UEF4.3 Object Oriented Programming

Mrs Sadeg and Mrs Bousbia
s_sadeg@esi.dz, n_bousbia@esi.dz
Ecole Nationale Supérieure d'Informatique
(ESI

Enum Types

Enum Type

Definition

- 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.

- Enumerations are used when we need to represent a fixed set of values like months, days, colors, deck of cards, etc.

Example: The variable CardinalPoint can take the values (NORTH, SOUTH, EAST, WEST).

Enum Type

Declaration

In Java, an enumeration is declared like a class but using the enum keyword instead of class.

Example:

```
public enum Season { WINTER, SPRING, SUMMER, AUTUMN }
```

Note: The fields of an enumerated type are written in uppercase letters because they are constants, but this is not mandatory.

Enum Type

Specific features

In Java, an enumeration is a special kind of class.

- It extends the Enum class, which itself extends the Object class, like all Java classes.
- The values of an enumeration are the only possible instances of this class. They are final (immutable) instances.
  - In the previous example, Season has four constant instances: Season.WINTER, Season.SPRING, Season.SUMMER, and Season.AUTUMN.
- An enumeration can have constructors, methods, fields, etc.

Enum methods (1)

- All enumerated types inherit from java.lang.Enum. They therefore inherit its methods:
  - int ordinal(): returns their position in order of declaration (starting from zero)
  - String name(): returns their name

Example:

```
Season season= Season.SPRING;
System.out.println("the Season " + season.name() + "
is at position: " + season.ordinal() ) ;
//displays
The Season SPRING is at position: 1
```

Enum methods (2)

All enum types also inherit Object methods overriden in java.lang.Enum

1/ equals: allows to test the equality of two values (instances) of an enumeration. We can also use the == operator, which compares addresses (because instances are constant).

2/ compareTo: It is used to order the values of an enumeration based on the order in which they are declared.

```
System.out.println(Season.SPRING.equals(Season.SPRING));
System.out.println(Season.SPRING == Season.SPRING);
System.out.println(Season.SPRING.compareTo(Season.WINTER));
System.out.println(Season.SPRING.compareTo(Season.SUMMER));
// Output:
True,
True,
1,
-1
```

Enum methods (3)

All enum types also have two static methods generated by the compiler.

1/ EnumType[] values(): returns a copy of an array containing all the values of the enumeration in the order of declaration.

```
Season seasons[] = Season.values();
System.out.println(Arrays.toString(seasons));
// Output:
[WINTER, SPRING, SUMMER, AUTUMN]
```

2/ EnumType valueOf (String name): returns the enumerated value from its string representation.

```
Season season = Season.valueOf("WINTER");
// season takes the value Season.WINTER
```

Using an enum type

in an "if" statement

We can compare a reference to an enumeration object with one of the possible values.

Example

```
public enum Color {GREEN, BLUE, RED}
……
Color color;
……
if (color == Color.GREEN) {……}
else if (color == Color.BLUE) {……}
else if (color == Color.RED) {……}
```

Using an enum type

in a switch

If an enumeration contains many constants, it is more appropriate to use a switch statement.

Example

```
public enum Level { FIRST_CPI, SECOND_CPI, FIRST_CS, SECOND_CS,
THIRD_CS }
……
Level level;
……
switch (level) {
case FIRST_CPI: ........ ; break;
case SECOND_CPI: ........; break;
case FIRST_CS: .........; break;
case SECOND_CS: .........; break;
case THIRD_CS: ......... ; break;
```

Iterating over an Enum

To iterate over the values of an enumeration, we use a foreach loop and an array containing the enumeration values, returned by the values() method.

Example:

```
for (Season s : Season.values())
   System.out.println(s.toString());
// output
 WINTER
 SPRING
 SUMMER
 AUTUMN
```

Add attributes and Methods to an Enumeration (1)

1) Methods: It is possible to define methods inside an enumeration. Note the specific syntax.

```
enum Season { WINTER, SPRING, SUMMER, AUTUMN;
public void display()
{        System.out.println(toString()); }
}
public class EnumTest {
public static void main(String[] args) {
for (Season s : Season.values())
   s.display();  }
}
```

```
//Output

WINTER
SPRING
SUMMER
AUTUMN
```

Add attributes and Methods to an Enumeration (2)

2) Constructors: An enumeration can have one or more constructors.

- It is called when the constant objects of the enumerated type are instantiated.
- It is used to associate a property that takes a specific value for each of its instances.
- If it requires arguments, they must be specified after the constant name.
- The constructors of an enum are implicitly private (even if the private access modifier is not specified). Therefore, they do not accept the public or protected access modifiers.

Add attributes and Methods to an Enumeration (3)

```
enum Season {
WINTER("rainy"), SPRING("sunny"), SUMMER("hot"), AUTUMN("variable"
private String weather;
private Season(String weather) {
this.weather = weather;
}
public String getWeather() { return this.weather;}
public void setWeather(String weather) { this.weather = weather; }
}

public class EnumTest {
public static void main(String[] args) {
Season.WINTER.setWeather("cold");
for (Season s : Season.values())
System.out.println(s.getWeather());
}
}
```

```
// output
cold
sunny
hot
variable
```

Enumerations and Inheritance

- Only enum types inherit from java.lang.Enum.
- A class cannot inherit from Enum.
- A class cannot inherit from an enumeration.
- An enumeration cannot inherit from a class.

```
public class A extends Enum { } // error
public class A extends Saison { } // error
public enum Saison extends A { } // error
public enum Meteo extends Saison { } // error
```

Enumerations and interfaces

- Enum types can implement interfaces just like classes

```
interface Describable {
String getDescription();
}
enum Season implements Describable {
WINTER("Cold and snowy"),
SPRING("Mild and blooming"),
SUMMER("Hot and sunny"),
AUTUMN("Cool and windy");
private String description;
private Season(String description){
this.description = description;}
public String getDescription() {return description;}
}
public class EnumTest {
public static void main(String[] args) {
for (Season s : Season.values()) {
System.out.println(s + ": " + s.getDescription());}}
}
```

```
// output
WINTER: Cold and snowy
SPRING: Mild and
blooming
SUMMER: Hot and sunny
AUTUMN: Cool and windy
```