enum (enumeration)

(*) Introduction

→ If we want to represent a group of named constants then we should go for "enum".

→ Ex.

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

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

semi-colon is Optional.

→ The main objective of enum is to define our own datatypes (enumerated data types)

→ enum concept introduced in "1.5v".

→ When compared with old language's enum, Java enum is more powerful.

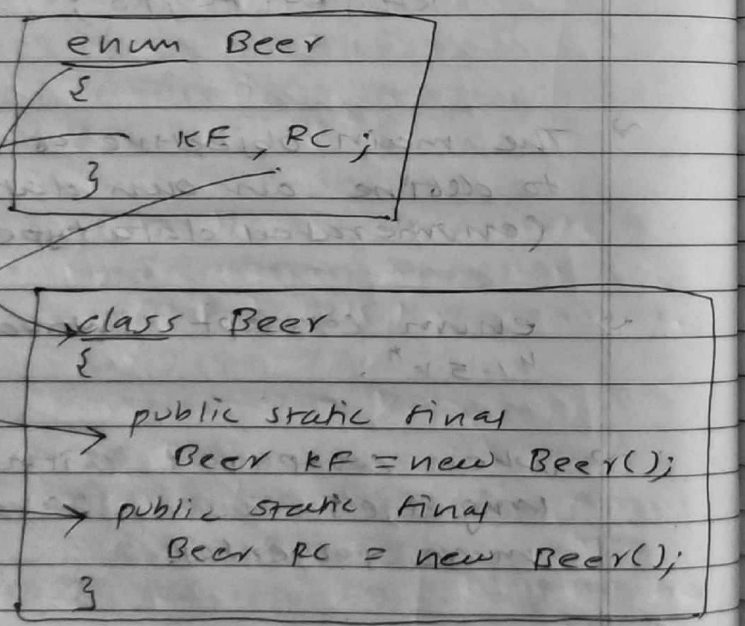
Internal implementation of enum

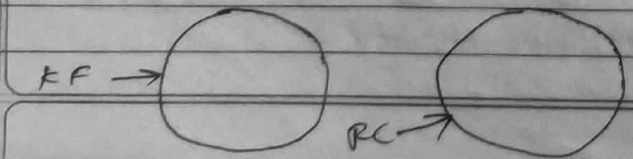
~ Every enum is internally implemented by using "class" concept only.

~ Every enum constant is always "public static final".

~ Every enum constant represents an object of the type enum.

~ Ex.





enum Declaration & Usage

~ Every enum constant is always "public static final" & hence we can access enum constant by using "enum name".

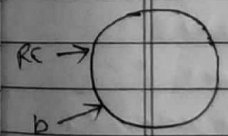
~ Ex.

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

class Test
{
    p s v main (Sn)
    {
        Beer b = Beer.RC;

        S.o.p(b); // RC

    }
}
```



~ Note :- Inside enum 'tostring()' meth.d is internally implemented to return name of the constant.

→ We can declare 'enum' either within a class or outside of a class, but not inside a method.

→ If we are trying to declare inside a method then we will get [C.E. enum types must not be local.]

→ enum X
{
}
class Y
{
}

class X
{
enum Y
{
}
}

class X
{
P S V M(C)
{
enum Y
{
}
}
}

[C.E. enum type must not be local]

→ If we declare enum outside of the class applicable modifiers are
public
default
strictfp

→ If we declare enum inside a class applicable modifiers are
public default strictfp + private protected static

→ enum outside the class, should be able to take modifiers like (final, abstract) just like class can take but enum are implicitly final so we can't declare final explicitly & as because enum are implicity final we can't declare ✱ abstract bcz final abstract is illegal combination.

✱ enum vs switch

→ Untill 1.4 V, the allowed argument types for the switch statement are
byte, short, char, int

~ But from 1.5 V onwards, corresponding wrapper classes & enum types are allowed.

~ From 1.7 V onwards, String types are also allowed.

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

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

~ Ex-

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

```
class Test
{
    p s v main(sa)
    {
        Beer b = Beer.RC;
        switch(b)
        {
            case KF:
                s.o.p("children's brand");
                break;
            case KO:
                s.o.p("Too light");
                break;
            case RC:
                s.o.p("Not much kick");
                break;
            case FO:
                s.o.p("Buy one get one free");
                break;
            default:
                s.o.p("Not Recomm.");
        }
    }
}
```

o/p :- Not much Pick

⇝ If we pass enum type as argument to switch statement then every case label should be valid enum constant, other wise we will get C.E.

[unqualified enumeration constant name required]

Ex:
```
switch(b)
{
    case FF:
    case RC:
    case KO:
    case FO:
    case KALYANI:
}
```

(*) enum (vs) Inheritance

⇝ Every enum is always direct child class of "j.l.Enum" & hence our enum can't extend any other enum (Bcz Java won't provide support for multiple inheritance)

⇝ Every enum is final implicitly & hence for our enum we can't create child enum.

⇝ Bcz of above reasons, we can conclude, Inheritance concept not applicable for enum & hence we can't use "extends" (explicitly) keyword for enum.

⇝ Ex.
```
enum X
{
}
enum Y extends X   (X)
{
}
```

⇝
```
enum x extends j.l.Enum
{
}
```
(X)

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

```
enum X
{
}
class Y extends X
{
}
```

C.E:1 cannot inherit from final X
C.E:2 enum types are not extensible

Anyway, An enum can implement any no. of interfaces.

```
interface X
{
}
enum Y implements X
{
}
```

java.lang.Enum

Every enum in java is the direct child class of "java.lang.Enum" & hence this class act as base class for all Java enums.

It is an abstract class & direct child class of Object.

It implements Serializable & Comparable interfaces.

values() method

Every enum implicitly contains values() method, to list out all values present in enum.

```
Beer[] b = Beer.values()
```

Note: values() method not present in java.lang.Enum & Object class but the twist here is that "enum" keyword provides implicitly this method

ordinal() method

Inside enum, order of constants is important & we can represent this order via "ordinal value"

we can find the "ordinal value" of

enum constant from "ordinal()" method.

```
public final int ordinal()
```

→ "ordinal value" is 0-based like Array index.

→ Ex.

```
enum Beer
{
    KF, KO, RC, FO;
}
class Test
{
    p s v main(SM)
    {
        Beer[] b = Beer.values();
        for(Beer bj : b)
        {
            s.o.p(" bj + "---" bj.ordinal());
        }
    }
}
```

O/P :- KF --- 0
KO --- 1
RC --- 2
FO --- 3

Speciality of java enum

→ In old language's enum, we can take only constants but in java enum we can take methods, constructors, normal variables in addition to constants, hence java-enum is more powerful than old language's enum.

→ Even in java-enum, we can take main() method & we can run enum class directly from cmd.

→ Ex.

```
enum Fish
{
    STAR, GUPPY, GOLD;
    p s v main(SM)
    {
        s.o.p("ENUM MAIN METHOD");
    }
}
```

javac Fish.java ↵
java Fish

O/P :- ENUM MAIN METHOD

⇝ Note :- In addition to constants, if we are taking any extra member like main() then list of constants compulsory should be at the first line & should be end with semi-colon.

Ex.
```
enum Fish
{
    STAR, GUPPY(;) ⟶ mandatory
    public void m1()
    {
    }
}
```

- ```
  enum Fish
  {
      STAR, GUPPY
      public void m1()
      {
      }
  }
  ```

- ```
  enum Fish
  {
      P void m1()
      {
      }
      STAR, GUPPY();
  }
  ```

⇝ Inside enum if we are taking any extra member like method, compulsory the first line should contain list of constants & atleast semicolon.

- ```
  enum Fish
  {
      P v m1()
      {
      }
  }
  ```

- ```
  enum Fish
  {
      ;
      P v m1()
      {
      }
  }
  ```

⇝ Anyway an empty enum is valid java syntax

```
enum Fish
{
}
```

```
enum Fish
{
    ;
}
```

✳ enum vs Constructor

→ An enum can contain constructor

→ enum constructor will be executed separately for every enum constant at the time of enum class-loading automatically.

→ Ex- enum Beer
{
KF, KO, RC, FO;
Beer()
{
S.O.P("constructor");
}
}
class Test
{
p s v main(Str)
{
Beer b = Beer.RC; ①
S.O.P("Hello");
}
}

→ javac Test.java

Beer.class Test.class

java Test

```
O/P :- constructor
       constructor
       constructor
       constructor
       constructor
       Hello
```

→ If we comment line-① , The O/P will be :-

```
Hello
```

→ we can't create enum object directly & hence we can't invoke enum constructor directly.

```
Beer b = new Beer();   (X)
```

[C.E. enum type may not be instantiated.]

→ Ex.

```
enum Beer
{
    KF(70), KO(80), RC(90), FO;

    int price;
    Beer(int price)
    {
        this.price = price;
    }
    Beer()
    {
        this.price = 65;
    }
    public int getPrice()
    {
        return price;
    }
}
class Test
{
    p s v main(Str)
    {
        Beer[] b = Beer.values();

        for (Beer bl : b)
        {
            S.O.p(bl + "---" +
                  bl.getPrice());
        }
    }
}
```

O/P :-

```
KF --- 70
KO --- 80
RC --- 90
FO --- 65
```

Note :-

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

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

↝ Inside enum we can declare method but should be concrete methods only & we can't declare abstract method ..

Reason 1 :- one abstract method means enum should be abstract but as it is final already final abstract combination will be illegal.

Reason 2 :- As enum is final, we can't create child for it, so no one can provide implementation for abstract method in that enum

↝ **Case-1** :- Every enum constant represents an object of type enum hence, whatever methods we can apply on normal java object can be applicable on enum constants also.

Ex-

Beer.KF.equals(Beer.RC) ✓

Beer.KF.hashCode() > Beer.RC.hashCode() ✓

Beer.KF < Beer.RC ✗

Beer.kf.ordinal < Beer.RC.ordinal() ✓

↝ **Case-2** :- If we want to use any class / Interface directly from outside package then the required import is normal import.

↝ If we want to use/access static members without classname then the required import is static import.

Ex.

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

```
import java.util.AL;

class Test
{
    p s v main(Sn)
    {
        AL al = new AL();
        s.o.p(sqrt(4));
    }
}
```

→ Importance of this topic related to enum:-

- package pack1;
  public ~~class~~ Enum Fish
  {
  STAR, GUPPY;
  }

- package pack2;
  public class Test
  {
  p s v main(Sn)
  {
  Fish f = Fish.GUPPY;
  s.o.p(f);
  }
  }

The required import is:-

import pack1.Fish / import pack1.*;

- package pack3;
  public class Test
  {
  p s v main(Sn)
  {
  s.o.p(STAR);
  }
  }

The required import is:-

import static pack1.Fish.STAR;
(or)
import static pack1.Fish.*;

- package pack4;
  public class Test
  {
  p s v main(Sn)
  {
  Fish f = Fish.STAR;
  s.o.p(GUPPY);
  }
  }

The required import is:-

import pack1.Fish;
(or)
import pack1.*;

{ import static pack1.Fish.GUPPY;
import static pack1.Fish.*;

→ **case 3** :-

```
enum Color
{
    BLUE, RED, GREEN;

    public void info()
    {
        S.O.P("Universal color");
    }
}
```

```
enum Color
{
    BLUE, RED
        {
            public void info()
            {
                S.O.P("Dangerous
                        color");
            }
        }, GREEN;

    public void info()
    {
        S.O.P("Universal color");
    }
}
```

```
class Test
{
    P S V main(SA)
    {
        Color[] C = Color.values();

        for(Color c1 : C)
        {
            S.O.P(c1.info());
        }
    }
}
```

O/P :- It 1st enum color considered

Universal color
Universal color
Universal color

- It 2nd enum color considered

Universal Color
Dangerous color
Universal color

→ This is nothing but just a Anonymous Inner class syntax in which we are creating anonymous child class & overriding original behaviour.

Date
Page

(*) enum (vs) Enum (vs) Enumeration

⇒ enum :-

- Keyword in java which can be used to define a group of named constants

⇒ Enum :-

- Enum is a class in java, present in "java.lang" package.
- Every enum in java should be direct child class of java.lang.Enum, hence this class acts as Base class for all enums in java

⇒ Enumeration :-

- Enumeration is an Interface present in "java.util" package.
- We can use Enumeration object to get objects one by one from the collection.

End of Enum

Start of Internationalization