

18. Enum

DURGA SOFTWARE SOLUTIONS

SCIP MATERIAL

Enum (Enumeration - 1.5 version) :-

→ If we want represent a group of constants then we should go for enum.

Ex: enum Month
{
 JAN, FEB, MAR, -----, DEC;
}

enum Beer
{
 KF, KO, RC, FO;
}
} optional (semicolon)

→ The main purpose of enum is to define our own data types like Enumerated data types.

→ Enum concept introduced in 1.5 version.

→ When compared with old languages enum Java enum is more powerful.

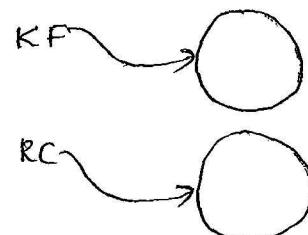
Internal implementation of Java enum :-

→ Every enum internally implemented by using class concept.

→ Every enum constant is always public static final.

→ Every enum constant represents an object of the type enum.

enum Beer
{
 KF, RC;
}
class Beer
{
 public static final Beer KF=new Beer();
 public static final Beer RC=new Beer();
}



Enum Declaration and Usage :-

→ Every enum constant is always static & hence we can access by using enum name.

```

Ex: enum Beer
{
    KF, KO, RC, FO;
}
class Test
{
    public static void main()
    {
        Beer b = Beer.RC;
        System.out.println(b); OP: RC
    }
}

```

Note:- Inside enum toString() method internally implemented to return name of the constant.

- We can declare enum either outside a class or within the class, but not inside a method.
- If we are trying to declare enum inside a method then we will get CE.

<u>Ex:</u> enum X	class X	class X
{	{	{
}	}	}
class Y	enum Y	public void m1()
{	{	{
}	}	}
✓	✓	✗

CE: enum types must not be local

- If we declare enum outside the class the applicable modifiers are public, <default> and strictfp.
- If we declare enum within the class the applicable modifiers are

public
<default> + private
strictfp protected
static

enum Vs switch statement:

- For the switch statement allowed argument types until 1.4 version are byte, short, char and int.
- But from 1.5 version onwards the corresponding wrapper classes & enum types allowed.

1.4 v	1.5 v	1.7 v
byte	Byte	
Short	Short	
Char	Character	
int	Integer + enum	String

- Hence from 1.5 version onwards we can pass enum type also as argument to switch statement.

Ex: enum Beer
{
 KF, KO, RC, FO;
}

```
class Test
{
    public void m()
    {
        Beer b = Beer.RC;
        switch(b)
        {
            case KF : S.o.p("It is children's brand");
                        break;
            case KO : S.o.p("It is too light");
                        break;
            case RC : S.o.p("It is not that much kick");
                        break;
            case FO : S.o.p("Buy one get one free");
                        break;
            default : S.o.p("Other brands are not
                           recommended");
                        break;
        }
    }
}
```

Q1: It is not that much kick

→ If we pass enum type as argument to switch statement then every case label should be valid enum constant, o.w. we will get CE.

Ex: switch (cb)

```
{
    Case KF;
    Case KO;
    Case RC;
    Case FO;
    X Case KALYANI;
```

CE: unqualified enumeration constant name required.

enum vs Inheritance:

1. Every enum in Java is direct child class of java.lang.Enum class. Hence our enum can't extend any other enum.

2. Every enum is always final implicitly & hence we can't create child enum.

→ Becoz of above reasons we can conclude inheritance concept not applicable for enum's explicitly.

→ Hence we can't use extends keyword for enum's.

Ex: enum X
{ }

enum Y extends X

{

}



enum X

{

}

class Y extends X

{

}

enum X extends j.l.Enum

{

}



class X

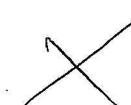
{

}

enum Y extends X

{

}



CE1: cannot inherit from final X

CE2: enum types are not extensible

→ An enum can implement any no. of interfaces simultaneously.

Ex: Interface X

```
{
}
enum Y implements X
{
}
```

java.lang.Enum:—

- Every enum in Java is the direct child class of j.l.Enum class.
- Hence this class acts as base class for all Java enums.
- It is an abstract class and direct child class of Object.
- It implements Comparable & Serializable interfaces.

values() method:—

→ We can use values() method to list out all values present inside enum.

Ex: Beer[] b = Beer.values();

ordinal() method:—

- Within the enum order of constants is important if we can represent order by using ordinal value.
- We can find ordinal value of enum constant by using ordinal() method.

public final int ordinal();

Ex:

```
enum Beer
{
    KF, KO, RC, FO;
}
class Test
{
    public void main()
    {
        Beer[] b = Beer.values();
    }
}
```

```
for(Beer b1 : b)
{
    System.out.println(b1+"..."+b1.ordinal());
}
```

Output:

KF	...	0
KO	...	1
RC	...	2
FO	...	3

Note:- ordinal value is zero based.

Speciality of Java enum:

- In old languages enum, we can take only constants but in Java enum in addition to constants we can take methods, constructors, normal variables etc.
- Hence Java enum is more powerful than old languages enums.
- Inside enum we can take main() method & hence we can invoke Enum class directly from command prompt.

Ex: enum Fish

```
{
    STAR, GUPPY, GOLD;
    public void m1()
    {
        System.out.println("Enum main method");
    }
}
```

javac Fish.java

java Fish

Op: Enum main method.

→ In addition to constants, if we are taking any extra member like a method then list of constants should be in the first line & should ends with semicolon(;).

Ex: enum Fish

```
{
    STAR, GUPPY;
    public void m1()
    {
    }
}
```

Mandatory

enum Fish

```
{
    STAR, GUPPY;
    public void m1()
    {
    }
}
```

X

enum Fish

```
{
    public void m1()
    {
    }
    STAR, GUPPY;
}
```

X

→ If we are taking any extra member like a method then first line should contain list of constants atleast semicolon.

Ex: enum Fish

```
{
    public void m1()
    {
    }
}
```

X

enum Fish

```
{
    ;
    public void m1()
    {
    }
}
```

✓

→ Anyway an empty enum is valid.

Ex: enum Fish
 {
 } ✓

enum Vs constructors :—
enum

→ enum can contain constructors and enum constructors will be executed at the time of enum class loading automatically for every enum constant.

Ex: enum Beer

```
{  

  KF, KO, RC, FO;  

  Beer()  

  {  

    System.out.println("constructor");  

  }  

}  

class Test  

{  

  public static void main(String[] args)  

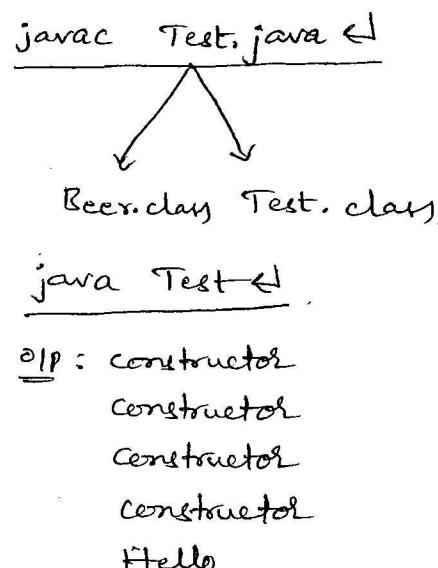
  {  

    Beer b = new Beer.KF; → ①  

    System.out.println("Hello");  

  }  

}
```



→ If we comment line ① then the output is Hello.

→ We can't create enum object explicitly by mistake if we are trying to create then we will get compile time error.

→ Hence we can't invoke enum constructor directly.

Ex: Beer b = new Beer(); X

cc: enum types may not be instantiated.

Note:-

KF ⇒ P s final Beer KF = new Beer();

KF(100) ⇒ P s final Beer KF = new Beer(100);

Ex: enum Beer

```

{
    KF(100), KO(90), RC(95), FO;
    int price;
    Beer(int price)
    {
        this.price=price;
    }
    Beer()
    {
        this.price=65;
    }
    public int getPrice()
    {
        return price;
    }
}
  
```

```

class Test
{
    P S R m()
    {
        Beer[] b=Beer.values();
        for(Beer b1:b)
        {
            S.o.p(b1+"..."+b1.getPrice());
        }
    }
}

  
```

OP:

KF	...	100
KO	...	90
RC	...	95
FO	...	65

Note:— Inside enum we can take methods, but should be concrete methods i.e.; we can't take abstract methods inside enum.

Ex(i): Every enum constant represents an object of the type enum. Hence whatever the methods we can call on normal Java objects we can call same methods on enum constants also.

- ✓ 1. Beer.KF.equals(Beer.RC)
- ✓ 2. Beer.KF == Beer.RC
- ✓ 3. Beer.KF.hashCode() > Beer.RC.hashCode()
- ✗ 4. Beer.KF > Beer.RC
- ✓ 5. Beer.KF.ordinal() > Beer.RC.ordinal()

Case(ii):

```

enum Color
{
    BLUE, RED, GREEN;
    public void info()
    {
        S.o.p("Universal Color");
    }
}
  
```

```

enum Color
{
    BLUE, RED;
    public void info()
    {
        S.o.p("Dangerous Color");
    }
}
  
```

```

class Test
{
    public void m()
    {
        Color[] c = Color.values();
        for(Color ct : c)
        {
            ct.info();
        }
    }
}

```

OIP:
Universal Color
Universal Color
Universal Color

Case(iii): enum Vs Enum Vs Enumeration:-

1) enum:-

→ enum is a keyword in java which can be used to define a group of named constants.

2) Enum:-

→ It is a class present java.lang package.

→ Every enum in Java should be direct child class of this Enum class.

→ Hence Enum class acts as base class for all Java enums.

3) Enumeration:-

→ It is an interface present in java.util package.

→ We can use Enumeration object to get objects one by one from the collection.

Case(iv):

Note:- If we want to use class name directly from outside package we have to use normal import.

```

public void info()
{
    System.out.println("Universal Color");
}

class Test
{
    public void m()
    {
        Color[] c = Color.values();
        for(Color ct : c)
        {
            ct.info();
        }
    }
}

```

OIP:
Universal Color
Dangerous Color
Universal Color

If we want to access static members directly without class name then we have to use static import.

Ex: package pack1;
public enum Fish
{
 STAR, GUPPY;
}

package pack2;

class Test1

{
 P S V m()
}

Fish f = Fish.GUPPY;

S.O.P(f)

The required import is

import pack1.Fish;

(or)

import pack1.*;

package pack3;

class Test2

{
 P S V m()
}

S.O.P(GUPPY);

}

The required import is

import static pack1.Fish.GUPPY;

(or)

import static pack1.Fish.*;

package pack4;

class Test3

{
 P S V m()
}

Fish f = Fish.GUPPY;

S.O.P(STAR);

}

import static pack1.Fish.STAR;

(or)

import static pack1.Fish.*;

import pack1.Fish;

(or)

import pack1.*;