SATURDAY, AUGUST 13, 2011

Java Enum Tutorial: 10 Examples of Enum in Java

What is Enum in Java

Enum in Java is a keyword, a feature which is used to represent fixed number of well known values in Java, For example Number of days in Week, Number of planets in Solar system etc. **Enumeration (Enum) in Java** was introduced in JDK 1.5 and it is one of my favorite features of J2SE 5 among Autoboxing and unboxing , Generics, varargs and static import. One of the common use of Enum which emerged in recent years is Using Enum to write Singleton in Java, which is by far easiest way to implement Singleton and handles several issues related to thread-safety and Serialization automatically. By the way, Java Enum as type is more suitable to represent well known fixed set of things and state, 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**. In this **Java Enum tutorial** we will see different *Enum example in Java* and learn using Enum in Java. Focus of this Java Enum tutorial will be on different features provided by Enum in Java and how to use them. If you have used Enumeration before in C or C++ than you will not be uncomfortable with Java Enum but in my opinion Enum in Java is more rich and versatile than in any other language. . By the way, if you like to learn new concepts using book than you can also see Java 5.0 Tiger : A Developers notebook, I had followed this book while learning Enum, when Java 1.5 was first launched. This book has excellent chapter not only on Enum but also on key features of Java 1.5 and worth reading.

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).

```
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; 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

`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 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 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 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);
```

► June (9)
► May (6)
► April (10)
► March (4)
► February (10)
► January (10)
► 2010 (33)

References

Java API documentation JDK 6
Spring framework doc
Struts
JDK 7 API
MySQL
Linux
Eclipse
jQuery

```
        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 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 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 or compareTo() method 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 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 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 and Comparable 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 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.

Another of my reader suggested about `ordinal()` 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." name() method can also be used for converting Enum to String in Java.

That's all on Java enum , Please share if you have any nice tips on enum in Java and let us know how you are using java enum in your work. You can also follow some good advice for using Enum by Joshua Bloch in his all time classic book Effective Java. Those advice will give you more idea of using this powerful feature of Java programming language

Further Reading on Java Enum

If you like to learn more about this cool feature, I suggest reading following books. Books are one of the best resource to completely understand any topic and I personally follow them as well. Enumeration types chapter from Thinking in Java is particularly useful.

Thinking in Java (4th Edition) By Bruce Eckel

Effective Java by Joshua Bloch

Java 5.0 Tiger: A Developers notebook

Java 7 Recipes

Last book is suggested by one of our reader @Anonymous, you can see his comment

> Check out the book, Java 7 Recipes. Chapter 4 contains some good content on Java enums. They really go into depth and the examples are excellent.

Some **Java Tutorials** you May Like

Generics in Java Explained with Example

How to use Variable Argument List in Java

How to Solve Java.lang.OutOfMemoryError: Java Heap Space

How to Solve UnSupportedClassVersionError in Java

Example of Polymorphism in Java

What is abstraction in Java with Example

Posted by Javin Paul at 7:47 PM

Labels: core java , java 5 tutorial
Location: North America

53 comments :

Jirka Pinkas said...

Nice! I use enum regulary, but I didn't know it is so versatile. Thank you very much for this complete summary.

One more thing - beware overuse of enum. Right now I'm refactoring a project, where programmer used enum for defining colors used in application (including their names), but now management wants to sell this application abroad and now I have to remove this enum and put these information into database.

August 13, 2011 at 10:37 PM

Anonymous said...

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.

August 30, 2011 at 10:40 PM

Anonymous said...

dude how can you miss ordenal() and name() method of enum in java. Ordinal method of java enumo 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.

August 30, 2011 at 11:04 PM

Anonymous said...

Can you write about reverse lookup using Enum in Java ? I am not able to understand that concept which is an important usage of java enum. thanks

October 5, 2011 at 2:08 AM

R. RamaKashyap said...

One of the most Complete tutorial I have read on Java Enum, didn't know that Enum in Java is this much versatile and we can do all this stuff with enum. I am going to recommend this article to all my students for Java Enum, I also want to distribute printed copy of this Java enum tutorial, let me know if its ok to you. Thanks.

October 7, 2011 at 3:13 AM

Anonymous said...

Thank you for this tutorial, i've found it very helpful!

October 19, 2011 at 12:38 PM

Ram said...

is there any difference in Java Enum in Java5 and Java6 ? is functionality of Enum in Java has enhanced in JDK 1.6 ?We are going to use Java6 in our application and would like to know if there is any significant change on Enum in Java6. Thanks

November 16, 2011 at 5:26 PM

Javin @ String split Java example said...

@Ram, As such I don't see any difference on Enum from Java5 to Java6. and you can safely use them on Java6.

November 18, 2011 at 5:48 PM

Gyananedra said...

Hi Can you please post example of Converting String to Enum in Java and opposite i.e. How to convert Enum to String in Java. we have an application where we need to convert from enum to string and vice-versa and I am looking for quick way of converting that. thanks

November 28, 2011 at 9:08 PM

Taner said...

I am trying to use Enum in Switch case but getting this error **"an enum switch case label must be the unqualified name of an enumeration constant"**, Looks like Enum in Switch are not allowed or only allowed with some restriction, Can you please help.about Code its simple WEEKDAY Enum with switch on every day.

December 1, 2011 at 5:36 PM

Anonymous said...

@Taner: It's just a guess (since you didn't provide any details), but apparently you're using qualified names in the case label.
i.e.
case Weekday.MONDAY:
instead of
case MONDAY:

i think you can't mix enums. and it would be very messy if you're always prepending the enum name

December 2, 2011 at 3:16 PM

Anonymous said...

Nice one

December 28, 2011 at 4:37 AM

Anonymous said...

Please provide EnumMap and EnumSet

January 19, 2012 at 11:24 PM

Anonymous said...

Best Tutorial on Enum in Java I have read. Indeed extensive and shows how differently one can use Enum in Java you could have also included EnumMap and EnumSet which are specifically optimized for Enums and much faster than there counterparts.

January 29, 2012 at 9:20 PM

Anonymous said...

Can we declare two constructors for enum. If we can how can we access that?

February 7, 2012 at 10:39 PM

Mansi said...

Another useful example of Enum in Java is using Enum for writing Thread-safe Singleton. its pretty easy and handles all thread-safety concern inherently:

public enum SINGLETON{
INSTANCE;
}

February 12, 2012 at 7:26 PM

Anonymous said...

hi this was more useful, i have some problems with my java program, i will post it late if any one could help thank you..

March 27, 2012 at 7:08 AM

Sushil said...

Excellent tutorial and example on Enum in Java.most extensive and useful coverage I have seen on Java 5 Enum.I would add on this on advantages of Java Enum. Though I see you have already described some benefits of Enum, mentioning some more advantages of Enum will certainly help:

1) Enum in Java is Type or you can say a Class.
2) Enum can implement interfaces.
3) You can compare two Enum without worrying about comparing orange with apples.
4) You can iterate over set of Enums.

In short flexibility and power is biggest advantage of Java Enum.

April 2, 2012 at 7:59 PM

Gesu said...

I have few interview questions related to Enum in Java, I hope you all help me to find answers:

Can one Enum extends another Enum in Java? or Can Enum extends another Class in Java ?

Can Enum implement interfaces in Java? if yes how many ?

April 10, 2012 at 3:34 AM

Javin @ spring interview questions said...

@Dinesh, Glad to hear that you like this Java enum tutorial. Enum is more useful than just for storing enumeration values and this can be used in variety of Java programs.

April 25, 2012 at 6:37 AM

Satya said...

Constructor of enum in java must be private

i think it works with default also

June 14, 2012 at 11:29 AM

Anonymous said...

Check out the book, Java 7 Recipes. Chapter 4 contains some good content on Java enums. They really go into depth and the examples are excellent.

July 10, 2012 at 1:44 AM

Robin said...

I was looking for an example of How to use Enum in Switch Case, when I found this tutorial. this shows many ways we can use Enum in Java, never thought of iterating all Enums in a for loop, Enum extending interface, enum overriding methods. I was only thinking enum in terms of fixed number of well know values but Java enum is way beyond that. Its full featured abstraction like Class. Now after reading your article my question why can't we use Enum in place of Class ? I expect we may not able to use Enum in every place but there must be certain cases where Enum in Java is more appropriate to Class.

July 30, 2012 at 7:59 PM

Anonymous said...

Some things that are good to remember about enumMaps are:

EnumMap does not allow null value for keys.If you try to put null as key in an EnumMap you will get an NullPointerException
If you want to declare an enumMap with generics note that you should declare your key as >
EnumMap's implementation uses an array and for that reason has slightly better performance than HashMaps

August 5, 2012 at 11:33 AM

Anonymous said...

One of the better example of Enum in Java API is java.util.concurrent.TimeUnit class which encapsulate Time conversion and some utility method related to time e.g. sleep. TimeUnit is an Enum it also implement several methods.

September 23, 2012 at 6:57 PM

Puskin said...

How to use Enum in Switch without using values method ? I want to switch on Enum where case can be individual Enum instances e.g.

switch(Button){
case Button.ON :
break;
case Button.OFF:
break;
}
Does this is a legal example of using Enum in Switch and case statement ?

September 26, 2012 at 6:26 PM

Tutorials said...

I guess below statement is wrong,

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.

Because values has been taken with 'final'.so, how can we change once it is created ?

October 5, 2012 at 5:50 AM

Javin @ CyclicBarrier Example Java said...

@Tutorials, Currency denomination which is Enum is final constants e.g. 100 , 10 or 5 etc. Currency variable which is int in case of enum int pattern can point out any valid int values instead of valid Currency value. Because Compiler doesn't throw compile time error in case of enum int pattern, its not type safe but in case Currency is Enum you can only assign valid Currency denomination which is Enum instances or constants.

October 5, 2012 at 10:37 PM

Srinivas said...

Nice articles..

October 7, 2012 at 7:17 PM

Anonymous said...

enum CoffeeSize{ BIG,HUGE,LARGE} ;

CoffeeSize cs=CoffeeSize.HUGE;
cs=CoffeeSize.LARGE;

i have tested above code on java6+ netbean7.2 it's working but it's contradict with the following

"4) Enum constants are implicitly static and final 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."

November 29, 2012 at 9:42 PM

Donald Arthur Kronos - Actor said...

Anonymous, Nov 29: That example....

enum CoffeeSize{ BIG,HUGE,LARGE} ;

CoffeeSize cs=CoffeeSize.HUGE;
cs=CoffeeSize.LARGE;

... is not reassigning a constant or final field, such as CoffeeSize.Large but rather reassigning the variable cs of enum type CoffeeSize.

Donald Kronos

December 29, 2012 at 6:36 PM

Anonymous said...

Here is an alternative way to storing additional information associated with each enum constant:
//First the enum:
enum Days {
MONDAY ("Lundi", "Montag", "indu vasaram"),
TUESDAY ("Mardi", "Dienstag", "bhauma vasaram"),
WEDNESDAY ("Mercredi", "Mittwoch", "saumya vasaram"),
THURSDAY ("Jeudi", "Donnerstag", "guru vasaram"),
FRIDAY ("Vendredi", "Freitag", "bhrgu vasaram"),
SATURDAY ("Samedi", "Samstag", "sthira vasaram"),
SUNDAY ("Dimanche", "Sonntag", "bhanu vasaram");

String inFrench;
String inGerman;
String inSanskrit;
Days(String inFrench, String inGerman, String inSanskrit)
{
this.inFrench = inFrench;
this.inGerman = inGerman;
this.inSanskrit = inSanskrit;
}
}
//Now an app to use the enum:
public class DaysTest {

public static void main(String[] args) {
System.out.println("\nDays of the week in Sanskrit:");
for (Days d : Days.values())
{
System.out.println(d.inSanskrit);

}
}
}
Thanks and regards,
Steve

March 31, 2013 at 10:05 AM

Anonymous said...

Hi Javin,
Sorry -- forgot to mention:
Excellent tutorial!
Thank you also other commentors for your information.

Regards,
Steve

March 31, 2013 at 10:07 AM

SARAL SAXENA said...

Hi Javin,

Gr8 article few things I want to add to simplify the things , Please let me know what is your point of view on this below understandings..!!

You should always use enums when a variable (especially a method parameter) can only take one out of a small set of possible values. Examples would be things like type constants (contract status: "permanent", "temp", "apprentice"), or flags ("execute now", "defer execution").

If you use enums instead of integers (or String codes), you increase compile-time checking and avoid errors from passing in invalid constants, and you document which values are legal to use.

BTW, overuse of enums might mean that your methods do too much (it's often better to have several separate methods, rather than one method that takes several flags which modify what it does), but if you have to use flags or type codes, enums are the way to go.

As an example, which is better?

```
/** Counts number of foobangs.
* @param type Type of foobangs to count. Can be 1=green foobangs,
* 2=wrinkled foobangs, 3=sweet foobangs, 0=all types.
* @return number of foobangs of type
*/
public int countFoobangs(int type)
```

versus

```
/** Types of foobangs. */
public enum FB_TYPE {
GREEN, WRINKLED, SWEET,
/** special type for all types combined */
ALL;
}

/** Counts number of foobangs.
* @param type Type of foobangs to count
* @return number of foobangs of type
*/
public int countFoobangs(FB_TYPE type)
```

In the second example, it's immediately clear which types are allowed, docs and implementation cannot go out of sync, and the compiler can enforce this.

April 11, 2013 at 11:30 PM

SARAL SAXENA said...

@Gesu yeah Nice questions you have asked here the solution for your first question ..

Q1) Can one Enum extends another Enum in Java? or Can Enum extends another Class in Java ?

No, you can't. If you look at the definition of Enum, instances of it are final and can't be extended. This makes sense if you understand enums as a finite, final set of values.

No it's not possible. The best you can do is make two enums implement and interface and then use that interface instead of the enum. So:

```
interface Digit {
int getValue();
}

enum Decimal implements Digit {
ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE;

private final int value;

Decimal() {
value = ordinal();
}

@Override
public int getValue() {
return value;
}
}

enum Hex implements Digit {
A, B, C, D, E, F;

private final int value;
```

```
Hex() {
value = 10 + ordinal();
}

@Override
public int getValue() {
return value;
}
}
```

April 12, 2013 at 12:24 AM

Márton Marczell said...

In the 8th example the implementation of toString() is very wrong. A toString() method should not print anything to System.out, but instead return all that information as a String.

April 13, 2013 at 11:44 AM

Javin @ Sort Hashtable on values in Java said...

@Marton, You are absolutely right on that toString() should return a String, instead of printing. This is more for demonstration purpose, any of those switch cases can return String equivalent of enum, instead of a call to super.enum at bottom.

April 13, 2013 at 9:13 PM

suresh said...

Another good example of enum from JDK is ThreadState enum, which is official guide on different thread state. This enum describe all possible thread states e.g. NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING etc, and removes lot of confusion around thread state. This is also an example of where to use Enum, Surely using it to represent state is one of the best usecase.

April 16, 2013 at 8:27 PM

Anonymous said...

OMG...im loving each nd every tutorial of urs..simple, neat, clear. Better u write a book.
I think u r d only admin who takes the comments given by readers seriously nd incorporates them in ur posts :) hats off to u... keep blogging. u hv great fan following

April 28, 2013 at 7:30 AM

m2wester said...

You should be carefull using ordinal().

It's final, so you cannot adapt it to your needs. If you want them to start with 1 - not possible. If you want one number not to be used (eg because you removed a value from your enum and you do not want the old ordinals to change), you have a problem.

If you need an arbitrary numeric representation of your values, you should consider providing your own method instead of using ordinal().

May 10, 2013 at 5:16 AM

Harry said...

Hey, I have a question, Can one Enum extends another Enum in Java? is it possible to declare Enum using extends clause? This was asked to me in a Java interview. Since enum is neither class nor interface, I find it tricky, but as you said Enum can implement interface, I am assuming it can also extend another class or enum?

May 19, 2013 at 7:36 PM

Anonymous said...

Is there any change on Enum in Java 6, Java 7 or even expecting change in java 8 ?

May 31, 2013 at 8:51 PM

James said...

WOW, lots of examples from Enum. Great works guys, including Commentators. Here is one of my Enum examples. It's from financial worlds. We have a clearer, normally banks, which clears trades and charge clearing fees ranging form $100- $500. If you are dealing with only few clearer, you can use Enum to represent them as shown below :

```
private enum Clearer{
BankOfAmerica(new BigDecimal(300)), Citibank(new BigDecimal(300)), Goldman(new BigDecimal(300));

private BigDecimal clearingCharge; //in dollars

private Clearer(BigDecimal fees){
```

```
this.clearingCharge = fees;
}

public BigDecimal getClearningCharge(){
return clearingCharge;
}
}
```

getClearingCharge() method is a nice convenient way to get clearing fees.

July 8, 2013 at 11:45 PM

Anonymous said...

Can we use abstract keyword along with Enum in Java? I want to use abstract method with enum, how to use that?

July 15, 2013 at 6:37 AM

Anonymous said...

Just to add my 2 cents on this wonderful article on Enum, I would like to share few tips on when to use Enum in Java programs :

1) Prefer Enum over String, for example if your method accepts String argument, which you compare against predefined one, than consider passing Enum, it will provide compile time type safety.

2) Prefer Enum over integer constant, Similar to reasons mentioned above, if you are accepting int and comparing with constants, consider accepting Enum.

3) Consider Enum for passing choices and options by wrapping Enum constants inside EnumSet e.g. EnumSet.of(Choice.ONE, Choice.TWO)

4) You can even implement Strategy design pattern and State design Pattern using Enum in Java.

Cheers

July 25, 2013 at 12:08 AM

Anonymous said...

constructor of enum need not to be private check it
once

September 19, 2013 at 3:33 AM

Sourabh said...

Read many article on Enum but was never clarified until i read your blog.Its very nice and crisp thnks for shring it in very simple way,helped me a lot to know the versatility of Enum in Java

November 4, 2013 at 10:28 PM

sudhanshu sharma said...

yes constructor in enum need not to be private.

March 16, 2014 at 10:15 AM

sudhanshu sharma said...

Constructor in enum need not be emply...Anonymous is right...anyway this tutorial is really nice. please make tutorial related to thread also.

March 16, 2014 at 10:16 AM

Anonymous said...

you are a brave java developer...for this(Further Reading on Java Enum...)

May 14, 2014 at 9:44 PM

Prats..! said...

The awesomest explanation ever. It is so detailed and very well explained. Thanks a ton!

June 4, 2014 at 4:26 PM

Anonymous said...

Is it mandatory to make constructor of enum as private

July 2, 2014 at 11:27 PM

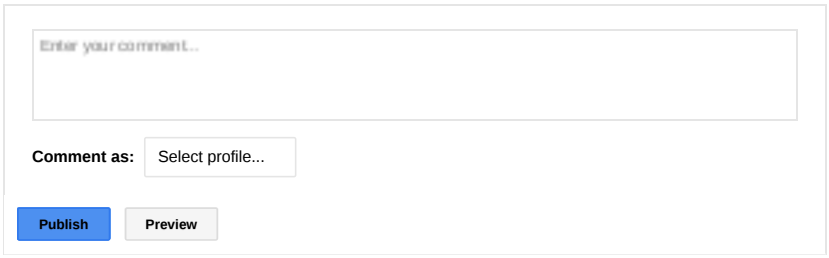
Anonymous said...

You have written that constructors of enum type can only be private...

But I have read on docs.orale.com that, it can be private or package-private
.
.
Can you please re-confirm this

December 21, 2014 at 4:48 AM

Post a Comment



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

About Me Privacy Policy