enums

Introduction

A class Days maintains information about the day of the year. One property of a day is the season in which it occurs —for example, in the northern hemisphere, Christmas is decidedly in the winter. You might have a method m that has the season as a parameter, so you might use 0, 1, 2, and 3 for the four seasons and write something like this:

```
/**  … s is 0, 1, 2, or 3 for spring, summer, fall, or winter. …*/
public void m(…, int s, …) { … }
```

This is poor programming. Who can remember what integer represents what season? When you see a call m(…, 2, ..), how can you remember what 2 means? And *any* int can be passed by mistake as an argument in call.

Better is to declare four constants at the top of class Days:

```
/** Constants representing the seasons. */
public static final int SPRING= 0;        public static final int SUMMER= 1;
public static final int FALL= 2;          public static final int WINTER= 3;
```

and then make it clear that the names Days.SPRING, Days.SUMMER, … are to be used for the seasons.

This is better, but it still has problems. Printing one of the constants is uninformative —all you get is an integer. Procedure m's parameter s is still an int, and any int value can be passed as an argument for parameter s.

To get around these problems, Java has a feature called the *enumeration type*, or *enum*, which solves these problems. We now introduce the enum and its basic properties.

The basic enum

Place this declaration in class Days:

```
public enum Season {SPRING, SUMMER, FALL, WINTER};
```

Season is a class. It has four constants as shown to the right, which are not ints but (pointers to) objects of the class. No other objects of the class can be created. Class Season is implicitly static; we can insert keyword static, but we don't have to. Here are important points.

Constants of class Season
Season.SPRING
Season.SUMMER
Season.FALL
Season.WINTER

1. Our method m will now look like this:

```
/**  … s is the season. …*/
public void m(…, Season s, …) { … }
```

2. The convention is to use capital letters for the names of the constants.

3.Within the method, to see whether s is WINTER, use an if-condition (use == and not function equals).

```
if (s == Season.WINTER) …
```

4. Each object of class Season has a toString() function, which returns its name. For example, the statement

```
System.out println(Season.SUMMER);
```

prints the characters SUMMER.

5. Function values() of class Season returns a Season[] that contains the four constants, in the order they appear in the declaration. This array can be used in a foreach loop to process each constant. For example, the following loop prints: "SPRING SUMMER FALL WINTER ".

```
String res= "";
for (Season se : Season.values())  res= res + se + " ";
System.out.println(res);
```

6. Java provides classes EnumSet and EnumMap to maintain sets and maps of enums. Use them instead of HashSet and HashMap. Since a set of constants of class Season has at most 4 elements, EnumSet implements the set in one variable of type long, with each constant represented by a 1-bit "flag". All basic operations run in constant time.

7. Document AdvancedEnums.pdf shows how to use a switch statement over the constants of an enum.

enums

Below, we provide a more complete, realistic example: implementing a deck of playing cards. More advanced features of enums are explained in document AdvancedEnums.pdf.

Each instance of class Card implements a card of a conventional deck of cards. There are two enum classes (arrows (1) and (2)), one for the rank of the card and the other for the suit. Both are declared static, since they don't refer to other components of class Card.

Fields rank and suit (arrow (3)) have been made public since they are final and cannot be changed. This obviates the need for getter methods.

Static method newDeck returns a new deck of cards. Look at the two for-each loops and the two calls on values() to iterate through the suits and ranks in building the deck.

Class CardGame is introduced just to show you how class Card, with its two enums, can be used. It doesn't refer directly to the enums.

Note the use of procedure Collections. shuffle(), which randomly swaps around the values of deck.

The number of players doesn't change, but the cards they hold may change. Therefore each hand is an ArrayList of Cards, and the hands are kept in an array, one for each player 0…p-1.

```
import java.util.*;

/** An instance is one card, with a suit and a rank. */
public class Card {
   public static enum Rank {TWO, THREE, FOUR,          <-(1)
                FIVE, SIX,  SEVEN, EIGHT, NINE,
                TEN, JACK, QUEEN, KING, ACE}

   public static enum Suit {CLUBS, DIAMONDS,           <-(2)
                            HEARTS, SPADES}

   public final Rank rank;  // the rank of this card    <-(3)
   public final Suit suit;     // the suit of this card

   /** Constructor: an instance with suit s and rank r */
   private Card(Suit s, Rank r) {
      suit= s;
      rank= r;
   }

   /** Return a representation of this card. */
   public @Override String toString() {
       return rank + " of " + suit;
   }

   /** Return a deck of cards. */
   public static List<Card> newDeck() {
      List<Card> deck= new ArrayList<Card>();
      for (Suit s : Suit.values())                     <-(4)
         for (Rank r : Rank.values())
            deck.add(new Card(s, r));
      return deck;
   }}
```

The last statement in the method (arrow (6)) is a loop to print the player's hands.

We ran this with 4 players and 5 cards. The first player's cards were:

[SEVEN of DIAMONDS,
QUEEN of CLUBS,
ACE of DIAMONDS,
DEUCE of DIAMONDS,
ACE of HEARTS]

If we commented out the shuffler in of the deck (arrow (5)), the first player's hands are what one expects:

[DEUCE of CLUBS,
SIX of CLUBS,
TEN of CLUBS,
ACE of CLUBS,
FIVE of DIAMONDS]

```
import java.util.*;

public class CardGame {
   /** Shuffle a new deck of cards, deal d cards to p players,
     * and print the hands. */
   public static void shuffleAndDeal(int d, int p) {
      List<Card> deck= Card.newDeck(); // the deck of cards.
      Collections.shuffle(deck);                       <-(5)

      // each element of hands will contain d cards
      ArrayList<Card>[] hands= new ArrayList[p];

      for (int player= 0; player < p; player= player+1)
         hands[player]= new ArrayList<Card>();

      for (int k= 1; k <= d; k= k+1) {
         // Deal a card to each player
         for (int player= 0; player < p; player= player+1) {
            Card card= deck.remove(0);
            hands[player].add(card);
         }
      }

      for (int player= 0; player < p; player= player+1)  <-(6)
         System.out.println(hands[player]);
   }}
```