Chapter 17. Enumerated Types

1) Introduction

➔ An *enum type* is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week. Because they are constants, the names of an enum type's fields are in uppercase letters.

➔ In the Java programming language, you define an enum type by using the enum keyword. For example, you would specify a days-of-the-week enum type as:

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

➔ Example

```
package enumerationTypes;

//An enumeration of apple varieties.
enum Apple
{
JONATHAN, GOLDENDEL, REDDEL, WINESAP, CORTLAND
}

class EnumDemo
{
  public static void main(String args[])
  {
  Apple ap;
  ap = Apple.REDDEL;

  // Output an enum value.
  System.out.println("Value of ap:  " + ap);
  System.out.println();
  ap = Apple.GOLDENDEL;

  // Compare two enum values.
  if(ap == Apple.GOLDENDEL)
  System.out.println("ap contains GoldenDel.\n");

  // Use an enum to control a switch statement.
        switch(ap)
        {
        case JONATHAN:
        System.out.println("Jonathan is red.");
        break;

        case GOLDENDEL:
        System.out.println("Golden Delicious is yellow.");
        break;

        case REDDEL:
        System.out.println("Red Delicious is red.");
        break;

        case WINESAP:
        System.out.println("Winesap is red.");
        break;

        case CORTLAND:
        System.out.println("Cortland is red.");
        break;
        }
        }
  }
  Output:
  Value of ap: RedDel

  ap contains GoldenDel.

  Golden Delicious is yellow.
```

2) <u>values() and valuesOf() methods</u> :

➔ All enumerations automatically contains two methods

```
public static enum-type[ ] values( )
public static enum-type valueOf(String str)
```

The **values( )** method returns an array that contains a list of the enumeration constants. The **valueOf( )** method returns the enumeration constant whose value corresponds to the string passed in *str.*

➔ <u>Example</u> :

```
package enumerationTypes;

enum Days {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THRUSDAY, FRIDAY, SATURDAY
}

public class EnumValuesCls {

  public static void main(String[] args)
  {
        Days [] daysArr = Days.values();

        System.out.println("Here is what all Days contains");
        for(Days d : daysArr)
        {
              System.out.println(d);
        }

        System.out.println();
        System.out.println("Using valueOf() method");
        System.out.println(Days.valueOf("FRIDAY"));
        //System.out.println(Days.valueOf("TODAY"));//ERROR:
```

```
    IllegalArgumentException
    }
  }
  Output:
  Here is what all Days contains
  SUNDAY
  MONDAY
  TUESDAY
  WEDNESDAY
  THRUSDAY
  FRIDAY
  SATURDAY

  Using valueOf() method
  FRIDAY
```

➔ In the above code one could directly used converted array from enum as shown below.

```
for(Days d : Days.values())
System.out.println(d);
```

3) <u>Enum as class types</u> :

➔ Enumeration types are class types in Java. They cannot instantiate with 'new' as constructors for enum is only private. Instance of enum is created when Enum constants are first called or referenced in code. Also all enum constants are implicitly static and final and cannot be changed once created. but they have other capabilities as other classes. Because of this enumeration types in Java have much capabilities which other enum types in other languages dont have. For example, you can give them constructors, add instance variables and methods, and even implement interfaces.

➔ It is important to understand that each enumeration constant is an **object** of its enumeration type. Thus, when you define a constructor for an **enum**, the constructor is called when each enumeration constant is created. Also, each enumeration constant has its own copy of any instance variables defined by the enumeration.

➔ <u>Example</u> :

```
package enumerationTypes;

//Use an enum constructor, instance variable, and method.
enum Apple1
{
  JONATHAN(10), GOLDENDEL(9), REDDEL(12), WINESAP(15), CORTLAND(8);
  private int price; // price of each apple
  //Constructor
  Apple1(int p) { price = p; }
  int getPrice() { return price; }
}

public class EnumAsClassType {
    public static void main(String[] args)
    {
          {
                Apple1 ap;

                // Display price of Winesap.
                System.out.println("Winesap costs "+ Apple1.WINESAP.getPrice() +
                " cents.\n");

                // Display all apples and prices.
                System.out.println("All apple prices:");

                for(Apple1 a : Apple1.values())
                System.out.println(a + " costs " + a.getPrice() + " cents.");
          }
    }
}
Output:
Winesap costs 15 cents.

All apple prices:
JONATHAN costs 10 cents.
GOLDENDEL costs 9 cents.
REDDEL costs 12 cents.
WINESAP costs 15 cents.
CORTLAND costs 8 cents.
```

- ➔ When the variable **ap** is declared in **main( )**, the constructor for **Apple** is called once for each constant that is specified. Notice how the arguments to the constructor are specified, by putting them inside parentheses after each constant, as shown here:

```
Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);
```

- ➔ Although the preceding example contains only one constructor, an **enum** can offer two or more overloaded forms, just as can any other class as shown below

```
// Use an enum constructor.

enum Apple1 {
JONATHAN(10), GOLDENDEL(9), REDDEL, WINESAP(15), CORTLAND(8);
private int price; // price of each apple
// Constructor
Apple1(int p) { price = p; }
// Overloaded constructor
Apple1() { price = -1; }
int getPrice() { return price; }
}
```

Notice that in this version, **RedDel** is not given an argument. This means that the default constructor is called, and **RedDel**'s price variable is given the value –1.

- ➔ Enumerations have only two restriction compare to classes. One they cannot extends other classes and second they cannot be a superclass.

4) java.lang.Enum class

- ➔ Although you can't inherit a superclass when declaring an **enum**, all enumerations automatically inherit one: **java.lang.Enum**. This class defines several methods which are available for use for all enumerations.
- ➔ One can get the contants positition in the enumeration which is called 'ordinal' value using ordinal() method. This and other methods of 'Enum' class like compareTo() and equals() has been demonstrated in the following programme.

```
package enumerationTypes;

enum Dog {LABRADOR, GERMANSHEPHERD, BULLDOG, DOBERMAN, ROTTWEILER}

public class EnumSuperCls {

  public static void main(String[] args)
  {
        Dog dog1, dog2, dog3;

        //Obtain all the values using ordinal
        System.out.println("Ordinal values for all dogs are");
        System.out.println();
        for(Dog d : Dog.values())
              System.out.println(d+"'s ordinal value is : "+d.ordinal());

        dog1 = Dog.BULLDOG;
        dog2 = Dog.LABRADOR;
        dog3 = Dog.BULLDOG;

        //Demonstrate compareTo() and equals()

        System.out.println();

        if(dog1.compareTo(dog2) < 0)//2-0
              System.out.println(dog1+" comes before "+ dog2);

        if(dog1.compareTo(dog2) > 0)//2-0
              System.out.println(dog2+" comes before "+ dog1);

        if(dog1.compareTo(dog3) == 0)//2-2
              System.out.println(dog1+" equals "+ dog3);

        System.out.println();

        if(dog1.equals(dog2))
              System.out.println(dog1+" is "+dog2);
        else
              System.out.println(dog1+" isn't "+dog2);

        if(dog1.equals(dog3))
```

```
                System.out.println(dog1+" is "+dog3);
          else
                System.out.println(dog1+" isn't "+dog3);
   }

}
Output:
Ordinal values for all dogs are

LABRADOR's ordinal value is : 0
GERMANSHEPHERD's ordinal value is : 1
BULLDOG's ordinal value is : 2
DOBERMAN's ordinal value is : 3
ROTTWEILER's ordinal value is : 4

LABRADOR comes before BULLDOG
BULLDOG equals BULLDOG

BULLDOG isn't LABRADOR
BULLDOG is BULLDOG

BULLDOG
```

As we can see above dog1.compareTo(dog2) methods compare the ordinal values of dog1 and dog2 and returns positive if dog1>dog2, 0 if dog1=dog2 and negative if dog1<dog2. The other method equal() compares values contained in e1 and e2 and returns a boolean true or false. Also there is a name() method in Enum class which returns the value contained in enum reference on which it is called.