

The background features abstract green geometric shapes. On the left, a solid green trapezoid points upwards. On the right, a complex arrangement of overlapping translucent green triangles and polygons creates a layered, crystalline effect. The text is centered in a clean, green, sans-serif font.

Object Oriented programming

Deze les

- ▶ UML: State transition diagrams
- ▶ Lift probleem uitwerken
- ▶ Sorteren en zoeken



AddKnoop

- ▶ Vorige week: Fout in ontwerp
- ▶ Deze logica hoort in de knoop te staan, niet in de boom
- ▶ Encapsulation: inkapseling van logica
 - ▶ De boom hoeft niet te weten hoe een knoop zijn werk doet

```
internal class BinaireBoom
{
    Knoop? root = null;
    3 references
    private void Add(Knoop NieuweKnoop, Knoop BestaandeKnoop)
    {
        if (NieuweKnoop.Data < BestaandeKnoop.Data)
        {
            // ga naar links
            if (BestaandeKnoop.Links == null)
            {
                BestaandeKnoop.Links = NieuweKnoop;
            }
            else
            {
                Add(NieuweKnoop, BestaandeKnoop.Links);
            }
        }
        else // ga naar rechts
        {
            if (BestaandeKnoop.Rechts == null)
            {
                BestaandeKnoop.Rechts = NieuweKnoop;
            }
            else
            {
                Add(NieuweKnoop, BestaandeKnoop.Rechts);
            }
        }
    }
}
```

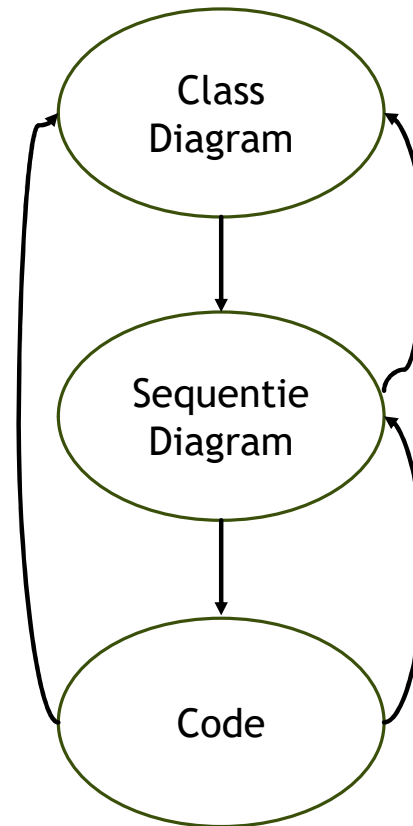
AddKnoop

- ▶ Methode in Knoop
- ▶ Geen parameter voor BestaandeKnoop
- ▶ Hetzelfde geldt voor de methode Zoek.

```
public void Add(Knoop NieuweKnoop)
{
    if (NieuweKnoop.Data <= Data)
    {
        // ga naar links}
        if (Links == null)
        {
            Links = NieuweKnoop;
        }
        else
        {
            Links.Add(NieuweKnoop);
        }
    }
    else // ga naar rechts
    {
        if (Rechts == null)
        {
            Rechts = NieuweKnoop;
        }
        else
        {
            Rechts.Add(NieuweKnoop);
        }
    }
}
```

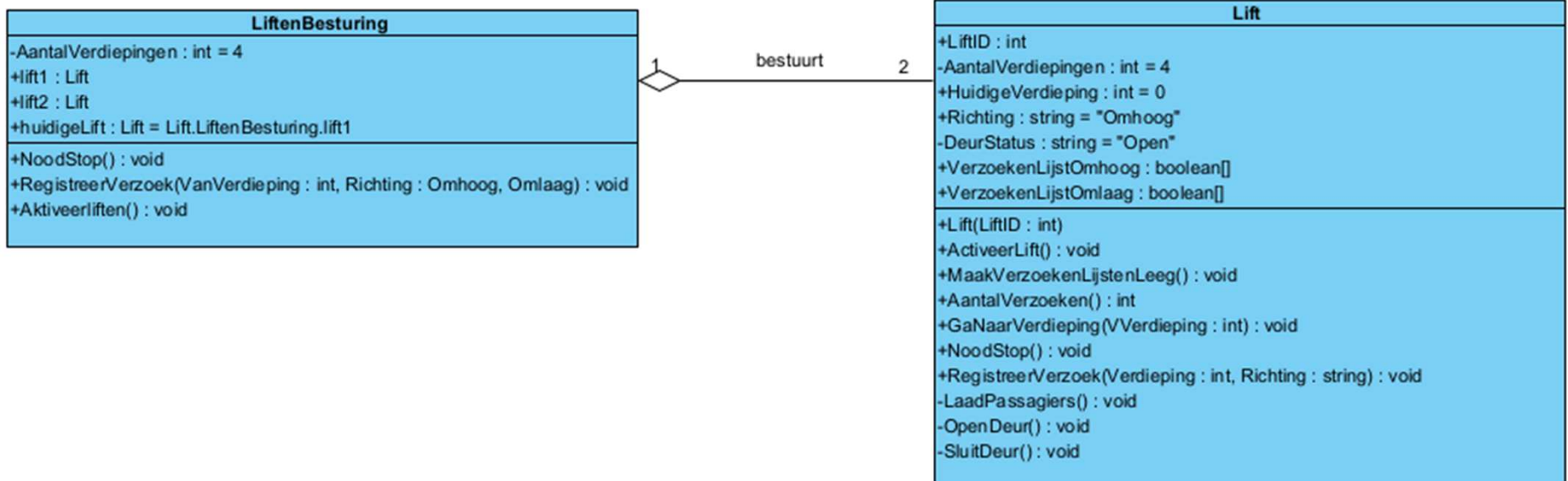
UML opdracht

- ▶ Liftcasus
 - ▶ Staat op Moodle
- ▶ Class diagram maken
- ▶ Sequence diagram(men) maken
- ▶ Werk iteratief
- ▶ Gebruik paradigm
- ▶ Mag in tweetallen
- ▶ Bouwen
- ▶ Zijn er al presentaties?



Lift Klassendiagram

- Console = paneel



Liftbesturing

- ▶ Twee liften
- ▶ Aktiveerliften
 - ▶ Technische oplossing

LiftenBesturing
-AantalVerdiepingen : int = 4 +lift1 : Lift +lift2 : Lift +huidigeLift : Lift = Lift.LiftenBesturing.lift1
+NoodStop() : void +RegistreerVerzoek(VanVerdieping : int, Richting : Omhoog, Omlaag) : void +Aktiveerliften() : void

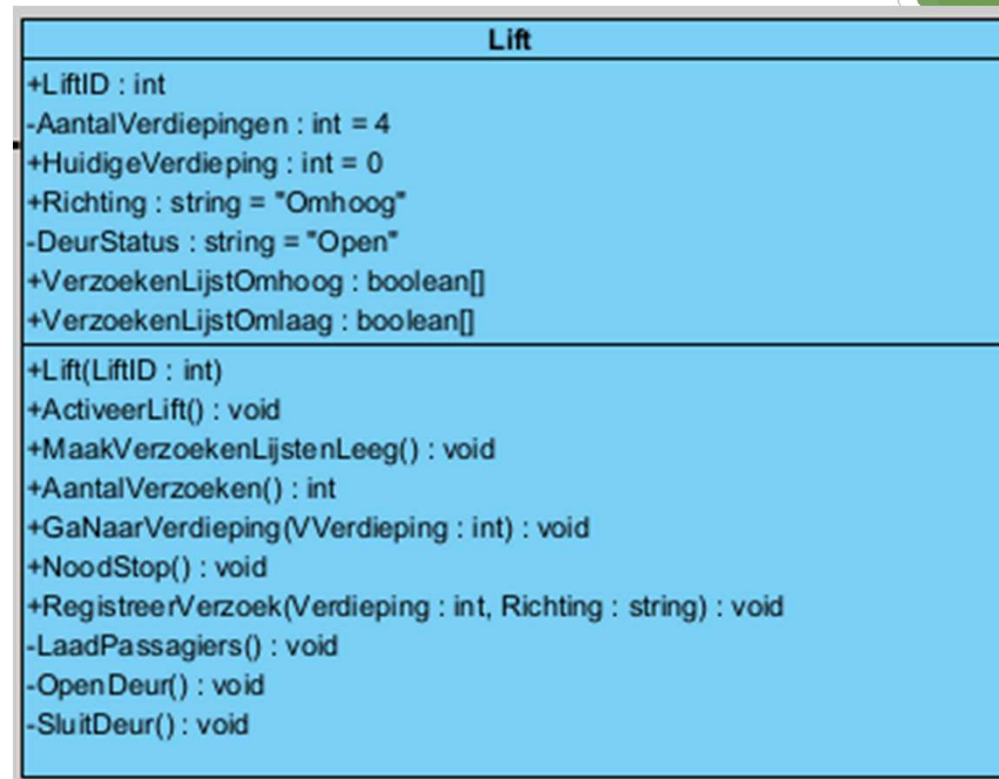
Lift

► Attributen

Lift
+LiftID : int
-AantalVerdiepingen : int = 4
+HuidigeVerdieping : int = 0
+Richting : string = "Omhoog"
-DeurStatus : string = "Open"
+VerzoekenLijstOmhoog : boolean[]
+VerzoekenLijstOmlaag : boolean[]

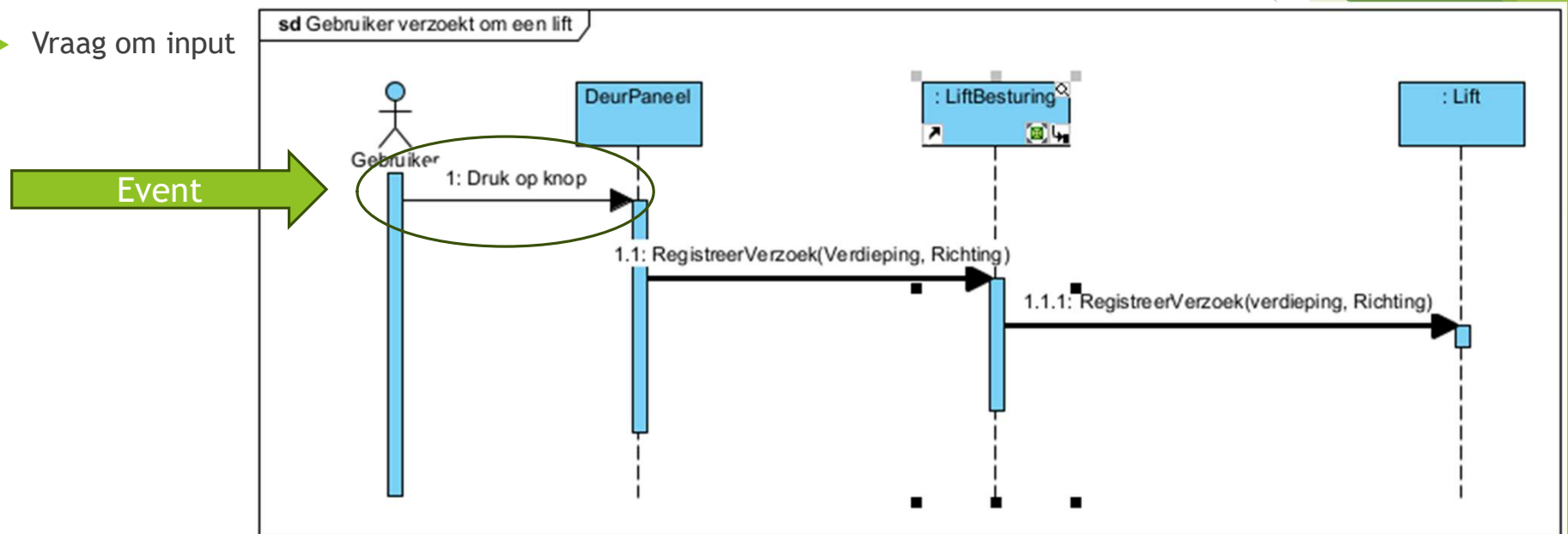
Lift

- ▶ Methodes
- ▶ Conventie
 - ▶ Eerst constructor(s)
 - ▶ Dan public methoden
 - ▶ Daarna private



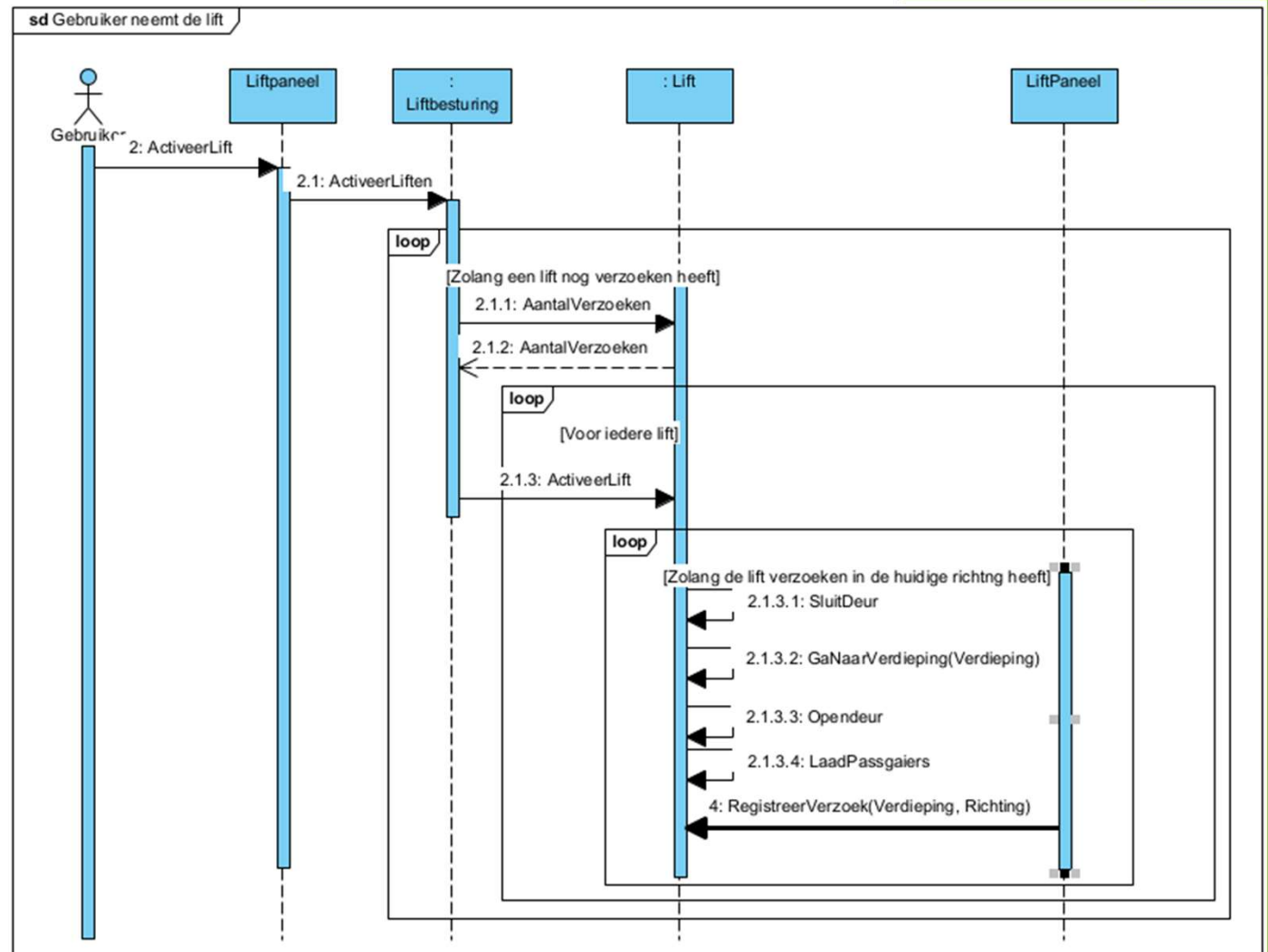
Sequentie diagram

- Use case: Gebruiker wil met de lift mee
- Functioneel: deurpaneel
 - Event driven
- Technisch: console
 - Vraag om input



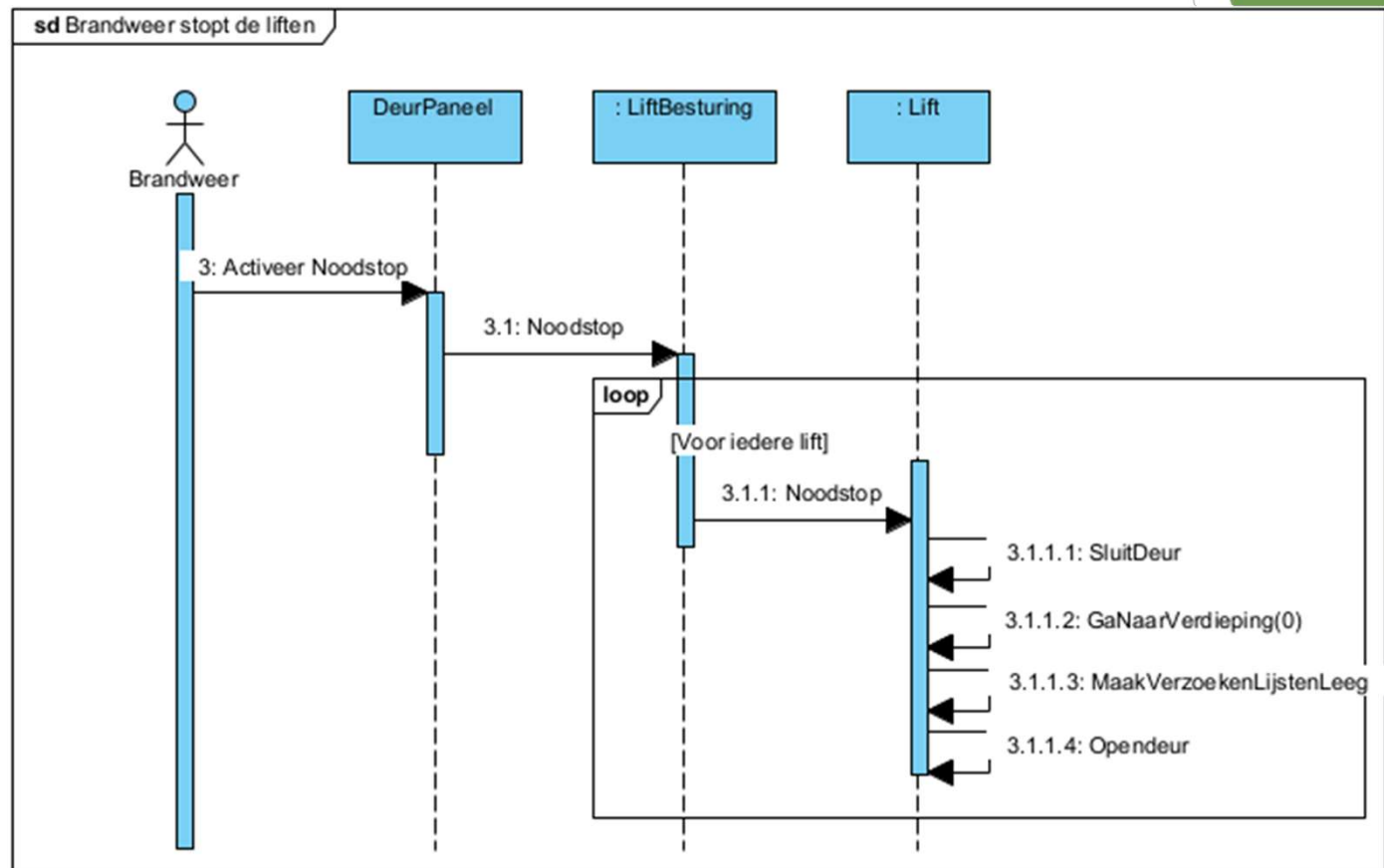
Sequentie

- Gebruiker neemt de lift
- Activeer lift
 - Console oplossing
- OO: delegeer operaties



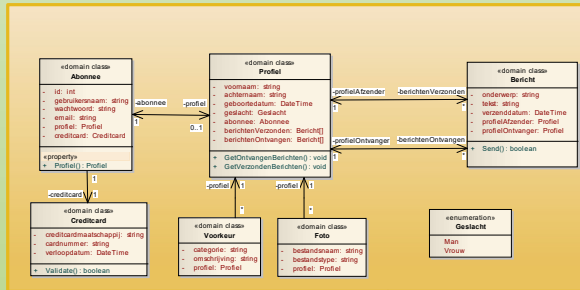
Sequentie diagram

► Noodstop

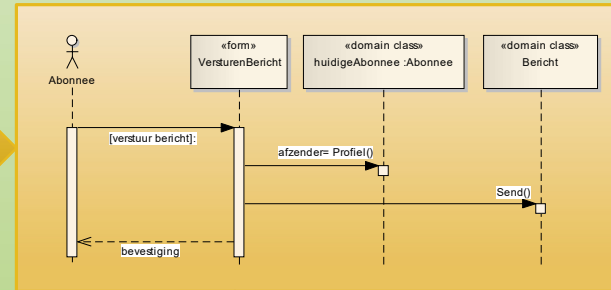


Ontwerpen

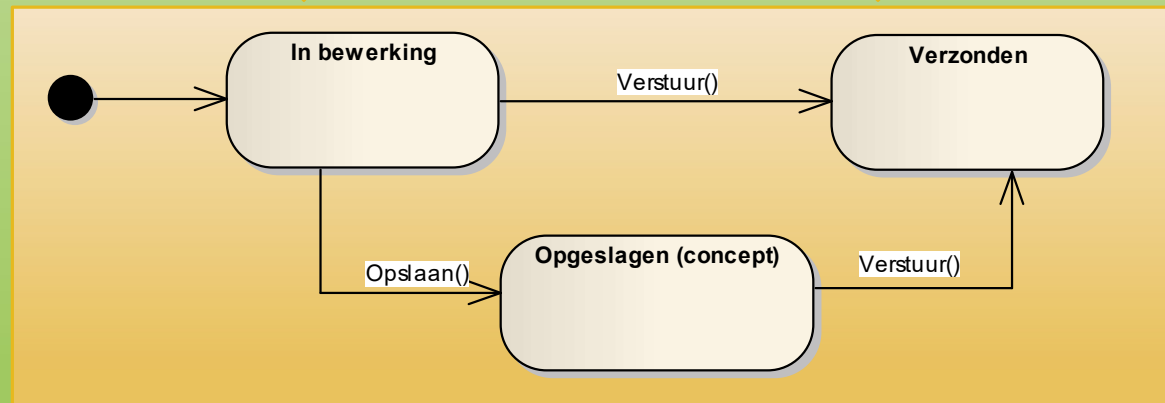
klassendiagram



sequentiediagram

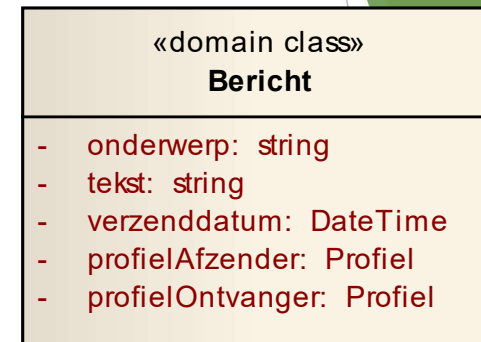


toestandsdiagram



Toestand van een klasse

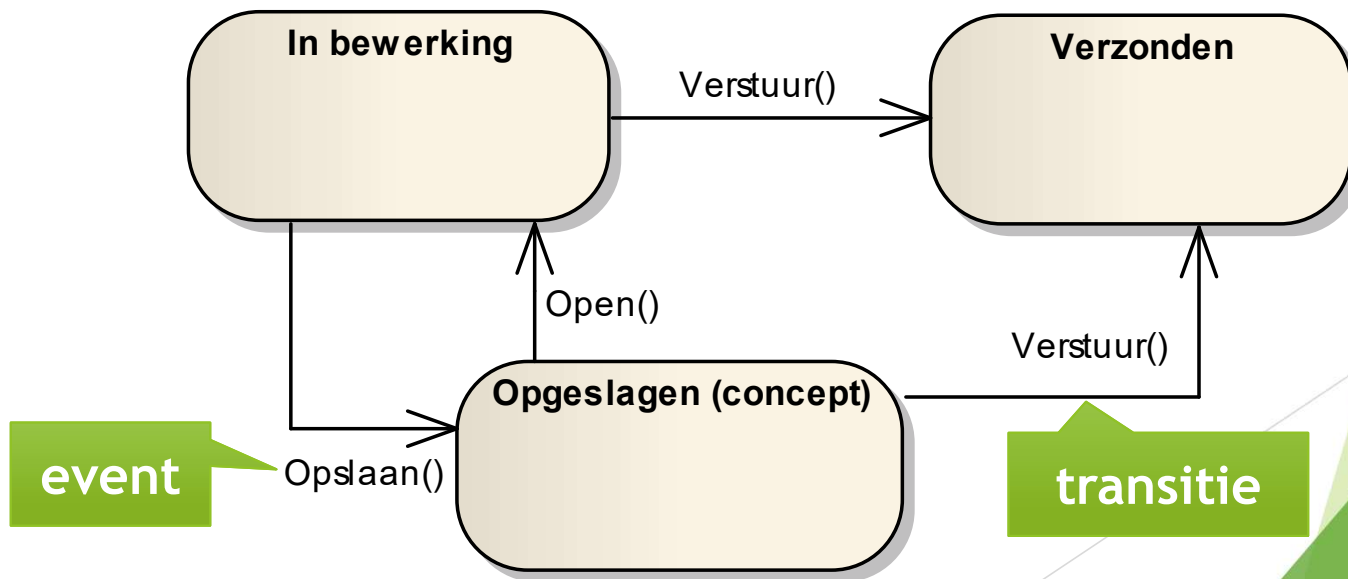
- ▶ Voorbeeld: mailbericht
- ▶ Telkens als één of meer attributen van waarde veranderen, verandert de toestand van de klasse
- ▶ Mogelijke toestanden voor een Bericht:



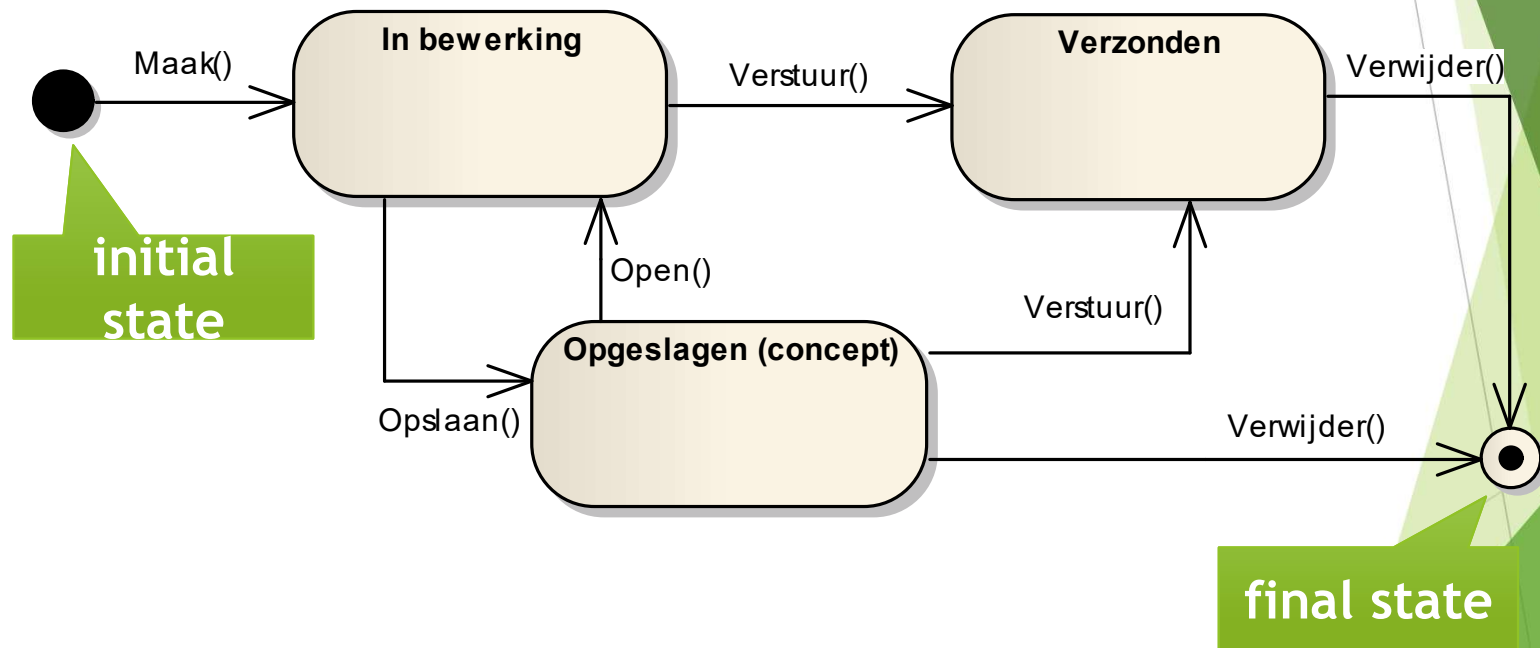
Veranderen van toestand

Een klasse gaat naar een nieuwe toestand wanneer er een gebeurtenis plaatsvindt:

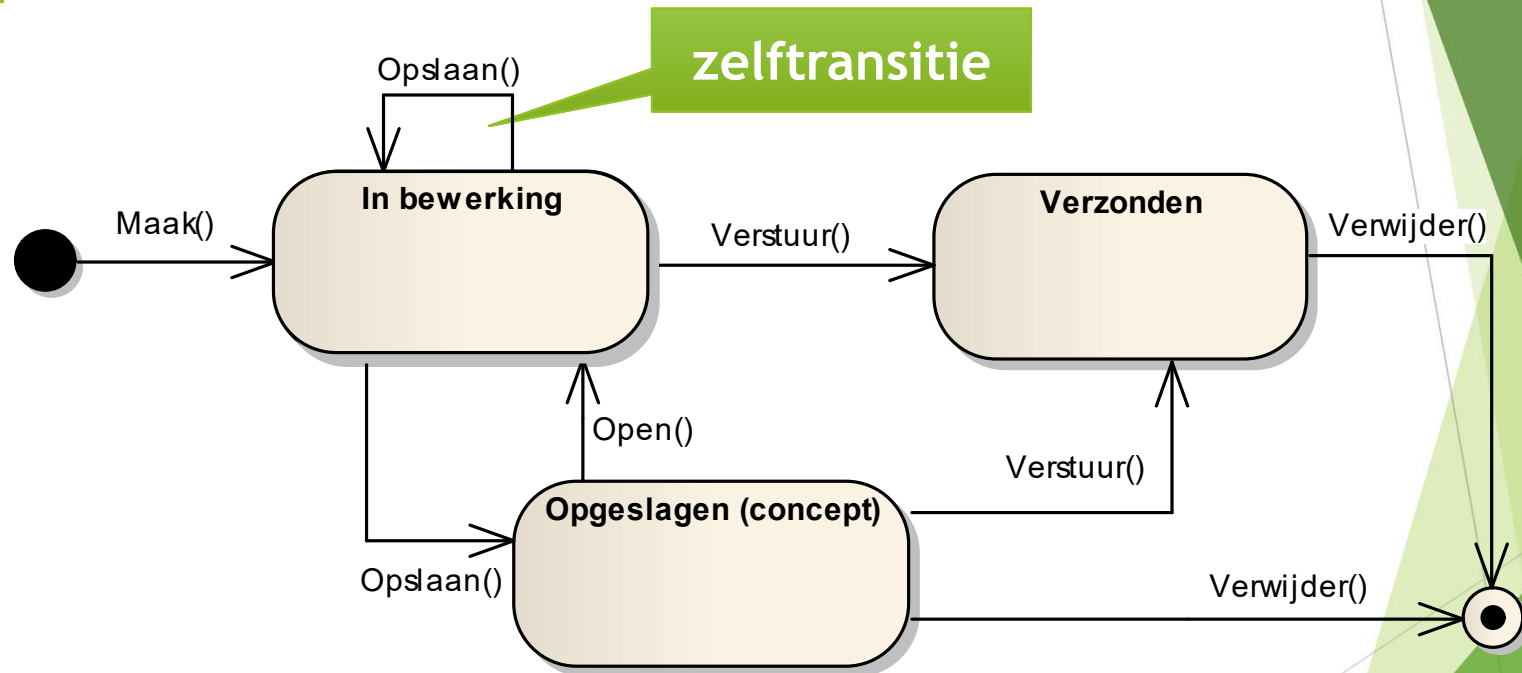
- ▶ Event (gestart door gebruiker)
- ▶ Methode (gestart door systeem)



Begin- en eindtoestand



Transitie die naar dezelfde toestand leidt



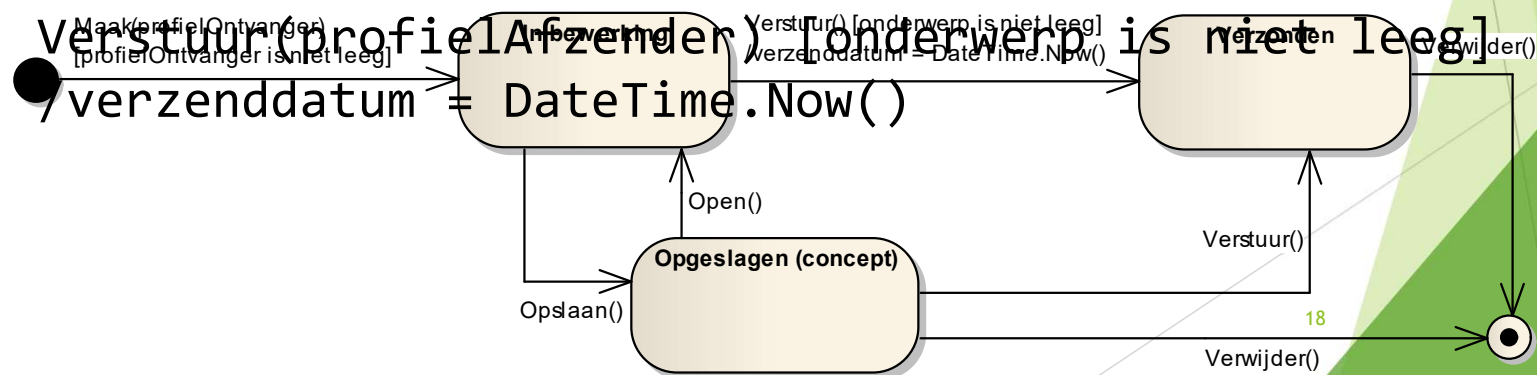
Parameters, guards en acties

Bij elke transitie kun je eventueel vermelden:

eventnaam(parameters) [guard] / actie

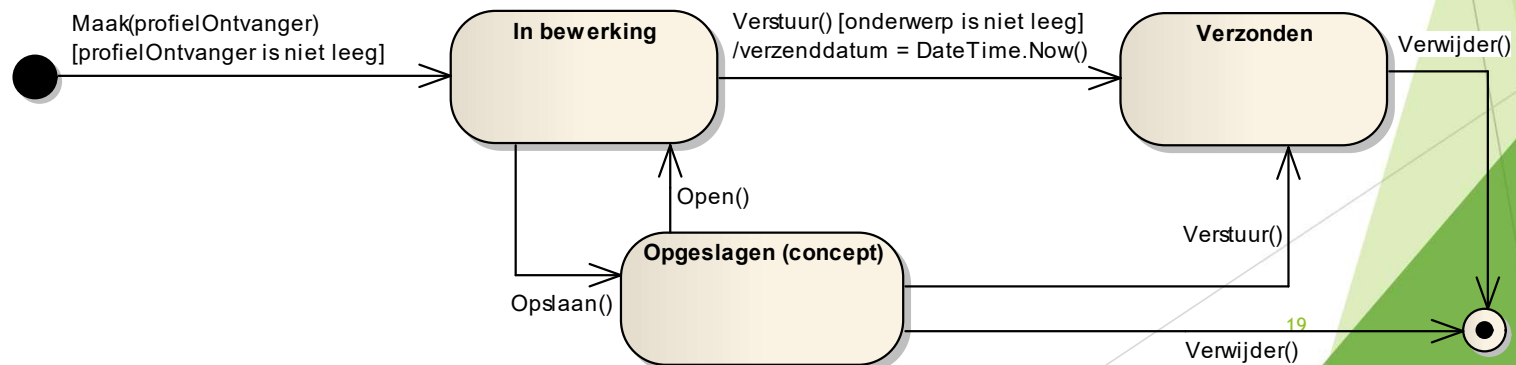
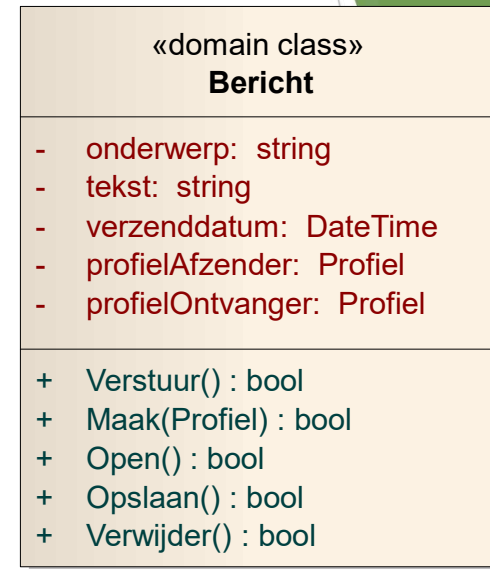
Voorwaarde waaraan moet zijn voldaan voordat de transitie mag plaatsvinden

- wijziging waarde attribuut
- aanroep van een methode van een andere klasse



Gevolgen voor klassendiagram

Je weet nu dat de klasse **Bericht** in ieder geval deze methoden moet hebben:



Gebruik van het toestandsdiagram

- ▶ Helpt bij het in kaart brengen van de vereiste methoden en attributen (klassendiagram, sequentiediagram)
- ▶ Beschrijft de levenscyclus van een kritiek object
- ▶ Vergroot betrouwbaarheid systeem: sommige operaties (events) zijn alleen toegestaan in een bepaalde toestand
- ▶ Maak alleen toestandsdiagrammen voor **kritieke** klassen die **dynamisch gedrag** vertonen
 - ▶ In administratieve systemen zijn dit meestal maar heel weinig klassen
 - ▶ In technische (real-time) systemen is een toestandsdiagram gebruikelijk en heel belangrijk

Stappenplan

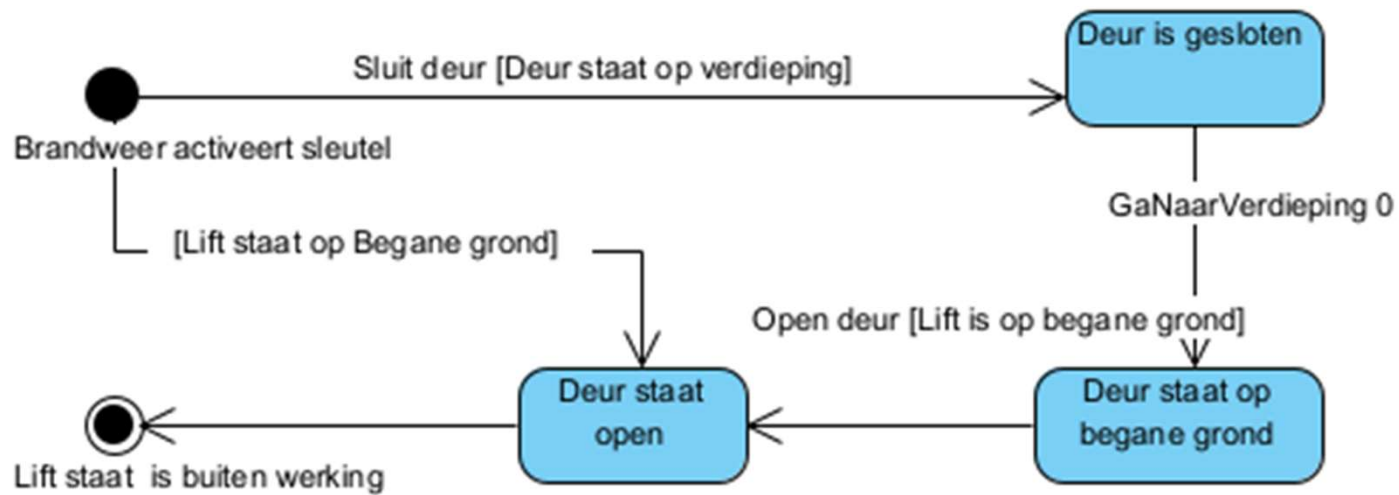
- ▶ Vind de toestanden waarin de klasse zich kan bevinden
- ▶ Vind de transitie en bijbehorende events
 - ▶ Klopt dit met je sequence- en klassendiagram?
- ▶ Voeg wanneer nodig begin- en eindtoestanden toe
- ▶ Voeg eventueel guards toe

Opdracht

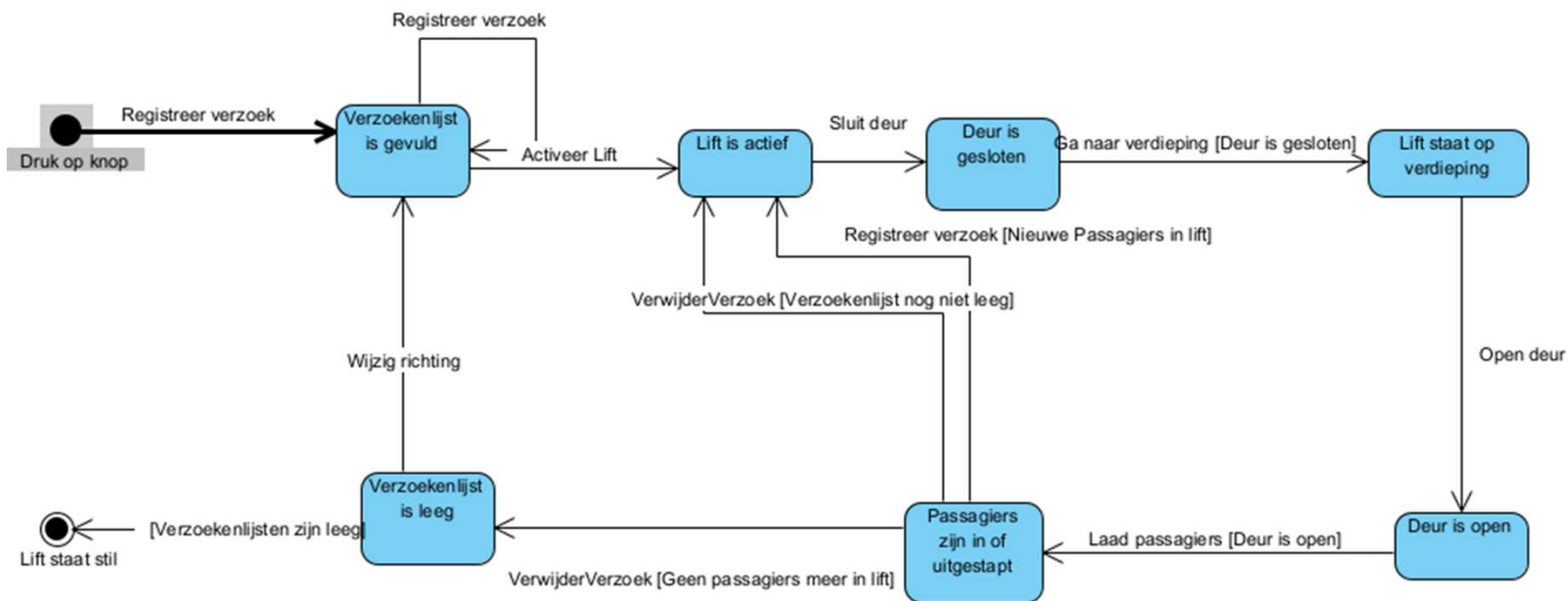
- ▶ Maak een toestandsdiagram (State machine diagram) voor de klasse LIFT:
 - ▶ 1 voor de brandweer
 - ▶ 1 voor normaal gebruik

Lift
+LiftID : int -AantalVerdiepingen : int = 4 +HuidigeVerdieping : int = 0 +Richting : string = "Omhoog" -DeurStatus : string = "Open" +VerzoekenLijstOmhoog : boolean[] +VerzoekenLijstOmlaag : boolean[]
+Lift(LiftID : int) +ActiveerLift() : void +MaakVerzoekenLijstenLeeg() : void +AantalVerzoeken() : int +GaNaarVerdieping(VVerdieping : int) : void +NoodStop() : void +RegistreerVerzoek(Verdieping : int, Richting : string) : void -LaadPassagiers() : void -MaakVerzoekenQueueLeeg() : void +RegistreerVerzoek(VanVerdieping : int, Richting : Omhoog, Omlaag) : void -OpenDeur() : void -SluitDeur() : void

Voorbeeld STD Brandweer



Voorbeeld uitwerking STD Normaal gebruik



Opdracht

- ▶ Werk het lift programma uit
 - ▶ Omdat we met een console applicatie niet event-driven werken gebruiken we commando's
- ▶ Commando's
 - ▶ V : verzoek (huidige verdieping, hoog, laag)
 - ▶ A: activeer liften. De liften gaan nu omhoog en omlaag tot de verzoekenlijsten leeg zijn. Als een lift stopt op een verdieping kan er weer een verzoek gedaan worden (naar verdieping, de lift berekent zelf omhoog of omlaag)
 - ▶ N: Noodstop
 - ▶ S: stop programma

Sorteren

- ▶ Sorteren van gegevens is een middel om snel te zoeken
- ▶ Veel verschillende sorteermethoden
 - ▶ Bubble sort
 - ▶ Selection sort
 - ▶ Shell's sort
 - ▶ Merge sort
 - ▶ Polyphase sort
 - ▶ Binaire boom
 - ▶ B-Tree
 - ▶ Insertion sort



Insertion sort

- ▶ Sorteermethode voor array's
 - ▶ Niet dynamisch, vaste grootte
- ▶ Sorteert een rij getallen in een array
- ▶ Begint links
- ▶ Zet de eerste twee op volgorde
- ▶ Tot alles gesorteerd is:
 - ▶ Selecteer volgende element
 - ▶ Schuif element op tot je de goede plek hebt gevonden
 - ▶ Voeg getal in

6 5 3 1 8 7 2 4

Insertion sort

- ▶ Maak de klasse public
 - ▶ Kan ook in andere programma's gebruikt worden
- ▶ Iedere methode retourneert de het object zelf!

```
2 references
public InsertionSortArray Print()
{
    for (int i = 0; i < rij.Length; i++)
    {
        Console.Write(rij[i]+ " ");
    }
    Console.WriteLine();
    return this;
}
```

```
public class InsertionSortArray
{
    int[] rij;
    2 references
    public int[] Rij { get {return rij;} }

    3 references
    public InsertionSortArray(int[] Rij)
    {
        rij = Rij;
    }
    3 references
    public InsertionSortArray insertionSort()
    {
        int positie = 1;

        while (positie < rij.Length)
        {
            int nieuwePositie = positie;
            while (nieuwePositie > 0 && rij[nieuwePositie - 1] > rij[nieuwePositie])
            {
                //swap
                int temp = rij[nieuwePositie - 1];
                rij[nieuwePositie - 1] = rij[nieuwePositie];
                rij[nieuwePositie] = temp;
                nieuwePositie--;
            }
            positie++;
        }
        return this;
    }
}
```

Insertion sort

- ▶ Maak ook een versie die de tussenresultaten afdrukt
- ▶ Kopieer en voeg de code toe




Afdrukken

```
public InsertionSortArray insertionSortMetPrint()
{
    int positie = 1;

    while (positie < rij.Length)
    {
        int nieuwePositie = positie;
        while (nieuwePositie > 0 && rij[nieuwePositie - 1] > rij[nieuwePositie])
        {
            //swap
            int temp = rij[nieuwePositie - 1];
            rij[nieuwePositie - 1] = rij[nieuwePositie];
            rij[nieuwePositie] = temp;
            nieuwePositie--;
        }
        positie++;
        Print();
    }
    return this;
}
```

Insertion sort

- ▶ Hoofdprogramma
 - ▶ InsertionSortArray wordt gecreëerd, maar niet aan objectvariabele toegekend!
 - ▶ Probeer beide sorteermethoden
- 
- ▶ Overbodige iteraties kun je met een booleanvariabele 'IsGesorteerd' voorkomen.

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("Aantal elementen:");
        int Aantal = int.Parse(Console.ReadLine());
        int[] Rij = new int[Aantal];
        for (int i = 0; i < Rij.Length; i++)
        {
            Console.WriteLine("Geef getal:");
            Rij[i] = int.Parse(Console.ReadLine());
        }
        new InsertionSortArray(Rij).insertionSort().Print();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Zoeken in een array

- ▶ Ongesorteerde gegevens: lineair zoeken
 - ▶ Begin op positie 0, dan 1, 2, 3, ... tot je de juiste waarde vindt
- ▶ Gesorteerde gegevens: binair zoeken
 - ▶ Begin halverwege
 - ▶ Gevonden? Klaar!
 - ▶ Niet gevonden: Kijk of de gezochte waarde kleiner of groter dan de waarde in het array
 - ▶ Herhaal dit in de linker- of rechterhelft

Binair zoeken

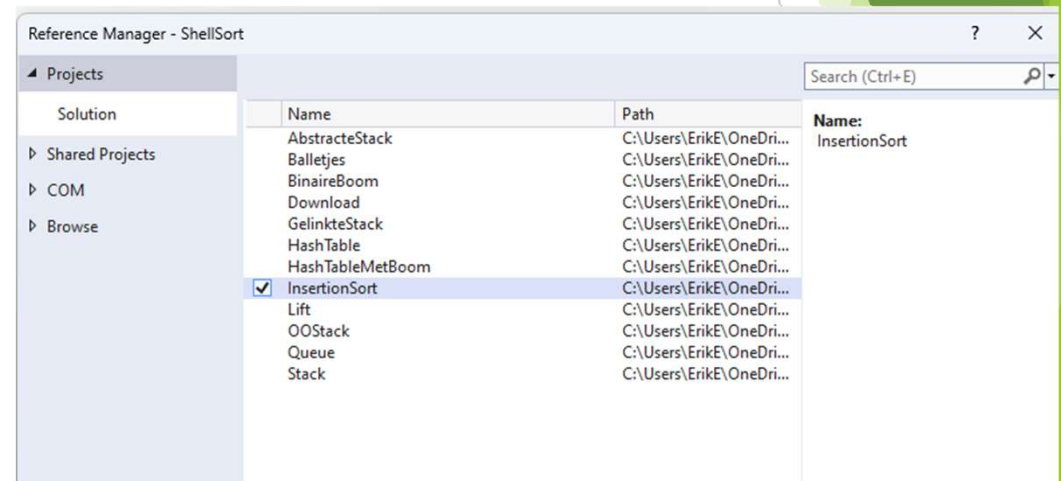
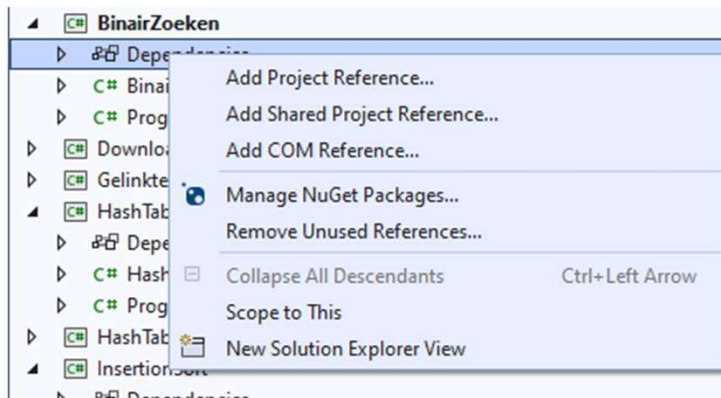
- Retourneert de positie waarop het getal is gevonden
- -1 betekent : niet gevonden

```
int[] rij;  
0 references  
public int[] Rij { get { return rij; } }  
  
1 reference  
public BinairZoekArray(int[] Rij)  
{  
    rij = Rij;  
}  
  
1 reference  
public int Zoek(int Getal)  
{  
    int min = 0;  
    int max = rij.Length - 1;  
    while (min <= max)  
    {  
        int mid = (min + max) / 2;  
        if (Getal == rij[mid])  
        {  
            return ++mid;  
        }  
        else if (Getal < rij[mid])  
        {  
            max = mid - 1;  
        }  
        else  
        {  
            min = mid + 1;  
        }  
    }  
    return -1;  
}
```


Binair zoeken

- ▶ Om te sorteren gaan we insertionsort gebruiken
- ▶ Project referentie toevoegen om insertionsort te gebruiken
- ▶ Voeg using... toe

```
1 using InsertionSort;  
2  
3 namespace BinairZoeken
```



Binair zoeken

```
try
{
    Console.WriteLine("Aantal elementen:");
    int Aantal = int.Parse(Console.ReadLine());
    int[] Rij = new int[Aantal];
    for (int i = 0; i < Rij.Length; i++)
    {
        Console.WriteLine("Geef getal:");
        Rij[i] = int.Parse(Console.ReadLine());
    }
    // sorteer het array
    BinairZoekArray ZoekArray = new BinairZoekArray(new InsertionSortArray(Rij).insertionSort().Rij);
    // Start zoeken
    int Getal = 0;
    Console.WriteLine("Zoek getal:");
    Getal = int.Parse(Console.ReadLine());
    while (Getal > int.MinValue)
    {
        int positie = ZoekArray.Zoek(Getal);
        if (positie > 0)
        {
            Console.WriteLine("Gevonden op positie " + positie);
        }
        else
        {
            Console.WriteLine("Niet gevonden");
        }
        Console.WriteLine("Zoek getal:");
        Getal = int.Parse(Console.ReadLine());
    }
}
catch
{
    Console.WriteLine("Einde programma");
}
```

Opdracht

- ▶ Lift programma verder uitwerken
- ▶ Casus Verkeerslichten op moodle
 - ▶ Klasse diagram
 - ▶ Sequence diagram
 - ▶ STD
- ▶ Tentamenvoorbereiding!
 - ▶ Geen programmeeropdracht

