Interface variables :

Inside the interface we can define variables

Inside the interface variables define requirement level constants(literals) .

Every variable present inside the interface is by default public static final

Eg:: interfaceI ISample {

int x =10;

}

the interface variables are made public static final by default because

In the interface completely SRS so variable cannot be changed so marked as final

These variables should be available to every implementation class so marked as public

That variable should be available directly to implementation class so marked as static (can be called with Interfacename.variable )

If the variable is final it should be initialized during the declaration itself.

Even if you don’t specify public static final also by default they are included in the interface variable .

Why variables in interface are made as static but not methods

static : complete information should be provided

method inside interface

(abstract incompleteness)

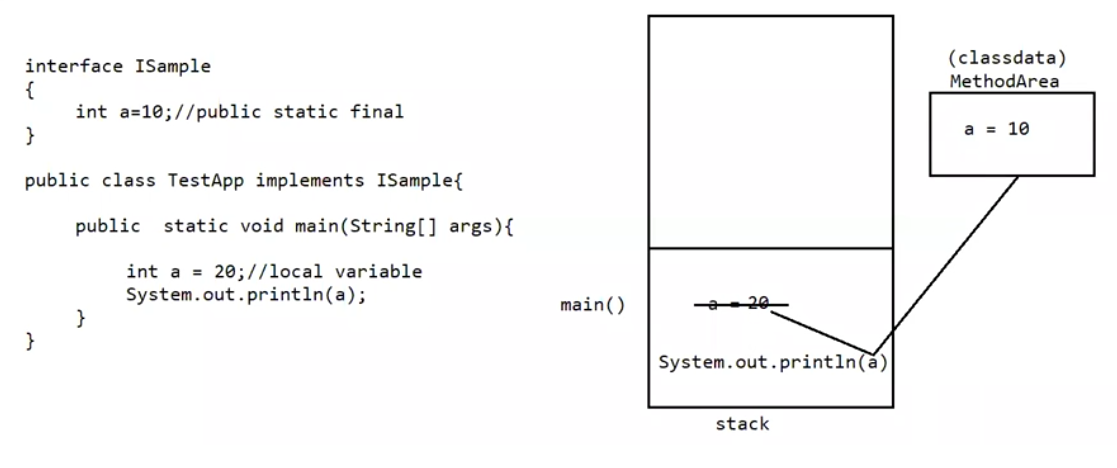
Eg: Variable\_Declaration\_Inside\_Interface // valid variable declarations inside interface.

// illegal declarations

Since the variable defined inside the interface is public static final , we cannot use modifiers private , protected , transient , volatile .

Since the variable is static and final in interface , compulsorily it should be initialized at the time of declaration otherwise it would result in the compile time error .

Eg: Interface\_Variable\_Vs\_Local\_Variable



main() method is brought is to the stack if variable with same name is present in the stack its value is taken . local variables are present in the stack.

Interface variables can be accessed from the implantation class , but cannot modify if we try to modify it would result in the compile time error.

Eg: IVariable

Case 1:

If 2 interfaces contain a method with same method signature ( method’s name and parameters list) and return type in the implementation class , only one method implementation is enough.

Eg: Two\_Interface\_With\_Same\_Method\_Signature

// go through the code

Case 2: If two interfaces contain a method with same name but different arguments in the implementation class we have to provide implementation for both the methods and these methods acts as overload methods

Eg: Two\_Interface\_With\_Same\_Name\_Different\_Parameter

// go through the code

Can a java class implements 2 interfaces simultaneously ?

Yes possible , except both the interfaces contains a method with same method signature , but different return types.

Case 3:

If two interfaces contains a method with same method signature but different return types then it is not possible to implement both the interfaces simultaneously.

Eg: Two\_Interface\_Same\_Method\_Signature\_Different\_Return\_Types

Variable Naming Conflicts :

Two variables can contain a variable with same name and there may be a chance of variable naming conflicts but we can resolve variable naming conflicts by using interface names .

Eg: Variable\_Naming\_Conflicts

Note :

Inside interface the methods are by default “public and abstract”

Inside interface the variables are by default “ public static and final” .

We can also write an interface without any variable (or) abstract methods .

An interface which does not contain any abstract methods those also will be supported by jvm in many ways.

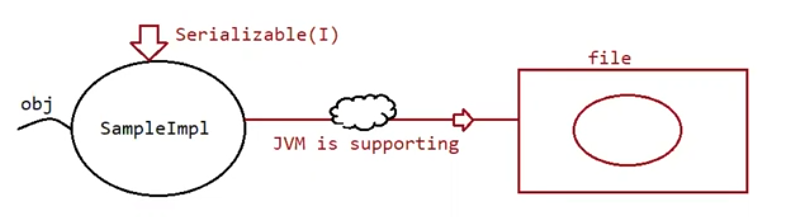
Interfaces that does not have any body is called “Marker interfaces”

Marker interface :

If an interface does not contain an interface and by implementing that interface if our object will get some ability such type of interface is called as Marker interface / Tag interface / Ability interface.

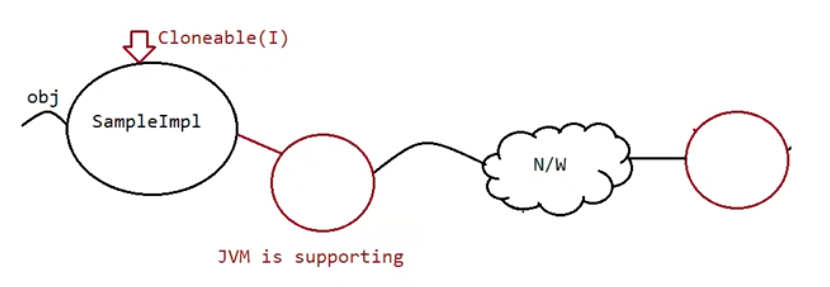
Example : Serializable , Cloneable , SingleThreadModel

Example1 : By implementing Serializable interface we can send the object across the network and we can save the state of the object into the file ( state of the object means data , whatever data is present in the obj it will go and sit in the file .)



Example2 : By implementing SingleThreadModel interface servlet can process only one client request at a time , so that we can get thread safety.

Example3 : By implementing Cloneable interface our object is in a position to provide exactly duplicate cloned object . And that cloned object can sent over the network.



Without having any methods in the marker interface how objects will get ability?

Ans : Jvm is responsible to provide required ability .

Why jvm is providing the required ability to Marker interfaces ?

Ans : To reduce the complexity of programming.

Can we create our own marker interface ?

Ans : yes but , customize the jvm by writing lines of code for your marker interface.

Adapter class ( It is a design pattern allowed to solve the direct implementation of interface

Methods )

It is a simple java class that implements an interface only with empty implementation of every method .

If we implement an interface compulsorily we should give body for all the methods whether it is required or not required . this approach increases the length of the code and reduces readability

Eg: interface X {

void m1();

void m2();

void m3();

void m4();

void m5();

}

class test implements X{

void m3() {

Sysytem.out.println(“ I am from m3”);

}

void() m1() { }

void() m2() { }

void() m4() { }

void() m5() { }

}

In the above approach even though we want only m3() , still we need to give body for all the abstract methods , which increase the length of the code , to reduce this we need “Adapter class” .

Instead of implementing the interface directly we opt for “Adapter class”

Adapter are such classes which implements thee interface and gives the dummy implementation for all the abstract methods , so if we extend adapter class we can easily give body only for those methods which are interested in giving the body .

Eg: Adapter\_Test

Example :

Servlet(I)

| implements

GenericServlet(abstract class )

| extends

HttpServlet(abstract class)

| extends

MyServlet ( class)

Here GenericServelt and HttpServlet acts as adapter classes.