Advanced enums

An enum is a class

View the enum declaration given below as syntactic sugar for the declaration to the right, although Java will not let you directly extend Enum yourself in this fashion.

```java
public enum Coin {PENNY, NICKEL}
```

Objects PENNY and NICKEL are the two enum constants of this class. They are the only Coin objects that can ever be created. It is a syntactic error to use the new expression `new Coin(…)`.

A constructor with no parameters and no statement in its body is included.

Functions `values()` and `valueOf()` are automatically inserted into the class. In discussions below, we won’t include their specs or bodies.

Declaring fields and methods in class Coin

You can declare fields and methods in class Coin. Just place a semicolon after the last constant in the enum declaration and then write the fields and methods.

As an example, we insert field value to contain the coin’s value and a constructor that initializes this field. Here’s the new enum; note that the constructor cannot have an access modifier.

```java
public enum Coin {PENNY, NICKEL;
    private int value; // value of coin
    /** Constructor: instance with value v. */
    Coin(int v) {value= v;}
}
```

This declaration is transformed into class Coin shown to the right above. But now we have a problem, a syntax error. Because we declared a constructor with 1 parameter, the constructor with no parameters was not included. But the new Coin() expressions have 0 parameters.

How do we indicate the arguments of the new-expressions? We place them on the constants PENNY and NICKEL, thus declaring the class as shown below; this declaration is transformed into the class to the right.

```java
public enum Coin {PENNY(1), NICKEL(5);
    private int value;
    /** Constructor: instance with value v. */
    Coin(int v) {value= v;}
}
```

One can place as many arguments as one likes on the names of the constants. There must be appropriate constructor with the appropriate parameter types.

That’s how one adds fields and methods to an enum. The next page provides a complete example, illustrating a few more points.

Below, we give a complete declaration of enum Coin. It has a few more interesting points, which we now explain.
Advanced enums

1. With each coin (e.g. Penny), we give an argument for a call on the constructor (e.g. 1). Note that the constructor stores its parameter in field value (arrows (1)).

2. Function toString should print not only the coin and its value but also its color. Therefore, we have declared a static nested class CoinColor, another enum (arrow (2)).

3. We then provide static function color, which returns the color of a Coin (arrow (3)). The body of function color shows you how to write a switch statement based on an object of an enum, in this case, on Coin c. (arrow (4)).

4. Function toString() in superclass Enum returns the name of the constant — thus, in object Penny it returns the String “Penny”. You see this superclass function being called in function toString() (arrow (5)).

5. Function toString is overridden so that it gives all information of the constant: its name (but changed to lower case), its color, and its value (arrow (6)).

6. Execution of the following loop produces the output in the box on the right. Note how it calls static function Coin.values(), so you know that that function is declared in class Coin. A foreach loop makes it easy to process all the constants of class Coin.

```java
for (Coin c : Coin.values())
    System.out.println(c);
```

```
penny: COPPER, worth 1 cents
nickel: NICKEL, worth 5 cents
dime: SILVER, worth 10 cents
quarter: SILVER, worth 25 cents
```