# Day-6

# Agenda

Variables
Interface

# Variables

## public

If we declare a member as public then we can access that member from anywhere but Corresponding class should be visible (public) I.e Before checking member visibility we have To check class visibility .

## default

If a member declared as the default, than we can access that member only with in the current package and we can not access from outside of the package , hence default access is also known as package level access.

# Variables

## private

If a member declared as private then we can access that member only within the current class

Corresponding class should be visible (public) I.e Before checking member visibility we have

## protected

If a member declared as protected, then we can access that member within current package any where

**But out side the package only in child class.**

Within the current package we can access protected members either by parent reference or by child reference .

But from outside package we can access protected members only by using child reference.  Parent reference throw Error.

# Variables

## final

In general for instance & static variables it is not required to perform initialization, explicitly JVM will always provide default value.

But if instance variable declared as final than compulsory we should perform initialization whether we are using or not

## static

**If a member declared as static, then explicitly JVM will always provide the default value**

And to access static  member we do not have to create the instance . With class reference  only we can access .

# Variables

## final

In general for instance & static variables it is not required to perform initialization, explicitly JVM will always provide default value.

But if instance variable declared as final than compulsory we should perform initialization whether we are using or not

## static

**If a member declared as static, then explicitly JVM will always provide the default value**

And to access static  member we do not have to create the instance . With class reference  only we can access .

Variables

| super() | this() |
| --- | --- |
| **This is to use to call the super class constructor** | This is to use to call the current class constructor |

| super | this |
| --- | --- |
| **To access the super class instance variables** | To access the current class instance variables |
| **Need to define explicitly to access the super class instance variable** | By default for method this is implemented |

**Note : super & this can not be called inside static method**

| this(), super() | this, super |
| --- | --- |
| **This is to call the super class and current class constructor .** | This is to call the super class and current class instance variables . |
| **We can use only this in constructor as as first statement .** | We can use anywhere except static area |
| **We can use only one at a time not both .** | We can any number of times. |

# BREAK

Back in 10:21 PM IST

# Interface

Any service requirement specification (SRS) is considered as Interface

From the client point of view an Interface defines the set of service what is expecting

From the service provider point of view an Interface defines the set of service what is offering

Hence, An Interface considered as contract between client & service provider

## Advantage of Interface

We can achieve security because we are not highlighting our internal implementation

Enhancement will become very easy because without effecting outside person we can change our internal implementation .

## Declaration of Interface

We can declare the interface by using interface keyword, we can implement an interface by using implements keywords.

## Extends Vs Implements

# Interface

## Extends Vs Implements

A class can extends only one class at a time .

A class can implements any no of interface at a time

A class can extends a class and implement any no of interface at a time

## Interface Method

Every method in interface by default default, public, abstract

## Interface Variables

Every variables in interface by default public, static, final

## Interface  Naming conflict

### Case -1

If two interface contains a method with same signature & same return type in the implementation class ,

we can provide implementation for only one method

### Case -2

If two interfaces contains a method with same name but different argument then, in the implementation class

We have to provide implementation for both methods & hence methods are considered as overloaded methods

### Case-3

If two interface contains a method with same signature but different return types than it is impossible to implement both interface at at time .

## Marker Interface

If an interface would not contain any method & by implementing that interface if our object will get ability

Such types of interface are called marker interface or tag interface or ability interface.

Ex : Serializable , clonable , Random Access etc…

# Interface

| Interface | Abstract Class |
| --- | --- |
| **If we don’t know any thing about implementation just we have the. Requirement than we should go for interface** | If we are talking about implementation  but not completely then we should go for abstract class |
| **Every method present inside the interface is by default public & abstract** | Every method present inside abstract class need not be public & abstract . We can take concrete method also |
| **The following modifiers are not allowed for interact method . Strictfp, protected, static, native, private , final , synchronized .** | There are no restriction for abstract class method modifiers . We can use any modifiers |
| **Every variable present inside interface is public , static , final , by default .** | Abstract variable need not be public , final , static |
| **For the interface variables we can not declare the following modifiers : private , protucted, transient, volatile** | **There are no restriction for abstract class variable modifiers .** |
| **For the interface variables compulsory we should perform initialization at the time of declaration only** | **For the abstract class variables there is no restriction like performing initialization** |
| **Inside interface we can not take constructor** | **Inside abstract class we can take constructor .** |