



Java Enums

Introduction to Enum Types

An enum type is a special data type that enables for a variable to be a set of predefined constants.

The variable must be equal to one of the values that have been predefined for it.

Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.

In the Java programming language, you define an enum type by using the enum keyword.



Example: days-of-the-week enum

```
public enum Day {  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
    THURSDAY, FRIDAY, SATURDAY  
}
```

```
public class EnumTest {
    Day day;

    public EnumTest(Day day) {
        this.day = day;
    }

    public void tellItLikeIts() {
        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 output is:

Mondays are bad.
Midweek days are so-so.
Fridays are better.
Weekends are best.
Weekends are best.

Example: programming-levels enum

```
enum Level { BEGINNER, INTERMEDIATE, EXPERT }
```

← The enum values are constant values.

```
class Game {  
    Level gameLevel;  
}
```

← Variable of type Level

```
class GameApp {  
    Game game = null;  
  
    public void startGame () {  
        game = new Game();  
        game.gameLevel = Level.BEGINNER;  
    }  
}
```

← Assigns constant BEGINNER

Decompile enum level

```
final class Level extends Enum
{
    public static final Level BEGINNER;
    public static final Level INTERMEDIATE;
    public static final Level EXPERT;

    private static final Level $VALUES [];

    static
    {
        BEGINNER = new Level("BEGINNER", 0);
        INTERMEDIATE = new Level("INTERMEDIATE", 1);
        EXPERT = new Level("EXPERT", 2);
        $VALUES = (new Level[] {
            BEGINNER, INTERMEDIATE, EXPERT
        });
    }

    public static Level[] values()
    {
        return (Level[])$VALUES.clone();
    }

    public static Level valueOf(String s)
    {
        return (Level)Enum.valueOf(Level, s);
    }

    private Level(String s, int i)
    {
        super(s, i);
    }
}
```

1 enum is implicitly declared final.

2 enum constants are implicitly public, static, and final.

3 Array to store reference to all enum constants

4 Creation of enum constants occurs in static initializer block

5 Method values return an array of all enum constants.

6 Method valueOf() parses a String value and returns corresponding enum constant

7 Private constructor

Enum Constructor

```
enum Direction {
    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;
    }
}
```


Non-abstract Methods

```
public enum Direction {  
  
    EAST, WEST, NORTH, SOUTH;  
  
    protected String message() {  
  
        String message = "Moving in " + this + " direction";  
        return message;  
    }  
}
```

Abstract Methods

```
public enum Direction
{
    EAST {
        @Override
        public String message() {
            return "You are moving in east. You will face sun in morning time.";
        }
    },

    WEST {
        @Override
        public String message() {
            return "You are moving in west. You will face sun in evening time.";
        }
    },

    NORTH {
        @Override
        public String message() {
            return "You are moving in north. Sea behind.";
        }
    },

    SOUTH {
        @Override
        public String message() {
            return "You are moving in south. Sea ahead.";
        }
    };


    public abstract String message();
}
```

EnumSet

```
Set enumSet = EnumSet.of(Direction.EAST,  
                          Direction.WEST,  
                          Direction.NORTH,  
                          Direction.SOUTH  
                          );
```

EnumMap

```
Map<Direction, Integer> enumMap = new EnumMap(Direction.class);  
  
enumMap.put(Direction.EAST, Direction.EAST.getAngle());  
enumMap.put(Direction.WEST, Direction.WEST.getAngle());  
enumMap.put(Direction.NORTH, Direction.NORTH.getAngle());  
enumMap.put(Direction.SOUTH, Direction.SOUTH.getAngle());
```



Rules to remember about enums

- An enum can define a main method. This means that you can define an enum as an executable Java application.
- The enum constant list must be defined as the first item in an enum, before the declaration or definition of methods and variables.



(continued)

- The enum constant list might not be followed by a semicolon, if the enum doesn't define any methods or variables.
- When an enum constant overrides an enum method, the enum constant creates an anonymous class, which extends the enum.
- An enum constant can define a constant specific class body and use it to override existing methods or define new variables and methods.
- An enum implicitly extends `java.lang.Enum`, so it can't extend any other class. But a class can't explicitly extend `java.lang.Enum`. An enum can implement interface(s).
- An enum can never be instantiated using the keyword `new`.
- You can define multiple constructors in your enums.
- An enum can't define a constructor with `public` or `protected` access level.
- An enum can define an abstract method. Just ensure to override it for all your enum constants.
- The enum method `values()` returns a list of all the enum constants.
- An enum can be defined as a top-level enum, or as a member of another class or interface. It can't be defined local to a method.

Conclusion

Enum declarations are full classes, and the values listed are constant names referring to separate instances of these classes. The enum declaration can contain fields, constructors, and methods, just like other classes.



Resources

List of resources you may use:

- <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>
- <https://blogs.oracle.com/javamagazine/post/how-to-make-the-most-of-java-enums>