### Afbeelding met tekst, schermopname, diagram, lijn Door AI gegenereerde inhoud is mogelijk onjuist.

### Association

Afbeelding met tekst, schermopname, Lettertype, lijn

Door AI gegenereerde inhoud is mogelijk onjuist.

* Een association geeft een algemene relatie tussen twee klassen aan.
* De ene klasse bevat een verwijzing naar een andere klasse.
* Er is sprake van samenwerking, maar geen eigendom of levenscyclus-koppeling.
* Beide klassen kunnen onafhankelijk bestaan.

**Voorbeeld in concept:**  
Een **Airline** heeft passagiers (**Passenger**). Ze zijn met elkaar verbonden, maar de passagier leeft los van de luchtvaartmaatschappij.

Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.

**Toegang:** Airline heeft toegang tot Passenger-objecten via een lijst, maar bezit ze niet strikt.

### Directed Association

### Afbeelding met tekst, Lettertype, schermopname, lijn Door AI gegenereerde inhoud is mogelijk onjuist.

* Een gerichte relatie: slechts één klasse wijst naar de andere.
* De pijl geeft de richting van afhankelijkheid aan.
* De klasse met de pijl "kent" of gebruikt de andere klasse.

**Voorbeeld in concept:**  
Een **Order** verwijst naar een **Customer**, maar een Customer weet niets van zijn orders.

**Afbeelding met schermopname, tekst

Door AI gegenereerde inhoud is mogelijk onjuist.**

**Toegang:** Order kent Customer, maar niet andersom. Directionele afhankelijkheid.

### Reflexive Association

Afbeelding met tekst, Lettertype, lijn, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.

* Een klasse heeft een relatie met een ander object van dezelfde klasse.
* Wordt vaak gebruikt bij hiërarchieën of zelfbeheerstructuren.
* Beide objecten zijn van hetzelfde type maar hebben verschillende rollen.

**Voorbeeld in concept:**  
Een **Employee** kan een manager hebben die ook een Employee is.

**Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.**

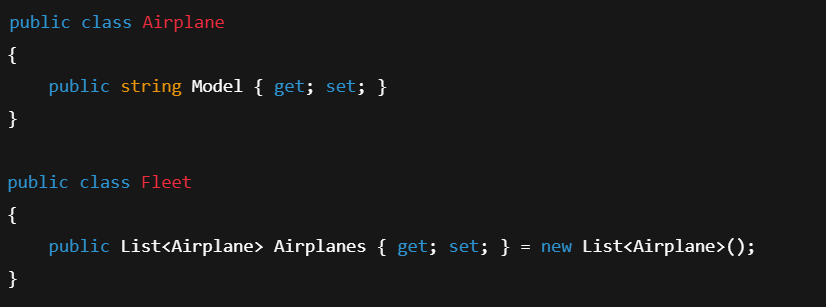
**Toegang:** Een Employee-object verwijst naar een ander Employee-object — zichzelf als hiërarchie.

### Multiplicity

Afbeelding met tekst, Lettertype, schermopname, diagram

Door AI gegenereerde inhoud is mogelijk onjuist.

* Geeft aan hoeveel objecten van de ene klasse gekoppeld zijn aan een ander object.
* Wordt aangeduid met notatie zoals 1, 0..1, 0..\*, 1..\*, enzovoort.
* Beschrijft kwantitatieve relaties in een associatie.

**Voorbeeld in concept:**  
Een **Fleet** heeft meerdere **Airplanes**, maar elk Airplane hoort bij exact één Fleet. 

**Toegang:** Fleet heeft toegang tot een lijst van Airplanes, met een 1:N (één-op-meerdere) multipliciteit.

### Aggregation

**Afbeelding met tekst, schermopname, Lettertype, symbool

Door AI gegenereerde inhoud is mogelijk onjuist.**

* "Heeft-een" relatie zonder eigendom.
* Kinderen (de componenten) kunnen blijven bestaan als de ouderklasse verdwijnt.
* Zwakke koppeling in vergelijking met composition.

**Voorbeeld in concept:**  
Een **Library** bevat meerdere **Books**. Als de bibliotheek sluit, blijven de boeken bestaan. Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.

**Toegang:** Library bevat Books, maar beheert hun levenscyclus niet. Geen eigendomsrelatie.

### Composition

### **Afbeelding met tekst, schermopname, Lettertype, lijn Door AI gegenereerde inhoud is mogelijk onjuist.**

* Sterke vorm van aggregatie met eigendom.
* Als de containerklasse verdwijnt, verdwijnen de componenten ook.
* De levenscyclus van de onderdelen is direct gekoppeld aan de hoofdklasse.

**Voorbeeld in concept:**  
Een **ShoulderBag** heeft een **SidePocket**. Als de tas vernietigd wordt, is het zakje ook weg. Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.

Afbeelding met tekst, schermopname, software

Door AI gegenereerde inhoud is mogelijk onjuist.

**Toegang:** ShoulderBag bezit de SidePocket volledig. Levenscyclus is gekoppeld.

### Inheritance / Generalization / Extends

**Afbeelding met tekst, Lettertype, lijn, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.**

* Een subklasse **erft** eigenschappen en methodes van een superklasse.
* Ondersteunt hergebruik en polymorfisme.
* Subklasse **is een** type van de superklasse.

**Voorbeeld in concept:**  
Een **Dog** is een **Animal**. Hij kan de maakgeluid methode overnemen en hem anders uitvoeren. Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.

**Toegang:** Subklasse Dog krijgt alles van Animal.

Een ander voorbeeld:

Afbeelding met tekst, schermopname, diagram, Parallel

Door AI gegenereerde inhoud is mogelijk onjuist.

De game applicatie, waar de Ridder, Tovenaar en Boogschutter alle **fields en methodes** overnemen. Tovenaar (als voorbeeld) bevat dus alle fields en methodes van Personage **+** Element. Deze methodes hoeven in de subclasses niet noodzakelijk aangepast te worden. Ridder **override de TakeDamage methode wel**, omdat die door zijn armor op een andere manier damage pakt. De Tovenaar en Boogschutter **implementeren die methode niet**, gezien die de standaard TakeDamage van Personage kunnen gebruiken. Het overriden van methodes uit de hoofdklasse is optioneel!

### Realization

### **Afbeelding met tekst, schermopname, Lettertype, lijn Door AI gegenereerde inhoud is mogelijk onjuist.**

* Een klasse implementeert een interface, oftewel: een contract van gedrag.
* De interface beschrijft alleen wat moet gebeuren, de klasse beschrijft hoe.
* Interface biedt abstractie, implementatie voegt concrete logica toe.

**Voorbeeld in concept:**  
Een **Printer** implementeert de interface **IPrinterSetup**, die een configuratiemethode vereist. Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.

**Toegang:** Printer belooft dat hij alles uitvoert wat IPrinterSetup definieert.

# Stappenplan

1. **Implementeer de klassen uit het UML-diagram**
   * Begin met het maken van alle klassen zoals ze in de UML staan, inclusief properties, methodes en constructors.
   * Let op toegangsniveaus (public/private) en inheritance (: baseClass). Als baseclass een constructor heeft ook : base() achter constructor in de subclass
   * **Tip:** Als je ergens op vastloopt, zet een comment zoals // TODO: hier moet nog een aggregratie komen en ga verder met de volgende klasse of methode.
2. **Maak een testproject aan (Unit Testing)**
   * Voeg een testproject toe aan je oplossing.
   * Maak alvast lege testmethodes aan voor de belangrijkste klassen en functies.
   * Je hoeft ze nog niet in te vullen — dit is puur voorbereiding.
3. **Zet een basis op in Program.cs**
   * Maak een Program.cs aan met de basisstructuur van Main.
   * Voeg alvast commentaar toe over wat je hier uiteindelijk wil runnen.
   * Gebruik de onderstaande boilerplate (vergeet hierbij niet de Namespace):
4. **Begin met de ‘Catalogus’-klasse van je systeem (als die er is)**
   * Zet deze class zo ver mogelijk op met wat je al weet
   * Maak de Initialize functie aan en zet deze comment er alvast in // TODO: vullen met data in
   * Vul de Initialize() functie met 1 eerste waarde, als dat al mogelijk is.
   * Maak een catalogus variabele in program.cs alvast aan:

var catalogus = new Catalogus();

catalogus.Initialize();

1. **Werk van onder naar boven: begin met simpele klassen**
   * Start met klassen die aan de ‘onderkant’ van de hiërarchie zitten (die geërfd worden).
   * Implementeer makkelijke methodes eerst, zoals ToString(), getters/setters, of simpele berekeningen.
   * Hierdoor kun je deze later hergebruiken bij de grotere klassen die deze objecten gebruiken.
2. **Loop je ergens te lang op vast? Laat het los en ga verder**
   * Als je merkt dat je te veel tijd kwijt bent aan één functie of klasse, zet er een comment bij en ga verder.
   * Laat je incorrecte code wel staan, misschien is het wel gewoon goed (alleen bijvoorbeeld een verkeerde hoofdletter)
   * Later kun je terugkomen met meer inzicht of als andere delen werken.
3. **Werk stap voor stap verder omhoog**
   * Als de basisklassen werken, bouw je de verdere hoofdklassen
   * Maak objecten in Program.cs aan wanneer dat kan. Als je bijvoorbeeld een kleine class hebt afgerond, maak die dan in program aan als die nodig is voor het scenario.

BOILERPLATE voor program.cs:

namespace NAAMCODEBASE

{

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine(Hello world);

}

}

}