# Object Oriented Programming

# Voorwoord

Deze opdrachtenreeks is bedoeld voor studenten van de opleiding Software Developer die studeren in periode 6 van het tweede leerjaar. Voorafgaand aan deze periode hebben de studenten het leermateriaal van Bit- Academy doorlopen. Verondersteld wordt dat de studenten beschikken over de basisvaardigheden van de C# programmeertaal en eenvoudige classes kunnen schrijven en instantiëren.

De reeks oefenopdrachten in deze syllabus zijn bedoeld voor het aanleren van Object Georiënteerd Programmeren. Om inzicht te krijgen in de wereld van Classes en Objecten zijn de opdrachten verduidelijkt doormiddel van Unified Modeling Language (UML). Hiermee hopen we te bereiken dat de materie als minder abstract wordt ervaren.

De opdrachten in deze syllabus zijn beschreven doormiddel van casussen die telkens in moeilijkheidsgraad worden opgevoerd door veranderingen door te voeren aan de casus. De opdrachten worden ondersteund door UML- diagrammen. In de casussen worden studenten ondersteund door een software engineer die telkens voorstellen doet voor een hernieuwd ontwerp wanneer de casus weer eens wordt aangepast. Op deze manier hopen we dat de student meer inzicht krijgt in het ontwerpen van een applicatie. Daar waar nodig wordt er verwezen naar studiemateriaal. Het is van belang dit studiemateriaal te doorlopen alvorens de opdracht uit te voeren.

Studenten werken met de didactische werkvorm Scrum At School. Tijdens elke sprint wordt er een opdracht opgeleverd. Bij elke opdracht zijn de leerdoelen vermeld. Bij de oplevering moeten de studenten aantonen dat de leerdoelen zijn behaald. Dit kunnen ze doen doormiddel van het bespreken van de uitwerkingen van de opdracht.

Voor het maken van de opdrachten krijgen de studenten ondersteuning van een vakdocent. Telkens wanneer een projectgroep begint aan een nieuwe opdracht zal een vakdocent de projectgroep ondersteunen door nieuwe programmeertechnieken te verduidelijken. Ook wordt er ondersteuning gegeven tijdens de uitvoering van de opdrachten. Ondersteuning zal worden gegeven aan de projecttafel in kleine groepen van maximaal 4 studenten.

De reeks opdrachten wordt afgesloten met een project en een toets.

# Inhoud

# Opdrachten Object Georiënteerd Programmeren in C#

## Casus.

Een pottenbakker wil keramische bloembakken voor in de buitentuin produceren en verkopen. Voor de startup van zijn bedrijf start hij met de verkoop van kubusvormige en de balkvormige bloembakken. Hieronder zijn afbeelding weergegeven van de producten die de pottenbakker wil gaan afzetten in de markt.

De pottenbakker heeft voor de startup een beperkt budget en kiest ervoor de opdrachten van klanten op te slaan op de lokale harde schijf van zijn computer.

Aan ons wordt gevraagd het programma te schrijven. We gaan daarbij gebruik maken van Object Oriented Programming (OOP).

## Opdracht 1.

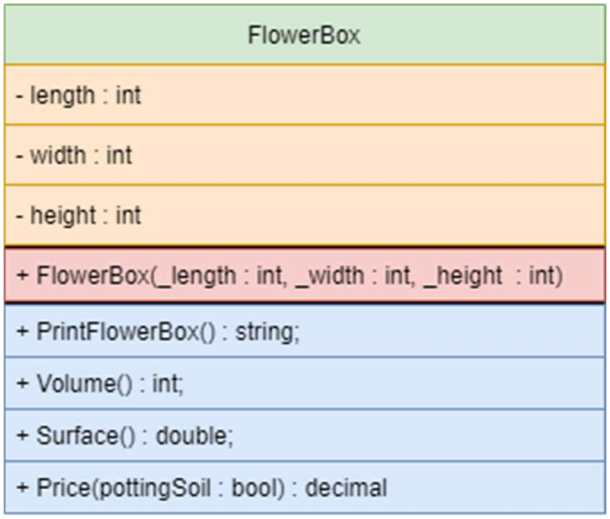
De pottenbakker wil voor iedere opdracht de klantgegevens en de afmetingen van de bloemenbak opslaan. Naderhand wil hij deze gegevens kunnen opvragen en afdrukken op het beeldscherm. We gaan deze opdracht in delen programmeren. We beginnen met het programmeren van de classes voor de bloemenbak en de klant. De gegevens die moeten worden opgeslagen op de harde schijf doen we in een latere fase.
Wanneer we de bloemenbak beschouwen als een object kunnen we vaststellen dat een balkvormige bloemenbak beschikt over de attributen lengte, breedte en hoogte. De pottenbakker heeft behoefte aan een aantal functionaliteiten die betrekking hebben op de attributen van de bloemenbak, te weten:

1. De pottenbakker wil voor een opdracht de afmetingen van een bloemenbak kunnen opnemen.
2. De pottenbakker wil op een later moment de afmetingen van een te produceren bloemenbak kunnen afdrukken op het beeldscherm.
3. Omdat de pottenbakker ook potgrond verkoopt wil hij het volume van de bloemenbak kunnen opvragen zodat hij weet hoeveel potgrond er in de bloemenbak gaat. Potgrond is een extra optie, een klant is niet verplicht potgrond te kopen.
4. Voor de klant wordt de oppervlakte van de bodem van de bloemenbak berekend zodat de klant weet hoeveel ruimte de bloemenbak inneemt in de tuin.
5. Tot slot wil de pottenbakker de verkoopprijs bepalen van de bloemenbak. De maatvoering van een bloemenbak is in cm. De prijs wordt bepaald door de inhoud van de bloemenbak te vermenigvuldigen met een factor 0.002. Voorbeeld prijs = 150.000$cm^3$ x 0.002. De prijs wordt twee cijfers achter de komma nauwkeurig berekend. Wanneer een klant potgrond bij de bloemenbak koopt moet deze worden doorberekend. De prijs van potgrond is de inhoud van de bloemenbak maal een factor 0.0002.

Een Software engineer heeft een klassendiagram ontworpen. Het klassendiagram is hieronder weergegeven.

*Uitleg klassendiagram*:

- *De naam van de class*: Een klassendiagram bestaat uit vier gedeelten. In het bovenste gedeelte van het klassendiagram wordt de naam van de class weergegeven.
- *Attributen van de class:* Onder de rechthoek waarin de naam van de class wordt weergegeven worden de attributen getoond. De attributen van een class zijn vrijwel altijd private. Dit geven we aan door het min teken voor de naam van het attribuut. Na de dubbele punt wordt het datatype van het attribuut weergegeven. We kunnen ook gebruik maken van properties, deze zijn public. Hier komen we later op terug.
- *Constructor:* In de rechthoek onder de attributen wordt de constructor getoond. In het klassendiagram wordt een constructor getoond met drie argumenten voor de afmetingen van de bloemenbak. Een constructor is altijd public, dit geven we aan met het plus teken "+". Een constructor heeft altijd de naam van de class en heeft geen return type. Een class kan beschikken over meerdere constructoren. Deze verschillen van elkaar door de argumenten van de constructor. Na de dubbele punt volgt het datatype van het argument van de constructor.
- *Methoden:* Na de constructor worden de functionaliteiten van de class getoond, ook wel methoden genoemd. Methoden zijn doorgaans public (+). Het is ook mogelijk methoden private te maken (-). Deze private methoden zijn dan slechts toegankelijk binnen de class zelf. Ook methoden kunnen beschikken over argumenten. Na de dubbele punt wordt weergegeven welk datatype de methode retourneert. Wanneer een methode niets retourneert is het return type void.




Schrijf een programma voor de pottenbakker. Heb je hulp nodig dan kun je een docent vragen of de volgende eenvoudige website bezoeken: https://www.w3schools.com/cs/cs_oop.php .

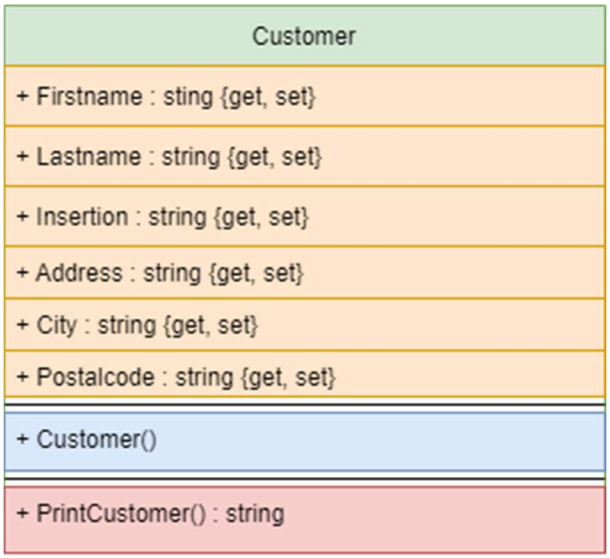
We beginnen met het realiseren van de class FlowerBox zoals getoond in het klassendiagram. Nadat de class is geschreven gaan we de class testen in een console applicatie. Daarvoor maken we een instantie van de class FlowerBox en roepen de verschillende methoden aan. Maak voor het testen gebruik van de tabel uit bijlage 1, opdracht 1.

*Beantwoord deze vraag:*
De class FlowerBox beschikt over private attributen. Wat wil dit zeggen voor het object van de class FlowerBox. Maak gebruik van de intellisense van Visual Studio om dit te onderzoeken.

Nu we de class FlowerBox hebben geschreven en getest is het tijd om de class “Customer” te realiseren. In het object van de class Customer slaan we de NAW- gegevens van de klant op. Bij de class FlowerBox hebben we gebruik gemaakt van private attributen. Bij de class Customer gaan we gebruik maken van Properties. Daarvoor bestuderen we eerst wat Properties nu eigenlijk zijn. We bezoeken daarvoor de volgende website: https://www.w3schools.com/cs/cs_properties.php .

*Beantwoord deze vraag*:
Wat is het verschil tussen Full Properties en Automatic Properties.

De software engineer heeft ons al een handje geholpen door het klassendiagram voor de klant te ontwerpen. Zie het klassendiagram hieronder.




Zoals je kunt zien beschikt de class Customer over vijf properties, twee constructoren waarvan één default (google hier eens naar) en een methode PrintCustomer welke de NAW-gegevens van de customer op het beeldscherm afdrukt. We kunnen dus de class Customer op twee manieren instantiëren. Een keer met de constructor met zes argumenten en een keer met de default constructor. De Property is een ontwikkeling in de C# taal en komt niet in alle programmeertalen voor. Daarom zijn er in de UML geen afspraken gemaakt hoe deze worden weergegeven in een klassendiagram. Onderling spreken we af dat properties worden weergegeven na het datatype door {get, set}. Het is ook mogelijk alleen een get of een set weer te geven. Zie het klassendiagram.

**Opdracht**
We beginnen met het realiseren van de class Customer zoals getoond in het klassendiagram. Nadat de class is geschreven gaan we de class testen in een console applicatie.

Realiseer de applicatie zodat de pottenbakker een opdracht van een klant kan opnemen. Maak gebruik van de classes Customer en FlowerBox. Opslaan op de harde schijf doen we in een latere

fase. Documenteer je code doormiddel van “Summery”, bekijk hiervoor eerst de volgende video: https://www.youtube.com/watch?v=z448dPJNXNs .

Maak gebruik van het volgende menu:
Menu:

1. Opdracht opnemen
2. Opdracht weergegeven
3. Prijs weergeven (met of zonder potgrond)
4. Inhoud weergeven
5. Oppervlakte weergeven
6. Stoppen

Maak uw keuze: _

## Testen van de applicatie

Test de class Order. Maak hiervoor gebruik van de tabel uit bijlage 1 en 2, opdracht 1. Schrijf voor het testen van de classes een methode waarin alle testitems uit de tabel worden getest. Neem de methode op in de applicatie zodat je kunt aantonen dat de classes zijn getest.

## Leerdoelen opdracht 1:

1. Eenvoudige class schrijven met private attributen (velden).
2. Kunnen verklaren wat een private attribuut betekent voor de toegankelijkheid van de instantie van de class.
3. Kunnen verklaren waarom we geen public attributen gebruiken (maar wel public properties).
4. Eenvoudige Constructor kunnen toevoegen aan de class welke de attributen van de class initieert.
5. Methoden toevoegen aan de class die een bewerking uitvoeren op, of met de attributen
6. Het verschil tussen private en public kunnen verduidelijken en kunnen aangeven wanneer en waar deze modifiers worden gebruikt.
7. Een eenvoudige class kunnen schrijven die gebruik maakt van Full Properties
8. Een eenvoudige class kunnen schrijven die gebruik maakt van Automatic Properties
9. Het verschil kunnen uitleggen tussen Full Properties en Automatic Properties en wanneer en in welke situatie deze worden gebruikt.

## Opdracht 2.

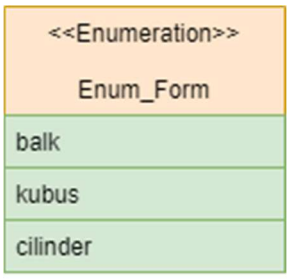
Het gaat goed met de pottenbakkerij. De opdrachten stromen binnen. Veel klanten vragen echter ook naar cilindervormige bloembakken. De pottenbakker springt hier direct op in.

We herschrijven het programma zodanig zodat de klant een keuze kan maken uit de volgende bloembakken:

- Kubusvormige bloemenbak
- Balkvormige bloemenbak
- Cilindervormige bloemenbak

De Software engineer bestudeert de nieuwe situatie en zegt genoodzaakt te zijn het klassendiagram aan te moeten passen omdat er nu sprake is van polymorfisme (google dit eens). Hetgeen wil zeggen dat objecten van de class FlowerBox meerdere vormen kunnen aannemen, namelijk bloembakken die cilindervormig, balkvormig of kubusvormig kunnen zijn. De software engineer wil dit bewerkstelligen door gebruik te maken van:

1. Drie constructoren, voor elke vorm van bloemenbak een constructor.
2. Door gebruik te maken van een “enum” datatype waarmee de vorm van de bloemenbak wordt vastgelegd. Bestudeer op w3schools het datatype enum: https://www.w3schools.com/cs/cs_enums.php , alvorens je verder gaat. In de afbeelding hieronder is een UML diagram weergegeven van een enum.
3. Door gebruik te maken van meerdere variabelen die beter passen bij de vorm van de bloemenbak.
4. Door variabelen die niet worden gebruikt null te maken. We maken hiervoor gebruik van “nullable value types”. Al vorens we verder gaan doen we eerst onderzoek naar “nullable value types”, zie hiervoor de volgende url’s: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types en https://josipmisko.com/posts/c-sharp-nullable-value-type .




De software engineer berust zich op onderzoek dat hij heeft gedaan naar de vormen van de bloembakken. Hieronder een overzicht van het onderzoek.

*Balkvormige bloemenbak:*

1. De zijden zijn niet gelijk
2. Dimensies: lengte, breedte en hoogte

3. Inhoud = lengte x breedte x hoogte
4. Oppervlakte grondvlak = lengte x breedte
5. Prijs = inhoud balk x 0.002 + eventueel potgrond = inhoud x 0.0002

*Kubusvormige bloemenbak*:

1. Alle 3 zijden van de kubus zijn gelijk.
2. Dimensies: 3 gelijke zijden
3. Inhoud = zijden$^3$ of ook wel, inhoud = zijde x zijde x zijde
4. Oppervlak grondvlak = zijde$^2$ of ook wel, oppervlak grondvlak = zijde x zijde
5. Prijs = inhoud kubus x 0.002 + eventueel potgrond = inhoud x 0.0002

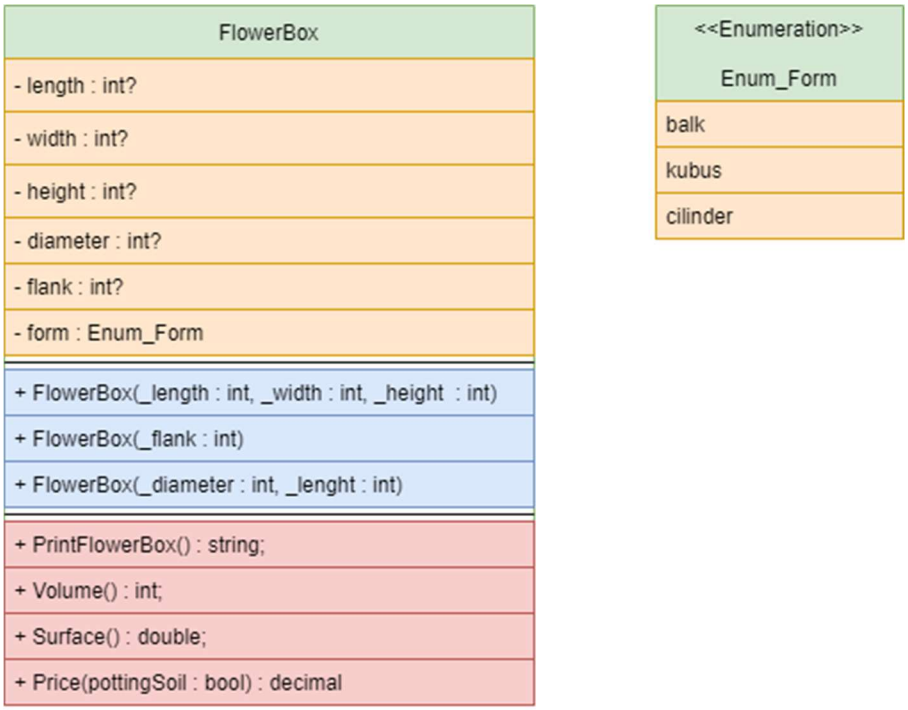
*Cilindervormige bloembakken:*

1. Dimensies: diameter en hoogte
2. Straal = diameter / 2
3. Hoogte is de hoogte van de cilinder
4. Inhoud = $\pi$ x Straal$^2$ x h
5. Oppervlakte = $\pi$ x Straal$^2$
6. Prijs = inhoud cilinder x 0.002 + eventueel potgrond = inhoud x 0.0002

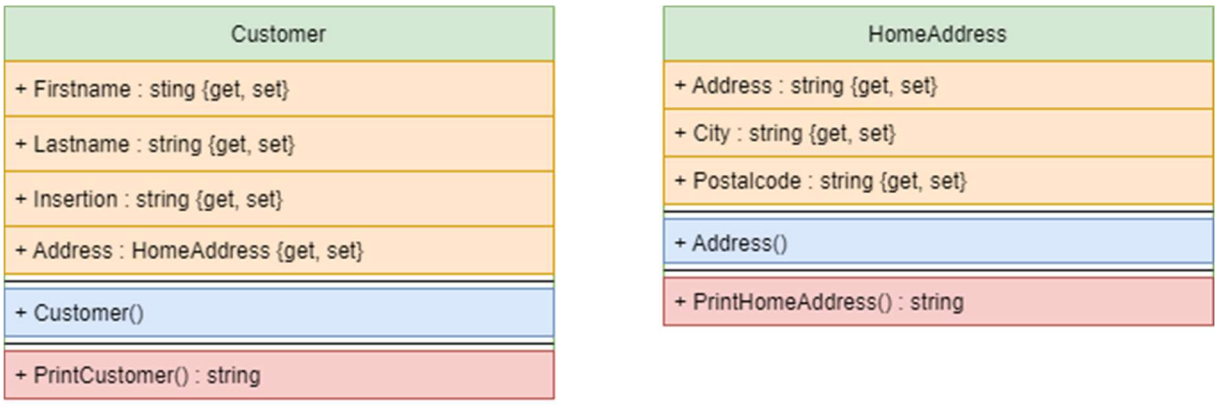
De software engineer trekt de conclusie dat het beter is voor iedere vorm van bloemenbak een aparte constructor te gebruiken. Wanneer men bijvoorbeeld een balkvormige bloemenbak construeert worden de variabelen lengte, breedte en hoogte gebruikt. Voor een cilindervormige bloemenbak worden de variabelen diameter en hoogte gebruikt.
Attributen van de class FlowerBox die door de constructor geen waarden krijgen worden automatisch door de compiler null gemaakt. Door gebruik te maken van een enum datatype (in ons geval “Enum_Form”) kan men van ieder willekeurig object van de class FlowerBox achterhalen uit welk type bloemenbak het object bestaat.

De pottenbakker kan in zijn oven potten bakken van maximaal 100cm x 100cm x 100cm. Zorg ervoor dat er een Exception wordt opgeworpen en afgehandeld wanneer deze maten worden overschreden. Door loop het volgende leermateriaal voor Exception Handeling: https://www.tutorialsteacher.com/csharp/csharp-exception-handling en voor het opwerpen van een Exception: https://www.tutorialsteacher.com/csharp/throw-csharp .

Het klassendiagram ziet er nu als volgt uit.




Nog steeds is de software engineer niet tevreden. Hij heeft nog eens de class Customer bestudeerd en kwam tot de conclusie dat de class moet worden afgesplitst in twee aparte classes omdat de adresgegevens van de klant op zichzelf ook weer een object is. Je kunt namelijk zeggen, “Een klant heeft adresgegevens”, maar adresgegevens bestaan uit zich zelf weer uit meerdere attributen, te weten: adres, woonplaats en postcode. De Software engineer heeft voor de class Customer een nieuw ontwerp gemaakt. Zie het klassendiagram hieronder.




De klas Customer heeft een associatie met de class HomeAddress. De class HomeAddress verleent een dienst aan de class Customer door de adresgegevens te verzorgen van de klant. In het klassendiagram is dit te zien doordat de class Customer beschikt over een property “Address” welke van het type HomeAddress is. Hieruit kun je concluderen dat er sprake is van een associatie is tussen de classes Customer en HomeAddress.

Om associaties vast te stellen wordt vaak in UML gebruik gemaakt van objectendiagrammen. In dergelijke diagrammen worden objecten bestudeerd en onderlinge associaties vastgesteld die later weer worden verwerkt in het klassendiagram. In het voorbeeld hieronder is een objectendiagram weergegeven. Het object “Jan Jansen” is van het type Customer en heeft een relatie met Address die van het type HomeAddress is. Op de website https://www.tutorialsteacher.com/csharp/association-and-composition wordt uitleg gegeven over associaties. Bestudeer eerst deze leerstof alvorens je

verder gaat. Onder het kopje “Composition” wordt bij toeval een uitleg gegeven die overeenkomt met de situatie zoals hier beschreven.


**Opdracht**
Het ontwerp is nu compleet, we kunnen nu de applicatie realiseren. Maak gebruik van het menu zoals hieronder is afgebeeld.

Menu:
1. Opdracht opnemen
2. Opdracht weergegeven
3. Prijs weergeven (met of zonder potgrond)
4. Inhoud weergeven
5. Oppervlakte weergeven
6. Afleveradres afdrukken
7. Stoppen

Maak uw keuze: _

*Geef antwoord op de volgende vraag*:
Waarom zijn de attributen (velden) van de class FlowerBox private en mogen deze **nooit** veranderen naar public.

## Testen van de applicatie

Test de classes FlowerBox, Customer en HomeAddress. Maak hiervoor gebruik van de tabel uit bijlage 3 en 4. Schrijf voor het testen van de classes een methode waarin alle testitems uit de tabel worden getest. Neem de methode op in de applicatie zodat je kunt aantonen dat de class is getest.

## Leerdoelen opdracht 2:

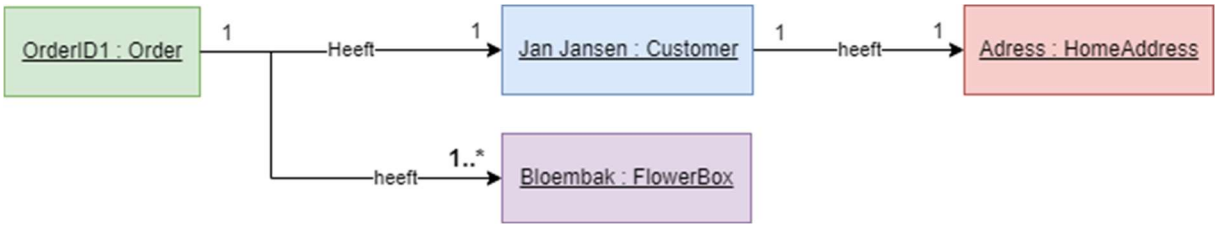
1. Een enum type kunnen schrijven en gebruiken
2. Class kunnen schrijven die beschikt over constructor overloading (meerdere constructors)
3. Kunnen uitleggen waarom constructor overloading van de class FlowerBox zorgt voor polymorfe objecten.
4. Gebruik kunnen maken van “nullable value types”.
5. Kunnen uitleggen in welke situatie nullable value types worden gebruikt en waarom.
6. Gebruik kunnen maken van Exception Handeling
7. Een Exception kunnen opwerpen, “throw new …….”
8. Kunnen uitleggen wanneer klassen associaties hebben.

# Opdracht 3.

**Vervolg van de casus**

Het loopt storm bij de pottenbakker, veel klanten bestellen in één order meerdere bloembakken voor hun tuin. Hierdoor is er echter een probleem ontstaan, de applicatie van de potenbakker kan slechts per klant één opdracht aanvaarden terwijl sommige klanten per order meerdere bloembakken willen bestellen.

De software engineer heeft voor de nieuwe situatie een objectendiagram ontworpen waarbij het mogelijk is voor één order meerdere bloembakken te bestellen. Zie het objectendiagram hieronder. Het symbool 1..* in het objectendiagram betekent, één order kan één of meerdere bloembakken bevatten. In het objectendiagram zien we dat er een class Order is bijgekomen. De class Order beschikt over een attribuut van de class Customer, en een attribuut die bestaat uit een collectie objecten van de class FlowerBox.




*Objectendiagram*

Nu het objectendiagram tot stand is gekomen gaat de gedachten van de software engineer uit naar de attributen en functionaliteiten van de class Order. De class Order beschikt over de volgende attributen:

1. Een collectie van bloembakken die de pottenbakker moet vervaardigen voor de klant.
2. Een object van de class Customer (de klant) die de order heeft gegeven.
3. Het adres waar de bloembak(ken) moet worden afgeleverd. Deze functionaliteit hadden we kunnen uitvoeren door gebruik te maken van een methode. Wanneer een methode geen argumenten heeft kan men overwegen om hiervoor een property te gebruiken. Dit is hoe wij het hier gaan doen.
4. Een property die de totale prijs weergeeft voor het vervaardigen van de bloembakken om de klant snel te informeren wat de opdracht gaat kosten.

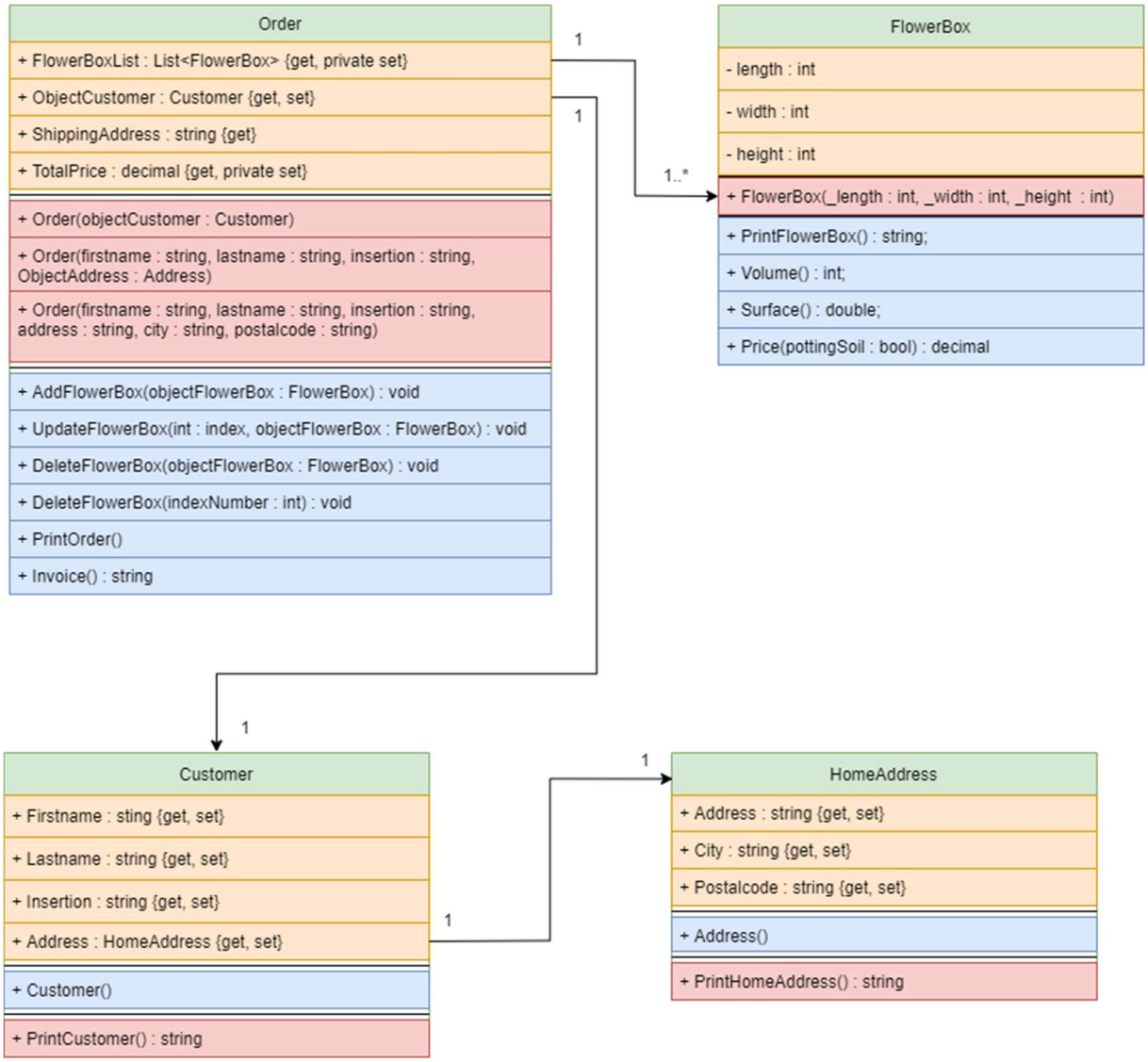
De class Order beschikt over de volgende functionaliteiten:

1. Een bloembak toevoegen aan de collectie van bloembakken.
2. De afmetingen van een bestelde bloembak aanpassen voor het geval de klant zich bedenkt.
3. Een bloembak verwijderen uit de collectie van bestelde bloembakken. De software engineer heeft ervoor gekozen om hier gebruik te maken van een “method overload”. Bestudeer de volgende url voor het gebruik hiervan: https://www.w3schools.com/cs/cs_method_overloading.php .
4. De gehele order (er zijn meerdere bloembakken mogelijk) afdrukken zodat de pottenbakker weet welke bloembakken hij moet vervaardigen voor de klant
5. Afdrukken van een eenvoudige rekening. Zie het voorbeeld hieronder.

**Klant:**
Karel de Groot
Gieterij 200
9153 VZ
Hengelo

| | | |
|---|---|---|
| Bloembak 1: balkvormig, | 100cm x 80cm x 50cm | €800,00- |
| Bloembak 2: Cilindervormig, | diameter 60cm x hoogte 60cm | €339, 30- |
| Totaal: | | €1139, 30- |

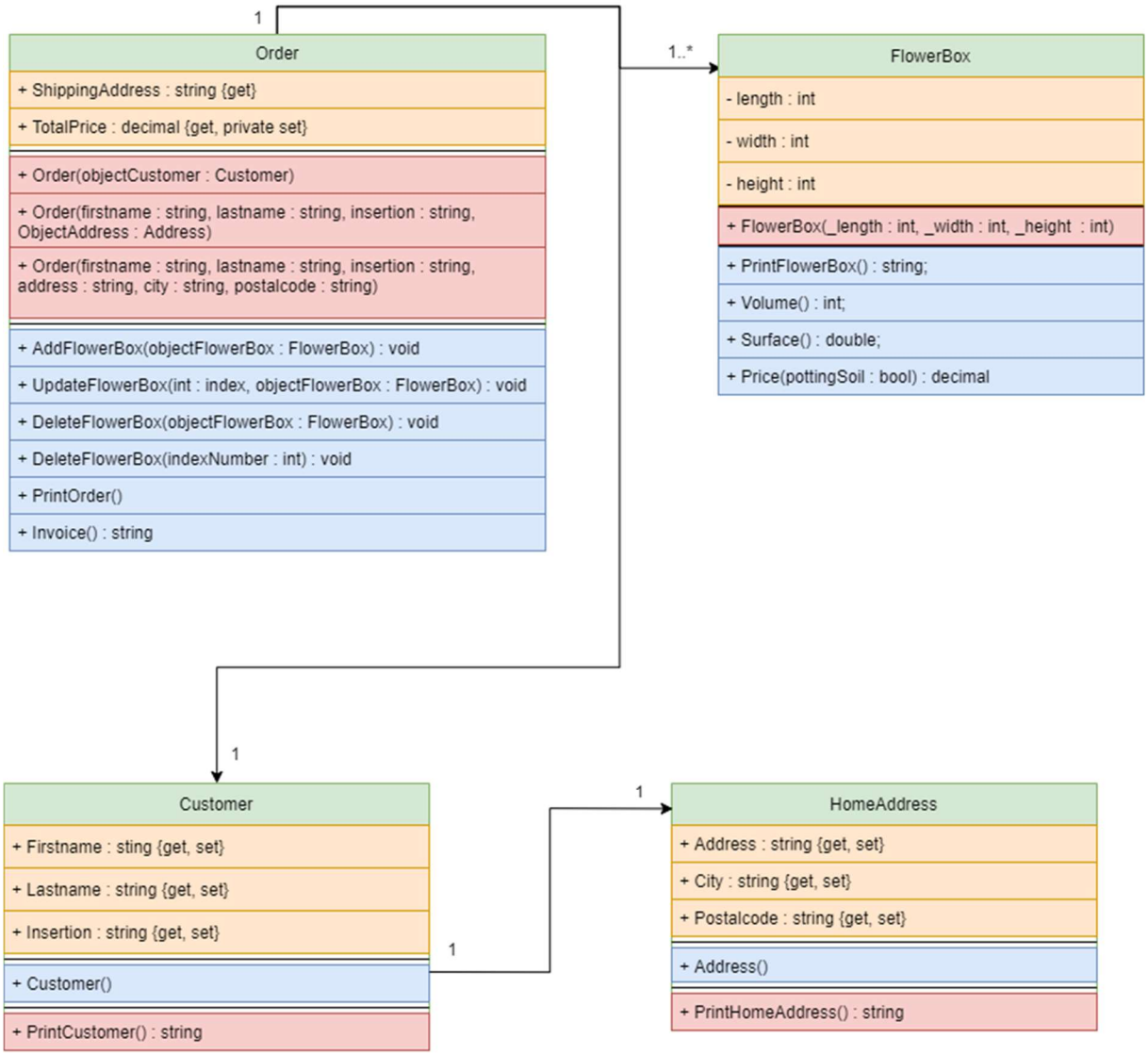
De software engineer heeft het objectendiagram verder uitgewerkt tot een klassendiagram. Zie het diagram hieronder.



*Volledig klassendiagram*

Het klassendiagram laat zien dat de class Order een relatie heeft met de class Customer en FlowerBox. Dit wordt weergegeven doormiddel van een lijn met een pijl. Wanneer de lijn met pijl de relatie weergeeft is het in de UML gebruikelijk het veld niet weer te geven. In ons klassendiagram is

dit wel gedaan, dit voor de duidelijkheid (zie b.v. het attribuut FlowerBoxList in de class Order). Hieronder is het klassendiagram nog eens afgebeeld, nu vereenvoudigd weergegeven. De lijn met pijl geeft de relatie aan. Hieruit maak je op dat de class Order over attributen beschikt van het type Customer en FlowerBox. En dat de class Customer beschikt over een attribuut Address. Het attribuut hoeft dus niet worden afgebeeld in het klassendiagram omdat dit kan worden afgeleid door de lijn met pijl. Het nadeel hiervan is dat de software developer de vrijheid krijgt om zelf in te vullen hoe hij het attribuut vorm gaat geven (private, public, property of benaderen door een methode). Wil je dit niet dan kun je beter het klassendiagram weergeven zoals in de eerste afbeelding. De software developer wordt dan gedwongen om het zo uit te voeren als de software engineer het heeft ontworpen. Vaak heeft dit een goede reden.




*Vereenvoudigd klassendiagram*

**Opdracht**

Het ontwerp is nu compleet. Herschrijf de applicatie van opdracht 2 zodanig dat de klant meerdere bloembakken kan bestellen per order, en neem de aanvullende functionaliteiten van de class Order op in het menu. Het menu zou er als volgt uit kunnen zien.

Menu:

1. Bloembak toevoegen

2. Bloembak updaten
3. Bloembak verwijderen
4. Opdracht weergegeven
5. Specifieke bloembak opvragen
6. Totaal prijs weergeven
7. Afleveradres afdrukken
8. Factuur afdrukken
9. De prijs van de duurste bloembak weergeven
10. De bloembak met het kleinste grondoppervlak weergegeven.
11. Stoppen

Maak uw keuze: _

*Optie 1:*
Voor optie 1 laat de applicatie een submenu zien waar men een keuze kan maken welke vorm van bloembak men wil toevoegen, balk, kubus of cilindervormig.

*Optie 2:*
Voor optie 2, bloembak updaten worden eerste alle bloembakken getoond die in de order voorkomen en kan men vervolgens weergeven welke bloembak moet worden geüpdatet. Een voorbeeld:
==== Bloembak updaten ====

1: vorm = balk, lengte 100cm, breedte = 80cm, hoogte = 50cm

2: vorm = kubus, zijde = 80cm

3: vorm = cilinder, diameter = 60cm hoogte = 40cm

Welke bloembak wilt u updaten: _

==========

*Optie 3:*
Voor optie 3 worden weer alle bloembakken getoond en wordt er gevraagd welke bloembak moet worden verwijderd.

*Optie 4:*
Optie 4 drukt de gehele bestelling af. Druk van iedere bloembak alle gegevens af.

*Optie 5:*
Met optie 5 is het mogelijk een specifieke bloembak uit de gehele verzameling op te vragen. Bijvoorbeeld bloembak nr. 3 uit de verzameling van 5 bloembakken. Druk alle gegevens van deze bloembak af. Maak hiervoor gebruik van een indexer:
https://www.tutorialsteacher.com/csharp/csharp-indexer .

*Optie 6..11:*
Verdere menu opties spreken voor zich.

Bestudeer de leerstof van de url's zoals hieronder weergegeven alvorens je de opdracht gaat uitvoeren:

- https://www.tutlane.com/tutorial/csharp/csharp-list
- https://www.tutorialsteacher.com/csharp/csharp-indexer

## Testen van de applicatie

Test de class Order. Maak hiervoor gebruik van de tabel uit bijlage 5. Schrijf voor het testen van de class Order een methode waarin alle testitems uit de tabel worden getest. Neem de methode op in de applicatie zodat je kunt aantonen dat de class is getest.

## Leerdoelen opdracht 3

1. Objectendiagram kunnen lezen.
2. Klassendiagram kunnen lezen.
3. Methoden van de class List<T> kunnen gebruiken.
4. Collectie objecten uit een List<T> kunnen afdrukken.
5. Een object uit een collectie List<T> kunnen updaten.
6. Objecten uit een collectie List<T> kunnen verwijderen.
7. Een List<T> kunnen doorzoeken.
8. Een indexer kunnen programmeren en gebruiken.
9. Gebruik kunnen maken van methoden die beschikken over een "method overload".

## Bijlage 1: Testen van class FlowerBox

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| FlowerBox | Maak een instantie van de class FlowerBox. Maak hiervoor gebruik van de volgende argumenten:<br>• Lengte = 100<br>• Breedte = 50<br>• Hoogte = 30 | Een object van de class FlowerBox. De lengte, breedte en hoogte van de bloemenbak kunnen niet meer worden aangepast. | |
| | Roep de methode PrintFlowerBox aan en controleer de uitvoer | Vorm bloembak: Balk vormig<br>Lengte: 100cm<br>Breedte: 50cm<br>Hoogte: 30cm<br>Volume: 150000cm3 | |
| | Roep de methode Volume aan en controleer of de uitvoer correct is | Bereken het volume en controleer dit met de uitvoer van de methode | |
| | Roep de methode Surface aan en controleer of de uitvoer correct is | Bereken de oppervlakte van de bodem van de bloemenbak en vergelijk dit met de uitvoer van de methode | |
| | Roep de methode Price aan om de prijs van de bloemenbak te berekenen. Controleer de berekening. Doe dit ook voor het geval de klant potgrond koopt voor de bloemenbak. | Bereken de prijs van de bloemenbak en controleer dit met de uitvoer van de methode. Voer de berekening ook door voor een bloemenbak met potgrond | |

## Bijlage 2: Testen van class Customer

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| Customer | Maak een instantie van de class Customer. Maak hiervoor gebruik van de constructor met 6 argumenten.<br>• voornaam = Jan<br>• achternaam = Hop<br>• tussenvoegsel = de<br>• adres = Bakkerstraat<br>• Woonplaats: Hengelo<br>• Postcode: 7553 VZ | Controleer met de intellisense van Visual Studio of de properties over de juiste NAW-gegevens beschikken | |
| | Maak een instantie van de class Customer met de default constructor. Geef de NAW properties dezelfde gegevens zoals hierboven weergegeven. Voorbeeld:<br>objCustomer.Voornaam = “Jan”; | Achterhaal met de intellisense van Visual Studio of de properties over de NAW- gegevens beschikken | |
| | Maak een instantie van de class Customer doormiddel van een Automatic Property Initializer (google dit eens). | Achterhaal met de intellisense van Visual Studio of de properties over de NAW- gegevens beschikken | |
| | Verander het adres van de klant door middel van de property. | Controleer met de intellisense van Visual Studio of de property daadwerkelijk is aangepast | |
| | Roep de methode PrintCustomer aan en controleer de uitvoer | De methode moet de NAW- gegevens van de klant tonen op het beeldscherm | |

## Bijlage 3: Testen van class FlowerBox hernieuwd ontwerp

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| FlowerBox | Maak een instantie van de class FlowerBox voor een balkvormige bloembak. Maak hiervoor gebruik van de volgende argumenten:<br>• Lengte = 100<br>• Breedte = 50<br>• Hoogte = 30 | Object is aangemaakt en de property “form” geeft de juiste vorm van bloembak weer | |
| | Maak een instantie van de class FlowerBox voor een kubusvormige bloembak. Maak hiervoor gebruik van de volgende argumenten:<br>• Zijde = 80 | Object is aangemaakt en de property “form” geeft de juiste vorm van bloembak weer | |
| | Maak een instantie van de class FlowerBox voor een cilindervormige bloembak. Maak hiervoor gebruik van de volgende argumenten:<br>• diameter = 30<br>• hoogte = 50 | Object is aangemaakt en de property “form” geeft de juiste vorm van bloembak weer | |
| | Maak voor iedere vorm van bloembak een instantie waarbij de maximale maatvoering van 100cm wordt overschreden | Object wordt niet gemaakt, de applicatie werpt een exception op | |
| | Test voor alle drie instanties de methoden. | De methoden geven de juiste waarden terug voor elke instantie | |

## Bijlage 4: Testen van class HomeAddress en Customer

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| HomeAddress | Maak een instantie van de class HomeAddress.<br>Doe dit met een object initializer zie de volgende url als voorbeeld: https://www.tutorialsteacher.com/csharp/csharp-object-initializer | Object is aangemaakt. De attributen Address, City en Postalcode van de class zijn benaderbaar, hun waarden hebben de functionaliteit lezen en schrijven | |
| | Roep de methode PrintHomeAddress aan. | De methode retourneert de adresgegevens | |

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| Customer | Maak een instantie van de class Customer.<br>Doe dit met een object initializer zie de volgende url als voorbeeld: https://www.tutorialsteacher.com/csharp/csharp-object-initializer | Object is aangemaakt. De attributen Firstname, Lastname, Insertion en Address van de class zijn benaderbaar, hun waarden hebben de functionaliteit lezen en schrijven | |
| | Roep de methode PrintCustomer aan. | De methode retourneert de NAW- gegevens van de klant | |
| | Overschrijf het adres van de klant. Maak hiervoor een new object van de class HomeAddress aan. | Het oude adres van de klant is overschreven. De methode PrintCustomer geeft het nieuwe adres weer. | |

## Bijlage 5: Testen van de class Order

| Testen van de classes | Test | Resultaat | Aanpassing (Indien niet correct) |
|---|---|---|---|
| Order | De class Order beschikt over drie constructoren. Test alle drie constructoren uitvoerig. | Drie objecten van de class Order. Alle drie objecten beschikken de volgende attributen:<br>• object van de class customer. De NAW-gegevens van de klant zijn opgeslagen in het object<br>• object van het type List<FlowerBox>. De Count property van het object = 0 | |
| | Roep de methode AddFlowerBox aan en voeg een object van het type FlowerBox toe aan de collectie | De Count property staat op 1. Het object is aantoonbaar toegevoegd aan de collectie van bloembakken | |
| | Roep de methode UpdateFlowerBox aan. Selecteer een flowerbox object en wijzig de dimensies van de bloemenbak | De dimensies van het object is aangepast. | |
| | Roep de methode DeleteFlowerBox aan en delete een bloembak uit de collectie van bloemenbakken. Test dit voor beide overloads van de methode | De bloemenbak is verwijderd uit de collectie van bloembakken.<br><br>Bloembakken kunnen met beide overloads van de methode worden verwijderd | |
| | Roep de methode PrintOrder aan. | De gehele order wordt afgedrukt. De pottenbakker heeft alle informatie om de bloembakken te vervaardigen. | |
| | Roep de methode Invoice aan. | De rekening voor de klant wordt getoond zoals eerder weergegeven in het voorbeeld. | |
| | Test de indexer van de class Order | De indexer geeft het juiste object terug.<br>Wanneer er een indexnummer wordt gebruikt dat buiten het bereik ligt van de verzameling wordt er een ArgumentOutOfRangeException opgeworpen | |