**Java Enum**

**#.** Java Enum was introduced in Java **1.5** as a new type whose fields consists of fixed set of constants. That means, enum is used to define constant values (does not change value). For example, we can create directions as Java Enum with fixed fields as EAST, WEST, NORTH and SOUTH.

**#.** Purpose of enum is also increasing the code readability for programmers.

**#.** After compilation we get things in two form class or interface. We get Enum in the form of class. It also creates separate .class file. If we execute “**javap MyClass”** tool on created enum class we will get that it converts enum codes to normal other class.

**Program:**

**Java enum example**

Java **enum** keyword is used to create an enum type. Let’s have a look at the java enum example program.

package com.journaldev.enums;

public enum ThreadStates {

START, RUNNING, WAITING, DEAD;

}

In above example, ThreadStates is the enum with fixed constants fields START, RUNNING, WAITING and DEAD.

Now let’s see how java enum is better than normal constants fields in java classes.

**Let’s create a similar constants class in java.**

package com.journaldev.enums;

public class ThreadStatesConstant {

public static final int START = 1;

public static final int WAITING = 2;

public static final int RUNNING = 3;

public static final int DEAD = 4;

}

**Now let’s see both enum and constants in usage/Implementation:**

// Code Readability and Advantages:

// Via Enum

enum ThreadStates **{**

START**,** RUNNING**,** WAITING**,** DEAD**;**

**}**

// via final static constants

public class ThreadStatesConstant **{**

// same as enum with final static data members

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

public static final int WAITING **=** 2**;**

public static final int RUNNING **=** 3**;**

public static final int DEAD **=** 4**;**

// main

public static void main**(**String**...**a**)** **{**

//Enum values are fixed

simpleEnumExample**(**ThreadStates**.**START**);**

simpleEnumExample**(**ThreadStates**.**WAITING**);**

simpleEnumExample**(**ThreadStates**.**RUNNING**);**

simpleEnumExample**(**ThreadStates**.**DEAD**);**

simpleEnumExample**(null);**

/\* we can’t figure out what 1 or other numbers are denoting. As we can see above that explains what parameter is about. \*/

simpleConstantsExample**(**1**);**

simpleConstantsExample**(**2**);**

simpleConstantsExample**(**3**);**

simpleConstantsExample**(**4**);**

//we can pass any int constant but not in case of enum

simpleConstantsExample**(**5**);**

**}**

/\* This method shows the benefit of using Enum over Constants \*/

private static void simpleEnumExample**(**ThreadStates th**)** **{**

**if(**th **==** ThreadStates**.**START**)** System**.**out**.**println**(**"Thread started"**);**

**else** **if** **(**th **==** ThreadStates**.**WAITING**)** System**.**out**.**println**(**"Thread is waiting"**);**

**else** **if** **(**th **==** ThreadStates**.**RUNNING**)** System**.**out**.**println**(**"Thread is running"**);**

**else** System**.**out**.**println**(**"Thread is dead"**);**

**}**

private static void simpleConstantsExample**(**int i**)** **{**

**if(**i **==** ThreadStatesConstant**.**START**)** System**.**out**.**println**(**"Thread started"**);**

**else** **if** **(**i **==** ThreadStatesConstant**.**WAITING**)** System**.**out**.**println**(**"Thread is waiting"**);**

**else** **if** **(**i **==** ThreadStatesConstant**.**RUNNING**)** System**.**out**.**println**(**"Thread is running"**);**

**else** System**.**out**.**println**(**"Thread is dead"**);**

**}**

**}**

If we look at the above example, we have two risks with using constants that are solved by enum.

1. We can pass any int constant to the simpleConstantsExample method but we can pass only fixed values to simpleEnumExample, so it provides type safety.
2. We can change the int constants value in ThreadStatesConstant class but the above program will not throw any exception. Our program might not work as expected but if we change the enum constants, we will get compile time error that removes any possibility of runtime issues.

**Now let’s see more features of java enum with an example.**

**import** java**.**io**.**Closeable**;**

**import** java**.**io**.**IOException**;**

/\*\* This Enum example shows all the things we can do with Enum types \*/

enum ThreadStatesEnum **implements** Closeable**{**

START**(**1**){**

@Override

public String toString**(){**

**return** "START implementation. Priority="**+**getPriority**();**

**}**

@Override

public String getDetail**()** **{**

**return** "START"**;**

**}**

**},**

RUNNING**(**2**){**

@Override

public String getDetail**()** **{**

**return** "RUNNING"**;**

**}**

**},**

WAITING**(**3**){**

@Override

public String getDetail**()** **{**

**return** "WAITING"**;**

**}**

**},**

DEAD**(**4**){**

@Override

public String getDetail**()** **{**

**return** "DEAD"**;**

**}**

**};**

private int priority**;**

public abstract String getDetail**();**

//Enum constructors should always be private.

private ThreadStatesEnum**(**int i**){**

priority **=** i**;**

**}**

//Enum can have methods

public int getPriority**(){**

**return** **this.**priority**;**

**}**

public void setPriority**(**int p**){**

**this.**priority **=** p**;**

**}**

//Enum can override functions

@Override

public String toString**(){**

**return** "Default ThreadStatesConstructors implementation. Priority=" **+** getPriority**();**

**}**

@Override

public void close**()** **throws** IOException **{**

System**.**out**.**println**(**"Close of Enum"**);**

**}**

**}**

public class EnumOverride**{**

public static void main**(**String**...**a**){**

System**.**out**.**println**(**ThreadStatesEnum**.**START**);** // START implementation. Priority=1

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**toString**());** // START implementation. Priority=1

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

ThreadStatesEnum**.**START**.**setPriority**(**10**);**

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

**}**

**}**

**Output:**

**START implementation. Priority=1  
START implementation. Priority=1  
1  
10**

**Java Enum Important Points**

Below are some of the important points for Enums in Java.

1. All java enum implicitly extends **java.lang.Enum** class that extends Object class and implements [**Serializable**](https://www.journaldev.com/2452/serialization-in-java) and [**Comparable**](https://www.journaldev.com/780/comparable-and-comparator-in-java-example) interfaces. **So we can’t extend any class in enum.**
2. Since enum is a keyword, we can’t end package name with it, for example **com.journaldev.enum** is not a valid package name.
3. Enum can implement [interfaces](https://www.journaldev.com/1601/interface-in-java). As in above enum example, it’s implementing Closeable interface.
4. Enum constructors are always private.If you don't declare private compiler internally creates private constructor.
5. We can’t create instance of enum using new operator.
6. We can declare [**abstract methods in java**](https://www.journaldev.com/1582/abstract-class-in-java) **enum**, then all the enum fields must implement the abstract method.
7. We can define a method in enum and enum fields can override them too. For example, toString()method is defined in enum and enum field START has overridden it.
8. Java enum fields has namespace, we can use enum field only with class name like ThreadStates.START
9. Enums can be used in [**switch statement**](https://www.journaldev.com/588/java-switch-case-string), we will see it in action in the later part of this tutorial.
10. We can extend existing enum without breaking any existing functionality. For example, we can add a new field NEW in ThreadStates enum without impacting any existing functionality.
11. Since enum fields are constants, java best practice is to write them in block letters and underscore for spaces. For example EAST, WEST, EAST\_DIRECTION etc.
12. Enum constants are implicitly static and final
13. Enum constants are final but its variable can still be changed. For example, we can use setPriority() method to change the priority of enum constants. We will see it in usage in below example.
14. Since enum constants are final, we can safely compare them using “==” and equals() methods. Both will have the same result.
15. Enum also creates **separate .class file**.
16. The enum can be defined **within or outside** the class because it is similar to a class.
17. enum can be traversed
18. If make enum as public then we have to make .java file same as enum name.

**Q. What is the purpose of values() method in enum?**

The java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

**values()** method that returns an array containing all of the values of the enum in the order they are declared. Note that this method is automatically generated by java compiler for every enum. **You won’t find values() implementation in java.util.Enum class.**

**valueOf (enumType, name)**

With the help of valueOf(enumType, name) we can create an enum object from String. It throws IllegalArgumentException if the specified enum type has no constant with the specified name, or the specified class object does not represent an enum type. It also throws NullPointerException if any of the arguments are null.

**Programs:**

**Creating an enum and its few methods.**

/\* creating enum. To create enum we have to use enum keyword.

Java enum keyword is used to create an enum type. Let’s have a look at the java enum example program.

\*/

enum MyCars**{**

// Defining constant fields:

/\* Since enum fields are constants, Java best practice is to write them in block letters and underscore for spaces. \*/

HONDA**,** BMW**,** LANDROVER

**};**

enum Direction**{** EAST**,** WEST**,** SOUTH**,** NORTH **};**

enum Week**{** MONDAY**,** TUESDAY**,** WEDNESDAY**,** THURSDAY**,** FRIDAY**,** SATURDAY**,** SUNDAY **};**

/\* going to use enum in our class \*/

public class MyEnum**{**

public static void main**(**String**...**a**){**

/\*1.\*/

System**.**out**.**println**(**"\n============= Accessing enums ===========\n"**);**

MyCars newCar **=** MyCars**.**BMW**;**

System**.**out**.**println**(**"My New Car is: " **+** newCar**);**

System**.**out**.**println**(**"\n============= All Constants ===========\n"**);**

/\* 2. The Java compiler internally adds the values() method when it creates an enum.

The values() method returns an array containing all the values of the enum.

\*/

Direction**[]** dir **=** Direction**.**values**();**

**for(**Direction d**:** dir**){**

System**.**out**.**println**(**d **+** " direction with the help of values()"**);**

**}**

/\* 3.

Enum's valueOf() method

\* @parameter name the name of the constant to return

\* @return the enum constant of the specified enum type with the specified name

\* @throws IllegalArgumentException if the specified enum type has no constant with the specified name, or the specified

class object does not represent an enum type @throws NullPointerException if {@code enumType} or {@code name} is null. \*/

System**.**out**.**println**(**"\n============= Conversion ===========\n"**);**

MyCars cr **=** MyCars**.**valueOf**(**"LANDROVER"**);**

System**.**out**.**println**(**"cr contains: " **+** cr**);**

Direction st **=** Direction**.**valueOf**(**"SOUTH"**);**

System**.**out**.**println**(**"st contains: " **+** st**);**

**try{**

Direction st1 **=** Direction**.**valueOf**(**"SOUTH1223"**);**

**}catch(**IllegalArgumentException e**){**

System**.**out**.**println**(**e**);**

**}**

/\* 4. Enum implements "Compareable" interface and it has compareTo() method.

\* When we create constants then JVM saves ordinal value of that constants value.

In compareTo() JVM compares them by ordinal value.

\* Compares this enum with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

\* Enum constants are only comparable to other enum constants of the same enum type. The natural order implemented by this method is the order in which the constants are declared.

return self.ordinal - other.ordinal;

\*/

System**.**out**.**println**(**"\n============= Comparison ===========\n"**);**

Week day1**,** day2**,** day3**;**

day1 **=** Week**.**MONDAY**;**

day2 **=** Week**.**TUESDAY**;**

day3 **=** Week**.**FRIDAY**;**

**if(**day1**.**compareTo**(**day2**)** **<** 0**){**

System**.**out**.**println**(**day1 **+** " comes before " **+** day2**);**

**}**

**if(**day2**.**compareTo**(**day3**)** **>** 0**){** // false

System**.**out**.**println**(**day2 **+** " comes before " **+** day3**);**

**}**

**if(**day1**.**compareTo**(**day3**)** **==** 0**){** // false

System**.**out**.**println**(**day1 **+** " equals " **+** day3**);**

**}**

**if(**day1**.**compareTo**(**day1**)** **==** 0**){** // true

System**.**out**.**println**(**day1 **+** " equals " **+** day1**);**

**}**

/\*5. Know Ordinal value of enum constants with the help of ordinal method\*

System**.**out**.**println**(**"\n============= Ordinal Values ===========\n"**);**

Week**[]** days **=** Week**.**values**();**

**for(**Week d**:** days**){**

System**.**out**.**println**(**d **+** ", Ordinal value is: " **+** d**.**ordinal**());**

**}**

**}**

**}**

**Output:**

============= Accessing enums ===========

My New Car is: BMW

============= All Constants ===========

EAST direction with the help of values()

WEST direction with the help of values()

SOUTH direction with the help of values()

NORTH direction with the help of values()

============= Conversion ===========

cr contains: LANDROVER

st contains: SOUTH

java.lang.IllegalArgumentException: No enum constant Direction.SOUTH1223

============= Comparison ===========

MONDAY comes before TUESDAY

MONDAY equals MONDAY

============= Ordinal Values ===========

MONDAY, Ordinal value is: 0

TUESDAY, Ordinal value is: 1

WEDNESDAY, Ordinal value is: 2

THURSDAY, Ordinal value is: 3

FRIDAY, Ordinal value is: 4

SATURDAY, Ordinal value is: 5

SUNDAY, Ordinal value is: 6

**Equals():**

enum Friends**{**

PREM**,** VINEET**,** JAVED**,** TAUSHIF**,** LOVELY**,** PARUL

**};**

enum MoreFriends**{**

ANIKA**,** PREM**,** VINEET**,** JAVED**,** TAUSHIF**,** LOVELY**,** PARUL

**};**

class MyFriends**{**

public static void main**(**String**...**a**){**

// Enum class has also overridden the equals() method that compares the reference id.

Friends f1**,** f2**,** f3**;**

f1 **=** Friends**.**VINEET**;**

f2 **=** Friends**.**VINEET**;**

f3 **=** Friends**.**PREM**;**

**if(**f1**.**equals**(**f2**))** **{** System**.**out**.**println**(**f1 **+** " and " **+** f2 **+** " is equal..."**);** **}**

**if(!**f1**.**equals**(**f3**))** **{** System**.**out**.**println**(**f1 **+** " and " **+** f3 **+** " is not equal..."**);** **}**

// As we know it compares the reference id then we can compare it through ( == ) operator

System**.**out**.**println**(**"\n========== Via Operators ============\n"**);**

**if(**f1 **==** f2 **)** **{** System**.**out**.**println**(**f1 **+** " and " **+** f2 **+** " is equal..."**);** **}**

**if(**f1 **!=** f3**)** **{** System**.**out**.**println**(**f1 **+** " and " **+** f3 **+** " is not equal..."**);** **}**

// Let's compare the different "Enum" class with the same constant value.

System**.**out**.**println**(**"\n====== Comparison Between Two Enum Classes Constants =====\n"**);**

Friends fj **=** Friends**.**JAVED**;**

MoreFriends mfj **=** MoreFriends**.**JAVED**;**

**if(**fj**.**equals**(**mfj**)){**

System**.**out**.**println**(**Friends**.**JAVED **+** " is equal to " **+** MoreFriends**.**JAVED**);**

**}else** **{** System**.**out**.**println**(**Friends**.**JAVED **+** " is not equal to " **+** MoreFriends**.**JAVED**);** **}**

/\* I don't know why it says at compile: error: incomparable types: \*/

/\* if(fj == mfj){

System.out.println(Friends.JAVED + " is equal to " + MoreFriends.JAVED);

}else { System.out.println(Friends.JAVED + " is not equal to " + MoreFriends.JAVED); }

\*/

**}**

**}**

**Enum Constructor**

// PrimitiveType constants

enum Apple**{**

// we can't send value to constructor. It will be hard coded.

A**(**10**),** B**(**9**),** C**(**12**),** D**(**15**),** E**(**8**);**

private int price**;** //data member

Apple**(**int x**){**

price **=** x**;**

**}**

// getter methods

int getPrice**(){**

**return** price**;**

**}**

**};**

class MyClass**{**

public static void main**(**String**...**a**){**

// In below code when cursor reach to 'D' then implicitly JVM calls the constructor

// Hard coded value assigned inside 'p' data member that we get via getPrice();

System**.**out**.**println**(**"D costs: " **+** Apple**.**D**.**getPrice**());**

// Same as can get other constants value.

System**.**out**.**println**(**"E costs: " **+** Apple**.**E**.**getPrice**());**

// We can get all apples price via calling values.

System**.**out**.**println**(**"\n====== Cost of all apples are =====\n"**);**

**for(**Apple ap**:** Apple**.**values**()){**

System**.**out**.**println**(**ap **+** " costs: " **+** ap**.**getPrice**()** **+** " cents"**);**

**}**

**}**

**}**

**abstract method in method.**

enum Day**{**

// public int y; It is an error, the first thing must be here is constants declaration anything else is compile time error.

MONDAY**(**1**){**

public Day next**(){** **return** TUESDAY**;** **}**

**},**

TUESDAY**(**2**){**

public Day next**(){** **return** WEDNESDAY**;** **}**

**},**

WEDNESDAY**(**3**){**

public Day next**(){** **return** THURSDAY**;** **}**

**},**

THURSDAY**(**4**){**

public Day next**(){** **return** FRIDAY**;** **}**

**},**

FRIDAY**(**5**){**

public Day next**(){** **return** SATURDAY**;** **}**

**},**

SATURDAY**(**6**){**

public Day next**(){** **return** SUNDAY**;** **}**

**},**

SUNDAY**(**7**){**

public Day next**(){** **return** MONDAY**;** **}**

**};**

// Data Member

private final int dayNumber**;**

// Constructor

Day**(**int d**){**

**this.**dayNumber **=** d**;**

**}**

//method

int getDayNumber**(){**

**return** dayNumber**;**

**}**

// Abstract method. needs to override for all constants.

public abstract Day next**();**

**}**

// Implementation class

public class MyClass**{**

public static void main**(**String**...**a**){**

System**.**out**.**printf**(**"Current Day is %s that comes in %sth place in the week" **+** " and next day is %s"**,** Day**.**TUESDAY**,** Day**.**TUESDAY**.**getDayNumber**(),** Day**.**TUESDAY**.**next**()**

**);**

System**.**out**.**println**();**

// Let's access all constants

**for(**Day d**:** Day**.**values**()){**

System**.**out**.**printf**(**"%s %d, next is %s\n"**,** d**,** d**.**getDayNumber**(),** d**.**next**());**

**}**

**}**

**}**

**Overriding**

**import** java**.**io**.**Closeable**;**

**import** java**.**io**.**IOException**;**

/\*\* This Enum example shows all the things we can do with Enum types \*/

enum ThreadStatesEnum **implements** Closeable**{**

START**(**1**){**

@Override

public String toString**(){**

**return** "START implementation. Priority="**+**getPriority**();**

**}**

@Override

public String getDetail**()** **{**

**return** "START"**;**

**}**

**},**

RUNNING**(**2**){**

@Override

public String getDetail**()** **{**

**return** "RUNNING"**;**

**}**

**},**

WAITING**(**3**){**

@Override

public String getDetail**()** **{**

**return** "WAITING"**;**

**}**

**},**

DEAD**(**4**){**

@Override

public String getDetail**()** **{**

**return** "DEAD"**;**

**}**

**};**

private int priority**;**

public abstract String getDetail**();**

//Enum constructors should always be private.

private ThreadStatesEnum**(**int i**){**

priority **=** i**;**

**}**

//Enum can have methods

public int getPriority**(){**

**return** **this.**priority**;**

**}**

public void setPriority**(**int p**){**

**this.**priority **=** p**;**

**}**

//Enum can override functions

@Override

public String toString**(){**

**return** "Default ThreadStatesConstructors implementation. Priority=" **+** getPriority**();**

**}**

@Override

public void close**()** **throws** IOException **{**

System**.**out**.**println**(**"Close of Enum"**);**

**}**

**}**

public class EnumOverride**{**

public static void main**(**String**...**a**){**

System**.**out**.**println**(**ThreadStatesEnum**.**START**);** // START implementation. Priority=1

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**toString**());** // START implementation. Priority=1

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

ThreadStatesEnum**.**START**.**setPriority**(**10**);**

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

System**.**out**.**println**(**ThreadStatesEnum**.**START**.**getPriority**());**

**}**

**}**

**Enum with Switch**

enum ThreadStatesEnum**{**

START**,** WAITING**,** RUNNING**,** DEAD

**};**

class MySwitch**{**

private static void usingEnumInSwitch**(**ThreadStatesEnum th**)** **{**

**switch** **(**th**){**

**case** START**:**

System**.**out**.**println**(**"START thread"**);**

**break;**

**case** WAITING**:**

System**.**out**.**println**(**"WAITING thread"**);**

**break;**

**case** RUNNING**:**

System**.**out**.**println**(**"RUNNING thread"**);**

**break;**

**case** DEAD**:**

System**.**out**.**println**(**"DEAD thread"**);**

**break;**

**default:**

System**.**out**.**println**(**"Invalid"**);**

**break;**

**}**

**}**

public static void main**(**String**...**a**){**

usingEnumInSwitch**(**ThreadStatesEnum**.**START**);**

usingEnumInSwitch**(**ThreadStatesEnum**.**WAITING**);**

usingEnumInSwitch**(**ThreadStatesEnum**.**RUNNING**);**

usingEnumInSwitch**(**ThreadStatesEnum**.**DEAD**);**

**}**

**}**