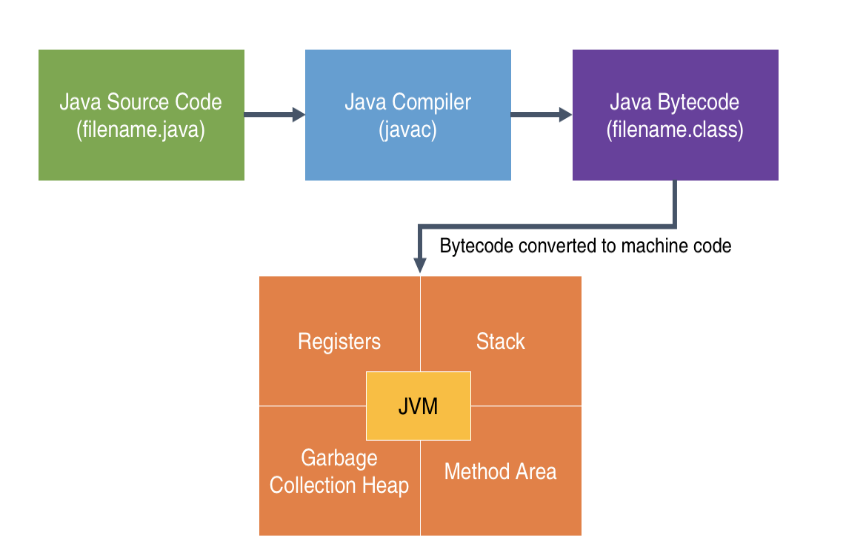
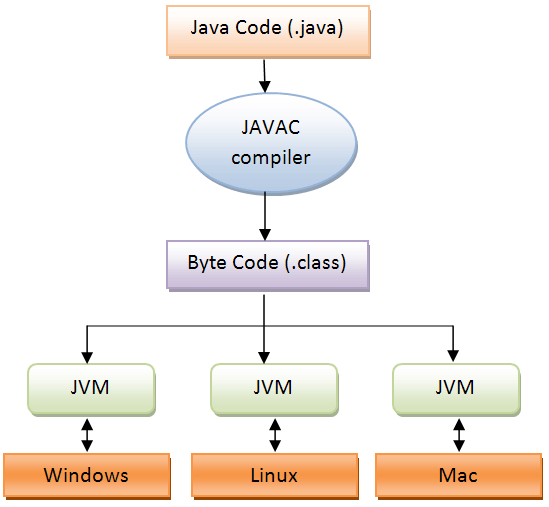
**Java Importance of Telesco:**

For developing java program, we required JDK. So, we require this, it will have compiler and runtime environment which will write the java program and execute them.

* When we install JDK along with JRE and JVM is available for us So, we execute a java program we required a JRE,

JVM -> is part of JRE -> is part of JDK.  
 

What is JRE -> JRE will contain library of java classes.

What is JVM -> For executing Java program we required JVM as a part of JRE, without JVM, JRE is cannot execute, Java program that will running inside JRE, that executed by JVM.

Look at the flow diagram:

EX: First.java -> Java c -> First. Class

Java c – Java c is a compiler that is the part of JDK.  
First. Class – This executes by java program; this will do by JRE and internally it uses JVM.

**Explanation of main method:**

Main method – public static void main (String args []) {}  
void -> main method should be void always it should not give any return value  
Why access specifier as:  
public – Because when the JVM is execute the program, call the main method so this main method is inside the class JVM should be able to see the main method, this reason we make it as always public.  
Why the static?  
static – If it is a static then we can directly call without creating a object of a class just by using class name.  
Command line arguments:  
String args[] – This we can called as commend line arguments, This is optional, But the java program we must write.   
println – println is a method printing anything on the screen.

out – is an object, in java nothing is outside of classes and object, so this inside some object called as out and that is inside a class System.

For the main method:  
public static void main () -> written like this JDK will compile the program, but JVM will not compile. When JVM do compile we written like this only public static void main (String args [])  
  
**How to read the data from the keyboard in java:**

Java provides a Scanner class that will use to read the data from the different sources. Here to read the data from keyboard.  
Scanner class is present in util package, there are some builds in packages in java one of is util, this class was provided by java version 5 onwards.

We are using a Scanner class, first we are creating an object.   
 Scanner s = new Scanner (System.in);   
Scanner – This is the class  
Scanner – This is the constructor  
System.in – This object is associated with keyboard. -> This object is given to constructor

Scanner and storing into a reference variable.  
**This are a reference for the keyboard:** Scanner class having some methods are:

1. nextInt () – Will read an integer
2. nextFloat () – will read a float value
3. nextDouble () – will read a double value
4. next () – If we want to read a String name or any word, we use this method, read single word
5. nextLine () – If we want to read a sentence like a line use this method, read multiple words.
6. nextByte () – it will read the byte value
7. nextShort () – it will read the short value
8. nextLong () – read the long value
9. nextBoolean () – read the Boolean value

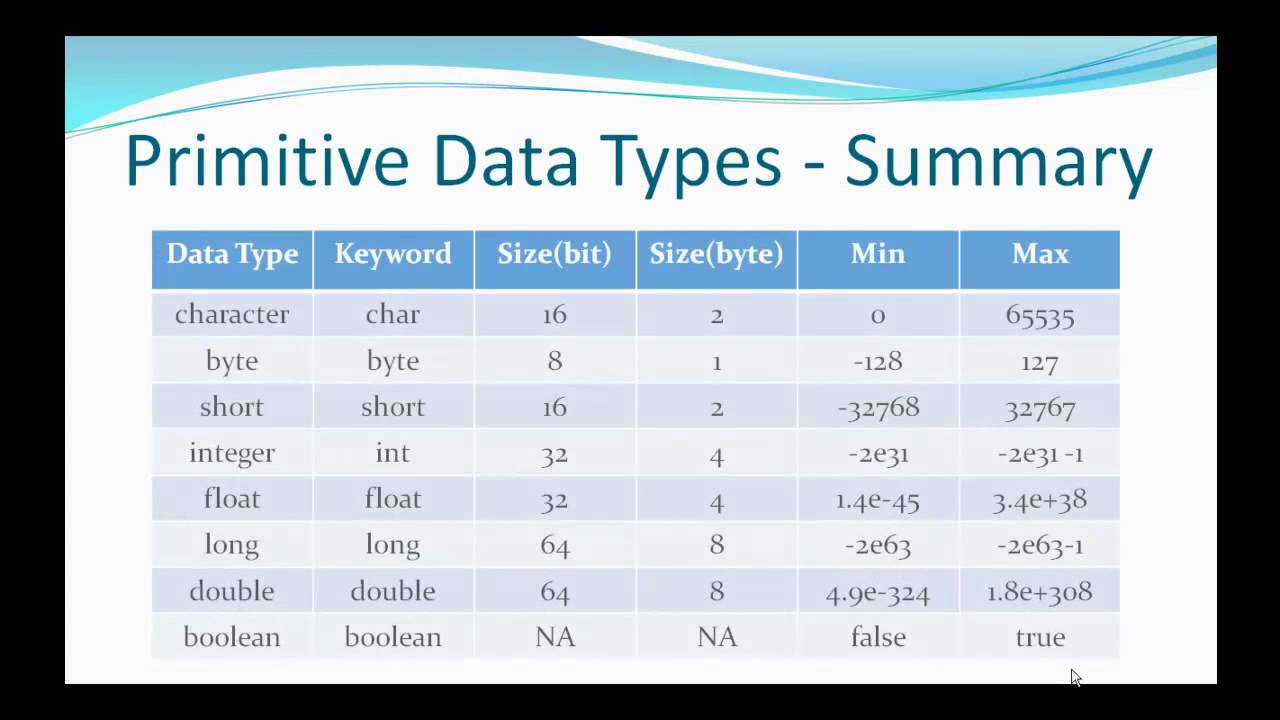
**There are other methods are also available are:**

1. hasNextInt () – Before reading an integer, I want to confirm next value is integer or not we use this method. So, it returns Boolean value.
2. hasNextFloat () - Before reading a float value, I want to confirm next value is float or not we use this method. So, it returns Boolean value.

**Important**: We scanner class; we would know which methods are there in scanner class.  
In CMD using this command we are getting an all methods. Using command javap java.util.Scanner  
javap – is a utility from java  
java.util – This is the package name  
Scanner – This is a class name  
Using this command it will show all the methods.

**Data types of variable literals of Java:**   
  
Data is an important part of a program, all process we doing upon data, program is running on the memory, program should hold the data temporarily in the memory not and permanently in the hard disk memory, during the execution of the program, program should hold the data.  
So, where do we store the data? We store the data using variables so variable is meant for storing the data.  
Variable will have some data type, means that type of the data is storing that, some pre-defined types of the data available in java are:

Primitive data types: It means the basic data types in java that is built in compiler of java  
1) Integer – It means only integer value without any decimal value   
 \* byte   
 \* short  
 \* int  
 \* long  
2) Floating point – They can have numeric value and with decimal point  
 \* float   
 \* double  
3) Character – This is just storing for a character  
4) Boolean – This is just storing for a true or false.

In java every data type there is a built-in class is available using this command in CMD:  
Integer – javap java.lang.Integer  
Float - javap java.lang.Float  
 

**What are variables?**  
A variable is used to storing the data, variable must have some data type.  
Variables - Naming rules

1. Case sensitive
2. Contains alphabets, Numbers, \_or$
3. Starts with alphabets, \_or$
4. Should not be a keyword
5. Should not be a class name, if class is also use
6. No limit on length of name
7. Follow camel cases

**Literals:**Literals are all constant value that are used in our program, literals are also called as data type.  
**Example:**

1) z = 5\*x+7\*y -> This is for integer literals.  
 2) int value = 25 -> This is for integer literals.  
 3) double price = 153.78 -> This is for double literals.  
 4) area = 3.1425\*radius\*radius; -> double literals  
 5) char c = ‘A’ -> This is for character literals  
 6) String s = “java” -> This is for the String literals.  
Integer literals number can be represented a various number systems like:   
1) Decimal – This will have a 0,1,2,3,4,5,6,7,8,9   
2) Binary – This will have a 0,1

3) Octal – This will have a 0,1,2,3,4,5,6,7  
4) Hexadecimal – This will have a 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F  
Our daily basis we are using a decimal value only.  
  
Byte, short, int = int literal Ex: byte b = 5; short s = 10; int I = 12;  
long is L or l literals Ex: long l = 10L; or long l = 10l;

Float is F or f literals Ex: float f = 10.2F; or float f = 10.2f;  
Double is D or d literals Ex: double d = 10.2D; or double d = 10.2d;  
char is ‘’ like this literals EX: char c = ‘a’;  
Boolean is true/false literals.

Floating point number: This number is don’t really store in the decimal in memory but the represent decimal number.  
Ex: 163.52\*100/100 = 16352. \*1/100 = 16352. \*10^-2 -> This how the represents the decimal number  
16352. –> This portion is called as mantises, 10^-2 -> This portion is called as exponent.  
Finally we representing as 16352E-2

Double number: For this up to 6-7 decimal value is suitable as float, if more than 6 decimal value we consider as the double it suitable up to 14-15.

Character: Basically, computer doesn’t support alphabets, then we use numeric value instead of these characters, for those numeric codes English alphabets we called as ASCII Code.  
So, these Ascii codes are uppercase, lowercase alphabets and the digits, other special characteries anything in the keyboard all is available for ascii code.

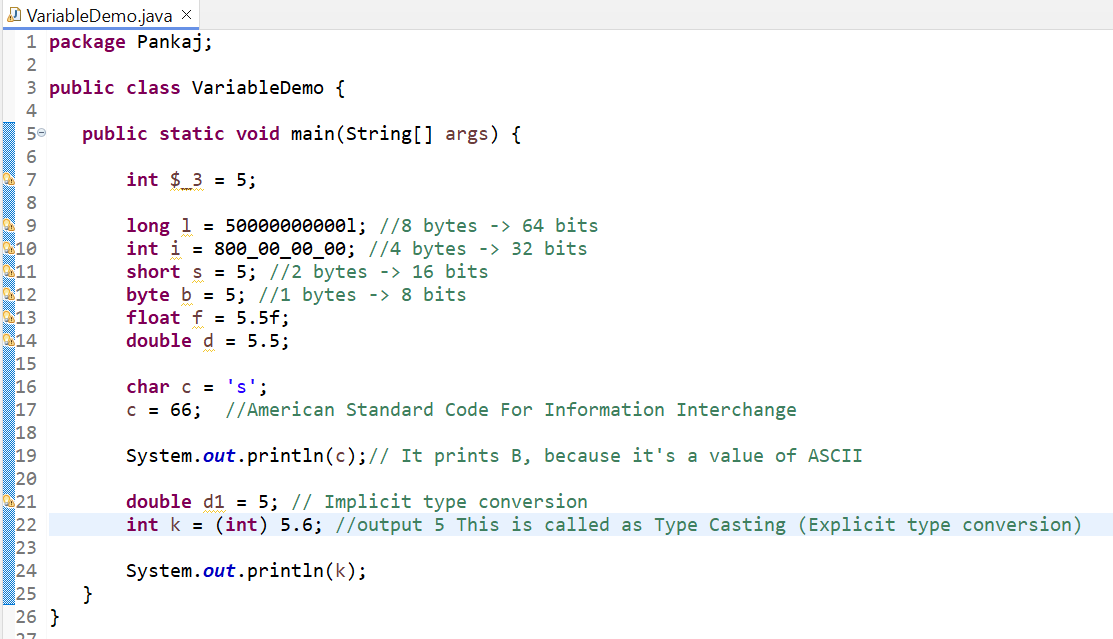
Code defines for all natural languages like: English, Kannada, Hindi etc. So, those code are called as Unicode.  
This ascii code is the part of Unicode. We need to find out our natural language code we use unicode.org.

EX: Ascii value are  
A, B, C, ------Z a, b, c, -------z 0, 1, 2, -----9 range is 0-127  
65 66 67 90 97 98 99 122 48 49 50 57 7 bits is sufficient

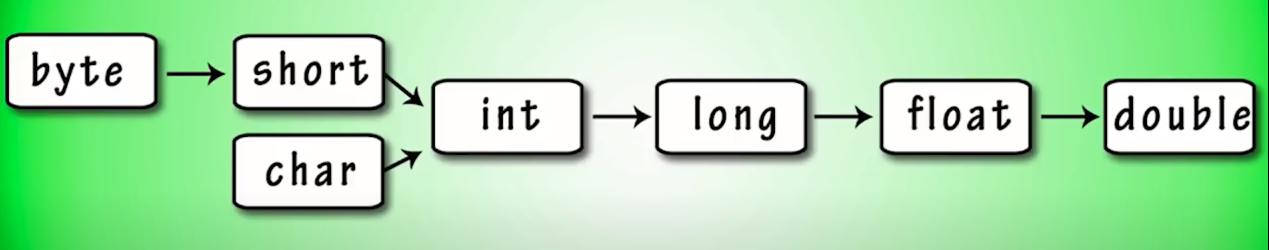
We talk about various supporting various natural languages 7 bits are not sufficient we need more spaces; this reason java takes 2 bytes for the character to support Unicode.

Boolean: This Boolean data type allows only true or false. For showing true or false 1 bit is sufficient 0 -> we can say positive and 1 -> we can say negative.

NetBeans -> NetBeans is an IDE (Integrated development environment) for writing programs mainly java programs, even though it supports other programming language also primarily it is used for java program netbeans.org.



A hand holding a pen

Description automatically generated

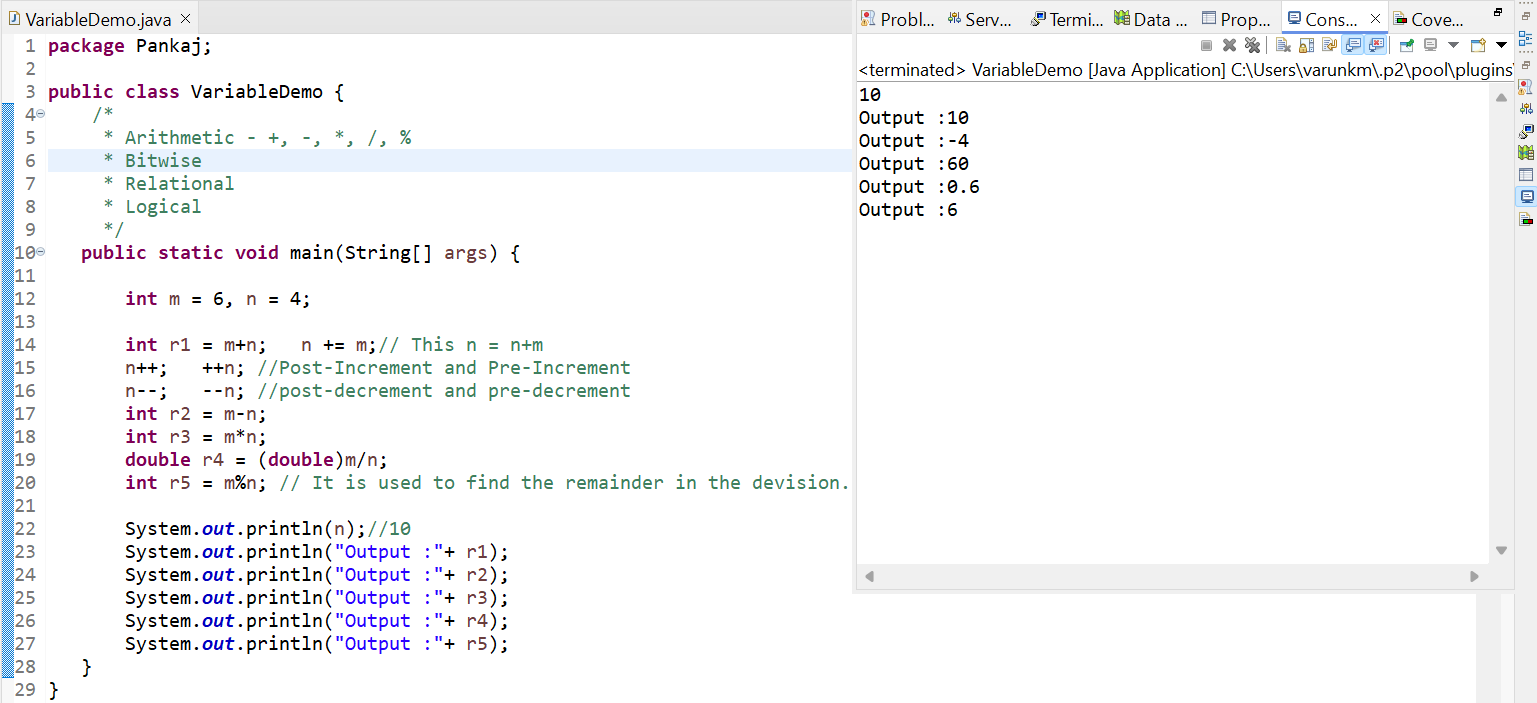
**Naming Convention in java Example:**

1. **Interface** name should be **Adjective**: Ex: Runnable, Readable, Remote
2. **Class** name should be **Noun** Ex: Student, Person, Computer, etc.
3. **Method** name should be **Verb** Ex: actionPerformed, run, print, write etc.
4. **Variables** Ex: stockprice, bankdetails
5. **Constant** Ex: PI, DENSITY, MAX\_PRICE, MIN\_PRICE etc.
6. **Packages** We are using everything small

Camal case rule: Class, Interface, Constructor  
Snake case rule: Method, Variable

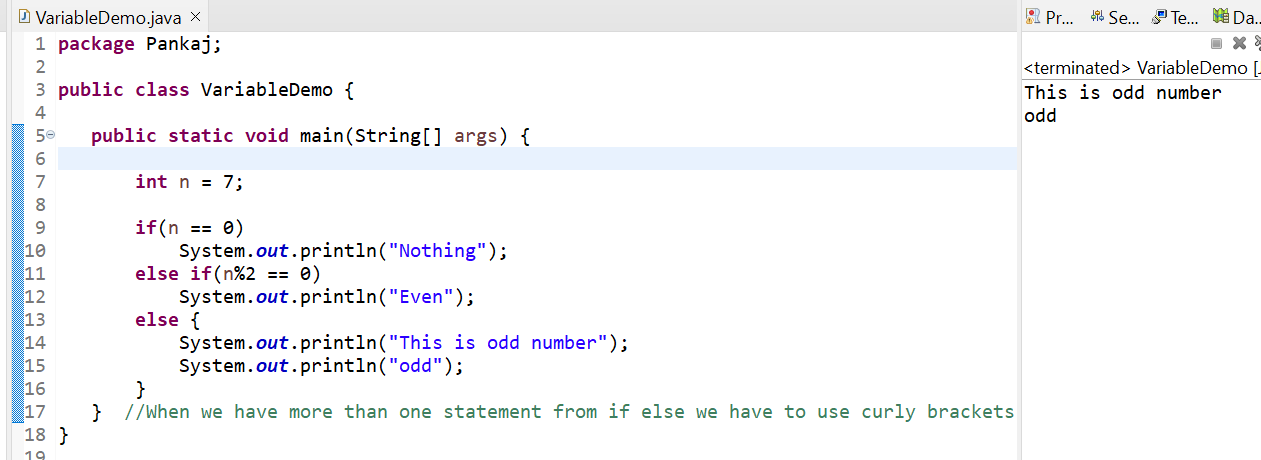
**operators in java:**

1. Arithmetic operator
2. Bitwise operator
3. Relational operator
4. Logical operator

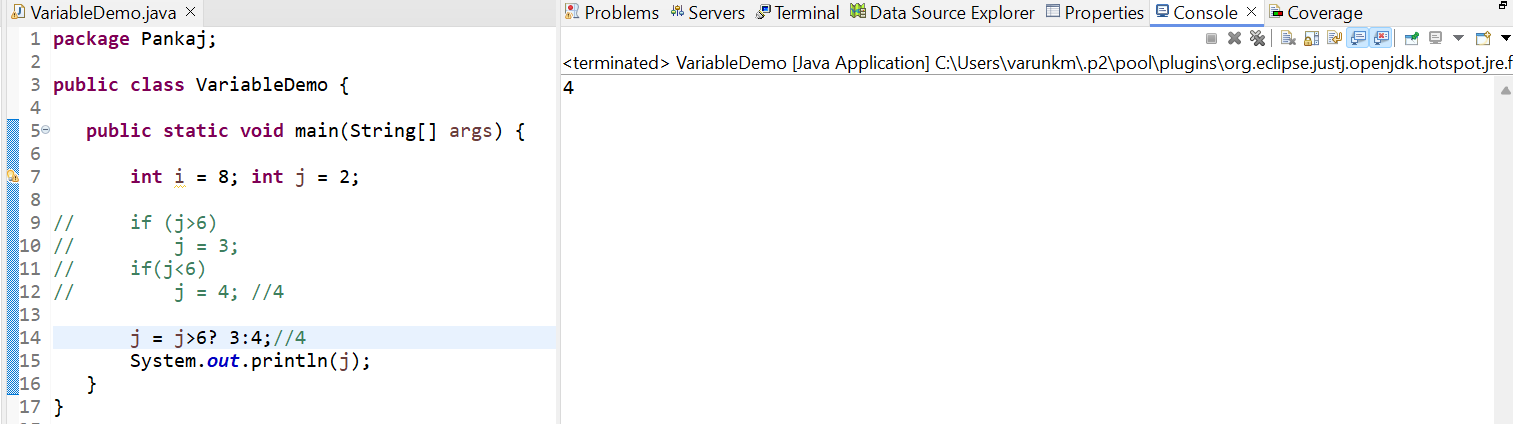


**If Else, switch and ternary operator Selection Statement:**

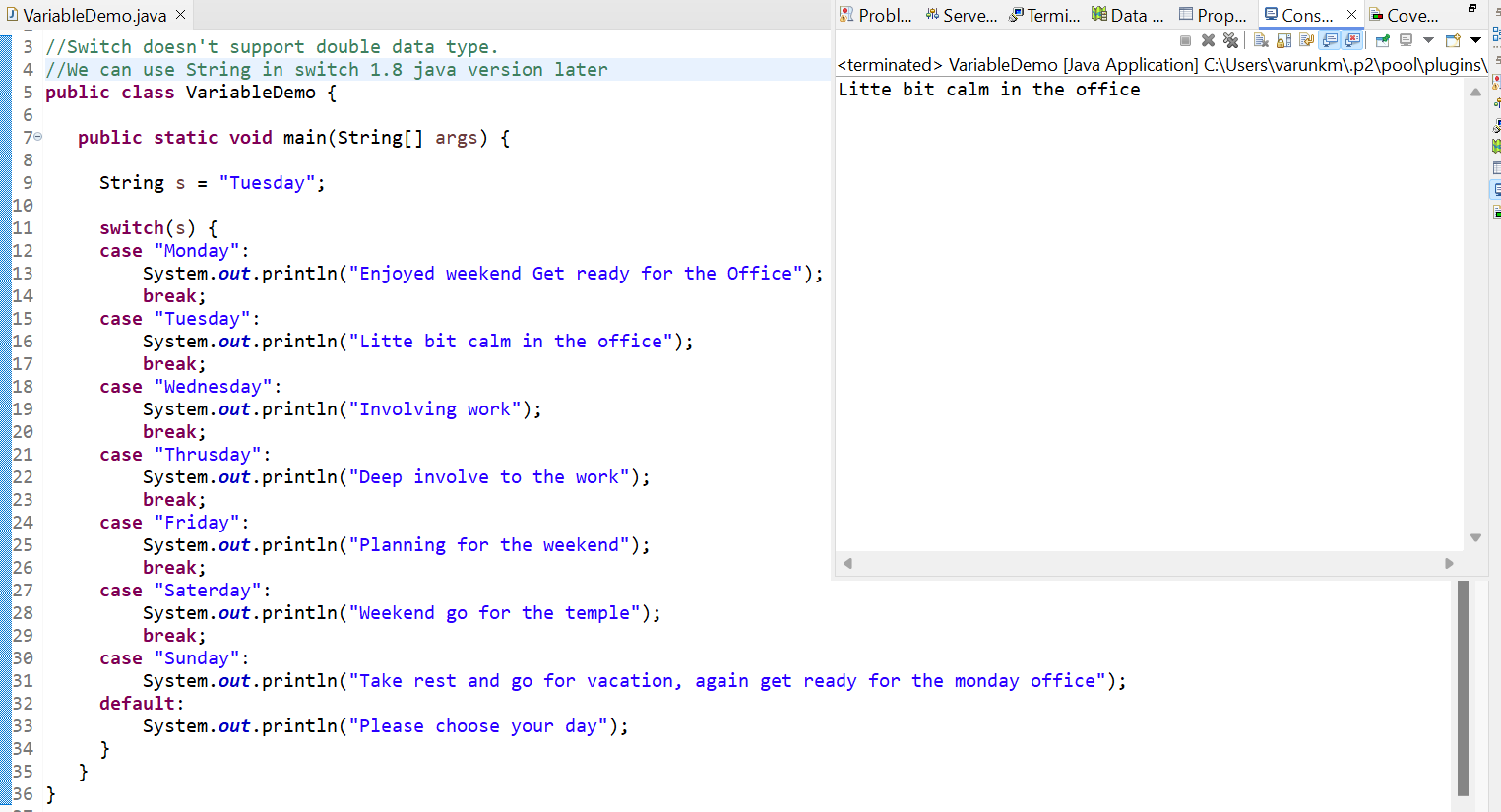
1. **If Else:**

****

1. **Ternary operator:**

****

1. **Fundamentals of selection and switch statement:**

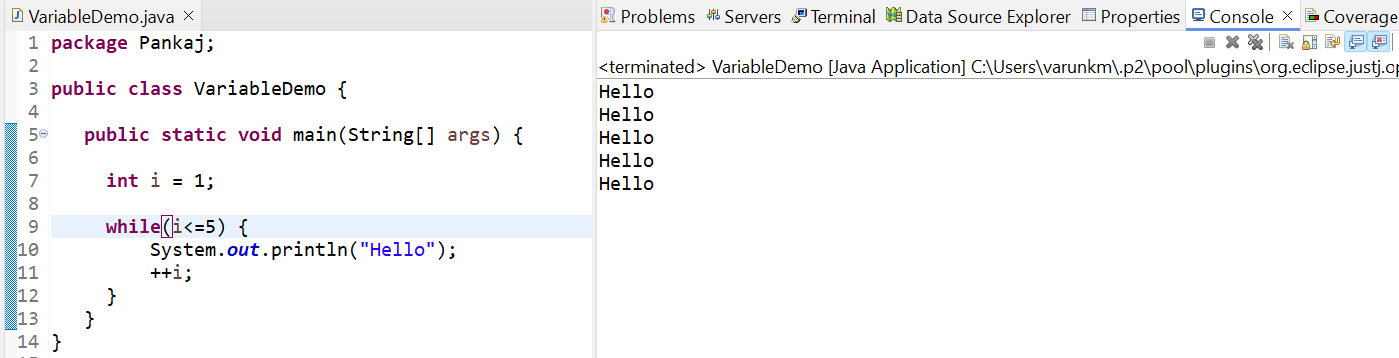
****

**While and for iteration Statements:**

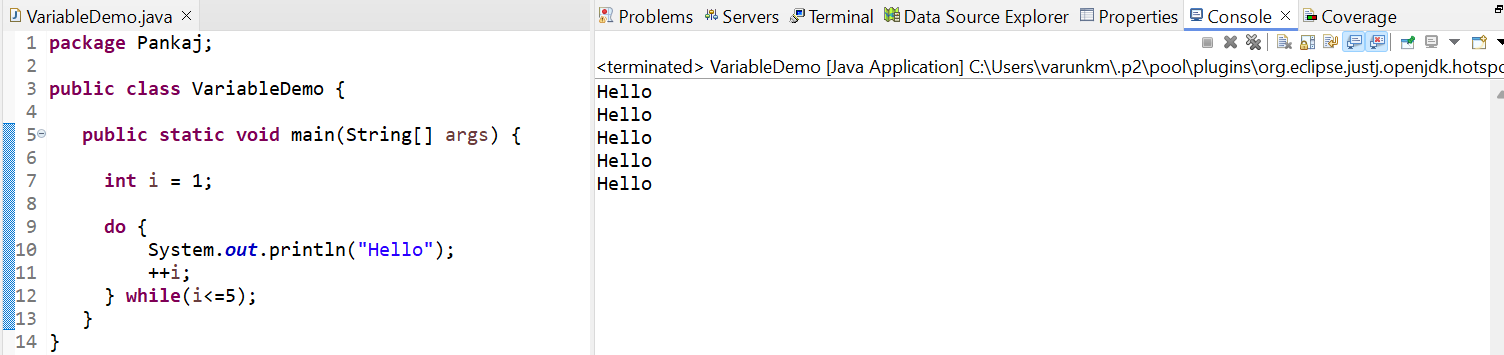
Iteration means when we want to repeat something that time we will go with iteration. Actual implementation of iteration or loop is we can **while loop, do while loop or for loop and we have advance loop called as for-each loop.**

1. **While loop** – Basically when we want to use counter for a loop, we can go with three steps are

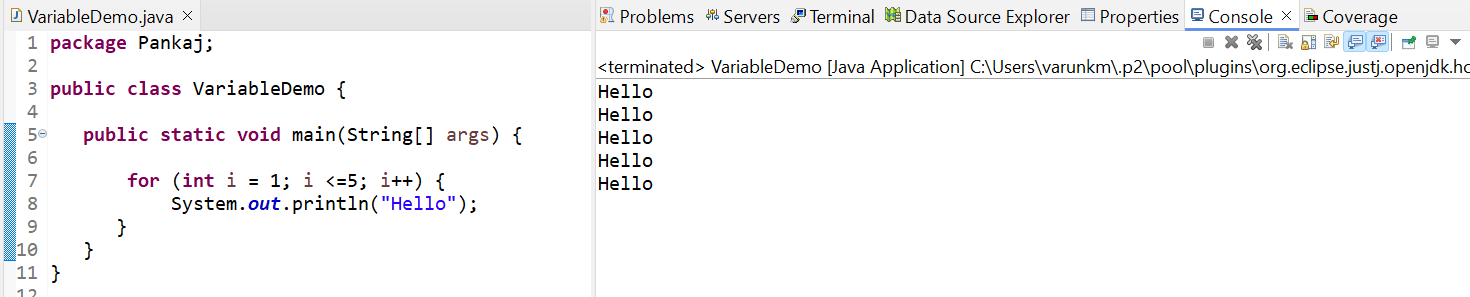
* Initializing a value
* Condition and
* Increment or decrement



1. **Do while loop** – It’s same as while loop but, checked the condition later.

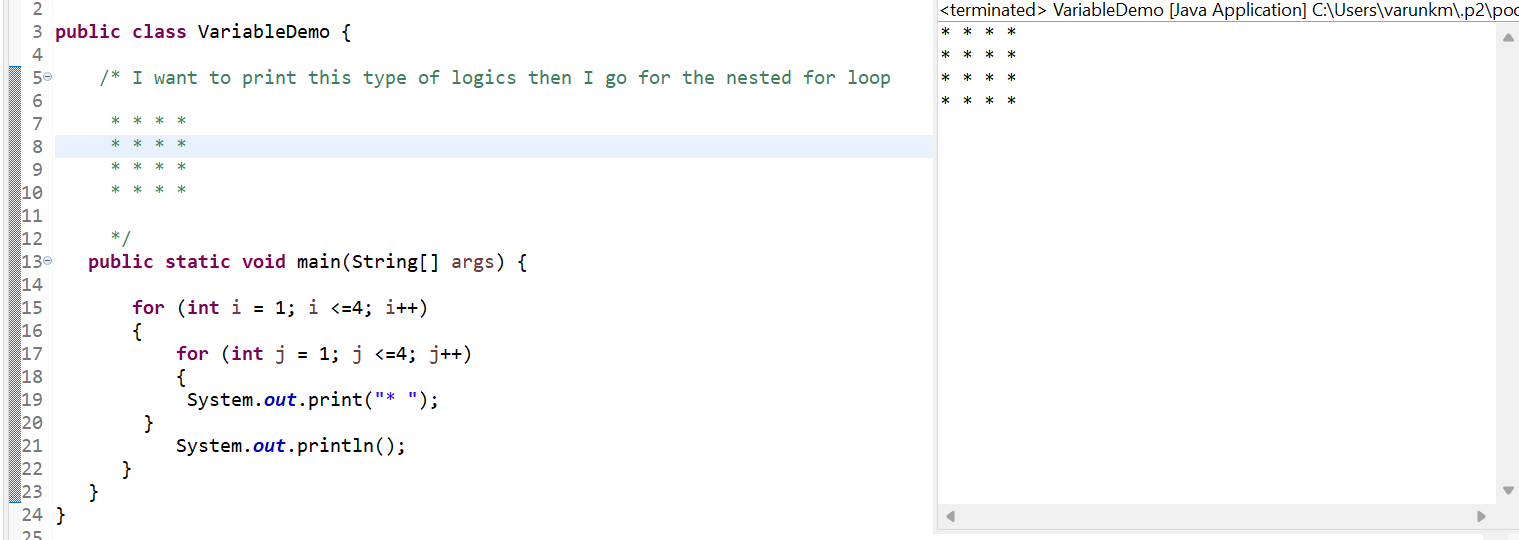
****

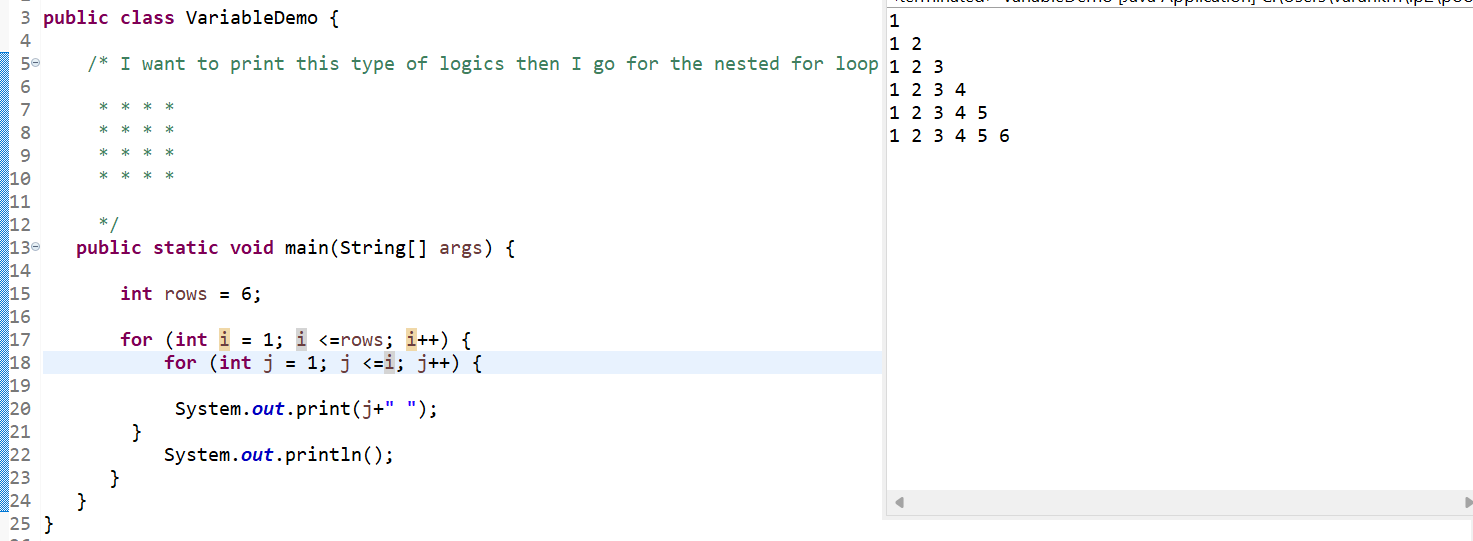
1. **For loop** – Using for loop everything we can write in one line.

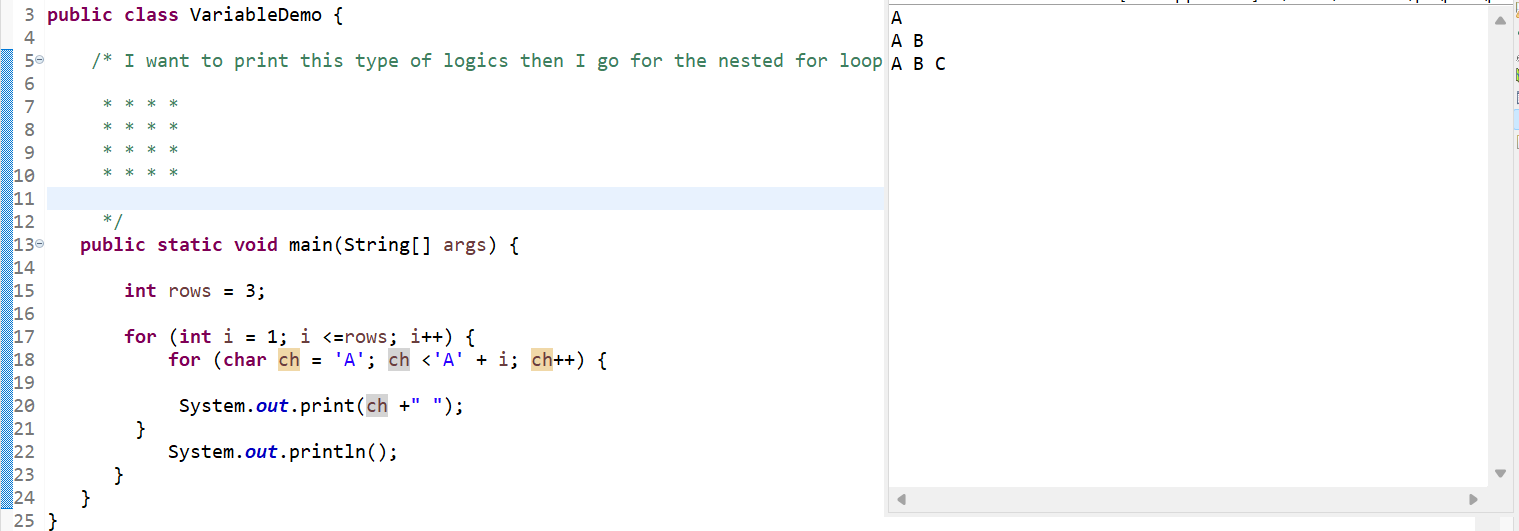
****

* We know that starting and the ending point as above example we **start 1 and end with 5**, we always go with the **for loop**.
* Sometimes **we don’t know the number of times start and end point** in this scenario always go with the **while loop**.
* **Even our condition false** we want to execute that statement once go for the **do while loop**.
* **For each** is the special type of loop especially made for arrays and collections.

1. **Nested loop** – I want to print logical patterns we go for the nested loop.





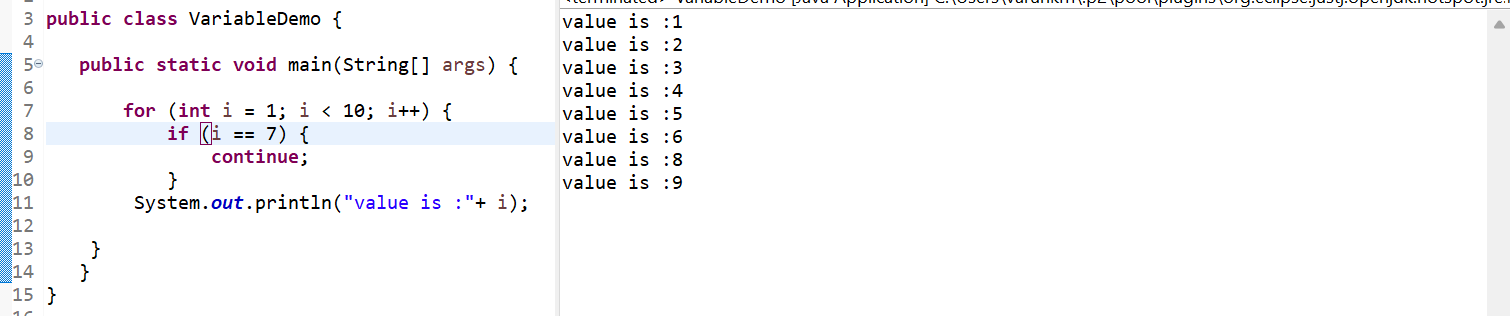


A screenshot of a computer program

Description automatically generated

**Break and Continue Statements:**

**Continue** – So, continue is skip the one iteration, the below example 7 has skipped. It is also called as jumping statement.



**Break** – The below example I want to print up to 5 I don’t want to the rest all numbers. It will break the loop it will not allow any further statement.

A screenshot of a computer

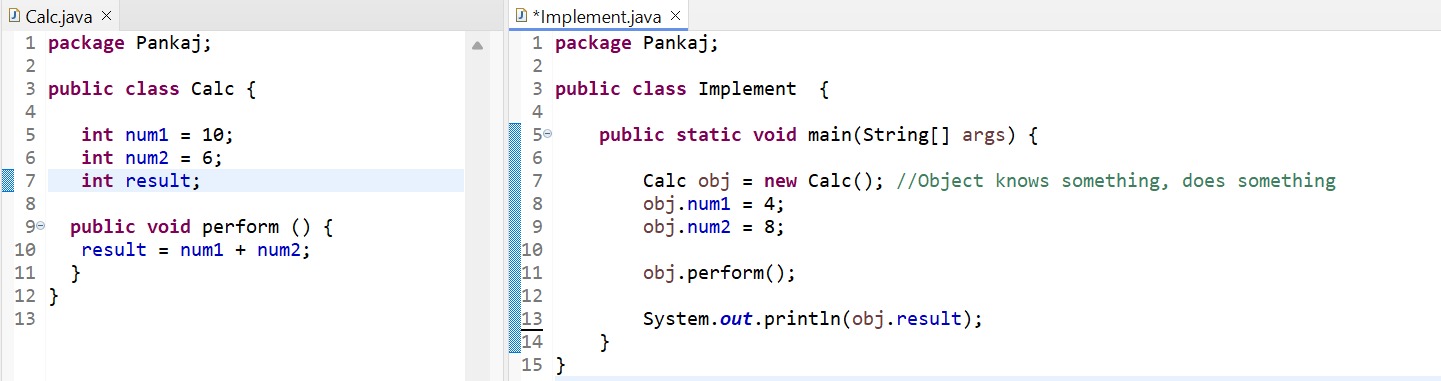
Description automatically generated

**Class and Object in Java**

We just imagine building is a one object, to build any object our JVM needs a structure or blueprint, that blueprint will be our class. So, class defining an object.

So, we have defined that what object knows and what object does we require a blueprint, in this blueprint we can define what is the behaviour and values of the object.

New keyword is responsible to create allocate the memory, **how much memory we need that will be define by constructor**.



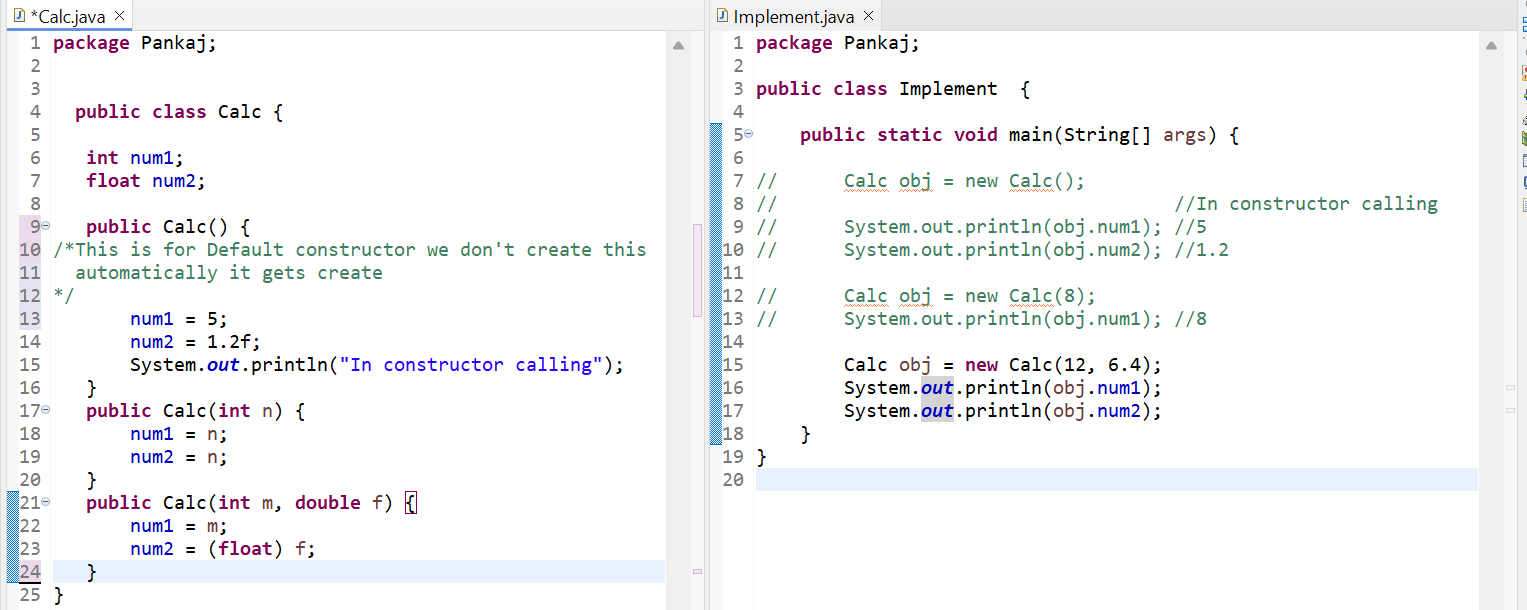
**Constructor in java**

**public** **class** Calc {   
 int i; //4 bytes  
 float f; //4 bytes  
}   
Create a variables **int** and **float** both occupies 4 bytes. What about the object? How much space is required for object? This answer will give by the constructor. For Every class even we don't mention their constructor will have. It will be used to allocate to the memory to the object.

* Constructor like a member of method which as the same name as a class name.
* Constructor will never return anything.
* Every time we have create an object we have to create a constructor.
* Even we don’t define the constructor it will be there inside the class. That’s why this constructor is called as **default constructor**.
* We want to provide the initial values of the variables in constructor.
* Constructor will be called automatically when the object is created.

**Constructor Overloading:**

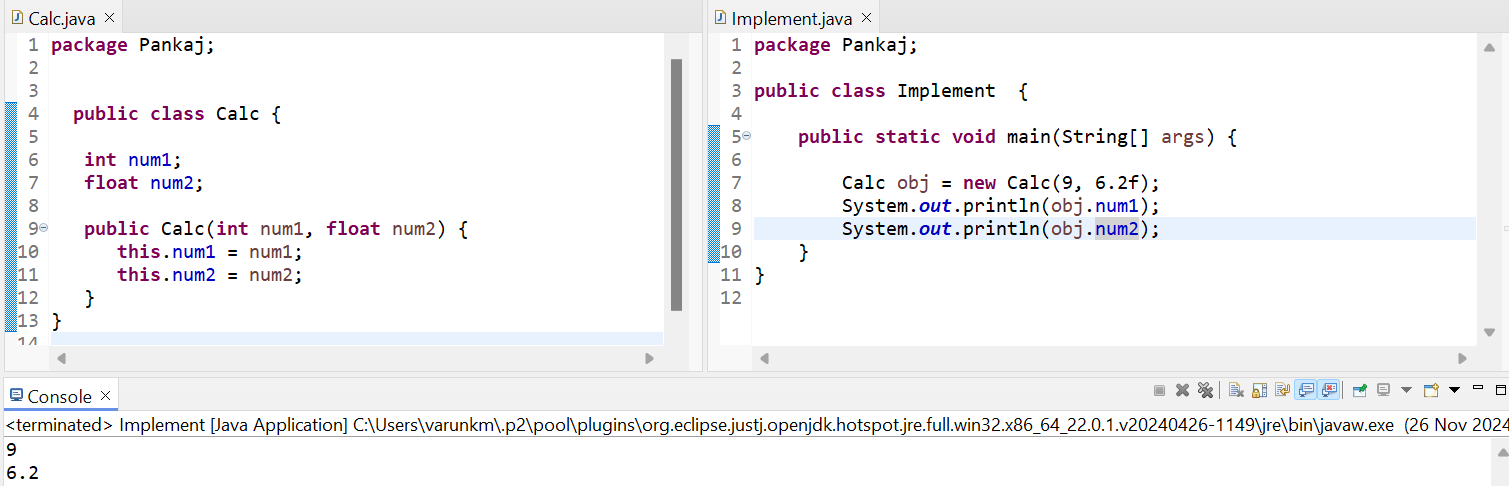
We can create a two constructor in the same class, provides they have different types of arguments and different number of arguments. This is called a constructor overloading.



**This Keyword:**

It is not compulsory use this keyword, when we have different variable name. But when it comes to local and instance variable will have the same name refer to the instance variable, we must use **this keyword**.

* So, this keyword represents the current object or instance.



**Method and Constructor Overloading**:

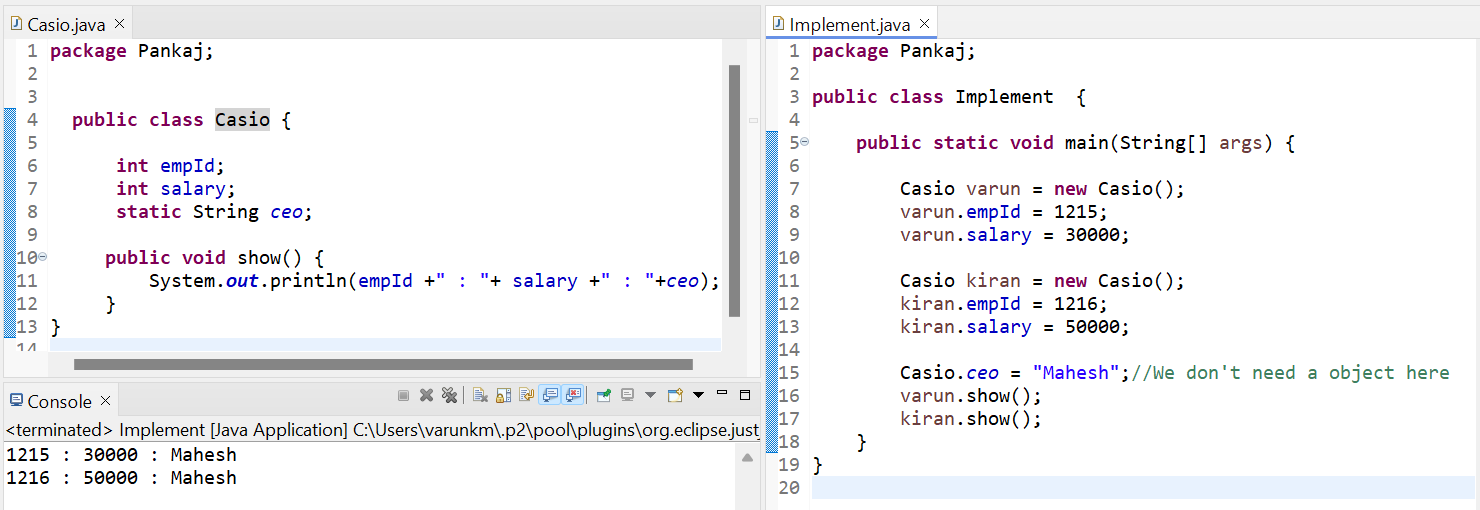
1. **Method Overloading:** Inside the class, if we have a same method name with different parameters, it is called as method overloading.



1. **Constructor Overloading**: We will create a different constructor in the same class with different number of arguments and types of arguments.



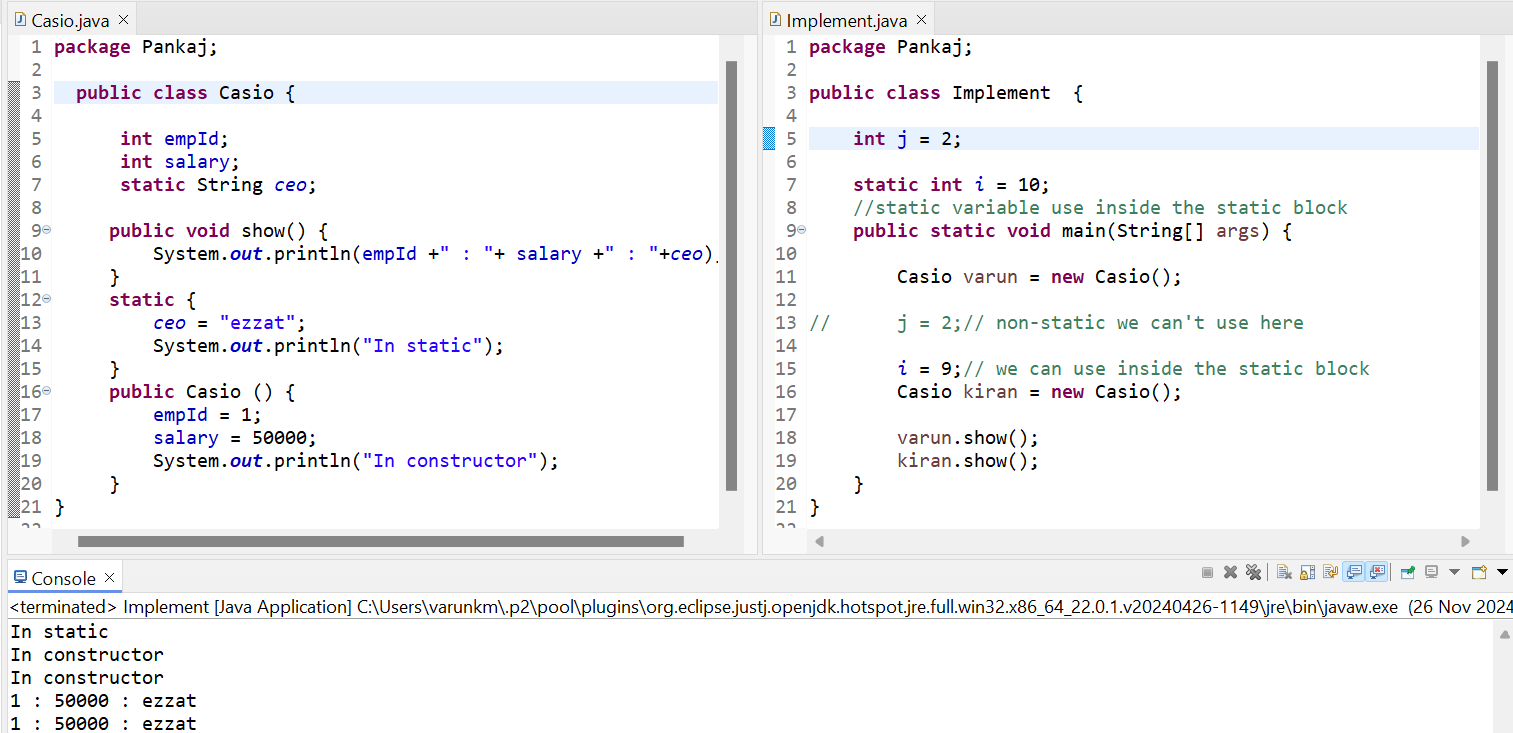
**Static Keyword:** If we make a variable as static, it will be same for all the objects



We can initialize the non-static variables we can use constructors. But we can initialize static variables we can use the static block.

Below example Constructor will execute when we create an object, static block will execute when we load a class. It doesn’t matter how many objects we create the class will load only once.

* Static block will be executed before the constructor, because class loads first then executes object.
* Static variables are same for the all the objects, non-static variables are different for all the objects. But we cannot use non-static variable in static block.



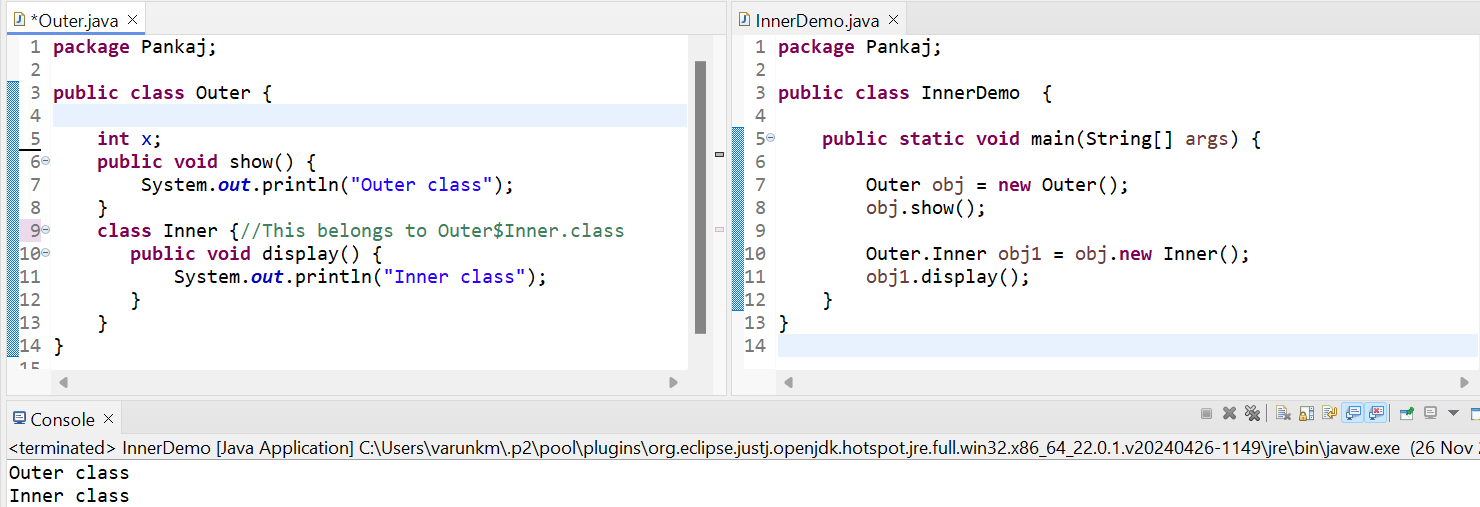
**Inner class:**

We can create a class inside the class, but whenever we are calling the inner class, we have use outer class or main class.

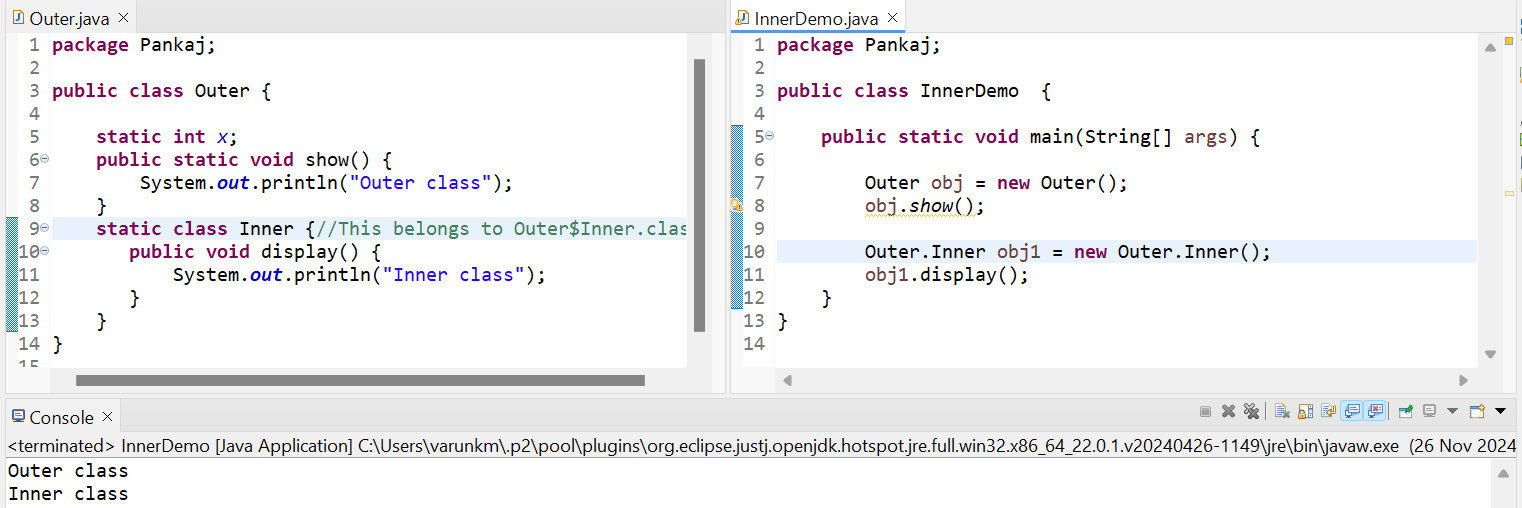
Why create an inner class?   
 Sometimes there is a requirement that we need for that class only for that specific purpose example we need inner class only for outer class there is not specific purpose for inner class. Only purpose is to work with inner class. Then we must go for inner class. Otherwise, we can create another class.

1. Member or non-static class
2. Static class
3. Anonymous class

* **Member or non-static inner class**

****

* **Static inner class**

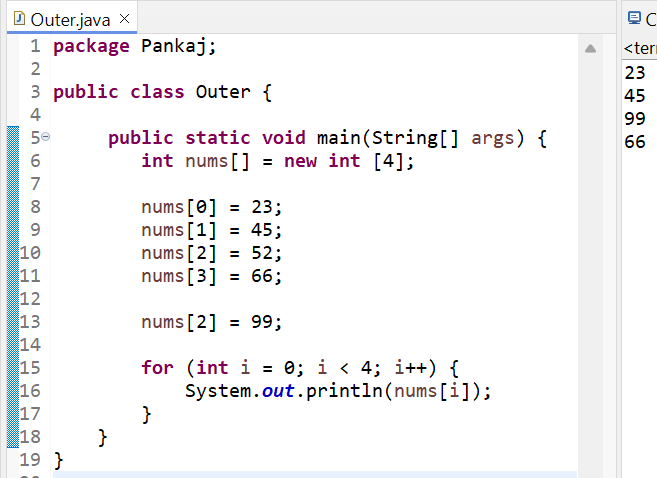
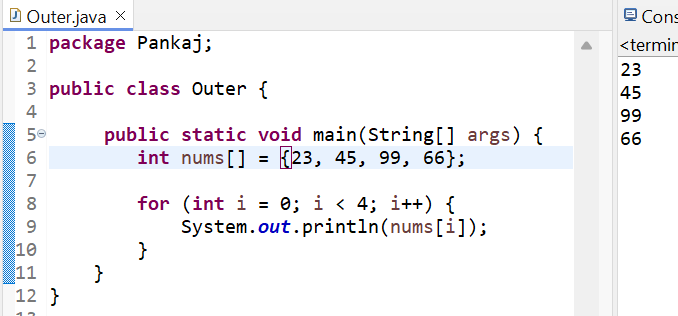
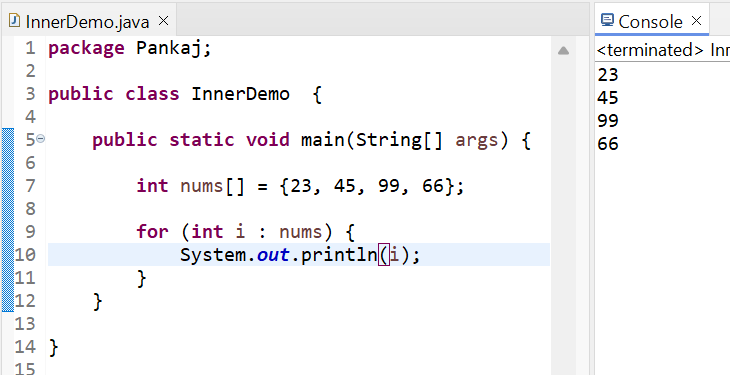
****

**Arrays:**

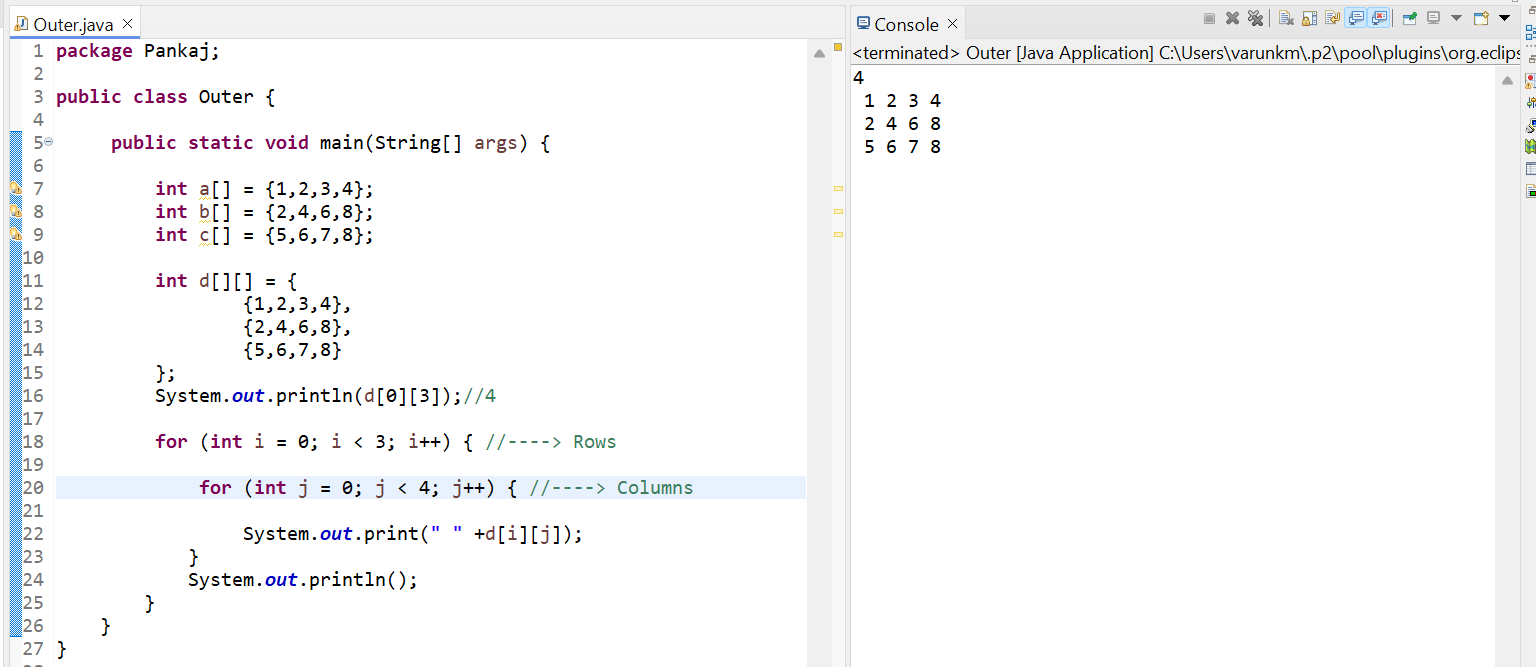
* Array means a collection of elements.
* Every element in our Array will be having a unique number and that number is called as index number. Index number starts with 0 1 2 3 …. Ex: we have 50 elements, but index ends with 0 1 2 3 \_ \_ \_ \_ 49.
* Suppose we have Array that stores 4 elements we are adding this Array 5 elements we get an **Array index out of bound exception**.
* In java Array is an object, when we create the Array, we use new keyword.
* When we are defining an Array, we must define the Array size or else we get an error.
* In Array once we define the value, we can change after.

Ex: I have teacups, I need to serve a people for this I am going to use trey and carry the cups how much is possible. In this example trey is an Array. Trey stores collection of teacups and Array stores collection of elements.

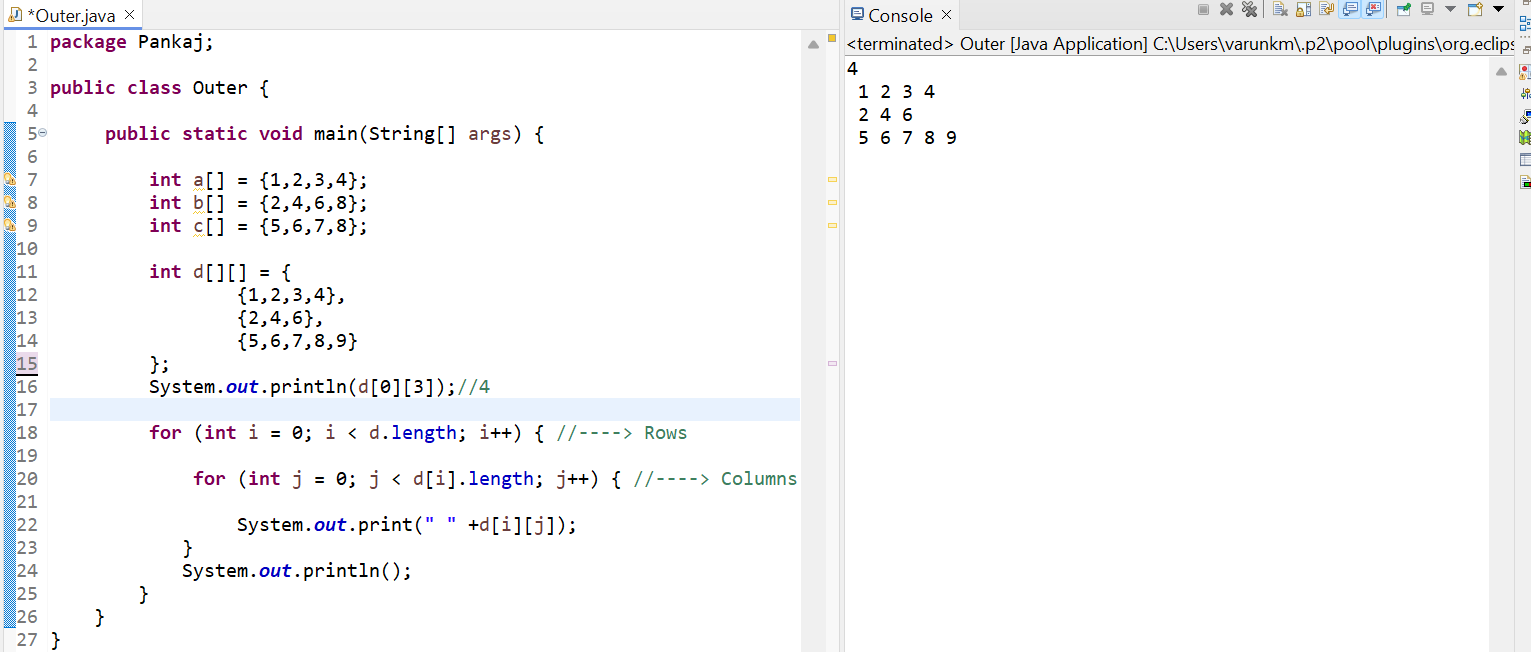
**1-Dimensional Array:** If we have multiple elements, but one Array this is called single-dimensional Array.  
**2-Dimensional Array:** If we have a multiple Array each Array, we have same elements that is the two-dimensional Array.  
**Jagged Array:** We have Array that is different sizes that Array will be called as Jagged Array

* Example of **1-Dimensional Array:**

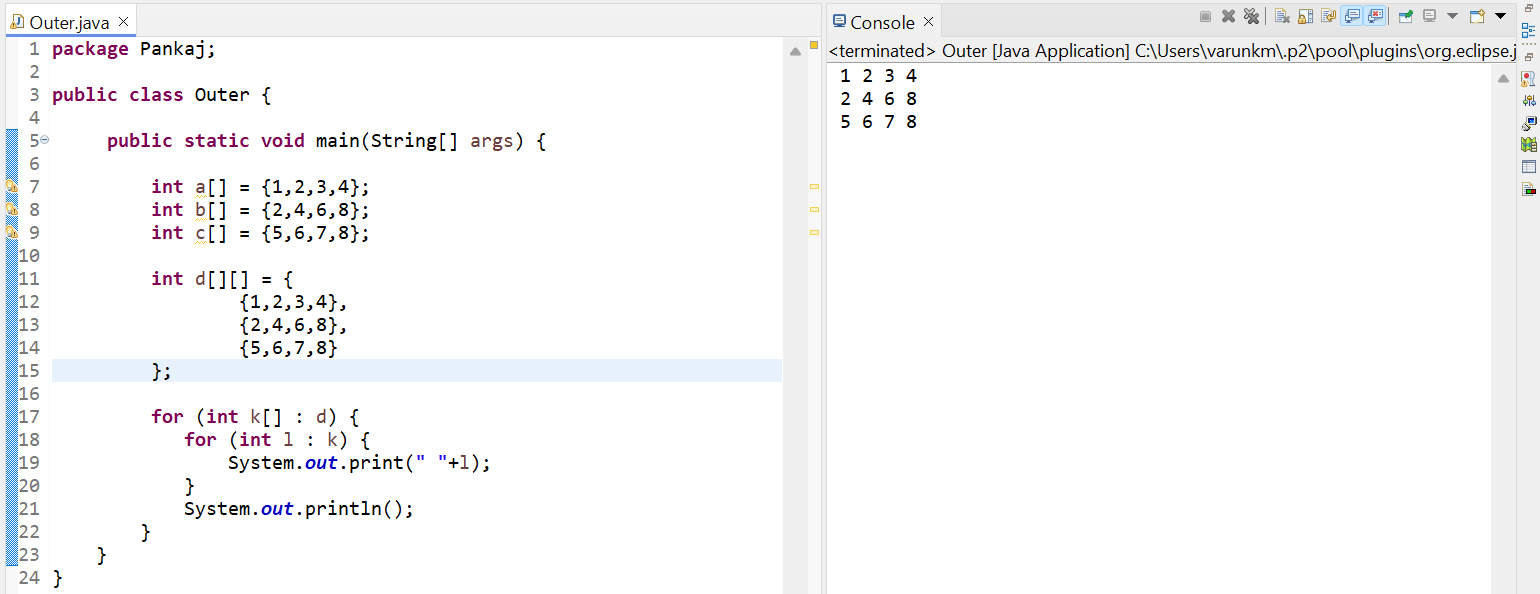
* Example of **2-Dimensional Array:** All the rows and columns are same that means 2-dimensional Array.



* Example of **Jagged Array:** The below example, this type of Array the number rows and columns are fixed are normally called jagged Array.



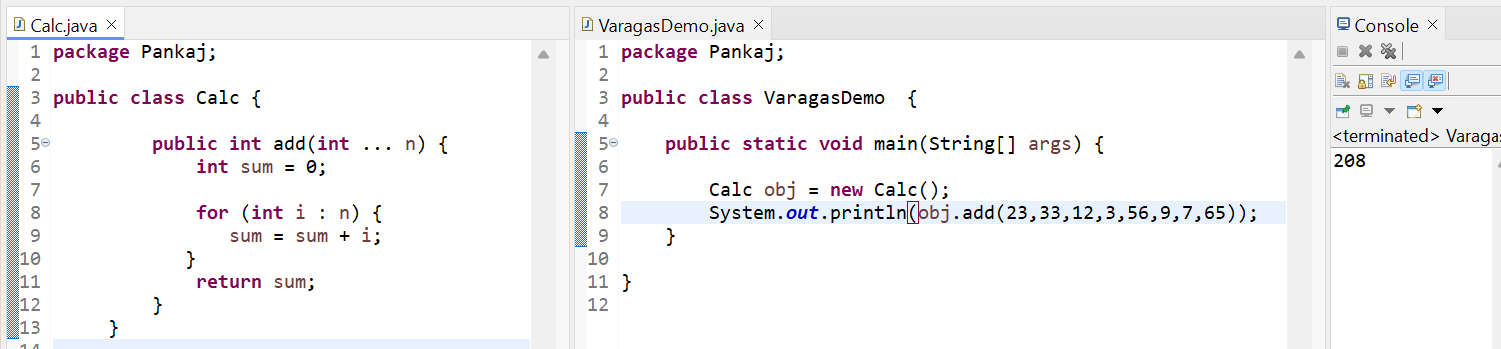
**Enhanced For loop or for each loop, with 2-dimensional Array:**



**Varagas:**  **Variable Length Arguments**

It means that we must pass the … dots in the parent method and then in the child class we create an object and pass any number of arguments this is called **Varagas (Variable Length Arguments)**.

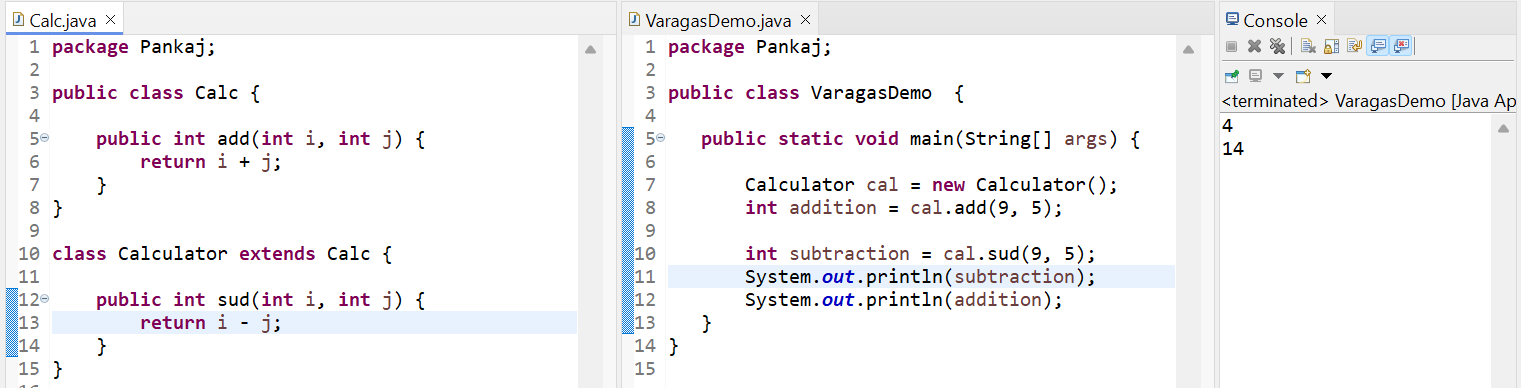
{23,33,12,3,56,9,7,65} The value is stores in an array.



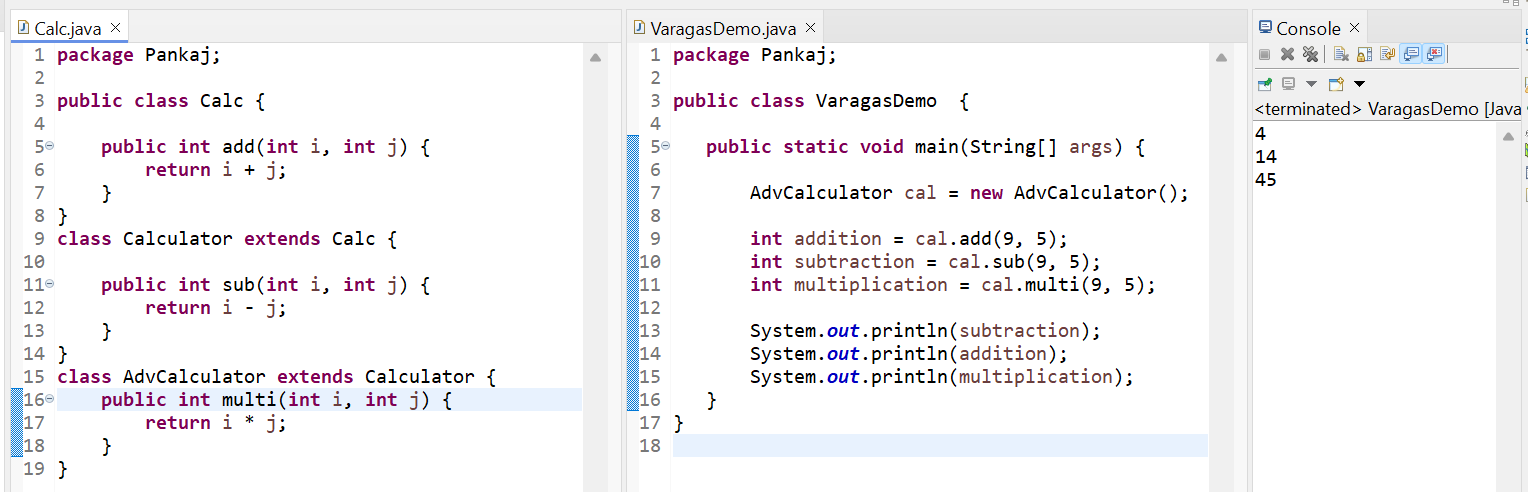
**Inheritance:**

We Inherit a member of parent to child class with the intension of reusing it. This is called as inheritance.

1. **Single level inheritance:**

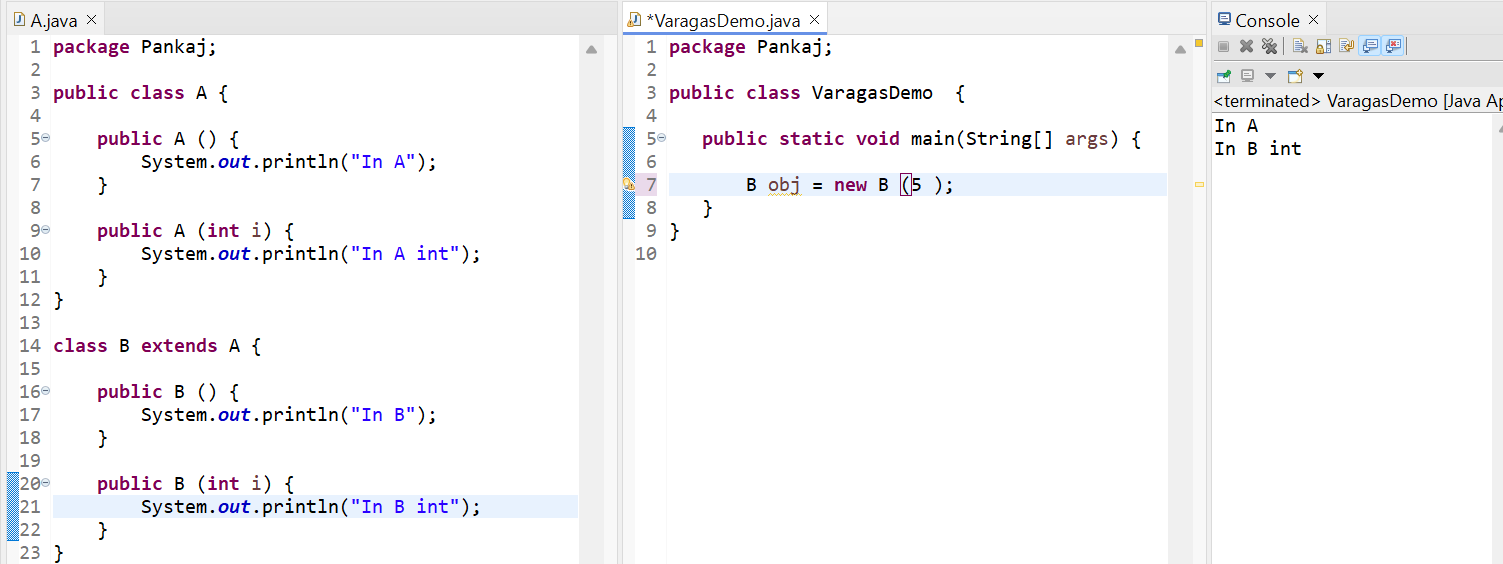
****

1. **Multi-level inheritance:**

****

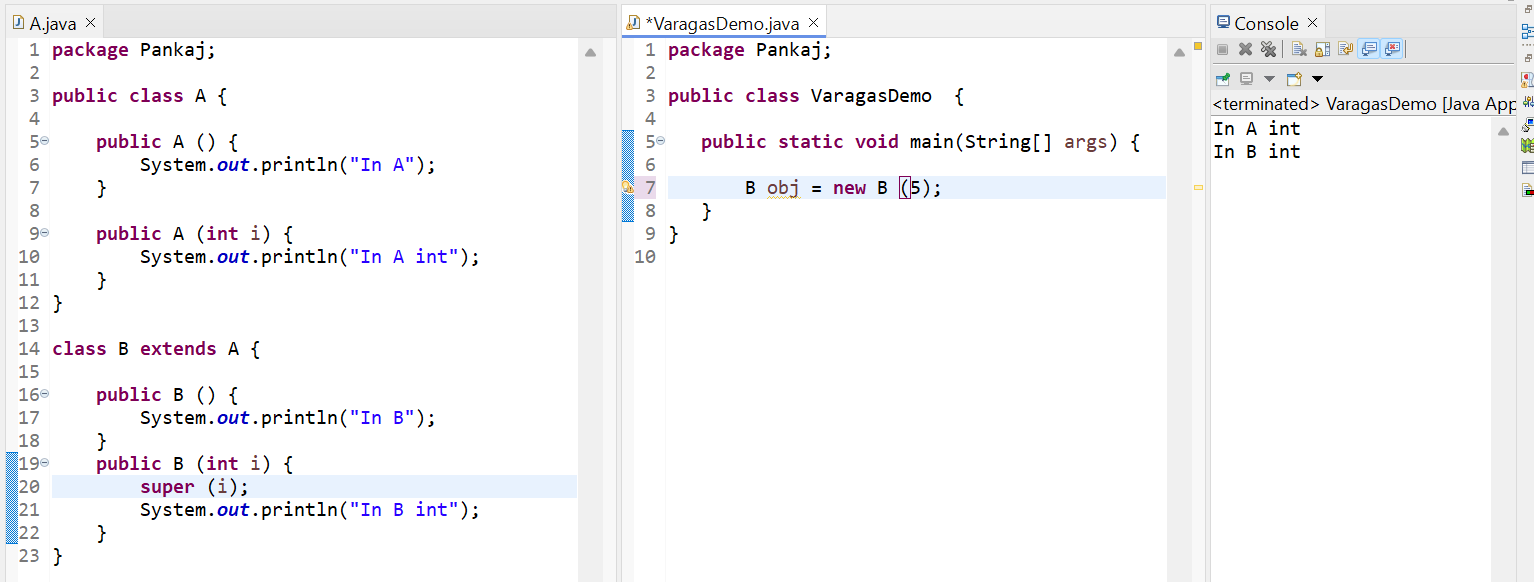
So, in java we can achieve single-level inheritance, multi-level inheritance, but how about **multiple**? But unfortunately, java doesn’t support **multiple inheritance**.

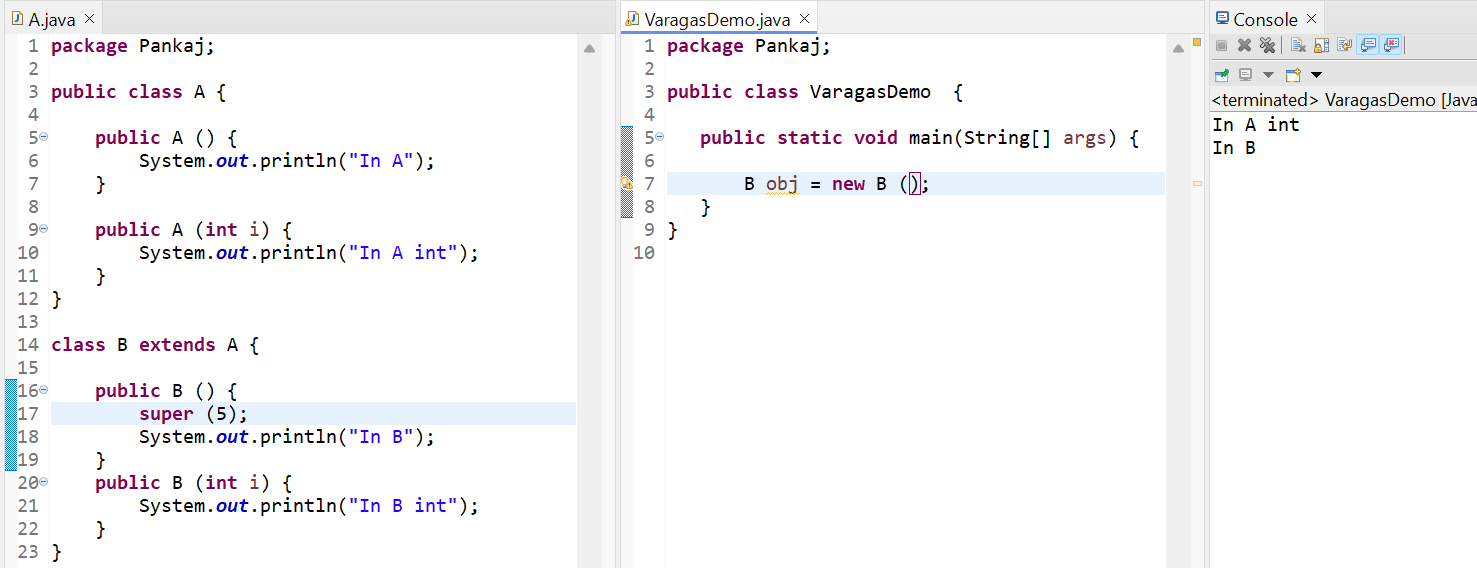
**Super Method:** Whenever we create an object of sub-class by specifying the parameter, it will call the specified constructor of sub-class, and it will call the default constructor of super class. Below example - 1.



So, what else I wanted to print the super class parameter constructor.

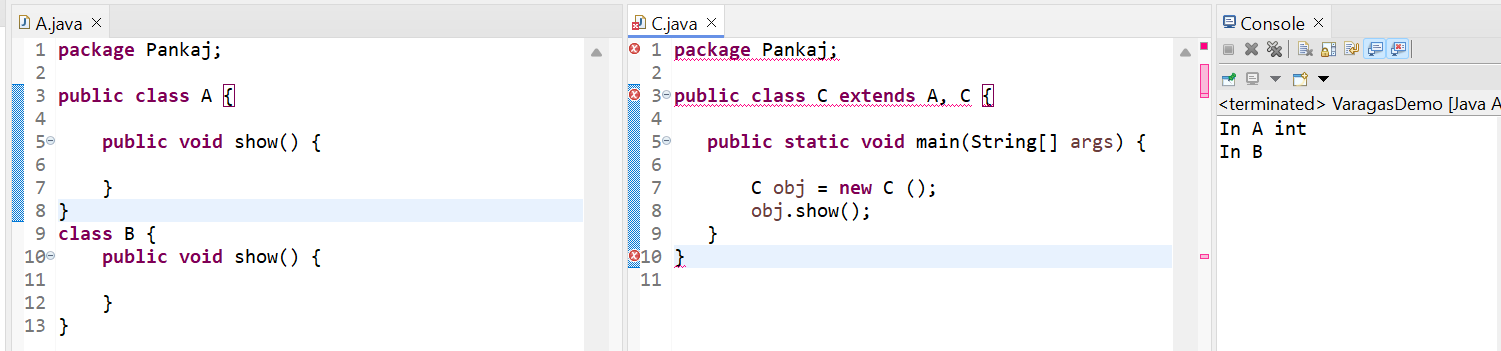
* Every constructor in sub-class will have a by default method call as super () even we don’t mention that by default it will be given by the compiler. So, that’s why calling the default constructor in the super class.
* But when we calling am parameter constructor, we have to pass the arguments in the super method, then we can call the parameter constructor in super class.





**Multiple Inheritance in Java:**

* Java doesn’t support multiple inheritance.
* Why exactly doesn’t support multiple inheritance.



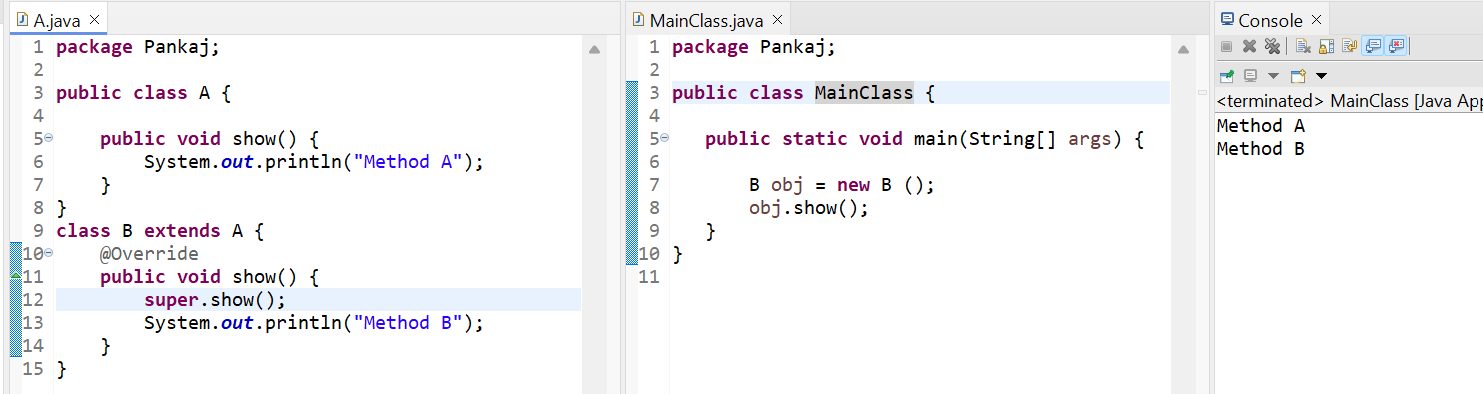
This is called an Ambiguity problem. So solve this problem what will do is just remove the multiple inheritance concept. That’s why java doesn’t support multiple inheritance using class level.

**Method Overriding:**

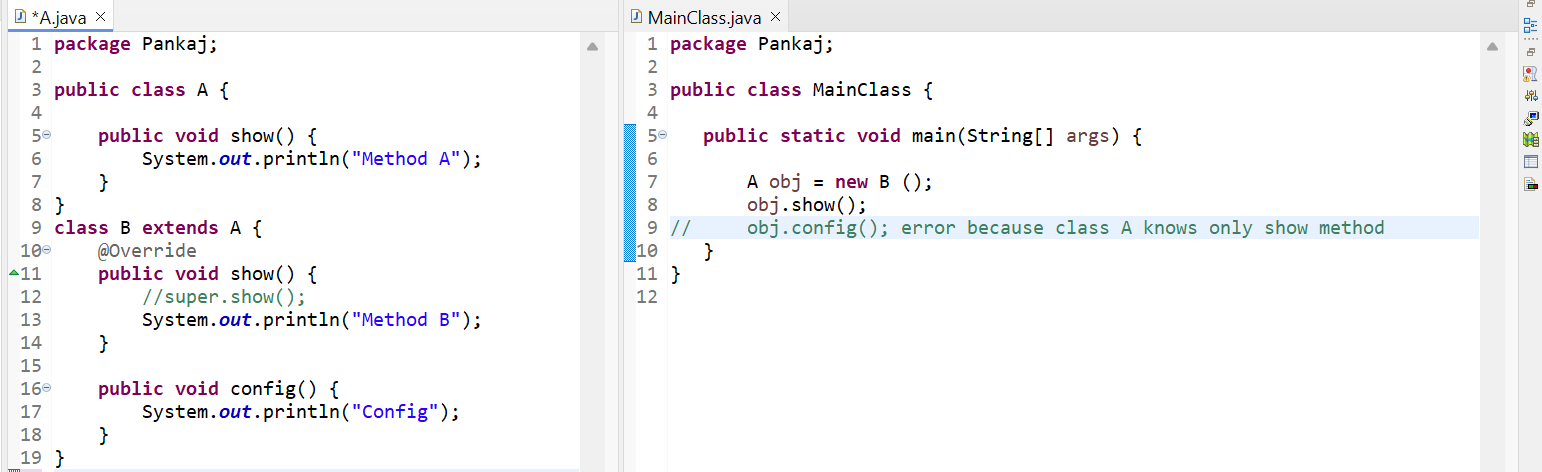
We are overriding the method from one class to another class with the same method name. The below example class B method override the method of class A this method is called method **overriding**. Child class method will override the parent class method. By mention the @Override annotation we understand this method is override to the parent method.

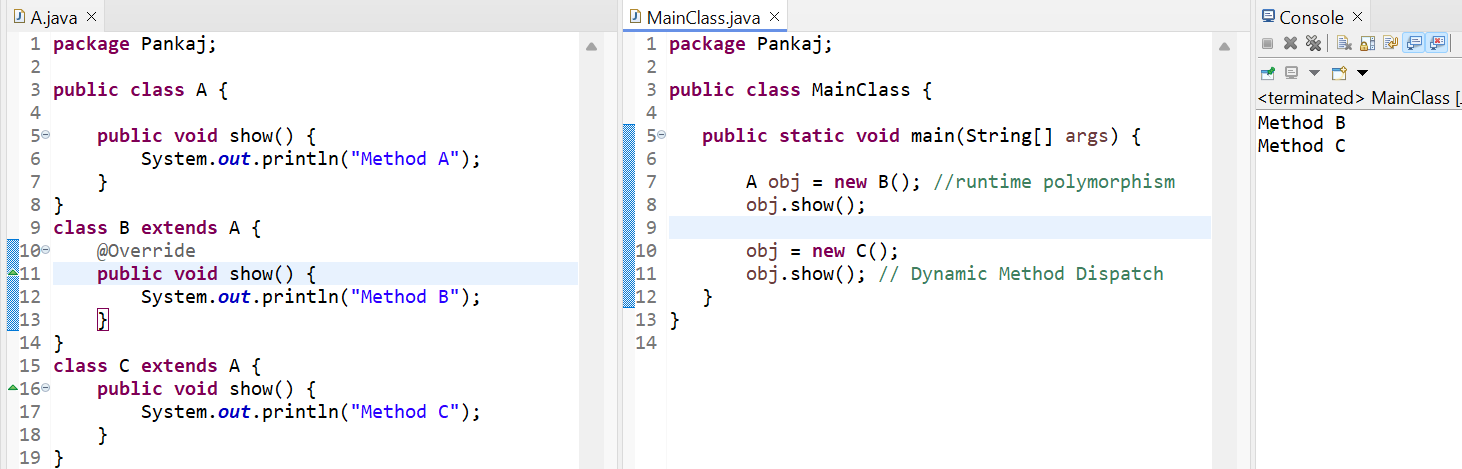


* I want to call the Method A, for that we must use the super keyword and call that method as well.



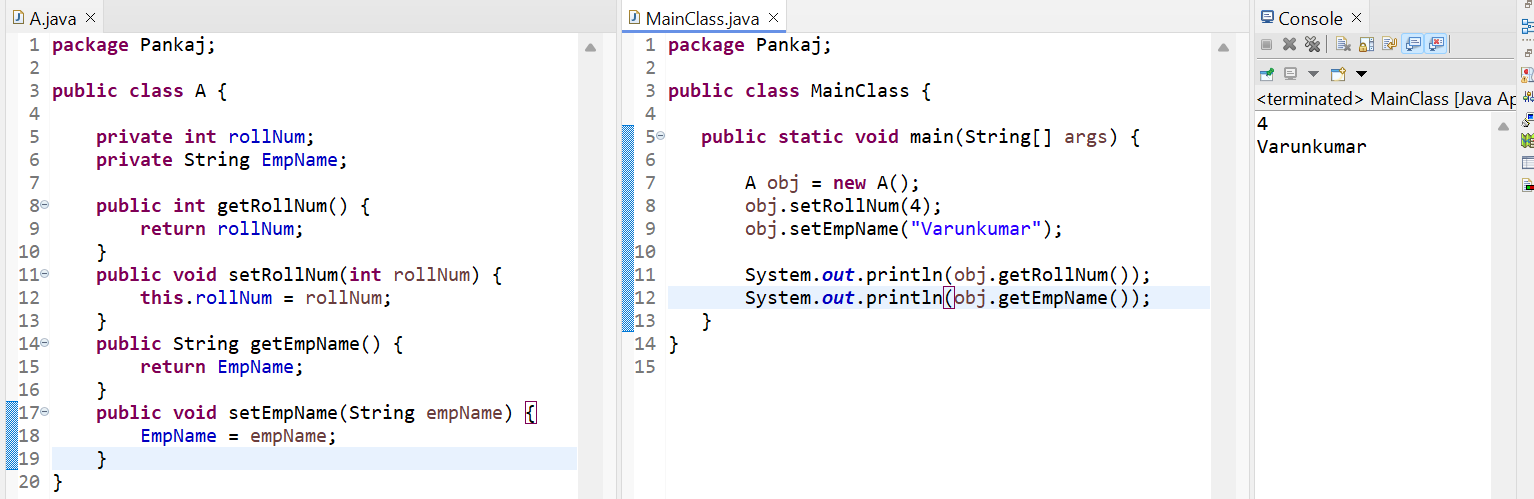
**Dynamic Method Dispatch:**





**Encapsulation:** Part – 1

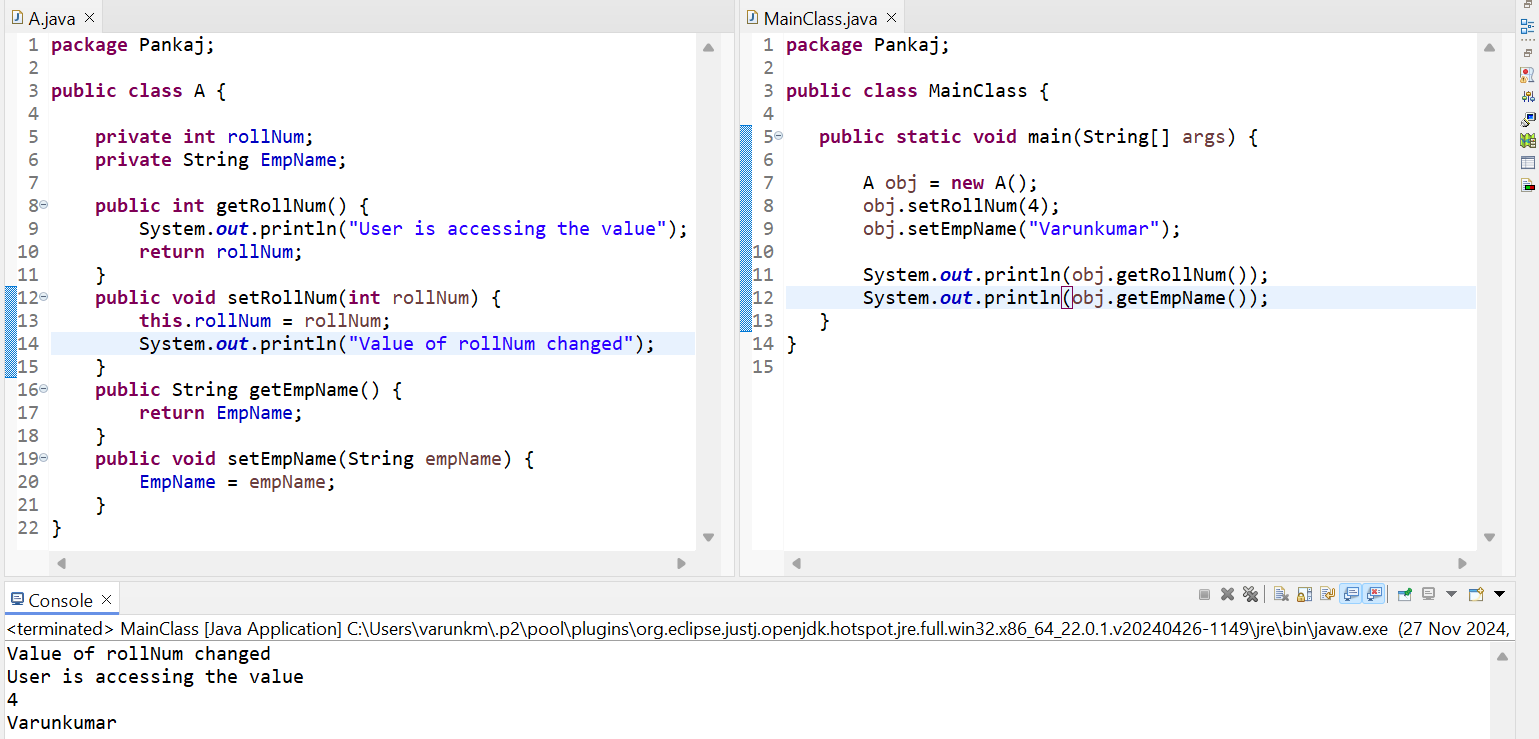
* Encapsulation means we are creating the variables as private then we have used the methods like getters and setters then access those variables. But methods which is access for the variables that is public.
* So, we are binding the data with the methods. Whenever we are bind our data with methods that concept is called **Encapsulation.**



Part – 2: Why the Encapsulation is needed?

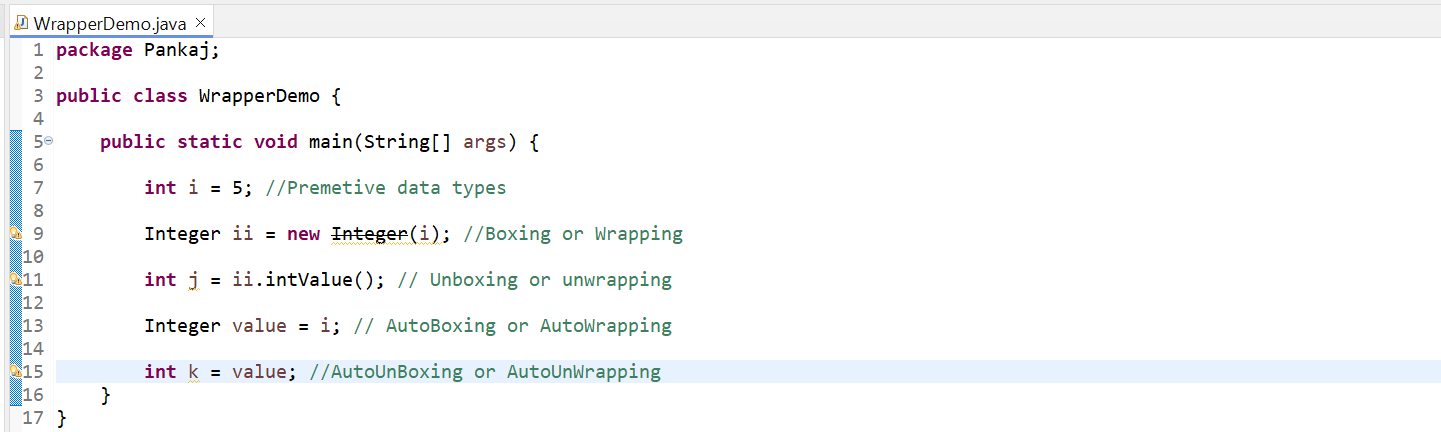
The need is we must make sure that our data is safe. Also,

* It useful for maintaining the log file
* Before fetching the rollNum, we can also check for the login of the user

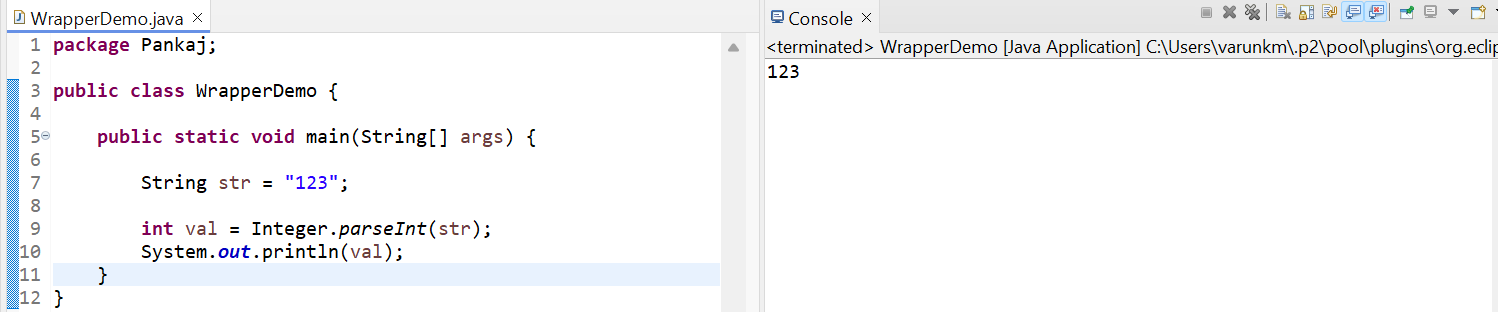


**Wrapper Class/Auto Boxing:**

* The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
* If we are using primitive in our code, it will work faster than wrapper class.
* Wrapper class is slow why we using a wrapper class?
* Because in java there are some certain frameworks which only works with wrapper classes it won’t support with primitive. Ex when we work with hibernate, Collection APIs they work with wrapper classes.

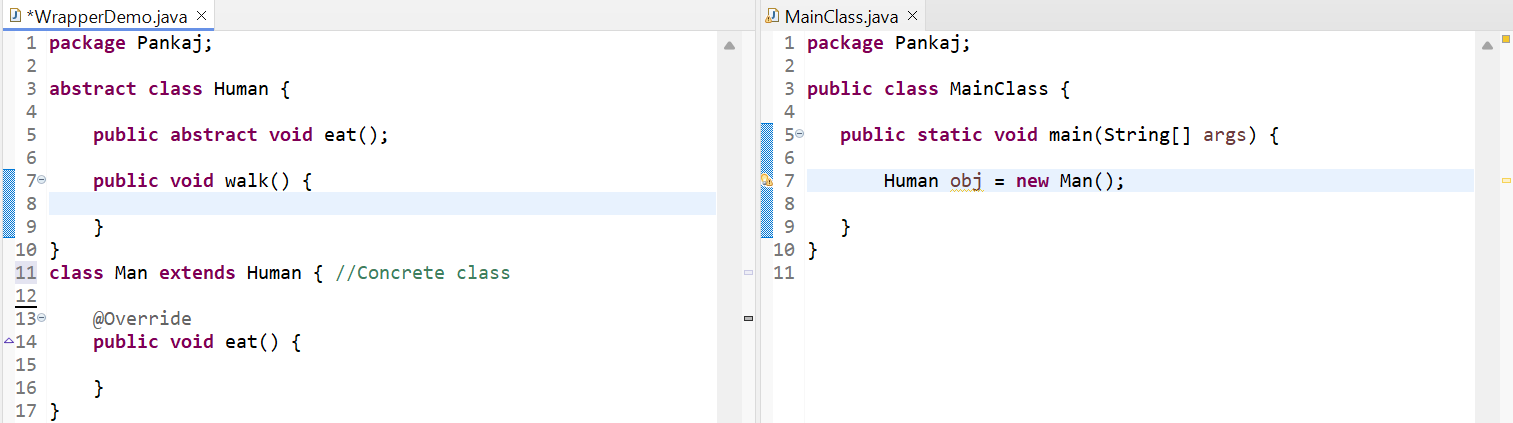


Another use of Wrapper class is to converts String into integer using the parseInt method.



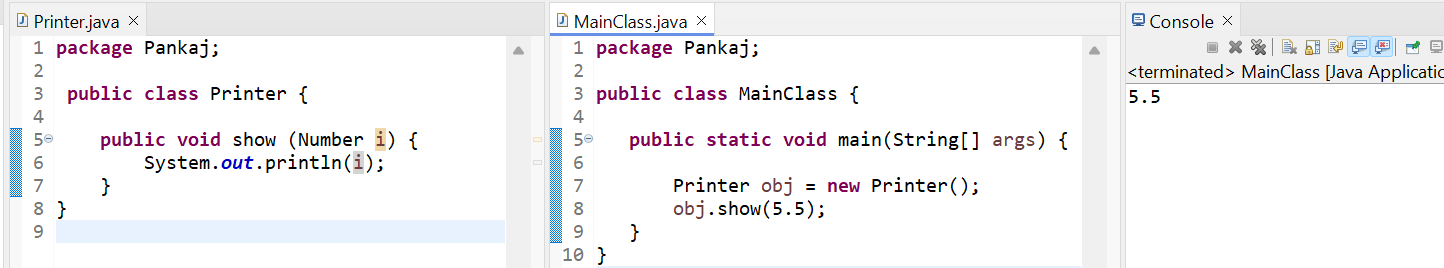
**Abstract Keyword:** We can use this keyword in class and method.

* If we make class is an abstract, we cannot create the object.
* If we make a method as an abstract our class should be also an abstract.



Why do we need an abstract class?

* We have **Number** class it is super class, using this class we can run any integer value ex Integer and as well as double value.
* Number is an abstract class, basically we don’t want to create object to anyone of abstract class and instead of wasting two methods we can use only one method which excepts all the sub-class objects.



* Integer extends Number Double extends Number
* Float extends Number Byte extends Number etc.

**Final Keyword:**

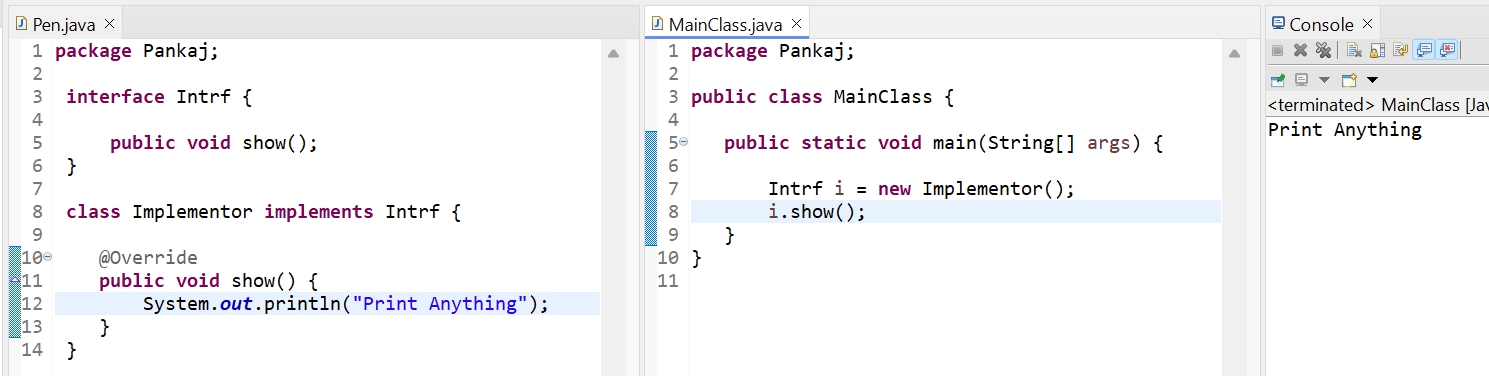
* We use Final keyword in a method, class and variables.
* If we make variable as a Final it becomes a constant, we can’t change the value in the variables, make class as a Final we can’t inherit it, make a method as a Final, can’t overriding it.
* Ex: The value of pie is 3.14 we can’t change this value at that time in the variable we are using Final keyword.
* Ex: We don’t want to our class extended in other class at that time I create a final class.
* Ex: In the method we are passing important stuffs at that time using final keyword in the method then anyone can’t override it.

**Interface:**

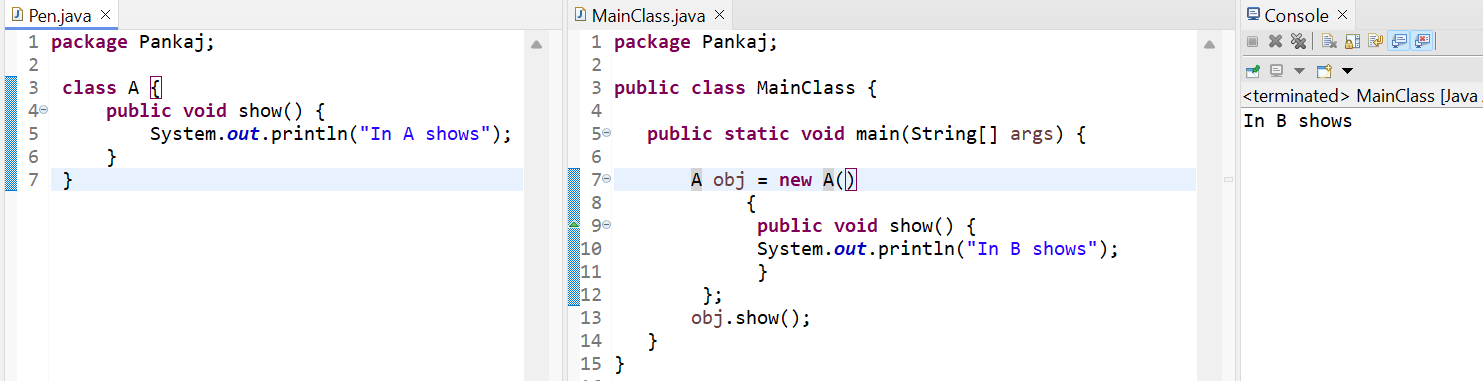
* Interface is a same as abstract class, in the abstract class we can use the normal and abstract method, but in interface only we use abstract methods means incomplete methods.
* In interface we cannot create an object, create only reference of it.
* We can achieve multiple inheritance. So, always better to use interfaces.

**More About Interface:**

* The only way to create an object in the interface to implementing the class

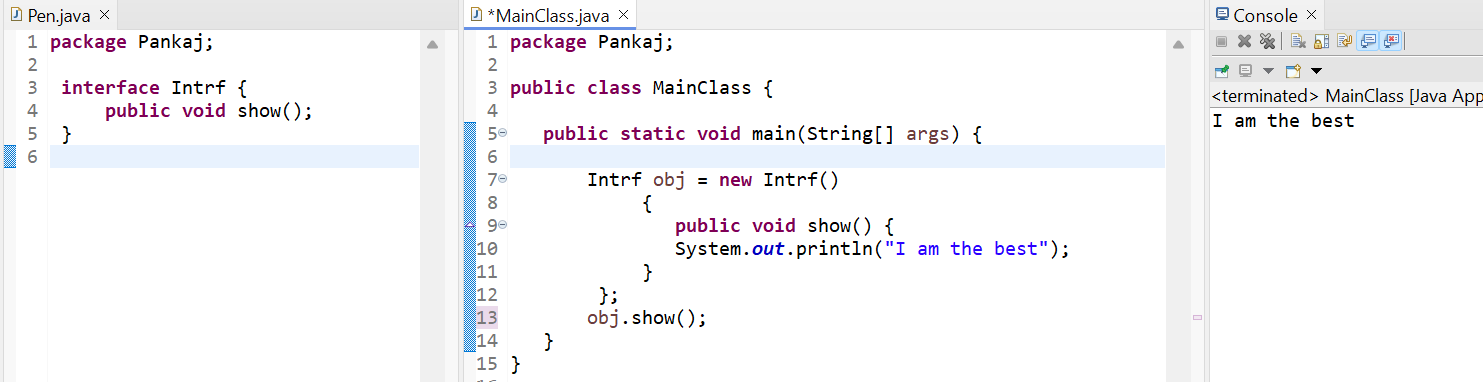
****

* **Anonymous Inner Class:** This is another way to create an object in the interface to implementing the Anonymous inner class.
* Anonymous class doesn’t have a class name.

****

**Anonymous class with Interface:**

The advantage is it will not consumption lots of memory.



**Functional Interface:**

**Types of interfaces:** Till 1.7 we are using these types of interfaces are:

1. **Normal interface** – In this interface we can create a more than one abstract method that is called normal interface.
2. **Single abstract method** – It has only one method.
3. **Marker interface** – It doesn’t have any method.

Ex: Serializable interface – It doesn't have any methods in it.

1. In java 1.8 introduced **Functional interface** – It contains only single abstract method. It contains any number of static and default methods.

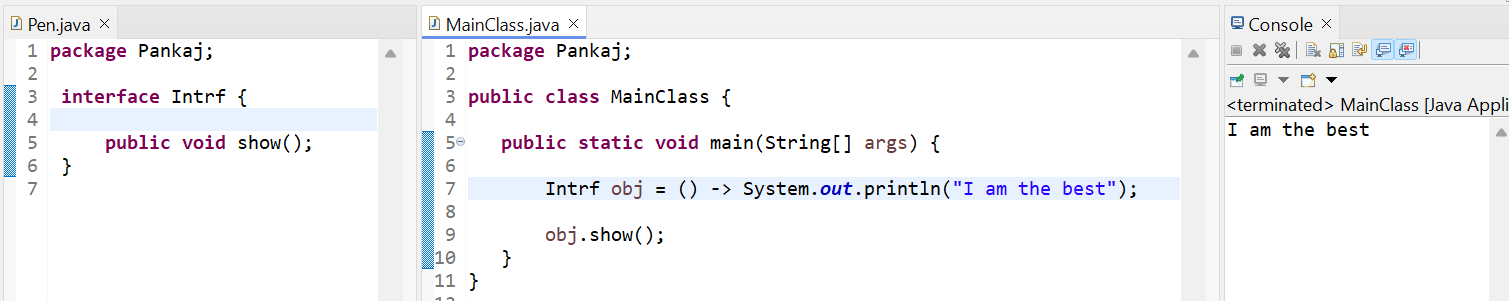
Whenever we are using Functional interface, we must invoke a **Lambda Expression.**

So, lambda expression has come from the language called **Scala**. Java adopted scala language. So, now in java 1.8 we have all the features which were there in scala.

* What is Scala? Why we need to learn it.   
  There is multiple reason we need to learn scala. Basically, Scala was created by person called **Martin Odersky**. The scala programming language is promoted by **Typesafe, Inc** company. So, SCALA stands for scalable language even all languages are scalable like java, phyton, C++, C# etc.
* Why Scala was programming language?  
  Scala is programming language which is build the main moto of building the language scala was make it scalable. This is the main important feature of in scala.
* Lightbend offers tools, support, and platforms for developing reactive applications using Scala, Akka, and Play Framework. They play a central role in the Scala ecosystem, ensuring its growth and adoption in the industry.
* Scala is a functional programming language. We must focus on functions, Scala language is faster compared to java, phyton, .Net etc.
* **Fun Note** – We are a java programmer, and we want to spend in a more time with our family and friends we should learn scala.

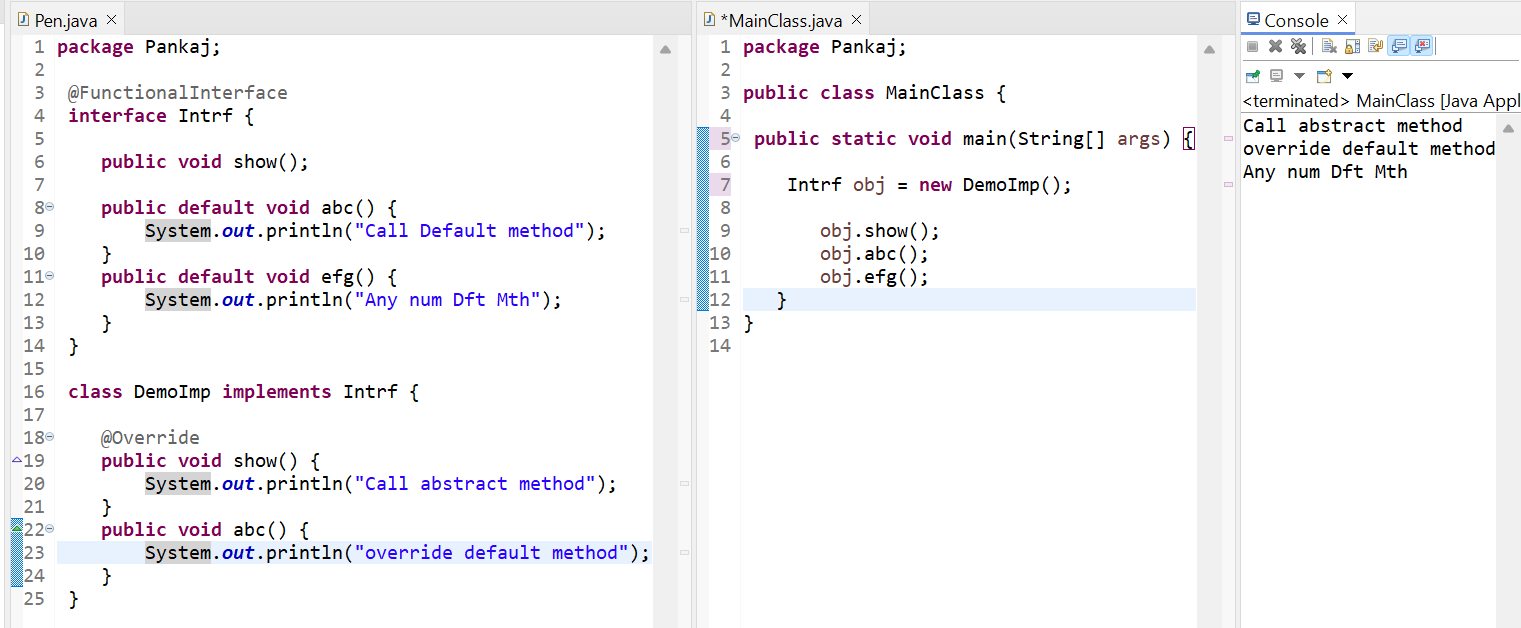
**Advantage:**

* We have task in java that task is on 20 lines but in scala that code is in 5-6 lines.
* When we write a develop a java application we have a lot of lines of code. If we have one bug it is difficult to detect the bug . But in Scala our code looks beautiful. Even if we have a bug we can removed easily.
* Scala is giving upgrade version of java. Both java and Scala are runs in JVM. When we are creating a java-based application then we can use scala code over there. As well as in scala code we can write java code.
* For the **Apache Spark** framework using **Scala** language.



**Default Method in Interface:**

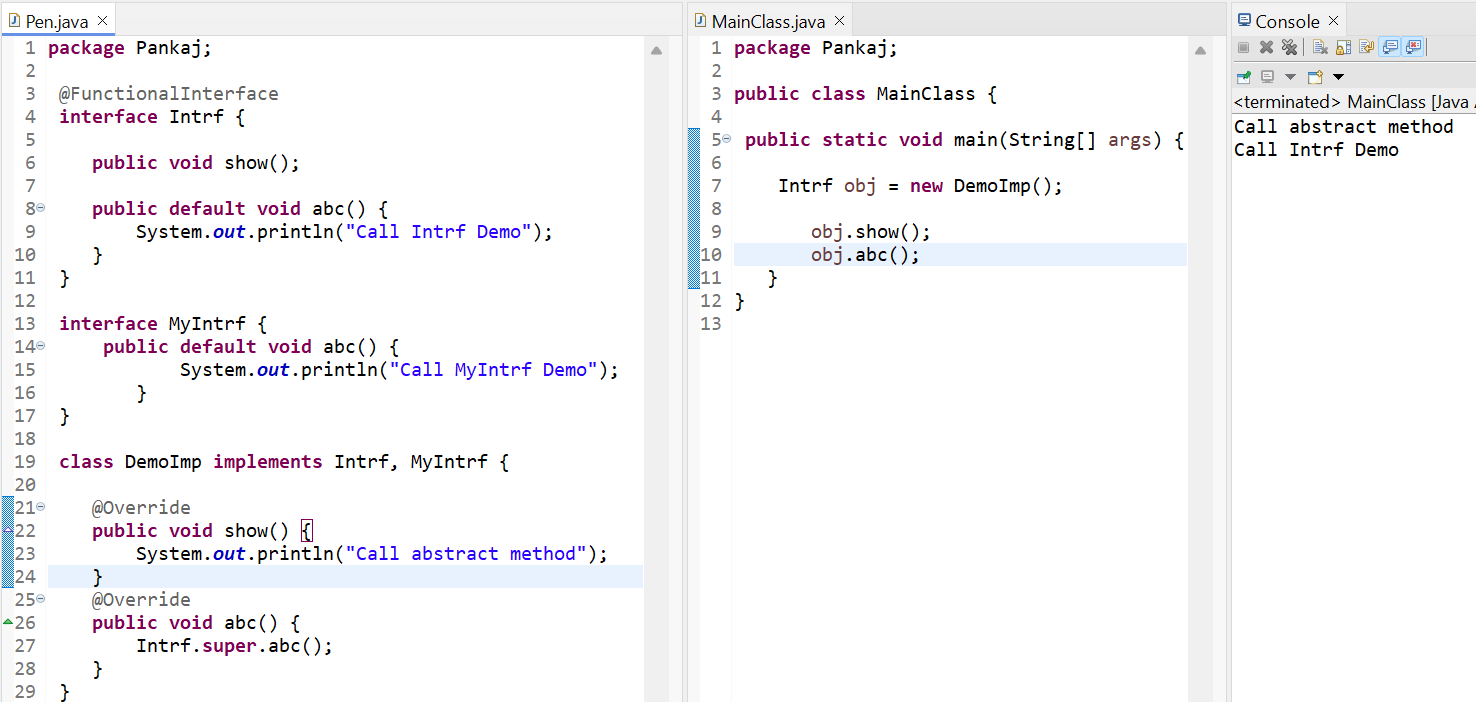
This method is introduced in java 1.8 we can use the default keyword; we can create any number of default methods inside the interface. That why this method was introduced in java 1.8   
We can also override the default methods from one class to another class. But here we have create an object to call this method.



**Multiple inheritance issue with interface:**



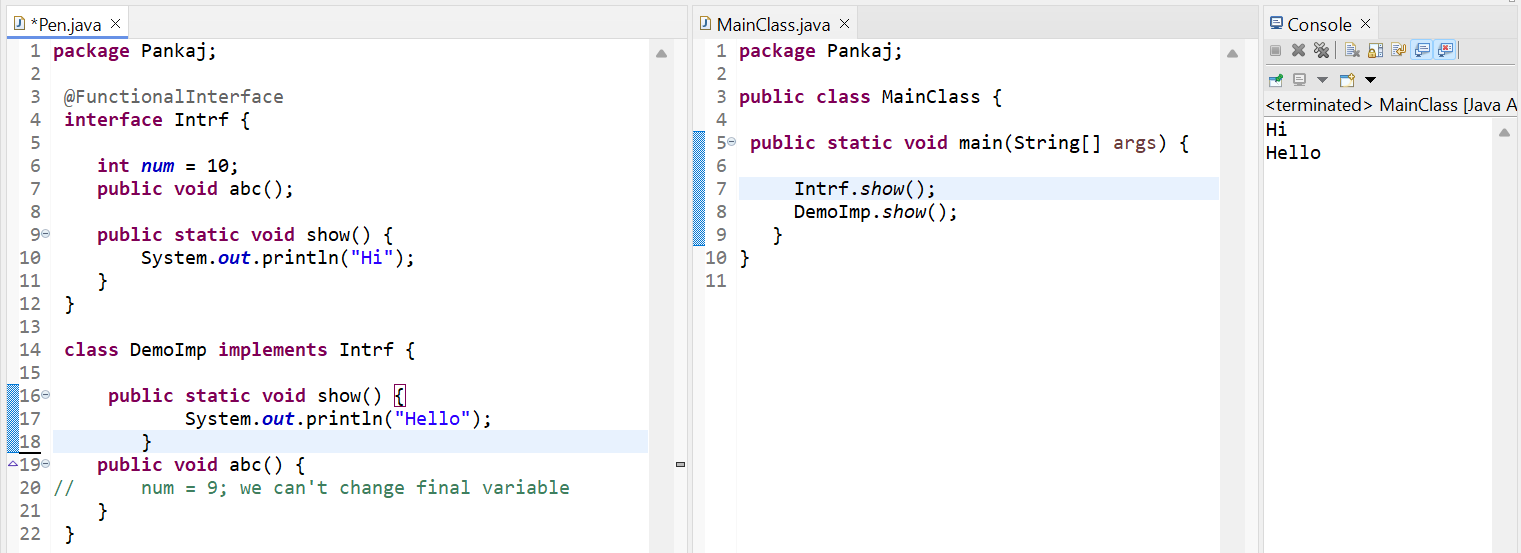
This is the issue with the interface we solve this ambiguity issue we create a method inside the implementation class.



**Static Method in Interface:**

In java 1.8 also provided one more feature called **static method** using inside the interface**.  
Ex:** When we create a static method inside the interface. Without creating an object, we can call the static method.

* In interface when we create a variable by default it becomes a constant. The final keyword automatically created in the variable.



**Packages:**

Domain name is **google.com**Package name should be **com.google**Sub package we create a package like this **com.google.entity** etc….

com. Google.entity \* This gives all the entity class to another class.

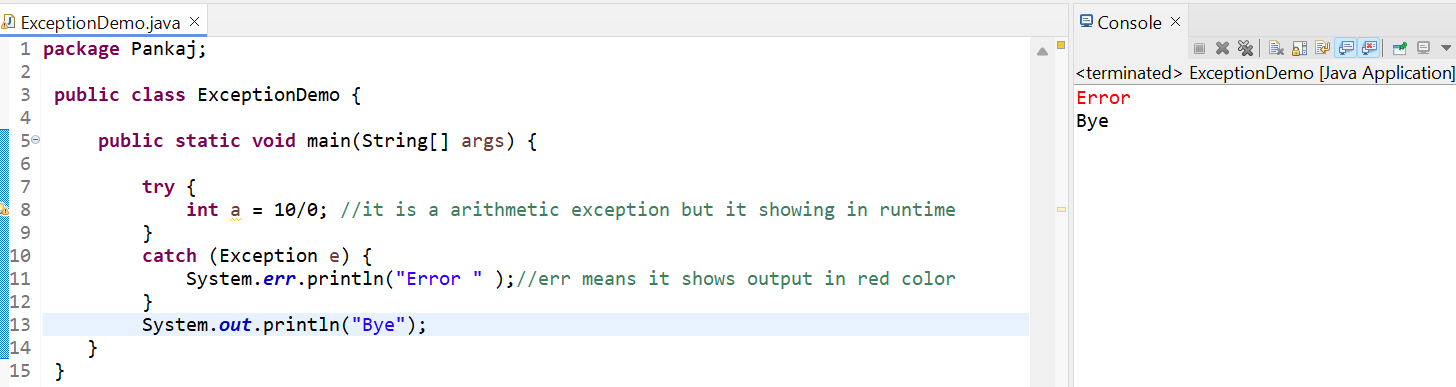
**Exception Handling Try Catch and Finally:**

**Unchecked Exceptions:**

1. **Arithmetic Exception:** Basically, Arithmetic exception is a sub-class of Runtime exception. Runtime time exception is a sub-class of Exception.

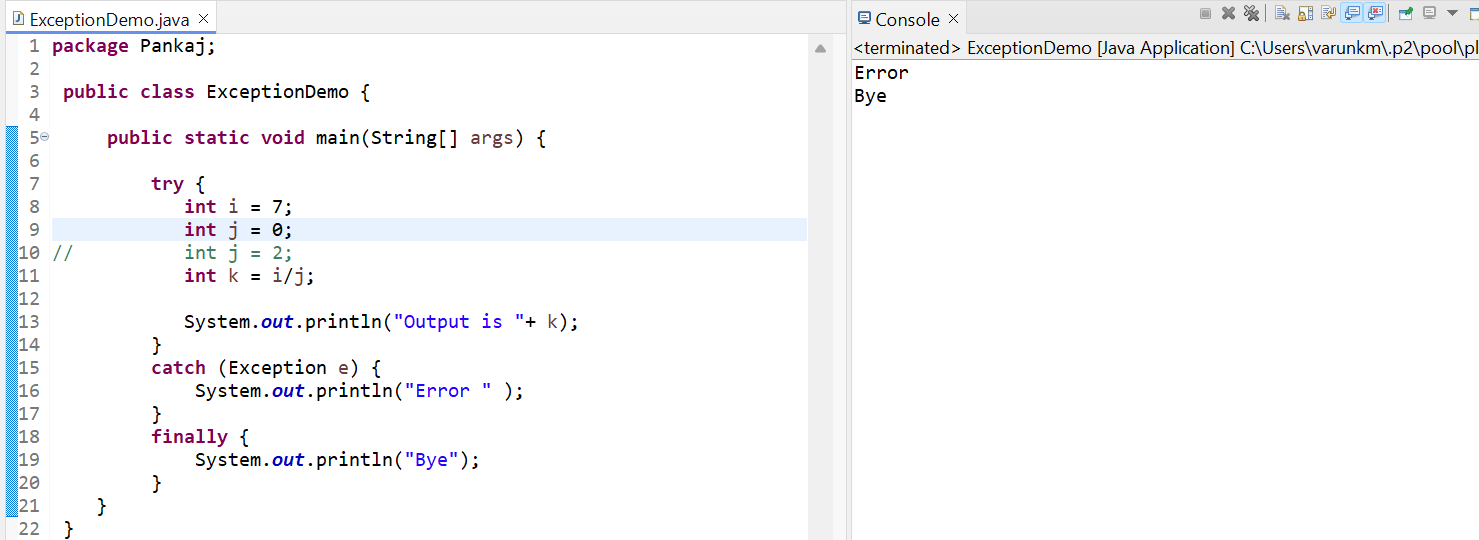
Try is a block where we must put all our critical statement. It throws the object of Exception.  
Ex: try {  
 int a = 10/0;  
 }

If it is throwing an exception, it’s our responsibility to catch it.



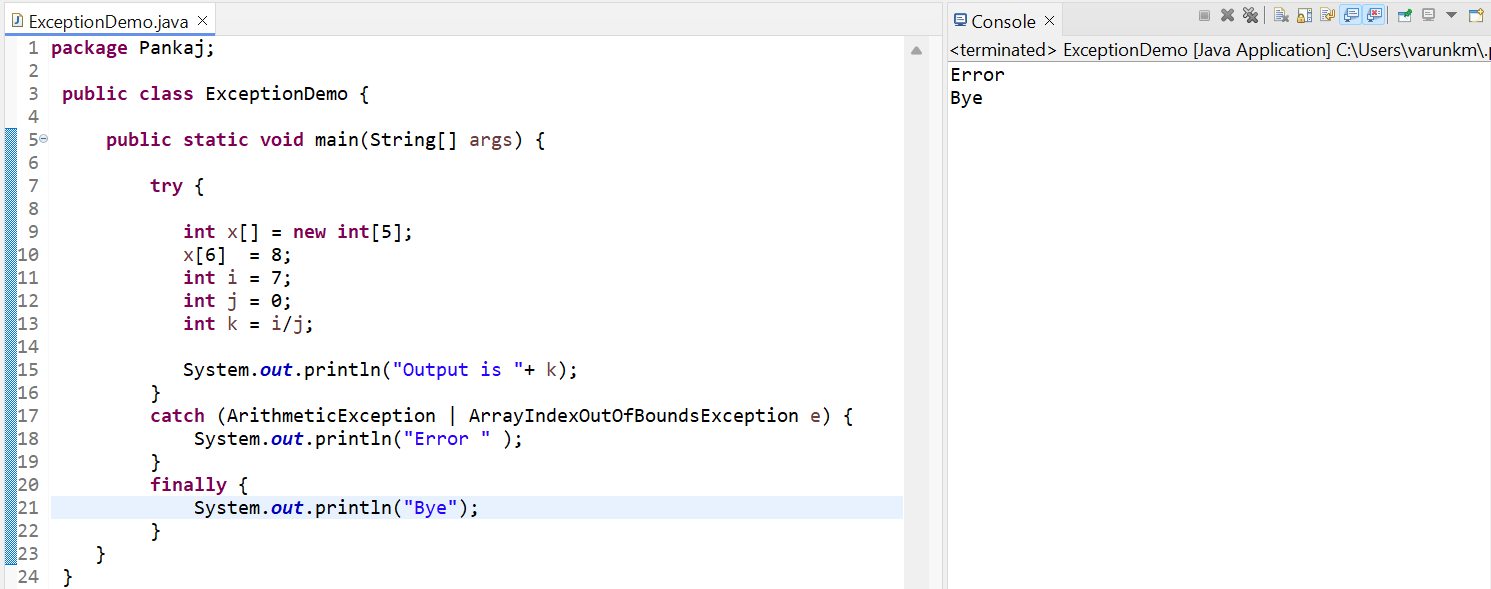
****

**Finally Block:** Finally, is a block which will executed even we are getting the exception or not getting the exception.

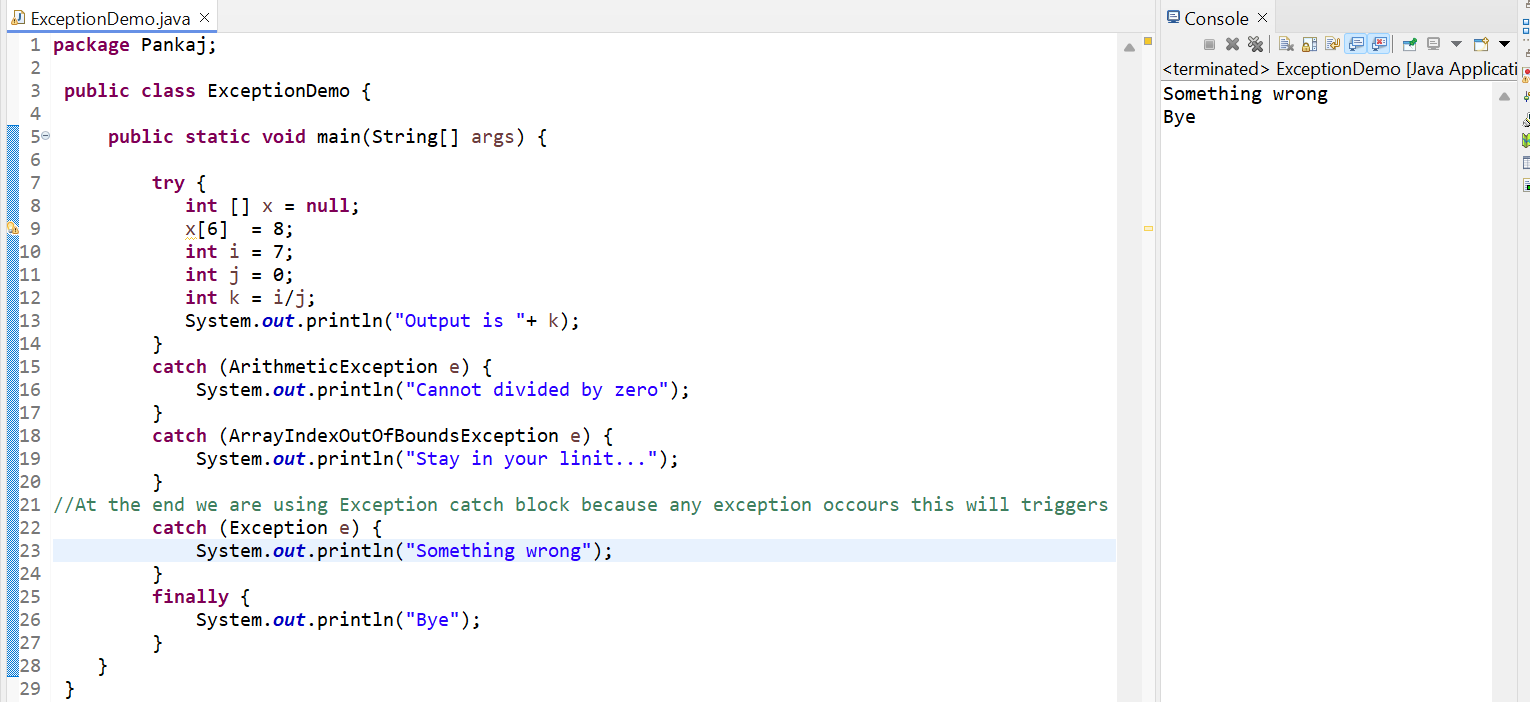
****

**Exception Handling Multiple Catch Blocks:**

We can write multiple catch and, we can write multiple exception in one catch block. This introduced in java 1.7 version. But we must use primary exceptions at the first then parent exception. If we parent exception in first it shows an error.



We want to give a message to the users for that we are using multiple blocks in that we are adding a message to the users.

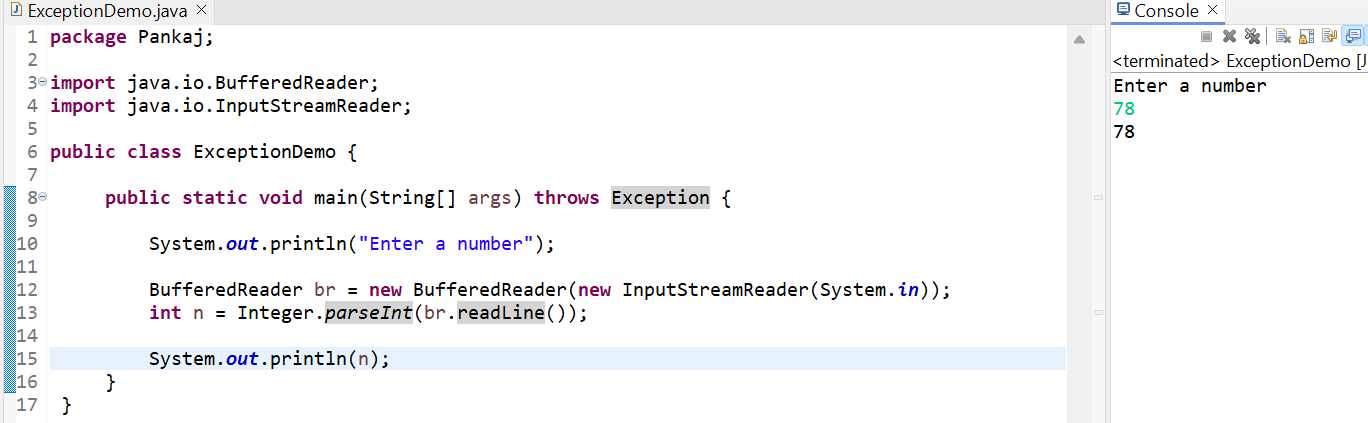


**Exception Handling Checked Exception | Finally Block:**

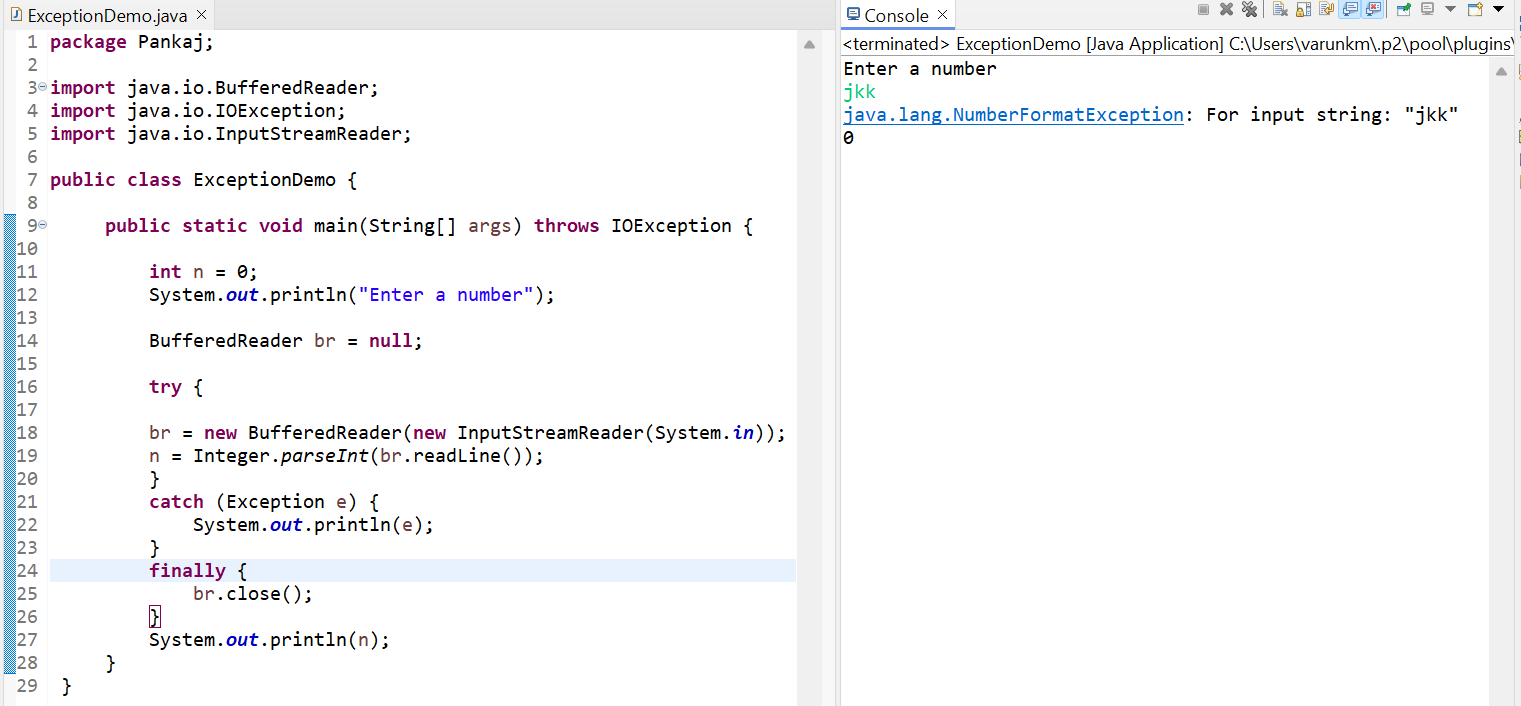
We have two types of exception **Checked** and **Unchecked,** So, when the compiler knows about the exception that is called checked exception and compiler don’t prompt about the exception that is unchecked exception.

**I O Exception** is not a part of runtime exception. So, only Runtime exceptions sub-classes are unchecked remaining all the exceptions are checked exception.

Either we can handle the exception using try catch block or throws exception.

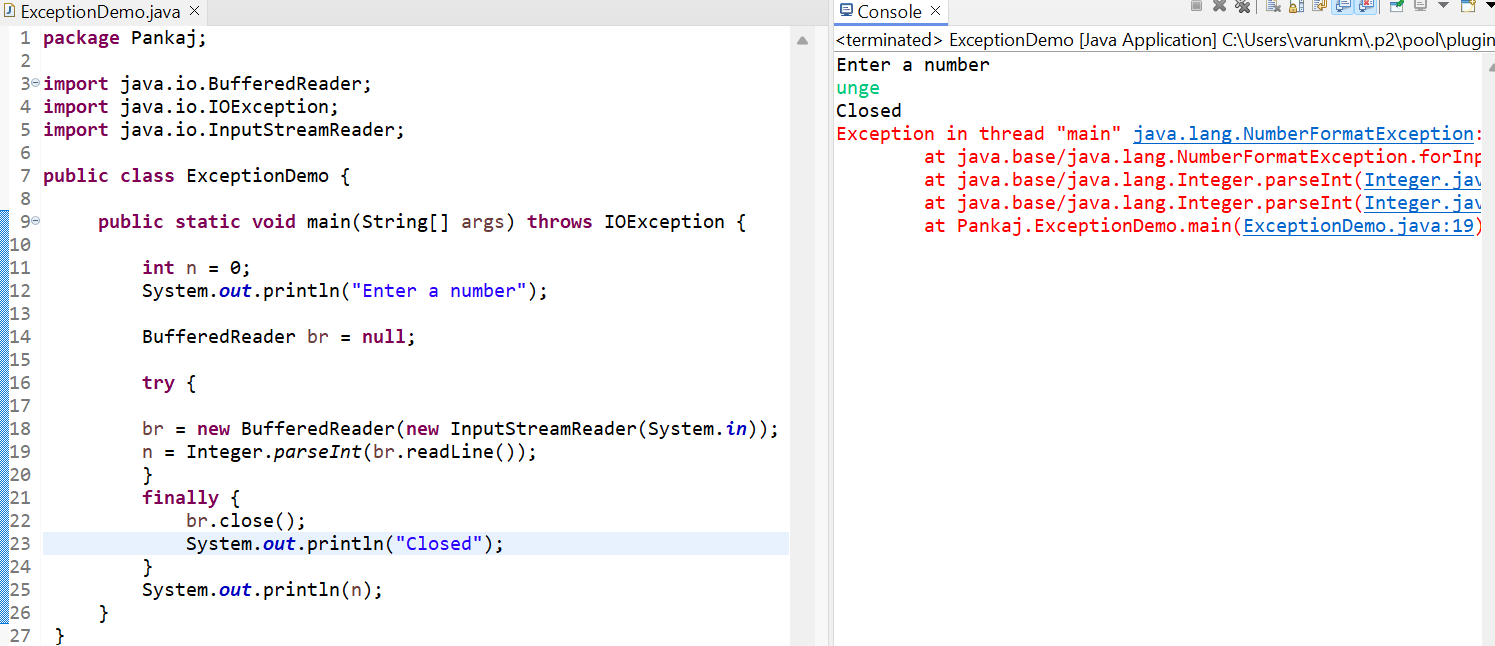


Sometimes don’t use throws exception we have to use try catch block. Also most importantly using the finally block to close the resources. The below example shows handled the checked exception using buffer reader and close the resources using finally block.



**Exception Handling Try with Resource:**

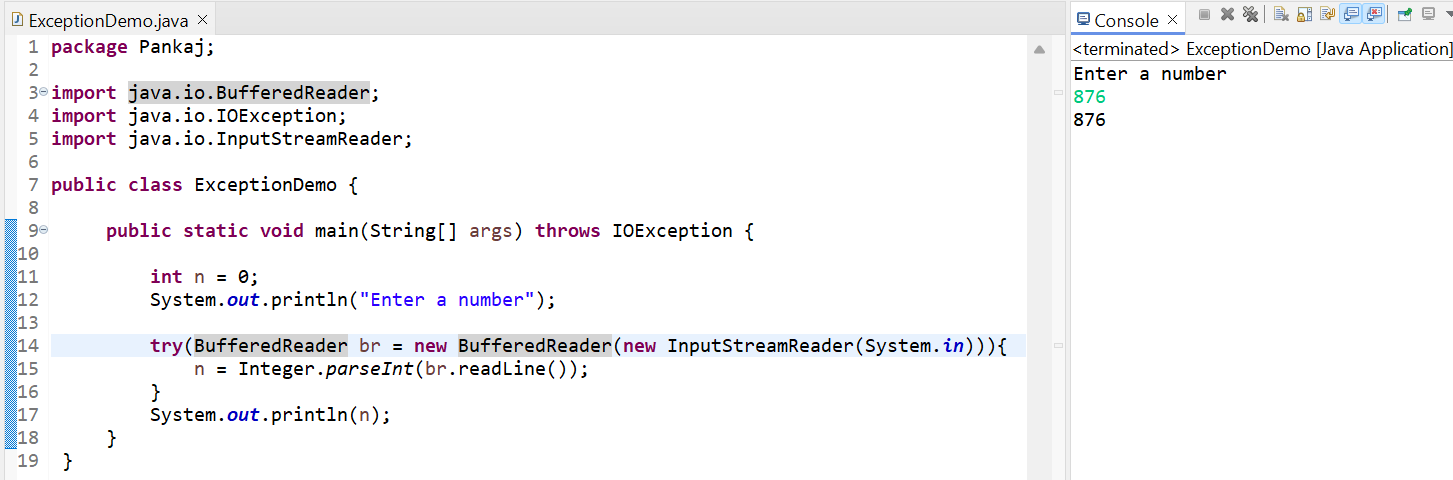
We can also handle the exception with using try and finally block. The below example is to close the resource.



Java 1.7 not compulsory to write the finally block, instead of using finally block syntax use inside the try using parentheses and add our object there.

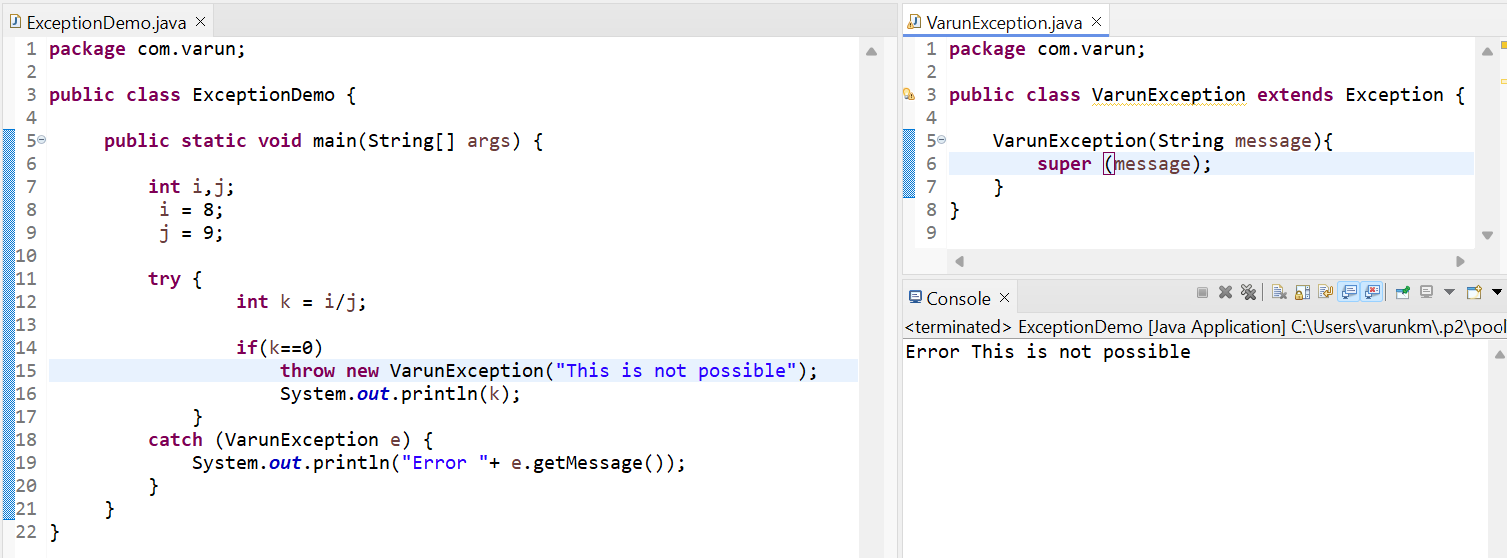
If we do like this as soon as our object goes out of scope it will close the resources automatically. Our object done with work it will close the resources even we get the exception here it will close automatically.

So, we can use try block without using catch and finally block. Using this below example syntax this is called as **try with resource**.



**Exception Handling | User Defined or Throw Exception:**

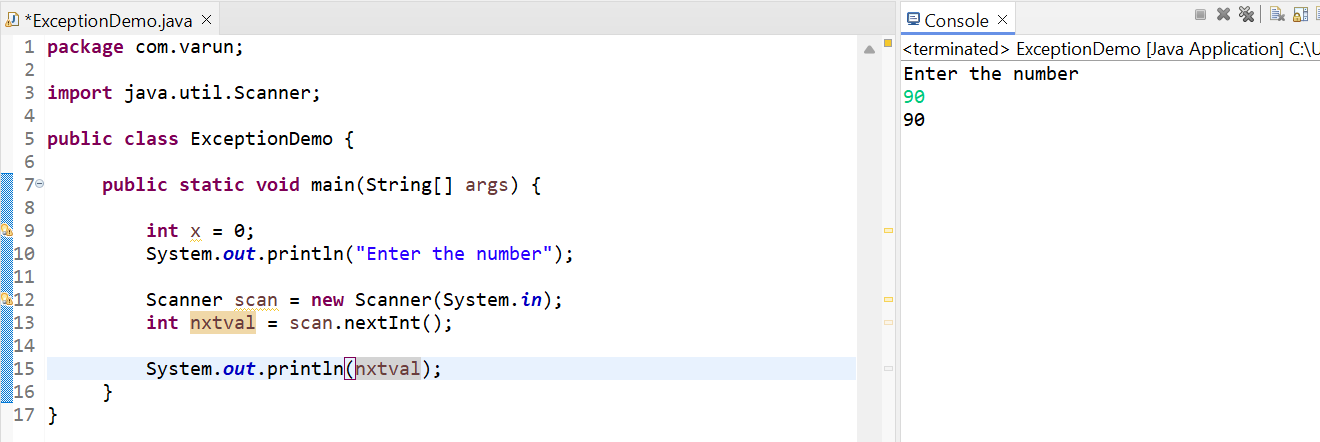
Using throw keyword, we are using customised exception whenever we want.



**User Input Using Scanner:**

nextInt() – This is a method inside a scanner this will fetch the integer value, we have not only this method so many method is there in this scanner class we have to use based on our requirement.

* The advantage here is we don’t have to handle the exception.
* We don’t have to convert String into integer value.

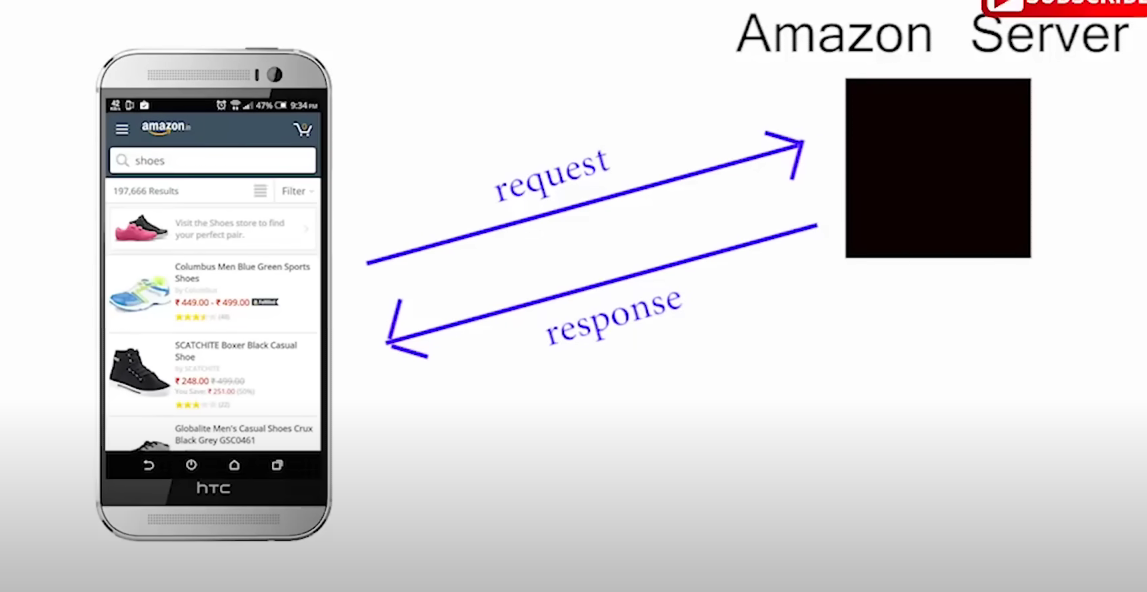


**Multi-Threading in java Theory:**

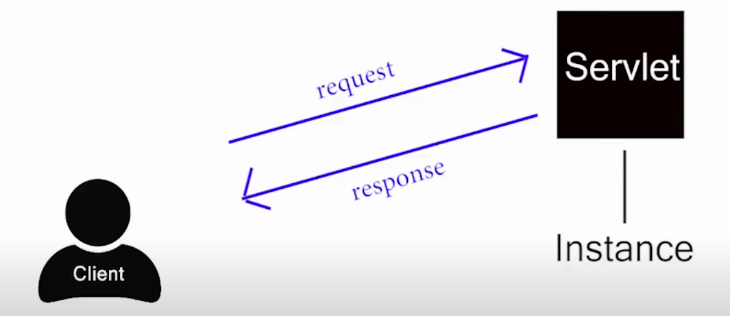
Threads – It is a unit of a process. Ex: if we take the process and break into the small pieces.

* Why we use threads?
* Now we have an octa core CPU -> It means we have 8 cores instead of having one core CPU.

1. When we create any java application, we are using only one thread that is called main method. Ex: we have 2000 elements of array {2,3,4 ….} we want to multiply into 2x {4,6,8 ….} so complete this task it takes at least 8 sec so in this concept we are using thread it reduces the time, and it will print 2 secs that is the power of threads.
2. Asynctask in Android:

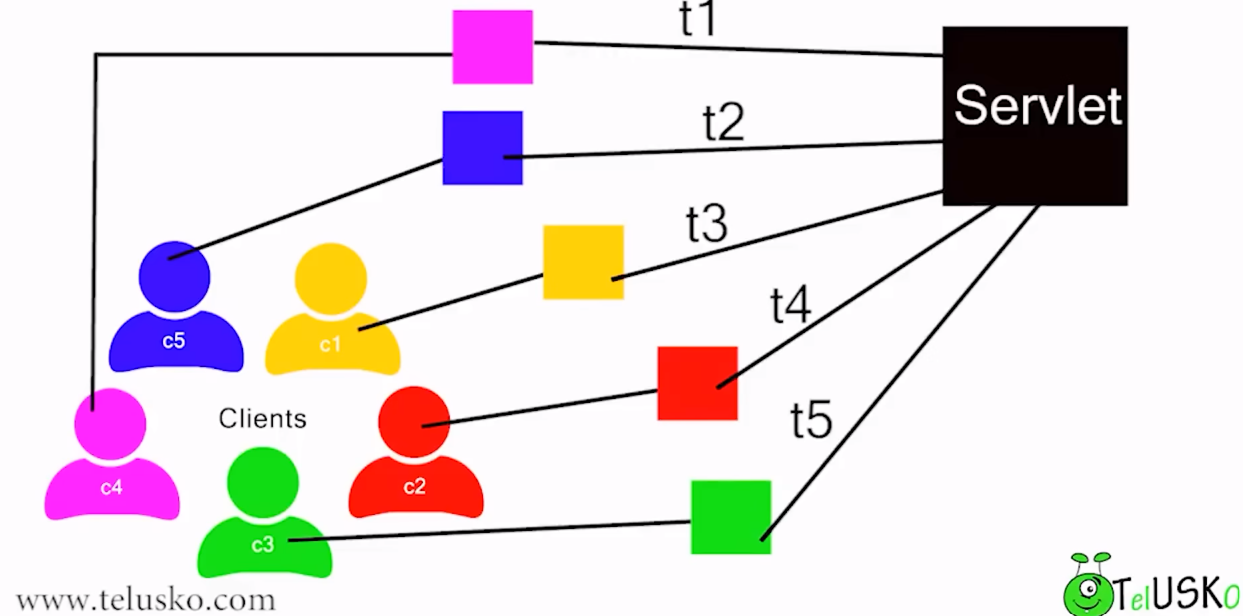


In this example main thread will not send a request we have to create two threads here on thread is interact with the user and another thread is going for the server request.

1. Web Application:

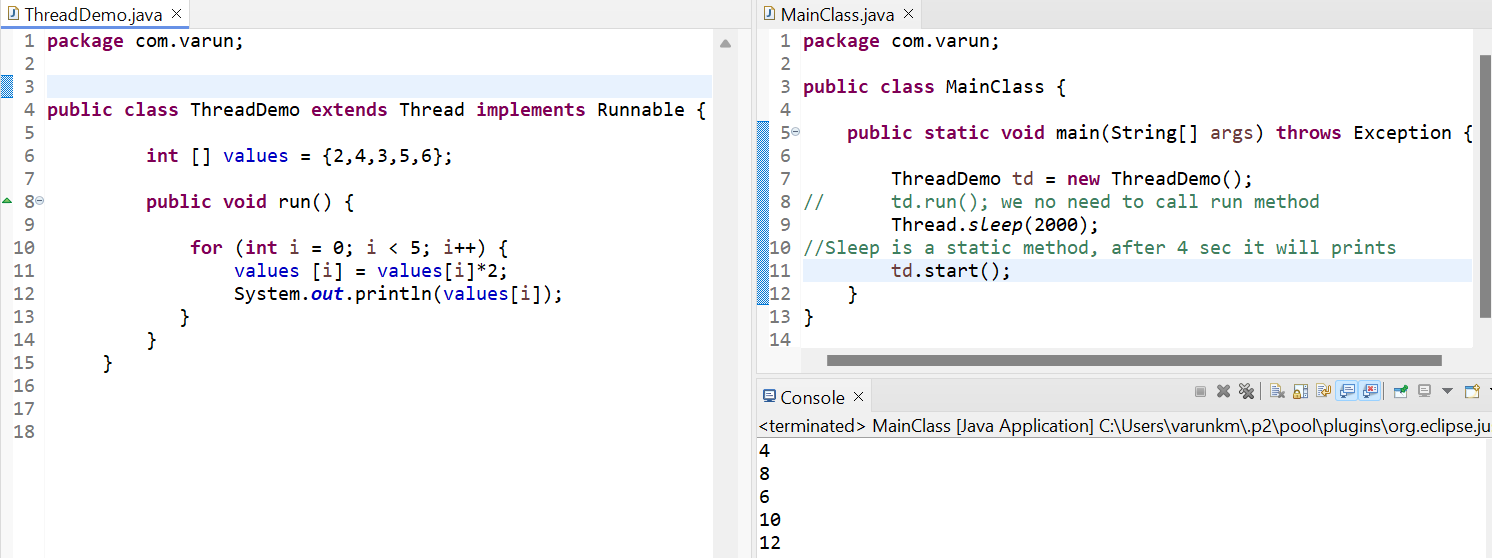
Whenever we send a request to client to server using servlet, servlet will process our request and give the response.

Let ‘s we have 5 users who are wants access to the same servlet the servlet creates 5 different threads and provide to clients.



1. Gaming: Threads are very important for the game developers, we developer, android developer etc.

How to create a Thread – We creating a Thread using extends the class and implements the Runnable interface these are the two ways to create a thread.

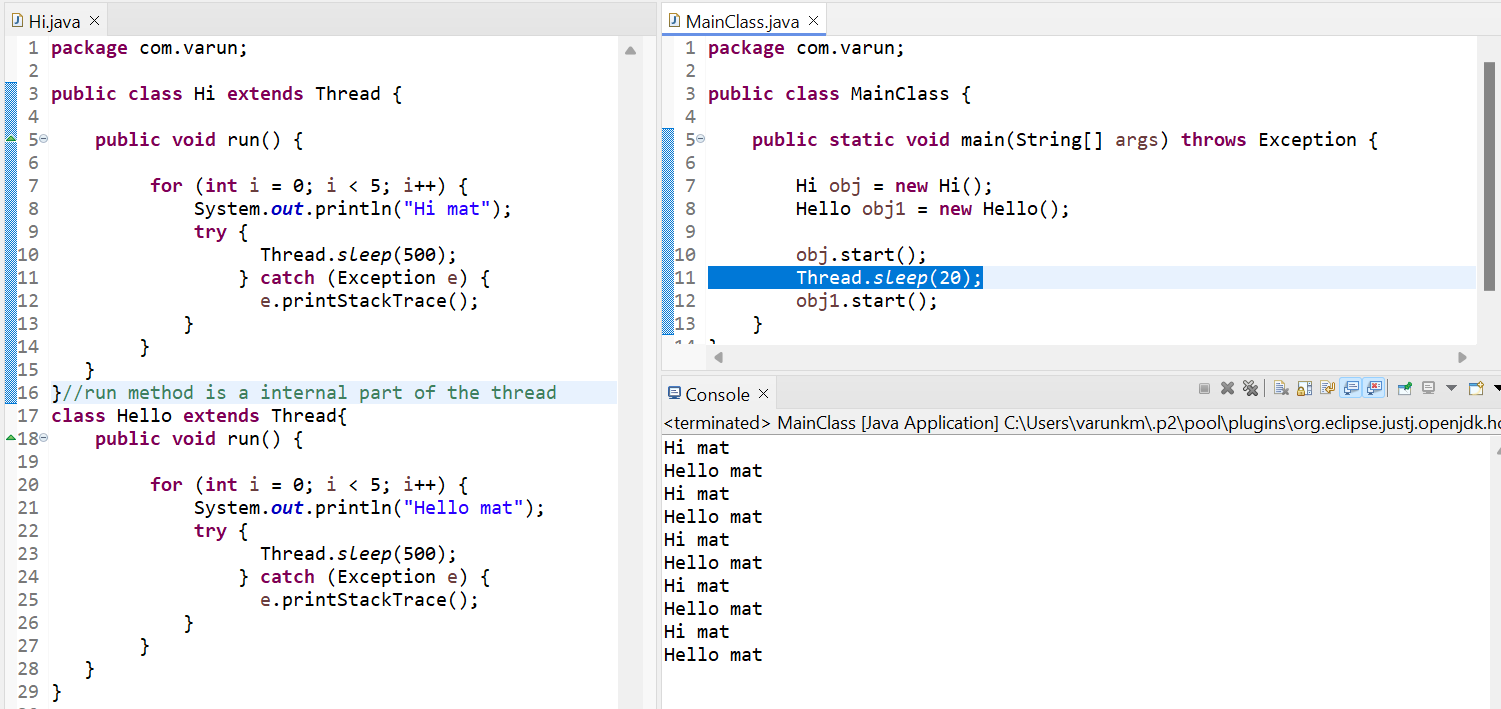


In this below example the schedular run through a random order and hence we can’t predict our output.

A screenshot of a computer

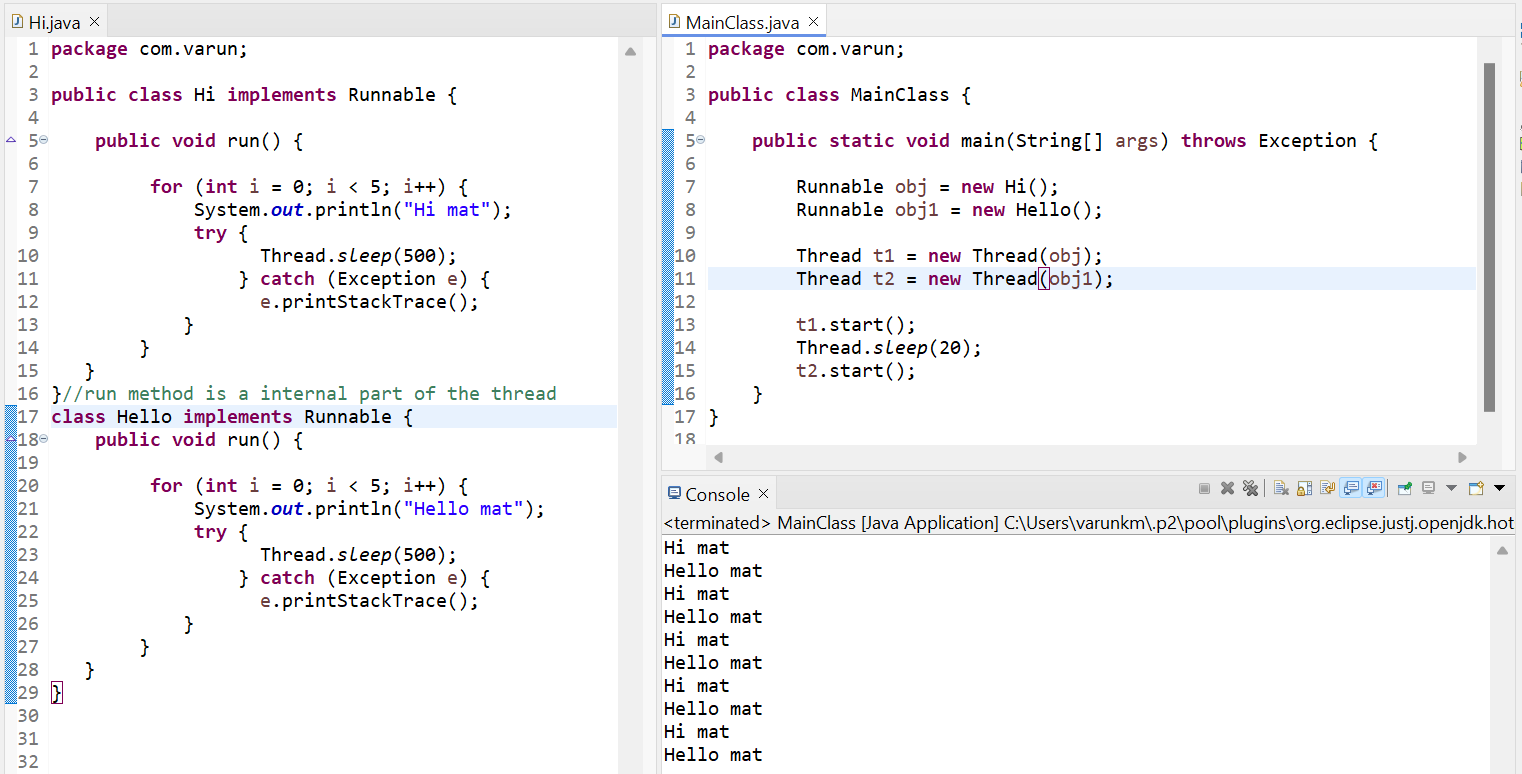
Description automatically generated

We are solving this issue we have pause a schedular some amount of secs example below



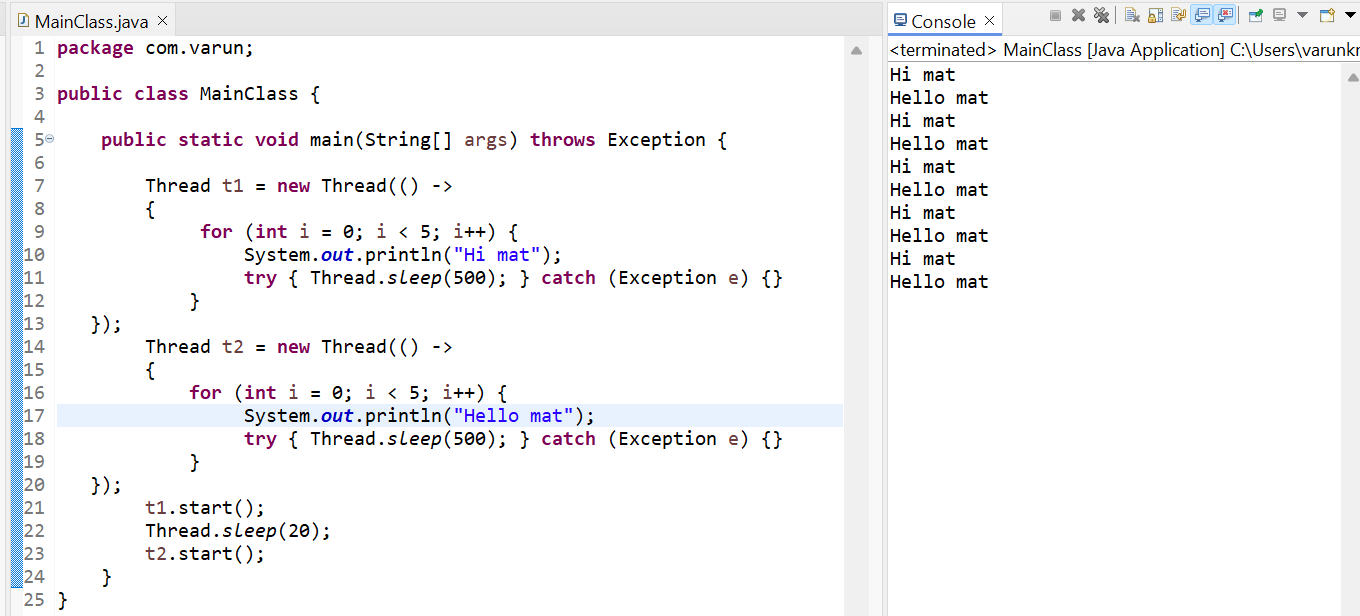
**Multi-Threading | Runnable Interface:**

Runnable interface is a functional interface, it has only one abstract method that is run.



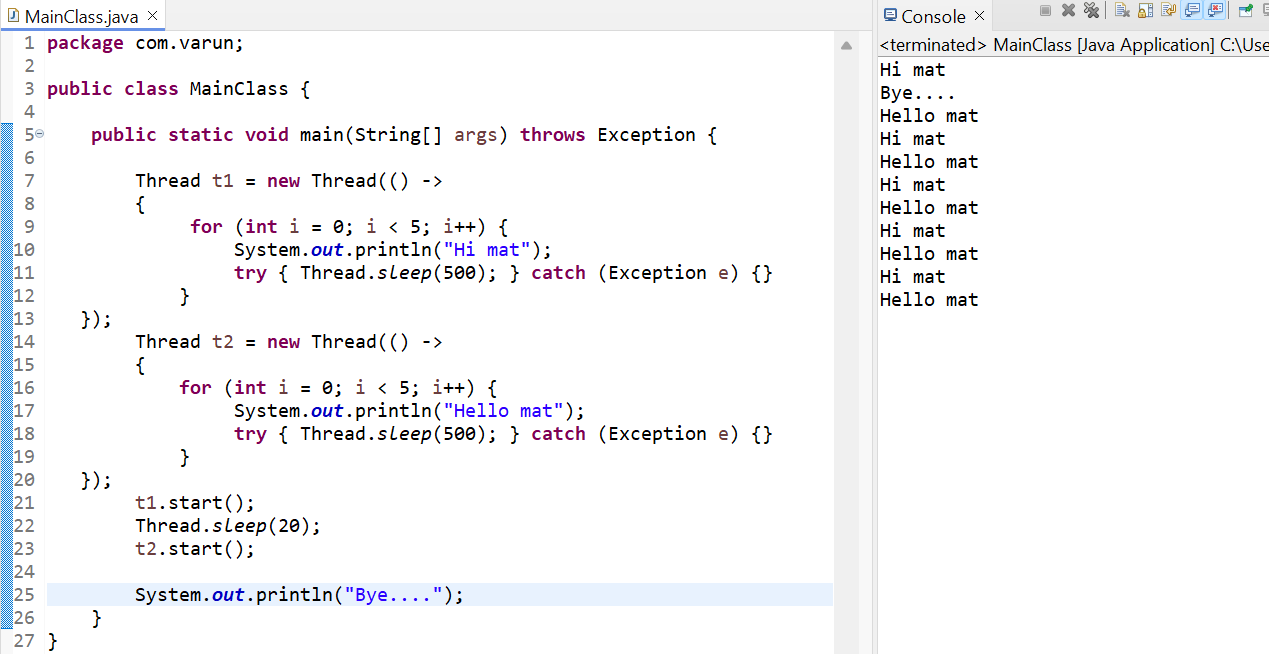
**Multi-Threading | Lambda Expression:**

1. The above program it increases the number of lines of code we are reducing the number of lines of code we are using lambda expression. Runnable is a SAM we can invoke lambda expression using Runnable interface.
2. Our code makes more efficient using lambda expression using a smaller number of classes using anonymous class. The below example shows how we are using lambda function and inner class. So, compare to the above program its lot of defference.

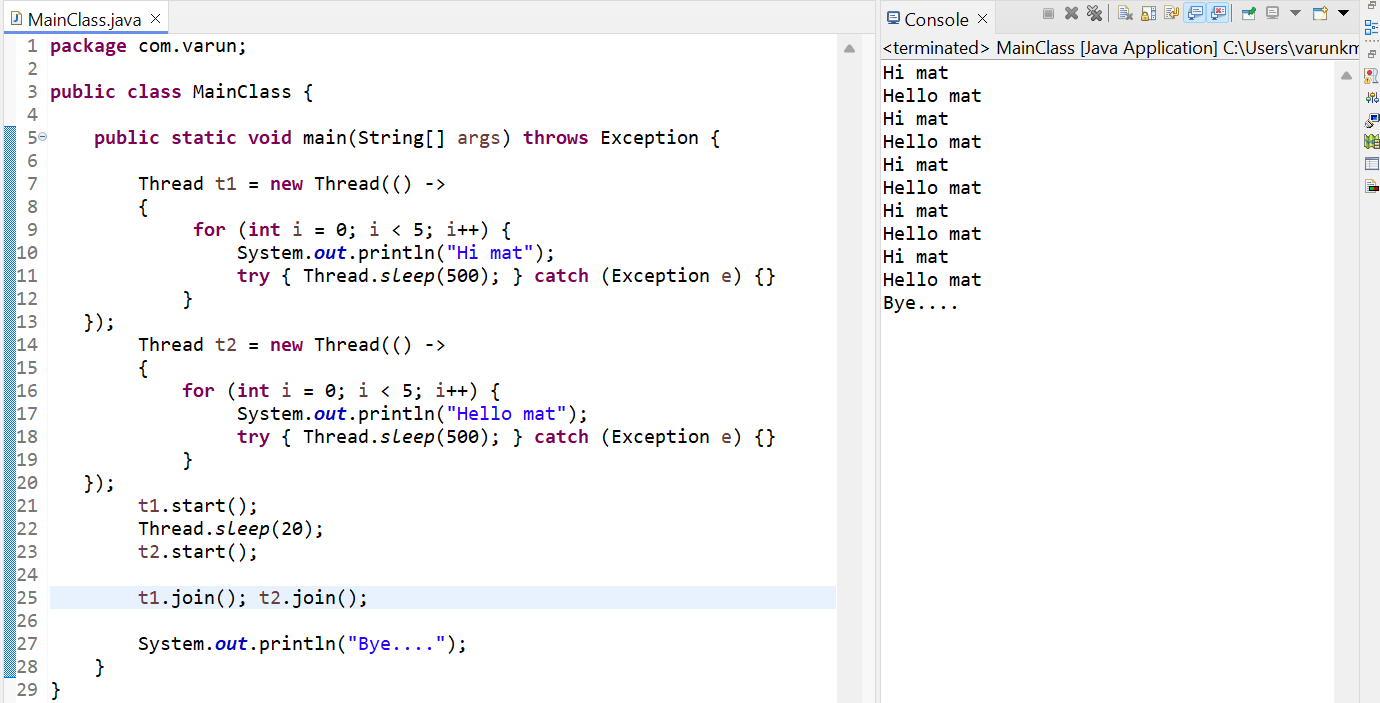


**Multi-Threading | Join |isAlive method:**

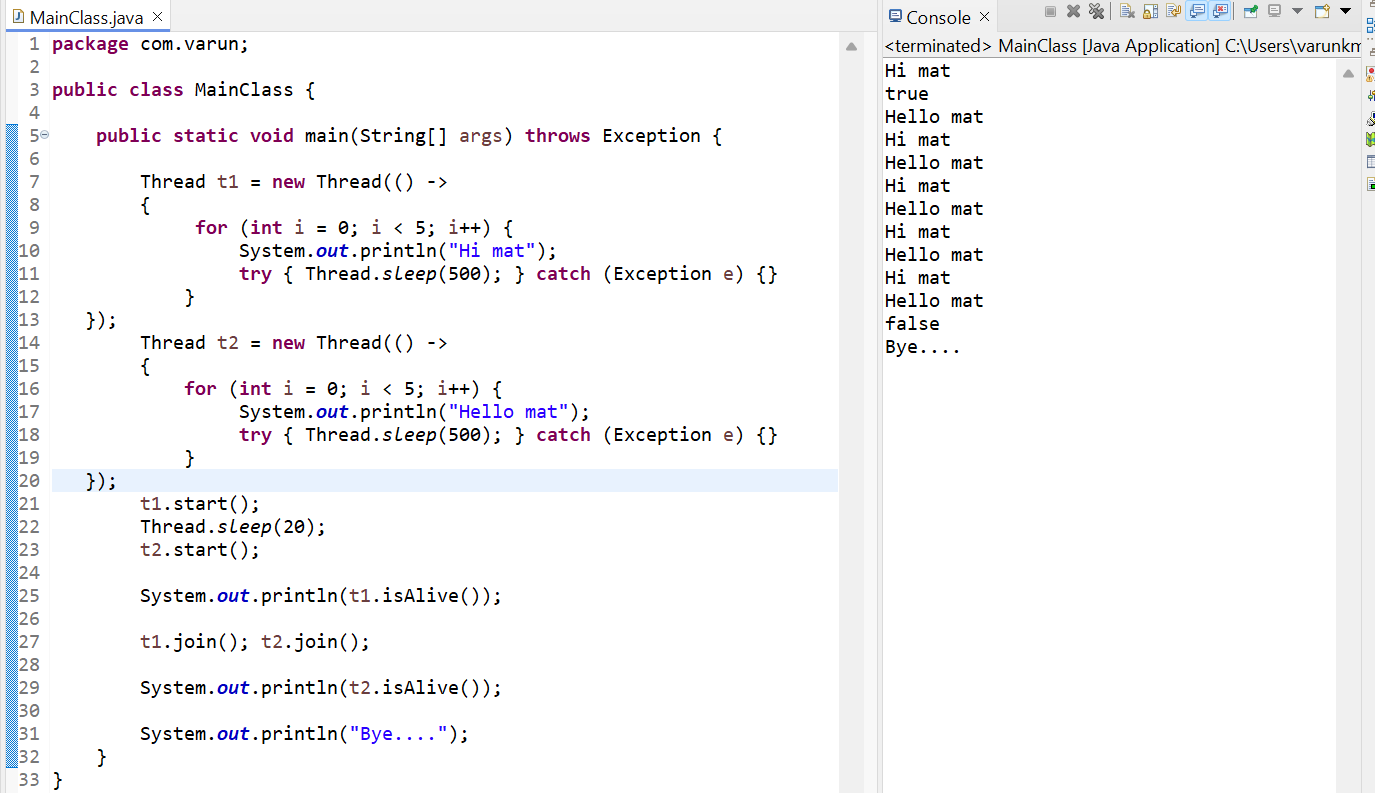
Compared to the above example, at last we must print bye, but when we do like this it will print in between. Because the t1 and t2 we have 2 threads are there. But main thread does have any work that’s why it prints bye …. After t1.



So, for this problem solution is join method, now we must tell the main method, as soon as completes the t1 and t2 you can print bye. This is the use of join () see below example. What join does join t1 and t2 each other.

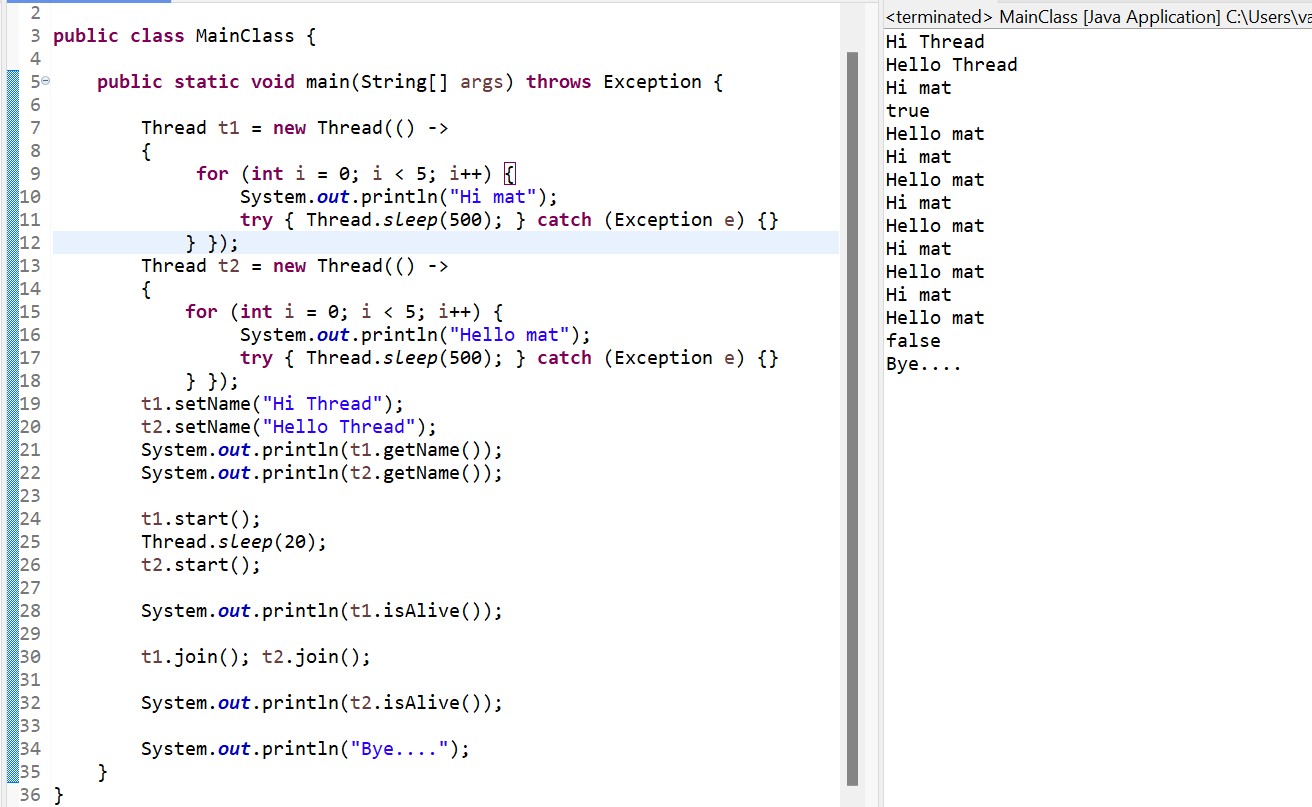


iAlive is a method using this method we detect our thread is running or not.

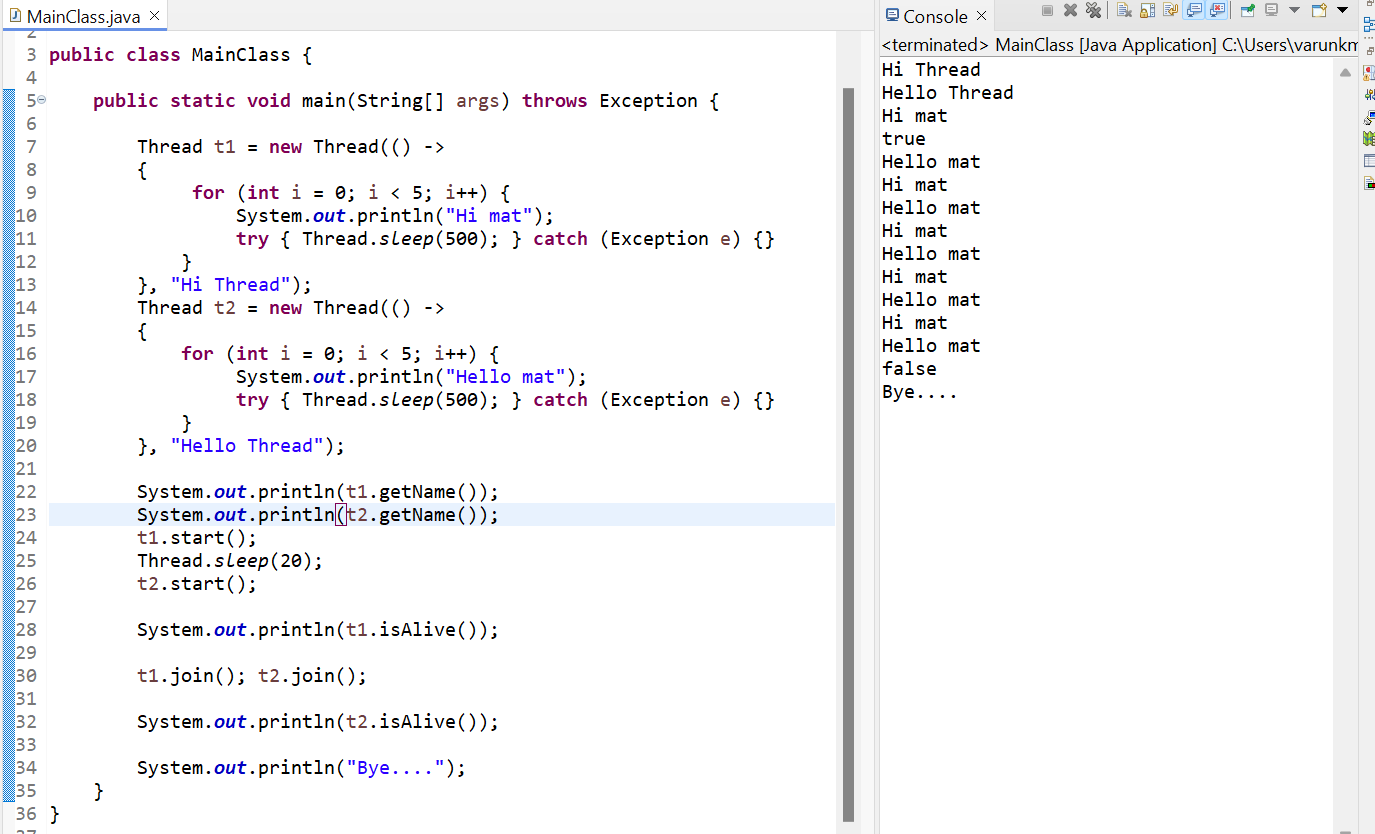


**Multi-Threading Thread Priority:**

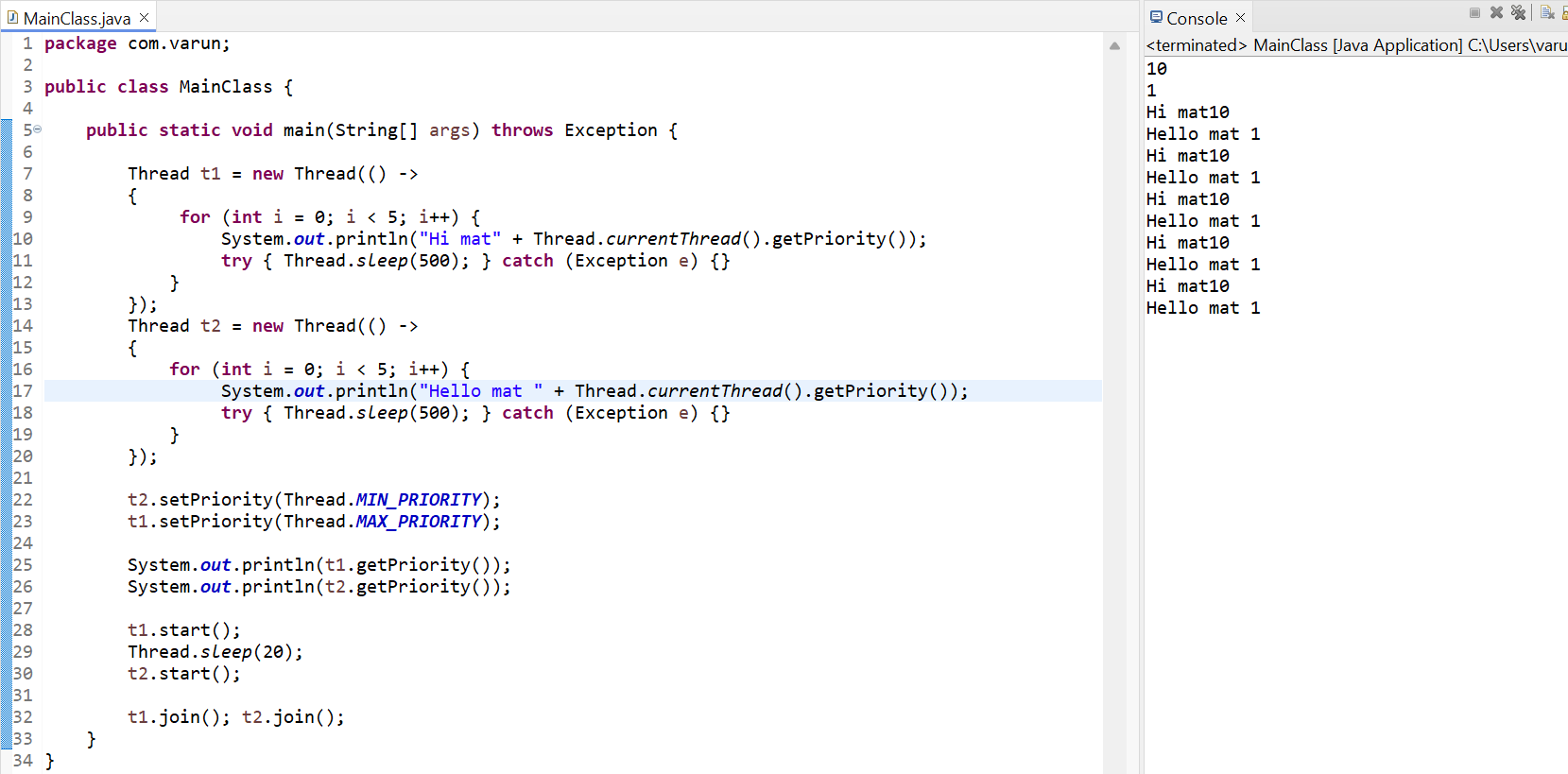
How to define a name for the Thread? We are using setName and getName methods set the name for the Thread. Example below  
Why this importance, suppose we are working with multiple Threads we don’t know which Thread is doing what sometimes its difficult to manage this is the reason we must provide a name for the thread.



Another way to give a name for the Thread is below example



Thread priority – By default whenever we create any thread it will have default priority. So, here 1 means least priority, 10 means highest priority, 5 means normal priority.



**Multi-Threading Synchronized Keyword:**

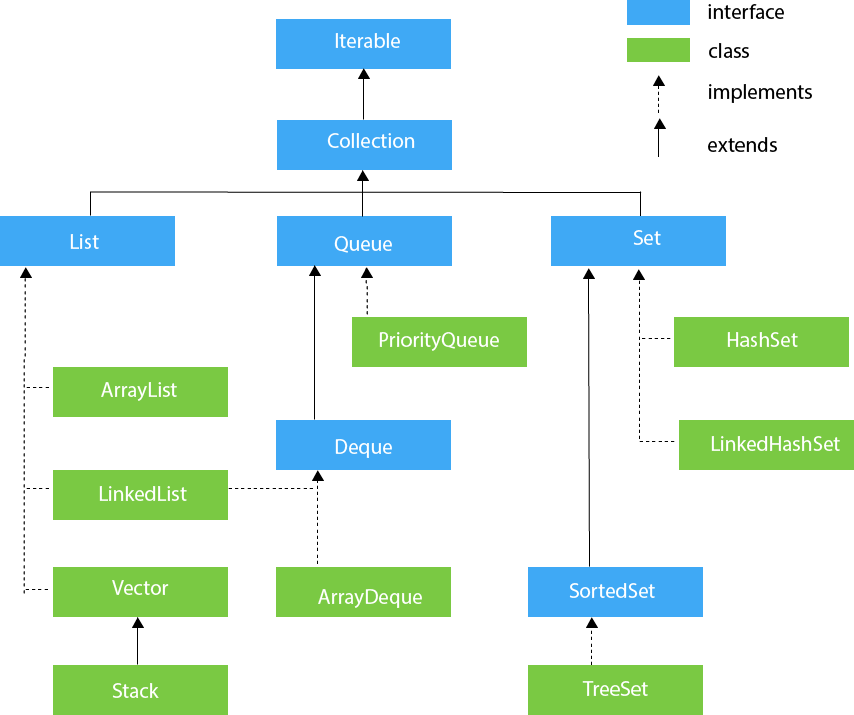
When we are using a synchronized method in a method one thread work with at a time. One thread execution is completed another thread will execute.

The below example t1 is working with increment method t2 must wait. While t2 is working with increment method t1 must be wait. So, maximum one thread should work.

When we don’t make method as synchronized that means our method, object or class is not thread safe.



**Collections and Generics in Java:**



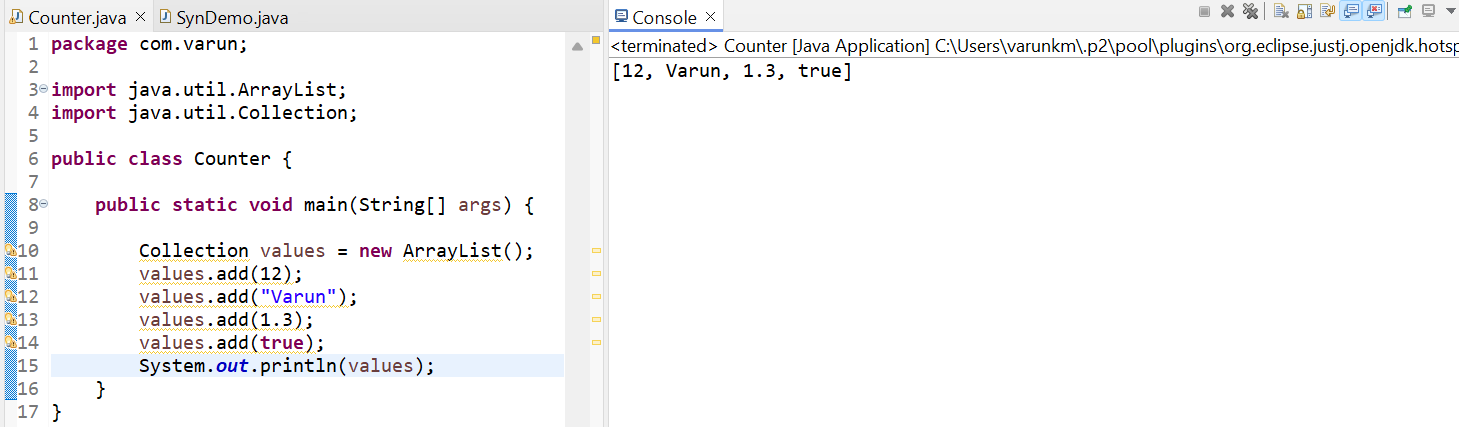
1. Collections – This is a class.
2. Connection – This is an interface.

Ex: I have a fixed array, I wat to extract

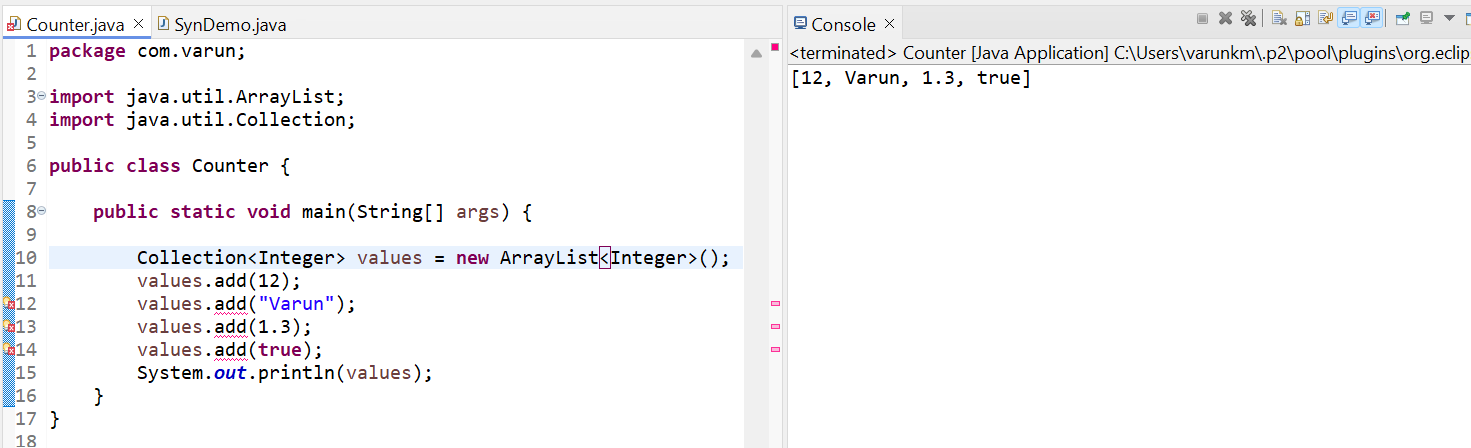
For this example, I am using **ArrayList -> it implements a List interface -> It extends a Collection interface** example below.

Note: The number of elements is fixed always go for the **ArrayList** because Array is fast compared to Collection. Then the size is not fixed always go for the Collection.

* List allows duplicate values.
* List get the element in sequence.



ArrayList stores homogenous and heterogenous values in it. But I need specific value like integer, String etc. for that wea are using **Generics**. Generally, it means flexibly and dynamically allocate the data.



**Set** -> It is an interface -> HashSet TreeSet and LinkedHashSet these are implements set interface.

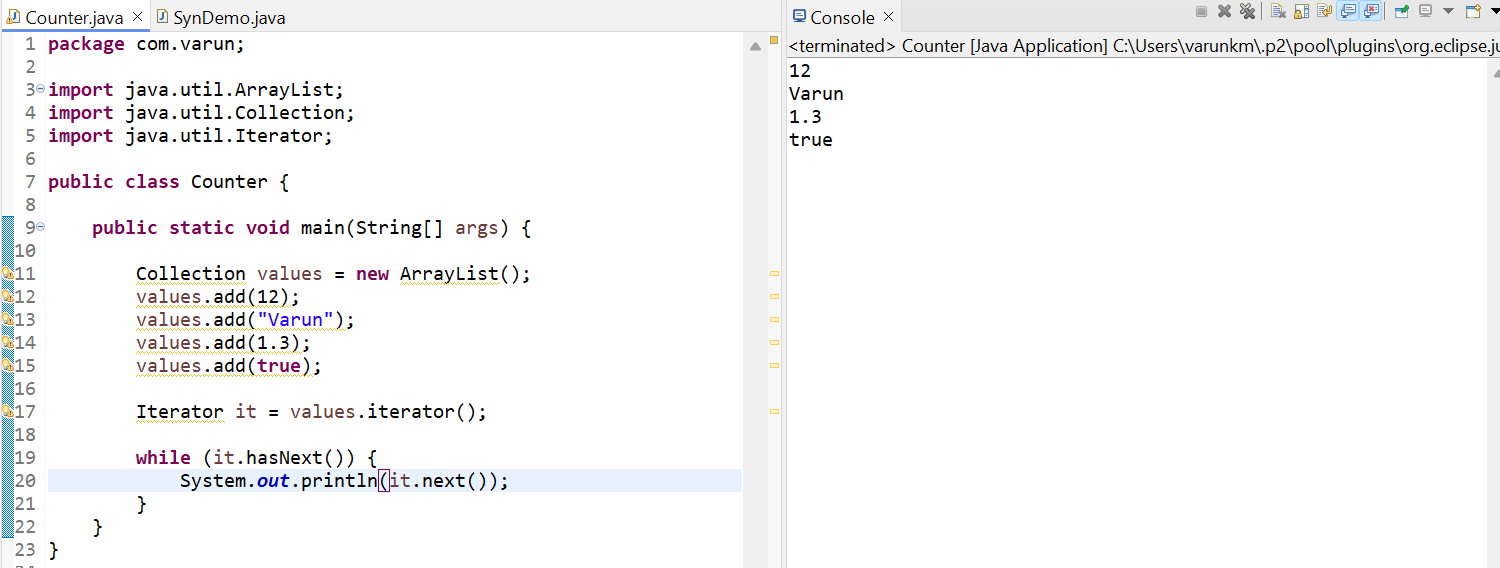
* Set doesn’t allow duplicate values. Every element should be unique.
* In set we are getting the element in random.

So, we have lots of classes and interfaces in one big package that is collection API.

**Map** -> It is an interface -> HashMap and HashTable is implements the map interface.

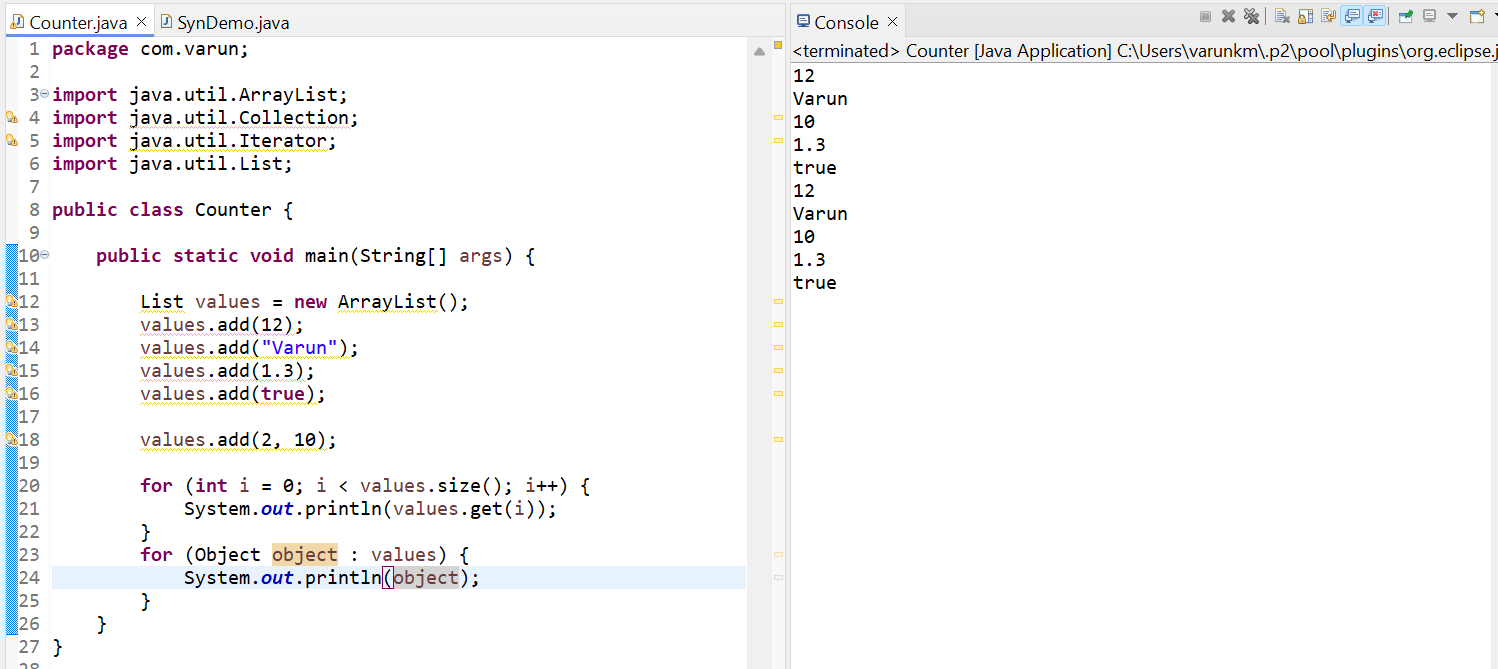
* it stores the data into key value pair.
* HashMap will not synchronize but HashTable is synchronized

**Collection and Generics | Iterator Interface:** Collection doesn’t support index number. We cannot fetch the values with index number. Another way is fetching the value is there is a special interface is called **Iterator** it is responsible to fetch the value one by one values in the collection.



We want to add an element in between the Array, for this scenario Collection won’t support we need to use **List** here

Both the loop works in this example enhanced for loop (foreach) and for loop.

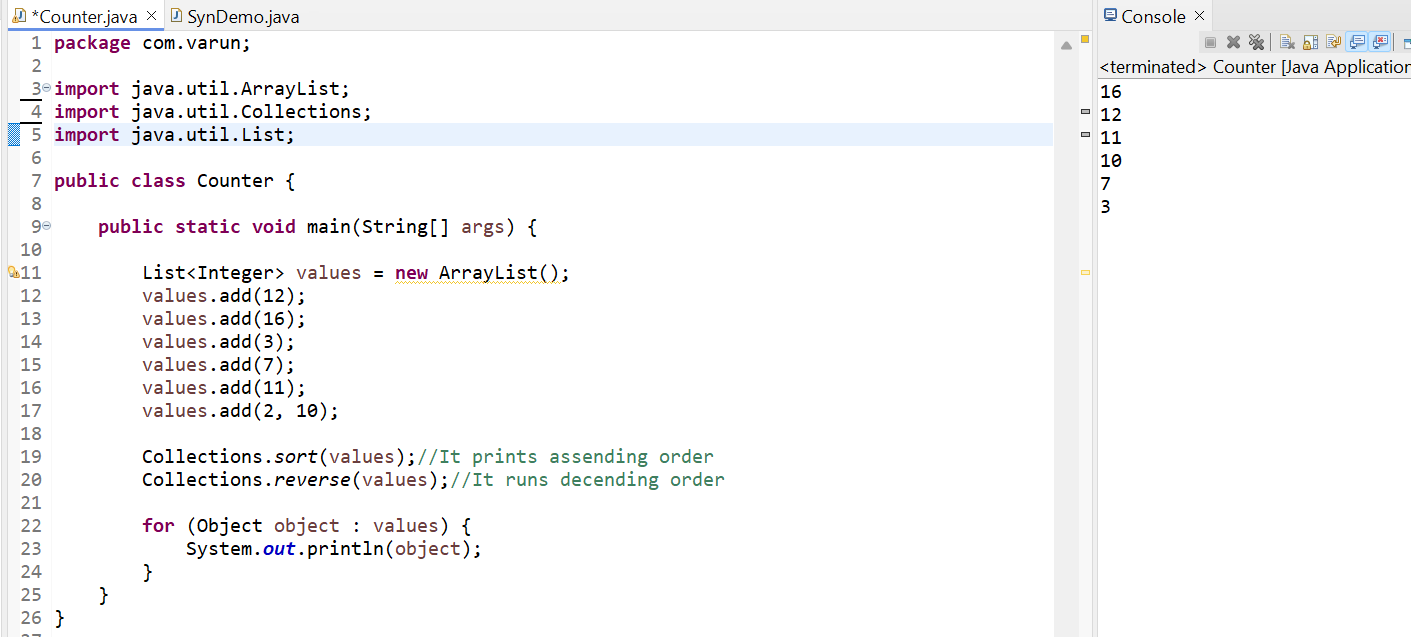


**Collection and Generics | Collections class:**

In ArrayList we get the elements in sequence manner. But we need an ascending order or anything sorting the order we want go with Collections class.

Collections is a class, and we are using sort () method is a static method. It will call our Collections class name.

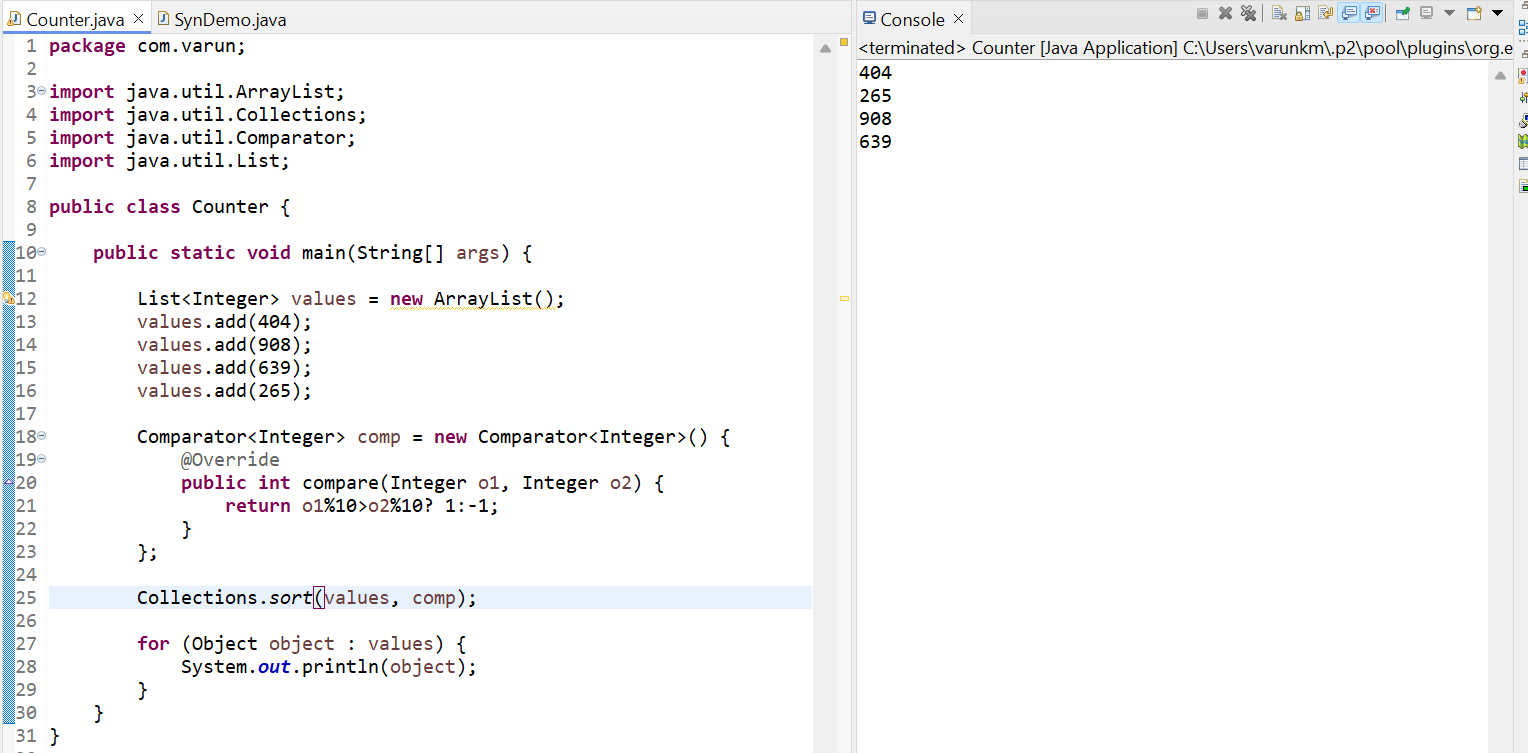
List is a **Mutable** class we can change the value in it.



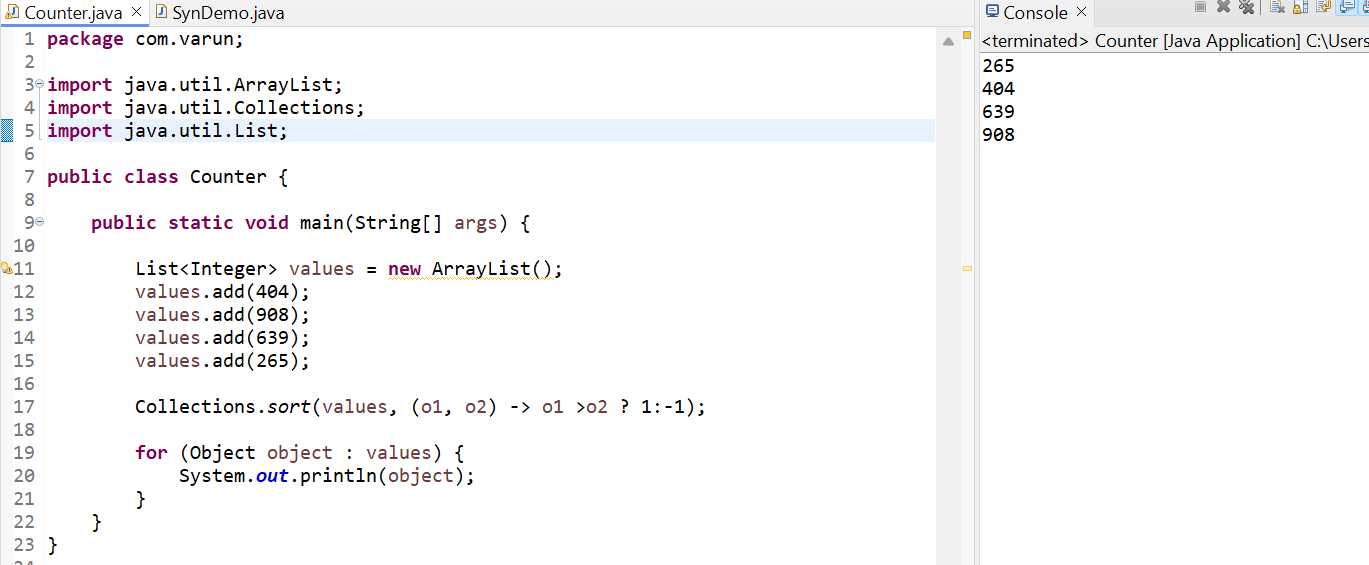
I have a number 897, 965, 743, 652, 523 I want to sort based on the last number. Means I want to customized sort.

We done this scenario using **Comparator**

**Collection Generics | Comparator Interface:**

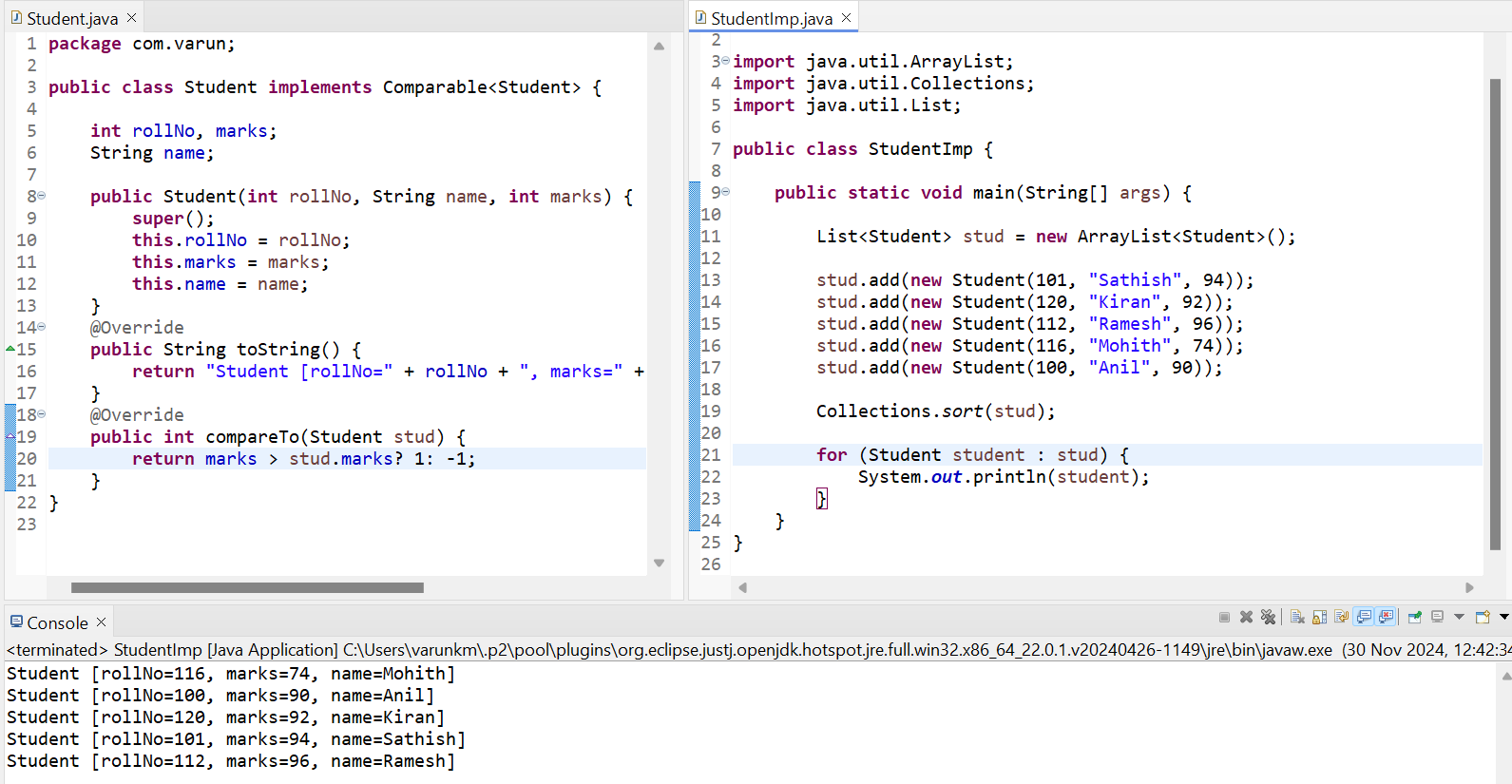
****

Above the same code prints using Lambda expression because Comparator have only one abstract method called compare () method. This time I am sorting whole 3 integer values.

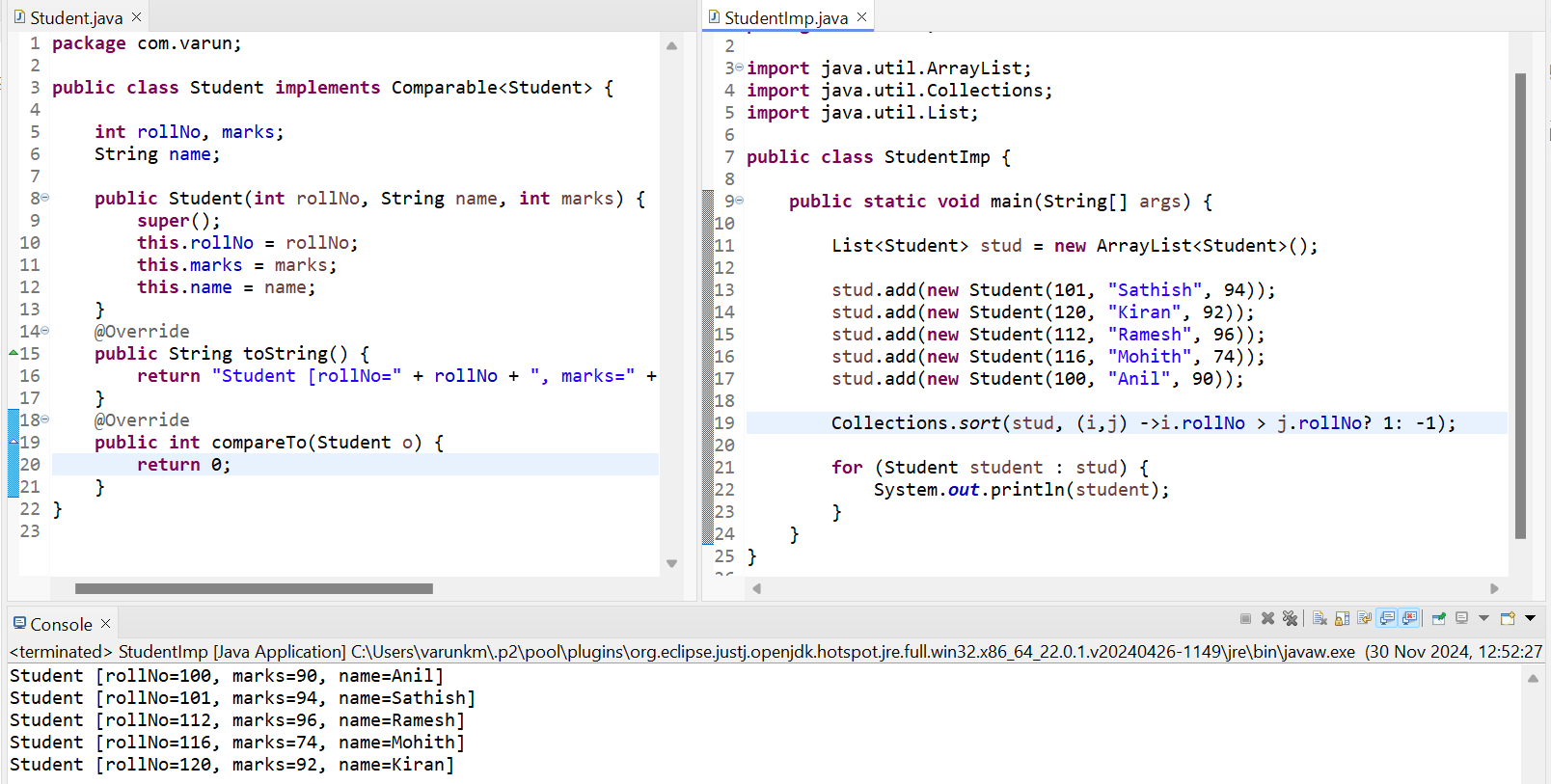


**Collection and Generics | Comparable Interface:**

We have a student objects; we need to sort based on the student’s marks. Using Comparable Interface, we are sorting the objects

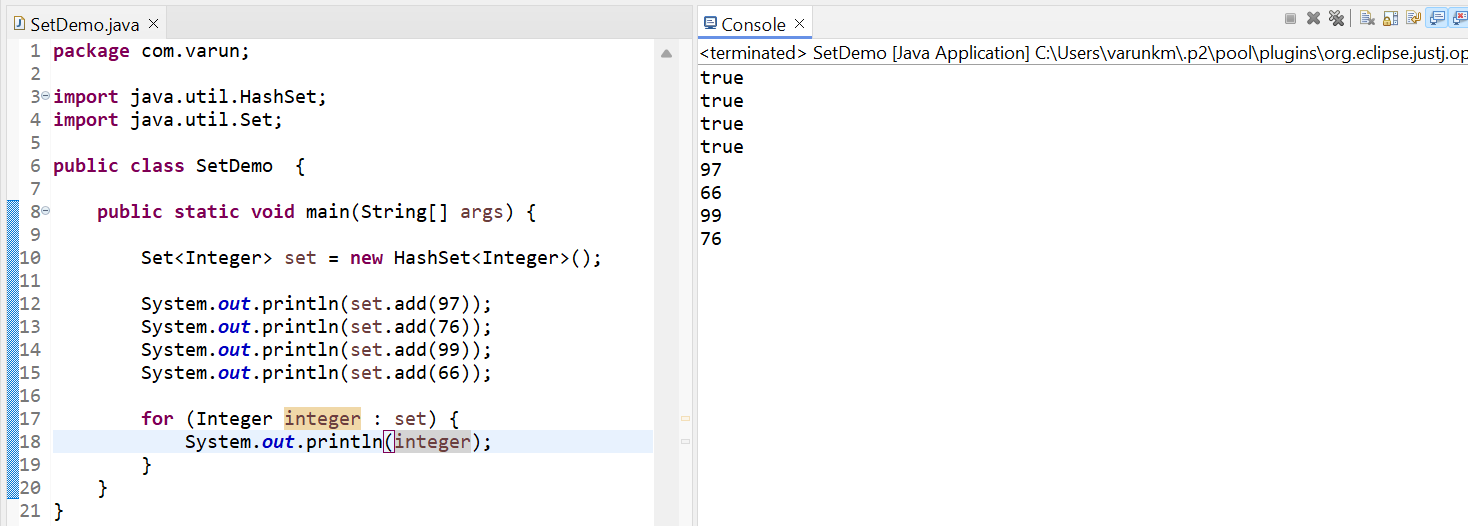


The same example, we can also use the Comparator interface into Comparable Interface example below. Here we comparing the rollNo.

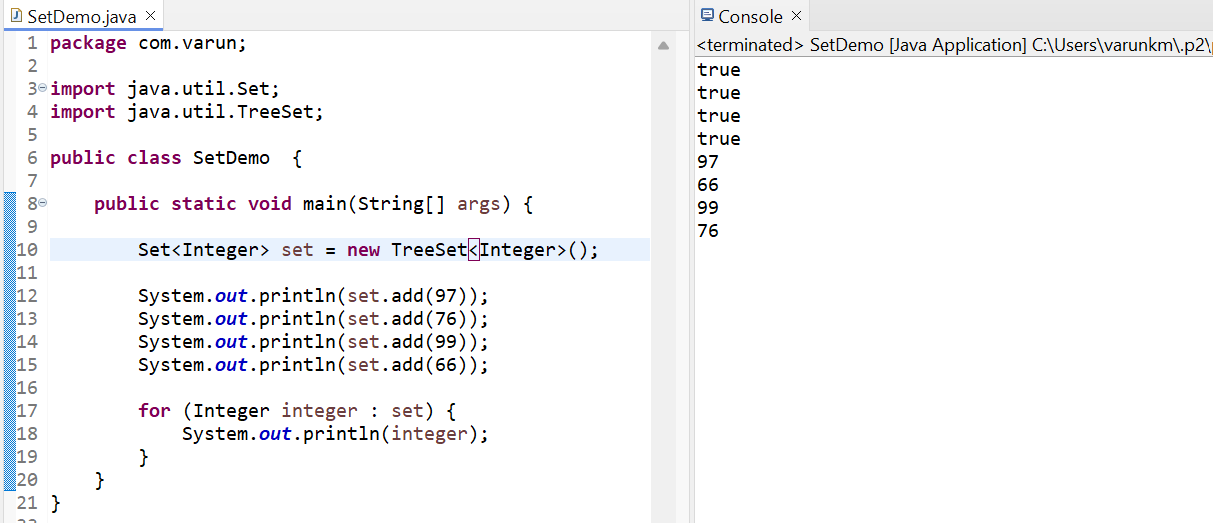


**Collection and Generics | Set Interface:**

Set interface we implemented using HasSet -> It will not ging the value in sequence.



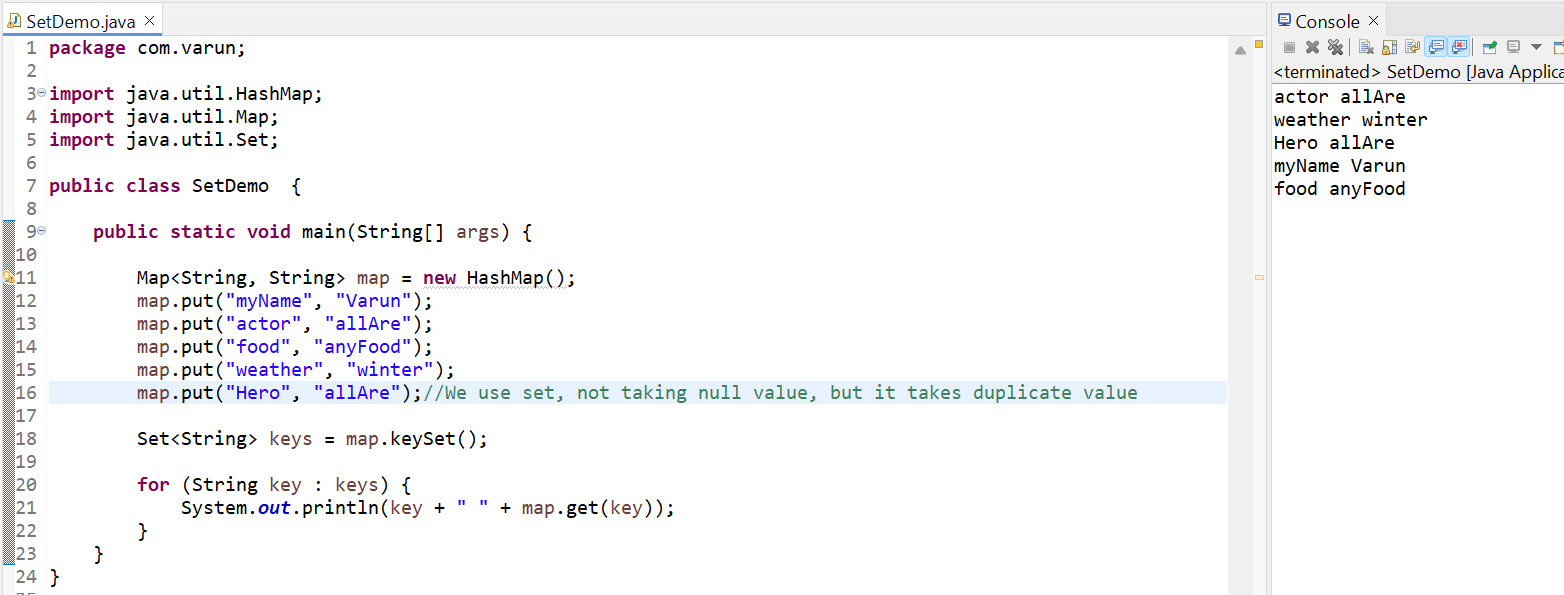
One more implementation class we have in set is TreeSet. It will be sorting the values in ascending order. We get the values in sequence.



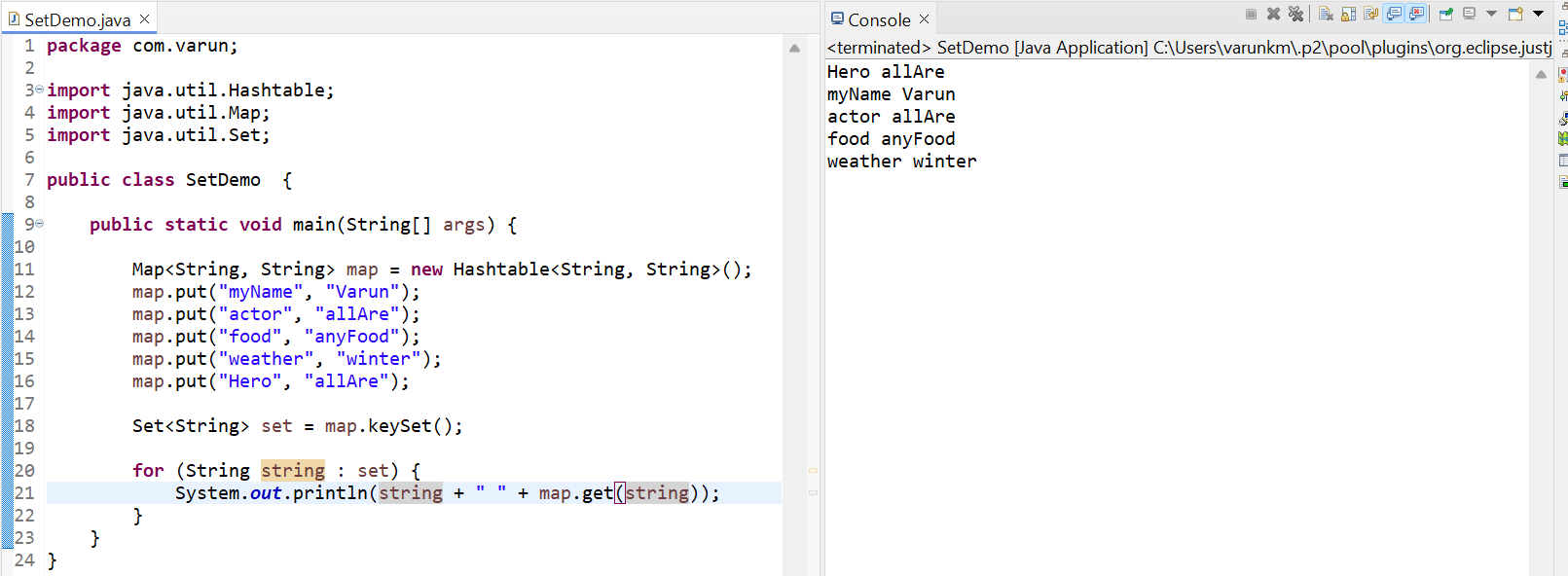
**Collection and Generics | Map Interface:**

We differentiate between two values using index numbers. We need a key value pair here to achieve we are using map. In map the value is not there also it will give null we don’t receive any exceptions.

We have keyset () method it will give us set of keys



Another way to implementing Map interface is HashTable. But is synchronized if we want Thread safety go with HashTable



**Flat Map:**

In Java, flatMap is a method provided by the **Stream API** that transforms each element of a stream into another stream, then flattens the resulting streams into a single stream.

1. **Purpose**:  
   flatMap is used when we have a stream of elements where each element can be transformed into a stream, and we want to flatten all these inner streams into a single continuous stream.
2. **Difference from map**:

* **map**: Transforms each element of the stream into another object. The result is a stream of streams (or nested streams).
* **flatMap**: Transforms each element into a stream and **flattens** the streams into a single stream.

**Example – 1: Basic usage:**

Let's say you have a list of lists of strings and want to process them as a single stream of strings.

