[**Java Enum**](http://javarevisited.blogspot.in/2011/08/enum-in-java-example-tutorial.html)

**Enumeration (Enum) in Java** was introduced in JDK 1.5 and it is one of my favorite features of J2SE 5. Java Enum as type is more suitable on certain cases for example representing state of Order as NEW, PARTIAL FILL, FILL or CLOSED. Enumeration (enum) was not originally available in Java though it was available in other language like C and C++ but eventually Java realized and introduced Enum on JDK 5 (Tiger) by **keyword enum**. 

### How to represent enumerable value without Java enum

Since **Enum in Java** is only available from **Java 1.5** its worth to discuss how we used to represent enumerable values in Java prior JDK 1.5 and without it. I use public static final constant to replicate enum like behavior. Let’s see an enum example in java to understand the concept better. In this example we will use US Currency Coin as enumerable which has values like PENNY (1) NICKLE (5), DIME (10), and QUARTER (25).

**class** CurrencyDenom {

**public** **static** **final** **int** *PENNY* = 1;

**public** **static** **final** **int** *NICKLE* = 5;

**public** **static** **final** **int** *DIME* = 10;

**public** **static** **final** **int** *QUARTER* = 25;

      }

**class** Currency {

**int** currency; //CurrencyDenom.PENNY,CurrencyDenom.NICKLE,

                 // CurrencyDenom.DIME,CurrencyDenom.QUARTER

}  
  
 Though this can server our purpose it has some serious limitations:  
  
**1) No Type-Safety**: First of all it’s not type-safe; you can assign any valid int value to currency e.g. 99 though there is no coin to represent that value.  
  
**2) No Meaningful Printing**: printing value of any of these constant will print its numeric value instead of meaningful name of coin e.g. when you print NICKLE it will print "5" instead of "NICKLE"  
  
**3) No namespace:** to access the currencyDenom constat we need to prefix classname e.g. CurrencyDenom.PENNY instead of just using PENNY though this    can also be achieved by using static import in JDK 1.5  
  
**Java Enum** is answer of all this limitation. Enum in Java is type-safe, provides meaningful string names and has there own namespace. Now let's see same example using Enum in Java:

**public** **enum** Currency {PENNY, NICKLE, DIME, QUARTER};

Here Currency is our **enum** and PENNY, NICKLE, DIME, QUARTER are **enum constants**. Notice **curly braces around enum constants** because enums are type like class and interface in Java. Also we have followed similar naming convention for enum like class and interface (first letter in Caps) and since *Enum constants are implicitly static final* we have used all caps to specify them like Constants in Java.

### What is Enum in Java

Now back to primary questions **“What is enum in java”** simple *answer enum is a keyword in java* and on more detail term java enum is type like class and interface and can be used to define a set of enum constants. Enum constants are implicitly static and final and you cannot change their value once created. Enum in Java provides type-safety and can be used inside switch statement like int variables. Since enum is a keyword you cannot use as variable name and since it’s only introduced in JDK 1.5 all your previous code which has enum as variable name will not work and needs to be re-factored.

### Benefits of Enum in Java:

1) **Enum is type-safe** you cannot assign anything else other than predefined enum constant to an **enum variable**.  
2) Enum has its own name-space.  
3) Best feature of Enum is **you can use Enum in Java inside Switch statement** like int or char primitive data type.

We will also see example of using java enum in switch statement in this java enum tutorial.  
4) Adding new constants on Enum in Java is easy and you can add new constants without breaking existing code.

## Important points about Enum in Java

1) **Enum in Java is type-safe** and has their own name-space. It means your enum will have a type for example "Currency" in below example and you can not assign any value other than specified in Enum Constants.

**public** **enum** Currency {*PENNY*, *NICKLE*, *DIME*, *QUARTER*};

Currency coin = Currency. PENNY;

coin = 1; //compilation error    
  
2**) Enum in Java are reference type like class or interface** and you can define constructor, methods and variables inside java enum which makes it more powerful than Enum in C and C++ as shown in next example of java enum type.  
  
3) You can **specify values of enum constants at the creation time** as shown in below example:  
**public** **enum** Currency {*PENNY*(1), *NICKLE*(5), *DIME*(10), *QUARTER*(25)};  
But for this to work you need to define a member variable and a constructor because PENNY (1) is actually calling a constructor which accepts int value , see below example.

**public** **enum** Currency {

*PENNY*(1), *NICKLE*(5), *DIME*(10), *QUARTER*(25);

**private** **int** value;

**private** Currency(**int** value) {

**this**.value = value;

        }

};     
**Constructor of enum in java** must be **private** any other access modifier will result in compilation error. Now to get the value associated with each coin you can define a public getValue () method inside java enum like any normal java class. Also semi colon in the first line is optional.  
  
4) Enum constants are implicitly static and final and cannot be changed once created. For example below code of java enum will result in compilation error:

Currency.PENNY = Currency.DIME;

The final field EnumExamples.Currency.PENNY cannot be re assigned.  
    
5) **Enum in java can be used as an argument on switch statment** and with "case:" like int or char primitive type. This feature of java enum makes them very useful for switch operations. Let’s see an example of how to use java enum inside switch statement:

   Currency usCoin = Currency.DIME;

**switch** (usCoin) {

**case** PENNY:

                    System.out.println("Penny coin");

**break**;

**case** NICKLE:

                    System.out.println("Nickle coin");

**break**;

**case** DIME:

                    System.out.println("Dime coin");

**break**;

**case** QUARTER:

                    System.out.println("Quarter coin");

    }  
    
6) Since **constants defined inside enum in java are final you can safely compare them using "==" equality operator** as shown in following example of   java enum:

Currency usCoin = Currency.DIME;

**if**(usCoin == Currency.DIME){

       System.*out*.println("enum in java can be"+

               "compared using ==");

    }

7) Java compiler automatically generates static values () method for every enum in java.Values() method returns array of enum constants in the same order they have listed in enum and you can use values() to iterator over values of enum in java as shown in below example:

**for**(Currency coin: Currency.values()){

        System.*out*.println("coin: " + coin);

    }  
And it will print:

**coin: PENNY**

**coin: NICKLE**

**coin: DIME**

**coin: QUARTER**  
                  
Notice the order its exactly same **with defined order in enum**.  
    
8) In Java Enum can override methods also. Let’s see an example of overriding toString () method **inside enum in java** to provide **meaningful description** for enum constants.

**public** **enum** Currency {

  ........

  @Override

**public** String toString() {

**switch** (**this**) {

**case** PENNY:

              System.out.println("Penny: " + value);

**break**;

**case** NICKLE:

              System.out.println("Nickle: " + value);

**break**;

**case** DIME:

              System.out.println("Dime: " + value);

**break**;

**case** QUARTER:

              System.out.println("Quarter: " + value);

        }

**return** **super**.toString();

 }

};          
And here is how it looks like when displayed:

Currency usCoin = Currency.*DIME*;

System.out.println(usCoin);

**output:**

**Dime: 10**  
        
9) Two new collection classes **EnumMap and EnumSet** are added into collection package to **support java enum**. These classes are high performance implementation of Map and Set interface in Java and we should use this whenever there is any opportunity.  
  
10**) You cannot create instance of enum by using new operator** in java because constructor of Enum in Java can only be private and Enum constants can only be created inside Enum itself.  
  
11) Instance of enum in java is created when any enum constants are first called or referenced in code.  
  
12) **Enum in Java can implement the interface** and override any method like normal class It’s also worth noting that enum in java implicitly implement both Serializable and Comparable interface. Let's see an example of **how to implement interface using java enum**:

**public** **enum** Currency **implements** Runnable{

  PENNY(1), NICKLE(5), DIME(10), QUARTER(25);

**private** **int** value;

  ............

  @Override

**public** **void** run() {

  System.out.println("Enum in Java implement interfaces");

   }

}  
  
13) **You can define abstract methods inside enum in Java** and can also provide different implementation for different instances of enum in java.  Let’s see an *example of using abstract method inside enum in java*

**public** **enum** Currency **implements** Runnable{

          PENNY(1) {

                  @Override

**public** String color() {

**return** "copper";

                  }

          }, NICKLE(5) {

                  @Override

**public** String color() {

**return** "bronze";

                  }

          }, DIME(10) {

                  @Override

**public** String color() {

**return** "silver";

                  }

          }, QUARTER(25) {

                  @Override

**public** String color() {

**return** "silver";

                  }

          };

**private** **int** value;

**public** **abstract** String color();

**private** Currency(**int** value) {

**this**.value = value;

          }

          ..............

  }         
In this example since every coin will have different color we made the color () method abstract and let each instance of enum to define their own color. You can get color of any coin by just calling color () method as shown in below example of java enum:

System.out.println("Color: " + Currency.DIME.color());

 **Enum java valueOf example**  
One of my reader pointed out that I have not mention about valueOf method of enum in Java, which is used to convert String to enum in java.  Here is what he has suggested, thanks @ Anonymous  
“You could also include **valueOf() method of enum** in java which is added by compiler in any enum along with values() method. **Enum valueOf()** is a static method which takes a string argument and can be used to convert a String into enum. One think though you would like to keep in mind is that valueOf(String) method of enum will throw "**Exception in thread "main" java.lang.IllegalArgumentException: No enum const class**" if you supply any string other than enum values.  
  
Another of my reader suggested about ordenal() and name() utility method of java enum **Ordinal method of java enum returns position of a enum constant** as they declared in enum **while name()of enum returns the exact string** which is used to create that particular enum constant.”