Theory: Enums in Java

21 minutes 0 / 5 problems solved

Skip this topic Start practicing

312 users solved this topic. Latest completion was about 7 hours ago.

When a variable can only take one out of a small set of possible values, it's a good idea to use **enums** in a program. Enum is a special keyword short for *enumeration* that allows us to create a list of constants grouped by their content: seasons, colors, states, etc. When we store a bunch of constants in one place and handle them together, it helps us to avoid errors and makes the code look more readable and clear.

Now, let's look closer at how enums work.

§1. Defining enum

We can create our own enumeration in the same way we normally declare a class. According to the Java Code Convention, constants in an enum are written in uppercase letters. All constants should be separated with commas. Take a look at the example of enum `Season`:

```
public enum Season {
    SPRING, SUMMER, AUTUMN, WINTER // four instances
}
```

It is possible to declare enum inside the class. In this case, we don't need to use `public` modifier in the enum declaration.

In general, an enum can be considered as a class with predefined instances. Here, we have four instances of seasons `SPRING`, `SUMMER`, `AUTUMN` and `WINTER` inside the storage `Season`. If we want to extend the list of constants, we can simply add another instance in our enum: mid-winter, Australian winter, etc. Don't forget that in real life they have to make sense.

Now that we've got the idea of how to define basic enums, let's learn how to use it in a program.

§2. Methods for processing enums

Suppose that we have to write a program with an enum that displays three possible user statuses. Let's create an enum `UserStatus` with these statuses:

```
public enum UserStatus {
    PENDING, ACTIVE, BLOCKED
}
```

And now we initialize a variable of the type `UserStatus` from the previous example:

```
UserStatus active = UserStatus.ACTIVE;
```

Each enum value has a name that can be accessed by using the method `name()`:

```
System.out.println(active.name()); // ACTIVE
```

Sometimes, we may need to access an enumeration instance by its name. This can be done with the `valueOf()` method. Here's another way to initialize a variable:

```
UserStatus blocked = UserStatus.valueOf("BLOCKED"); // BLOCKED
```

An important thing to remember about this method is that it is case-sensitive. That means that if the given string doesn't exactly match any constant, we will get the IllegalArgumentException.

Current topic:

Enums in Java

Topic depends on:

✓ Switch statement (Stage 3)

✓ Iterating over arrays

✓ Instance methods (Stage 6)

Topic is required for:

Fields and methods in enum

Table of contents:

```
UserStatus blocked = UserStatus.valueOf("blocked"); // IllegalArgumentException, valueOf is case-sensitive
```

If we want to look at all enumeration's constants, we can get them in an array by using the `values()` method:

```
UserStatus[] statuses = UserStatus.values(); // [PENDING, ACTIVE, BLOCKED]
```

Another method `ordinal()` returns the ordinal position of instance among **enums**:

```
System.out.println(pending.ordinal()); // 0 (starting with 0)
System.out.println(UserStatus.BLOCKED.ordinal()); // 2
```

Although an enum is a reference type, two variables can be correctly compared by using both the `equals` method and the operator `==`.

```
System.out.println(active.equals(UserStatus.ACTIVE)); // true
System.out.println(active == UserStatus.ACTIVE); // true
```

§3. Enumerations in the switch statement

An **enum** can be successfully used in the `switch` statement. Depending on the status, our program can perform different actions indicated by the `switch` statement. In this case, it prints out different responses:

```
UserStatus status = ... // some status

switch (status) {
    case PENDING:
        System.out.println("You need to wait a little.");
        break;
    case ACTIVE:
        System.out.println("No problems, you may pass here.");
        break;
    case BLOCKED:
        System.out.println("Stop! You can't pass here.");
        break;
    default:
        System.out.println("Unsupported enum constant.");
}
```

The message that our program outputs depends on the value of the variable `status`.

§4. Iterating over enum

One of the best ways to iterate over enum is to use a `for` or a `for-each` loop. Let's apply it to our sample enum:

```
    for (UserStatus status : UserStatus.values()) {
        System.out.println(status);
    }
/* the output is
PENDING
ACTIVE
BLOCKED
*/
```

Here, we used `values()` method to return an array of enum values. This loop comes in handy when iterating over enums with a large number of constants.

§5. Conclusion

An enum is a special keyword that helps us to define named constants grouped together according to its content. By defining enums you can make the code more readable and avoid invalid values being passed in.
The number of constants in enum may be extended whenever we want. Also, you can use `name()`, `valueOf()`, `ordinal()` and `equals()` methods to process the enum. `switch` statements and `for-each` loops are widely used while working with enums in simple programs.

Now you know how to use it, so get ready for some practice!