***enum(enumeration)***

When we want to go for group of named constants then we can go for enum. Almost every programming language contains enum concept.

You can define your own data type. **For eg**: byte is a predefined data type.

**enum** Month{  
 ***JANUARY***, ***FEBRUARY***….;  
}

Here, Month is our own data type(Main objective of enum is to define our own data-types)

* If you want to represetnt a group of named constants then we should go for enum.
* It was introduced in 1.5v. When compared to old lang enum, Java enum is more powerful as we can declare methods, variables etc also.

**Internal implementation of enum.**

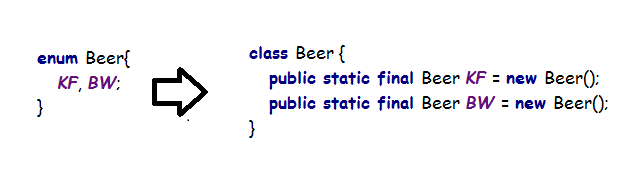
Internally enum is also converted into a java class.

**Eg:**

**enum** Beer{  
 ***KF***, ***BW***;  
}

Here, KF and BW are the types of Beer Object. Every enum constants represents an Object of the type enum. Hence, for Beer two Objects are present.

Internally what happens is defined below:



As we can see, every constants on enum is nothing but an object of enum itself.

**enum** Beer{  
 ***KF***, ***BW***;  
}  
  
**class** Test{  
 **public static void** main(String[] args) {  
 Beer beer = Beer.***KF***;  
 System.***out***.println(beer); **// KF**  
 }  
}

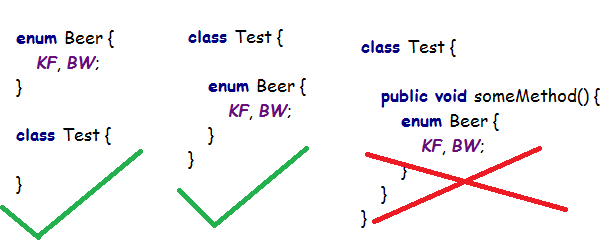
Here, KF is printed as internally enum uses toString() method that will print the name of enum constant. Every enum constant is **public static final** and hence we can access enum constant by using enum name.

Q. Can we declare enum inside a class??

* Yes, it is possible

Q. Can we declare enum inside a method?

* No, we will get compile time error.



**NOTE: Every enum is final implicitly**

If we are defining enum **outside** the class then these are the modifiers that are allowed:

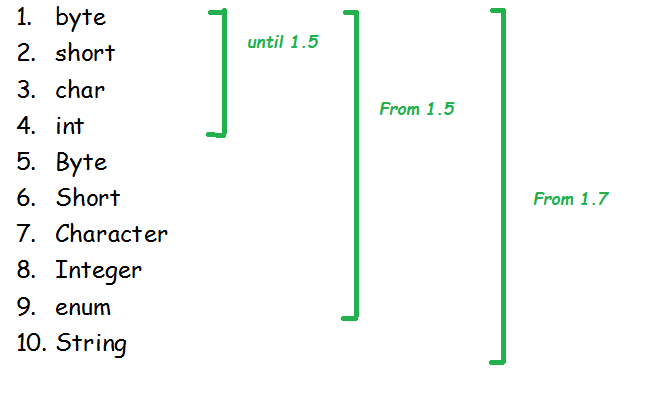
1. public
2. default
3. strictfp

If we are defining enum **inside** the class then these are the modifiers that are allowed:

1. private
2. protected
3. static

**enum vs switch:**

Arguments that are allowed as an argument in switch.

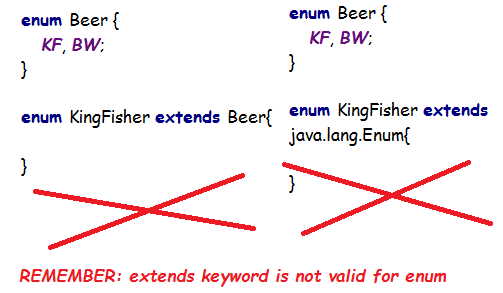


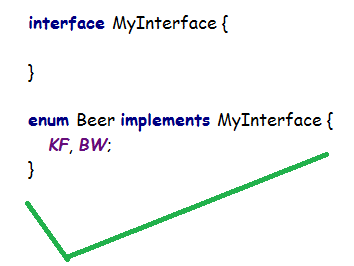
**enum** Beer {  
 ***KF***, ***BW***;  
}  
  
**class** Test {  
 **public static void** main(String[] args) {  
  
 Beer beer = Beer.***KF***;  
 **switch** (beer) {  
 **case *KF***:  
 System.***out***.println(**"KF is not good"**);  
 **break**;  
 **case *BW***:  
 System.***out***.println(**"BW is better"**);  
 **break**;  
 **default**:  
 System.***out***.println(**"Which beer do you want?"**);  
 }  
 }  
}

***enum VS inheritance***

Every enum in java is a direct child class of Enum class.

* As enum is directly extending Enum class so our enum class cannot extends any other other class as multiple inheritance is not possible in java. As Enum class is a direct parent to our enum class, hence it is not multilevel also.
* As our enum is implicitly final, hence we cannot create its child class as well.
* Hence, for enum inheritance concept is not valid for enum.
* extends keyword is not applicable to our enum.





**Enum class**

1. Every enum that we create is a direct child class of predefined **Enum class**.
2. It is an abstract class and direct child class of Object class
3. It implements Serializable and Comparable interfaces.

**values() method**

**enum** Beer {  
 ***KF***, ***BW***;  
}  
  
**class** Test {  
 **public static void** main(String[] args) {  
 Beer[] beers = Beer.*values*();  
 **for** (Beer beer : beers) {  
 System.***out***.println(beer); *// KF BW* }  
 }  
}

**Q. Where exactly is values() method??**

* As enum is a direct child of Enum, we can look the method in Enum class, But it is not present.

Methods in Enum class:

1. name
2. finalize
3. equals
4. toString
5. hashCode
6. clone
7. compareTo
8. compareTo
9. valueOf
10. getDeclaringClass
11. readObject
12. ordinal
13. readObjectNoData

Further we can look into Object class for the value() method as Enum is direct child class od Object clas, But it still is not present.

Methods in Object class:

1. finalize
2. wait
3. wait
4. wait
5. equals
6. toString
7. hashCode
8. getClass
9. clone
10. notify
11. notifyAll
12. registerNatives

**So, where is the values() method?**

You cannot see values() method inside Enum as well as Object method. Every enum keyword implicitly contains values() method.

**ordinal() method**

It is used to find the index of enum constant.

**enum** Beer {  
 ***KF***, ***BW***;  
}  
**class** Test {  
 **public static void** main(String[] args) {  
 Beer[] beers = Beer.*values*();  
 **for** (Beer beer : beers) {  
 System.***out***.println(**"value is: "** + beer + **" ordinal value(index): "** + beer.ordinal()); }  
 }  
}

***Output:***

value is: KF ordinal value(index): 0

value is: BW ordinal value(index): 1

**Speciality of Java enum**

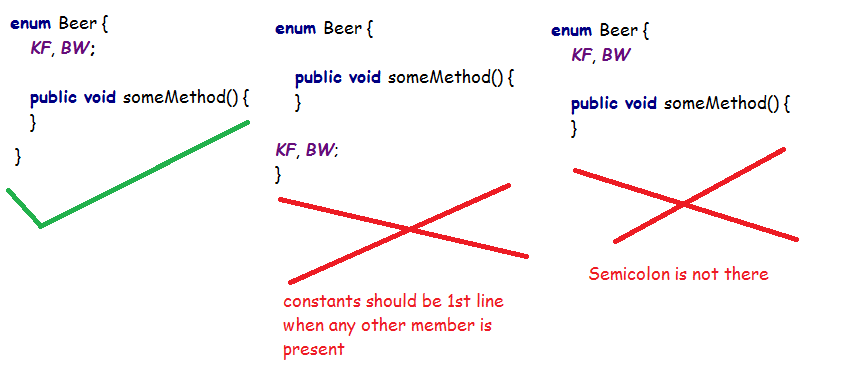
In old languages enum we could take only constants but in java in addition to constants we can take normal variables, methods, constructors etc.

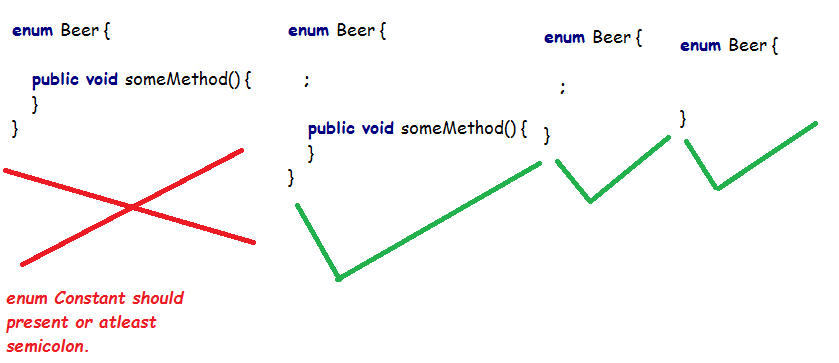
**Q. Can we take main method inside enum?**

* Yes, and also we can run it directly
* **enum** Beer {  
   ***KF***, ***BW*;**  
    
   **public static void** main(String[] args) {  
   System.***out***.println(**"This is my enum class"**);  
   }  
  }

**Note: Semicolon is not optional when you are declaring methods or variables in enum.**

**Note: list of Enum constants should be first line and ends with semicolon when you are declaring methods or variables in enum.**

****

****

***Enum constructor:***

An enum can contain constructor.

Enum constructor will be executed for every enum constant at the time of class loading (.class file generation) seperatly.

**enum** Beer {  
 ***KF***, ***BW***;  
  
 Beer() {  
 System.***out***.println(**"Constructor"**);  
 }  
}  
  
**class** Test {  
 **public static void** main(String[] args) {  
  
 Beer beer = Beer.***KF***;  
 System.***out***.println(**"HERE"**);  
 }  
}

***Output:***

Constructor

Constructor

HERE

**Q. Can we initialize Enum i.e, Beer beer= new Beer()?**

* No, we cannot. As at the time of class loading every enum constants is instantiated, so it will give compile time errors saying enum may not be instantiated.

**Q. So, what is the way to pass an argument at time of class loading of an enum?**

**enum** Beer {  
 ***KF***(40),

***BW***(50),

***KR***;  
  
 **private int price**;  
  
 Beer() {  
 **this**.**price** = 65;  
 }  
  
 Beer(**int** price) {  
 **this**.**price** = price;  
 }  
  
 **public void** getPrice(Beer beer) {  
 System.***out***.println(**"Price of "** + beer + **" is: "** + **this**.**price**);  
 }  
}  
  
**class** Test {  
 **public static void** main(String[] args) {  
  
 Beer kingfisher = Beer.***KF***;  
 kingfisher.getPrice(kingfisher);  
  
 Beer budwiser = Beer.***BW***;  
 budwiser.getPrice(budwiser);  
  
 Beer korona = Beer.***KR***;  
 korona.getPrice(korona);  
 }  
}

Output:

Price of KF is: 40

Price of BW is: 50

Price of KR is: 65

**Here,**

***KF***(40) = **public static final** Beer ***KF*** = **new** Beer(40);

***BW***(50) = **public static final** Beer ***BW*** = **new** Beer(40);

***KR =* public static final** Beer ***KR*** = **new** Beer();

* We can not declare abstract method inside enum as we can not create child of any enum.

**Valid examples:**

**enum** Beer {

***KF***, ***BW***, ***KR*** {  
 @Override  
 **public void** info(Beer beer) {  
 System.***out***.println(**"Best beer in the market"**);  
 }  
 };  
  
 **public void** info(Beer beer) {  
 System.***out***.println(beer + **"is Normal beers"**);  
 }  
}  
  
**class** Test {  
 **public static void** main(String[] args) {  
  
 Beer[] beers = Beer.*values*();  
 **for** (Beer beer : beers) {  
 beer.info(beer);  
 }  
 }  
}

Normal beers

Normal beers

Best beer in the market

***enum VS Enum VS Enumeration***

enum is a keyword used to define group of enum constants.

Enum is a base class for all the enums we create.

Enumeration is an interface used in Collections.