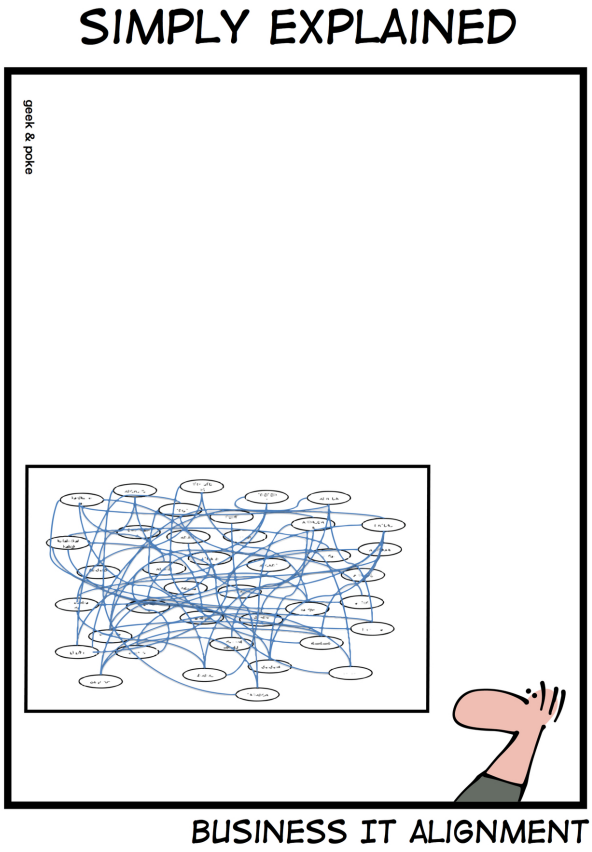
Universiteit van Amsterdam
Inleiding Programmeren

Opgave 6

Stephen Swatman - Jordy Perlee

Hotel



Deadline: zaterdag 12 oktober 2019, 18:00

1 Leerdoelen

Aan het einde van deze opdracht kun je:

- een ingewikkelder programma bouwen op een object-georiënteerde manier,
- simpele datastructuren implementeren,
- gebruik maken van abstracte klassen,
- simpele strings formatteren,
- gebruik maken van overerving,
- klassen implementeren die gebruik maken van generieke types,
- omgaan met polymorfische klassen,
- gebruik maken van constante variabelen,
- op juiste wijze omgaan met foutmeldingen en exceptions.

2 Introductie

Folkert en Florine dachten even snel wat bij te kunnen verdienen. Een internationale hotelketen heeft hen benaderd om een nieuw online receptie-systeem te schrijven voor hotels met een eindig aantal kamers (Hibert's hotel[1] gebruikt vanzelfsprekend een ander systeem). Echter, Florine en Folkert hadden onderschat hoeveel tijd hierin ging zitten, en ze hebben het veels te druk met het nakijken van Inleiding Programmeren opgaves om de software voor aanstaande zondag af te leveren. In eerste instantie hadden ze oud-assistent Stephen om hulp gevraagd, maar die bedankte voor de eer. Als laatste redmiddel stelde Koen voor dat ze hier ook studenten voor mochten inzetten, dus vragen ze nu om jouw hulp.

Folkert en Florine hebben al een hele start gemaakt met het web-gedeelte van dit systeem. Het is nu aan jou om de backend op te zetten dat het mogelijk maakt om een hotel aan te maken, de kamers binnen dit hotel op te zetten en groepen van gasten in- en uit te checken. Om zeker te zijn dat alles onderling goed gaat werken, leveren ze een aantal interfaces aan die jij moet gaan implementeren. Je kunt de interfaces vinden op Canvas.

Waar voorgaande opdrachten vooral gericht waren op het maken van ingewikkelde methoden in één klasse zul je in deze opdracht meerdere klassen moeten maken met relatief simpele methoden en die op een juiste manier koppelen. Vooral van belang is dat de elementen die je gaat schrijven losjes gekoppeld zijn aan elkaar. We letten dan ook op de access modifiers die je gebruikt.

We verwachten dat deze opdracht veel tijd zal kosten en het is dus absoluut aan te raden om *op tijd te beginnen*. Dit geeft je ook meer tijd om vragen te stellen op de werkcolleges.

3 Programmaopbouw

Het programma zal bestaan uit de volgende basisklassen en een aantal kinderen. Deze zijn als volgt:

HotelInterface Beschrijft welke methoden het Hotel moet hebben.

Hotel De meest belangijke klasse van het project omdat deze het hotel representeert. Er zitten kamers in en het handelt de logica af voor het in- en uitchecken.

KamerInterface Een interface die weinig meer doet dan een getter specificeren voor de gasten van de kamer.

Kamer Een enkele kamer in het hotel.

GroepInterface Een interface die een aantal methoden specificeert voor de Groep klasse. Lijkt een beetje op de standaard `ArrayList`.

[1] `https://nrich.maths.org/5788`

Groep Een groep van een generiek type.

Gast Iemand die in het hotel verblijft.

ReceptieInterface Een marker-interface die geen methoden afdwingt. Lekker makkelijk!

Receptie Een gebruikersinterface die een controlemenu biedt door middel van de terminal.

WebReceptie Een receptie die je in de browser toestaat om mensen in- en uit te checken. Deze krijg je van ons.

4 Implementatie

In de implementatie van dit programma zit een hoop vrijheid wat betreft opbouw en structuur. Er zijn dan ook meerdere juiste oplossingen. Belangrijk is dat je goed nadenkt over elke keuze die je maakt en dat je deze met behulp van commentaar onderbouwt.

Hieronder volgt een globaal stappenplan om wat meer houvast te bieden. In dit stappenplan wordt voorgedaan hoe je zo een groot project kunt opdelen in kleinere, minder complexe delen. Deze delen kun je dan gaandeweg aan elkaar knopen om zo tot het volledige programma te komen. Bij elk onderdeel staat globaal uitgelegd wat het moet doen. Er staan ook steeds bepaalde eisen en tips bij. De uitleg zelf is met opzet vaag gehouden zodat je zelf moet gaan nadenken over wat je aan het doen bent.

Van dit stappenplan mag natuurlijk afgeweken worden, maar zorg er wel voor dat alle beschreven klassen en alle beschreven functionaliteit aanwezig is in het door jou ontworpen programma.

4.1 De `Gast` klasse

Het kleinste en simpelste element om mee te beginnen is een gast. Een gast is simpelweg een persoon die in jouw hotel zal gaan verblijven. Maak in **Gast.java** een klasse `Gast`. Een object van de `Gast` klasse bevat niets meer dan een voornaam en achternaam. Geef de `Gast` klasse twee constructoren: eentje die twee strings neemt als voor- en achternaam en eentje die willekeurige namen kiest uit een constante lijst met namen. Vooral die laatste gaat je veel werk besparen bij het testen (dan hoef je namelijk niet zelf steeds namen te tikken). Als je geen inspiratie hebt voor namen, kun je de volgende arrays overnemen en gebruiken. Als je het leuk vindt mag je die zelf ook nog aanvullen.

```
private static final String[] VOORNAMEN = {
    "Adam", "Sofie", "Johan", "Yuri", "Marie", "Fred", "Lisa", "Robin",
    "Claartje", "Freek", "Piet", "Pietje", "Erik", "Bas", "Pavlov", "Igor",
    "Ivan", "Geertje", "Klaas", "Snorri",  "Anna", "Lotte", "Sara", "Roos",
    "Noor", "Thor", "Jean", "Karel", "Dick", "Richard", "Dzjengis", "Emma",
    "Howard Phillips", "Peter", "Sint", "Albert", "Dirk Jan", "Taylor",
    "Linus", "James", "Bjarne", "Jurriaan", "Jan-Klaassen"
};
private static final String[] ACHTERNAMEN = {
    "de Vries", "Smit", "Petersen", "Vonk", "Janssen", "Klaassen",
    "de Trompetter", "Baantjes", "de Jong", "Sanchez", "Bakker", "Eggertsson",
    "Sturluson", "Valjean", "de Grote", "Precies", "Khan", "Snorremans",
    "de Cock met C-O-C-K", "Stallman", "Lovecraft", "Erklaas", "Gagarin",
    "Einstein", "Heijn", "de Geer", "Swift", "Torvalds", "Gosling",
    "Stroustrup"
};
```

Geef de `Gast` klasse een `toString` methode die simpelweg de volledige naam teruggeeft en maak voor jezelf een simpel testprogramma (bijvoorbeeld in de `main` methode van de `Gast` klasse) dat een paar gasten aanmaakt en hun naam print, om zo te kunnen controleren of alles naar behoren werkt.

Eis De voor- en achternaam van een `Gast` moeten private zijn.

Eis De `toString` methode moet de volledige naam returnen, met spatie.

Eis De twee genoemde constructoren moeten aanwezig zijn en de tweede moet willekeurige namen kiezen.

Tip Gebruik constructor chaining.

Tip Besteed niet teveel tijd aan het bedenken van grappige namen ☺.

4.2 De `Groep` klasse

Maak in **Groep.java** een klasse `Groep` aan die zowel de `GroepInterface` implementeert als de `Iterable` interface. Een `Groep` is eigenlijk niets meer dan een verzameling objecten van een bepaald type, net zoals een `ArrayList`. Omdat de `Groep` de `Iterable` interface[2] implementeert kun je er ook for-each loops op toepassen. Zorg ervoor dat de klasse een generiek type neemt om op te slaan. Intern is de `Groep` klasse een array van objecten die in grootte verdubbeld wordt op het moment dat hij vol zit. Maak een constuctor die een initiele groote voor de array neemt en een constructor die de array begint met een enkel element.

Groepen verwijzen veelal naar groepen mensen (gasten) die komen inchecken of die al ingecheckt zijn, maar dat hoeft niet altijd zo te zijn. Door de implementatie generiek te houden, zou je een Groep kunnen gebruiken voor elk willekeurig object. Om deze reden is de groep een generieke klasse, die je niet alleen voor gasten maar ook voor een verzameling kamers gaat gebruiken.

Omdat de `Groep` ook de `Iterable` interface moet implementeren moet er een `iterator` methode geimplementeerd worden die een `Iterator`[3] object returnt. Om jullie hiermee op weg te helpen, krijg je van ons de volgende methode en bijna complete inner klasse (dat wil zeggen, een klasse binnen een object die toegang heeft tot de inhoud van zijn outer klasse) cadeau.

```
public Iterator<T> iterator() {
    return this.new GroepIterator();
}

private class GroepIterator implements Iterator<T> {
    int teller = 0;

    public T next() {
        return get(teller++);
    }

    public boolean hasNext() {
        return teller < [[ het aantal elementen in het Groep object ]];
    }
}
```

Maak wederom een klein testprogramma aan waarin je allerlei verschillende objecten in een `Groep` gaat zetten en ze er weer uit haalt. Test de methoden van de klasse en probeer ook met een for-each loop over jouw `Groep` te lopen.

Eis De `Groep` klasse implementeert zowel de `Iterable<T>` interface als de `GroepInterface<T>`.

Eis De `Groep` klasse neemt een generiek type.

Eis Je implementeert de `Groep` als een array die verdubbelt in grootte als hij vol is. Je gebruikt **geen** `ArrayList`.

Tip Je kunt geen array maken van een generiek type maar wel van het type `Object`.

[2] `https://docs.oracle.com/javase/10/docs/api/java/lang/Iterable.html`
[3] `https://docs.oracle.com/javase/10/docs/api/java/util/Iterator.html`

Tip Als je waarschuwingen krijgt over unchecked casts kun je selectief een decorator boven de methode zetten: `@SuppressWarnings("unchecked")`.

Tip Als je Java expert wil worden, neem dan vooral de documentatie van de `Iterator` interface door maar prioritiseer de opdracht.

4.3 De `VasteGrootteGroep` klasse

Maak nu een `VasteGrootteGroep` subklasse aan van de `Groep` klasse. De `VasteGrootteGroep` kan alles wat een `Groep` kan maar vergroot nooit de innerlijke array. Eenmaal aangemaakt behoudt een `VasteGrootteGroep` dus altijd de grootte waarmee hij begon. Bedenk hoe je dit gaat implementeren.

Schrijf ook een `GroepVolException` exception. Dit moet een unchecked exception zijn wat wil zeggen dat het een `RuntimeException` is. Deze exception gooi je op het moment dat er een element toegevoegd wordt aan een volle `VasteGrootteGroep`.

Eis De `VasteGrootteGroep` klasse is een subklasse van de `Groep` klasse.

Eis De `add` methode van de `VasteGrootteGroep` klasse moet een `GroepVolException` gooien als de groep vol zit.

Eis De `GroepVolException` is een subklasse van `RuntimeException`.

Tip In principe heeft de `VasteGrootteGroep` klasse alleen een constructor nodig en een aangepaste `add` methode.

4.4 De `Kamer` klasse

Om alle groepen die het hotel binnenkomen een slaapruimte te kunnen bieden, heeft ons hotel kamers nodig. Omdat niet alle kamers hetzelfde zijn, zul je gebruik moeten maken van overerving om meerdere soorten kamers te kunnen maken.

Maak in **Kamer.java** een abstracte klasse `Kamer` aan die de `KamerInterface` implementeert en een `VasteGrootteGroep` bevat om de gasten op te slaan. De abstracte `Kamer` geef je een constructor die de grootte van de kamer als argument krijgt. Daarna maak je de verschillende subclasses van kamer namelijk `EenPersoonsKamer`, `TweePersoonsKamer` en `VierPersoonsKamer`. De constructoren van deze subclassen roepen de kamer constructor aan met het juiste aantal kamers maar nemen zelf geen argumenten. De `Kamer` klasse bevat een `Groep` van `Gast`en en de functionaliteit om gasten in kamers te stoppen en uit kamers te halen.

Het is zo dat je de grootte van een kamer normaliter zou implementeren met behulp van instantievariabelen. Dit voorbeeld dient dan ook enkel als basis om eens een keer met abstracte klassen te kunnen werken. Een leuke uitbreiding zou zijn om bijvoorbeeld een subclass Penthouse te maken die, naast de functionaliteit van de normale, nog extra functies heeft om de gasten in te schrijven voor een VIP diner of om de gasten toegang geeft tot het zwembad. Dit hoef je echter niet te doen.

Voeg een methode toe om een `Groep` gasten in te checken en maak checked (dus subklassen van `Exception`[4]) exceptions `KamerAlBezetException` en `KamerTeKleinException` aan die door de `Kamer` worden gegooid als deze respectievelijk vol zit of te klein is voor de groep die daar in wil checken.

Eis De `Kamer` klasse is abstract en heeft de drie genoemde subklassen.

Eis De `Kamer` klasse gooit de twee genoemde checked exceptions.

Eis De gasten in een `Kamer` zijn private en worden opgeslagen in een final `VasteGrootteGroep`.

4.5 De `Hotel` klasse

Om alles samen te voegen maak je in **Hotel.java** een klasse `Hotel` aan die de `HotelInterface` implementeert. Geef het `Hotel` een `VasteGrootteGroep` van `Kamer`s en vul een kwart (afgerond naar beneden)

[4] `http://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html`

van het totale aantal kamers (een argument aan de constructor en standaard 10) met éénpersoonskamers, een kwart (wederom afgerond naar beneden) met vierpersoonskamers en voor de rest met tweepersoonskamers.

Bij het inchecken van een `Groep` gasten controleer je **niet** van tevoren of de groep in de kamer past en dat de kamer leeg is. Je probeert simpelweg in te checken en gaat door naar de volgende kamer als de kamer een exception gooit bij het inchecken. Is er geen enkele kamer beschikbaar, dan gooit het hotel een `GroepPastNietException` die je zelf nog moet implementeren als checked exception.

Je geeft het hotel een naam die bestaat uit een sjabloon waar je de naam van een locatie in kunt vullen, bijvoorbeeld *Grand Budapest Hotel* of *Hotel California*. Dit doe je door uit de volgende gegeven arrays steeds een element te kiezen en met de `format` methode van de `String`[5] klasse. Hoe deze methode werkt, moet je zelf uitvogelen door middel van de documentatie. Wederom mag je natuurlijk je eigen namen toevoegen. De `toString` methode returnt de naam van het hotel.

```
private static final String[] NAAM_STEDEN = {
    "Budapest", "Bucharest", "Chisinau", "Tbilisi", "Munich", "Warsaw",
    "Cluj", "Moscow", "Zagreb", "Prague", "Ljubljana", "California"
};
private static final String[] NAAM_TEMPLATES = {
    "Grand %s Hotel", "Royal %s Hotel", "Hotel %s", "%s Hotel",
    "Hotel of %s"
};
```

Eis De `Hotel` klasse implementeert de `HotelInterface`

Eis De `Hotel` klasse bevat een `VasteGrootteGroep` van `Kamers`.

Eis Het inchecken gebeurt door middel van exceptions zoals hierboven uitgelegd.

Eis De naam van het hotel wordt door middel van `String.format` samengesteld.

Tip De meeste hotels hebben geen kamer nul, maar hier mag dat wel.

4.6 De `Receptie` klasse

Om nu de functionaliteit van jouw hotel volledig voor de gebruiker toegankelijk te maken, maak je een `Receptie` klasse die het mogelijk maakt om het hotel te besturen door middel van commando's. De commando's die ondersteund moeten worden zijn `checkin`, `checkuit`, `status` en `stop`. Hoewel we er in het begin van de cursus erg op gehamerd hebben, is het nu niet belangrijk dat je uitgebreid controleert op foutieve invoer. Je kunt een voorbeelduitvoer vinden van de `Receptie` klasse aan het eind van dit document.

De `Receptie` klasse mag, als je dat handig vindt, slechts bestaan uit een `main` methode. Het aantal kamers in het hotel lees je uit het eerste argument van de command line. Je hoeft geen bibliotheek te gebruiken om dit in te lezen.

4.7 De `WebReceptie` klasse

De WebReceptie klasse is het gevolg van een uit de hand gelopen experiment van Stephen en Folkert. Deze klasse zet lokaal een webserver op, waardoor je via een webpagina het hotel kunt bedienen. Je kunt de klasse vinden op Canvas. Als je WebReceptie start zal er een URL getoond worden. Door deze te openen kom je bij de web interface en kun je gasten inchecken en uitchecken (Zodra je dit hebt geïmplementeerd in jouw eigen code). Afsluiten van de webserver kan met `Ctrl + C`.

Je hoeft verder weinig met deze WebReceptie klasse te doen. Het idee erachter is dat hierdoor duidelijk wordt waar interfaces nuttig voor kunnen zijn, namelijk tussen verschillende teams communiceren over hoe bepaalde klassen zich dienen te gedragen. Als je het fijn vindt om de `WebReceptie` te gebruiken om je programma te testen dan mag dat natuurlijk.

[5] `http://docs.oracle.com/javase/10/docs/api/java/lang/String.html`

Wel is het belangrijk dat jouw functionaliteit dusdanig is geimplementeerd dat de WebReceptie klasse werkt. Dit toont namelijk aan dat je inderdaad de Interfaces en beschrijvingen in de opgave hebt opgevolgd. Je bent ons immers aan het helpen met het opzetten van een incheck systeem voor een hotel, dus als onze code niet goed op elkaar aansluit hebben we er alsnog niks aan ☺.

5 Inleveren

Pak de bestanden in en lever ze in op Canvas.

A Voorbeelduitvoer

```
java Receptie 4
Welkom bij het mainfraim van Hotel of Munich.
Geldige commando's zijn `checkin`, `checkuit`, `status` en `stop`.

Voer een commando in: status
Kamer 0 (maximaal 1 gast)
    Leeg
Kamer 1 (maximaal 2 gasten)
    Leeg
Kamer 2 (maximaal 2 gasten)
    Leeg
Kamer 3 (maximaal 4 gasten)
    Leeg
Voer een commando in: checkin
Hoeveel gasten? 2
Voer een commando in: checkin
Hoeveel gasten? 3
Voer een commando in: checkin
Hoeveel gasten? 2
Voer een commando in: checkin
Hoeveel gasten? 2
Deze groep past helaas niet in het hotel.
Voer een commando in: checkin
Hoeveel gasten? 1
Voer een commando in: checkin
Hoeveel gasten? 1
Deze groep past helaas niet in het hotel.
Voer een commando in: status
Kamer 0 (maximaal 1 gast)
    Sara Vonk
Kamer 1 (maximaal 2 gasten)
    Lotte de Trompetter
    Yuri Precies
Kamer 2 (maximaal 2 gasten)
    Sara Eggertsson
    Howard Phillips Gosling
Kamer 3 (maximaal 4 gasten)
    Lotte Sturluson
    Roos Torvalds
    Lisa Stallman
Voer een commando in: checkuit
Welk kamernummer? 1
Voer een commando in: status
Kamer 0 (maximaal 1 gast)
    Sara Vonk
Kamer 1 (maximaal 2 gasten)
    Leeg
Kamer 2 (maximaal 2 gasten)
    Sara Eggertsson
    Howard Phillips Gosling
Kamer 3 (maximaal 4 gasten)
    Lotte Sturluson
    Roos Torvalds
    Lisa Stallman
Voer een commando in: stop
```