HO
GENT

OOSDII

Exception handling Werkcollege

Table of Contents

1. DivideByZeroWithExceptionHandling voorbeeld

Gegeven de classe `DivideByZeroNoExceptionHandling` uit de theorie.

In de functionaliteit is nog geen rekenening gehouden met het opvangen en afhandelen van exceptions, waardoor het programma soms bruusk tot einde komt.

We passen de code aan om de fouten op te vangen en af te handelen.

1.1. Aanpassing programma delen door 0

Bij een foute invoer van de noemer (0 of geen integer getal) treedt een exception op. We gaan deze exceptions als volgt afhandelen:

- Uitschrijven van de fout (System.err)
- Duidelijke foutmelding voor de gebruiker uitschrijven (System.out)
- Nieuwe invoer vragen aan de gebruiker

We hebben twee mogelijkheden om de exceptions af te handelen:

- afhandeling op de plaats waar de fout optreedt (duidelijk en vlugger) → try-catch-finally-statement
- afhandeling op een andere plaats → impliciete of expliciete exception propagation

1.2. Afhandeling via try-catch-finally

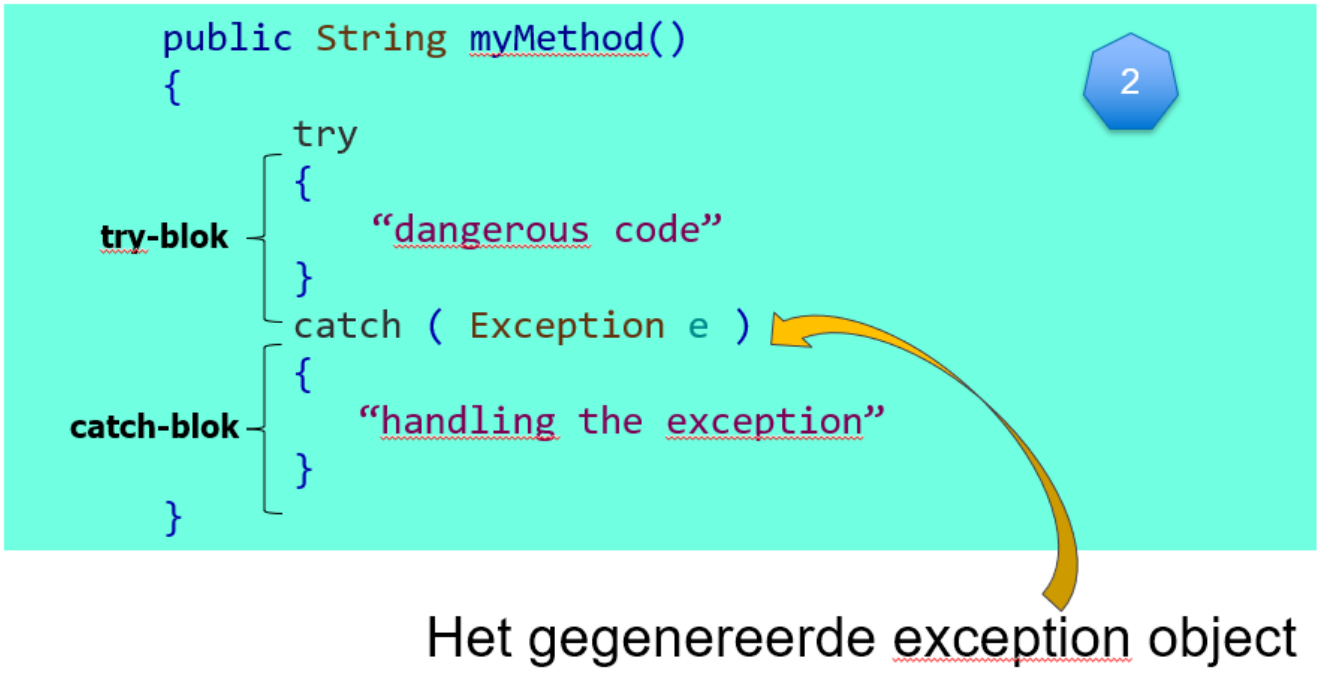
- Alle instructies waarin zich fouten kunnen voordoen en alle instructies die niet mogen uitgevoerd worden wanneer een exception optreedt komen in een try-blok
- als er een fout optreedt in het try-blok, dan wordt er een instantie van een exception-class gemaakt en deze kan opgevangen worden in één van de catch-blokken; elk catch blok verwerkt een bepaald soort fout; als er geen geschikte catcher is in het programma zal dit bruusk eindigen
- code die hoe dan ook moet uitgevoerd worden (bv. vrijgeven van resources): finally-blok [optioneel]

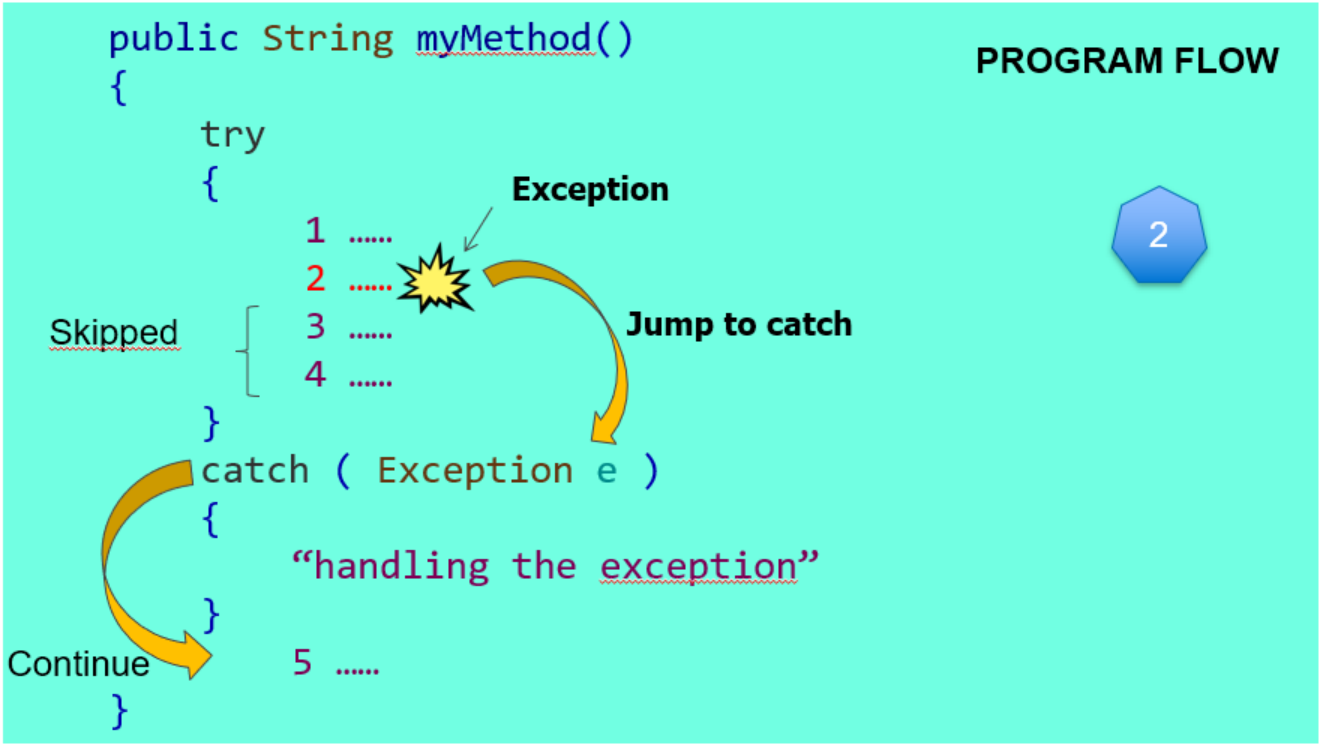
Het try-catch-finally statement:

```
try
{
    // code die mag falen
}
catch (Exception e)
{
    // afhandelen Exception
}
// meerdere catchers mogelijk
[finally
{
    // code die altijd uitgevoerd wordt
}]
```

Let op:

- Wanneer een exception optreedt in een try-blok, dan wordt dat blok verlaten (zelfs als het throw punt in een geneste blok zit) en wordt verdergegaan met het corresponderende catch-blok.
- In een catch-blok kan je geen gebruik maken van objecten lokaal gedeclareerd in het corresponderende try-blok.
- Tussen de catchers en het try-blok kan geen andere code staan.






```
6   public class DivideByZeroWithExceptionHandling {
7
8       public static int berekenQuotient(int teller, int noemer){
9                   return teller / noemer;
10      }
11
12      public static void main(String[] args) {
13          Scanner scanner = new Scanner(System.in);
14          boolean blijvenHerhalenFlag = true;
15          do {
16              try {
17                  System.out.print("Geef een integere waarde voor de teller: ");
18                  int teller = scanner.nextInt();
19                  System.out.print("Geef een integere waarde voor de noemer: ");
20                  int noemer = scanner.nextInt();
21
22                  int result = berekenQuotient(teller, noemer);
23                  System.out.printf("%nResultaat: %d / %d = %d%n", teller,
24                          noemer, result);
25                  blijvenHerhalenFlag = false;
26              } catch (InputMismatchException inputMismatchException) {
27                  System.err.printf("%nException: %s%n",
28                          inputMismatchException);
29                  scanner.nextLine();
30                  System.out.printf(
31                          "You must enter integers. Please try again.%n%n");
32              } catch (ArithmeticException arithmeticException) {
33                  System.err.printf("%nException: %s%n", arithmeticException);
34                  System.out.printf(
35                          "Zero is an invalid denominator. Please try again.%n%n");
36              }
37          } while (blijvenHerhalenFlag);
38      }
39  }
```

3

GEEL

Code kan falen

ORANJE

Niet uitvoeren als er zich een fout voordoet

1.3. Stap voor stap

Volgende methodes kunnen verkeerd lopen. Beide methodes propageren impliciet een Exception.

```
1     int teller = scanner.nextInt();
2     int noemer = scanner.nextInt();
3     int result = berekenQuotient(teller, noemer);
```

- **Propageren**: Dit wil zeggen dat ze de fout niet afhandelen, maar gewoon “verder gooien” naar de methode die hen opgeroepen heeft.
- **Impliciet**: Aan de signatuur van de methode zie je niet dat er een Exception kan optreden.

1.3.1. Impliciet propageren

nextInt

```
public int nextInt()
```

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:
the int scanned from the input

Throws:
`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range
`NoSuchElementException` - if input is exhausted
`IllegalStateException` - if this scanner is closed

1.3.2. Expliciet propageren

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value or an ASCII plus sign `'+'` (`'\u002B'`) to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:
`s` - a String containing the int representation to be parsed

Returns:
the integer value represented by the argument in decimal.

Throws:
`NumberFormatException` - if the string does not contain a parsable integer.

We kunnen ook zelf een exception expliciet propageren, bv. bij berekenQuotient(...)

```
1     public static int berekenQuotient(int teller, int noemer) throws
  ArithmeticException
2     {
3         return teller / noemer;
4     }
```

In de signatuur van de methode leg je vast dat dit type Exception kan optreden in de body van de methode. Soms is het verplicht dit op te nemen.

1.4. Verschillende exceptions afhandelen

In het try-blok kunnen uiteindelijk 2 verschillende soorten Exceptions optreden:

- Een InputMismatchException gegooid door nextInt()

- Een ArithmeticException gegooid door berekenQuotient()

```
try {
   System.out.print("Geef een integere waarde voor de teller: ");
   int teller = scanner.nextInt();
   System.out.print("Geef een integere waarde voor de noemer: ");
   int noemer = scanner.nextInt();

   int result = berekenQuotient(teller, noemer);
   System.out.printf("%nResultaat: %d / %d = %d%n", teller,
         noemer, result);
   blijvenHerhalenFlag = false;
}
```

We voorzien voor elk type een verschillend catch-blok, de afhandeling is per type verschillend:

- Andere boodschap uitschrijven voor gebruiker.
- Scanner object klaar zetten voor volgende input (enkel nodig waar het inlezen zelf fout ging)

```
catch (InputMismatchException inputMismatchException) {
   System.err.printf("%nException: %s%n",
         inputMismatchException);
   scanner.nextLine();
   System.out.printf(
         "De invoer moeten integere getallen zijn. Probeer opnieuw.%n%n");
} catch (ArithmeticException arithmeticException) {
   System.err.printf("%nException: %s%n", arithmeticException);
   System.out.printf(
         "Het cijfer 0 kan geen noemer zijn. Probeer opnieuw.%n%n");
}
catch (Exception exception)// ALL-catcher
{
   System.err.printf("%nException: %s%n", exception);
}
```

We voorzien voor elk type een verschillend catch-blok en voor de volledigheid ook de ALL-catcher:

- Deze catcher vangt de fouten op die we eventueel niet voorzien hadden.

```
catch (Exception exception)// ALL-catcher
{
   System.err.printf("%nException: %s%n", exception);
}
```

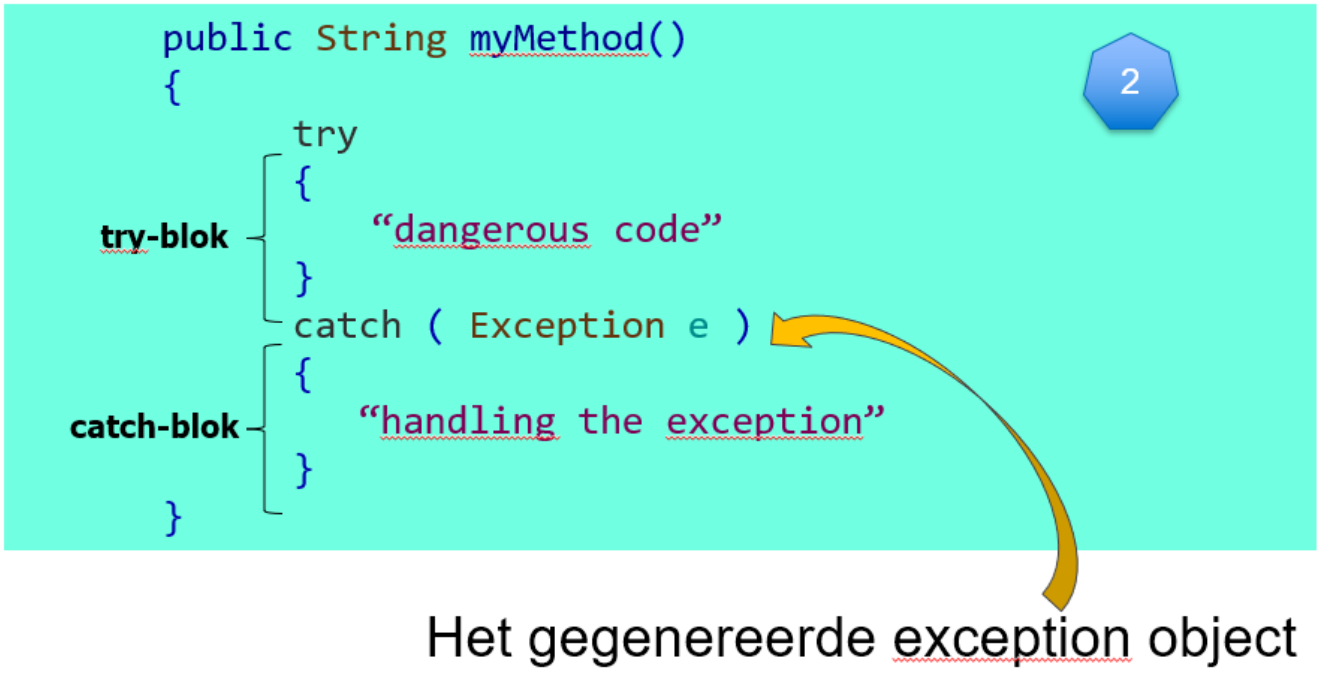
Denk goed na: een all-catcher is niet altijd een goede oplossing. Je moet heel goed weten of de afhandeling in alle gevallen dan wel ok is!

Elk catch-blok start met het keyword catch, gevolgd door een parameterlijst met één parameter, die aangeeft welk soort exceptie kan opgevangen worden meerdere parameters gescheiden door "|"

```
catch ( Type1 | Type2 | Type3 e) {….}
```

Het programma stopt als er geen geschikte catcher is.

- De eerste catcher na een try-blok die overeenkomt met het exception-type wordt uitgevoerd; alle andere worden genegeerd!
- Zet nooit een catcher van een superklasse object vóór een catcher van een subklasse object!
- Een exception kan op verschillende wijzen afgehandeld worden.
- Het is niet mogelijk om nog terug te keren naar het throw punt!



Merk op:

- Aan de hand van de boolean (vlag) blijvenHerhalenFlag zorgen we ervoor dat de do-while-lus enkel verlaten wordt als de try-blok geen enkel probleem ondervonden heeft. In dat geval wordt effectief het laatste statement van de try-blok uitgevoerd (`blijvenHerhalenFlag = false;`) en wordt bijgevolg de do-while-lus afgesloten. In alle andere (probleem)situaties blijven we in de do-while-lus!

- Er is geen finally-blok opgenomen. Er is geen code die sowieso uitgevoerd moet worden, zelfs al treedt er een exception op.

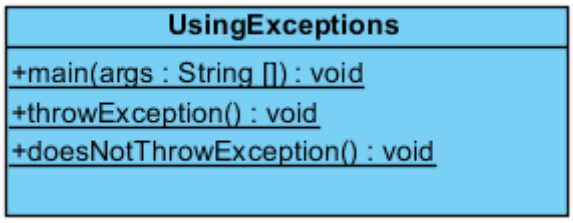
```
1  public class DivideByZeroWithExceptionHandling {
2
3      public static int berekenQuotient(int teller, int noemer) {
4          return teller / noemer;
5      }
6
7      public static void main(String[] args) {
8          Scanner scanner = new Scanner(System.in);
9          boolean blijvenHerhalenFlag = true;
10         do {
11             try {
12                 System.out.print("Geef een integere waarde voor de teller: ");
13                 int teller = scanner.nextInt();
14                 System.out.print("Geef een integere waarde voor de noemer: ");
15                 int noemer = scanner.nextInt();
16
17                 int result = berekenQuotient(teller, noemer);
18                 System.out.printf("%nResultaat: %d / %d = %d%n", teller, noemer,
   result);
19                 blijvenHerhalenFlag = false;
20             } catch (InputMismatchException inputMismatchException) {
21                 System.err.printf("%nException: %s%n", inputMismatchException);
22                 scanner.nextLine();
23                 System.out.printf("De invoer moeten integere getallen zijn. Probeer
   opnieuw.%n%n");
24             } catch (ArithmeticException arithmeticException) {
25                 System.err.printf("%nException: %s%n", arithmeticException);
26                 System.out.printf("Het cijfer 0 kan geen noemer zijn. Probeer
   opnieuw.%n%n");
27             } catch (Exception exception)// ALL-catcher
28             {
29                 System.err.printf("%nException: %s%n", exception);
30             }
31         } while (blijvenHerhalenFlag);
32     }
33 }
```

2. Even tussendoor

Wat wordt uitgeschreven op het scherm?

```
    public void myMethod()
    {
        try
        {
            int aNumber;
            aNumber = Integer.parseInt("this is not a number");
            System.out.println("Wrong input");
        } catch (NumberFormatException e)
        {
            System.out.println("The given number is not a number");
        }
        System.out.println("Continue");
    }
```

3. UsingExceptions voorbeeld



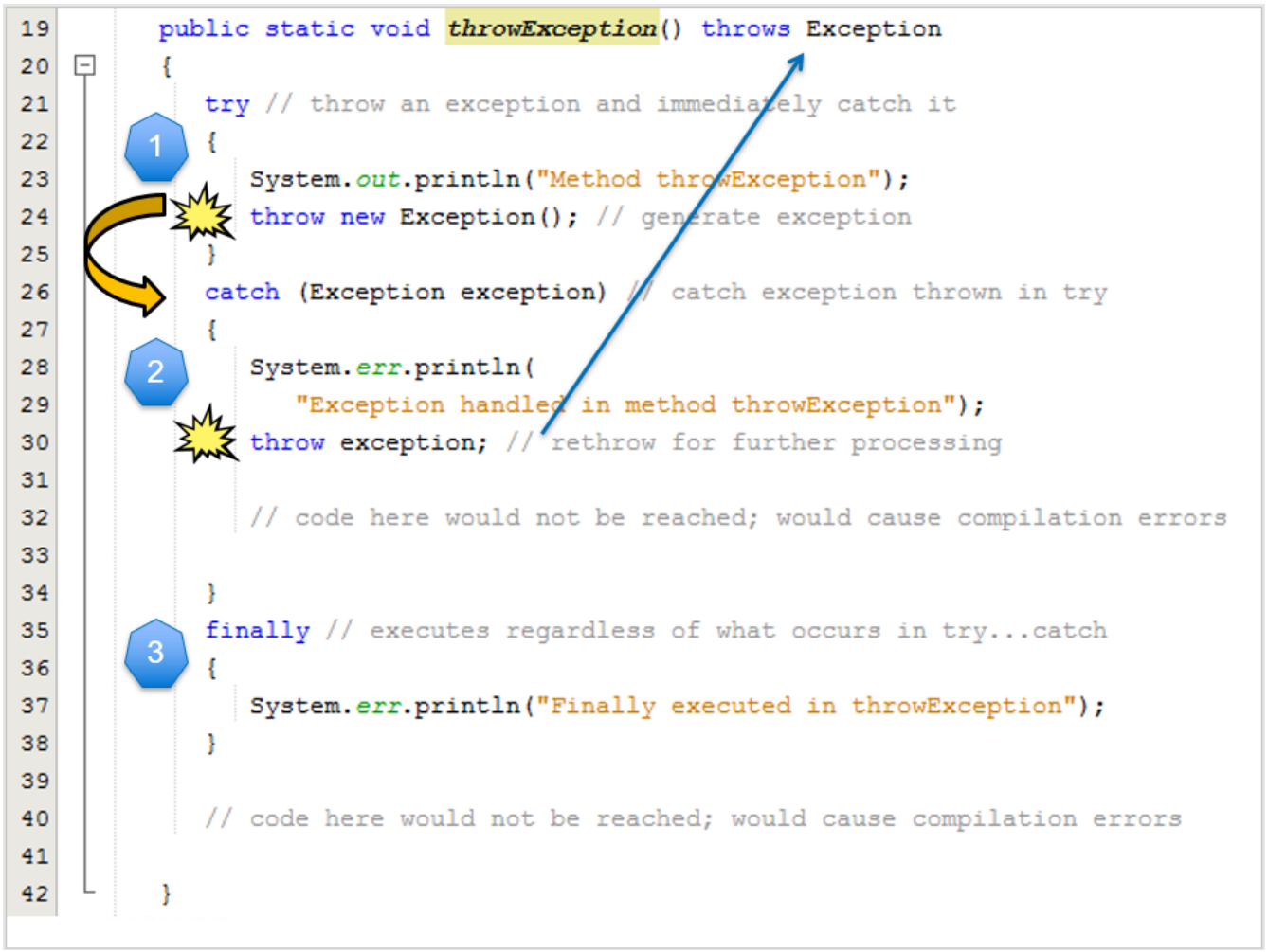
- Static main methode roept de twee andere static methodes aan;
- **throwException**: er kan een Exception optreden, die wordt deels afgehandeld en dan wordt de Exception verder doorgegooid
- **doesNotThrowException**: hier treden geen Exceptions op
- De laatste 2 methodes hebben elk een try-catch-finally- blok. Wat wordt precies uitgevoerd?

```
public static void main(String[] args)
{
    try
    {
        throwException();
    }
    catch (Exception exception) // exception thrown by throwException
    {
        System.err.println("Exception handled in main");
    }

    doesNotThrowException();
}
```

Oproepen van 2 static methodes uit dezelfde klasse.

- In de eerste methode kan een Exception optreden, dus deze aanroep moet in een try-blok staan.
- Bij de tweede methode kan er geen Exception optreden.




```
19  public static void throwException() throws Exception
20  {
21     try // throw an exception and immediately catch it
22     {
23        System.out.println("Method throwException");
24        throw new Exception(); // generate exception
25     }
26     catch (Exception exception) // catch exception thrown in try
27     {
28        System.err.println(
29           "Exception handled in method throwException");
30        throw exception; // rethrow for further processing
31
32        // code here would not be reached; would cause compilation errors
33
34     }
35     finally // executes regardless of what occurs in try...catch
36     {
37        System.err.println("Finally executed in throwException");
38     }
39
40     // code here would not be reached; would cause compilation errors
41
42  }
```

- Uitvoeren van try-blok
  - Er treedt een Exception op. Dit object wordt hier aangemaakt en gegooid via de instructie **throw**
- Catch-blok vangt de Exception op
  - Het afhandelen hier bestaat uit iets afdrukken in de err-stream. Verder afhandelen is niet mogelijk, datzelfde object wordt verder gegooid (via throw op lijn 30).
  - Exception is een checked Exception en wordt hier niet meer verder opgevangen moet dus vermeld worden in de signatuur van de methode op lijn 19 *Finally-blok wordt als laatste stap uitgevoerd.

```
44      public static void doesNotThrowException()
45      {
46         try // try block does not throw an exception
47         {
48            System.out.println("Method doesNotThrowException");
49         }
50         catch (Exception exception) // does not execute
51         {
52            System.err.println(exception);
53         }
54         finally // executes regardless of what occurs in try...catch
55         {
56            System.err.println(
57               "Finally executed in doesNotThrowException");
58         }
59
60         System.out.println("End of method doesNotThrowException");
61      }
```

- Uitvoeren van try-blok
  - Er treedt geen Exception op. Er wordt dus geen enkel catch-blok uitgevoerd. Finally-blok wordt uitgevoerd.
- Daarna gaan we verder met de code onder het finally-blok.

Opgelet De uitvoer van dit programma ziet er niet altijd identiek uit.

```
Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Finally executed in doesNotThrowException
Method doesNotThrowException
End of method doesNotThrowException
```

```
Exception handled in method throwExceptionMethod throwException

Finally executed in throwException
Exception handled in main
Finally executed in doesNotThrowException
Method doesNotThrowException
End of method doesNotThrowException
```

- **Zwart**: System.out (standard output stream)

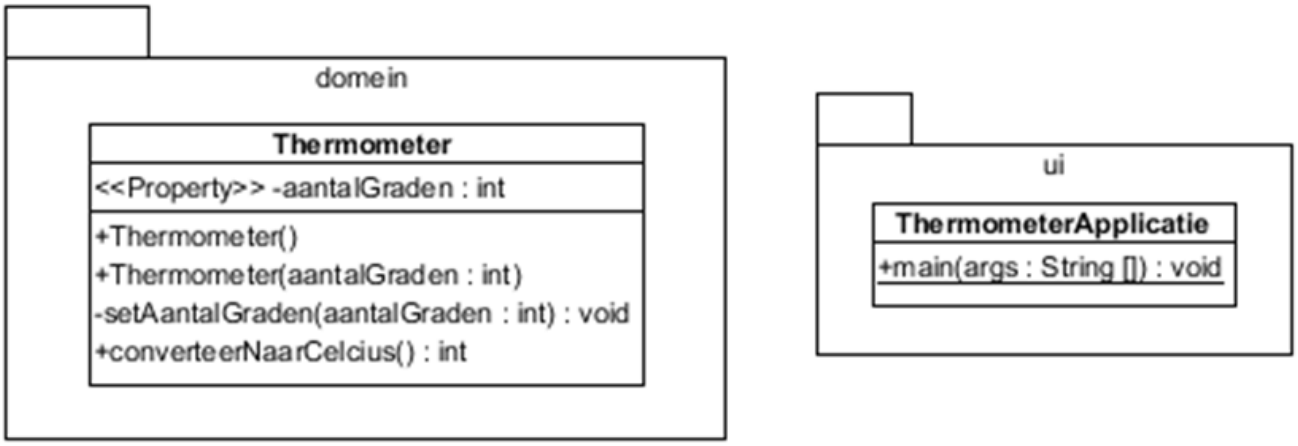
- **Rood**: System.err (standard error stream)

Volgorde zwarte boodschappen en volgorde rode boodschappen ligt vast, maar de volgorde onderling kan altijd anders zijn.

4. Oefening - Afhandelen van een exception

4.1. Deel 1

Fouten gooien en afhandelen in applicatie



Schrijf een consoleapplicatie die een temperatuur in Fahrenheit omzet in een temperatuur in Celsius.

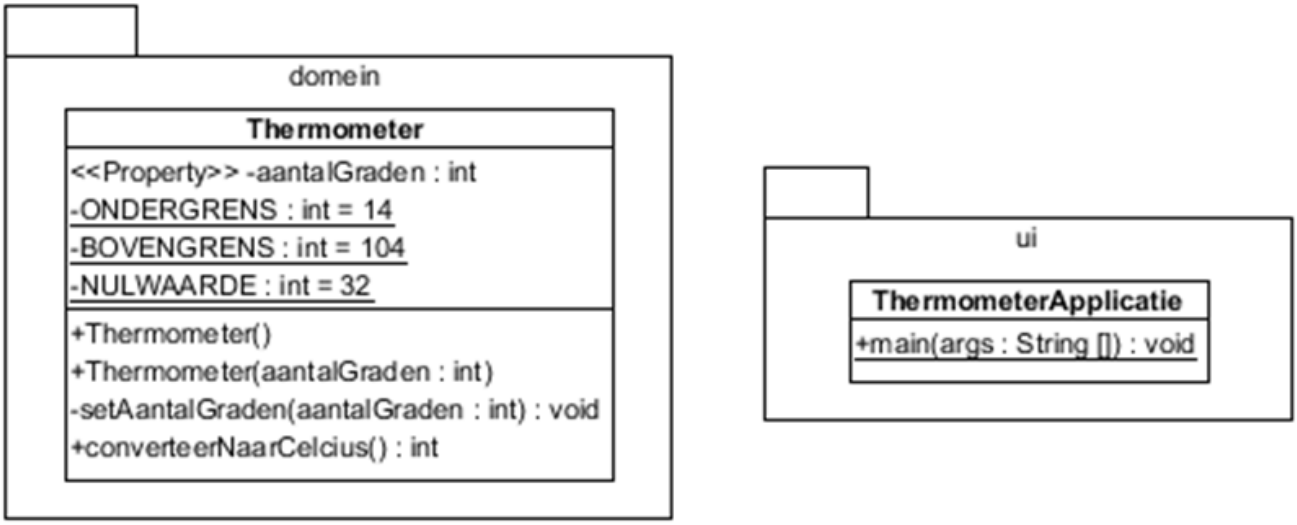
Gebruik de domeinklasse Thermometer om de temperatuur om te zetten! (°C = 5/9 * (°F – 32)). De defaultconstructor maakt een thermometer aan waarbij de temperatuur ingesteld is op 32°F.

Zorg ervoor dat een foutieve input (= niet numeriek, buiten de grenzen van het interval [14°F,104°F]) wordt gemeld. Handel alle fouten ter plaatse (= in de applicatie) af!

```
run:
Geef een temperatuur (eenheid Fahrenheit) in het interval [14,104]: blablabla
Foutieve invoer! Moet numeriek zijn!
Geef een temperatuur (eenheid Fahrenheit) in het interval [14,104]: 500
De temperatuur ligt niet in het gevraagde interval.
Geef een temperatuur (eenheid Fahrenheit) in het interval [14,104]: 10
De temperatuur ligt niet in het gevraagde interval.
Geef een temperatuur (eenheid Fahrenheit) in het interval [14,104]: 20
De temperatuur is -6°C
BUILD SUCCESSFUL (total time: 23 seconds)
```

4.2. Deel 2

Fouten gooien deels in domein en deels in applicatie, afhandelen in applicatie



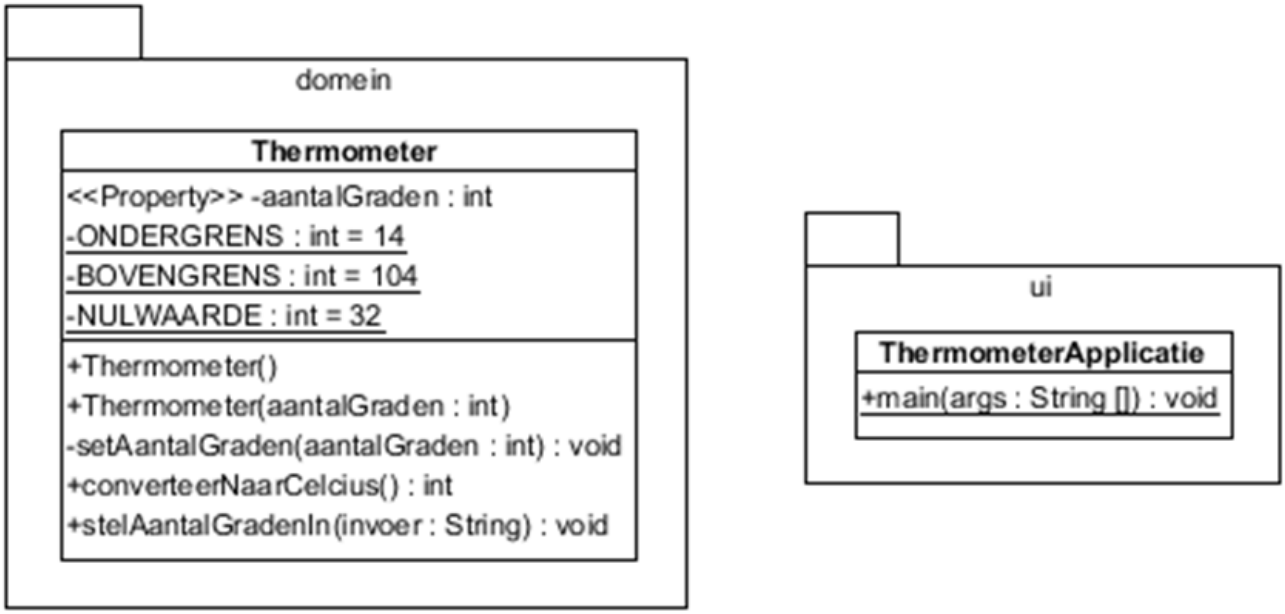
Wijzig deel 1 van de oefening

- Een niet-numerieke invoer handelen we nog altijd ter plaatse (= in de applicatie) af.
- Alle numerieke invoer wordt in eerste instantie aanvaard, maar in de klasse Thermometer moet op de waarde gecontroleerd worden. Blijkt de waarde buiten het interval te vallen, dan moet de (private) setter() een exceptie teruggooien. Deze wordt dan zoals in oefening1 afgehandeld.

De controle op het interval [14,104] is een domeinregel. Dit zijn de grenzen van de thermometer. Die controle MOET altijd in het domein zitten, het mag ook – extra bovenop de controle in het domein – opgenomen worden in de user interface.

4.3. Deel 3

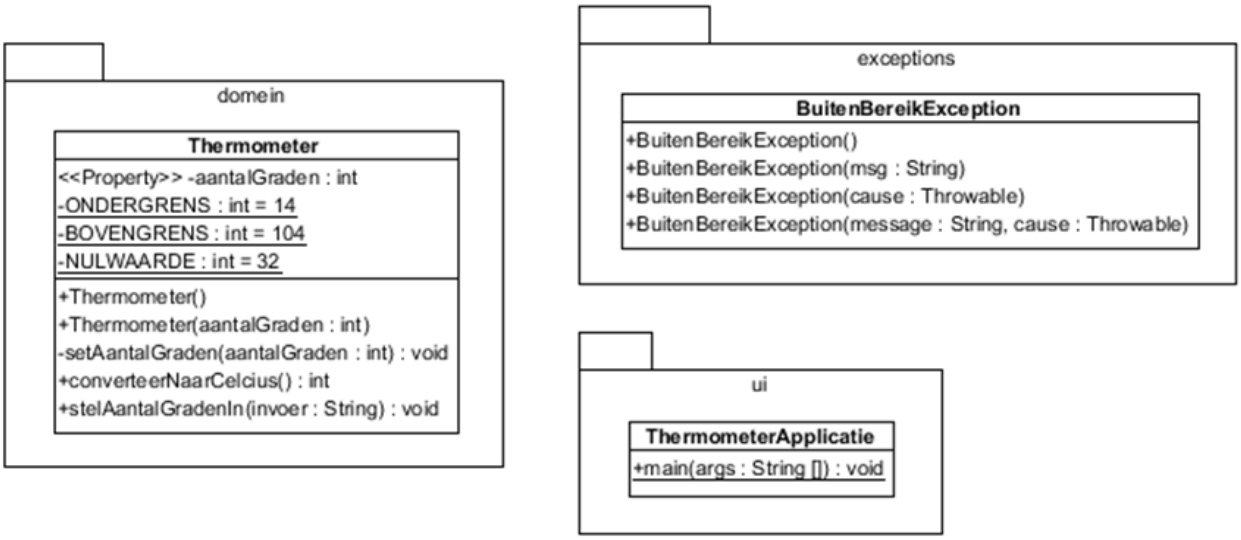
Fouten gooien in domein, afhandelen in applicatie



Herneem deel 2 van de oefening:

- Elke invoer van de gebruiker wordt doorgestuurd naar het domein. Daar wordt indien nodig een Exception gegooid als de invoer fout was.
- Maak hiervoor een nieuwe methode `stelAantalGradenIn(invoer:String):void`

4.4. Deel 4 - Schrijf je eigen Exception klasse



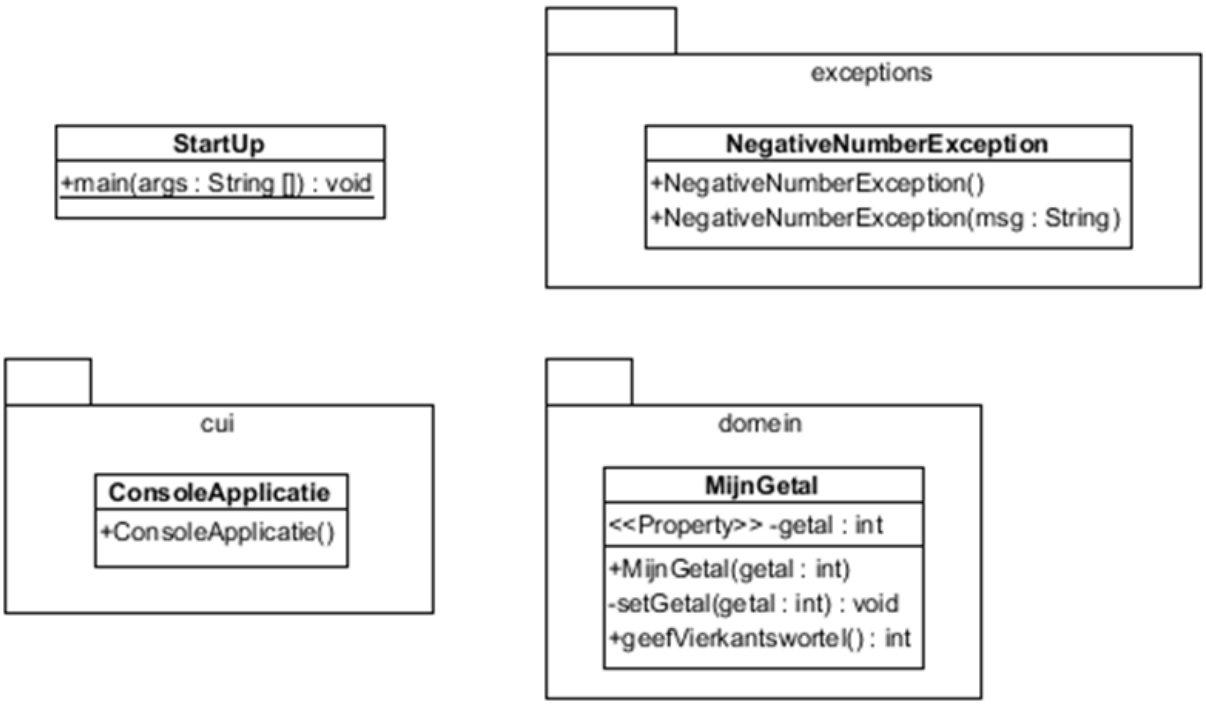
Herneem deel 3 van de oefening.

Schrijf je eigen exceptionklasse: BuitenBereikException, afgeleid van IllegalArgumentException

De exceptie wordt gegooid als de invoer niet voldoet aan de volgende voorwaarden: * minimum 14°F * maximum 104° F

Zorg ook voor een correcte afhandeling van deze fout.

5. Oefening - MijnGetal



Schrijf een eigen exceptieklasse NegativeNumberException, afgeleid van Exception

Gebruik deze klasse in een applicatie, die invoer krijgt via het toetsenbord.

De invoerlijn bevat normaal twee positieve gehele getallen. Splits de invoerlijn dus eerst op (zie tip). Krijg je geen twee elementen in de array, gooi dan een NoSuchElementException.

Probeer vervolgens de twee Strings om te zetten naar gehele getallen.

Als je zeker weet dat het om gehele getallen gaat, maak dan objecten van MijnGetal. Hierbij controleer je of ze wel degelijk positief zijn (controle in domeinklasse). Gooi een fout indien foutieve waarde.

Bepaal van deze beide getallen de gehele vierkantswortel.

```
run:
Geef twee gehele positieve getallen in (gescheiden door een spatie):
12
Twee getallen gescheiden door een spatie invoeren. Opnieuw!
Geef twee gehele positieve getallen in (gescheiden door een spatie):
a
Twee getallen gescheiden door een spatie invoeren. Opnieuw!
Geef twee gehele positieve getallen in (gescheiden door een spatie):
12.5 12.6
Zorg voor twee gehele getallen! Probeer opnieuw.
Geef twee gehele positieve getallen in (gescheiden door een spatie):
12 46
res1 = 3
res2 = 6
BUILD SUCCESSFUL (total time: 42 seconds)
```

Klasse String, methode split(...)

split

`public String[] split(String regex)`

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument `split` method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string `"boo:and:foo"`, for example, yields the following results with these expressions:

Regex	Result
:	{ "boo", "and", "foo" }
o	{ "b", "", ":and:f" }

Parameters:
`regex` - the delimiting regular expression

Returns:
the array of strings computed by splitting this string around matches of the given regular expression

Throws:
`PatternSyntaxException` - if the regular expression's syntax is invalid

Since:
1.4

See Also:
`Pattern`