Table of Contents

1 Introduction

Enum concept is used to represent a group named constants by using a single name. Below is an example of Enum.

```
Enum Month {
        JAN, FEB, MAR, .., DEC;
}
Enum Beer {
        RC, FO, KF, KO
}
```

Ex- Java byte data allowed values are total 256 (-128 to 127).
So Enum main object of enum is define our own data types.
In PL, PPL we having concept like EDT's and ADT's, which are missing in Java in its First release.

- ";" at the end of Enum's constant is optional.
- Enum concept introduced in java 1.5.
- Before Java 1.5, "public static final" variables (constants) are used in place of Enum's.
- The main objective of defining Enum is to define user defined data type (Enumerated Data types) like primitive data types.
- Java Enum concept is so power then its earlier old language as we can declare variables, constructor's, methods, etc inside Enum class. Which is not possible in most of the old languages.

1.1 Internal implementation

- Every enum is inernally implemented using a class concept.
- Every Enum constant is internally "public static final".
- Every enum constant within Enum (class) declaration represents an Object of the Type Enum.
- Enum simplifies developer job by simplifying the lengthy code.

Examples –

```
Enum Beer {
        RC, FO, KF, KO
}
```

Above Enum declaration will be internally converted as below –

```
Class Beer {
        public static final Beer RC = new Beer();   =➔ is an Beer type Object.
        public static final Beer FO = new Beer();
        public static final Beer KF = new Beer();
        public static final Beer KO = new Beer();
}
```

2 Enum Declarations

- Every enum constant is public static final. Hence we can access any enum constant by using its enum Name directly.
- Inside enum toString() method implemented to return name of the constant directly.

```
Enum Beer {
        RC, FO, KF, KO
}
Class Test {
        public static main (String[]  args){
                Beer b = Beer.KF;  - calling enum constants by its enum name.
                System.out.println(b) → Prints "KF".

        }
}
```

2.1 Allowed Enum declarations

We can declare enum's as below :-

- Outside a class
- inside a class
- In its own file.

2.2 Not allowed Enum declarations

We cannot declare an enum as below and If we try to declare inside a method then we will get a compile time error (Enum type must not be local).

- Inside a method like local classes.
- Inside a block.

2.3 Applicable modifiers

- Applicable modifiers for For top level classes
  public, default , strictfp, final, abstract
- Applicable modifiers for For inner classes
  public, default , strictfp, final, abstract + private, protected, static

Enum is implicitly final so "final" is not allowed, and abstract is not applicable for final types hence "abstract" is not allowed.

- If enum declared outside of a class or within its own file then applicable modifiers are –
  public, default and strictfp
- If enum declared inside a class then applicable modifiers are –
  Public, default, strictfp + private, protected, static.

3 Enum with switch statement

- Until Java 1.4 the only allowed argument types for "switch" statement are – byte, short, int and char.
- From Java 1.5 version onwards corresponding **wrapper classes and Enum** also allowed because of Autoboxing and Unboxing concept.
- From java 1.7 version onwards String classes also allowed.

So Enum type is an allowed value for switch type from Java 1.5.

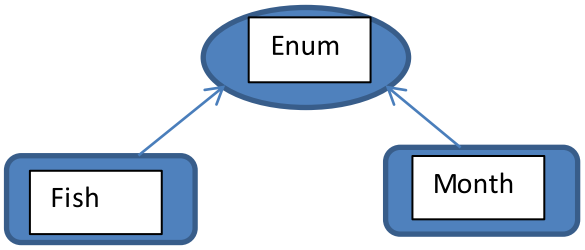
Example –

```
Enum Beer {
        RC, FO, KF, KO
}
Class Test {
        public static main (String[]  args){
                Beer b = Beer.KF;
                Switch (b) {
                        Case RC:
                                System.out.println("child")
                                break;
                        Case FO:
                                System.out.println("buy one get one")
                                break;
                        Case KF:
                                System.out.println("No kick")
                                break;
                        Case KO:
                                System.out.println("Light")
                                break;
                        Case default:
                                System.out.println("Not valid")
                                break;
                }
        }

}
```

If the enum constant is not a valid constant then we will get a compiler type error like - Unqualified enumeration constant name required

4 Enum with Inheritance



- Every enum in java is direct child class of java.lang.Enum class. Hence we can't extend any other class.
- Every enum is java is implicitly final, hence we can create child enums.
- Because of the above reasons we can conclude that inheritance concept is not supported/applicable for enum. Hence we can't use extends keyword for enums.

Below error message you will see when you try to inherit enums.

- Cannot inherit from final "XX"
- Enum types are not extensible

Below **Class inheritance examples** **are not valid** with respect to Enums : -

```
Enum X {
}
Enum Y extends X {
}
```

```
Class X {
}
Enum Y extends X {
}
```

```
Enum X  {
}
Class Y extends X {
}
```

```
Enum Y extends java.lang.Enum {
}
```

But you **can implement** as many number of **interfaces** as you like –

```
Interface X  {
}
Enum Y implements X {
}
```

5 Java.lang.Enum

- Every enum is java is **direct child class of java.lang.Enum** class.
- This is class act as a base class for all the enums.
- It is a direct child class of Object
- It is an **Abstract class**, So you cannot create an object directly for this class.
- It implements **Serializable and Comparable** interfaces.

6 values() and ordinal() methods

These two methods are useful methods in provided by java language for enums.

6.1 values () method

- This method useful to list of all values present in Enum type.
- It is a **static method**
- This method is not available in java.lang.Enum, Java compiler implicitly provide this method with enum keyword.

Example -

```
Beer[]  beers = Beer.values();
```

6.2 ordinal () method

- Inside enums, order of constants are important which can be derived by using ordinal() method.
- This method is available in java.lang.Enum class.
- Ordinal value is array based index.

Example –

```
Enum Beer {
        KF, KO, RC, FO
}
Class Test {
        Public static void main(String[] args){
                Beer[] beers = Beer.values();
                For (Beer b : beers){
                        System.out.print( b + "---" + b.ordinal());
                 }
        }
}
```

7 Enum constants with other members

- In old languages, mostly enum allows declare only as constants.
- But in Java, In addition to constants we can also declare methods, constructors, and Normal variables etc.
- Hence Java enum is more powerful than old language enums.
- Even we can declare main() methos inside enum, and can directly invoke enum from command prompt.
- In addition to constants if we are declaring any extra members like methods, variables, constructors, etc then **"List of constants" should be declared in first line** and **should end with semicolon (;)**

  Example -

```
Enum Beer{
        Public static void main (String[] argd) {
                System.out.println ("enum main");
        }
}
```

Below **members combination is not allowed** with enums : -

```
Enum Beer {
        KF, RC [No semicolon]
        Public void m1(){
        }
}
```

```
Enum Beer {
        Public void m1(){
        }
        KF, RC [Not 1st line  ]
}
```

```
Enum Beer {
        Public void m1(){
        }
}
[Only method, without constants]
```

Below **members combination is allowed** with enums :

```
Enum Beer {
        KF, RC;
        Public void m1(){
        }
}
```

```
Enum Beer {
        ;
        Public void m1(){
        }
}
```

```
Enum Beer {

}
```

8 Enum constants with Constructor

Below enum contains constructor and it is executed separately for enum constant at the time of enum class loading.

```
Enum Beer {
        KF, KO, RC, FO;
        Beer() {
                System.out.println("Constructor")
        }
}
Class Test {
        Public static void main(String[] args){
                Beer b = Beer.KF // (1)
                System.out.println("Hello")
        }
}
```

Output of above program is – 3 times "constructor" followed by "Hello".
If we comment Line 1, then the output is Hello.

- Inside enum we can declared only concrete methods, **abstract methods are not allowed**.
- Inside enum only **private or default** constructors are allowed but not Public.
- We can't create enum object explicitly and hence we can't invoke enum constructor directly.

```
Beer KM = new Beer(); // Compile time error
we can't create an object of enum explicitly
```

Below is the sample implementation of enum with constructor and a method.

```
enum Beer {
        KF(50), KO(60), RC(70), FO(80), KM; // Calling enum constructor.
        int price;
        Beer(int price) {
                this.price = price;
        }
        Beer() {
                this.price = 25;
        }
        Public int getPrice(){
                return price;
        }
}
Class Test {
        Public static void main(String[] args){
                Beer[] beers = Beer.values();
                for (Beer b : beers) {
                        System.out.println(b + "--- "+ b.getPrice()); // call enum methods
                }
        }
}
```

Note:

Beer constants are internally treated as Beer Object like below -

```
BEER.RC ➔ public static final Beer RC = new Beer();
BEER.KF ➔ public static final Beer KF = new Beer();

BEER.RC(50) ➔ public static final Beer RC = new Beer(50);
BEER.KF(60) ➔ public static final Beer KF = new Beer(60);
```

Another example:

```
enum GameStatus {
        START(“Start”), PLAY(“Paly”), PAUSE(Pause“”), RESUME(“Resume”), STOP(“Stop”);

        Srting status;
        Beer(iString status) {
                this. status = status;
        }
        Public String getString(){
                return status;
        }
}
Class Test {
        Public static void main(String[] args){
                String status = GameStatus.START.getStatus(); // Access enum Method directly.
        }
}
```

9 Enum with Methods

Case 1 :

Every enum constant represent an object of type Enum.

Hence we can call all methods available in Object class on enum constants like any other normal java object.

```
Beer.KF.equals(Beer.RC)
Beer.KF == Beer.RC
Beer.KF.hashcode() > Beer.RC.hashcode()
Beer.KF.ordinal() > Beer.RC.ordinal ()
```

Below syntax is not allowed

```
Beer.KF > Beer.RC
```

Case 2: Methods can be declared anonymously for an enum constant like below

```
enum Game {
        CRICKET , FOOTBALL{
                Public void Play (){
                        System.out.println(“So much fitness is required”);
                }
        , TENIS;

        Public void play(){
                System.void.println(“players are crazy”);
        }
}
Class Test {
        Public static void main(String[] args){
                For (Game gm : Game.values()){.
                        gm.play();
                }
        }
}
```

Output is :

```
players are crazy
So much fitness is required
players are crazy
```

10 Nested Enum Declarations

Nested enum types are treated analogously to nested interface in this regard: they are static.