OOPS2

Lec 1.

Components of OOPS

There are 3 major components of OOPS

- Encapsulation
  - ↓ Abstraction
- Inheritance
- Polymorphism

1 ENCAPSULATION

Encapsulation simply means encapsulating things.

Putting things in 1 capsule like structure.

* Wrapping Data and functions together in a Class.

```
public class Student
{   int name;
    int marks.
    private int rollno;

    public int getrollno()
    {  return rollno;
    }
}
```

This class has this data/var

as well as these functions

So, we are encapsulating data and functions in 1 capsule/class

So, this is what Encapsulation is.

↓ getter func

* Another component that lies under ENCAPSULATION is **Abstraction** (some people consider it as an all different component)

Abstraction → You are not revealing all the details to someone outside.

eg→ We have Scanner class using which we generally take input.

We create an object named → scan or s or scn, etc

```
Scanner scan = new Scanner ( System.in );
```

scan → Using this object, we get the access to many functions of Scanner class

System.in → Source

eg→
```
scan.nextInt()
scan.next()
scan.nextLine()
```
We don't know how these function are actually working but we still use them.

The thing that we are really concerned about is what input is this function expecting from me and what kind of output is this function giving me.

* So, the details about how this function is working is abstract from me.

& I as a user is not diving deep into its working. So, this is what ABSTRACTION is about

* for eg1:→ We use Breaks in Car.
We don't really care about how it works.
We don't really need the details as a user.

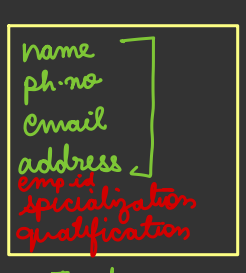
for eg2. we use math.pow()
Integer.MAX_VALUE()
Integer.MIN_VALUE()
} without worrying how these func actually work internally.

2. INHERITANCE

Someone Inheriting properties & functionalities somebody (Generally children inherit properties from their parent)

If I create a class Student and a Teacher class. There are certain set of properties that will be common in both these classes for eg → name
→ ph.no
→ email id
→ address

If both teachers & students have some common set of properties

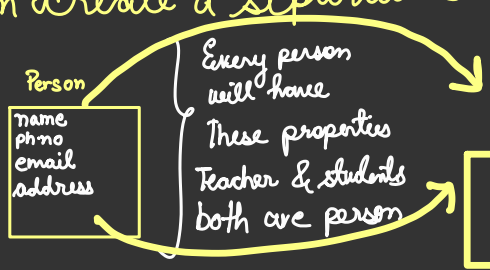


Teacher

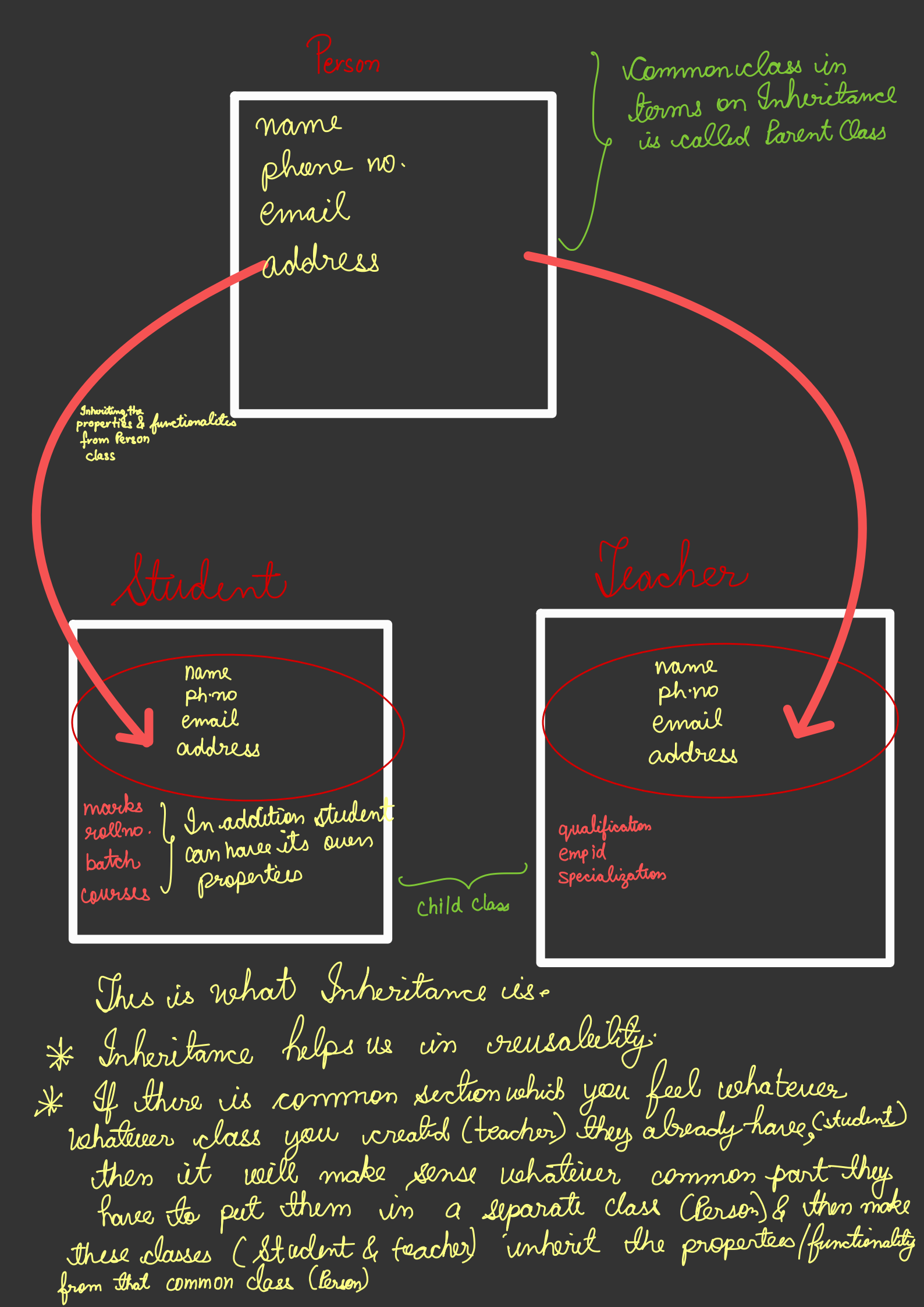
Student

} We are doing redundant (same) work here again by creating same properties in 2 class

So, we can create a separate class Person for all properties in common

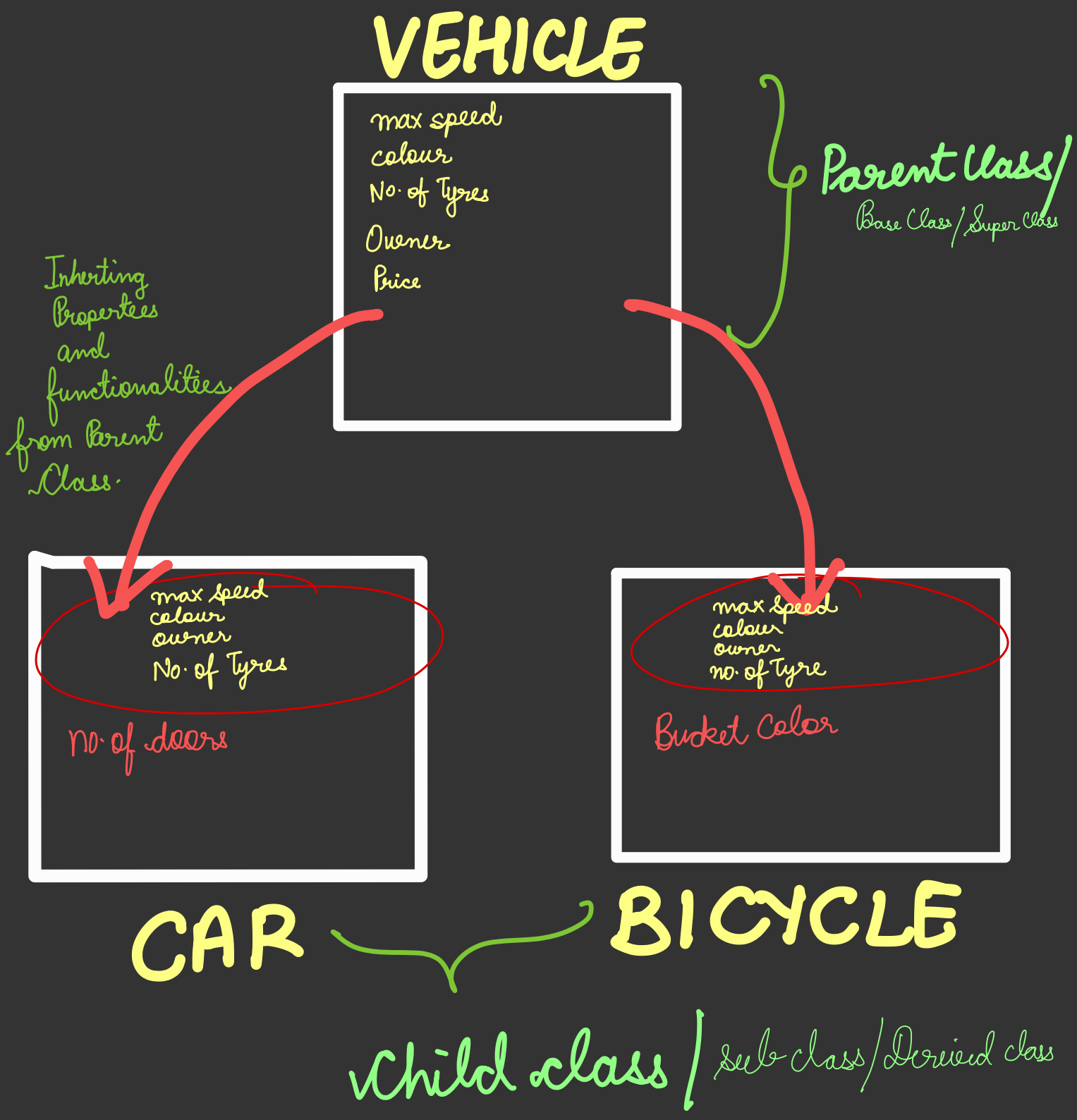


} Teacher and Student are going to inherit these properties from Person Class because they both are Person



This is what Inheritance is.

* Inheritance helps us in reusability.
* If there is common section which you feel whatever whatever class you created (teacher) they already have, (student) then it will make sense whatever common part they have to put them in a separate class (Person) & then make these classes (Student & teacher) inherit the properties/functionality from that common class (Person)


VEHICLE
max speed
colour
No. of Tyres
Owner
Price
Parent Class/
Base Class / Super Class
Inheriting Properties and functionalities from Parent Class.
max speed
colour
owner
No. of Tyres
No. of doors
max speed
colour
owner
no. of Tyre
Bucket Color
CAR
BICYCLE
Child class / sub class / Derived class

Lec 2. Writing code using Inheritance

1. Inheritance helps us in establishing that parent-child relationship where child class inherit the properties and functionalities from the parent class.
2. Inheritance helps us in reusability of code and giving a proper structure to our code.

```
package inheritance;
public class Vehicle
{
  String color;
  int maxspeed;
  public void print()
  { System.out.println
  ("Vehicle "+ "color "+
   color + "maxSpeed "+
   maxSpeed);
```

Extends means we are inheriting the properties & function of Vehicle.

```
package inheritance;
public class Car extends Vehicle
{
```

```
package inheritance;
public class Bicycle extends Vehicle
{
```

```
package inheritance;
public class VehicleUse
{ public static void main(String args[])
  {
    Vehicle v = new Vehicle();
    v.color = "red";
    v.print();
  Car c = new Car();
  c.color = "Black";
  c.maxSpeed = 100;
  c.numDoors = 4;
  c.print()
Bicycle b = new Bicycle();
b.print();
  }
}
```

→ Vehicle color Red maxspeed 0 (0 ↓ default int value)

→ Vehicle color Black maxSpeed 100

→ Vehicle color null maxSpeed 0

Lec 3.

INHERITANCE AND PRIVATE MEMBERS

```
package inheritance;

public class Vehicle
{ private String color;
  int maxspeed;
  public void print()
  { System.out.println
    ("Vehicle color"+color+"maxSpeed"+maxspeed);
  }
  private String getColor()
  { return color;}
  private String setcolor()
  { this.color=color;}

}
```

Using getters & setters we can access private variables.

Inherit properties & Functionalities

Inherit properties & functionalities

Car

```
package inheritanc;
public class Car extends Vehicle {
 int numDoors;
public void printCar()
 { System.out.println
   ("Car color"+getcolor()+
     "maxspeed"+maxspeed+
     "numdoor"+numDoors);
 }
```

BiCycle

```
package inheritanc;
public class BiCycle extends Vehicle {

 }
```

```
package inheritance;
public class VehicleUse{
public static void main(String[]args){
   Vehicle v= new Vehicle();
    v.setColor("red");
    v.maxSpeed=80;
    v.print();

  Car c=new Car();
  c.setcolor("Black");
  c.maxSpeed=100;
  c.numDoors=4;
  c.printCar();

  }
 }
```

4. FUNCTION OVERLOADING & FUNCTION OVERRIDING

FUNCTION OVERLOADING

1. In function overloading, we declare more than 1 function with same name and different types of parameters.

   Eg → public int sum();
   public int sum(int x);

2. It can occur without inheritance
3. In function overloading, there must be different signatures of function.

   eg → → one doesn't have parameters.
   → another has parameters
   → another has many parameters
   → another has different return type
   etc.

4) It is the concept of compile-time polymorphism

5) In function Overloading, every function behave in a diff. way depending on the parameters passed

This concept is called as Function Overriding

FUNCTION OVERRIDING

1. In function overriding, we declare a function in the base class as well as in the derived class with the same name and the same parameters

eg →

```
In vehicle class
  └→ public void print()
     {
     }
```

```
In Car class (car extends vehicle)
  └→ public void print()
     {
     }
```

```
In main func
  └→ Car c = new Car();
     c.print();
```

First, the function is checked in the child class, if the function is found there with same arguments, then the function in the Parent class will be overridden by the function in the child class, otherwise if the function isn't present in child class, then the function in parent class will run.

2. It can occur when one class is inherited from another class
3. In function overriding, function must have same signature of functions.
4. It is the concept of compile time polymorphism
5. In overriding, every class function explains its own behaviour.

VEHICLE → PARENT CLASS

```
package inheritance;
public class Vehicle {
private String color;
int maxSpeed;
public String getColor()
 { return color;
 }
public void setColor(String color)
 { this.color = color;
 }
public void print()
{ System.out.println("Vehicle color "+ color
                     + "maxSpeed "+ maxSpeed);
}
}
```

CAR → CHILD CLASS

```
package inheritance;
public class Car extends Vehicle {
 int numDoors;
public void print()
 { System.out.println
   ("Car color "+ getColor()+
      "maxspeed "+ maxSpeed +
     "numdoor "+ numDoors);
 }
```

BICYCLE → CHILD CLASS

```
package inheritance;
public class BiCycle extends Vehicle {
}
```

```
package inheritance;
public class VehicleUse {
public static void main(String[] args) {
  Vehicle v = new Vehicle();
  v.setColor("red");
  v.maxSpeed=80;
  v.print();

 Car c=new Car();
 c.setColor("Black");
 c.maxSpeed = 100;
 c.numDoors = 4;
 c.print();

 BiCycle b = new BiCycle();
 b.print();
 }
}
```

Vehicle color red maxspeed 80

Vehicle has print() function so,

Car color black maxspeed 100 numDoors 4

Vehicle print()

Car print()

If child class has same func other child class func overrides the parent class func

Car c = new Car()
c.print()

Vehicle color null maxspeed 0

Vehicle print()

Parent class func is not overridden by the same func is not present in child class

BiCycle

BiCycle b = new BiCycle();
b.print();

5. Protected Modifiers

```
package inheritance;
public class Vehicle {
private String color;
protected int maxspeed
public String getcolor()
 { return color;
 }
public void setcolor(String color)
 { this.color= color.
 }
 ublic void print()
{ System.out.println ("Vehicle color "+ color
                      + "maxSpeed"+maxspeed);
}
}
```

Public was package friendly modifier (can be accessed outside pack) by all classes

Protected is also package friendly
i.e can be accessed outside package
but only by derived / child class

Package Inheritence

Package Inheritence_temp

```
package inheritance_temp;
public class Truck extends Vehicle {
 int maxLoadingCapacity;
public void print()
 {
  System.out.println
  ("Truck color "+getcolor()+
     "maxspeed"+ maxspeed +
   "maxLoadingCapacity" +maxLoadingCapacity);
 }
```

maxspeed can be accessed because this class is a derived class

```
package inheritance_temp;

public class VehicleUse2
{  public static void main(String []args)
 { Vehicle v = new Vehicle();
   v.maxSpeed=10;
 }
}
```

maxspeed can't be accessed because it isn't a derived class

So, we have discussed 4 types of Modifiers

1 → default access modifier.
2 → private access modifier.
3 → protected access modifier.
4 → public access modifier.

		Same package		different package	
	same class	derived class	non-derived class	derived class	non-derived class
private	✓	✗	✗	✗	✗
default	✓	✓	✓	✗	✗
protected	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓

6. Super Keyword

Parent class can be called as
Parent Class / Base Class / Super Class

Child class can be called as
Child Class / derived class / subclass

VEHICLE → PARENT CLASS

```
package inheritance;
public class Vehicle {
private String color;
protected int maxSpeed;
public String getColor()
 { return color;
 }
public void setColor(String color)
 { this.color = color;
 }
public void print()
{ System.out.println("Vehicle color "+ color
                     + "maxSpeed"+ maxSpeed);
}
}
```

CAR → CHILD CLASS

```
package inheritance;
public class Car extends Vehicle {
int numDoors;
int maxSpeed;
public void print(){
 super.print();
System.out.println(
   "Car numDoors"+ numDoors);
}
public void printMaxSpeed()
 { super.maxSpeed=150
   System.out.println(maxSpeed +
    super.maxSpeed);
}
```

→ calling the parent's print function

Parent class maxspeed

car's maxspeed

BICYCLE → CHILD CLASS

```
package inheritance;
public class BiCycle extends Vehicle {
}
```

```
package inheritance;
public class VehicleUse{
public static void main(String[]args){
   Vehicle v= new Vehicle();
    v.setColor("red");
    v.maxSpeed=80;
    v.print();

  Car c=new Car();
  c.setcolor("Black");
  c.maxSpeed=100;
  c.numDoors=4;
   c.print();
  c.printMaxSpeed();
  BiCycle b=new BiCycle();
  b.print();
 }
}
```

Vehicle color red maxspeed 80

vehicle has print() function so,

Vehicle color black maxspeed 150

Car numDoors 4

100 150

Parent class maxspeed

Vehicle print()

Car print()

If child class has same func other child class func overrides the parent class func

Car c = new Car()
c.print()

Vehicle color null maxSpeed 0

Vehicle print()

Parent class func is not overridden by the same func not present in child class

BiCycle

BiCycle b = new BiCycle();
b.print();

7. Inheritance and Constructors

```
package inheritance;

public class Vehicle
{ private String color;
  int maxspeed;
  public Vehicle()
  { System.out.println("Vehicle's Constructor");
  }
public void print()
{ System.out.println
("Vehicle color"+color+"maxSpeed
}
private String getColor()
{ return color;}
private String setColor()
{ this.color=color;}
}
```

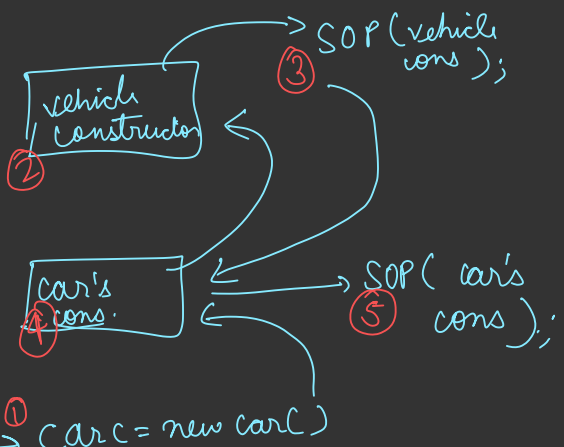
If this takes parameter, then we need to put the parameter earlier

The default constructors only work unless we create our own constructors

Inherit properties & Functionalities

Car

```
package inheritance;
public class Car extends Vehicle {
 int numDoors;
  public Car()
 { System.out.println("Car's constructor");
 }
public void printCar()
  { System.out.println
    ("Car color"+getColor()+
     "maxSpeed"+maxSpeed + "numdoor"+numDoors);
  }
```




```
package inheritance;
public class VehicleUse{
public static void main (String[]args){
  Vehicle v= new Vehicle();
  v.setColor("red");
  v.maxSpeed=80;
  v.printC();
Car c=new Car();
c.setcolor("Black");
c.maxSpeed = 100;
c.numDoors=4;
c.printCar();
  }
}
```

Vehicle's constructor

Vehicle's constructor
Car's Constructor

Whenever we call the constructor class, it is first gonna call constructor of parent class

```
package inheritance;

public class Vehicle
{ private String color;
  int maxspeed;
  public Vehicle (int maxSpeed)
  { System.out.println ("Vehicle's Constructor " +
                        + maxspeed
  }

public void print()
{ System.out.println
  ("Vehicle color"+color+ "maxSpeed

}
private String getColor()
 { return color;}
private String setcolor()
 { this.color=color;}

}
```

Car

```
package inheritance;
public class Car extends Vehicle {
 int numDoors;
 public Car(int numDoors, int maxspeed)
 { super (maxspeed)
  System.out.println ("Car's constructor");
    this.numDoors = numDoors;
  }
public void printCar()
 { System.out.println
   ("Car color "+getcolor()+
     "maxspeed"+ maxspeed + "numdoor" + numDoors);

  }
```

In child cons. first thing that need to be done is call parent class cons.

```
package inheritance;
public class VehicleUse{
public static void main (String[] args){
  Vehicle v= new Vehicle(40);
  v.setColor("red");
   v.maxSpeed=80;
   v.printC();

Car c=new Car(4,100)
c.setcolor("Black");
c.maxSpeed = 100;
c.numDoors=4;
 c.printCar();

  }
}
```

→ Vehicle's constructor 40

→ Vehicle's constructor 100
Car's Constructor

8. Third Component of OOPS → Polymorphism

We discussed these 2 components of OOPS

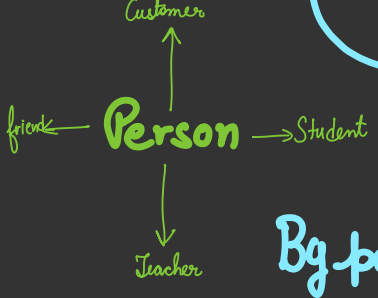
1. Encapsulation → Abstraction
2. Inheritance

Polymorphism → A thing that is capable of taking multiple forms is polymorphic.

Poly - morphism

many behaviour/forms

Many forms

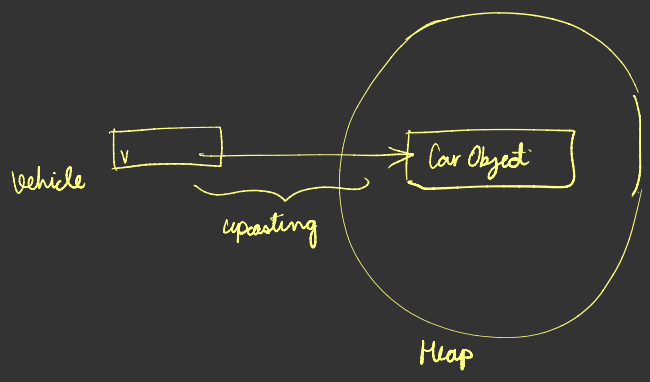


By passing diff. parameters, functions behave differently.

Polymorphism is the greek word whose meaning is "same object having different behaviour"

```
Vehicle V= new Car();
Vehicle V2= new Bicycle();
```

Reference of Parent Class can refer to **Object of child class**




```
package inheritance;
public class Vehicle
{
  public static void main(String[] args)
  {
    Vehicle v=new Car(4,100);
  }
}
```

This is called Upcasting in Java

Because if we create an object of Car class, it calls Car constructor & a child class constructor will first call Parent Class Constructor so Vehicle Constructor will be called and it need maxspeed as a parameter.

numDoors maxspeed

```
v.maxspeed= 100;
```

↓ vehicle specific properties can work

```
v. numDoors = 3;
```

↳ giving compile time error; Because here the codes runs line by line and in this line compiler doesn't know whether vehicle is a car or not

} Compile Time Check because every vehicle isn't having numDoors

```
v.print();
```

```
int n= scan.nextInt();
if(n<10){ V= new Car(4,100);}
else{ V= new Vehicle();}
V.print();
}
}
```

→ We have print() function in car & vehicle but here car's function will run because first the child's object will be checked & then parents

1. Reference variable → Own class object.
2. Reference variable → child " " } upcasting
3. Reference variable can not point to Parent class object

Polymorphism is of 2 types →

→ 1. Compile-Time Polymorphism

It is also called Static Polymorphism, Early Binding.

METHOD OVERLOADING

```
Class A
{
    public void show ()
    {
        ———— //1
    }
    public void show (int i)
    {
        ———— //2
    }
    public void show (float f)
    {
        ———— //3
    }
}

main ()
{
    A obj = new A ();
    obj.show ();
}
```

Why do we call function/method overloading as Compile Time Polymorphism
It is because at compile time, java knows which method to call by checking method signatures. So, this is called Compile time polymorphism or Static or Early Binding. So, compile time polymorphism is achieved through method overloading.

जैसे error आना होगा, तो compile time पर ही आएगा।

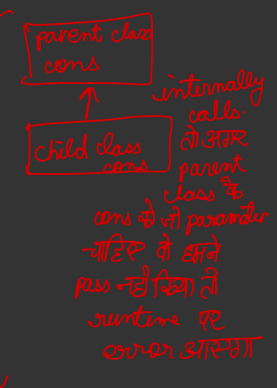
→ 2. Runtime Polymorphism

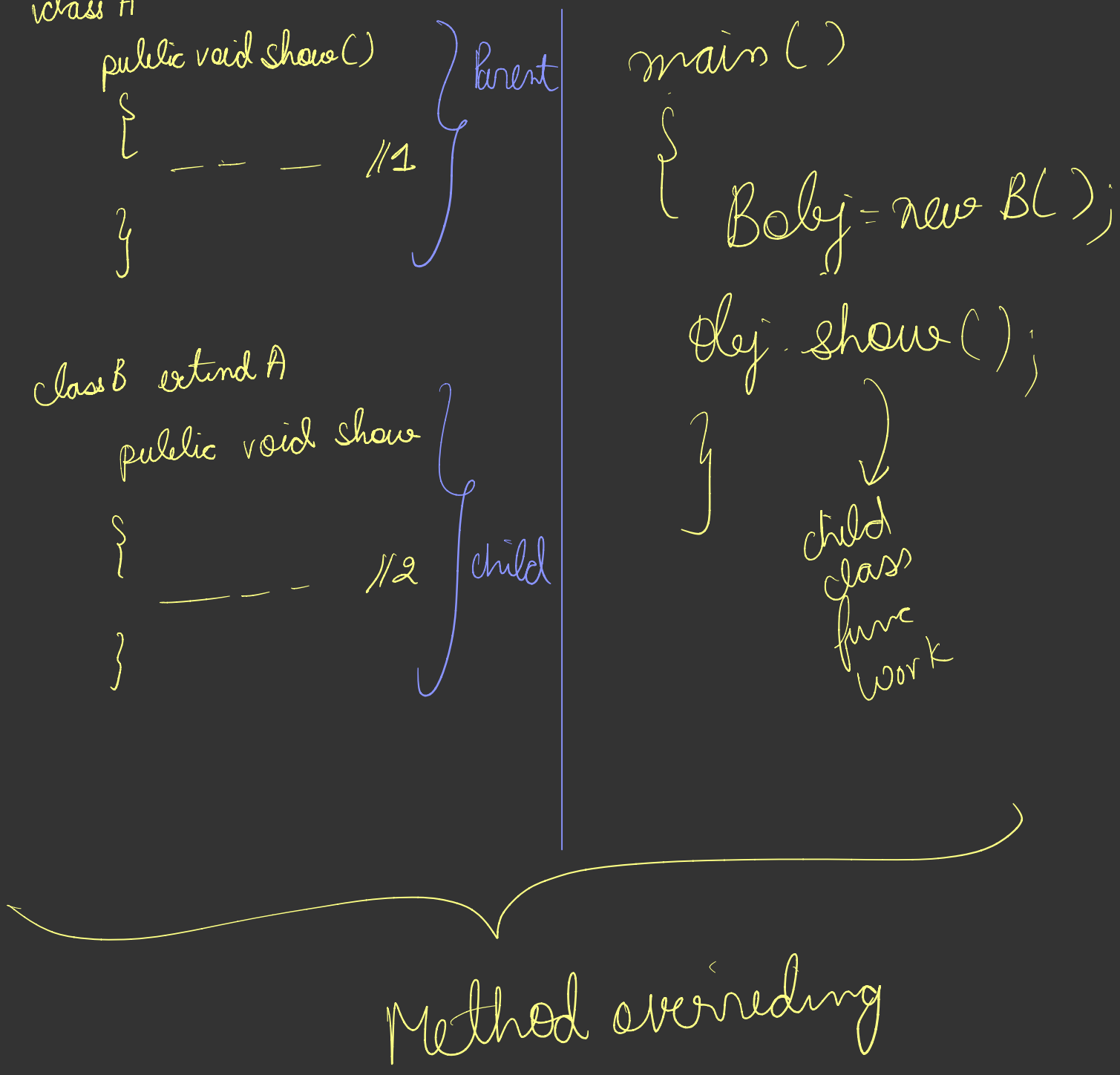
It is also called Dynamic Polymorphism, late binding.

The runtime polymorphism can be achieved by Method overriding. Method overriding says that the child class has the same method as declared in the parent class. Then if child class obj calls a func that is present in both child & parent class, then child class func runs, here child class function override parent class function.

This is called Method / Function overriding. It is an example of Runtime Polymorphism.

जो error आएगा वो run-time पर आएगा।




class A
public void show()
{
---- //1
}
Parent
class B extend A
public void show
{
---- //2
}
child
main()
{
Bobj = new B();
obj.show();
}
child class func work
Method overriding

OVERLOADING	OVERRIDING
1. Occurs within ONE CLASS.	1. Occurs in TWO classes: SUPER class and SUB class ie inheritence is involved.
2. Name of the method is same but parameters are different.	2. Name and Parameters both are same
3. Purpose :- Increases Readability of Program.	3. Purpose: Use the method in the child class which is already present in Parent Class
4. Return type can be same or different	4. Return type is always same.
5. It is an example of Compile Time Polymorphism.	5. It is an example of RUN TIME POLYMORPHISM.
6. Which method to call is decided by Compiler.	6. Method call is decided by JVM in the run time

9. Class cast & Object Class

```
package inheritance;
package class vehicletest{
  public static void main (String[]args)
  {
    Vehicle v = new Car(4,100);
    Car c = (Car) v;
    c.numDoors = 3;
  }
}
```

Compile Time Polymorphism

Explicit Type conversion

If v was a Bicycle then he would have got a Class cast Exception

OBJECT CLASS

There is a class called Object class in Java & it is the Parent class of all/any class we create in Java.
It is inbuilt.

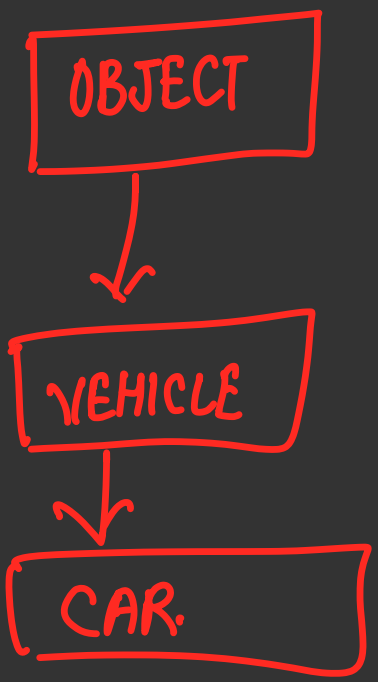
Any class we create in Java are subclasses of object class

```
Object o = new Car();
```

↳ Car constructor will call

Vehicle constructor will call

Object constructor

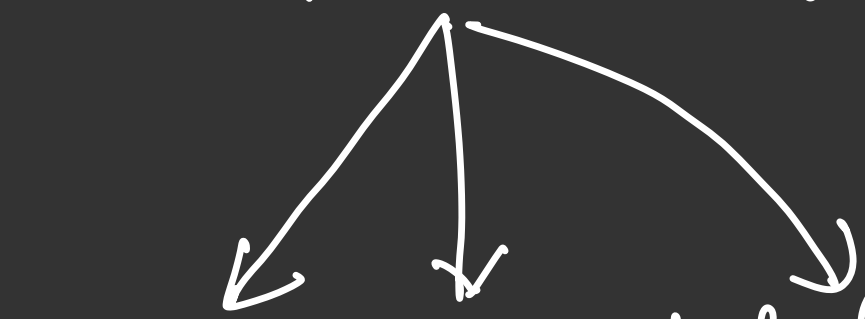


10. EXCEPTION HANDLING

Exception is actually an abnormal event happened in your code because of some error

If something wrong event is happening, it will disrupt my code flow, we will see it on console if its thrown

Exceptions are of 3 Types in Java :-



Errors
↓
extreme cases (like Harddisk failure)

Unchecked
↓
Arithematic (Maths), Array IndexOut of Bound,
→ Logical error.
→ We aren't aware

Checked.
↳ We know that they can occur.

How to throw Exception?

```
package exceptions;
public class ExceptionDemo;

public static int divide (int a, int b)
  { if (b==0) {
                throw new ArithematicException ();
              }
    return a/b;
  }
public static void main( String[] args)
  {
   divide(10, 0);
  }
}
```

→ Unchecked Exception

This is how we create a new Exception

If $\frac{4}{0} = \infty$

If exception occur, nothing after exception in code will be done.

Exception → An Exception is an unexpected/unwanted/abnormal situation occured at runtime.

```
String s = null;
SOPC(s);
```

Eg → ArithmaticException

Or

ArrayIndexOutOfBoundException

Exception Handling → If in case we come across an Exception, then we must have an alternate source through which we can handle the Exception, so that code is not disrupted in between. So that code flow goes on, (so that our entire code runs)

Exception Handling

```
public class A
{
    public static void main()
    {
        int a=10, b = 0, c;
        c = a/b;
        SOP(c);
    }
}
```

If JVM encountered AE here, so code written after this point will not run & an exception will be thrown.

So, we need to handle this exception

The Object Orientation Mechanism has provided the following Techniques to work with Exceptions :-

1. try
2. catch
3. throw
4. throws
5. finally

We got 2 option
→ Either handle the exception or throw it (pass

Try & Catch →

```
class A
{ main()
  {
    int a=10, b=0, c;
    try {
          c = a/b;
          SOP(c);
        }
    catch (exception e)
        {
          S.OP(e)
        }
    SOP("Till Last");
  }
}
```

This part of code has some risk so put it in Try Block

Catch block will catch the exception thrown by try block

If Exception comes then anything written in try written after, won't run

यदि हमें इस exception को handle करदें & हमारा program end तक run करदे है।

```
package exceptions;
public class ExceptionDemo;

public static int divide (int a, int b) throws DivideByZero Exception
  { if (b==0) {
        throw new DivideByZeroException();
    }
   return a/b;
}
public static void main( String[] args)
  {
     try {
        divide (10,5);
        System.out.println ("Within try");
     }
     catch (DivideByZeroException)
     {
       System.out.println (" Exc raised");
     }
  System.out.println (" Main");
}
```

→ Checked Exception

This is how we create a new Exception

$\frac{4}{0} = \infty$

```
package exceptions;
public class DivideByZeroException extends Exception
{

}
```

That's how we create a new Exception

Main doesn't throw Exception here

```
package exceptions;
public class ExceptionDemo;

public static int divide (int a, int b) throws divideByZeroException
  { if (b==0) {
        throw new DivideByZeroException();
    }
   return a/b;
}
public static void main( String[] args) throws divideByZeroException
  {
     int x= divide (10/0);
     System.out.println(x);
  }
```

throw → throw

new DivideByZeroException() → object of Exception Class that we want to throw

This function can throw DivideByZeroException

Main function can throw DivideByZeroException

So it will be shown in the console.

exception raised here

```
Exception in Thread "main" exceptions.DivideByZeroException
    at exceptions.ExceptionDemo.divide(ExceptionDemo.java:16)
    at exceptions.ExceptionDemo.main(ExceptionDemo.java:28)
```

call

```
class Test
{ public static void main(String[] args)
  { SOP("main method started");
    int a=10, b=0, c;
    c = a/b;
    SOP(c);
    SOP("main method ended");
  }
}
```

Java में EP class की main thread होती है जो which is responsible for running the entire main function

OUTPUT

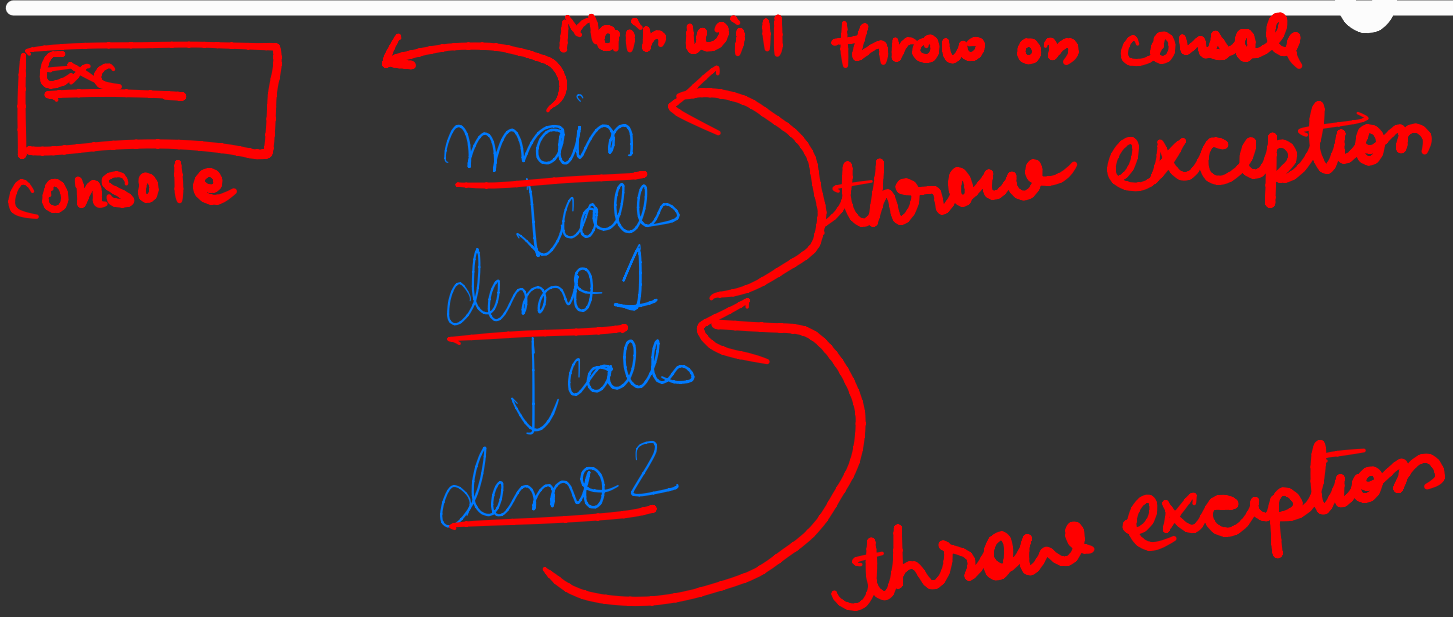
→ main method started

→ Exception is thread main → java.lang.ArithmaticException: / by 0

Reason is this

AE is a class in lang package in java

11. Exception Handling 2



→ Each func got 2 option

- Handle the Exception using try catch
- throw back to the func who called you

```
package exceptions;
public class ExceptionDemo {
  public static int fact(int n)
  { if(n<0){
      throw new NegativeNumberException();
    }
    int ans=1;
    for(int i=2; i<=n; i++)
    {
      ans=ans*i;
    }
    return ans;
  }
  public static int divide(int a, int b) throws DivideByZeroException
  { if(b==0)
    {
      throw new DivideByZeroException();
    }
    return a/b;
  }
```

throws means passing it to user: (User of this function)

```
package exceptions;
public class DivideByZeroException extends Exception
{
}
```

That's how we create a new Exception

```
package exceptions;
public class NegativeNumberException extends Exception
{
}
```

That's how we create a new Exception

```
public static void main (String[] args)
{
  try{ divide(10,5);
       SOP(fact(2));
       SOP("within try");
  }
  catch(DivideByZeroException e)
  { SOP("DivideByZeroException");
  }
  catch(NegativeNumberException)
  { SOP("NegativeNumberException");
  }
  catch(Exception e)
  { SOP("GenericException");
  }
  finally
  { SOP("finally");
  }
}
```

whatever is suspicious of getting exception goes in try

If this exception occur, then catch block code run

finally block is always executed whether we get exception or not

finally Block