|  |  |  |
| --- | --- | --- |
| **No.** | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| 2) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| 5) | In java, method overloading can’t be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. no.** | | **Key** | **final** | **finally** | **finalize** |
| 1. | Definition | | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| 2. | Applicable to | | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| 3. | Functionality | | (1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited. | (1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| 4. | Execution | | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.  It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed. |

Method vs Constructor:

* Method can be Override but Constructor cannot.
* Method may or may not has return type, constructor is void.
* Method can be static or non-static, constructor is not.
* Method doesn’t need to have same name as that of class, but constructor has to be.

23)Explain the situation when finally block will not run?

Ans. If before finally block we run an infinite for loop , then we will not reach finally block.

If we write System.exit();

| Sr.No | Compile Time Polymorphism | Run time Polymorphism |
| --- | --- | --- |
| 1 | In Compile time Polymorphism, the call is resolved by the compiler. | In Run time Polymorphism, the call is not resolved by the compiler. |
| 2 | It is also known as Static binding, Early binding and overloading as well. | It is also known as Dynamic binding, Late binding and overriding as well. |
| 3 | Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type. | Method overriding is the runtime polymorphism having same method with same parameters or signature, but associated in different classes. |
| 4 | It is achieved by function overloading and operator overloading. | It is achieved by virtual functions and pointers. |
| 5 | It provides fast execution because the method that needs to be executed is known early at the compile time. | It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime. |
| 6 | Compile time polymorphism is less flexible as all things execute at compile time. | Run time polymorphism is more flexible as all things execute at run time. |

Compile Time Polymorphism vs. Run Time Polymorphism

Compile Time Polymorphism: Whenever an object is bound with their functionality at the compile-time, this is known as the compile-time polymorphism. At compile-time, java knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding. Compile-time polymorphism is achieved through [method overloading](https://www.geeksforgeeks.org/overloading-in-java/). Method Overloading says you can have more than one function with the same name in one class having a different prototype. Function overloading is one of the ways to achieve polymorphism but it depends on technology that which type of polymorphism we adopt. In java, we achieve function overloading at compile-Time. The following is an example where compile-time polymorphism can be observed.

Run-Time Polymorphism: Whenever an object is bound with the functionality at run time, this is known as runtime polymorphism. The runtime polymorphism can be achieved by [method overriding](https://www.geeksforgeeks.org/overriding-in-java/). [Java virtual machine](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) determines the proper method to call at the runtime, not at the compile time. It is also called dynamic or late binding. Method overriding says child class has the same method as declared in the parent class. It means if child class provides the specific implementation of the method that has been provided by one of its parent class then it is known as method overriding. The following is an example where runtime polymorphism can be observed.

# THIS:-

1) “this” is a special reference variable or a Global variable that holds object address.

2)When an object is created, it’s address is to a reference variable, and also “This” is created automatically, which points to OR holds the object address.

3)Reference variable is created by Developers, but this() is created by compiler, at the time of creating Objects.

4) We can declare “this” in the static block, but it’s a wrong way.compiler auto-corrects this to class\_name like “this.x” to “class\_name.x”.

5)”this” cannot be used to call a constructor from inside non-static and static method.

6)we can also call constructor , using “this”, but that call from another constructor only.

8)this should be the first statement inside constructor.

# POLYMORPHISM:-

1)Depending on the situation, it can take more than one form.

Has 2 category:-

a)Method Overriding

b)Method Overloading

Method Overriding:-

here we create a method in parent class and then inherit it.

In child class we create another method with same name as that of parent but modify its logic by creating another method in child class

# CONSTRUCTOR:-

1)Constructor should have same name as that of class

2)Constructors are internally by Default void in nature, so no return type.

3)Constructors are automatically called when we create objects inside it.

4)Constructors cannot be inherited.

5)Constructor cannot be Overrided as inheritance is not possible.

6)If we don’t create constructor explicitly, then compiler by default creates a Non-args Constructor

7)We can have as many constructors as want inside class, provided, they should have either different no. of Arguments or Diff. types of Arguments,thtswhy we achieve overloading here and not overriding as cannot be inherited.

# CONSTRUCTOR OVERLOADING:-

1)Overloading means involvement of more than one constructor but with different no. of argument or diff. types of Args.

# DEFAULT CONSTRUCTOR:-

1)when an object without argument is created, and we don’t create constructor, then compiler creates a .class file.

## 2)While compiling it automatically creates an empty nO-Args Constructor in .class

3)Compiler creates Default constructor only when there is no arguments passed while creating objects.

//constructor overloading

class overload{

//in default constructor we can have empty message or content inside it, but we cannot have any arguments.

overload(char x){

System.out.println();

}

overload(int x){

}

main(){

}

### **Example of Static and Dynamic Binding**

public class FastFood {

   public void create() {

      System.out.println("Creating in FastFood class");

   }

}

public class Pizza extends FastFood {

   public void create() {

      System.out.println("Creating in Pizza class");

   }

}

public class Main {

   public static void main(String[] args) {

      FastFood fastFood= new FastFood();

      fastFood.create();

      //Dynamic binding

      FastFood pza= new Pizza();

      pza.create();

   }

}

**Can we declare an interface final or not?**

**Ans. No, we can not declare interface as final** . Interface in Java is similar to a class but it contains only abstract methods and fields, which are final and static . Since all the methods are abstract ; therefore, we cannot instantiate interface.

No, an abstract class can not be final in Java. Making them final will stop the abstract class from being extended, which is the only way to use an abstract class. They are also opposite of each other, abstract keyword enforces to extend a class, for using it, on the other hand, [final keyword](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) prevents a class from being extended.

In real-world also, abstract signifies incompleteness, while final is used to demonstrate completeness. The bottom line is, you can not make your class abstract and final in Java, at the same time, it’s a compile-time error.

**What is blank final Variable?**

**Ans.**

**class** Test

{

    // We can initialize here, but if we

    // initialize here, then all objects get

    // the same value.  So we use blank final

**final** **int** i;

    Test(**int** x)

    {

        // Since we have initialized above, we

        // must initialize i in constructor.

        // If we remove this line, we get compiler

        // error.

        i = x;

    }

}

**What is Final Parameter?**

**Ans.** public class Test{

public void sample(final int data){

data =600;

System.out.println(data);

}

public static void main(String args[]) throws Exception{

Test t = new Test();

t.sample(500);

}

}

**Advantage of Java inner classes:**

* Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class,** including private.
* Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
* **Code Optimization**: It requires less code to write.

### **Difference between nested class and inner class in Java**

An inner class is a part of a nested class. Non-static nested classes are known as inner classes.

### **Types of Nested classes**

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

* Non-static nested class (inner class)
  1. Member inner class
  2. Anonymous inner class
  3. Local inner class
* Static nested class

## What IS JDK?

**Ans**. Inside JDK there is a compiler which is used to convert a A.java file to A.class file. Compiler also converts the code into ByteCode and then checks for errors.

What is JRE?

Ans. JRE is also included in JDK, and is used to provide a runtime environment for the program.

Class Upcasting:-

When we are creating a parents reference variable and storing child’s object address inside it, that’s called Upcasting. A small man wants a big house, like that a parent class will love to store child class

Parent p = new child();

Class Downcasting:-

When we are creating a child’s variable and storing parent’s object address inside it, that’s called Downcasting.When a bigger scope is stored into smaller scope, that’s called Downcasting.

Ex. while using a method “.readObject()” in ObjectOutputStream, we have to store supreme object class inside “sub class A”. So, something bigger content is stored into smaller container, which is done forcefully, so Downcasting.

Child c = new Parent();

Advantage of “this” :- As a1 is a reference variable storing object address created inside main method, so it’s local variable, so cannot be used inside other methods, whereas, we can use “this” in place of reference, because “this” points to current object address, and is like global variable, so “this” can be used inside other methods in place of reference variable.

## Advantage of Interface:-

a)It helps developers to achieve multiple inheritance

b)When we are defining a method name, before using it, then it helps us in code readability, and we can easily know what kind of methods are we using, its like we write the name of food menu in a template name Interface, and we present the original food by modifying in some other class.

**1) Can we have both abstract and static method in interface?**

Ans. No, abstract is incomplete, so we have to complete it in child class, BUT , if we declare that incomplete method as static, then we cannot override it in child class, so it shows error.

2)can we create incomplete method in interface?

3)can we override a static method?

4)why main method is static?

**Ans.** The main() method is static **so that JVM can invoke it without instantiating the class**. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the main() method by the JVM

**5) why constructor cannot be overridden?**

**6)Why run() method of Thread class is overrided in child class?**

**Ans.** It is highly recommended to override run() method **because it improves the performance of the system**. If we don’t override Thread class run() method in our defined thread then Thread class run() method will be executed and we will not get any output because Thread class run() is with an empty implementation.

**7)why Super Keyword cannot be placed in Static methods? OR Why can’t static method access ‘this’ or ‘super’ in Java?**

**ANS.** Now, **this**keyword in java is a **reference variable** that refers to the **current object**. Also the **super** keyword in java is a **reference variable** which is used to refer immediate **parent class object**.

So, we can say that **this**and **super** both keyword are **reference variable** that refers to some object. In other words these both keywords belong to instance of the class.

Whereas, **static method** belongs to the class than instance of the class. And so **static method** can’t access **this** and **super** keyword in java.

**7)Can Super be used to call static methods of parent?**

Ans. Yes, it’s possible, but in a wrong way. Compiler is smart enough to change the calling statement from super.xx() to className.xx() just like it does for this, when this() is used to call static method.

**8)Can Super keyword supported in SIB?**

Ans. No, as super is related to OOP, as it’s a reference variable. So SIB don’t support Super().

**9)Why in constructor two Super Keywords cannot be used?**

Ans. Because , we know Super keyword cannot be a second statement, so when we write the second super(), then it definitely shows error.

**10) can we call a non-static from another non-static method without “this” keyword?**

Ans. Yes, but it’s a wrong way. Compiler by default puts a this keyword with the method calling statement.

## Ex. public void test() {

**this**.test2(**this**.x);

test2(**this**.x);//when we don’t use this inside a non static method to call another non static method, then compiler by default puts a “this” with the method.

}

**public** **void** test2(**int** val) {

// this.test();//if we call again the test function from here, then we get StackOverflow error

}

**11) Can we create constructor inside abstract class?**

**Ans.** Yes, we can create constructor, but its useless when defined in an abstract class because abstract class contains incomplete methods. And we know, due to the presence of incomplete methods we cannot create it’s object. So if we don’t create object we cannot call constructor.

**12) constructor or IIB should be used for initialization?**

**Ans. Instance Initialization Blocks** or IIB are used to initialize instance variables . So firstly, constructor is invoked and the java compiler copies the instance initializer block in the constructor after the first statement super(). They run each time when object of the class is created.

**13) what happens when we write start() method?**

**Ans.** When we use start method then a new thread is created and then code inside the run method will be executed for each new Thread.

Use of start method creates two stack for each thread ,Stack and native stack.

But Run method call just execute the code inside the run method sequentially as run method call does not create different stacks.

**14) The synchronization is mainly used to:-**

* To prevent thread interference.
* prevent consistency problem.

Java Synchronization is better option where we want to allow only one thread to access the shared resource or same module like “balance”, where two Threads operating on same module ,i.e., balance.

**15)How to get an ID of a current running Thread?**

Ans. Public void run(){

System.out.println(Current Thread Name: “+ Thread.currentThread().getName());

**}**

**16)** **Why static is declared in main method in java?**

**Ans.** Java main() method is always static, **so that compiler can call it without the creation of an object or before the creation of an object of the class**. In any Java program, the main() method is the starting point from where compiler starts program execution. So, the compiler needs to call the main() method.

**17) Why constructor cannot be overrided in java?**

**Ans.** Constructor Overriding **is never possible** in Java. This is because, Constructor looks like a method but name should be as class name and no return value. Overriding means what we have declared in Super class, that exactly we have to declare in Sub class it is called Overriding.

**18) Use of functional interface in java?**

**Ans.** A functional interface is an interface annotated with @FunctionalInterface annotation and contains only one abstract method, but the interface can have multiple default methods.

Lambda expressions are anonymous,which means which don’t have a name and no access Modifier and can be used to complete the incomplete abstract method in Interface.

**Types of Functional Interface in JAVA?**

**Consumer:** The Consumer interface of the Functional Interface is the one that accepts only one argument. The Consumer surface has no return Value.

Consumer<Integer> consumer = (value) -> System.out.println(“Hello”);

**Predicate:** It’s a Functional Interface which accepts a single Value or argument and does some sort of processing on it, and returns a Boolean value.

Predicate predicate = (value) -> System.out.println(“Hello”);//Using Lamda’s Expression

Public interface Predicate<T>{

Boolean test(T t);

}

**Function:** It receives only a single argument and returns a value after the required processing.

Public interface BitFunction<T, U, R>{

R apply(T t, U u);

}

**Unary Operator** extends Function Interface.

**Binary Operator** extends Bi-Function Interface.

**Supplier:** This Interface doesn’t take any input or argument and yet returns a single output. It’s used in lazy generation of Values. It’s also used for defining the logic for the generation of any sequence. Ex. Logic behind Fibonacci Series using Stream.generate() method.

Public interface Supplier<T>{

T.get();

}

**19) why static method cannot be overridden in java?**

**Ans.** Static methods cannot be overridden **because they are not dispatched on the object instance at runtime**. The compiler decides which method gets called. Static methods can be overloaded (meaning that you can have the same method name for several methods as long as they have different parameter types).

**20)Use of methods in JAVA?**

Ans. A method is a collection of statements that perform some specific task and return the result to the caller. We can reuse those method modules also.

# 21)W[**hat exactly does this do Class.forName(“com.mysql.jdbc.Driver”).newInstance();**](https://stackoverflow.com/questions/15039265/what-exactly-does-this-do-class-fornamecom-mysql-jdbc-driver-newinstance)

**Ans.** The Class class is located in the java.lang package, so it is distributed with java, and imported automatically into every class.

What the forName() method does, is just return the Class object for the paramater that was loaded by the class loader. The newInstance() method then returns a new instance of the class.

So then what happens is you call Class.forName(...) it returns com.mysql.jdbc.Driver.class. You then call newInstance() on that class which returns an instance of the class, with no paramaters, so it’s basically calling new com.mysql.jdbc.Driver();.

**22)What are Integrals?**

Ans. Integral means all char, int, long are non decimal values and are converted into (int) by compiler.so Downcasting also it shows error. If we write L with the value then error is removed.

Now if short s =(s)12345; **short** s=(**short**)12345;

In java if we write inside “if” block as shown, if(10){} will show error, because inside if(true) && if(false) is only supported in java. If we write if(10){}, then it shows “cannot convert from int to Boolean”.

So 12345 is considered as an int type, so this integral part doesnot show error. Now when

**23))What are session variables?**

Ans. A Session is **a period of a user’s interaction with the server**. Whenever a user accesses any page of the server, then the server creates session for the user. ... A servlet can use the session of user to create some variables. These variables occupy server memory. Users cannot deny creation of session variables. These session variables are like passbook or own identity of an user, and using which one can user can travel from one page to another.

**24)Can Wrapper class be used to store values of “var” inside objects?**

**Ans.** No, because first of all, it’s a type, and not a **datatype**. Secondly var has no Wrapper class, to store values of **var** inside an object, just like other data types has.

**25) Difference between Throws and Throw?**

**Ans. “**Throw” is used to build our own customized Exception, if may arise relating to our code, as per the requirement of the Developer, but we should make the customized Exception class extends the “Throwable” class, whereas,

“Throws” is used with only methods to catch the exception arised in the method block, if any, and throwing it to its method calling statement.

**26) Can we compile a program without a main method?**

Yes You can compile and execute without main method By using static block. But after static block executed (printed) you will get an error saying no main method found.

* jvm loads class
* executes static blocks
* looks for main method and invokes it

So, if there's code in a static block, it will be executed. But there's no point in doing that.

How to test that:

public final class Test {

static {

System.out.println("FOO");

}

}

Then if you try to run the class (either form command line with java Test or with an IDE), the result is:

FOO  
java.lang.NoSuchMethodError: main

##### Concurrency and Parallelism

In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution. Concurrency indicates that more than one thread is making progress, but the threads are not actually running simultaneously.

The switching between threads happens quickly enough that the threads might appear to run simultaneously.

In the same multithreaded process in a shared-memory multiprocessor environment, each thread in the process can run concurrently on a separate processor, resulting in parallel execution, which is true simultaneous execution. Only then simultaneously or parallelly two tasks are done.

When the number of threads in a process is less than or equal to the number of processors available, the operating system’s thread support system ensures that each thread runs on a different processor.

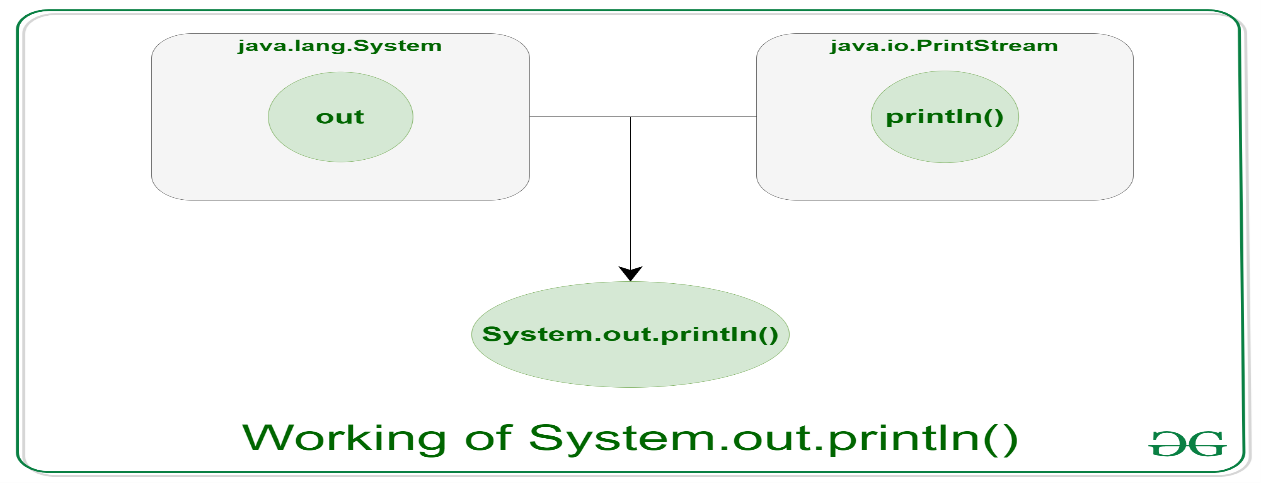
26) What is abstraction?

Ans. It hides the Implementation details. Now, we can achieve 100% abstraction ,creating incomplete methods by using Interfaces. We can also use Abstract class to create abstract incomplete methods.

**EXPLAIN “System.out.println”?**

Java **System.out.println()** is used to print an argument that is passed to it. The statement can be broken into 3 parts which can be understood separately as:

* [System](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** It is a final class defined in the [java.lang package](https://www.geeksforgeeks.org/java-lang-package-java/).
* **out:** This is an instance of [PrintStream](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/)type, which is a public and static member field of the [System class](https://www.geeksforgeeks.org/java-lang-system-class-java/).
* [println()](https://www.geeksforgeeks.org/difference-between-print-and-println-in-java/)**:** As all instances of [PrintStream class](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/) have a public method println(), hence we can invoke the same on out as well. This is an upgraded version of print(). It prints any argument passed to it and adds a new line to the output. We can assume that System.out represents the Standard Output Stream.



**THREADS:**

Using threads means we cannot predict which thread will run, so any thread can execute

**Why main() method is public and static?**

Static methods belongs to the class and they are the first ones to be executed. So, main method is something which is called by compiler first, even before creation of object. So, we declare static.

If we don’t declare a method static then, whenever we will create object, copy of the main method will be passed inside the object. But, we want main method only one time & first time. So multiple copies of main method will cause wastage of memory.

###### ANONYMOUS CLASS:

Anonymous class are that type of class which have no name. these classes are created inside another class. we denote anonymous class by writing flower brackets “{ }”

2)Thead t1 = new Thread({ });

here how will the anonymous class run or execute?

Ans. Here when we will create thread object then Anonymous class is executed automatically.And then we will get the .class file for this anonymous class.

3)Now we used to implement runnable interface for main class by writting “Class A implements Runnable”.

Now if we want to implement the same Runnable Interface for Anonymous class also, then we write it like “new Thread( new Runnable(){ }); “. When we write this, that means we are implementing Runnable Interface to the Anonymous class. We cannot use “implements” keyword with Anonymous.

Now inside the curly brackets we cannot keep it empty, otherwise shows errors, because we have an incomplete run() method inside Runnable Interface, which has to be overrided with @Override Annotation.

Thread() - >

Here each thread gets its own module or run() method.

Now each thread executes its run() method when it’s chance comes, but for small duration. So, each module is executed by it’s own Threads for sometime and switches to the other. Here we cannot predict which thread will run first.

**join() ->** Earlier we cannot predict which thread runs first. So, to solve this problem we introduce join() method. Using join() method, we get a special power by which we can control which Thread will run first and and which Thread will run second. And we can decide the order of those threads. To refer code, go to Eclipse->JoinInternalWorking.java

**synchronizable() ->** Earlier Thread can be executed in sorted order, but stillusing join() also each task was not done completely. Switching between modules still exist. Now if we make a method synchronizable means the first thread to enter this block has to execute and complete all lines of code inside that method and then exit and make other threads to run.

We thus avoid data corruption.How? Earlier two threads, simultaneously fighting for executing a non synchronized methods sometime leads to data loss or corruption. But synchronizable Method force the threads to execute a method completely, called from inside ‘run()’ and then next Thread starts executing.

**Join() + synchronization Method ->** Using synchronization() we can force any Thread to run() completely. But we cannot predict which Thread will enter the synchronized block first and starts executing. So, to control and decide that, we use join() + synchronize block together. Using join() we decide which thread will start first, and using synchronize block we force that Thread to execute the method completely, and then next thread starts.

**Real Life Example -> S**uppose we have a module or method balance(), where some value is stored in balance. Now we create two methods debit() and credit(). Now debit() will decrease the balance and credit() method will increase the balance.

But if suppose we have balance zero. Now, if we use thread without synchronize + join(), then if debit() is first started by thread then it will decrease the balance to some negative values which will lead to data corruption.

To solve that we first use join() method to make that Thread runs first which contains add() method. And now if add() method is declared synchronized then that method will be executed completely. And then second thread is made to run and debit() method inside it is executed. This debit() is also declared synchronised to make that block run completely.

Otherwise balance will not be credited or debited properly if not synchronised.

Another example of Tea making Example by Pankaj Sir:-

Sir told that if we use join() with Thread and operates on same module, then the second thread starts from that place where first thread stopped executing.

So, thus sir’s example of making tea is proved. Where tea is our same module and thread are two boys operating on the same module(tea). Now if 1st Thread started boiling the water and went two sleep mode-> the 2nd Thread adds tea leaves and sugar and goes to sleep -> Now 1st Thread wakes and adds milk to it and again sleeps. So, in this way, where 1st thread stopped working, 2nd Thread starts from there. **Refer to code :** TeajoinThreadsOperatesSameModuleAdd.java in threadCodes,

Where two Thread operated on same module balance().

**But sometime data corruption happens, as some datas are not printed or missed or interchanged. S that leads to data corruption. Thatswhy synchronized block is used with join.**

Which means, one Thread(boy) makes is own tea first instead of waiting for other Thread to do the task. Ins simple word self independent. Each Thread has to do its task or complete module independently.

notify()

notify all()

if I make the method synchronized means the complete method or block of that code is synchronised or this to be executed by thread.

but “Synchronize Block” means if inside a method if we want to make a small part of method to be synchronised, then we use Synchronise Block.

wait() is a method which belongs to Object class and it’s kept inside that thread which we want to pause or wait. Now make the other Thread to run first and complete the execution, and then the thread containing the wait() method will start.

When multiple wait() is declared for multiple Threads, then Notify() method randomly choose which Thread to resume first.

Wait() cause the Thread to sleep until notify() method is called.

**INTER-SERVELET Communication. How do you perform that?**

Ans. This means Servlet 1 will communicate with servelet2 and return some data.

setAttribute() ->in java is used to set some values. getAttribute() is used to get the values in the second Servlet.

Now again if we wat to go to 3rd Servlet, then in second Servlet we have to set the value, and in 3rd Servlet we get the values.

form -> 1st Servlet -> 2nd Servlet -> final html page , get

### **What is Prepared Statement?**

* Prepared statements are little faster in terms of execution when compared to our traditional approach using create statement.
* Prepared statement prevents SQL injection attacks. How? by dividing the queries into two parts. First a query is set using “con.prepareStatemnt(‘sql query with “?” in place of values’)”. And the values ,like ‘email’,’city’ , are later set dynamically from outside using ‘setString()’ methods. So, as the values of queries are later set dynamically from inside the code but outside sql Query , using “getString(int, String)” method . Sql Query is present inside PreparedStatement(“sql query”) in form of argument. So, hackers cannot inject sql codes from URL, as they has to be set from inside JAVA code, after declaring the pre-compiled query.
* setString(int,String);//here the first parameter is index where the value is to be placed In query.
* Values are dynamically injected into the memories.
* PrepareStatement is said to be pre-compiled. Because PrepareStatement compiles the SQL statement only once in the first time when we run the file. Next time no compilation takes place, only runs directly. Compilation only one time, runs multiple times. So time is saved which was earlier needed for every compilation.

###### **JSP LIFECYCLE**

A jsp file consists of html and java code. Now tomcat reads the JSP file but picks only the JAVA code. The java code of JSP file is converted into a Servlet file behind the scene. Now, it means what ever code we run in a JSP file, everything gets converted into a Servlet. Then the Servlet has already 3 methods: init(), service(doGet(), doPost()), destroy(). \_JSP(init()) is executed first and It will run only once after tomcat is started. Then \_jspServlet() method runs. When finally \_jspDestroy() methods runs, then jsp lifecycle comes to an end.

###### **THREAD PRIORITY**

### **Can we developer decide which thread will run first by using concept of priority?**

**Ans**.No, we cannot decide. If we give a particular thread maximum priority, still we don’t have 100% chances that Thread will execute first. Among multiple Threads, the Thread with highest priority gets a chance to send a request to the Thread Scheduler. Now it depends on Thread Scheduler that it will execute it or execute some other Threads.

Real-life example of Thread Priority: Now among multiple students(Thread) in our class, we have one topper who is very reputed among teachers and gets highest Priority. Now he went to principle(Thread Scheduler) and request for holidays. Though he is topper with highest priority, but still it depends on the principle that he will give holiday to him or to some other Students(Threads). NOW RELATE.

Thread-Priority: It decides which Thread is going to run first and which Thread will run later. If we set the priority then it is a request made to the Thread scheduler where there is no assurity that it will be processed and approved.

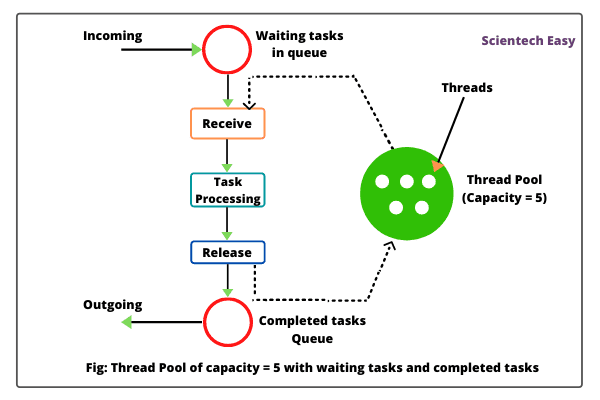
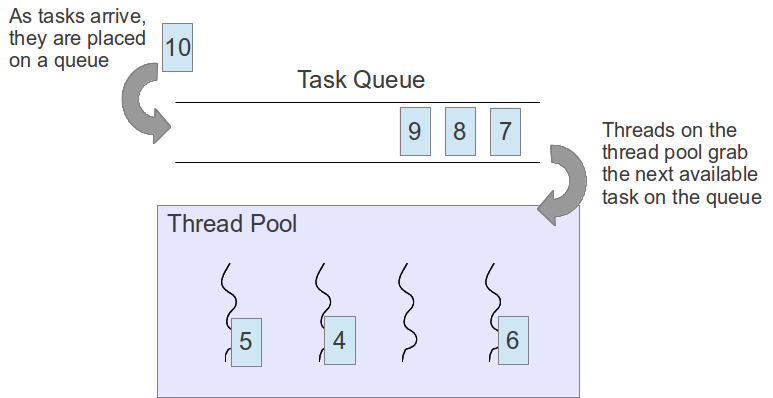
The minimum Thread priority is one. Maximum priority of Thread is 10, and the normal Thread priority is 5. However we can set the Thread priority with a number anything between 1 to 10.

###### **THREAD NAMING CONVENTION:**

**Suppose** there aretwo Threads , and one is used for adding and the other is used for subtracting the balance. Now while we run it, then we should knowthe name of **which Thread is running** at that **time** and what **amount** is it deducting from the balance module.

So, we use t1.setName(“Adding Thread”), to set the name of thread, and now to get the name of thread and print it, we use, S.O.Pln(t1.getName()).

###### **THREAD POOL:**



Here we have a pool of multiple Threads containing **1000 Threads** suppose. Now I have a server where my application is hosted, suppose the application is “Naukri.com” which is hosted in the “Application Server”.

Now, suppose there are multiple Web Browsers which are opened in our local machine. Now, we will send some kind of Request from all these browsers to the application Server(Naukri.com) and then these requests enters a queue one by one and the first **request** to enter the queue is assigned with a Thread for executing the task. Now, every time server cleverly picks one **Thread** from the **Thread Pool** and assign that **Thread** with one of the Browser request one by one to process the **Request**. And after the Task is completed, these Threads are freed and returned back to the pool.

Here the **time taken** to create a Thread and destroy a Thread is **saved**, because the Thread is already created in the Thread Pool, and we just have to assign it to every Browser Request, and after its work is done, we just return it instead of destroying. And thus performance is improved.

Whenever we are dealing with servers, the best performance of the servers happens only when we keep switch on for a very long time. Because if we close the application Server, then all the threads in Thread Pool are killed. Now when we will start our Server again, then all these Threads of Thread Pool are again created and loaded into Thread Pool for handling Request. So, its best not to close server much.

A Thread pool has 1000 Threads means it can handle 1000 reqests at a time not more then that, like “**ZOMATO”** a booking app, where Thousand users can request for booking meals at a time. It cannot handle more then 1000 request, otherwise it will show “The Server is too busy, please connect after sometime” and if I don’t do that and still make request, then the Server might crash.

Thus another advantage of Thread is we can control the no. of Threads in the application Server depending on the user visit rate to the site. Like **ZOMATO Application will have more requests as compare to any government application.**  So, more Threads in ZOMATO then Government application.

**Advantage:** We use Thread pools when we need to limit the number of threads running in your application at the same time. So if Naukri Server need only 1000 Threads in Thread Pool, then there is no use of creating infinite Threads in **Thread** Pool when we know how many Threads our **Application Server** needs.And also creating huge amount of excess threads also takes time. So, this is **another Advantage of Thread Pool**, that we can control the no. of **Threads**  and create only that amount of Threads that will be needed by Application Server of **Naukri**.

**Another Advantage is** it will help us to improve the performance of your application,i.e, instead of starting a new Thread for every tasks to execute concurrently, the tasks can be assigned to Threads fetched from the Thread pool.

A **thread pool** contains a collection of **Threads**. As soon as, the **pool** has any **idle Threads**, the tasks is assigned to one of them and executed.

Thread pool are often used in Servers. Each connection arriving at the server via the network **Http** is wrapped as a task and is passed on to Thread Pool. The Threads in the Thread pool will process the request on the connections concurrently assigning a Thread to every request. This is how we can use existing Thread instead of creating a new Thread and thereby improve the performance in terms of Execution.

# **Thread Scheduler in Java**

A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**. In Java, a thread is only chosen by a thread scheduler if it is in the runnable state. However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones. There are some criteria that decide which thread will execute first. There are two factors for scheduling a thread i.e. **Priority** and **Time of arrival**.

**Priority:** Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

**Time of Arrival:** Suppose two threads of the same priority enter the runnable state, then priority cannot be the factor to pick a thread from these two threads. In such a case, **arrival time** of thread is considered by the thread scheduler. A thread that arrived first gets the preference over the other threads.

**What is a thread?**

Ans. A **thread** is a basic execution unit which has its **own program counter, set of the register, stack** but it shares the code, data, and file of the process to which it belongs.

**ENUMS:**

Enums is a special type of class and keyword too, are collection of constants. In Java, we can also add variables, methods and constructors to it.

We don’t use Enums much, but there are some built-in features where Enums is used.

* Enum declaration can be done outside a Class or inside a Class but not inside a Method.
* In Enum first line should be having all the constants or fixed values, and then we can have constructors or methods.
* Enum in java are type safe.
* Enum can contain main methods, but cannot be used to create Objects.
* You can define Constructor in Java Enum.
* Enum constants are implicitly static and final.
* Enum can implement Interface in Java.
* Enum has implicit values() method which return all Enum constants in array.
* You can use Enum in switch case like int, String data type.
* You can compare two Enum constant using both == and equals() method.

**Use->**

* We use Enum when we create some dropdown or calendar or some other menu in our website where values will be constant or fixed after compilation, and will not change.

**WRAPPER CLASSES**

Primitive DataType -> Wrapper class of the Data type

byte -> Byte

short -> Short

int -> Integer

long -> Long

There are 2 ways I can store the data, ways are:-

i)I can use the datatype to store the data. Ex. Int x = 10.

ii) we can store the data using Wrapper class. Ex. Integer i =1;

**Important Points of Wrapper class**

* **Wrapper classes** helps to **store** the **values or literals** inside the **Object**. This **classes** has lot of **built in** methods. And this **built in** methods that the **Integer class** has got, will help to **manipulate** or **modify** the data quiet easily.
* **Primitive data types** cannot be **converted** to **objects**. So these **primitive data-types** like **int, long** are needed to be **converted** to Objects to perform some **special operations** like,
* I)**Storing** the primitive data types in **collection**. But we know **Primitive data types** cannot be stored in **collection**. So that why they are to be **converted** to **Objects** first using **Wrapper Class**. And then they can be **stored** inside **Collection**.
* ii)**In** **Serialization**, we know only **objects** can be **serialised**. Now if we use **Primitive data types** then it wouldn’be possible to **use** for **serialization**. So now **primitive data type** is **converted** into **object** using **Wrapper Class**. Then we **use** it for **Serialization**.
* **String** is a class, and not a **data type**, so its **not a part** of **Wrapper class**, but **String** has same **functionalities** as that of **WrapperClass**. It is used to **create** **objects** as well as **String Literals** and store **values** inside them just like Wrapper class do.
* **Integer.MAX\_VALUE**-> gives the maximum value we can store in the Integer.In one word it gives the range of Integer. **Integer** is **class**, and “**MAX\_VALUE”** is a **non-static** member.
* If we want to **convert** any other **primitive data type (here int)** to **String Object**, we use **Integer.parseInt(String s)** or **Float** to **String** by using “**Float.parseFloat(String s);”.**
* The process of value being stored inside an object is called as wrapping.
* When ever we are reading the value from inside the object, then it is called unwrapping or unboxing.
* Boxing means storing the value inside an object.
* Here the values are stored in objects.The process of storing the value inside the object is called as wrapping or boxing.
* Reading the Value from the object is called as unboxing.

**There are 2 ways to store Integer values inside objects using Wrapper class:-**

* One is bycreating **“Literals”** like :
* **Integer Literals** “Integer i = 1” , **OR**
* **String Literals** “String str = Deba”// Its not a part of Wrapper class, just used to relate the same functionalities
* Another ways is to create **Objects** using **new** keyword like creating:
* **Integer Object** “Integer I = new Integer(10)  **OR,**
* **String Object** “String str = new String(“Deba”);”

**FINALIZE() METHOD:**

**Protected Void Finalize()** is a **non static** method present in **Object class**, and we know by default every **class** is a **subclass** of **Object class**, so need to **extend** that class where finalize() method will be used. And we Override the method of finalize() in the subclass, after inheriting from Object class, and modifying the logic.

Finalize() is special method present inside Object class. Here the logic of Garbage collection is defined or implemented.

**Garbage collection** is operated by **internal implementation** of **JVM**, but still if we want to call that, we write **“System.gc()”.** This calls the finalize() method.

**Why finalize method is used()?  
Ans.** finalize() method releases system resources before the garbage collector runs for a specific object. JVM allows finalize() to be invoked only once per object.

### **How to override finalize() method?**

**Ans.** Now, the finalize method which is present in the [Object class](https://www.geeksforgeeks.org/object-class-in-java/), has an empty implementation, in our class clean-up activities are there, then we have to override this method to define our own clean-up activities.

In order to [Override this method](https://www.geeksforgeeks.org/overriding-in-java/), we have to explicitly define and call finalize within our code.

#### **THROWS KEYWORD:**

Throws keyword is applied on a method only.We can use multiple Exceptions with “throws” keyword.

If any exception occurs in the method, then the exception will be passed on to the calling statement of the method.

Throws keyword is only used with methods and neither class nor variable. Now Work of Throws keyword is whenever any exception arises in the test() method, then “Throws” will throw the exception back to the method calling statement.

Throws is like playing catch catch. When a method block has an exception, if “throws” keyword is used with that method, then it will throw it to its method calling statement.

Now the Exception will be transferred and shown with the method calling statement. Now Suppose this method calling statement is present inside main() method, then we can use “throws” keyword with main method, and this Exception will be gone to the JVM directly, because JVM is the first one to call main method, so indirectly JVM is the method calling statement for main() method.

* One way to handle exception ,which arises when we use “FileWriter”, is by using try catch Block.

**public** **void** test(){

try{

FileWriter fw = **new** FileWriter(“D://test.txt”);//here if we don’t write this line inside try catch block, then it will give compile time Exception.

}catch(Exception e){

Sopln(e);

}

}

* Another way to handle the Exception is by using Throws Keyword. **And we can have multiple Exception class after throws keyword, because it may happen that the method arise multiple Exceptions.** Below is an Example:=

**public** **void** tets()**throws** Exception, SQLException {

FileWriter fw = **new** FileWriter(“D://test.txt”);

}

##### **THROW:**

In Java, there are many exceptions like Null Pointer Exception,etc.,

But Throw helps us to create object of our own Customized Exceptions **class** which is not their in java **and throw it inside “catch“ block**. Now, for some reason we want to create our own Exception but extending that customized Exception class to “Throwable” is mandatory. **A practical scenario is mentioned below**:-

**Real-Life example:-**

Suppose we went to a ATM machine to withdraw money. Now, our balance is 500, and I want to withdraw Rs 1000. So, if we want to do it.Then it should show some kind of Exceptions like “Insufficient Funds”.

Now, we can create this Customized Exception using Throw keyword.

**I as a programmer created our own Exception for this particular business logic using “throw”.**

**Note:** To use “throw” inside a class, then we have to extend that class to parent of all exceptions i.e.,Throwable.

**Importance of try-catch:** Suppose if by somehow our application is crashed, Then we can use “catch” block to restart the app. “catch” has other usage also except suppressing errors.

REGULAR EXPRESSIONS IN JAVA:

**Html Form**

Suppose we create a form containing name, mobile text boxes in html.

Now we have to validate those mobile numbers like:-

* Its 10 digit or not.
* Every mobile No. should starts with 6/7/8/9 only.

**We** also need to validate for “Name” input box like:-

* We should not enter numbers into it.
* We should not enter special characters into it.
* We should enter only Alphanumeric characters into it.

Now for doing this, we need Regular Expressions.

* Pattern p = Pattern.compile();

Ex. Of compile()method,

**public** **static** Pattern compile(String regex) {

**return** **new** Pattern(regex, 0);

}

* Matcher m = P.matcher(“a6b#@z9D”);

Ex. Of matcher() method,

**public** Matcher matcher(CharSequence input) {

**if** (!compiled) {

synchronized(this) {

**if** (!compiled)

compile();

}

}

Matcher m = **new** Matcher(**this**, input);//here Matcher constructor is called below

return m;

}

Matcher(Pattern parent, CharSequence text) {

**this**.parentPattern = parent;

**this**.text = text;

}

**.compile(String regex) ->**Compiles a given regular expression to a pattern. And that pattern “[abc]” is stored inside an object, and is returned to reference “p”. When we print “p”, we get the pattern as value inside it,i.e.,[abc], instead of address.

**p.matcher(“a6b#@z9D”)-> “matcher”** is a special type of method, which takes **string**  as an **Argument,** where internally the given sequence of characters(“a6b#@z9D”), is matched for “**common characters”** with the current object value. And here the current object value is “p” which contains the pattern [abc]. After finding out the common characters , now they are stored into a newly created “matcher” object, which will contain the matching value “a&b” and address is returned to “m” reference.

While(m.find()){

Sopln(m.start()+”…………”+m.group());

}

We have to first check if an alphabet exist in the sequence, where do they exist, if exists, then give me the alphabet and its index number.

**[abc] ->**The [] brackets means, It is searching for the alphabets or contents present inside the square brackets “[]” in the given string. This abc is regular expresiion.

Whatever we write in square brackets, compiler will search for that in the given String.

* [a-z]->Regular expression to returns index of all alphabets present in given string.
* [0-9]->returns index of digits, if present in given string
* [A-z]-> returns index of Uppercase characters.
* [a-z0-9A-Z] ->returns all index of characters ranging between a->z, A->Z, 0->9, in our given string.
* [^a-z0-9A-Z] ->returns everything index except the characters ranging between a->z, A->Z, 0->9, in our given string.
* [^a-z] -> returns everything except the index of all alphabets in that range, present in given string
* [0-9]{3} -> in our String containing phone no. “7003543238” “{3}” will make groups of 3, But those groups which will not meet the minimum requirement will be excluded.{700},{354},{323},will be considered. Group {8} does not contain 3 member, so it will be excluded.

REGULAR EXPRESSION ADVANCED:

[\\s](file:///\\s), [\\S](file:///\\S) , [\\d](file:///\\d), [\\D](file:///\\D) ->

Find out white space characters in a given string->

As single “\” has special meaning like new line. There are more of them. So, thatswhy while defining path we use “\\” and even in regular expression we us”\\s” for removing white space from a given string.

### In [\\s](file:///\\s), Lowercase “s” will be used in Regular Expression to return the index where white space is used in String.

### In [\\S](file:///\\S), Uppercase “S” is just opposite. It will return index of all other characters except the index of “ “(space).

### In [\\d](file:///\\d) , LowerCase “d” will search for only numeric digits from the given String and return it’s index.

### “\\D” is used to return everything except numeric digits in a given String.

### In [\\w](file:///\\w), Lowercase “w” will exclude everything except “Lowercase”, “Uppercase” and “Digits”.

### In [\\W](file:///\\W), UpperCase “W” will return everything **except** “Lowercase”, “Uppercase” and “Digits”.

**\*\*Find and research -> matches(), and, return (m.find() && m.group().equals(str));\*\***

**Some Special Characters and their meaning:-**

* **“a\*”->** Writting “a\*” means it gives us 0 occurences of “a” or group of occurences of “a”, of a particular character which is written with “\*” in Regular Expression.
* **“a+” -> H**ere “a+” will print only for the occurence of it’s own type . Its shelfish, so doesnot donates to the other characters except itself.
* **“a+” -> H**ere “a?” will print for all the occurrences(both “a” and those characters excluding “a”) of characters, independently and not in a group, whether it’s “a” or not a.

**TOKENIZER:**

### Splitting a given string on the basis of a “delimiter” like “space” or “-“ or ”.”,etc,. Ex. There is a string “Hellow Debanjan Sarkar”. On basis of (space)“ “ we can divide the entire String into Tokens like [“Hellow”, “Debanjan”, “Sarkar”] are the 3 tokens, splitted on the basis of “space” present inside the string.

### Now to split the string based on some pattern, we have a built-in class in java called as **“String Tokenizer”.**

**Tokens:-** After splitting a string what words I gate are called as Tokens.

### StringTokenizer str = new StringTokenizer(“hellow deba”) -> In StringTokenizer Constructor, we have two Arguments. First one is the String, and second one is the “Delimiter”. Depending on Delimiter String splitting is done. If we don’t define any Delimiter, then by default “ “(space) Delimiter is considered, and whenever “space” is found, it will split those String in Tokens.

### **Internal Working of StringTokenizer Object**: Firstly constructor of StringTokenizer class is called, and a string argument is passed inside a constructor. Now that String is splitted into “Tokens” when “space” is found and address of an object, containing the tokens is returned.

1. **“str.hasMoreTokens()” -> T**he **str.hasmoreTokens()** in **StringTokenizer class,** checks whether the String contained in “str” reference has any more tokens available, after splitting that String stored in “str” reference.
2. **Parameters Of .hasMoreTokens()** : The method does not take any parameters.
3. **Return Type of** **str.hasMoreTokens()”:** The method returns Boolean **True** if the availability of at least one more token is found in the string after the current position, if not, then false.
4. **Str.nextToken() :-** here **.nextToken()** will read the **Token** for us and print it. If there are 3 tokens, then we have to write this method 3 times. Another way is to use a while loop. Suppose we have a String **“Hellow Debanjan Sarkar”**. Whenever “**str.hasMoreTokens()”** return **true,** loop will run for 3 times, and **“str.nextToken”** will also read 3 times and print the Tokens.

Assignment: print the words independently from a given String without Tokenizer, but by split function and Regular Expression.

**CLONING:**

* **Defination:** Cloning means exact copy of a real object into subsequent multiple clone objects with same properties and behaviours.
* The clone() method belongs to object class.
* To achieve cloning, java.lang.Cloneable (**Marker**)interface should be implemented by that class whose objects will be cloned. If our class implements Cloneable Marker Interface, some instruction are send to the JVM Compiler to give some features to make this class applicable for Cloning.
* If we don’t implement Cloneable interface, clone() method generates **CloneNotSupportedException**.
* **Advantage:** In a class, when we use “new” keyword multiple times, then overall processing of Object creation takes a lot more time by JVM, like every time loading methods, variables into new Objects. It’s a lengthy process. **Solution: Use clone() method,** to create copy of Objects, if we need Object multiple times in our code. This cloning() method Saves extra processing time.
* It is the easiest way to implement cloning even if for age-old projects, you just need to define a parent class, implement Cloneable in it, provide the definition of clone() method and you are ready every child of your parent will get the cloning feature.
* Cloning requires very less line of code, just an abstract class with 4 or 5 line long clone() method if you don’t require deep copy.

###### **DISADVANTAGES:**

* Object.clone() doesn’t invoke any constructor so we don’t have any control over object construction.
* If you want to write a clone method in a child class then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
* We have to implement cloneable interface while it doesn’t have any methods in it. We just have to use it to tell the JVM that we can perform clone() on our object.

**Concurrent Modification:** Concurrent Modification in programming means to modify an object concurrently when another task is already running over it. For example, in Java to modify a collection when another thread is iterating over it. Some Iterator implementations (including those of all the general purpose collection implementations provided by the JRE) may choose to throw *ConcurrentModificationException* if this behaviour is detected.

***Fail Fast And Fail Safe Iterators in Java***

[Iterators](https://contribute.geeksforgeeks.org/iterators-in-java/) in java are used to iterate over the Collection objects. Fail-Fast iterators immediately throw ConcurrentModificationException if there is structural modification of the collection. Structural modification means adding, removing any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes are some examples of fail-fast Iterator.  
Fail-Safe iterators don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes are examples of fail-safe Iterator.  
 Important points of fail-fast iterators : 

These iterators throw ConcurrentModificationException if a collection is modified while iterating over it.

Ex : Iterators returned by ArrayList, Vector, HashMap.

Note 2 : If you remove an element via Iterator remove() method, exception will not be thrown. However, in case of removing via a particular collection remove() method, ConcurrentModificationException will be thrown.

ArrayList<Integer> al = **new** ArrayList<>();

        al.add(1);

        al.add(2);

        al.add(3);

        al.add(4);

        al.add(5);

        Iterator<Integer> itr = al.iterator();

**while** (itr.hasNext()) {

**if** (itr.next() == 2) {

                // will not throw Exception

                itr.remove();

            }

        }

        System.out.println(al);

        itr = al.iterator();

**while** (itr.hasNext()) {

**if** (itr.next() == 3) {

                // will throw Exception on

                // next call of next() method

                al.remove(3);

            }

        }

**Fail Safe Iterator:**

First of all, there is no term as fail-safe given in many places as Java SE specifications does not use this term. I am using this term to demonstrate the difference between Fail Fast and Non-Fail Fast Iterator. These iterators make a copy of the internal collection (object array) and iterates over the copied collection. Any structural modification done to the iterator affects the copied collection, not original collection. So, original collection remains structurally unchanged.

* Fail-safe iterators allow modifications of a collection while iterating over it.
* These iterators don’t throw any Exception if a collection is modified while iterating over it.
* They use copy of original collection to traverse over the elements of the collection.
* These iterators require extra memory for cloning of collection. Ex : ConcurrentHashMap, CopyOnWriteArrayList

**HASHCODE:**

The **hashCode()** method is a [Java](https://www.javatpoint.com/java-tutorial) Integer [class](https://www.javatpoint.com/object-and-class-in-java#class) method which returns the hash code for the given input values. This method overrides the hashCode() of Object class.

Now, using hashCode() gives some 4 byte code of every input values, to identify them Uniquely.

Ex.

String str = “a”;

System.***out***.println(“String Object to HashCode: “+str.hashCode());//97, Returns an HashCode(integer of 4 bytes) of the String input value, here int value of input “a” is ASCII Value, So prints 97 .

Ex.2

Integer i = 10;

System.***out***.println(“Integer Object to Hashcode3: “+i.hashCode());//10, returns int value only, because “10” is itself a 4 byte integer, so it remains unchanged

Ex.4(Here Objects Address, which is Hexadecimal, is converted to Integer Code)

WrapperClass\_Code w1 = **new** WrapperClass\_Code();

System.***out***.println(“Input Value is Object’s Address to Hashcode: “+w1.hashCode());

**Output:** 1392838282

**Explanation:** Here hashCode will return integer representation of memory Address of “WrapperClass\_Code” Object .

Ex.3

//Here we convert the object value “Deba” into its Integer format

String str2 = **new** String(“Deba”);

System.***out***.println(“String Object Value to Hashcode3:”+str2.hashCode());//10336

**ANNOTATIONS**:

**DEFINITION:** The @SuppressWarnings annotation type **allows Java programmers to disable compilation warnings for a certain part of a** program (type, field, method, parameter, constructor, and local variable).So programmers can choose to tell the compiler ignoring such warnings if needed.

Annotation helps us to communicate with the compiler and tell it to do some task on my behalf. Annotations starts with “@”.

Java has many built-in Annotations.

* @Override,
* @SuppressWarnings,
* @Deprecated.

@ SuppressWarnings->

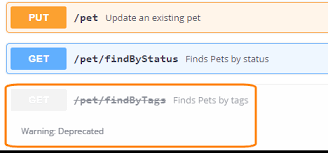
@**SuppressWarnings** can be used in 2 places:

1) 1st, just right on top of the line where warning is shown

2) On top of the whole method block, to suppress all warnings inside method together by just declaring it once.

**(“unused”) -> To suppress warning relative to unused code only like, we have declared a variable, but we didn’t initialized it.**

**@Deprecated ->**

****

Application of Deprecated:

**Suppose** in **Spring Boot**, for example we have a built-in method like “**findmyName**()”. And it’s outdated as the **creator** of the **method** has marked it with **@Deprecated Annotation**. **But** still if **another developer** is using it, then he will get an **Warning**, as he is using or calling an **outdated Deprecated** method inside is code. So, to make **other developers** know that it’s **outdated**, and shouldn’t be further continued with those methods, we use @**Deprecated**.

**Real-Life Example of @Deprecated Annotation: Suppose** a **School** (**Creator**) created ID Cards(**methods)** for students(**Application**). Now after some years **some new ID Cards** are **created** and re-designed with new contents and features.. And then declared the that **New ID Cards** will be used, and **Old ID** cards are **Discarded** and **Deprecated**.

Now Debanjan goes to School, and shows the old ID cards only, it will work, but he will be warned by saying, that there is a replacement of old ID Cards with the new ID’S, and the old ID’s are suspended, and may be phased out in near future. So, not to use the old one anymore.

**NOTE:** A deprecated class or method is like that. It is no longer important. It is so unimportant, in fact, that you should no longer use it since it has been superseded and may cease to exist in the future. Because a class’s API (application programming interface) changes over time and redesigned with new methods or features.

Java provides a special way to express this old stuffs my denoting them with @**deprecated**. Methods are renamed, new and better methods are added, and fields change. However, such changes pose a problem. Because things have updated, and if I still use the OLD ones, then its bad, so in our code if some methods are already built, but that cannot be changed because whole flow of Java will get disturbed. Ins such cases we mark them with @Deprecated, to male other developers know that some methods in the Application is outdated, and should not be continued with that.

**COLLECTION:**



Array is a collection of similar items. Like “int” array will contain only Integer items, && String Array holds String Items.

Int[] x = new int[3] => an array is created of size 3.

**Collection** is an area in which all the **contents(like variable or some tabular data)** are stored in the form of objects.

**DEFINITION:** **Collection** is a **framework** in java, that provides an **architecture** or **container** to store group of **Objects**, and **manipulate** them **explicitly**. **Collection** “Framework” has lot of readily available logic in it so that we can deal with different **DSA** and **achieve** all the **operations** that you perform on a **data** such as **searching, sorting, insertion, manipulation, and deletion**.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet). 143231 112334

**List** :- It’s an **Interface** which is a **subInterface** or **childInterface** of **Collection Interface**. It’s a **list type Data Structure** where we can store a **collection of Objects** in an **order**. It can have **Duplicate** Values also. To **instantiate** the List interface, we must use :

* List <data-type> list1= **new** ArrayList(); //in a list type reference, we store an objects address
* List <data-type> list2 = **new** LinkedList();

**List interface** is implemented by the **classes ArrayList, LinkedList, Vector, and Stack**. There are various **methods** in the **List Interface**, which are used to **manipulate** the **elements** in the List. Like **Inserting** values into **list**, or **deleting**, or **access** the **elements** from the **List.**

ArrayList and LinkList are classes implementing List Interface. In List interface some Incomplete methods are there[Ex. public void add();], 😉 and those incomplete methods are completed in ArrayList and LinkList classes[Ex. public void add(){ } ], Overriding is happening here

😂.

**VECTOR:** Vector uses dynamic Array to store the data elements. It is similar to ArrayList. **Every** **method** in **vector** is **synchronised**. Synchronised means Thread safety, i.e., **only one Thread** can enter the **synchronised** block at a **time**. This reduces the performance and increases time. Because Multiple Threads cannot execute a synchronised block simultaneously. Also contains some methods not part of Collection framework.

**Difference between ArrayList and ARRAY?**

**Ans.** ArrayList is an advanced form of array, by,

* Array is static, sizes are fixed but ArrayList is dynamic because it can expand its size as per requrements, where sizes has no limits.
* Array contains similar type of Items, but ArrayList can contain different types of values, like Boolean, char, string, etc. only if any specific datatype is not mentioned inside generics,
* Array contains values, but ArrayList contains Objects.
* We can manipulate the ArrayList using different **in-built** methods present in List Interface, like adding, removing, but in Array, we have to do this manually.
* Array can hold both Objects and primitive Datatypes, whereas ArrayList can hold only Objects.

**ARRAYLIST:-** **ArrayList** is a **collection** of **objects** containing **values**. Here **primitive datatypes** like**, int, char**, etc, are **not used** to **store values**. Instead, by **default** if we **add** some values to an **ArrayList**, then the values are **converted** into **Objects** using **Wrapper classes**, like **Integer**, **Character**, **Long**, etc., and then **stored** in **ArrayList**. And **ArrayList** is a **Collection**, containing **objects**. So, no place for primitive data-types.

In ArrayList when we write : **ArrayList x = new ArrayList(**) ,then a dynamic array will get created internally to store elements, and the default size of the ArrayList will going to be 10 . But when ArrayList is fully filled and we try to insert the 11th element, then internally a new ArrayList is created with new size, 1.5 time more then previous size. And all the contents of OLD ArrayList will now be copied into the new ArrayList with increased size. This is the process by which ArrayList increases its size dynamically. We can also write **List<Integer> x = new ArrayList<Integer>();//Class Upcasting**

* As it is dynamic, it has no fixed size or limit. We can manipulate the ArrayList externally by adding or removing, or sorting, etc, elements. So, it Is more flexible then traditional array.
* ArrayList can contains duplicate elements also.
* It implements List Interface so that we can use all the methods of List Interface.
* The ArrayList maintain the order, in which the values are inserted in it. It follows Insertion Order.
* ArrayList is not synchronised, So performance is faster then VECTOR.

**Ex. Code for Converting ArrayList to List:**

String[] array={"Java","Python","PHP","C++"};

System.out.println("Printing Array: "+Arrays.toString(array));

//Converting Array to List

List<String> list=**new** ArrayList<String>();//class Upcasting

**for**(String lang:array){

list.add(lang);

}

**DIFFERENCE BETWEEN ARRAYLIST AND LINKLIST?**

**Ans.** ArrayList allows random access to it contents, as it operates on index-based structure occupying contagious memory location, one after another,

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **LinkedList time complexity** | **ArrayList time complexity** | **Preferred** |
| Insert at last index | O(1) | O(1)  (If array copy operation is Considered then O(N)) | LinkedList |
| Insert at given index | O(N) | O(N) | LinkedList |
| Search by value | O(N) | O(N) | ArrayList |
| Get by index | O(N) | O(1) | ArrayList |
| Remove by value | O(N) | O(N) | LinkedList |
| Remove by index | O(N) | O(N) | LinkedList |

* But LinkList doesn’t allow random access to its data, just like ArrayList, as it does not have continuous indexes to access elements directly. It is splitted in different areas or corners in RAM. To access elements in LinkList, we have to traverse the LinkList up to that element, and then only we can access the element.
* But LinkList doesn’t wastes memory just like ArrayList because, LinkList can occupy any space from any corners, but ArrayList picks a continuous memory Location depending on size, leaving small space unoccupied, So wastage of memory occurs in Array List, but memory is saved in LinkList.
* ArrayList can fetch you any element in O(1) complexity as the array has random access property. You can access any index directly without iterating through the whole array.

LinkedList has a sequential access property. It needs to iterate through each element to reach a given index, so time complexity to get a value by index from LinkedList is O(N).

* The difference is that to **remove** an **element** from **LinkedList**, we just need to **modify** **pointers**, which is **O(1)** complexity, but In **ArrayList**, we need to **shift** all **elements** after the **index** of the **removed** value to fill the gap created.

As shifting is costly operation then modifying pointers, so even after the same overall complexity O(N),  we **prefer** **LinkedList** where **more** **delete** by **value** **operation** is **required**.

* ArrayList provides constant time for search operation, so it is better to use ArrayList if searching is more frequent operation than add and remove operation. The LinkedList provides constant time for add and remove operations. So it is better to use LinkedList for manipulation.
* ArrayList has O(1) time complexity to access elements via the get and set methods.
* LinkedList has O(n/2) time complexity to access the elements.
* In sort, ArrayList is better to access data whereas LinkedList is better to manipulate data

**Performance of ArrayList:** We can read or fetch the value from an ArrayList, directly using get(int index) method with O(1) time complexity.

**Disadvantage of ArrayList:** When we try to insert a value in the middle of an array,then we have to shift all the elements by one position to its right, and then insert the new element in the middle.In worst case it will take O(n) time complexity.So time is more, and performance is reduced. Instead use LinkList to insert elements in middle.

**Difference between ArrayList and LinkList:**

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

### **Java Non-generic Vs. Generic Collection**

Let's see the old non-generic example of creating java collection.

* ArrayList list=**new** ArrayList();//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

* ArrayList<String> list=**new** ArrayList<String>();//creating new generic arraylist

**ITERATOR:**

In java, an Iterator, is one of the Java Pointers or cursors. Java Iterator is an Interface that is used to iterate over an List, like ArrayList, LinkList, etc., and printing the values one by one. When we use Java Iterator, then it’s compulsory to create an object of the Iterator Interface first{(Interface itr = x.iterator()}, then use the reference to call inbuilt methods of Iterator Class. It has methods:

* **hasNext() ->** Checks if there exists any next elements in List, if yes, returns true and points to that value now, OR else false. In this way goes on pointing other objects in List, and returns true if value exist.
* **Next() ->** this reads the values in ArrayList, if exists, then points to next Index, and returns the current index value, and we print it.

###### Ex.Code.

###### **int** cursor = 0; // index of next element to return, cursor points to next index

###### **int** lastRet = -1; // index of last element returned; -1 if no such, lastRet points to current Index

###### **public** E **next**() {

###### **int** i = cursor;//initially cursor points to zero, so i=0.

###### **if** (i >= size)

###### **throw** **new** NoSuchElementException();

###### cursor = i + 1; //it’s the next index (0+1=1), the cursor will point to now

###### **return** (E) elementData[lastRet = i];//lastRet=0 initially, always returns the current index

###### }

* **Remove()** -> It’s used to remove the last element of the ArrayList. The next() method should be called once before using the remove() method. Now remove() will remove that index value, which next() method will return currently. We cannot use remove() until we declare next() first. And also we cannot use remove() two time. One remove() for one next() only. Next()can stay alone without remove and be used for returning values, but remove() cannot be kept alone without next().

#### **public** **void** remove() {

#### **if** (lastRet < 0)

#### **throw** **new** IllegalStateException();

#### **try** {

#### ArrayList.**this**.remove(lastRet);

#### cursor = lastRet;

#### lastRet = -1;

#### } **catch** (IndexOutOfBoundsException ex) {

#### **throw** **new** ConcurrentModificationException();

#### }

#### }

* **forEachRemaining(Consumer action)**

**Advantages of Iterator:**

* If a user is working with for-loop, for Traversing the ArrayList, then he cannot use inbuilt add() or remove() methods, because there aren’t any, for loops. But if Iterator is used for Traversing an ArrayList, then while traversing we can use Inbuilt remove() methods of “Iterator” parallelly to manipulate the data of ArrayList.
* Iterator can be used to iterate or traverse maximum classes in the Collection framework like ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet, HashMap, LinkedHashMap, TreeMap, etc.
* As Iterator is fail-fast in nature and doesn’t allow modification of a collection by other threads while iterating, it is considered as safe and secure than Enumeration.

**Disadvantages of Iterator:**

The java Iterator iterates in the forward direction only. So, Iterator Is a uni-directional Iterator.

**Enumeration Vs Iterator In Java :**

Enumeration and Iterator are two interfaces in java.util package which are used to traverse over the elements of a Collection object. Though they perform the same function i.e traversing the Collection object, there are some differences exist between them. Using Enumeration, you can only traverse the Collection object. But using Iterator, you can also remove an element while traversing the Collection. This is the one major difference between Enumeration and Iterator in java. You can say Iterator is some what advanced version of Enumeration.

|  |  |
| --- | --- |
| **Enumeration** | **VS Iterator** |
| Using Enumeration, you can only traverse the collection. You can’t do any modifications to collection while traversing it. | Using Iterator, you can remove an element of the collection while traversing it. |
| Enumeration is introduced in JDK 1.0 | Iterator is introduced from JDK 1.2 |
| Enumeration is used to traverse the legacy classes like Vector, Stack and HashTable. | Iterator is used to iterate most of the classes in the collection framework like ArrayList, HashSet, HashMap, LinkedList etc. |
| Methods: hasMoreElements() and nextElement() | Methods : hasNext(), next() and remove() |
| Enumeration is fail-safe in nature. | Iterator is fail-fast in nature. |
| Enumeration is not safe and secured due to it’s fail-safe nature. | Iterator is safer and secured than Enumeration. |

**LINKLIST :**

### A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the Abstract List class and implements List and Deque interfaces. **By default Internally “Doubly LinkedList” is implemented in Collection.**

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to occur like ArrayList.
* Java LinkedList class can be used as a list, stack or queue.

### **LinkedList class declaration for java.util.LinkedList**

Let's see the declaration class.

1. **public** **class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Cloneable, Serializable

**2 TYPES:**

**SINGLE LIST->** This traverse is uni-directional. A singly linked list is **a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail)**. Each element in a linked list is called a node. A single node contains data and a pointer to the next node which helps in maintaining the structure of the list.

Our Node holds Object Value as well as address of next Node. So**, LinkList occupies more memory than array**.

They are splitted everywhere in the RAM and doesn’t occupies contagious memory location.

Head pointer points to the first element in LinkList. Last node of LinkList points to Null.

When we insert a new node value, then we store the address, head points to the new Node, and head will now point to new node.

DSA DEFINATION: We are forming a structure to store a data in it, so that the retrieval of data or data manipulation becomes easy.

**DOUBLE LINKLIST ->** A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in doubly linked list. This traversal is bi-directional, i.e., from left to right and also from right to left, both.



Here ONLY FOR first node, it’s divided into 3 cells, 1st Cell points to null, 2nd contains Object value, and make head point to it.,3rd cell contains next Node Address, if any, or else null. But starting from Second Node onwards, value changes, like, 1st cell contains previous values address, 2nd cell contains Value Objects, 3rd cell contains next Nodes address.

Last Node of Double LinkList : 3rd cell will contain null.

**Disadvantages of DLL over singly linked list** :  
1) Every node of DLL occupies more memory, as contains Object value as well as two Address. An extra space is needed for the “previous pointer”. It is possible to implement DLL with single pointer though (See [this](https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-1/)and [this](https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-2/)).   
2) All operations require an extra pointer “previous” to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in insertions at different positions of DDL, we need to write 1 or 2 extra steps for making the previous pointer point to previous Node’s address.

3) But if we try to insert in the middle of DDL, then we have to traverse up to that length, and then insert, so performance is bit reduced. Adding node at the end(DoubleLinkList), will take O(n) time Complexity in worst case, because we have to traverse the entire List.

**Advantage of DLL and Disadvantage of DLL:**

* Adding every Node at the beginning of DoubleLinkList will take O(1) Time Complexity.
* DLL is bidirectional so we can traverse forward as well as print LinkList in reverse order.

**Disadvantage of Single LinkList:** But in Single LinkList, inserting Node at the beginning is good with O(1) time complexity, but if we try to insert from end, then we have to traverse the whole LinkList first, up to the end, and then we can insert from back.

**Important points to remember:**

* By default, In Java collection concept, DoubleLinkList is implemented internally instead of Single LinkList due to some advantage like printing LinkedList in reverse Order with DLL. Not possible with SLL.
* **List** is an **Interface**, and it is implemented by ArrayList, LinkList, etc., Now, if we write,

**List<Integer> x = new LinkList<Integer>();//** here inside **parent reference** we store **child’s object’s** **address , so Class Upcasting** occurs. **List** has many **incomplete** **methods** like size(), add(),etc., which are completed in LinkedList and all methods are defined in List Interface, so we use “**List x**” as reference to call such methods. Most of the Methods of LinkList and ArrayList are common except some few **Special Methods**. **Some common methods(same as ArrayList) of LinkedList implemented in “List” Interface are:**

* x.add(10);
* x.remove();
* Collections.sort(x);
* x.size();

**Before implementing some exclusive methods of LinkedList ,we use the reference of LinkList class only, like, LinkedList<Integer> x = new LinkedList<>() because some exclusive methods are present in Linked class only and not “List” Interface. Some Special Exclusive methods to implement LinkList are:**

* **addFirst() ->** It is used insert an element at the beginning of List.
* **addLast() ->** It is used to insert the element to the end of a List.
* **removeFirst() ->** It removes and returns the first element from a list.
* **removeLast() ->** It removes and returns the last element from a list.
* **peek() ->** It fetches the first element in LinkedList
* **peekFirst() ->** It fetches the first element of a list or returns null if a list is empty.
* **peekLast() ->** It fetches the last element of a list or returns null if a list is empty.
* **set(int index, E element) ->** It replaces the element at the specified position in a list with the specified element.

**HASHTABLE:**

* Hashtable is a Legacy Class -> It means, this concept was first in JDK 1.0 but it later transferred from its parent JDK 1.0 to its next Versions upto JDK 11.
* HashTable is synchronised whereas a HashMap is not synchronised.
* But both HashMap and HashTable uses HashTable Internally and data stored in key value pair. As HashMap is not synchronised, its performance is better, but HashTable is synchronised, so performance is slowed as each Thread has to execute the whole synchronise block, and others have to wait.
* But Thread safety is there in HashTable because of being synchronised, whereas, Thread Safety is not there in HashMap. When we are achieving Thread safety, we are compromising with the performance.
* Here as it is synchronised, so it has methods like wait(), notify(), notifyAll().

**METHODS OF HashTable:**

* Put(key, value) -> for setting key and Its values inside HashTable by the process of Hashing.
* Get(key) -> for fetching the value inside HashTable with the “key”.
* Size() -> returns the size of HashTable.
* isEmpty() -> Checks if the HashTable is empty or not.
* keyset() -> returns all the keys of HashTable
* values() -> Returns all the values of HashTable only, not Keys.
* Remove(key) -> Removes the value as well as the key present in the argument, and returns the removed value.
* Replace(key, value) - >Replace or Update the previous value with the current value.
* Clear() -> Clears the HashTable so that it contains no keys.
* Clone() -> Creates a shallow copy of this HashTable
* Contains(Object value) -> checks if the value present in the method exist in HashTable or not and returns Boolean.
* ContainsKey(key) -> checks if that key exist in HashTable or not. Returns Boolean.
* ContainsValue(Object value) - >checks if value is present in HashTable or not, returns Boolean.

**DEFINATION:** The HashTable class creates a hash table, where values of different data types are stored and some unique identical keys are mapped with those values, to give them unique identity. And also fetch them with those unique KEYS when required. It’s a table Storing values in key Value pair. **Key** can be an **address** and Object value can be a **String,float,etc.,**. **Key** can be of any data type like, **String name**, and value can be **a number, or some other data types. Key** is an **identity** given to some value.

To successfully store and retrieve objects from a HashTable, all the keys(it may be Strings, Objects or numbers) must be converted into hashCode using **hashCode()** method, and divide the hashCode with the size of HashTable.

* It is similar to HashMap, but is synchronized.
* It doesn’t maintain Insertion Order as uses process of Hashing for storing data in the indexes. But Reading of data starts from initial Index only.
* The Hash table does not allow null value/key.
* HashTable stores key/value pair in hash table.
* Each Index is known as Bucket/slot.
* Any non-null object can be used as a key or as a value.
* In HashTable we specify an object(**can be String or Numbers)** that is used as a key, and the value we want to associate to that key like **“hash.map(101,”deb”)”**. The key is then hashed(**converted to hashCode/size of HashTable=nth Index)**, and then the Object value is stored in the Resulting Index within the table. **Here everything is converted to Object with Wrapper Class and then stored in HashTable.**
* **The initial default capacity** of HashTable class is **11** whereas **LoadFactor** is 0.75. **LoadFactor** means ***Load Factor*** *= Total number of items stored / Size of the array.* LoadFactor is that saturated point, after which the HashTable increases its size dynamically by creating a new HashTable with double the size of its initial capacity.
* Iterators in HashMap doesn’t provide any Enumeration, Iterator in the Hashtable is fail-safe because enumerator for the Hashtable is not throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator's own remove() method.
* Array also has key value pair(like index, and its corresponding value). HashTable is internally implemented just like an array only.

**Declaration:**

* public class HashTable <K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable
* Type Parameters:
  + K – the type of keys maintained by this map
  + V – the type of mapped values

**Hashing:**

Hashing is a technique where we are representing any entity in the form of Integer of 4 bytes, and it is done in java using hashCode() method**.** hashCode() is a method present inside Object Class. So we can get the hashCode method in any class, as Object class is inherited by all classes.

* To convert string into integer, uses a method hashCode. And this process of conversion of key=>“Mia” into an integer value=>77349 is known as hashing.
* HashTable Is an associated Array, where in the values are stored as a key-value Pair.

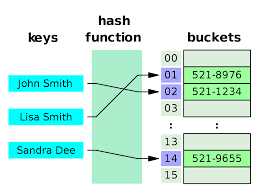
**hashCode has the potential to convert any datatype to its Integer Value.**

**“Mia” ->converted to hashCode->(Gives an integer value)/(Size of an array) = Index No. of HashTable**

* **REAL-Life example:** Someone has booked a hotel room(**one cell of hashtable**), and store laptopBags(**some integer value**) there. Now a thief(**get() methods for fetching values**) wants to access the laptop for fetching some info, so needs the key to unlock the room and then fetch.
* **Using** the key we are fetching the data also, and using the key we are storing the data also.
* HashCode not only converts the hexadecimal object into an integer value but it has the potential to convert any datatype to its Integer value. Ex. String x = “Mia” -> convert to hashCode ->77349

Now how the key and its value is stored is shown below, Suppose we have some names like:

* Mia -> 77349/(size of HashTable=11) => “Mia” stored in 8th Index of HashTable.
* Tom ->84274/( size of HashTable=11) => “Tom” stored in 3th Index of HashTable.
* Kia ->75427/( size of HashTable=11) => “Tom” stored in 0th Index of HashTable.



Now suppose it may happen that at any Interval, while inserting values in above way, one index came two times after division, so there will be a collision of index. This is **Collision** in hashing.

**Collision** is nothing but suppose in particular index already there is a value stored, now there is one more value, we get, that also now points to the same index. **Collision** occurs. When two values are being stored at same index number is called as collision. To solve this problem in HashTable, we store the data in the form of Nodes of LinkList mapped to the same index Number.

**Collision**  are 2 types:

* Chaining(Closed Addressing)
* Linear Probing(Open Chaining)

Collision->When two values are being stored at same index number is called as collision. To solve this problem in HashTable, we store the data in the form of Nodes of LinkList mapped to the same index Number.

When at a particular index, we are storing one value.

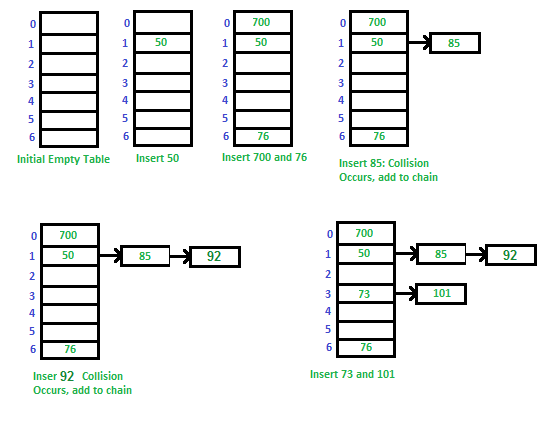
Internally in HashTable an Array is only implemented.

Another way to do with collisions is known as chains sometimes referred to as CLOSED ADDRESSING. In this, the one slot can form a chain by adding the various values into one bucket/index.

How to handle Collisions?   
There are mainly two methods to handle collision:   
1) Separate Chaining   
2) Open Addressing

Separate Chaining:   
The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Let us consider a simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



**Advantages:**   
1) Simple to implement.   
2) Hash table never fills up, we can always add more elements to the chain.   
3) Less sensitive to the hash function or load factors.   
4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

**Disadvantages:**  
1) Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.   
2) Wastage of Space (Some Parts of hash table are never used)   
3) If the chain becomes long, then search time can become O(n) in the worst case.   
4) Uses extra space for links.

**Performance of Chaining:**   
Performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of table (simple uniform hashing).

#### m = Number of slots in hash table

#### n = Number of keys to be inserted in hash table

#### Load factor α = n/m

#### Expected time to search = O(1 + α)

#### Expected time to delete = O(1 + α)

#### Time to insert = O(1)

#### Time complexity of search insert and delete is

#### O(1) if α is O(1)

**Question**. We know, load factor of a HashTable is defined as n/N where n=number of items N=Number of buckets in the hash table. Its recommended you increase the size of your HashTable when load factor is 0.75.

**Now my question is this:-**

#### If I have 75 items in my HashTable and the size of the hash HashTable table is 100, is it not possible that we put multiple items in the same bucket to save space? Which means I could also be using just 40 buckets. So why not resize when number of buckets/cells used reaches close to 100(size) instead of looking at number of items?

**ANS.** The reason you want number of buckets N to be greater than number of keys n, is that no algorithm (HashValue + compression) gives a perfect even distribution of keys though the buckets. So when n starts approaching nearer to N(size), the odds of collisions become higher. Which means you will have longer linked lists in certain buckets. Thus increasing the time to find that key.

| S.No. | **Separate Chaining** | **Open Addressing** |
| --- | --- | --- |
| 1. | Chaining is Simpler to implement. | Open Addressing requires more computation. |
| 2. | In chaining, Hash table never fills up, we can always add more elements to chain. | In open addressing, table may become full. |
| 3. | Chaining is Less sensitive to the hash function or load factors. | Open addressing requires extra care to avoid clustering and load factor. |
| 4. | Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted. | Open addressing is used when the frequency and number of keys is known. |
| 5. | Cache performance of chaining is not good as keys are stored using linked list. | Open addressing provides better cache performance as everything is stored in the same table. |
| 6. | Wastage of Space (Some Parts of hash table in chaining are never used). | In Open addressing, a slot can be used even if an input doesn’t map to it. |
| 7. | Chaining uses extra space for links. | No links in Open addressing |

**Open Addressing or Linear Probing** :  
Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

**Insert**(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

**Search**(k): Keep probing until slot’s key doesn’t become equal to k or an empty slot is reached.

**Delete**(k): Delete operation is interesting. If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as “deleted”.   
The insert can insert an item in a deleted slot, but the search doesn’t stop at a deleted slot.

Open Addressing is done in the following ways:

a) **Linear Probing:** In linear probing, we linearly probe for next slot. For example, the typical gap between two probes is 1 as seen in the example below.   
Let hash(x) be the slot index computed using a hash function and S be the table size

If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

..................................................

..................................................

Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2015/08/openAddressing1.png)

Collection leftout:

**1)java Hashset**

**2)Java Map Interface**

**3) Java HashMap**

**4)Working Of HashMap**

5)Java TreeSet

6)Java TreeMap

**7)Java HashTable**

**8)HashMap vs Hash Table**

9)ArrayList vs Vector

10)java Stack & Vetor

11) Java Queue

**12)Java Concurrent HashMap**

13)priority Queue

**Collection SET Interface:**

**DEFINATION:** The set interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the mathematical set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.

Doesn’t maintain any insertion order.

Cannot contain duplicate values.

After the introduction of [Generics](https://www.geeksforgeeks.org/generics-in-java/) in Java 1.5, it is possible to restrict the type of object that can be stored in the Set. Since Set is an interface, it can be used only with a class that implements this interface. HashSet is one of the widely used classes which implements the Set interface. Now, let’s see how to perform a few frequently used operations on the HashSet. We are going to perform the following operations as follows:

* Adding elements
* Accessing elements
* Removing elements
* Iterating elements
* Iterating through Set

| Method | Description |
| --- | --- |
| [add(element)](https://www.geeksforgeeks.org/set-add-method-in-java-with-examples/) | This method is used to add a specific element to the set. The function adds the element only if the specified element is not already present in the set else the function returns False if the element is already present in the Set. |
| [addAll(collection)](https://www.geeksforgeeks.org/set-addall-method-in-java-with-examples/) | This method is used to append all of the elements from the mentioned collection to the existing set. The elements are added randomly without following any specific order. |
| [clear()](https://www.geeksforgeeks.org/set-clear-method-in-java-with-examples/) | This method is used to remove all the elements from the set but not delete the set. The reference for the set still exists. |
| [contains(element)](https://www.geeksforgeeks.org/set-contains-method-in-java-with-examples/) | This method is used to check whether a specific element is present in the Set or not. |
| [containsAll(collection)](https://www.geeksforgeeks.org/set-containsall-method-in-java-with-examples/) | This method is used to check whether the set contains all the elements present in the given collection or not. This method returns true if the set contains all the elements and returns false if any of the elements are missing. |
| [hashCode()](https://www.geeksforgeeks.org/set-hashcode-method-in-java-with-examples/) | This method is used to get the hashCode value for this instance of the Set. It returns an integer value which is the hashCode value for this instance of the Set. |
| isEmpty() | This method is used to check whether the set is empty or not. |
| [iterator()](https://www.geeksforgeeks.org/set-iterator-method-in-java-with-examples/) | This method is used to return the [iterator](https://www.geeksforgeeks.org/iterators-in-java/) of the set. The elements from the set are returned in a random order. |
| [remove(element)](https://www.geeksforgeeks.org/set-remove-method-in-java-with-examples/) | This method is used to remove the given element from the set. This method returns True if the specified element is present in the Set otherwise it returns False. |
| [removeAll(collection)](https://www.geeksforgeeks.org/set-removeall-method-in-java-with-examples/) | This method is used to remove all the elements from the collection which are present in the set. This method returns true if this set changed as a result of the call. |
| [retainAll(collection)](https://www.geeksforgeeks.org/set-retainall-method-in-java-with-example/) | This method is used to retain all the elements from the set which are mentioned in the given collection. This method returns true if this set changed as a result of the call. |
| [size()](https://www.geeksforgeeks.org/set-size-method-in-java-with-example/) | This method is used to get the size of the set. This returns an integer value which signifies the number of elements. |
| [toArray()](https://www.geeksforgeeks.org/set-toarray-method-in-java-with-example/) | This method is used to form an array of the same elements as that of the Set. |

*Classes that implement the Set interface in Java Collections can be easily perceived from the image below as follows and are listed as follows:*

* *HashSet*
* *EnumSet*
* *LinkedHashSet*
* *TreeSet*

**HASHSET:**

* **HashSet** uses **HashTable** **internally** thus creating 16-sized HashTable when we call HashSet.
* It **doesn’t** maintain **Insertion Order** as value are set in any place of HashTable using process of **Hashing**. But **Reading** of **data** starts from **1st Index** only.
* **Important:** As doesn’t maintain **Insertion** Order, so all indexes are not placed In contagious manner like array. So, using **get()** method to fetch random elements is not allowed. Instead we use Iterator() method to **iterate** from end to the other and printing them one by one.
* It uses hashing to inject the data into the DB table.
* It **doesn’t store** the **Values** inside a **Set Table** in **Key-Value** Pair but only the single data.
* It will contain only **unique** elements and **no duplicates**. **Doesn’t maintain insertion order**
* This is not **synchronised**, so not ThreadSafe.
* This **class** **permits** **null** element, and **storing** of “**null”** values can be done.
* Initial size of **HashSet**(internally HashTable created) is **16**.Now Increase of size will happen when **size** is filled by (**LoadFactor**)75% of 16(**Initial**) =**12**. When **HashSet** is filled upto 12(**buckets/cells**), then a **New** **HashTable** is **created** with **double** the **size** of Initial(16) = 32. And the **old values** of **HashSet** is copied to the new one.

In HashSet also we use the key and convert it into hashCode-> divide the hashCode with (Size of Array) to give an Index -> then store the data into that index.

**Set -> HashSet-> LinkedHashSet.**

**Internal working of a HashSet**: All the classes of Set interface internally backed up by HashMap. A Map Object is created when we call “HashSet” Constructor. HashSet uses HashMap for storing its object internally. You must be wondering that to enter a value in HashMap we need a key-value pair, but in HashSet, we are passing only one value, which is considered as a key, and a constant value “Present” is taken in place of value.

**Storage in HashMap:** Actually the value we insert in HashSet acts as a key to the map Object and for its value, java uses a constant variable. So in key-value pair, all the values will be the same.

**// Constructor - 1**

**// All the constructors f HashSet oare internally creating HashMap Object.**

#### private transient HashMap map;

#### public HashSet()

#### { // Creating internally backing HashMap object

#### map = new HashMap();

#### }

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**HashSet**](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html#HashSet())()  Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75). |
| [**HashSet**](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html#HashSet(java.util.Collection))([**Collection**](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<? extends [**E**](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html)> c)  Constructs a new set containing the elements in the specified collection. |
| [**HashSet**](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html#HashSet(int))(int initialCapacity)  Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75). |
| [**HashSet**](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html#HashSet(int,%20float))(int initialCapacity, float loadFactor)  Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor. |

**If we look at the add() method of HashSet class:**

#### public boolean add(E e){

#### return map.put(e, PRESENT) == null;

#### }

We can notice that, add() method of HashSet class internally calls the put() method of backing the HashMap object by passing the element you have specified as a key and constant “PRESENT” as its value. remove() method also works in the same manner. It internally calls remove method of Map interface. 

#### public boolean remove(Object o){

#### return map.remove(o) == PRESENT;

#### }

HashSet not only stores unique Objects but also unique Collection of Objects like [ArrayList<E>](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList<E>](https://www.geeksforgeeks.org/linked-list-in-java/), [Vector<E>](https://www.geeksforgeeks.org/java-util-vector-class-java/#:~:text=The%20Vector%20class%20implements%20a%20growable%20array%20of%20objects.&text=They%20are%20very%20similar%20to,AbstractList%20and%20implements%20List%20interfaces.),..etc.

Let us understand this with the help of the below example:

|  |
| --- |
| // Java program to illustrate the concept  // of Collection objects storage in a HashSet  **import** java.io.\*;  **import** java.util.\*;  **class** CollectionObjectStorage {  **public** **static** **void** main(String[] args){          // Instantiate an object of HashSet          HashSet<ArrayList> set = **new** HashSet<>();          // create ArrayList list1          ArrayList<Integer> list1 = **new** ArrayList<>();          // create ArrayList list2          ArrayList<Integer> list2 = **new** ArrayList<>();          // Add elements using add method          list1.add(1);          list1.add(2);          list2.add(1);          list2.add(2);          set.add(list1);          set.add(list2);            // print the set size to understand the          // internal storage of ArrayList in Set          System.out.println(set.size());      }  } |

**Output:**

1

#### Before storing an Object, HashSet checks whether there is an existing entry using [hashCode() and equals() methods.](https://www.geeksforgeeks.org/equals-hashcode-methods-java/) In the above example, two lists are considered equal if they have the same elements in the same order. When you invoke the [hashCode()](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractList.html#hashCode()) method on the two lists, they both would give the same hash since they are equal.

#### HashSet does not store duplicate items,  if you give two Objects that are equal then it stores only the first one, here it is list1.

#### Time Complexity of HashSet Operations: The underlying data structure for HashSet is hashtable. So amortize (average or usual case) time complexity for add, remove and look-up (contains method) operation of HashSet takes O(1) time.

Some Methods of HashSet:

Add(Object value) -> Adds values to the specified Set,if not present. If present, returns false.

Contains(Object value) -> Checks if value present inside set, then returns true.

Remove(Object value) -> Used to remove the element if it is present in set.

Iterator() -> Used to return an iterator over the element in the set.

isEmpty() -> checks if set is empty or not.

Size() -> returns the size of set.

Equals() -> Used to verify the equality of an Object with a HashSet and compare them. The list returns true only if both HashSet contains same elements, irrespective of order

addAll() -> This method is used to append all the elements of another newly created HashSet from the existing set. The elements are added randomly without following any specific order

Clear() ->remove all the elements from the set.

**LinkedHashSet:**

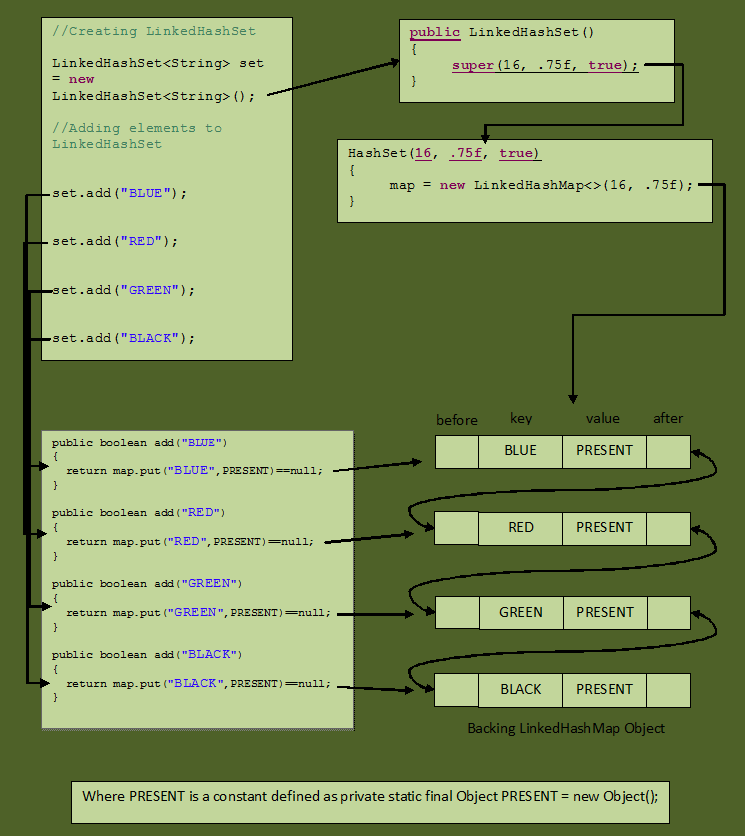
The **LinkedHashSet** is an **ordered version** of **HashSet** that **maintains** a **doubly-linked List** across all **elements**. When the **iteration order** is needed to be **maintained** this **class** is used. When iterating through a [**HashSet**](https://www.geeksforgeeks.org/hashset-in-java/) the **order** is **unpredictable**, while a **LinkedHashSet** lets us **iterate** through the **elements** in the **order** in which they were **inserted**. When cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

* LinkedHashSet uses LinkedHashMap object to store it’s elements. It’s a class implementing Set Interface.
* Its not synchronised.
* It maintains an Insertion Order.
* It uses **Double Link-List internally** to **add** elements just like **LinkedHashMap**, which are **connected** to each other. And when we print the **LinkedHashSet**, it starts with the **First node** of **Double-LinkedList** that was inserted, and **traverse** upto the end.
* Can contains only Unique Elements. Doesn’t allow and ignore Duplicate Values if Inserted.
* It can contain null Values.

### **LinkedHashSet class declaration**

### **public** **class** LinkedHashSet<E> **extends** HashSet<E> **implements** Set<E>, Cloneable, Serializable  .

**::Methods are all same as that of HashSet.**



**TREE\_SET:**

**DEFINATION:** Tree Set class implements SortedSet Interface and SortedSet Interface extends Set Interface. TreeSet uses a tree data structure for storage in a sorted Manner. Objects are stored in sorted, ascending order. But we can iterate in descending order using the method TreeSet.descendingIterator().

* TreeSet sorts our data in ascending Order and stores the data into it.
* TreeSet cannot contain **Heterogeneous** elements, like String, Integer, etc., altogether.
* This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).
* **This implementation is not synchronized.** If multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it *must* be synchronized externally.
* The iterators returned by this class's iterator method are *fail-fast*: if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a [ConcurrentModificationException](eclipse-javadoc:%E2%98%82=app_java_Collections/C:%5C/Program%20Files%5C/AdoptOpenJDK%5C/jdk-14.0.1.7-hotspot%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava.util(TreeSet.class%E2%98%83TreeSet%E2%98%82ConcurrentModificationException). Thus, in the face of concurrent modification, the iterator fails quickly.
* If we try to insert a null value to “TreeSet” then it gives me a NullPointerException.
* It implements the SortedSet and NavigableSet interface to keep the elements sorted and structured.
* It also implements the Cloneable and Serializable interfaces.
* Contains Unique elements only. Doesn’t allow Duplicate Values.
* The HashSet uses a [Hashtable](https://www.javatpoint.com/java-hashtable), and the TreeSet uses a self-balancing tree.

The TreeSet stores the objects in the ascending order, which is a natural ordering of a tree. We can also specify a comparator to sort the elements based on it during the creation of the TreeSet. It implements the **SortedSet** and **NavigableSet** interface to maintain and navigate the order of the elements.

It is a useful class for finding the comparison between the elements, such as greater than, less than, etc., between the available elements of the tree.

### **Internal Working of the TreeSet**

The data structure for the TreeSet is TreeMap; it contains a **SortedSet** & **NavigableSet** interface to keep the elements sorted in ascending order and navigated through the tree.

As soon as we create a TreeSet, the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) creates a TreeMap to store the elements in it and performs all the operations on it. It's working is similar to HashSet.

When we implement a TreeSet the following code will be executed by the [Java](https://www.javatpoint.com/java-tutorial) Compiler:

1. **public** TreeSet() {
2. **this**(**new** TreeMap<>());
3. }

It instantiate a TreeMap and executes the following code to implement the interfaces:

1. **private** **transient** NavigableMap<E,Object> m;
2. TreeSet(NavigableMap<E,Object> m) {
3. **this**.m = m;
4. }

It automatically assigns elements to the corresponding position using the **NavigableMap**. The TreeMap can not have a duplicate value. So, when we execute the add() method, it performs a put operation and adds a value to the tree.

By default, It sorts values in ascending order, but we can specify the comparator if we want to sort the elements in any specific order. It uses a comparator or comparable variable to compare the elements.

**HashMap->**

* It’s an class that doesn’t belong or part of collection.
* HashMap uses **HashTable** **Internally** to **store Key-Value Pair** by **process** of **Hashing** **Technique** with **hashCode**(). So, its data is inserted into the Tables using process **Hashing**.
* **As HashMap uses HashTable internally, thatswhy doesn’t maintains Insertion Order**.
* A key can be null here and we can even store multiple null values in it.
* Default Initial Capacity of HashMap is 16.
* HashMap is not synchronized.
* //In HashSet the data doesn’t stores value in the form of Key Value Pair.
* //Both HashMap and HashSet are not synchronized.

**Important Note:**

Practical Example of HashMap: Suppose in a college every student has a Unique Roll Number. **Now “Roll Number”** is the **key,** and **value= ”name of Student”.**

In **Java JDBC**, we had **‘request.setAttribute(“keyname”,”value=deba”)’,** which is also kind of **HashMap.**

[equals() method](https://www.geeksforgeeks.org/equals-hashcode-methods-java/)

equals method is used to check that 2 objects are equal or not. This method is provided by Object class. You can override this in your class to provide your own implementation.   
HashMap uses equals() to compare the key whether the are equal or not. If equals() method return true, they are equal otherwise not equal. 

Buckets

A bucket is one element of HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case link list structure is used to connect the nodes. Buckets are different in capacity. A relation between bucket and capacity is as follows:

A single bucket can have more than one nodes, it depends on hashCode() method. The better your hashCode() method is, the better your buckets will be utilized.

**Difference between HashMap vs HashTable:**

1. Synchronization or Thread Safe :This is the most important difference between two . HashMap is non synchronized and not thread safe. On the other hand, HashTable is thread safe and synchronized. HashMap should be used if your application do not require any multi-threading task, in other words HashMap is better for non-threading applications. HashTable should be used in multithreading applications.
2. Null keys and null values :HashMap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.
3. Performance :hash.map is much faster and uses less memory than Hashtable as former is unsynchronized . Unsynchronized objects are often much better in performance in compare to synchronized object like Hashtable in single threaded environment.
4. HashMap inherits Abstract Map class while Hashtable inherits Dictionary class.
5. The significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally by adding or removing any element except Iterator’s own remove() method. But this is not a guaranteed behaviour and will be done by JVM on best effort. This is also an important difference between Enumeration and Iterator in Java.

**HashTable(Class) -> extends -> Dictionary(class) -> implements -> Map(Interface)**

Difference between HashSet and HashTable.

**HashMap -> (directly implements) -> Map(Interface) ->**

**CONCURRENT HASHMAP:**

* In Concurrent HashMap, the HashTable is implemented internally. That HashTable is divided into Segments(**Index)**. Consider Each Segments to be each Index of HashTable.
* This doesn’t allow “**null” keys & “null Values”. ConcurrentHashMap** is an enhanced version of **HashTable.**
* By Default ConcurrentHashMap has a default Size of 16 Segments.

**Now There is an Unique mechanism of HashMap:**

As HashMap is Synchronized. So, it’s Thread Safe. But we know those concepts (like HashTable or synchronizedMap) which are synchronized are Thread Safe ,but lower Performance, as each Thread gets a chance to execute a block at a time, and other Thread are made to “wait”. Now, using Synchronised mechanism, we made the Performance fast, using Concurrent HashMap. **How?**

By dividing the Concurrent HashMap into Segments, and making each segments Synchronised, so that only one Thread enters the Segments at a Time. So, now we don’t block the whole Synchronised HashTable for just one Thread at a time. **Instead,** we make the small **Segments** or Index of HashTable, to be synchronised, and making each Thread to **add values** on these segments at a time.

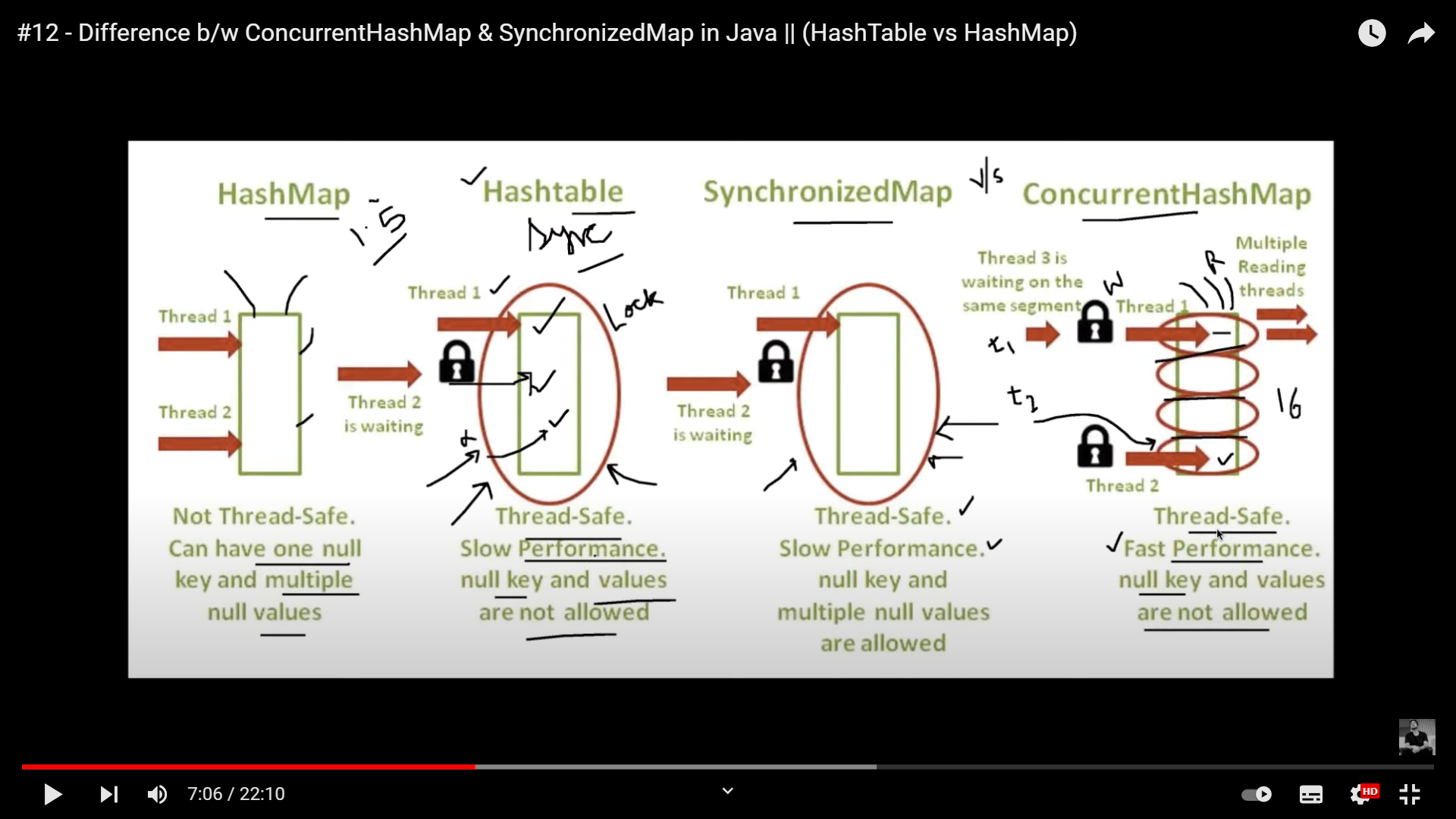
Thus, we can make multiple Threads to add values to The small segments of HashTable in the same time.

**Now** Problem is solved with HashMap. For inserting one value to the Index by Thread, we don’t need to block the whole synchronised HashTable for that and make other Thread to wait for adding values.

This was for “add” operation. But in a “**read”** operation, like if we want to fetch value from HashTable from one segment or Index. Then many Threads can operate at a single time to fetch or red values of a single segment or Index using **get()**.

For read Segment Level Lock is applied, and not the Object level of whole Hashtable.

But In HashTable without ConcurrentHashMap, suppose, one Thread is adding value to the Index, and blocking the whole table, then suppose another Thread is for reading the value. But that second Thread has to wait for the First Thread to complete its execution of adding. Thus reducing performance by increasing execution Time.



**CONCLUSION**: If we need to add some values to HashTable, then ConcurrentHashMap is preferred over other Synchronized Collection Concepts like HashTable because multiple Threads can be used to add values to Segments or Index at a time to insert in the Segment Level Synchronised block segment. Execution speed is faster.

**Q)** Why Concurrent is Fail-Fast, and why “**ConcurrentModificationException”** is not shown here**?**

In Concurrent HashMap, we can use Multiple Threads to operate on the same Module. So it’s **fail-safe**. As multiple Threads can manipulate or modify the Values of Concurrent HashMap. So, it doesn’t show **“ConcurrentModificationException”.**

**But for others Like synchronised HashMap which are fail-fast, if we try to manipulate the values with one Thread t2 while another Thread t1 was already Traversing and adding values to HashMap, then as it’s synchronised, So multiple Threads cannot Operate and perform two tasks concurrently in the same module at the same time. So this throws “ConcurrentModificationException”.**

# **Java LinkedHashMap class**



Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

### **Points to remember**

* Java LinkedHashMap contains values based on the key.
* Java LinkedHashMap contains unique elements.
* Java LinkedHashMap may have one null key and multiple null values.
* Java LinkedHashMap is non synchronized.
* Java LinkedHashMap maintains insertion order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

### **LinkedHashMap class declaration**

Let's see the declaration for java.util.LinkedHashMap class.

1. **public** **class** LinkedHashMap<K,V> **extends** HashMap<K,V> **implements** Map<K,V>

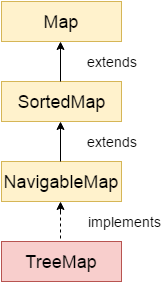
**TREE MAP :**

Tree Map is a concept which is neither connected to HashTable nor HashMap nor Hashing.

It’s based on Self-Balancing Tree.

All **Homogeneous** values to be inserted in the Tree. Duplicate and null values cannot be kept there.

# **Java TreeMap class**



Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

* Java TreeMap contains values based on the key. It implements the **Navigable Map** interface and extends AbstractMap class.
* Java TreeMap contains only unique elements.
* Java TreeMap cannot have a null key but can have multiple null values.
* Java TreeMap is non synchronized.
* Java TreeMap **sorts**  elements w.r.t **keys, and** maintains ascending order.
* We cannot use Iterator to traverse tree map. We use Comparator.

**IMPORTANT POINTS:**

* entrySet() is a method is a method of map Interface, used to return a “Set” of values.
* HashMap<Integer> hash = new HashMap<Integer>();
* hash.entrySet() -> converts values of HashMap and returns a “Set” and stored into a Reference of “entr” of type “Set” and the Object values are of type inside Generics “<>” is of type Entry.
* “Entry” is an child inner Interface inside parent “Map” Interface.
* Ex. Set<Integer> entr = new Set<Integer>();
* //In above line, Set is the type, <Integer> determines the type of Integer Object values passed to the “Set”.
* //In same way, Set is the type, <Entry<K,V>> determines the type of Entry Object values passed to the “Set”.
* Set<Entry<K,V>> entr = hash.entrySet();//returns the values of type Set.

**COMPARATOR:**

Comparator is used to compare the Objects Content.

If in sorting, Object1 comes first and then Object2, then it will return negative Value.

If in sorting, Object2 comes first and then Object1, then it will return positive Value.

If in sorting, Object1 and Object2, both are same, then it will return 0. Actually if we write **String x=”abc” && String y=”abc”, then returns 0,** because actually it **creates** only **one object** containing **abc, and make both x and y reference point to the same Object.**

**GENERICS:**

**Generics** gives me flexibility to dynamically allocate the datatypes in my program. Whenever <T> constant is used, that constant is dynamically replaced to “String” or “Integer” from outside class.

**JDK 9:**

A new Feature Jshell was introduced, which is like a terminal command window. This helps us to quickly built the java code in it. To access Jshell -> go to java->JDK -> bin ->double click on jshell.exe file.

**Jshell**-> We can basically write some small program logic in a cut-short manner, and execute them.

**JDK 10:**

Var was introduced, which can store values of any datatypes.

**JDK 11:**

Even we can directly write some Unicode inside “**S.O.Pln()”** and execute for getting emojis and symbols.

Ex.\u265A, gives the emoji of King in chess.

Sorting an HashMap and printing the values.

HashMap follows insertion order or not?

LinkList is more preferable for inserting elements as more optimized.

HashMap internal APPLICATION working.

FailFast & Fail-safe ->

HashTable:

It stores the content as key value pair

HashTable is synchronised.

**Optional Class:**

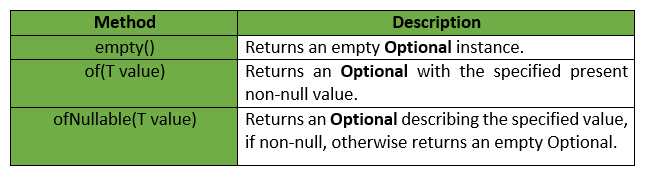
**Definition:** Optional class was in version 8 of JAVA. This class helps us to check if the reference variable has null value or not, and easy way to handle Null Pointer exceptions. Optional class reduces repeated us of try-catch block to handle null Exception. Define one method containing Optional class and pass reference of any class, and It’s a container object which may or may not contain a non-null value. You must import java.util package to use this class. If a value is present, isPresent() will return true and get() will return the value. Every programmer is familiar with “*NullPointerException”.* It’s very hard to avoid it without using too many null Checks and to handle multiple Exceptions, we have to use several try-catch Blocks. But writing everytime try-catch blocks makes our code lengthy.

To avoid using multiple try-catch Blocks for handling our most common Exception “*NullPointerException”,* we rather use Optional Class for writing a neat code without using too many null checks. And our Overall *NullPointerException* in our Project Can be handled with just one Code with Optional Class. And also using **Optional Class** we can specify some **alternate values** to **return** or some **message** to display instead of showing“*NullPointerException”.*

//Optional. ofNullable - throws NullPointerException if passed parameter is null

Optional<Integer> b = Optional.ofNullable(value2);

To avoid **abnormal** termination like ***“Exception in thread "main" java.lang.NullPointerException”,*** we use the Optional class to make our program execute without crashing and display some **understandable message** to the user.



**STREAM API:**

Some of the most prominent methods used in these examples are the [filter()](http://www.java67.com/2016/08/java-8-stream-filter-method-example.html) -  which allows elements that match the predicate, count() - which counts the number of items in a stream, [map()](http://www.java67.com/2015/01/java-8-map-function-examples.html) - which applies a function in each element of Stream for transformation, and [collect()](http://www.java67.com/2018/06/java-8-streamcollect-example.html) - which collects the final result of Stream processing into a Collection.  
  
Now, let's walk through each example to understand what they are doing and how they are doing.

### 1. Java 8 Filter Example: Counting Empty String

Here is an example of counting how many elements are in the stream at any stage of pipeline processing using the count() method of Stream class.

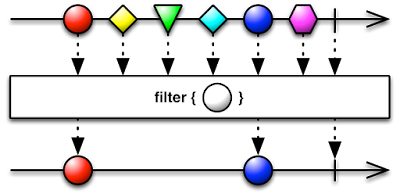
List<String> strList = Arrays.asList("abc", "", "bcd", "", "defg", "jk");

long count = strList.stream()

.filter(x -> x.isEmpty())

.count();

This is an excellent example to demonstrate how you can filter specific objects from [Collection](http://java67.blogspot.com/2012/08/how-to-create-read-only-collection-in-java-example.html) and create a subset of elements that satisfy the given criterion. In second-line strList.stream() returns a Stream, and then we use the filter() method, which accepts a Predicate.  
  
Since the java.util.function.Predicate is a [functional interface](https://javarevisited.blogspot.com/2018/01/what-is-functional-interface-in-java-8.html) ( an interface with just one abstract method), we can pass lambda expression instead of an instance of the Predicate interface. Here we can define code to specify a condition.  
  
This code will go to the test() method of Predicate and will be applied to each element during internal iteration. All Strings which are [empty](https://javarevisited.blogspot.com/2013/02/5-ways-to-check-if-string-is-empty-in-java-examples.html) are counted by the count() method, which is a terminal operation.  
  
After this line, you can not call any method on this Stream. Remember filter() is a tricky method, **it does not filter element from the original collection; instead**, it **selects** item which satisfies criterion and returns them in new Collection.  
  
You can read more about that in this excellent [**From Collections to Streams in Java 8 Using Lambda Expressions**](https://javarevisited.blogspot.com/2020/04/top-5-courses-to-learn-java-collections-and-streams.html#axzz6nwXUSoGH) course on Pluralsight.

[](https://medium.com/javarevisited/7-best-java-collections-and-stream-api-courses-for-beginners-in-2020-3ad18d52c38)

Read more: <https://www.java67.com/2014/04/java-8-stream-examples-and-tutorial.html#ixzz7JrJPckni>

This is another example of using the [map()](https://javarevisited.blogspot.com/2018/05/java-8-filter-map-collect-stream-example.html) method, here we are mapping each element to their square and then filtering out all duplicate elements by calling the distinct() method.

List<Integer> numbers = Arrays.asList(9, 10, 3, 4, 7, 3, 4);

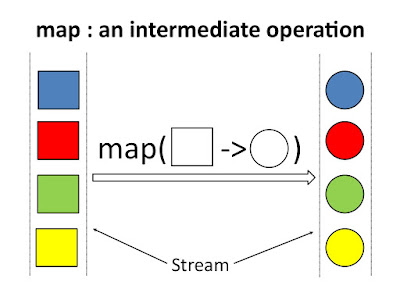
List<Integer> distinct = numbers.stream()

.map( i -> i\*i)

.distinct()

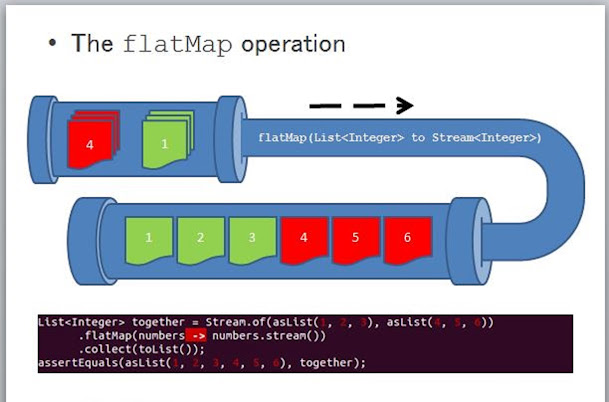
.collect(Collectors.toList());

Finally, by using the collect() method, we are gathering output into a List.

//Get count, min, max, sum, and average for numbers List<Integer> primes = Arrays.asList(2, 3, 5, 7, 11, 13, 17, 19, 23, 29); IntSummaryStatistics stats = primes.stream() .mapToInt((x) -> x) .summaryStatistics(); System.out.println("Highest prime number in List : " + stats.getMax()); System.out.println("Lowest prime number in List : " + stats.getMin()); System.out.println("Sum of all prime numbers : " + stats.getSum());  
mapping square to circle, by changing the property of square and making it circle.

FlatMap() Method:

if the map operation uses a function which instead of returning a single value returns a Stream of values like when you give a number and it returns all [prime factors](http://javarevisited.blogspot.sg/2014/05/how-to-find-prime-factors-of-integer-number-java.html) of the number then you have a *Stream of Stream of integers*. That's where the flatmap function helps.  
  
The flatMap() method can be used to flatten that stream into a stream of integers. For example, suppose, you have a list of numbers like [21, 23, 42] and we call the getPrimeFactors() method along with the [map() operation](http://javarevisited.blogspot.com/2014/02/10-example-of-lambda-expressions-in-java8.html) to transform this stream.  
  
The result would be [[3,7],[23],[2,3,7]]. If you want to flatten this stream of a stream into a stream of values, you can use the flatMap() which will finally return [3,7,2,3,2,3,7].



**Designing Synchronized and Thread Safe Design Pattern:**

|  |
| --- |
| // Java program to create Thread Safe  // Singleton class  **public** **class** GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method  **private** **static** GFG instance;    **private** GFG()    {      // private constructor    }     //synchronized method to control simultaneous access  **synchronized** **public** **static** GFG getInstance()    {  **if** (instance == **null**)      {        // if instance is null, initialize        instance = **new** GFG();      }  **return** instance;    }  } |

1. **Pros:**
   1. Lazy initialization is possible.
   2. It is also thread safe.
   3. getInstance() method is synchronized so it causes slow performance as multiple threads can’t access it simultaneously.
2. **Lazy initialization with Double check locking:** In this mechanism, we overcome the overhead problem of synchronized code. In this method, getInstance is not synchronized but the block which creates instance is synchronized so that minimum number of threads have to wait and that’s only for first time.

* JAVA

|  |
| --- |
| // Java code to explain double check locking  **public** **class** GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method  **private** **static** GFG instance;    **private** GFG()    {      // private constructor    }    **public** **static** GFG getInstance()    {  **if** (instance == **null**)      {        //synchronized block to remove overhead  **synchronized** (GFG.**class**)        {  **if**(instance==**null**)          {            // if instance is null, initialize            instance = **new** GFG();          }          }      }  **return** instance;    }  } |

1. **Pros:**
   1. Lazy initialization is possible.
   2. It is also thread safe.
   3. Performance overhead gets reduced because of synchronized keyword.
   4. First time, it can affect performance.

### HashSet vs HashMap

| BASIS | HashSet | HashMap |
| --- | --- | --- |
| *Implementation* | HashSet implements Set interface. | HashMap implements Map interface. |
| *Duplicates* | HashSet doesn’t allow duplicate values. | HashMap store key, value pairs and it does not allow duplicate keys. If key is duplicate then the old key is replaced with the new value. |
| *Number of objects during storing objects* | HashSet requires only one object add(Object o). | HashMap requires two objects put(K key, V Value) to add an element to the HashMap object. |
| *Dummy value* | HashSet internally uses HashMap to add elements. In HashSet, the argument passed in add(Object) method serves as key K. Java internally associates dummy value for each value passed in add(Object) method. | HashMap does not have any concept of dummy value. |
| *Storing or Adding mechanism* | HashSet internally uses the HashMap object to store or add the objects. | HashMap internally uses hashing to store or add objects |
| *Faster* | HashSet is slower than HashMap. | HashMap is faster than HashSet. |
| *Insertion* | HashSet uses the add() method for add or storing data. | HashMap uses the put() method for storing data. |
| *Example* | HashSet is a set, e.g. {1, 2, 3, 4, 5, 6, 7}. | HashMap is a key -> value pair(key to value) map, e.g. {a -> 1, b -> 2, c -> 2, d -> 1}. |

### HashSet vs TreeSet

| BASIS | HashSet | TreeSet |
| --- | --- | --- |
| *Speed and internal implementation* | For operations like search, insert and delete. It takes constant time for these operations on average. HashSet is faster than TreeSet. HashSet is Implemented using a hash table. | TreeSet takes O(Log n) for search, insert and delete which is higher than HashSet. But TreeSet keeps sorted data. Also, it supports operations like higher() (Returns least higher element), floor(), ceiling(), etc. These operations are also O(Log n) in TreeSet and not supported in HashSet. TreeSet is implemented using a Self Balancing Binary Search Tree (Red-Black Tree). TreeSet is backed by TreeMap in Java. |
| *Ordering* | Elements in HashSet are not ordered. | TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java. TreeSet elements are sorted in ascending order by default. It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc. |
| *Null Object* | HashSet allows the null object. | TreeSet doesn’t allow null Object and throw NullPointerException, Why, because TreeSet uses compareTo() method to compare keys and compareTo() will throw java.lang.NullPointerException. |
| *Comparison* | HashSet uses equals() method to compare two objects in Set and for detecting duplicates. | TreeSet uses compareTo() method for same purpose. If equals() and compareTo() are not consistent, i.e. for two equal object equals should return true while compareTo() should return zero, then it will break the contract of the Set interface and will allow duplicates in Set implementations like TreeSet |

| Categories | LinkedHashMap | LinekdHashSet |
| --- | --- | --- |
| Operation | Usd to store key-value pairs. | Used to store collection of things |
| Duplicates | Take unique an no duplicate keys but can takeduplicate values | Stores no duplicate element |
| Implements | HashMap | HashSet |
| Example | Map<String, Integer> lhm = new LinkedHashMap<String, Integer>(); | Set<String> lhs = new LinkedhashSet<String>(); |

**Small Story of Hashmap, ArrayList & LinkList:**

* The downfall of ArrayList and LinkedList is that when iterating through them, depending on the search algorithm, the time it takes to find an item grows with the size of the list.
* The beauty of hashing is that although you sacrifice some extra time searching for the element, the time taken does not grow with the size of the map. This is because the HashMap finds information by converting the element you are searching for, directly into the index, so it can make the jump.
* Long story short... **LinkedList**: Consumes a little more memory than ArrayList, low cost for insertions(add & remove) **ArrayList**: Consumes low memory, but similar to LinkedList, and takes extra time to search when large. **HashMap**: Can perform a jump to the value, making the search time constant for large maps. Consumes more memory and takes longer to find the value than small lists.
* Lists and Maps are different data structures. Maps are used for when you want to associate a key with a value and Lists are an ordered collection.
* Map is an interface in the Java Collection Framework and a HashMap is one implementation of the Map interface. HashMap are efficient for locating a value based on a key and inserting and deleting values based on a key. The entries of a HashMap are not ordered.
* ArrayList and LinkedList are an implementation of the List interface. LinkedList provides sequential access and is generally more efficient at inserting and deleting elements in the list, however, it is it less efficient at accessing elements in a list. ArrayList provides random access and is more efficient at accessing elements but is generally slower at inserting and deleting elements.

Actuator:

Using Actuator we get multiple end-points. We get production-grade tools without having to actually implement these features ourselves.

They are mainly used to expose Operational Information about the running Application –like health, info, dump, etc. It uses HTTP ENDPOINTS OR JMX Beans to enable us to interact with it. /health is used to monito the status of the application.

/beans is used to create an return of All the beans used in our Application.

### What is an Optional class?

Optional is a container type which may or may not contain value i.e. zero(null) or one(not-null) value. It is part of java.util package. There are pre-defined methods like isPresent(), which returns true if the value is present or else false and the method get(), which will return the value if it is present.

**static** Optional<String> **changeCase**(String word) {

**if** (name != **null** && word.startsWith("A")) {

**return** Optional.of(word.toUpperCase());

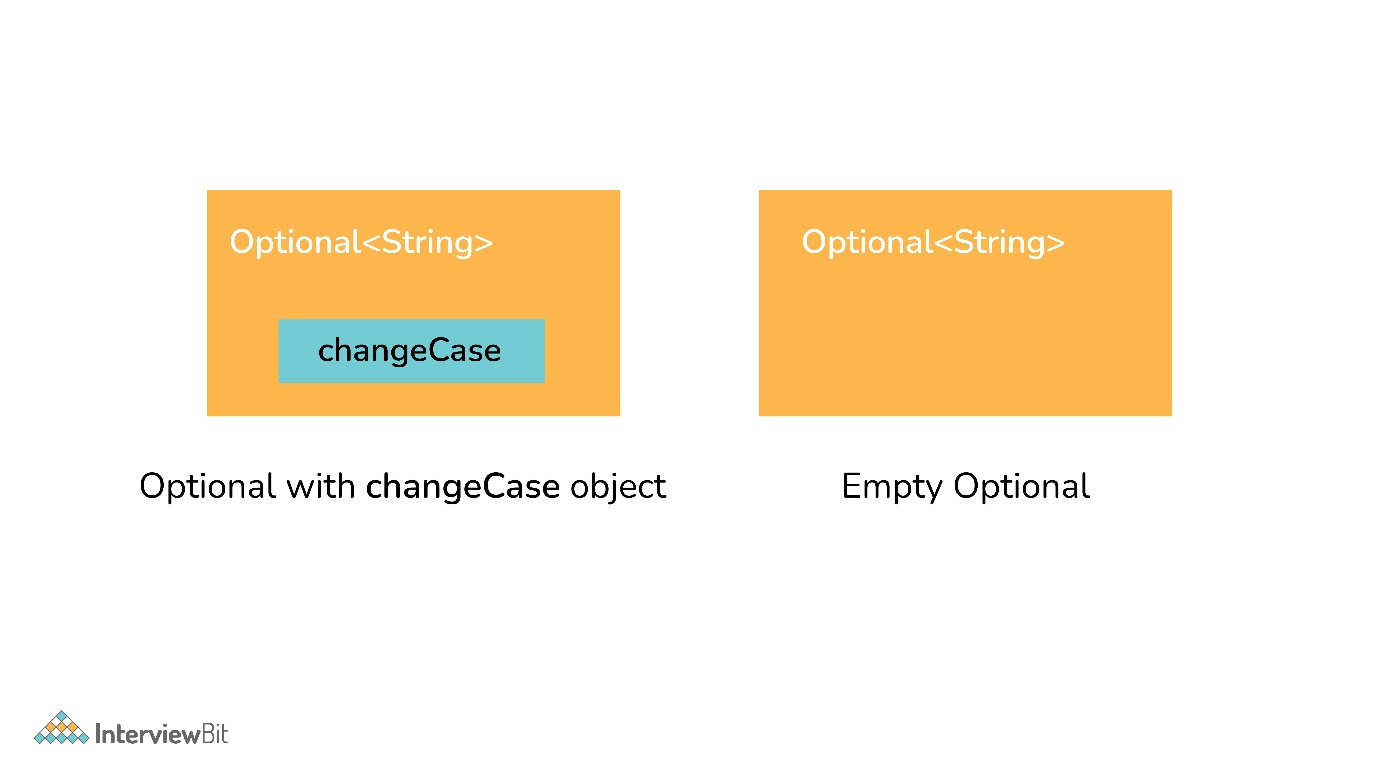
}

**else** {

**return** Optional.ofNullable(word); // someString can be null

}

}

Optional Class

### 19. What are the advantages of using the Optional class?

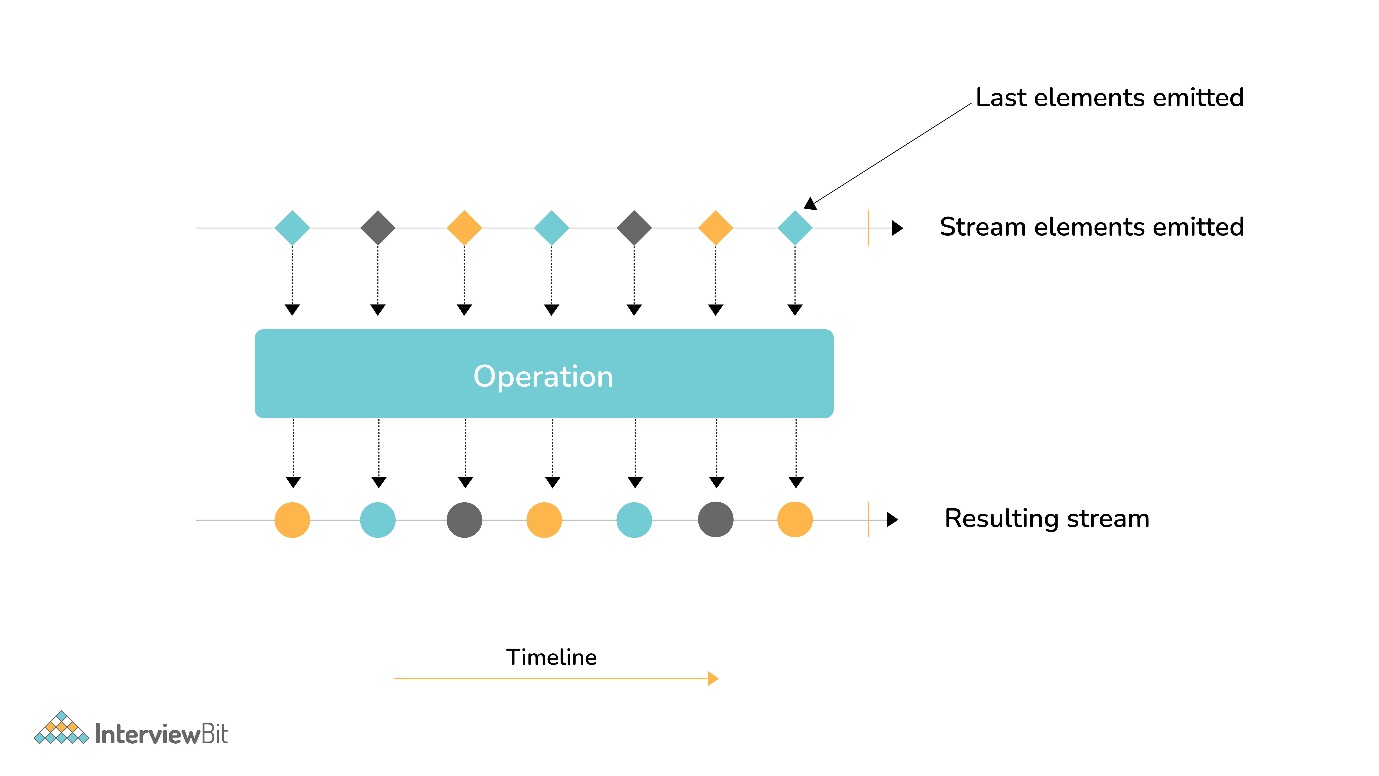
Below are the main advantage of using the Optional class:

It encapsulates optional values, i.e., null or not-null values, which helps in avoiding null checks, which results in better, readable, and robust code It acts as a wrapper around the object and returns an object instead of a value, which can be used to avoid run-time NullPointerExceptions.

### 20. What are Java 8 streams?

A stream is an abstraction to express data processing queries in a declarative way.

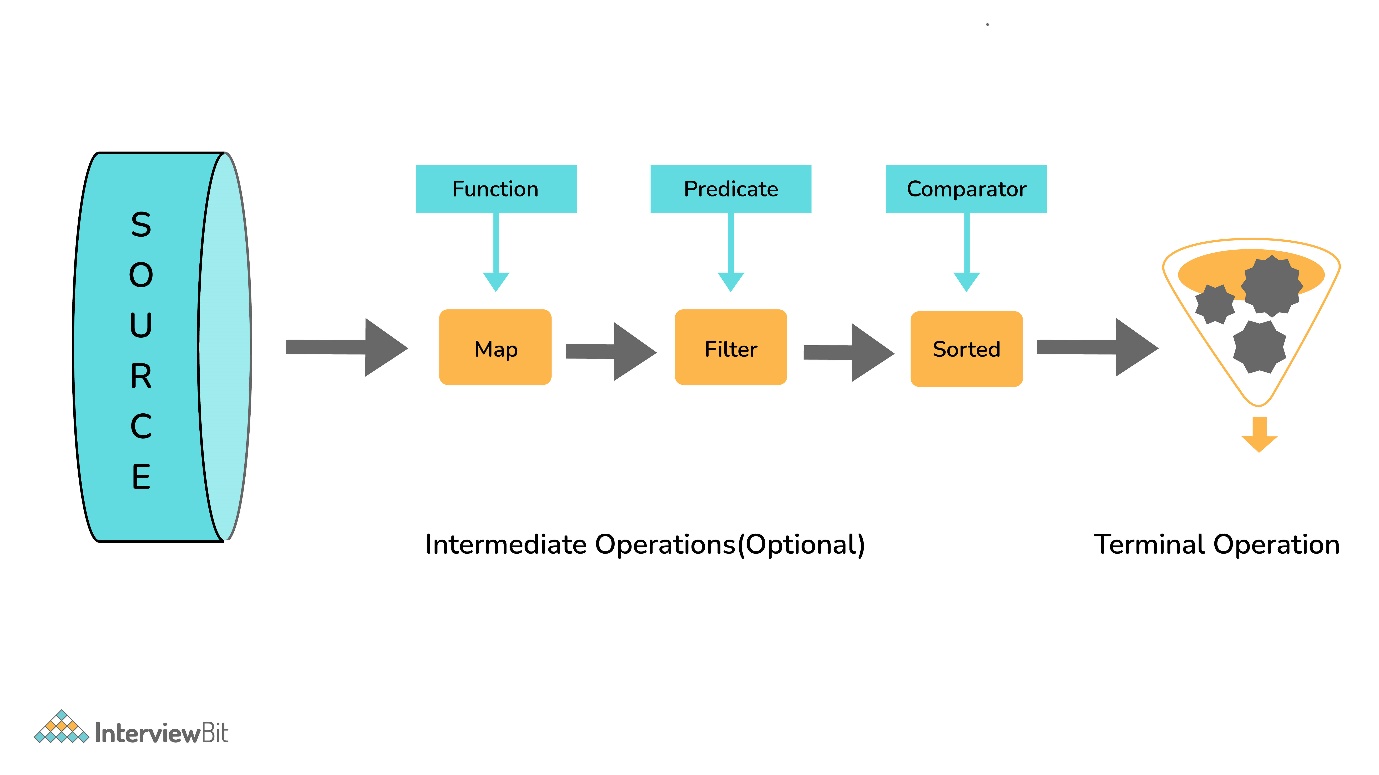
A Stream, which represents a sequence of data objects & series of operations on that data is a data pipeline that is not related to Java I/O Streams does not hold any data permanently.  
The key interface is java.util.stream.Stream<T>. It accepts Functional Interfaces so that lambdas can be passed. Streams support a fluent interface or chaining. Below is the basic stream timeline marble diagram:

Java 8 Streams

### 21. What are the main components of a Stream?

Components of the stream are:

* A data source
* Set of Intermediate Operations to process the data source
* Single Terminal Operation that produces the result

Components of Stream

### 22. What are the sources of data objects a Stream can process?

A Stream can process the following data:

* A collection of an Array.
* An I/O channel or an input device.
* A reactive source (e.g., comments in social media or tweets/re-tweets)
* A stream generator function or a static factory

**Intermediate Operations:**

* Process the stream elements.
* Typically transforms a stream into another stream.
* Are lazy, i.e., not executed till a terminal operation is invoked.
* Does internal iteration of all source elements.
* Any number of operations can be chained in the processing pipeline.
* Operations are applied as per the defined order.
* Intermediate operations are mostly lambda functions.

**Terminal Operations:**

* Kick-starts the Stream pipeline.
* used to collect the processed Stream data.

**int** count = Stream.of(1, 2, 3, 4, 5)

.filter(i -> i <4) // Intermediate Operation filter

.count(); // Terminal Operation count

### 24. What are the most commonly used Intermediate operations?

**Filter(Predicate<T>)** - Allows selective processing of Stream elements. It returns elements that are satisfying the supplied condition by the predicate.

**map(Funtion<T, R>)** - Returns a new Stream, transforming each of the elements by applying the supplied mapper function.= sorted() - Sorts the input elements and then passes them to the next stage.

**distinct()** - Only pass on elements to the next stage, not passed yet.

**limit(long maxsize)** - Limit the stream size to maxsize.

**skip(long start)** - Skip the initial elements till the start.

**peek(Consumer)** - Apply a consumer without modification to the stream.

**flatMap(mapper)** - Transform each element to a stream of its constituent elements and flatten all the streams into a single stream.

### 25. What is the stateful intermediate operation? Give some examples of stateful intermediate operations.

To complete some of the intermediate operations, some state is to be maintained, and such intermediate operations are called stateful intermediate operations. Parallel execution of these types of operations is complex.

For Eg: sorted() , distinct() , limit() , skip() etc.

Sending data elements to further steps in the pipeline stops till all the data is sorted for sorted() and stream data elements are stored in temporary data structures.

### 26. What is the most common type of Terminal operations?

* collect() - Collects single result from all elements of the stream sequence.
* reduce() - Produces a single result from all elements of the stream sequence
  + count() - Returns the number of elements on the stream.
  + min() - Returns the min element from the stream.
  + max() - Returns the max element from the stream.
* Search/Query operations
  + anyMatch() , noneMatch() , allMatch() , ... - Short-circuiting operations.
  + Takes a Predicate as input for the match condition.
  + Stream processing will be stopped, as and when the result can be determined.
* Iterative operations
  + forEach() - Useful to do something with each of the Stream elements. It accepts a consumer.
  + forEachOrdered() - It is helpful to maintain order in parallel streams.

### 27. What is the difference between findFirst() and findAny()?

| **findFirst()** | **findAny()** |
| --- | --- |
| Returns the first element in the Stream | Return any element from the Stream |
| Deterministic in nature | Non-deterministic in nature |

### 28. How are Collections different from Stream?

Collections are the source for the Stream. Java 8 collection API is enhanced with the default methods returning Stream<T> from the collections.

| **Collections** | **Streams** |
| --- | --- |
| Data structure holds all the data elements | No data is stored. Have the capacity to process an infinite number of elements on demand |
| External Iteration | Internal Iteration |
| Can be processed any number of times | Traversed only once |
| Elements are easy to access | No direct way of accessing specific elements |
| Is a data store | Is an API to process the data |

Interview Question: Which is more advantage: String Literals or String Objects “new”?

Ans. String Literals is more advantage because helps in caching things, like, if we create two references with same values, inside String Pool, then only one Object is created, and both reference are made to point to the same Object. So, thus memory is saved, of creating another Object.

ArrayList l = new ArrayList(int initial Capacity) -> This Constructor is used, if we directly know the size of the Array, then we can create an Array directly ins

String x=”10”;

Int x =10

// Converting String to int without inbuilt methods

String num = "27";

**int** result=0;

**for**(**int** i=0;i<num.length();i++) {

result = result\*10 + num.charAt(i) - '0';

System.***out***.println(result);

}

System.***out***.println(result);