# Programmeren Gevorderd

Les 3 // Design class diagram

# Design Class Diagrams

# DCD 101

Unified Modeling Language (UML) Schema

Al gezien voor één klasse, breiden nu uit

Analyseren en ontwerpen van klassen voor implementatie

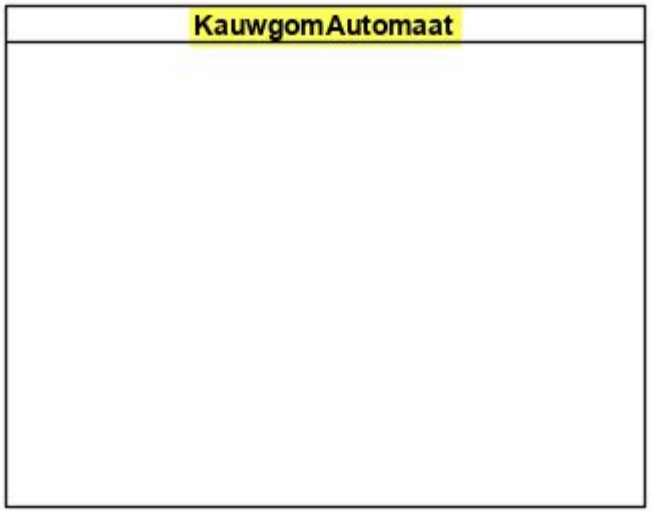
Kan gebruikt worden als documentatie

Uitleg aan de hand van praktisch voorbeeld

Relaties tussen klassen in applicatie

# Herhaling klassediagram

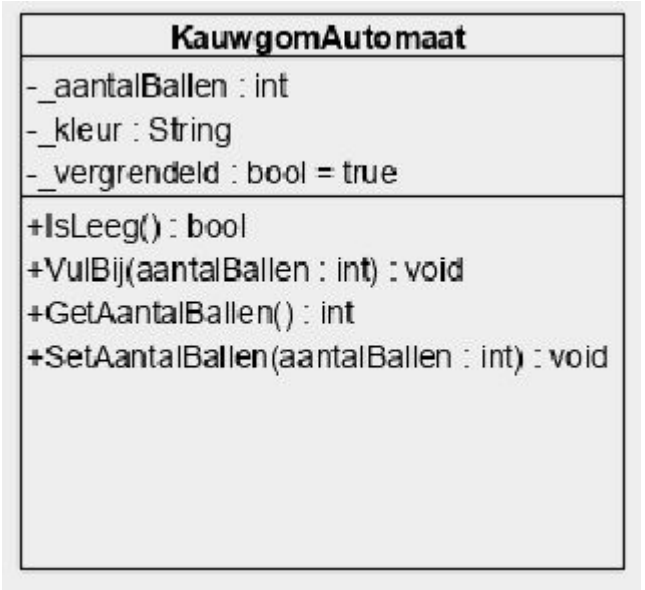
# KlasseNaam



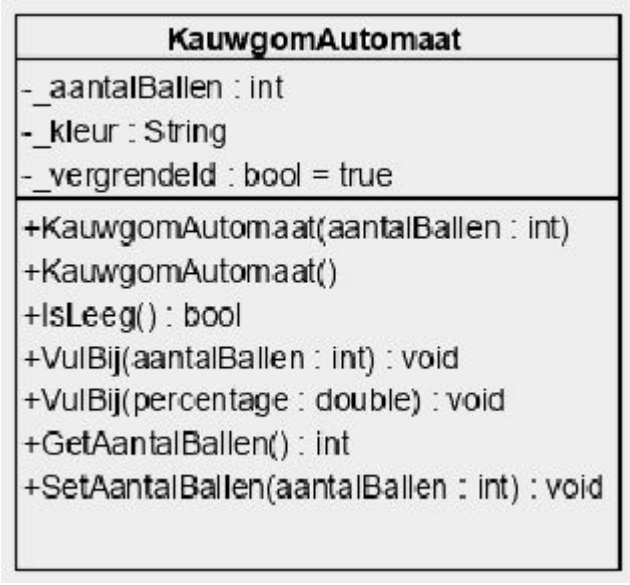
# Fields


KauwgomAutomaat
-_aantalBallen : int
-_kleur : String
-_vergrendeld : bool = true

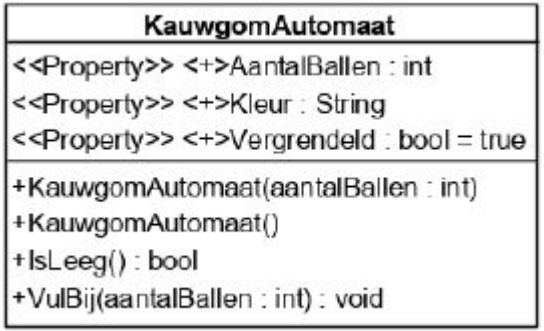
# Methods


KauwgomAutomaat
-_aantalBallen : int
-_kleur : String
-_vergrendeld : bool = true
+IsLeeg() : bool
+VulBij(aantalBallen : int) : void
+GetAantalBallen() : int
+SetAantalBallen(aantalBallen : int) : void

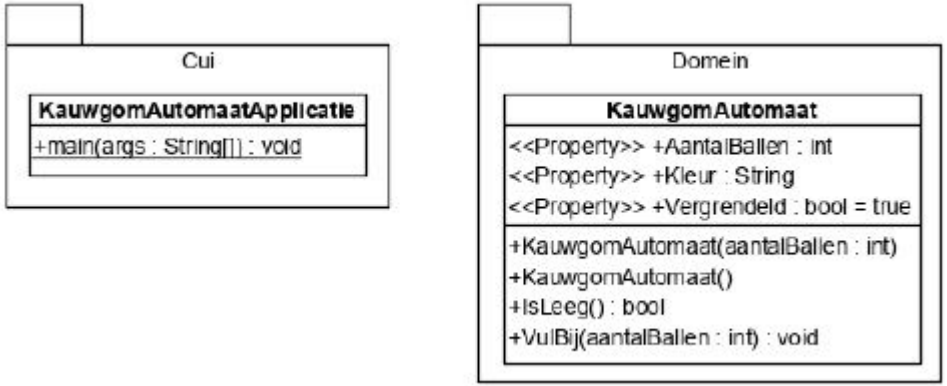
# Constructor



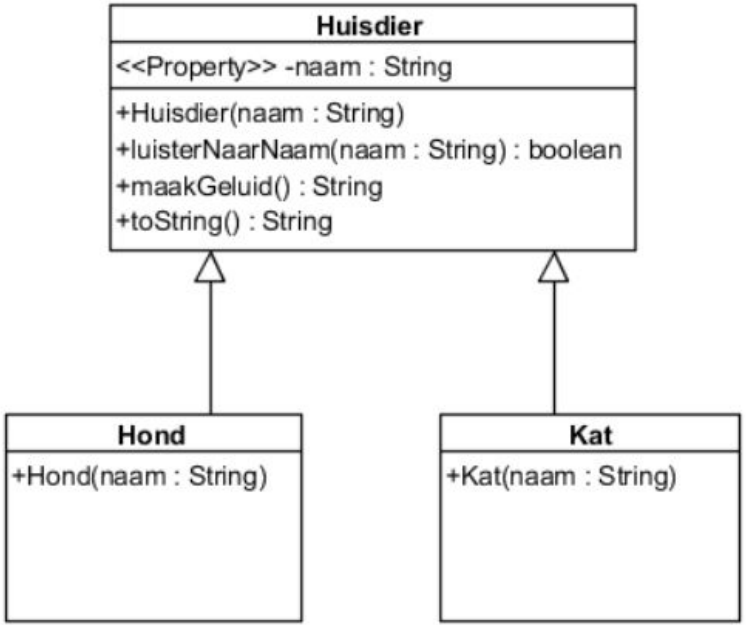
# Properties


KauwgomAutomaat
<<Property>> <+>AantalBallen : int
<<Property>> <+>Kleur : String
<<Property>> <+>Vergrendeld : bool = true
+KauwgomAutomaat(aantalBallen : int)
+KauwgomAutomaat()
+IsLeeg() : bool
+VulBij(aantalBallen : int) : void

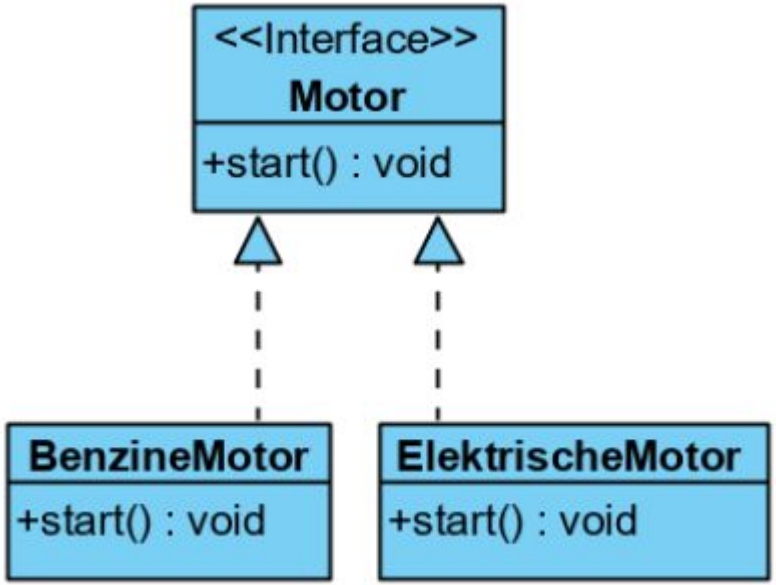
# Applicatie UML



# Overerving


Huisdier
<<Property>> -naam : String
+Huisdier(naam : String)
+luisterNaarNaam(naam : String) : boolean
+maakGeluid() : String
+toString() : String
Hond
+Hond(naam : String)
Kat
+Kat(naam : String)

# Interface implementatie



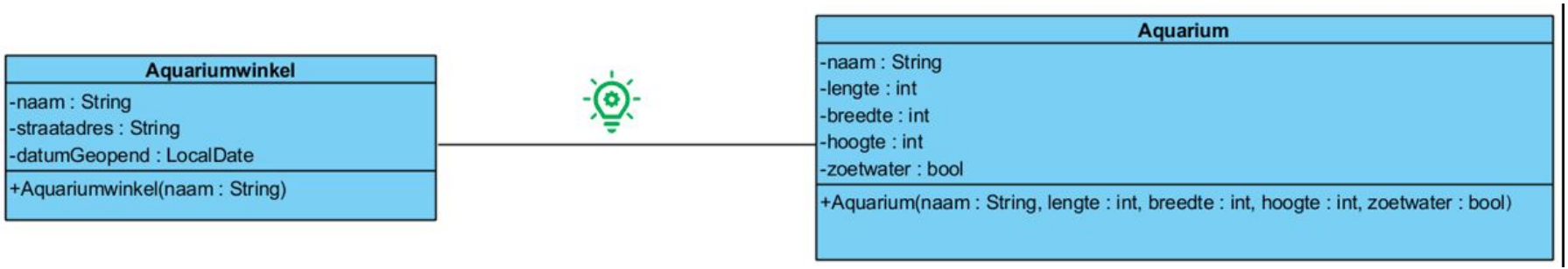
# Uitbreiding: relaties tussen klassen

# Praktisch voorbeeld: Aquarium Winkel

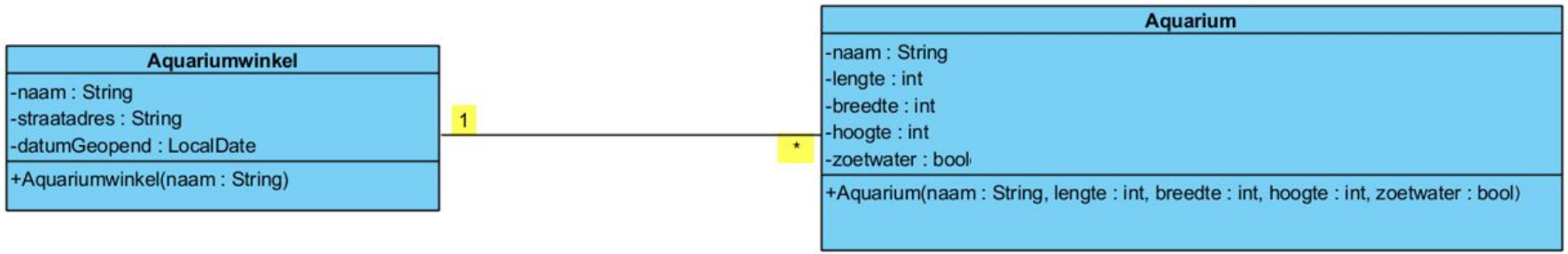
- AquariumWinkel
  Bevat naam, straatadres en datumGeopend
  Bevat ook meerdere aquariums
- Aquarium
  Bevat unieke naam, zout of zoutwater, lengte, breedte, hoogte

# AquariumWinkel DCD

# Associatie


Aquariumwinkel
-naam : String
-straatadres : String
-datumGeopend : LocalDate
+Aquariumwinkel(naam : String)
Aquarium
-naam : String
-lengte : int
-breedte : int
-hoogte : int
-zoetwater : bool
+Aquarium(naam : String, lengte : int, breedte : int, hoogte : int, zoetwater : bool)

# Associatie Multipliciteit



# Mogelijke multicipliteiten

* -> meerdere

1 -> exact 1

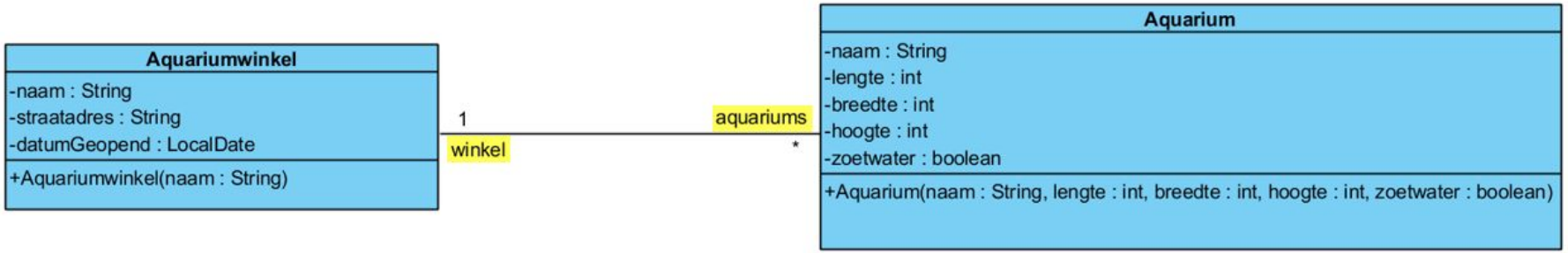
0..1 -> maximaal 1, maar mogelijks geen

1..* -> meerdere, maar minimaal 1

n -> meerdere zijnde exact n (n = integer >= 0)

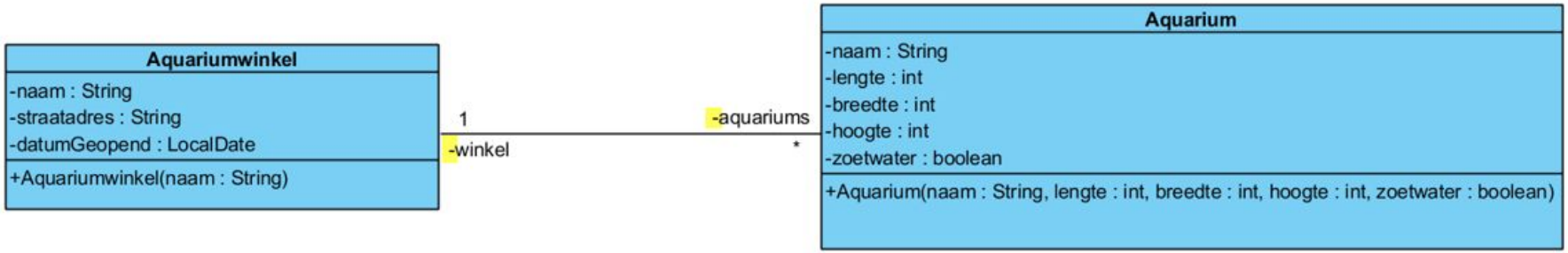
n..m meerdere, minimaal n en maximaal m

0..* komt overeen met *




Volg de regels van de naamgeving van fields in een klasse

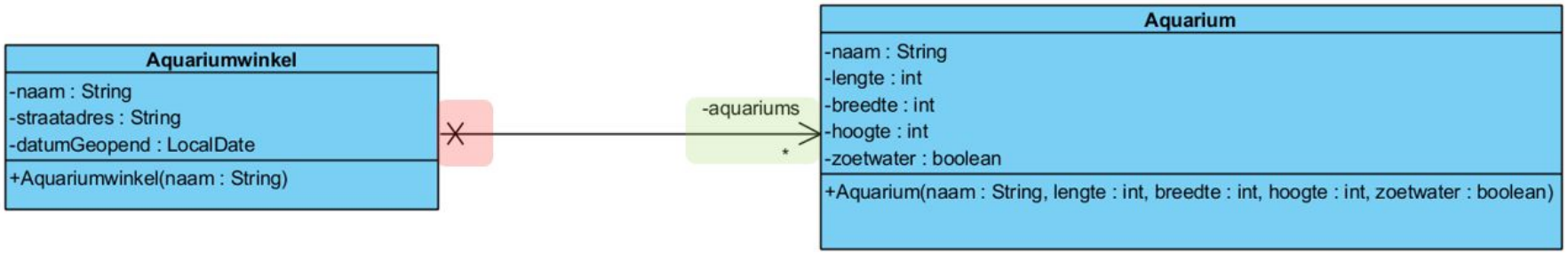
# Associatie Visibiliteit




Encapsulatie: maak associaties enkel toegankelijk binnen object.

Toch externe aanpassingen? Maak methodes.

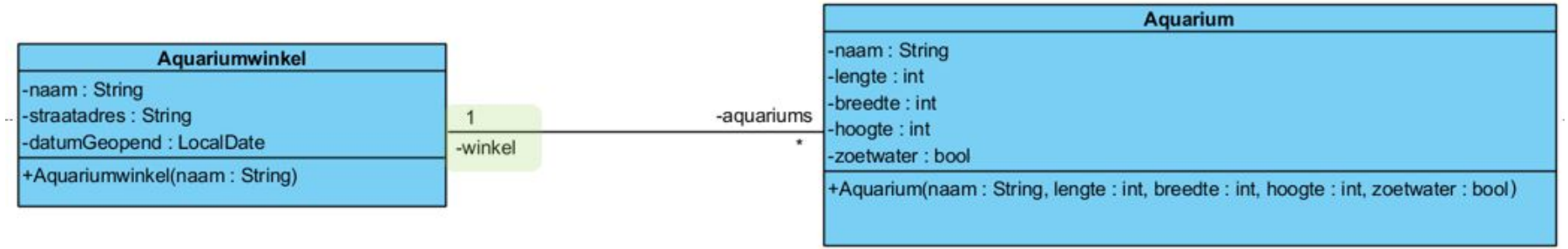
# Associatie Navigeerbaarheid




In welke richting wordt relatie gelegd?

Kruis = Aquarium houdt niet bij in welke winkel hij staat

Pijl = Winkel weet hoeveel en welke aquariums gekoppeld zijn




Geen pijltjes of kruisjes indien beide associaties elkaar kunnen zien

# Exceptions

# Soorten fouten

1. Syntaxfouten
    a. Applicatie compileert niet
    b. Opgevangen door compiler
2. Logische fouten
    a. Applicatie compileert
    b. Programma werkt maar doet niet wat we willen
3. Runtime fouten
    a. Applicatie compileert
    b. (deel van) programma stopt

# Exception

Exception -> opgetreden fout

Niet negatief, gemaakt om je te helpen voorzien op het onvoorziene

- Interne code issues -> vallen op te lossen
- Externe problemen -> vallen niet altijd op te lossen, enkel op te vangen

# Wat doen we nu?

```
MethodeA();
if (MethodeA misliep)
{
// handel het methodeA-probleem af
}
else
{
    MethodeB();
    if (methodeB misliep)
    {
        // handel het methodeB-probleem af
    }
    else
    {
        MethodeC();
        if (methodeC misliep)
        {
            // handel het methodeC-probleem af
        }
    }
}
```

# Betere manier?

Exception handling:

**Try** -> probeer iets te doen

**Catch** -> vang mogelijke problemen op

**Finally** -> voer iets sowieso uit

# Voorbeeld: ParseDouble

# Wat gebeurt er precies

Try blok rond te controleren code

Fout treedt op -> code in try blok wordt direct afgebroken

Springen naar een catch blok -> exception afhandelen

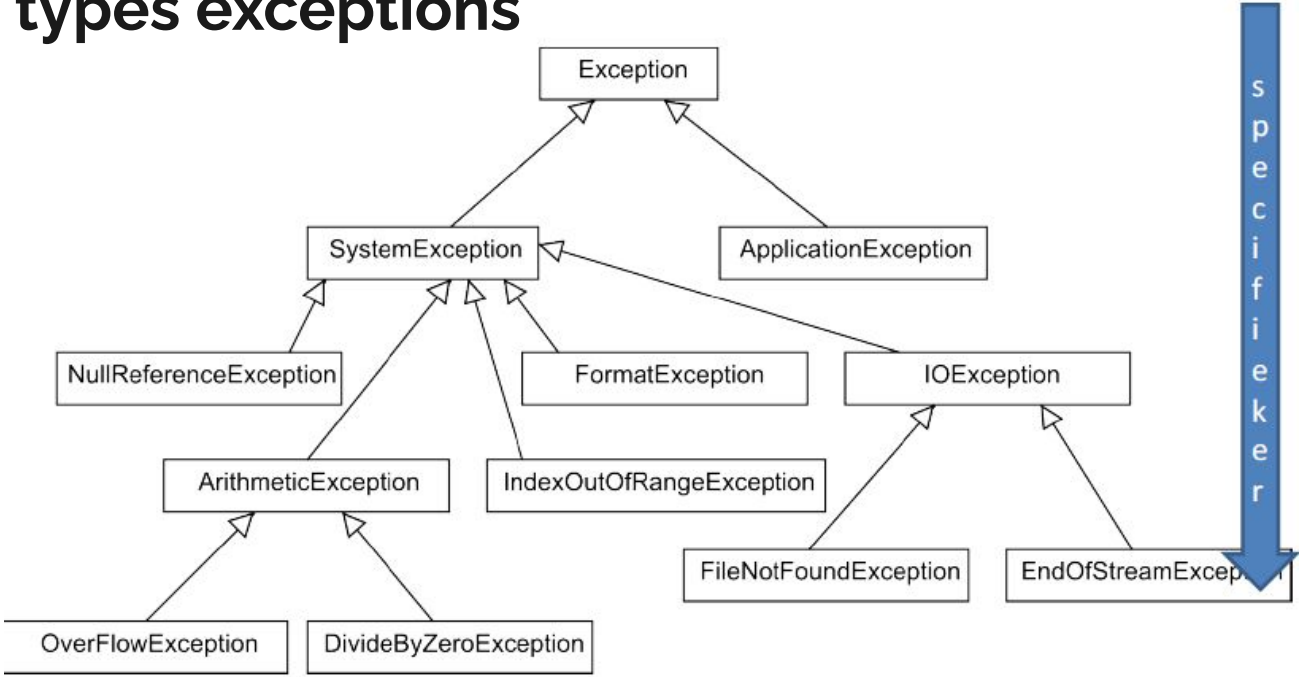
Exception niet opgevangen? Throwen naar oproepende methode

# Exception object bevat nuttige info

Ook tijdens ontwikkeling -> helpt je zoeken waar fouten zitten

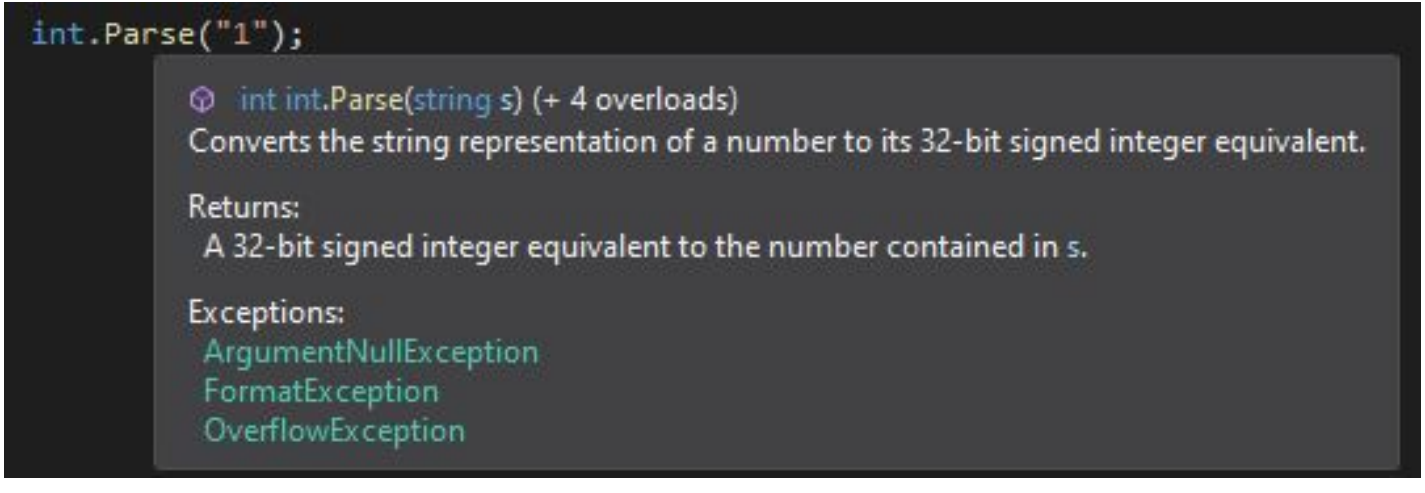
- Message: korte uitleg
- StackTrace: hierarchie van functies die tot exception hebben geleid
- ToString(): Uitprinten als tekst

# Verschillende  types exceptions



# Tip!

Exceptions die bestaande methodes kunnen throwen zijn gedocumenteerd.



# Meerdere soorten exceptions afhandelen

Meerdere catch blokken mogelijk om andere exception types op te vangen

Worden afgelopen in volgorde tot één matched

Verschillende mogelijkheden:

1. Verschillende catches die specifieke types opvangen
2. Catch die overkoepelende klasse opvangt (IOException)
3. Algemeen Exception type opvangen (vangt alles)

Exception propagation -> gaat omhoog in stack trace tot matchende catch gevonden wordt voor Exception type

# Voorbeeld: ExceptionHandling

# Afhandelen best practices

Zinvolle foutmelding voor gebruiker (ook al is de fout niet hun schuld)

Exceptions loggen -> in logfile of externe dienst

**Nooit lege catch statements om exceptions te negeren**

```
try
{
    int.Parse("boot");
}
catch (Exception)
{

}
```

# Finally

Wordt altijd uitgevoerd

- Zelfs bij return in try
- Zelfs bij niet opgevangen catch

Achteraan try-catch blok

**Dus**: beter dan gewoon na try-catch schrijven

# Zelf exceptions schrijven

Applicatie specifieke problemen

Overerven van ApplicationException

Best practice: constructor overerving waar je message aan kan meegeven

# Voorbeeld: CustomException