

Java

Les 5

jeroen.devos01@ap.be

Lesdoelen

- Exception
- Stack trace
- Debugging





Wat is een Exception?

- Exception is een probleem / fout dat opduikt tijdens het uitvoeren van je programma.
- Exception is géén compilatie fout.
- Wanneer een Exception optreedt wordt het programma onderbroken.
- Wanneer een Exception niet correct wordt afgehandeld zal het programma stoppen.

Voorbeeld : IndexOutOfBoundsException

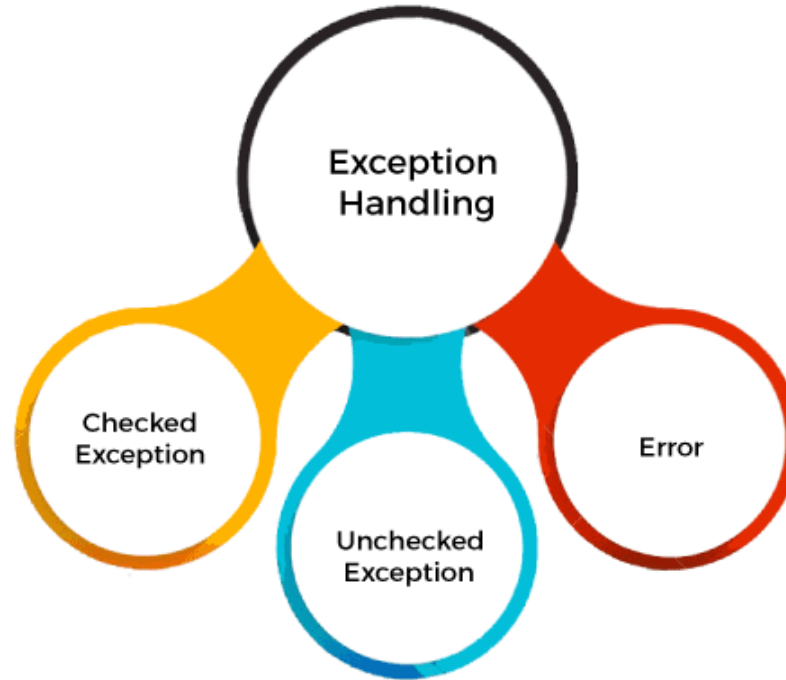
```
List<String> names = new ArrayList<>();  
System.out.println(names.get(0));
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 0 out of bounds for length 0  
    at java.base/java.util.Objects.checkIndex(Objects.java:372)  
    at java.base/java.util.ArrayList.get(ArrayList.java:459)  
    at Main.main(Main.java:48)
```

Fout : index 0 bestaat niet

→ Stack trace leren lezen

Soorten exceptions



Checked Exception

- Exception die verplicht moet opgevangen worden. Je kan niet compileren zonder iets met die Exception te doen.
- Je code verwacht dat er een fout kán optreden.
- Enkele bekende voorbeelden:
 - IOException : I/O operaties met bestanden
 - ClassNotFoundException : dynamisch classes inladen
 - ParseException : gegevens omzetten naar een ander formaat

Unchecked Exception

- Exception die op een onverwacht moment kan optreden. Tijdens coderen wordt niet afgedwongen om de fout op te vangen.
- Zal steeds overerven van RuntimeException.
- Enkele bekende voorbeelden:
 - NullPointerException : methode van een null object oproepen
 - IndexOutOfBoundsException : index bestaat niet op lijst
 - ArithmeticException : rekenfout, bv delen door 0

Error

- Geen echte Exception
- Kan niet opgevangen worden
- Zal steeds het programma beëindigen (crash)

- Enkele bekende voorbeelden:
 - OutOfMemoryError
 - StackOverflowError

Exception handling

```
DecimalFormat format = new DecimalFormat("00.00");  
try {  
    double parsed = format.parse(12.02).doubleValue();  
} catch(ParseException e) {  
    e.printStackTrace();  
}
```

- try block : bevat de code die de Exception kan gooien
- catch block :
 - argument de fout
 - code block : wat te doen met de fout

Multiple Exception handling

```
DecimalFormat format = new DecimalFormat("00.00");  
try {  
    double parsed = format.parse(12.02).doubleValue();  
    Class.forName("DemoClass");  
} catch (ParseException e) {  
    e.printStackTrace();  
} catch (ClassNotFoundException e) {  
    e.printStackTrace();  
}
```

In geval van meerdere Exceptions binnen één try block : meerdere catch blokken schrijven.

Finally

```
DecimalFormat format = new DecimalFormat("00.00");  
double parsed;  
try {  
    parsed = format.parse(12.02).doubleValue();  
} catch(ParseException e) {  
    e.printStackTrace();  
} finally {  
    parsed = 10;  
}
```

- finally block : deze code zal altijd uitgevoerd worden, ongeacht of er een Exception optreedt.
- Wordt typisch gebruikt om op te kuisen (bv databaseconnectie sluiten)

Zelf een exception schrijven

- isFather() methode geeft true of false terug
- Wat als we willen dat als de input van de method “Darth Vader” of “Anakin Skywalker” is, er nog een ander resultaat is?

→ Eigen Exception schrijven

```
public class DarthVader extends StarWarsCharacter {  
    public boolean isFather(StarWarsCharacter character) {  
        if(character.getName().equals("Luke Skywalker")) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```



StarWarsFatherException

```
public class StarWarsFatherException extends Exception {  
  
}
```

- Maak een nieuw class aan
 - Conventie : naam eindigt op Exception
- De class erft over van Exception
 - Checked Exception

Throw en Throws

```
public class DarthVader extends StarWarsCharacter {  
    public boolean isFather(StarWarsCharacter character) throws StarWarsFatherException {  
        if(character.getName().equals("Luke Skywalker")) {  
            return true;  
        } else if(character.getName().equals("Darth Vader")) {  
            throw new StarWarsFatherException();  
        } else {  
            return false;  
        }  
    }  
}
```

Throw en Throws

```
} else if(character.getName().equals("Darth Vader")) {  
    throw new StarWarsFatherException();  
} else {
```

throw new ...Exception()

- new -> maak een nieuwe foutmelding aan
- throw -> 'gooi' de fout om daarna 'gevangen' te worden (catch)

Throw en Throws

```
public class DarthVader extends StarWarsCharacter {  
    public boolean isFather(StarWarsCharacter character) throws StarWarsFatherException {  
    }  
}
```

throws ...Exception

- throws : aanduiding op een methode dat deze een Exception kan geven
→ Checked Exception
- Als je deze methode gebruikt zal je een try catch block moeten voorzien

Throws - catch

```
DarthVader vader = new DarthVader();  
try {  
    boolean father = vader.isFather(new LukeSkywalker());  
} catch (StarWarsFatherException e) {  
    e.printStackTrace();  
}
```

- `.isFather()` throws `StarWarsFatherException`
- Deze lijn moet in een try / catch block komen.

StarWarsFatherException - message

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

Een eigen foutmelding verplichten:

- Implementeer constructor mét message
- Geef message door aan Exception via super

```
starwars.characters.StarWarsFatherException Create breakpoint : You cannot be your own father  
at starwars.characters.DarthVader.isFather(DarthVader.java:31)  
at Main.main(Main.java:50)
```

Exception methods

```
} catch(Exception e) {  
    e.printStackTrace();  
    System.out.println(e.getMessage());  
}
```

- `.printStackTrace()` = default gedrag, print de volledige stack in de system out.
- `.getMessage()` : haalt enkel het foutbericht op.

Custom methods?

```
public class StarWarsFatherException extends Exception {  
    private String name;  
  
    public StarWarsFatherException(String message, String name) {  
        super(message);  
        this.name = name;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
}
```



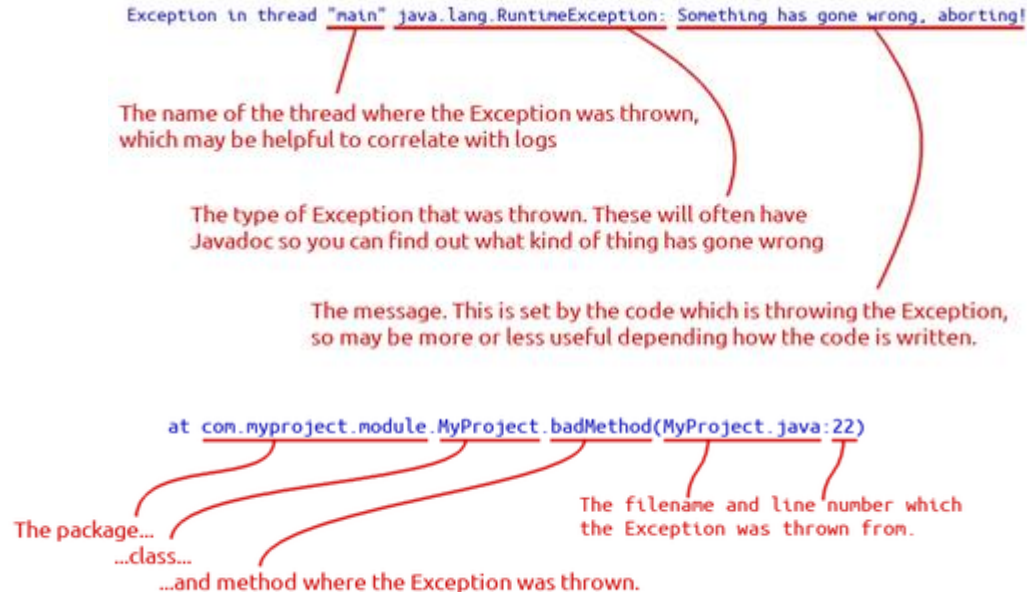
Code together or code with me

- Haal demo code van Digitap
- We gaan samen bug hunting



Stack trace

- De stack trace stelt het pad voor dat gevolgd werd om tot aan de fout te komen.
- We krijgen alle informatie van de methodes die opeenvolgend aangeroepen werden.

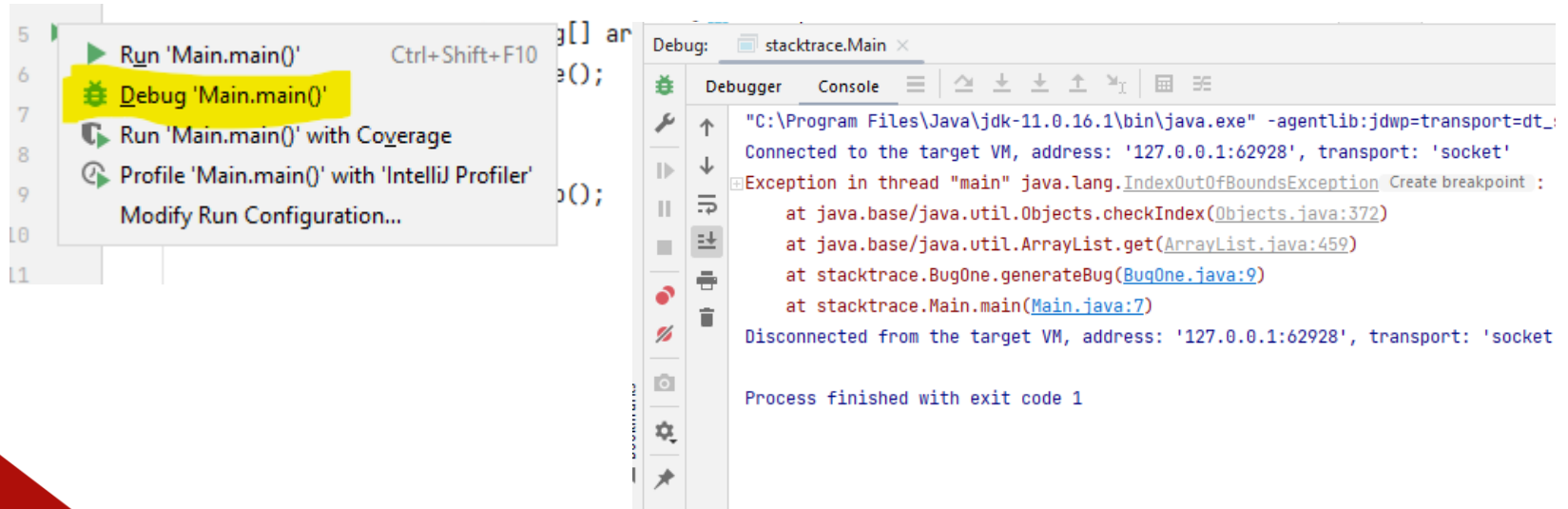


BugHunting code

- BugOne
- BugTwo
- BugThree
- BugFour
- BugFive
- BugSix

Debugging in IntelliJ

- Debugging is een krachtige tool om een fout te localiseren
- Start debugging op -> debugger console opent



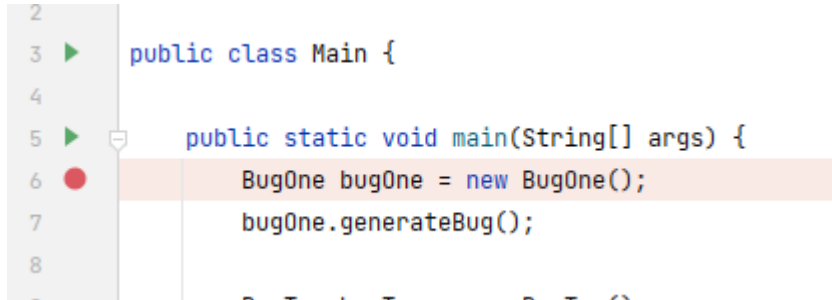
Debugging - breakpoint

De kracht van debugging zit in:

- Breakpoints
- Stepping

Plaats een break point door naast de lijnnummer te klikken

→ Rode bol verschijnt + lijn wordt rood



```
2  
3 ▶ public class Main {  
4  
5 ▶ public static void main(String[] args) {  
6 ● BugOne bugOne = new BugOne();  
7   bugOne.generateBug();  
8
```

The screenshot shows a code editor with a Java class named 'Main'. A breakpoint, represented by a red dot, is set on line 6, which contains the code 'BugOne bugOne = new BugOne();'. The line is highlighted in light orange. The line numbers 2 through 8 are visible in the left margin. The code on line 7 is 'bugOne.generateBug();'.



**FOR TODAY
ANYWAY...**