Java Enumerations

Enumerations was added to Java language in JDK5. **Enumeration** means a list of named constant. In Java, enumeration defines a class type. An Enumeration can have constructors, methods and instance variables. It is created using **enum** keyword. Each enumeration constant is *public*, *static* and *final* by default. Even though enumeration defines a class type and have constructors, you do not instantiate an **enum** using **new**. Enumeration variables are used and declared in much a same way as you do a primitive variable.

How to Define and Use an Enumeration

1. An enumeration can be defined simply by creating a list of enum variable. Let us take an example for list of Subject variable, with different subjects in the list.

```
2. //Enumeration defined
3. enum Subject
4. {
5.       Java, Cpp, C, Dbms
}
```

Copy

6. Identifiers Java, Cpp, C and Dbms are called **enumeration constants**. These are public, static and final by default.
7. Variables of Enumeration can be defined directly without any **new** keyword.

Example of Enumeration

Lets create an example to define an enumeration and access its constant by using enum reference variable.

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


class Demo
{
     public static void main(String args[])
     {
          WeekDays wk;              //wk is an enumeration variable of
type WeekDays
```

```
            wk = WeekDays.SUNDAY;     //wk can be assigned only the
constants defined under enum type Weekdays

            System.out.println("Today is "+wk);

      }

}
```

Copy

```
Today is SUNDAY
```

Example of Enumeration using switch statement

Enumeration can be used in switch case to create decision making application, here we have created an enum of restaurants that can be used to pick user choice restaurant.

```
enum Restaurants {

dominos, kfc, pizzahut, paninos, burgerking

}

class Test {

public static void main(String args[])

{

Restaurants r;

r = Restaurants.paninos;

switch(r) { //The name of the enumertion constants are used without
their enumeration

type name i.e only r, not Restaurants.r

case dominos: //only constants defined under enum Restaurants can be
used

System.out.println("I AM " + r.dominos);

break;

case kfc:

System.out.println("I AM " + r.kfc);

break;

case pizzahut:

System.out.println("I AM " + r.pizzahut);

break;

case paninos:
```

```
System.out.println("I AM " + r.paninos);

break;

case burgerking:

System.out.println("I AM " + r.burgerking);

break;

}

}

}
```

Copy

```
I AM PANINOS
```

Example : Enumeration in If-Else

Enumeration can be used in if statement to compare a value with some predefined constants. Here we are using an enumeration with if else statement.

```
enum WeekDays{

     SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}


class Demo {

     public static void main(String args[])

     {

          WeekDays weekDays = WeekDays.WEDNESDAY;


          if(weekDays == WeekDays.SUNDAY || weekDays ==
WeekDays.SATURDAY)

               System.out.println("It is Weekend");

          else

               System.out.println("It is weekday: "+weekDays);


     }

}
```

Copy

```
It is weekday: WEDNESDAY
```

Example: Traversing Enumeration Elements

We can iterate enumeration elements by calling its static method values(). This method returns an array of all the enum constants that further can be iterate using for loop. See the below example.

```
enum WeekDays{

      SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}


class Demo {

      public static void main(String args[])

      {

            WeekDays[] weekDays = WeekDays.values();


            for(WeekDays weekday : weekDays ){


                  System.out.println(weekday);


            }

      }

}
```

Copy

```
SUNDAY

MONDAY

TUESDAY

WEDNESDAY

THURSDAY

FRIDAY

SATURDAY
```

Values() and ValueOf() method

All the enumerations predefined methods **values()** and **valueOf()**. values() method returns an array of enum-type containing all the enumeration constants in it. Its general form is,

```
public static enum-type[ ] values()
```

Copy

valueOf() method is used to return the enumeration constant whose value is equal to the string passed in as argument while calling this method. It's general form is,

```
public static enum-type valueOf (String str)
```

Copy

Example of enumeration using values() and valueOf() methods:

Value and valueOf both are static methods of enum type and can be used to access enum elements. Here we are using both the methods to access the enum elements.

```
enum Restaurants {

     DOMINOS, KFC, PIZZAHUT, PANINOS, BURGERKING

}

class Demo {

     public static void main(String args[])

     {

          Restaurants r;

          System.out.println("All constants of enum type Restaurants
are:");

          Restaurants  rArray[] = Restaurants.values(); //returns an
array of constants of type Restaurants

          for(Restaurants  a : rArray) //using foreach loop

               System.out.println(a);



          r = Restaurants.valueOf("DOMINOS");

          System.out.println("It is " + r);

     }

}
```

Copy

```
All constants of enum type Restaurants are:

DOMINOS

KFC

PIZZAHUT

PANINOS

BURGERKING

It is DOMINOS
```

Points to remember about Enumerations

1. Enumerations are of class type, and have all the capabilities that a Java class has.
2. Enumerations can have Constructors, instance Variables, methods and can even implement Interfaces.
3. Enumerations are not instantiated using **new** keyword.
4. All Enumerations by default inherit **java.lang.Enum** class.

Enumeration with Constructor, instance variable and Method

Enumeration is similar to class except it cannot be instantiated. It can have methods, constructors, variables etc. here in this example, we are creating constructor and method in the enum and accessing its constants value using these.

```
enum Student

{

     John(11), Bella(10), Sam(13), Viraaj(9);

     private int age;                    //variable defined in enum
Student

     int getage() { return age; }  //method defined in enum Student

     private Student(int age)  //constructor defined in enum Student

     {

          this.age= age;

     }

}

class Demo

{

     public static void main( String args[] )

     {

          Student S;
```

```
          System.out.println("Age of Viraaj is "
+Student.Viraaj.getage()+ " years");

      }

}
```

Copy

```
Age of Viraaj is 9 years
```

In this example as soon as we declare an enum variable(*Student S*), the constructor is called once, and it initializes age for every enumeration constant with values specified with them in parenthesis.