Enums

- ✓ Enum is a keyword. Enums in java are used to represent fixed number of well known values in java. For example Boolean value , number of months in a year etc.
- ✓ Enum constants are implicitly static and final ,we cannot change their values once created.
- ✓ Enum in java need to be declared with public or default modifier.
- ✓ One common use of Enum is it is used to design singleton pattern and it is easiest method to handle thread safety and serialization.
- ✓ Enum in java is more rich and versatile than any other language.
  Enum is introduced in JDK 1.5.
- ✓ Before usage of Enum ,the enumerable values are represented like public static final to replicate enum behavior.
- ✓ Let's take one example to discuss the use of Enum

```
class StatusWithoutEnum{
        public static final int STATUS_OPEN = 0;
        public static final int STATUS_STARTED = 1;
        public static final int STATUS_INPROGRESS = 2;
        public static final int STATUS_ONHOLD = 3;
        public static final int STATUS_COMPLETED = 4;
        public static final int STATUS_CLOSED = 5;
}

public class StatusEnum {

}
```

The above technique has some limitations

- **Type safety**: There is no Type safety, that is we can change the values of constants if we want.
- **Compile time constants:** All these constants are compiled and used in the program. If we want to add new constants, add that in the list and recompile everything.
- **Uninformative**: When printed, these are just numbers to the reader. Ex: status =1 then the user have to go and find out what does it exactly mean. Amount of information available here is minimal.
- **Restricted Behavior**: If we want to use these values for remote call or compare them, then we have to handle it explicitly. Serializable and comparable features are not used. Also there is no space to add behavior for statuses. ex: hashcode(),toString(),equals() methods.
- **Meaningless switch-case use:** Her not possible to use variable names to switch statement, instead it uses uninformative numbers. Therefore readability of program goes down.
- **Non iterative list:** This is a list of values. But it is not possible to iterate like the way used in collections.

The solution to above limitations is Java Enum.

```java
public class RequestStatus {

   private final int status;

   private RequestStatus(int status){
      this.status = status;
   }

   public static final RequestStatus STATUS_OPENED = new RequestStatus(0);
   public static final RequestStatus STATUS_STARTED = new RequestStatus(1);
   public static final RequestStatus STATUS_INPROGRESS = new RequestStatus(2);
   public static final RequestStatus STATUS_ONHOLD = new RequestStatus(3);
   public static final RequestStatus STATUS_COMPLETED = new RequestStatus(4);
   public static final RequestStatus STATUS_CLOSED = new RequestStatus(5);

}
```

- ✓ In the above code RequestStatus is actually an object. even though it is Internally an int value, it is represents an object. If we convert int value to String or object we can add any behavior to it.(Serializable , comparable ,toString(),hashcode(),equals()).
- ✓ In Java 5 ,this pattern is further modified to provide a cleaner implementation to bunch of constants. Instead of using list of public static final we can use enum types to group the constants. observe below example

```
package Notes;

public class ReqStatus {
   public enum Status{
               STATUS_OPEN,
      STATUS_STARTED,
      STATUS_INPROGRESS,
      STATUS_ONHOLD,
      STATUS_COMPLETED,
      STATUS_CLOSED;
   }
   public static void main(String[] args) {
      for(Status stat : Status. values()){
         System.out.println(stat);
      }
   }

}
```

Output

```
STATUS_OPEN
STATUS_STARTED
STATUS_INPROGRESS
STATUS_ONHOLD
STATUS_COMPLETED
STATUS_CLOSED
```

Here there is no need to define constants .Instead we can use enum types to store them directly.

Different uses of Java Enums

- We can declare an Enum type inside the class
- Enum with additional fields and custom constructors
- Enum constructors must private or default and public ,protected constructors are not allowed. When constructor is declared ,all elements declaration must match with that constructor. In Enum Constructors we can use this() statement to call current class constructor.

```
enum Season4{

   SPRING(), SUMMER(20.0f), FALL("Keerthana"), WINTER;

   Season4(int i){
      System.out.println("value of i =" +i);
   }

   Season4(float f){
      System.out.println("value of f = " +f);
   }

   Season4(String s){
      System.out.println("value of s = " +s);
   }

   Season4(){
      System.out.println("default constructor");
   }
}
public class O {
   public static void main(String[] args) {

      System.out.println("main begin");
      Season4 s1 = Season4.FALL;
   }

}
```

Output:

```
main begin
default constructor
value of f = 20.0
value of s = Keerthana
default constructor
```

```
enum Season4{

   SPRING(10), SUMMER(20.0f), FALL("Keerthana"), WINTER;

   Season4(int i){
      System.out.println("value of i =" +i);
   }

   Season4(float f){
      System.out.println("value of f = " +f);
   }

   Season4(String s){
      System.out.println("value of s = " +s);
   }

   Season4(){
      System.out.println("default constructor");
   }
}
public class O {
   public static void main(String[] args) {

      System.out.println("main begin");
      Season4 s1 = Season4.FALL;
      Season4 s2 = Season4.SPRING;
      Season4 s3 = Season4.SUMMER;
      Season4 s4 = Season4.WINTER;
   }
}
```

Output:

```
main begin
value of i =10
value of f =20.0 value of s = Keerthana
default constructor
```

```
enum Season4{

  SPRING(10), SUMMER(20.0f), FALL("Keerthana"), WINTER;

  Season4(int i){
    System.out.println("value of i =" +i);
  }

  Season4(float f){
    this("Pallavi");
    System.out.println("value of f = " +f);
  }

  Season4(String s){
    this();
    System.out.println("value of s = " +s);
  }

  Season4(){
    this(20);
    System.out.println("default constructor");
  }
}

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

    System.out.println("main begin");
    Season4 s1 = Season4.FALL;
    Season4 s2 = Season4.SPRING;
    Season4 s3 = Season4.SUMMER;
    Season4 s4 = Season4.WINTER;
  }

}
```

Output:

```
main begin
value of i =10
value of i =20
default constructor
value of s = Pallavi
value of f = 20.0
value of i =20
default constructor
value of s = Keerthana
value of i =20
default constructor
```

```java
enum Season4{

  SPRING(10), SUMMER(20.0f), FALL("Keerthana"), WINTER;

   Season4(int i){
     System.out.println("value of i =" +i);
  }

  Season4(float f){
     this("Pallavi");
     System.out.println("value of f = " +f);
  }

  Season4(String s){
     this();
     System.out.println("value of s = " +s);
  }

  Season4(){
     this(20);
     System.out.println("default constructor");
  }
}

public class O1 {
  public static void main(String[] args) {

     System.out.println("main begin");
     Season4 s1 = Season4.FALL;
     System.out.println("------");
     Season4 s2 = Season4.SPRING;
     System.out.println("----------");
     Season4 s3 = Season4.SUMMER;
     System.out.println("----------");
     Season4 s4 = Season4.WINTER;
  }

}
```

Output:

```
main begin
value of i =10
value of f = 20.0
value of s = Keerthana
default constructor
------
----------
---------
```

```
enum Season4{

  SPRING(10), SUMMER(20.0f), FALL("Keerthana"), WINTER;

  Season4(int i){
    System.out.println("value of i =" +i);
  }

  Season4(float f){
    this("Pallavi");
    System.out.println("value of f = " +f);
  }

  Season4(String s){
    this();
    System.out.println("value of s = " +s);
  }

  Season4(){
    this(20);
    System.out.println("default constructor");
  }
}

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

    System.out.println("main begin");
    Season4 s1 = Season4.FALL;
    System.out.println("------");
    Season4 s2 = Season4.SPRING;
    System.out.println("----------");
    Season4 s3 = Season4.SUMMER;
    System.out.println("----------");
    Season4 s4 = Season4.WINTER;
  }

}
```

Output:

```
main begin
value of i =10
value of f = 20.0
value of i =20
default constructor
value of s = Keerthana
value of i =20
default constructor
------
----------
----------
```

```
enum Season4{

   SPRING(10), SUMMER(20.0f), FALL("Keerthana"), WINTER;

   Season4(int i){
      System.out.println("value of i =" +i);
   }

   Season4(float f){
      System.out.println("value of f = " +f);
   }

   Season4(String s){
      System.out.println("value of s = " +s);
   }

   Season4(){
      System.out.println("default constructor");
   }
}

   enum Season5{
      SPRING,SUMMER,FALL,WINTER;
      Season5(){
         System.out.println("default constructor");
      }
   }

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

      System.out.println("main begin");
      Season4 s1 = Season4.FALL;
      System.out.println("-------");
      Season5 s5 = Season5.SPRING;

   }

}
```

```
Output:
main begin
value of i =10
value of f = 20.0
value of s = Keerthana
default constructor
--------
default constructor
default constructor
default constructor
default constructor

We can declare any number of enum within the same class.
```

Enum that implement interfaces.
We can implement any interfaces. Within interface we can declare Enum.

```java
public class A1 {
   enum Season{
      SPRING,SUMMER,FALL,WINTER;
      interface A2{
         void print();
         void print1();
      }
   }

   public static void main(String[] args)
{
      Season s1 = Season. FALL;
      Season s2 = Season. SPRING;
      Season s3 = Season. FALL;
      System.out.println(s1);
      System.out.println(s2);
      System.out.println(s3);
      A3 a3 = new A3();
      a3.print();
   }

}

import enum1.A1.Season.A2;

public class A3 implements A2 {
   public void print(){
      System.out.println("Hello");
   }

}
```

Output :

```
FALL
SPRING
FALL
Hello
```

If we are using interface inside an Enum it is not possible to implement the methods within the same class. we can instantiate the class implemented in the class we declare interface.

```
public class A2 {

   void print(){
      enum Days{
         SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY;
      }
   }

   public static void main(String[] args) {

   }

}
```

Output : SUNDAY

- Inside Enum we cannot declare abstract class and inside abstract class we can declare Enum.

```
   abstract class A3{
      enum Days{
         SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY;
      }
   }

 public class A2 extends A3{
  public static void main(String[] args) {
      Days d1 = Days. SUNDAY;
      Days d2 = Days. SATURDAY;
      System.out.println(d1);
      System.out.println(d2.ordinal());
   }
}
```

Output :

```
SUNDAY
6
```

```
public class A4 {

  enum Enum1{
    SPRING, SUMMER, FALL, WINTER;

    abstract class A5{
      abstract void display();
       void display1(){
         System.out.println("Keerthana Technologies");
      }
     }
    }
    public static void main(String[] args) {
      Enum1 s1 = Enum1.SPRING;
      A6 a6 = new A6();
      a6.display();
      a6.display1();

    }
   }

import enum1.A4.Enum1.A5;

public class A6 extends A5 {
  A6 a6 = new A6();
  void display(){
    System.out.println("groups");
  }



}
```

Output: CTE

```
  No enclosing instance of type A4.Enum1 is available due to some
intermediate constructor invocation
```

- Inside Enum we can declare IIB , SIB ,Static members and Instance members.

```
public class C1 {

  enum Pallavi{
    MALE,FEMALE;
    static int x;
    int y =30;

    static{
      System.out.println("SIB");
    }

    {
      System.out.println("IIB");
    }

    static void m1(int...s){
      System.out.println("int");
    }

    void m2(){
        System.out.println("hello");
      }
    }

    public static void main(String[] args) {
      Pallavi p1 = Pallavi.FEMALE;
      System.out.println(Pallavi.FEMALE.ordinal());
      System.out.println(p1.y);
      System.out.println(p1.x);
      p1.m1();
      p1.m2();
    }
  }
Output :
IIB
IIB
SIB
1
30
0
int
hello
```

- Enum is not possible to declare inside a method or a block.

```
public class A2 {

   void print(){
      enum Days{
         SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY;
      }
   }

   public static void main(String[] args) {

   }
}

Output :
CTE
```

- We can use Enum type in switch case also like int and char values.

```
public class B2 {

   enum Season{
      SPRING,SUMMER,FALL,WINTER;
   }
   public static void main(String[] args) {
      Season s1 = Season. SPRING;
      Season s2 = Season. SUMMER;
      Season s3 = Season. FALL;
      Season s4 = Season. WINTER;

      switch(s3){
        case SPRING :
            System.out.println("I am SPRING");
            break;
        case SUMMER :
             System .out. println("I am SUMMER");
             break;
        case FALL:
             System.out.println("I am FALL");
             break;
        case WINTER:
             System.out.println("I am WINTER");
             break;
         default:
            System.out.println("NOTHING");
            break;
      }
   }
}
Output : I am FALL
```

- All Enums and Enum constants have some built in or predefined methods such as values(),valueOf(),name(),ordinal().

  values(): This method returns the array of enumeration constants.

  valueOf(): This method returns enum constant that corresponds to the String passed as an argument.

  Ordinal(): This method is used to get the position of enum declaration. It is not mandatory.

  name(): This method returns the name of the enum constant. Its return type is String.

```
public class A {
   enum Month{
      JANUARY,FEBRAUARY,MARCH,APRIL,MAY,JUNE,JULY,
      AUGUST,SEPTEMBER,OCTOMBER,NOVEMBER,DECEMBER;
   }

   public static void main(String[] args) {
      Month m1[] = Month. values();
      for(Month temp: m1){
         System.out.println(temp);
      }
   }
}
```

Output :

```
JANUARY
FEBRAUARY
MARCH
APRIL
MAY
JUNE
JULY
AUGUST
SEPTEMBER
OCTOMBER
NOVEMBER
DECEMBER
```

- ✓ If we don't specify anything to values() method, there is run time error like IllegalArgumentException.

```
public class B {
   enum Nature{
      TREE,STONE,WIND,WATER,FIRE;
   }

   public static void main(String[] args) {
    Nature n1 = Nature.valueOf("TREE");
    Nature n2 = Nature.valueOf("STONE");
    Nature n3 = Nature.valueOf("WIND");
    System.out.println("n1=" +n1+ "n2 = " +n2+ " n3 = " +n3);
   }
}
```

Output : `n1 = TREE n2 = STONE n3 = WIND`

```
package enum1;

public class B1 {
   enum Boolean{
      FALSE,TRUE
      ;
      abstract class B2{
         void print(){
            System.out.println("Hello");
         }
         abstract void print1();
      }
       }
      public static void main(String[] args) {
         System.out.println(Boolean. TRUE);
         System.out.println(Boolean.FALSE.ordinal());
      }
}
```

Output: `True 0`

```
import Note.A.Month;

public class C {

   enum Month1{
      JANUARY,FEBRAUARY,MARCH,APRIL,MAY,JUNE,JULY,
      AUGUST,SEPTEMBER,OCTOMBER,NOVEMBER,DECEMBER;
   }

   public static void main(String[] args) {
      Month1 m1 = Month1.JANUARY;
      Month1 m2 = Month1.JULY;
      String s1 = m1.name();
      System.out.println(s1);
      System.out.println(m2.ordinal());
      System.out.println(Month.AUGUST.ordinal());
      System.out.println(Month. valueOf("APRIL"));

   }

}
```

Output :

```
JANUARY
6
7
APRIL
```