Java : Extensible Enum with Interface

# Overview

In Java Enum is a powerful data type. Many places user uses Enum To get advantages from Enum type. Like In a Static Factory method it wise decision to pass Enum type rather than pass String. Below , I mention some situations where using Enum is advantageous.

# Enum Advantages

1. In a Static Factory method passing Enum type as argument make the method typesafe. Method users can’t pass arbitrary value to this method.
2. Instead of using bit fields using Enum and EnumSet make the code cleaner and Easy to maintain.
3. Using Enum one can achieve Singleton pattern which is inherently Thread Safe .
4. Using Strategy Enum Pattern one can get rid of If/Switch statements.

Although, Enum has several advantages but it has one major disadvantages.

# Enum DisAdvantage

1. Unlike Class we can not extend Enum.

So the Question is

## “Can we Extend Enum and When it is needed?”

Yes we can emulate the extensibility in Enum but we have to play in a strategic way ,

We know that Enum can’t be extended but we know the fact that an Interface can be extended and Enum can implement the Interface. So combining these two statements we can achieve extensibility. Where Enum can’t be extended but interface can so If we use “***Coding to an Interface***” Strategy we can easily replace BaseType enum with Our Extended Enum.

Now coming to the use case,

According to Joshua Bloch ,

***“There is at least one compelling use case for extensible enumerated***

***types, which is operation codes, also known as opcodes. An opcode is an enumerated***

***type whose elements represent operations on some machine, such as the***

***Operation”***

***“Sometimes it is desirable to let the users of an API provide their own operations,***

***effectively extending the set of operations provided by the API.”***

According to Josh Bloch In case of Operation we can provide some basic operation in a Enum like a simple calculator and basic operations are plus ,minus etc but if API Users want more like XOR , Square root, they can achieve it through extending Operation Interface and create a new Enum.

# Steps For Achieving Extensibility

Let’s take an Use Case , We have a Enum call **Direction** in a API ,it has entries for NORTH,SOUTH,EAST,WEST but most of the cases API Users use these basic direction but when It is about to showing the Shortest path between source and destination API Users need some advanced direction like NORTH-EAST,NORTH-WEST etc.

How judiciously we implement the API so Users can have option to extend the Basic Direction Enum?

Steps.

1. Create an Interface called Direction and declares a method called showDirection().
2. Create a BasicDirection enum by implement Direction interface.
3. Put entries for NORTH,SOUTH,EAST,WEST.
4. Please note that as it implements Direction interface each Enum has to override showDirection method.
5. Create an AdVANCEDIRECTION enum with NORTH-EAST,NORTH-WEST etc entries.
6. Use this Enum anywhere.

Java Code :

package com.example.enumtest;

public interface Direction {

public void showDirection();

}

package com.example.enumtest;

public enum BasicDirection implements Direction{

NORTH{

@Override

public void showDirection() {

System.out.println("I am NORTH");

}

},

SOUTH{

@Override

public void showDirection() {

System.out.println("I am South");

}

},

EAST{

@Override

public void showDirection() {

System.out.println("I am EAST");

}

},

WEST{

@Override

public void showDirection() {

System.out.println("I am WEST");

}

}

}

package com.example.enumtest;

public enum AdvanceDirection implements Direction{

NORTHEAST{

@Override

public void showDirection() {

System.out.println("I am NORTH-EAST");

}

},

NORTHWEST{

@Override

public void showDirection() {

System.out.println("I am NORTH-WEST");

}

},

SOUTHEAST{

@Override

public void showDirection() {

System.out.println("I am SOUTH-EAST");

}

},

SOUTHWEST{

@Override

public void showDirection() {

System.out.println("I am SOUTH-WEST");

}

}

}

package com.example.enumtest;

public class DirectionDriver {

public static void printDirection(Direction op)

{

op.showDirection();

}

public static <T extends Enum<T> & Direction> void printDirections(Class<T> clazz)

{

for(Direction direction : clazz.getEnumConstants())

{

direction.showDirection();

}

}

public static void main(String[] args) {

DirectionDriver.printDirection(BasicDirection.EAST);

DirectionDriver.printDirection(AdvanceDirection.SOUTHWEST);

DirectionDriver.printDirections(BasicDirection.class);

DirectionDriver.printDirections(AdvanceDirection.class);

}

}

Output :

I am EAST

I am SOUTH-WEST

I am NORTH

I am South

I am EAST

I am WEST

I am NORTH-EAST

I am NORTH-WEST

I am SOUTH-EAST

I am SOUTH-WEST

Please pay attention to the class DirectionDriver here I create two methods

1. printDirection which takes Direction as Input and I pass an Enum to it.
2. Another version is printDirections method it prints all Enum entries to achieve that

I make a generic type which says ***T must be an Enum and a subtype of Direction by***

<T extends Enum<T> & Direction> . then pass class instance so I can iterate over EnumConstants.