


## Enumerated User-Define Data Type:

Enum data type is used when we want to represent a group named constants then we should use enum keyword. Enum concept introduced in java 1.5 version. The main objective of enum is to define our own data types(enumerated data types). Compared with old languages enum java's enum is more powerful than other old languages. Syntax is as follows. Enum month { Jan, feb, mar, apr,....., dec; }

The internal implementation of enum keyword: 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.

<pre>enum month { Jan, feb, mar ; }</pre>		<pre>class month { public static final month jan = new month[]; public static final month feb = new month[]; public static final month mar = new month[]; }</pre>
---	---	---

Every enum constant is static. Hence we can access by using enum name. inside enum toString() method implemented to return name of the constant directly.

```
/*ENUM DATA TYPE */
enum Name {
    ATISH, LIPUN, IMMORTAL, MRITUNJAY
}

public class JVP04 {
    public static void main(String args[]) {

        Name n1 = Name.MRITUNJAY;
        Name n2 = Name.ATISH;

        System.out.println("name = " + n1); // name = MRITUNJAY
        System.out.println("name = " + n2.toString()); // name = ATISH
    }
}
```

We can declare enum either outside class or inside class, but inside a method If we try to declare enum inside a method then we will get compile time error.

<pre>enum { }</pre>	<pre>class { enum { }</pre>	<pre>class { Public static void main() { enum { }</pre>
allowed	allowed	Not allowed

If we declare enum outside of the class then applicable modifiers are public, default and strictfp.

If we declare enum with in a class then the applicable modifiers are public, default, strictfp, private, protected, static.

Q. WHY NOT FINAL AND ABSTRACT IS NOT ALLOWED IN ENUM?

Every enum is always with final implicitly but we cannot declare final in explicitly so final is not allowed. Final and abstract use simultaneously is an illegal so no abstract is allowed in enum.

## Enum vs Switch Statement:

until 1.4 version the only a allowed argument types for the switch statements are byte, short, char & int. from 1.5 version onwards the corresponding wrapper class and enum type also allowed. From version 1.7 onwards string type is also allowed. Hence version 1.5 version onwards we can pass enum type also as an argument. If we are taking enum type as an argument to switch statement, then every case label should be valid enum constant. If you don't do that you will get the compile time error "an enum switch case label must the unqualified name of an enumeration constant."

JAVA VERSION 1.4	JAVA VERSION 1.5	JAVA VERSION 1.7
byte	Byte	String
short	Short	
char	Char	
int	Int	
	enum	

```
/*enum vs switch statement*/

enum COLOUR {
    RED, BLUE, GREEN, ORANGE
}

public class JVP05 {
    public static void main(String args[]) {
        COLOUR clr = COLOUR.BLUE;

        switch (clr) {
            case RED:
                System.out.println("the colour is = " + clr);
                break;

            case BLUE:
                System.out.println("the colour is = " + clr); // the colour is = BLUE
                break;

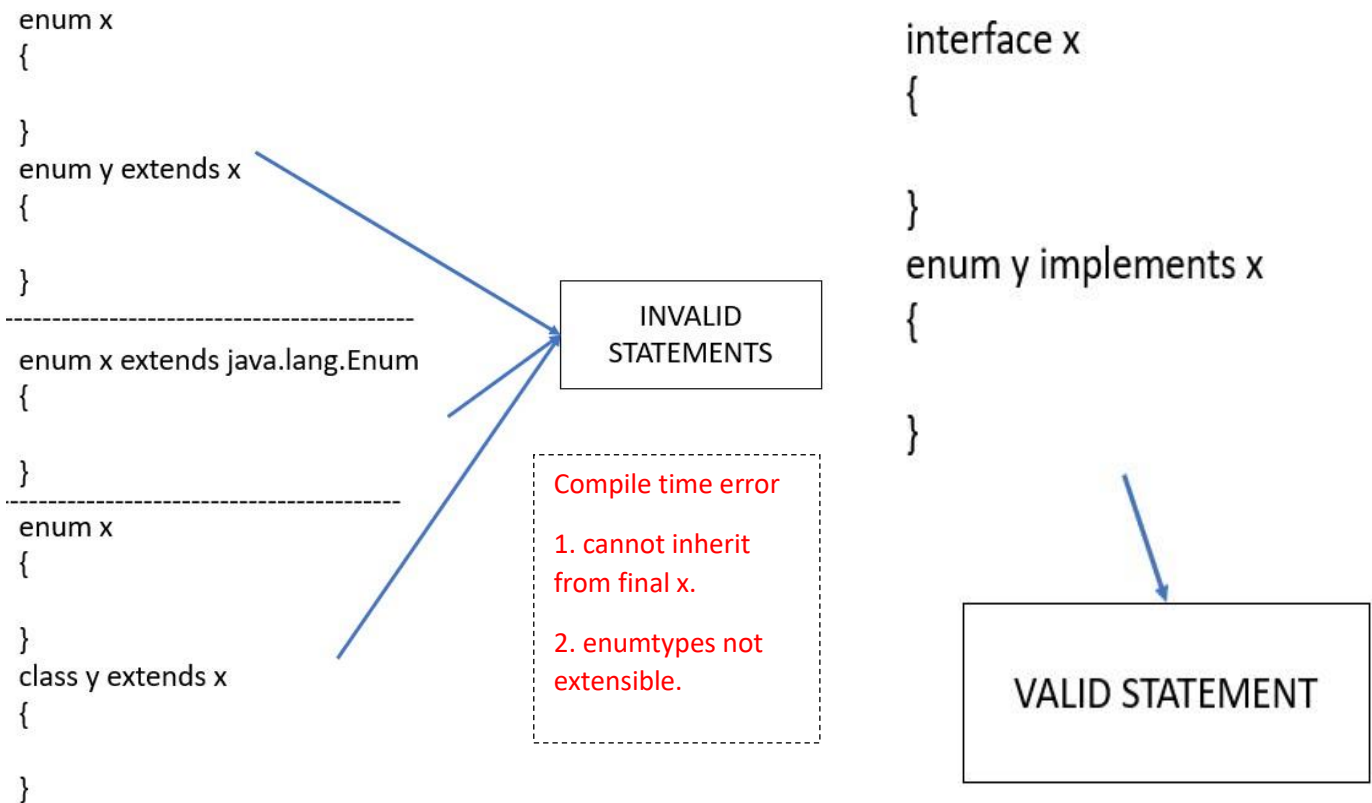
            case GREEN:
                System.out.println("the colour is = " + clr);
                break;

            case ORANGE:
                System.out.println("the colour is = " + clr);
                break;

            default:
                System.out.println("invalid information");
        }
    }
}
```

## enum vs inheritance concept:

every enum in java is a direct child class of "java.lang.Enum;" hence we can't extend any other enum. Every enum is always "final" implicitly and hence we cannot create child enum. Hence because of above reason we can conclude that inheritance concept not applicable for enum. Hence we can't use extend keyword for enum.



## Java.lang.Enum:

Every enum in java is the direct child class of java.lang.Enum hence this class acts as base class for all java enum. It is the direct child class of object and it is an abstract class. it implements "serializable" and "comparable" interfaces.

```
C:\Users\Atish kumar sahu>javap java.lang.Enum
Compiled from "Enum.java"
public abstract class java.lang.Enum<E extends java.lang.Enum<E>> implements java.lang.Comparable<E>, java.io.Serializable {
    public final java.lang.String name();
    public final int ordinal();
    protected java.lang.Enum(java.lang.String, int);
    public java.lang.String toString();
    public final boolean equals(java.lang.Object);
    public final int hashCode();
    protected final java.lang.Object clone() throws java.lang.CloneNotSupportedException;
    public final int compareTo(E);
    public final java.lang.Class<E> getDeclaringClass();
    public static <T extends java.lang.Enum<T>> T valueOf(java.lang.Class<T>, java.lang.String);
    protected final void finalize();
    public int compareTo(java.lang.Object);
}

C:\Users\Atish kumar sahu>
```

## value() & ordinal() in enum:


values() is to list out all values present in enum. Enum keyword implicitly contains to this method. Every enum implicitly contains values() to return all values present inside enum.

Within the enum the order of the constants are implements and we can represent this order by using ordinal values. We can find the ordinal values of enum constant by using ordinal() method. Ordinal values are 0 based like array index.

```
/*value() & ordinal() in enum concept*/
enum alphabet {
    A, B, C, D, E, F, G, H,
    I, J, K, L, M, N, O, P,
    Q, R, S, T, U, V, W, X,
    Y, Z;
}

public class JVP06 {
    public static void main(String args[]) {
        alphabet[] alpha = alphabet.values();
        for (alphabet a1 : alpha) {
            System.out.println("value " + a1.ordinal() + " = " + a1);
        }
    }
}

/*
 * values() = to list out all the constant of enum ordinal() = to give the order
 * value of constant present in enum.
 */
```

 Problems @ Ja

<terminated> JVPC

```
value 0 = A
value 1 = B
value 2 = C
value 3 = D
value 4 = E
value 5 = F
value 6 = G
value 7 = H
value 8 = I
value 9 = J
value 10 = K
value 11 = L
value 12 = M
value 13 = N
value 14 = O
value 15 = P
value 16 = Q
value 17 = R
value 18 = S
value 19 = T
value 20 = U
value 21 = V
value 22 = W
value 23 = X
value 24 = Y
value 25 = Z
```

```
enum enum_test {
    ;
    public static void main(String args[])
    {
        System.out.println("ENUM MAIN METHOD"); //ENUM MAIN METHOD
    }
}
```

In old programming language enum can declare only constants. But in java enum in addition to constants we can also declare methods, constructors, and normal variables etc. even we can declare main() method inside enum. Hence we can invoke enum directly from the command prompt.

In addition to constants if we are declaring any extra members like method then list of constants should be in the first line and should ends with semicolon(;). Inside enum if we are declaring members like methods then it should contain list of constants end with semicolon(;) or at least one semicolon(;). An empty enum is a valid java syntax.

enum A { A,B,C,D ;    //";" is mandatory psvm() { }}            // correct statement	enum A { ; psvm() { } }            //correct statement	enum A {  }            //correct statement
---	--	---

---

enum A { A,B,C,D psvm() { } } }            //invalid	enum A { psvm() { } A,B,C,D ; }            //invalid	enum A { psvm() { }            } }            //invalid
---	--	--

## Enum vs constructor:

enum can contain constructor and it is executed separately for every enum constant at the time of enum class loading. We cannot create enum object explicitly and hence we cannot invoke enum constructor directly. Inside enum we can declare methods but should be concrete methods only we can't declare abstract method.

```
/*constructor in enum concept*/
enum product
{
    A(100),B,C(110),D(235),E;

    int PRICE ;

    product(int price)
    {
        this.PRICE = price; // A = 100,C = 110,D = 235
    }

    product()
    {
        this.PRICE = 114 ; //B = 114, E = 114
    }

    public int getprice()
    {
        return PRICE;
    }
}

public class JVP07 {
    public static void main(String args[])
    {
        product[] p = product.values();

        for(product p1:p)
        {
            System.out.println("product "+p1+ " = "+p1.getprice());
        }
    }
}

/* OUTPUT
product A = 100
product B = 114
product C = 110
product D = 235
product E = 114
*/
```



Every enum constant represent an object of type enum. Hence whatever the methods we can call on normal java object we call same methods on enum constant also.

Beer.kf.equals(Beer.rc);       //correct statement

Beer.kf == Beer.rc ;    //correct statement

Beer.kf.hashCode() > Beer.rc.hashCode();    //correct statement

Beer.kf.ordinal() > Beer.rc.ordinal();    //correct statement

Beer.kf > Beer.rc ;       // this is an invalid statement

```
enum LETTER {
    A, B, C, D, E, F;
}

public class JVP08 {
    public static void main(String args[]) {
        // == between two constants
        System.out.println(LETTER.A == LETTER.B); // false

        System.out.println("-----");

        // show the hashCode of any constants
        System.out.println(LETTER.C.hashCode()); // 366712642
        System.out.println(LETTER.D.hashCode()); // 1829164700

        System.out.println("-----");

        // compare between two hasCode of two constants
        System.out.println(LETTER.D.hashCode() > LETTER.C.hashCode()); // true

        System.out.println("-----");

        // show the order of any constants
        System.out.println(LETTER.E.ordinal()); // 4
        System.out.println(LETTER.F.ordinal()); // 5

        System.out.println("-----");

        // compare between two order of two constants
        System.out.println(LETTER.F.ordinal() < LETTER.E.ordinal()); // false

        System.out.println("-----");

        // equals() between two constants
        System.out.println(LETTER.C.equals(LETTER.F)); // false
    }
}
```

If we want to use class name directly from outside package then required import is normal import. If we want to access static member directly without class name, then required import is static import.

```
import java.util.ArrayList ;
import static java.lang.Math.sqrt;
public class JVP09 {
    public static void main(String args[])
    {
        ArrayList i = new ArrayList();

        System.out.println(sqrt(8)); //2.8284271247461903
    }
}
```

## Package Concept In Enum:

There are two types of package are there general package and static package.

```
package epack1;

public enum fish {
    shark, whale, star, guppy;
}

package epack2;

import epack1.fish;

public class testfish {
    public static void main(String args[]) {
        fish f = fish.star;
        System.out.println(f); // star
    }
}

package epack3;

import static epack1.fish.*;

public class testfish1 {
    public static void main(String args[]) {
        System.out.println(shark); // shark
    }
}

package epack4;

import epack1.*;
import static epack1.fish.*;

public class testfish12 {
    public static void main(String args[]) {
        fish f = fish.whale;
        System.out.println(f); // whale
        System.out.println(guppy); // guppy
    }
}
```

```

/*anonymous class in enum*/
enum random {
    book, pen {
        void info() {
            System.out.println("TITAN PEN");
        }
    },
    bag;

    void info() {
        System.out.println("random product");
    }
}

public class JVP10 {
    public static void main(String args[]) {
        random[] r = random.values();

        for (random r1 : r) {
            r1.info();
        }
    }
}
/*OUTPUT
random product
TITAN PEN
random product
*/

```

## enum vs Enum vs Enumeration:

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

Enum is a class present in java.lang package. Every enum in java is direct child class of Enum class. hence it acts as a base class for all java enum.

Enumeration is an interface present in java.util package. We can use Enumeration to get objects from collection one by one.