* **JVM architecture is very important for Placements**
* Identifiers­­

Rules­­­

1) start with alphabet ,, $(dollar)

2) we can’t start with 0-9

3) $ is used to start in java

or we can’t use dollar to start identifiers in c or c++

* **Variables**

1)) We are using camel case for naming convention like (ranjeetTiwari)

In **methods**.

2) Use pascal Naming convention like (RanjeetTiwari) in **Class .**

3) Use capital letters like (MY\_CONST) in **constant**.

* **Keywords**

1. classes or function that are predefined
2. 49 keyword are predefined in java.

* **Data Type**

**Primitive**

1. int -size(4byte), char -size(1byte) , float - size(4byte),

double - size(8yte)

short - size(2byte), , long - size(8byte), Boolean - size(1bit)

Byte -size(1byte).

1. int java 0 and 1 is not considered as true or false in java

**Non- Primitive**

1. String , array

HOME WORK RANGE HOME WORK SIZE DEFAULT

Minus decimal convert into binary

* **Operators**

1. Arithmetic : + , - , \* , / , %
2. Logical : And , Or , Not .
3. Relational : > , < , => , =< , == , !=
4. Assignment : == , -= , += , \*= , /=
5. Increment, decrement : ++ , --
6. Conditional : condition? (true): (false)
7. Bitwise : left shift ,right shift , & , ^

Bitwise is to increase the speed

* **Loops**

1. For loop

For(initialisation ;condition ; increment / decrement)

{

Statement

}

Use when length is given.

1. While loop

While()

{

statement

}

When don’t know how many times loop runs.

1. Do while

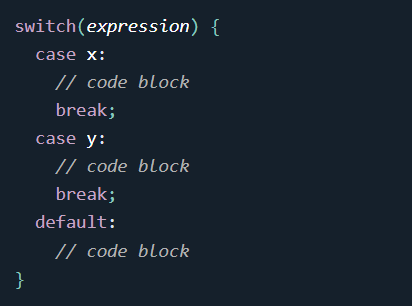
Do{

Statement

}while()

When don’t know how many times loop runs.but at least one time runs.

* **Switch**
* Instead of writing **many** if..else statements, you can use the switch statement.
* The switch statement selects one of many code blocks to be executed:
* Switch is different from if else because if else check every time expression . but in switch the only checking on value.

****

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.
* The break and default keywords are optional, and will be described later in this chapter

The break Keyword

* When Java reaches a break keyword, it breaks out of the switch block.
* This will stop the execution of more code and case testing inside the block.
* When a match is found, and the job is done, it's time for a break. There is no need for more testing.

The default Keyword

The default keyword specifies some code to run if there is no case match:



* **ARRAY**
* Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
* To declare an array, define the variable type with **square brackets**:

String[] cars;

1. Array is object new keyword is use for initialisation.
2. Int [] arr={1,2,3,4,5,5}

in this condition new keyword is not used so java is automatically correct this and apply new in backend.

1. Int arr[]=new int[5];

The size of array is 5.

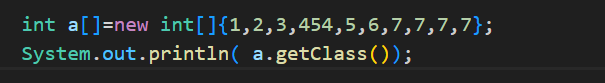
1. **Int []a={1,2,3,44,5,};**

**Or**

**Int a[]=new int[]{1,2,3,454,5,6,7,7,7,7}**

**Array is object.**

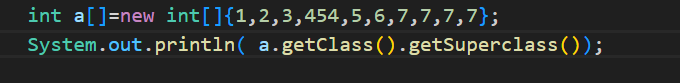
**5) We can test with a.getClass()**

****

**Output**

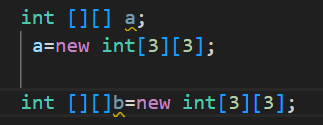
****

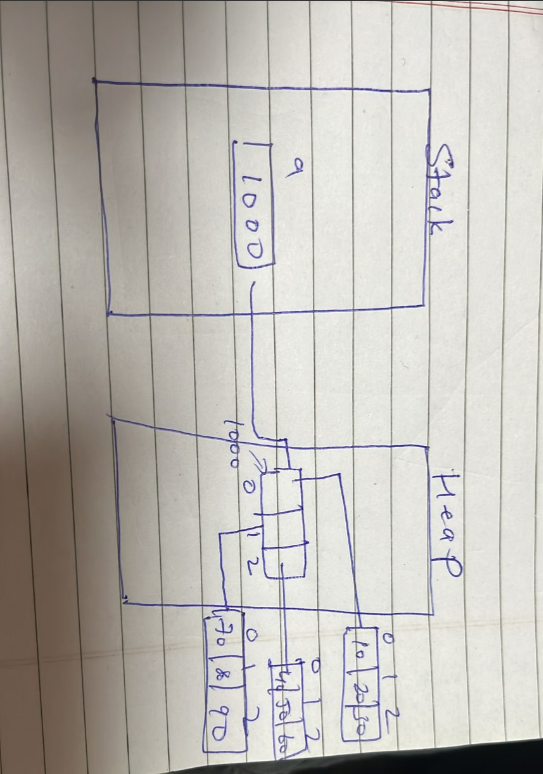
**I is data type of array . if array is byte type then B.**

****

* **2D array**

**Pointer in java but internally**

****

****

**Int this case data is store**

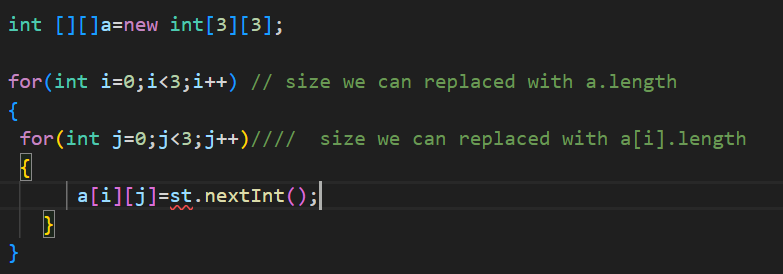
Also store a[0][0]=10

**Or**

Int [][]a={{2,34,45},{55,67,,878}{9,899,9}}

Int c++ column size define is compulsory

**If we declared**

****

C++ is not support boundary checking. If size we declared 3\*3

But we are inserting 4\*4 . then c++ do.

but in java . java is not do this.

This is work on fix size.

Int java Array indexOutOf bounds supports

Array uses properties

a.length. length Is not a method

**Jagged Array**

1. In jagged array column can variated

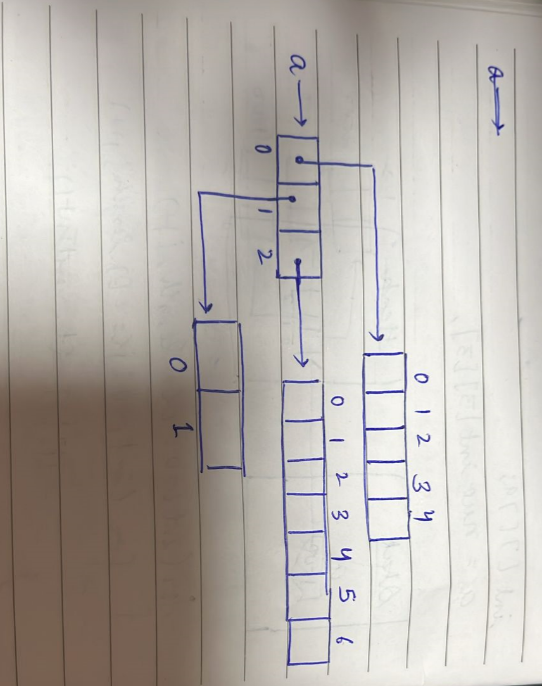
Int [][]a=new int[3][];

1. In jagged array row size is compulsory. But column size is not given .its given but row by row.

a[0]=new int[5]

a[1]=new int[2]

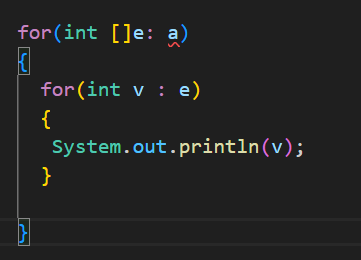
a[2]=new int[7]

****

**Enhanced for loop**

****

Enhance for loop is used only for data accessing. not for updating the data.



**This is defined java.language**

**Inbuilt class**

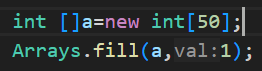
1)For compare two array.



2)



3)



4)



> for descending order / which algorithm is used for sorting / complexity

**5)**

****

**>** it return the first occurrence.

If the element is not present then it return the -5 index

* **Classes and Object**

Placement Question

1)how a language is oops based

Ans; class and object

And support

Encapsulation, abstraction, inheritance, polymorphism

1. difference between c and c++.

Encapsulation:- wrapping of data and allowed behaviour single unit.

Why we are using encapsulation

* Important component are secured
* for data security.

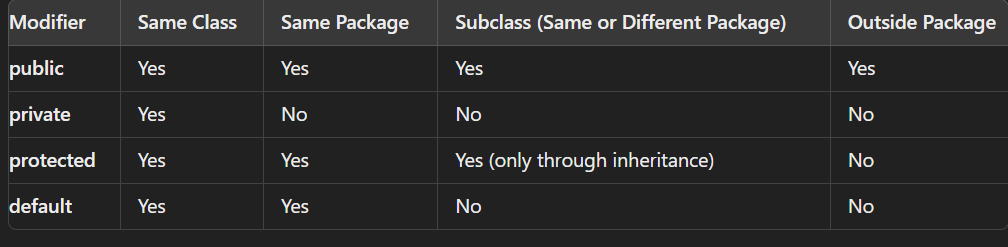
Java is class is the example of encapsulation.

* **Abstraction:**
* **Definition:** Abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object to the user. This allows the user to interact with the object without needing to understand the underlying complexity.
* **Rule of Thumb:**
  + Data Members: Should be declared as private to protect the internal state of the object from unauthorized access and modifications.
  + Member Functions: Should be declared as public to provide controlled access to the data members and to expose only the necessary operations to the user.
* **Access Modifiers**

Access modifiers in Java determine the visibility and accessibility of classes, methods, and variables. The four main access modifiers are:

* **public:**
  + Visibility: The class, method, or variable can be accessed from any other class in the program, regardless of the package.
  + Use Case: Use public when you want to expose a class, method, or variable to all other parts of the application.
* **private:**
  + Visibility: The class, method, or variable is accessible only within the class where it is defined.
  + Use Case: Use private to encapsulate and protect the internal state of a class, ensuring that it cannot be accessed or modified directly from outside the class.
* **protected:**
  + Visibility: The class, method, or variable is accessible within its own package and by subclasses (derived classes) in other packages.
  + Use Case: Use protected when you want to allow subclasses to access certain members of a class while still restricting access from non-related classes.
* **default (no modifier):**
  + Visibility: The class, method, or variable is accessible only within the same package. This is also known as "package-private" access.
  + Use Case: Use the default access modifier when you want to restrict access to other classes in the same package but not to subclasses outside the package.

**Summary Table:**

****

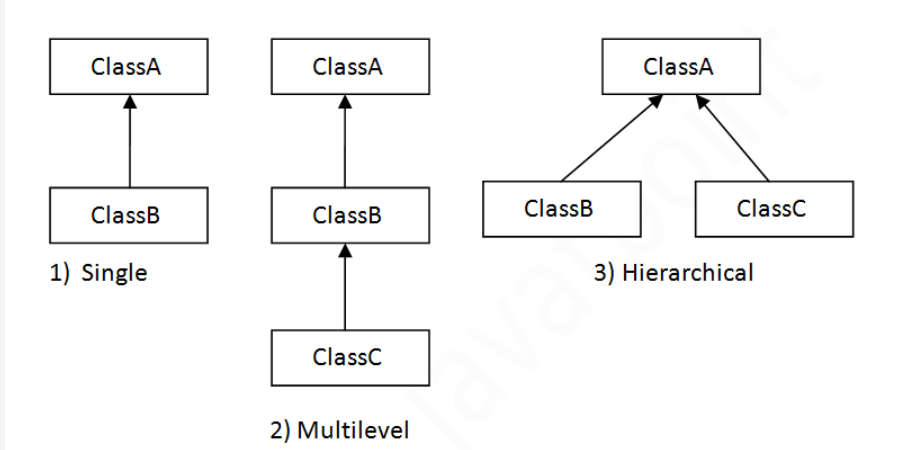
* **Inheritance :-**
* Inheritance represents the IS-A relationship which is also known as a *parent-child* relationship.
* Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system)
* when one object acquires all properties and behaviours of parent object, it is known as inheritance

**Why use inheritance in java**

* For [Method](https://www.javatpoint.com/method-overriding-in-java)  (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.
* In C++ we use colon (:) . but in java extends keyword is used .

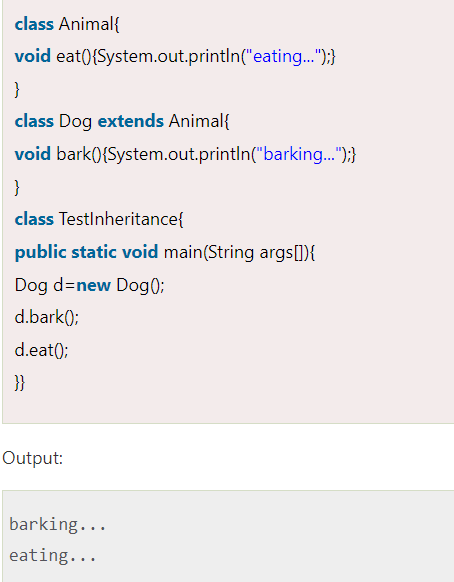
Extends:- The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality

* **Types of inheritance in java**
* There can be three types of inheritance in java:
  1. single
  2. multilevel
  3. hierarchical.
* In java programming, multiple and hybrid inheritance is supported through interface only

****

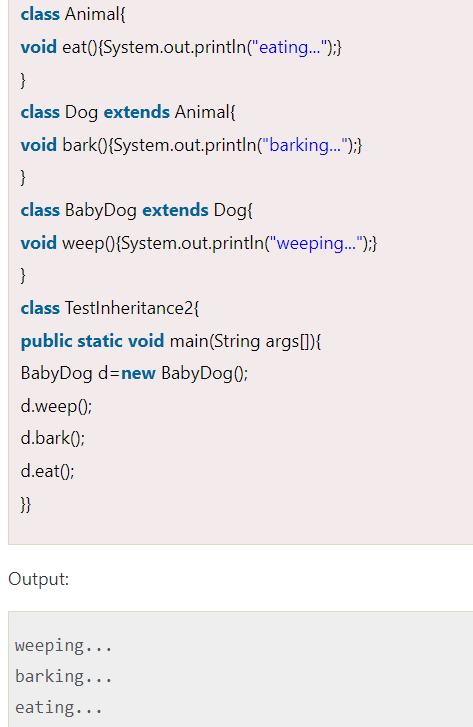
* **Single inheritance**

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

****

* **Multilevel-Inheritance**

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.



## Hierarchical Inheritance

## When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

## 

## Interview Question:

## Why multiple inheritance is not supported in java?

## Ans) To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

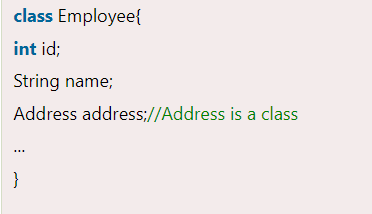
## Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

## Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

## 

## Msg is defined in both classes .

* **Aggregation:-**
* If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.
* Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.



In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

* For Code Reusability.
* When use Aggregation?
* Code reuse is also best achieved by aggregation when there is no is-a relationship.k
* **Polymorphism:-**

# **Method Overloading in Java**

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as Method Overloading.

* **Different ways to overload the method**

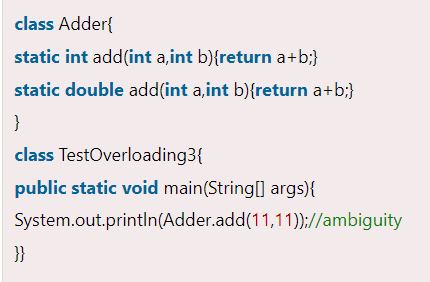
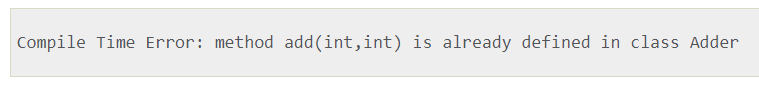
There are two ways to overload the method in java

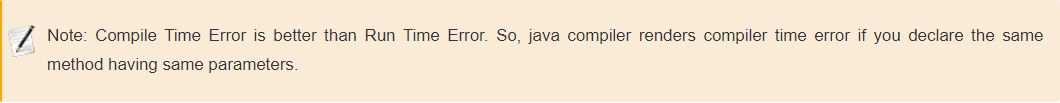
1. By changing number of arguments
2. By changing the data type of argument.

* **InterView Question**

### Why Method Overloading is not possible by changing the return type of method only?

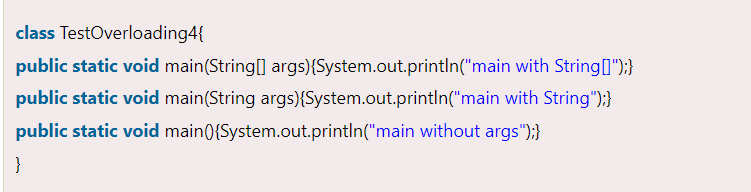
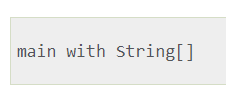
Ans-) In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:



### **Can we overload java main() method?**

Ans- Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) calls main() method which receives string array as arguments only. Let's see the simple example:

**Class Notes**

* **in java method overloading is use but operator overloading is not support.**

**Method overloading :- same name of method in same class box.**

**In one box compeletely same method is not use . but we change the prototype or signature.**

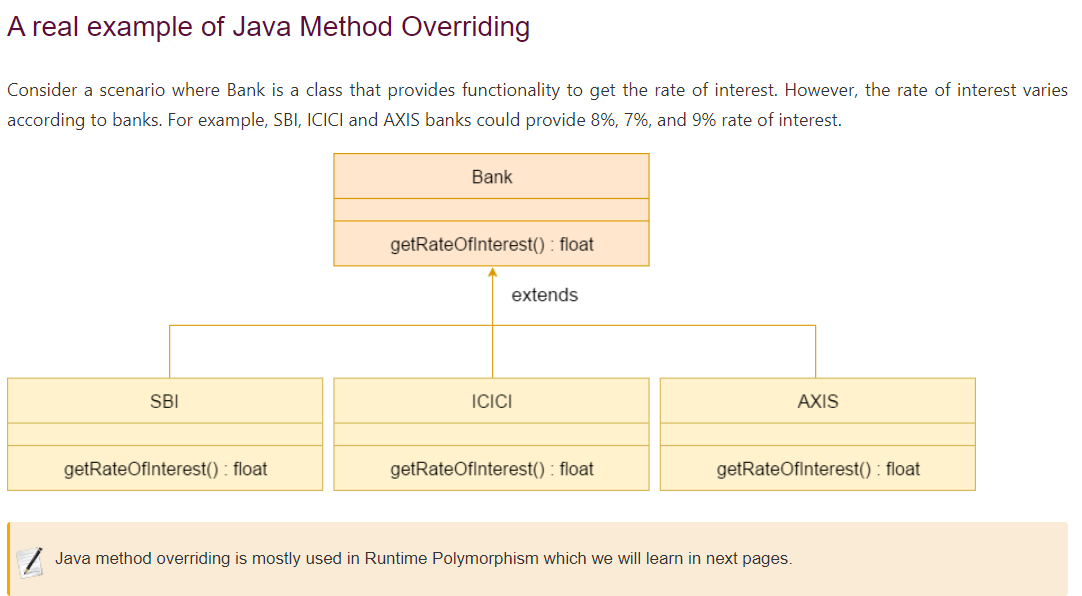
**Number of parameter or change the datatype.**

* **Method in Java**

If subclass (child class) has the same method as declared in the parent class, it is known as method in Java.

* **Usage of Java Method Overriding**
* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism
* **Rules for Java Method Overriding**

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).



* **InterView Question**

### **Can we override static method?**

### Ans- No, a static method cannot be overridden. It can be proved by runtime polymorphism.

* 1. **Why can we not override static method?**

Ans- It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

* 1. Can we override java main method?

Ans- No, because the main is a static method.

* **Object: a class is a collection of field**

**an entity that has state and behaviors is known as object**

**Object has three characteristic:**

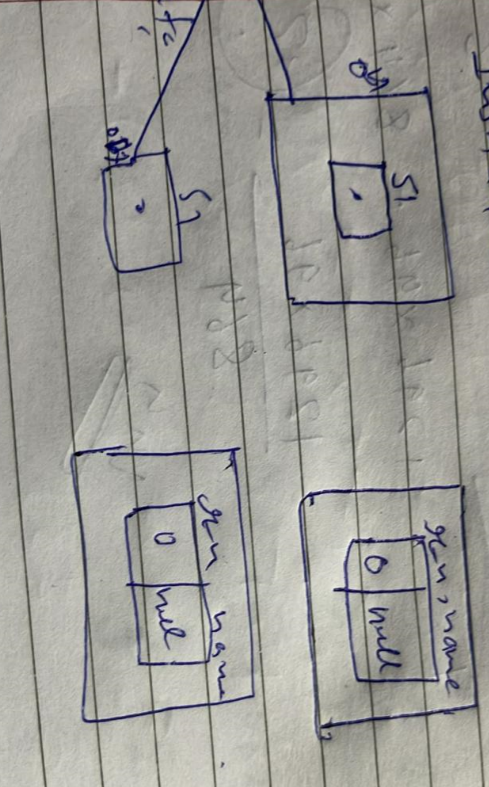
**State: it represented as attribute of class**

**Behavior:**

**Identity:**

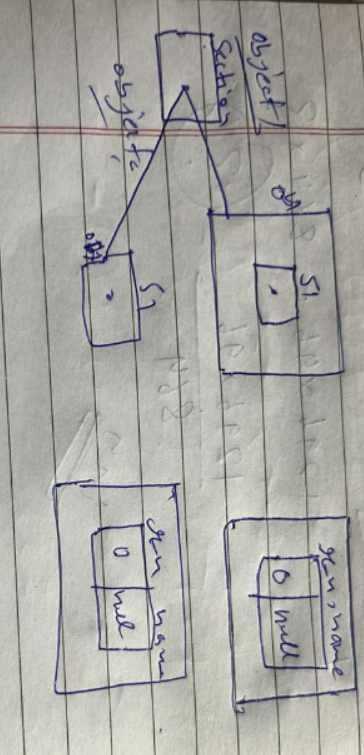
* **Three types of variables in class**

1. **Instance variable : different memory as per object.**

****

1. **Local variable – the variable that act as parameters. Eg in getData () method a,b**
2. **Class variable – the variable that use with modifiers. But in this case**

**Static keyword is used to define a variable that is use a same memory.**

****

**Class student**

**{ static int section**

**Int r\_n;**

**String name:**

**Public void getData(int a ,string b)**

**{**

**r\_n=a;**

**Name=b**

**}**

**Public void putdata()**

**{**

**System.out.print(r\_n \_ name)**

**}**

**}**

**Student s1=new Student();**

* **Creating an object**
* **Method :- mehtoad are function and procedure or sub -routines.**

**To excute a method , you call it from another methoad.**

**.) there is no rule to define parameter in methods**

* **( ) :- function call operator**
* **Method overloading:-**

**If two or more in a class have same name but different parameter. It is known as method overloading.**

**If If two or more in a class have same parameter but differs in return type not said to be overloaded methoad.**

* **Constructor:-**

**1) Special type of method .that are calls automatically .**

**2) It has no return type.**

**3) Same name as class**

**Types of construction:**

1. **No. arg constructor**
2. **Parameterised**
3. **Non-parameterised construction**

**ClassName()**

**{**

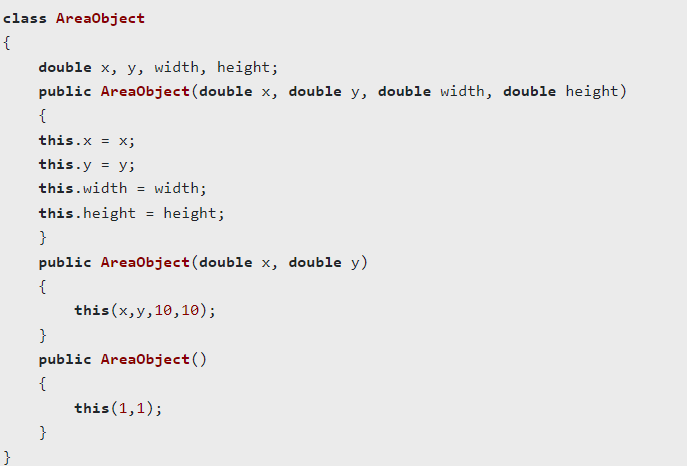
**}**

* **(This ) keyword behaving as pointer.**

**It acts as a reference variable in Java that represents the current object.**

**1)This resolve the naming conflict.**

**Between instance variable and local variable.**

****

**Eg.**

**Class A**

**{**

**Int a;**

**A(int a)**

**{**

**This.a=a;**

**}**

**}**

* **This is also use for contructor**

**Class A**

**{**

**Int a;**

**A()**

**{**

**This(a,b);**

**}**

**A(int x,int y)**

**{**

**a=x;**

**b=y;**

**}**

**}**

1. **For using same class method we are using this key word**

**Class student**

**{**

**Int rn**

**String name;**

**Void getData()**

**{**

**rn=10;**

**name=”abc”;**

**this.print() // that’s why this is using**

**}**

**Void print()**

**{**

**S.o.p(rn + “ ” + name);**

**}**

**}**

**Public class Test**

**{**

**P . s . v. main(string []s)**

**{**

**Student s1=new Student();**

**S1.getData() //this=&s1**

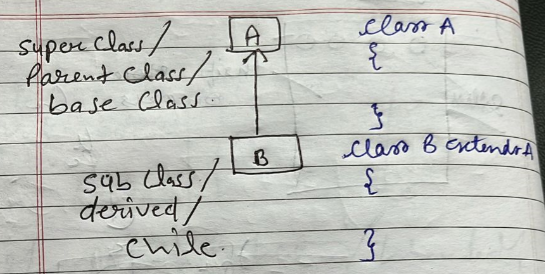
**}**

**}**

* **In constructor by default have a constructor**
* **INHERITANCE**

**it is a mechanism in which one object acquires all the properties and beahaviors of parent object.**

**Ineheritance define the relationship bet superclass and sub class**

****

**Class A**

**}**

**Class B extends A**

**{**

**}**

* **Java uses extends keyword but in c++ uyse : .**
* **Java is not support multiple inheritace directelly but indirectely supports.**
* **hybrid inheritance is not support because of diamond problem**
* **More than multiple class and same name or same signature .**

**One problem is arrises Overriding problem.**

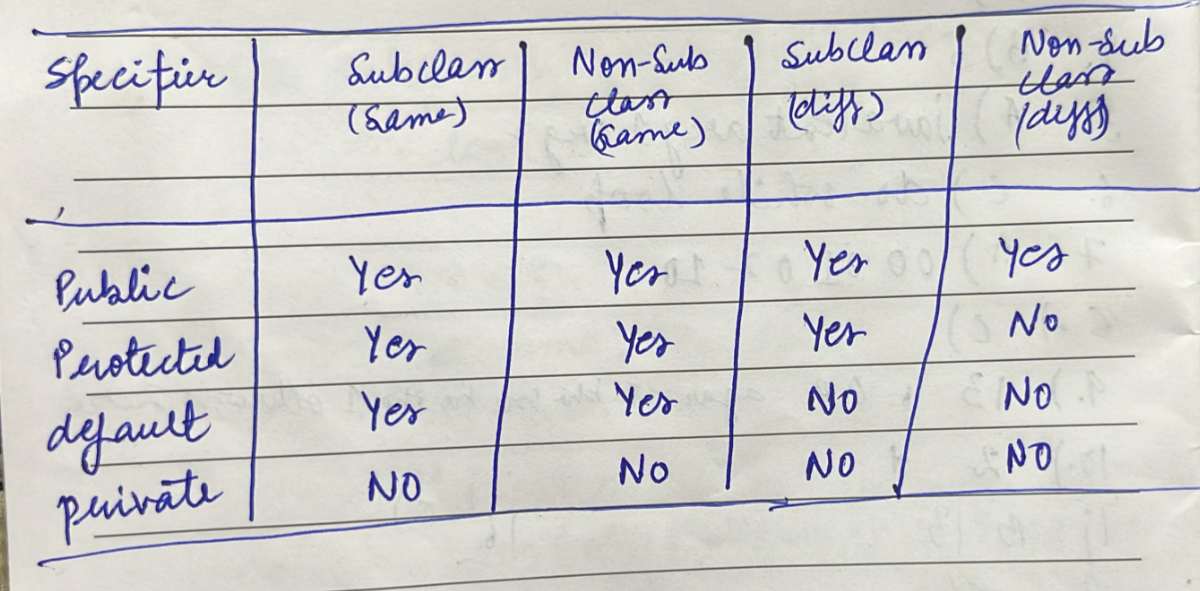
* **Access specifier**

**1)public –**

**2)protected – it is modifier that allies between private and public**

**3)default**

**4) private**

****

**If we are not defining any modifier then by default it uses (default) modifier.**

**Sub class means – child class**

**Non subclass – simple class**

**Class A**

**{**

**Void print();**

**}**

**Class B extend A**

**{**

**Void print();**

**}**

**We create object of child class**

**B obj=new B();**

**b.print() // it print the local method that are Class of B;**

**but B class inherit the functionality of class A. what is the benefit of inherit. Why we are inherit the super class’s methoid is not execute.**

**That why we are using super to execute super class’s method.**

**Local version of class is execute in case of overriding.**

**This problem is solved by super keyword**

**Contructuctor is required in both class base class and child class.**

* **It is responsibility of child class to call the constructor of base class.**

**That why we are using**

**In child class constructor to call base call constructor**

**That is super() this is put in child class ‘s constructor**

* **Final keyword**

**Class A**

**{**

**Final Int a;**

**Final Void print()**

**{**

**S.O.P(a);**

**}**

**}**

1. **With final we create constant .**
2. **Not override . that it gives the error**

**Eg. Class A**

**{**

**Final Int a;**

**Final Void print()**

**{**

**S.O.P(a);**

**}**

**}**

**Now we creating**

**Class B extend A**

**{**

**Void print(); /// that are not allowed because method is iniitilise in base class with final   
}**

1. **If we apply final keyword on class . it restrict the class to inherit .**
2. **Eg.**

**Final Class A{**

**}**

**Class B extends A // this is not allowed because of final keywords**

**{**

**}**

**--------------------- Dynamic Dispatch--------**

**We creating a object of child class intead of base class.**

**Class Test{**

**Int a;**

**Public Test()**

**{**

**a=10;**

**}**

**Public void print()**

**{**

**S.O.P(a);**

**}**

**}**

**Class demo extends Test{**

**Int b;**

**Public Demo()**

**{**

**Super();**

**b=20;**

**}**

**Public void print()**

**{**

**S.O.P(b);**

**}**

**}**

**Class Example{**

**P.S.V. main(string[] a)**

**{**

**Test d1= new Demo()**

**D1.print() ;//it print the demo class// it is the dyanamic dispatch**

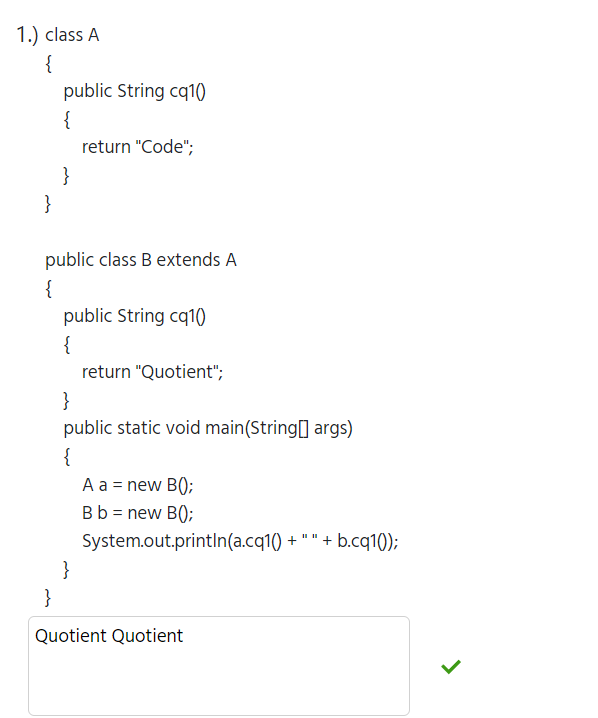
**Test d2=new Test()**

**D2.print()//it print parent classggg!!!!**

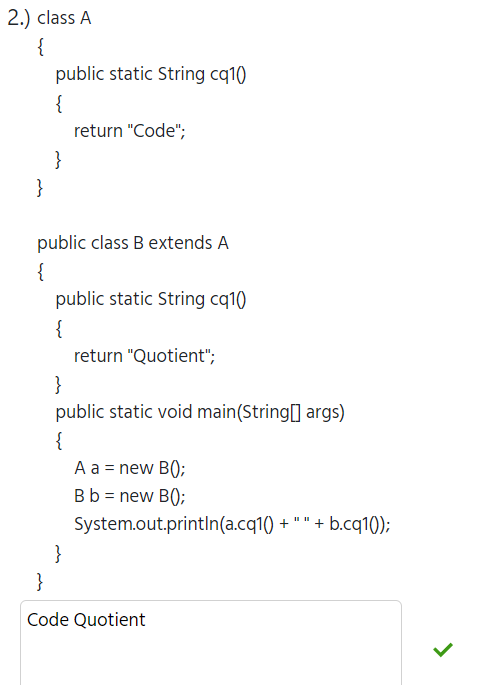
**}**

**}**

**Static methods are not polymorphic. This means that the method called is determined at compile time based on the reference type, not the object type.**

****

**In this case (a) is reference of A and object of B. and when we call (a.cq1()). Then it call to the object of class**

****

**But if cq1() method are static then things does’t like Previos .**

**Int which the reference of (a) is A and object is B .. then we call a.cq1().**

**Then it call to the reference type class.**

**Bitwise operator**

* **Binary**

1. **OR**
2. **AND**
3. **EXOR**
4. **SIGNED RIGHT SHIFT**
5. **SIGNED LEFT SHIFT**
6. **UNSIGNED RIGHT SHIFT**

* **Unary**

1. **N (complement)**

**Precedence**

1. **~ --right to left**

**Eg. ~N=-(N+1)**

**public static void main(String[ ]a)**

**{       System.out.println(~(-9));**

**}**

1. **>>, << - left to right**

**Left shift**

**a<<b = a\*pow(2,b)**

**Right shift**

**a>>b = a/pow(2,b)**

**in right shift last bit is filled on the basis of signed bit**

1. **& -- left to right**
2. **^ -- left to right**
3. **| -- left to right**

**public class IntegerMethod {**

**public static void main(String[ ]a)**

**{**

**System.out.println( Integer.toBinaryString(9));**

**}**

**}**

**Output: 1001**

**ABSTRACT**

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

* Abstraction in Java

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the [object](https://www.javatpoint.com/object-and-class-in-java) does instead of how it does it.

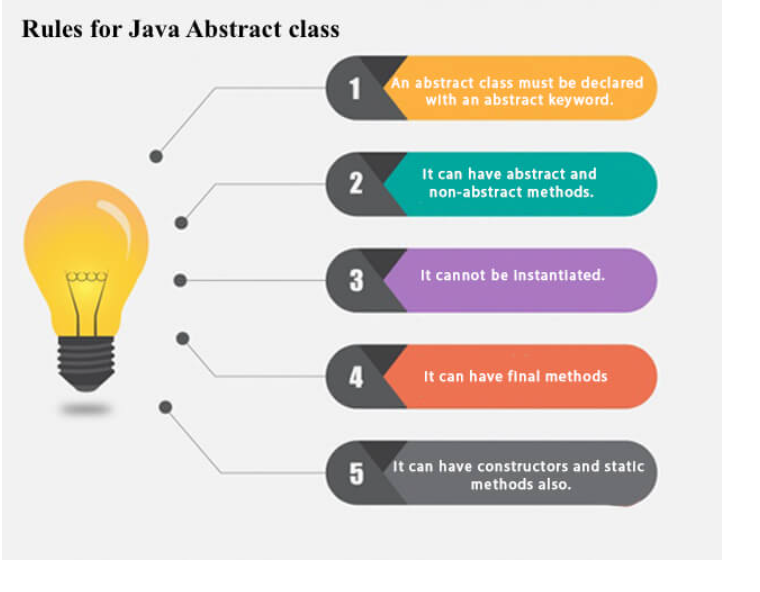
Ways to achieve Abstraction

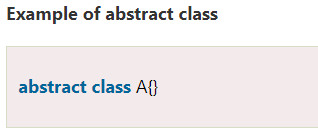
There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

* **Abstract class in Java**

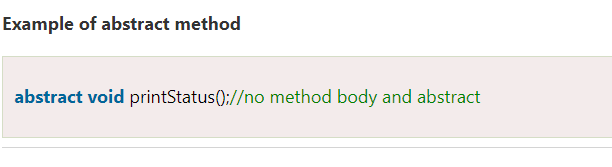
A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.



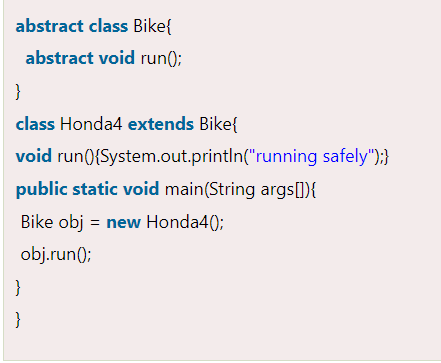


* **Abstract method in java**

A method which is declared as abstract and does not have implementation is known as an abstract method.



* Example of Abstract class that has an abstract method

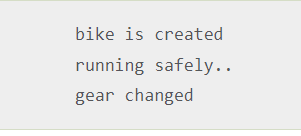


* Real Life Scenario

### **Abstract class having constructor, data member and methods**

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

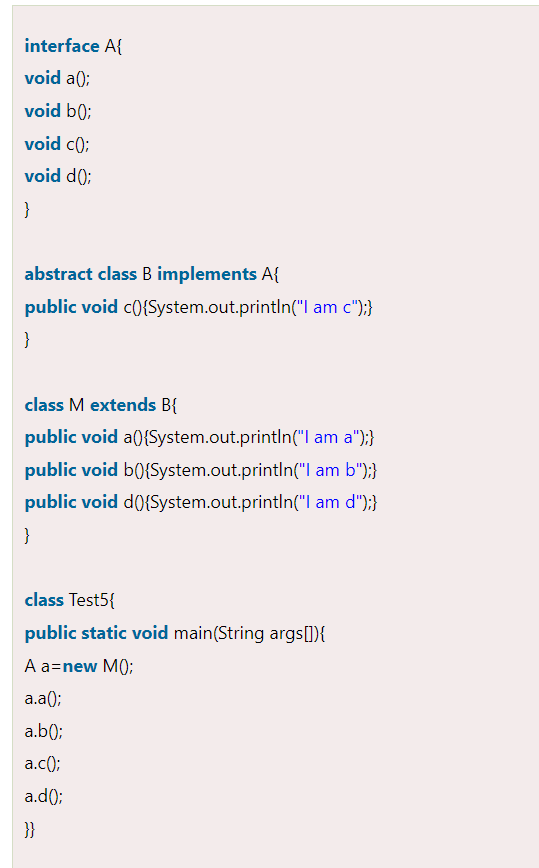




* Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the [interface](https://www.javatpoint.com/interface-in-java). In such case, the end user may not be forced to override all the methods of the interface.





Class Note

**For decalaration only**

**Abstract-**

* 1. a method without any definition.

2) Compulsorily define in child class.

3) Class- abstract class. – have at least one abstract method.

Class can be also a abstarct

1. The object cannot create of abstract class type.
2. If a class have all methods are abstract that class having 100% data hiding.

**CLASS A**

**{**

**abstract void print(); //there is no need of statement**

**}**

**CLASS B extend A**

**{**

**Void print()**

**}**

**Print method is must be override in subclass**

* **INTERFACES**
  + 1. An interface in Java is a blueprint of a class. It has static constants and abstract methods.
    2. The interface in Java is *a mechanism to achieve*[*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).
    3. Java Interface also **represents the IS-A relationship**.
    4. In java method are public and abstract by default.

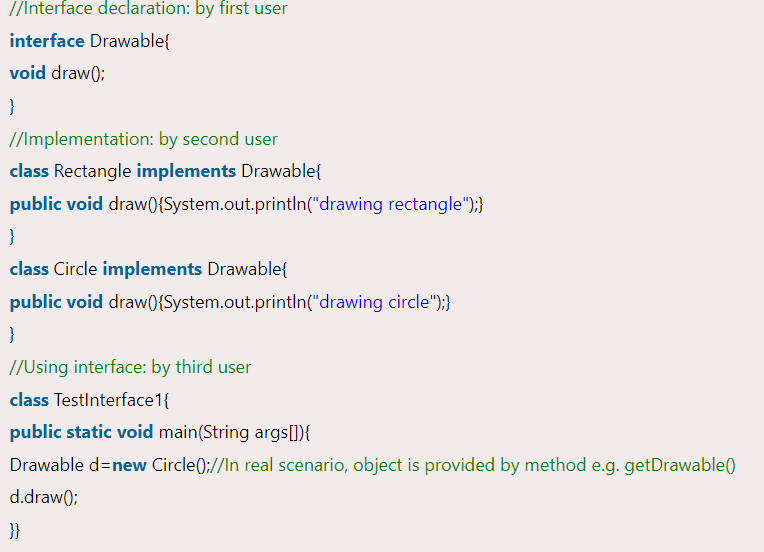
## How to declare an interface?

## An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

## 

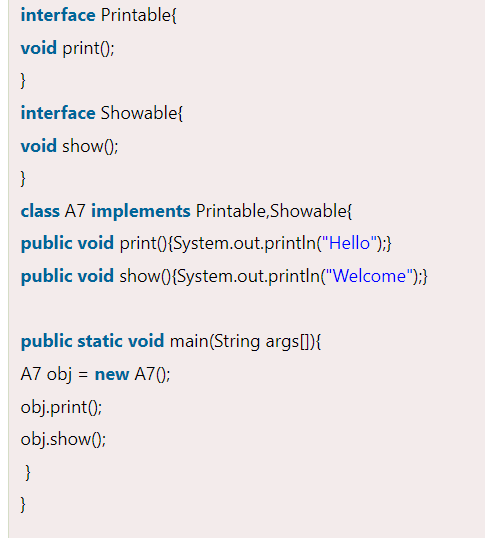
* Java Interface Example: Drawable

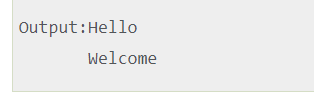
In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.



## Multiple inheritance in Java by interface

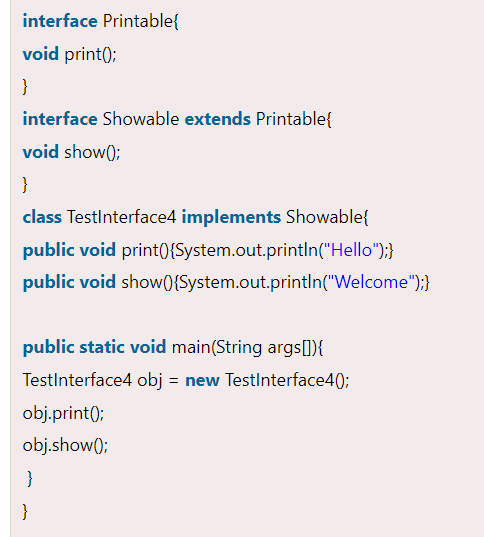
If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

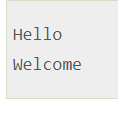




* Interface inheritance

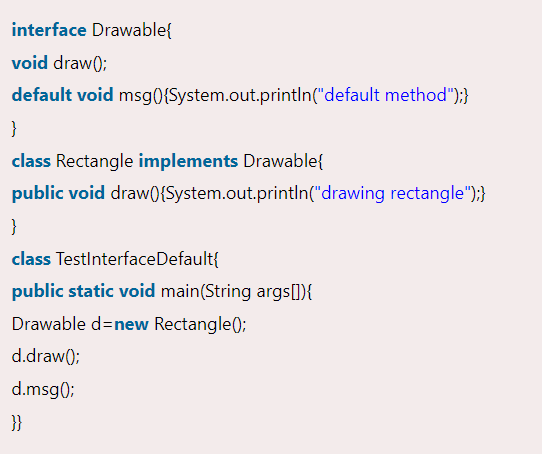
A class implements an interface, but one interface extends another interface.

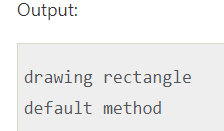




* Java 8 Default Method in Interface

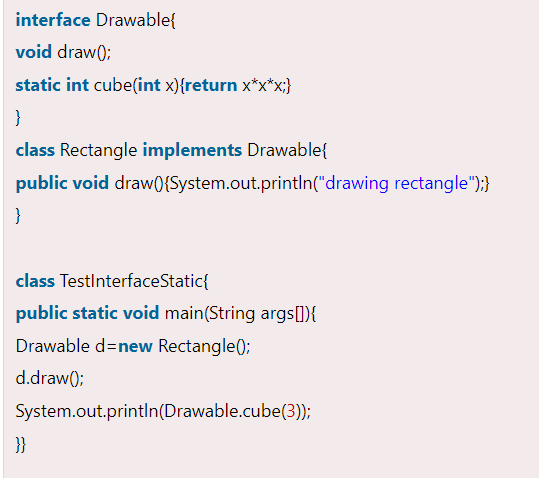
Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

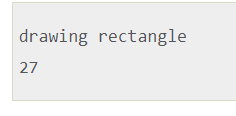




* Java 8 Static Method in Interface

Since Java 8, we can have static method in interface. Let's see an example:

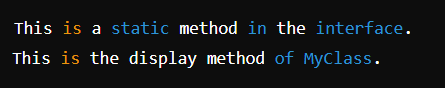




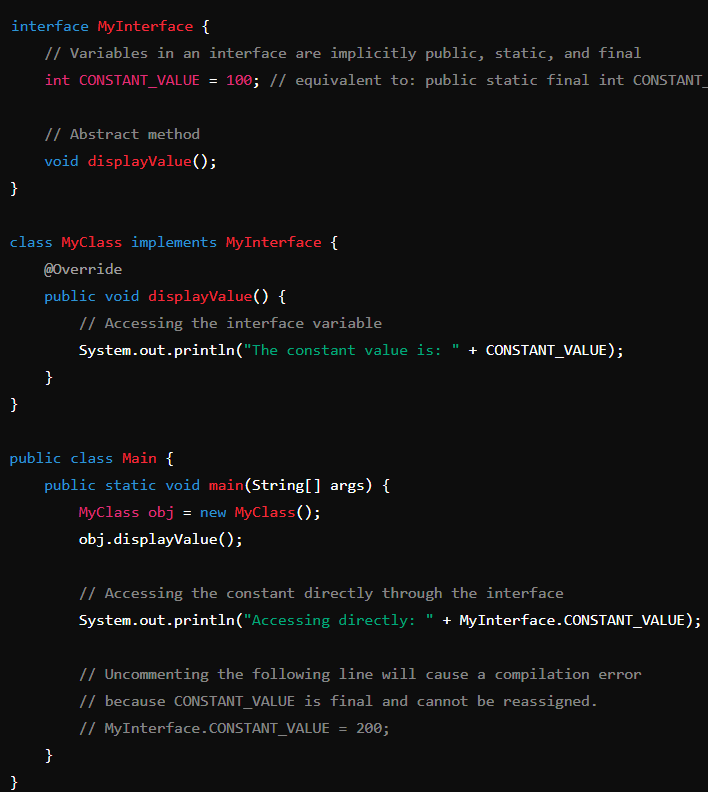
* 1. It more understandable



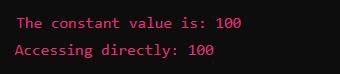
Out put



* Java 8 Varibles are Static ,final and public



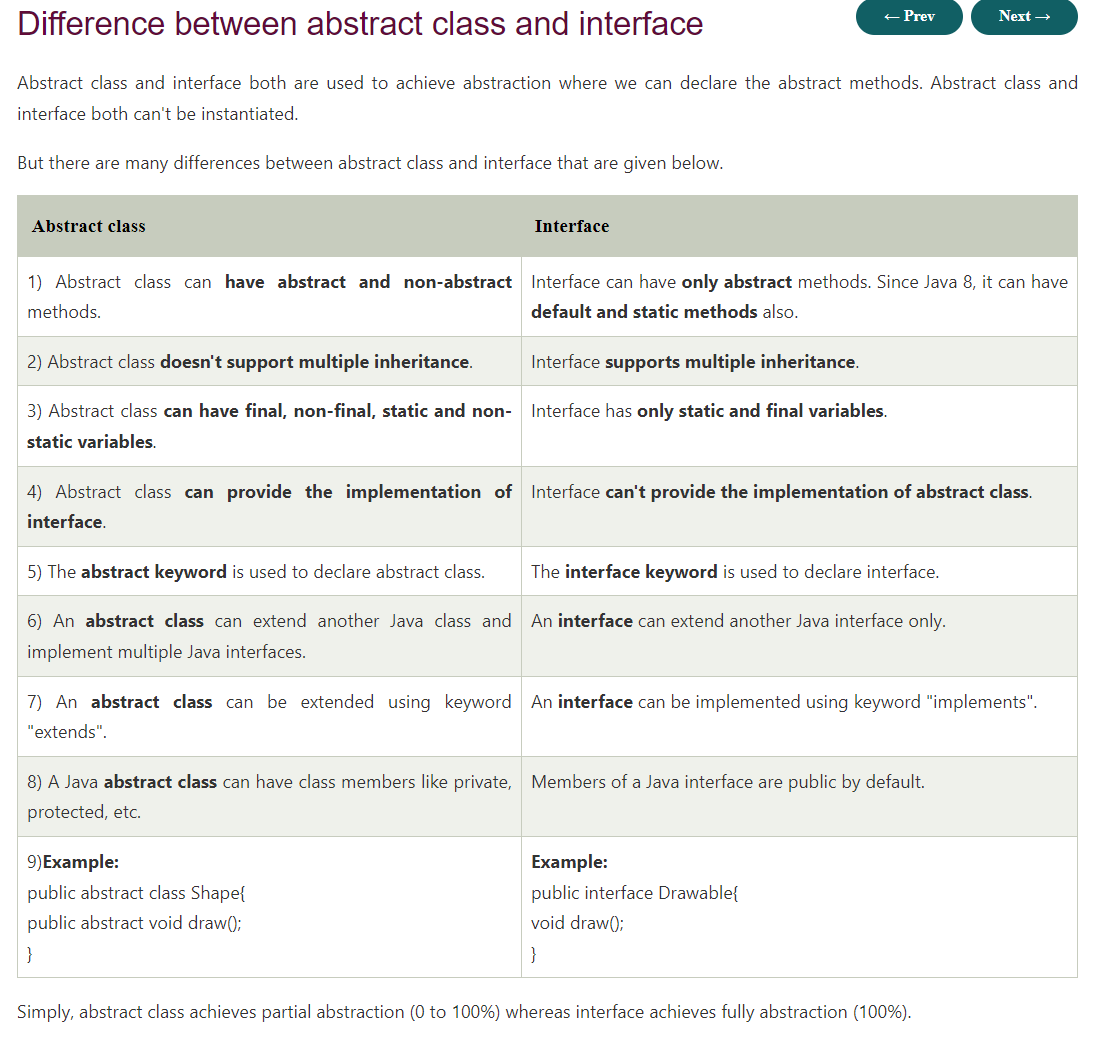
Output:



Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.



**Class Notes**

**1)Blueprint**

**2)General Structure**

**3)Has A**

**4) 100% abstraction – all the methods are abstract**

**5) to use Interface we are using keyword (Implement).**

**6) how to create interface.**

**We are using keyword interface.**

**Eg. Interface interfaceName**

**{**

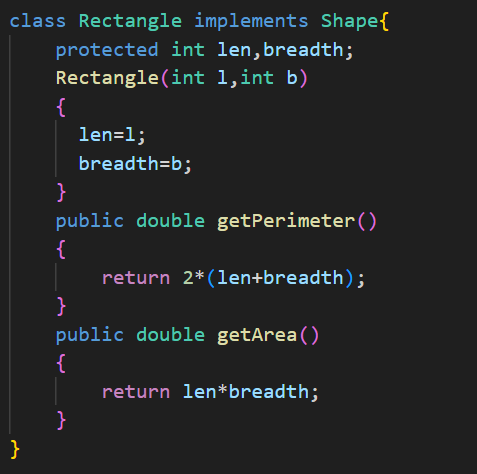
**Data members // by default (public ,final , static )**

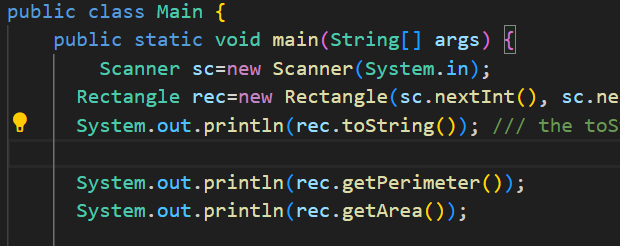
**Member function // by default (public,abstract) but in java 8 it can be default, static;**

**}**

**7) The object is not create of Intercace.**

* **Things that are arises at the time of practice**

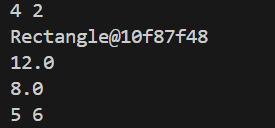
****

****

Ques) Inside main creating object and calling toString method

But I not implement toString method in Rectangle class

But it give the ouput of toString() method why ?



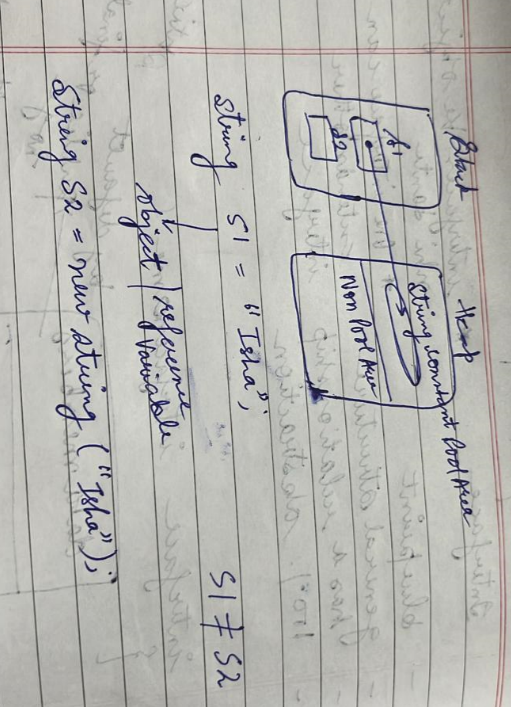
Ans.)

1. When you create a class like Rectangle without explicitly defining a toString() method, the default toString() method from the Object class is used.
2. returns a string that consists of the class name followed by the "@" character and the hexadecimal representation of the object's hash code

**STRING**

1. **String class is used to create string object .**
2. **In c++ String has no null character at last if we create with String keyword**
3. **Declaration**

* **String s1=”Ranjeet”**
* **String s2=”Ranjeet” // will not create new instance**

****

1. **Java strings are immutable.it cannot modifiable in change place**

**Eg**

**S1=”RAnjeet”/// A can’t change**

1. **String s2=new String(“Ranjeet”)**

**It goes to the non-pool area**

1. **S1.equal(S2) // if S1,S2 is Created with litral for check only content**
2. **String s1=”Isha”;**

**String s2=”Sharma”**

**S1=s1+s2;**

**JVM uses string buffer concept behind it**

1. String s=new StringBuffer("").append(s1).append(s2).toString();
2. **Java is safe it is not giving the exact address. If we want to extract we cant use  System.out.println(s.hashCode());**

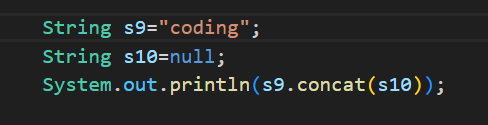
**It generate the hashcode of adsress**

1. **Java String Class Methods**

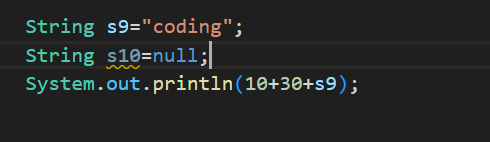
* **substring()**

**eg.**

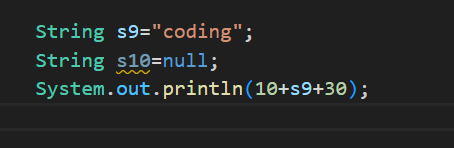
1. **Concate**

****

**Gives Error because it cannot concate NULL**

****

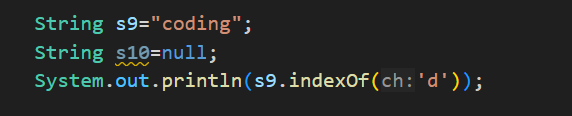
**Ans It gives 40coding**

****

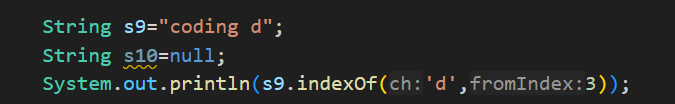
**Ans 10Coding30**

1. **IndexOf()**

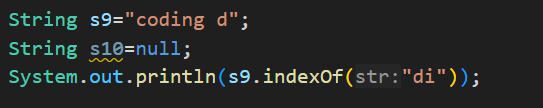
**Eg. Return the index of character. and give the first occurance**

****

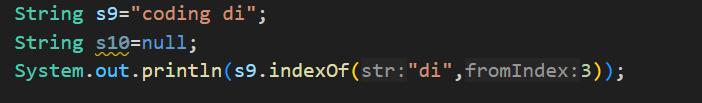
**Ans – 2**

****

**Ans – 7**

****

**Starting index of string that found**

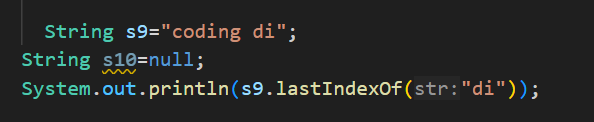
****

**Ans – 7**

1. **LastIndexOf()**

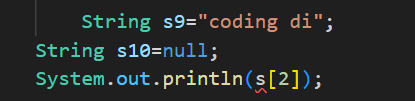
**It search from the and. If character or sting is not found then gives the**

**Answer is -1**

****

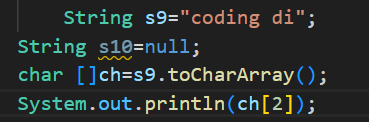
**Ans – 7**

1. **toCharArray()**

****

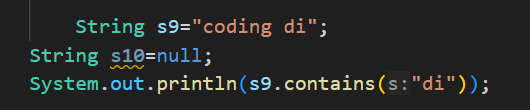
**Gives error because because string cannot use as array**

**For using convert into array**

****

**Ans is d**

1. **Contains()**

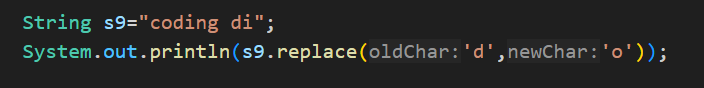
****

**Ans true**

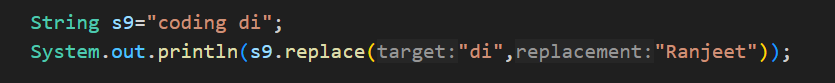
**Return the answer in form of Boolean**

1. **Replace**

**It is replace all the occurance**

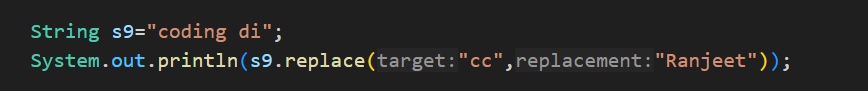
****

**Ans – cooing oi**

****

**Replace the location**

**Ans- coRanjeeting Ranjeet**

****

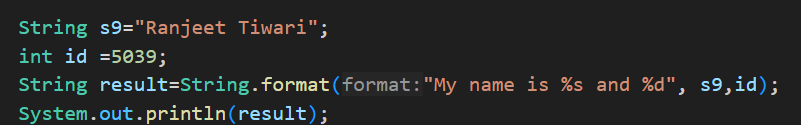
**cc is not found in s9 then it gives the same string in syso**

1. **String.format**

**String formatting allow us to control the presentation of text and numeric value in the string.**

**It is static method**

**Format method has system class**

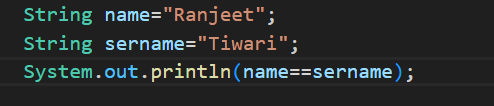
****

**Output**

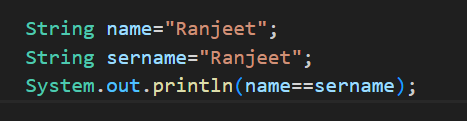
****

1. **==**

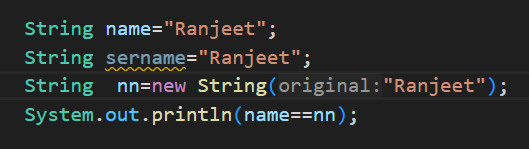
**It check the address**

****

**Output is false**

****

**True because Ranjeet at one address it same that’s why**

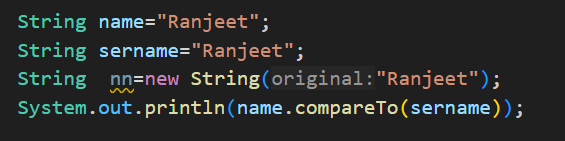
****

**Ranjeet create at new address it give False answer**

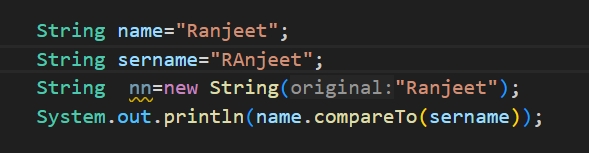
1. **CompareTo()**

**The java compares two string based on the inocode value of each character in string.**

**Output in Integer**

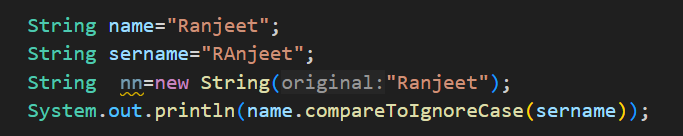
****

**output is 0 . if it is 0 then the string is equals else not equal;**

****

**Output**

****

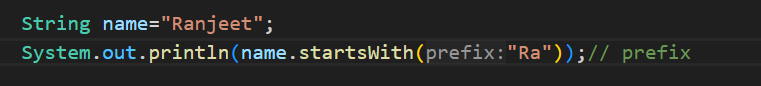
****

**output**

****

1. **startWith()**

**for prefix checking**

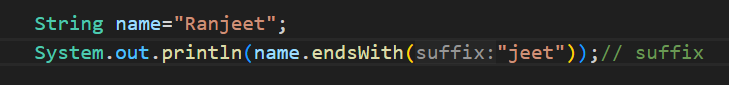
****

**Ouput**

****

**In this ignore case is not used**

1. **endWith()**

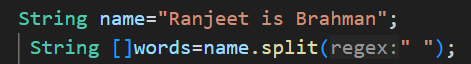
****

**Ouput**

****

**In this ignore case is not used**

1. **Split()**

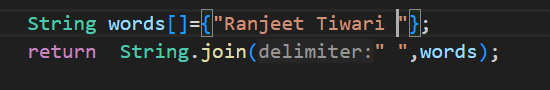
****

**It gives the Array**

****

**Words look like this.**

1. **Join()**

****

**Output**

****

**The output is display with space or in string form.**

* **StringBuffer()**

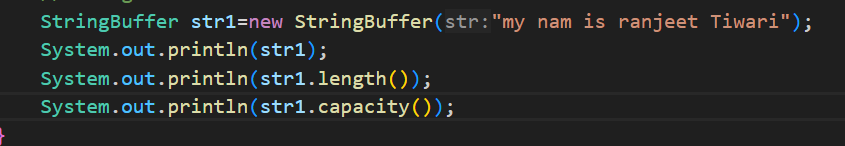
1. **StringBuffer is Synchronised .. means sequence by sequence.**
2. **StringBuffer take more time run.**

**Because of synchronised.**

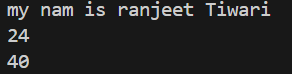
1. **thread safe. Because it is Synchronized.**
2. **it is not java,lang pakage no need to import.**

**Example**

1. **Length() , capacity()**

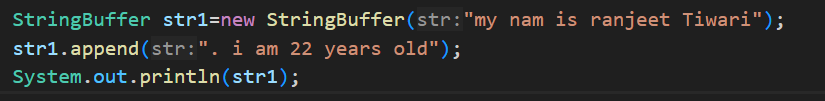
****

**Output**

****

**Capacity is 40 because 24 is length and by default it is 16 so answer of capacity is 40**

1. **Append()**

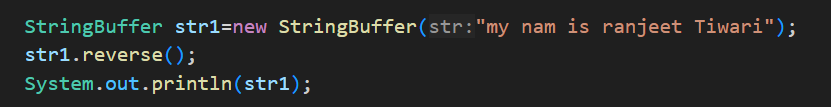
****

**Output**

****

**­**

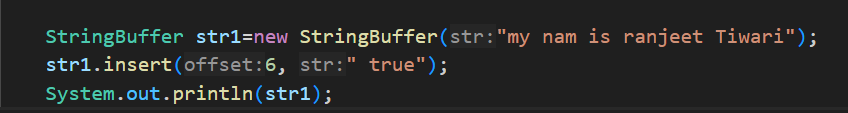
1. **Reverse()**

****

**Output**

****

1. **Insert()**

****

**Output**

****

* **StringBuilder**

1. **Mutable String**

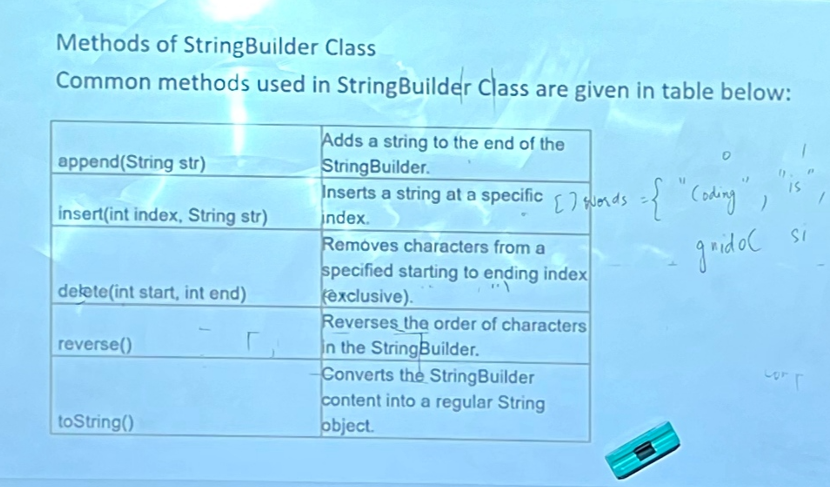
**Ans JDK provide this mutable classes**

****

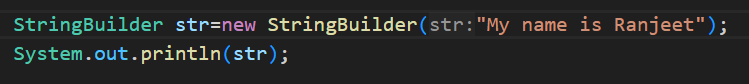
1. **empty string create by default have 16 bits**
2. **StringBuilder create the string Mutable.mutable means modifiable on same address.**
3. **It is same as StringBuffer but one thing is it is Asysnchronized .**

**It is efficient then stringBuffer.**

1. **Thread unsave . because it unSynchronised.**
2. **it is not java,lang pakage not need to import.**

****

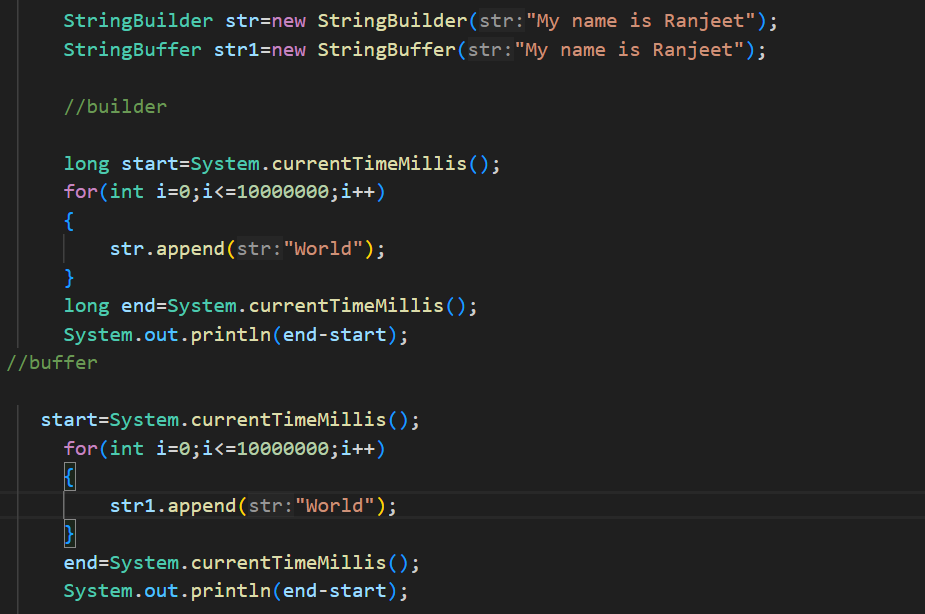
**Example**

****

**Output**

****

1. **Testing time difference between buffer and builder.**

****

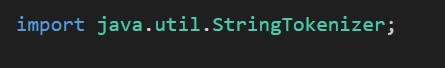
**Output**

** this time difference**

* **String Tokenizer()**

1. **Create token of string with the help of deliminator.**
2. **Tokenizer need to import because it is in java.lang.utill.**

**And utill need to import.**

****

**Example.**



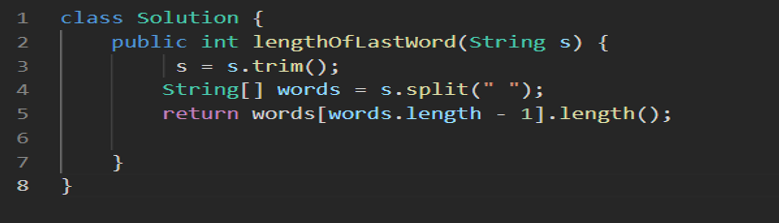
**Output**

****

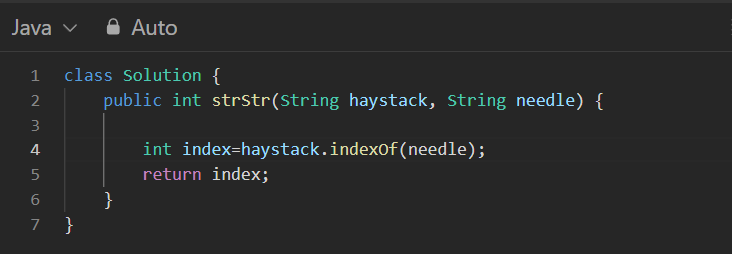
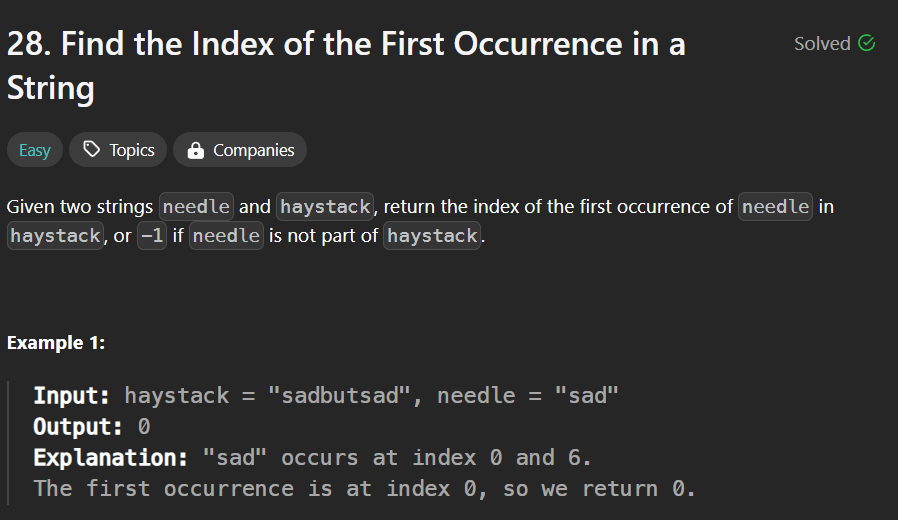
**Question of String, StringBuilder, StringBuffer**

**Ques. 1)**

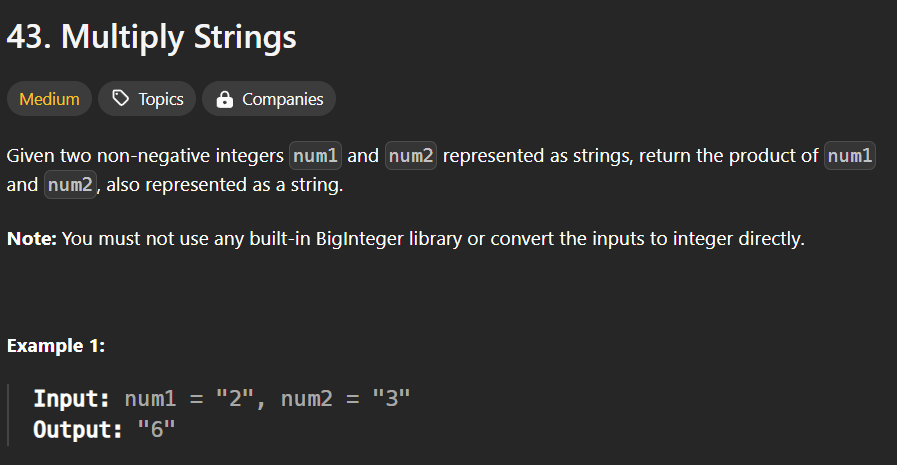
****

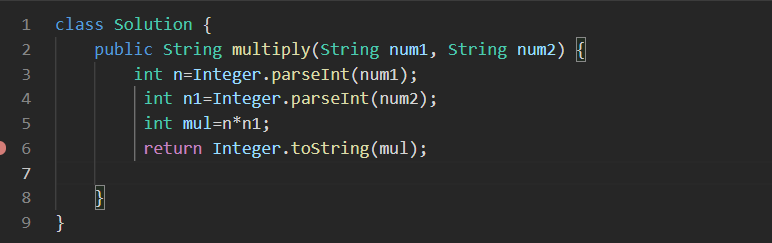
****

**Ques. 2)**

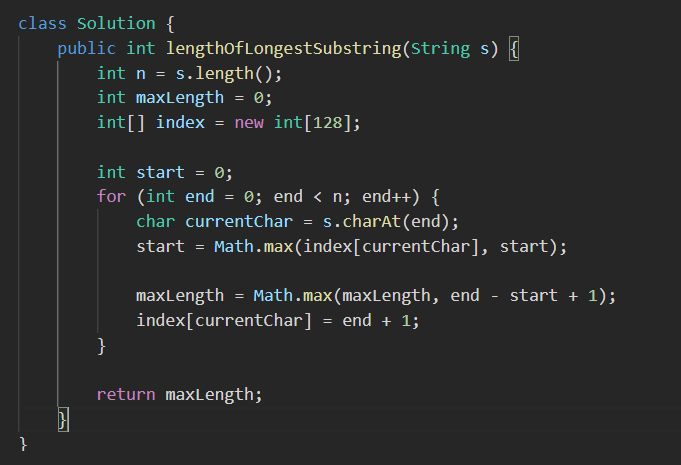
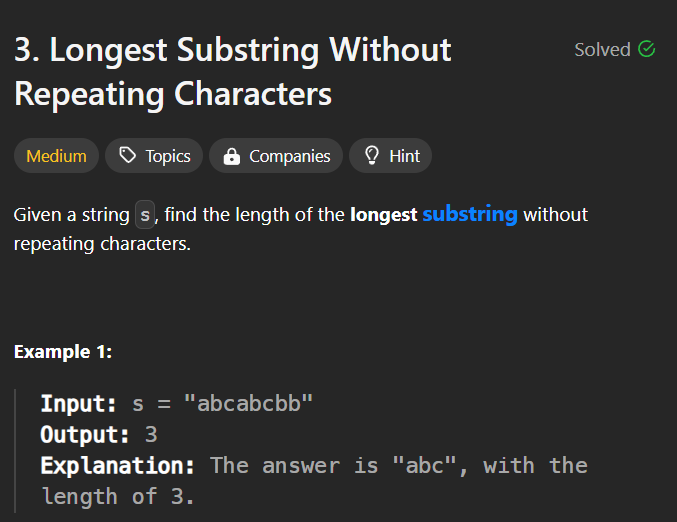
****

**Ques. 3)**

****

****

**Ques. 4)**

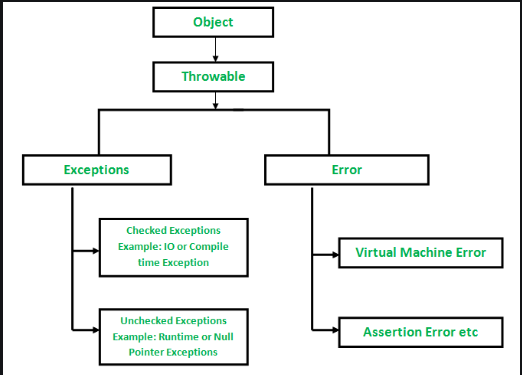
****

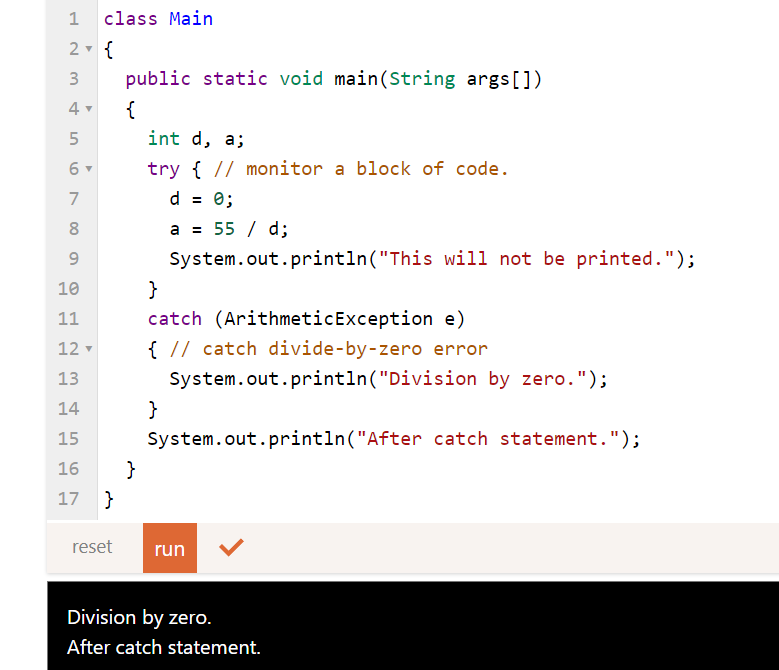
* **Exception Handling**

Java provides superior support for runtime error and exception handling, enabling programs to check for anomalous conditions and respond to them with minimal impact on  the normal flow of program execution. This allows error- and exception-handling code to be added easily to existing methods.

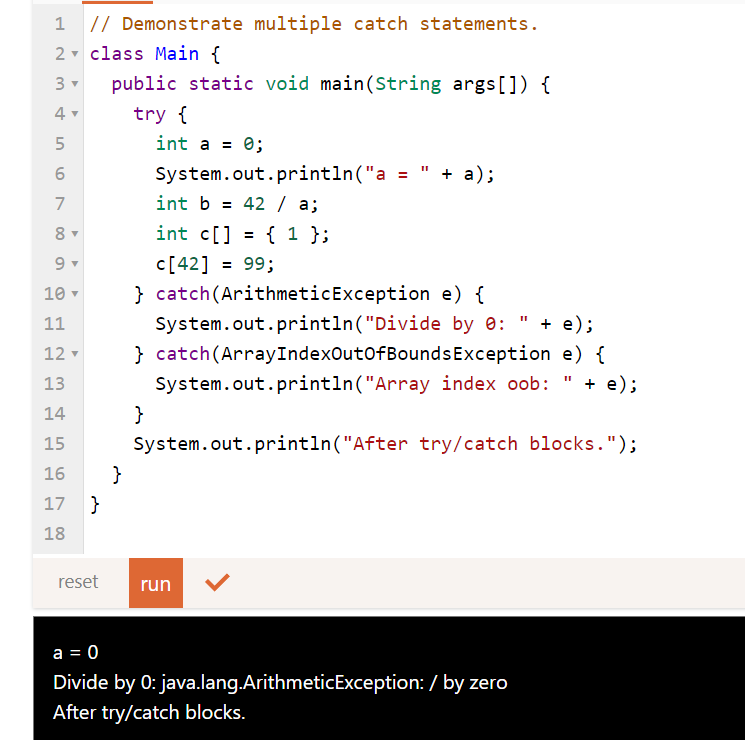
1. The **try** statement contains a **catch** clause that identifies what processing is to be performed for different types of exceptions. When an exception occurs, the Java runtime system matches the exception to the appropriate catch clause.

All exception type are subclass of built-in class thowable.





1. Multiple Catch Statements



It will call Arithmetic Exception, but if we set a to a non-zero value then, it will call ArrayIndexOutOfBoundsException as c[42] will raise an exception.

1. **Throw Statement**

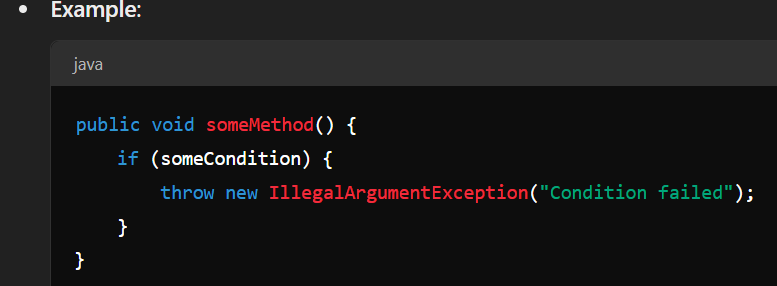
Exceptions are thrown using the throw statement. Its syntax is as follows:

**throw** Expression;

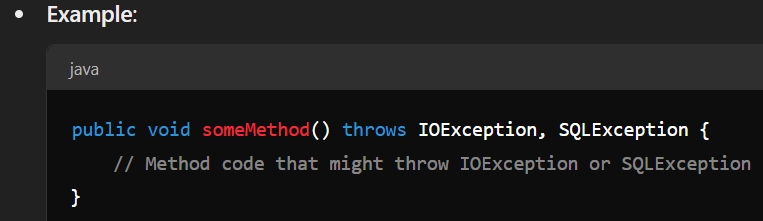
Note - What to Throw? A throw statement can throw an object of any class that is a subclass of java.lang.Throwable;

**throw** **new** ExampleException();

* **Throw**
* **Purpose**: Used to actually throw an exception from a method or a block of code.
* **Usage**: You use throw when you want to create and throw an instance of an exception. It is used within the method's body to indicate that an error has occurred.

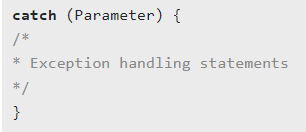


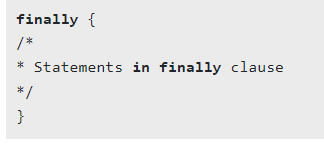
* Throws
* **Purpose**: Used in the method signature to declare that a method can throw one or more exceptions. This is a way to inform the caller of the method that they should handle or declare these exceptions.
* **Usage**: You use throws in the method declaration to specify that the method may throw certain types of exceptions. It is part of the method's signature.



* **try-catch-finally**

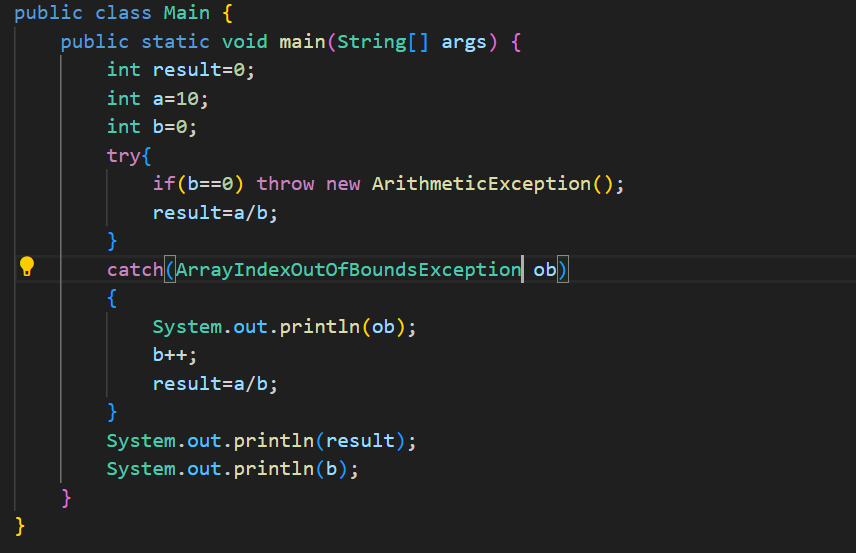
At least one catch clause or finally clause must be defined. More than one catch clause may be used, but no more than one ly clause may be identified





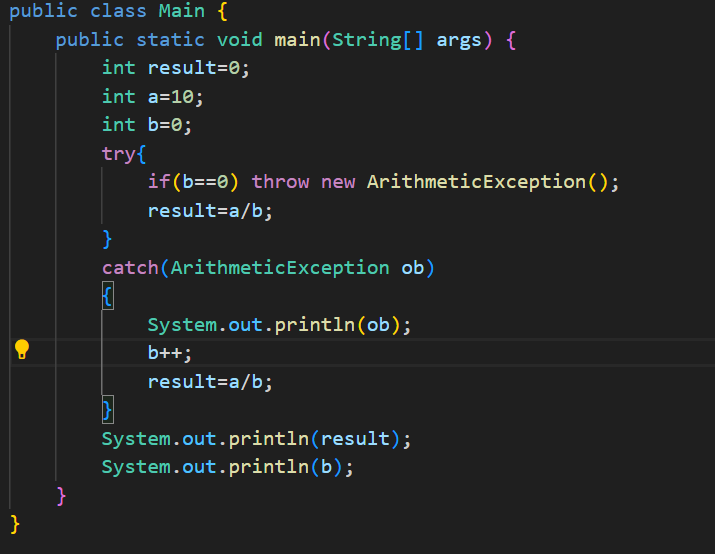
* The ly clause is always executed, no matter whether or not an exception is thrown.
* Step to be follows:

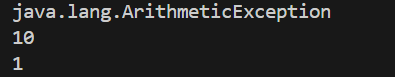
1. The try statement executes a statement block.
2. If an exception is thrown during the block's execution, it terminates execution of the statement block.
3. and checks the catch clauses to determine which, if any, of the catch clauses can catch the thrown exception.
4. If none of the catch clauses can catch the exception, the exception is propagated to the next level try statement.

****

****

**Mismatch exception**

****

****

**Solved like this**

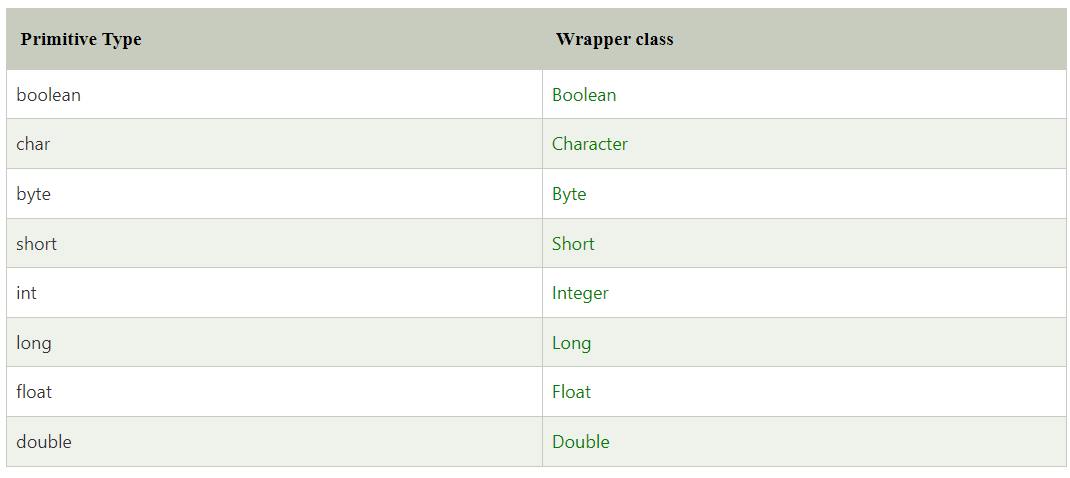
**Exception catch will be in the end.if we put in starting then ambiguity arises**

**// try with out catch is not allowed**

**Try with At least one catch or try with one ly is ok**

**ly block execute in every case.**

* **Wrapper class**
  1. The wrapper class in Java provides the mechanism *to convert primitive into object and object into primitive*.Generic Class.
  2. Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.
  3. **Change the value in Method**: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.

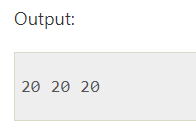


* **AutoBoxing**
  1. The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing.

for example, byte to Byte, char to Character, int to Integer, etc.

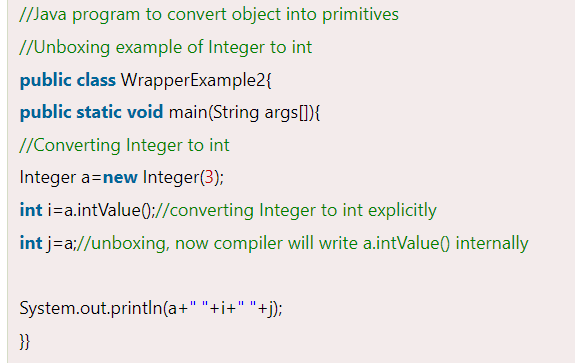
* 1. Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

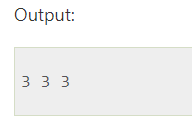




* **Unboxing**

1. The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.
2. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.





* **Generics**

1. The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.
2. Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

**There are mainly 3 advantages of generics. They are as follows:**

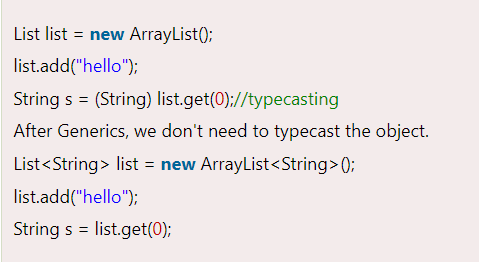
1. **Type-safety:-** : We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of object

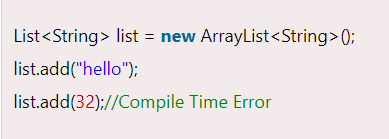


1. **Type casting is not required:** There is no need to typecast the object.

Before Generics, we need to type cast.

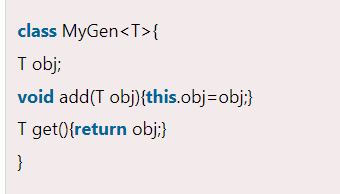


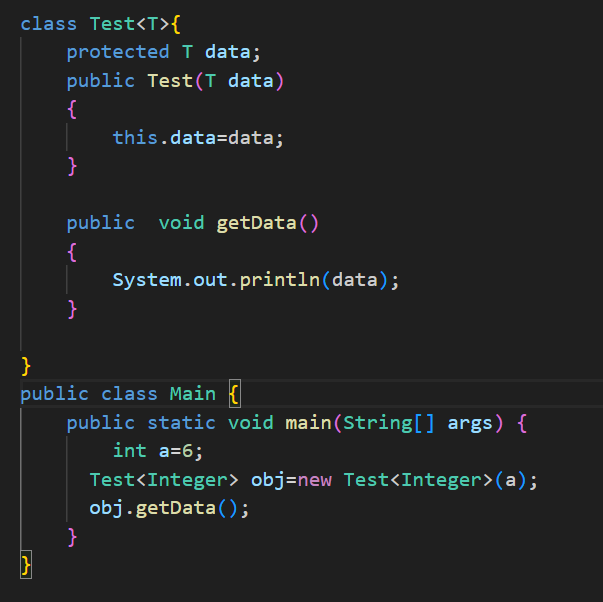
1. **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.



* Generic class

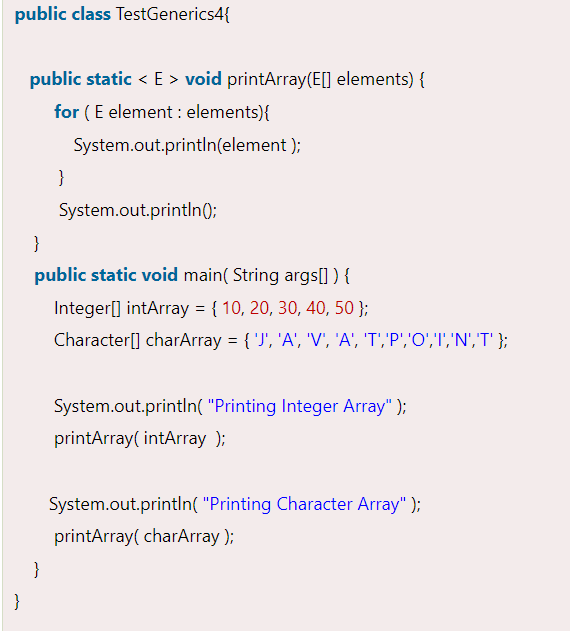
1. A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.
2. Let's see a simple example to create and use the generic class.



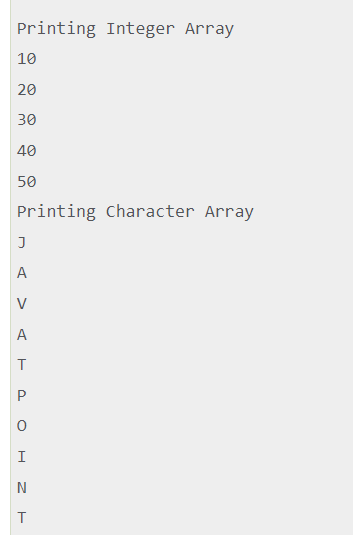


* **Generic Method**

1. Like the generic class, we can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.
2. Let's see a simple example of java generic method to print array elements. We are using here **E** to denote the element



Output



* **Type Parameters**

The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

* T - Type
* E - Element
* K - Key
* N - Number
* V - Value
* **Collection**

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

* List Interface

A list is an ordered collection of elements which may contain duplicate . it is an interface that extends the collection collection interface . lists are further classified into into the following.

* 1. ArrayList

Arraylist is the implementation of list interface where

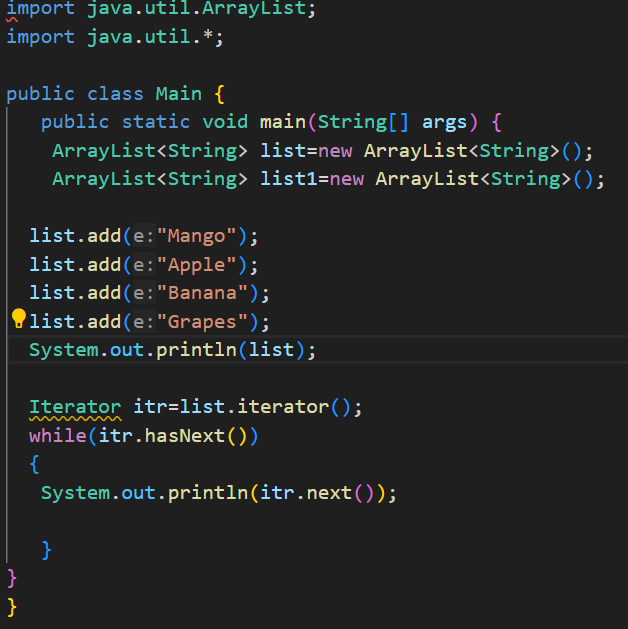
Internally dyanamic array to store elements.

10 size buffer value size initiate

Growable size=(old capacity\*2)+2

Syntax

ArrayList<Interger> marks = new Arraylist<integer>();



* 1. LinkList

Linked list is a sequence of links which contains items. Each link contains a connection to another link.

In which uses doubly linkedlist.

Syntax

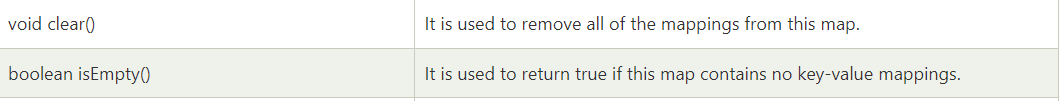
LinkedList<Wrapper class type> object =new LinkedList<Wrapper Class type>();

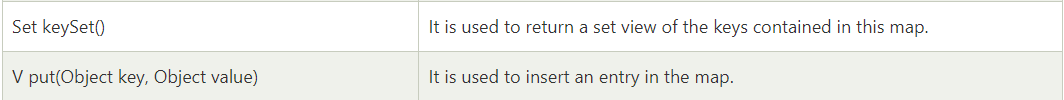
In which size is growable.

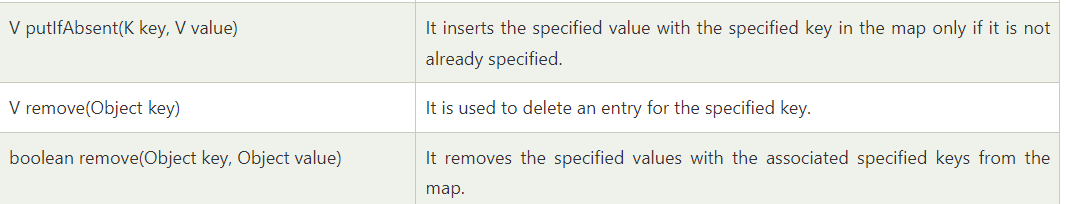
* **Hashmap**

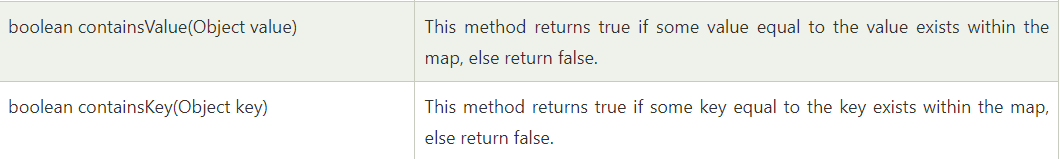
Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key.

Sysntax

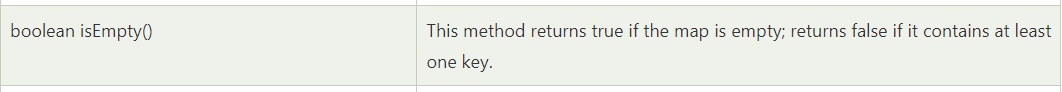














HashMap<keyType,valuetype> mymap=new HashMap<keyType,valueType>

Put and get

Mymap.put(key,value);

Mymap.get(key);

Int size=mymap.size()// returns the size of pairs added into map

Mymap.containesKey(key);// returns the true or false if key is in map

Mymap.keySet() //return the keys in map in collection set

Mymap.values()///return the value in map in collection set

Mymap.isEmpty() // return true if this map contains no key value mapping.

Iterate

For(keyTtype key :Mymap.keySet()

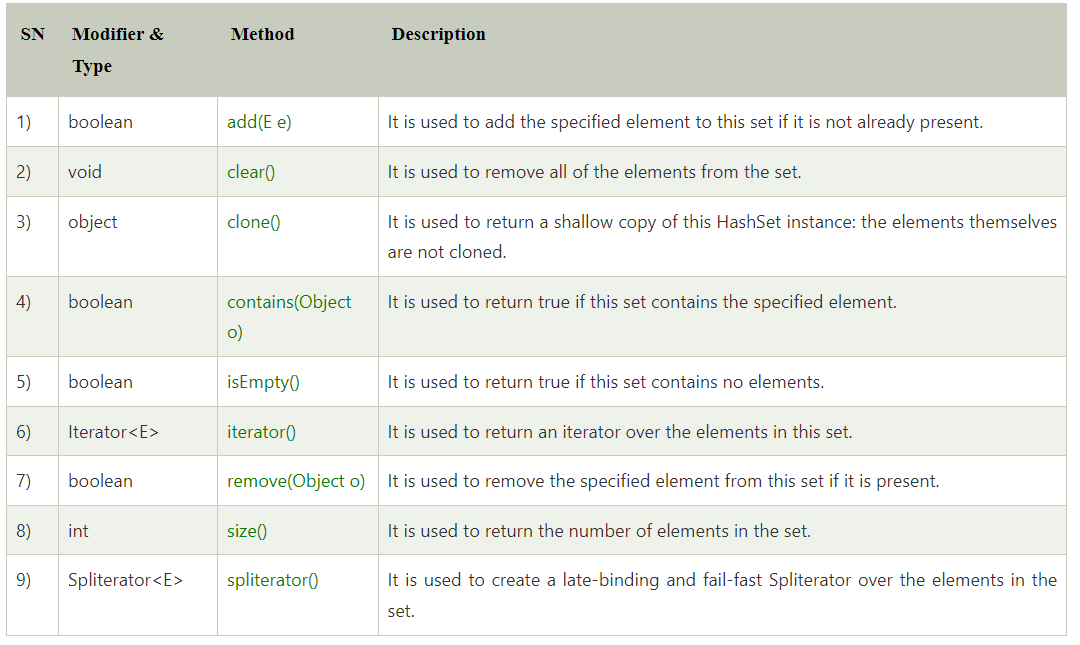
{

Mymap.get(key);

}

* **Hashset**

**A list can contain duplicate elements whereas Set contains unique elements only.**

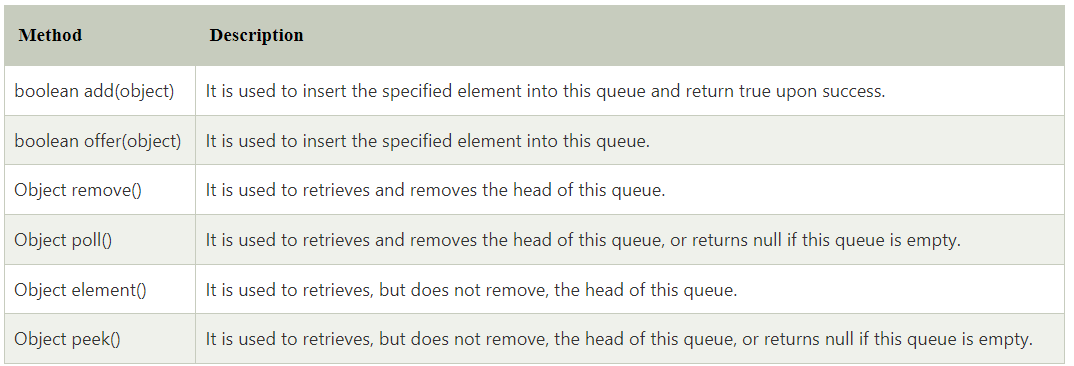
****

Syntax

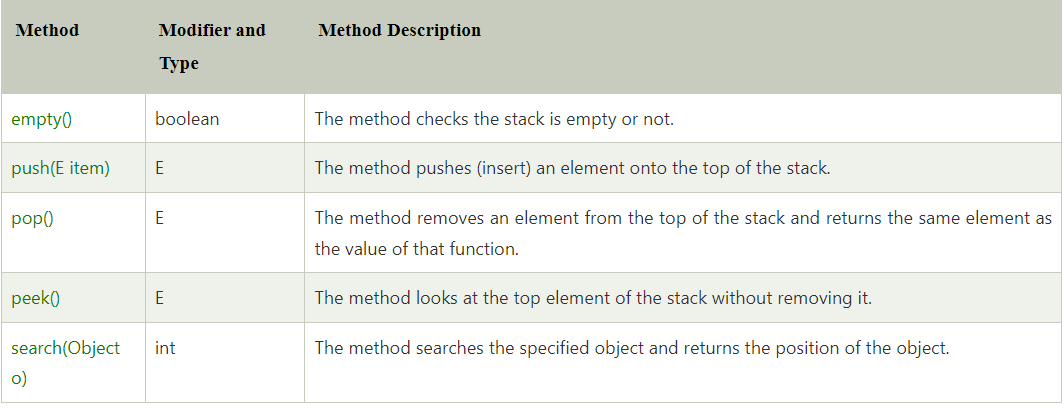
HashSet<E> hs=new HashSet<E>();

Can store 16 capacity and default load factor is 0.75

* **Queue**
  1. The interface Queue is available in the java.util package and does extend the Collection interface
  2. It is used to keep the elements that are processed in the First In First Out (FIFO) manner.
  3. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

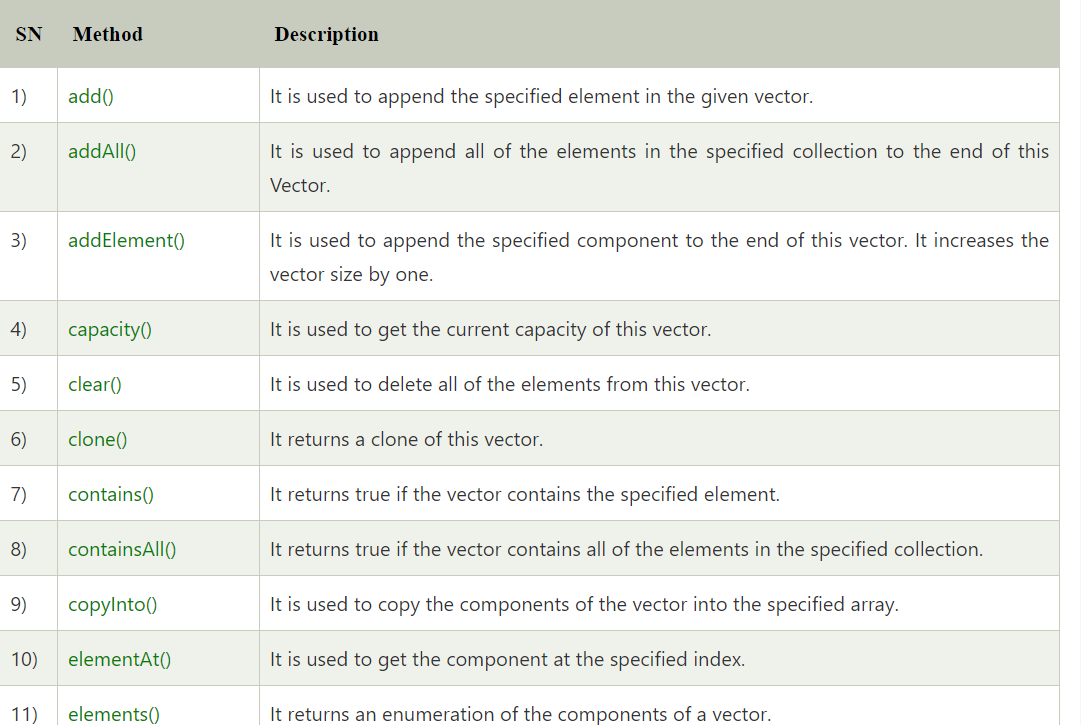
****

* **Java Stack**
  1. The stack is a linear data structure that is used to store the collection of objects
  2. It is based on Last-In-First-Out (LIFO).

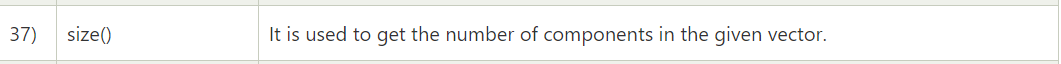


* Vector

**Vector** is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.







* **Multithreading**
* Java supports single-thread as well as multi-thread operations. A single-thread program has a single entry point (the main() method) and a single exit point. A multi-thread program has an initial entry point (the main() method), followed by many entry and exit points, which are run concurrently with the main(). The term "concurrency" refers to doing multiple tasks at the same time.
  1. Simultaneuously execution of multiple thread.
  2. Threads are independent. If there is any exception in thread , is does not affect other threads.
  3. Thread is a single unit of work.

Thread is a light weight sub process and smallest unit of processing

At least one thread in every program

Creation of thread

1. Thread class

policy

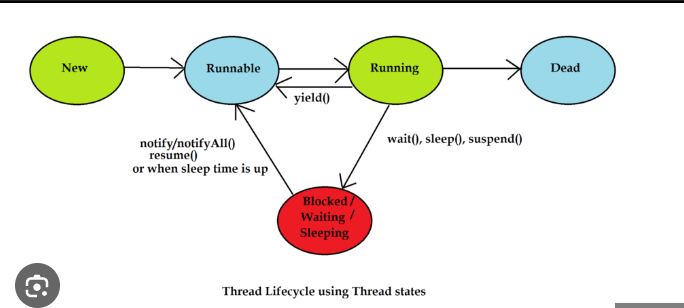
MIN\_PRIORITY=1

NORM\_PRIORITY=5

MAX\_PRIORITY=10

1. Runnable interface

* Life cycle of thread



* Constructor
  1. Thread()
  2. Thread(String name)
  3. Thread(Runnable r)
  4. Thread(Runnable r, String name)
* **Files**
  + 1. FileWriter

This is used to write character-oriented data to a file.it is character oriented class. Which is used for file handing in java

**KeyPoint for exam**

**Inheritance**

1. **If set a default value. The method is that means that cannot override. So we can do that this with super keyword in contructor**

**Eg. Is in marketer question.**