HO
GENT

Exception handling Werkcollege

Ludwig Stroobant

Table of Contents

1. DivideByZeroWithExceptionHandling voorbeeld

Gegeven de klasse `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
- Duidelijke foutmelding voor de gebruiker uitschrijven
- 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

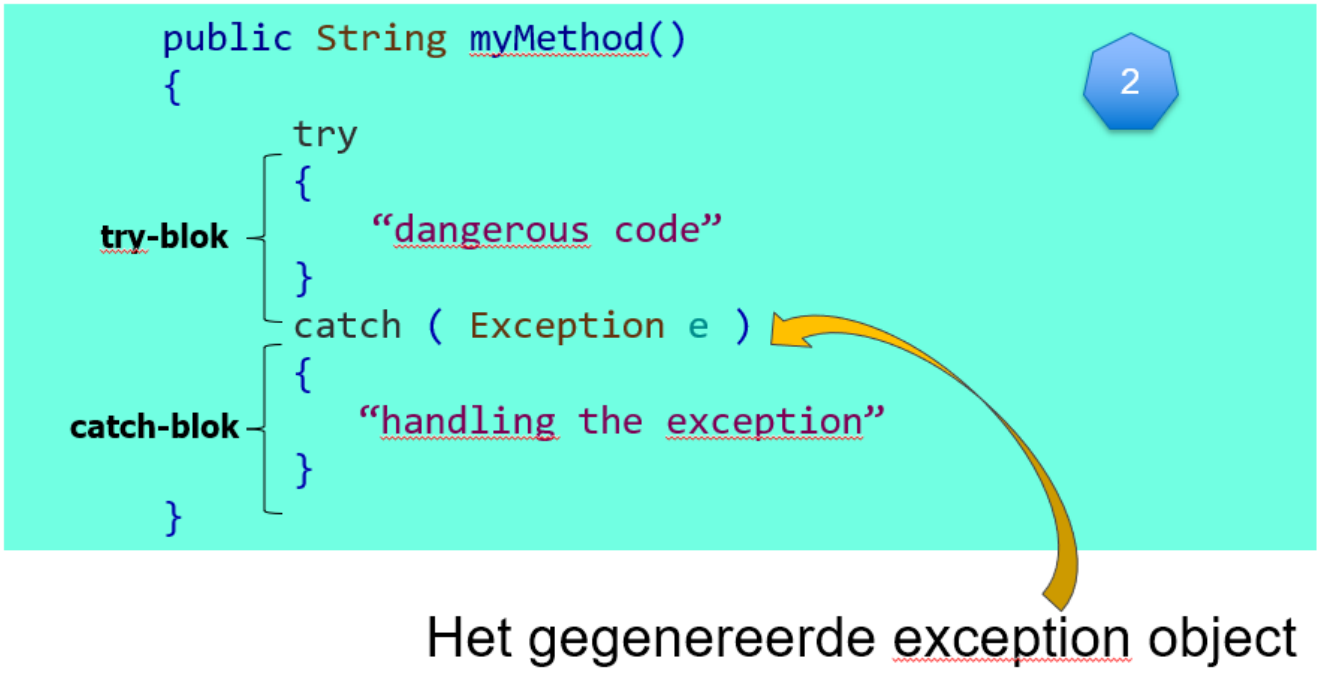
- Alle instructies waarin zich fouten kunnen voordoen en alle instructies die niet mogen uitgevoerd worden wanneer een exception optreedt komen in een try-blok
- als er een fout optreedt in het try-blok, dan wordt er een instantie van een exception-class gemaakt en deze kan opgevangen worden in één van 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]

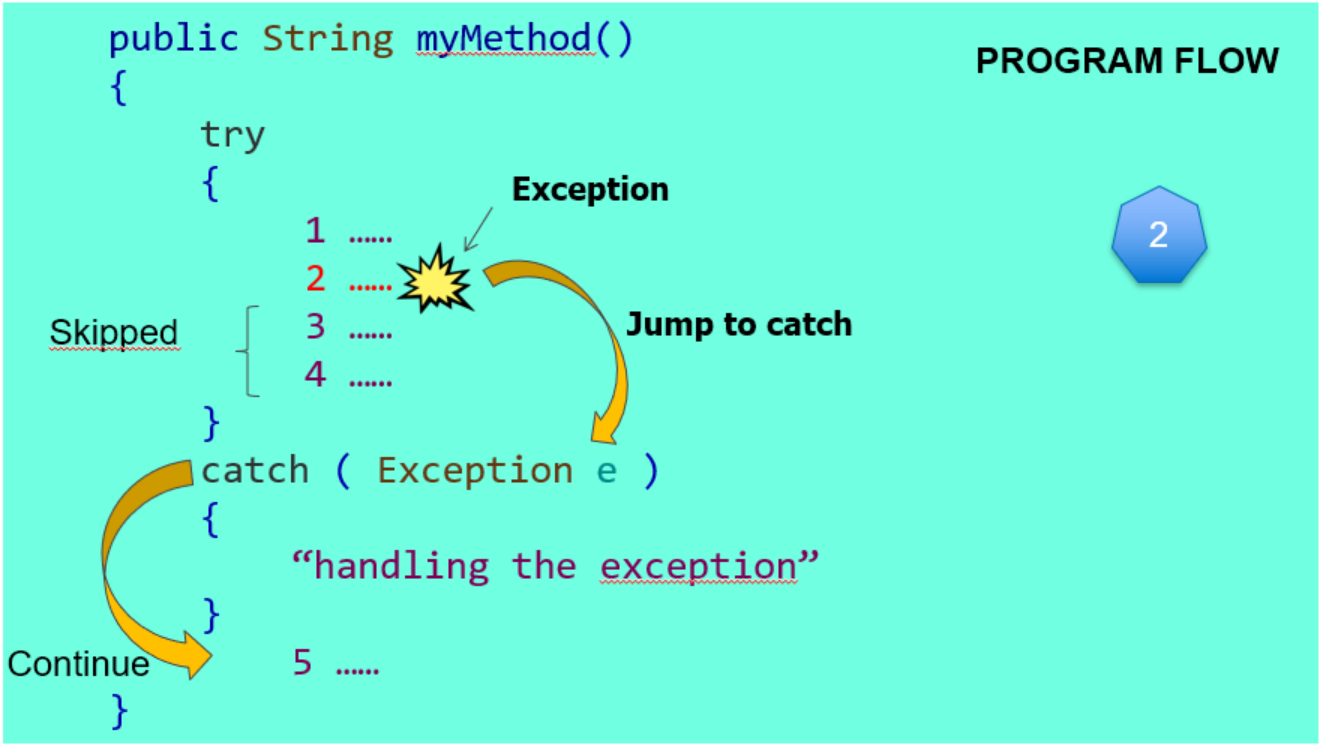
Het try-catch-finally statement:

```
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.





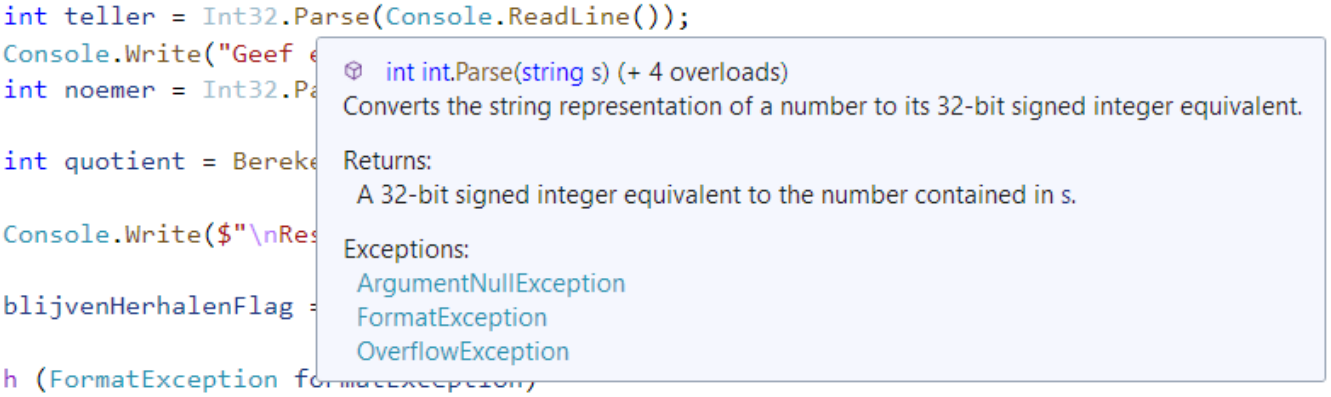
1.3. Stap voor stap

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

```
1     int teller = Int32.Parse(Console.ReadLine());
2     int noemer = Int32.Parse(Console.ReadLine());
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.
- Aan de signatuur van de methode zie je niet dat er een Exception kan optreden.

Onderstaande afbeelding toont info over de `Int32.Parse()` methode. Deze kan drie verschillende exception types gooien, maar handelt deze zelf niet af. Ze worden gepropageerd naar boven om op een hoger niveau te worden afgehandeld. M.a.w. diegene die de `Parse` methode oproept moet ook zijn mogelijke fouten afhandelen.



1.4. Verschillende exceptions afhandelen

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

- Een FormatException gegooid door Parse()
- Een ArithmeticException gegooid door berekenQuotient()

```
                try
                {
                    Console.Write("Geef een integere waarde voor de teller: ");
                    int teller = Int32.Parse(Console.ReadLine());
                    Console.Write("Geef een integere waarde voor de noemer: ");
                    int noemer = Int32.Parse(Console.ReadLine());

                    int quotient = BerekenQuotient(teller, noemer);

                    Console.Write($"\nResultaat: {teller} / {noemer} =
{quotient}\n");

                    blijvenHerhalenFlag = false;
                }
```

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

- Andere boodschap uitschrijven voor gebruiker.

```
                catch (FormatException formatException)
                {
                    Debug.Write("\nException: {0}\n", formatException.Message);

                    Console.Write("De invoer moeten integere getallen zijn. Probeer
opnieuw.\n\n");
                }
                catch (ArithmeticException arithmeticException)
                {
                    Debug.Write("\nException: {0}\n", arithmeticException.Message);
                    Console.Write("Het cijfer 0 kan geen noemer zijn. Probeer
opnieuw.\n\n");
                }
                catch (Exception exception) // ALL-catcher
                {
                    Console.Write("\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.

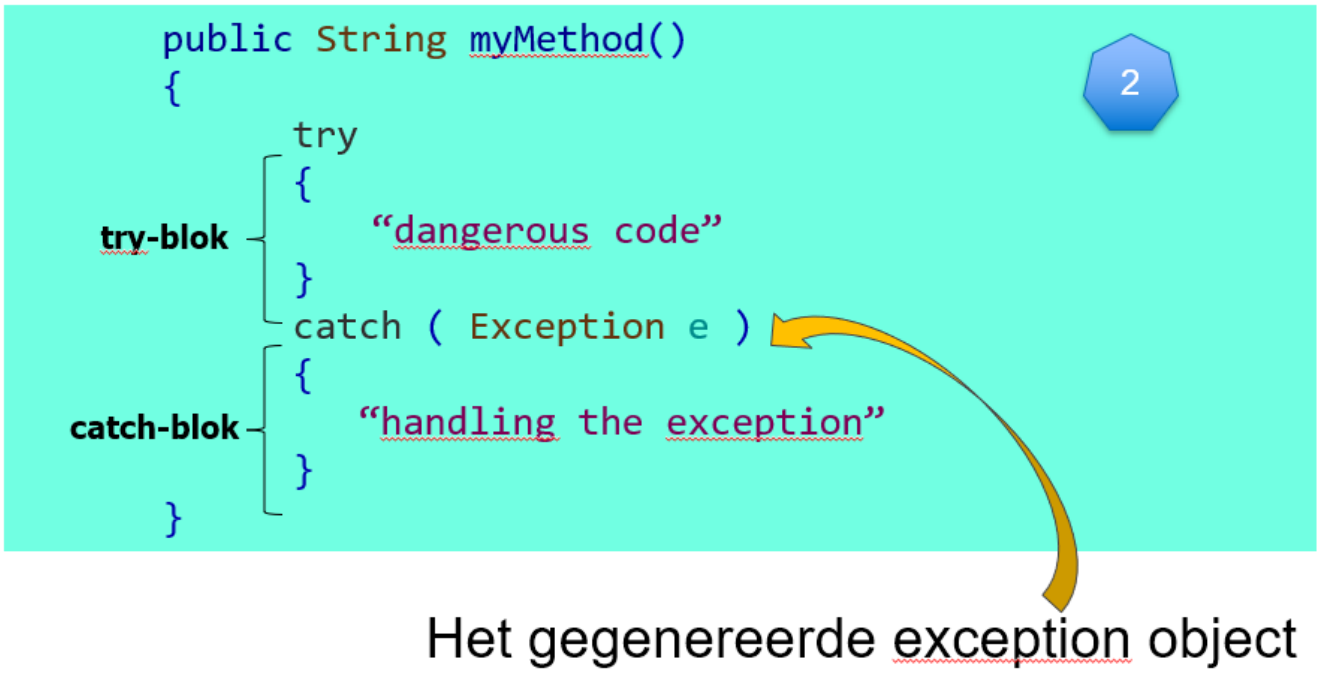
```
1               catch (Exception exception) // ALL-catcher
2               {
3                   Console.Write("\nException: %s\n", exception);
4               }
```

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.

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 van die superklasse!
- 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
4         public static int BerekenQuotient(int teller, int noemer)
5         {
6             return teller / noemer;
7         }
8
9         public static void Main(string[] args)
10        {
11
12            bool blijvenHerhalenFlag = true;
13            do
14            {
15                try
16                {
17                    Console.Write("Geef een integere waarde voor de teller: ");
18                    int teller = Int32.Parse(Console.ReadLine());
19                    Console.Write("Geef een integere waarde voor de noemer: ");
20                    int noemer = Int32.Parse(Console.ReadLine());
21
22                    int quotient = BerekenQuotient(teller, noemer);
23
24                    Console.Write($"\nResultaat: {teller} / {noemer} =
  {quotient}\n");
25
26                    blijvenHerhalenFlag = false;
27                }
28                catch (FormatException formatException)
29                {
30                    Debug.Write("\nException: {0}\n", formatException.Message);
31
32                    Console.Write("De invoer moeten integere getallen zijn. Probeer
  opnieuw.\n\n");
33                }
34                catch (ArithmeticException arithmeticException)
35                {
36                    Debug.Write("\nException: {0}\n", arithmeticException.Message);
37                    Console.Write("Het cijfer 0 kan geen noemer zijn. Probeer
  opnieuw.\n\n");
38                }
39                catch (Exception exception) // ALL-catcher
40                {
41                    Console.Write("\nException: %s\n", exception);
42                }
43            } while (blijvenHerhalenFlag);
44        }
45    }
```

2. Even tussendoor

Wat wordt uitgeschreven op het scherm?

```
1     public void MyMethod()
2     {
3         try
4         {
5             int aNumber;
6             aNumber = Int32.Parse("this is not a number");
7             Console.Writeln("Wrong input");
8         } catch (NumberFormatException e)
9         {
10            Console.Writeln("The given number is not a number");
11        }
12        Console.Writeln("Continue");
13    }
```

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? Bekijk de code en voorspel de flow bij uitvoer.

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.

Ga met de debugger na of de voorspelde flow ook gevolgd wordt.

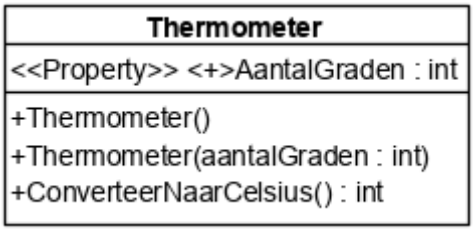
- Uitvoeren van try-blok
  - Er treedt een Exception op. Dit object wordt hier aangemaakt en gegooid via de instructie **throw**
- Catch-blok vangt de Exception op
  - Het afhandelen hier bestaat uit iets afdrukken. Verder afhandelen is niet mogelijk, datzelfde object wordt verder gegooid. *Finally-blok wordt als laatste stap uitgevoerd.
- 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.

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.

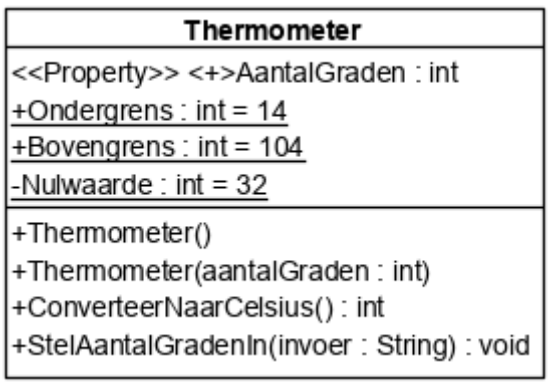
Gebruik de domeinklasse Thermometer om de temperatuur om te zetten! (°C = 5/9 * (°F – 32)). De defaultconstructor maakt een thermometer aan waarbij de temperatuur ingesteld is op 32°F.

Zorg ervoor dat een foutieve input (= niet numeriek, buiten de grenzen van het interval [14°F,104°F]) wordt gemeld. Handel deze 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



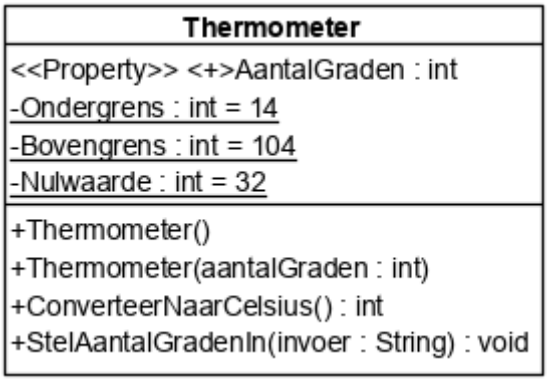
Wijzig deel 1 van de oefening

- 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

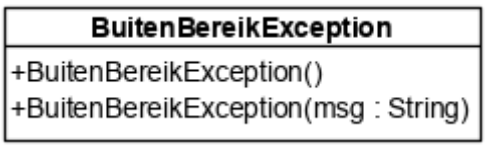
Fouten gooien in domein, afhandelen in applicatie



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 Exception

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.