**Declarations and Access Modifiers – Part-12- Interfaces-02**

* Interface Methods:
  + Every method present inside interface is always public and abstract whether we are declaring or not.
  + Example:

interface Interf{

void m1();

}

public:

To make this method available to every implementation class.

Abstract:

Implementation class is responsible to provide implementation.

Hence inside interface the following method declarations are equal:

void m1();

public void m1();

abstract void m1();

public abstract void m1();

* + As every interface method is always public and abstract we can declare interface method with the following modifiers.

By default, the access specifier is pubic, so we can’t declare:

private, protected

By default, the method is abstract, so we can’t declare the following modifiers:

static

final

synchronized

strictfp

native

* + Which of the following method declarations are allowed inside interface?

public void m1(){} // Invalid

private void m1(); // Invalid

protected void m1(); //Invalid

static void m1(); // Invalid

public abstract native void m1(); // Invalid

abstract public void m1(); // valid

* Interface Variables:
  + An interface can contain variables the main purpose of interface variable is to define requirement level constants.
  + Every interface variable is always public static final whether we are declaring or not.

interface Interf{

int x = 10;

}

public:

To make this variable available to every implementation class.

static:

Without existing object also, implementation class has to access this variable.

final:

If one implementation class changes value, then remaining implementation classes will be affected. To restrict this every interface variable is always final.

Hence within the interface the following variable declarations are equal.

int x = 10

public int x = 10;

static int x = 10;

final int x = 10;

public static int x = 10;

public final int x = 10;

static final int x = 10;

public static final int x = 10;

* + As every interface variable is always public static final, we can’t declare with the following modifiers.

Default its public so we can’t use private, protected.

We can’t create an object for an interface, so serialization is not possible (because serialization is used to save the object). Hence, we can’t use the transient.

By default, its final hence we can’t use “volatile”

* + For interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error.

Note: Inside interface there is no static block.

interface Interf{

int x;

}

CE: = expected

Inside interface which of the following variable declarations are allowed.

int x; // Invalid

private int x = 10; // Invalid

protected int x = 10; // Invalid

volatile int x = 10; //Invalid

transient int x = 10; //Invalid

public static int x = 10; // Valid

* + Inside implementation class, we can access interface variables but we can’t modify values.

Example:

interface Interf{

int x = 10;

}

class Test implements Interf{

public static void main(String[] args){

x = 777l;

System.out.println(x);

}

}

CE: cannot assign a value to final variable x

class Test implement Interf{

public static void main(String[] args){

int x = 777;

System.out.println(x);

}

}

Output: 777

* Interface Naming Conflicts:
  + Method Naming Conflicts:
    - Case\_01:

If two interfaces contain a method with same signature and same return type, then in the implementation class we have to provide implementation for only one method.

interface Left{

public void m1();

}

interface Right{

public void m1();

}

class Test implements Left, Right{

public void m1(){

}

}

* + - Case\_02:

If two interfaces contain a method with same name and but different argument types then in the implementation class we have to provide implementation for both methods and these methods acts as overloaded methods.

Example:

interface Left{

public void m1();

}

interface Right{

public void m1(int i);

}

class Test implements Left, Right{

public void m1(){

}

Public void m1(int i){

}

}

* + - Case\_03:

If two interfaces contain method with same signature but different return types then it is impossible to implement two interfaces simultaneously (if return types are not co-variant).

Example:

interface Left{

public void m1();

}

interface Right{

public int m1(int i);

}

Note: We can’t write any Java class which implements both interfaces simultaneously.

* + - Is a Java class can any number of interfaces simultaneously?

Yes, except a particular case.

If two interfaces contain a method with same signature but different return types. Then it is impossible to implement both interfaces simultaneously.

* Interface Variable Naming Conflict:
  + Two interfaces can contain a variable with same name, and there may be a chance of variable naming conflicts, but we can solve this problem by using interface names.

Example:

interface Left{

public int x = 777;

}

interface Right{

public int x = 888;

}

class Test implements Left, Right{

public static void main(String[] args){

System.out.println(x); // CE: x is ambigious

//We can resolve this using interface name as the variable is by default static.

System.out.println(Left.x);

System.out.println(Right.x);

}

}