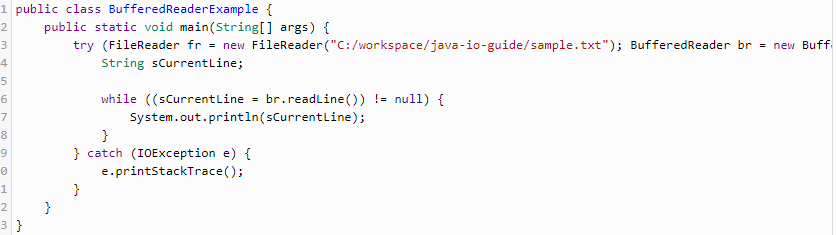


**Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?**

No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but can not be overridden with checked exceptions.

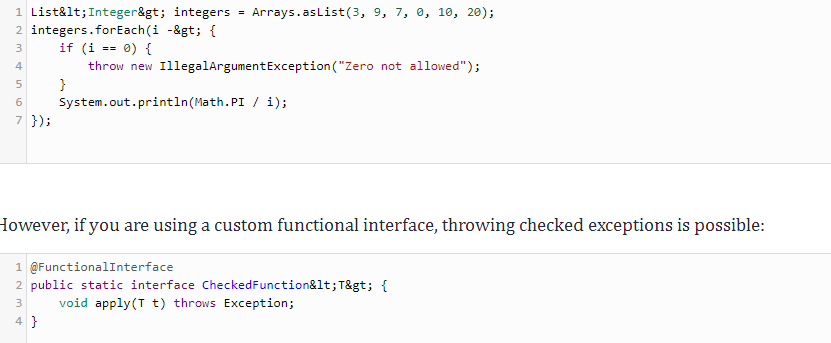
## **What Is the Try-With-Resources Statement?**

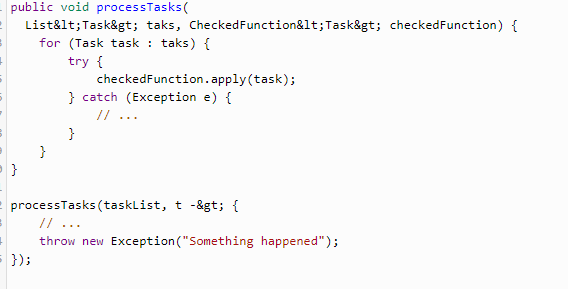
In Java, the try-with-resourcesstatement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.



## **Can You Throw any Exception Inside a Lambda Expression’s Body?**

When using a standard functional interface already provided by Java, you can only throw unchecked exceptions because standard functional interfaces do not have a “throws” clause in its method signatures:





**Oracle:**

**Difference between varchar and varchar2 data types?**

Varchar can store upto 2000 bytes and varchar2 can store upto 4000 bytes. Varchar will occupy space for NULL values and Varchar2 will not occupy any space. Both are differed with respect to space.

**What is RAW datatype?**

RAW datatype is used to store values in binary data format. The maximum size for a raw in a table in 32767 bytes.

**What is the use of NVL function?**

The NVL function is used to replace NULL values with another or given value. Example is –

NVL(Value, replace value)

**What is BLOB datatype?**

**BLOB** : Variable-length binary large object string that can be up to 2GB (2,147,483,647) long. ... **CLOB** : Variable-length character large object string that can be up to 2GB (2,147,483,647) long. A **CLOB** can store single-byte character strings **or** multibyte, character-based data. A **CLOB** is considered a character string.

**What do you mean by GROUP BY Clause?**

A GROUP BY clause can be used in select statement where it will collect data across multiple records and group the results by one or more columns.

**How do we create privileges in Oracle?**

A privilege is nothing but right to execute an SQL query or to access another user object. Privilege can be given as system privilege or user privilege.

[sql]GRANT user1 TO user2 WITH MANAGER OPTION;[/sql]

**What is a View?**

View is a logical table which based on one or more tables or views. The tables upon which the view is based are called Base Tables and it doesn’t contain data.

**What is a cursor variable?**

A cursor variable is associated with different statements which can hold different values at run time. A cursor variable is a kind of reference type.

**What are the various constraints used in Oracle?**

Following are constraints used:

* NULL – It is to indicate that particular column can contain NULL values
* NOT NULL – It is to indicate that particular column cannot contain NULL values
* CHECK – Validate that values in the given column to meet the specific criteria
* DEFAULT – It is to indicate the value is assigned to default value

**What is the parameter mode that can be passed to a procedure?**

IN, OUT and INOUT are the modes of parameters that can be passed to a procedure.

**What are the different Oracle Database objects?**

There are different data objects in Oracle –

* Tables – set of elements organized in vertical and horizontal
* Views – Virtual table derived from one or more tables
* Indexes – Performance tuning method for processing the records
* Synonyms – Alias name for tables
* Sequences – Multiple users generate unique numbers
* Tablespaces – Logical storage unit in Oracle

**What is the maximum number of triggers that can be applied to a single table?**

12 is the maximum number of triggers that can be applied to a single table.

**How can we view last record added to a table?**

Last record can be added to a table and this can be done by –

MySQL

| 1 | Select \* from (select \* from employees order by rownum desc) where rownum<2; |
| --- | --- |

**How to display employee records who gets more salary than the average salary in the department?**

**Select \* from employee where salary>(select avg(salary) from dept, employee where dept.deptno = employee.dept**

**What is a primary key?**

A primary key is a combination of fields which uniquely specify a row. This is a special kind of unique key, and it has implicit NOT NULL constraint. It means, Primary key values cannot be NULL.

**What is a unique key?**

A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns.

A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key.

There can be many unique constraint defined per table, but only one Primary key constraint defined per table.

**What is a foreign key?**

A foreign key is one table which can be related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table.

* **Right Join.**

Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

* **Left Join.**

Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

**What is a View?**

A view is a virtual table which consists of a subset of data contained in a table. Views are not virtually present, and it takes less space to store. View can have data of one or more tables combined, and it is depending on the relationship.

**What is an Index?**

An index is performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and it will be faster to retrieve data.

**What is a Cursor?**

A database Cursor is a control which enables traversal over the rows or records in the table. This can be viewed as a pointer to one row in a set of rows. Cursor is very much useful for traversing such as retrieval, addition and removal of database records.

**What is a stored procedure?**

Stored Procedure is a function consists of many SQL statement to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required.

**What is a trigger?**

A DB trigger is a code or programs that automatically execute with response to some event on a table or view in a database. Mainly, trigger helps to maintain the integrity of the database.

Example: When a new student is added to the student database, new records should be created in the related tables like Exam, Score and Attendance tables.

**What is the difference between DELETE and TRUNCATE commands?**

DELETE command is used to remove rows from the table, and WHERE clause can be used for conditional set of parameters. Commit and Rollback can be performed after delete statement.

TRUNCATE removes all rows from the table. Truncate operation cannot be rolled back.

**Advantages and Disadvantages of Stored Procedure?**

Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data.

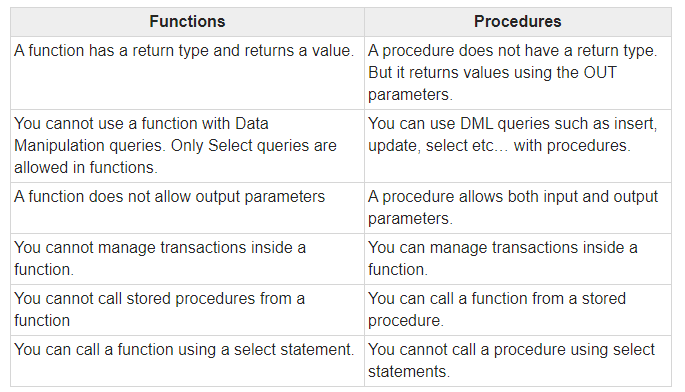
Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

**What is the difference between TRUNCATE and DROP statements?**

TRUNCATE removes all the rows from the table, and it cannot be rolled back. DROP command removes a table from the database and operation cannot be rolled back.

**Stored procedure vs Functions:**

The **function** must return a value but in **Stored Procedure** it is optional. Even a **procedure** can return zero or n values. **Functions** can have only input parameters for it whereas **Procedures** can have input or output parameters. **Functions** can be called from **Procedure** whereas **Procedures** cannot be called from a **Function**

****

# [**Finding duplicate values in a SQL table**](https://stackoverflow.com/questions/2594829/finding-duplicate-values-in-a-sql-table)

Select name,email,count(\*) from Emp where group by name,email having count(\*)>1

# **SQL GROUP BY and HAVING Clause with Examples**

The GROUP BY clause is a SQL command that is used to group rows that have the same values. The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

**Suppose we want total number of males and females in our database.**

Select gender,count(gender) from emp group by gender;

**Nth highest sal of emp:**

[**Ask Question**](https://stackoverflow.com/questions/ask)

Select \* from (select \* from emp order by sal desc) where row\_num=n;

### **If you are a SQL Developer, how can you delete duplicate records in a table with no primary key?**

We can delete the duplicate records in the table with no primary key by using the SET ROWCOUNT command. It limits the number of records affected by a command. For example, if you have 2 duplicate rows, you would SET ROWCOUNT 1, execute DELETE command and then SET ROWCOUNT 0

### **Write SQL query to fetch employee names having a salary greater than or equal to 5000 and less than or equal 10000.**

By using BETWEEN in the where clause, we can retrieve the Employee Ids of employees with salary >= 5000and <=10000. This can be used as a subquery to find the Full name of Employees having those Ids from the EmployeeDetails table.

SELECT FullName

FROM EmployeeDetails

WHERE EmpId IN

(SELECT EmpId FROM EmployeeSalary

WHERE Salary BETWEEN 5000 AND 10000)

### **Write an SQL Query to find maximum salary in each department of an organisation.**

By using the MAX() function and grouping on the basis of Department ID, we can calculate the maximum salary in each department. The following query will help us achieve the desired result.

SELECT DeptId, MAX(Salary) FROM Employee

GROUP BY DeptId

In case we need the department name too, we will need to join the Employee table with Department table using foreign key DepId. The following query can be used.

SELECT DeptName, MAX(Salary) FROM Employee e RIGHT JOIN Department d ON e.DeptId =d.DeptId GROUP BY DeptName

**List all tables in DB:**

select table\_name from user\_tables;

**List all columns in a table :**

select column\_name from user\_tab\_cols where table\_name='CUSTOMER';

**Collections:**

### **What is Java Collections Framework? List out some benefits of Collections framework?**

* Reduced development effort by using core collection classes rather than implementing our own collection classes.
* Code quality is enhanced with the use of well tested collections framework classes.
* Reduced effort for code maintenance by using collection classes shipped with JDK.
* Reusability and Interoperability

### **What is the benefit of Generics in Collections Framework?**

Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation.

### **What are the basic interfaces of Java Collections Framework?**

[Collection](https://www.journaldev.com/1260/collections-in-java-tutorial#collection-interface) is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

[Set](https://www.journaldev.com/1260/collections-in-java-tutorial#set-interface) is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

[List](https://www.journaldev.com/1260/collections-in-java-tutorial#list-interface) is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.

A [Map](https://www.journaldev.com/1260/collections-in-java-tutorial#map-interface) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are [Queue](https://www.journaldev.com/1260/collections-in-java-tutorial#queue-interface), [Dequeue](https://www.journaldev.com/1260/collections-in-java-tutorial#dequeue-interface), [Iterator](https://www.journaldev.com/1260/collections-in-java-tutorial#iterator-interface), [SortedSet](https://www.journaldev.com/1260/collections-in-java-tutorial#sortedset-interface), [SortedMap](https://www.journaldev.com/1260/collections-in-java-tutorial#sortedmap-interface) and [ListIterator](https://www.journaldev.com/1260/collections-in-java-tutorial#listiterator-interface).

### **Why Map interface doesn’t extend Collection interface?**

If Map extends Collection interface, then where are the elements? The map contains key-value pairs and it provides methods to retrieve the list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

### **What is an Iterator?**

The Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using *iterator()* method. Iterators allow the caller to remove elements from the underlying collection during the iteration. Java Collection iterator provides a generic way for traversal through the elements of a collection and implements [**Iterator Design Pattern**](https://www.journaldev.com/1716/iterator-design-pattern-java).

### **What is difference between Enumeration and Iterator interface?**

Enumeration is twice as fast as Iterator and uses very little memory. Enumeration is very basic and fits basic needs. But the Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.

### **What is different between Iterator and ListIterator?**

* We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
* Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
* ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

### **What do you understand by iterator fail-fast property?**

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

### **What is difference between fail-fast and fail-safe?**

Iterator fail-safe property work with the clone of underlying collection, hence it’s not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe.

Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException.

### **How to avoid ConcurrentModificationException while iterating a collection?**

We can use concurrent collection classes to avoid ConcurrentModificationException while iterating over a collection, for example CopyOnWriteArrayList instead of ArrayList.

### **What is UnsupportedOperationException?**

UnsupportedOperationException is the exception used to indicate that the operation is not supported. It’s used extensively in [JDK](https://www.journaldev.com/546/difference-jdk-vs-jre-vs-jvm) classes, in collections framework java.util.Collections.UnmodifiableCollection throws this exception for all add and remove operations.

### **How HashMap works in Java?**

### **What is difference between HashMap and Hashtable?**

1. HashMap allows null key and values whereas Hashtable doesn’t allow null key and values.
2. Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.

### **How to decide between HashMap and TreeMap?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

### **What are similarities and difference between ArrayList and Vector?**

1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
2. ArrayList is faster than Vector because it doesn’t have any overhead because of synchronization.
3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

### **What is difference between Array and ArrayList? When will you use Array over ArrayList?**

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.

Arrays are fixed-size whereas ArrayList size is dynamic.

Arrays don’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator, etc.

### **What is difference between ArrayList and LinkedList?**

ArrayList and LinkedList both implement List interface but there are some differences between them.

1. ArrayList is an index based data structure backed by Array, so it provides random access to its elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to its previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.
2. Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.
3. LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

### **Which collection classes are thread-safe?**

Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment.

### **What are concurrent Collection Classes?**

Java 1.5 Concurrent package (java.util.concurrent) contains thread-safe collection classes that allow collections to be modified while iterating. By design Iterator implementation in java.util packages are fail-fast and throws ConcurrentModificationException. But Iterator implementation in java.util.concurrent packages are fail-safe and we can modify the collection while iterating. Some of these classes are CopyOnWriteArrayList, ConcurrentHashMap, CopyOnWriteArraySet.

### **What is Queue and Stack, list their differences?**

Both Queue and Stack are used to store data before processing them. java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it’s not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.

The stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.

Stack is a class that extends Vector whereas Queue is an interface.

### **How can we sort a list of Objects?**

If we need to sort an array of Objects, we can use Arrays.sort(). If we need to sort a list of objects, we can use Collections.sort(). Both these classes have overloaded sort() methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator).

Collections internally uses Arrays sorting method, so both of them have same performance except that Collections take sometime to convert list to array.

### **While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?**

We can create a read-only collection using Collections.unmodifiableCollection(Collection c) method before passing it as argument, this will make sure that any operation to change the collection will throw UnsupportedOperationException.

### **What is Comparable and Comparator interface?** Java provides a Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. The comparable interface has a compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as an argument. But, in most real-life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can’t choose the field on which we want to sort the Object. Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if the first argument is less than the second one and returns zero if they are equal and positive int if the first argument is greater than the second one. Check this post for use of Comparable and Comparator interface to [sort objects](https://www.journaldev.com/780/comparable-and-comparator-in-java-example).

**What is difference between Comparable and Comparator interface?**Comparable and Comparator interfaces are used to sort collection or array of objects.  
Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic.  
Comparator interface is used to provide different algorithms for sorting and we can choose the comparator we want to use to sort the given collection of objects.

**How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of the null key?**

The null key is handled specially in HashMap, there are two separate methods for that putForNullKey(V value) and getForNullKey(). Later is an offloaded version of get() to look up null keys. Null keys always map to index 0.

This null case is split out into separate methods for the sake of performance in the two most commonly used operations (get and put), but incorporated with conditionals in others. In short, equals() and hashcode() method are not used in case of null keys in HashMap.

Read more: <https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz6lZI4vDJy>

**HashMap Changes in JDK 1.7 and JDK 1.8**

There is some [performance improvement done on HashMap and ArrayList from JDK 1.7](http://javarevisited.blogspot.com/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html), which reduces memory consumption. Due to this empty Map are lazily initialized and will cost you less memory. Earlier, when you create HashMap e.g. new HashMap() it automatically creates an array of default length e.g. 16.

After some research, the Java team found that most of this Map are temporary and never use that many elements, and only end up wasting memory. Also, From JDK 1.8 onwards HashMap has introduced an improved strategy to deal with a high collision rate.

Since a poor hash function e.g. which always return the location of the same bucket, can turn a HashMap into linked list, i.e. converting get() method to perform in O(n) instead of O(1) and someone can take advantage of this fact, Java now internally replace linked list to a binary true once certain threshold is breached. This ensures performance or order O(log(n)) even in the worst case where a hash function is not distributing keys properly.

**ConcurrentHashMap:**

It allows concurrent access to the map. Part of the map called *Segment (internal data structure)* is only getting locked while adding or updating the map. So ConcurrentHashMap allows concurrent threads to read the value without locking at all. This data structure was introduced to improve performance.

**Key points of ConcurrentHashMap:**

* The underlined data structure for ConcurrentHashMap is [Hashtable](https://www.geeksforgeeks.org/hashtable-in-java/).
* ConcurrentHashMap class is thread-safe i.e. multiple threads can operate on a single object without any complications.
* At a time any number of threads are applicable for a read operation without locking the ConcurrentHashMap object which is not there in HashMap.
* In ConcurrentHashMap, the Object is divided into a number of segments according to the concurrency level.
* The default concurrency-level of ConcurrentHashMap is 16.
* In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updation in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as **Segment locking or bucket locking**. Hence at a time, 16 update operations can be performed by threads.
* Inserting null objects is not possible in ConcurrentHashMap as key or value.

[How to make Java ArrayList Read-Only](https://www.javatpoint.com/how-to-make-java-arraylist-read-only)

An **ArrayList** can be made **read**-**only** easily with the help of Collections. unmodifiableList() method. This method takes the modifiable **ArrayList** as a parameter and returns the **read**-**only unmodifiable view** of this **ArrayList**.

**MultiThreading:**

**What is the difference between Process and Thread?**

A process is a self contained execution environment and it can be seen as a program or application whereas Thread is a single task of execution within the process. Java runtime environment runs as a single process which contains different classes and programs as processes. Thread can be called lightweight process. Thread requires less resources to create and exists in the process, thread shares the process resources.

### **What are the benefits of multi-threaded programming?**

In Multi-Threaded programming, multiple threads are executing concurrently that improves the performance because CPU is not idle incase some thread is waiting to get some resources. Multiple threads share the heap memory, so it’s good to create multiple threads to execute some task rather than creating multiple processes. For example, Servlets are better in performance than CGI because Servlet support multi-threading but CGI doesn’t.

### **What is difference between user Thread and daemon Thread?**

When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

**Daemon thread** is a low priority **thread** that runs in background to perform tasks such as garbage collection. Properties: They can not prevent the JVM from exiting when all the user **threads** finish their execution. ... If JVM finds running **daemon thread**, it terminates the **thread** and after that shutdown itself.

### **How can we create a Thread in Java?**

There are two ways to create Thread in Java – first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class.

### **What are different states in lifecycle of Thread?**

### **New**

When we create a new Thread object using *new* operator, thread state is New Thread. At this point, thread is not alive and it’s a state internal to Java programming.

### **Runnable**

When we call start() function on Thread object, it’s state is changed to Runnable. The control is given to Thread scheduler to finish it’s execution. Whether to run this thread instantly or keep it in runnable thread pool before running, depends on the OS implementation of thread scheduler.

### **Running**

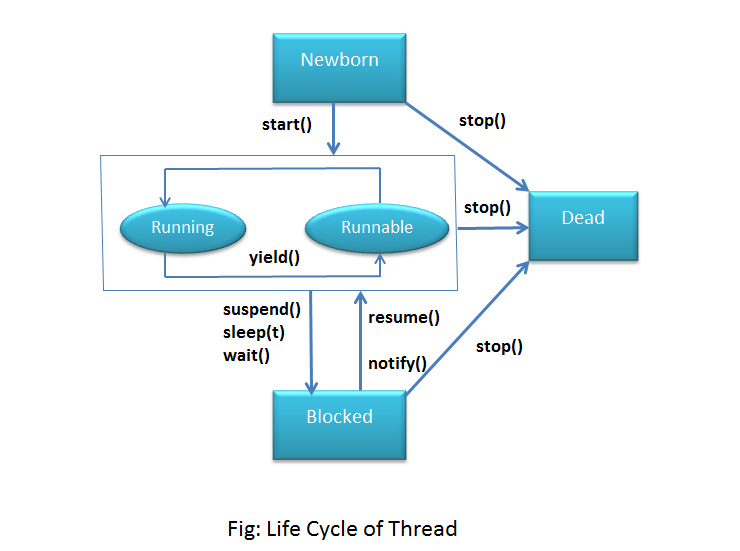
When thread is executing, it’s state is changed to Running. Thread scheduler picks one of the thread from the runnable thread pool and change it’s state to Running. Then CPU starts executing this thread. A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

### **Blocked/Waiting**

A thread can be waiting for other thread to finish using [thread join](https://www.journaldev.com/1024/java-thread-join-example) or it can be waiting for some resources to available. For example [producer consumer problem](https://www.journaldev.com/1034/java-blockingqueue-example) or [waiter notifier implementation](https://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example) or IO resources, then it’s state is changed to Waiting. Once the thread wait state is over, it’s state is changed to Runnable and it’s moved back to runnable thread pool.

### **Dead**

Once the thread finished executing, it’s state is changed to Dead and it’s considered to be not alive.



### **Can we call run() method of a Thread class?**

Yes, we can call run() method of a Thread class but then it will behave like a normal method. To actually execute it in a Thread, we need to start it using **Thread.start()** method.

### **How can w e pause the execution of a Thread for specific time?**

We can use Thread class sleep() method to pause the execution of Thread for certain time. Note that this will not stop the processing of thread for specific time, once the thread awake from sleep, it’s state gets changed to runnable and based on thread scheduling, it gets executed.

### **What do you understand about Thread Priority?**

Every thread has a priority, usually higher priority thread gets precedence in execution but it depends on Thread Scheduler implementation that is OS dependent. We can specify the priority of thread but it doesn’t guarantee that higher priority thread will get executed before lower priority thread. Thread priority is an *int* whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

### **What is Thread Scheduler and Time Slicing?**

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads. Once we create and start a thread, it’s execution depends on the implementation of Thread Scheduler. Time Slicing is the process to divide the available CPU time to the available runnable threads. Allocation of CPU time to threads can be based on thread priority or the thread waiting for longer time will get more priority in getting CPU time. Thread scheduling can’t be controlled by java, so it’s always better to control it from application itself.

### **How can we make sure main() is the last thread to finish in Java Program?**

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function.

### **How does thread communicate with each other?**

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource.

### **Why thread communication methods wait(), notify() and notifyAll() are in Object class?**

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

### **Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?**

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

### **Why Thread sleep() and yield() methods are static?**

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

### **How can we achieve thread safety in Java?**

There are several ways to achieve thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes.

### **What is volatile keyword in Java**

When we use volatile keyword with a variable, all the threads read it’s value directly from the memory and don’t cache it. This makes sure that the value read is the same as in the memory.

Suppose that two threads are working on **SharedObj**. If two threads run on different processors each thread may have its own local copy of **sharedVariable**. If one thread modifies its value the change might not reflect in the original one in the main memory instantly. This depends on the [write policy](https://en.wikipedia.org/wiki/CPU_cache#Write_policies) of cache. Now the other thread is not aware of the modified value which leads to data inconsistency.

### **Which is more preferred – Synchronized method or Synchronized block?**

Synchronized block is more preferred way because it doesn’t lock the Object, synchronized methods lock the Object and if there are multiple synchronization blocks in the class, even though they are not related, it will stop them from execution and put them in wait state to get the lock on Object.

### **How to create daemon thread in Java?**

Thread class setDaemon(true) can be used to create daemon thread in java. We need to call this method before calling start() method else it will throw IllegalThreadStateException.

### **What is Deadlock? How to analyze and avoid deadlock situation?**

Deadlock is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

To analyze a deadlock, we need to look at the java thread dump of the application, we need to look out for the threads with state as BLOCKED and then the resources it’s waiting to lock, every resource has a unique ID using which we can find which thread is already holding the lock on the object.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation, read this post to learn how to [analyze deadlock in java](https://www.journaldev.com/1058/deadlock-in-java-example) with a sample program.

Difference between java.util.Date and java.sql.Date in Java

java.sql.Date just represent DATE without time information while java.util.Date represent both Date and Time information. This is the major differences why java.util.Date can not directly map to java.sql.Date.

**Java 8 features:**

## Date/Time API

Lazy initialization

Binary tree in hashmap

**Spring:**

### **What do you understand by Dependency Injection?**

Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable.

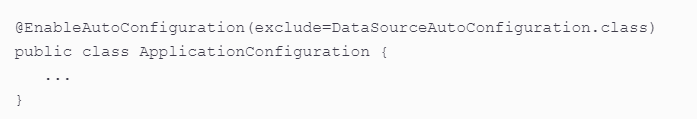
**Spring Boot:**

**What does the @SpringBootApplication annotation do internally?**

Spring Boot @SpringBootApplication annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. It’s same as declaring a class with @Configuration, @EnableAutoConfiguration and @ComponentScan annotations.

# **Disabling specific auto-configuration classes**

At any point in time, if you don't want to use some of the specific auto-configuration classes or if they don't suit your needs, you can disable those auto-configuration classes. For this, you can use the exclude attribute of the @EnableAutoConfiguration annotation. Let's see the following example:

****

**Spring Acutator**

Spring Boot Actuator is a sub-project of the Spring Boot Framework. ... It contains the actuator endpoints (the place where the resources live). We can use HTTP and JMX endpoints to manage and monitor the Spring Boot application.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it. Once this dependency is on the classpath, several endpoints are available for us out of the box.

As we know, Spring Boot provides lots of auto-configuration features that help developers quickly develop production components. But if you think about debugging and how to debug, if something goes wrong, we always need to analyze the logs and dig through the data flow of our application to check to see what's going on. So, the Spring Actuator provides easy access to those kinds of features. It provides many features, i.e. what beans are created, the mapping in the controller, the CPU usage, etc. Automatically gathering and auditing health and metrics can then be applied to your application.

**How to enable spring boot actuator:**

To enable Spring Boot actuator endpoints to your Spring Boot application, we need to add the Spring Boot Starter actuator dependency in our build configuration file.

In the application.properties file, we need to disable the security for actuator endpoints.

management.security.enabled = false

**what is spring initializr why should you use it**

The Spring Initializr is ultimately a web application that can generate a Spring Boot project structure for you. It doesn't generate any application code, but it will give you a basic project structure and either a Maven or a Gradle build specification to build your code with.

It is very easy to develop **Spring** Based applications with Java or Groovy. It reduces lots of development time and increases productivity. It avoids writing lots of boilerplate Code, Annotations and XML Configuration.

**How to exclude any package without using the basePackages filter?**

There are different ways you can filter any package. But Spring Boot provides a trickier option for achieving this without touching the component scan. You can use the exclude attribute while using the annotation @SpringBootApplication. See the following code snippet:

****

**Is this possible to change the port of Embedded Tomcat server in Spring boot?**

Yes, it's possible to change the port. You can use the application.properties file to change the port. But you need to mention "*server.port*" *(i.e. server.port=8081)*. Make sure you have application.properties in your project classpath; REST Spring framework will take care of the rest. If you mention *server.port=0* , then it will automatically assign any available port.

**Can we override or replace the Embedded Tomcat server in Spring Boot?**

Yes, we can replace the Embedded Tomcat with any other servers by using the Starter dependencies. You can use spring-boot-starter-jetty or spring-boot-starter-undertow as a dependency for each project as you need.

**Can we disable the default web server in the Spring Boot application?**

The major strong point in Spring is to provide flexibility to build your application loosely coupled. Spring provides features to disable the web server in a quick configuration. Yes, we can use the application.properties to configure the web application type, i.e. spring.main.web-application-type=none.

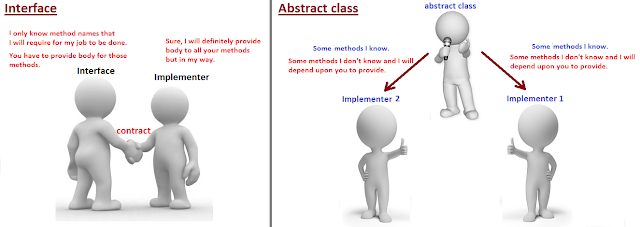
Difference between setter & constructor injection:

Abstract class VS Interface:

[Real time example of abstract class and interface in Java project. | JavaByPatel: Data structures and algorithms interview questions in Java](https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html)

Real time example of abstract class and interface in java projects. abstract class real world example. real time example of Interface in java.

This is very popular interview question for the beginners as well for experienced.



### **interface Vs abstract class in Java.**

Interface:

*Interface is used when you want to define a contract and you don't know anything about implementation. (here it is total abstraction as you don't know anything.)*

Abstract class:

*Abstract class is used when you know something and rely on others for what you don't know.(here it is partial abstraction as some of the things you know and some you don't know.)*

Now, Let's understand above difference between Interface and Abstract class with real world project example.

When to use Interface

Scenario,

Consider we want to start a service like "[makemytrip.com](http://makemytrip.com/)" or "[expedia.com](http://expedia.com/)", where we are responsible for displaying the flights from various flight service company and place an order from customer.

Lets keep our service as simple as,

1. Displaying flights available from vendors like "airasia", "british airways" and "emirates".
2. Place and order for seat to respective vendor.

*How should we design our application considering interfaces and abstract class? In this scenario, interface is useful or abstract class?*

Remember, In this application, we don't own any flight. we are just a middle man/aggregator and our task is to first enquire "airasia", then enquire "british airways" and at last enquire "emirates" about the list of flights available and later if customer opts for booking then inform the respective flight vendor to do booking.

For this, first we need to tell "airasia", "british airways" and "emirates" to give us list of flights, internally how they are giving the list that we don't care.

1. This means I only care for method "getAllAvailableFlights()"  
     
   "getAllAvailableFlights()" from "airasia" may have used SOAP service to return list of flights.  
   "getAllAvailableFlights()" from "british airways" may have used REST service to return list of flights.  
   "getAllAvailableFlights()" from "emirates" may have used CORBA service to return list of flights.  
     
   but we don't care how it is internally implemented and what we care is the contract method "getAllAvailableFlights" that all the flight vendor should provide and return list of flights.
2. Similarly, for booking I only care for method "booking()" that all vendors should have, internally how this vendors are doing booking that I don't care.

To conclude: We know contract.

So we can say that we know the contract that irrespective of who the Flight vendor is, we need "getAllAvailableFlights()" and "booking()" method from them to run our aggregator service.

*In this situation, Interface is useful because we are not aware of the implementation of all the 2 methods required, and what we know is the contract methods that vendor(implementer) should provide. so due to this total abstraction and for defining the contract, interface is useful in this place.*

Technically, we need to design our interface somewhat like below,

FlightOpeartions.java(Contract)

[?](https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html#)

| 1  2  3  4 | interface FlightOpeartions{  void getAllAvailableFlights();  void booking(BookingObject bookingObj);  } |
| --- | --- |

BookingObject.java

[?](https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html#)

| 1 | class BookingObject{} |
| --- | --- |

BritishAirways.java (Vendor 1)

[?](https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html#)

| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class BritishAirways implements FlightOpeartions{    public void getAllAvailableFlights(){  //get british airways flights in the way  //they told us to fetch flight details.  }    public void booking(BookingObject flightDetails){  //place booking order in a way British airways  //told us to place order for seat.  }    } |
| --- | --- |

Emirates.java (Vendor 2)

[?](https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html#)

| 1  2  3  4  5  6  7  8  9  10  11  12 | class Emirates implements FlightOpeartions{    public void getAllAvailableFlights(){  //get Emirates flights in the way  //they told us to fetch flight details.  }    public void booking(BookingObject flightDetails){  //place booking order in a way Emirates airways  //told us to place order for seat.  }  } |
| --- | --- |

When to use Abstract class

Scenario,

Consider we want to start a service like Bulk SMS sender, where we take orders from various telecom vendors like Airtel, France Telecom, Vodafone etc.

For this, we don't have to setup our own infrastructure for sending SMS like Mobile towers but we need to take care of government rules like after 9PM, we should not send promotional SMS, we should also not send SMS to users registered under Do Not Disturb(DND) service etc. Remember, we need to take care of government rules for all the countries where we are sending SMS.

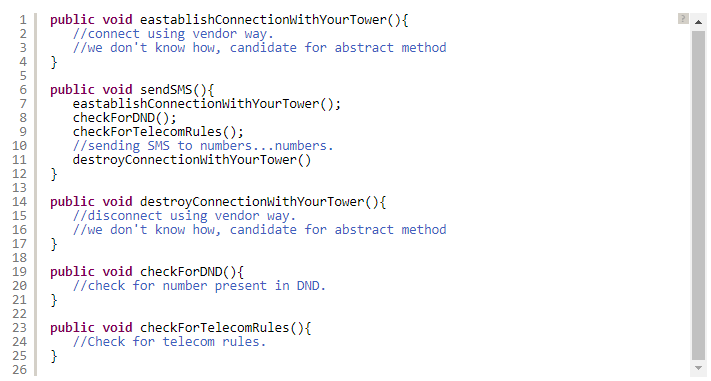
*Note: for infrastructure like towers, we will be relying on vendor who is going to give us order.*

*Example, In case of,*

*Vodafone request us for bulk messaging, in that case we will use Vodafine towers to send SMS.*

*Airtel request us for bulk messaging, in that case we will use Airtel towers to send SMS.*

*What our job is to manage Telecom Regulations for different countries where we are sending SMS.* So what all methods we require would be somewhat like below,



Out of above 5 methods,

1. Methods we know is "sendSMS()", "checkForDND()", "checkForTelecomRules()".
2. Methods we don't know is "eastablishConnectionWithYourTower()", "destroyConnectionWithYourTower()".

we know how to check government rules for sending SMS as that is what our job is but

we don't how to eastablish connection with tower and how to destroy connection with tower because this is purely customer specific, airtel has its own way, vodafone has its own way etc.

*So in the given scenario, we know some methods but there also exist some methods which are unknown and depends on customers.*

In this case, what will be helpful, abstarct class or interface?

*In this case, Abstract class will be helpful, because you know partial things like "checkForDND()", "checkForTelecomRules()" for sending sms to users but we don't know how to eastablishConnectionWithTower() and destroyConnectionWithTower() and need to depend on vendor specific way to connect and destroy connection from their towers.*

Let's see how our class will look like,

| package javabypatel;    abstract class SMSSender{    abstract public void eastablishConnectionWithYourTower();    public void sendSMS(){  /\*eastablishConnectionWithYourTower();  checkForDND();  checkForTelecomRules();    sending SMS to numbers...numbers.\*/  }    abstract public void destroyConnectionWithYourTower();    public void checkForDND(){  //check for number present in DND.  }  public void checkForTelecomRules(){  //Check for telecom rules  }  }      class Vodafone extends SMSSender{    @Override  public void eastablishConnectionWithYourTower() {  //connecting using Vodafone way  }    @Override  public void destroyConnectionWithYourTower() {  //destroying connection using Vodafone way  }    }    class Airtel extends SMSSender{    @Override  public void eastablishConnectionWithYourTower() {  //connecting using Airtel way  }    @Override  public void destroyConnectionWithYourTower() {  //destroying connection using Airtel way  }    } |
| --- |

### **So to summarize,**

### **For Interface:**

### *Interface is used when you don't know anything about implementation but know the contract that implementer should have to accomplish the task.*

### **For Abstract class:**

### *Abstract class is used when you know partial implementation, where say out of 5 methods, you know implementation of 3 methods and don't know implementation of 2 methods in that case 2 methods will be abstract and you need to rely on implementer as a contract to must provide body of abstract methods to accomplish the task.*

**Singleton Design Pattern:**

In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one “single” instance. This is useful when exactly one object is needed to coordinate actions across the system.

These classic design patterns every developer should understand, because it helps us to communicate with other developer abstract level, and it makes us better designer.

**Note:** Adding brief definition with real life and Java API examples.

# **Creational**

Way of creating objects.

**Prototype :** A fully initialized instance to be copied or cloned

Example : initial status of chess game

* java.lang.Object#clone()

**Builder** - Separates the construction of a complex object from its representation so that the same construction process can create different representations.

Example : when we make (or order) a pizza(it can de deferent in size and flavours )

* java.lang.StringBuilder

**Singleton** - A class of which only a single instance can exist

Example : President of a country

* java.lang.Runtime#getRuntime()

**Factory Method** - Creates a **f**amily of object types.

Example : In an organisation HR works as factory method. Here development team request type of resource need to HR. Based on request type, HR provide resource to Development team.

* java.util.Calendar#getInstance()

**Abstract Factory** - Creates an instance of several families of classes

Example : HP, Samsung and Dell laptops are uses Intel and AMD processor.

* javax.xml.parsers.DocumentBuilderFactory#newInstance()

[Factory Method vs Abstract Factory](https://stackoverflow.com/a/5740020/1697099)

# **Structural**

This design patterns is all about Class and Object composition i.e. How do you want structure the software component. This helps us guarantee that when one of the parts changes, the entire structure does not need to change.

**Proxy** - An object representing another object.

Example : check book leaf, credit card, debit card are proxy for Money and a customer representative to order a product.

* java.rmi.\*, the whole API actually.

**Composite** - Gives an unified interface to a leaf and composite.

Example : File System in Operating Systems, Directories are composite and files are leaves. System call Open is single interface for both composite and leaf.

**Decorator** - Gives additional feature to objects, while giving unified interface.

Example : 1) Adding **d**iscounts on an order 2) gun is a deadly weapon on it's own. But you can apply certain "decorations" to make it more accurate, silent and devastating.

* All subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor taking an instance of same type.

**Facade** - Single interface that represents entire subsystem.

Example : Control Panel, Event Manager.

* javax.faces.context.ExternalContext, which internally uses ServletContext, HttpSession, HttpServletRequest, HttpServletResponse, et

**Adapter** - Provides different interfaces for an interface.

Example : Power Adapters

* java.util.Arrays#asList()

**Flyweight** - A fine-grained instance used for efficient sharing

Example : *The Flyweight uses sharing to support large numbers of objects efficienthttps://refactoring.guru/design-patternsly*. The public switched telephone network is an example of a Flyweight. There are several resources such as dial tone generators, ringing generators, and digit receivers that must be shared between all subscribers. A subscriber is unaware of how many resources are in the pool when he or she lifts the handset to make a call. All that matters to subscribers is that a dial tone is provided, digits are received, and the call is completed.

* java.lang.Integer#valueOf(int) (also on Boolean, Byte, Character, Short and Long)

# **Behavioral**

This design patterns specially concerned with communication between Objects.

**Chain of Responsibility** - A way of passing a request between a chain of objects

Example : Loan or Leave approval process, Exception handling in Java.

* javax.servlet.Filter#doFilter()

**Iterator** - Sequentially access the elements of a collection

Example : Next/Previous buttons on TV

* All implementations of java.util.Iterator & java.util.Enumeration

**State** - Alter an object's behaviour when its state changes

Example : A fan wall control

**Observer** - A way of notifying change to a number of classes

Example : Bidding or auction

* Publish/Subscribe JMS API

**Visitor** - Defines a new operation to a class without change. Example : Taxi

**Template** - Defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.

Example : A blue print

* All non-abstract methods of java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer.
* All non-abstract methods of java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap.
* javax.servlet.http.HttpServlet, all the doXXX() methods by default sends a HTTP 405 "Method Not Allowed" error to the response. You're free to implement none or any of them.
* JMSTemplate HibernateTemplate and JdbcTemplate in Spring

**Command** - Encapsulate a command request as an object

Example : The "Guest Check" at a diner is an example of a Command pattern. The waiter or waitress takes an order or command from a customer and encapsulates that order by writing it on the check. The order is then queued for a short order cook. Note that the pad of "checks" used by each waiter is not dependent on the menu, and therefore they can support commands to cook many different items.

* All implementations of java.lang.Runnable

**Memento** - Capture and restore an object's internal state

Example : save the state in a game & Undo/Redo operation in Windows

* All implementations of java.io.Serializable

**Mediator** - Defines simplified communication between classes

Example : Air Traffic Controller(ATC)

**Strategy** - A Strategy defines a set of algorithms that can be used interchangeably.

Example : Modes of transportation

* java.util.Comparator#compare(), executed by among others Collections#sort().
* javax.servlet.http.HttpServlet, the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them (and not to get hold of them as instance variables!).
* javax.servlet.Filter#doFilter()

[Java implemented Design Patterns](https://stackoverflow.com/a/2707195/1697099)

[Design Pattern with Simple examples](https://refactoring.guru/design-patterns)

﻿