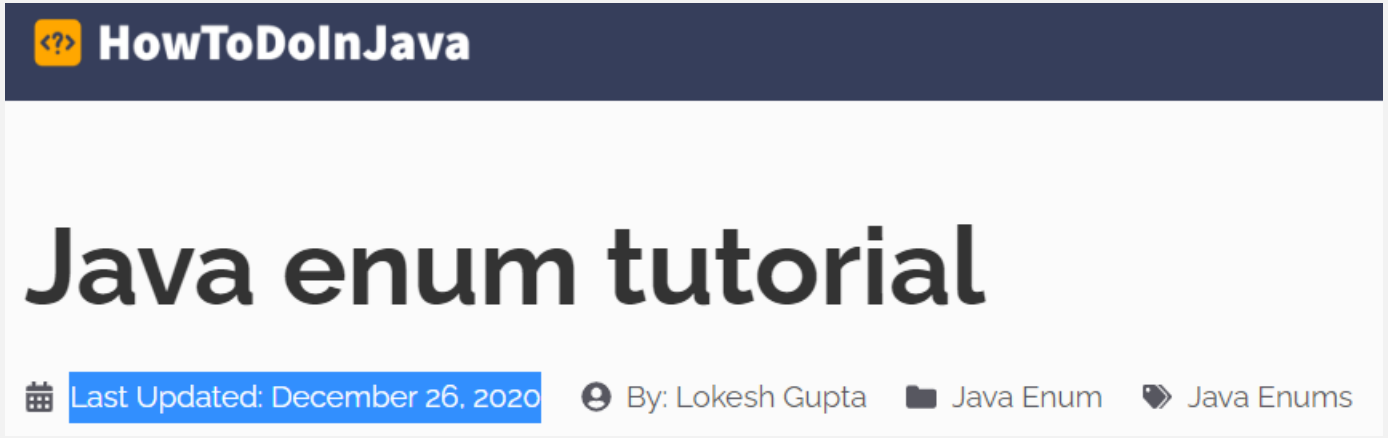


05506004

https://howtodoinjava.com/java/enum/enum-tutorial/

https://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html

# Outline

- What is enum in Java
- enum constructors
- enum methods
- ~~enum inheritance~~
- Compare enums
- EnumMap and EnumSet
- Summary

Java enum, also called Java enumeration type, is a type whose fields consist of a fixed set of constants. The very purpose of enum is to enforce compile time type safety. enum keyword is reserved keyword in Java.

# 1. enum in Java

- Enumerations was officially released in JDK 1.5 release.
- Java enum declaration
- Java enum naïve usage

```
public enum Direction
{
   EAST, WEST, NORTH, SOUTH;
}
```

We can use the enum just like we use `final static` class fields.

```
public class EnumExample
{
    public static void main(String[] args)
    {
        Direction north = Direction.NORTH;

        System.out.println(north);        //Prints NORTH
    }
}
```

Logically, **each enum is an instance of enum type** itself. So given enum can be seen as below declaration. **JVM internally adds ordinal and value methods** to this class which we can call while working with enum.

```
final class Direction extends Enum   ;Direction   ;
{
    public final static Direction EAST = new Direction();
    public final static Direction WEST = new Direction();
    public final static Direction NORTH = new Direction();
    public final static Direction SOUTH = new Direction();
}
```

- notice the naming convention (UPPER_CASE)

# .values() .ordinal()

The **enum values()** method returns all the enum values in an **enum array**.

```
Direction[] directions = Direction.values();

for (Direction d : directions) {
    System.out.println(d);
}

//Output:

EAST
WEST
NORTH
SOUTH
```

The `ordinal()` method returns the order of an enum instance. It represents the **sequence in the enum declaration**, where the initial constant is assigned an ordinal of `'0'`. It is very much like **array indexes**.

It is designed for use by sophisticated enum-based data structures, such as `EnumSet` and `EnumMap`.

```
Direction.EAST.ordinal();     //0

Direction.NORTH.ordinal();    //2
```

# 2. & 3. constructors and methods (getter?)

By default, **enums don't require constructor** definitions and their default values are always the string used in the declaration. Though, you can give define your own constructors to initialize the state of enum types.

For example, we can add `angle` attribute to direction. All directions have some angle. So let's add them.

```
public enum Direction
{
    // enum fields
    EAST(0), WEST(180), NORTH(90), SOUTH(270);

    // constructor
    private Direction(final int angle) {
        this.angle = angle;
    }

    // internal state
    private int angle;

    public int getAngle() {
        return angle;
    }
}
```

- new keyword can not be used to intialize an enum, even within the enum type itself
  - Enum instance is a singleton instance

If we want to access angle for any direction, we can make a simple method call in enum field reference.

```
Direction north = Direction.NORTH;

System.out.println( north );                    //NORTH

System.out.println( north.getAngle() );         //90

System.out.println( Direction.NORTH.getAngle() ); //90
```

# (common) concrete method
vs
# abstract methods in enum

```
public enum DirectionConcreteMethod {
  EAST, WEST, NORTH, SOUTH;

  public String method() {
    return String.format("moving %sward“
                                    , this);
  }
}
```

```
public enum DirectionAbstractMethod {
    EAST {
        @Override
        public String method() {
            return "east is right";
        }
    },
…
    SOUTH {
        @Override
        public String method() {
            return "south is down";
        }
    };

    public abstract String method();
}
```

# 4. comparing enums

All enums are by default **comparable and singletons** as well. It means you can compare them with `equals()` method, even with "==" operator.

```
static void q4() {
    DirectionConcreteMethod north = DirectionConcreteMethod.NORTH;
    DirectionConcreteMethod another = DirectionConcreteMethod.valueOf("NORTH");
    // valueOf(str) return Enum instance
    System.out.println(north == another);
    System.out.println(north.equals(another));
}
```

> You can **compare enum types** using `'=='` operator or `equals()` method, because enums are singlton and comparable by default.

- Singleton allows enum instance to be compared like primitives.

```
static void q6(USCOIN coin) {
    int reward = 0;
    switch (coin) {
        case PENNY : reward = 20; break;
        case NICKLE : reward = 22; break;
        case DIME : reward = 27; break;
        case QUARTER : reward = 28;
    }
    System.out.println(reward);
}
```

# 5. Enum collections – EnumSet and EnumMap

```
static void q5() {
    Map<USCOIN, Integer> coinMap = new EnumMap<>(keyType: USCOIN.class);
    coinMap.put(USCOIN.PENNY, USCOIN.PENNY.value());
    coinMap.put(USCOIN.NICKLE, USCOIN.NICKLE.value());
    System.out.println(coinMap);
}
```

```
public enum USCOIN {
    PENNY(1), NICKLE(5), DIME(10), QUARTER(25);
    private int val;
    private USCOIN(int v) {
        this.val = v;
    }
    public int value() {
        return val;
    }
}
```

# Remark

- Enum constructors should be declared as private. The compiler allows non private constructors, but this seems misleading to the reader
- name() and valueOf() methods simply use the text of the enum constants, while toString() method may be overridden to provide any content, if desired
- Since these enumeration instances are all effectively singletons, they can be compared for equality using identity ("==").
  - for enum constants, equals() and "==" evaluates to same result, and can be used interchangeably
- enum constants are implicitly public static final
- if an enum is a member of a class, it’s implicitly static