KOTLIN

```
package com.example.myfirstapp


my 1st app
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import android.widget.Toast



class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val btnclickme = findViewById<Button>(R.id.mybutton)
        val textview2 = findViewById<TextView>(R.id.textView)
        var count = 0


        btnclickme.text="woow"
        btnclickme.setOnClickListener {
            count +=1
            textview2.text = count.toString()
            //Toast.makeText(context: this, text:"it's me" , Toast.LENGTH_LONG).show()
           // btnclickme.text="here u r"
            //textview2.text="Hiiiiiiiii"
        }
```

}

}

/* Our first project - the UI and how to change the size of the text

SDK- Software development kit(collection of SD tools & one installable package & faciliate the creation of applications by having a compiler

xml file take care of UI

setContentview(R.layout.activity_main) :- means screen should use file having name activity_main*/

/*AVD Android Virtual Device is an emulator which emulates Android on our pc & use to test app on our pc directly without involving phone

*/

/*Class - used to hold & execute our code

Activity = Screen

sp - scale independent pixels

There r packages u can import, different libraries that allow u to use their code*/

/*VAR keyword bcz this is going to be a variable that'll changeable

we r importing class names

there is method called setOnClickListner that we can then execute & what we add as code inside those curly brackets will be executed once we click on button*/

/*To check app in your phone enable developer option and USB debugging in your phone & then run from your PC....app will be installed automatically*/

kotlin basics

```
package com.example.kotlinbasics


fun main(){
    // type string
    var ad = "Bharadwaj"
    ad = "Kumar"
    print("Akshay" + ad)
    // type int
    // it's not necessary to assign data type to variables
    val type: Byte = 24
    val type2: Short = 5456
    val type3:Int =5446444
    val type4: Long = 585_555566_55555545
    val type5 : Float=55.25F
    // we need to write F for assigning float
    // otherwise it too Double by default
    val type6 :Double=3.21465464654454
    //Boolean
    var hi=true
    hi=false
    val letterchar='A'
    val digitchar='1'
```

```
val mystr = "Hello world"
var firstcaharacter = mystr[0]
var lastcaharacter = mystr[mystr.length -1]
print("first character" + firstcaharacter )
//String template expression or string interpolation
print("First cahracter $firstcaharacter and the length of mystr is ${mystr.length}")
//o/p is :- First cahracter H and the length of mystr is 11
// Arithmatic operator
var res = 5+4
res /=3
print(res)
// o/p will be 3
var res1 = 5+3
var a = 5
var b = 3
res1 = a / b
print (res1)// prints 1
val c=5.0
b=3

var res2 : Double
res2 = c/b
print(res2)// prints 1.66666
// comparision operator(==,!=.<,>,<=,>=)
val isEqual = 5==3
println("isEqual is" + isEqual) // prints isEqual is false
//we can also write it as
println("isEqual is $isEqual") // prints isEqual is false
println("is-5greater3 ${-5>3}") // prints is-5greater3 false
// if u want to execute expression before printing on screen then us ${}
```

```
// Assignment operator(+=,-=,*=,/=,%=)
var p =5
p+=3
println("sum is $p")// prints sum is 8

// Increament & Decreament operator(++,--)
p++
println("sum is $p")// prints 9
println("sum is ${p++}")// prints 9
println("sum is ${++p}")// prints 11

// conditional statements
var e=456
var f=562
if(e>f){
   println("use raw")
}else if(e==f){
   println("equal")
}else{
   println("use technique")
}
var name="Akshay"
if(name=="Akshay")
   println("true")
else{
   println("no")
}
//when statement
var season=3
  when(season){
    1 -> print("hi") // means if season is 1 then compiler will print "hi"
```

```
    2 -> print("hello")
    3 -> {
        // can be multiline statement also
        print("hiii")
        print("hello")
    }
    else -> {
        println("invalid")
    }

}
var month=3
when(month){
    in 3..5 -> println("woow") // means from month 3 to 5 it'll print "woow"
    in 6..8 -> println("summer")
    in 12 downTo 2 -> println("spring")
    // we can also write
    12 , 1 , 2 -> println("winter") // means if those three values are true then it print winter
    !in 0..5 -> println("hhh") // means if month is not from 0 to 5 then this expression ll run
    else -> {
        println("invalid")
    }

}
var x : Any = 13.25
when(x){
    is Int -> println("$x is an Int")
    !is Double -> println("not a Double number")// if it isn't double then this statement will print
    //means if it is integer then this statement will print
    //similarly check for Double , String
    else -> println("$x is none of the above")
```

```
}
// while loop
var y=1
while(y<=10){
    println("$y")
    // u can use both print & println , in println it works like endl like in this case it will print 1-10 vertically whereas if we use print then it print 1-10 horiz0ntally
    y++
}
println("\n loop is done")
// \n is line break
// Do while loop
y=1
do{
    print("$y")
    y++
}while(y<=10)
println("\n do while loop is done")
var feltTemp="cold"
var roomTemp =10
while(feltTemp=="cold"){
    roomTemp++
    if(roomTemp >=20){
        feltTemp="comfy"
        println("it's comfy")
    }

}
for(num in 1..10){
    print("$num")
}
```

```
    // we can also write
    for(i in 1 until 10){
        print("$i")
    }
    // other ways also to write for loop
    for(i in 10 downTo 1 step 2){// means it will print in diff of 2
        print("$i")
        // break & continue keywords

    for(i in 1 until 20){
        //print("$i")//print from 1 upto 19
        if(i/2 ==5){
            //break
            continue
        }
        print("$i")

    }
        print("Done with the loop")


}
fun main() {
    myfunction()// function calling
    var result = addup(a:5, b:3)
    print("result is $result")//prints 8
    var result1 =avg(a:2.36,b:3.25)
    print("$result1")

}
fun myfunction() {
```

```
    print("called my function")
}
fun addup(a:Int,b:Int):Int//return type of function{
    return a+b
}
fun avg(a:Double,b:Double):Double{
    return (a+b)/2
}
```

/*Create a new file in java folder

Main fun is the starting point of our application

var variable can be overridden whereas val can't (we can't reassign a new value to val variable)

Int types : byte(8 bit), short(16),int(32),long(64)

Float(32),Double(64)*/

/*Boolean is used to represent logical values .

It can have two possible values true & false

u don't need to specify type bcz kotlin supports type inference & it finds the type & size from context.*/

/*String template expression or string interpollation*/

/*null can't be a value of non null type string*/

kotlin basics 1.0

```
package com.example.kotlin10

import java.sql.DriverManager.println

fun main(){
    var name : String = "Akshay"
    //name=null -> complilation error
    var name2 : String? = "Akshay"
    name2=null

    var len=name.length
    //var len2=name2.length// here will get error
    //use this
    var len2=name2?.length
    println(name2?.toLowerCase())//it'll print in lower case
    // ? called Safe call operator
    name2?.Let { println(it.length)}// means we can write this instead od if else statement
    // println statement only executed if name2 is not null
    //u can use this also(i.e given below)    both are same
    /*
    if(name2!=null){
    var len2=name2.length}
    else{
    null}
     */
```

```
    // here we r assigning nullable variable to  a non nullable variable
    // here we r using Elvis operator(?:) which assign the value of name2 to  name3 & if name2 is null then it'll print a default value given by us  ie "Guest"


    val name3 = name2 ?: "Guest"
    print("name is $name3")// if name 2 has some assigned value then it'll print that otherwise it'll print Guest
    // not null assertion operator(!!)
    // which converts nullable type to non nullable type & throws a exception if a nullable type holds a null value
    //use only u r 100% sure that there'll be a value in variable
    print(name2!!.toLowerCase())// here we got error bcz name2 is null
    // if it's not null it's going to print akshay



}
```

Kotlin basics 2.0

```
package com.example.kotlinbasics20

import android.app.backup.BackupAgent

fun main(){
    //person("Denis", "panjuta")// oject of class person
```

```
    //we can pass age also after making primary constructor
    var akshay=person("Denis", "panjuta,age:21")
    akshay.age=21
    println("akshay is ${akshay.age} years old")
   // var akshay=person("Denis", "panjuta")
    // A variable is a object if it is a type which is a class
    //print as we gave above
    akshay.stateHobby()// prints My hobby is watch netflix
    //if u want to change hobby
    akshay.hobby="cricket"//chnaging properties
    akshay.stateHobby()//prints My hobby is cricket
    // Now we can have individual hobby for every single person
    //similarly as we do above
    var new =person()//creating new object
    //prints as we gave in class parameter
    // for this object it'll autmatically take default
    //value what we gave in class parameter
    var new2 =person(lastName="bhardwaj")
    //it'll print first name from class paramater & second as we gave above
    // created 3 objects of a class

}
class person(firstName: String="aksh", lastName: String="bharadwaj"/*primary constructor*/){
    //Member variables (or) properties is a variable within a class
    // primary constructor run by initialiser block
    var age : Int?=null
    var hobby : String="watch netflix"
    var firstName : String ? = null
    //Initializer block
    init{
        this.firstName=firstName
```

```
        // whenever we create a class we have an init there
        // this is initialiser which is called once the object is created
        //once we run code 5 it'll automatically run this initialiser
        print("person created")
        //once we run code person created will be printed
        print("first name $firstName"+"lasname $lastName")// u can also do this

    }
    //Member secondary constructor
    constructor(firstName: String,lastName: String,agent: age:Int):this(firstName,lastName){
        this.age=age
        print("first name $firstName"+"lasname $lastName and age $age")
    }
    //member functions (or) Methods
    fun stateHobby(){
        println("My hobby is $hobby")
        // we can't use $firstName bcz it's not a varaible
        // it's visible only in primary& secondary constructor
        //Now we can use after creating variable
        println("$firstName hobby is $hobby")
        // now it prints Denis hobby is cricket

    }
}
```

oops concept

```
package com.example.oops3


data class user(val id:Long,var name:String)
fun main(){
    val user1=user(id:1, name:"Denis")
    //val name=user1.name comment out for line 17
    println(name)//prints Denis
    user1.name="Michael"
    val user2=user(id:1,name:"Michael")
    println(user1 == user2)//prints true
    val updateuser=user1.copy(name="Denis panjuta")
    println(user1)//prints user(id=1,name=Michael)
    println(updateuser)//prints user(id=1,name=Denis panjuta)
    println(updateuser.component1())//prints 1
    println(updateuser.component2())//prints Denis panjuta
    val (id,name)=updateduser//took id & name from updateduser & store them in separate variables
    println("id=$id,name=$name")//prints id=1 name=Denis panjuta
}
/*
5 basic cocepts


variables & types , Control flows , Functions , collections , Classes & Objects (Including inheritance)
```

Variables & types

- A variable is a location in memory

- To indicate the storage area each variable should given a unique name

fun main(){

//entry point of lotlin programme }

Control flows

- Allow to do something conditionally

- to repeat run code (conditionally)

- if statement

Functions

- To separate code

- To run code blocks when needed

eg :- calling fun inside main fun

collections

- Enables us to store multiple enable at one place

- Iterate through multiple elements(with the help of control flow

classes

Allows us to

- Create our own datatypes

- Keep data members & method together In one place

- write more readable and maintainable code

*/A class is just a blueprint where we define all of these values such as properties & skills & objects are individual cars that are produced.

Kotlin allows to declare variable with the same name as parameter inside the method.

Kotlin internally generates a default getter and setter for mutable properties, and a getter (only) for read-only properties.

When creating data class you must have at least one parameter that primary constructor mus have at least 1 parameter

Data class can't be abstract,open,sealed or inner classes.

*/

INHERITENCE AND INTERFACES

```
package com.example.inheritance


/*class that inherts the feature of another
class called sub class or child class or
Derived class and class whose features are
inherited called super class or parent class or
Base class.
 */
interface  Drivable{
    val maxSpeed :Double//property
    fun drive():String
    fun brake(){
        println("The drivable is braking")
    }
}
//super class
/*open class vehicle{
    //properties
    //methods
}*/
//sub class  of vehicle
//super class of vehicle
open class car(override val maxspeed:Double,val name:String,val brand:String):Drivable{
    open var range:Double=0.0
```

```
    fun extendRange(amount:Double){
        if(amount>0)
            range+=amount
    }
    override fun drive():String{
        return "driving the interface drive"
    }
    //u can also write it as
    //override fun drive():String="driving the interface drive"

   open fun drive(distance:Double){//used open keyword to override drive functionality
        println("Drove for $distance KM")
    }

}
//sub class of car
class ElectricCar(maxspeed: Double,name:String,brand:String,batterylife:Double)
    :car(maxspeed,name,brand)/*extends the functanility
    of it's parent class(car)
    if Electric car wants to inherit from car it also has to follow
    the structure of our interface*/{
    var cahrgerType="Type1"
     override var range =batterylife*6
    override fun drive(distance:Double){
        println("Drove for $distance KM on electricity")
    }
    /*
    fun drive(){
        println("Drove for $range KM on electricity")

    }*/
```

```
    //after interface
    override fun drive():String {
        return "Drove for $range KM on electricity"

    }
    override fun brake(){
        super.brake()
        println("break inside of electric car")
    }


}
//Any class inherits from any class
//Any is a class in kotlin,it's an open class
//the root of kotlin class heirarchy,every kotlin class has any
//as a super class & that's why we can always ues
//the function equals,hashcode, toString()
fun main(){
    var myCar=car(maxSpeed:200.0,name:"A3",brand:"Audi")
    var myECar=ElectricCar(maxSpeed:240.0,name:"S-Model",brand:"Tesla",batterylife:85.0)
    myECar.cargerType="Type2"//can change if u wnat
    myECar.extendRange(amount:200.0)
    myECar.drive()/*prints Drove for 200.0 KM on electricity
                    Drove for 710.0 KM on electricity*/
    myECar.brake()// prints The drivale is braking
              //        break inside of electric car
    myCar.brake()//The drivale is braking
    //polymorphism(ability to treat objects
    //with similar traits in a common way
    myCar.drive(distance:200.0)//prints Drove for 200.0 KM
    myECar.drive(distance:200.0)//prints Drove for 200.0 KM}
```

ABSTRACT CLASS

```
package com.example.abstractclass

import android.webkit.WebStorage
/*Abstract classes little bit similar to
interfaces but they are still different .*/

/*An abstract class can't be instantiated
(u can't create objects of an abstract class).
However,you can inherit subclasses from an abstract class.
The members(properties & methods) of an abstract class are non abstract
unless u explicitly use the abstract keyword to make them abstract.
 */
abstract class Mammal(private val name:String,private val origin: WebStorage.Origin:String,
private val weight :Double){//Concrete(Non Abstract)properties
   //Abstract property(Must be overridden by subclasses)
abstract var maxspeed :Double
//Abstract Methods (Must be implemented by subclasses)
abstract fun run()
abstract fun breath()
//Concrete (Non Abstract)Method
fun displayDetails(){
   println("Nmae:$name,Origin:$origin,Weight:weight"+"Max Speed :$maxspeed")
}

}
class Human(name:String,origin:String,weight: Double,override
var maxspeed:Double):Mammel(name,origin,weight){
```

```
    override fun run(){
        //code to run
        println("Runs on two legs")
    }
    override fun breath(){
        //code to breath
        println("Breath throught mouth or nose")
    }

}
class Elephant(name: String,origin:String,weight: Double,
override var maxspeed:Double):Mammal(name,origin,weight){
    override fun run(){
        //code to run
        println("Runs on four legs")
    }
    override fun breath(){
        //code to breath
        println("Breath through the trunk")
    }
}
fun main(){
    val human=Human(name:"Denis",origin:"Russia",
    weight:70.0,maxSpeed:28.0)
    val elephant=Elephant(name:"Rosy",origin:"India",
    weight:5400.0,maxSpeed:25.0)

    //val mammal=Mammal(name:"Denis",origin:"Russia",weight:70.0,maxSpeed:28.0)
     //in above code we got error bcz u can't create an instance of an abstract class
    // u can do that with subclasses that is inherited from that.
    /*an interface can't hold state & we can implement multiple interfaces but only 1 class.
```

we can inherit from one class.class have constructor even an abstract class has a constructor.

abstract class have everything that interface have & additionally they have constructor.

therefore we can hold state in an abstract class.*/

human.run()//prints Runs on two legs

elephant.run() // Runs on four legs

human.breath()//Breath through mouth or nose

elephant.breath()//Breath through the trunk

}

//INHERITANCE

/*Inheritance is one of the main concepts of oop & it allows a class to inherit features such as properties and methods from another class and reuse them.

all classes in kotlin are non inheritable by default , to use inheritance we need to make class open (use open keyword to make a class inheritable).

use sealed keyword if don't wan't to make class inheritable*/

//INTERFACE

/*Interfaces are feature which allow us to extend the functionality of classes.

Super keyword to call the break function of superclass.

An interface can inherit from another interface

interfaces is useful when u have certain functions in mind that u definitely want to be implemented later & also properties of class that u want to*/

TYPECASTING

```
package com.example.typecasting


fun main(){
    val stringList:List<String>=listOf("Denis",
    "Frank","Michael","Garry")
    val mixedTypeList:List<Any>=listOf("Denis",
    31,5,"BDay",70.2,"weighs","kg")
    for(value in mixedTypeList){
        if(value is Int){// is keyword
            println("Integer:'$value'")
        }else if (value is Double){
            println("Double:'$value' with Floor value" +
                    "${floor(value)}")


        }else if (value is String){
            println("String: '$value'of length" +
                    "${value.length}")
        }else{
            println("unknown Type")
```

```
    }
  }
  //Alternatively
  for(value in mixedTypeList){
    when(value){
      is Int ->println("Integer :$value")
      is Double-> println("Double:$value with Floor value ${Math.floor}")
      is String->println("String:'$value' of length ${value.length}")
      else->println("unknown Type")
    }
  }
  //SMART CAST
  val obj1:Any="I have a dream"
  if (obj1 !is String){
    println("Not a String")
  }else{
    //obj is automatically cast to String in this scope
    println("Found a string of length ${obj1.length}")
  }
  //Explicit (unsafe) casting using the "as" keyword -can go wrong
  val str1.String=obj1 as String
  println(str1.length)
  val obj2:Any =1337
  val str2:String=obj2 as String
  println(str2)// here will get bcz it's not a string
  //Explicit (safe) casting using the "as?" keyword
  val obj3:Any=1337
  val str3:String?=obj3 as? String//Works
  println(str3)//prints null
}
```

Arrays

```
package com.example.a2nd
fun main() {
    println("hi")
  /*  val numbers:IntArray=intArrayOf(1,2,3,4,5,6)
    val numbers=intArrayOf(1,2,3,4,5,6)
    both of the above statements are valid
   */
    val numbers =arrayOf(1,2,3,4,5,6)
    //println(numbers)//will print the address of above array
    //print(numbers.contentToString())//prints array
    for(ele in numbers )
       // print(ele) //prints all elements of array
   // print("$ele+2")//it just prints 1+2 2+2 3+2 & so on...
   // print("${ele+2}")//increase each value of array by 2
    //here above we didn't change(override) the value we just displayed it
   // print(numbers[0])//prints 1
    numbers[0]=8
   // print(numbers[0])//prints 8
    val numbersD:DoubleArray = doubleArrayOf(1.0,2.0,3.0,4.0,5.0,6.0)
    numbersD[0]=8.0//we overrided here
    val days= arrayOf("sun","mon","tue")//arrayOf can holds diff types of data
    //print(days.contentToString())//prints above strings
    val fruit= arrayOf(Fruit(name:"apple",price:2.5),Fruit(name:"orange",price:3.5))//creating object
   // print(fruit.contentToString())//prints [Fruit(name:"apple",price:2.5)
// ,Fruit(name:"orange",price:3.5)]
    for(ele in fruit ){
        print("${fruit.name}")//prints apple orange
```

```
    }
    for(index in fruit.indices){
        print("${fruit[index].name}is in index $index")//prints apple is in index 0
                                                //      orange is in index 1
    }



}
data class Fruit(val name:String,val price:Double)
```

List

```
package com.example.list


fun main(){
    val months= listOf("jan","feb","mar")
    // to add item in existing list
    //we've to convert it into mutable list
    //& then u can add items
    val anytypes=listOf(1,2,true,false,"hi")
   // print(anytypes.size)//prints 5
   //print(months[1])//prints feb
    /*for(ele in months)
        print(ele)//prints all months of months array*/
    val addionalMonths=months.toMutableList()
    val newMonths= arrayOf("Apr","may","jun")
    addionalMonths.addAll(newMonths)
    addionalMonths.add("july")//u can also add like this
    print(addionalMonths)//prints all months of months array including Apr may jun


    //u can also do this(as below)
    val days = mutableListOf<String>("mon","tue","wed")
    days.add("thu")
    days[2]="sun"//this is how u can override items
    //after this it'll print sun in place of wed
    days.removeAt(index = 1)//delete tue from list
```

```
    val removeList= mutableListOf<String>("mon","wed")
    days.removeAll(removeList)//using this we can remove particular elements from
    //list
    days.removeAll(days)//this'll remove all of the items
    print(days)// prints [mon,tue,wed,thu]

}
```

Sets & Maps

```
package com.example.set

fun main(){

    val  fruits= setOf("oren","app","mang","grap","app","oren")
    print(fruits.size)//prints 4
    //ignoring duplicates
    print(fruits)//prints elements of set ignoring duplicates
    print(fruits.toSortedSet())//prints in sorted alphabetically
    val newfruits=fruits.toMutableList()
    newfruits.add("water melon")
    newfruits.add("pear")
    print(newfruits)//prints all elements of set including above two
    print(newfruits.elementAt(index=4))//prints elements present at index 4
    //map is type of collection that holds data in form of key value pair
```

```
    //map keys are unique & hold one value for each key
    val daysOfTheWeek= mapOf(1 to "mon",2 to "tue",3 to "wed")
    print(daysOfTheWeek[2])//prints tue
    for (key in daysOfTheWeek.keys){
        print("$key is to ${daysOfTheWeek[key]}")//prints 1 is to mon .....
    }
    val fruitsMap= mapOf("Favorite" to fruit(name: "Grape",price:2.5),"ok" to fruit
    (name:"app",price:1.0))
    val newDaysOfWeek=daysOfTheWeek.toMutableMap()
    newDaysOfWeek[4]="thu"
    newDaysOfWeek[5]="fri"
    print(newDaysOfWeek.toSortedMap())//prints whole map in sorted order including above two

}
data class fruit(val name: String,val price:Double )
```

Array List

```
package com.example.arraylist


fun main(){
   /*Constructor of ArrayList
   ArrayList<E>(): Is used to create an empty ArrayList
   ArrayList(capacity:Int):Is used to create an ArrayList of specified capacity
   ArrayList(elements:Colllection<E>):Is used to create an ArrayList filled with
                                the elemets of collection.
   open fun add(element:E):Boolean-> used to add the specific elements into
   the collection.
   open fun clear()-> used to remove all elements from the collection
   open fun get(index:Int):E->used to return the elements at specified index in
                           the list
    open fun remove(element:E):Boolean-> used to remove a single instance of the
  specific element from current collection ,if it is available. There r more functios in array list class.
   val arrayList=ArrayList<String>()//creating an empty array list
   arrayList.add("one")//adding an object in arrayList
   arrayList.add("two")
   println("print ArrayList")
   for(i in arrayList){
      println(i)
   }
}
```

```
/*o/p print ArrayList
one
two
 */
/*              ArrayList using collections
    val arrayList:ArrayList<String>=ArrayList<String>(5)
    var list:MutableList<String>=mutableListOf<String>()

    list.add("one")
    list.add("two")
    arrayList.addAll(list)
    println("print arrayList")


    val itr =arrayList.iterator()

    while(itr.hasNext()){
    println(itr.next())
    }
    println("Size of arrayList="+arrayList.size)


    o/p  is
    print ArrayList
    one
    two
    Sizeof arrayList 2



    println(arrayList.get(1))//means we r accessing ele of index 1*/
```

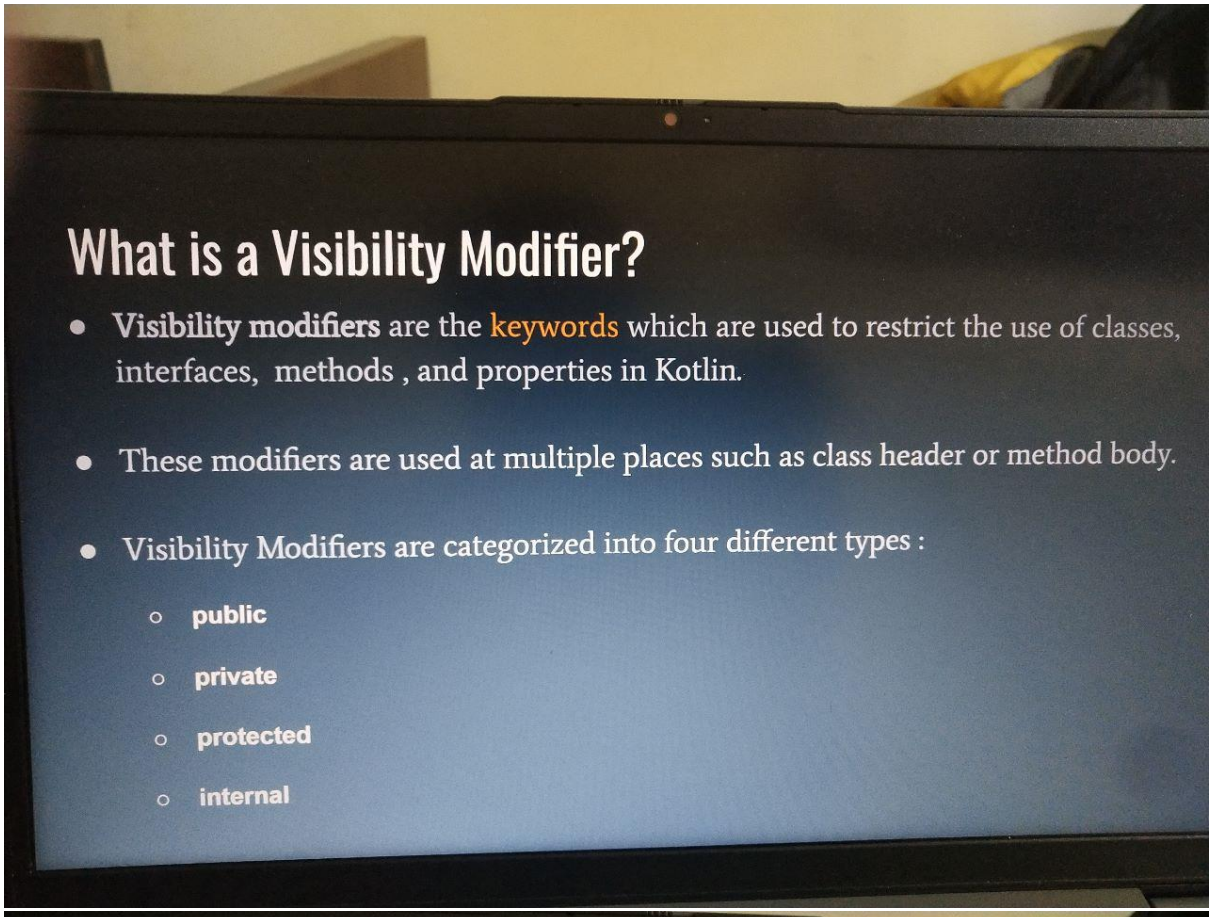
Lambda expression: addition of two numbers

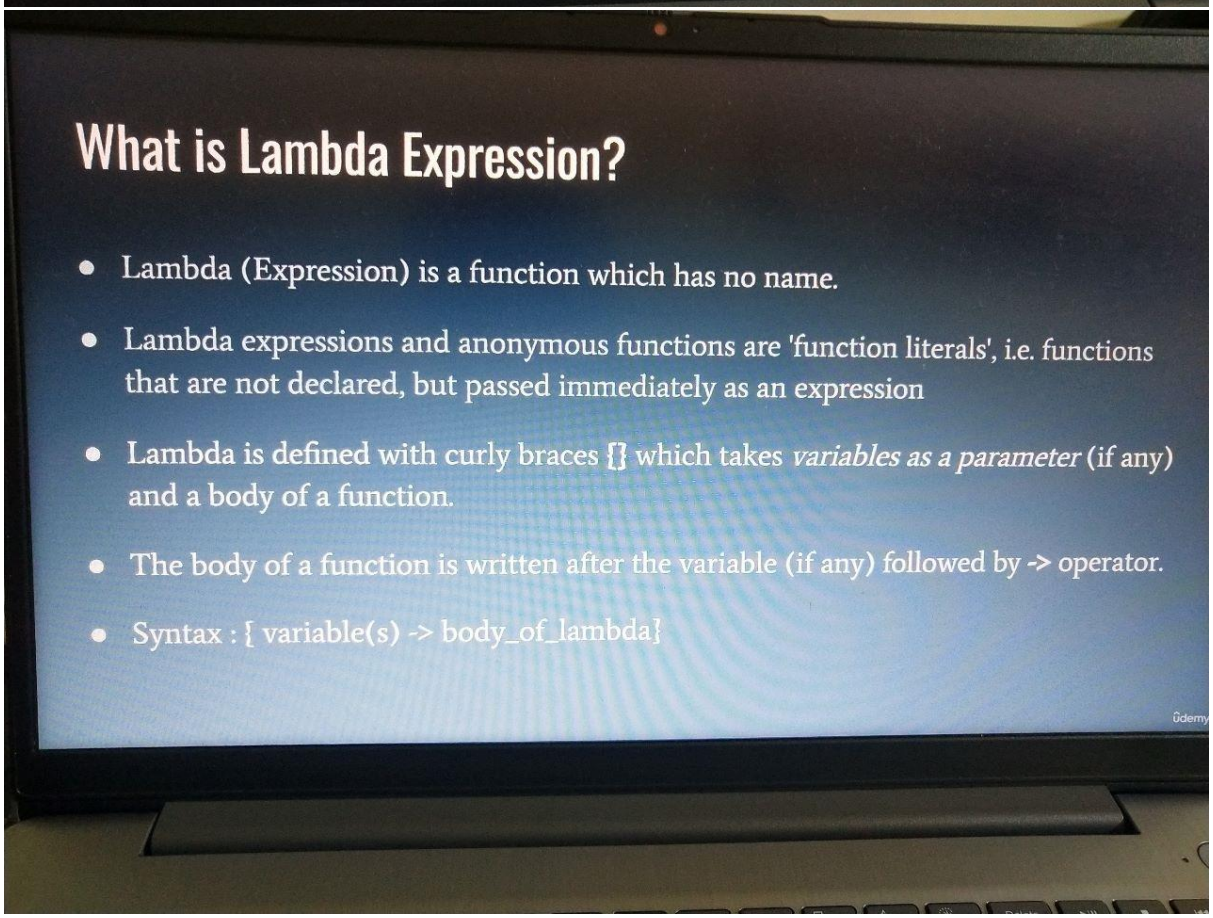
We will write the same example using a lambda expression.

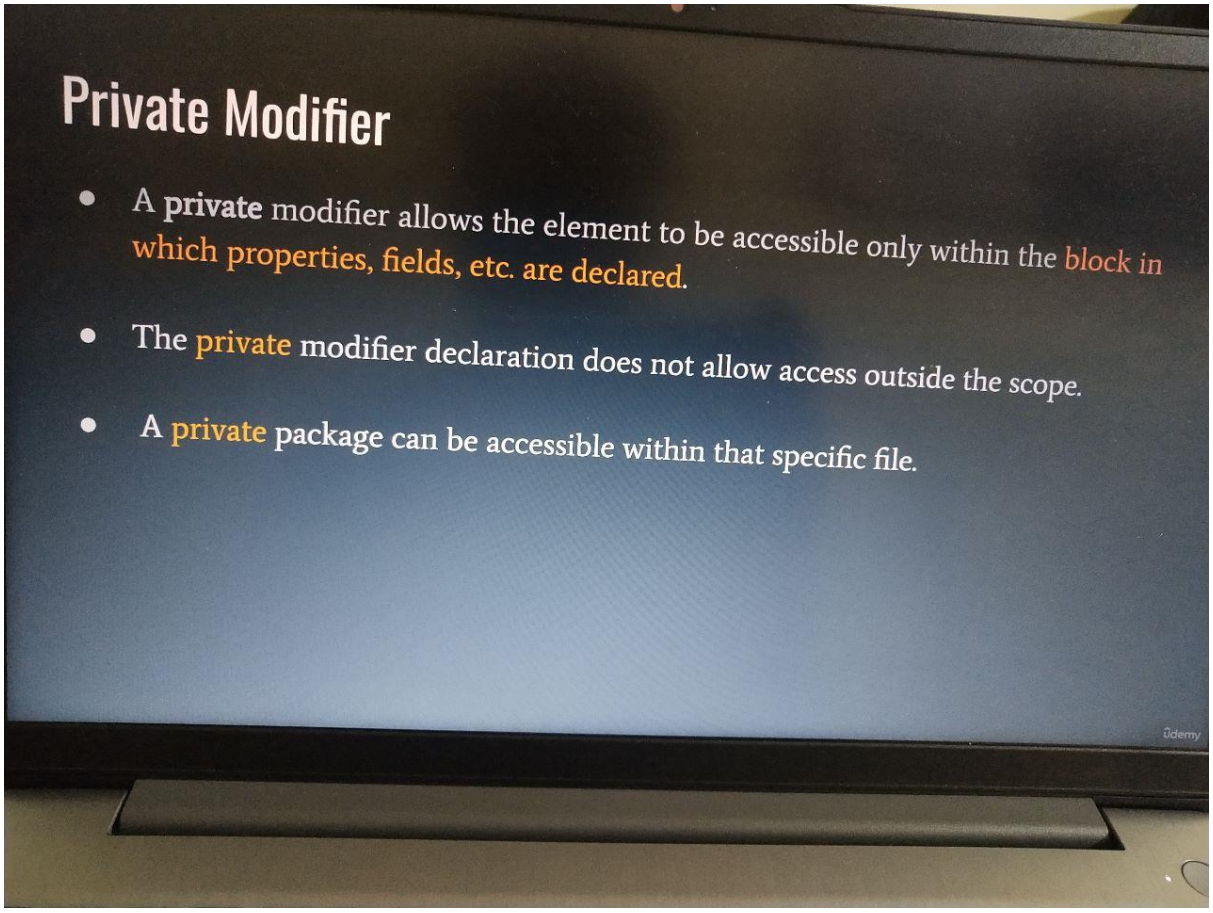
```
val sum: (Int, Int) -> Int = { a: Int, b: Int -> a + b }
println(sum(10,5))

// even shorter
val sum = { a:Int, b:Int -> println(a + b) }
sum(10,5)
```

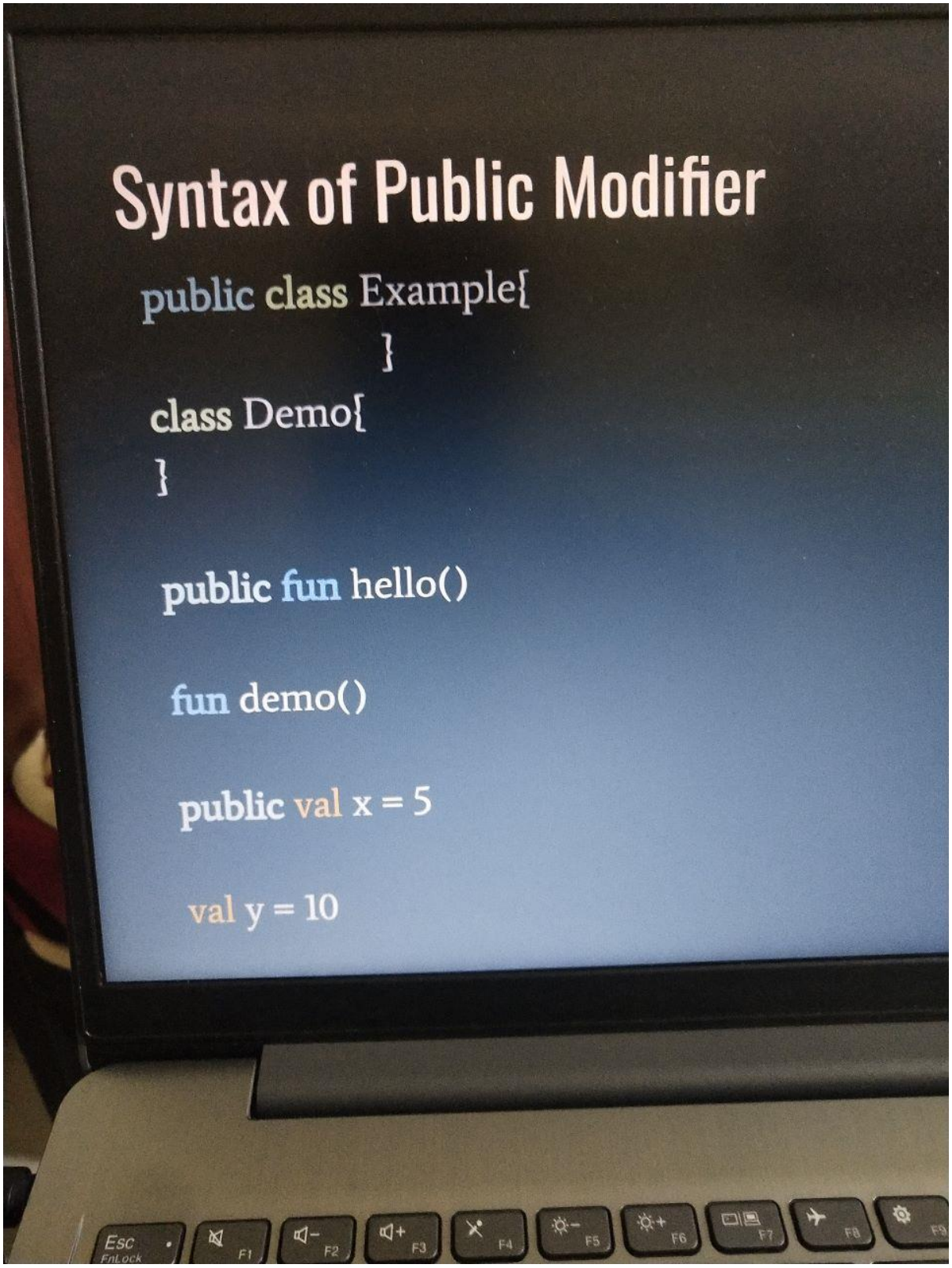
Public Modifier

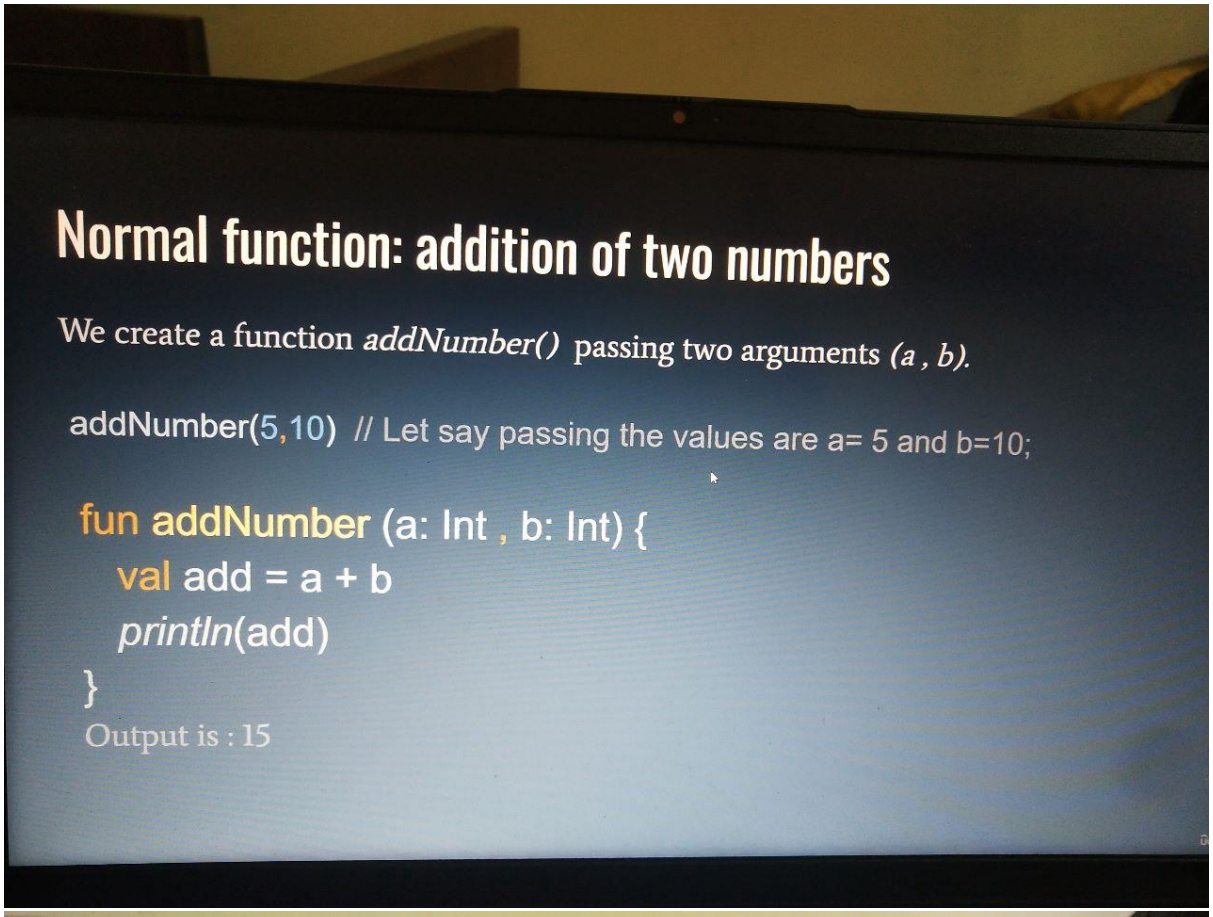
- A **public** modified element is accessible from everywhere in the project.
- It is a **default modifier** in Kotlin. If any class, interface etc. are not specified with any access/visibility modifier then that class, interface etc. is used in a public scope.
- All public declarations can be placed at the top of the file.
- If a member of a class is not specified then it is **by default public**.

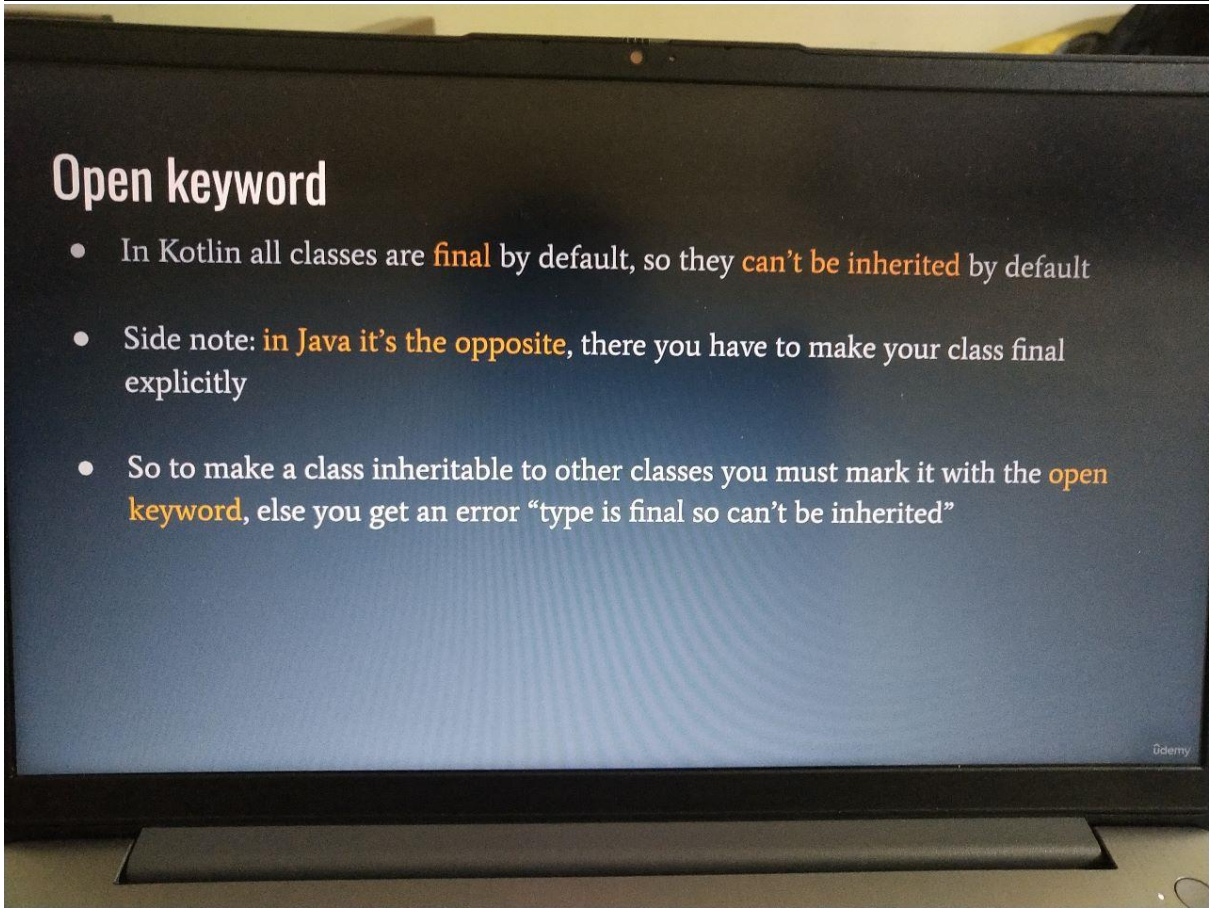

What is a Visibility Modifier?
Visibility modifiers are the keywords which are used to restrict the use of classes, interfaces, methods , and properties in Kotlin.
These modifiers are used at multiple places such as class header or method body.
Visibility Modifiers are categorized into four different types :
public
private
protected
internal

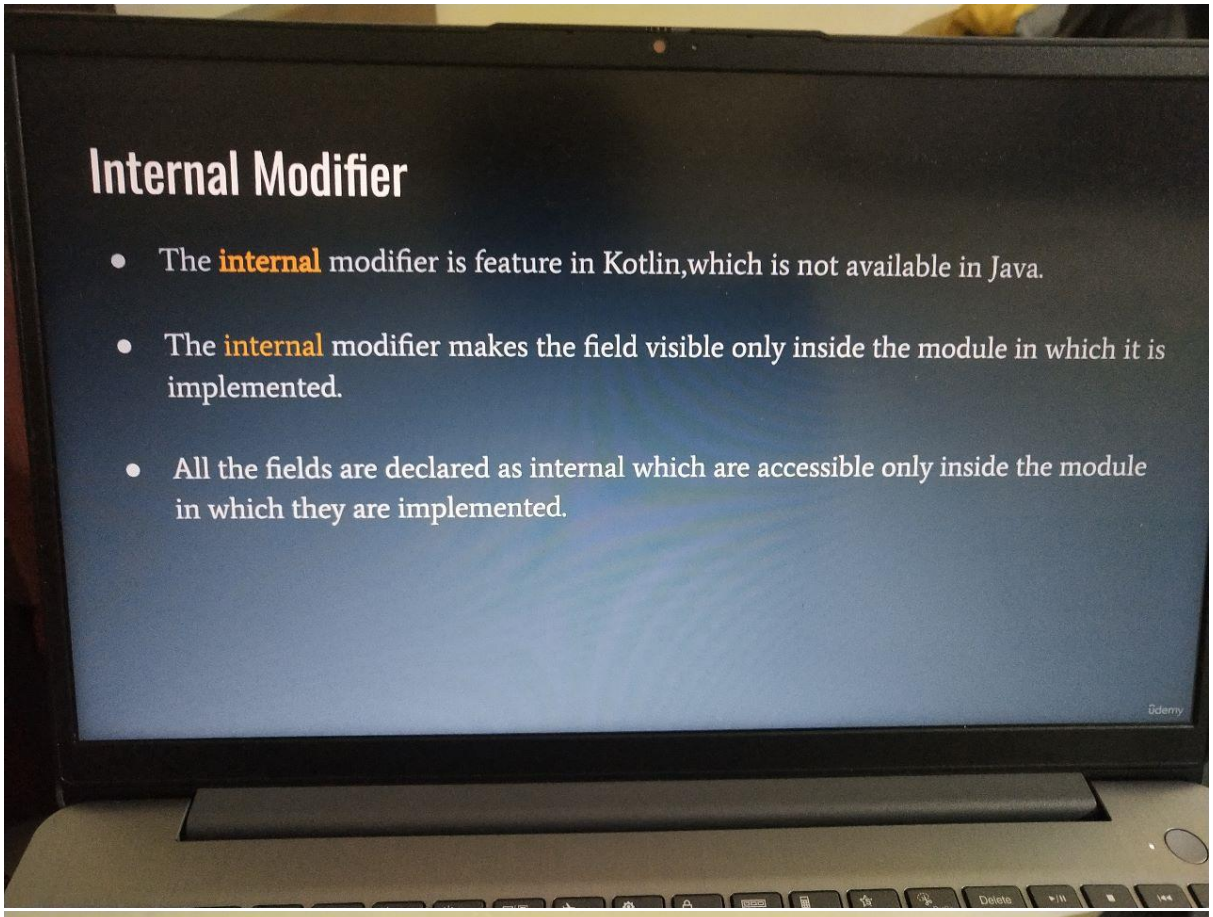

What is Lambda Expression?
Lambda (Expression) is a function which has no name.
Lambda expressions and anonymous functions are 'function literals', i.e. functions that are not declared, but passed immediately as an expression
Lambda is defined with curly braces {} which takes variables as a parameter (if any) and a body of a function.
The body of a function is written after the variable (if any) followed by -> operator.
Syntax : { variable(s) -> body_of_lambda}

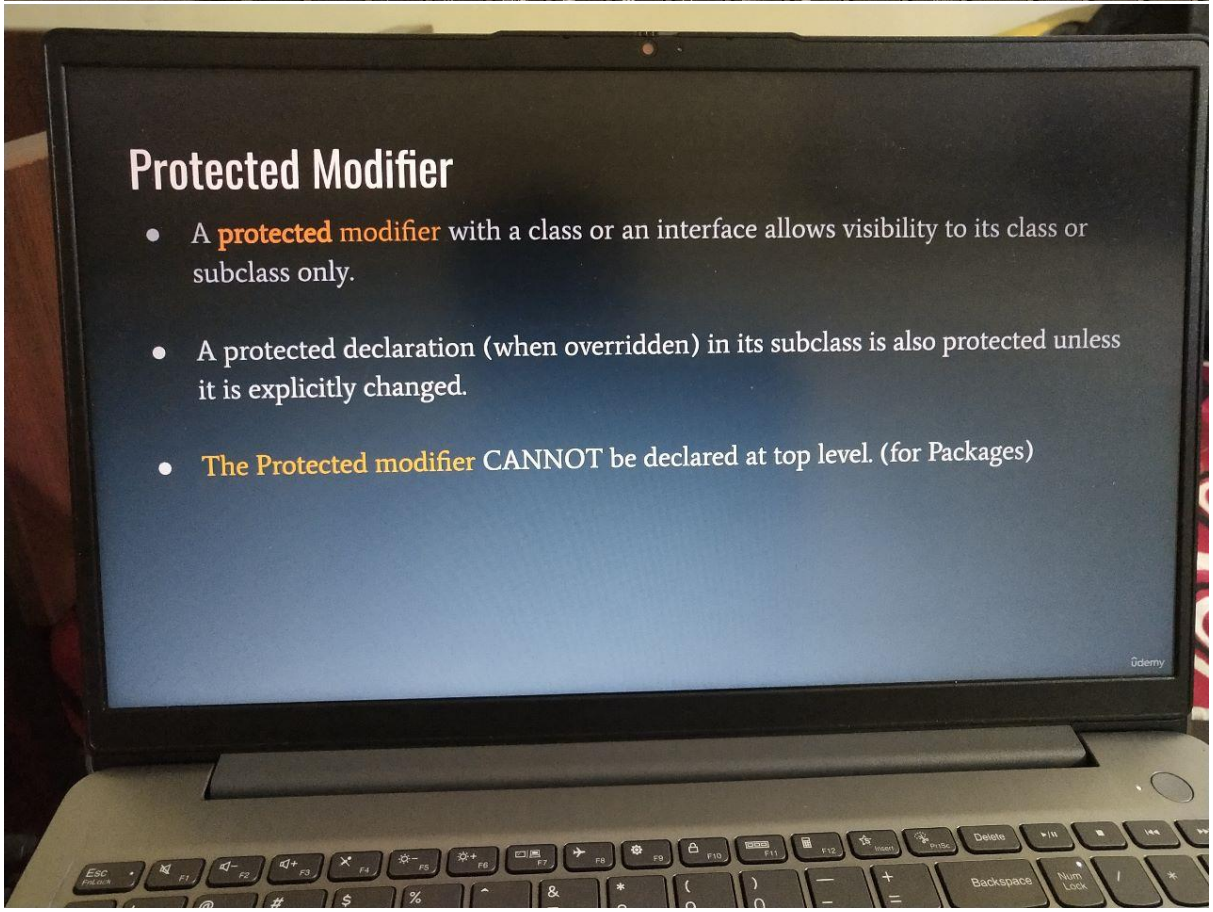

Private Modifier
A private modifier allows the element to be accessible only within the block in which properties, fields, etc. are declared.
The private modifier declaration does not allow access outside the scope.
A private package can be accessible within that specific file.
Udemy


Syntax of Public Modifier
public class Example{
}
class Demo{
}
public fun hello()
fun demo()
public val x = 5
val y = 10


Normal function: addition of two numbers
We create a function addNumber() passing two arguments (a , b).
addNumber(5,10) // Let say passing the values are a= 5 and b=10;
fun addNumber (a: Int , b: Int) {
val add = a + b
println(add)
}
Output is : 15


Open keyword
In Kotlin all classes are final by default, so they can't be inherited by default
Side note: in Java it's the opposite, there you have to make your class final explicitly
So to make a class inheritable to other classes you must mark it with the open keyword, else you get an error "type is final so can't be inherited"
Udemy


Internal Modifier
The internal modifier is feature in Kotlin,which is not available in Java.
The internal modifier makes the field visible only inside the module in which it is implemented.
All the fields are declared as internal which are accessible only inside the module in which they are implemented.
Udemy


Protected Modifier
A protected modifier with a class or an interface allows visibility to its class or subclass only.
A protected declaration (when overridden) in its subclass is also protected unless it is explicitly changed.
The Protected modifier CANNOT be declared at top level. (for Packages)
Udemy

Syntax Internal Modifier

```
internal class Example {
    internal val x = 5

    internal fun getValue(){

        }
    }

internal val y = 10
```

Syntax of Protected Modifier

```
open class Base {
   protected val i = 0
}

class Derived : Base(){

   fun getValue() : Int  {
      return i
   }
}
```

Syntax Private Modifier

```
private class Example {

    private val x = 1

    private doSomething() {
      }
    }
```

- In above class Example, val x and function doSomething() are declared as private. The class "Example" is accessible from the same source file, "val x" and "fun doSomething()" are accessible within Example class.

Nested class

- **Nested class** is such class which is created inside another class.
- In Kotlin, a nested class is by default **static**, so its data members and member functions can be accessed without creating an object of the class.
- Nested classes cannot access the data members of outer classes.

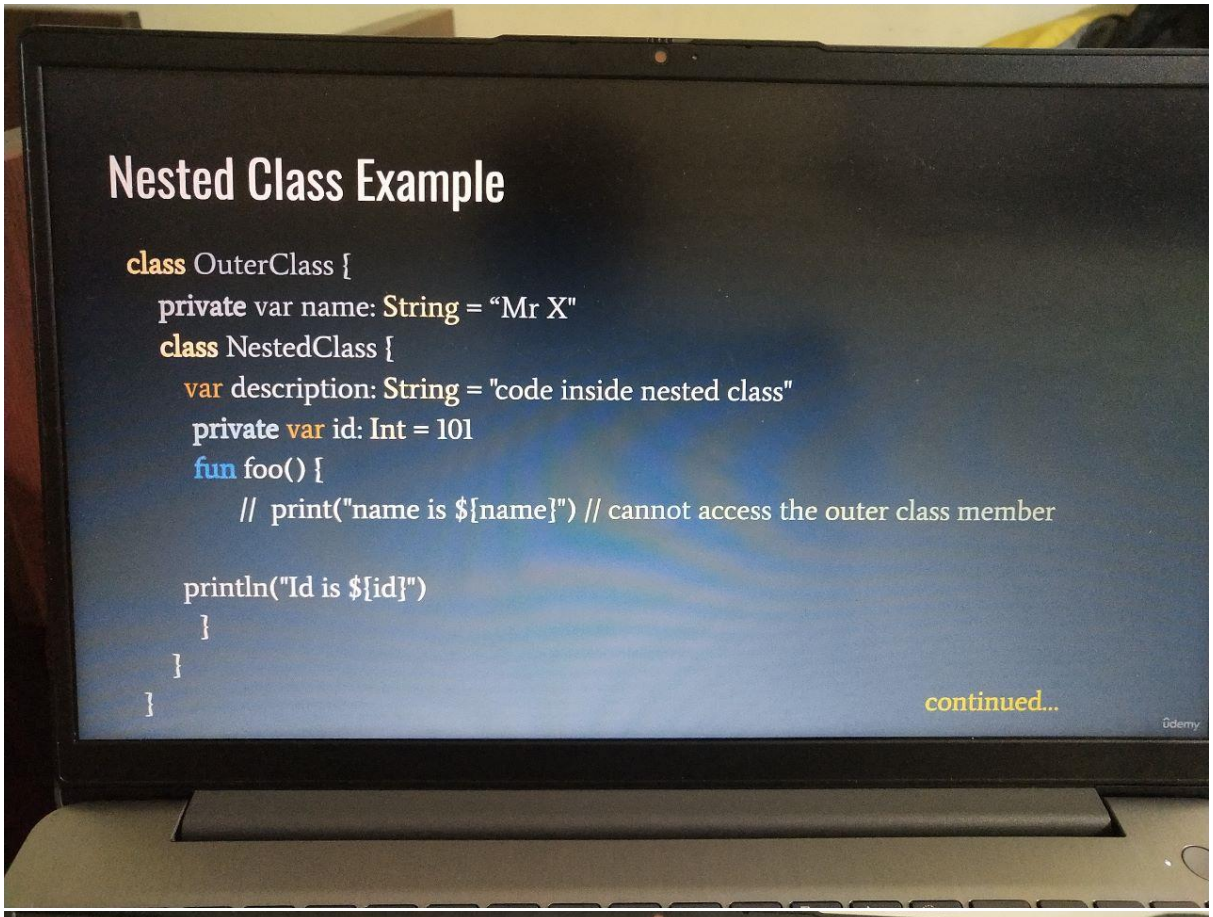
Nested Class Example

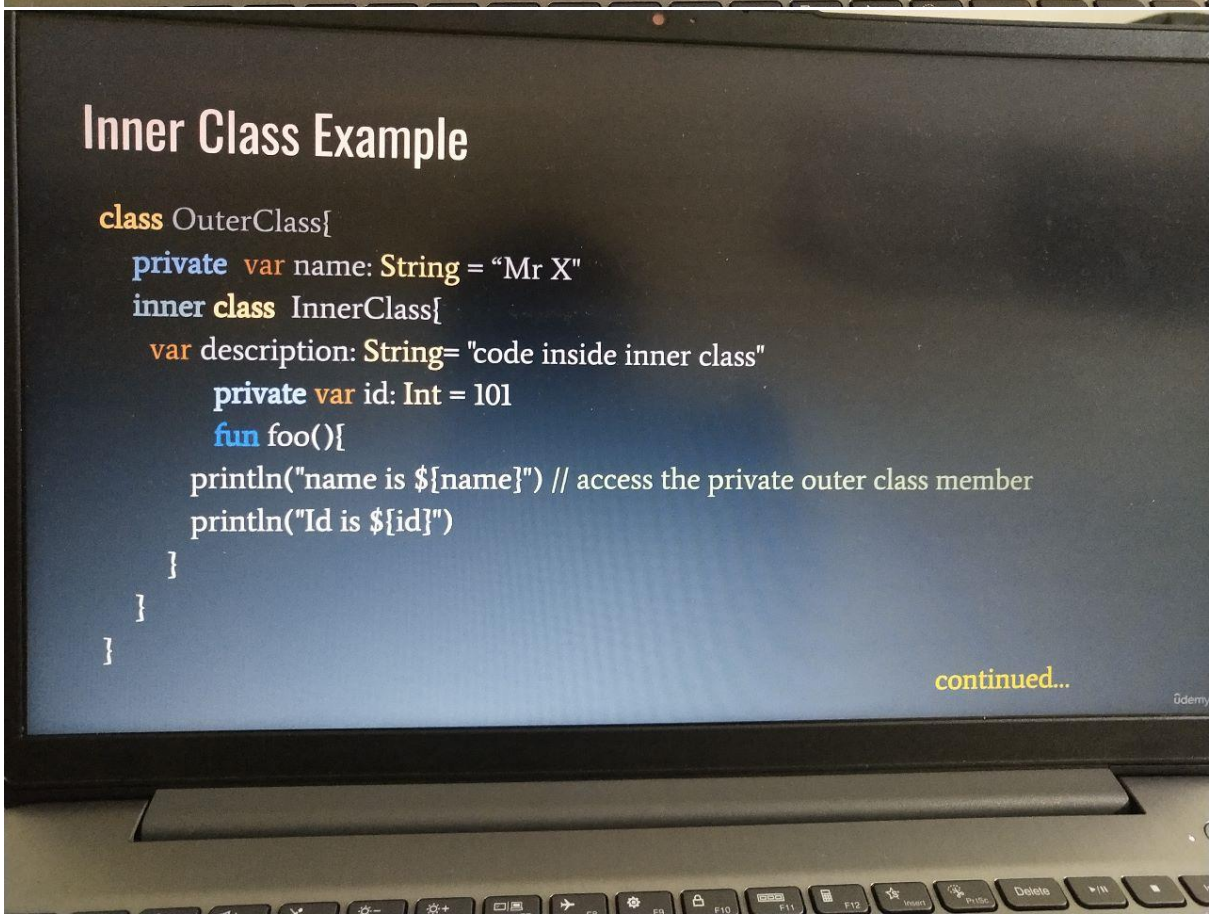
```
fun main(args: Array<String>){
        // nested class must be initialized
 println(OuterClass.NestedClass().description) // accessing property

 var obj = OuterClass.NestedClass() // object creation
   obj.foo() // access member function
 }
```

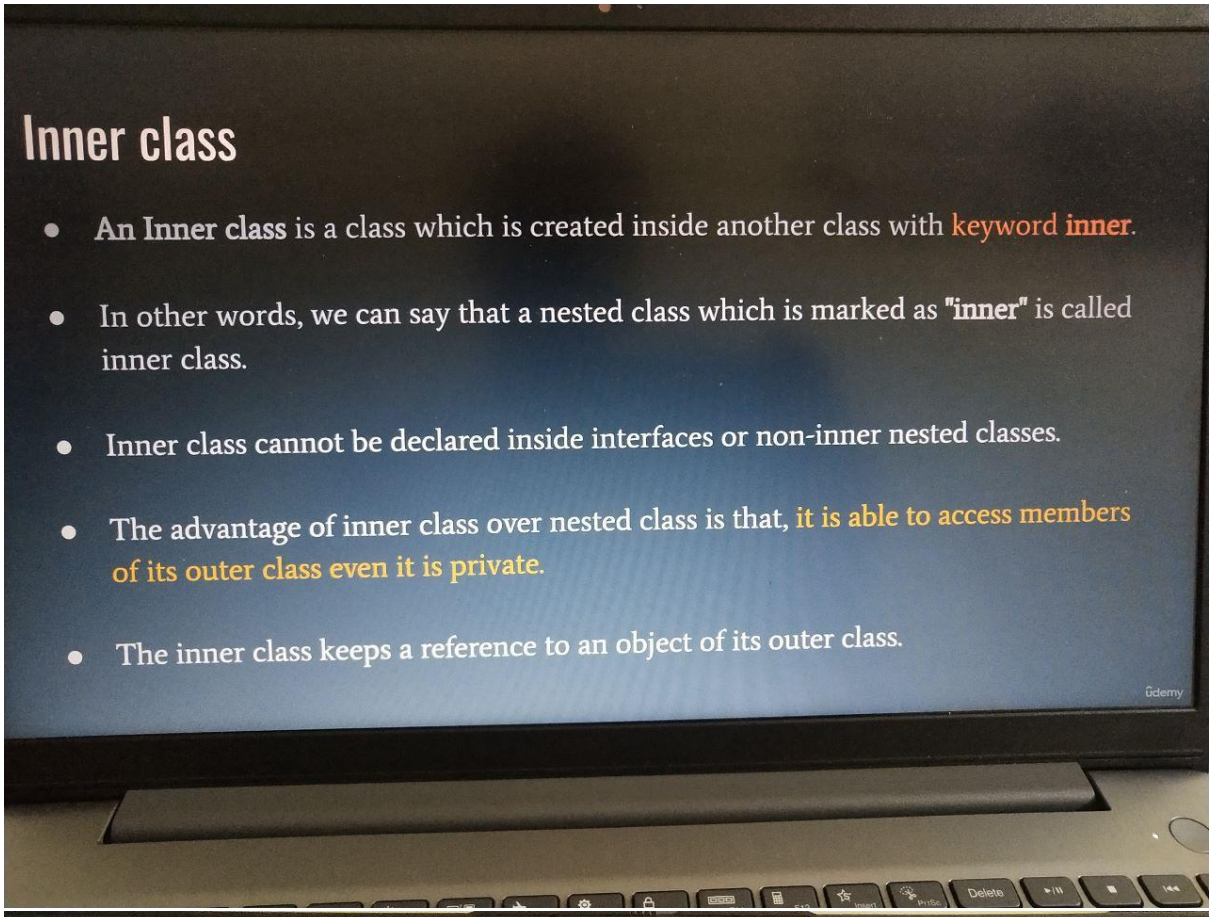
Output : code inside nested class
Id is 101

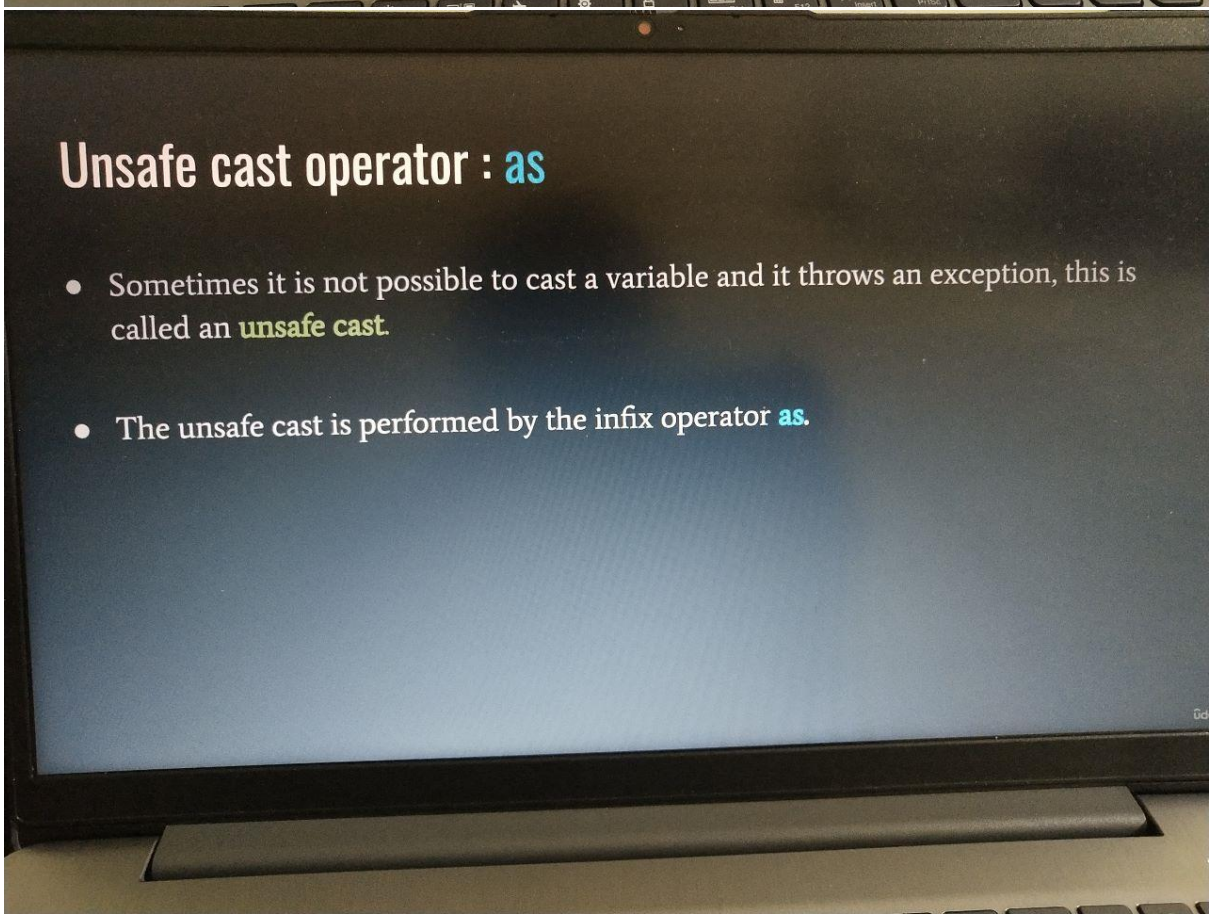
nple of Visibility Modifier

```
open class Base() {
var a = 1 // public by default
   private var b = 2 // private to Base class
   protected open val c = 3  // visible to the Base and the Derived class
   internal val d = 4 // visible inside the same module
   protected fun e() { } // visible to the Base and the Derived class
}

class Derived: Base() {
   // a, c, d, and e() of the Base class are visible
   // b is not visible
   override val c = 9 // c is protected
}

fun main(args: Array<String>) {
val base = Base()
   // base.a and base.d are visible
   // base.b, base.c and base.e() are not visible
val derived = Derived()
   // derived.c is not visible
}
```


Nested Class Example
class OuterClass {
private var name: String = “Mr X"
class NestedClass {
var description: String = "code inside nested class"
private var id: Int = 101
fun foo() {
// print("name is ${name}") // cannot access the outer class member
println("Id is ${id}")
}
}
}
continued...


Inner Class Example
class OuterClass{
private var name: String = “Mr X"
inner class InnerClass{
var description: String= "code inside inner class"
private var id: Int = 101
fun foo(){
println("name is ${name}") // access the private outer class member
println("Id is ${id}")
}
}
}
continued...


Inner class
An Inner class is a class which is created inside another class with keyword inner.
In other words, we can say that a nested class which is marked as "inner" is called inner class.
Inner class cannot be declared inside interfaces or non-inner nested classes.
The advantage of inner class over nested class is that, it is able to access members of its outer class even it is private.
The inner class keeps a reference to an object of its outer class.
Udemy


Unsafe cast operator : as
Sometimes it is not possible to cast a variable and it throws an exception, this is called an unsafe cast.
The unsafe cast is performed by the infix operator as.

Inner Class Example

```
fun main(args: Array< String >){

    println(OuterClass().InnerClass().description) // accessing property

    var obj = OuterClass().InnerClass() // object creation

    obj.foo() // access member function
}
```

Output : code inside inner class
name is Mr X

Example :

- A nullable string (String?) cannot be cast to non nullable string (String), this throws an exception.

```
fun main(args: Array<String>){
    val obj: Any? = null
    val str: String = obj as String
    println(str)
}

// Output :
Exception in thread "main" kotlin.TypeCastException: null cannot be cast to non-null type kotlin.
String .
```

Nullable for Casting to work :

- Source and target variable needs to be a nullable for casting to work:

```
fun main(args: Array<String>){
    val obj: Any? = "String unsafe cast"
    val str: String? = obj as String?     // Works
    println(str)
}

// Output : String unsafe cast
```


Safe cast operator: as?

- as? provides a safe cast operation to safely cast to a type.
- It returns a null if casting is not possible rather than throwing an ClassCastException exception.

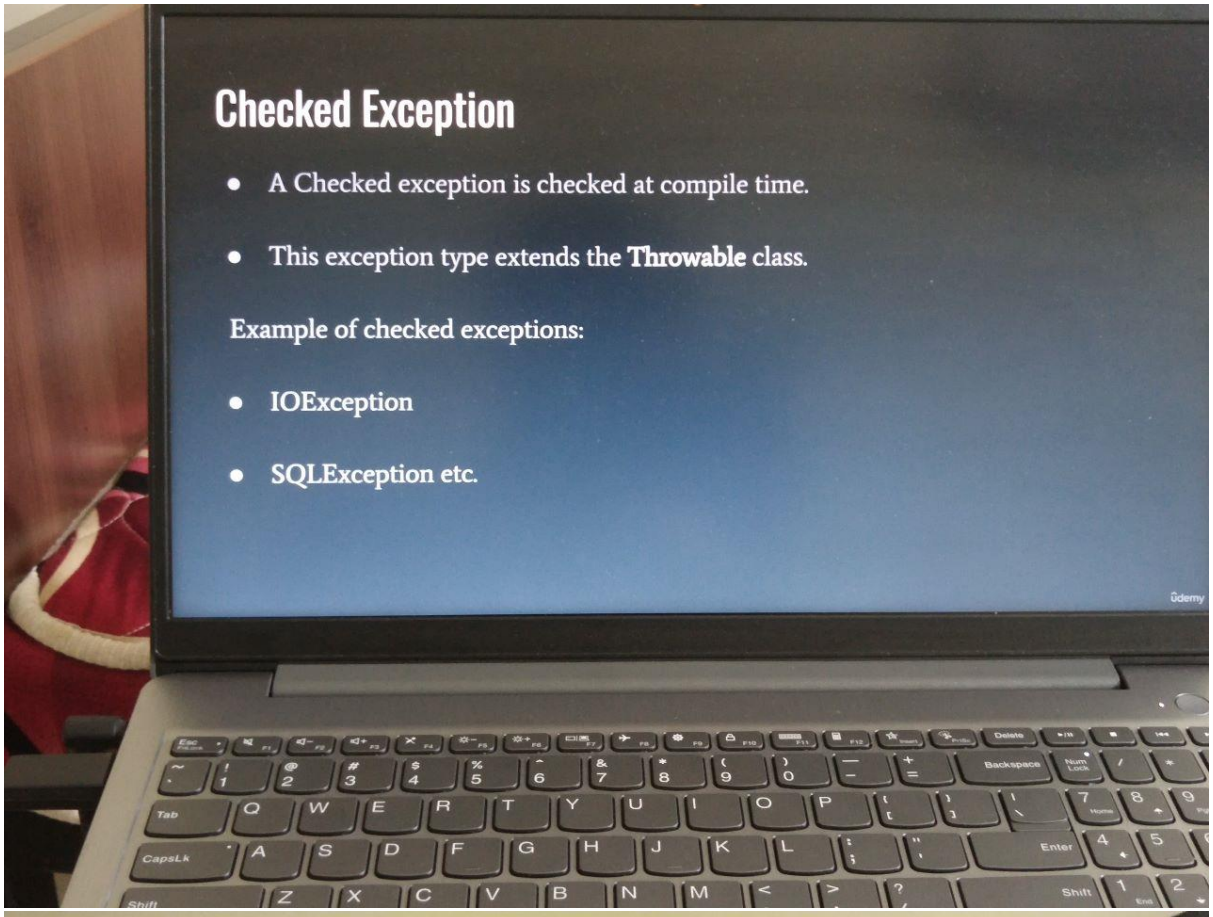
Keywords used in exception handling :

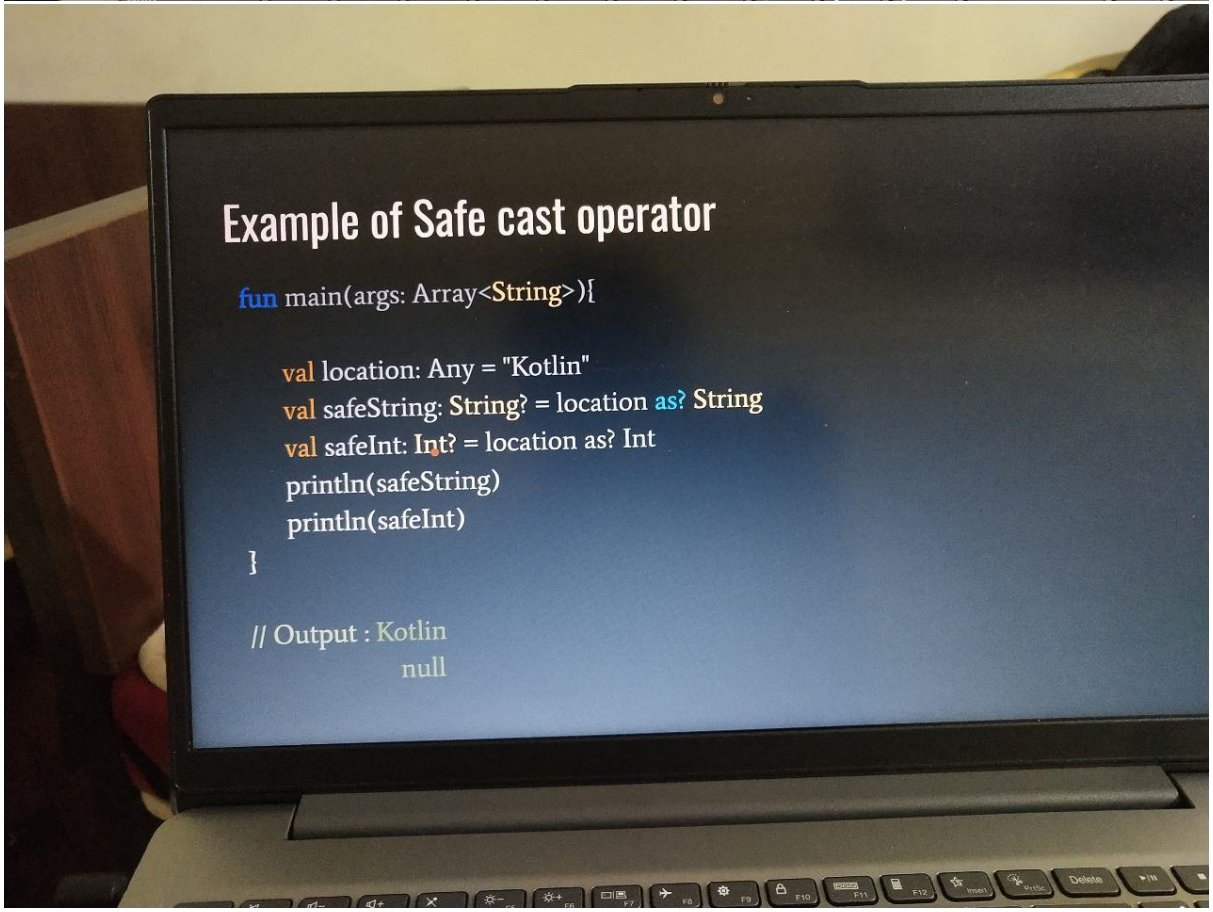
- **try** : the try block contains a set of statements which might generate an exception. It must be followed by either catch or finally or both.
- **catch** : the catch block is used to catch the exception thrown from try block.
- **finally** : finally block always execute whether exception is handled or not. So it is used to execute important code statement. (like closing buffers)
- **throw** : the throw keyword is used to throw an exception explicitly.

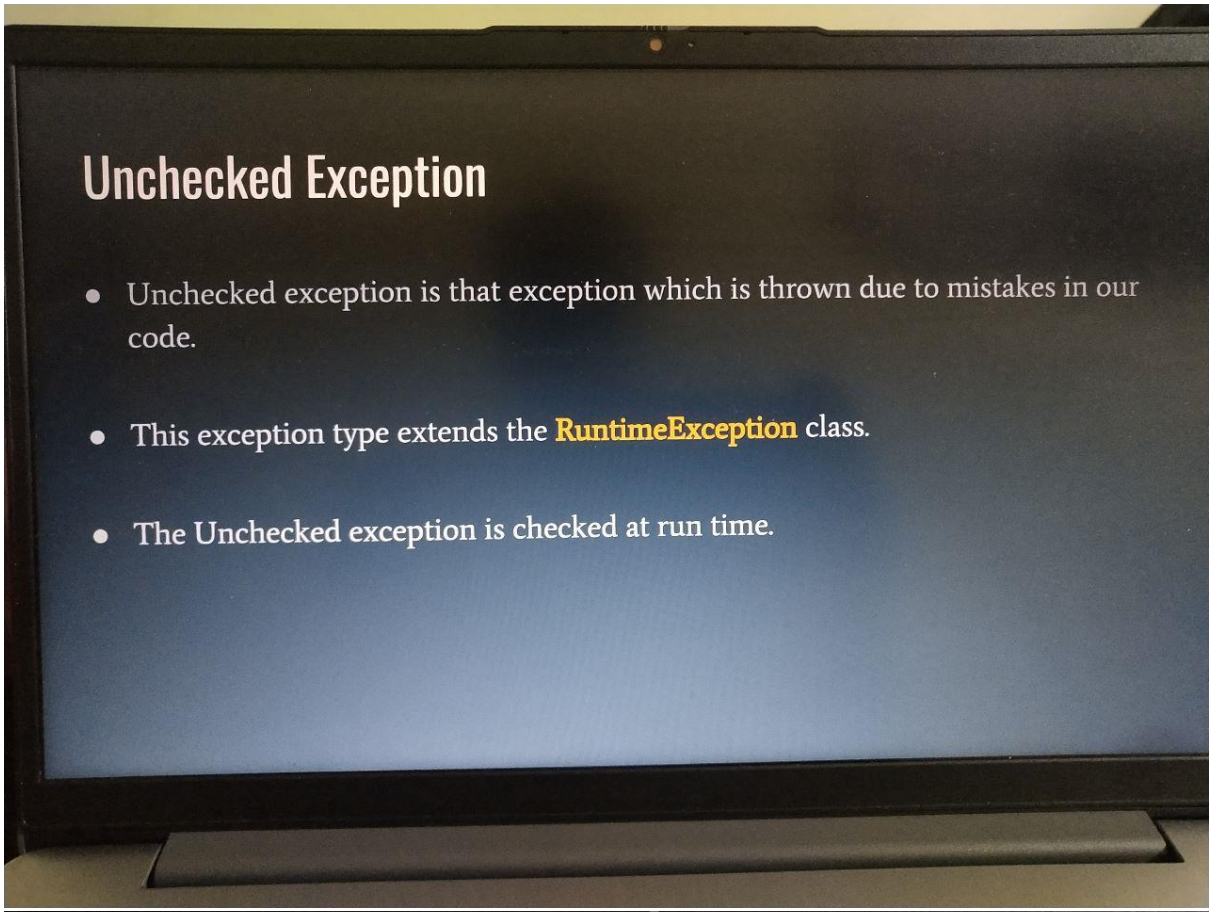
Generates a ClassCastException

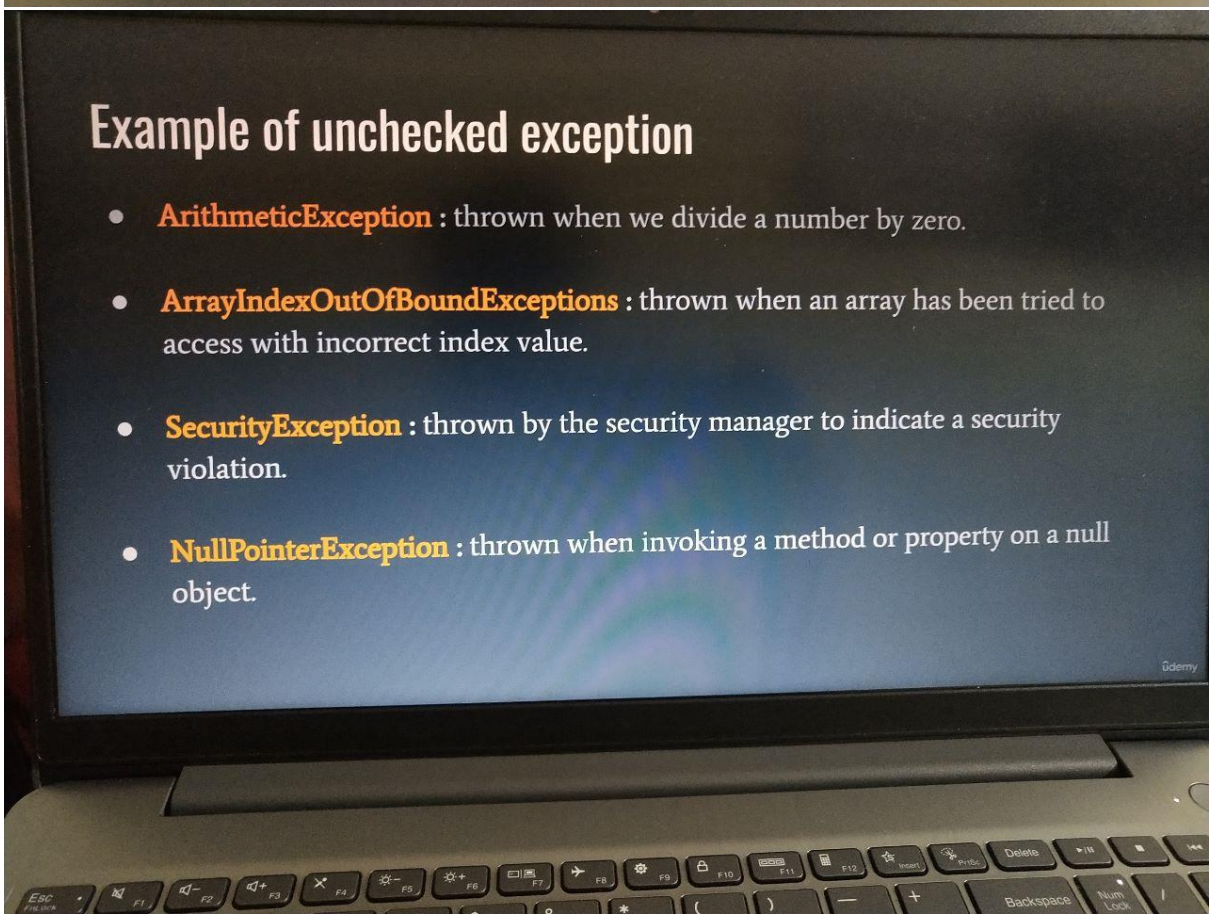
- Trying to cast an integer value of the Any type into a string type leads to a ClassCastException.

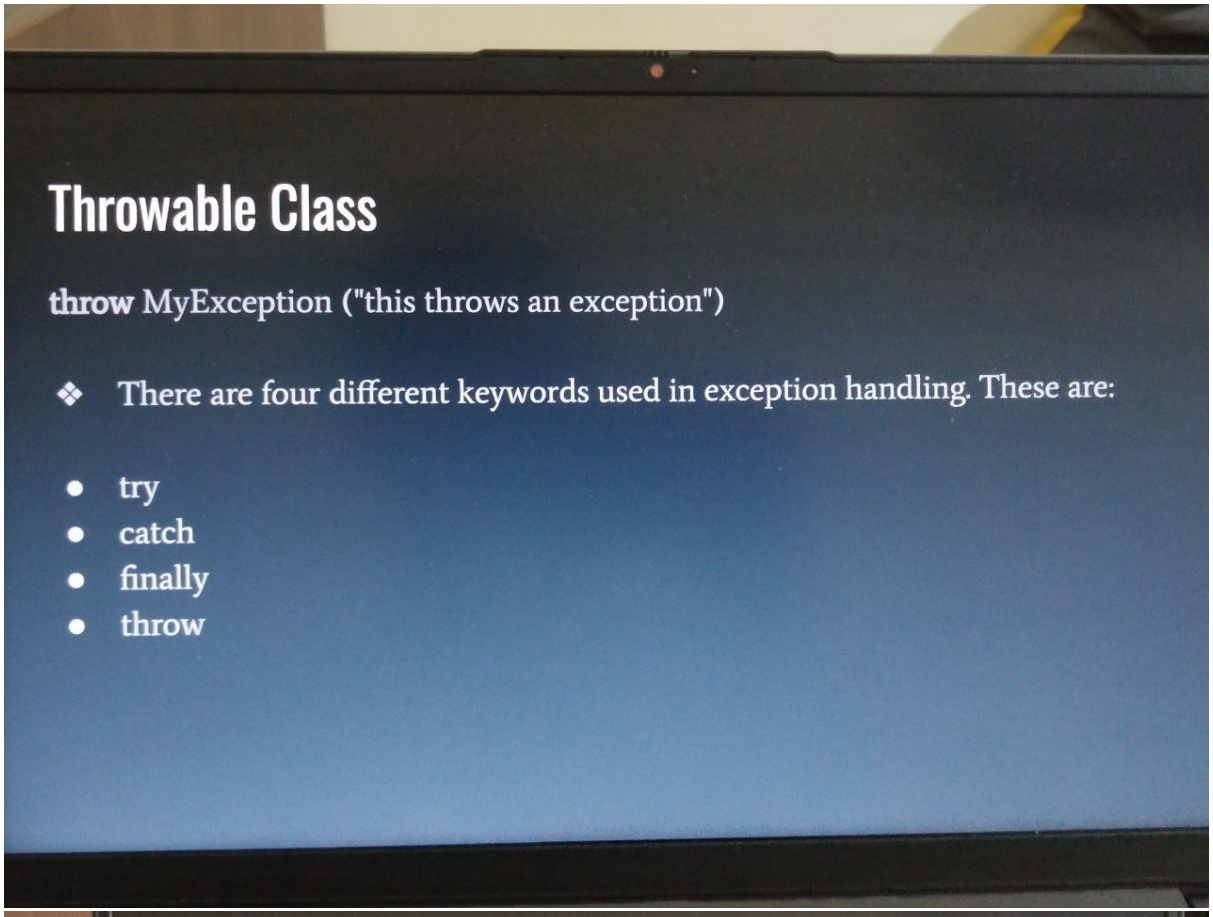
```
val obj: Any = 123

val str: String = obj as String

// Throws java.lang.ClassCastException: java.lang.Integer cannot be cast to
java.lang.String
```

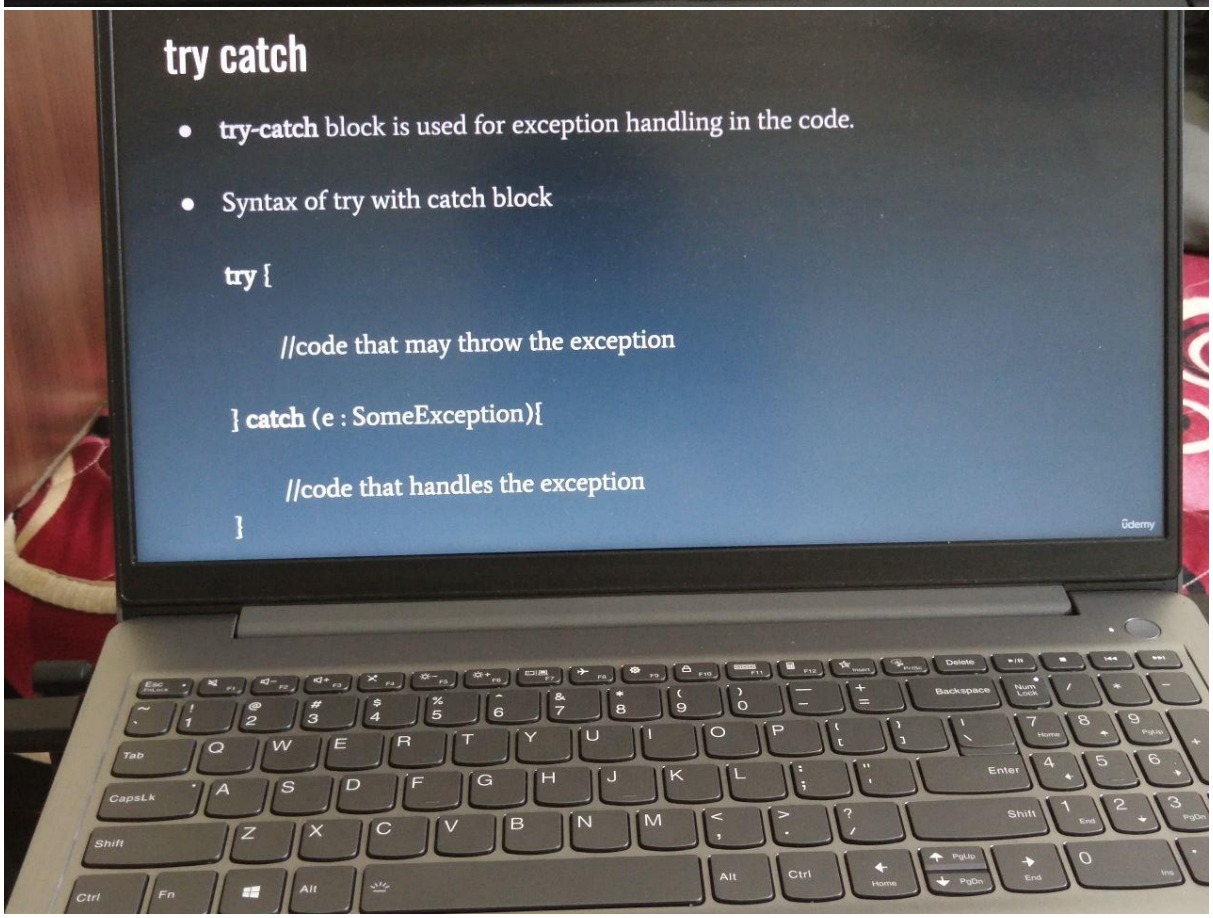

Checked Exception
A Checked exception is checked at compile time.
This exception type extends the Throwable class.
Example of checked exceptions:
IOException
SQLException etc.
Udemy


Example of Safe cast operator
fun main(args: Array<String>){
val location: Any = "Kotlin"
val safeString: String? = location as? String
val safeInt: Int? = location as? Int
println(safeString)
println(safeInt)
}
// Output : Kotlin
null


Unchecked Exception
Unchecked exception is that exception which is thrown due to mistakes in our code.
This exception type extends the RuntimeException class.
The Unchecked exception is checked at run time.

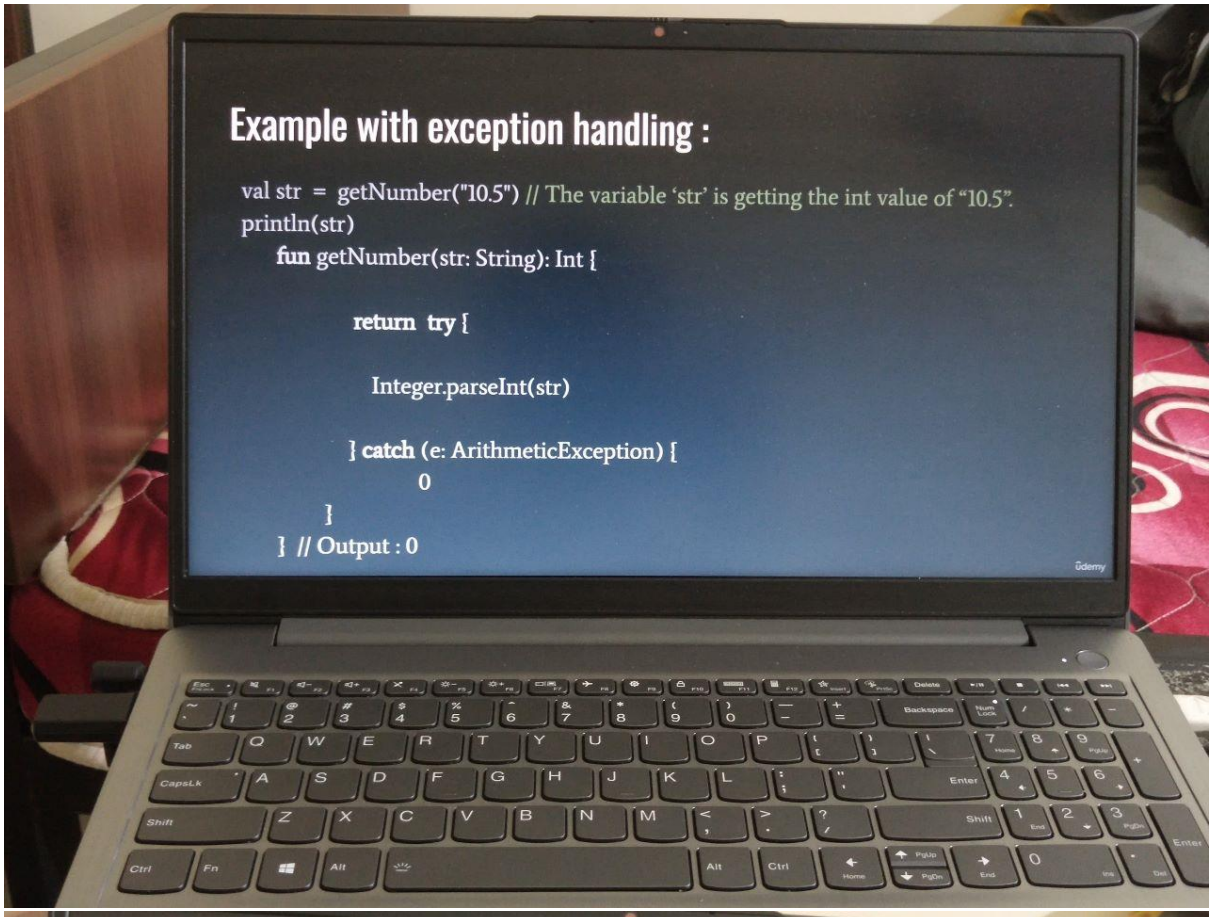

Example of unchecked exception
ArithmeticException : thrown when we divide a number by zero.
ArrayIndexOutOfBoundExceptions : thrown when an array has been tried to access with incorrect index value.
SecurityException : thrown by the security manager to indicate a security violation.
NullPointerException : thrown when invoking a method or property on a null object.
Udemy


Throwable Class
throw MyException ("this throws an exception")
There are four different keywords used in exception handling. These are:
try
catch
finally
throw


try catch
try-catch block is used for exception handling in the code.
Syntax of try with catch block
try {
//code that may throw the exception
} catch (e : SomeException){
//code that handles the exception
}
Udemy

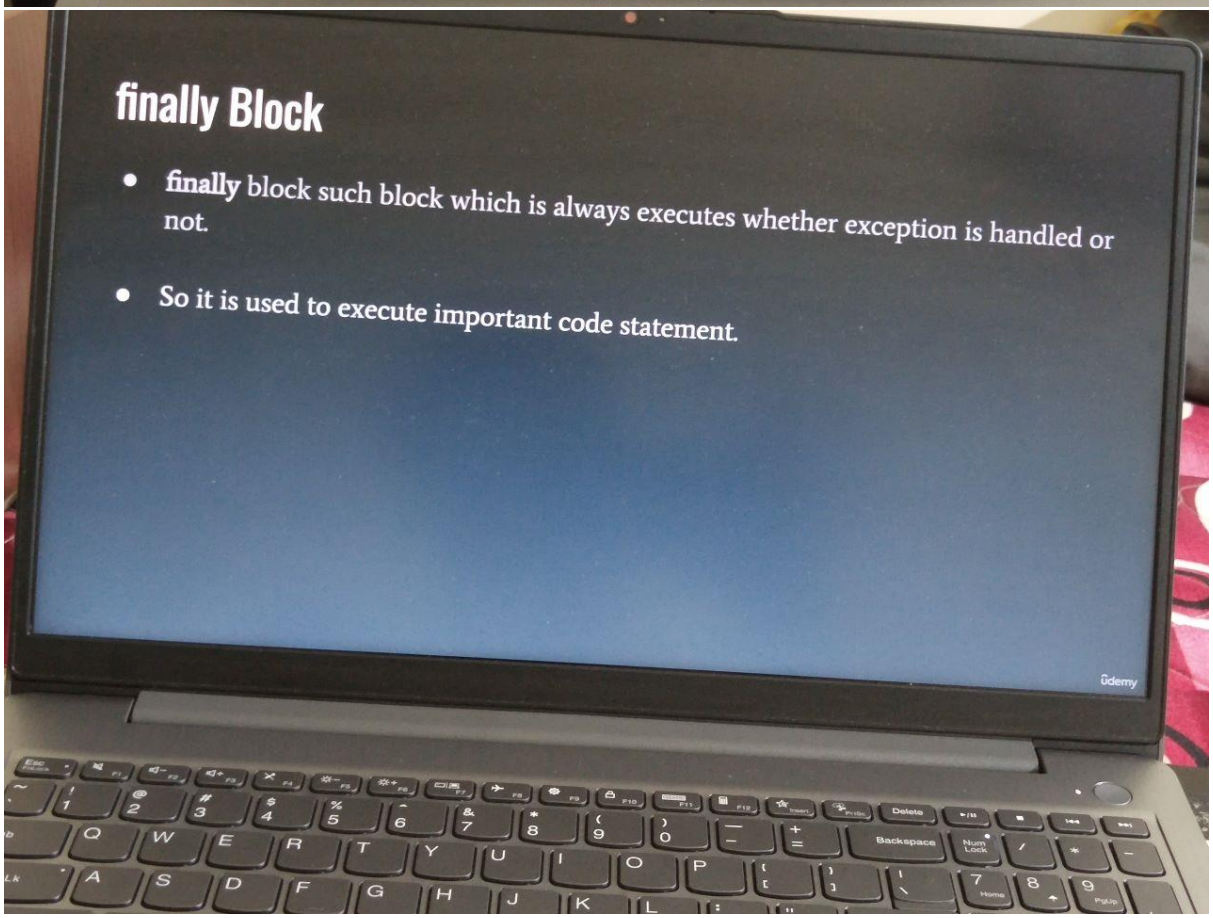
Example with exception handling :

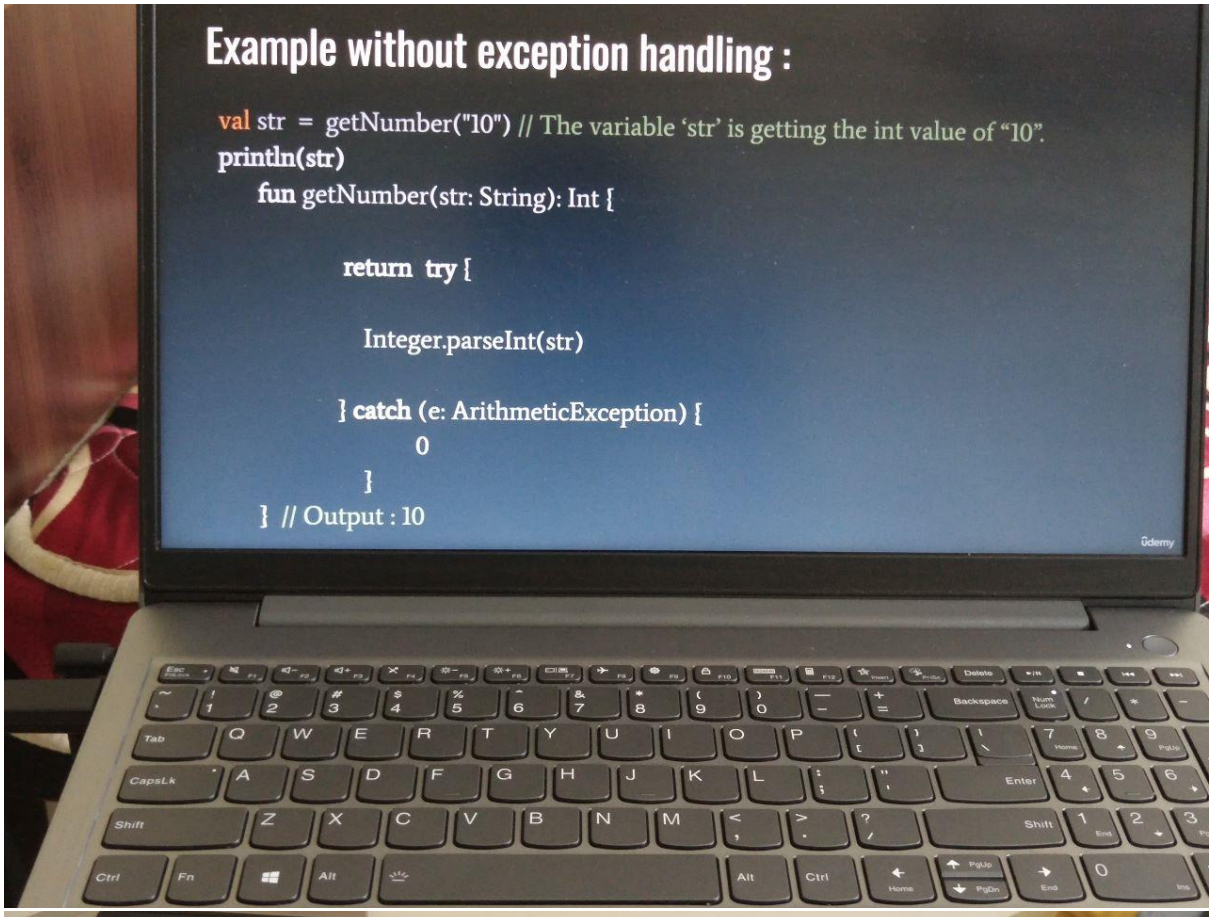
```
val str  =  getNumber("10.5") // The variable 'str' is getting the int value of "10.5".
println(str)
    fun getNumber(str: String): Int {

            return  try {

              Integer.parseInt(str)

            } catch (e: ArithmeticException) {
                    0
        }
    }  // Output : 0
```

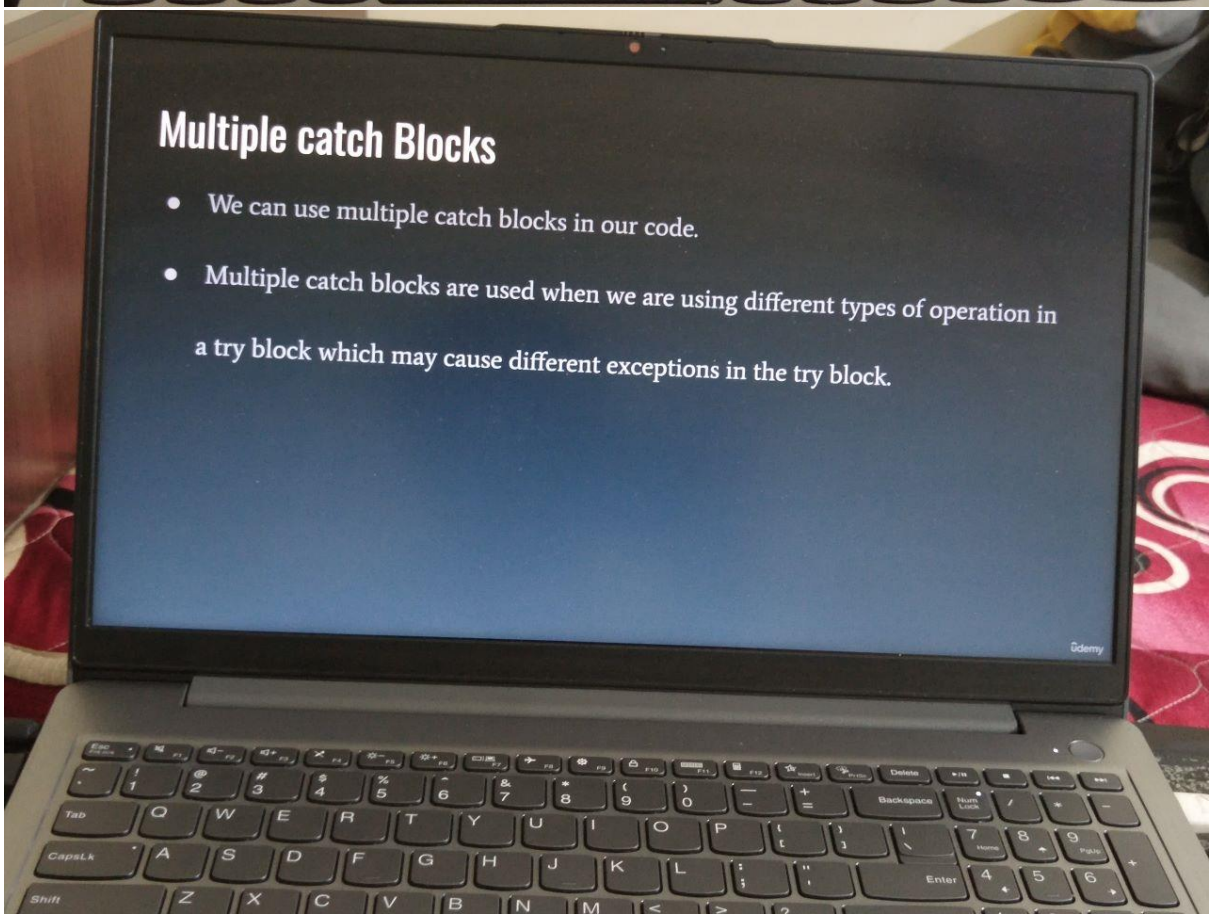


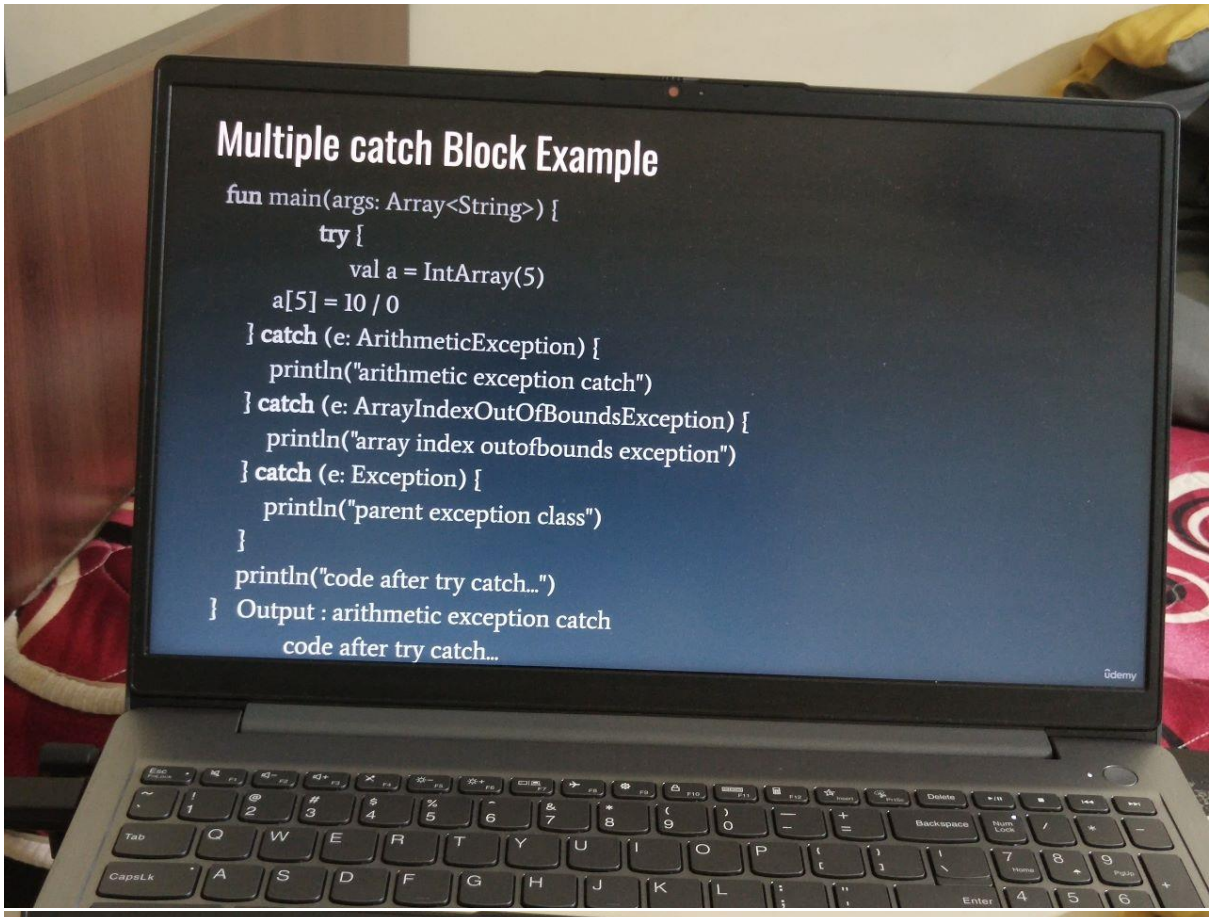
finally Block

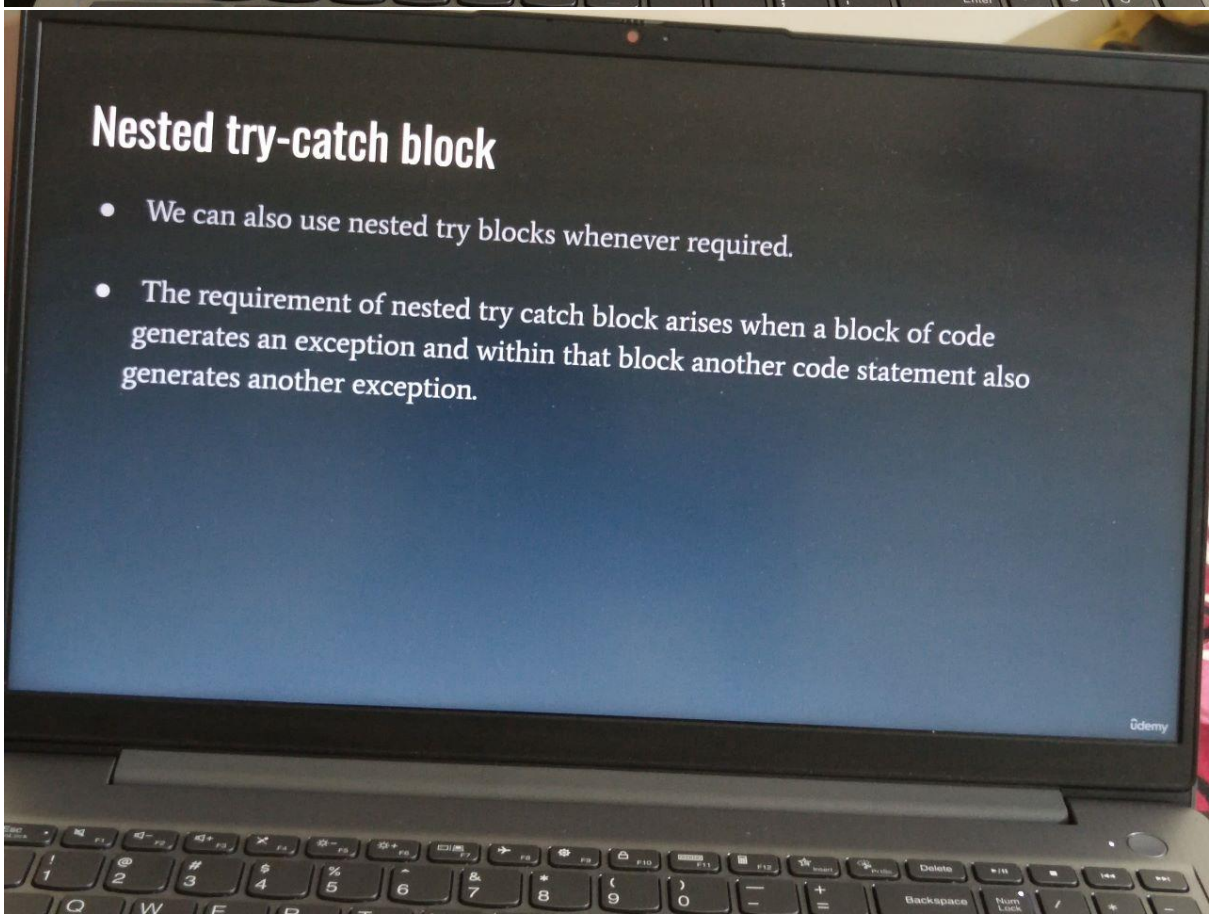
- **finally** block such block which is always executes whether exception is handled or not.
- So it is used to execute important code statement.

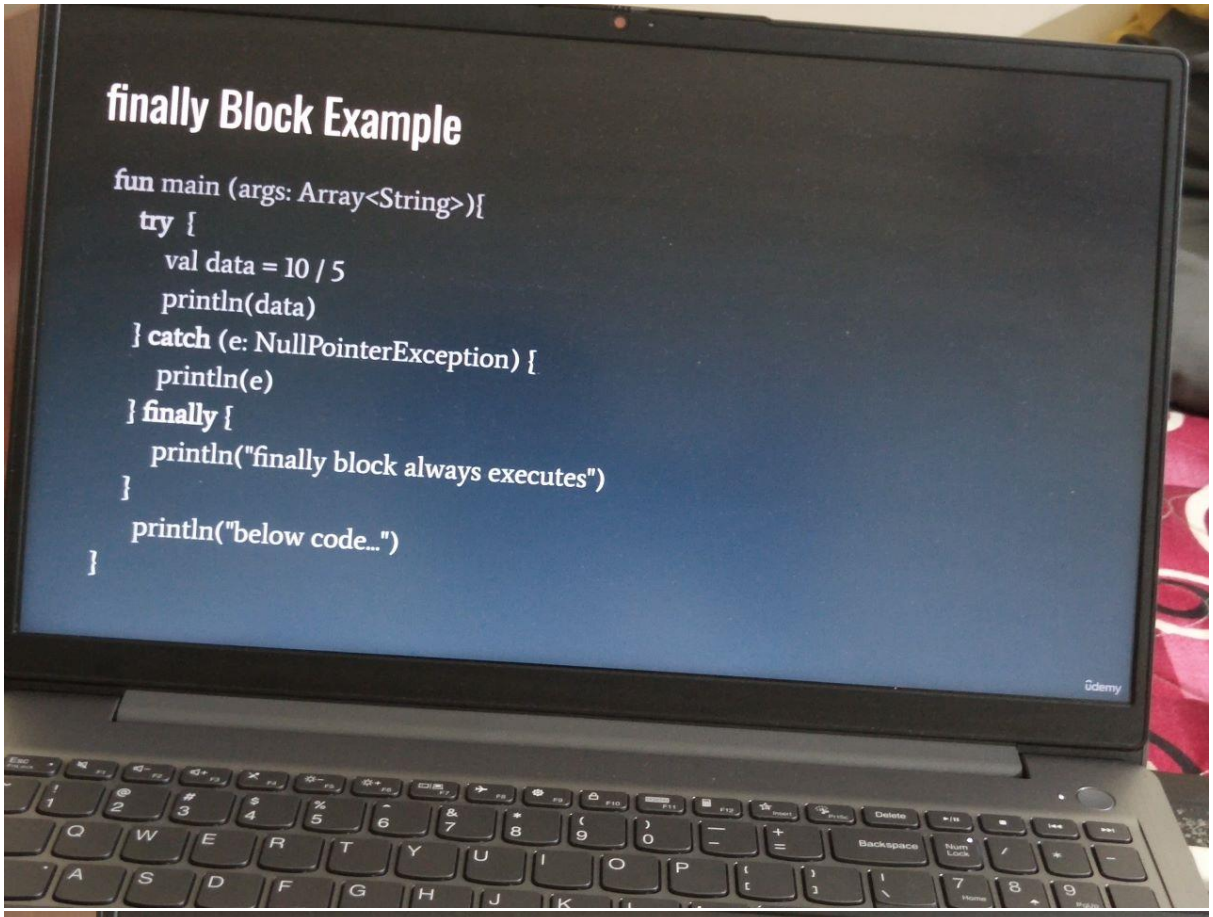


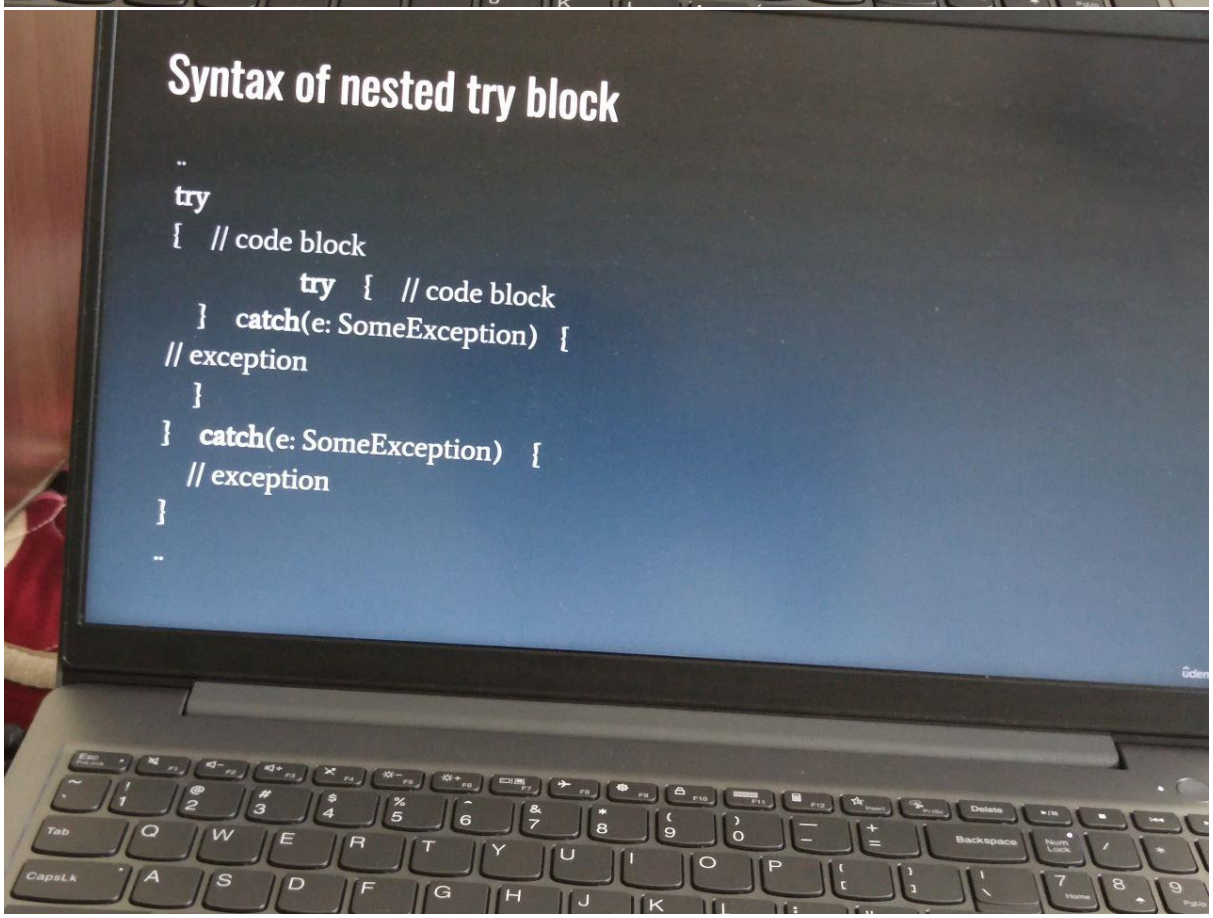

Example without exception handling :
val str = getNumber("10") // The variable 'str' is getting the int value of "10".
println(str)
fun getNumber(str: String): Int {
return try {
Integer.parseInt(str)
} catch (e: ArithmeticException) {
0
}
} // Output : 10
Udemy

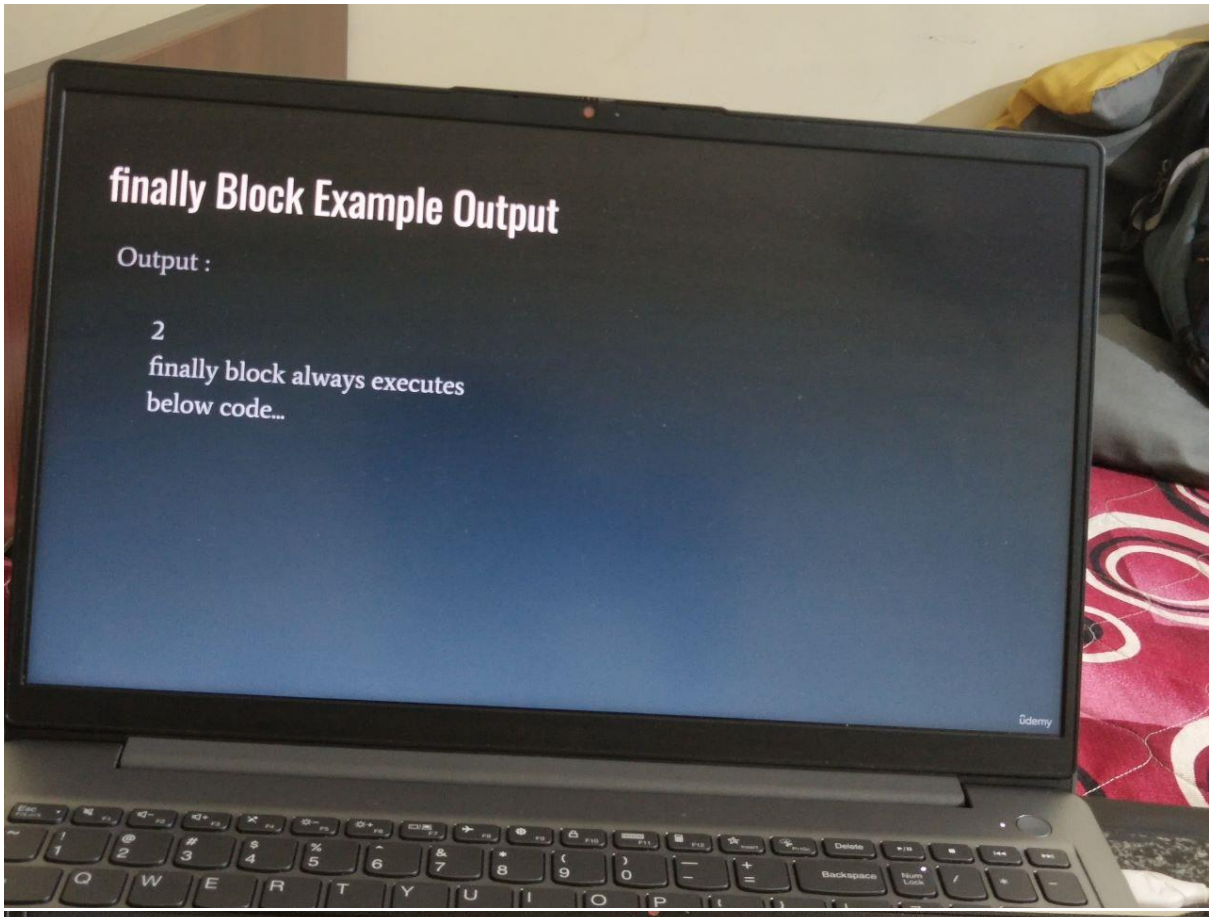

Multiple catch Blocks
We can use multiple catch blocks in our code.
Multiple catch blocks are used when we are using different types of operation in a try block which may cause different exceptions in the try block.
Udemy

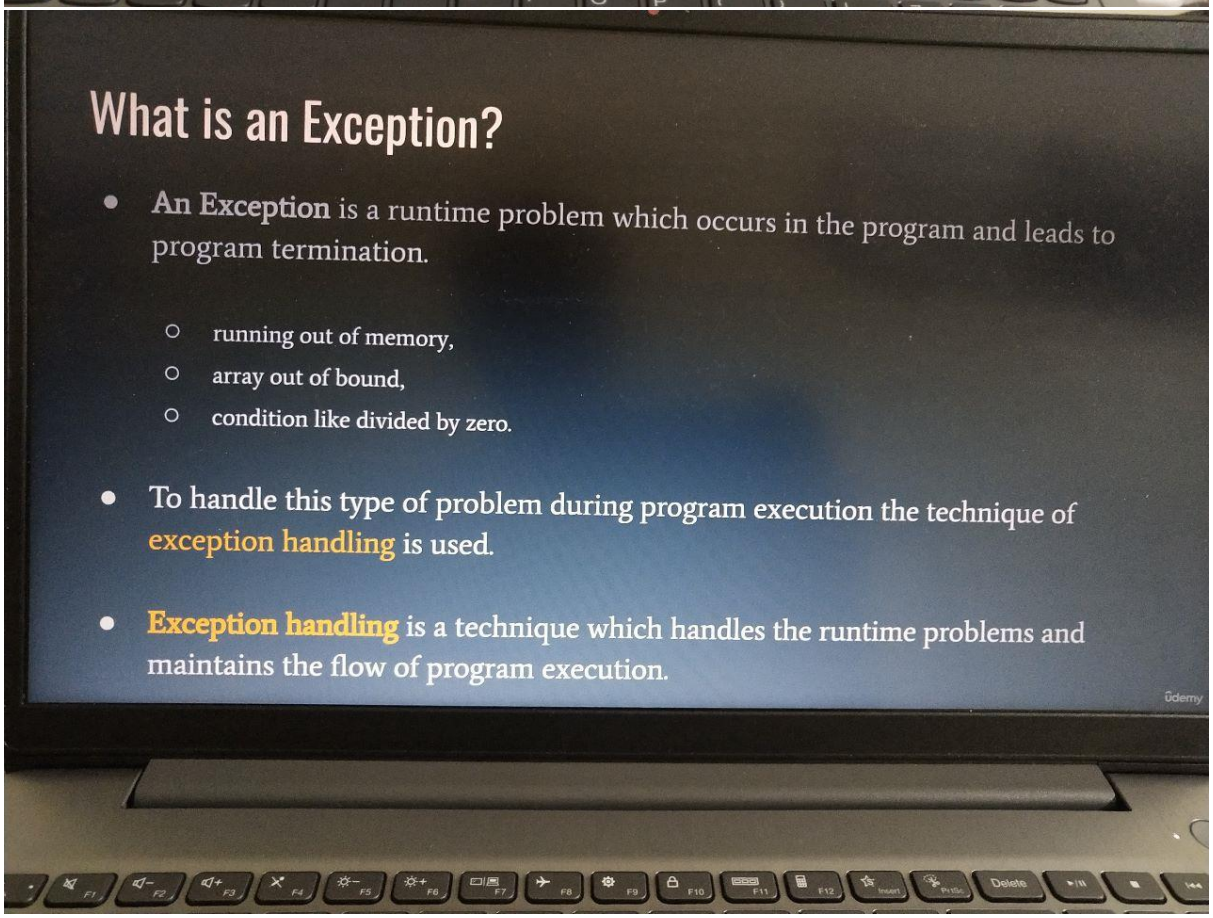

Multiple catch Block Example
fun main(args: Array<String>) {
try {
val a = IntArray(5)
a[5] = 10 / 0
} catch (e: ArithmeticException) {
println("arithmetic exception catch")
} catch (e: ArrayIndexOutOfBoundsException) {
println("array index outofbounds exception")
} catch (e: Exception) {
println("parent exception class")
}
println("code after try catch...")
}
Output : arithmetic exception catch
code after try catch...
Udemy

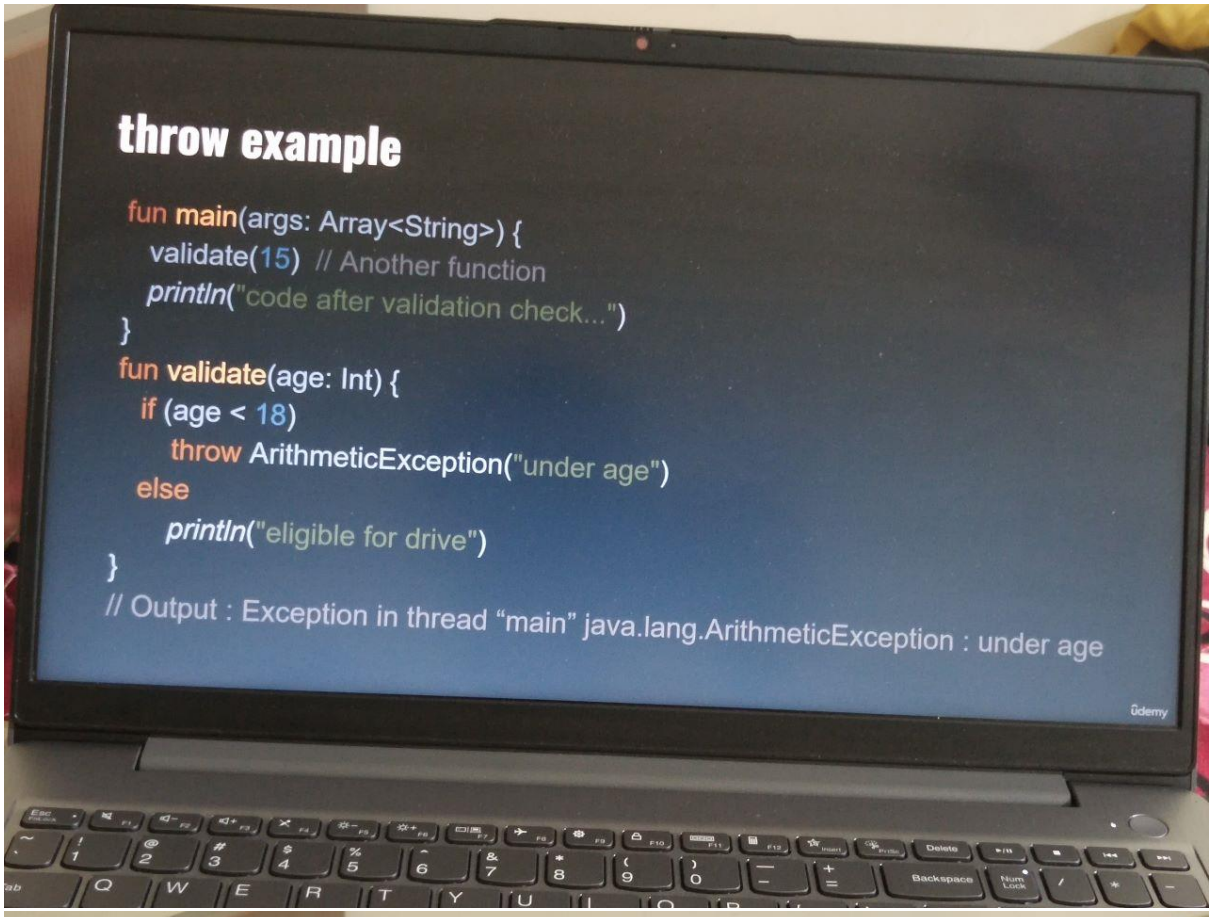

Nested try-catch block
We can also use nested try blocks whenever required.
The requirement of nested try catch block arises when a block of code generates an exception and within that block another code statement also generates another exception.
Udemy

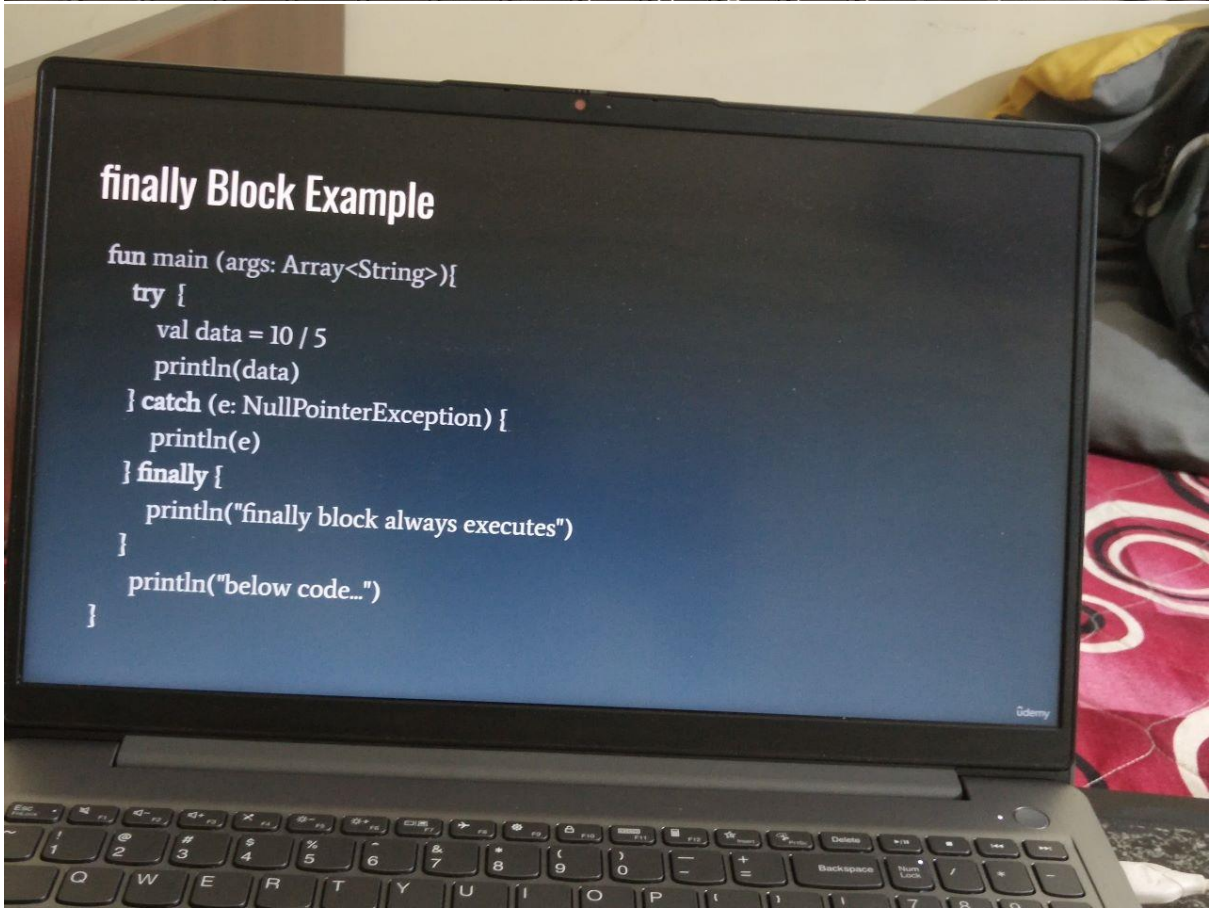

finally Block Example
fun main (args: Array<String>){
try {
val data = 10 / 5
println(data)
} catch (e: NullPointerException) {
println(e)
} finally {
println("finally block always executes")
}
println("below code...")
}

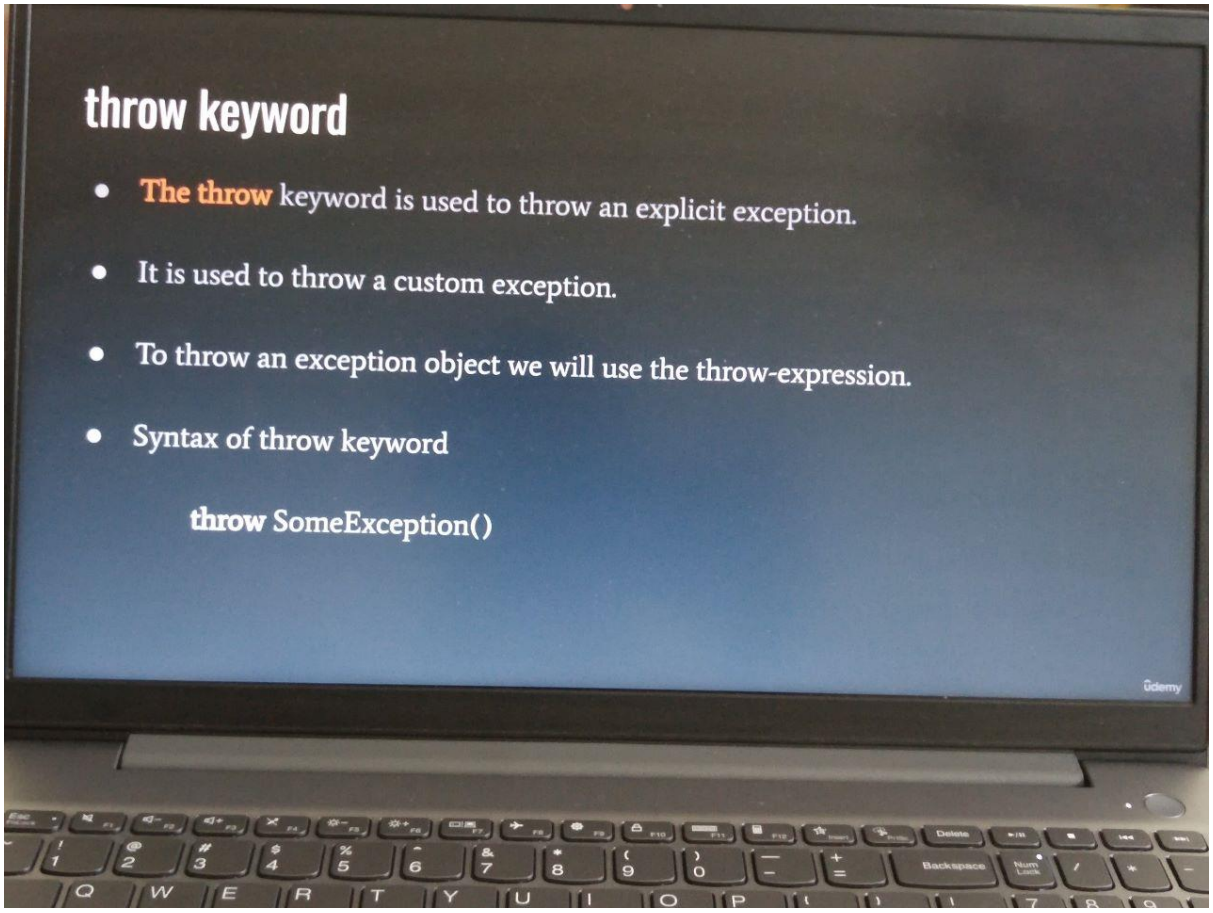

Syntax of nested try block
..
try
{ // code block
try { // code block
} catch(e: SomeException) {
// exception
}
} catch(e: SomeException) {
// exception
}
..


finally Block Example Output
Output :
2
finally block always executes
below code...
Udemy


What is an Exception?
An Exception is a runtime problem which occurs in the program and leads to program termination.
running out of memory,
array out of bound,
condition like divided by zero.
To handle this type of problem during program execution the technique of exception handling is used.
Exception handling is a technique which handles the runtime problems and maintains the flow of program execution.
Udemy


throw example
fun main(args: Array<String>) {
validate(15) // Another function
println("code after validation check...")
}
fun validate(age: Int) {
if (age < 18)
throw ArithmeticException("under age")
else
println("eligible for drive")
}
// Output : Exception in thread “main” java.lang.ArithmeticException : under age
Udemy


finally Block Example
fun main (args: Array<String>){
try {
val data = 10 / 5
println(data)
} catch (e: NullPointerException) {
println(e)
} finally {
println("finally block always executes")
}
println("below code...")
}
Udemy



KHATAM TATA BYE BYE!!