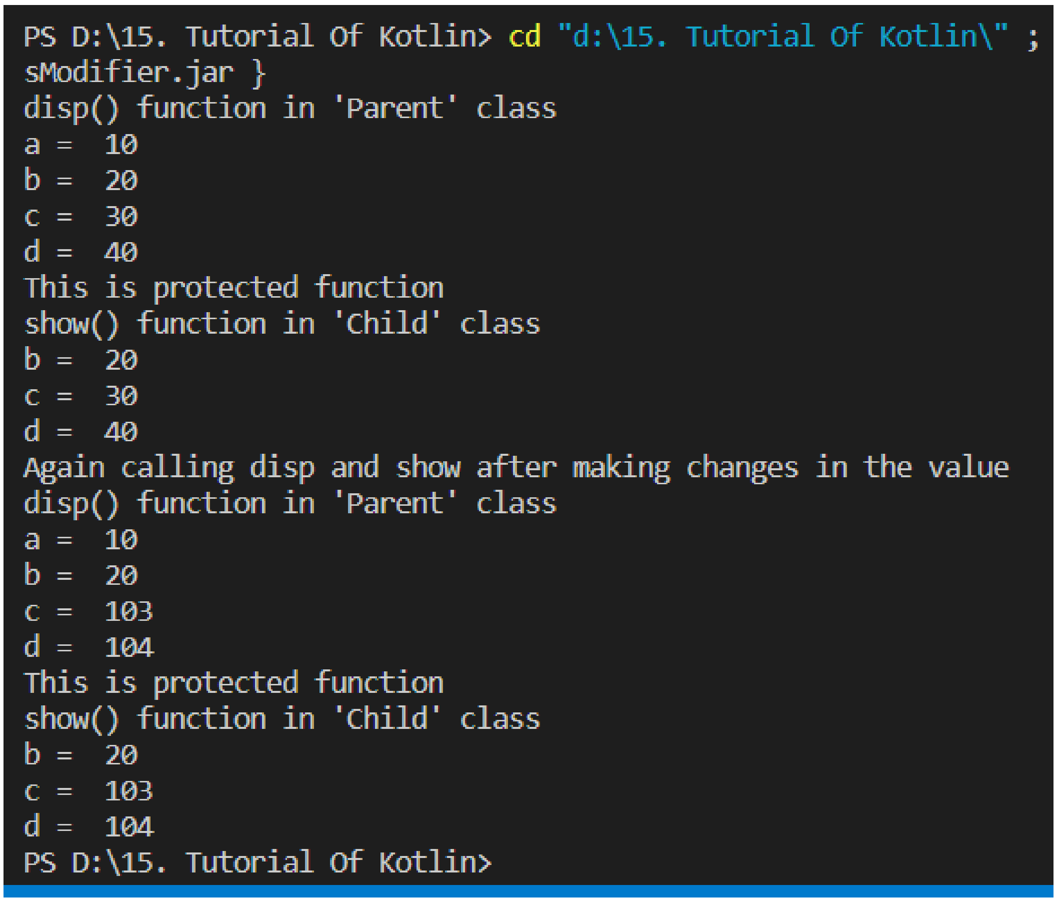
INDEX :

Access Modifier :

```kotlin
open class Parent{
    private var a: Int = 10
    protected var b: Int = 20
    internal var c: Int = 30
    var d: Int = 40   // -> by default it is 'public'

    fun disp(){
        println("disp() function in 'Parent' class")
        println("a =  $a")
        println("b =  $b")
        println("c =  $c")
        println("d =  $d")
    }

    protected fun greet(){
        println("This is protected function")
    }
}

class Child : Parent() {
    fun show(){
        greet()
        println("show() function in 'Child' class")
        // println("a =  $a")    // -> cann't access as 'a' is private data type
        println("b =  $b")
        println("c =  $c")
        println("d =  $d")
    }
}

fun main() {
    val obj = Child()
    obj.disp()
    obj.show()
    // obj.a = 101   // -> cann't access as 'a' is private data type
    // obj.b = 102   // -> cann't access as 'b' is protected data type
    obj.c = 103
    obj.d = 104

    println("Again calling disp and show after making changes in the value")
    obj.disp()
    obj.show()

    // obj.greet()   // cann't access form the main function as it is protected data type.


}
```

Output :-

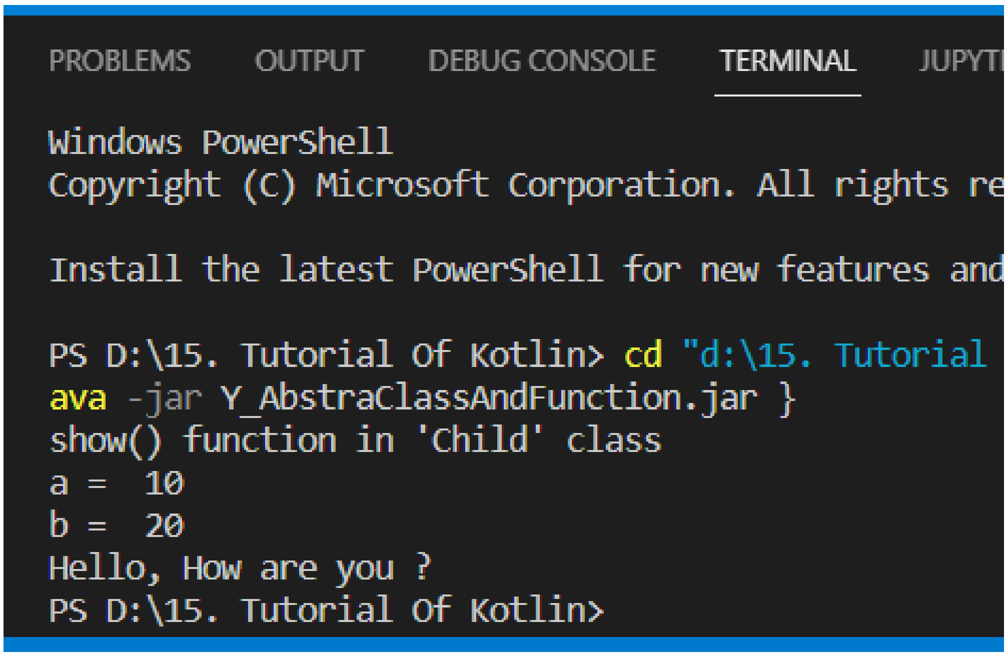


```
PS D:\15. Tutorial Of Kotlin> cd "d:\15. Tutorial Of Kotlin\" ;
sModifier.jar }
disp() function in 'Parent' class
a =  10
b =  20
c =  30
d =  40
This is protected function
show() function in 'Child' class
b =  20
c =  30
d =  40
Again calling disp and show after making changes in the value
disp() function in 'Parent' class
a =  10
b =  20
c =  103
d =  104
This is protected function
show() function in 'Child' class
b =  20
c =  103
d =  104
PS D:\15. Tutorial Of Kotlin>
```

Abstract Class And Function :

```kotlin
abstract class Parent{       // -> by default abstract class is open.

    var a: Int = 10

    var b: Int = 20


    fun disp(){

        println("disp() function in 'Parent' class")

        println("a =  $a")

        println("b =  $b")

    }


    abstract fun greet()

}


class Child : Parent() {

    fun show(){

        println("show() function in 'Child' class")

        println("a =  $a")

        println("b =  $b")

    }


    override fun greet(){

        println("Hello, How are you ?")

    }

}


fun main() {

    val obj = Child()

    obj. show()

    obj.greet()


}
```

Output :-

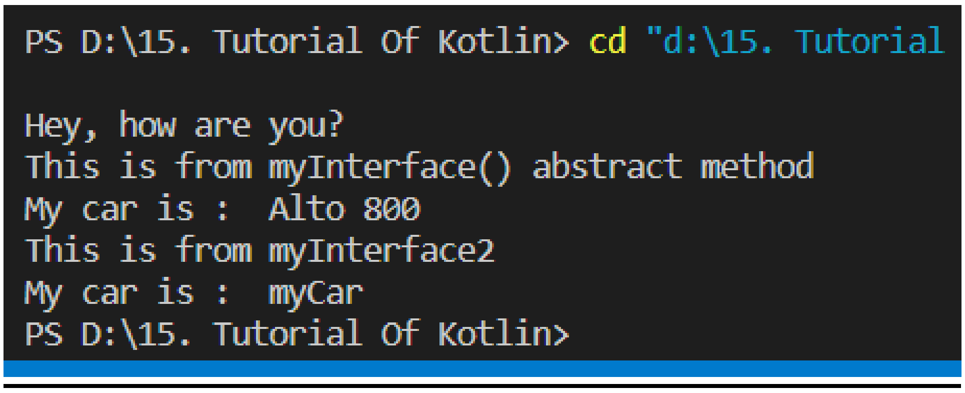


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights re

Install the latest PowerShell for new features and

PS D:\15. Tutorial Of Kotlin> cd "d:\15. Tutorial 
ava -jar Y_AbstraClassAndFunction.jar }
show() function in 'Child' class
a =  10
b =  20
Hello, How are you ?
PS D:\15. Tutorial Of Kotlin>
```

Interface :

```kotlin
interface myInterface{
    var car: String           // Abstract property, so we cann't initialize the value.

    fun disp(){
        println("My car is :  $car")
    }

    fun hello()
    // -> By default it is 'Abstract' method
}

interface myInterface2{
    fun disp(){
        println("This is from myInterface2")
    }
}

class myClass : myInterface{
    var bike: String = "KTM"

    override var car : String = "Alto 800"
    override fun hello(){
        println("Hey, how are you?")
        println("This is from myInterface() abstract method")
    }
}

class Derived : myInterface, myInterface2{
    override var car : String = "myCar"

    override fun disp(){
        super<myInterface2>.disp()  // If we want to call specific function
        super<myInterface>.disp()
    }

    override fun hello(){

        println("This is coming from myInterface")

    }

}


fun main() {

    val s1 = myClass()

    s1.hello()

    s1.disp()


    val obj = Derived()

    obj.disp()


    // We cann't make the object of the 'Interface'


}
```

Output :-

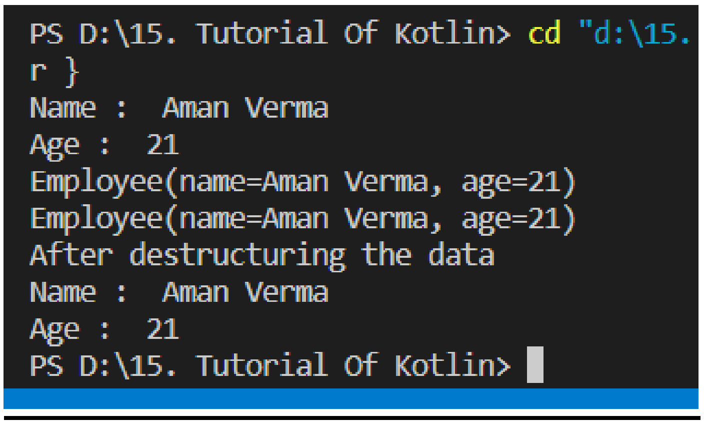


```
PS D:\15. Tutorial Of Kotlin> cd "d:\15. Tutorial

Hey, how are you?
This is from myInterface() abstract method
My car is :  Alto 800
This is from myInterface2
My car is :  myCar
PS D:\15. Tutorial Of Kotlin>
```

Data Class :

```kotlin
// Data Class -> Where we need to create class solely to hold data.


data class Employee(val name: String, val age: Int)


fun main() {

    val emp = Employee("Aman Verma", 21)

    println("Name :  ${emp.name}")

    println("Age :  ${emp.age}")

    println(emp)

    println(emp.toString())


    // Destructuring

    val(name, age) = emp

    println("After destructuring the data")

    println("Name :  $name")

    println("Age :  $age")


}
```

Output :-

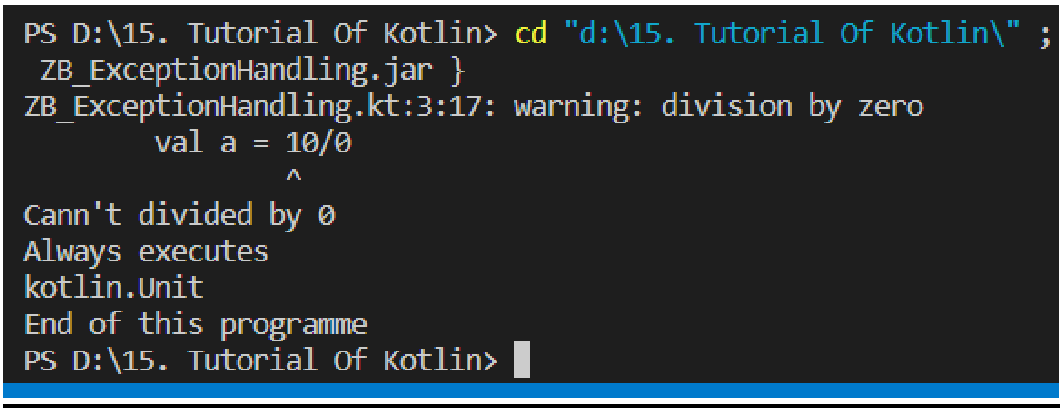


```
PS D:\15. Tutorial Of Kotlin> cd "d:\15.
r }
Name :  Aman Verma
Age :  21
Employee(name=Aman Verma, age=21)
Employee(name=Aman Verma, age=21)
After destructuring the data
Name :  Aman Verma
Age :  21
PS D:\15. Tutorial Of Kotlin> 
```

Exception Handling :

```kotlin
fun main() {

    val result = try{

        val a = 10/0

        a

    } catch(e: Exception){

        e.message

        println("Cann't divided by 0")

    } finally {

        println("Always executes")

    }


    println(result)

    println("End of this programme")


}
```

Output :-



```
PS D:\15. Tutorial Of Kotlin> cd "d:\15. Tutorial Of Kotlin\" ;
 ZB_ExceptionHandling.jar }
ZB_ExceptionHandling.kt:3:17: warning: division by zero
        val a = 10/0
                ^
Cann't divided by 0
Always executes
kotlin.Unit
End of this programme
PS D:\15. Tutorial Of Kotlin>
```

Calling Java :

```kotlin
fun main() {
    val obj = ZC_CallingJavaClass()
    obj.setValue(21)
    println(obj.getValue())
}


fun add(a: Int, b: Int) : Int{
    return (a+b)
}
```

Calling Kotlin Through Java :

```java
public class ZC_CallingJavaClass{
    private int a;

    public void setValue(int value){
        this.a = value;
    }

    public int getValue(){
        return a;
    }
    public static void main(String[] args) {
        System.out.println("Hey, you are in main method of the java class");
        int sum = ZC_CallingJavakt.add(5, 6);  // doesn't work


    }
}
```