ince **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](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) 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).

**public 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;

}

**public class** Currency {

**private 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](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html); 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 constant we need to prefix class name e.g. CurrencyDenom.PENNY instead of just using PENNY though this can also be achieved by using [static import in JDK 1.5](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html)  
  
**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 Enum are type like [class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) and [interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html). 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](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) and you can not change there value once created. Enum in Java provides type-safety and can be used inside switch statment like int variables. Since enum is a keyword you can not use as variable name and since its 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 Enums in Java:**

1) **Enum is type-safe** you can not assign anything else other than predefined Enum constants to an Enum variable. It is compiler error to assign something else unlike the public static final variables used in Enum int pattern and Enum String pattern.  
  
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) **Enums in Java are type-safe** and has there 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](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) or [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html)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](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) 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**](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html) 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](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html) and [final](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html) and can not 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");

    }  
    
from JDK 7 onwards you can also [String in Switch case in Java](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html) code.  
  
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 ==");

}

By the way comparing objects using == operator is not recommended, Always use [equals() method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) or [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) to compare Objects.

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 [iterate](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) 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 enums**.  
  
  
    
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 enums 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](http://javarevisited.blogspot.sg/2012/07/create-read-only-list-map-set-example-java.html) and we should use this whenever there is any opportunity.  
  
  
  
10**) You can not create instance of enums by using new operator** in Java because constructor of Enum in Java can only be private and Enums constants can only be created inside Enums 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](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html) and [Comparable](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) interface. Let's see and 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*](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html) *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   there 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.

enum Car {

lamborghini(900),tata(2),audi(50),fiat(15),honda(12);

private int price;

Car(int p) {

price = p;

}

int getPrice() {

return price;

}

}

public class Main {

public static void main(String args[]){

System.out.println("All car prices:");

for (Car c : Car.values())

System.out.println(c + " costs "

+ c.getPrice() + " thousand dollars.");

}

}

public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY

}

You should use enum types any time you need to represent a fixed set of constants. That includes natural enum types such as the planets in our solar system and data sets where you know all possible values at compile time—for example, the choices on a menu, command line flags, and so on.

Here is some code that shows you how to use the Day enum defined above:

public class EnumTest {

Day day;

public EnumTest(Day day) {

this.day = day;

}

public void tellItLikeItIs() {

switch (day) {

case MONDAY:

System.out.println("Mondays are bad.");

break;

case FRIDAY:

System.out.println("Fridays are better.");

break;

case SATURDAY: case SUNDAY:

System.out.println("Weekends are best.");

break;

default:

System.out.println("Midweek days are so-so.");

break;

}

}

public static void main(String[] args) {

EnumTest firstDay = new EnumTest(Day.MONDAY);

firstDay.tellItLikeItIs();

EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);

thirdDay.tellItLikeItIs();

EnumTest fifthDay = new EnumTest(Day.FRIDAY);

fifthDay.tellItLikeItIs();

EnumTest sixthDay = new EnumTest(Day.SATURDAY);

sixthDay.tellItLikeItIs();

EnumTest seventhDay = new EnumTest(Day.SUNDAY);

seventhDay.tellItLikeItIs();

}

}

**The java.lang.Enum.valueOf() method returns the enum constant of the specified enumtype with the specified name. The name must match exactly an identifier used to declare an enum constant in this type.**

import java.lang.\*;

// enum showing Mobile prices

enum Mobile {

Samsung(400), Nokia(250),Motorola(325);

int price;

Mobile(int p) {

price = p;

}

int showPrice() {

return price;

}

}

public class EnumDemo {

public static void main(String args[]) {

System.out.println("CellPhone List:");

for(Mobile m : Mobile.values()) {

System.out.println(m + " costs " + m.showPrice() + " dollars");

}

Mobile ret;

ret = Mobile.valueOf("Samsung");

System.out.println("Selected : " + ret);

}

}