Enum in Java

[**2.5**](https://www.geeksforgeeks.org/easy/)

Enumerations serve the purpose of representing a group of named constants in a programming language. For 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, colours, directions, etc.).  
Enums are used when we know all possible values at **compile time**, such as choices on a menu, rounding modes, command line flags, etc. It is not necessary that the set of constants in an enum type stay **fixed** for all time.

In Java (from 1.5), enums are represented using **enum** data type. In Java, we can also add variables, methods and constructors to it. The main objective of 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.

|  |
| --- |
| // 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

|  |
| --- |
| // 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
* First line inside enum should be list of constants and then other things like methods, variables and constructor.
* According to [Java naming conventions](http://www.oracle.com/technetwork/java/codeconventions-135099.html), it is recommended that we name constant with all capital letters

**Important points of enum :**

* Every enum internally implemented by using Class.
* /\* internally above enum Color is converted to
* class Color
* {
* public static final Color RED = new Color();
* public static final Color BLUE = new Color();
* public static final Color GREEN = new Color();
* }\*/
* Every enum constant represents an **object** of type enum.
* enum type can be passed as an argument to **switch** statement.

|  |
| --- |
| // A Java program to demonstrate working on enum  // in 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.
* Every enum constant is always implicitly **public static final**. Since it is **static**, we can access it by using enum Name. Since it is **final**, we can’t create child enums.
* We can declare **main() method** inside enum. Hence we can invoke enum directly from the Command Prompt.

|  |
| --- |
| // A Java program to demonstrate that we can have  // main() inside enum class.  enum Color  {      RED, GREEN, BLUE;        // Driver method      public static void main(String[] args)      {          Color c1 = Color.RED;          System.out.println(c1);      }  } |

* Output :
* RED

**Enum and Inheritance :**

* All enums implicitly extend **java.lang.Enum class**. As a class can only extend **one** parent in Java, so 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.

**values(), ordinal() and valueOf() methods :**

* These methods are present inside **java.lang.Enum**.
* **values() method** can be used to return all values present inside enum.
* Order is important in enums. By using **ordinal() method**, each enum constant index can be found, just like array index.
* **valueOf() method** returns the enum constant of the specified string value, if exists.

|  |
| --- |
| // 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

**enum and constructor :**

* enum can contain constructor and it is executed 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 enum constructor directly.

**enum and methods :**

* enum can contain **concrete** methods only i.e. no any **abstract** method.