# Unchecked/Checked exceptions

**Unchecked exceptions**

Unchecked exceptions zijn fouten in het programma(bugs). Deze exceptions zijn subclasses van de class RunTimeException.

Methodes hoeven hier niet rekening mee te houden, aangezien het bugs zijn en als je van de bug af weet dan kun je beter de bug fixen ipv methods aanpassen.

**Checked exceptions**

Checked exceptions zijn fouten buiten het programma, zoals invalid user inputs, database problemen, netwerk storing, en absente bestanden.

Checked exceptions zijn subclasses van de class Exception

Methodes moeten hier rekening mee houden.

**Wanneer moet je exceptions gebruiken**

Als je denkt dat er een kans is dat een stukje code errors gaat geven doordat bijvoorbeeld een filepath/schrijf niet meer bestaat of dat er foute user input kan gebeuren. Wanneer er een exception error komt die niet wordt opgevangen in een exception kapt je hele programma ermee met een “Uncaught Exception…”.

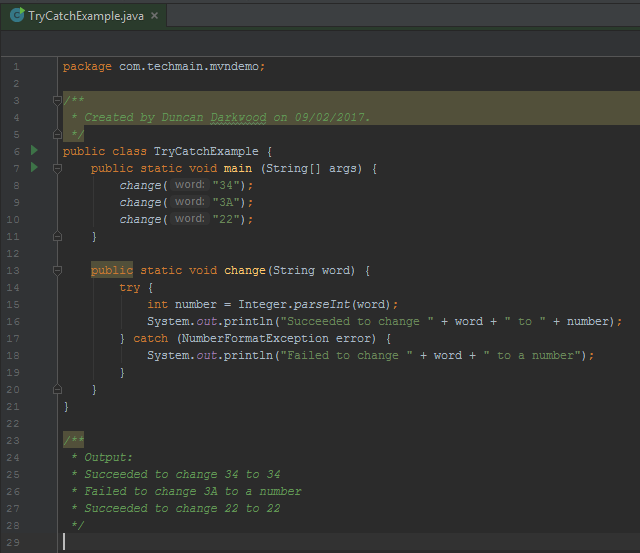
Hier nog een 20 minuten video die aanvullende en uitgebreide informatie geeft op het gebruik van exceptions. <https://www.youtube.com/watch?v=R86ObiKhMNc>

# 

# Try/Catch

Het idee van try/catch is vrij simpel. Je doet ongeveer hetzelfde als in een if/else structuur, maar dan met try/catch. Je probeert iets in try, en als dat niet werkt dan ga je naar catch. Wanneer er in try een error komt, wordt er in catch iets uitgevoerd, indeed er in try geen error komt, dan wordt er niets met catch gedaan.

Hieronder een voorbeeld van een try/catch structuur



Zoals te zien is, is poging 2, het veranderen van 3A naar een nummer, niet gelukt. Hierbij is het programma het catch stuk ingeslagen.

Als laatste, na een catch kun je een finally plaatsen. Alles wat er in de finally staat wordt ten alle tijden uitgevoerd, of de try is gelukt of niet.



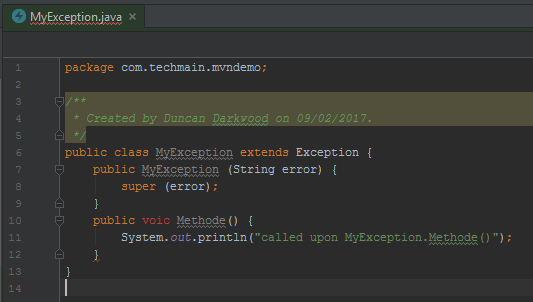
Hierboven is een try te zien met 2 mogelijke catches. Daarna is de finally te zien.

Wat hier het nut van is is dat je aan het eind altijd de output -\_-\_-\_-\_- hebt in de console (in dit voorbeeld), zonder dat het uitmaakt wat ervoor gebeurd is.

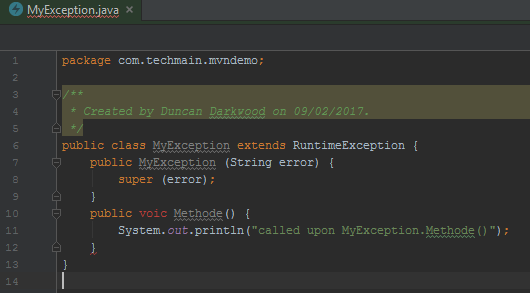
# Defineren

Voor het definiëren van een eigen exception moet er eerst bepaald worden of het een checked, of unchecked exception variant is. Dit moet omdat elke eigen exception een class is die extend naar een bestaande exception class. “Exception” voor checked exceptions en “RuntimeException” voor unchecked exceptions.

Hieronder een voorbeeld voor een checked exception

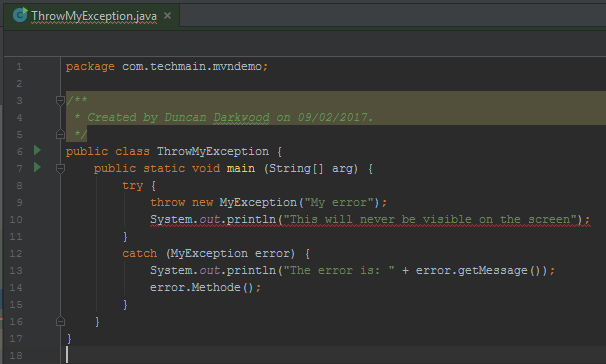


Hieronder een voorbeeld van een unchecked exception



# Throws/Throw

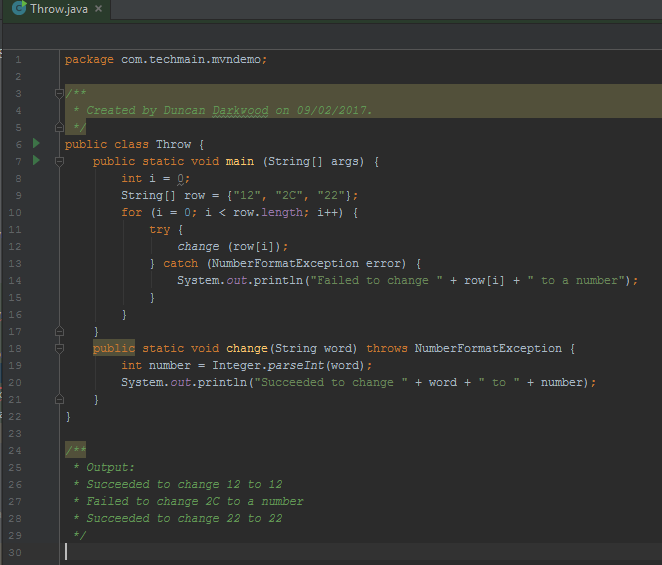
Met Throws en Throw worden er exceptions gegooid. Dit is om de gebruiker te informeren over een exception die is opgetreden en moet worden afgehandeld door de aanroeper.

Met als voorbeeld van ons eigen gedefineerde Exceptions kan Throw worden gebruikt.

Hier is te zien dat door middel van een try/catch de zelf gedefinieerde exception genaamd “MyException” wordt gegooid.

Er kan ook gebruik gemaakt worden van Throws, een keyword dat wordt gebruikt in een methode. Deze methode kan in de try worden aangeroepen, en geeft een specifieke exception terug in de catch die dan wordt afgehandeld.

Dit ziet er ongeveer zo uit



Zoals in dit voorbeeld te zien is, wordt er in try een methode aangeroepen genaamd “change”. Als er in deze methode een exception voorkomt, dan gooit (“throws”) hij het terug in de vorm van een NumberFormatException. Deze wordt bij catch dan weer uitgelezen en uitgevoerd.

Het verschil tussen throw en throws wordt op deze pagina uitgebreid uitgelegt http://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/

In je App.java zet je dit

**package** com.techmain.mvndemo;

**import** java.util.InputMismatchException;

**import** java.util.Scanner;

*/\*\**

*\* Hello world!*

*\*/*

**public class** App {

**public static void** main(String[] args) {

**int** i = 0;

**boolean** done = **false**;

**do** {

**try** {

Scanner input = **new** Scanner(System.***in***);

System.***out***.println(**"enter an integer under or equal to 100"**);

i = input.nextInt();

**if** (i > 100) {

**throw new** intOutOfRangeException();

}

System.***out***.printf(**"you entered %d\n"**, i);

} **catch** (InputMismatchException e) {

System.***out***.println(**"That's not an integer, try again."**);

} **catch** (intOutOfRangeException e) {

System.***out***.println(**"That integer is too high, try again."**);

}

**finally** {

System.***out***.println(**"-\_-\_-\_-\_-"**);

}

} **while** (!done) ;

}

}

In je intOutOfRangeException class zet je

**package** com.techmain.mvndemo;

*/\*\**

*\* Created by allard on 09/02/2017.*

*\*/*

**public class** intOutOfRangeException **extends** Exception{

}