Session 5 (of 24)

PGR112
Objectorientert programmering

Marcus Alexander Dahl /
marcusalexander.dahl@kristiania.no

Lecture lecture = new Lecture(5);

- lecture.printGoals();
  - Go more in depth using the Scanner-class, from validating input to understanding how it actually works
  - Debugging our code to understand what happens
  - Logging information during development
  - Writing a simple test and explore that briefly

String – taking a closer look

- Strings in Java are immutable.
  You can read more here.

- Can be seen as an array of the primitive data type **char**

- Be aware, as we should use A.equals(B) and A.equals(C) instead!

```
public class StringComparison {
    public static void main(String[] args) {
        String A = "String";
        String B = "String";
        String C = new String("String");

        if (A == B) {
            System.out.println("A == B");
        }

        if (A == C) {
            System.out.println("A == C");
        }
    }
}
```

Java I/O

- Input and output of data
  - For input, we have so far used **`System.in`** as an argument to the constructor of the Scanner-class
    - Available as an import from java.util.Scanner
  - For output, we have so far used **`System.out`** and its available methods such as print, println, printf.
  - Respectively, they are the following types: *InputStream* and *PrintStream*.

Java I/O – Scanner

- Taking a closer look at the Scanner we have used to read input so far
  - It works by reading and parsing input data, creating tokens by splitting up the data using a delimeter, by default using white space (space, enter, tab, etc)

  - It has different methods which allows us to detect the state of the scanner, seeing as the Scanner-class is a state machine.
  - Our program will stop while waiting on input!
  - Remember to close our scanner, we are working with streams!

boolean	hasNext()	Returns true if this scanner has another token in its input.
boolean	hasNextInt()	Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the nextInt() method.
String	next()	Finds and returns the next complete token from this scanner.
int	nextInt()	Scans the next token of the input as an int.
String	nextLine()	Advances this scanner past the current line and returns the input that was skipped.

Java I/O – Scanner

- Let us explore by writing some code…

Java I/O – Creating a (terminal) menu

- Into IntelliJ we go!

Problems

- For a Java-program to run, it must be compiled first. The compiler makes sure that the programmer is following the rules that the compiler expects to be followed.

- The compiler will display errors during compilation and display where in our code an error / exception presented itself.

- There can still be errors when running our code and is often due to a logical error (if statements, loops, indexing, etc.) or user input.

- Where should we start when errors occur, or we are stuck?

Debugging

- A simple way of debugging is to simply use the terminal, as in printing out information to the terminal so that we as the programmer can read it

- This can get really messy with bigger code bases, and especially when we're dealing with many classes and methods.

- IntelliJ has a powerful debugger!

Debugger

- It is not only IntelliJ that has a debugger, any IDE with any dignity and respect for itself, do have a debugger.
- What does the debugger allow us to do? We can step into our code and run it step by step.
- During each step, we can examine both local variables and fields.
- A little demonstration;
- You can read more about basics of Debugger in IntelliJ here.

Writing a simple (for now) test

- Tests can make sure your code doesn’t break old functionality when new functionality is added.

- Testing alone is a big topic, but for now we will keep it simple with assertions, making sure our data stays and acts as we expect it to do.

- Let create a simple test and explore some basic concepts when setting up tests.

Before we end

- Goals for this sessions:
  - I understand more about how the Scanner works
  - I understand in simple terms that we are working with streams of data

[Bonus]: Newer Java-feature: **Records**

- Instead of using classes to store data, as in, a class being used only as data storage using its fields, we can use records instead.
  - Data is immutable, so there are no setters, only getters with the same name as the field, instead of a **get** prefix.
  - Values are declared when given to the constructor.
  - Let us explore by coding an example together