Default Methods Inside Interfaces

From java 1.8, we can declare concrete methods in interfaces (they are called default methods)

**public** **interface** MyInterface {

**public** **void** test();

**public** **static** **final** **double** ***PI*** = 3.14;

**public** **default** **int** add(**int** i, **int** j) {

**return** i + j;

}

**public** **default** **void** m1() {

System.***out***.println("interface my Interface");

}

}

Using default keyword, we can make a method concrete in interface .

We can override default methods (see below)

**public** **class** Test **implements** MyInterface {

**public** **static** **void** main(String[] args) {

Test t = **new** Test();

t.m1();

}

@Override

**public** **void** test() {

}

**public** **void** m1() {

System.***out***.println("Test class");

}

}

Output : “Test class”

Differences between Interface with Default Methods and Abstract classes

==

**public** **interface** IntOne {

**public** **default** **void** m1() {

System.***out***.println("interface my Interface");

}

}

**public** **interface** IntTwo {

**public** **default** **void** m1() {

System.***out***.println("interface my Interface");

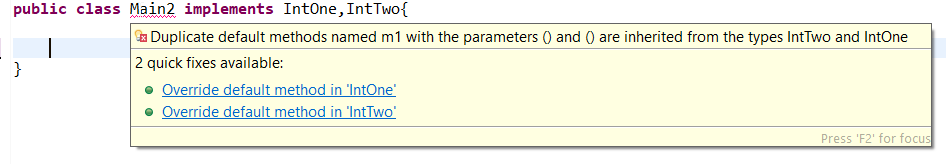
}

}

**public** **class** Main2 **implements** IntOne,IntTwo{

}

There is an ambiguity problem in above class (same method in both interfaces)



--- To solve above problem, we can override common method and call the method (from which interface you want to call);

**public** **class** Main2 **implements** IntOne, IntTwo {

**public** **void** m1() {

IntOne.**super**.m1();

}

}

OR (you can implement by your self)

**public** **class** Main2 **implements** IntOne,IntTwo{

**public** **void** m1() {

System.***out***.println("interface my Interface");

}

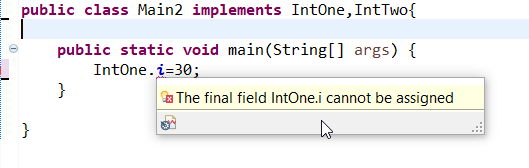
}

|  |  |
| --- | --- |
| Interface with default method | Abstract class |
| Inside interface every variable is always public static and final. We cannot declare instance variables | Inside abstract class we can declare instance variables required for child class |
| Interface never talks about state of object | Abstract class can talk about state of object |
| Inside interface we can’t declare constructors | Inside abstract class we can declare constructors |
| Inside interface we can’t declare instance and static blocks | Inside interface we can declare instance and static blocks |
| Functional interface with default method can refer lambda expression | Abstract class can’t refer lambda expression |
| We can’t override object class methods | We can override abstract class methods |

**public** **interface** IntOne {

**public** **int** ***i*** =10;

}



Static methods and interfaces are not related to objects.

What is the purpose of static methods in interface?

If we want to declare any utility methods that are not related to state of object then we can declare those methods as static in Interface.

**public** **interface** IntOne {

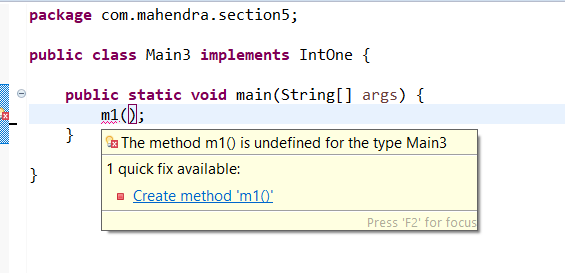
**public** **static** **void** m1() {

System.***out***.println("IntOne.m1()");

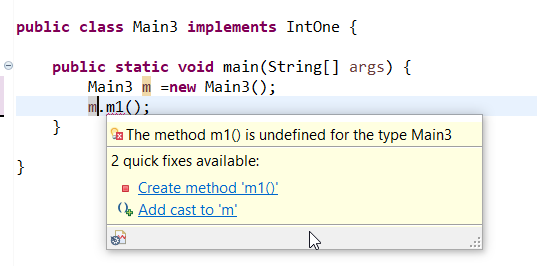
}

}

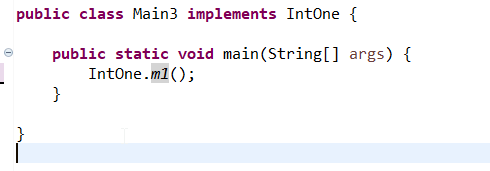
You can access interface static methods in subclass directly (but you can access concrete methods)



You can access interface static methods in subclass objects (but you can access concrete methods through objects)



The only way you can access interface static methods is through its name (see below)



Below is not overriding (it’s a separate method on object). If you can’t access static methods directly, then there is no concept of overriding)

**public** **class** Main3 **implements** IntOne {

**public** **static** **void** main(String[] args) {

*m1*();

}

**public** **static** **void** m1() {

System.***out***.println("IntOne.m1()");

}

}

Below will work (interface has static method but implementation class we removed static. In normal scenario (One class extending another class, we can’t override a method by removing static key word, but using static methods we can do it, this is because its not overriding, it’s a separate method)

**public** **class** Main3 **implements** IntOne {

**public** **static** **void** main(String[] args) {

}

**public** **void** m1() {

System.***out***.println("IntOne.m1()");

}

}

Below will work ( interface has public but implementation class modifier is private)

**public** **class** Main3 **implements** IntOne {

**public** **static** **void** main(String[] args) {

*m1*();

}

**private** **static** **void** m1() {

System.***out***.println("IntOne.m1()");

}

}

|  |  |  |
| --- | --- | --- |
| **public** **interface** IntOne {  **public** **static** **void** m1() {  System.***out***.println("IntOne.m1()");  }  } | **public** **interface** IntOne {  **public** **static** **void** m1() {  System.***out***.println("IntOne.m1()");  }  } | **public** **interface** IntOne {  **public** **static** **void** m1() {  System.***out***.println("IntOne.m1()");  }  } |
| **public** **class** Main3 **implements** IntOne {  **public** **static** **void** m1() {System.***out***.println("IntOne.m1()");  }  } | **public** **class** Main3 **implements** IntOne {  **private** **void** m1() {System.***out***.println("IntOne.m1()");  }  } | **public** **class** Main3 **implements** IntOne {  **private** **static** **void** m1() {  System.***out***.println("IntOne.m1()");  }  } |

**public** **interface** IntOne {

**public** **static** **void** m1() {

System.***out***.println("IntOne.m1()");

}

**public** **static** **void** main(String[] args) {

*m1*();

}

}

You can write public static void main in interface and execute it in interface (see above)

You can run interface from command prompt (from 1.8 version)