PGR112 – Step 5: Reading writing to/from file, Exception Handling

Object Oriented Programming

Bogdan Marculescu/ bogdan.marculescu@kristiania.no

Plans for Wednesday? (lille Lørdag, right?)

- Programming!

Agenda

- Exception handling
- Read from file
- Write to file
- Date

Status

- Step 1: Development environment. Hello World.
- Step 2: Methods (loops, if, array, String etc)
- Step 3: Classes and objects. Encapsulation.
- Step 4: ArrayList, Scanner, user input, debugger
- Now: Read from file. Write to file. Exception handling
- Step 6: Larger task (with options)
- …
- We are thus approaching an important point in the subject. In step 6, there will be no new topics but working with a larger case based on what you have learned so far. And you can choose between two tasks.

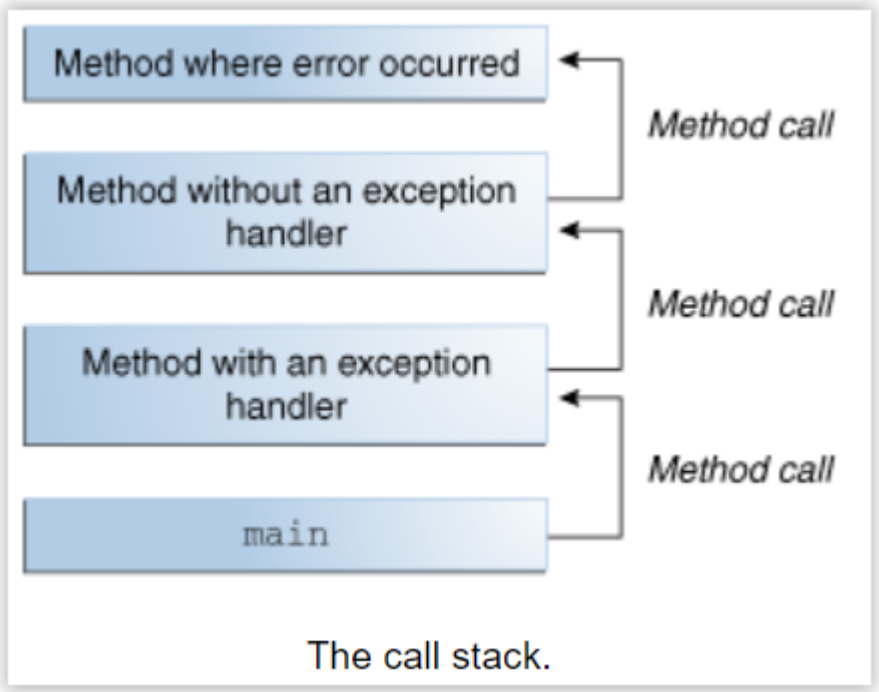
Exceptions

- In running programs, situations may arise where something abnormal is happening.
- "Abnormal" means that the code does not run as it is intended to run.
- Java definition:
  - An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

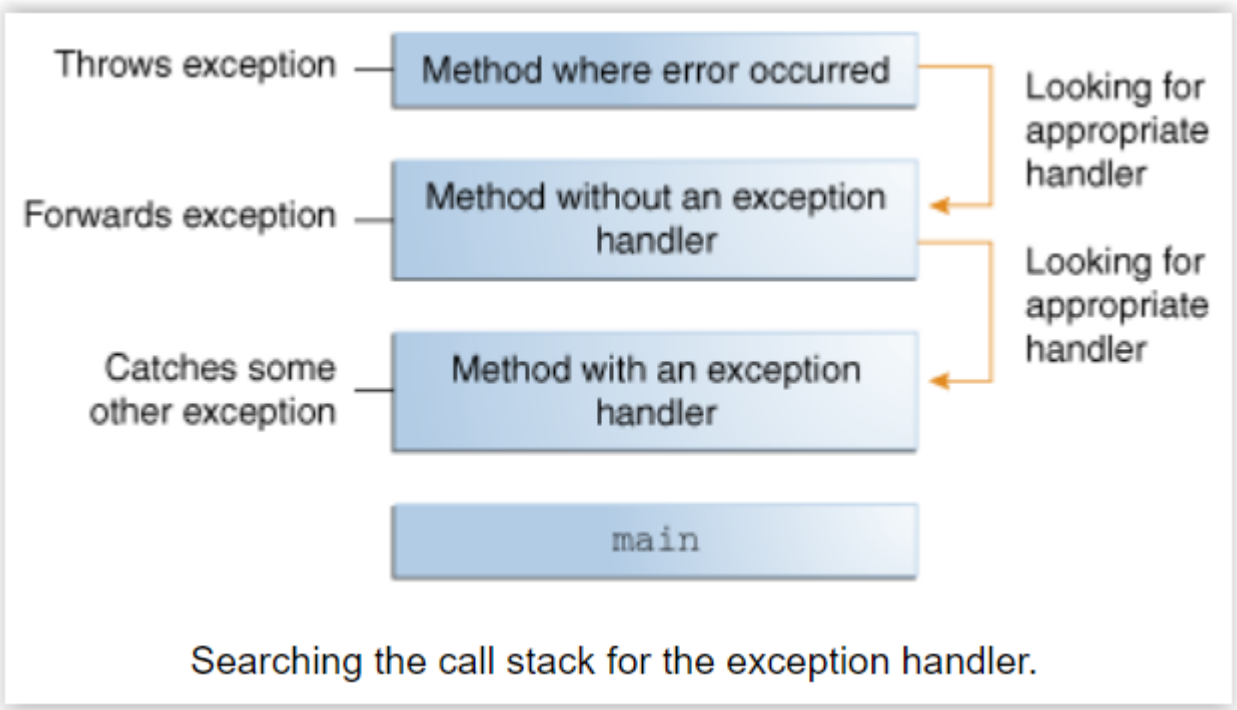
When an exception occurs

- When an error occurs in a method, the method creates an object and passes it to the runtime system.
- The object, called an exception object, contains information about the error, including the type and condition of the program when the error occurred.
- Creating an exception object and delivering it to the runtime system is called "throw an exception".
- After a method throws an exception, the runtime system tries to find something that can handle it.
- This "something" to deal with the exception is the list of methods that had been called to get to the method where the error occurred. The list of methods is known as the "call stack".

Call stack



The call stack.



Searching the call stack for the exception handler.

Demo call stack and exception

Three main groups of exceptions

- Checked exception: The compiler checks if these are handled.
- Error: Something exceptional. Something that is not expected to be taken into account (eg hardware failure)
- Unchecked exception: Bugs or logical errors. Programmer's error ☺

- For checked exceptions we must follow the requirement «catch or specify»
  - Catch the exception, or
  - Specify that the method throws the exception further

Checked exceptions

Some examples (relevant this week especially):

- FileNotFoundException
- IOException

Unchecked exceptions

Some we already know and love:

- NullPointerException
- ArithmeticException
- IllegalArgumentException
- IndexOutOfBoundsException

Catch or specify

- The main difference between checked and unchecked exceptions is that we MUST take checked exceptions into account. The compiler checks if we do.

So how do we handle an exception and how do we pass it on?

Exception Handling

- Using try/catch

```
try
{
    // statement(s) that might cause exception
}
```

```
catch
{
   // statement(s) that handle an exception
   // examples, closing a connection, closing
   // file, exiting the process after writing
   // details to a log file.
}
```

Finally

In addition to try / catch, we can have a final finally that runs after catch. Example:

```
class Division {
    public static void main(String[] args)
    {
        int a = 10, b = 5, c = 5, result;
        try {
            result = a / (b - c);
            System.out.println("result" + result);
        }

        catch (ArithmeticException e) {
            System.out.println("Exception caught:Division by zero");
        }

        finally {
            System.out.println("I am in final block");
        }
    }
}
```

```
Exception caught:Division by zero
I am in final block
```

Throws

- So we can process an exception by using try / catch. But what if we'd rather throw it further?

- Example from Oracle:

```
public void writeList() throws IOException {
```

```
public void writeList() {
    PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++) {
        out.println("Value at: " + i + " = " + list.get(i));
    }
    out.close();
}
```

Constructor Details

FileWriter

```
public FileWriter(String fileName)
           throws IOException
```

Java API

Catch or specify?

We can therefore choose whether to catch an exception or discard it further. When should we choose what?

Catch the exception if you can choose an alternative strategy and get to the end of it.

Capture the exception in a central location where it can be logged (not so central to us in this topic).

Put another way: If you can do something about it; do it. If not, forward it.

Using finally

Used for «cleanup code». For example, close resources that are in use in the try / catch.

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

There are newer ways to close resources when try / catch is in use: try-with-resources. We will return to it later. But if you are the curious type, then you can use it in this week's practice.

Demo – catch exception

Some methods in the Exception objects

- getMessage(): Description of what went wrong.
- printStackTrace(): Prints out information about where the exception occurred and how it traveled further in the stack. Example:

```
java.lang.NullPointerException
        at MyClass.mash(MyClass.java:9)
        at MyClass.crunch(MyClass.java:6)
        at MyClass.main(MyClass.java:3)
```

Oracle

Exceptions - summary

- When we use (call for) methods that throw checked exceptions, then we must decide whether we want to throw the possible exception further, or whether we should process it.
  - Process it (try / catch / finally).
  - Throw it further: specify in the method that it throws exceptions further with the use of throws
    - example: public void method throws IOException {}

- Exceptions are a large topic, and we will return to them later. For example:
  - How can we throw our own exceptions?
  - Multi-catch
  - Try-with-resources
  - Chained exceptions

Reading from file

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files
```

- We already know Scanner! Example fra w3schools:

```
public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Demo, reading from file

Writing to file

```
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors
```

- Example from w3schools:

```
public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Files in Java might be tricky, but it is fun enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Demo, writing to file

A bit about date and time

- Java struggled for a long time to sort out data that had to do with time and place. But «now everything is going so much better».
- The java.time package contains a lot of goodies you can use.
- In this week's assignments, the LocalDate class in particular is relevant. It represents a date.

Create a LocalDate object from textual representation

`static LocalDate parse(CharSequence text)`	Obtains an instance of `LocalDate` from a text string such as `2007-12-03`.

Oracle

Eksempel:

LocalDate date = LocalDate.parse("2007-12-03");

Ojoj! This is an example of calling a static method! A static method is a class-level method (not a method in an object).

More about that later in the topic…

Textual representation of a LocalDate object

Eksempel:

String s = localDate.toString();

We will also return to this toString method later. It is available to ALL objects!

null

- You probably (should?) remember the null from DB topic.
- We have the same phenomenon in Java (and JavaScript)
- NullPointerException is a very common exception to encounter (and one of the easiest to correct)
- Occurs when you perform something with an object (for example, calls a method in the object), but the object is null.
- Let’s have a look.

Wrap up

- The goals of this step are:
  - I can use Scanner to read from file.
  - I can use FileWriter to write to file.
  - Overall, I understand how exception handling works in Java.
  - I understand how I can catch and throw exceptions.
  - I can use a suitable class to handle a date.

- Good luck with the tasks!