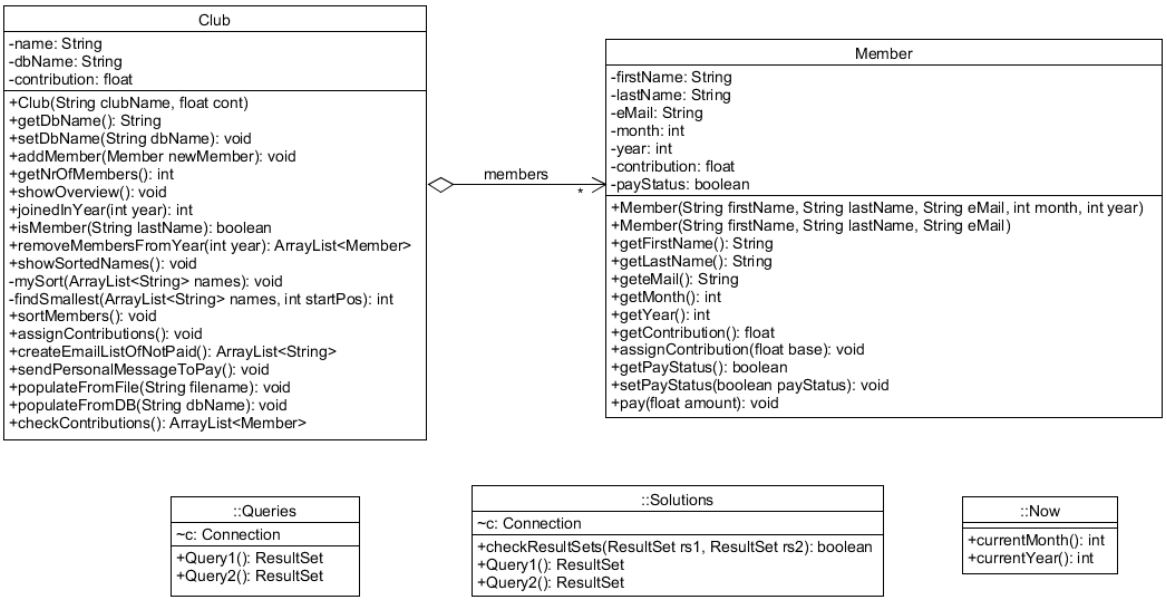
Labo opgave over collections

1. De opzet is om het bestaand Products project van hoofdstuk 4 uit te breiden. Wordt eerst vertrouwd met de bestaande methodes. Voorzie de nieuwe methodes van betekenisvolle namen en de correcte parameters. Gebruik de StockDemo klasse om de nieuw toegevoegde functionaliteit uit te testen.
De StockDemo klasse is een voorbeeld van hoe je een extra klasse gebruikt om de geïmplementeerde functionaliteit te testen i.p.v. dit interactief in BlueJ te doen of met een JUnit testklasse. Als je kijkt naar de constructor van StockDemo zie je dat hierin een StockManager object aangemaakt wordt en hier een aantal Product objecten aan toegevoegd worden. In de demo methode zie je een voorbeeld van een (klein) testscenario: eerst de huidige toestand opvragen, dan een levering simuleren en dan terug de toestand opvragen om na te gaan of het aantal in stock effectief gewijzigd is. Je kan dit later aanpassen om zo ook de nieuw toegevoegde functionaliteit uit te testen.

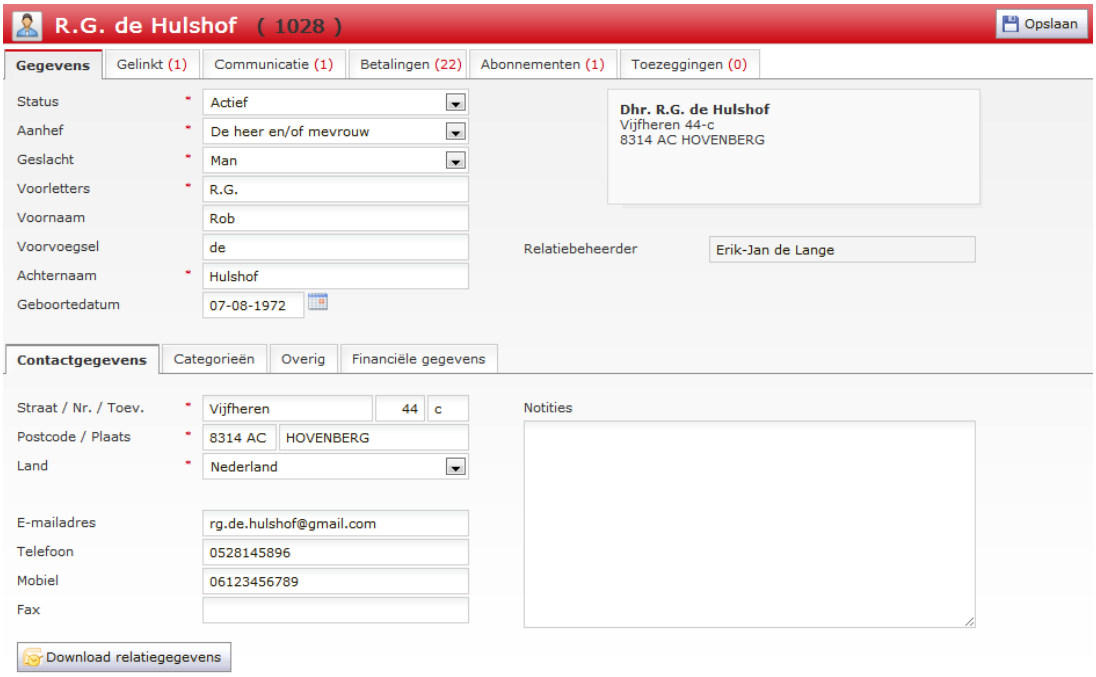
In de eerste versie van StockDemo wordt er een StockManager object aangemaakt en nadien worden er 3 verschillende Product objecten aangemaakt om zo de initiële inhoud van de stock aan te genereren.
Skeleton van a. tot d. is al aanwezig in de Stockmanager van dit project.

a. Implementeer de methode die een overzicht geeft van alle aanwezige producten. Gebruik hiervoor de toString() methode van Product.
b. Implementeer de methode om een product te vinden gebaseerd op het ID (als het product niet aanwezig is in de lijst wordt null teruggegeven).
c. Voorzie een methode om te weten hoeveel items er in stock zijn van een gegeven product. Als het product niet aanwezig is wordt er 0 teruggegeven. Gebruik vorige methode om het product op te zoeken.
d. Voorzie de mogelijkheid om de hoeveelheid in stock van een bepaald product te laten toenemen (simulatie van een levering).
e. Voeg een methode toe die een overzicht geeft van alle producten die minder in stock zijn dan een gegeven aantal.
f. Creëer een methode om een product te krijgen op basis van zijn naam. (vergeet niet om equals() te gebruiken!)
g. Wijzig de methode addProduct(). Controleer eerst of het product al in de lijst aanwezig is en wijzig dan alleen het aantal.
h. Gebruik de aanwezige methode populateFromDB() om alle informatie uit de SQLite databank Products.db in te voeren in de StockManager.

2. We willen software ontwikkelen om de ledenadministratie van een vereniging te automatiseren. Van elk lid houden we familienaam, voornaam, e-mailadres, de maand en jaar dat ze toegetreden zijn bij. Een vereniging kan natuurlijk een willekeurig aantal leden bevatten.
Maak zo veel mogelijk gebruik van bestaande functionaliteit uit de JAVA bibliotheken.



Start van het gegeven project clubstart met 2 lege klassen Club en Member, en een testklasse die een aantal testen met toenemende complexiteit implementeert. In de Queries klasse moet je enkel de query string invullen op de juiste plaats. De uitgewerkte testen coveren topics a tot en met s. Van de klassen Solutions and Now bezit je enkel de compiled code. Zij worden door de testklasse gebruikt. Het is natuurlijk perfect mogelijk dat je later nog attributen en/of methoden moet toevoegen aan deze vertreksituatie.



a. De ***constructor*** van Club verwacht een naam en het basisbedrag dat leden moeten betalen als jaarlijks lidgeld. Verder wil je weten hoeveel leden de vereniging telt: ***getNrOfMembers()***

b. Test de mogelijkheid om leden toe te voegen: ***addMember(Member).*** Zorg er voor dat je geen twee leden kan toevoegen die zelfde voornaam en familienaam hebben.

c. Integreer dan een databank in je applicatie die alle nodige informatie bevat. De databank heeft 3 tabellen: Club, Member en ContributionPayment en zit ook in de zip file als clubs.db. Implementeer de methode ***populateFromDB(String)*** die alle informatie uit de tabel Member gebruikt om alle overeenkomstige Member objecten te creëren en toe te voegen aan de collectie met behulp van addMember(). De methode slaat ook de naam van de DB op in het attribuut dbName. De databank bevat gegevens over meerdere clubs, hier zijn we alleen geïnteresseerd in de leden van onze club (clubnaam is gebruikt als primary key)

d. Geef een tekstueel overzicht van alle aangesloten leden: ***showOverview().*** Gebruik de informatie uit de testklasse om te weten in welk formaat je de informatie moet weergeven. (test5, lijnen 104 en 106)

e. Om te weten hoe succesvol een club is willen we weten hoeveel leden zijn er aangesloten in een gegeven jaar: ***joinedInYear(int)***

f. Controleer of er een lid is met een gegeven familienaam: ***isMember(String)***

g. Verwijder alle leden die aangesloten zijn in een bepaald jaar. Geef de verwijderde leden terug als een aparte collectie: ***removeMembersFromYear(int)***

h. Genereer een lijst van alle familienamen gesorteerd in alfabetische volgorde (hint: gebruik de sort methode van java.util.ArrayList) en toon ze lijn per lijn: ***showSortedNames()***

i. Elk lid moet een jaarlijkse bijdrage betalen. Voor "The red Devils" is dit bedrag vastgelegd op €300, maar een lid krijgt 10% korting voor elk jaar dat hij/zij al lid is: ***assignContributions().*** Voeg deze informatie toe aan het overzicht.

j. In het overzicht willen we maandnamen zien in plaats van getallen, dus "mei" ipv "5". Het veld maand in Lid blijft ongewijzigd van type int. (***Hint: gebruik een static array met de namen van de maanden***)

k. Voorzie de mogelijkheid om te weten of een lid al betaald heeft: ***getPayStatus()***

l. Voorzie de mogelijkheid om de betaalstatus van een lid aan te passen: ***setPayStatus(boolean)*** of een betaling uit te voeren met ***pay(float).*** Als je het nodige bedrag betaald hebt wordt je status automatisch op true gezet.

m. We gaan nu ook een aantal queries uitvoeren vanuit ons Java programma. Je beschikt hiervoor over de klasse Queries. Jij moet enkel de String invullen met de correcte query. Je mag natuurlijk altijd eerst de query uitproberen in SQLiteStudio.
- Query1: geef een overzicht van het aantal leden per jaar van aansluiting van onze club .. Oudste leden eerst.
- Query2: geeft voornaam, familienaam en totaal bedrag van het lid dat in de hele periode het meeste lidgeld betaald heeft.

n. Genereer een lijst met e-mailadressen van alle leden die nog niet betaald hebben voor het huidig jaar: ***createEmailListOfNotPaid()***

o. Gebruik deze lijst om naar deze mensen een gepersonaliseerde (minstens naam en bedrag moeten specifiek zijn) e-mail te sturen om hen aan te sporen om te betalen. Schrijf deze boodschappen naar de terminal als simulatie.

p. In plaats van alle leden manueel in te laden willen we alle informatie inlezen van een bestand. Dit bestand bevat per lijn alle info over een lid gescheiden door blanco's familienaam voornaam e-mail startmaand startjaar betaalstatus (neem voor de eenvoud aan dat familienaam of voornaam geen blanco's bevat): ***populateFromFile(String)***

q. Voorzie een 2e versie van de constructor van Member() waar je maand en jaar niet moet specifiëren. Gebruik gewoon de huidige datum. (***Hint: gebruik de Java.time.LocalDate klasse***)

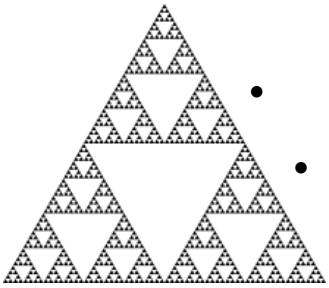
r. The club tabel heeft informatie over het standaardbedrag dat een nieuw lid jaarlijks moet betalen. Vergeet niet dat je per jaar dat je al lid bent 10% korting krijgt op dit bedrag (dus ja, na 10 jaar hoef je niets meer te betalen). Alle betalingen van leden zijn opgeslagen in ContributionPayment. Implementeer nu de method ***checkContributions()*** die een lijst van Members teruggeeft die minstens voor 1 jaar niet voldoende betaald hebben. Leden die niet meer moeten betalen hebben geen nieuwe records meer in de ContributionPayment tabel. Je mag er van uitgaan dat iedereen reeds een betaling deed in 2018.

s. Bedenk wat er allemaal moet gebeuren bij de start van een nieuw werkingsjaar. Welke acties moeten ondernomen worden? Welke informatie heb je nodig? Automatiseer dit.

t. Zet dit project om in een versie die je ook zonder BlueJ kan oproepen (executable jar). Simuleer in de main method de start van een nieuw werkjaar en genereer dan de lijst van e-mails naar iedereen die nog niet betaald heeft.

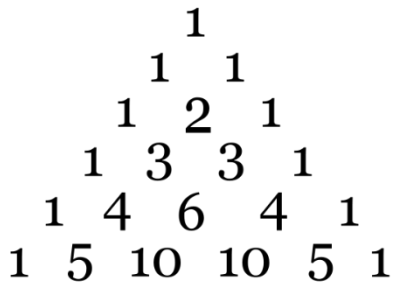
3. **Automatiseren van bestellingen in de KursusDienst [Herbruik zo veel mogelijk bestaande klassen uit voorgaande 2 opgaven]**
We willen software ontwerpen voor de KursusDienst van Industria. Als studenten zich inschrijven krijgen ze een formulier waarmee ze handboeken kunnen bestellen. Het formulier zelf gaan we niet ontwerpen, maar onderstel dat jouw applicatie volgende informatie binnenkrijgt per student: naam, voornaam en een lijst van bestelde boeken. (we onderstellen even dat de combinatie naam + voornaam uniek is) De cursusdienst beschikt over een voorraad boeken. Deze informatie zit opgeslagen in een databank. Neem aan dat elk boek over een unieke titel beschikt en verder weet je enkel hoeveel exemplaren er beschikbaar zijn. Deze informatie wordt ingelezen bij de start van het programma. Telkens een formulier ingevoerd wordt moet je controleren of er nog voldoende handboeken in voorraad zijn, en indien wel het aantal in voorraad aanpassen. Als er geen handboek meer voorradig is, krijgt de student een e-mail (simuleer dit met een gepersonaliseerde String) dat het boek niet voorradig is en wordt de bestelling opgenomen in een wachtlijst. Als alle boeken beschikbaar zijn genereert de applicatie een overzicht van de gekochte boeken samen met de totale prijs. Je hoeft dus alleen volledige bestellingen te verwerken.
Verder moet de voorraad natuurlijk kunnen aangevuld worden. Na een levering wordt de wachtlijst gecontroleerd om te zien of bepaalde bestellingen nu wel volledig afgewerkt kunnen worden. Zorg er voor dat de student hiervan op de hoogte gebracht wordt.
Voor je het programma afsluit moet je ook nog de mogelijkheid hebben om de informatie in de databank up te daten.
Gebruik je eigen creativiteit om onduidelijkheden, aanvullingen en zo te maken.
[Extra] Maak een primitieve versie van een invulformulier. Maak een programma dat je kunt oproepen van uit een terminal dat een KursusDienst object aanmaakt met een zeker aantal boeken in stock. Vraag dan in een lus via de terminal telkens naam, voornaam en titel van boek op. Verwerk dan de bestelling en toon resultaat. Maak gebruik van de java.util.Scanner klasse om gegevens van de terminal te lezen.

4. **“Pascal meets fractals”**



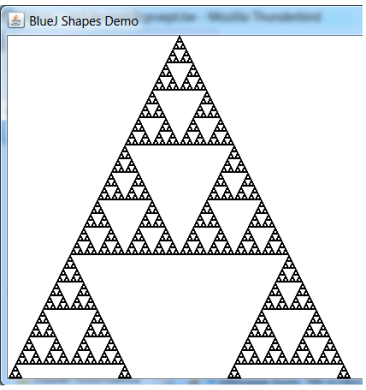
De Sierpinski zeef is een voorbeeld van een fractal, een geometrische figuur met een niet-gehele dimensie (de Hausdorf dimensie is 1.585). Dergelijke figuren worden gegeneerd met een simpel recursief proces

- Bepaal een driehoek en neem een willekeurig startpunt in de driehoek
- Kies willekeurig 1 van de 3 hoekpunten en zet telkens een punt exact op de helft van het startpunt en het gekozen hoekpunt
- Maak van dit nieuwe punt het startpunt en herhaal deze procedure zoveel je maar wil.



Anderzijds is er de driehoek van Pascal, waarbij elk getal (buiten de 2 schuine zijden met allemaal 1-en) de som is van zijn 2 bovenburen. Die driehoek wordt o.m. gebruikt om de waarden te kennen bij de uitwerking van $(x + a)^n$. Bijvoorbeeld $(x + a)^5 = 1x^5 + 5x^4a + 10x^3a^2 + 10x^2a^3 + 5xa^4 + 1a^5$.

Nu is er een grappige overeenkomst tussen beiden: als je in de driehoek van Pascal alleen een punt tekent op de plaatsen waar een oneven getal staat, krijg je ook een (gedeeltelijke) Sierpinski zeef, zoals je in bijgevoegd voorbeeld kunt zien.

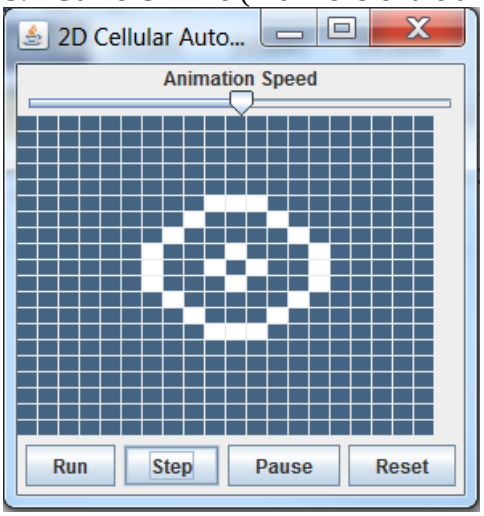


(zie http://en.wikipedia.org/wiki/Pascal%27s_triangle) Gebruik deze overeenkomst om een afbeelding te genereren van een Sierpinski zeef door gebruik te maken van een driehoek van Pascal. Je kan hiervoor terug gebruikmaken van het Figures project en punten voorstellen als vierkanten met zijde 2.

TIP: je hoeft niet de hele driehoek van Pascal in geheugen te houden, je hebt alleen de voorgaande rij nodig om een nieuwe te berekenen.

Probeer een aantal rijen te vinden dat een mooie gesloten driehoek oplevert (zoals bvb. bij 4)

5. Game Of Life (2-dimensionale arrays)



Je vertrekt van het project GameOfLife op Toledo. Dit voorziet al een klasse om de visualisatie te realiseren (EnvironmentView). Hier moet je niets aan wijzigen. De andere klassen moet je zelf nog aanvullen. Een Cel is de basis van alles en heeft een bepaalde State (ALIVE of DEAD). Environment bevat een 2-dimensionale array van Cellen die door elke oproep naar de methode step() in hun nieuwe toestand gebracht worden. De klassieke regel is: als een levende cel minder dan 2 of meer dan 3 buren heeft, sterft ze. Een levende cel met exact 2 buren blijft leven. Als een dode cel exact 3 levende buren heeft wordt ze ook levend. Elke cel heeft 8 buren. In de methode reset() creëer je een startsituatie (je maakt een aantal cellen levend). Een aantal configuraties die mooie simulaties opleveren kan je vinden op https://en.wikipedia.org/wiki/Conway's_Game_of_Life#Examples_of_patterns.

De View heeft een aantal controls: je kan met de slider de animatiesnelheid aanpassen. De 4 knoppen onderaan zijn:

- Run: start de animatie (oneindige lus)
- Step: doe 1 update
- Pause: stop de animatie
- Reset: gebruik terug de startconfiguratie

Extra oefeningen zijn te vinden op Toledo > OOP&DB > Evaluatie. Daar vind je examenopgaven van vorig academiejaar.