UNIVERSITEIT GENT

Taak 1: het collections framework

Algoritmen en Datastructuren I

Academiejaar 2010-2011

1 Inleiding

Deze eerste taak bestaat uit een aantal experimenten met het Java Collections Framework. Er wordt gevraagd een aantal algoritmen te implementeren en een aantal tests uit te voeren. Hierover schrijf je ook een **verslag van 5 à 10 bladzijden** dat je samen met je code indient. Voor je verslag gebruik je de structuur van de **template** die je vindt op Minerva (`TemplateVerslag.odt`). Het is echter niet verplicht Open Office te gebruiken. Je kan dus gerust dezelfde structuur overnemen in LaTeX of in een andere tekstverwerker.

De data voor alle experimenten worden gegenereerd door de gegeven klasse `RandomPersonGenerator`. Het is *niet* de bedoeling dat je deze klasse aanpast. De statische methode `getInstance()` geeft een object van deze klasse terug waarna je met behulp van de methode `getRandomPerson()` random personen kan genereren. De data worden door deze methode als een array van 6 Strings teruggegeven in deze volgorde:

```
[ID, voornaam, naam, telefoonnummer, gemeente, leeftijd]
```

Omdat deze ruwe voorstelling niet voor elke vraag geschikt is om verder te gebruiken zal je zelf één of meerdere klassen moeten schrijven om een persoon voor te stellen. Merk ook op dat de random generator standaard steeds een andere reeks random personen zal geven. Indien je voor testdoeleinden wenst dat steeds dezelfde reeks personen geretourneerd wordt, kan je gebruik maken van de `setSeed` methode. Als parameter gebruik je een willekeurig geheel getal (de *seed*). Deze methode zorgt ervoor dat de random generator steeds dezelfde reeks personen zal retourneren. Uiteraard zal je voor al je tests eerst een aantal random personen moeten genereren, maar het is **niet** de bedoeling dat je het genereren van de random personen meeneemt in je tijdsmetingen en complexiteitsbesprekingen.

2 Opgave

Vraag 1: Gegevens verwijderen

Voor deze oefening is het de bedoeling een vergelijking te maken van de **verwijderbewerking** bij een `LinkedList` en een `ArrayList`. Hiervoor ontwerp en implementeer je een algoritme dat alle inwoners van een gegeven gemeente uit de databank verwijdert. Je schrijft twee versies: een die de gegevens opslaat in een `LinkedList` en een die de gegevens opslaat in een `ArrayList`. De oorspronkelijke volgorde van de data moet behouden blijven. Voor de tijdscomplexiteit en tijdsmetingen zijn we hier enkel geïnteresseerd in de verwijderbewerkingen.

Implementatie

- Schrijf twee implementaties `ArrayListPlaceFilter` en `LinkedListPlaceFilter` van de interface `PlaceFilter`, die resp. de data opslaan in een `ArrayList` en in een `LinkedList`.
- De methode `removeInhabitants` heeft als returntype `Collection<String>`. Hier retourneer je een collectie met enkel de ID 's van de verwijderde personen.

Verslag

- Beschrijf **duidelijk** op welke manier je de beide lijsten gebruikt. Motiveer je keuzes.
- Bespreek de tijdscomplexiteit van de verwijderbewerking bij je beide versies.
- Voer tijdsmetingen uit op de beide verwijderbewerkingen en geef de resultaten weer in duidelijke grafieken met gelabelde assen.
- Bespreek je tijdsmetingen en verklaar je bevindingen.
- Welke van beide datastructuren is volgens jou het meest geschikt?

Vraag 2: Frequenties van initialen

Voor de tweede vraag zullen we enkel gebruik maken van de voor- en familienaam van de gegenereerde personen. We willen namelijk onderzoek doen

naar de initialen van personen en meer bepaald naar het voorkomen ervan. De initialen van een persoon zijn de combinatie van de eerste letter van elk woord van zijn voornaam gevolgd door de eerste letter van elk woord van zijn familienaam. De initialen van *Arthur Dent* zijn bijvoorbeeld *AD* en van *Jean-Claude Van Damme*, *JVD*. Het is de bedoeling twee varianten te ontwerpen voor een algoritme om te tellen hoe vaak elke combinatie van initialen voorkomt. De eerste variant moet gebaseerd zijn op een datastructuur uit het Java Collections Framework die de `List` interface implementeert. De tweede variant van het algoritme moet gebaseerd zijn op de `HashMap` datastructuur.

Implementatie

- Je variant die gebruik maakt van een `List` komt in een klasse `ListInitialsCounter`. Je variant met `HashMap` komt in een klasse `HashMapInitialsCounter`. Beide klassen implementeren de interface `InitialsCounter`.
- De methode `getNrOccurrences` moet zo efficiënt mogelijk zijn. Voer hier dus geen tijdrovende bewerkingen uit die in de methode `addPerson` ook al hadden kunnen gebeuren.

Verslag

- Beschrijf **duidelijk** op welke manier je de datastructuren gebruikt. Motiveer je keuzes. Merk ook op dat je voor de eerste variant de keuze hebt uit verschillende datastructuren die de interface `List` implementeren. Welke heb je hier gekozen en waarom?
- Bespreek de tijdscomplexiteit van de twee methoden van beide versies.
- Voer tijdsmetingen uit en geef de resultaten weer in duidelijke grafieken met gelabelde assen.
- Bespreek je tijdsmetingen en verklaar je bevindingen.
- Welke implementatie is volgens jou de beste?

Vraag 3: Telefoonboek

In deze vraag is het de bedoeling om een telefoonboek te implementeren. Je dient zelf te beslissen welke voorstelling je intern gebruikt om het boek en de

personen voor te stellen. Een persoon wordt net zoals in een echt telefoonboek opgezocht aan de hand van zijn gemeente, familienaam en voornaam. Een persoon kan meerdere telefoonnummers hebben in het telefoonboek!

Implementatie

- Schrijf een klasse `MyPhoneBook` die de interface `PhoneBook` implementeert.
  - Schrijf een methode `void add(String[] persoon)` die een persoon toevoegt aan het telefoonboek.
  - Schrijf een methode `void addPhoneNumber(String place, String lastName, String firstName, String phoneNumber)` die een extra telefoonnummer toevoegt voor een persoon die reeds in het telefoonboek voorkomt.
  - Schrijf een methode `Collection<String> getNumbers(String place, String lastName, String firstName)` die een collectie retourneert met alle telefoonnummers voor de gegeven persoon.
  - Het moet ook eenvoudig zijn om je klasse in boekvorm om te zetten. Dit wil zeggen dat er per gemeente een alfabetische namenlijst (eerst op familienaam, dan op voornaam) met alle telefoonnummers afgedrukt wordt. Ook de gemeenten staan uiteraard in alfabetische volgorde. Noem deze methode `void write()`.

    Deze methode schrijft enkel lijnen uit van de vorm (op één lijn):
    `<Gemeente> <spatie> <Familienaam> <spatie> <Voornaam> <spatie> <Telefoonnummer 1> <spatie> <Telefoonnummer 2>` enz.
    Hou je **zeer strikt** aan deze specificatie. Schrijf verder geen enkele extra lijn, scheidingsteken of spatie uit!

Verslag

- Beschrijf uitvoerig je datastructuur en de manier waarop je die gebruikt. Motiveer de gemaakte keuzes.
- Bespreek de tijdscomplexiteiten van de geïmplementeerde bewerkingen.

Vraag 4: Oudste inwoners

Met het oog op het organiseren van feesten voor eeuwelingen e.d. wil men er bij de gemeenten zicht op hebben wie de oudste inwoners zijn. Er wordt gevraagd om een algoritme te ontwerpen dat voor een gegeven getal n en een gegeven gemeente, de n oudste personen in deze gemeente bepaalt. Bij ex aequo geef je de betreffende inwoners in alfabetische volgorde (eerst op familienaam, dan op voornaam). Indien de gemeente minder inwoners telt dan het gevraagde aantal, retourneer je alle inwoners van de gemeente, in de gevraagde volgorde.

Implementatie

- Schrijf een klasse `MyOldestPeopleFinder` die de interface `OldestPeopleFinder` implementeert.
- Je algoritme komt in de methode `findOldestPeople`. Ook hier geldt dat deze methode zo snel mogelijk moet zijn. Voer hier dus geen tijdrovende bewerkingen uit die in de methode `addPerson` ook al hadden kunnen gebeuren.
- Het returntype van de methode is `Collection<String>`. Hier retourneer je een collectie met de ID's van de geselecteerde personen. Deze moeten echter wel degelijk gesorteerd zijn volgens leeftijd: oudste eerst, en vervolgens alfabetisch gesorteerd op familienaam en vervolgens op voornaam.

Verslag

- Beschrijf uitvoerig je datastructuur en de manier waarop je die gebruikt. Motiveer de gemaakte keuzes.
- Bespreek de tijdscomplexiteit.
- Voer tijdsmetingen uit en geef de resultaten weer in duidelijke grafieken met gelabelde assen.
- Bespreek je tijdsmetingen en verklaar je bevindingen. Komen je tijdsmetingen overeen met je theoretische tijdscomplexiteit?

3 Nog enkele algemene opmerkingen

- Je verslag is een zeer belangrijk onderdeel van dit project. Besteed hier dan ook voldoende aandacht aan. Label steeds duidelijk je grafieken en lees je verslag grondig na voordat je het indient. Natuurlijk kan een typfoutje iedereen overkomen, maar 20 typfouten op één blad is uiteraard te veel van het goede.
- Test zeer grondig of je implementaties wel **correct** zijn voordat je met tijdsmetingen begint. Het heeft geen enkele zin tijdsmetingen te doen op een implementatie die foute resultaten geeft.
- Wees niet bang om je computer lange tijd te laten rekenen. Uit tijdsmetingen in een grootteorde van 16 à 31 ms kan je **niets** besluiten.
- Plaats je code **niet in een package-structuur**.
- Alle gevraagde klassen moeten een **default-constructor** (zonder parameters) hebben. Heb je daarnaast zelf nog klassen geschreven waarvoor de naam niet door ons werd opgegeven, dan mag je in deze klassen uiteraard wel een constructor naar keuze gebruiken.
- Op Minerva vind je een klasse `ConstructorTest`. Deze klasse heeft als enig doel na te gaan of alle gevraagde klassen een default-constructor hebben en de gegeven interfaces implementeren. Op Indianio wordt deze klasse automatisch mee gecompileerd.
- Aan de bijgeleverde klassen en interfaces verander je niets.
- Voorzie je code van voldoende duidelijke **commentaar**.
- Zeer belangrijk: **schrijf niets naar het scherm dat niet gevraagd is**, ook geen debuginformatie of foutmeldingen, niet naar stdout en niet naar stderr.

4 Indienen

Om in te dienen bundel je je verslag en code in een zip bestand. Dit bestand bevat **enkel** een bestand `verslag.pdf` en je zelfgeschreven javabestanden. Gecompileerde code en de door ons geleverde klasse en interfaces dien je **niet** mee in. We verwachten dat je dit bestand indient voor **donderdag 17 maart 2011 om 14u30** via `http://indianio.ugent.be` bij het project: "Algoritmen en Datstructuren I: Taak 1".

Hiernaast dient er ook een **papieren versie** van het verslag ingediend te worden. Dit kan tot de deadline in bureau 40.09.110.016 (Bart) of 40.09.110.032 (Stéphanie) of ten laatste in het practicum van donderdag 17 maart 2011.

5 Belangrijke opmerking

Het project is **individueel** en dient dus door jou persoonlijk gemaakt te worden. Het is uiteraard toegestaan om andere studenten te helpen of om ideeën uit te wisselen maar het is ten strengste verboden code uit te wisselen, op welke manier dan ook. Het overnemen van code beschouwen we als fraude van beide betrokken partijen en zal in overeenstemming met het examenreglement behandeld worden.