Interface variables:

Inside the interface we can define variables

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

public :: to make it available for implementation class object

static :: to access it without using the implementation class

final :: implementation class can access the value without any modification

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 ( static methods means methods with body)

method inside interface

(abstract incompleteness)

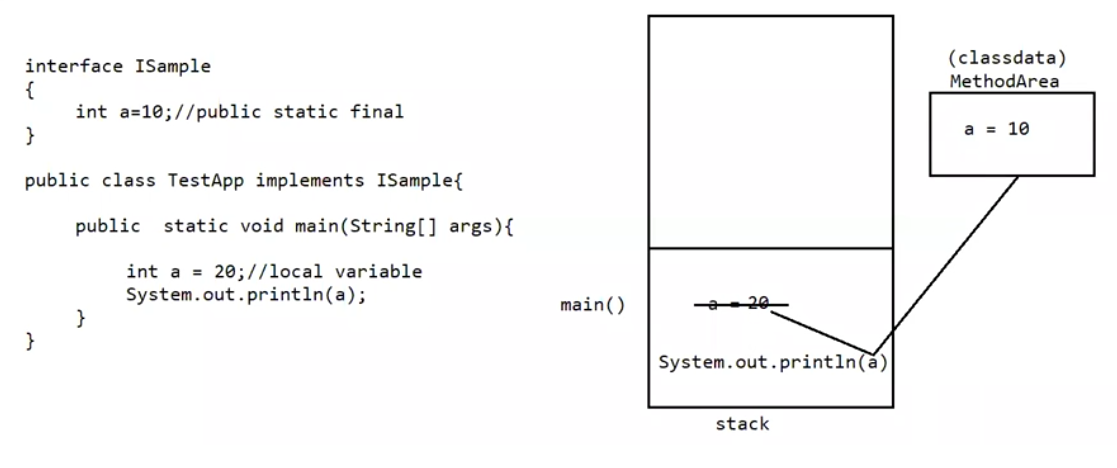
Eg: Variable\_Declaration\_Inside\_Interface // valid variable declarations inside interface.

// 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 overloaded methods

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

// go through the code

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.

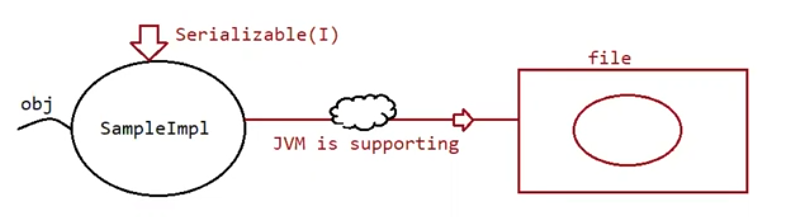
Interfaces that does not have any body is called “Marker interfaces”

Marker interface:

If an interface does not contain an method 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.

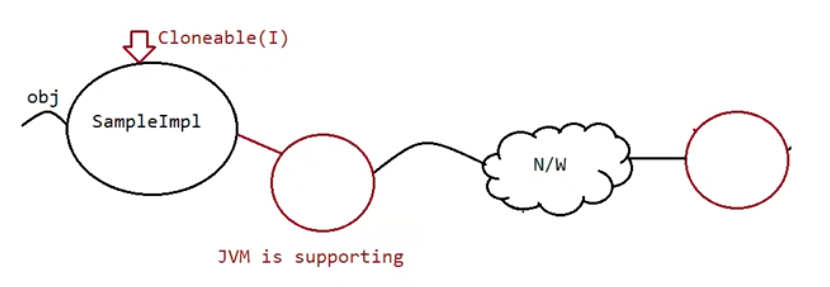
Example: Serializable, Cloneable, SingleThreadModel

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.

Concrete class

Abstract class

Interface

When to use concrete class, abstract class, interface?

It is preferred when we speak only about specification (When I know nothing about implementation) => interface

It is preferred when we speak about partial implementation (When I know partial implementation) => Abstract class

It is preferred when we speak about complete implementation and ready to provide service we can go for concrete class. (Complete implementation and ready to provide the service) => Concrete class

Difference between interface and abstract class?

Interface: if we don’t know anything about implementation, just we have requirement specification then we should go for interface

Abstract class: if we are talking about implementation but not completely then we should go for abstract class.

Interface: Every method declared inside the interface is public and abstract whether you are declaring it or not.

Abstract class: every method present inside abstract class need not be public and abstract.

Interface: we can’t declare interface methods with modifiers like private, protected, strictfp, final, static, synchronized, native.

Abstract class: there are no restrictions for abstract class method modifiers

Interface: every interface variable is public static final whether we are declaring it or not

Abstract class: every abstract class variable need not be public static final

Interface: every interface variable is always public static final, we can’t declare with other following modifiers like private, protected, transient, volatile.

Abstract class: no restriction on access modifiers.

Interface: for every interface variable we should perform initialization during the time of declaration otherwise we get compile time error.

Abstract class: not required to perform initialization for abstract class variables during the time of declaration.

Interface: inside interface we can’t write static and instance block.

// Static blocks are used to initialize static variables, but the variables in the interface are always public static final, since final the variable must be initialized during the declaration itself. so no need of static block in the interface

// instance block, constructor mainly deals with object but interface doesn’t have any implementation (code written in it) to create the object, so there is no point of creating the object for the interface.

Abstract class: inside abstract class we can write static and instance block

We can’t create the object for abstract class.

Interface: inside interface we can’t writer constructor

Abstract class: inside abstract class we can write constructor

Note:

Static block: when .class file loading happens static variables are intialized

Instance block: during the creation of the object, just before the constructor call, it is used for initialization of instance variables

Constructor: during the creation of an object, used for initialization of instance variable.

During the child class object creation, only child class object will be created but not parent class object (still the constructor of parent is called to bring the properties of parent to child)

Eg: Objects\_Created\_During\_Child\_Class\_Object\_Creation

Program flow:

When you create an object of child class, child class constructor is called. And the first line of the child class constructor is super() keyword . then the control comes to parent class constructor, again in the parent class constructor super() keyword is added by jvm. By default the Object class is the parent of all the classes.

So control goes to Object class, after the execution of Object class, parent class constructor and child class constructor are executed.

Why abstract class can contain constructor whereas interface does not contain a constructor?

Abstract class => It is used to perform initialization of the object

It is used to provide the value for the instance variable

It contains instance variables which are required for the child class object to

perform initialization for those instance variables.

can abstract class be instantiated / object be created ? no

can abstract class contain constructor ? yes

Eg: Abstract\_Class\_Constructor

Can a reference be created for abstract class? Yes

// go through the above example

interface => every variable is public static final by default, there is no chance of existing instance variable inside the interface.

We should perform initialization at the time of declaration, so constructor is not required for interface.

Interface object cannot be created.

Can reference be created for Interface ?

Yes // Isample obj = null; here Isample is a interface.

Note: Every method present inside the interface is abstract, but in abstract class also we can have abstract methods then what is the need of interface concept?

We can replace interface with abstract class, but it is not a good programming practice

Eg1: interface Ix{

…

…

}

class TestImpl implements Ix{

…

…

}

Test t = new Test()

1. Performance is high
2. While implementing x we can extend one more class, through which we can bring reusability.

Eg2: abstract X{

…

…

}

class Test extends X{

…

…

}

1. Performance is low
2. While extending X we can’t extend any other classes, so reusability is not brought

If everything is abstract then it is recommended to go for interface.

Interface => high performance

Interface ISample{

}

class Object{

Object();

}

class SampleImpl extends Object implements Isample{

SampleImpl(){

super();

}

}

ISample sample = new SampleImpl();

When we create object of the control goes SampleImpl() constructor and super() method is present in 1st line by default so control goes to parent class constructor (Object class constructor), so here two levels of constructor chaining is happening, which gives better performance(fast execution speed) than abstract class

Here class SampleImpl extends Object class (by default) as well as implementing Isample, so we can bring code reusability. Which is a good practice.

Abstract class => lower performance

class Object{

Object(){

}

}

abstract class Sample extends Object {

Sample(){

super();

}

class SampleApp extends Sample{

SampleApp(){

super()

}

}

Sample obj = new SampleApp();

When we create object of SampleApp() control goes to SampleApp() constructor, and super() method is present in 1st line by default so control goes to parent class constrctor( Sample() constructor) , since super() method is present control goes to Object() class constructor , so here three levels of constructor chaining is happening , here performance is a bit low when compared to interface.

Here class SampleApp extends class Sample, so we can’t extend another class. therefore code reusability is not brought here.