Lecture 2: Enumerated Types

Reading materials
Enum Types - Oracle tutorial http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
Enum Abstract Class - Oracle documentation http://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html

Topics Covered

1 Enumerated Types in General  2

2 (Simple) Enumerated Types in Java  2
  2.1 Enum Abstract Class  2
  2.2 Source Code  3

3 (Full) Enumerated Types in Java  4
  3.1 Source Code  5

4 Game of Life  5
  4.1 Source Code  5
1 Enumerated Types in General

Many programming languages provide means of defining enumerated types. These are usually user defined types for which the values are simply enumerated (hence the name) in the definition.

For example the type Day should have values representing the name of each day of the week: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday.

We could agree on representing the days by integer and using numbers to encode each day of the week, but there are several problems with that:

1. seeing number 5, it is not immediately clear what day of a week we are dealing with (we may be starting to count from zero or from one, we may be starting counting with Sunday, Monday or some other day);

2. integers have much larger range than the days of the week do, so what would 54 represent?

Using enumerated types provides

- more readable and self documenting code,
- allows the compiler to check for valid values within the type,
- prevent the programmer from mixing days of the week and integers.

2 (Simple) Enumerated Types in Java

The enumerated types in java use the keyword enum (in place where you normally see the keyword class). Syntax:

```java
public enum EnumeratorName {
   //enumerator definition
}
```

The simplest definitions just list the possible values. Example:

```java
public enum Day01 {
   MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;
}
```

The values can be thought of a constants so the naming convention is to use all uppercase letters. Notice also that the values are separated by commas.

2.1 Enum Abstract Class

All Java enumerators inherit automatically from Enum Abstract Class [http://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html](http://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html). This means that with a simple definition like the one above, you can take advantage of several methods inherited from the abstract class.
Example: Using `compareTo()` method. You can use the `compareTo` method with values of enumerated types. The order is defined by the order in which the values are listed in the definition. The following code uses the `compareTo` method:

```java
Day01 d1 = Day01.THURSDAY;
Day01 d2 = Day01.MONDAY;

if (d1.compareTo(d2) < 0)
    System.out.printf("%s comes before %s\n", d1.toString(), d2.toString());
else if (d1.compareTo(d2) > 0)
    System.out.printf("%s comes after %s\n", d1.toString(), d2.toString());
else
    System.out.printf("%s and %s are the same day.\n", d1.toString(), d2.toString());
```

Output (as expected):

```
THURSDAY comes after MONDAY
```

Example: Using an array of values and `equals()` method. The method `values()` returns an array containing all the values of the given enumerator type. The following code iterates through all days of the week and prints information about each day.

```java
Day01[] days = Day01.values();
for (Day01 day : days) {
    System.out.printf("Today is %10s.\n", day.toString());
    if (day.equals(Day01.THURSDAY) || day.equals(Day01.TUESDAY))
        System.out.printf("Go to Data Structures class.\n");
    else
        System.out.println("\n");
}
```

Output:

```
Today is MONDAY.
Today is TUESDAY.Go to Data Structures class.
Today is WEDNESDAY.
Today is THURSDAY.Go to Data Structures class.
Today is FRIDAY.
Today is SATURDAY.
Today is SUNDAY.
```

2.2 Source Code

Files `Day01.java`, `Day01Driver.java` contain the two examples from the previous section.
3 (Full) Enumerated Types in Java

Enumerated types can contain constructors and methods. They are in many ways similar to classes in Java with an exception of how the values are defined and created.

A more advanced version of the enumerator to represent days of the week may look as follows.

```java
public enum Day02 {
    MONDAY ("Monday"),
    TUESDAY ("Tuesday"),
    WEDNESDAY("Wednesday"),
    THURSDAY ("Thursday"),
    FRIDAY ("Friday"),
    SATURDAY ("Saturday"),
    SUNDAY ("Sunday"),
    
    String name;

    Day02(String name) {
        this.name = name;
    }

    public boolean isWeekend () {
        if (this.equals(Day02.SATURDAY) || this.equals(Day02.SUNDAY))
            return true;
        else
            return false;
    }

    public boolean isWeekday () {
        return !isWeekend();
    }

    public Day02 getTomorrow (){
        switch (this) {
            case MONDAY: return TUESDAY;
            case TUESDAY: return WEDNESDAY;
            case WEDNESDAY: return THURSDAY;
            case THURSDAY: return FRIDAY;
            case FRIDAY: return SATURDAY;
            case SATURDAY: return SUNDAY;
            case SUNDAY: return MONDAY;
            default: return null;
        }
    }

    public String toString () {
        return name;
    }
}
```

The values are listed this time with constructor calls. This allows us to set the name of the day to the appropriate string that is not in all uppercase letters. The constructor can be used to specify values of data fields specific to the particular value defined by the enumerated type.

The methods are defined just as if they were defined for a class.
3.1 Source Code

Files Day02.java and Day02Driver.java provide an example of how the enumerator types with additional methods can be used.

4 Game of Life

The game of life is a mathematical zero-player game that is "played" on a rectangular infinite grid. Its outcome depends solely on the initial state of the game. The rules for the original version of the game of life are very simple.

Each location on the grid is referred to as a cell.

At any point of the game some cells are alive, some cells are dead.

The state of the next "step" is determined as follows:

- if the cell is alive and it has fewer than one neighbor that is alive, it dies (cells die of loneliness);
- if the cell is alive and it has more than three neighbors that are alive, it dies (cells die of overcrowding);
- if the cell is alive and it has two or three neighbors that are alive, it stays alive;
- if the cell is dead and it has exactly three neighbors that are alive, it is born (do not ask me why three);
- if the cell is dead and it has fewer or more than three neighbors that are alive, it stays dead.

The initial state determines the entire sequence of steps.

How would you implement a simulation of the game of life?

4.1 Source Code

Files State.java, Cell.java, GameOfLife_Console.java implement the game of life.