**Java -** Object oriented language & all operation we can perform using objects

**Project**

**Packages ->collection of similar classes – Package name follow with pascal case(i.e start with small char)**

Eg Packages name: com.lao.javalearning..in real project they will follow these-> company.sub.products [ something like that]

**Class** – Data + Methods(functions)

**Function** will work on the data only & do the works.[User defined function & system define function]

eg deposit/withdraw the amount in the banking account. Here amount be the data & deposit/withdraw be methods..so class is a single unit which compose of Data and methods. Without data, methods cant do anything & vice versa.

**Variables**

**Data** will be stored in the variables & we define the **data Types** like(Int, string, double etc)

Int X1 = “5”

Int ->Data Type & X1 ->variables& 5 – Data

**Starting point of Java class/Entry point of execution is Main() method**

Public static void main(String[] args){

}

In Java, we can create object for class & cant create object for **primitive data types (int, string, double,long,float etc)**

So to overcome this java developed **wrapper class (Integer, String, Double, Long, Float)** using this we can create objects for data types

For small numbers, we use Int & long number(lakhs) use Long.

**package** com.lao.javalearning;

**public** **class** Bankaccount {

//Here Long, String, Integer are wrapper class[its a object we can call inside the methods

//for long datatype need to append "l" at the last

Long accountnumber = 1234566l;

//String datatype, value need to give in double quotes

String accountname = "Suku";

Integer accountbalance = 1000;

**public** **void** getbalance() {

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

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//In order to call the getbalance fucntion, we need to create a object for class & then call method

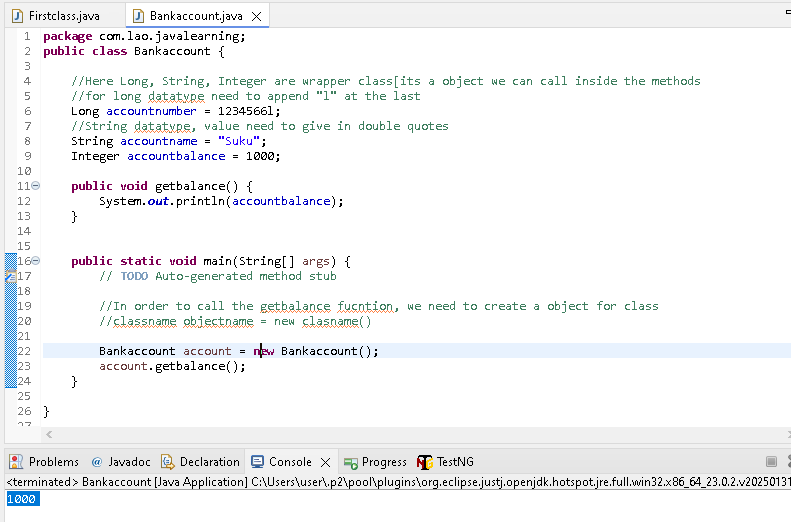
//classname objectname = new clasname()

Bankaccount account = **new** Bankaccount();

account.getbalance();

}

}



**Return type**

Void –null return values & not getting any returnvalue from the function

**public** void collectamountandgivetome()

//here Integer is the returntype & retuenvalue send back to the main() method

**public** Integer collectamountandgivetome() {

System.***out***.println("Son collected the amount from other person" +collectedamount );

//need to add below type for returntype

**return** collectedamount;

}

----------------------------------

**package** com.lao.javalearning;

**public** **class** Collectamount {

//eg dad ask son to collect amount from other person & give it back

**public** Integer collectedamount = 2000;

//here Integer is the returntype & retuenvalue send back to the main() method

**public** Integer collectamountandgivetome() {

System.***out***.println("Son collected the amount from other person" +collectedamount );

//need to add below type for returntype

**return** collectedamount;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

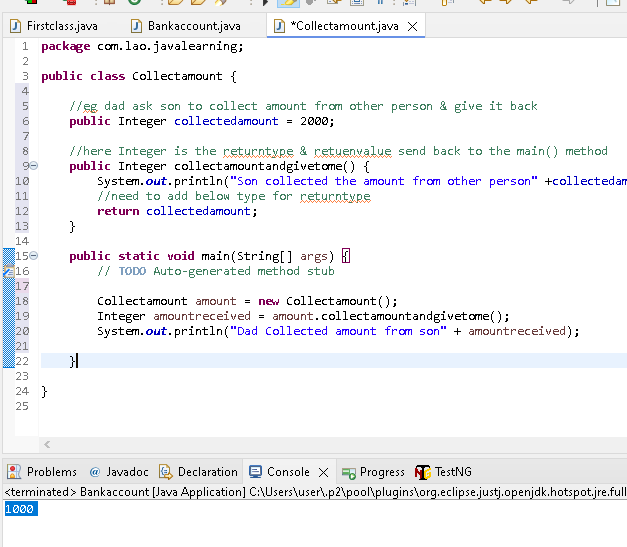
Collectamount amount = **new** Collectamount();

Integer amountreceived = amount.collectamountandgivetome();

System.***out***.println("Dad Collected amount from son" + amountreceived);

}

}

****

**Constructor ->** it’s a block of code that initializes the newly created object. Like special type of function but don’t have any returntype

initializes the new created object ->means assigning the values to the object

{

//block of code

}

**Properties of Constructor:**

1. Constructor name same as class name
2. Must not have explicit return type
3. Cannot be abstract, static, final, and synchronized

**Type of Constructor:**

1. Default 2.No Arguments 3. Parameterized

**Default Constructor explained:**

When we don’t provide any Constructor, compiler will provide one.

We cannot see with naked eye, it will be in class file

Used to provide the default values to the object like 0, null etc depending on the type

**Below code, we have not assign any values to the student name & roll number..so default Constructor assign values depending on the type**

**public** **class** Constructorexamples {

String studentname;

**int** studentrollnumber;

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

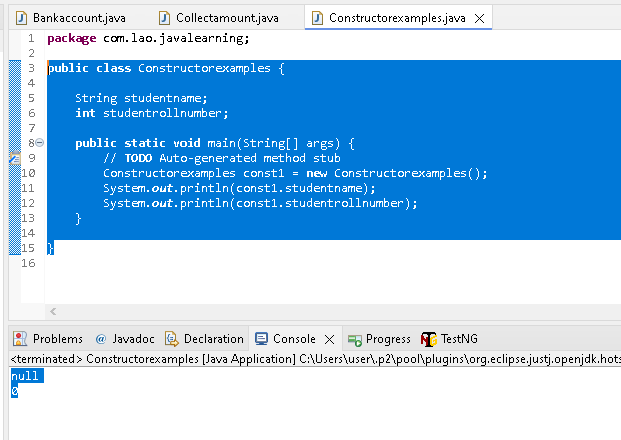
Constructorexamples const1 = **new** Constructorexamples();

System.***out***.println(const1.studentname);

System.***out***.println(const1.studentrollnumber);

}

}



**No Arguments constructor**

No argument(parameter) will be there

It will allow us to write logic when object is created

**package** com.lao.javalearning;

**public** **class** Noargumentconstructor {

**int** employeeid;

String employeename;

//below is a No argument constructor

Noargumentconstructor(){

employeeid=1;

employeename="Suku";

System.***out***.println("Employee object is created");

}

**public** **static** **void** main(String[] args) {

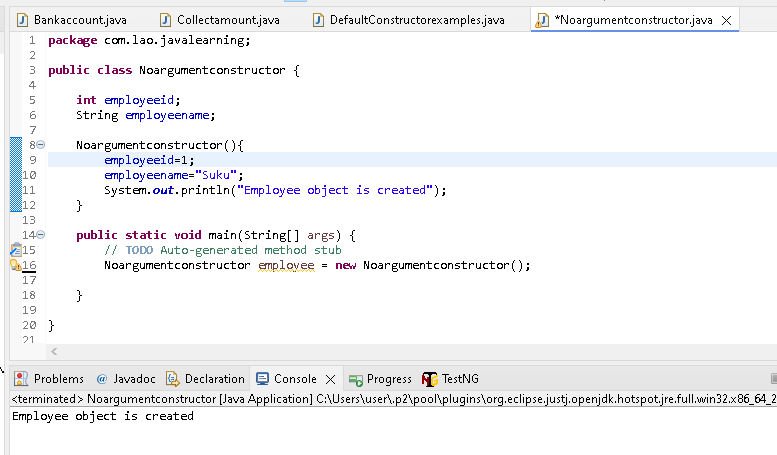
// **TODO** Auto-generated method stub

//below line will calling the Noargumentconstructor automaticllay when object is created

Noargumentconstructor employee = **new** Noargumentconstructor();

}

}



**Parameterised constructor**

Argument(parameter) will be there

**package** com.lao.javalearning;

**public** **class** Parameterizedconstructor<animalname> {

String animalname;

String animaltype;

Parameterizedconstructor(String string, String string2){

animalname =string ;

animaltype = string2;

}

**public** **void** Animaldetails() {

System.***out***.println("Animal name is" +animalname+ "Animal Type is"+animaltype);

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//while creating the object itself we are assign/sending the paramter

Parameterizedconstructor Animal = **new** Parameterizedconstructor("Duck", "Omnivorses");

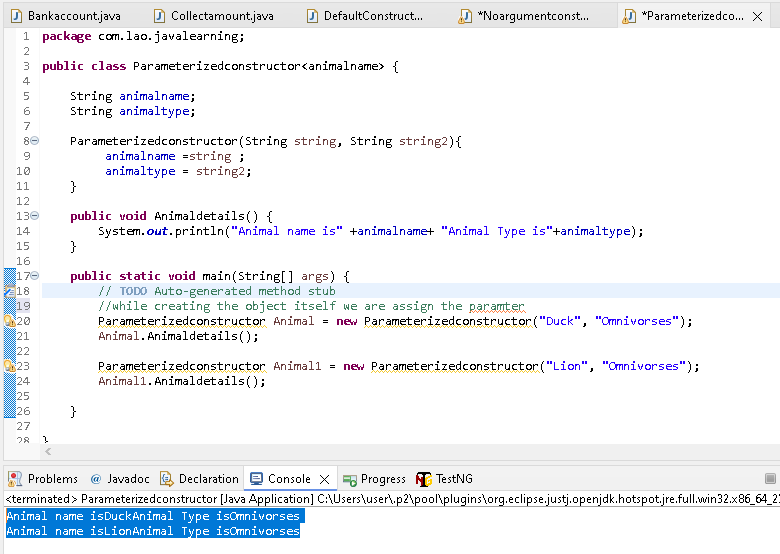
Animal.Animaldetails();

Parameterizedconstructor Animal1 = **new** Parameterizedconstructor("Lion", "Omnivorses");

Animal1.Animaldetails();

}

}

****

**Constructor overloading**

Same person at different/different situation will exploring/behaving differently

Eg myself will behave differently with my mom / my wife / my son / friends

**package** com.lao.javalearning;

**public** **class** Constructoroverloading {

Constructoroverloading(){

System.***out***.println("Drawing any object");

}

Constructoroverloading(String str){

System.***out***.println("Drawing" +str);

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

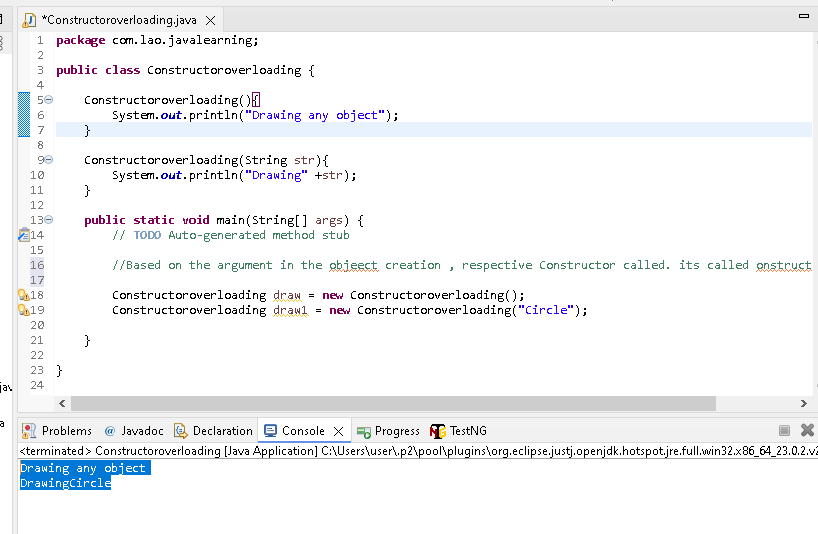
//Based on the argument in the objeect creation , respective Constructor called. its called onstructoroverloading

Constructoroverloading draw = **new** Constructoroverloading();

Constructoroverloading draw1 = **new** Constructoroverloading("Circle");

}

}



Super Keyword ->works in inheritance

This & Super

This ->it will update in current class

Super ->it will update parent class first by using access modifier

Constructor can have access modifier . extends is a access modifier

**Childclass extends Parentclass**

**Child class**

**package** com.lao.javalearning;

**public** **class** Childclass **extends** Parentclass{

**public** Childclass() {

//by using extends[Access modifier], it will go and check the other class[eg parent class]

//by default super() keyword will be present in the first line of constructor[cant see in naked eyes]

//super() keyword will execute the parent class first

System.***out***.println("This is Child class");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Childclass class1 = **new** Childclass();

}

}

**Parent class**

**package** com.lao.javalearning;

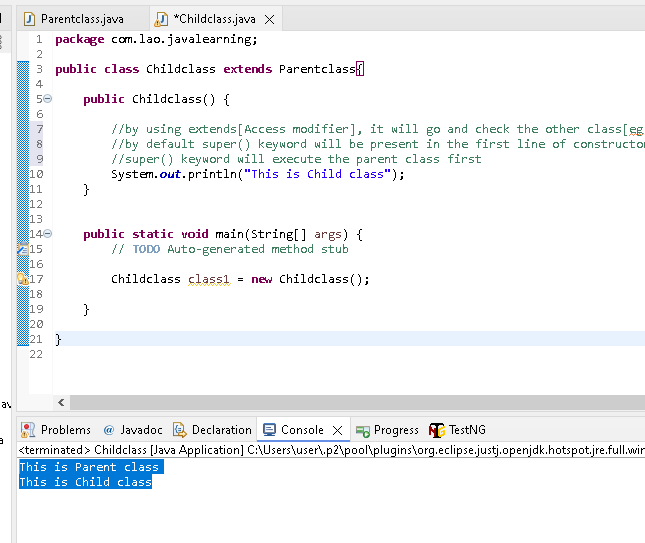
**public** **class** Parentclass {

**public** Parentclass() {

System.***out***.println("This is Parent class");

}

}

****

**Looping statement in Java**

When we have to perform certain operation repeatedly

For loop

While loop

Do while loop

Loop with out condition is infinite loop & system will hang during run. Always have exit condition need to call in any of the loop.

**package** com.lao.Loop;

**public** **class** Forloop {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//For loop ->[Initialization; condition; increment in one line]

**int** i;

**for**(i=0;i<=10;i++) {

System.***out***.println("My name is Sukumar");

}

//While Loop->same but Initialization; condition; increment all in different lines]

//Her first check the condition & then allow to execute the code

//its called as Entry control loop

**int** j=0;

**while** (j<=10) {

System.***out***.println("My name is Sukumar");

j++;

}

//Do while Loop->Here condition at the last.Here first execute the code & then check the condition

//its called as Exit control loop

**int** k=0;

**do** {

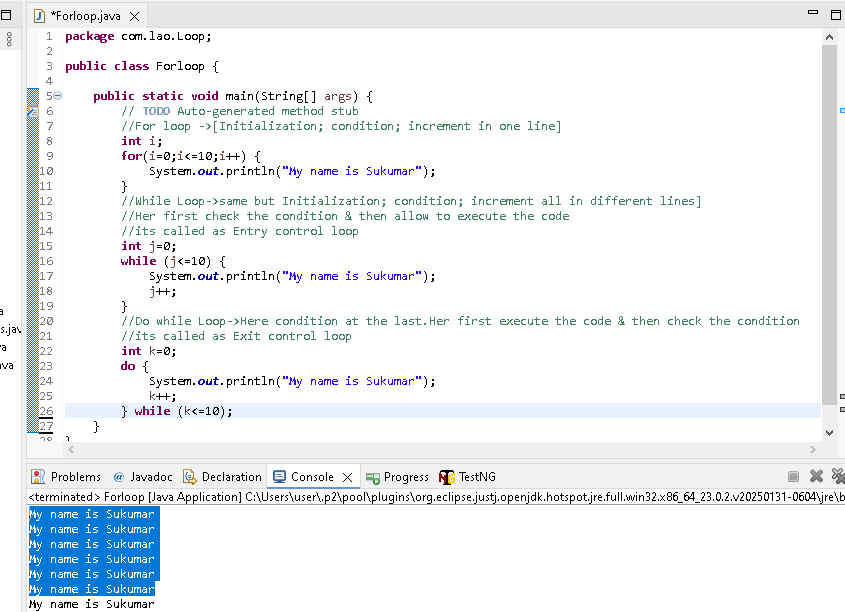
System.***out***.println("My name is Sukumar");

k++;

} **while** (k<=10);

}

}



**Conditional Statement in Java**

1.If condition -> The statement gets executed only when the given condition is true.If the condition is false then the statement inside if statement body are completely ignored

2.If else condition ->

The statement inside “If” would execute when the given condition is true. If the condition is false then the statement inside “else” statement body will be executed

**public** **class** Ifleseexamples {

**static** String *name* = "Suku";

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**if** (*name*.equals("Suku")) {

System.***out***.println("Executed if statement");

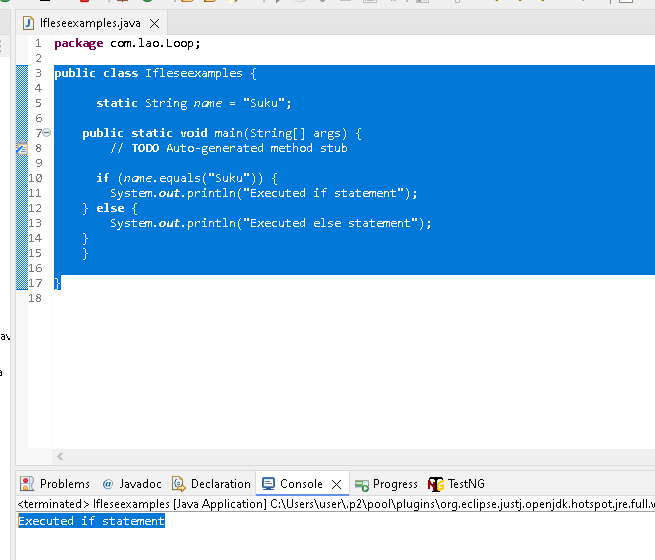
} **else** {

System.***out***.println("Executed else statement");

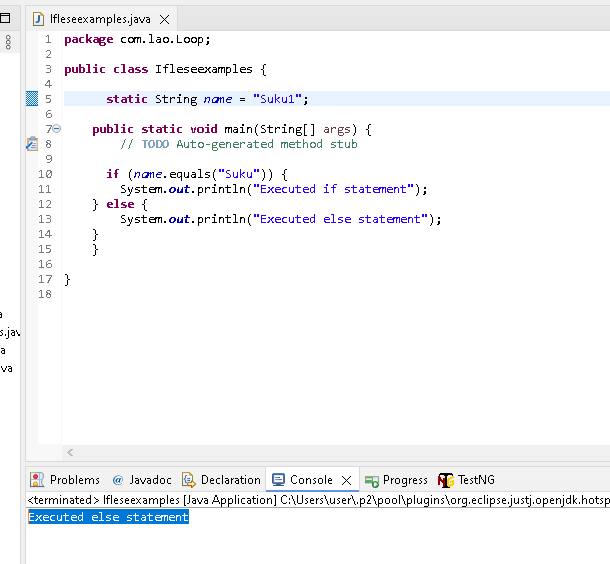
}

}

}



Else



3. if-else-if ->used when we need to check multiple conditions..

If one condition is true, then remaining else-if wont execute in if-else-if statement

If 2 to 3 condition to check, then go with if-else-if statement..if more than 5 condition then go with switch case

**package** com.lao.Loop;

**public** **class** ifelseifexamples {

**static** String *name* = "Suku1";

**public** **void** names() {

**if**(*name*.equals("Babu")) {

System.***out***.println("Matched at first run");

}

**else** **if**(*name*.equals("Jothi")){

System.***out***.println("Matched at second run");

}

**else** **if**(*name*.equals("Suku1")){

System.***out***.println("Matched at third run");

}

**else** {

System.***out***.println("Not at all Matched in all run");

}

}

**public** **static** **void** main(String[] args) {

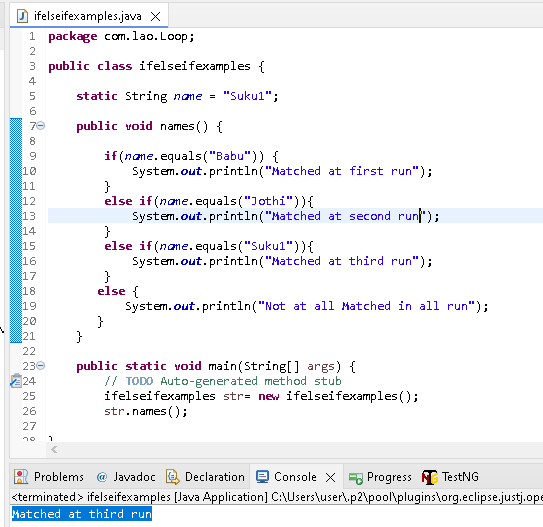
// **TODO** Auto-generated method stub

ifelseifexamples str= **new** ifelseifexamples();

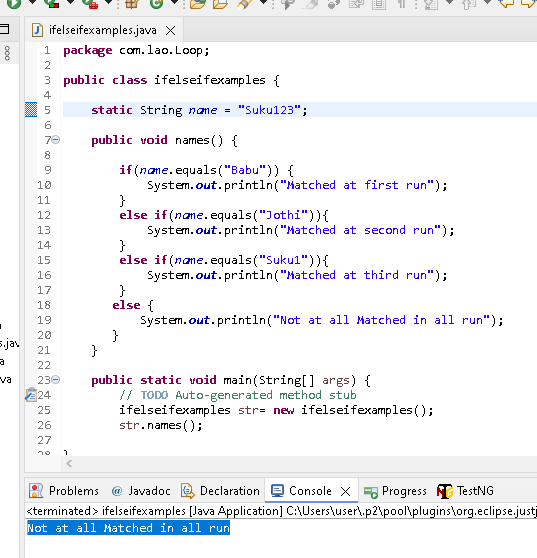
str.names();

}

}



Else



4.Switch Case ->Used when we have number of options(or choices) & we may need to perform a different task for each choice

If one case is true, it will always execute all the cases present below..Always use break statement if any of the case is true.

If None of the case is true, then it will execute Default case

Eg..here first case is true without break statement, so all below cases executed

**package** com.lao.Loop;

**public** **class** switchcaseexamples {

**static** String *name* = "Suku123";

**public** **void** names() {

**switch** (*name*) {

**case** "Suku123": {

System.***out***.println("Matched at first run");

}

**case** "Babu": {

System.***out***.println("Matched at first run");

}

**case** "Hari": {

System.***out***.println("Matched at first run");

}

**default**:

System.***out***.println("Not at all Matched in all run");

}

}

**public** **static** **void** main(String[] args) {

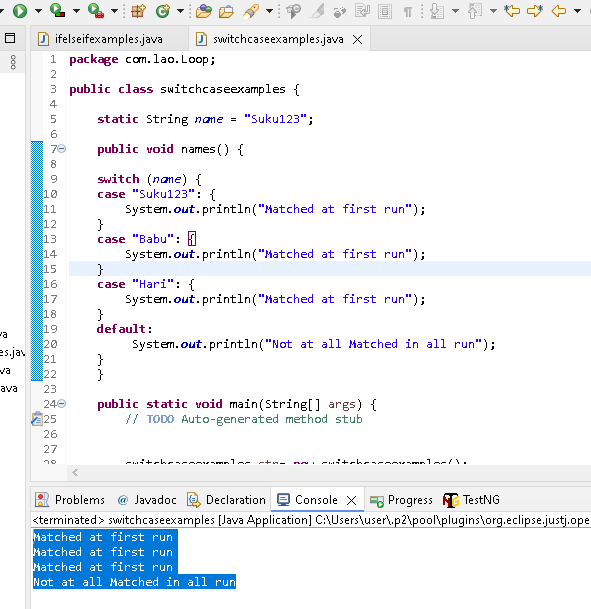
// **TODO** Auto-generated method stub

switchcaseexamples str= **new** switchcaseexamples();

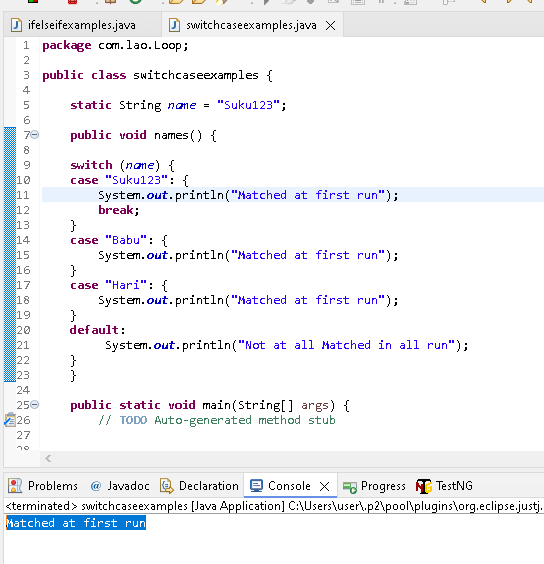
str.names();

}

}



With Break



**Static Keyword concepts**

1. It can be used with class, variables, method & block

2.belong to the class instead of specific instance/object..This means it you make a member static, then you can access it without object.

**Static Method**

Eg.Create a static method & call it without object

Note: We can call static method from non static method but not otherwise.

**public** **class** Staticmethod {

**public** **static** **void** staticmethod1() {

System.***out***.println("It a Static Method");

//We cant call Non static method from static method

//staticmethod2();

}

**public** **void** staticmethod2() {

//We can call static method from non static method but not otherwise

*staticmethod1*();

System.***out***.println("It a Non Static Method");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//if the method declare with static(public static void staticmethod1(), then we can the method

//without creating object.

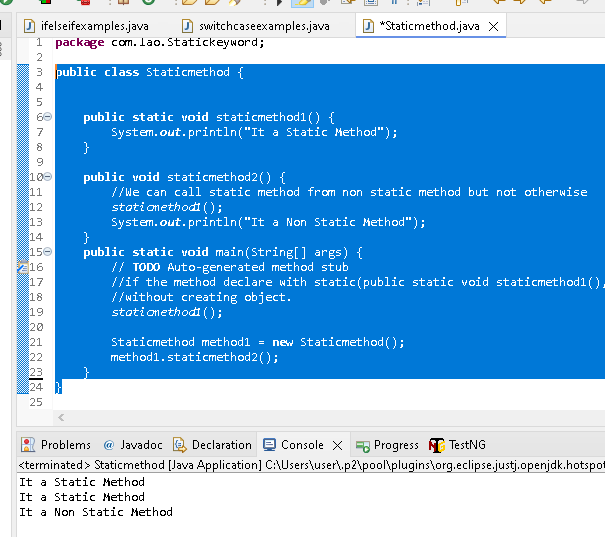
*staticmethod1*();

Staticmethod method1 = **new** Staticmethod();

method1.staticmethod2();

}

}

****

**Static Block**

{} - It’s a block

In class, we can have n number of static block & execution in written order

Before main method(public static void main(String[] args)), static will execute first in class

It used to initializing the static variables

**package** com.lao.Statickeyword;

**public** **class** Staticblock {

//Before main method(public static void main(String[] args)), static will execute first in class

**static** {

System.***out***.println("It a First static method");

}

**static** {

System.***out***.println("It a second static method");

}

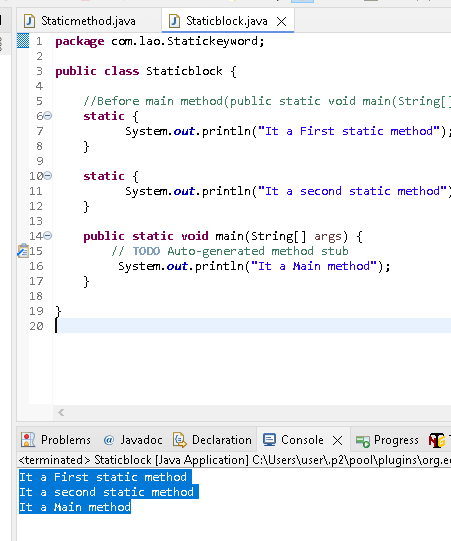
**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println("It a Main method");

}

}



**Static Variables:**

**1**. Common to all the instance (or object) of the class because it’s a class level variables

2. Only a single copy of static variables is created & shared among all the instance(object) of the class

**public** **class** Staticvaribles {

**static** **int** *intvarible* = 0;

String intvariable1l;

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//we can call the static variable without creating object & assign the value

*intvarible*=2000;

//we cant call the non static variable without creating objects.it will throw error

intvariable1l = "suku"

}

}

--------------------------------

**package** com.lao.Statickeyword;

**public** **class** Staticvaribles {

**static** **int** *intvarible* = 0;

String intvariable1l;

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Staticvaribles object1 = **new** Staticvaribles();

object1.*intvarible* = 1000;

object1.intvariable1l = "Suku";

Staticvaribles object2 = **new** Staticvaribles();

object2.*intvarible* = 2000;

object2.intvariable1l = "Jothi";

//statis variables(intvarible) will have last stored values only..Only a single copy of static variables is created

//and shared among all the instance(object) of the class

System.***out***.println(object1.*intvarible*);

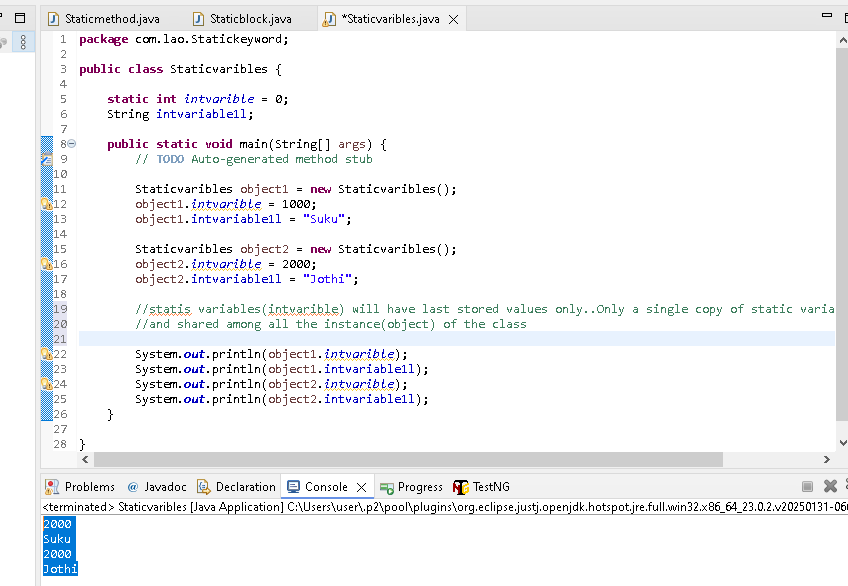
System.***out***.println(object1.intvariable1l);

System.***out***.println(object2.*intvarible*);

System.***out***.println(object2.intvariable1l);

}

}

****

**Static class**

A class can be made static only if is a nested class

We don’t need this. Because we won’t use nested class in real time of testing area.

**Inheritance (Access Modifier)**

Process of acquiring the properties (Data + methods)

The one which takes ->child class

The one which gives ->Parent class

Note:

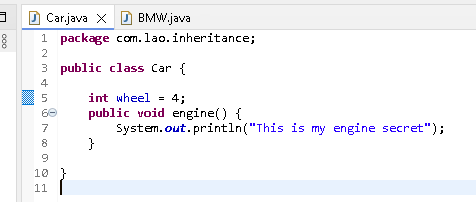
Child can be parent of another class

Parent can be a child of another class

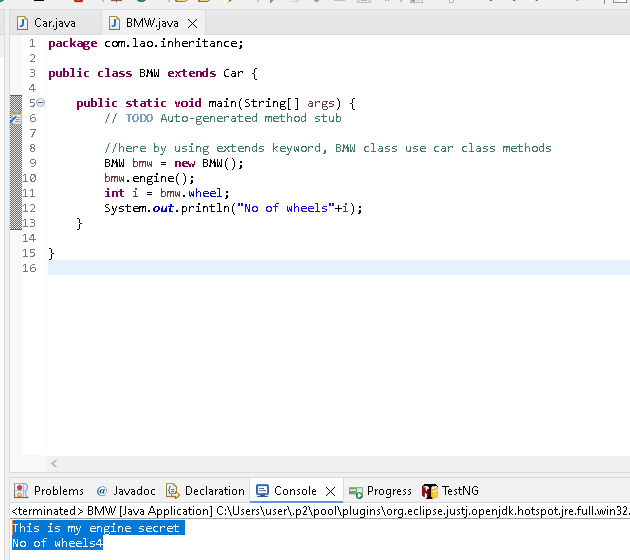
How to inherit? ->Using Extends keyword

IS- A relationship or parent- child relationship or Inheritance– all same only in Java..

**Parent class**

**\ **

**Child class –**BMW child class & its acquiring the properties of Car class



**Type of inheritance**

1. Single : A->B
2. Multi level : A->B->C
3. Multiple (Not supported in Java)

A+ B->C (By using interface we can achive..will learn later/upcoming module)

A & B -> both parent class for C child class

1. Hierarchical : A->B, A->C, A->D

**Access Modifiers: (Public, private, protected) all are access modifier**

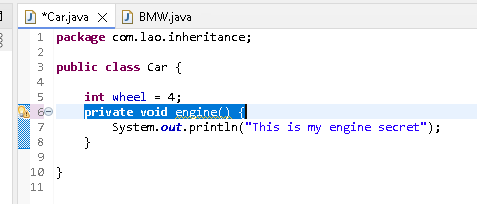
1. Default(**No access modifier**)-> When no access modifier is specified’
2. Private-> only within the class in which they are declared
3. Protected-> within same packages/sub classes(Parent- child) in different package
4. Public -> from everywhere in the program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Same Class** | **Same Package** | **sub class** | **Different Package** |
| **Public** | Y | Y | Y | Y |
| **Protected** | Y | Y | Y | N |
| **No access modifier** | Y | Y | N | N |
| **Private** | Y | N | N | N |

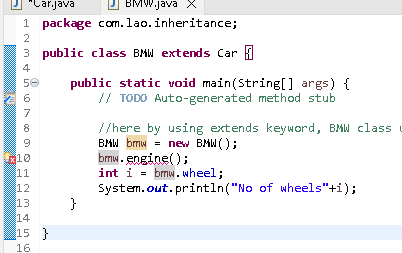
**Public** ->refer above tables

**Private** ->its only to class, not even to other class in same packages

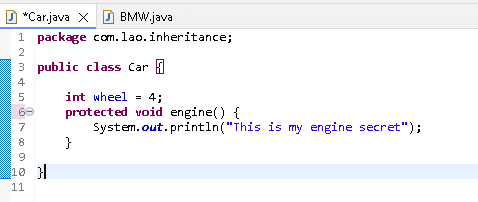
If we mention the parent class methods as Private, then even using **extends keyword, child class cant** acquiring the parent class methods



Error throwing at line 10

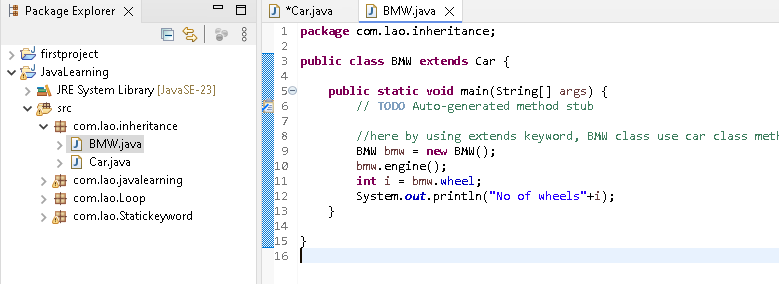


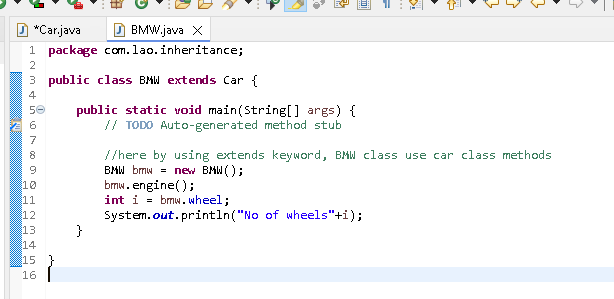
**Protected**->Its accessible to same class, same packages & only sub class(Parent- child) in another packages



Error disappears by using Protected, as its accessible to Same packages

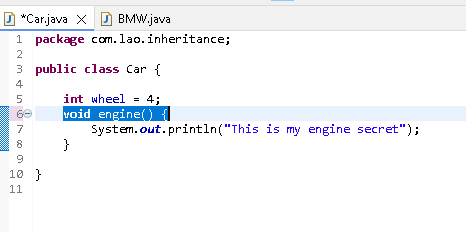
Car & BMW class under same packages





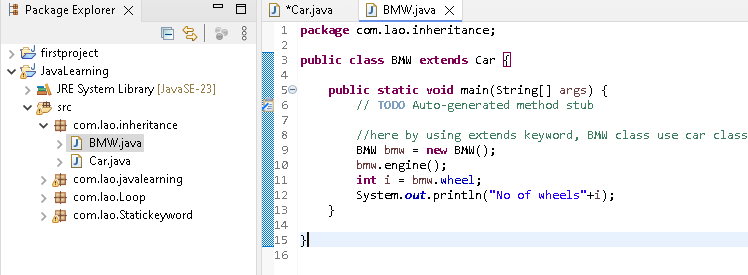
**Default access modifier**

Engine method have no access modifier in it.

****

**No error in BMW class, because its accessible to same packages**

Car & BMW class under same packages

****

**Polymorphism**

Q1: will you talk the same way with your friend as if you were talking to your GF/BF?

Q2:can we behave the same way in office as if we are in home?

No...Our activity changes based on the whom we are interacting with

**So what is polymorphism?**

Its is the capability of a method(function) to do different things based on the object that it is acting upon

Types:

1.Overloading [also called as static/compile/early binding]

2.Overriding [ also called as Dynamic/runtime/late binding]

**Overloading**

Method name should be same

If there is any change(change in data type or change in order of passing in the parameter passed to a method, then its overloading

Happen at compile time [ by clicking the method, it will navigate to that method position]

Done in the same class

**package** com.lao.polymarphism;

**public** **class** Waywetalk {

//we can give class names also in the argument(eg Parents)..Parents class also created in same packages

//created 2 method with same name , then only its Overloading

//Both method, argument is different

//all done in the same class (Waywetalk)

**public** **void** talk(Parents styleoftaking) {

System.***out***.println("Polite");

}

**public** **void** talk(parter styleoftaking) {

System.***out***.println("Love");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Waywetalk str = **new** Waywetalk();

//In order to call talk , created object for parents class

Parents parents = **new** Parents();

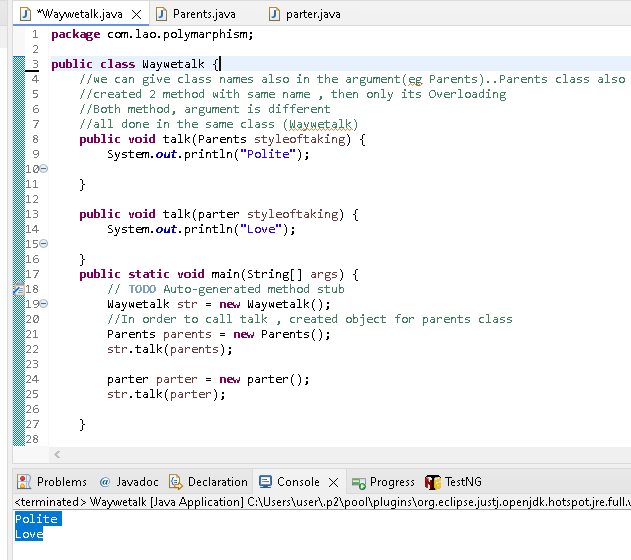
str.talk(parents);

parter parter = **new** parter();

str.talk(parter);

}

}



**Overriding** will happen in parent-child class or inheritance

Method name should be same

No change in anything[Parameters]

Happen at run time

Done in the different class[ parent/child relationship]

**Parent class**

**package** com.lao.polymarphism;

**public** **class** Parents {

**public** **void** property() {

System.***out***.println("My son - use my property");

}

**public** **void** Marriage() {

System.***out***.println("My son - Marry uncle daugter");

}

}

**Child class**

**package** com.lao.polymarphism;

**public** **class** Son **extends** Parents{

//child class ready to use property method & not ready with Marriage method of parent class

//in that we can use override..it will not execute the Marriage method of parent class

//override means->child class is not satisfy with implementation of parent class

//during run time, it will execute the Marriage method in child class eventhough same class name & same parameter

@Override

**public** **void** Marriage() {

System.***out***.println("My Marriage My rule");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//Parentclass ref = new chilsclass() ->its a dynamic polymorphism

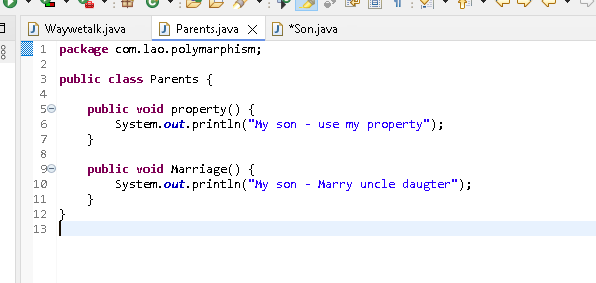
Parents str = **new** Son(); //->its a dynamic polymorphism

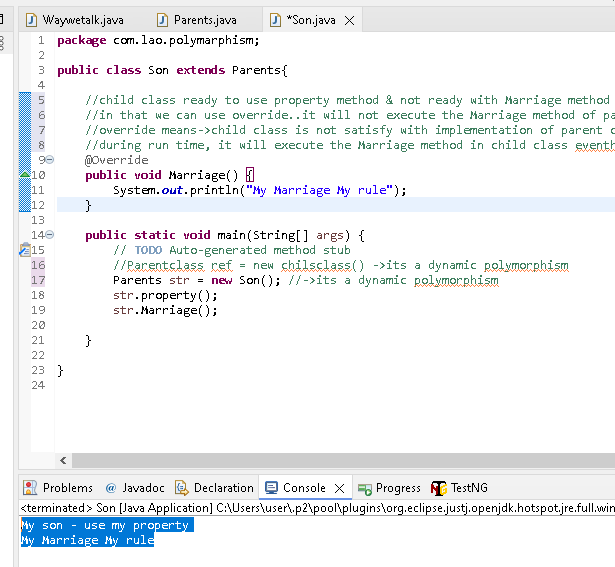
str.property();

str.Marriage();

}

}

****



|  |  |  |
| --- | --- | --- |
| **Aspects** | **Overloading** | **Overriding** |
| **Name of the methods** | Must be same | Must be same |
| **Argument Type** | Must be different | Must be same |
| **Return Type(Void, string)** | can be anything | Must be same |
| **Access Modifier(Public, private, prtected)** | can be anything | Must be same or greater modifier |
| **Throwing Exception** | can be anything | Parents should throw the same or greater exception |
| **Private Methods** | can be overloaded | Cant be overridden |

**ABSTRACTION**: Hiding the implementation details

How can we achieve **ABSTRACTION?**

1.Abstract class(0 to 100% ABSTRACTION)

2.Interface(100% ABSTRACTION)

**Abstract class**

1. If a class is having an unimplemented method(hiding the details), then the method should be declared abstract
2. If a class is having one abstract method then class should be declared as abstract
3. Class may or may not have implementation method(eg one method with hiding the details & other method without hiding the details also allow] **[That why 0 to 100%]**
4. And also we can declare the class as abstract, even if doesn’t have any abstract method
5. Can we create object for abstract class? Nooooooooooooooooo
6. If a child does not implement all the abstract method of parent class, then the child class must need to be declared abstract as well

**Parent Class**

**package** com.lao.abstraction;

//2.If a class is having one abstract method then class should be declared as abstract

**public** **abstract** **class** Car {

//1.below method Body {} not present, hiding the details(unimplemented method), then the method should be declared abstract

**public** **abstract** **void** enginesecret();

**public** **abstract** **void** Comapanyvalut();

//3.Class may or may not have implementation method(eg one method with hiding the details & other method without hiding the details also allow]

//below method showing the details(implementation method)

**public** **void** carrevenue() {

System.***out***.println("Car revenue details");

}

//5.we cant create object for abstract class

//Car car new Car();

}

**Child class**

**package** com.lao.abstraction;

**public** **class** BMW **extends** Car{

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Car bmw = **new** BMW();

bmw.enginesecret();

bmw.Comapanyvalut();

}

//6.If a child(BMW) does not implement all the abstract method of parent class,

//then the child class must need to be declared abstract as well

//Here we implement the abstract method(enginesecret() & Comapanyvalut()) of parent class

//if we cant implement the abstract method(enginesecret() & Comapanyvalut()) of parent class, it will throw error

@Override

**public** **void** enginesecret() {

// **TODO** Auto-generated method stub

System.***out***.println("BMW engine secret");

}

@Override

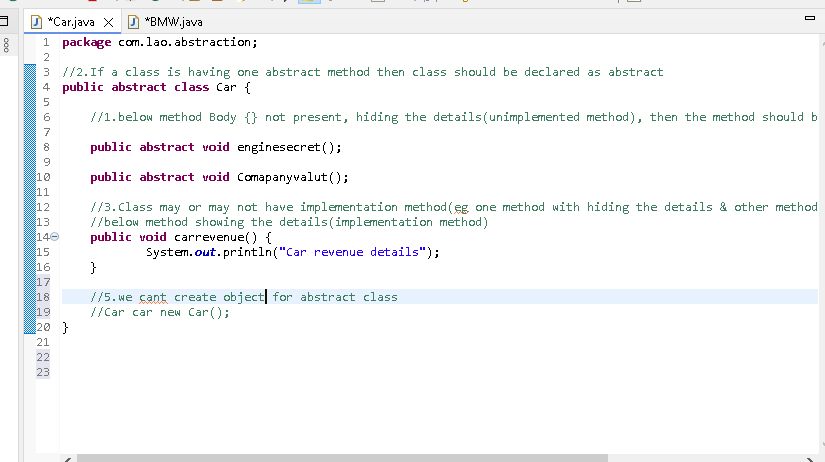
**public** **void** Comapanyvalut() {

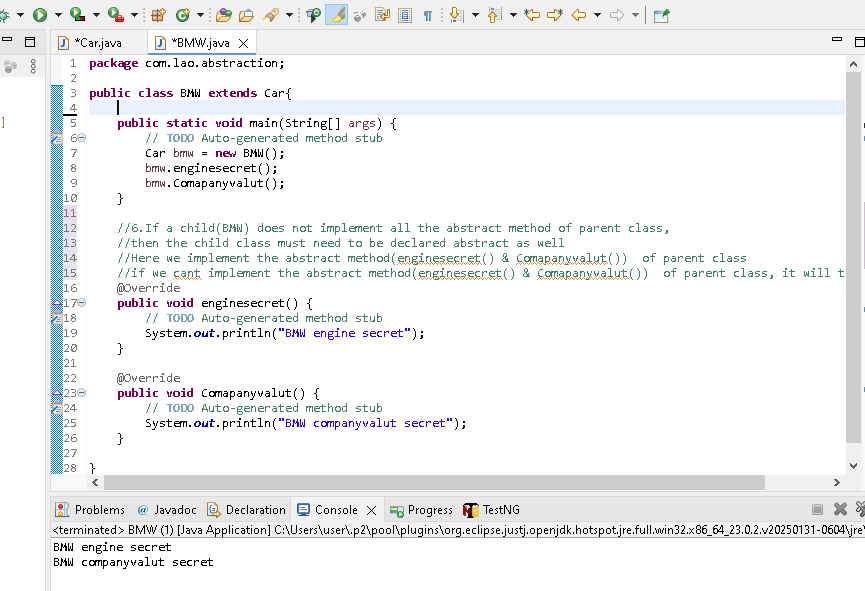
// **TODO** Auto-generated method stub

System.***out***.println("BMW companyvalut secret");

}

}





**Interface**

What is an interface?

Like a class but it is not a class

**Why like a class but Not a CLASS?**

An interface can have methods and variables just like the class.

But The variables are always public, static & final by default. [i.e In Interface cant use protected, private & in class we can use public,protected]

The Methods declared in interface are always abstract by default [ only abstract methods allowed in interface, whereas in class both abstract methods & normal methods allowed]

**Why interface is needed?**

To achieve absolute abstraction[100%] and multiple inheritance[A+B ->C]

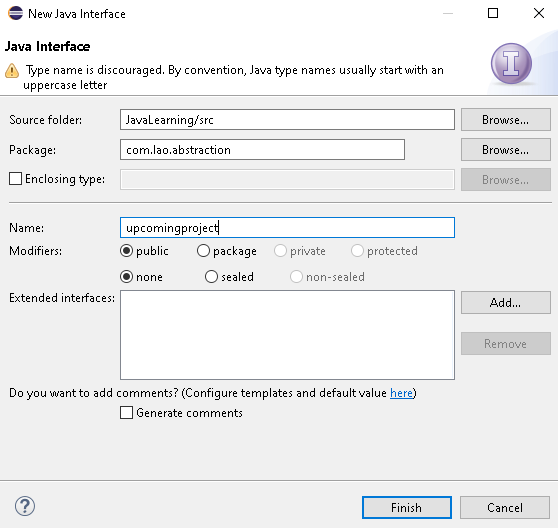
**How a class can use an interface?**

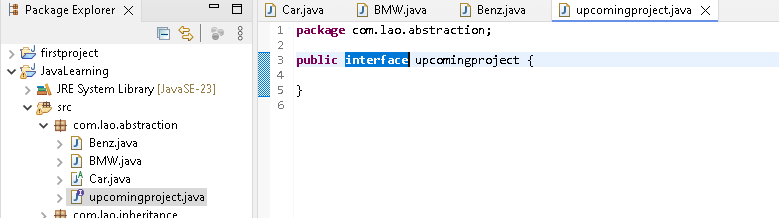
Using implement keyword

**How interfaces use another interface?**

By using extends keyword

**File->New->Interface**

****

****

**Interface**

**package** com.lao.abstraction;

**public** **interface** upcomingproject{

//The variables are always public, static & final(final means need to assign value & its final across] by default

//we have not mention public, static here..by default it will public, static & final..cant see with naked eye\

//public static String upcomingpoject = "new musis";

String ***upcomingpoject*** = "new musis";

//The Methods declared in interface are always abstract & public by default

//we have not mention public, abstract keyword here..by default it will public,abstract l..cant see with naked eye

//Public abstract void employee1();

//all methods should be in abstract in interface..that why 100% abstraction archived in interface & we cant create object for interface

**void** Methods1();

**void** Methods2();

}

**Benz class implements upcomingproject interface**

**package** com.lao.abstraction;

//How a class can use an interface?

//Using implement keyword

//Multiple inheritance archived thru interface as below (class Benz extends Car implements upcomingproject, Interface2 or class Benz implements upcomingproject, Interface2)

//in java [public class Benz extends Car, BMW] - not support will throw error.. Multiple inheritance archived thru interface only

**public** **class** Benz **extends** Car **implements** upcomingproject {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Car benz = **new** Benz();

benz.enginesecret();

benz.Comapanyvalut();

}

//If a child(Benz) does not implement all the abstract method of parent class (here car & upcomingproject(interface),

//Here we implement the abstract method(enginesecret() & Comapanyvalut()) of parent class "Car"

//also //Here we implement the abstract method(employee() & employee1()) of parent class upcomingproject(interface)

//if we cant implement the abstract method(enginesecret() & Comapanyvalut()) of parent class, it will throw error

//below for Car class abstract methods

@Override

**public** **void** enginesecret() {

// **TODO** Auto-generated method stub

System.***out***.println("Benz engine secret");

}

@Override

**public** **void** Comapanyvalut() {

// **TODO** Auto-generated method stub

System.***out***.println("Benz companyvalut secret");

}

//below for upcomingproject interface abstract methods

@Override

**public** **void** Methods1() {

// **TODO** Auto-generated method stub

}

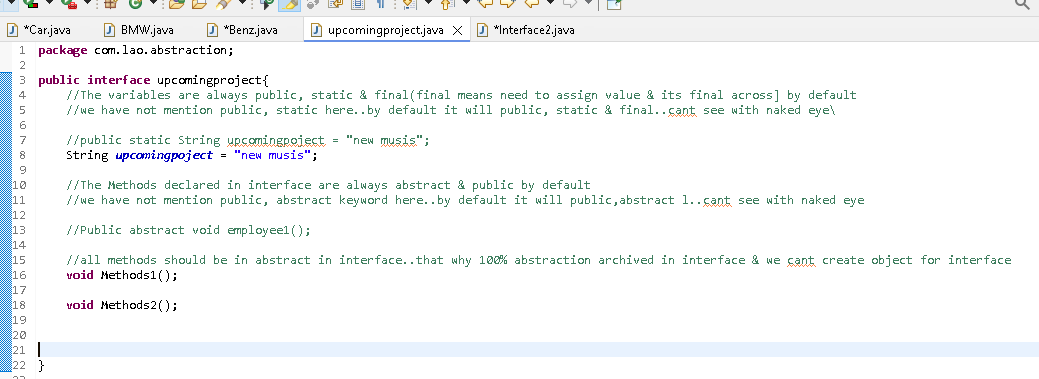
@Override

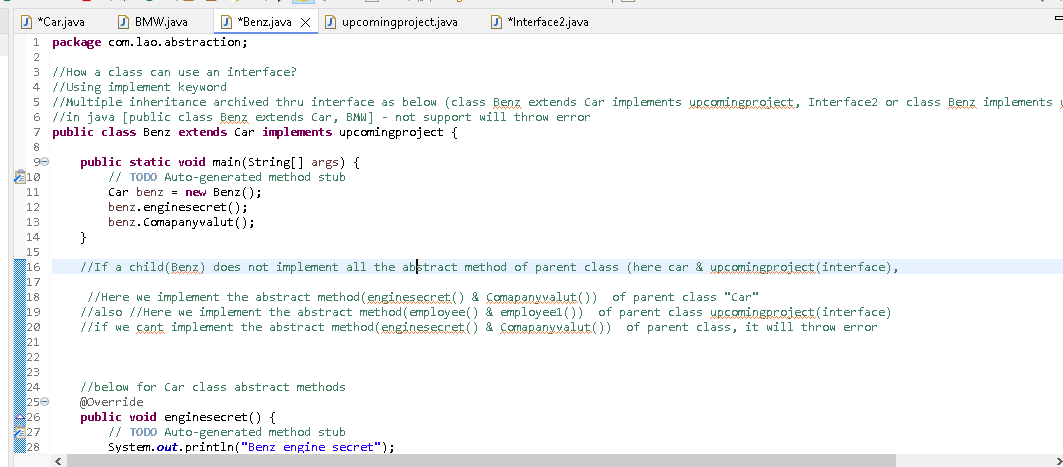
**public** **void** Methods2() {

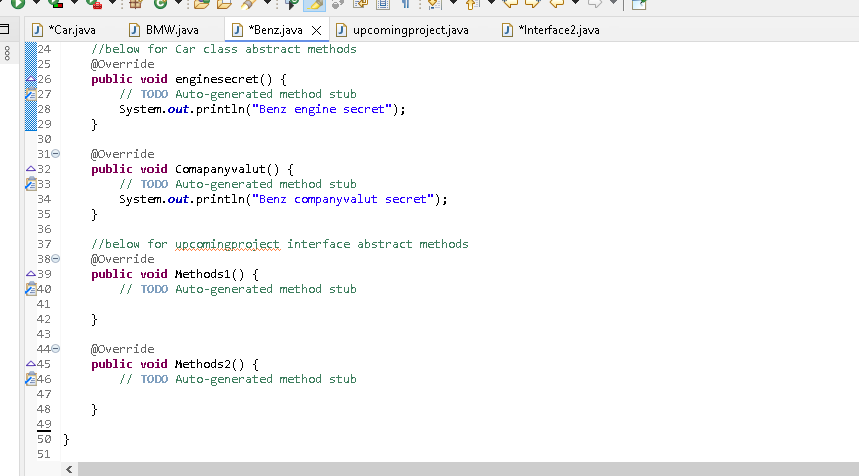
// **TODO** Auto-generated method stub

}

}

****

****

****

**Marker Interface**

**An empty interface is called as marker interface**

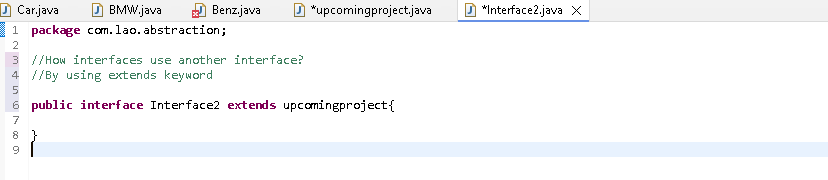
**package** com.lao.abstraction;

//How interfaces use another interface?

//By using extends keyword

**public** **interface** Interface2 **extends** upcomingproject{

}

****

**Encapsualtion:**

Process of binding data and methods together into a single unit(Single unit is class..bec in class only having data and methods together) so class is a encapsualtion.

**Java Strings**

An object that represents sequence of char values [Eg, “Agni is a string of 4 char]

In Java, string is an **immutable object(eg below)** which means it is constant & cannot be changed once it has been created

String str = “Kiru”

Str = “Suku”

Print str

Result: Suku..that means Kiru wont remove always available in the garbage memory..so its a **immutable object**

The Java.lang.String class implement serializable, comparable & charsequence interfaces.

**How to create a string**

1. By string literal

Java string literal is created by using double quotes

String string1 =”Agni”

String string2 =”Agni”

In Heap memory, Agni will create one only & map to string1 & string2..so memory use less in string literal. Whenever value is same it will map to the respective string

String 1

Agni

String 2

2.By using keyword

String string1 = new String(“Agni”)

String string1 = new String(“Agni”)

Here In Heap memory, Agni will create twice even though the value is same bec of using new to create a string. so memory use high in keyword.

String1

String2

**String Method->**No of inbuild string method(eg get string length) available in java & stored in Java.lang.String class

The Java.lang.String class implement serializable, comparable & charsequence interfaces

String is child class of charsequence interfaces

**PFB the list of inbuilt string method**

**package** com.lao.string;

**public** **class** Javaexamples {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String name = "Agniprasath";

**int** number = 3;

System.***out***.println(name.charAt(1));

System.***out***.println(name.charAt(1));

System.***out***.println(name.length());

System.***out***.println(name.equals("Arul"));

System.***out***.println(name.equalsIgnoreCase("AGNIPRASATH"));

System.***out***.println(name.isEmpty());

System.***out***.println(name.contains("A"));

System.***out***.println(name.substring(1));

//Beginning index(1) included & end index(6) not included

System.***out***.println(name.substring(1,3));

System.***out***.println(name.concat("Arulprasath"));

System.***out***.println(name.replace("g", "G"));

System.***out***.println(name.replace("Agni", "Riya"));

System.***out***.println(name.indexOf("A"));

System.***out***.println(name.indexOf("a",7));

System.***out***.println(name.indexOf("i",2));

System.***out***.println(name.indexOf("sath",1));

System.***out***.println(name.trim());

//Convert the give data type to string

System.***out***.println(String.*valueOf*(number));

String uppercase="SUKU";

System.***out***.println(uppercase.toLowerCase());

String lowercase="suku";

System.***out***.println(lowercase.toUpperCase());

//Returns a joined string with given delimiter

System.***out***.println(String.*join*("-", "Learn","Automation","Online"));

System.***out***.println(String.*join*("/", "30","05","1990"));

//split

String str= "Am, I, teaching";

String[] str1 = str.split(",");

**for** (String string : str1) {

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

}

}

}

**StringBuffer**

1.StringBufferis also a child class of charsequence interfaces

2.Its a mutable object [opposite to a Immutable object as explained above]

3.Its a synchronised (Thread safe)..That is multiple thread cannot access it simultaneously]

**How to create a StringBuffer object**

1. By using new keyword

StringBuffer s = new StringBuffer(“Agni”);

String literal not work in Stringbuffer

**package** com.lao.string;

**public** **class** Stringbuffer {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//String ->IMMAUTABLE EXAMPLES

String name ="Suku";

System.***out***.println(name.concat("mar"));

//after concat , name displayed as "Suku"(Not changed, so its a ImMAUTABLE object)

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

//StringBUFFER ->MAUTABLE EXAMPLES

StringBuffer name1 = **new** StringBuffer("Suku");

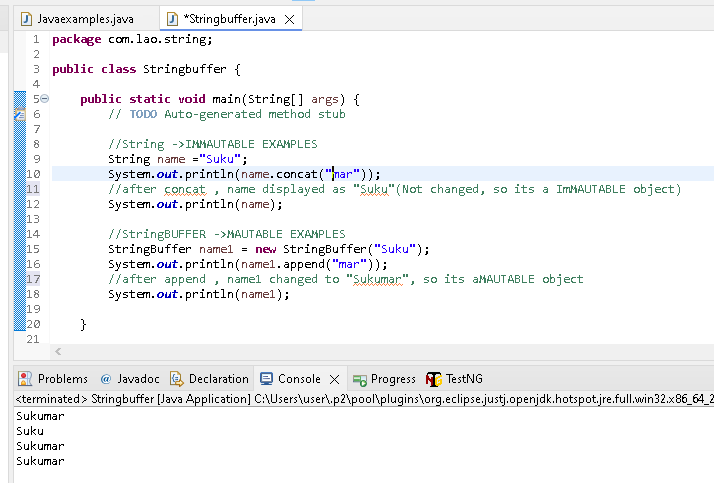
System.***out***.println(name1.append("mar"));

//after append , name1 changed to "Sukumar", so its aMAUTABLE object

System.***out***.println(name1);

}

}



**StringBuffer methods [less only when compare with String method]**

//------Stringbuffer methods

//Reserve--in string we cant use reserve method

StringBuffer name2 = **new** StringBuffer("Kirubha");

System.***out***.println(name2.reverse());

//Replace

System.***out***.println(name2.replace(0, 3, "Riya"));

//delete

System.***out***.println(name2.delete(0, 3));

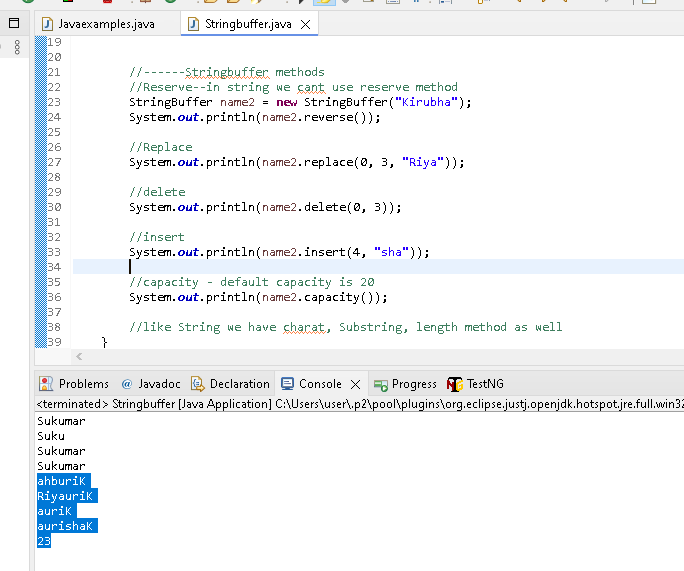
//insert

System.***out***.println(name2.insert(4, "sha"));

//capacity - default capacity is 20

System.***out***.println(name2.capacity());

//like String we have charat, Substring, length method as well

****

**Stringbuilder**

1.Stringbuilderis also a child class of charsequence interfaces

2.Its a mutable object

3.Its a Non synchronised (Not Thread safe)..That is multiple thread can access it simultaneously]..Because of this more efficient then StringBuffer

**Stringbuilder Method: Same as StringBuffer methods[Refer above]**

**Exception handing**

**What is an exception?**

An exception is an undesirables or unexpected event, which occur during the execution of a program.(At run time), that disrupts the normal flow of the program instruction.It can be handled by our program.

**An exception is NOT an ERROR.**

An error is very severe problem, which can’t be handled by the program. Eg when your computer runs out of space, there is nothing our program can do except crashing.

**Type of exception:**

1.Checked exception - Occur at compile time..it will be corrected then only we are able to running the code..(eg syntax error etc)..in ellipse will show the error with red underline, easily corrected

2. Unchecked exception - Occur at run time..during code run time, java will throw the Unchecked exception.[eg Null pointer exception, arithmetic exception]

Unchecked exception – Handle thru Try & catch

**In Try ->** we need to find & write code what are possible exception(unexpected behaviour) will occur during run time based on our code

**Try proper definition:** This a a block where we will write our code which may give run time exception

**Catch**->if the exception occur, how to handle those exception..those code will write in Catch block

**Catch proper definition:** This is where the exceptions are handled. Catch block is immediately followed by try block

Try{

} catch(){

}

**Important point:**

1. A Try block can have multiple catch block
2. If there are multiple catch blocks, they should not be of genetic Exception Type(like Arithmetic Exception use one time, all exception be different in multiple catch)

Try{

}

**catch** (ArithmeticException e) {

}

**catch** (NullPointerException e) {

}

1. If there are no exceptions then the catch blocks will not get executed
2. catch (Exception e) Exception parent of all exception , under this only child excpetion like Arithmeticexception(number), nullpointer exception(object not present) etc present
3. //always use this catch (Exception e) Exception at the last in catch, if u use at first then other exception wont execute..as its a parent it have all type of child exception to handle

**package** Exceptionexamples;

**public** **class** Trycatch {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

**int** a = 10;

**int** b= 0;

**int** c = a/b;

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

}

//A Try block can have multiple catch block..developer analyze and have multiple catch based on the need

**catch** (ArithmeticException e) {

System.***out***.println("Number exception");

}

**catch** (NullPointerException e) {

System.***out***.println("Object exception");

}

//Exception parent of all exception , under this only Arithmeticexception(Mumber), nullpointer exception(objectnot present) etc present

//always use this Exception at the last in catch, if u use at first then below exception wont execute..as its a parent it have all type of child exception to handle

//always write child handle first, then parent exception at last

**catch**(Exception e) {

//printStackTrace will show what are the jave exception occur during run time

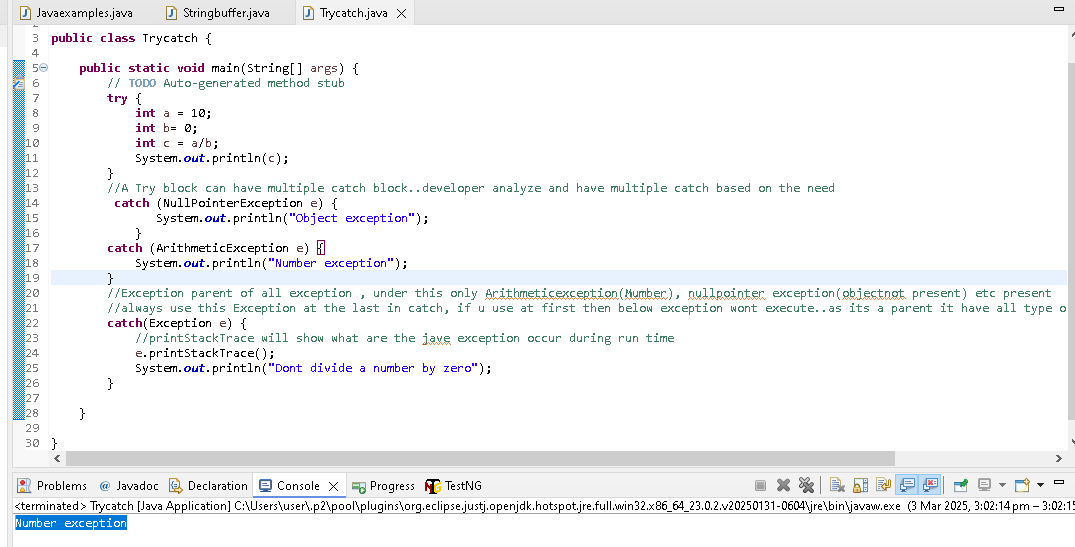
e.printStackTrace();

System.***out***.println("Dont divide a number by zero");

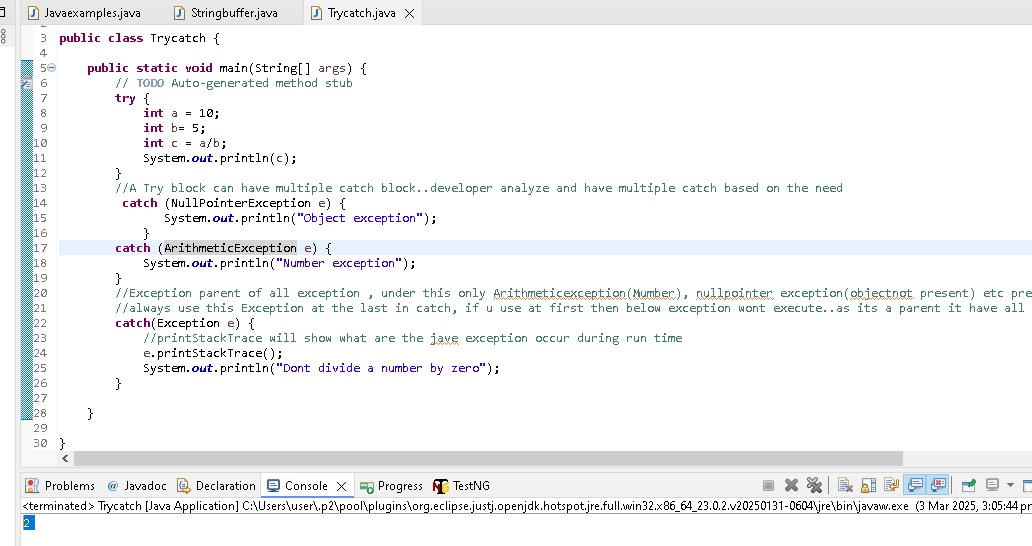
}

}

}



Eg If there are no exceptions then the catch blocks will not get executed[refer below]

****

**Finally block in Try catch:**

This is a block of code which will get executed even if the exception occur or not. In real time, This block will contains codes like closing connection of DBs, buffers, stresms etc.

**package** Exceptionexamples;

**public** **class** Tryctachfinally {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

System.***out***.println("Inside Try block");

}

//no exception, so it wont execute Catch block..but finally always execute

**catch**(Exception e) {

System.***out***.println("Inside Catch block");

}

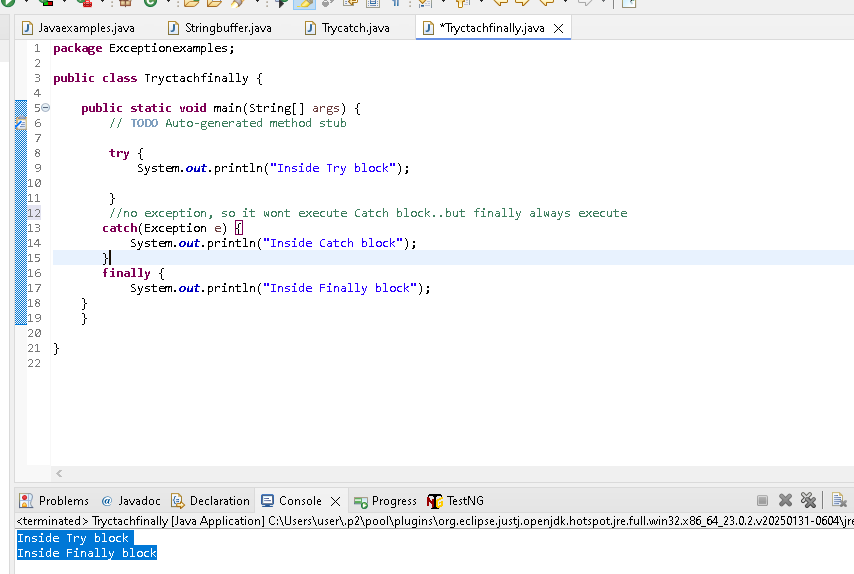
**finally** {

System.***out***.println("Inside Finally block");

}

}

}



**Important points:**

1.There is no finally block without a Try-catch. A finally block must be associated with a try catch

2.Fianlly block is not mandatory

3.Fianlly block also can have exception..in that we can use try & catch in finally block

4.The statement present in the finally block execute even if the try block contains control transfer statement like break,return

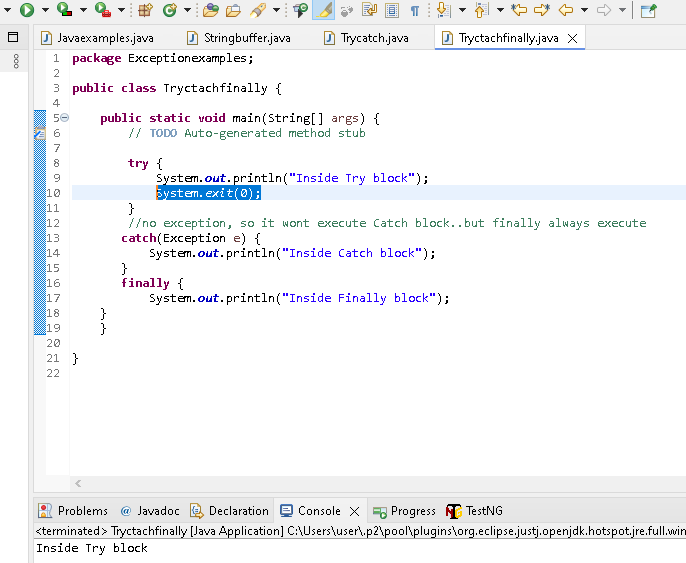
**Will Finally block always execute?**

Nope...Finally will not get executed if

1. the thread is dead [**public** **static** **void** main(String[] args) its a main thread]

2.when system.exit() is called

3.when an unrecoverable exception happens in finally block



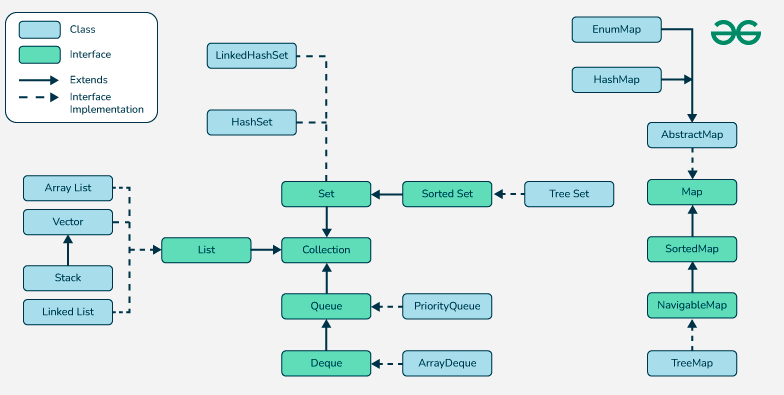
**Java collection**

A library is a collection of books

A school is a collection of students

So can we define the collection is a group of individual objects.?

Yes..that will be perfect definition.



**Collection** [InterFace]..child interfaces below..all used to store the individual objects with different functionality in it.

1.List [InterFace]

2.Set [InterFace]

3.Queue [InterFace] [Not imp & so not covered ]

4.Map(Interface) ->it wil not comes under collection..its also have same functionlonlity(storing group of individual objects.)

**List [Interface] ->child class below**

1.Arraylist [Class]

2.Linkedlist [Class]

3.vector [Class]

4.Stack [Class]

**Note**: vector & stack its old list operation..Nowadays mostly Arraylist & Linkedlist used

**1.Arraylist**

Arraylist will accept duplicate values(we can same values multiple times)

Arraylist wil maintain insert order[they way we adding..it will display same way]

Arraylist is non – synchronized.. in below eg we are reading the array list value thru for loop in that only we are adding a new value to the list..java will throw the ConcurrentModificationException as its non – synchronized(multiple operation at same time not allowed)

List<String> str1 = **new** ArrayList<String>();

str1.add("Suku");

str1.add("Suku");

str1.add("Hari");

str1.add("Jothi");

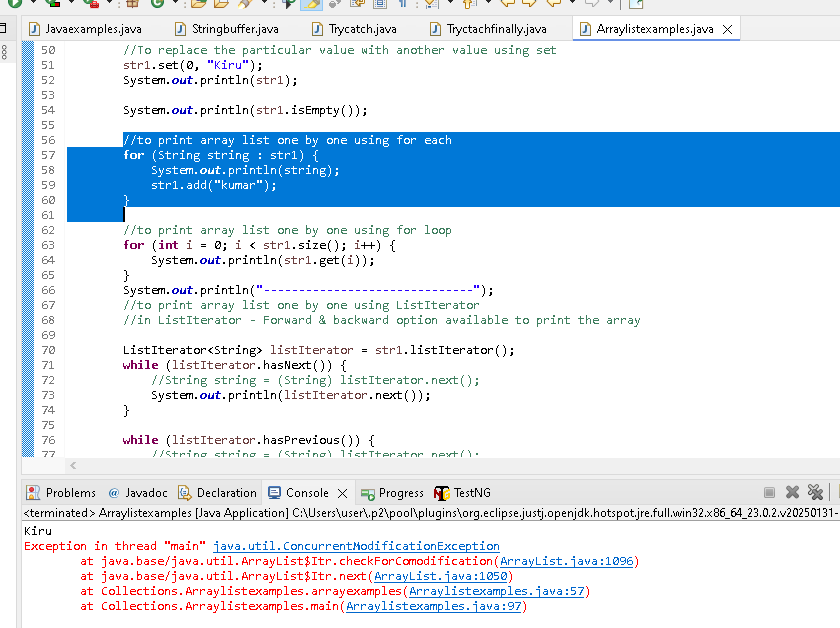
//to print array list one by one using for each

**for** (String string : str1) {

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

str1.add("kumar");

}



**Important Points:**

Arraylist(c) is one of the classes provides implementation for the list(I)

Duplicate values allowed

Insertion order is maintained

Underlying DS(Data structure) is resizable Array or Growable Array[ Based on the size(need), we can add/delete the elements in the arraylist]

Can insert heterogeneous object(if generics are not used) [ we can add both number & string if if generics are not used..best practice always use generics]

List str1 = **new** ArrayList ();

str1.add("1");

str1.add("Suku");

**Generics ->only accept string**

List<String> str1 = **new** ArrayList<String>();

Arraylist is not synchronised

Its is not thread safe

Default capacity- 10

Fill radio or load factor : 1 to 100% [ in the 10 capacity we can load full as per need, once max size 10 is filled, next growth rate as below]]

Growth rate: current\_size + current\_size/2

List str1 = **new** ArrayList ();->that means Default capacity- 10

Growth rate = 10 + 10/2 =>15

Implements RandomAccess, Serializable & Cloneables interfaces (Cloneables – copy the arraylist to another arraylist))

Best suited for search operation[bec of RandomAccess] and performs , worst on insertion and deletion

Because if you insert any value inbetween then remaining all array position should move to the right accordingly..same way if we delete any array , then remaining array position should move accordingly

**package** Collections;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.ListIterator;

**public** **class** Arraylistexamples {

**public** **void** arrayexamples() {

//below line -its dynamic polymorphism {parent = new child() reference)

//<String> its called as generic..we can add only string value to the list

List<String> str1 = **new** ArrayList<String>();

//Arraylist will accept duplicate values(we can same values multiple times)

//Arraylist wil maintain insert order[they way we adding..it will display same way]

//add used to add the stringvalues to the array list

str1.add("Suku");

str1.add("Suku");

str1.add("Hari");

str1.add("Jothi");

System.***out***.println(str1);

//get to read the stored values in the array

System.***out***.println(str1.get(0));

//indexOf to get the position of the string

System.***out***.println(str1.indexOf("Hari"));

//particlular string not present, it will return "-1"

System.***out***.println(str1.indexOf("Kirubha"));

//copy the arraylist to another arraylist bu using addall

List<String> str2 = **new** ArrayList<String>();

str2.addAll(str1);

System.***out***.println(str2);

//remove the value based on index or direct string by using remove

str2.remove(0);

System.***out***.println(str2);

str2.remove("Hari");

System.***out***.println(str2);

//to clear all by using clear

str2.clear();

System.***out***.println(str2);

//we can add null string in array list

str1.add(**null**);

System.***out***.println(str1);

//To replace the particular value with another value using set

str1.set(0, "Kiru");

System.***out***.println(str1);

System.***out***.println(str1.isEmpty());

//to print array list one by one using for each

**for** (String string : str1) {

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

//str1.add("kumar");

}

//to print array list one by one using for loop

**for** (**int** i = 0; i < str1.size(); i++) {

System.***out***.println(str1.get(i));

}

System.***out***.println("------------------------------");

//to print array list one by one using ListIterator

//in ListIterator - Forward & backward option available to print the array

ListIterator<String> listIterator = str1.listIterator();

**while** (listIterator.hasNext()) {

//String string = (String) listIterator.next();

System.***out***.println(listIterator.next());

}

**while** (listIterator.hasPrevious()) {

//String string = (String) listIterator.next();

System.***out***.println(listIterator.previous());

}

System.***out***.println("------------------------------");

//to print array list one by one using Iterator

//in Iterator - only Forward option available to print the array

Iterator<String> iterator = str1.iterator();

**while** (iterator.hasNext()) {

//String string = (//String) iterator.next();

System.***out***.println(iterator.next());

}

System.***out***.println("------------------------------");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Arraylistexamples str = **new** Arraylistexamples();

str.arrayexamples();

}

}

**LinkedList**

Insertion & Deletion is effective in linked list [only pointer between 2 element changes when new element inserted in between in the linkedlist]

Search operation is worst in linked list [Bec of RandomAccess implement not present in Linkedlist] if we searching if will go first element & next & next..one by one so not effective

Duplicate values allowed

Insertion order is maintained

**package** Collections;

**import** java.util.LinkedList;

**public** **class** Linkedlist {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//add the value to the list

LinkedList<Integer> linkedlist = **new** LinkedList<Integer>();

linkedlist.add(1);

linkedlist.add(5);

linkedlist.add(6);

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

//Add the element to the first position

linkedlist.addFirst(5);

//Add the element to the last position

linkedlist.addLast(10);

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

//Get the firstelement of the list

System.***out***.println(linkedlist.get(0));

//another method Get the firstelement of the list

System.***out***.println(linkedlist.getFirst());

//Get the lastelement of the list

System.***out***.println(linkedlist.getLast());

//REmove the first & last element in the list

System.***out***.println(linkedlist.removeFirst());

System.***out***.println(linkedlist.removeLast());

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

//Poll method & pollfirst & remove ->delete the first element in the list

System.***out***.println(linkedlist.poll());

//polllast delete the last element in the list

System.***out***.println(linkedlist.pollLast());

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

//Add again

linkedlist.add(1);

linkedlist.add(5);

linkedlist.add(6);

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

//removefirstoccurence

System.***out***.println(linkedlist.removeFirstOccurrence(5));

//removelastoccurence

System.***out***.println(linkedlist.removeLastOccurrence(6));

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

//retrieve the values from the arraylist using For each

//by using simple for loop, while loop, iterator also we retrieve the values from the arraylist

**for** (Integer integer : linkedlist) {

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

}

}

}

**Note**: vector & stack its old list operation..Nowadays mostly Arraylist & Linkedlist used

Just high level below for reference

**Stack**

**package** Collections;

**import** java.util.Stack;

**public** **class** Stackexamples {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//push to add

Stack<String> stack = **new** Stack<String> ();

stack.push("A");

stack.push("B");

stack.push("C");

//pop to delete - delete last stored element

stack.pop();

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

//search

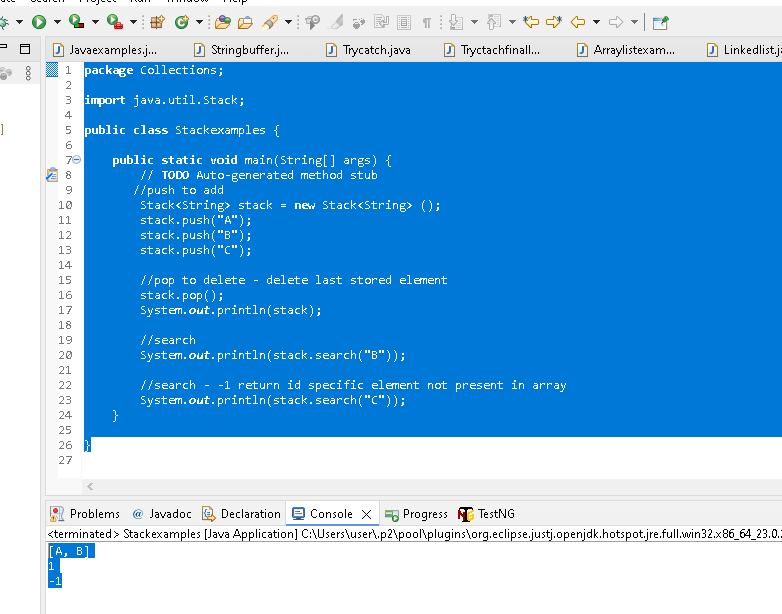
System.***out***.println(stack.search("B"));

//search - -1 return id specific element not present in array

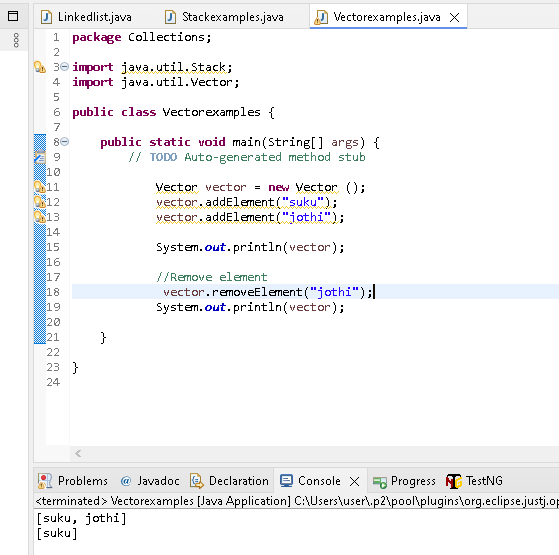
System.***out***.println(stack.search("C"));

}

}

****

**Vector**

****

**Set [Interface] & childclass & child interface below**

1.**Hashset(class) ->child -> Linked Hashset(class)**

2.Sortedset(Interface)->child-> Navigable set(Interface) ->child->**Tree set(class)**

We are going to see only classes - Hashset (C) -> Linked Hashset (C) & Treeset(C)

**Nature of SET:**

1.Set(I) is one of the child interface of collection(I)

2.duplicate values are not allowed

3.insertion order is not maintained

4.Can insert heterogeneous objects(if generics are not used)

**Handset**

1.Hashset(1) is implementation class for SET(I)

2.duplicate values are not allowed

3.insertion order is not maintained

Underlyting DS is hashtable which is actually a hashmap instance

4.Can insert heterogeneous objects(if generics are not used)

5.null element is allowed

6.implements serialization & cloneable interface[cloneable - >we can copy the set to another set]

7.Data are stored based on hashcode[i.e means it will store the values in code format in backend], so search is very effective

8.fill ratio or load factor: 0.75 or 75% [once 75% reached of 16 default capacity, new load created]

9.Default capacity -16

**package** Collections;

**import** java.util.HashSet;

**import** java.util.Iterator;

**public** **class** Hashsetexamples {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

HashSet<String> hashSet = **new** HashSet<String>();

hashSet.add("A");

hashSet.add("A");

hashSet.add("F");

hashSet.add("B");

hashSet.add(**null**);

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

System.***out***.println(hashSet.contains("B"));

//like arraylist..most of the methods available in hashset(clear, remove, search etc)..do handson

//iteration using loop

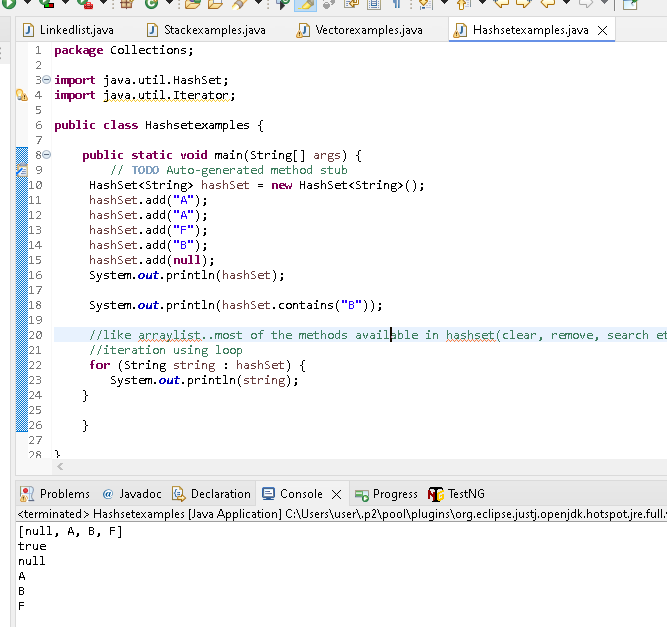
**for** (String string : hashSet) {

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

}

}

}



**LinkedHashSet -> Same as Hashset , only diff is insertion order will be maintained here**

1.LinkedHashset(1) is implementation class for SET(I)

2.duplicate values are not allowed

3.**Insertion order is maintained**

Underlyting DS is **hashtable & linkedlist**

4.Can insert heterogeneous objects(if generics are not used)

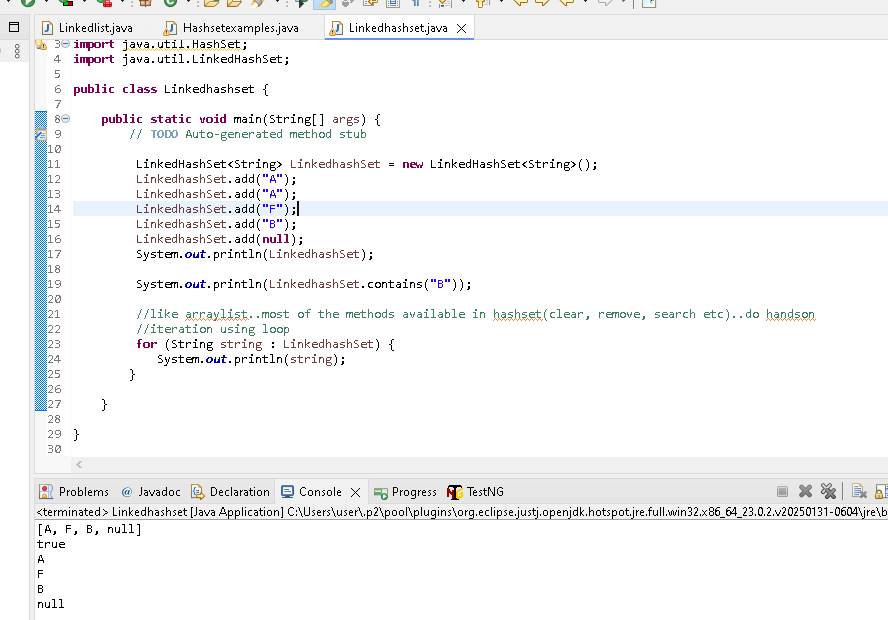
5.null element is allowed

6.implements serialization & cloneable interface[cloneable - >we can copy the set to another set]

7.Data are stored based on hashcode[i.e means it will store the values in code format in backend], so search is very effective

8.fill ratio or load factor: 0.75 or 75% [once 75% reached of 16 default capacity, new load created]

9.Default capacity -16

****

**Treeset**

Treeset(C) is implementation class for Sortedset(Interface),Navigable set(Interface)

Duplicate values & null values are not allowed

Elements wil be retrieved on **Natural sorting order**

Underlying DS is Binary search tree

Only homogeneous

Heterogeneous objects is not allowed[ as points 3 told Natural sorting order, so we give 1 & A how it will compare & sort..so not allowed even if <generic> not provided]

**package** Collections;

**import** java.util.Iterator;

**import** java.util.TreeSet;

**public** **class** Treesetexamples {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

TreeSet<Integer> integers = **new** TreeSet<Integer>();

integers.add(10);

integers.add(2);

integers.add(1);

integers.add(5);

integers.add(9);

integers.add(3);

//

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

//first()

System.***out***.println(integers.first());

//last()

System.***out***.println(integers.last());

//headset()

System.***out***.println(integers.headSet(3));

//tailset()

System.***out***.println(integers.tailSet(5));

//subset

System.***out***.println(integers.subSet(1,9));

//comparator() -> it will return null if the sorting is default natural order

System.***out***.println(integers.comparator());

//immediate higher of 3

System.***out***.println(integers.higher(3));

//immediate lower of 3

System.***out***.println(integers.lower(9));

//display first element & remove also

System.***out***.println(integers.pollFirst());

//display last element & remove also

System.***out***.println(integers.pollLast());

//display after poll

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

//descending order

System.***out***.println(integers.descendingSet());

//loop

Iterator<Integer> iterator = integers.iterator();

**while** (iterator.hasNext()) {

System.***out***.println(iterator.next());

}

}

}

**Map(Interface)**

it wil not comes under collection..its also have same functionlonlity(storing group of individual objects.)

**Map(I)**

1.**HashMap(C) ->LinkedHashmap(C)**

2.Sortedmap(I)->Navigablemap(I)->**Treemap(C)**

We are going to see only classes - HashMap(C) ->LinkedHashmap(C) & Treemap(C)

**Nature of Map:**

Map(I) is not the child interfaces of collection(I) & hence its is not considered to be a true collection

It represents data in Key-Value pair

Duplicate Key are not allowed in map

Duplicate Values are allowed

Eg

Employee No(Key) -> Employee name(Value)

Pin code(key) -> city(Name)

**HashMap**

Hashmap(C) is implementation class for Map(I)

Duplicate keys are not allowed & Duplicate Values are allowed

Only one null key is allowed[ as Duplicate keys are not allowed, 2nd null will be duplicate but it will override with latest value..No null point exception]

Uses a technique called hashing to store elements

The order is not preserved as ist uses hashing. So we cant predict the retrieval order of the element inserted

Value of the map can only be fetched using its key

Initial capacity: 16 & load factor:.75

It is not synchronised.so if it has to be used on multithreading env, it has to be synchronised explicity

**package** Collections;

**import** java.util.HashMap;

**public** **class** Hashmapexamples {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

HashMap<Integer, String> hashMap = **new** HashMap<Integer, String>();

hashMap.put(1, "Suku");

hashMap.put(2, "Jothi");

hashMap.put(5, "Babu");

hashMap.put(3, "Kiru");

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

//Copy one map to another

HashMap<Integer, String> hashMap1 = **new** HashMap<Integer, String>();

hashMap1.putAll(hashMap);

System.***out***.println(hashMap1);

//to check the key is present in the map

System.***out***.println(hashMap.containsKey(2));

//to check the value is present in the map

System.***out***.println(hashMap.containsValue("Suku"));

//clone the map without using put all

System.***out***.println(hashMap.clone());

//check map is empty or not

System.***out***.println(hashMap.isEmpty());

//Fetch the set of key only

System.***out***.println(hashMap.keySet());

//Fetch the set of values only

System.***out***.println(hashMap.values());

//Fetch the particular values

System.***out***.println(hashMap.get(2));

//fetch the entry set

System.***out***.println(hashMap.entrySet());

}

}

**LinkedHashmap(C) ->**same as hashmap, only diff is in LinkedHashmap(C) insertion order is maintained

Hashmap(C) is implementation class for Map(I)

Duplicate keys are not allowed & Duplicate Values are allowed

Only one null key is allowed[ as Duplicate keys are not allowed, 2nd null will be duplicate, but it will override with latest value..No null point exception]

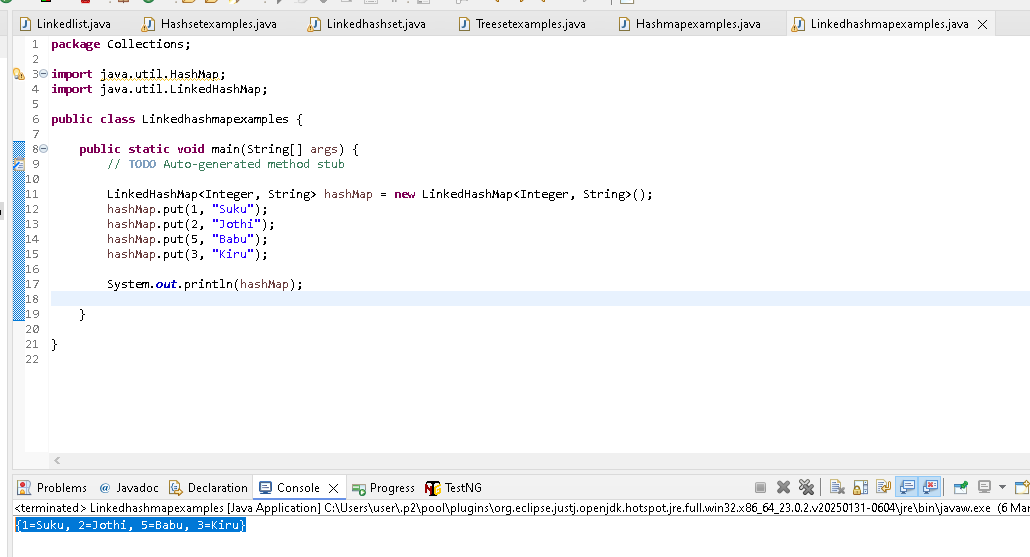
Uses a technique called hashing to store elements

**The order is preserved**

Value of the map can only be fetched using its key

Initial capacity: 16 & load factor:.75

It is not synchronised.so if it has to be used on multithreading env, it has to be synchronised explicity



**TreeMap**

Treemap(C) is implementation class for Map(I)

Duplicate keys are not allowed & Duplicate Values are allowed

NULL key is not allowed [if Null value added then how it will compare with other values & get sorted, so null insertion is not allowed] & throw No null point exception

The elements will be retrieved on natural sorting order.

