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");

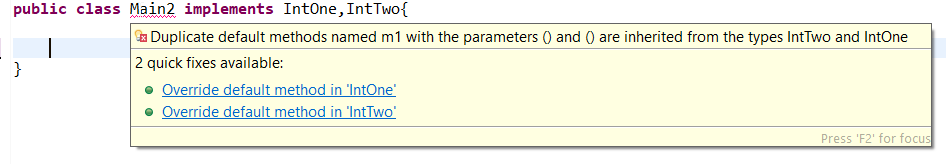
}

}

**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 instant 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;

}

