Enumeration

If we want to represent a group of named constants the we should go for enum. Purpose of enum is to define our own data type (Enumerated Data Type). Like Byte is predefined data type Month is own defined data type. Enum concept introduced in 1.5 version. When compared with old languages enum Java enum is more powerful.

Internal Implementation of Enum

- Every enum is internally implemented by class concept.
- Every enum constant is always public static final.
- Every enum constant represents an object of the type enum.

```
enum Month
{
	JAN,FEB,MARCH;
}
```

Is equivalent to

```
class Month
{
	public static final JAN=new Month();
	public static final FEB=new Month();
	public static final MARCH=new Month();

}
```

Enum Declaration and Usage

Since enum members are static so it can be accessed through enum name itself. Since it is object of type enum so it will be referenced through enum variable.

```
enum Month
{
	JAN,FEB,MARCH; //semicolon is optional
}

public class EnumDemo {

	public static void main(String[] args) {
		Month month=Month.JAN;
		System.out.println(month);
	}
}
```

Output-JAN

Note- Whenever we try to print reference variable internally toString() method will be called. Inside enum toString() method is internally implemented to return name of the constant.

Enum can be declared inside a class also like inner class.

Can enum be declared inside a method like local inner class?

Class can be defined inside a method but enum cannot as all the constants are static and static declaration inside a method is not defined in java as static is class level.

```
public class EnumDemo {
	enum Month
	{
		JAN,FEB,MARCH;//semicolon is optional
	}

	public static void main(String[] args) {
		Month month=Month.FEB;
		System.out.println(month);
	}


}
```

Access modifier with enum

- ✓ *Enum outside class*

Modifiers allow with class is public, default, final, abstract and strictfp. So these modifiers should be allowed with enum as well. Every enum is final implicitly though we cannot declare it is final. If enum is final definitely abstract cannot be declared. So allowed modifiers are public, default and strictfp. Same thing we covered in inner class.

- ✓ *Enum inside a class*

Along with these three modifiers private, protected and static.

Enum vs Switch

Until 1.4 version the allowed arguments type for the switch statement are byte, short, int, char but from 1.5 version onwards corresponding wrapper class and enum types are allowed (Because autoboxing and unboxing was introduced in this version). From 1.7 version onwards string type also allowed. Hence from 1.5 version onwards we can pass enum type argument in switch statement.

```
public class EnumDemo {
	enum Month
	{
		JAN,FEB,MARCH,APRIL,MAY,JUNE//semicolon is optional
	}

	public static void main(String[] args) {
		Month month=Month.FEB;
		switch(month)
		{
		case JAN :
			System.out.println("January");
			break;
		case FEB :
			System.out.println("Febuary");
			break;
		}
	}
}
```

Suppose in case we are passing SUNDAY in case what will happen. It will be compile time error as it is not present in enum. In case we can pass only those values which are present in enum.

Can we define Enum inside a method like method local inner class?

Answer- Since all the members of enum is static so we cannot declare it. Static members are not allowed inside static/non-static methods.

Enum vs Inheritance

Every enum is direct child class of Java.lang.Enum. So it cannot extend any other enum (because multiple inheritance is not allowed in Java). Every is always final implicitly and hence for our enum we cannot create child enum. We cannot use extends keyword with enum.

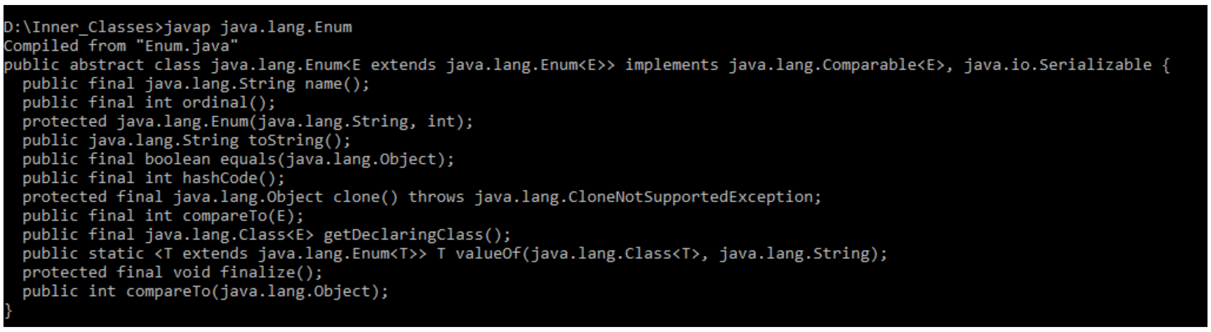
Enum extends java.lang.Enum – Though enum implicitly implicitly extends java.lang.Enum but we cannot use. Compiler will check extends keyword after enum and will give error.

Note : **public class** Switch_Case **extends** Object is happily accepted by compiler.

Because of above reason we can conclude Inheritance concept not applicable for enum.

Anyway an enum can implement any number of interfaces.

Java.lang.Enum

1. Every enum in Java is the direct child class of Java.lang.Enum and hence this class acts as base class for all Java enums.
2. It is an abstract class. It implements serializable and comparable interfaces so it can be passed over the network and can be compared with other enum as well.
3. 

```
D:\Inner_Classes>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);
}
```

<u>Values() method</u>

Every enum implicitly provides values() method to list out all the values. Since values are object of enum type so it will be hold in array of enum type.

Months all[]=Months.*values*();

Note- As per syntax of values() method it should be static method of Months enum but it is not. So it should be present in super of Months i.e in java.lang.Enum but it is not present here also. So it should be in super of java.lang.Enum i.e in Object class. But it is not present here also. Enum keyword implicitly provides values() method.

<u>Public final int ordinal() method</u>

Inside enum order of constant is important and we can represent this order using ordinal value (like index). We can find it by using ordinal() method. Ordinal value is zero based like array.

```
public class EnumDemo {
	public enum Months
	{
		JAN,FEB,MAR,APR,MAY,JUN
		
	}
	
	public static void main(String[] args) {
		
		Months all[]=Months.values();
		for(Months months : all)
		{
			System.out.println("    "+months+" index= "+
months.ordinal());
		}
	}

}
```

Output-

```
JAN index= 0
FEB index= 1
MAR index= 2
APR index= 3
MAY index= 4
JUN index= 5
```

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 enum.
- ❖ Even in Java enum we can declare main method and we can run enum class directly from command promt.

```
public enum Enum1 {
	Constant1,Constant2;
	public static void main(String args[])
	{
		System.out.println("enum main method");
	}

}
```

Output- enum main method

Note – in enum if we are taking constants only then semicolon is optional.

But in addition to constants if we are taking anything in enum then constants must be in first line and should must end with semicolon.

```
public enum Enum1 {

	public static void main(String args[])
	{
		System.out.println("enum main method");
	}
	Constant1,Constant2;//compile time error in this line
}
```

<u>Inside enum if we are taking any extra member like a method compulsory first line should constants if not at least semicolon.</u>

```
public enum Enum1 {

	public static void main(String args[])
	{
		System.out.println("enum main method");
	}

}
```

The above code is invalid as there is no constants defined in first line.

```
public enum Enum1 {
	;

	public static void main(String args[])
	{
		System.out.println("enum main method");
	}

}
```

Now the code compiles fine due to semicolon. Which is list of zero constants.

An empty enum is valid syntax in Java as it does not have any members.

```
public enum Enum1 {

}
```

Even semicolon not required as there is no constants defined.

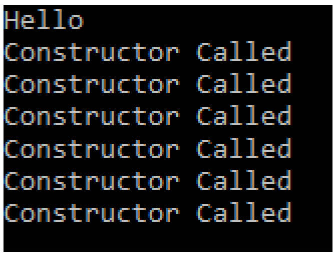
Enum vs Constructor

An enum can contain constructor. Enum constructor separately for every enum constant at the time enum class loading automatically.

Example-

```
enum Months
{
	JAN,FEB,MAR,APR,MAY,JUN;
	Months()
	{
		System.out.println("Constructor Called");
	}
}

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

		System.out.println("Hello");
		Months month=Months.JAN;
	}
}
```

Output-



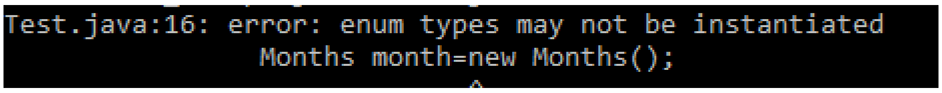
```
Hello
Constructor Called
Constructor Called
Constructor Called
Constructor Called
Constructor Called
Constructor Called
```

If we remove the line `Months month=Months.JAN;`
Constructor will not be called because nowhere enum is getting called. As we know all static members are created at the time of class loading. *As soon as enum will be called all the enum objects will be created one by one and constructor* will be called. Whether we call Months once or twice constructor will be created as number of enum objects.

Can we create enum object using new keyword?

Ans – We cannot create directly and hence we cannot invoke enum constructor directly.

Months JUL=new Months();



```
Test.java:16: error: enum types may not be instantiated
                Months month=new Months();
```

Compile time error because we are misusing enum syntax. Why to use new keyword. We can directly declare it along with remaining six months. It will create objects automatically for you.

If we cannot instantiate enum using new keyword then how value will be passed to parameterized constructor?

Ans- There is other way. Let us think this way.

Enum JAN what means internally is-

Public static final Months JAN=new Months();// it is like no arg constructor

If we internally implement like this

Public static final Months JAN=new Months(1);// it is like no arg constructor

Then parameterized constructor will be invoked. The equivalent declaration will be ***JAN(1)***

Let us take a full fledge example of enum

```
enum Months
{
	JAN(1),FEB(2),MAR(3),APR(4),MAY(5),JUN();//objects

	int monthNumber;//normal variable
	Months(int monthNumber)//parameterized constructor
	{
		this.monthNumber=monthNumber;
	}
	Months()// no-arg constructor
	{
			this.monthNumber=6;
		}
		public int getMonthNumber()//method
		{
			return monthNumber;
		}
}

class Enum1
{
		public static void main(String args[])
		{

			Months months[]=Months.values();
			for(Months sixMonths : months)
			{
				System.out.println(sixMonths+" : 
"+sixMonths.getMonthNumber());
			}
		}

}
```

Output-

```
JAN : 1
FEB : 2
MAR : 3
APR : 4
MAY : 5
JUN : 6
```

Can we define abstract method inside an enum?

Ans- Definitely NO. If we are writing abstract method on a class then its implementation must be provided in child class. As enum is final so cannot be extended. So we can define only concrete methods.

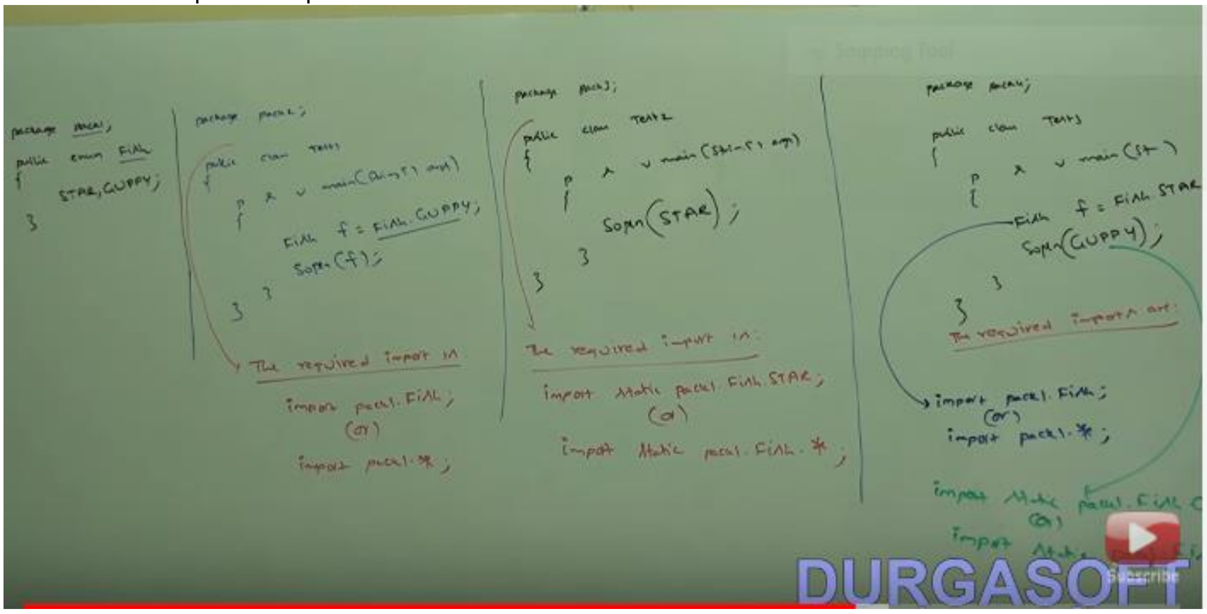
Important Cases

- ✓ Every enum constant represents an object of the type enum hence whatever method we can apply on normal java object, can be applicable on enum constants also.
  Some valid statements-

  ```
  System.out.println(Months.APR.equals(Months.FEB));
  System.out.println(Months.FEB.hashCode()>Months.MAR.hashCode());
  System.out.println(Months.JAN.ordinal()<Months.JUN.ordinal());
  ```
- ✓ If we want to access any class or interface directly from outside package then the required import is normal import.
  If we want to access static members directly without class name then the required import is static import.

```
package basic_syntax;

import java.util.ArrayList;
 import static java.lang.Math.sqrt;;
```

```java
public class Arithmetic_Operators {

	public static void main(String[] args) {

		ArrayList al=new ArrayList();
		System.out.println(sqrt(36));
	}

}
```
1-normal import required
2-static import required
3-both the import required



```
package pack1;
public enum Fish
{
   STAR,GUPPY;
}
```

```
package pack2;
public class Test1
{
   p s v main(String[] args)
   {
      Fish f = Fish.GUPPY;
      Sopln(f);
   }
}
The required import is
import pack1.Fish;
   (or)
import pack1.*;
```

```
package pack3;
public class Test2
{
   p s v main(String[] args)
   {
      Sopln(STAR);
   }
}
The required import is
import static pack1.Fish.STAR;
   (or)
import static pack1.Fish.*;
```

```
package pack4;
public class Test3
{
   p s v main(St
   {
      Fish f = Fish.STAR
      Sopln(GUPPY);
   }
}
The required imports are
import pack1.Fish;
   (or)
import pack1.*;
import static pack1.Fish.G
   (or)
import static
```

✓ Overriding over constants

Consider the following code-

```java
package basic_syntax;
enum Month
{
	JAN,JUN
	{
		public  void getMonthInfo()

		{
			System.out.println("JUN - SUMMER");//Overridden method
		}
	},
	AUG
	{
	public  void getMonthInfo()

	{
		System.out.println("AUG - RAINY");
	}
	}

	,MAR;
	public  void getMonthInfo()
```

```
        {
                System.out.println("Not a special month");
        }

}


public class EnumDemo {


        public static void main(String[] args) {

                Month sixMonth[]=Month.values();
                for(Month month : sixMonth)
                {
                        month.getMonthInfo();
                }


        }

}
```
Output-

```
Not a special month
JUN - SUMMER
AUG - RAINY
Not a special month
```

What is error in the below code?

```
package basic_syntax;
enum Month
{
        JAN,JUN
        {
                public  void getMonthInfo()

                {
                        System.out.println("Not a special month");// static method
cannot overridden - Compile time error
                }
        },
        AUG,DEC;
        public static void getMonthInfo()// Since it is static method

        {
                System.out.println("Not a special month");
        }

}


public class EnumDemo {


        public static void main(String[] args) {

                Month.getMonthInfo();
        }

}
```

- ✓ enum vs Enum vs Enumeration
  - enum – it is keyword in Java used to define a group of named constants.
  - Enum – it is a class in Java acts as base class for enum.
  - Enumeration -it is an interface present in java.util package. We can use enumeration object to get objects one by one from the collection like iterator.

  hjhj