Gevorderd Programmeren Opdrachten

Week 1

Beschrijving

Het maken van een programma voor het bijhouden van mensen en hun huisdieren.

Doel

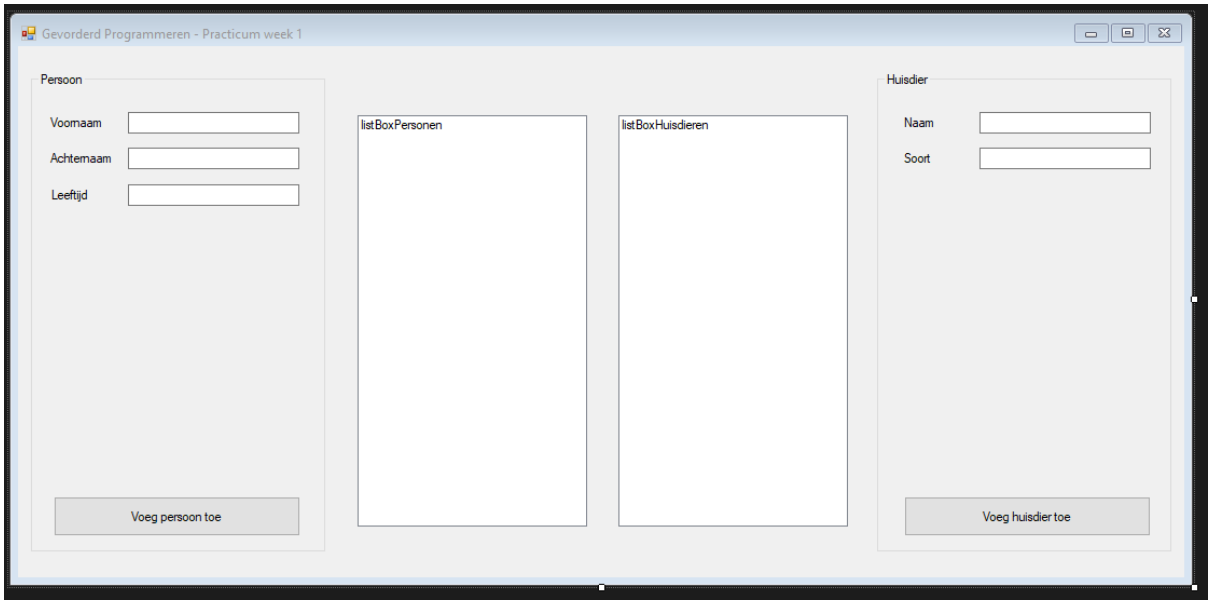
Het doel van deze opdracht is om te leren:

- omgaan met classes
- gebruikmaken van lijsten
- uitbreiden van bestaande code

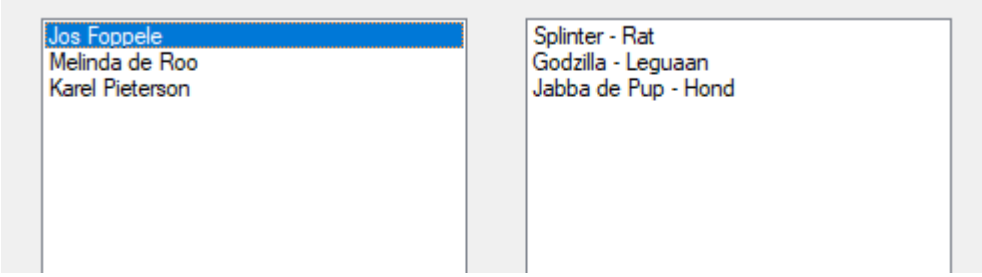
Opdracht

Loop de volgende stappen door:

1. Download de zip "*GevorderdProgrammerenPracticumWeek1 – opdracht*" van elo en pak deze uit op een plek waar jij hem makkelijk kunt terugvinden.
2. Open de solution in Visual Studio, als het goed is zie je onderstaand design.



Dit scherm bestaat uit 2 lijsten en 2 invoergroepen. De bedoeling is om in de lijsten personen en hun huisdieren te laten zien. Als het programma wordt uitgevoerd kunnen de lijsten er dus als volgt uit zien:

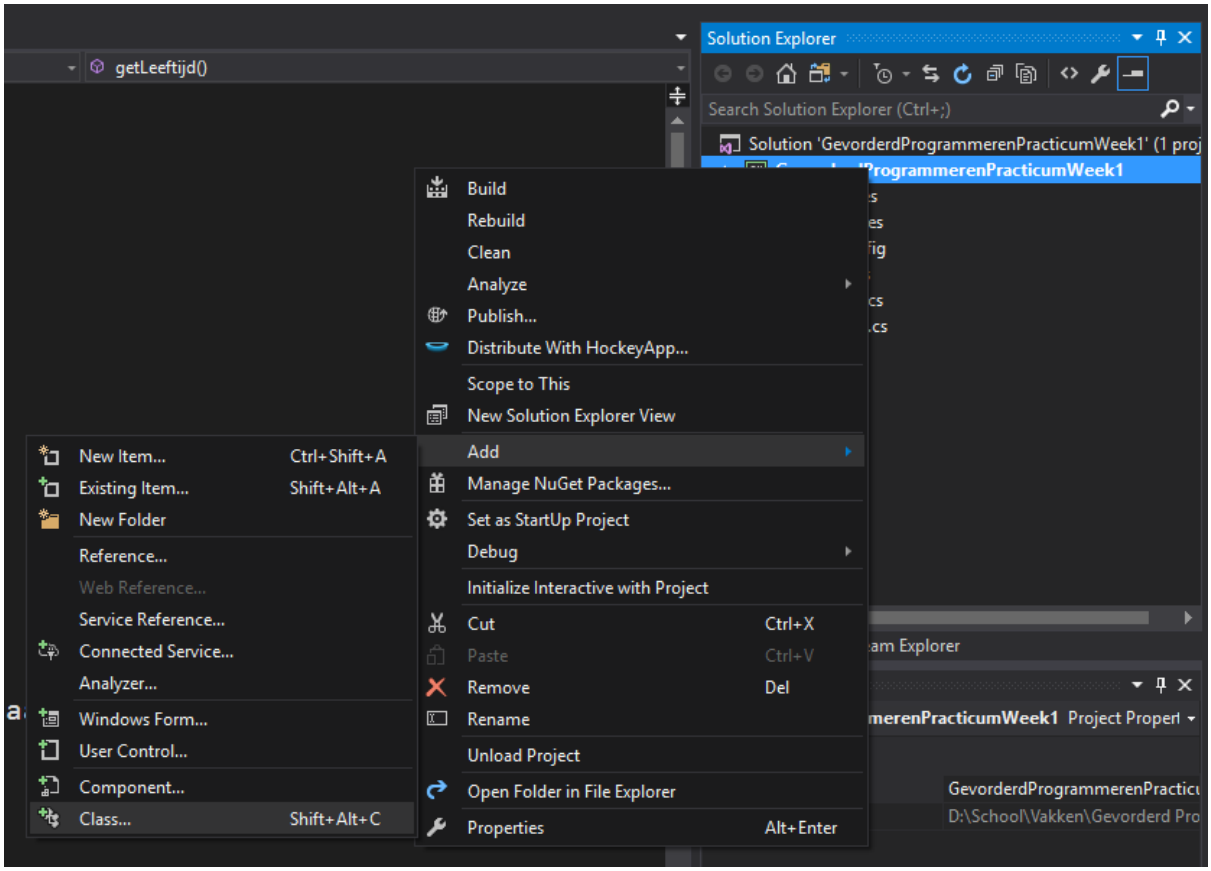


3. Om dit voor elkaar te krijgen moet je in de class Persoon alle methoden vullen met code.
4. Maak daarna de class Huisdier

5. Zorg dat een huisdier een naam en een soort heeft (beide strings) en maak de bijbehorende constuctor(en).
6. Zorg voor de methodes getNaam() en getSoort() en vul deze.
7. Maak de methode getBeschrijving(). Laat deze een string teruggeven waarin de naam en de soort van het huisdier terugkomen.
8. Voorzie Persoon en Huisdier van XML Doc
9. -OPTIONEEL- Breid de interface en de classes uit met extra velden. Je kunt denken aan een woonplaats (string) bij een persoon of een checkbox voor zindelijkheid (boolean).

Tip

Een nieuwe class aanmaken kan lastig zijn als je dit nog nooit hebt gedaan. Je kunt een nieuwe class aanmaken (in Windows) door in de Solution Explorer met rechts te klikken op het project en dan te kiezen voor Add -> Class (zie afbeelding).



Week 2

Beschrijving

In tegenstelling tot mensen en hun huisdieren, gaan we ons nu bezig houden met helden en schurken (heroes and villains).

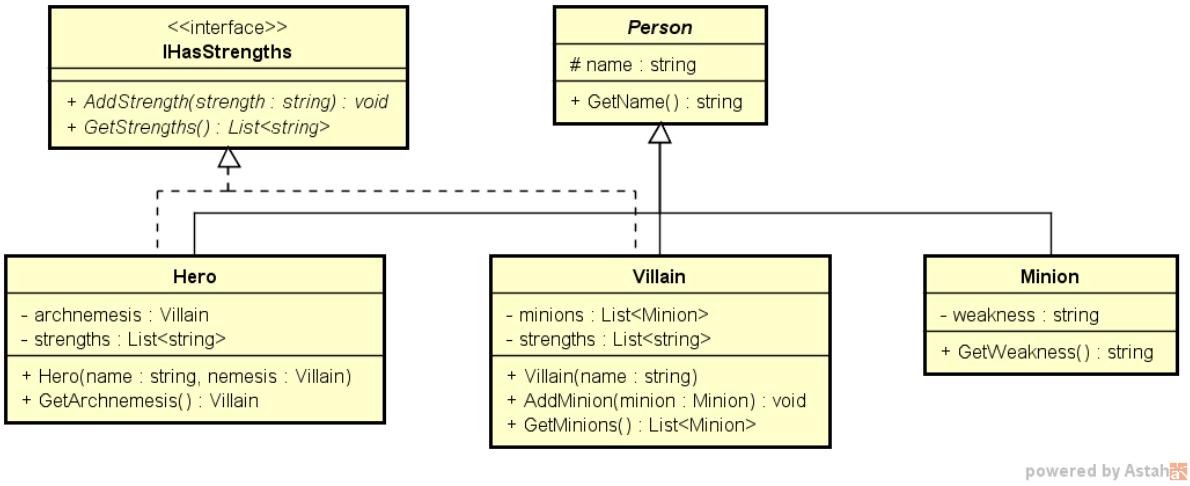
Doel

Het doel van deze opdracht is om te leren:

- omgaan met overerving
- gebruikmaken van abstracte class
- gebruikmaken van een interface

Opdracht

Gegeven is het volgende klasse diagram.



Maak onderstaande opdrachten.

1. Werk het gegeven klassediagram uit in code. Bedenk hierbij zelf de constructor voor Minion.
2. Breid de klasse Hero uit met een sidekick. Een hero kan maximaal 1 sidekick hebben. Een sidekick kan een andere hero zijn, maar ook een villain of een minion. Zorg er voor dat een Sidekick aanpasbaar is met de methode UpdateSidekick() en opvraagbaar is met de methode GetSidekick().
3. Een persoon is vrijwel nooit geheel goed of slecht. Voeg hiertoe aan Person een evilnessIndicator toe die kan varieren tussen 0 en 100. Als een evilness buiten deze range wordt opgegeven, moet de evilness op 50 gezet worden. De evilness moet alleen gezet kunnen worden in de constructor. Zorg voor nieuwe constructors zonder de oude constructors weg te halen of te veranderen. Evilness moet opvraagbaar zijn met behulp van de methode "getEvilness()".
   P.S. evilness loopt van 0 (totale goedheid) tot 100 (most evil ever a.k.a. docent)
4. Maak XML Doc bij al de classes en methodes.
5. Maak een interface (mag grafisch) waarbij je bewijst dat alle classes volledig werken. Ter inspiratie staat een voorbeeld main method op blackboard. Een GUI hierbij is optioneel.
6. Bedenk wat je zou moeten doen om een lijst met personen te sorteren op evilness (je hoeft dit niet uit te programmeren).
7. -OPTIONEEL- Programmeer jouw oplossing van vraag 6 uit en test of het werkt.

Week 3

Beschrijving

Je kunt data bewaren in variabelen, arays, lijsten of databases. Maar natuurlijk kun je ook een eigen datastructuur hiervoor maken. Deze week gaan we een eigen stack en queue maken.

Doel

Het doel van deze opdracht is om te leren:

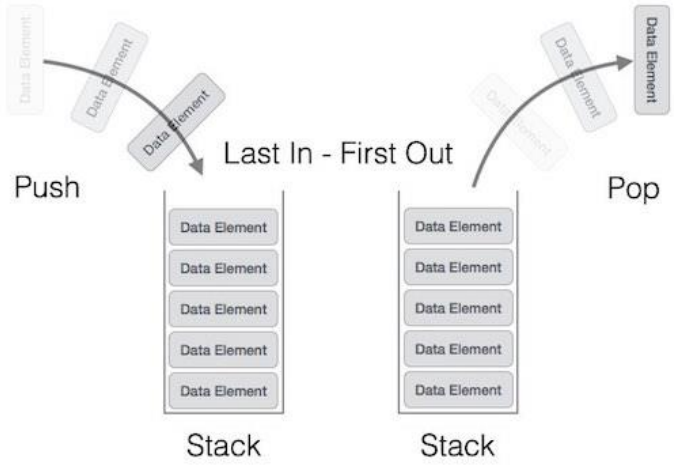
- hoe je een eigen datastructuur opzet en gebruikt
- hoe een stack en een queue werkt

Opdracht

Maak onderstaande opdrachten.

1. Creeer een project met een GUI. Binnen dit project werk je onderstaande opdrachten uit.
2. Maak de klasse Stack. De Stack heeft een private lijst met elementen van type T, gebruik dus generics. Daarnaast heeft een Stack drie publieke methoden, namelijk:
    a. void Push(T item)
    b. T Pop()
    c. bool IsEmpty()

    Push en Pop zijn in onderstaand plaatje uitgelegd, IsEmtpy geeft true terug als de stack leeg is (geen items bevat).



3. Maak de methode ControleerString (ergens buiten de Stack) die met behulp van een Stack controleert of de haakjes in een string goed gevormd zijn. Voorbeelden van goedgevormde strings:

    []{}(<>)
    [[]{}<()>]
    List<string> strings = new List<string>();

    Voorbeelden van niet goed gevormde strings zijn:

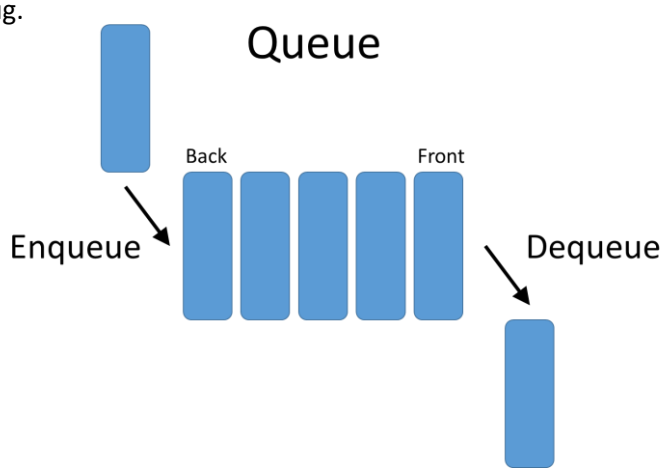
    [{]}
    [](){}
    {{}]
    {{{{}}
    ((()))

    Je kunt deze controle uitvoeren door ieder openhaakje (, {, [, <, dat je tegenkomt op de stack te pushen en bij ieder sluithaakje te controleren of het matchende haakje bovenop staat.

Verwerk deze methode in je interface zodat je makkelijk in strings kunt controleren of de haakjes goed gevormd zijn.

4. Maak de klasse Queue. De Queue heeft een private lijst met elementen van type T. De Queue heeft 3 publieke methoden, namelijk:
    a. void Enqueue(T item)
    b. T Dequeue()
    c. int Length()

    Enqueue en Dequeue zijn in onderstaande afbeelding uitgelegd. Length geeft de lengte van de Queue terug.



5. Maak in je GUI de lengte van een Queue<int> zichtbaar. Je mag er ook voor kiezen om de gehele queue zichtbaar te maken met bijvoorbeeld een listbox. Zorg er voor dat een timer iedere seconde een element dequeued uit deze Queue. Daarnaast moet je met een knop random elementen aan de queue toe kunnen voegen.

Week 4

Beschrijving

Deze week gaan we meerdere verschillende recursieve functies maken. We gaan dit testen met het Unit Test Framework binnen Visual Studio.

Doel

Het doel van deze opdracht is om te leren:

- omgaan met recursie
- in aanraking komen met Unit Tests

Opdracht

Maak onderstaande opdrachten.

1. Download de zip "*GevorderdProgrammerenPracticumWeek4 – opdracht*" van elo en pak deze uit op een plek waar jij hem makkelijk kunt terugvinden. Controleer dat je het project kun openen en dat je de Unit Tests kunt runnen (er moeten 2 tests passen en 23 failen). Dit doe je door de "Test Explorer"te openen en dan op Run All te klikken. Werkt de Solution niet, kijk dan hieronder bij "zelf aanmaken testsolution" hoe je de testsolution zelf kunt opzetten.
   Tijdens het maken van onderstaande opdrachten is het de bedoeling dat je de tests ongewijzigd laat.
2. Maak de methode Row. De definitie is gegeven in Recursion.cs. Row levert een getal op volgens de gegeven som:
   $$Row = \sum_{i=0}^{n} \frac{1}{2^n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots + \frac{1}{2^n}$$
   Bereken de uitkomst van Row recursief.
3. Implementeer de methode SimplePow(b, e) die b^e berekent. Deze berekening werkt als volgt:
   $2^7 = 2 \cdot 2^6$
   $2^6 = 2 \cdot 2^5$
   $2^5 = 2 \cdot 2^4$
   $\vdots$
   $2^1 = 2 \cdot 2^0$
   $2^0 = 1$
   Je mag de methode Math.Pow dus **NIET** gebruiken.
4. Implementeer de methode FastPow(b, e) die b^e berekent op een efficiëntere manier. Deze berekening is als volgt gedefinieerd:
   a. $b^{2e} = (b \cdot b)^e$ // recursieve definitie voor **even** exponent
   b. $b^{2e+1} = b \cdot (b \cdot b)^e$ // recursieve definitie voor **oneven** exponent
   c. $b^0 = 1$ // stopconditie
   Je mag de methode Math.Pow of SimplePow **NIET** gebruiken
5. *Een palindroom is een woord, dat indien het omgedraaid wordt dit hetzelfde woord is.* Implementeer de methode IsPalindroom recursief zodat deze kan bepalen of een gegeven string een palindroom is of niet. Gebruik hierbij alleen de index operator, en dus **GEEN** methoden van de string class.
6. Implementeer de methode Sum die recursief de som berekent van een lijst door iedere keer het eerste element van de lijst op te tellen bij de som van de rest van de lijst.
7. Implementeer de methode IsSorted die recursief bepaald of een gegeven lijst gesorteerd is van klein naar groot.

8. Implementeer de methode Merge die twee gesorteerde lijsten samenvoegt tot 1 gesorteerde lijst. Omdat de gegeven lijsten gesorteerd zijn kun je het volgende doen:
   Kijk naar het eerste element van beide lijsten en bepaal welke van de twee de kleinste is.
   Haal deze kleinste uit de oorspronkelijke lijst en zet hem vooraan in de resultaatlijst.
   Voeg daarna de gemerge resterende lijsten toe aan de resultaatlijst.
9. Implementeer de methode MergeSort die recursief een lijst sorteerd met behulp van de bij opdracht 8 gemaakte Merge methode. MergeSort splitst de gegeven lijst door de helft in 2 lijsten. De twee lijsten worden gesorteerd (met MergeSort) en daarna gemerged (met Merge).
10. Breid de klasse MyList uit met de methode GetLast. GetLast geeft recursief het laatste element van de lijst terug.
11. Breid de klasse MyList uit met de methode Count. Count telt recursief het aantal elementen in de lijst.
12. Breid de klasse MyList uit met de methode Reverse. Reverse draait recursief de volgorde van de elementen in de lijst om.

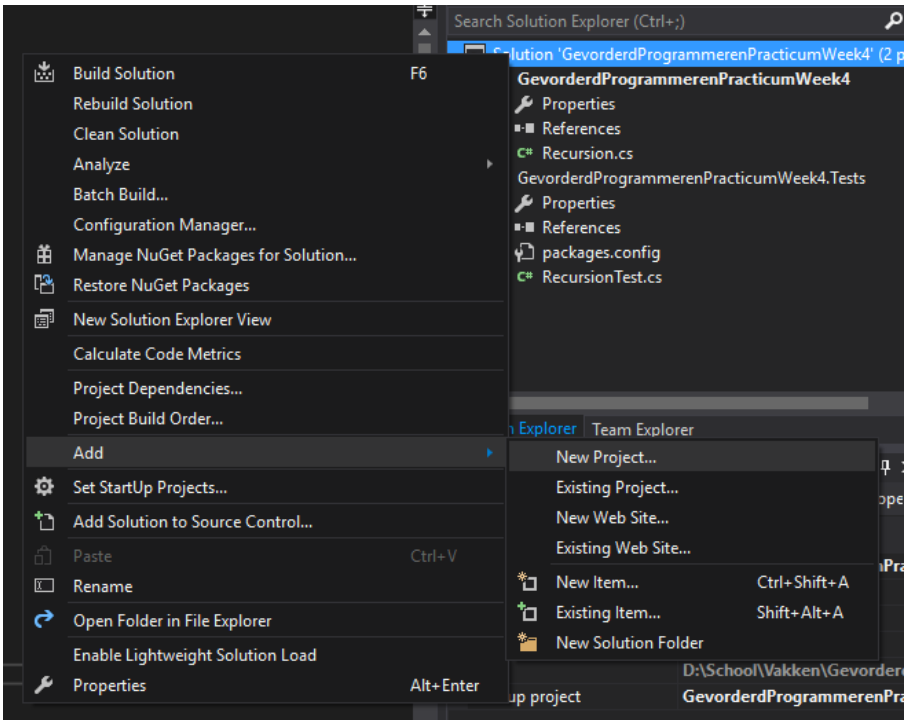
Extra info

Voor meer info over Unit Test kun je bijvoorbeeld kijken naar https://stackify.com/unit-testing-basics-best-practices/ of https://www.youtube.com/watch?v=8YFZBNFm0OM.

Zelf aanmaken testsolution

Als het openen van de testSolution niet werkt, volg dan de volgende stappen om alsnog de Unit Tests aan de praat te krijgen:

1. Maak een nieuw project aan. Dit moet een “Class Library (.NET Framework)” zijn.
2. Voeg aan de gemaakte solution nog een project toe, door in de solution explorer met rechts te klikken op de solution (zie afbeelding).



3. Voeg nu een “Unit Test Project (.Net Framework)” toe aan de solution. Je kunt deze vinden onder C# Test.
4. Voeg in het test project een referentie toe naar het originele project. Doe dit door met rechts te klikken op de References en dan via “Add Reference” het andere project uit de Solution te selecteren.
5. Verwijder de aangemaakte .cs files uit de projecten en voeg de files MyList.cs en Recursion.cs toe aan het originele project.
6. Voeg de file RecursionTest.cs toe aan aan het test project.

Week 5

Beschrijving

Bomen zijn belangrijke datastructuren. Deze week staat geheel in het teken van binaire bomen.

Doel

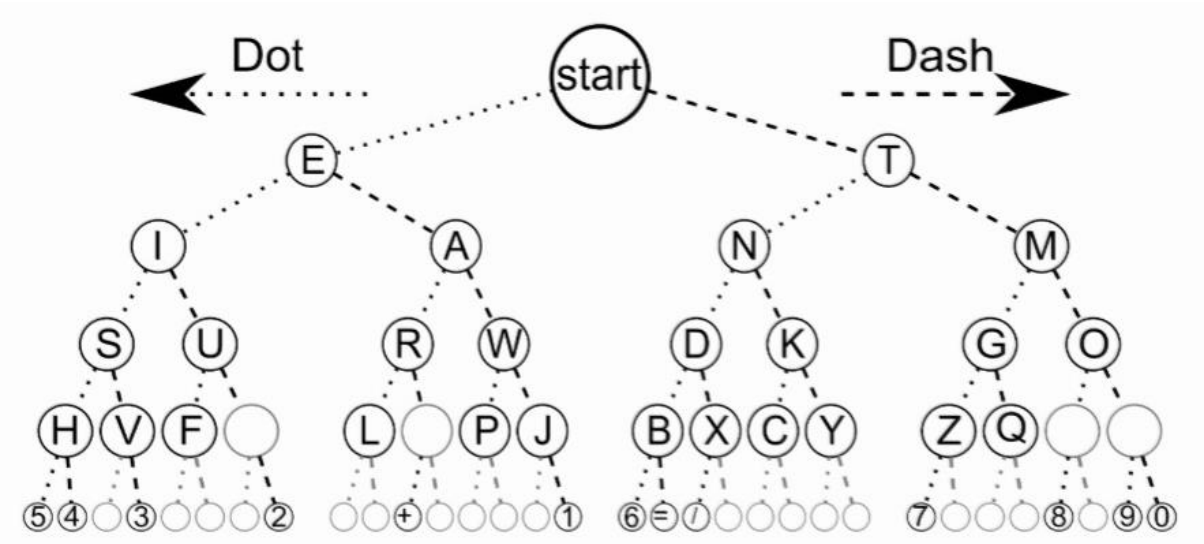
Het doel van deze opdracht is om te leren:

- toepassen van recursieve binaire (zoek)bomen

Opdracht

Maak onderstaande opdrachten.

1. Download de zip "*GevorderdProgrammerenPracticumWeek4 – opdracht*" van elo en pak deze uit op een plek waar jij hem makkelijk kunt terugvinden. Controleer dat je het project kun openen en dat je de Unit Tests kunt runnen (er zijn 19 tests, ze failen allemaal).
2. Vul de gegeven IntTree aan door de volgende methodes te implementeren:
    a) Implementeer de methode Min() die het kleinste getal uit de boom teruggeeft.
    b) Implementeer de methode Max() die het grootste getal uit de boom teruggeeft.
    c) Implementeer de methode Count() die het aantal elementen (getallen) in de boom telt en teruggeeft.
    d) Implementeer de methode InOrder(). Deze methode geeft een string terug met daarin alle getallen in de boom gesorteerd van klein naar groot. De getallen worden gescheiden door een spatie. Je kunt dit vanuit een node (een punt in de boom) doen door eerst de kleinere onderliggende getallen inOrder te sorteren, dan het getal in de node er aan toe te voegen en daarna de grotere getallen inOrder te plakken.
3. Gegeven is de class MorseTree. In deze class kunnen morsecodes gecodeerd worden opgeslagen (zie onderstaande afbeelding). Door deze boom te gebruiken zie je bijvoorbeeld dat de letter e in morse . is, en dat de letter c -.-. is in morse.



Het aanmaken van de morseboom is al verzorgd in de code.
Implementeer nu de volgende methodes:

    a) Implementeer de methode GiveCharacterByMorseCode(string morsecode) die een karakter uit de boom haalt op basis van een gegeven morsecode. De methode geeft het karakter '?' terug als een code geen resultaat oplevert/niet klopt.
    b) Implementeer de methode GiveMorsecode(char c, string code). Deze methode wordt aangeroepen door GiveMorsecode(char c). Deze methode bepaalt welke morsecode bij een bepaald karakter hoort. Deze methode zzal bijvoorbeeld bij input

'r' de string ".-." teruggeven. Doe dit door de code uit te breiden met een punt of een streep als je dieper in de boom gaat en bij het vinden van het juiste character de code terug te geven. Indien een karakter niet gevonden kan worden moet een lege string terug gegeven worden.
De in deze opgave gebruikte techniek heet een opsparende parameter.

4. Gegeven is een Person klasse. Een persoon bevat een naam en een evilness (werking is vergelijkbaar met de Person uit week 2). De Person class hoeft niet aangepast te worden. Evilness is niet publiek beschikbaar, maar wel de methode IsMoreEvil() welke bepaald of een persoon slechter (more evil) is dan meegegeven persoon.
    a) Implementeer de methode Add() die een persoon toevoegt aan de boom. Beschouw de boom als een binaire zoekboom waarbij persoonen die slechter zijn (more evil) in de moreEvil tak komen. Je hoeft hierbij geen rekening te houden met personen die even evil zijn.
    b) Implementeer de methode GetAllNamesSorted() welke alle namen in de boom oplevert in 1 string. De namen zijn gesorteerd van goed naar slecht en zijn gescheiden door de tekens " - " (spatie – streepje – spatie).
    c) [OPTIONEEL] Integreer de Villain en Hero class uit week 2 en probeer Heroes en Villains in de boom te zetten. Bedenk ook de bijbehorende tests.

Afsluitende opdracht (ter inlevering op blackboard)

Beschrijving

Een HP calculator is een calculator die berekeningen volgens de RPN (postfix notatie) uitvoert. Voorbeeld van een postfix berekening: 7 2 + is 9.

Doel

Het doel van deze opdracht is aan te tonen dat je:

5. kunt omgaan met overerving
6. gebruik kunt maken van abstracte class
7. recursie kunt implementeren
8. datastructuren recursief kunt gebruiken

Opdracht

Maak een HP calculator die aan de volgende punten voldoet (zie volgende pagina voor weging):

- De HP calculator heeft 10 digit knoppen waarmee een cijfer in de inputbox geplaatst kan worden.
- Berekeningen worden postfix uitgevoerd (6 3 / is 2)
- Cijfers in de inputbox moeten op de stack gepusht kunnen worden.
- De HP calculator ondersteunt drie eigengemaakte typen stack (ArrayStack, ListStack en MyListStack)
- Er moet een abstracte klasse Stack zijn waar de drie eigengemaakte stacks van erven.
- Er mag runtime maar één stack in het geheugen staan.
- Runtime moet er geswitcht kunnen worden tussen verschillende stack typen.
- MyListStack maakt gebruik van een zelf gedefinieerd type MyList.
- De inhoud van de stack moet zichtbaar zijn voor de gebruiker.
- Vermenigvuldigen, delen, optellen en aftrekken wordt door de calculator ondersteund.
- De GUI moet zoveel mogelijk dynamisch gegenereerd worden (dynamisch betekent dat de knoppen niet vooraf gemaakt zijn met de designer, maar door de eigen code worden gegenereerd)

Een voorbeeld HP calculator is te vinden op blackboard. Kijk ook goed naar de rubric ter beoordeling om te zien waar je HP calculator aan moet voldoen.

Beoordeling

De beoordeling van de opdracht vindt plaats met behulp van onderstaande rubric.

	Slecht	Onvoldoende	Voldoende	Goed
GUI	**0** GUI werkt niet	**0.3** GUI functioneel in de basis. Geen weergave van de stack	**0.6** GUI functioneel met weergave van de stack	**1** GUI functioneel met weergave van de stack. Nummerknoppen zijn dynamisch opgebouwd en reageert op numpad toetsaanslagen.
Stack	**0** Geen drie eigengemaakte stack typen	**0.75** Drie eigengemaakte stack typen die erven van een abstracte Stack class	**1.5** Drie eigengemaakte stack typen die erven van een abstracte Stack class. Er staat maar 1 stack in het geheugen.	**2.5** Drie eigengemaakte stack typen die erven van een abstracte Stack class. Er staat maar 1 stack in het geheugen en de stacks zijn efficient geprogrammeerd (geen dubbele code).
Recursie	**0** Geen recursie	**0.75** Recursie geimplementeerd in MyList maar dit werkt niet goed.	**1.5** Recursie geimplementeerd in MyList.	**2.5** Recursie efficient en logisch in MyList.
Code technisch	**0** Geen code	**0.6** Spaghetticode	**1.2** Code opgesplitst in aparte classes en methoden.	**2** Code opgesplitst in aparte classes en methoden zonder dubbel stukken code. Hoge efficientie.
Code opmaak	**0** Onleesbare code	**0.6** Leesbare code zonder onnodige witregels en goed uitgelijnd	**1.2** Leesbare code voorzien van commentaar	**2** Goed leesbare code voorzien van commentaar en XML Doc volledig ingevuld

Tabel 1: Rubric beoordeling HP calculator