**Enum Data Type:**

**Theory:**

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.**

In Java (from 1.5), enums are represented using enum data type. Java enums are more powerful than C/C++ enums . 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.

**Outside a class:**

// 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);

}

}

**Inside a class:**

// 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);

}

}

**Enum can not declared as local to a function.**

**Note:**

* First line inside enum should be list of constants and then other things like methods, variables and constructor.
* According to Java naming conventions, it is recommended that we name constant with all capital letters
* Java language designers make it optional to put a semicolon at the end of the enum declaration. So, what gets created when you make an enum? The most important thing to remember that an enum is not string or int. Each of the enumerated Color typesare actually an instance of Color.

**So,**

enum Color

{

RED, GREEN, BLUE;

}

public class Test

{

// Driver method

public static void main(String[] args)

{

Color c1 = Color.RED;

System.out.println(c1);

}

}

**It can be rewritten as:**

enum Color

{

RED, GREEN, BLUE;

};

public class Test

{

// Driver method

public static void main(String[] args)

{

Color c1 = Color.RED;

System.out.println(c1);

}

}

**Since,** each of the enumerated Color types are actually an instance of Color. You can think it as the following:

**class Color**

**{**

**public static final Color RED=new Color(“RED”,0);**

**public static final Color GREEN=new Color(“GREEN”,1);**

**public static final Color BLUE=new Color(“BLUE”,”2”);**

**Color(String colorName, int index)**

**{**

**//stuff here**

**}**

**}**

You can get the exact idea, by doing the following things:

First, compile the following program:

enum Color

{

RED, GREEN, BLUE;

}

public class Test

{

// Driver method

public static void main(String[] args)

{

Color c1 = Color.RED;

System.out.println(c1);

}

}

Using javac Test.java

Now, disassembles it using javap

**Use :** javap Color

**It will generate the following output:**

final class Color extends java.lang.Enum<Color> {

public static final Color RED;

public static final Color GREEN;

public static final Color BLUE;

public static Color[] values();

public static Color valueOf(java.lang.String);

static {};

}

**Declaring Constructors, Methods, Variables In An Enum:**

**Constructors:**

Because, enum is a special kind of class, you can do more than just list the enumerated constant values. You can add constructors, instance variables, methods and something really strange known as a constant specific class body.

For instance, consider the following example:

For instance, in your coffee shop application, you might want to restrict your coffee size size selections to BIG, HUGE, and OVERWHELMING. If you let and order for a **LARGE** or **GRANDE** slip in, it might cause an error.

Now, if a java application handles this, you need to declare an enum (declare an enum when all the possible values of a variable is known at the compile time).   
  
Hence, you could declare an enum like the following:

**enum CoffeeSize{BIG, HUGE, OVERWHEELMING};**

Now, suppose, you need more. Imagine people want to know the actual size, in ounces, that map to the three CoffeeSize constants. Now, you could make some kind of lookup table using some other data structures. But that will be a poor design and hard to maintain. The simplest way to treat your enum values as objects, each of which can have its own instance variables and own values.

enum CoffeeSize

{

BIG(8), HUGE(10), OVERWHELMING(16);

private int ounces;

CoffeeSize(int ounces)

{

this.ounces=ounces;

}

public int getOunces()

{

return ounces;

}

}

public class Coffee

{

CoffeeSize size;

public static void main(String[] args)

{

Coffee drink1=new Coffee();

drink1.size=CoffeeSize.BIG;

System.out.println("In "+drink1.size+" we get "+drink1.size.getOunces());

}

}

**It will produce:**

In BIG we get 8

**Important Points About Enum Constructor:**

* You can never invoke an enum constructor directly. The enum constructor is invoked automatically, with the arguments you defined after the constant value.
* You can define more than one argument to the constructor, and you can overload the enum constructors. Just as you overload a normal class constructor.
* This feature is also called **Enum with Customized Value in Java**

**Methods:**

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

**Variables:**

enum CoffeeSize

{

BIG(8), HUGE(10), OVERWHELMING(16);

private int ounces;

CoffeeSize(int ounces)

{

this.ounces=ounces;

}

public int getOunces()

{

return ounces;

}

}

Note the variable ounces with in CoffeeSize.

**More Important Points About Enum In General:**

* All java enum implicitly extends java.lang.Enum class that extends Object class and implements Serializable and Comparable interfaces. So we can’t extend any class in enum. **(Because, multiple inheritances are not allowed in enum)**
* Since enum is a keyword, we can’t end package name with it, for example com.dev.enum is not a valid package name. You cannot import that to some other class with other package to access the functionality of the defined enum.
* Enum can implement interfaces.
* Enum constructors are always private. (probably, implicitly. That’s why Enum constructors cannot be called from outside the definition of enum)
* We can’t create instance of enum using new operator.

Check the following example:

enum CoffeeSize

{

BIG(8), HUGE(10), OVERWHELMING(16);

private int ounces;

CoffeeSize(int ounces)

{

this.ounces=ounces;

}

public int getOunces()

{

return ounces;

}

}

**BIG(8) here, constructor is called implicitly. It is not called in the manner:**

**CoffeeSize BIG=new CoffeeSize(8)**

* enum constants are implicitly static and final. Disassembles a enum definition with javap and you will see:

For instance, if you disassembles the enum definition:

enum CoffeeSize{BIG, HUGE, OVERWHELMING}

With javap, it will show you the following:

final class CoffeeSize extends java.lang.Enum<CoffeeSize> {

public static final CoffeeSize BIG;

public static final CoffeeSize HUGE;

public static final CoffeeSize OVERWHELMING;

public static CoffeeSize[] values();

public static CoffeeSize valueOf(java.lang.String);

static {};

}

* Since enum constants are final, we can safely compare them using “==” and equals() methods. Both will have the same result.