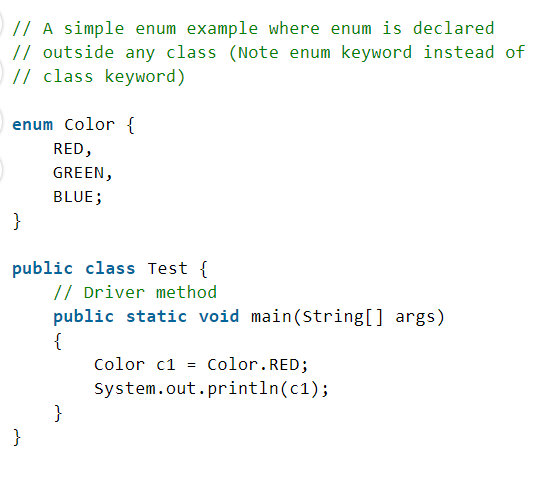
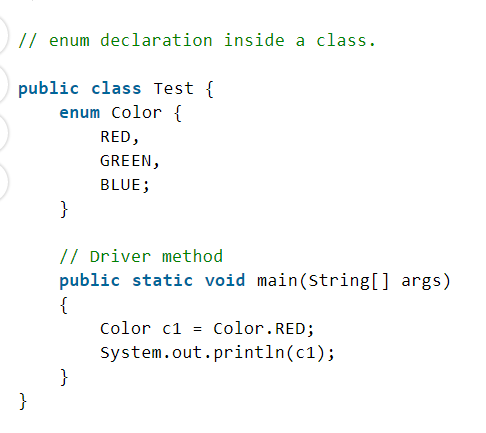
**Enumeration in Java**

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, colors, 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.

**Declaration of enum in Java:**Enum declaration can be done outside a Class or inside a Class but not inside a Method.

* 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

**Important Points of enum:**

Every enum is 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** statements.

Example:

// 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 the enum Name. Since it is **final**, we can’t create child enums.
* We can declare the **main() method** inside the 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);      }  } |

**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 the enum.
* Order is important in enums.By using the **ordinal() method**, each enum constant index can be found, just like an array index.
* **valueOf() method** returns the enum constant of the specified string value if exists.

Example:

|  |
| --- |
| // 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"));      }  } |

**enum and constructor:**

* enum can contain a 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.
* **The constructor of the enum in java** must be **private** any other access modifier will result in a compilation error.

**enum and methods:**

* enum can contain both **concrete** methods and **abstract** methods. If an enum class has an abstract method, then each instance of the enum class must implement it

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

// An enum (Note enum keyword inplace 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

1. **Enum in Java can implement the interface** and override any method like a normal class It’s also worth noting that Enum in java implicitly implements both Serializable and Comparable
2. **You can define abstract methods inside Enum in Java** and can also provide a different implementation for different instances of enum in java.
3. **You can not create an instance of enums by using a new operator** in Java because the constructor of Enum in Java can only be private and Enums constants can only be created inside Enums itself.
4. An instance of Enum in Java is created when any Enum constants are first called or referenced in code.