Object-Oriented Programming in Java

Lecture 8 - Exception Handling

Emily Lucia Antosch

HAW Hamburg

15.08.2025

Contents

1. Introduction

1.1 Where are we now?

- In the last lecture, we dealt with creating graphical user interfaces
- You can now
  - create windows in which other elements can live,
  - arrange elements using layouts and panels
  - and create graphics directly in Java.
- Today we continue with **Exception Handling**.

1.1 Where are we now?

1. Imperative Konzepte
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. Schnittstellen
6. Graphische Oberflächen
7. **Exception Handling**
8. Input and Output
9. Multithreading (Parallel Computing)

1.2 The goal of this chapter

- You handle exceptions and errors that occur during program execution to establish an orderly program flow in exceptional situations.
- You define your own exception classes adapted to the needs of your specific application.

2. Exception Handling

2.1 Introductory Example

? Question

- What output is produced by the following program?

Java

```java
public class ProvokeException {
    public static void main(String[] args) {
        int a = 3;
        int b = 2;
        printRatio(a, b);
        System.out.println("Exiting main()");
    }

    public static void printRatio(int a, int b) {
        int ratio = a / b;
        System.out.println("Ratio = " + ratio);
    }
}
```

? Question

- And what output is produced for a = 7 and b = 0?

2.1 Introductory Example

? Question

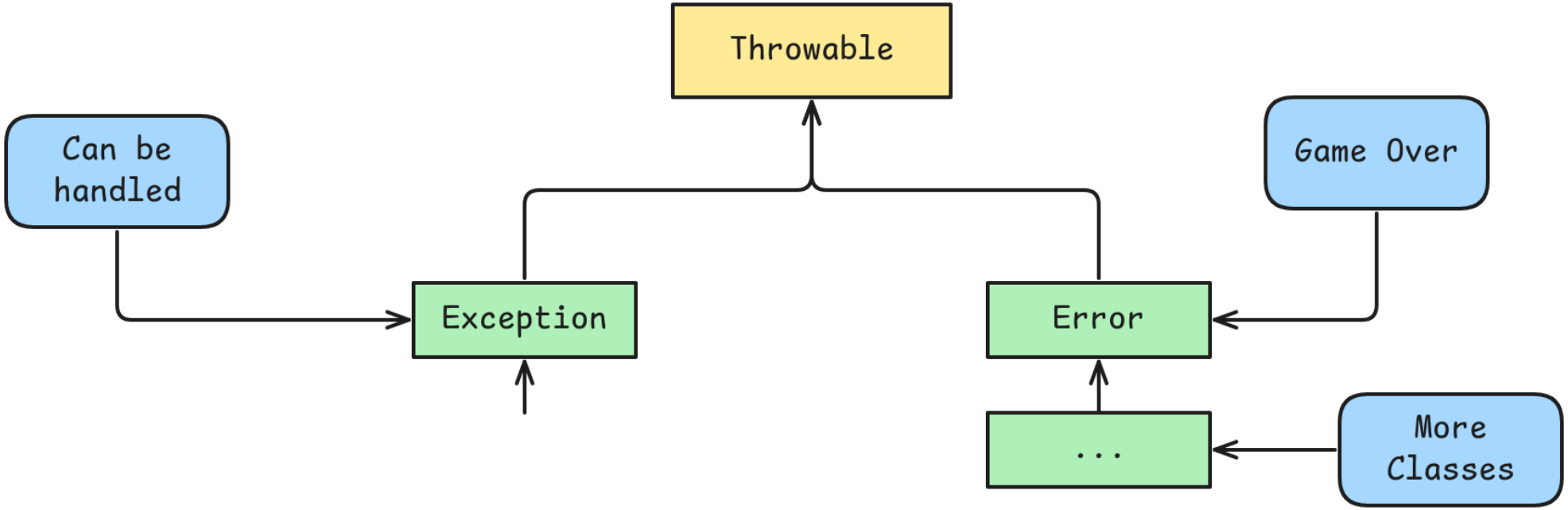
- What can go “wrong” in a program?
- When must the normal program flow be interrupted?
- When must a program be terminated, when can it be continued?

Example

- Division by zero
- Calling `a.method()` although variable `a` has the value null
- Negative or too high index for arrays
- Converting the string “This is text” to an integer of type `int`
- File not found
- No more memory available

2.1 Introductory Example

- Exceptions and errors are represented by objects of special classes
- Base class of all exception classes is Throwable



- One distinguishes:
  - Exception: Handleable, program can be continued
  - Error or fatal error: Not handleable, terminate program

2.1 Introductory Example

Memorize

- Exception is also used as a collective term for exceptions and errors.
- Exception handling is also called Exception handling.

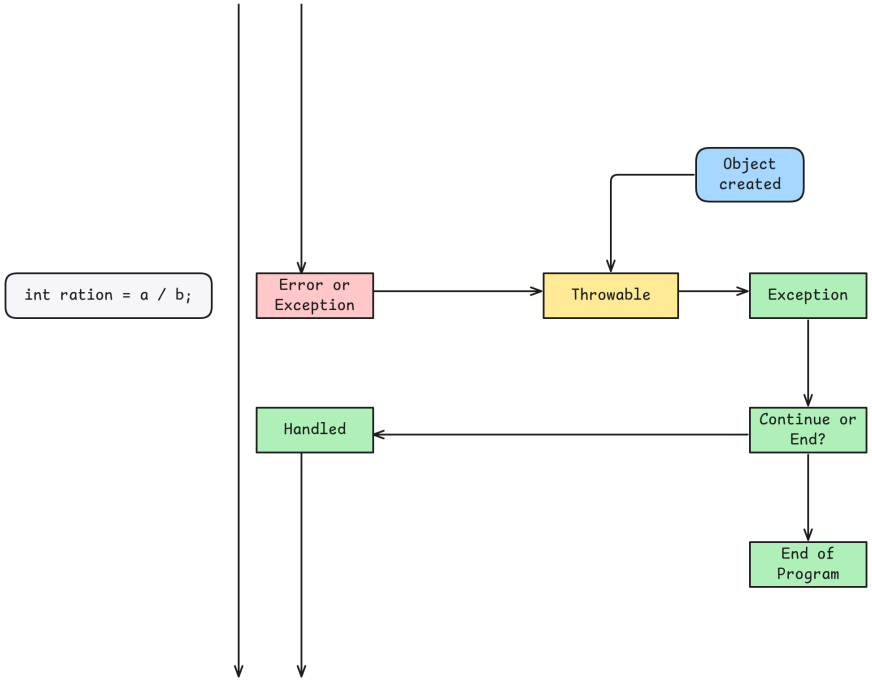
- Some classes for exceptions:
  - Division by zero (ArithmeticException)
  - Access to method or attribute via null reference (NullPointerException)
  - Invalid array index (ArrayIndexOutOfBoundsException)
  - Invalid characters when reading a number (NumberFormatException)
  - File not found (FileNotFoundException)

Java

```java
int[] array = {1, 2, 3, 4};
System.out.println(array[4]);

String message;
System.out.println(message.length());

int code = Integer.parseInt("12a4");
```

2.2 Exception handling flow

1. Throwing an exception:
   - Program flow is immediately interrupted
   - Object is created that represents the exception
2. Catching an exception:
   - Programmer can catch and handle the exception



3. Throwing exceptions

3.1 Throwing exceptions

- In case of error, exceptions are automatically generated (e.g. division by zero).
- However, exceptions can also be thrown explicitly.

```java
throw ExceptionObject;
```

- Constructor can be passed a string (e.g. as error description)

Example

```java
throw new Exception();
throw new Exception("Division by zero");
Exception exception = new Exception(); throw exception;
```

3.1 Throwing exceptions

- For illustration:
  - Throw an exception before attempting to divide by zero.

Java

```java
public class ThrowException {
    public static void main(String[] args) {
        int a = 3;
        int b = 0;
        printRatio(a, b);
        System.out.println("Exiting main()");
    }

    public static void printRatio(int a, int b) {
        int ratio = a / b;
        System.out.println("Ratio = " + ratio);
    }
}
```

3.1 Throwing exceptions

- Example solution:

Java

```java
public class ThrowException {
    public static void main(String[] args) {
        int a = 3;
        int b = 0;
        printRatio(a, b);
        System.out.println("Exiting main()");
    }

    public static void printRatio(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero");
        }
        System.out.println("Ratio = " + (a / b));
    }
}
```

3.1 Throwing exceptions

- Output in case of error:
  - Exception type (e.g. ArithmeticException)
  - Error message (e.g. “Division by zero”)
  - Stack trace (i.e. chain of called methods)

Example

```
1 Exception in thread "main" java.lang.ArithmeticException: Division by zero at
2   kapitel8_exceptions.ThrowException.printRatio(E02_ThrowException.java:20) at
3   kapitel8_exceptions.ThrowException.main(E02_ThrowException.java:14)
```

- Method `main()` called `printRatio()` at line 14
- Method `printRatio()` threw the exception at line 20

4. Catching exceptions

4.1 Exception Handling

- Exceptions can be caught and handled:

```java
try {
    // Statements ...
} catch (ExceptionType e) {
    // Statements ...
}
```

- Try block contains code that can throw an exception
- If an exception is thrown in the try block:
  1. Try block is immediately terminated
  2. Catch block is executed if the exception type (ExceptionType) matches
  3. Program continues after the catch block
- Exception type of catch block doesn't match: Exception is not caught!
- No exception thrown: Catch block is skipped

4.1 Exception Handling

- Avoid the “crash”:
  - Catch the thrown exception!

Java

```java
public class TryCatch {
    public static void main(String[] args) {
        int a = 3;
        int b = 0;
        printRatio(a, b);
        System.out.println("Exiting main()");
    }

    public static void printRatio(int a, int b) {
        int ratio = a / b;
        System.out.println("Ratio = " + ratio);
    }
}
```

4.1 Exception Handling

- Example solution:

```java
public static void printRatio(int a, int b) {
    try {
        int ratio = a / b;
        System.out.println("Ratio = " + ratio);
    } catch (ArithmeticException e) {
        System.out.println("Exception caught in printRatio()");
        System.out.println("e.getMessage(): " + e.getMessage());
        System.out.println("e.toString(): " + e + "\n");
    }
    System.out.println("Exiting printRatio()");
}
```

- Selected methods for exception objects:
  - `getMessage()`
  - `printStackTrace()`
  - `toString()`

4.1 Exception Handling

? Question

And now?

Java

```java
public class TryCatchChain1 {
    public static void main(String[] args) {
        int ratio = getRatio(3, 0);
        System.out.println("Ratio = " + ratio);
        System.out.println("Exiting main()");
    }
    public static int getRatio(int a, int b) {
        int ratio = 0;
        try {
            ratio = a / b;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught in getRatio()");
        }
        System.out.println("Exiting getRatio()");
        return ratio;
    }
}
```

4.1 Exception Handling

? Question

And now?

Java

```java
public class TryCatchChain2 {
    public static void main(String[] args) {
        try {
            int ratio = getRatio(3, 0);
            System.out.println("Ratio = " + ratio);
        } catch (ArithmeticException e) {
            System.out.println("Exception caught in main()");
        }
        System.out.println("Exiting main()");
    }

    public static int getRatio(int a, int b) {
        int ratio = a / b;
        System.out.println("Exiting getRatio()");
        return ratio;
    }
}
```

4.1 Exception Handling

- If multiple exception types can occur, multiple catch blocks are needed.
- Exception types of catch blocks must be different
- The first matching catch block is executed.

Java

```java
try {
    // ...
} catch (ExceptionType1 e) {
    // ...
} catch (ExceptionType2 e) {
    // ...
} catch (ExceptionType3 e) {
    // ...
}
```

4.1 Exception Handling

? Question

- The following source code contains two error sources. Which ones?
- What output does the program produce?

Java

```java
public class ExceptionTypes1 {
    static int recursiveIncrease(int i) {
        return recursiveIncrease(i + 1);
    }

    public static void main(String[] args) {
        int[] a = new int[4];
        try {
            a[4] = recursiveIncrease(7);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException");
        }
        System.out.println("Exiting main()");
    }
}
```

4.1 Exception Handling

Task 1

- Modify the previous source code so that both error sources are caught.

Java

```java
public class ExceptionTypes2 {
    static int recursiveIncrease(int i) {
        return recursiveIncrease(i + 1);
    }

    public static void main(String[] args) {
        int[] a = new int[4];
        try {
            a[4] = recursiveIncrease(7);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException");
        } catch (StackOverflowError e) {
            System.out.println("Caught StackOverflowError");
        }
        System.out.println("Exiting main()");
    }
}
```

4.1 Exception Handling

? Question

- Oops, something goes wrong in the catch block below!
- Is the new exception handled? What is output?

Java

```java
public class ExceptionTypes3 {
    static int recursiveIncrease(int i) {
        return recursiveIncrease(i + 1);
    }
    public static void main(String[] args) {
        int[] a = new int[4];
        try {
            a[4] = 0;
        } catch (ArrayIndexOutOfBoundsException e) {
            recursiveIncrease(7);
        } catch (StackOverflowError e) {
            System.out.println("Caught StackOverflowError");
        }
        System.out.println("Exiting main()");
    }
}
```

4.1 Exception Handling

- A catch block only refers to its associated try block.
- If catch block throws an exception, it is not caught by subsequent blocks

? Question

- How can we catch the exception generated in the catch block?

4.1 Exception Handling

- Source code that generates exception in nested try block

Java

```java
public static void main(String[] args) {
    int[] a = new int[4];
    try {
        a[4] = 0;
    } catch (ArrayIndexOutOfBoundsException e1) {
        try {
            recursiveIncrease(7);
        } catch (StackOverflowError e2) {
            System.out.println("Caught inner StackOverflowError");
        }
    } catch (StackOverflowError e) {
        System.out.println("Caught outer StackOverflowError");
    }
    System.out.println("Exiting main()");
}
```

4.1 Exception Handling

- Sometimes certain code must be executed in any case.
- Example: Closing open files or data streams
- Optional finally block:
  - Always comes last (i.e. after try and catch blocks)
  - Code is executed at the end of the construct … really always … honestly!

Java

```java
try {
    // ...
} catch (ExceptionType1 e) {
    // ...
} catch (ExceptionType2 e) {
    // ...
} finally {
    // Is guaranteed to be executed
}
```

4.1 Exception Handling

? Question

- What is output?

Java

```java
public class TryCatchFinally1 {
    static int recursiveIncrease(int i) {
        return recursiveIncrease(i + 1);
    }
    public static void main(String[] args) {
        int[] a = new int[4];
        try {
            a[4] = 0;
        } catch (ArrayIndexOutOfBoundsException e1) {
            recursiveIncrease(7);
            System.out.println("Caught ArrayIndexOutOfBoundsException");
        } finally {
            System.out.println("Finally");
        }
        System.out.println("Exiting main()");
    }
}
```

4.1 Exception Handling

? Question

- What is output?

Java

```java
public class TryCatchFinally2 {
    public static void main(String[] args) {
    System.out.println("Ratio = " + getRatio(3, 0));
    }
    public static int getRatio(int a, int b) {
        int ratio = 0;
        try {
            ratio = a / b;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught in getRatio()");
            return 0;
        } finally {
            System.out.println("Finally");
        }
        System.out.println("Exiting getRatio()");
        return ratio;
    } }
```

4.1 Exception Handling

- Rules for blocks:
  - Exactly one try block as the first block
  - None or any number of catch blocks after the try block
  - None or one finally block as the last block
  - A try block must have at least one catch or finally block.
- The following structure is allowed:

```java
try {
    // ...
} finally {
    // ...
}
```

5. Defining custom exceptions

5.1 Custom exceptions

- Let's consider the following program:

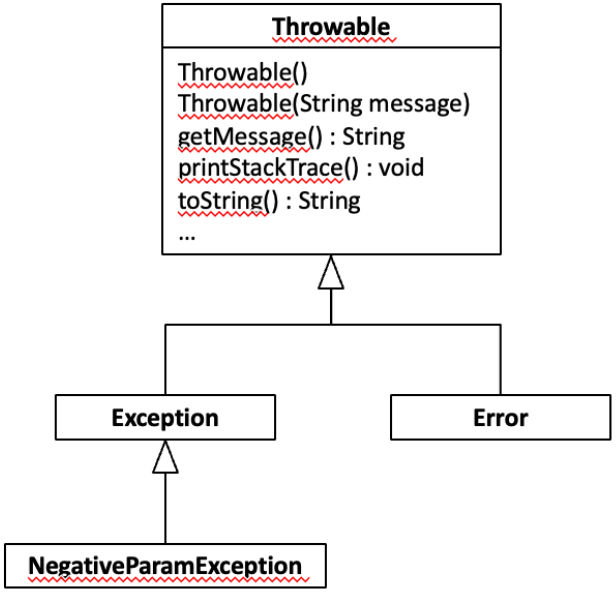
```java
public class OwnException1 {
    public static void main(String[] args) {
        double x = 25.0;
        System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
    }

    public static double squareRoot(double x) {
        return Math.sqrt(x);
    }
}
```

? Question

- Method squareRoot() should throw an exception for negative parameters
- How could we define our own type (e.g. NegativeParameterException)?

5.1 Custom exceptions

- Custom exception type by deriving from an existing class
- First approach: Derive from the Exception class



5.1 Custom exceptions

- Approach generates error message ("Unhandled exception")
- Why is that now?!

```java
class NegativeParamException extends Exception {
}

public class OwnException2 {
    public static void main(String[] args) {
        double x = 25.0;
        System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
    }

    public static double squareRoot(double x) {
        if (x < 0.0) {
            throw new NegativeParamException();
        }
        return Math.sqrt(x);
    }
}
```

5.1 Custom exceptions

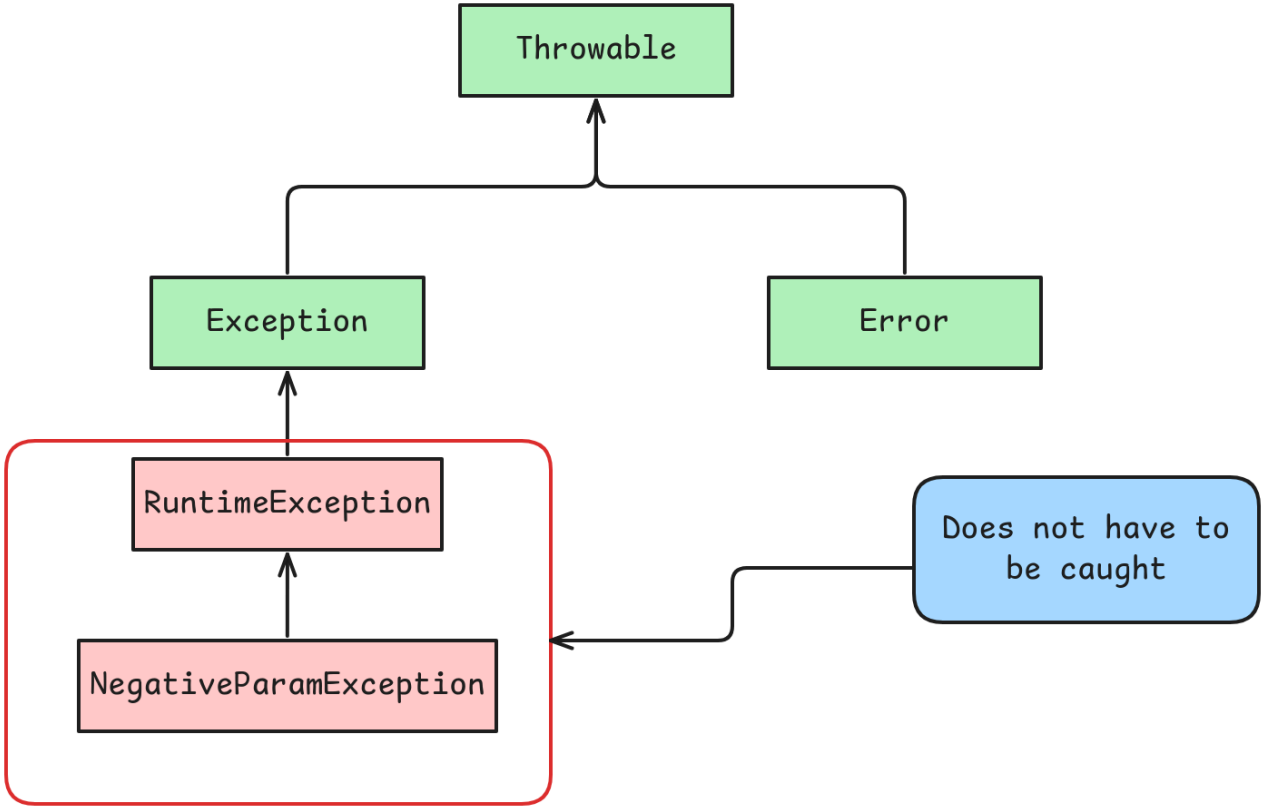
- Background:
  - Exceptions must be caught OR
  - Method must declare via throws that it can throw an exception type.

Java

```java
public class OwnException2 {
    public static void main(String[] args) throws NegativeParamException {
        double x = 25.0;
        System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
    }

    public static double squareRoot(double x) throws NegativeParamException {
        if (x < 0.0) {
            throw new NegativeParamException();
        }
        return Math.sqrt(x);
    }
}
```

5.1 Custom exceptions

- This applies to all exception types (i.e. Throwable and derived from it) except for:
  - Class RuntimeException
  - Classes derived (directly or indirectly) from RuntimeException



5.1 Custom exceptions

Java

```java
class NegativeParamException extends RuntimeException {
}

public class OwnRuntimeException {
    public static void main(String[] args) {
        double x = 25.0;
        System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
    }

    public static double squareRoot(double x) {
        if (x < 0.0) {
            throw new NegativeParamException();
        }
        return Math.sqrt(x);
    }
}
```

5.1 Custom exceptions

- Description ("message") passed to constructor of base class

Java

```java
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}

public class OwnExceptionWithMessage {
    public static void main(String[] args) {
        try {
            throw new MyException("An exception just for fun :-) ...");
        } catch (MyException e) {
            System.out.println("Message: " + e.getMessage());
        }
    }
}
```

6. License Notice

6.1 Attribution

- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.