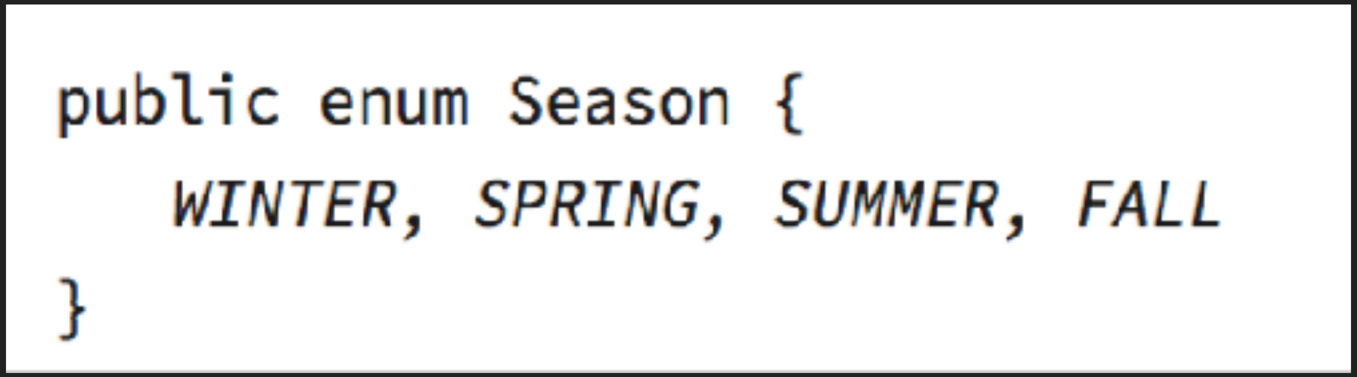
ENUMERATION

ENUMS ?

- An *enum* is a class that represents an enumeration, meaning that represents a **FIXED** set of constants and can't be reassigned.
- Enums are type-safe constructs
- Java 5 introduced the data type *enum*.



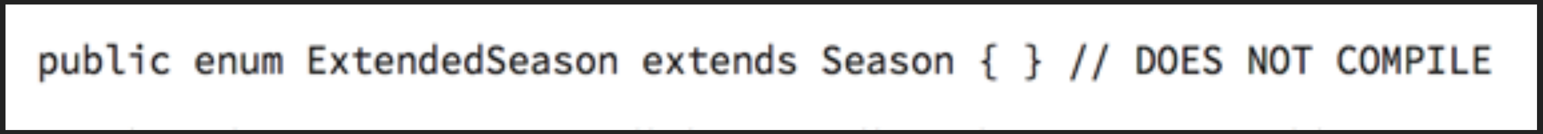
```
public enum Season {
   WINTER, SPRING, SUMMER, FALL
}
```

RULE

- You cannot create an instance of an enum by using the **new** operator.
    - A constructor in an *enum* class can only be specified as private.
      Enums are not allowed to have a public constructor, or the compiler will complain with following message: "Illegal modifier for the enum constructor; only private is permitted”.
- An instance of an enum is created when the enum is first referenced.
- Enums are singletons.

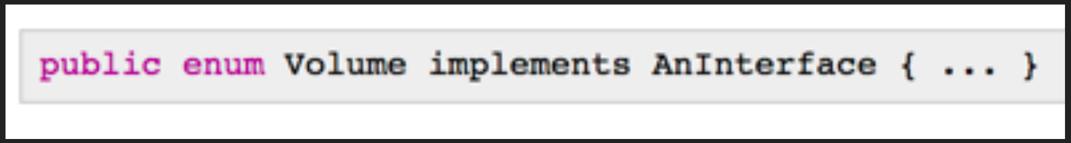
RULE

- Enums are <u>implicitly</u> declared public, static, and final, which means you **cannot** extend because all enums extend from java.lang.Enum



```
public enum ExtendedSeason extends Season { } // DOES NOT COMPILE
```

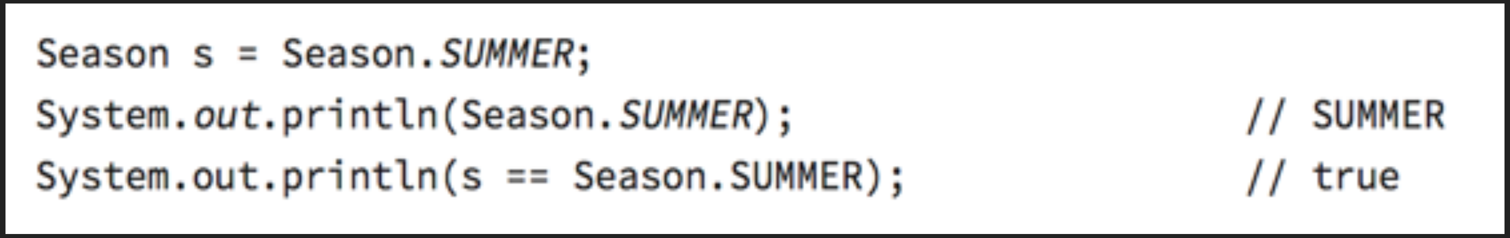
- **CAN** do is implement interfaces



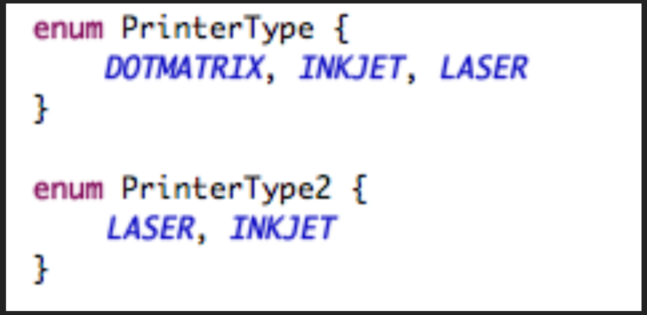
```
public enum Volume implements AnInterface { ... }
```

- Enumeration constants cannot be cloned. An attempt to do so will result in a CloneNotSupportedException.

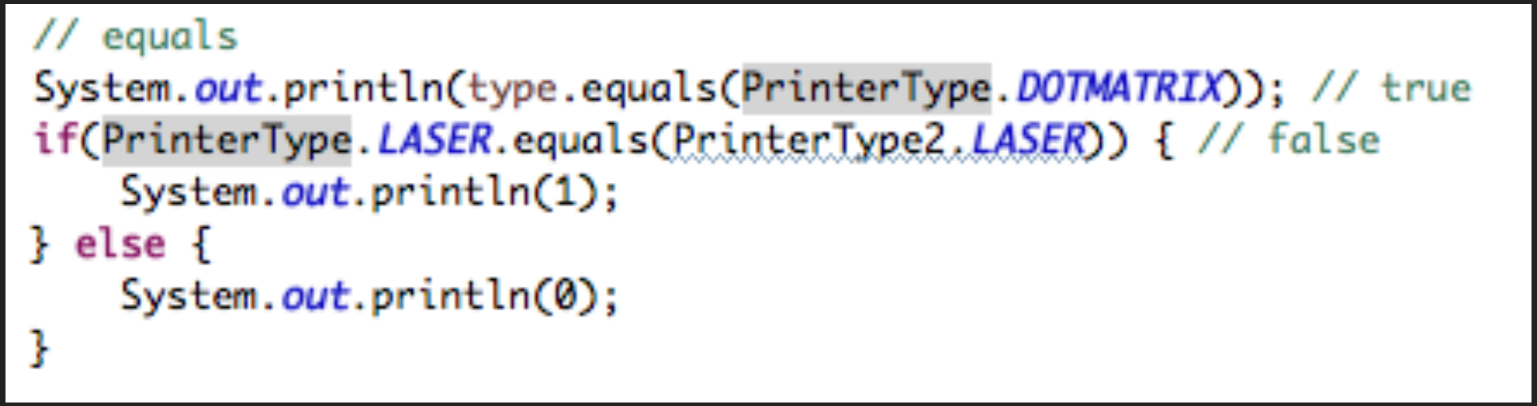
- You can compare two enumerations for equality using == operator.
- If enumeration constants are from two different enumerations, the equals() method does not return true.



```
Season s = Season.SUMMER;
System.out.println(Season.SUMMER);                          // SUMMER
System.out.println(s == Season.SUMMER);                     // true
```



```
enum PrinterType {
    DOTMATRIX, INKJET, LASER
}

enum PrinterType2 {
    LASER, INKJET
}
```

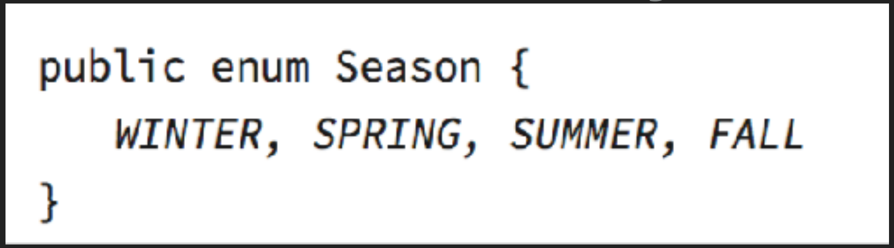


```
// equals
System.out.println(type.equals(PrinterType.DOTMATRIX)); // true
if(PrinterType.LASER.equals(PrinterType2.LASER)) { // false
    System.out.println(1);
} else {
    System.out.println(0);
}
```

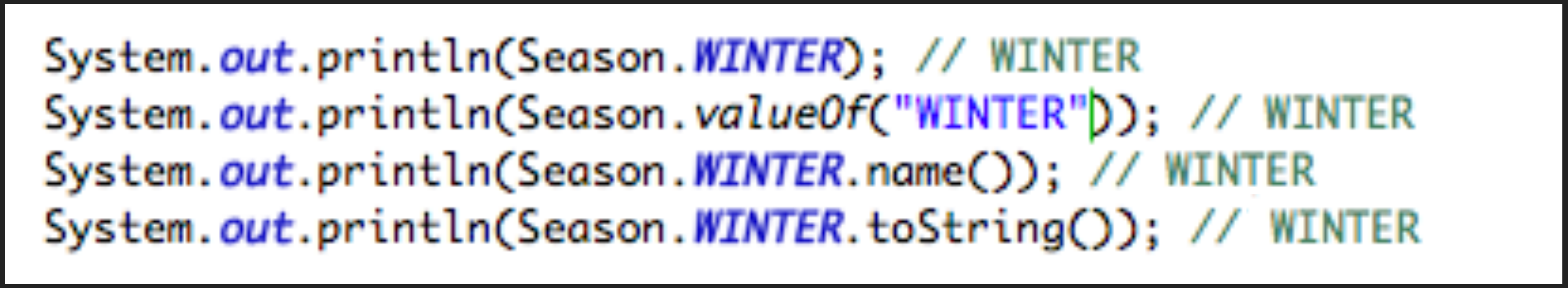
WORKING WITH ENUMS

HOW TO GET A REFERENCE OF AN ENUM.

- Reference directly



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

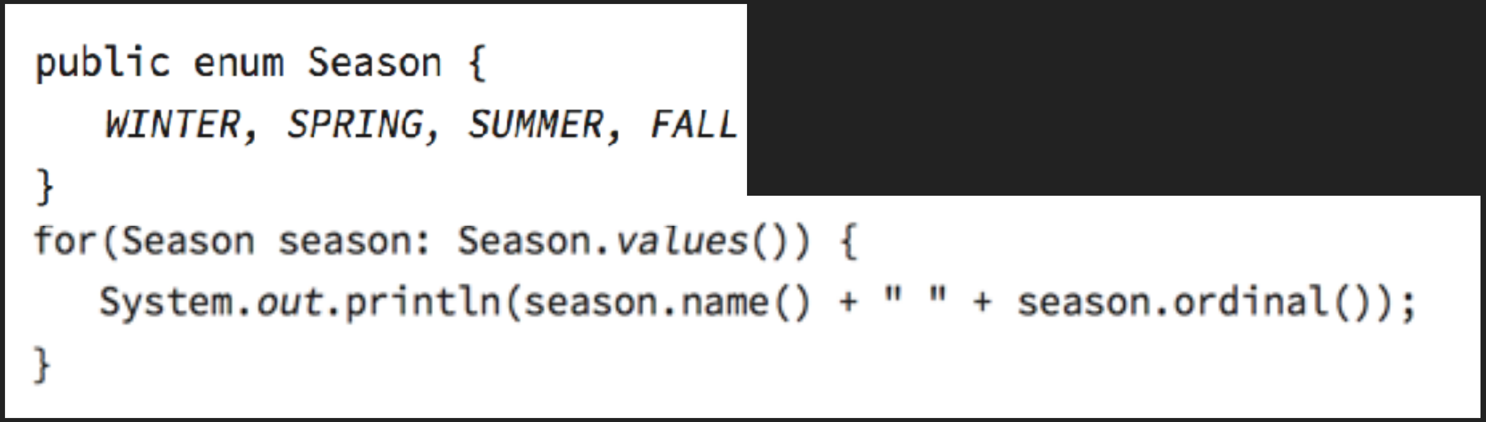


```
System.out.println(Season.WINTER); // WINTER
System.out.println(Season.valueOf("WINTER")); // WINTER
System.out.println(Season.WINTER.name()); // WINTER
System.out.println(Season.WINTER.toString()); // WINTER
```

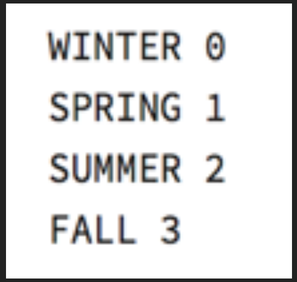
- You can apply the valueOf() and name() methods to the enum element to return the name of the enum element.

- valueOf() be careful* case sensitive

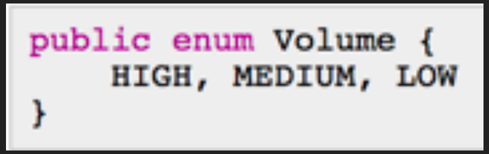
- values() that returns an array of enums in the same order in which they were declared
- ordinal() method of an enum returns its corresponding int value. Like arrays, enums are zero based. Remember that the index of an enum may change when you recompile the code and should not be used for comparison.



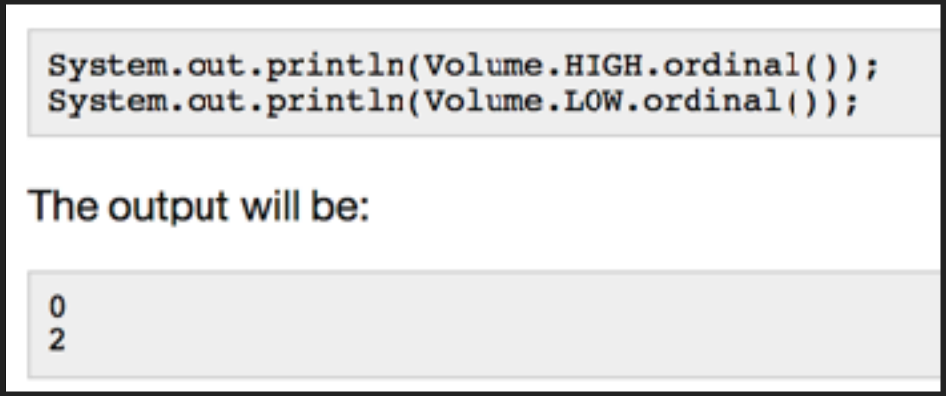
```
public enum Season {
    WINTER, SPRING, SUMMER, FALL
}
for(Season season: Season.values()) {
    System.out.println(season.name() + " " + season.ordinal());
}
```



```
WINTER 0
SPRING 1
SUMMER 2
FALL 3
```



```
public enum Volume {
    HIGH, MEDIUM, LOW
}
```

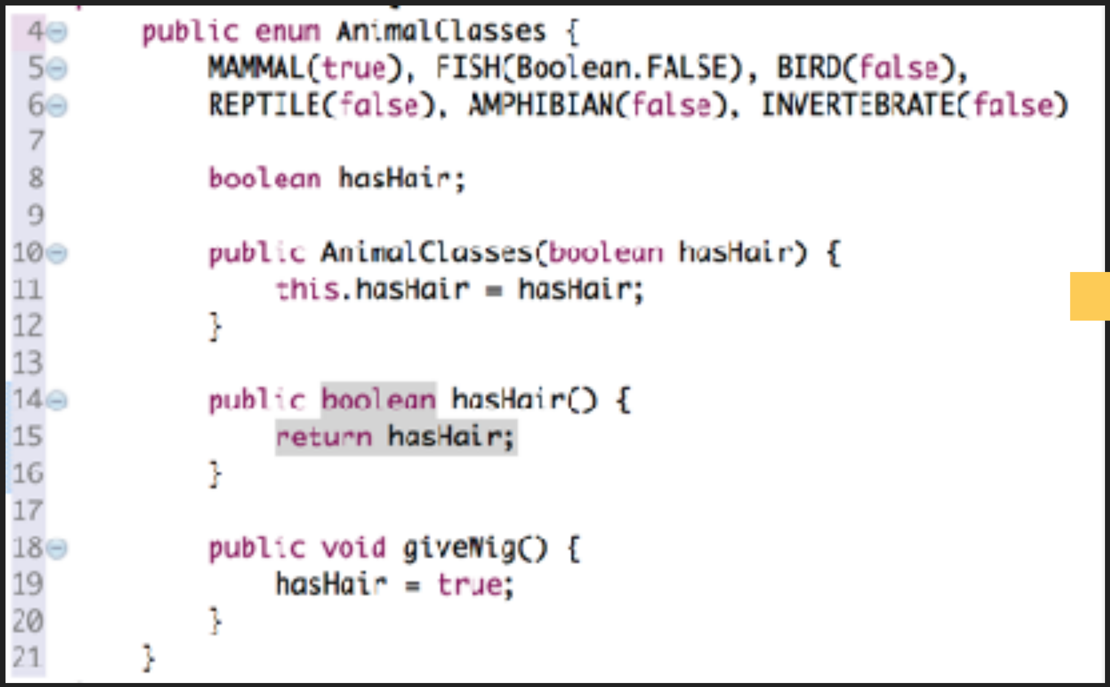


```
System.out.println(Volume.HIGH.ordinal());
System.out.println(Volume.LOW.ordinal());
```

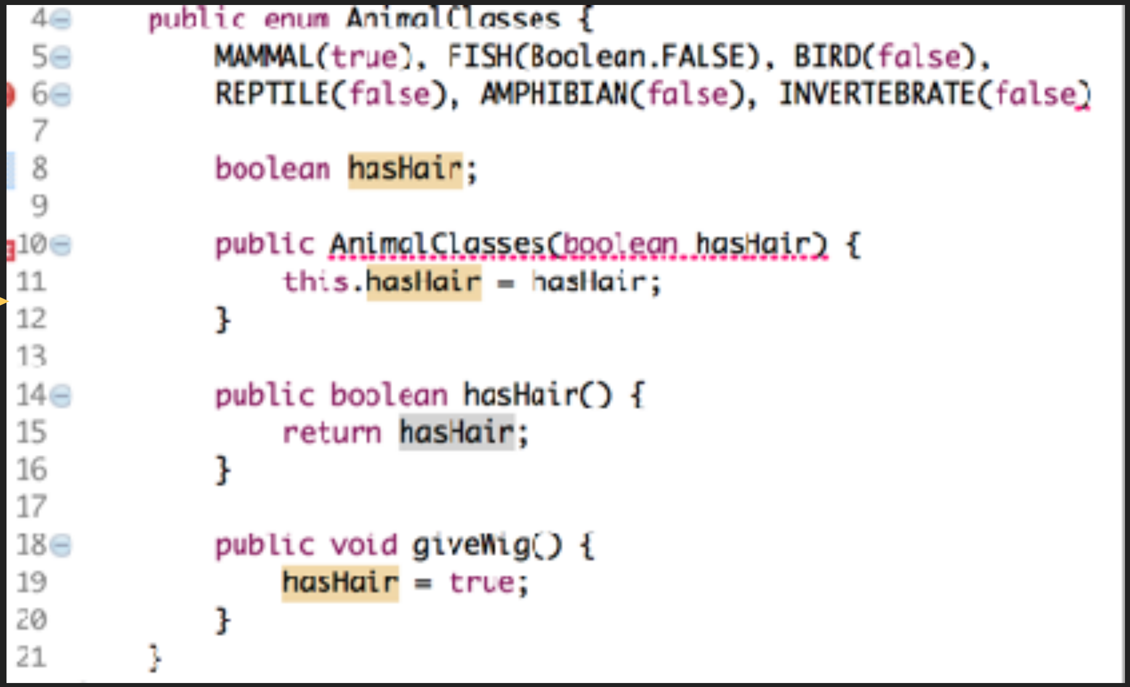
The output will be:

```
0
2
```

- notice that the values are comma-separated and because the `enum` only contains a list of values, required to have a semicolon after **the list of value.**



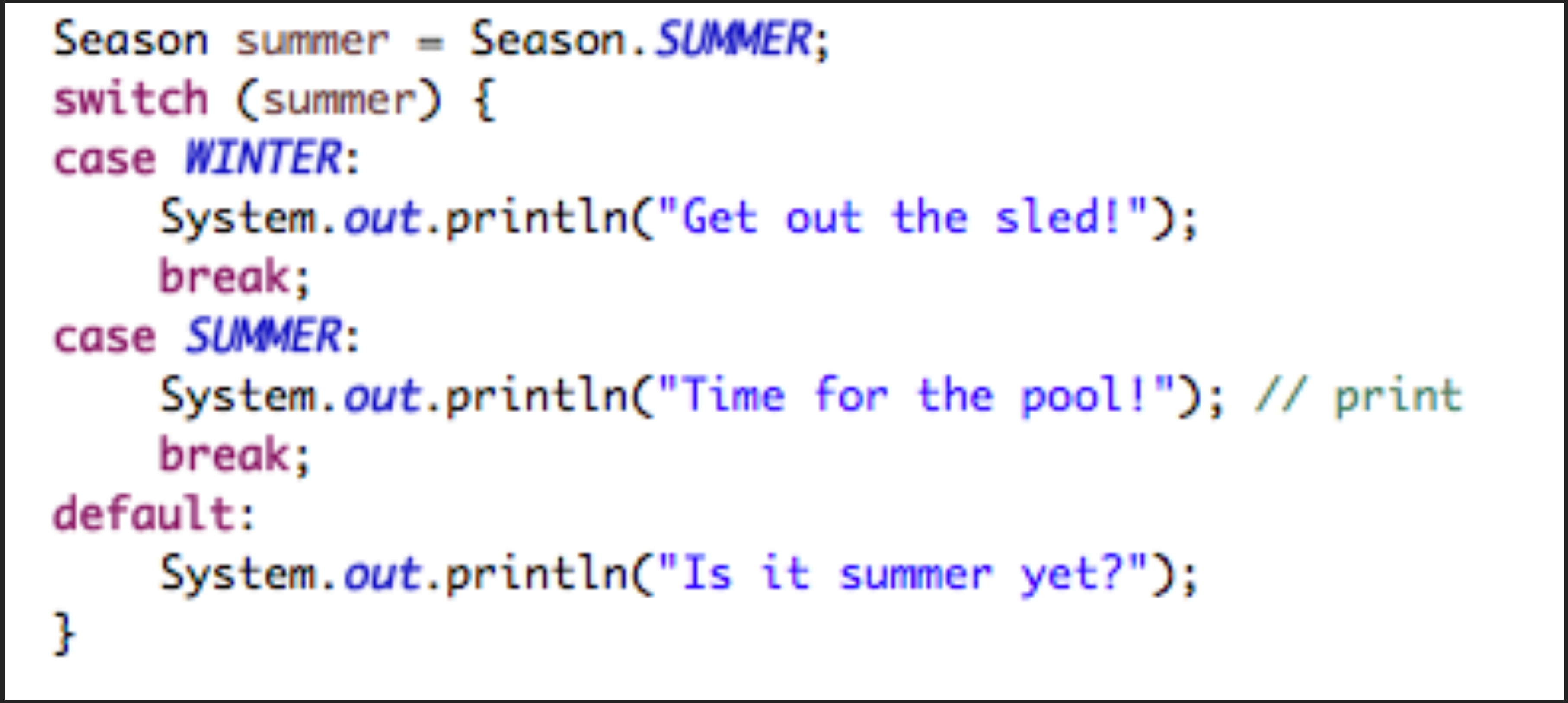
```java
public enum AnimalClasses {
    MAMMAL(true), FISH(Boolean.FALSE), BIRD(false),
    REPTILE(false), AMPHIBIAN(false), INVERTEBRATE(false)

    boolean hasHair;

    public AnimalClasses(boolean hasHair) {
        this.hasHair = hasHair;
    }

    public boolean hasHair() {
        return hasHair;
    }

    public void giveWig() {
        hasHair = true;
    }
}
```



```java
public enum AnimalClasses {
    MAMMAL(true), FISH(Boolean.FALSE), BIRD(false),
    REPTILE(false), AMPHIBIAN(false), INVERTEBRATE(false)

    boolean hasHair;

    public AnimalClasses(boolean hasHair) {
        this.hasHair = hasHair;
    }

    public boolean hasHair() {
        return hasHair;
    }

    public void giveWig() {
        hasHair = true;
    }
}
```

SWITCH-CASE

- A case statement on an enum data type must be the unqualified name of an enumeration constant.



```
Season summer = Season.SUMMER;
switch (summer) {
case WINTER:
    System.out.println("Get out the sled!");
    break;
case SUMMER:
    System.out.println("Time for the pool!"); // print
    break;
default:
    System.out.println("Is it summer yet?");
}
```

CONCLUSION

- E*num* is a type that represents a **FIXED** list of values, providing type-safe.
- Enums can define constructors, but they must be private.
  - You cannot create an instance of an enum by using the new operator.
- Enums are implicitly `public`, `static` and `final`.
- Enums can be compared against other enums using the == operator and the `equals()` method.
- Enums can be used in `switch` statements.
- Enums can implement interfaces, but they cannot extend from a class since they implicitly extend from `java.lang.Enum`.
- Enums are the easiest way to implement singletons.