Enum (Enumeration)

- If we want to represent a Group of Named Constants, then we should go for Enum Concept.

```
enum Month{
      JAN,FEB,MARCH,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC;
}
```

- Here Semicolon (;) is Optional.
- The main objective of enum is to define our own data types (Enumerated Data Types).
- Enum concept was introduced in Java version 1.5.
- When compared with Old Languages enum, Java enum is more powerful.

Internal Implementation of Enum

Whenever we declares an enum as:

```
enum Person{
    ENGINEER,DOCTOR,LAWYER;
}
```

Internally, a class called Person is created and each of the enum constants is declared as:

```
public static final <EnumName> <constantName> = new <EnumName>
```

So, internally above enum is represented as:

```
class Person{
public static final Person ENGINEER = new Person();
public static final Person DOCTOR = new Person();
public static final Person LAWYER = new Person();
}
```

So,

- Every enum is internally represented as Class.
- Every enum constant is always declared as public static final Object of enum type.

Enum Declaration and Usage

- Enum is declared by using enum keyword.

```
enum <enum_Name>{
    <constant_Name_1>,<contanst_Name_2>,….<constant_Name_n>;
}
```

Example:

```
enum Month{
      JAN,FEB,MARCH,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC;
}
```

To Access Enum Constants:

Since every enum constants are declared as public static final. So we can access enum constant with the help of Enum Name.

Main.java

```
enum Month{
    JAN,FEB,MARCH,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC;
}
class Test{
    public static void main(String args[]){
        Month m = Month.MAY;
        System.out.println(m);
    }
}
```

Output

```
java -cp /tmp/ir4Hw2hHni Test
MAY
```

Internally, whenever we are printing the reference variable of enum type, a toString() method is called which is internally implemented in every enum for printing the name of enum constant.

We can declare enum type either outside the class as declared in above example or inside the class, but cannot be written inside a method. If we declare enum type inside a method, then a compile time error is thrown.

Enum inside the class

Main.java

```
class Test{
    enum Month{
    JAN,FEB,MARCH,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC;
    }
    public static void main(String args[]){
        Month m = Month.MAY;
        System.out.println(m);
    }
}
```

Output

```
java -cp /tmp/ir4Hw2hHni Test
MAY
```

Enum inside the method

Main.java

```
class Test{
    public static void main(String args[]){
        enum Month{
    JAN,FEB,MARCH,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC;
    }
        Month m = Month.MAY;
        System.out.println(m);
    }
}
```

Output

```
ERROR!
javac /tmp/ir4Hw2hHni/Test.java
/tmp/ir4Hw2hHni/Test.java:3: error: enum types must not be local
        enum Month{
        ^
1 error
```

If we declare enum outside a class, then it can be public, default, strictfp. If we declare an enum inside the class then apart from public, default, strictfp it can also be private, protected, static as well. It cannot be abstract, and final in either of the two cases.

enum vs switch

Until 1.4 version, the allowed argument types for switch statement are byte, short, int, char. In 1.5 version, apart from byte, short, int, char the corresponding wrapper classes like Byte, Short, Integer, Character are also allowed along with enum type. In 1.7 version, String class was allowed as an argument type for switch statement.

1.4	1.5	1.7
byte	Byte	String
short	Short	
int	Integer	
char	Character	
	enum	

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

Main.java

Run

Output

```
enum Fruit{
    APPLE,MANGO,BANANA;
}
class Test{
    public static void main(String args[]){
        Fruit f = Fruit.APPLE;
        switch(f)
        {
            case APPLE:
                System.out.println("APPLE IS APPY");
                break;
            case MANGO:
                System.out.println("MANGO IS MAZZA");
                break;
            case BANANA:
                System.out.println("BANANA IS GYM SYRUP");
                break;
            default:
                System.out.println("No Healthy Fruits !!");
        }
    }
}
```

```
java -cp /tmp/ir4Hw2hHni Test
APPLE IS APPY
```

If we pass enum type as an argument to switch statement, then its mandatory that every case Label should be valid enum constant. i.e., they should belong to enum.

enum vs Inheritance

- Every enum in java is a direct child class of java.lang.Enum class. Hence, our enum can't extend any other enum as multiple Inheritance is not possible in Java.
- Every enum is declared as final implicitly. Hence, our enum cannot be extended further. i.e., for our enum we cannot create child enum.
- Because of above reasons we can conclude that inheritance concept is not applicable for enum explicitly and we can't use extends keyword for enum.

Main.java

```
enum X
{

}
enum Y extends X
{

}
class EnumDemo
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Output

```
ERROR!
javac /tmp/iPxbgkTMWe/EnumDemo.java
/tmp/iPxbgkTMWe/EnumDemo.java:5: error: '{' expected
enum Y extends X
^
/tmp/iPxbgkTMWe/EnumDemo.java:5: error: ',', '}', or ';' expected
enum Y extends X
      ^
/tmp/iPxbgkTMWe/EnumDemo.java:5: error: '}' expected
enum Y extends X
           ^
3 errors
```

Main.java

```
enum X extends java.lang.Enum
{

}
class EnumDemo
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Output

```
ERROR!
javac /tmp/iPxbgkTMWe/EnumDemo.java
/tmp/iPxbgkTMWe/EnumDemo.java:1: error: '{' expected
enum X extends java.lang.Enum
     ^
/tmp/iPxbgkTMWe/EnumDemo.java:1: error: ',', '}', or ';' expected
enum X extends java.lang.Enum
^
/tmp/iPxbgkTMWe/EnumDemo.java:1: error: '}' expected
enum X extends java.lang.Enum
           ^
3 errors
```

Main.java

```
enum X
{

}
class EnumDemo extends X
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Output

```
ERROR!
javac /tmp/iPxbgkTMWe/EnumDemo.java
/tmp/iPxbgkTMWe/EnumDemo.java:5: error: cannot inherit from final X
class EnumDemo extends X
                       ^
/tmp/iPxbgkTMWe/EnumDemo.java:5: error: enum types are not extensible
class EnumDemo extends X
^
2 errors
```

Main.java

```
class EnumDemo
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
enum X extends EnumDemo
{

}
```

Output

```
ERROR!
javac /tmp/iPxbgkTMWe/EnumDemo.java
/tmp/iPxbgkTMWe/EnumDemo.java:8: error: '{' expected
enum X extends EnumDemo
     ^
/tmp/iPxbgkTMWe/EnumDemo.java:8: error: ',', '}', or ';' expected
enum X extends EnumDemo
      ^
/tmp/iPxbgkTMWe/EnumDemo.java:8: error: '}' expected
enum X extends EnumDemo
           ^
3 errors
```

Anyways, an enum can implement any number of Interfaces.

Main.java

```
interface Y
{

}
enum X implements Y
{

}
class EnumDemo
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Output

```
java -cp /tmp/iPxbgkTMWe EnumDemo
Hello, World!
```

java.lang.Enum

- java.lang.Enum is an abstract class and is a direct child of Object class.
- This class implements Serializable and Comparable Interfaces.
- Every enum is a direct child of this class.

values() Method

- To get all values present in an enum, this method is used.
- This Method returns an array of Enum Type which contains all the values present in an enum.
- Enum keyword implicitly provides values() method. This class is not present in java.lang.Enum class and also not present in Object class. This is the speciality of enum keyword which provides values() method. No other keywords provides any implicit methods.

Main.java

Run

Output

```
enum Alphabet
{
    A,B,C,D,E,F,G,H,I,J,K;
}
class EnumDemo
{
    public static void main(String[] args)
    {
        Alphabet[] a = Alphabet.values();
        for(int i=0; i < a.length; i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

```
java -cp /tmp/iPxbgkTMWe EnumDemo
A
B
C
D
E
F
G
H
I
J
K
```

ordinal() Method

- Inside enum, order of Constants is important and we can represent this order of enum constant through Ordinal Value.
- To get the Ordinal Value of each Enum constant, ordinal() method is used.
- This is a final method present in java.lang.Enum class and returns an int Ordinal Value.
- Ordinal Value is just like Index of Array. In Array, each element is located at an index value while in enum each constant is present at Ordinal Value.

Main.java

Run

Output

```
enum Alphabet
{
    A,B,C,D,E,F,G,H,I,J,K;
}
class EnumDemo
{
    public static void main(String[] args)
    {
        Alphabet[] a = Alphabet.values();
        for(int i=0; i < a.length; i++)
        {
            System.out.println(a[i] + "---" + a[i].ordinal());
        }
    }
}
```

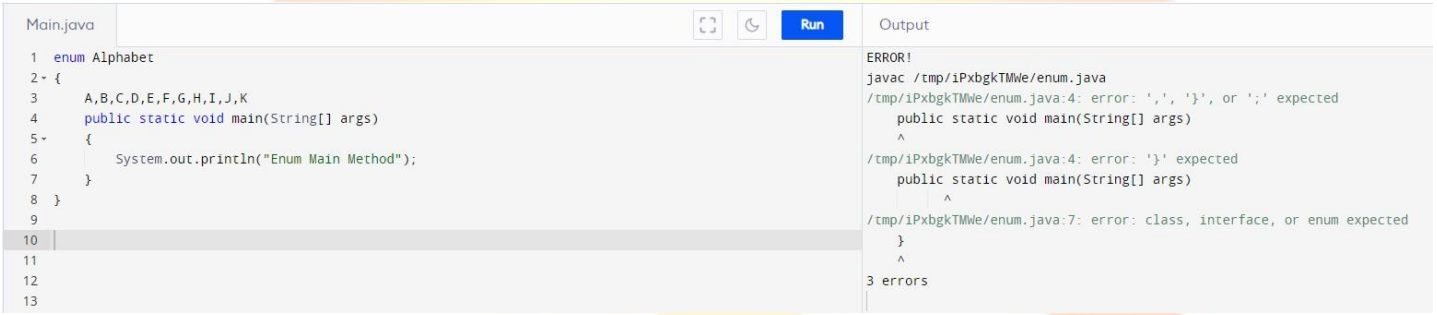
```
java -cp /tmp/iPxbgkTMWe EnumDemo
A---0
B---1
C---2
D---3
E---4
F---5
G---6
H---7
I---8
J---9
K---10
```

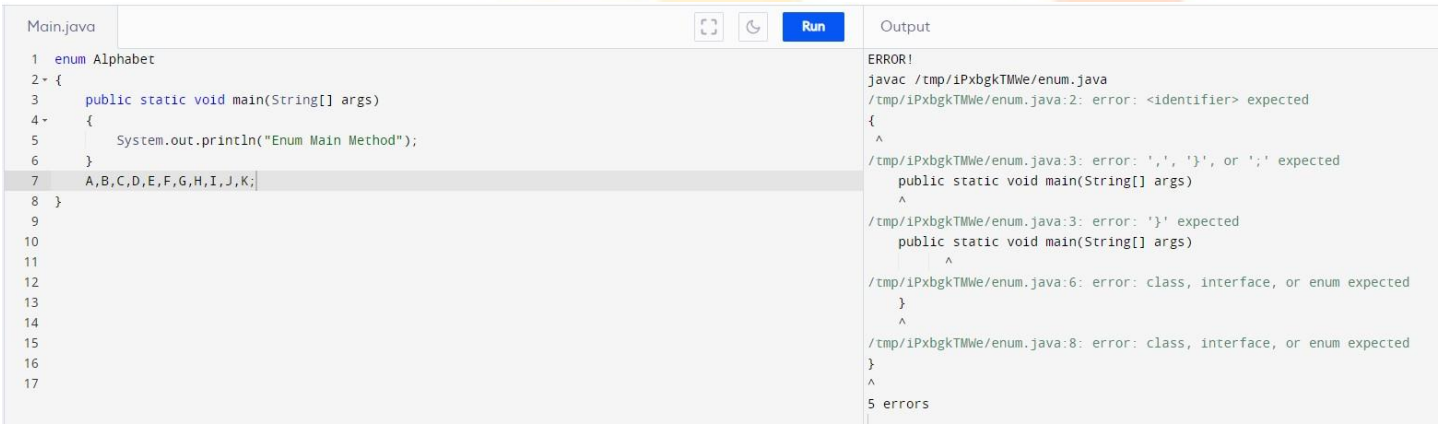
Speciality of Java Enum

- In Old programming language enum, we can have only constant values. While in Java enum, apart from constants we can have methods, constructors, normal variables etc, Hence Java enum is more powerful than old languages enum.
- Even inside java enum we can declare main method and we can run that java class directly from command prompt.

```
enum Alphabet
{
    A,B,C,D,E,F,G,H,I,J,K;
    public static void main(String[] args)
    {
        System.out.println("Enum Main Method");
    }
}
```

- If apart from List of Constants, our enum contains of any extra members like Methods, then it is mandatory that list of enum constants should be at the first line of the enum and compulsory should end with semicolon, if there is no constants to declare in an enum then mandatory semicolon should be there then only we can have our extra methods. If enum contains only constants no other members then this semicolon is completely optional.





Anyways, an empty enum is a valid java syntax.

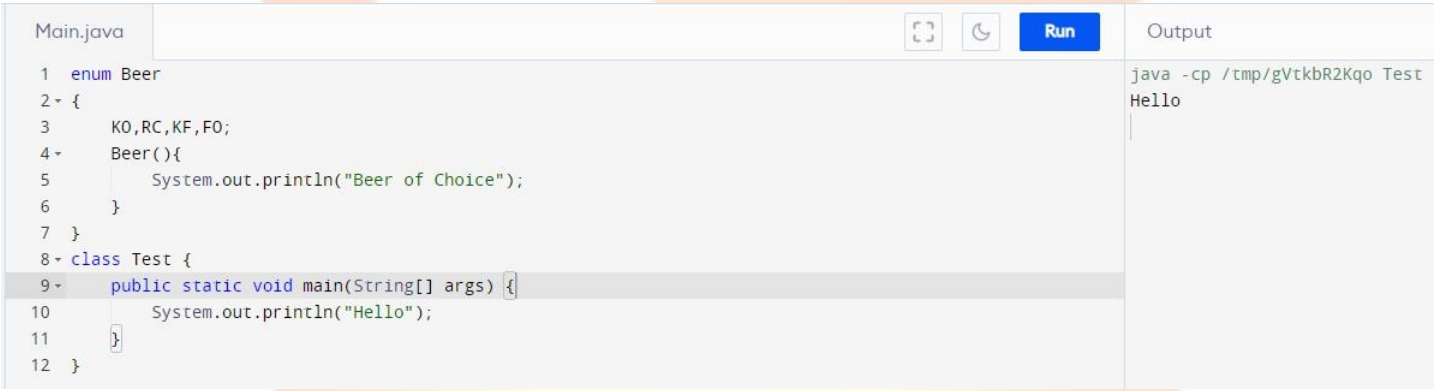
```
enum Alphabet
{

}
enum Vowels
{
    ;
}
```

enum vs Constructors

An enum can contain constructor. An enum constructor can get executed for each enum constants at the time of enum class loading automatically.

When no enum is called in main method:



When enum is called:

When we compile this program with javac Test.java. It creates two class files. Test.class and Beer.class files. When we run this program Line number 10 loads enum class beer which creates an object of all four enum constants and calls constructor 4 times.

We can't create enum object directly and hence we can't invoke enum constructor directly. See the below program on how Java behaves when we try to invoke enum constructor or try to create enum object explicitly.

Main.java

```
enum Beer
{
    KO,RC,KF,FO;
    Beer(){
        System.out.println("Beer of Choice");
    }
}
class Test {
    public static void main(String[] args) {
        Beer b = new Beer();
        System.out.println("Hello");
    }
}
```

Output

```
ERROR!
javac /tmp/gVtkbR2Kqo/Test.java
/tmp/gVtkbR2Kqo/Test.java:10: error: enum types may not be instantiated
Beer b = new Beer();
                ^
1 error
```

We can also pass or set any value to enum constant as well. Example:

enum Beer{

KF(70), KF

}

This enum internally converts enum to class Beer as:

class Beer{

    public static final Beer KF = new Beer(70);

    public static final Beer KF = new Beer();

}

To achieve this, we need to specify two constructors in enum. One is default constructor and while other is a parametrized constructor.

Main.java

```
enum Beer
{
    KO(70),RC(80),KF(90),FO;
    int price;
    Beer(int price){
        this.price = price;
    }
    Beer(){
        this.price = 65;
    }
    public int getPrice()
    {
        return price;
    }
}
class Test {
    public static void main(String[] args) {
        Beer[] b = Beer.values();
        for(Beer b1:b)
        {
            System.out.println(b1+"--->"+b1.getPrice());
        }
    }
}
```

Output

```
java -cp /tmp/gVtkbR2Kqo Test
KO--->70
RC--->80
KF--->90
FO--->65
```

Note: Inside enum, we can declare methods but those methods should be concrete methods. We cannot declare abstract methods inside enum.

There are some cases with respect to enum.

Case 1

Every constant in an enum represents an Object. So whatever methods are applicable for any object, same we can call on enum constant. For example:

Main.java

```
enum Beer
{
    KO,RC,KF,FO
}
class Test {
    public static void main(String[] args) {
        System.out.println(Beer.KF.equals(Beer.KO));
        System.out.println(Beer.KF.hashCode() > Beer.KO.hashCode());
        System.out.println(Beer.KF.ordinal() > Beer.KO.ordinal());
    }
}
```

Output

```
java -cp /tmp/gVtkbR2Kqo Test
false
false
true
```

Case 2

If we want to use any class or Interface from outside package then we have to use normal import in java program. For example –

To use ArrayList we can use

import java.util.ArrayList;

But if we want to access the static contents of a Class without using the class name, then we should go for static import.

import static java.lang.Math.*;

Now, suppose we have an enum called Beer in package say package1;

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

Now, we have class Test.java in package say package2;

```
class Test {
    public static void main(String[] args) {
        Beer b = Beer.KO;
    }
}
```

To have line Beer b = Beer.KO; It is compulsory to go for Normal Import Statement.

And

```
class Test {
    public static void main(String[] args) {
        System.out.println(KO);
    }
}
```

To have System.out.println(KO) directly, we have to go for static import statement.

If we want both Beer b = Beer.KO; and System.out.println(KO), then we have to involve both the imports.

Case 3

We can redefine the normal behaviour for any of the enum constant by overriding the specific method.

Main.java

Run

Output

```
enum Color
{
    BLUE, RED{
        public void info()
        {
            System.out.println("Dangerous Color");
        }
    }, GREEN;
    public void info()
    {
        System.out.println("Universal Color");
    }
}
class Test {
    public static void main(String[] args) {
        Color[] c = Color.values();
        for(Color c1:c)
        {
            c1.info();
        }
    }
}
```

```
java -cp /tmp/gVtkbR2Kqo Test
Universal Color
Dangerous Color
Universal Color
```

Difference between enum, Enum and Enumeration

- Enum is a Class in Java present in java.lang package which is a superclass of all the enum classes which we declare.
- enum is a keyword to declare an user defined enum in Java.
- Enumeration is an interface in Java which can be used to get objects from collection one by one. It is one of the Cursor in Java.