Tutorial 8: Enums, Switch Expressions, Sealed Classes, and Records

CSE1100 - Introduction to Programming

1 Rewriting with Lambdas

You are given a writer application. It asks the user for input and then uses a thread to write to file what the user put in. The `WriteTask` is responsible for writing. Since we only use `WriteTask` at this one point in the application and do not plan to reuse it, we can rewrite to not have a separate class.

1.1 Anonymous class

Rewrite the `WriterApplication` to use an anonymous class. You can use this reference.

1.2 Lambda

The anonymous class still looks bulky and there is a lot of syntax to write for a small amount of functionality. Now rewrite the anonymous class to a lambda.

2 Threads

We have created a `DataProcessingApplication` that reads 10,000,000 numbers from `numbers.txt` and sums them. As this can take some time, we print a progress bar every time we read a number, but this makes the programme slow. Using threads, make the progress bar print in parallel every 10 milliseconds. You are allowed to change anything (including moving methods into different classes or changing access modifiers). When implemented correctly, the programme should finish in a few seconds rather than more than a minute.

3 Synchronisation

We have created an implementation of a queuing system. Students can ask questions in the form of a `Request`, which then will get added to the `Queue`. There is one problem however: when two assistants call `getNext` at the same time, they sometimes will receive the same `Request`. We also want to make sure `enqueue` and `getNext` always work in the order they are called.

Change the `Queue` class such that the `enqueue` and `getNext` methods work as we want them to, regardless of the thread that is calling them and the time it takes to add and remove requests from the queue.

Testing multithreaded behaviour is almost impossible, so you will not have to write any tests. There are already tests provided in the `QueueSyncTest` class, note that you do **not** have to understand these tests.

4 Enums and Switch

We have created a `BankingApplication`. We can make some improvements to this application to make it more easy to read. One idea is to store all possible input options in an `enum`, so we can read `case SHOW_BALANCE` instead of `case 1` (and the same for the other cases). A second improvement we can make is to change the switch statement to a switch expression.

- Create an `enum` that represents the different options of the application.
- Change the type of `option` to this enum.
- Change the switch statement in `executeOption` to a switch *expression*.

5 Sealed and Records

Given is a small calculator application. In the `expression` package, you can find one `Expression` interface and four implementations of that interface: `Constant`, `Add`, `Subtract`, and `Multiply`. These represent constant numbers (e.g. '1', '-5'), addition (e.g. '1 + 2'), subtraction (e.g. '4 - 2'), and multiplication (e.g. '4 * 2') respectively. The `Calculator` class contains one method `calculate` that will calculate the value of any expression. This application works perfectly fine, but it could still use some improvements.

5.1 Records

If you look closely, you will see that the classes `Constant`, `Add`, `Subtract`, and `Multiply` only store some constant data. We can make the implementation of those classes a lot more concise by converting them to `record`s. Convert these classes to `record`s.

5.2 Sealed

The last line of the `calculate` function should ideally never be executed. We can enforce this by making the `Expression` interface sealed. Seal the `Expression` interface and make it permit `Constant`, `Add`, `Subtract`, and `Multiply` as subclasses.

Although, the last line of the `calculate` method should now not be called, we unfortunately cannot remove it yet.

5.3 Instanceof Pattern

Another improvement we can make is to remove the casts from the `calculate` function. Whenever we do `x instanceof C` and then a cast `(C) x`, we can replace this by `x instanceof C y`, where `y` is an instance of class `C`. Remove all the casts from the calculate function.

5.4 Pattern Matching with Switch

Finally, we can replace the `if` statements with a `switch` expression. This also allows us to get rid of the `IllegalArgumentException`. Change the function such that its body is just one statement: `return switch { /* ... */ };`

Hint: You can convert an if statement of the form `if (... instanceof C y) ...` to `case C y -> ...`. Additionally, if `C` is a record with two fields `A a` and `B b`, then we can write `case C(A a, B b) -> ...` to immediately have access to `a` and `b` without needing to call `a()` and `b()`.