ENUM

- We can use enum to define a group of named constants.

Example 1:

```
enum Month
{
JAN,FEB,MAR,DEC;
}
```

Example 2:

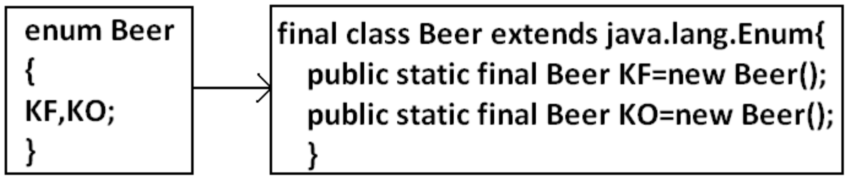
```
enum Beer
{
KF,KO,RC,FO;
}
```

- Enum concept introduced in 1.5 versions.
- When compared with old languages enum java's enum is more powerful.
- By using enum we can define our own data types which are also come enumerated data types.

Internal implementation of enum:

- Internally enum's are implemented by using class concept. Every enum constant is a reference variable to that enum type object.
- Every enum constant is implicitly **public static final** always.

Example 3:



```
enum Beer
{
KF,KO;
}
```

```
final class Beer extends java.lang.Enum{
   public static final Beer KF=new Beer();
   public static final Beer KO=new Beer();
   }
```

Diagram:

Declaration and usage of enum:

Example 4:

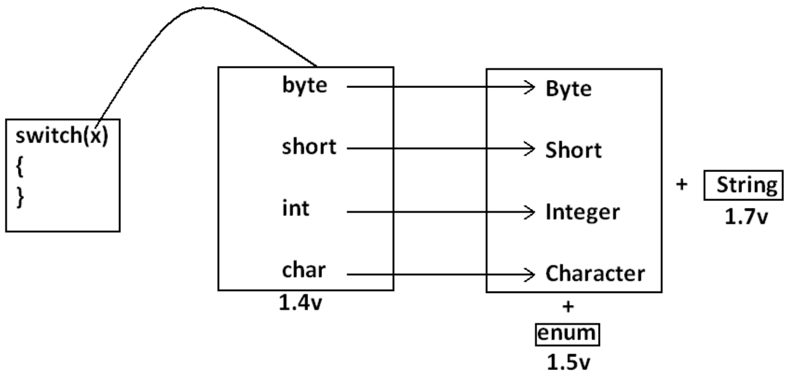
```
enum Beer
{
KF,KO,RC,FO;//here semicolon is optional.
}
class Test
{
public static void main(String args[]){
```

```
Beer b1=Beer.KF;
System.out.println(b1);
}
}
```
Output:
```
D:\Enum>java Test
KF
```
Note: Every enum constant internally static hence we can access by using “enum name”.

Enum vs switch statement:

- Until 1.4 versions the allowed types for the switch statement are byte, short, char int. But from 1.5 version onwards in addition to this the corresponding wrapper classes and enum type also allowed. That is from 1.5 version onwards we can use enum type as argument to switch statement.

Diagram:



Example:
```
enum Beer
{
KF,KO,RC,FO;
}
class Test{
public static void main(String args[]){
Beer b1=Beer.RC;
switch(b1){
case KF:
	System.out.println("it is childrens brand");
	break;
case KO:
	System.out.println("it is too lite");
	break;
case RC:
```

```
            System.out.println("it is too hot");
            break;

case FO:
            System.out.println("buy one get one");
            break;
default:
            System.out.println("other brands are not good");
            }
}}
```

Output:

```
D:\Enum>java Test
It is too hot
```

- If we are passing enum type as argument to switch statement then every case label should be a valid enum constant otherwise we will get compile time error.

Example:

```
enum Beer
{
KF,KO,RC,FO;
}
class Test{
public static void main(String args[]){
Beer b1=Beer.RC;
switch(b1){
case KF:
case RC:
case KALYANI:
}}}
```

Output:

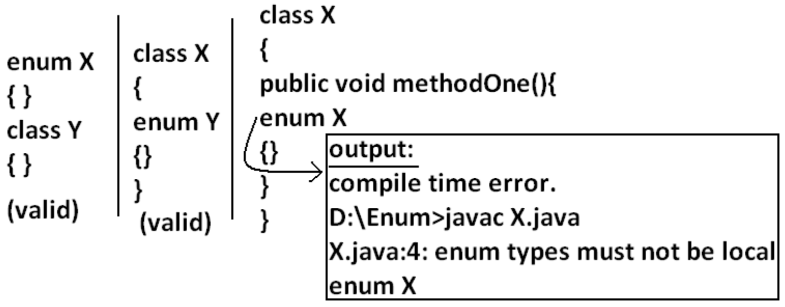
Compile time error.

```
D:\Enum>javac Test.java
Test.java:11: unqualified enumeration constant name required
case KALYANI:
```

- We can declare enum either outside the class or within the class but not inside a method. If we declare enum outside the class the allowed modifiers are:
    1) public
    2) default
    3) strictfp.

- If we declare enum inside a class then the allowed modifiers are:
  1) public   private
  2) default + protected
  3) strictfp   static

Example:



```
enum X
{ }
class Y
{ }
(valid)
```

```
class X
{
enum Y
{}
}
(valid)
```

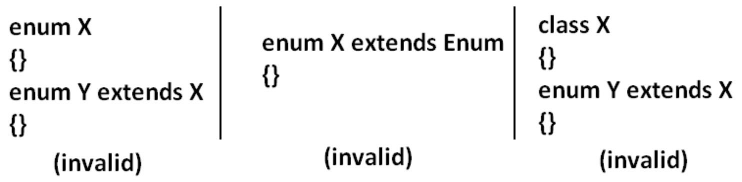
```
class X
{
public void methodOne(){
enum X
{}
}
}
```

```
output:
compile time error.
D:\Enum>javac X.java
X.java:4: enum types must not be local
enum X
```

Enum vs inheritance:

- Every enum in java is the direct child class of java.lang.Enum class hence it is not possible to extends any other enum.
- Every enum is implicitly final hence we can't create child enum.
- Because of above reasons we can conclude inheritance concept is not applicable for enum's explicitly.
- But enum can implement any no. Of interfaces simultaneously.

Example:

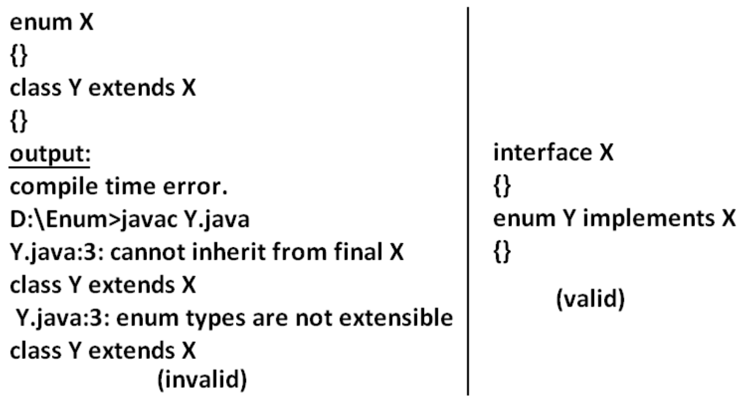


```
enum X
{}
enum Y extends X
{}
(invalid)
```

```
enum X extends Enum
{}
(invalid)
```

```
class X
{}
enum Y extends X
{}
(invalid)
```

Example:



```
enum X
{}
class Y extends X
{}
output:
compile time error.
D:\Enum>javac Y.java
Y.java:3: cannot inherit from final X
class Y extends X
 Y.java:3: enum types are not extensible
class Y extends X
(invalid)
```

```
interface X
{}
enum Y implements X
{}
(valid)
```

Java.lang.Enum: Every enum in java is the direct child class of java.lang.Enum. The power of enum is inheriting from this class only.

- It is abstract class and it is direct child class of “Object class” it implements **Serializable** and **Comparable**.

values() method: Every enum implicitly contains a static values() method to list all constants of enum.

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

ordinal() method: Within enum the order of constants is important we can specify by its ordinal value.

- We can find ordinal value(index value) of enum constant by using ordinal() method.

Example: public int ordinal();

Example:

```
enum Beer
{
KF,KO,RC,FO;
}
class Test{
public static void main(String args[]){
Beer[] b=Beer.values();
for(Beer b1:b)//this is forEach loop.
{
System.out.println(b1+"......."+b1.ordinal());
}}}
```

Output:

```
D:\Enum>java Test
KF.......0
KO.......1
RC.......2
FO.......3
```

Specialty of java enum: When compared with old languages enum java’s enum is more powerful because in addition to constants we can take normal variables, constructors, methods etc which may not possible in old languages.

- Inside enum we can declare main method and even we can invoke enum directly from the command prompt.

Example:

```
enum Fish{
GOLD,APOLO,STAR;
public static void main(String args[]){
```

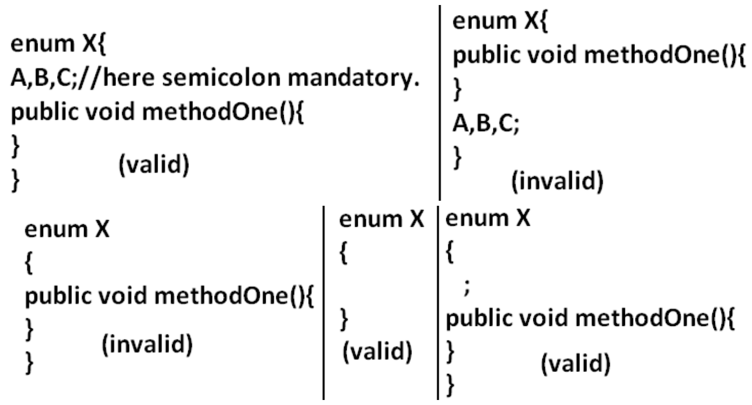
```
System.out.println("enum main() method called");
}}
```

Output:

```
D:\Enum>java Fish
enum main() method called
```

- In addition to constants if we are taking any extra members like methods then the list of constants should be in the 1st line and should ends with semicolon.
- If we are taking any extra member then enum should contain at least one constant. Any way an empty enum is always valid.

Example:



```
enum X{
A,B,C;//here semicolon mandatory.
public void methodOne(){
}
}          (valid)
```

```
enum X{
public void methodOne(){
}
A,B,C;
}
       (invalid)
```

```
enum X
{
public void methodOne(){
}
}          (invalid)
```

```
enum X
{

}
(valid)
```

```
enum X
{
  ;
public void methodOne(){
}          (valid)
}
```

Enum vs constructor: Enum can contain constructor. Every enum constant represents an object of that enum class which is static hence all enum constants will be created at the time of class loading automatically and hence constructor will be executed at the time of enum class loading for every enum constants.

Example:

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

Output:

```
D:\Enum>java Test
Constructor called.
Constructor called.
Constructor called.
Constructor called.
Hello.
```

- We can’t create enum object explicitly and hence we can’t invoke constructor directly.

Example:

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

Output:

Compile time error.

```
D:\Enum>javac Test.java
Test.java:9: enum types may not be instantiated
Beer b=new Beer();
```

Example:

```
KF==>public static final Beer KF=new Beer();
KF(100)==>public static final Beer KF=new Beer(100);
```

```
enum Beer
{
KF(100),KO(70),RC(65),Fo(90),KALYANI;
int price;
Beer(int price){
this.price=price;
}
Beer()
{
this.price=125;
}
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());
}}}
```

- Inside enum we can take both instance and static methods but it is not possible to take abstract methods.

Case 1:

- Every enum constant represents an object hence whatever the methods we can apply on the normal objects we can apply the same methods on enum constants also.

Which of the following expressions are valid?

1) Beer.KF==Beer.RC---------------------------> false
2) Beer.KF.equals(Beer.RC) ------------------->false
3) Beer.KF<Beer.RC----------------------------->invalid
4) Beer.KF.ordinal()<Beer.RC.ordinal()------>valid

Case 2:

Example 1:

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

Example 2:

```
package pack2;
//import static pack1.Fish.*;
import static pack1.Fish.STAR;
class A
{
public static void main(String args[]){
System.out.println(STAR);
}
}
```

1) Import pack1.*; --------------------------->invalid

2) Import pack1.Fish; ------------------------>invalid
3) import static pack1.Fish.*; --------------->valid
4) import static pack1.Fish.STAR; ---------->valid

Example 3:

```
package pack3;
//import pack1.Fish;
import pack1.*;
//import static pack1.Fish.GUPPY;
import static pack1.Fish.*;
class B
{
public static void main(String args[]){
Fish f=Fish.STAR;
System.out.println(GUPPY);
}
}
```

Case 3:

```
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();
}
}
}
```
<u>Output:</u>
```
Universal color
Dangerous color
Universal color
```