Enums

(and a review of switch statements)

13-Nov-18

Enumerated values

- Sometimes you want a variable that can take on only a certain listed (enumerated) set of values
- Examples:
  - dayOfWeek: SUNDAY, MONDAY, TUESDAY, …
  - month: JAN, FEB, MAR, APR, …
  - gender: MALE, FEMALE
  - title: MR, MRS, MS, DR
  - appletState: READY, RUNNING, BLOCKED, DEAD
- The values are written in all caps because they are constants
- What is the actual type of these constants?

Enumerations

- In the past, enumerations were usually represented as integer values:
  - public final int SPRING = 0;
    public final int SUMMER = 1;
    public final int FALL = 2;
    public final int WINTER = 3;
- This is a nuisance, and is error prone as well
  - season = season + 1;
  - now = WINTER; …; month = now;
- Here's the new way of doing it:
  - enum Season { WINTER, SPRING, SUMMER, FALL }

enums are classes

- An enum is actually a new type of class
  - You can declare them as inner classes or outer classes
  - You can declare variables of an enum type and get type safety and compile time checking
    - Each declared value is an instance of the enum class
    - Enums are implicitly public, static, and final
    - You can compare enums with either equals or ==
  - enums extend java.lang.Enum and implement java.lang.Comparable
    - Hence, enums can be sorted
  - Enums override toString() and provide valueOf()
  - Example:
    - Season season = Season.WINTER;
    - System.out.println(season); // prints WINTER
    - season = Season.valueOf("SPRING"); // sets season to Season.SPRING

Advantages of the new enum

- Enums provide compile-time type safety
  - int enums don't provide any type safety at all: season = 43;
- Enums provide a proper name space for the enumerated type
  - With int enums you have to prefix the constants (for example, seasonWINTER or S_WINTER) to get anything like a name space.
- Enums are robust
  - If you add, remove, or reorder constants, you must recompile, and then everything is OK again
- Enum printed values are informative
  - If you print an int enum you just see a number
- Because enums are objects, you can put them in collections
- Because enums are classes, you can add fields and methods

Enums have weird constructors

- Except for constructors, an Enum is an ordinary class
- Each name listed within an Enum is actually a call to a constructor
- Example:
  - enum Season { WINTER, SPRING, SUMMER, FALL }
  - This constructs the four named objects, using the default constructor
- Example 2:
  - public enum Coin {
    private final int value;
    Coin(int value) { this.value = value; }
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);
    }
- Enum constructors are only available within the Enum itself
  - An enumeration is supposed to be complete and unchangeable

Enums extend and inherit from Enum

- String toString() returns the name of this enum constant, as contained in the declaration
- boolean equals(Object other) returns true if the specified object is equal to this enum constant
- int compareTo(E o) compares this enum with the specified object for order; returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object
- static enum-type valueOf(String s) returns the enumerated object whose name is s
- static enum-type[] values() returns an array of the enumeration objects

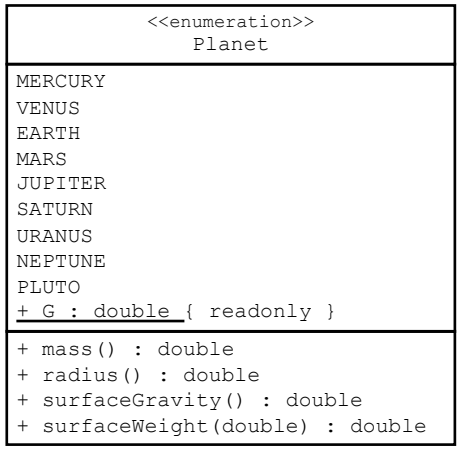
Enumeration types

```
public enum Color {RED, YELLOW, BLUE};
...
for (Color myColor : Color.values())
  System.out.println(myColor);
```

`values()` is a static method of an enumeration type returning an array

containing all the values of the enum type in the order they are declared.

Enum types - UML class diagram




```
public enum Planet {
  MERCURY (3.303e+23, 2.4397e6),
  VENUS (4.869e+24, 6.0518e6),
  EARTH (5.976e+24, 6.37814e6),
  MARS (6.421e+23, 3.3972e6),
  JUPITER (1.9e+27, 7.1492e7),
  SATURN (5.688e+26, 6.0268e7),
  URANUS (8.686e+25, 2.5559e7),
  NEPTUNE (1.024e+26, 2.4746e7),
  PLUTO (1.27e+22, 1.137e6);

  private final double mass; // in kilograms
  private final double radius; // in meters
```

```
  Planet(double mass, double radius) {
      this.mass = mass;
      this.radius = radius;
  }

  public double mass() { return mass; }
  public double radius() { return radius; }

  // universal gravitational constant (m3 kg-1 s-2)
  public static final double G = 6.67300E-11;

  public double surfaceGravity() { return G * mass /
            (radius * radius); }
  public double surfaceWeight(double otherMass) {
            return otherMass * surfaceGravity(); }
}
```

```
public enum Operation {
   PLUS { double eval(double x, double y) { return x + y; } },
   MINUS { double eval(double x, double y) { return x - y; } },
   TIMES { double eval(double x, double y) { return x * y; } },
   DIVIDE { double eval(double x, double y) { return x / y; }};

   // Do arithmetic op represented by this constant abstract
   double eval(double x, double y);
}

public static void main(String args[]) {
   double x = Double.parseDouble(args[0]);
   double y = Double.parseDouble(args[1]);
   for (Operation op : Operation.values())
       System.out.printf("%f%s%f=%f%n",x,op,y,op.eval(x,y));
}
```

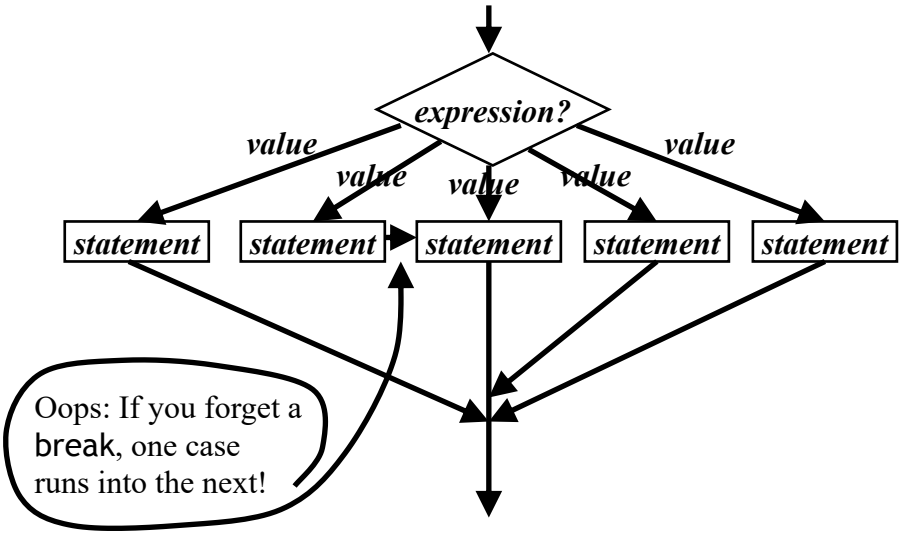
Syntax of the switch statement

- The syntax is:

```
switch (expression) {
   case value1 :
      statements ;
      break ;
   case value2 :
      statements ;
      break ;
   ...(more cases)...
   default :
      statements ;
      break ;
}
```

- The ***expression*** must yield an integer or a character
- Each ***value*** must be a literal integer or character
- Notice that colons (**:**) are used as well as semicolons
- The last statement in every case should be a break;
  - I even like to do this in the *last* case
- The default: case handles every value not otherwise handled
  - The default case is usually last, but doesn't have to be

Flowchart for switch statement



Example switch statement

```
switch (cardValue) {
    case 1:
        System.out.print("Ace");
        break;
    case 11:
        System.out.print("Jack");
        break;
    case 12:
        System.out.print("Queen");
        break;
    case 13:
        System.out.print("King");
        break;
    default:
        System.out.print(cardValue);
        break;
}
```

Enums and the switch statement

- switch statements can now work with enums
  - The switch variable evaluates to some enum value
  - The values for each case must (as always) be constants
- switch (***variable***) { case ***constant***: ...; }
  - In the switch constants, do not give the class name—that is, you *must* say case SUMMER:, *not* case Season.SUMMER:
- It's still a very good idea to include a default case

Example enum and switch

```
public void tellItLikeItIs(DayOfWeek day) {
    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;
    }
}
```

Source: http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html

Value-specific enum methods

```
// These are the the opcodes that our stack machine can execute.
abstract static enum Opcode {
  PUSH(1),
  ADD(0),
  BEZ(1); // Remember the required semicolon after last enum value
  int numOperands;
  Opcode(int numOperands) { this.numOperands = numOperands; }

  public void perform(StackMachine machine, int[] operands) {
    switch(this) {
       case PUSH: machine.push(operands[0]); break;
       case ADD: machine.push(machine.pop( ) + machine.pop( )); break;
       case BEZ: if (machine.pop( ) == 0) machine.setPC(operands[0]); break;
       default: throw new AssertionError( );
    }
 }
}
```

From: http://snipplr.com/view/433/valuespecific-class-bodies-in-an-enum/