Objectgerichte Softwareontwikkeling 2021-2022
Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Labosessie week 9-10
Collections: ArrayList

Theorie
Op het moment dat deze labosessie plaats vindt, zijn alle hoorcolleges gegeven.

Doel
In dit labo introduceren we de ArrayList, een van de belangrijkste klassen in het Java Collections Framework. Net zoals bij een array kan je meerdere objecten, meestal van hetzelfde type, opslaan in een ArrayList maar zal je merken dat een ArrayList "meer" object-georiënteerd is dan een array. Het abstractieniveau is hoger. De belangrijkste verschillen zijn:

- Een Array heeft een vaste grootte die bij de initialisatie moet opgegeven worden en niet meer kan wijzigen; een ArrayList is dynamisch, dat wil zeggen dat je de grootte vooraf niet moet vastleggen en dat de ArrayList automatisch groeit als je meer elementen wil toevoegen.
- Een ArrayList kan alleen objecten opslaan; geen waarden van een primitief type (bv. int, float, …). Wil je toch een waarde van een primitief type opslaan, moet je deze waarde "inpakken" ("wrappen") in een object. Hiervoor kan je de wrapper classes voor primitieve types, zoals Integer, Float… gebruiken. In de Java api kan je zien dat er voor elk primitief type een wrapper class voorzien is.
- ArrayList's zijn "echte" objecten met veel methoden (zie java api); arrays hebben enkel een length property en de [] operator.

Formaat
Elke labosessie is onderverdeeld in drie verschillende delen: startoefeningen, labo-oefeningen en thuisoefeningen. **Startoefeningen** zijn oefeningen waarvan we verwachten dat je ze thuis oplost, vóór de labosessie. Op die manier geraak je vertrouwd met de materie en check je of je de theorie van de hoorcolleges goed begrepen en verwerkt hebt. **Labo-oefeningen** zijn oefeningen die tijdens de labosessie worden behandeld. **Thuisoefeningen** zijn oefeningen die dieper ingaan op de materie. Ze zijn optioneel voor wie sneller werkt, of als extra oefeningenmateriaal bij het studeren/oefenen thuis. Neem contact op met jouw labo-assistent voor feedback op deze oefeningen.

Leerdoelstellingen
Na deze sessie moet je in staat zijn om de volgende concepten te begrijpen en te implementeren:

- Het verschil kennen tussen array en ArrayList.
- Complexere herhalingslussen gebruiken om naar specifieke elementen te zoeken, elementen te wijzigen of te verwijderen in een ArrayList.
- Informatie terugvinden in de Javadocumentatie.

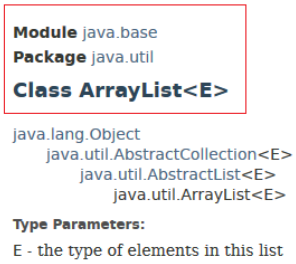
Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Startoefeningen

Oefening 1: ArrayList basics

*Gebruik de Java-API-documentatie over ArrayList voor de volgende oefeningen. ArrayList is een onderdeel van de **java.util** package. Tot en met Java versie 8 (figuur aan de linkerkant) zijn er geen modules in Java. Sinds Java versie 9 (figuur aan rechterkant), kan je deze package terugvinden in de java.base module Voor deze lab sessie over ArrayLists kan je gelijk welke versie van de Java API van de klasse ArrayList gebruiken. Denk eraan dat je al de extra klassen die je gebruikt moet importeren in een "import" statement; dit hoef je niet te doen voor de klassen in de java.lang package, zoals String, Integer, Double…, zij worden automatisch geïmporteerd.*



Voor wie meer informatie wil, zijn er meerdere goede tutorials die de werking van de ArrayList uitleggen, zoals bijvoorbeeld:

- *The Java™ Tutorials: Collections*
- *W3Schools Java ArrayList tutorial (beknopte inleiding)*

Open het ArrayListExercise project en vervolledig de ArrayListExercise klasse. Je kan in de commentaar de opdrachten terugvinden. Voor sommige methoden is een testmethode voorzien in de bijhorende testklasse, zodat je deze onmiddellijk ook kan testen.

Dit zijn de opdrachten:

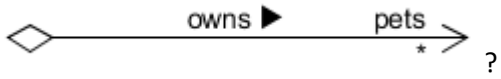
1. Vervolledig het import statement, zodat je een ArrayList kan gebruiken in je code.
2. Vervolledig de myFriends field declaratie (het moet een ArrayList worden die namen van het type String kan bevatten).
3. Initialiseer myFriends in de constructor.
4. Implementeer de addFriends- en printFriends-methodes.
5. Implementeer een getter voor myFriends.
6. Implementeer een getter (getNumberOfFriends()) die het aantal vrienden als returnwaarde heeft (= het aantal elementen in de myFriends collection).
7. Implementeer de addUniqueFriend methode (= zorg ervoor dat een naam niet toegevoegd wordt als deze al in myFriends voorkomt); om je code beter leesbaar te maken kan je een private helper methode gebruiken of … misschien is er wel een geschikte methode in de ArrayList klasse.

Oefening 2: Van UML tot Java classes

Start met een nieuw project en geef het de naam "Petowners"; als je het volgende UML-diagram zou moeten implementeren, hoe ga je deze aggregatie implementeren:

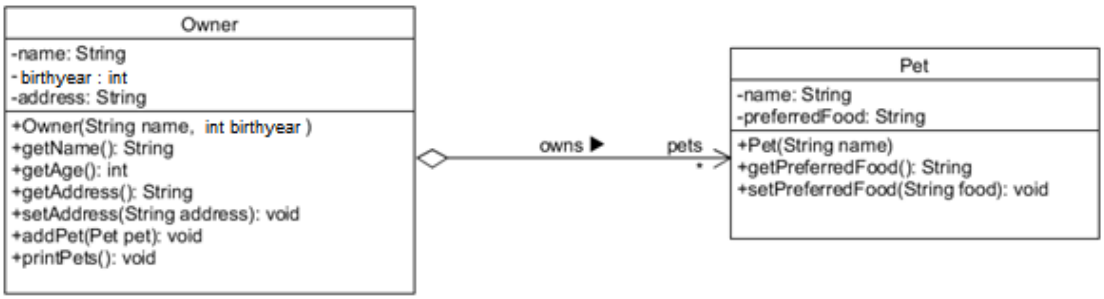
Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.



?

Hint: kijk naar het onderwerp van dit labo…



Objectgerichte Softwareontwikkeling 2021-2022

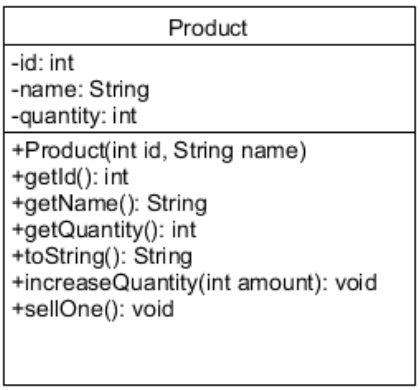
Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Labo-oefeningen

Oefening 1: Bingo

Schrijf een programma dat lottogetallen genereert. Deze getallen gaan van 1 tot en met 75; elk getal mag slechts één keer voorkomen én ze moeten in willekeurige volgorde afgedrukt worden. Je kan hiervoor de klasse `java.util.Random` gebruiken. Gebruik een ArrayList. Je kan deze klasse aan het ArrayListExercise project toevoegen.

Oefening 2: Stock Management



Download 'StockManagement' van Toledo. Er is al heel wat code voorzien. Zoek eerst uit wat de bestaande methoden in de klassen Product en StockManager doen. Lees ook de documentatie bij deze methoden. Je hoeft de klasse Product niet aan te passen.

De skeletoncode voor de eerste opdrachten is voorzien in de klasse StockManager.

a. Implementeer de findProduct-methode om een product terug te vinden op basis van de ID. Als het product niet in de lijst voorkomt, heeft deze functie returnwaarde null, anders is de returnwaarde het gevonden product. Test je code door gebruik te maken van de voorziene test in de testklasse.
b. Implementeer de numberInStock-methode. Deze heeft het aantal items in stock van een gegeven product als returnwaarde. Als het product niet in de lijst voorkomt, moet de returnwaarde 0 zijn. Gebruik de functie uit de vorige vraag. Test je code door gebruik te maken van de voorziene test in de testklasse.
c. Implementeer de delivery-methode. Hiermee kan je de hoeveelheid stock van een bepaald product verhogen. Maak gebruik van de vorige methodes. Test je code door gebruik te maken van de voorziene test in de testklasse.
d. Vervolledig de methode printProductDetails zodat ze alle productinformatie van de producten in stock afdrukt. Gebruik de toString()-methode van de klasse Product. Hiervoor is geen testfunctie voorzien.
e. Schrijf een methode met als returnwaarde een ArrayList, die alle producten in stock met een stockhoeveelheid onder een als parameter gegeven drempelwaarde bevat. Test je code door gebruik te maken van de voorziene test in de testklasse.
f. Pas de addProduct()-methode aan zodat je geen producten kan toevoegen die al in de stock voorkomen; een product wordt uniek geïdentificeerd door de ID. Druk een boodschap af als het product reeds in de stock aanwezig is. Hiervoor is geen testfunctie voorzien.
g. Implementeer de functie totalItemsInStock() die als returnwaarde het totaal aantal items in de stock heeft. Test je code door gebruik te maken van de voorziene test in de testklasse.

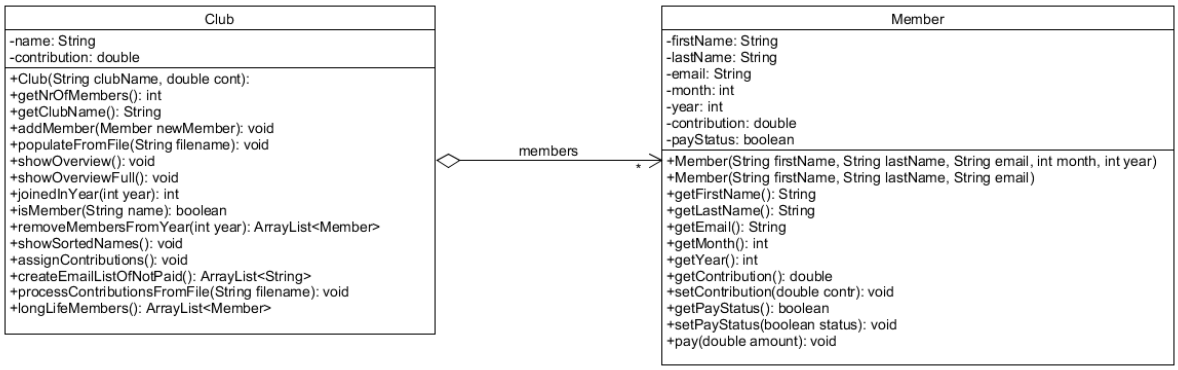
Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Oefening 3: Member Management

Ontwerp software om de administratie van het lidmaatschap van een club te ondersteunen. Voor elk clublid wordt de achternaam, voornaam, e-mailadres en maand en jaar van aansluiting bewaard. Bij een club kunnen een onbepaald aantal leden aansluiten.

Het uiteindelijke UML-diagram ziet er zo uit:



Gebruik het project MemberManagement als start. Je zal hierin twee klassen (Member en Club) terugvinden en een testklasse, die een aantal testen met toenemende complexiteit heeft. Member heeft een aantal vooraf gedefinieerde methoden. Club heeft templatecode die je kan gebruiken om de inhoud van een bestand in te lezen (ook op het examen zal je, indien nodig, dit soort van templatecode krijgen). Gebruik het UML-diagram om de signatuur van de toe te voegen methoden terug te vinden. De testklasse test de volgende functionaliteit:

1. De constructor verwacht als parameters de naam van de club en het basisbedrag dat clubleden jaarlijks moeten betalen. Om te weten hoeveel leden een club heeft implementeer je getNrOfMembers().
2. Bekijk of je een lid kan aansluiten bij een club: addMember(Member m). Zorg ervoor dat je geen lid met dezelfde voor- en familienaam meer dan een keer kan toevoegen.
3. Er is een bestand met naam ”members.txt” aanwezig in het zip-bestand. Dit bestand bevat namen van mogelijke clubleden. Open het bestand zodat je kan zien welke informatie aanwezig is (de eerste rij is een zogenaamde “header”-rij en geeft aan welke informatie in de volgende rijen terug te vinden is). De gegevens in één rij zijn gescheiden door een “;”-teken.

   Implementeer de methode populateFromFile(). Deze methode heeft één parameter: de bestandsnaam (fileName). Gebruik de templatecode. Opgelet: het bestand bevat informatie over spelers van meerdere clubs; wij zijn enkel geïnteresseerd in de leden van “onze” club (test op clubnaam).
   Het inlezen van gegevens uit een bestand is niet altijd eenvoudig. Je moet goed weten wat de bestandsstructuur is, welk scheidingsteken gebruikt wordt en welke methoden van de Scanner klasse je kan toepassen. Gebruik de debugger om te bekijken hoe de informatie uit het bestand exact ingelezen wordt. Nuttige methods uit de Scanner klasse – zoek ze zeker op in de API om te weten hoe je ze moet gebruiken – zijn: hasNextLine(), userDelimiter(), next(), nextInt(), skip(), nextLine(),… Indien je getallen moet inlezen, kan je dit ook als “String” doen en dan conversie-methoden uit de wrapper klassen (Integer, Double, …) gebruiken om er echte getalwaarden van te maken (int, double, …).

4. Voeg de mogelijkheid toe om een tekstueel overzicht van alle clubleden te krijgen: showOverview(). Gebruik de informatie in de testklasse om het formaat waarin de uitvoer moet staan terug te vinden (test04, lijn 95 en 98).
5. Om te weten hoe succesvol een club is, willen we weten hoeveel mensen lid werden in een bepaald jaar: joinedInYear(int year) (test05).
6. Kijk na of een persoon met een bepaalde achternaam lid is van de club: isMember(String name).

   Hint: er is een belangrijk verschil tussen test 5 en test 6: vergelijken van primitieve types (int) vs. vergelijken van objecten (String)

7. Verwijder alle leden die toetraden in een bepaald jaar. De functie heeft als returnwaarde een collection (ArrayList van Member objecten) van de verwijderde leden: removeMembersFromYear(int year)

   Hint: denk eraan om een iterator te gebruiken wanneer je elementen uit een collection verwijdert, anders: ConcurrentModificationException.

8. Druk een overzicht van alle **familienamen** van leden, alfabetisch gesorteerd. Je mag gebruik maken van bestaande sorteermethodes (bv. uit java.util.Collections of java.util.ArrayList): showSortedNames() .

   Hint: denk aan de import-statements wanneer je bestaande klassen en methodes gebruikt; zoek in de API op hoe je een bepaalde methode moet gebruiken.

9. Implementeer showOverviewFull(): deze print de lijst van clubleden zoals bij showOverview() maar nu met de naam van de maand van aansluiting in het Engels in plaats van het nummer van de maand (bijvoorbeeld “May” in plaats van 5); het attribuut “month” blijft van het type int.

   Hint: gebruik een static array met de namen van de maanden.

10. Elk clublid moet jaarlijks lidgeld betalen (tweede parameter in de constructor van Club – bekijk test 10: The red Devils betalen €300.00). Een lid krijgt 10% korting voor elk jaar dat hij of zij reeds lid is: assignContributions() berekent het correcte te betalen bedrag en stelt het in voor elk clublid. Voeg een setter en getter toe voor contribution **in de Member klasse.**

    Hint: gebruik java.time.LocalDate om het huidige jaar te vinden in plaats van dit hard te coderen; anders zal je code na nieuwjaar niet meer werken…

11. Voeg de mogelijkheid toe om te weten of een lid zijn of haar lidmaatschap al betaalde: voeg een status veld en een methode getPayStatus() toe in de Member-klasse. Voeg de mogelijkheid toe om aan te geven of een lid betaald heeft: setPayStatus(boolean newStatus) en om een nieuwe betaling uit te voeren met pay(double). Wanneer het verschuldigde bedrag volledig betaald is, wordt de status op true gezet.
12. Maak een lijst van alle e-mailadressen van leden die nog niet betaalden voor het huidige jaar: createEmailListOfNotPaid().

    Hint: je moet wellicht assignContributions aanpassen om een correcte payStatus in te stellen voor “long-time members”, die geen lidgeld meer moeten betalen.

13. Elk kwartaal ontvangt de club een bestand van de bank, met de door de spelers betaalde contributie. In dit bestand staan voornaam, achternaam (de combinatie van beiden is uniek in deze

toepassing, zie addMember()), datum van betaling (formaat: "yyyy-mm-dd"), betaald bedrag (double). De gegevens zijn gescheiden door een ";". Implementeer een functie die dit bestand inleest en verwerkt: processContributionsFromFile(String filename)

Hint1: het inlezen van niet-gehele getallen uit een tekstbestand is altijd een beetje "tricky": niet overal wordt bijvoorbeeld hetzelfde decimale scheidingsteken gebruikt ("," vs. "."). In contributions.txt wordt een "." gebruikt. De Scanner class gebruikt standaard de "regional settings" van je operating system (opgeslagen in een Locale object); voor be-nl is het standaard decimale teken ",". Je kan de te gebruiken standaard (of Locale) voor het inlezen indien nodig omzetten naar een locale waar wél een "." als scheidingsteken gebruikt wordt, zoals bv. Locale.ENGLISH. De scannerklasse heeft hiervoor de method "useLocale(Locale)".

Hint2: als je na enige tijd zoeken de juiste oplossing voor deze test niet gevonden hebt, kan je dit beter even laten rusten en aan de volgende test beginnen

14. Overload de constructor van Member met een constructor waarin je geen maand en jaar van toetreden aangeeft, maar de gegevens van "vandaag" voor maand en jaar gebruikt.

    Hint: gebruik java.time.LocalDate en zoek in de API welke method(s) je hiervoor kan gebruiken.

15. Elk jaar wil de club de leden die al het langs lid zijn speciaal in de bloemetjes zetten. Implementeer de functie: longLifeMembers() die een Arraylist van de leden die al het langst lid zijn (jaartal én maand) als resultaat teruggeeft. Als de ledenlijst leeg is, moet heeft deze functie "null" als returnwaarde.

KU LEUVEN

Objectgerichte Softwareontwikkeling 2021-2022

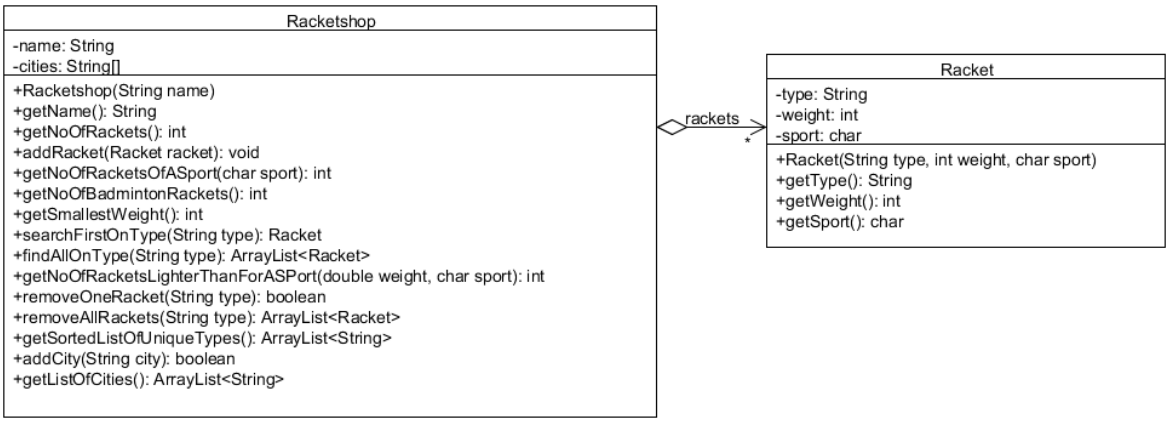
Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Thuisoefeningen

Oefening 1: The Racketshop

We gaan een systeem bouwen dat het mogelijk maakt het aanbod van tennis-, badminton- en padelrackets in een racketshop te beheren. Het systeem zoals je het moet maken, zou in realiteit veel complexer kunnen zijn; dit is slechts een aanzet.

Je kan volgend klassediagram gebruiken:



Een racketshop heeft een naam, is in maximaal 4 steden gevestigd en heeft een aanbod van rackets.

Elk racket heeft een type (bv. “Yonex Z-slash”), een gewicht (in gram) en is voor een bepaalde sport bestemd; dit wordt als volgt aangegeven in een char: ‘B’ voor badminton, ‘P’ voor padel en ‘T’ voor tennis (je zou hiervoor ook een enum-type kunnen gebruiken).

De volledige implementatie van de klasse Racket is gegeven; hier hoef je niets meer aan te wijzigen. De signatuur van de methoden die je moet implementeren in Racketshop kan je terugvinden in het UML-diagram. Zoals je kan zien wordt voor de lijst van locaties een array gebruikt en geen ArrayList.

Je programma moet het volgende kunnen:

1. Racketshop aanmaken, met een naam (test01).
2. Rackets toevoegen aan een racketshop (test02); enkel rackets van de voorziene types zijn toegelaten (test02bis).
3. Het aantal rackets in stock voor een bepaalde sport tellen (test03).
4. Het aantal rackets in stock voor badminton tellen (test04).
5. Het kleinste gewicht van alle rackets in stock opzoeken (test05).
6. Een racket met een bepaald type opzoeken (als een type meerdere keren voorkomt moet het eerste voorkomen gebruikt worden) (test06).
7. Alle rackets van een bepaald type opzoeken en deze in een ArrayList teruggeven; deze rackets moeten niet uit de shop gewist worden (test07).
8. Aantal rackets in stock lichter dan een gegeven gewicht en geschikt voor een gegeven sport berekenen (test08).
9. Verwijderen van een racket van een bepaald type uit de shop (= simulatie van een verkoop) (test09).

10. Verwijderen van alle rackets van een bepaald type uit de shop; deze rackets worden in een aparte ArrayList als returnwaarde gegeven (test10).
11. Alfabetische gesorteerde lijst van unieke types aanmaken (test11).
12. Namen voor locaties (steden) toevoegen aan een shop. Hierbij mag elke stad maar één keer voorkomen en kunnen er maximaal 4 locaties toegevoegd worden. Als een stad kon toegevoegd worden is de returnwaarde van de functie “true”, anders “false” (test12).
13. Een lijst van de namen van de locaties van een shop maken (test13).