modifier	Classes		methods	variables	blocks	interfaces	enum	Constructors
	Outer	Inner						
public	✓	✓	✓	✓	×	✓	✓	✓
<default>	✓	✓	✓	✓	×	✓	✓	✓
private	×	✓	✓	✓	×	×	×	✓
protected	×	✓	✓	✓	×	×	×	✓
final	✓	✓	✓	✓	×	×	×	×
abstract	✓	✓	✓	×	×	✓	×	×
static	×	✓	✓	✓	✓	×	×	×
synchronized	×	×	✓	×	✓	×	×	×
native	×	×	✓	×	×	×	×	×
strictfp	✓	✓	✓	×	×	✓	✓	×
transient	×	×	×	✓	×	×	×	×
volatile	×	×	×	✓	×	×	×	×

→ The modifiers which are applicable for Inner classes but not for Outer Classes are private, protected, Static

Interfaces

(1) Introduction

(2) Interface declaration & Implementation
   (a) extends Vs implements.

(3) Interface methods

(4) Interface Variables

(5) Interface Naming Conflicts

   (1) method naming Conflicts

   (2) Variable " "

(6) marker Interface

(7) Adapter class

(8) Abstract Class Vs Concrete Class vs interface.

(9) diff.. b/w abstract class & interface

Interface :-

② Any Service requirement Specification (SRS) is Considered as Interface.

→ from the client point of view ~~Con~~ an Interface defines the Set of Services what is expecting.

→ from the Service provider point of view an Interface defines the Set of Services what is offering.

③ Hence an Interface Considered as Contract b/w Client & Service provider.

Ex:-

→ By using Bank ATM GUI Screen, Bank people will hightLate the Set of Services what they are offering At the Same time The Same Screen describes the Set of Services what End-User is Expected.

Hence this GUI Screen acts as Contract b/w ~~the~~ bank people & customers.

→ with in the Interface we Can't write any implementation because it has to highlight Just the Set of Services what we are offering or what you are Expecting. Hence every method present inside interface Should be abstract. Due to this interface is Considered as 100% pure abstract class

What is an Interface :-

→ Any Service requirement Specification (SRS) (or) Any Contract b/w Client & Service provider (or) 100% pure abstract class is nothing but an Interface.

→ The main Advantages of Interfaces are

(i) we Can acheive Security. because we are not highlighting our internal implementation.

(ii) Enhancement will become Very easy, because with out effecting outside person we Can change our internal implementation.

(iii) Two different Systems Can Communicate via Interface (A Java application Can talk with Mainframe System through Interface).

Declaration & Implementation of an Interface :-

→ We Can declare an Interface by using Interface Keyword, we Can implement an Interface by using implements Keyword.

Ex!-

```
interface Interf
{
  Void m1();    // by default public abstract void m1();
  void m2();
}
abstract class ServiceProvider implements Interf
{
→ public void m1()
  {
  }
}
```

→ If a class implements an interface Compulsary we should provide implementation for every method of that interface otherwise we have to declare Class as abstract. Violation leads to Compile-time Error.

→ when ever we are implementing an interface method Compulsary it should be declared as public otherwise we will get CompiletimeError.

Extends Vs implements :-

1. A class Can extend only one class at a time.
2. A class Can implement any no. of interfaces at a time.
3. A class Can extend a class and Can implement any no. of interfaces Simultanciously.
4. An Interface Can extend any no. of interfaces at a time.

ex:-

```
interface A
{
}
interface B
{
}
interface C extends A, B
{
}
```

Q) which of the following is True?

(1) A class Can extend any no. of classes at a time. X

(2) A class Can implement only one Interface at a time. X.

(3) A class Can extend a Class ~~and~~ or Can implement an interface but not both Simultaneously X

(4) An Interface Can extend only one interface at a time X

(5) An Interface Can implement any no. of classes at a time X

(6) None of the above ✓

Q) Consider the expression

$\boxed{X \text{ extends } Y}$ — for which of the following possibilities this Expression is True?

① Both should be classes

② Both should be interfaces

✓ ③ Both can be either classes or interfaces

④ No Restriction.

A:-

① X extends Y, Z

(a) X, Y, Z should be interfaces

② X extends Y implements Z

X, Y → classes

Z → interfaces

③ X implements Y extends Z ✗

→ C.E

Interface methods :-

wheather we are declaring or not, every interface method is by-default, public & abstract

ex:-

```
interface Interf
{
    void m1();
}
```

public:-

→ To make this method availability for every implementation class.

abstract:-

Because interface methods specifies requirements but not implementation.

Hence the following method declarations are equal inside interface.

(1) void m1(); ✓

(2) public void m1(); ✓

(3) abstract void m1(); ✓

(4) Public abstract void m1(); ✓

→ As every interface method is bydefault public & abstract the following modifier are not applicable for interface methods.

(1) private
(2) protected
(3) <default>
(4) final
} ✗

(5) Static
(6) Strictfp
(7) Synchronized
(8) native
} ✗

→ which of the following method declaration are valied inside interface:

(1) public void m1() {} X

(2) public static void m1(); X

(3) public synchronized void m1(); X

(4) private abstract void m1(); X

(5) public abstract void m1(); ✓

Interface variables:-

→ An interface can contain variables the main purpose of these variables is to specify.

Constents at requirement level:-

→ Every interface variable is always public, static, final whether we are declaring (or) not.

```
interface Inter
{
 int x=10;
}
```

public:- To make this variable available for every implementation class.

static:- without existing object also implementation class can access this variable.

final:- implementation class can access this variable but can't modify.

→ Hence inside interface the following declaration are valid & equal.

1) int x=10;
2) public int x=10;
3) public static int x=10;
4) public static final int x=10;
5) public static int x=10;
6) final int x=10;
7) public final int x=10;

8) Static final int x=10;

→ As interface variables are public static & final we can't declare with the following modifiers.

(1) private (3) <default> (5) volatile.

(2) protected (4) transient

→ for the interface variable compulsary are should perform initialization at the time of declaration only otherwise will get compile time error.

⊛
```
interface Interf
{
   int x; X      C.E:- = Expected.
}
```

→ which of the following variable declarations are allowed inside interface.

(1) int x=10; ✓

(2) int x; X

(3) private int x=10; X

(4) public int x=10; ✓

(5) transient int x=10; X

(6) volatile int x=10; X

(7) public static final int x=10; ✓

→ Inside implementation classes we can access interface variables but we can't modify these values.

Ex:-

```
interface Interf
{
    int x=10;
}
```

```
class Test implements Interf
{
 p.s.v.m(String[] args)
 {
     x=888;  ✗
     S.o.pln(x);
   }
}
        C.E:
```

```
class Test implements Interf
{
 p.s.v.m(String[] args)
  {
     int x=88;
    S.o.pln(x); 88
  }
}       ✓
```

Interface nameing Conflicts :-

① method naming Conflicts :-

Case1:-

→ If Two interfaces Contains a method with Same Signature & Same return type in the implementation class we Can provide implementation for only one method.

Ex:-

```
interface Left
{
    Public void m1();
}
```

```
interface Right
{
  Public void m1();
}
```

```
Class Test implements Left, Right
{
    Public void m1()
    {
    }
}
```

Case2 :-

→ If Two interfaces Contains a method with Same name but different args then, in the implementation class we have to provide implementation for both methods & these methods are Considered as overloaded methods.

Ex!-

```
interface Left
{
    Public void m1();
}
```

```
interface Right
{
  public void m1(int i);
}
```

```
Class Test implements Left, Right
{
    Public void m1()
overloaded  {
methods     }
    public void m1(int i)
    {
    }
}
```

Case3 :-

→ If Two interfaces Contains a method with Same Signature but different returntypes. Then it is impossible to implement both interfaces at a time.

Ex :-

```
interface Left               |  interface Right
{                            |  {
   public void m1();         |     public int m1();
}                            |  }
```

→ We Cant write any Java Class which implements both interfaces Simultaneously.

Q) is It possible A Java class Can implement any no. of interfaces Simultaneously.

A) yes, Except if Two interfaces Contains a method with Same Signature but different return-types.

② Variable naming Conflicts :-

```
interface Left
{
    int x=888;
}
interface Right
{
   int x=999;
}
```

```
Class Test implements Left, Right
{
  p.s.v.m(——)
  {
   S.o.pln(x);
  }
}
```

C.E:- reference to x is abeguoues.

→ There may be a chance of 2 interfaces contains avaiable with same name & may rise variable naming conflicts But we can resolve these naming conflicts by using Interface names.

```
S.o.p(Left.x) ; 888
S.o.p(Right.x) ; 999
```

** Marker Interface :-

Ex:- Kenya

→ If an interface wont contain any method & by implementing that interface if our objects will get ability such type of interfaces are called marker interface (or) Tag interface (or) ability interface.

Ex:- Serializable, Clonable, RandomAccess, Single Thread mode.

→ these interfaces are marked from some ability.

Ex:- By implementing Serializable interface we can send object across the N/w and we can save state of object to a file. This extra ability is provided through Seriable interface.

A:- By implementing Cloneable interface our Object will be in a position to provide exactly duplicate Object

Q) Marker interface wont Contain any method then how the Objects will get that Special ability?

A) JVM is responsible to provide required ability in marker interfaces.

Q) why JVM is providing required ability in marker interface?

A) To reduce Complexity of the programming.

Q) Is it possible to Create our own Marker Interface?

A) yes, But Customization of JVM is required.

Ex:- Sleepable, Eatable, Jumpable, Lovable, Funnable.

Adapter Class:-

→ Adapter class is a Simple java class that implements an interface, an interface only with Empty implementation.

```
interface X
{
  m1();
   m2();
    :
    :
  m1000();
}
```

```
abstract class Adapter X implements X
{
  m1() {}
   m2() {}
     :
     :
   m1000() {}
}
```

If we create an object for this Empty result So for this class are declare as abstract. by default abstract

→ If we implement an interface directly ~~coz~~ Compulsary we should provide implementation for every method of that interface, whether we are intrested or not & whether it is required (or) not. It increases length of the code, so that readability will be reduced.

```
Class Test implements X
{
  m1() {}
  m2() {}
  m3()
  {
    ==
  }
  :
  m100() {}
}
```

If we extends adapter class instead of implementation interface directly then we have to provide implementation of only for required method but not all this approach reduce Length of the Code & improves readability.

```
⇒ Class Test extends Adapter X
{
  m4()
  {
    ==
  }
}
```

Concreate class Vs abstract Class Vs interface :-

→ we don't know any thing about implementation Just we have requirements Specification, then we should go for interface

Ex1. Serulet.

→ we are talking about implementation but not Completly (Just partially implementation) then we should go for abstract Class.

Ex:- Generic – Servlet
HTTP_Servlet

→ we are talking about implementation Completly & ready to provide Service, then we should go for Concrete Class.

Ex:- Our own Servlet.

Defference b/w interfaces & abstract class:-

interface	abstract class
1) If we don't know any thing about implementation Just we have requirement Specification. then we should go for interface.	1) If we are talking about implementation but not Completly (partially implementation) then we should go for abstract class.
2) Every method present inside interface is by default public & abstract.	2) every method present inside abstract class need not be public & abstract. we can take concrete methods also.
3) The following modifiers are not allowed for interface methods: strictfp, protected, static, native private, final, synchronized,	3) There are no restrictions for abstract class method modifier i.e, we can use any modifier.

4) every variable present inside interface is public, static final, bydefault wheather we are declare (or) not	4) abstract class variables need not be public, final static.
5) for the interface variables we can't declare the following modifiers private, protected, transient, volatile	5) There are no restriction for abstract class variable modifiers.
6) for the interface variables compulsory we should perform initialization at the time of declaration only	6) for the abstract class variables there is no restriction like performing initialization at the time of declaration
7) Inside interface we can't take instance & static blocks.	7) Inside abstract class we can take static block & instance blocks.
8) Inside Interface we can't take Constructor.	8) Inside abstract class we can take Constructor.

**

Q) Inside abstract class we Can take Constructor but we Can't Create an object of abstract class, what is the need?

A) → abstract Class Constructor will be executed whenever we are Create child class object to perform initialization of parent class instance variable at parent Level only and this Constructor meant for child object Creation only

Q) Inside Interface every method should be abstract where as in abstract class also we Can take only abstract methods Then what is the need of interface?

A) → Interface purpose we Can replace abstract class but it is not a good programming practice we are miss using the role of abstract class.

→ we should bring abstract class into the picture whenever we are talking about implementation.