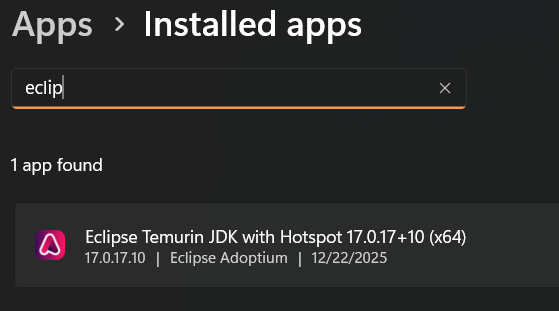
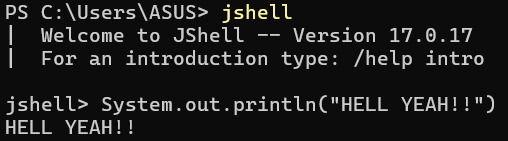
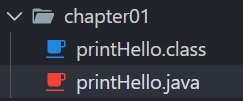
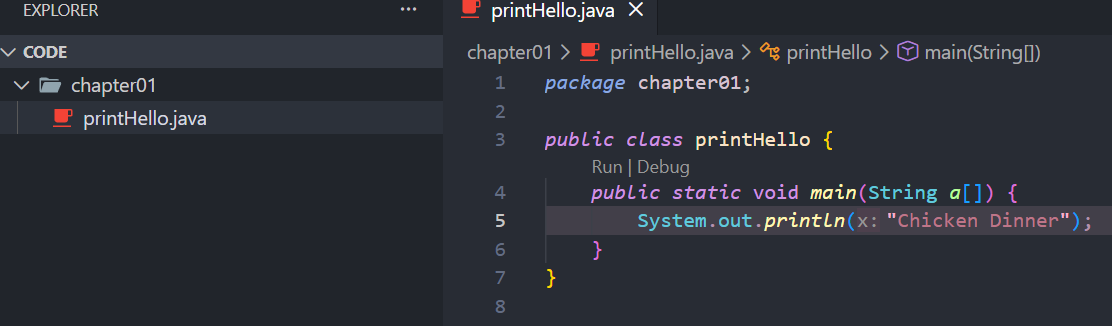
JAVA

Install jdk from oracle or I installed adoptium (as it is good)  


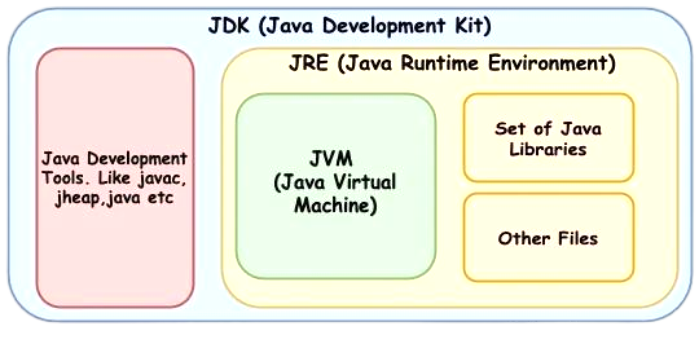
Java shell a place in terminal where you can execute java codes  


If there are multiple coding file, we have to execute them all => then we need to provide first file to jvm, first file will always be the one which contains main method

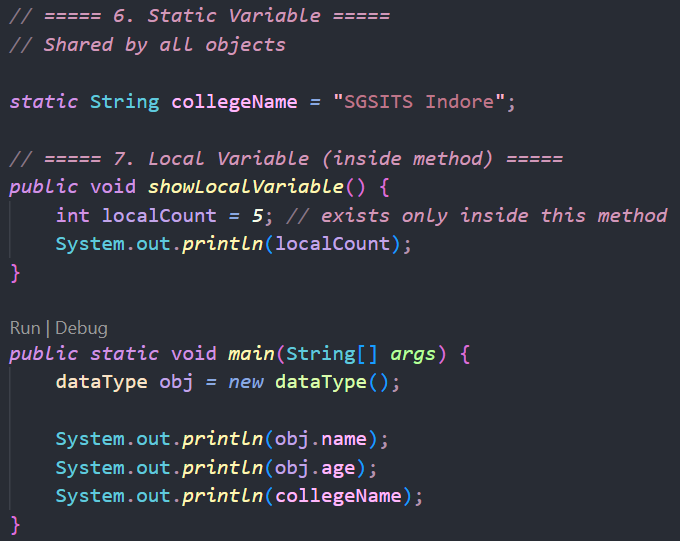
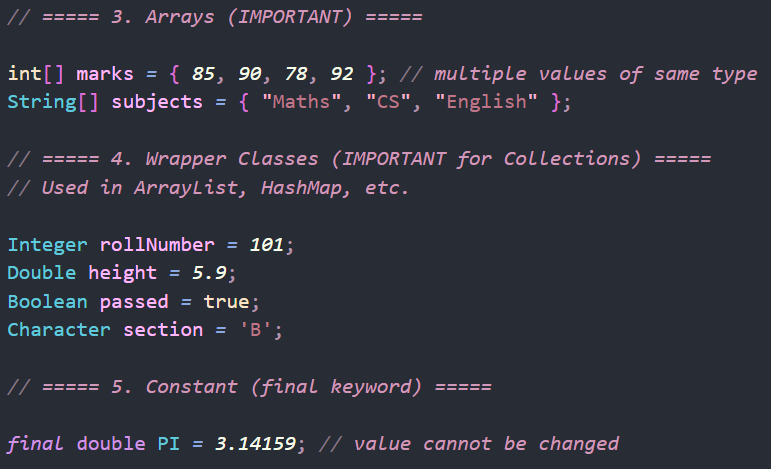
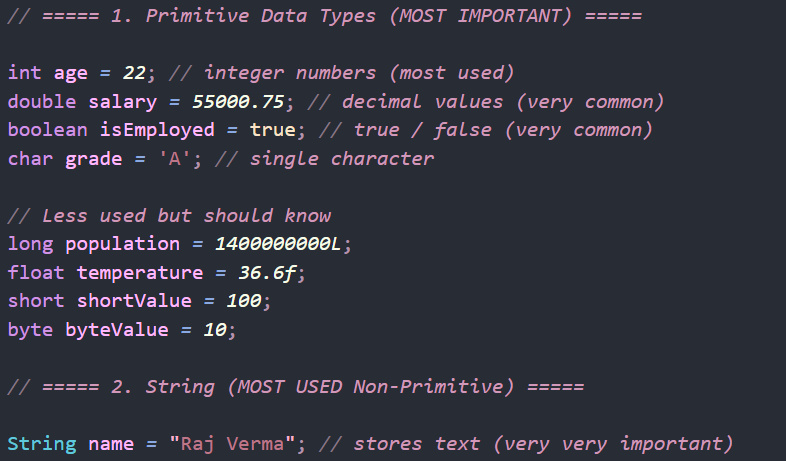
Write our first code  
  
Now we run code,  
  
Now a byte-code file is created   
   
🡨

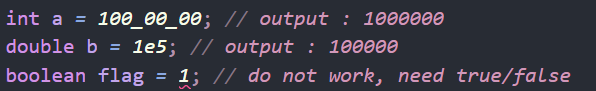
extension for java file is .java & for byte code is .class

Now we play the code,  
  
Note : as I am in a folder, which is package in java term => if I want to play a class file then I have to provide a full path i.e. package.javafile  
But for javac file direct addressing the file address is followed

Java Ecosystem  


Java Variables



Some things to remember  


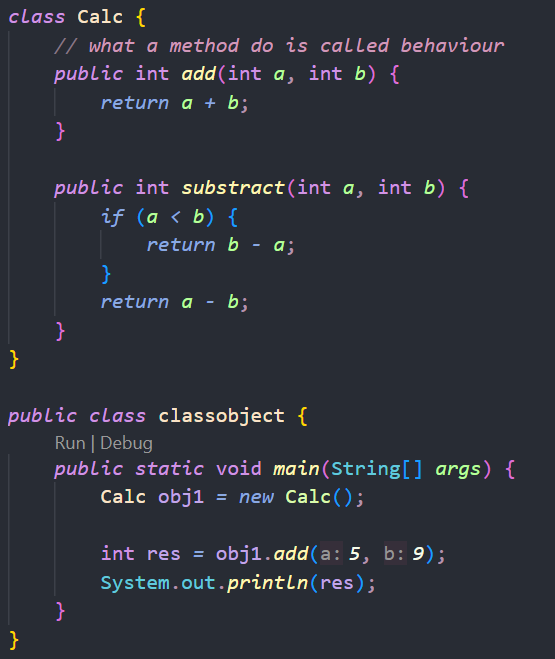
Type Conversion & Type Casting

Widening = java auto converts small to large type

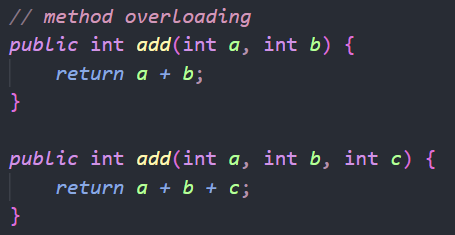
Narrowing = manually converting data type

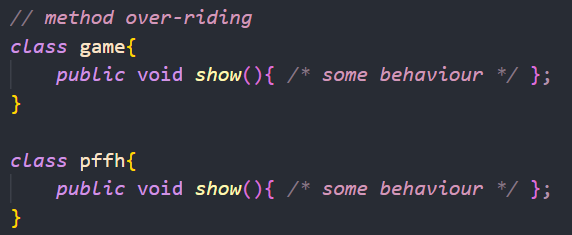
Type promotion => conversion of data from lower to higher types

Class & Object

  
**obj1** is a **reference** to the object [generally in stack shape] which is present in heap memory

Method

Overloading :   
 [ main method can also be overloaded ]

Overriding :  


Stack & Heap Memory Allocation  
-> variable lives in stack, and actual data lives in heap

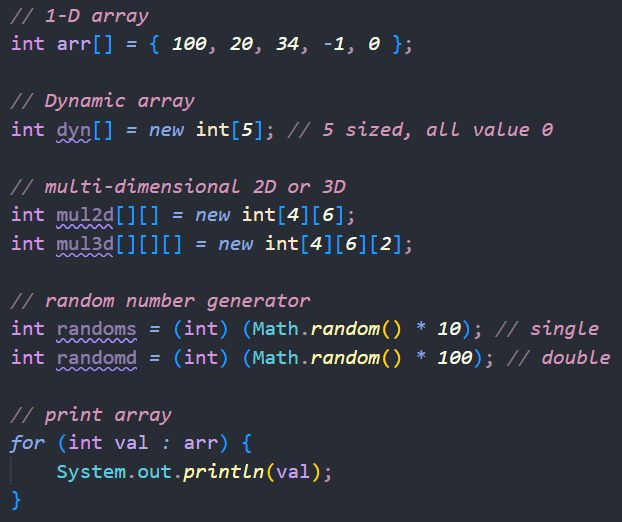
Stack stores -> local variable, object reference, method call

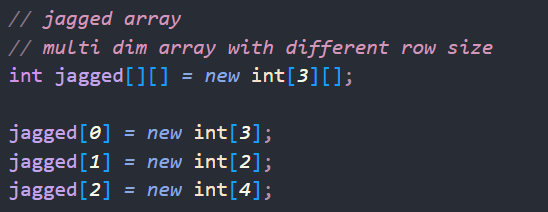
Heap stores -> object data, instance variable, string object

Note:  
 1. Stack is **fast**, heap is **slow  
 2.** Stack memory is **auto freed  
 3.** Heap memory is freed by **Garbage Collector  
 4.** Multiple references can point to **same heap object**

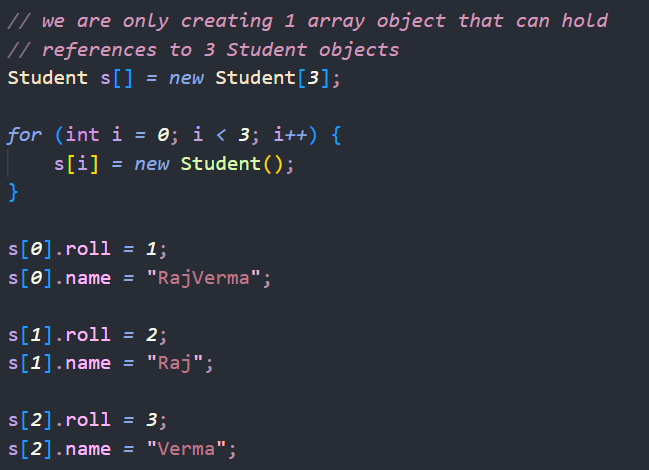
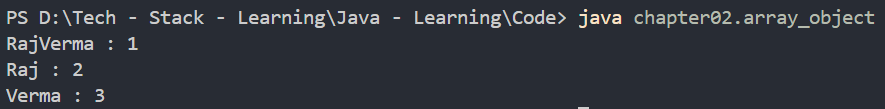
 [ s1, s2 are instances of an class student ]

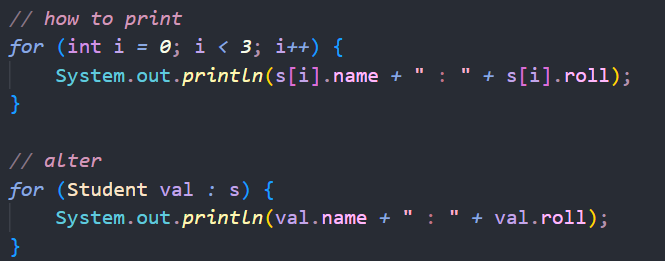
Array

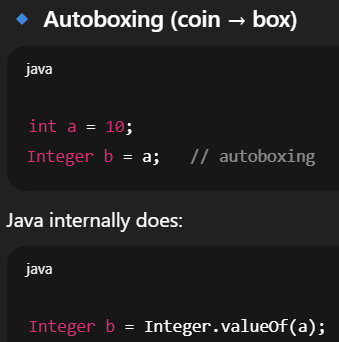
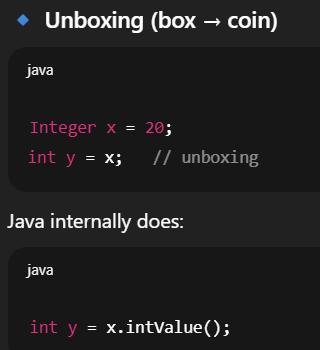




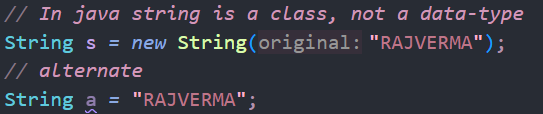
Array with Object

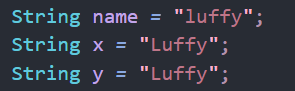


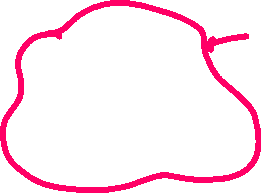
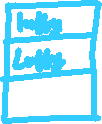
 ( printing )

Autoboxing & Unboxing  
auto -> means auto converting a wrapper class to its primitive type  
   
Integer is an object where x is a reference to a object which contain value 20  
int is a primitive data holder, directly stores value i.e. a = 10

\*Wrapper class are created to deal with collection, framework

String  
-> a class to store sentences  
  
-> string are immutable means they cant be changed, they get allocated and then if they are not needed they got erase by garbage collector

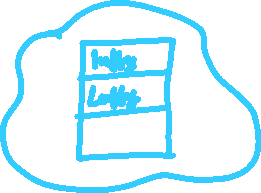
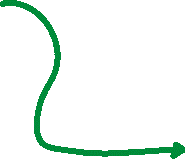
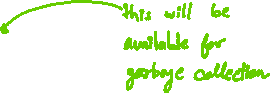




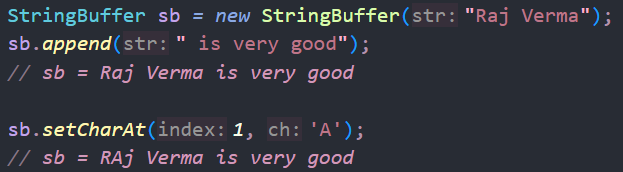
|  |  |
| --- | --- |
| y | 103 |
| x | 103 |
| name | 101 |

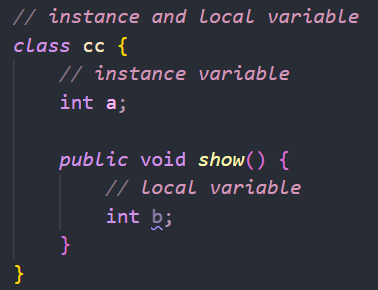


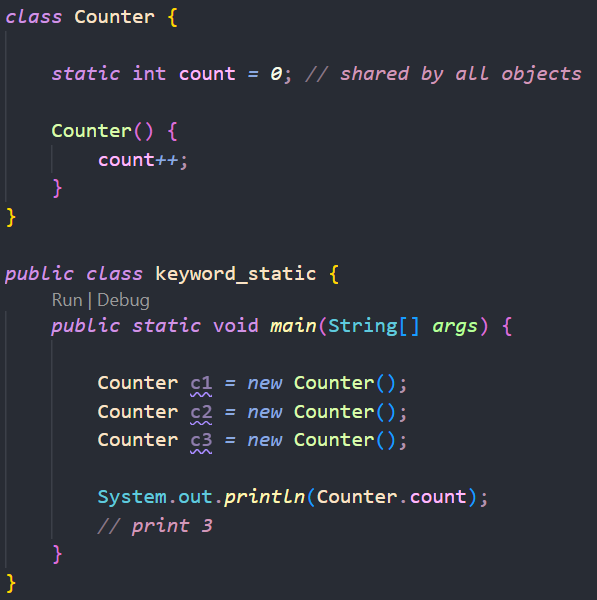
Now let say we change name  

To implement a ***mutable string*** we use string buffer or string builder

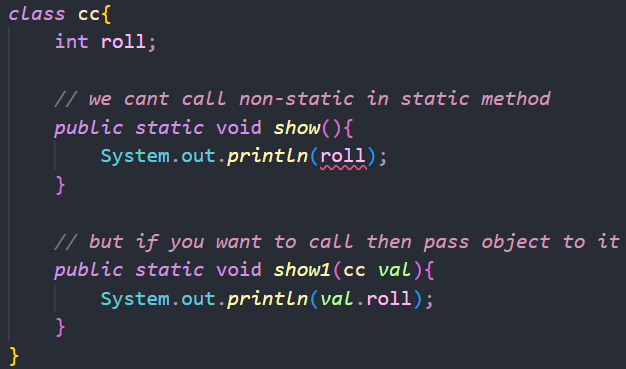
  
Here reference will be same, and string is manipulated at the same address

Instance & Local Variable  


Keyword – static  
-> static variable must be modified by calling it with class name  
-> static variable is same for all the objects of class, so if you change for one it will be updated with same to others  


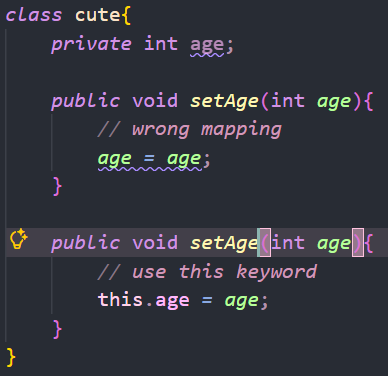
\* so using *static* we make variable/method of class not for object

\* main method contain static in it because at start we don’t create any object, and to run a method we need to initiate an object, but when we use static problem resolves and main method easily executes

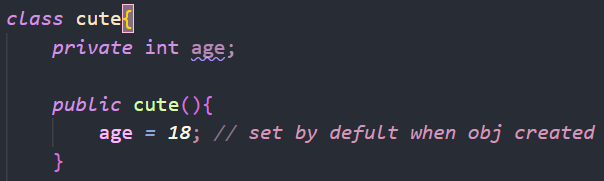
**static** **method**  


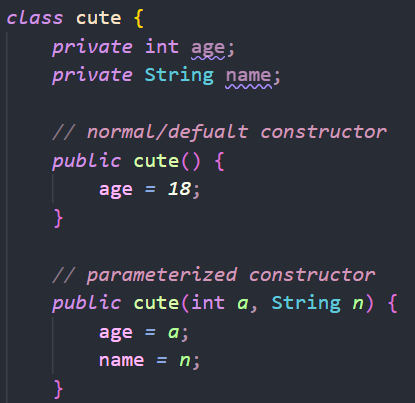
Getter & Setter method

-> in simple terms we create private instance variable and then create public method to manipulate those variable or fetch the manipulated variables

this keyword  


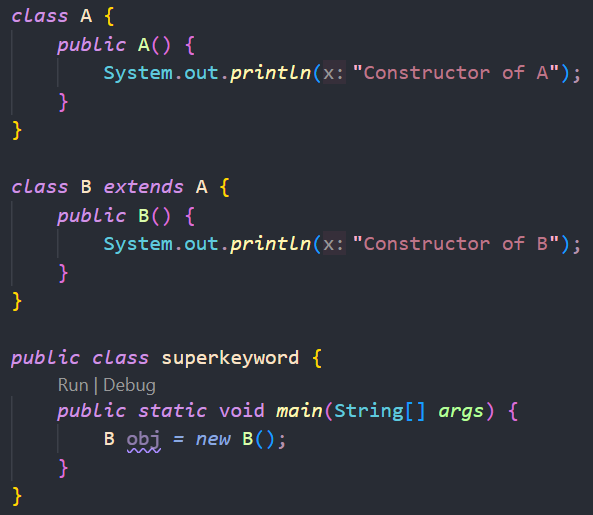
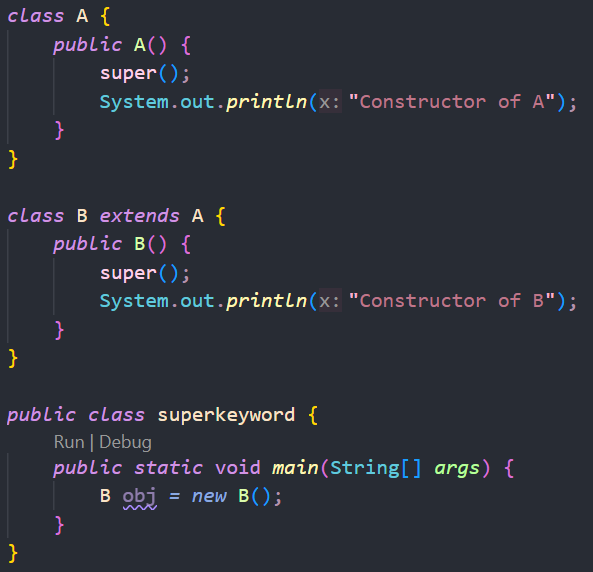
-> this denotes the current object, like to which we called the method with

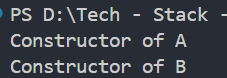
Constructor  
-> always called when an object is created  
-> it is a method which have same name as class  


**Overloading** a constructor  


Tip : Right click on vs\_code code area and go to **Source Action > generate constructor**

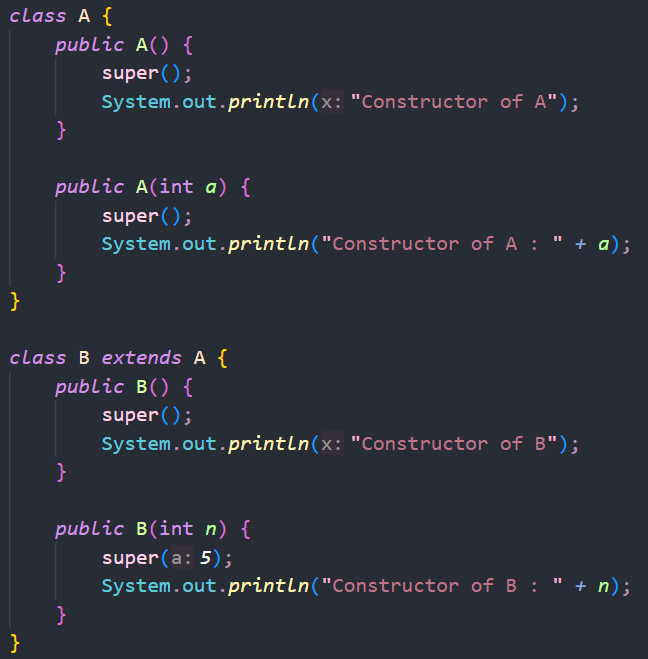
super keyword

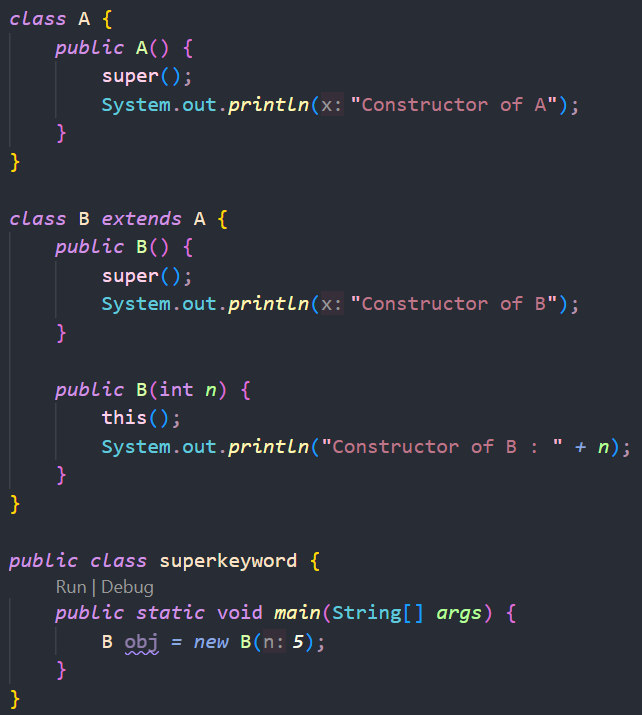
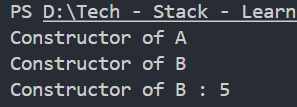
-> every class have a super method which invoke first, so even if you don’t mention it *super* is present eventhough not visible , (refer diagram to for the position of *super*   
output for above will be : 

-> when you need to use super ?  
so, when we have to call a parameterized constructor of superclass

Lets take an example below,  
if you pass **n** in super then second function of A is called,  
if you don’t pass **n** then default constructor of A is called

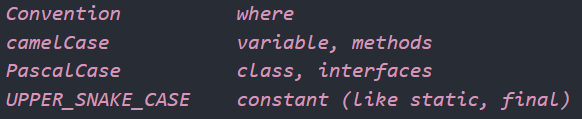


Another Point : in class A whom super is calling to ?  
-> so every class in java extends to a class called as ***Object***   
-> so is B also extends to A, Object => **NO**, it only extend to A then A extend to Object, its is also know as **Multi-level Inheritance**

Another Point : if you want to call both constructor of B, how to achieve it ?  
----> output : 



Naming Convention

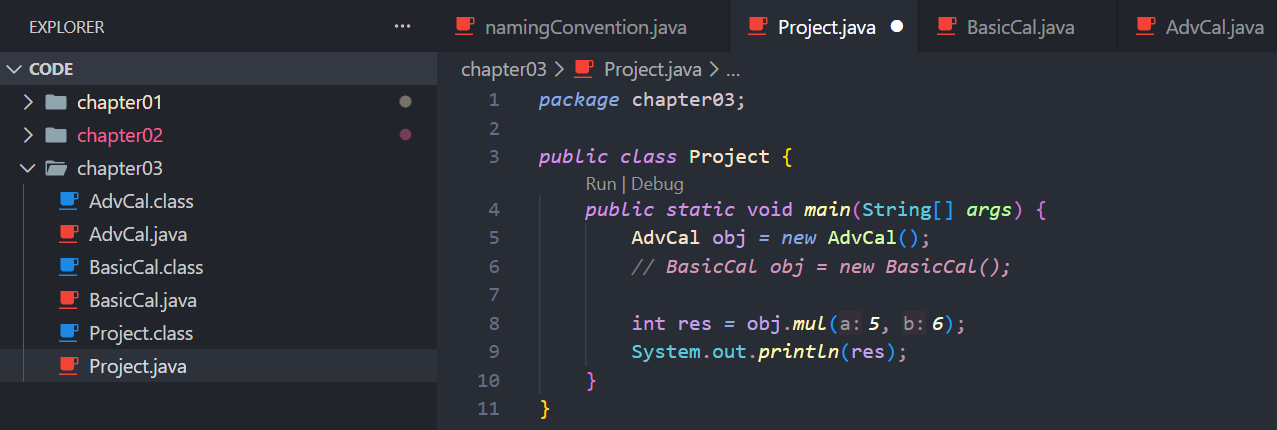
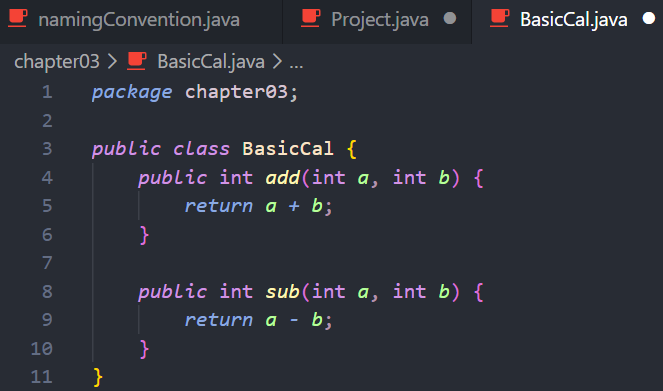
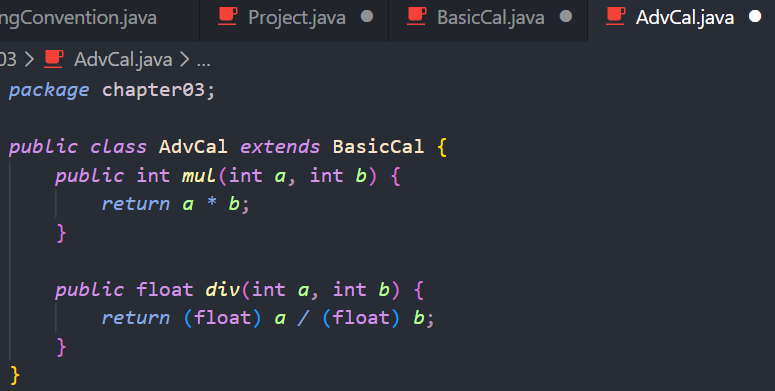


How to Compile and link multiple file if they are in a same package  
-> Compile all files of a package i.e. javac package/\*.java

-> if you don’t do this, then error like *cannot file symbol* occurs

Inheritance

**Multi-Level Inheritance**



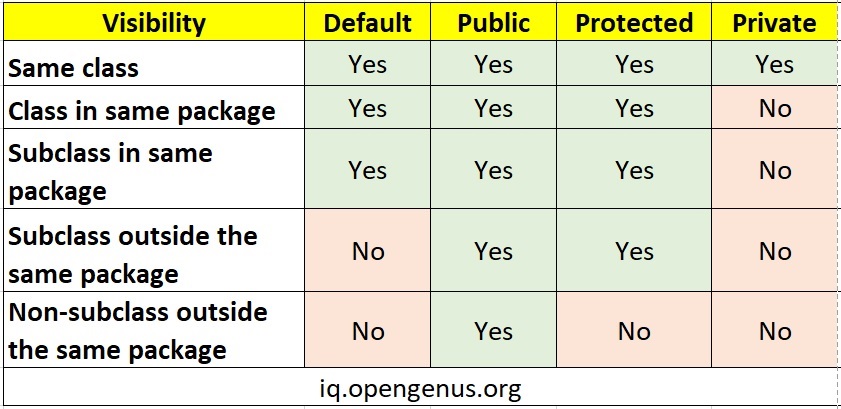
In above example we are using BasicCal functions in AdvCal object,   
basically we have one parent one child relation

**Multiple Inheritance**  
-> Java Do not Support it in classes, but supports it in Interface



Ambiguity:  
let say class A have method y() and class B also have same method y()  
then how C choose which method to invoke !?

Importing:  
->A package can be imported in any java file you just have to use import file destination  
-> When we import a complete package then it import all file in it but not subfolders   
For subfolders we need to import the subfolder too

Access Modifiers

Poly-Morphism

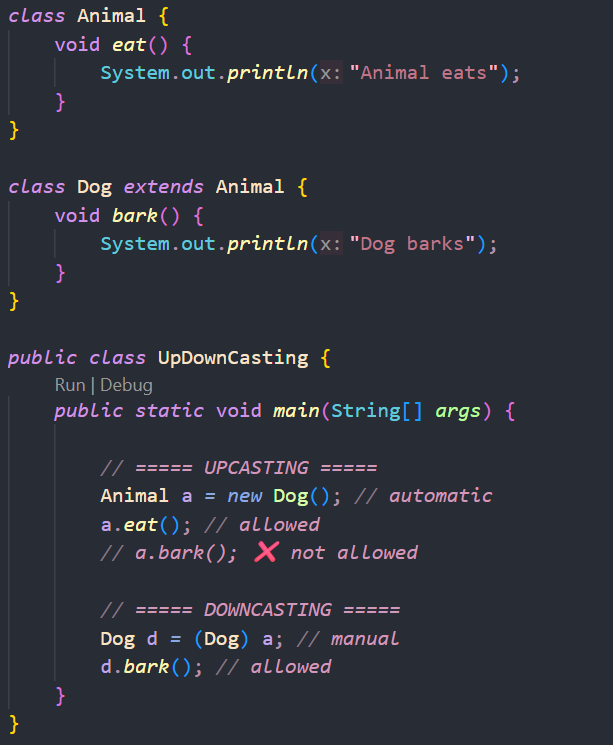
If let say B inherit A then we can create a A type of B object   
i.e. A obj = new B();

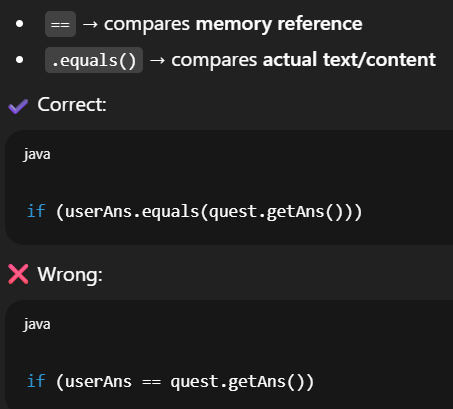
If B do not inherit A, then above will not work

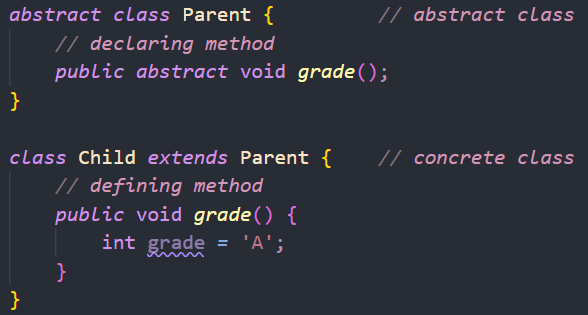
Now, if both A & B have show() method and obj.show() is called, then which show work ?  
-> answer is that show() method of the new object will work i.e. of B in our case

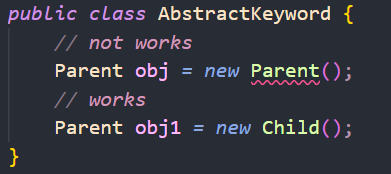
Final keyword  
-> making something a constant, same as const in C++

If use with method -> stop overriding  
If use with class -> stop inheritance  
If use with data type -> stop anyone to change value of it

Up-Down Casting  
  
It simply is referencing an object which is of different type accessing method of the object type  
Like we cant access Dog bark() method when object is of type Animal

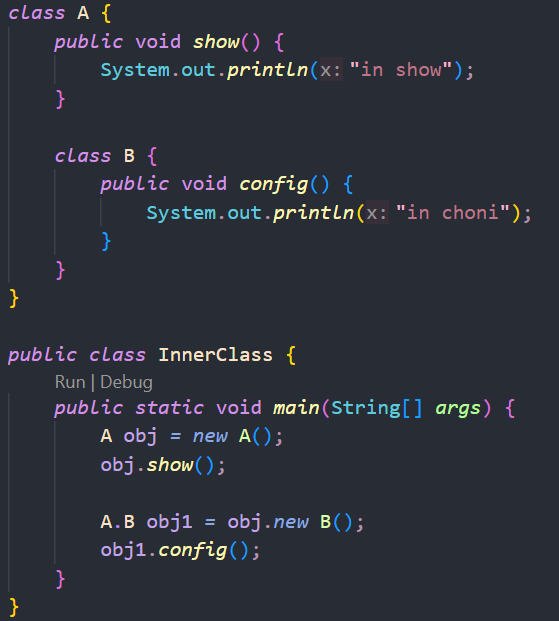
String Comparison  


Abstract Keyword  
-> We declare a method which we don’t use in parent class but its child might have an implementation for it  


-> Note : we **cant create object of a abstract class** but we can use type of abstract class  


-> to create a abstract method, a class also marked as abstract

-> You have to define all abstract method in child, else you get error   
or if you don’t want to define it in child, mark child as abstract then

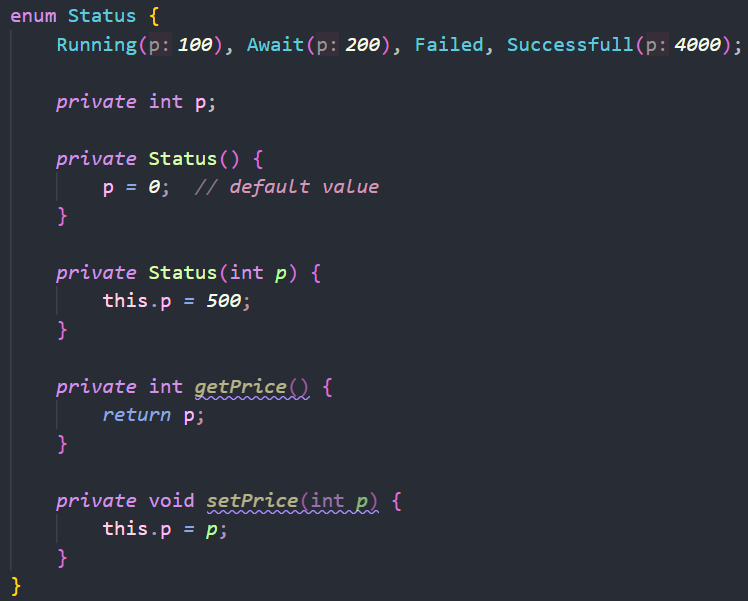
Inner Class  
-> class ke andr class  


-> we can mark class B with static keyword  
then we can create object like : A.B obj1 = new A.B();

Anonymous Inner Class  


-> Know this concept will come in handy

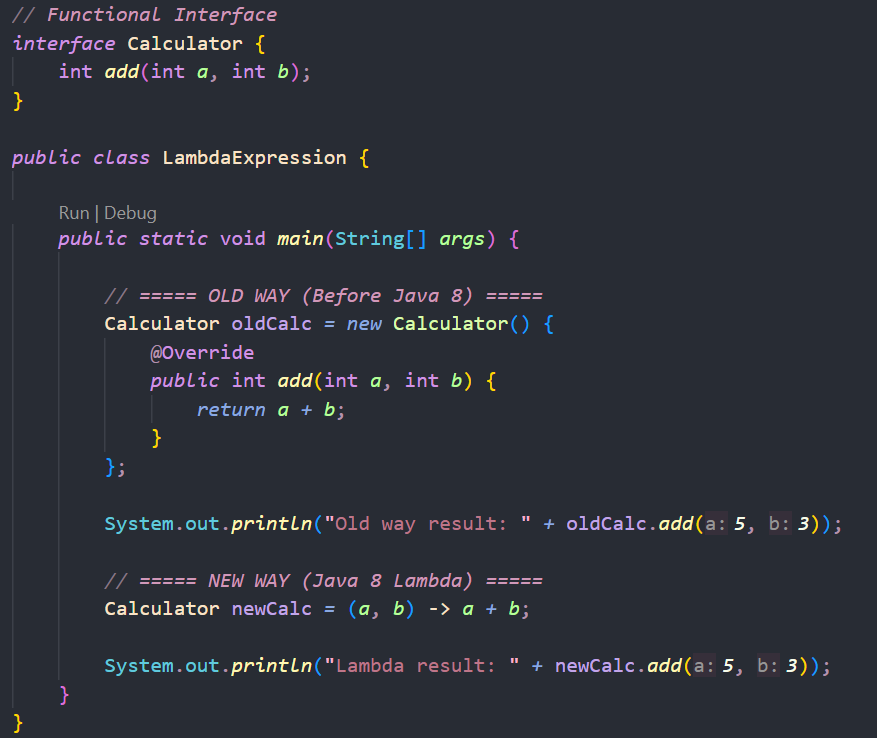
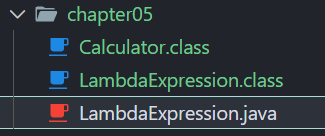
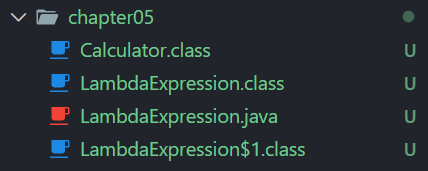
Enum  
they are basically use to define constant values, we can use this value anywhere  


-> with enum we can also define values for constant values,   
you can also use setter and getter to set their values  


Annotation  
**Annotations** in Java are special markers that give **extra information** to the compiler, JVM, or frameworks.  
They **do not change logic**, but they **change behavior / checks**

e.g. @override => let say you want to override a method showMe() of class A in showMe() of class B  
but you wrote wrong method name “showme()” now compiler treat it as different method but you know it is meant to override the parent method, so override check behavior and tell that you write this spelling wrong

Serialization & De-serialization  
-> serialization means storing object value in hard drive and delete object  
-> de-serialization means loading/fetching those value from hard drive and update the object with those values  
-> e.g. when you save gave data is saved and on quitting object created is deleted, now when you re-login to game you will load the saved data

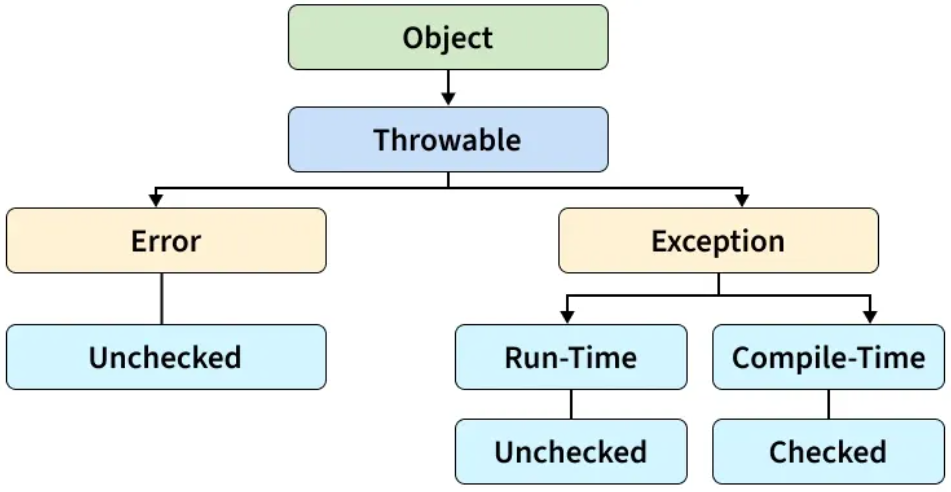
Lambda Expression  
-> a shorten way to define a function/method  
  
-> when we use lambda then a new anonymous class for the method is not created separately  
   
 on using lambda not using lambda  
**Syntactical Sugar**A **shorter, cleaner syntax** added to a language that **does NOT add new power**,  
it just makes code **easier to write and read**.

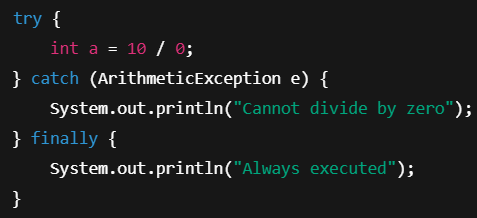


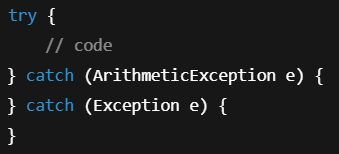
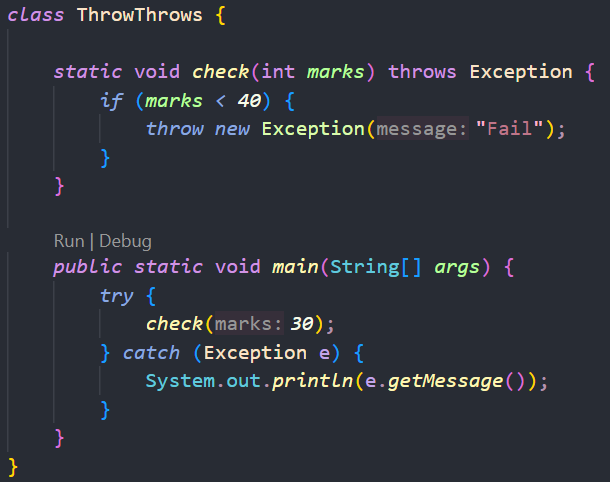
Java 8 **lambda expressions** are syntactical sugar over **anonymous classes**.

Exception  
-> **unexpected event** that occurs at runtime and **disrupts the normal flow** of a Java program

e.g. divide by zero, out of bound indexing, etc…


Keywords used : try, catch, finally, throw, throws  
 ( using all )

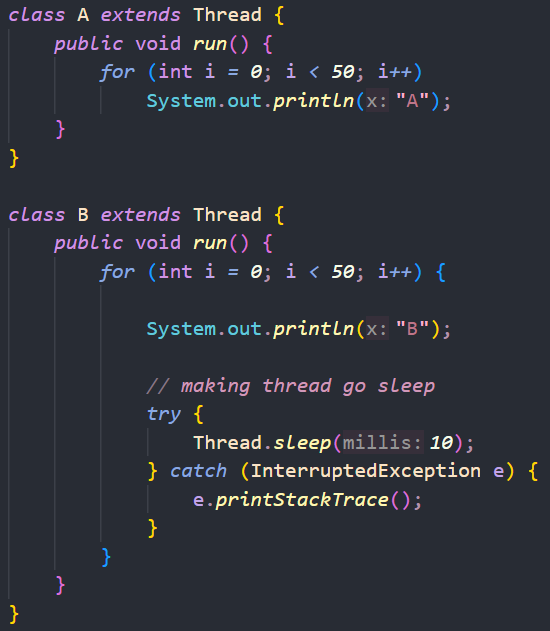
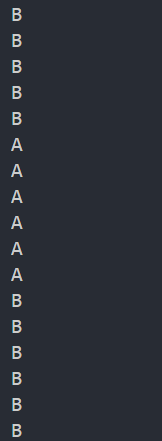
 ( multiple catch example ) ( throw/throws example )

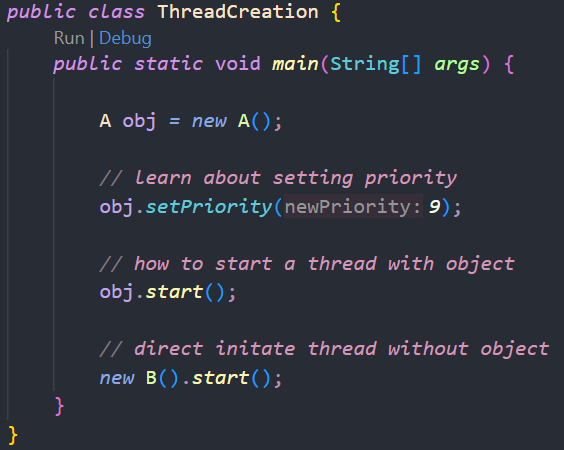
Throws is like a warning sign ( e.g. stone may fall on you ) [ Method signature, Declare Exception ]  
Throw is like actual thing happened ( e.g. stone fallen ) [ Inside method, throw excep. ]

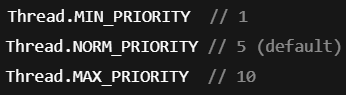
Threads  
**Thread** is a **smallest unit of execution** inside a program.  
Java supports **multithreading**, meaning:  
 Multiple tasks can run **concurrently** within one program.

Process = program  
Thread = task inside program

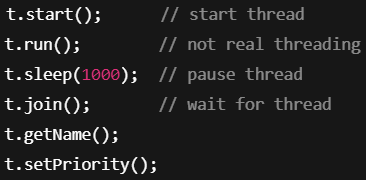
Single thread => function/method/process executed in order one by one  
Multi thread => all start execution at same time

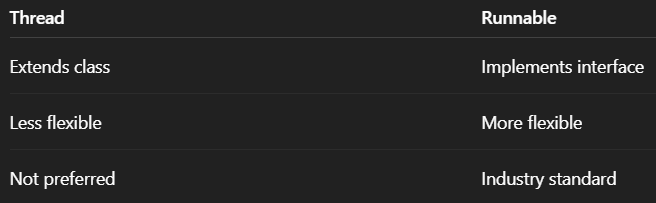
 Output : 

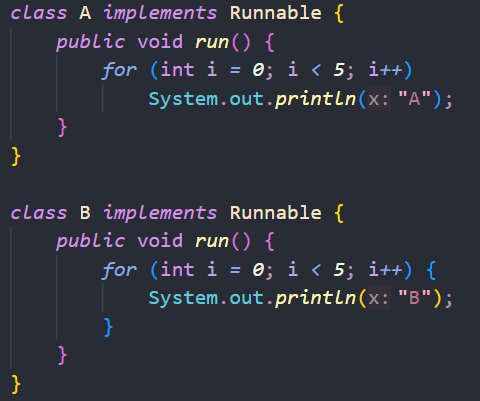
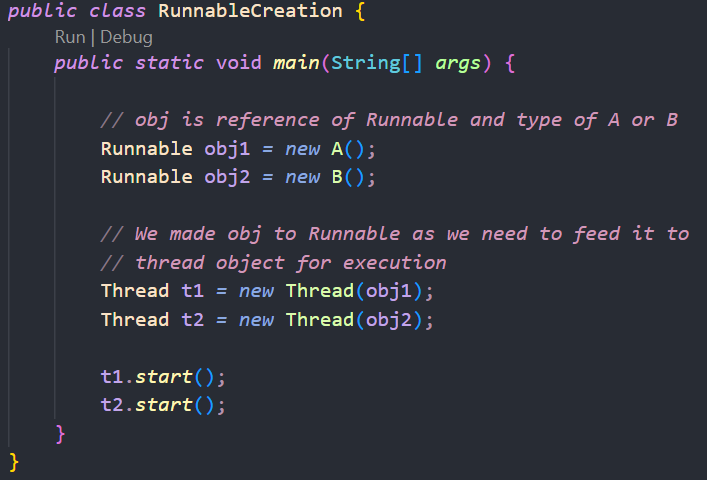


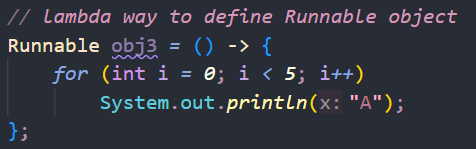
**Priority** in threads  


-> Remember setting a priority with java for a thread doesn’t guarantee, OS will be the one which decide the priority  
We can just suggest from java program

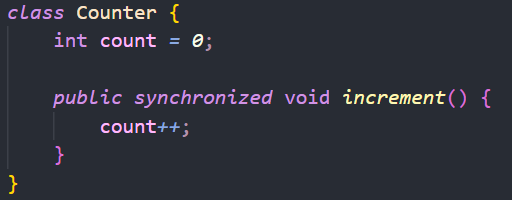
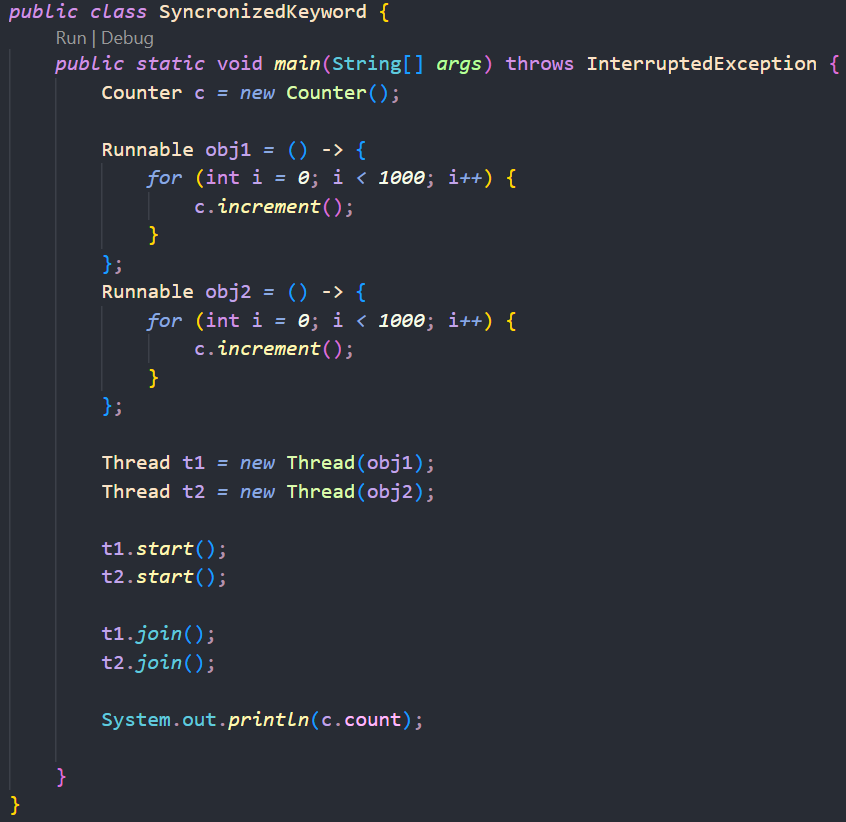
Common **Methods** used  


Runnable   


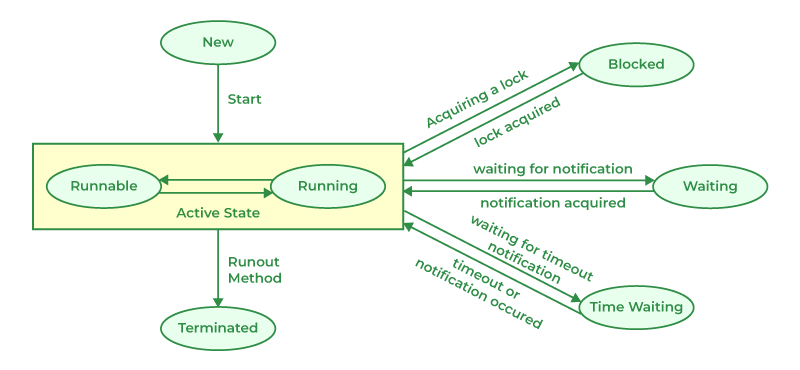
  
  
-> Use Runnable instead of direct thread because, java support single inheritance



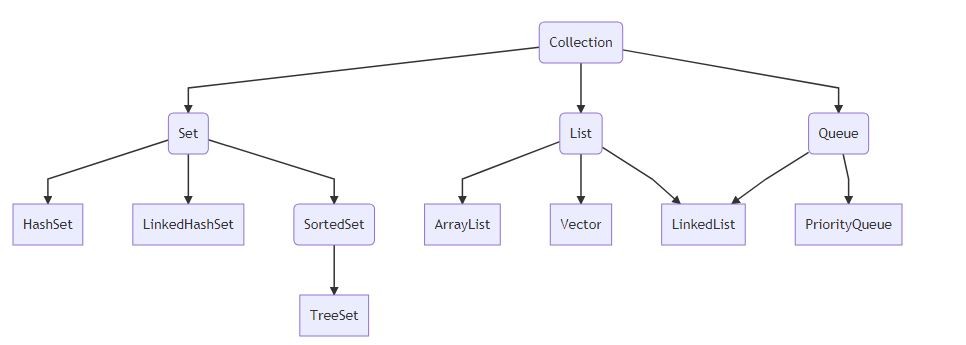
Race Condition  
-> when two threads work on a shared variable they sometimes used old values which result in not updating the variable carefully  
to resolve it we 1) join the threads 2) use *synchronized* keyword



States of Thread  


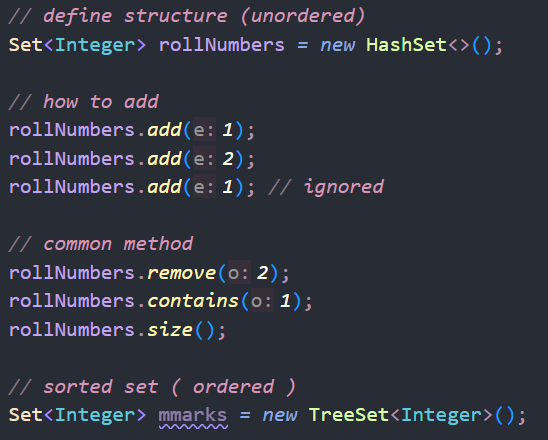
Collection API in Java  
1. Collection – we talking about interfaces and some class which implements it  
2. Collections – we talking about class and their method

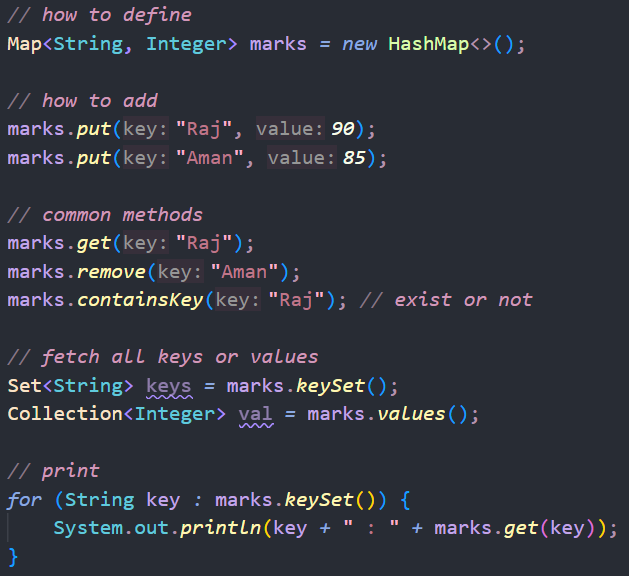


1. Array List ( similar to vector )

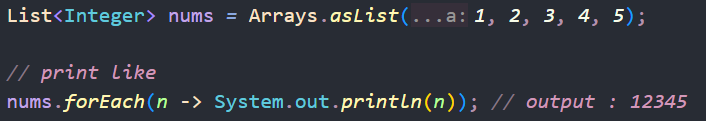
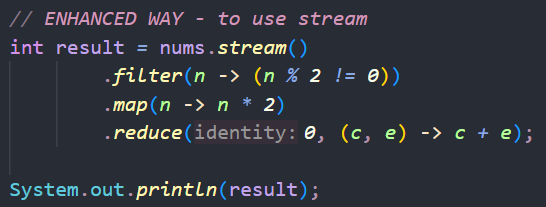


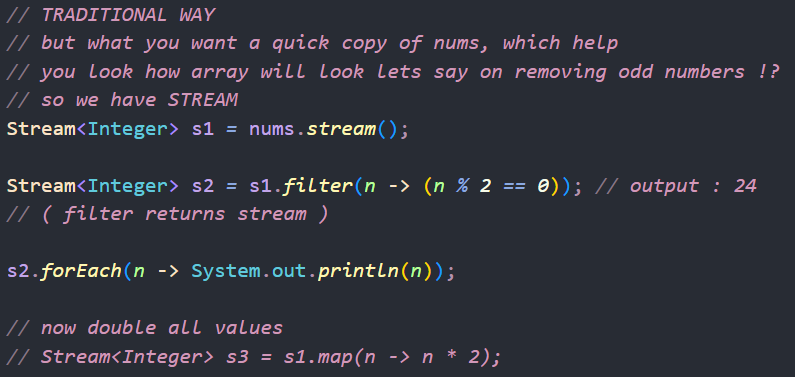
-> arraylist and vector are same in terms of resize ability, **but** vector is synchronous in nature and used when thread safety needed & arraylist are not-sync.

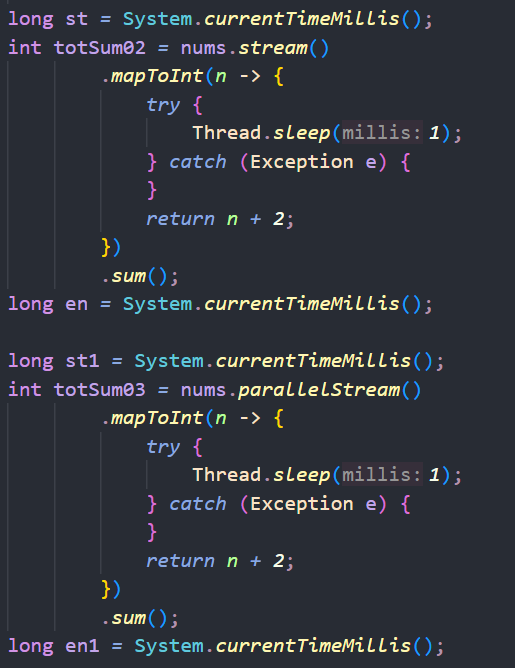
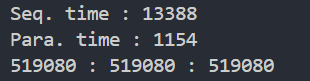
2. Set  


3. HashMap  


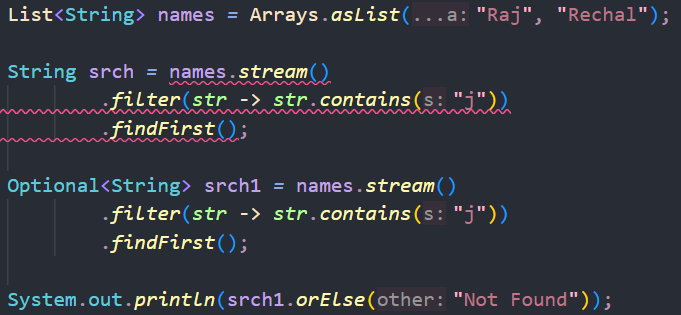
Stream API  
-> they are like one time use array, you use it once then cant use it again & it doesn’t matter you printing or performing some operation

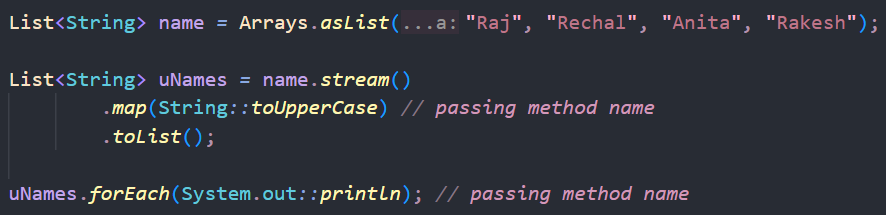


Parallel Stream  
-> Use it when we need wait for sometime to process a function  
 Output ----> 

So in above example we have an array of size 10,000 and we need to increase all its value by 2 then sum all those value  
If we don’t use ***Thread.sleep()*** then Sequential Stream is faster  
But if we use sleep() then Parallel Stream is a lot faster

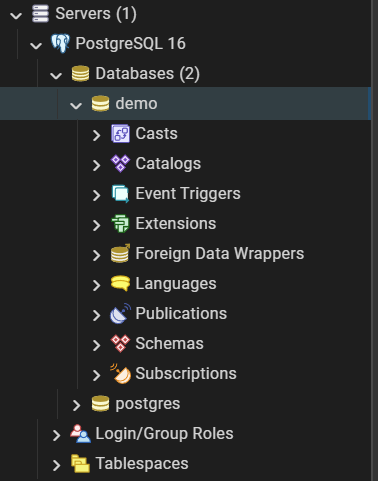
Optional Class  
-> It needed when you search for a value and it is not found, so stream return **null** which cant be stored in normal data types

-> In above we observe that if we find some word and it is not found then stream return *null* which we cant store in normal datatype primitive datatype  
So, we use Optional class which at run time decide if the returned value is quantitive or null

Method Reference  


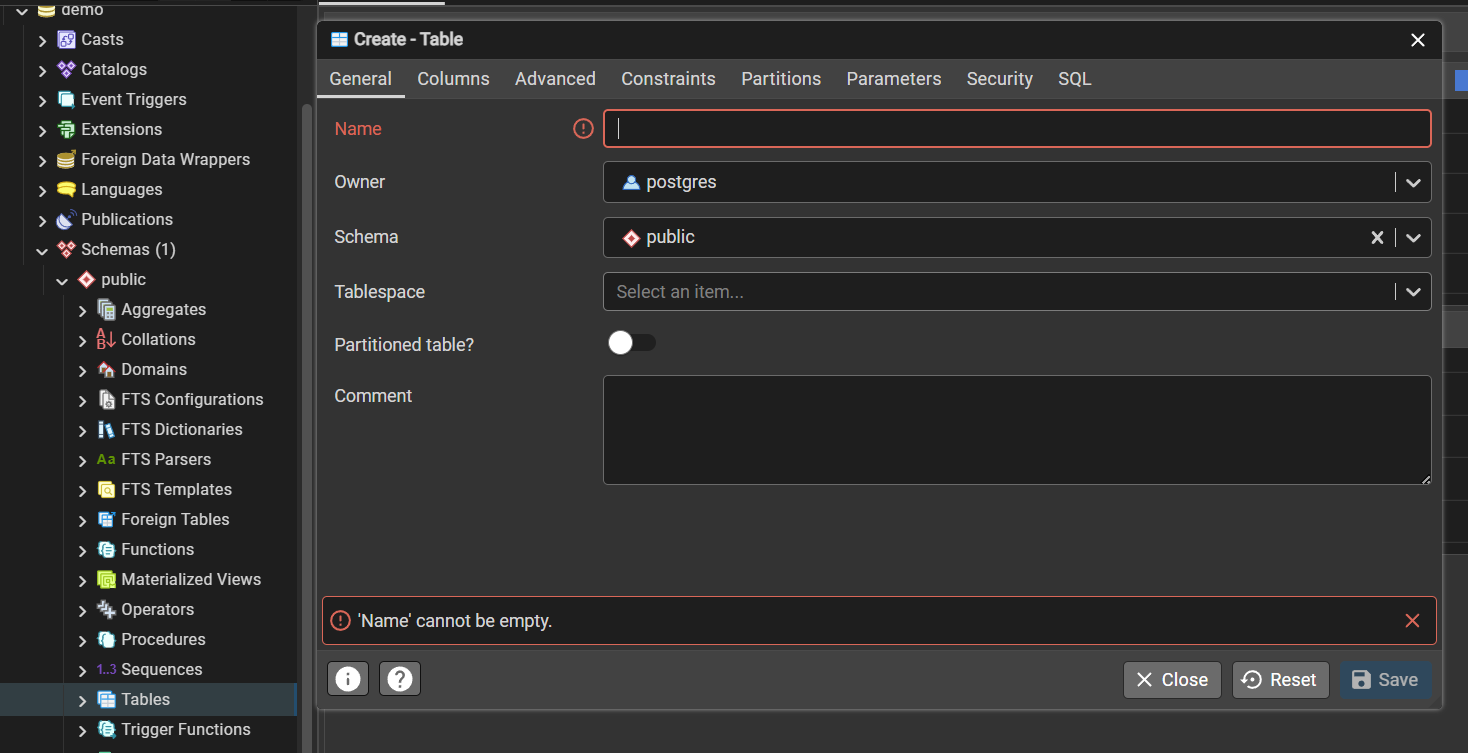
**JDBC**

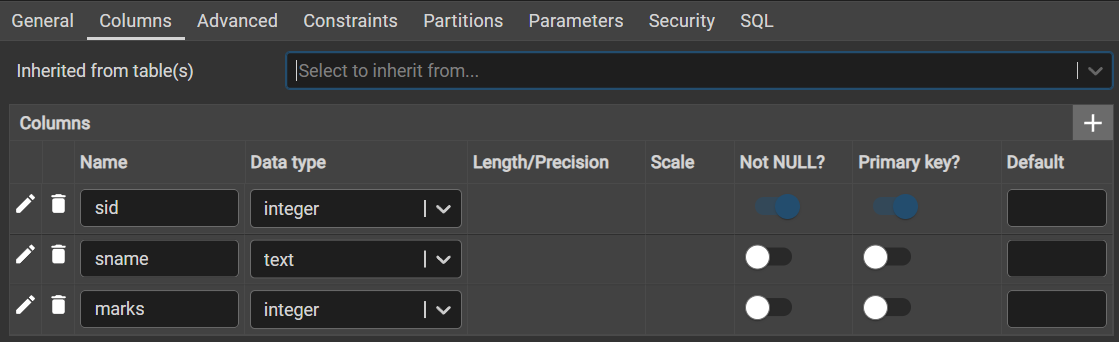
Install postgres sql from their website [ in installed 16.11 version ]  
It ask you set password during installation : [ I use “raj” ]

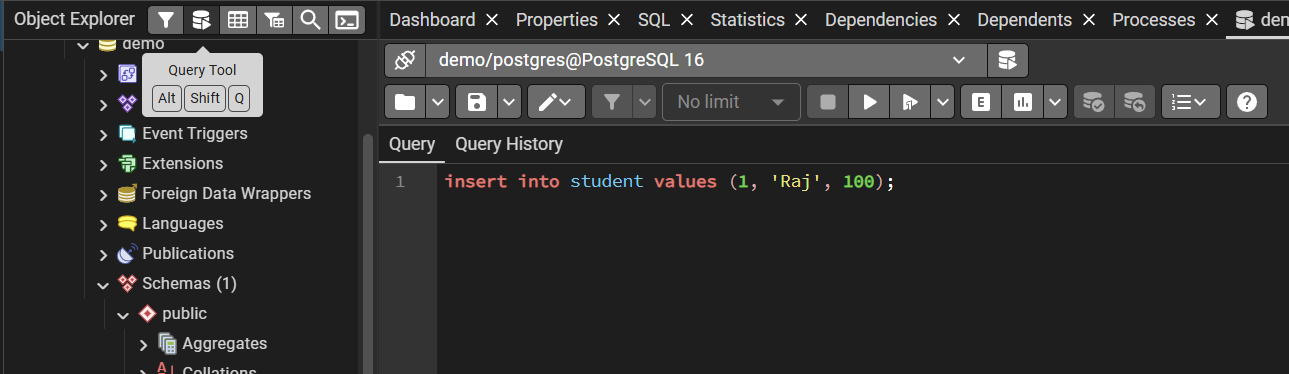
Now open **pgAdmin 4** which is available in your device  


Now click on *Servers*

Right click on *Databases*, create new database name & config ( as per req. )

Now create a table : *Schema* > Right click *Tables* > new table  


Name database [ in our case student ], & set some columns  


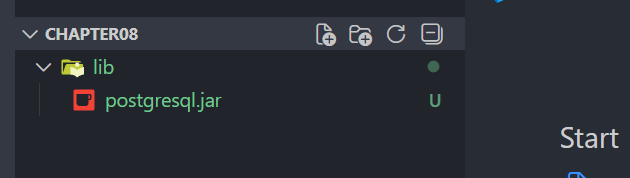
Now further, right click on student table you built and start entering the data  


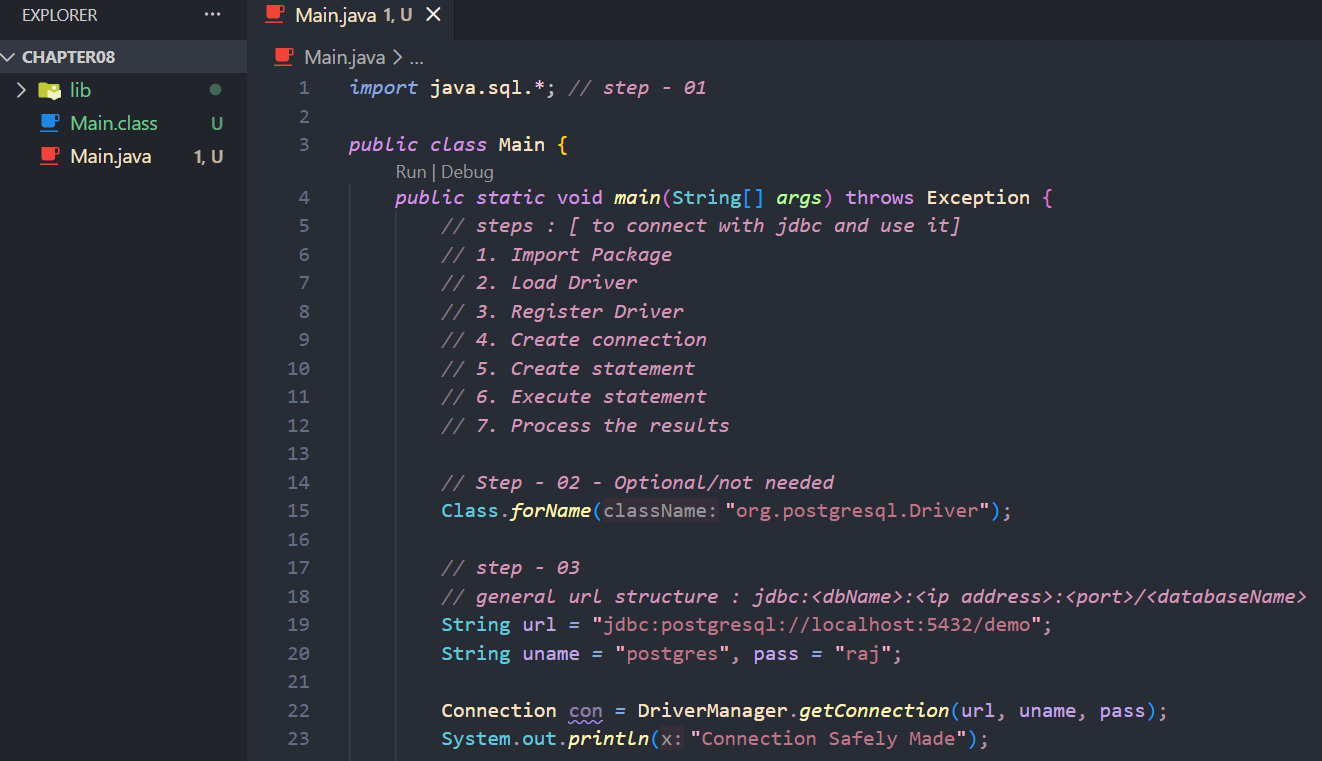
Enter query to input data in table, Use ***Query Tool***

Now to Operate it with **jdbc** we follow 7 steps  
1. Import Package  
2. Load Driver  
3. Register Driver   
4. Create connection  
5. Create statement   
6. Execute statement  
7. Process the results ---->> 8. Close the connection

Now install jdbc Driver from official or maven website

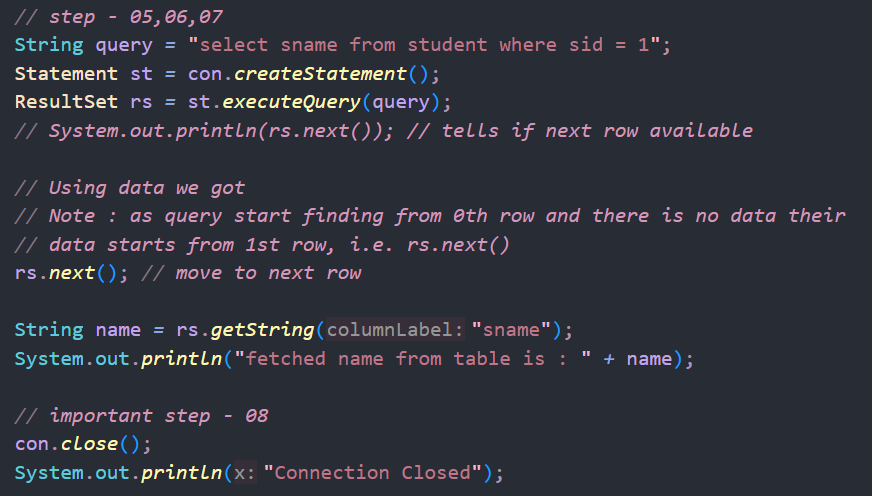
 [ rename file, move it to project location ]

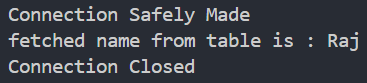
 [ move it to lib for cleanliness & structure ]

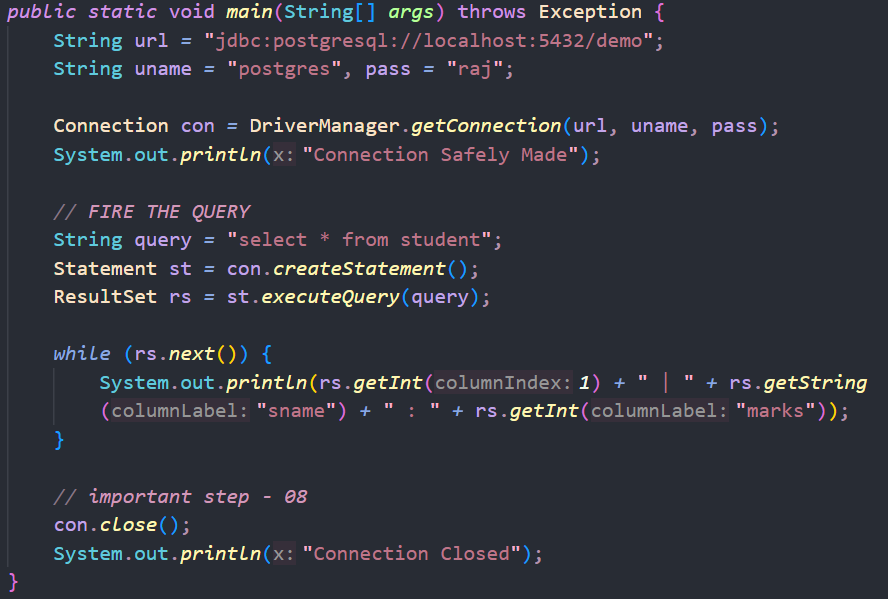


Now for running java code of Main.java you have to first compile the code, but wait if you do normal javac Main.java you will get **error**  
As now we are using jdbc API we need to pass our code through it, So use :- 

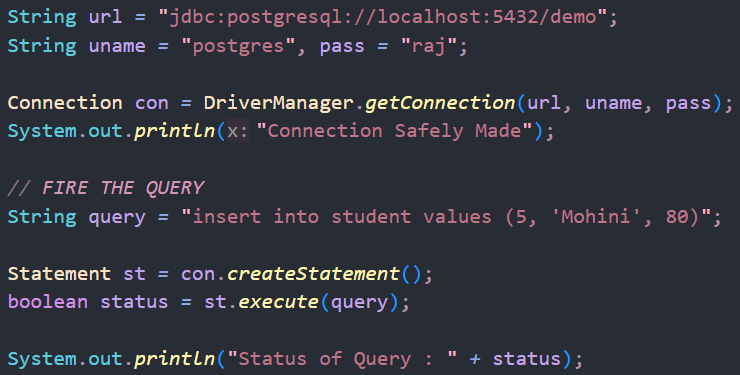
**-cp** tells *classpath* of *postgresql.jar* file which help as translator/API



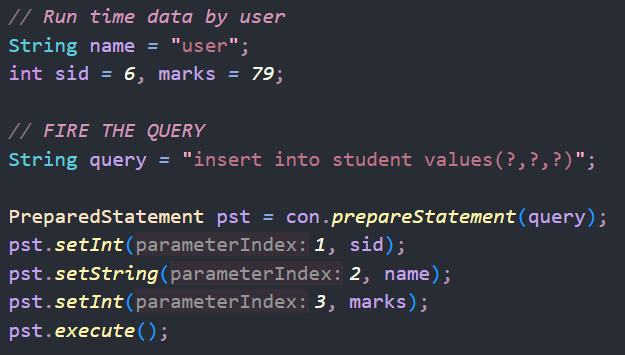
Output : 

**Fetch all rows**

**CRUD OPERATIONS**-> we just need to update the query for update, delete, insert, create. But still e.g.

****



**Prepared Statement**-> help us to pass runtime variables to SQL QUERIES  
-> not vulnerable to SQL Injection Attacks like normal *Statements*

To Do

Interface

Variable shadowing and method hiding

Statements have 3 type -> statement , preparedstatement, **CallableStatement** [this is not completed ]