enum in Java

Last Updated : 04 Oct, 2024

In Java, **Enumerations or Java Enum** serve the purpose of representing a group of named constants in a programming language. Java Enums are used when we know all possible values at compile time, such as choices on a menu, rounding modes, command-line flags, etc.

What is Enumeration or Enum in Java?

A Java enumeration is a class type. Although we don't need to instantiate an enum using **new,** it has the same capabilities as other classes. This fact makes Java enumeration a very powerful tool. Just like classes, you can give them constructors, add instance variables and methods, and even implement interfaces.

Enum Example:

> *The 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit. Other examples include natural enumerated types (like the planets, days of the week, colors, directions, etc.).*

One thing to keep in mind is that, unlike classes, enumerations neither inherit other classes nor can get extended(i.e become superclass). We can also add variables, methods, and constructors to it. The main objective of an enum is to define our own data types(Enumerated Data Types).

Declaration of enum in Java

Enum declaration can be done outside a class or inside a class but not inside a method.

1. Declaration outside the class

```java
// A simple enum example where enum is declared
// outside any class (Note enum keyword instead of
// class keyword)
enum Color {
    RED,
    GREEN,
    BLUE;
}

public class Test {
    // Driver method
    public static void main(String[] args) {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

Output

```
RED
```

2. Declaration inside a class

```java
// enum declaration inside a class.
public class Test {
    enum Color {
        RED,
        GREEN,
        BLUE;
    }

    // Driver method
    public static void main(String[] args) {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

Output

```
RED
```

- The first line inside the enum should be a list of constants and then other things like methods, variables, and constructors.
- According to Java naming conventions, it is recommended that we name constant with all capital letters

Properties of Enum in Java

There are certain properties followed by Enum as mentioned below:

- **Class Type:** Every enum is internally implemented using the `Class` type.
- **Enum Constants:** Each enum constant represents an object of type enum.
- **Switch Statements:** Enum types can be used in switch statements.
- **Implicit Modifiers:** Every enum constant is implicitly `public static final`. Since it is static, it can be accessed using the enum name. Since it is final, enums cannot be extended.
- **Main Method:** Enums can declare a `main()` method, allowing direct invocation from the command line.

Below is the implementation of the above properties:

```java
// A Java program to demonstrate working on enum
// in a switch case (Filename Test.java)

import java.util.Scanner;

// An Enum class
enum Day {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY;
}

// Driver class that contains an object of "day" and
// main().
public class Test {
    Day day;

    // Constructor
    public Test(Day day) {
        this.day = day;
    }
    // Prints a line about Day using switch
    public void dayIsLike() {
        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;
        }
    }

    // Driver method
    public static void main(String[] args) {
        String str = "MONDAY";
        Test t1 = new Test(Day.valueOf(str));
        t1.dayIsLike();
    }
}
```

Output

```
Mondays are bad.
```

Java Enum Programs

1. Main Function Inside Enum

Enums can have a `main` function, allowing them to be invoked directly from the command line.

Below is the implementation of the above property:

```
// A Java program to demonstrate that we can have
// main() inside enum class.

public enum Color {
    RED,
    GREEN,
    BLUE;

    // Driver method
    public static void main(String[] args) {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

Output

```
RED
```

The above program will not run on online compiler. To run this program, save it as `Color.java` and use the `java` command to compile and execute.

2. Loop through Enum

We can iterate over the Enum using the `values()` method, which returns an array of enum constants.

Below is the implementation of the loop through Enum:

```java
// Java Program to Print all the values
// inside the enum using for loop
import java.io.*;

// Enum Declared
enum Color {
    RED,
    GREEN,
    BLUE;
}

// Driver Class
class GFG {

    // Main Function
    public static void main(String[] args) {
        // Iterating over all the values in
        // enum using for loop
        for (Color var_1 : Color.values()) {
            System.out.println(var_1);
        }
    }
}
```

Output

```
RED
GREEN
BLUE
```

3. Enum in a Switch Statement

Enums can be used in switch statements to handle different cases based on the enum constants.

```java
// Java Program to implement
// Enum in a Switch Statement
import java.io.*;

// Driver Class
class GFG {
    // Enum Declared
    enum Color {
        RED,
        GREEN,
        BLUE,
        YELLOW;
    }

    // Main Function
    public static void main(String[] args) {
        Color var_1 = Color.YELLOW;

        // Switch case with Enum
        switch (var_1) {
        case RED:
            System.out.println("Red color observed");
            break;
        case GREEN:
            System.out.println("Green color observed");
            break;
        case BLUE:
            System.out.println("Blue color observed");
            break;
        default:
            System.out.println("Other color observed");
        }
    }
}
```

Output

```
Other color observed
```

Enum and Inheritance

- All enums implicitly extend **java.lang.Enum class**. As a class can only extend **one** parent in Java, an enum cannot extend anything else.
- **toString() method** is overridden in **java.lang.Enum class**, which returns enum constant name.

- enum can implement many interfaces.

Enum and Constructor

- Enums can contain constructors, which are called separately for each enum constant at the time of enum class loading.
- We can't create enum objects explicitly and hence we can't invoke the enum constructor directly.

Enum and Methods

Enums can have both concrete and abstract methods. If an enum class has an abstract method, each enum constant must implement it.

```java
// Java program to demonstrate that enums can have
// constructor and concrete methods.

// An enum (Note enum keyword in place of class keyword)
enum Color {
    RED,
    GREEN,
    BLUE;

    // Enum constructor called separately for each
    // constant
    private Color() {
        System.out.println("Constructor called for : " + this.toString());
    }

    public void colorInfo() {
        System.out.println("Universal Color");
    }
}

public class Test {
    // Driver method
    public static void main(String[] args) {
        Color c1 = Color.RED;
        System.out.println(c1);
        c1.colorInfo();
    }
}
```

Output

```
Constructor called for : RED
Constructor called for : GREEN
Constructor called for : BLUE
```

```
RED
Universal Color
```

> ***Note:*** *While the set of constants in a Java enum is fixed at runtime and cannot be changed dynamically, you can modify the source code of the enum to add or remove constants and then recompile the application to reflect these changes.*

Creating a Class with Enum Member

When combined with classes, enum can serve as powerful tool for organising program.

```java
/*package whatever //do not write package name here */

import java.io.*;

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

public class EnumTest {
    // Enum member variable
    Day day;

    // constructor which takes enum value
    public EnumTest(Day day) { this.day = day; }

    // method to execute action as per enum value
    public void tellItLikeItIs()
    {
        switch (day) {
        case MONDAY:
            System.out.println("Mondays are tough");
            break;
        case TUESDAY:
            System.out.println("Tuesday are better");
            break;
        case WEDNESDAY:
            System.out.println("Wednesday are okay");
            break;
        case THURSDAY:
            System.out.println("Thursdays are hopeful");
            break;
        case FRIDAY:
            System.out.println("Fridays are exciting");
            break;
        case SATURDAY:
            System.out.println("Saturdays are relaxing");
            break;
        case SUNDAY:
            System.out.println("Sunday are for rest");
            break;
        default:
            System.out.println("Everyday are good");
            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();
    }
}
```

Output

```
Mondays are tough
Wednesday are okay
Fridays are exciting
Saturdays are relaxing
Sunday are for rest
```

The EnumTest class in above code is created with member of type Day. It has constructor which takes Day enum as an argument and assigns it.

The class has method tellItLikeItIs(), which prints message based on value of day.

The main method includes objects of EnumTest using different Day enum values and calling tellItLikeItIs() method on each.

> *NOTE: In the main method new is used when creating instances of EnumTest because EnumTest is a regular Java class not enum*

itself. So, the new keyword is used to create new instance of EnumTest class, and passing enum value to its constructor.

Implementing Abstract Methods in Enum Class

1. Create an enum for each day of a week

```
/*package whatever //do not write package name here */
import java.io.*;

enum Day {
    MONDAY(){ },
    TUESDAY(){ },
    WEDNESDAY(){ },
    THURSDAY(){ },
    FRIDAY(){ },
    SATURDAY(){ },
    SUNDAY() { };
}
```

2. Add an abstract method getNumberOfDay()

```
/*package whatever //do not write package name here */

import java.io.*;

enum Day {
    MONDAY(){ },
    TUESDAY(){ },
    WEDNESDAY(){ },
    THURSDAY(){ },
    FRIDAY(){ },
    SATURDAY(){ },
    SUNDAY() { };
  public abstract String getNumberOfDay();
}
```

3. Implement the abstract method in each enum

```
/*package whatever //do not write package name here */

import java.io.*;

enum Day {
    MONDAY(){
      @Override
      public String getNumberOfDay(){
        return "1st day of a weak";
    },
    TUESDAY(){
      @Override
      public String getNumberOfDay(){
        return "2nd day of a weak";
    },
    WEDNESDAY(){
      @Override
      public String getNumberOfDay(){
        return "3rd day of a weak";
    },
    THURSDAY(){
      @Override
      public String getNumberOfDay(){
        return "4th day of a weak";
    },
    FRIDAY(){
      @Override
      public String getNumberOfDay(){
        return "5th day of a weak";
    },
    SATURDAY(){
      @Override
      public String getNumberOfDay(){
        return "6th day of a weak";
    },
    SUNDAY() {
      @Override
      public String getNumberOfDay(){
        return "7th day of a weak";
    };
  public abstract String getNumberOfDay();
}
```

4. Create a class called EnumTest with an instance variable day that we had just created.

```
/*package whatever //do not write package name here */

import java.io.*;

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

5. Create a main method in which create the object and call the method.

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {
    public static void main (String[] args) {
      EnumTest daynum = new EnumTest(Day.MONDAY );
      System.out.println("The "+daynum.day.name()+"is 
"+daynum.daynum.getNumberOfDay());

      EnumTest daynum = new EnumTest(Day.SATARDAY );
      System.out.println("The "+daynum.day.name()+"is 
"+daynum.daynum.getNumberOfDay());

    }
}
```

Now combine all above code :

```
/*package whatever //do not write package name here */

import java.io.*;

enum Day {
    MONDAY(){
      @Override
      public String getNumberOfDay(){
        return "1st day of a weak";
      }
    },
    TUESDAY(){
      @Override
      public String getNumberOfDay(){
        return "2nd day of a weak";
      }
    },
    WEDNESDAY(){
      @Override
      public String getNumberOfDay(){
        return "3rd day of a weak";
      }
    },
    THURSDAY(){
      @Override
      public String getNumberOfDay(){
        return "4th day of a weak";
      }
    },
    FRIDAY(){
      @Override
      public String getNumberOfDay(){
        return "5th day of a weak";
      }
    },
    SATURDAY(){
      @Override
      public String getNumberOfDay(){
        return "6th day of a weak";
      }
    },
    SUNDAY() {
      @Override
      public String getNumberOfDay(){
        return "7th day of a weak";
      }
    };
  public abstract String getNumberOfDay();
}
public class EnumTest {
      public Day day;
    public EnumTest(Day day) {
        this.day = day;
    };

    public static void main (String[] args) {
      EnumTest daynum = new EnumTest(Day.MONDAY );
      System.out.println("The "+daynum.day.name()+"is
"+daynum.day.getNumberOfDay());

      daynum = new EnumTest(Day.SATURDAY );
      System.out.println("The "+daynum.day.name()+"is
"+daynum.day.getNumberOfDay());

    }
}
```

Iterating from specific range through ENUM_SET

```
/*package whatever //do not write package name here */

import java.io.*;
import java.util.EnumSet;

public class EnumSetExample {
    // Define an enum for days of the week
    public enum Day {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;
    }

    public static void main(String[] args) {
        // Define a specific range from TUESDAY to FRIDAY
        EnumSet<Day> workDays = EnumSet.range(Day.TUESDAY, Day.FRIDAY);

        // Iterate over the specific range of days
        System.out.println("Work days in the range are:");
        for (Day day : workDays) {
            System.out.println(day);
        }
    }
}
```

FAQs on Enum in Java

Can we create the instance of Enum by the new keyword?

> *No, we can't create the instance of the Enum keyword because it contains private constructors only.*

Can we have an abstract method in the Enum?

> *Yes, we have an abstract method in Enum.*

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

> *In Java, the values( ) method can be used to return all values present inside the enum.*

What is the purpose of the valueOf() method in the enum?

> *The valueOf() method returns the enum constant of the specified string value if exists.*

What is the purpose of the ordinal() method in the enum?

> *By using the ordinal() method, each enum constant index can be found, just like an array index.*

Write a program in Java to describe the use of values(), valueOf(), and ordinal() methods in the enum.

```
// Java program to demonstrate working of values(),
// ordinal() and valueOf()

enum Color {
    RED,
    GREEN,
    BLUE;
}

public class Test {
    public static void main(String[] args)
    {
        // Calling values()
        Color arr[] = Color.values();

        // enum with loop
        for (Color col : arr) {
            // Calling ordinal() to find index
            // of color.
            System.out.println(col + " at index "
                                   + col.ordinal());
        }

        // Using valueOf(). Returns an object of
        // Color with given constant.
        // Uncommenting second line causes exception
        // IllegalArgumentException
        System.out.println(Color.valueOf("RED"));
        // System.out.println(Color.valueOf("WHITE"));
    }
}
```

Output

```
RED at index 0
GREEN at index 1
BLUE at index 2
RED
```

Next Article on enum: Enum with Customized Value in Java