TENTAMEN

Programmeren 1

vakcode: 213500
datum: 10 juli 2004
tijd: 9:00-12:30 uur

VOORBEELDUITWERKING

Algemeen

- Bij dit tentamen mag gebruik worden gemaakt van het boek van Niño/Hosch, en van de handleiding van Programmeren 1.
- Dit tentamen bestaat uit 4 opgaven, waarvoor in het totaal 100 punten behaald kunnen worden. Het minimale aantal punten per opgave bedraagt 0.
- Bij de opdrachten in dit tentamen hoeven *geen* pre- en postcondities te worden gegeven, tenzij *expliciet* anders vermeld. Neem wel commentaar op waar dat nuttig is voor het begrijpen van uw oplossing.

Opgave 1 (15 punten)

Een database van personen slaat voor elke persson ook de *huwelijkse staat* op; mogelijke waarden zijn in ieder geval "gehuwd" en "ongehuwd". Bespreek tenminste vier verschillende datatypen die voor de representatie van dit gegeven gebruikt zouden kunnen worden, op de volgende manier:

- Geef het type en de representatie van de verschillende (mogelijke) waarden, waaronder in ieder geval de hierboven genoemde.
- Bedenk tenminste drie criteria waaraan uw representatie zou moeten voldoen, en beoordeel elk van uw representatiekeuzes volgens elk criterium, met uitleg (plm. 1 regel). Een voorbeeldcriterium is: *uitbreidbaarheid* (d.w.z., kunnen er nieuwe mogelijke waarden worden gedefinieerd).

Antwoord op Opgave 1

Bij deze opgave gaat het om reflectie op de keus van een datatype: wat zijn de mogelijkheden, en wat de voordelen? Enig inzicht in de implementatiekenmerken van de diverse datatypen is vereist.

Beoordeling: Per type 0–4 punten; -2 per ontbrekend of dubbel beoordelingscriterium, -1 indien mogelijke waarden niet of fout gegeven. Alleen punten geven voor werkelijk verschillende typen, dus niet twee referentietypen of `byte` enerzijds en `int` anderzijds.

Een aantal criteria volgen hieronder.

Uitbreidbaarheid: zoals in de opgave genoemd;

Efficiëntie van de opslag: de hoeveelheid geheugen die voor het opslaan van een waarde nodig is;

Efficiëntie van de executie: de hoeveelheid tijd die voor het rekenen met een waarde nodig is;

Efficiëntie in het gebruik: de hoeveelheid code er nodig is om het type te definiëren en te gebruiken;

Begrijpelijkheid: de mate waarin de waarden van het type op een direct begrijpelijke manier worden gerepresenteerd;

Flexibiliteit: of het mogelijk is speciale operaties te definiëren;

Type-checking: of de Java-typechecker helpt bij het herkennen van geldige en ongeldige waarden;

Een aantal representaties. Niet precies deze argumenten hoeven gebruikt te worden, maar er moet wel een toelichting gegeven worden! Alleen "goed" of "slecht" is niet voldoende.

- Als `boolean`.

  Mogelijke waarden: `true` en `false`, bijv. voor resp. gehuwd en ongehuwd.

  Uitbreidbaarheid: Slecht, meer dan 2 waarden kan niet.

  Efficiëntie van de opslag: Goed, een `boolean` gebruikt in principe maat 1 bit. [De implementatie in de JVM is hierin helaas inefficiënt en gebruikt uiteindelijk 4 bytes, maar dat weet bijna geen mans.]

  Efficiëntie van de executie: Goed, met `boolean`s wordt uitermate snel gerekend.

  Efficiëntie in het gebruik: Goed, `boolean` is een primitief type en wordt goed ondersteund.

  Begrijpelijkheid: Redelijk, we hoeven allen te weten waar `true` voor staat — gehuwd ligt daar het meest voor de hand.

  Flexibiliteit: Slecht, er is geen mogelijkheid eigen operaties te definiëren.

  Type-checking: Goed, er zijn 2 mogelijke waarden en die worden door de type-checker herkend.

- Als `int` of ander getaltype.

  Mogelijke waarden: Van tevoren gekozen coderingen, bijv. 0 voor ongehuwd, 1 voor gehuwd.

  Uitbreidbaarheid: Goed, het aantal `int`s is ruim voldoende

  Efficiëntie van de opslag: kan beter, een `int` gebruikt 4 bytes. (Andere getaltypen gberuiken minder of meer.)

  Efficiëntie van de executie: Goed, met `int`s wordt zeer snel gerekend.

  Efficiëntie in het gebruik: Goed, `int` is een primitief type en wordt goed ondersteund.

  Begrijpelijkheid: Matig, de keuze van de codering moet onthouden worden. Constantendefinities kunnen helpen dit te verbeteren.

  Flexibiliteit: Slecht, er is geen mogelijkheid eigen operaties te definiëren.

  Type-checking: Matig, de type-checker heeft geen benul van geldige en ongeldige waarden.

- Als `char`.

  Mogelijke waarden. Van tevoren gekozen coderingen, bijv. `'o'` voor ongehuwd, `'g'` voor gehuwd.

  Uitbreidbaarheid: Matig, aangezien de latters opraken; bijv. "gescheiden" zou ook `'g'` moeten zijn maar dat gaat niet meer.

  Efficiëntie van de opslag: Goed, een `char` neemt 2 bytes in beslag.

  Efficiëntie van de executie: Goed, met `char`s wordt zeer snel gerekend.

  Efficiëntie in het gebruik: Goed, `char` is een primitief type en wordt goed ondersteund.

  Begrijpelijkheid: Iets beter dan `int` omdat de beginletter gekozen kan worden (maar zie "uitbreidbaarheid")

  Flexibiliteit: Slecht, er is geen mogelijkheid eigen operaties te definiëren.

  Type-checking: Matig, de type-checker heeft geen benul van geldige en ongeldige waarden.

- Als `String`.

  Mogelijke waarden: De volledige omschrijving, bijv. `"gehuwd"`, `öngehuwd"`.

  Uitbreidbaarheid: Prima, nieuwe waarden kunnen zonder meer toegevoegd worden.

  Efficiëntie van de opslag: Slecht, een `String` is een referentietype en heeft veel ruimte nogid (behalve als er geshared wordt).

  Efficiëntie van de executie: Matig, alle operaties gaan d.w.v. methoden-aanroepen.

  Efficiëntie in het gebruik: Redelijk, `String` wordt voor een referentietype goed ondersteund (bijv. constantendefinitie, concatenatie).

Begrijpelijkheid: Prima, de string kan de waarde geheel uitdrukken

Flexibiliteit: Slecht, er is geen mogelijkheid eigen operaties te definiëren.

Type-checking: Matig, de type-checker heeft geen benul van geldige en ongeldige waarden.

- Als zelfgedefinieerde klasse, bijv. `HuwelijkseStaat`

Mogelijke waarden. De interne implementatie staat vrij, bijv. een van de bovengenoemde mogelijkheden. Voordeel is dat de keus van de gebruiker afgeschermd is en eventueel gewijzigd kan worden.

Uitbreidbaarheid: Goed, door eventuele keuze andere implementatie (zie boven)

Efficiëntie van de opslag: Slecht, elke waarde is een object met een referentie er naar toe. Minder slecht als objecten geshared worden.

Efficiëntie van de executie: Slecht, creëren van waarden vergt geheugen-allocatie en gebruik ervan methodenaanroepen.

Efficiëntie in het gebruik: Matig, het definiëren en creëren van waarden is betrekkelijk omslachtig.

Begrijpelijkheid:

Flexibiliteit: Goed, nieuwe operaties kunnen zelf gedefinieerd worden.

Type-checking: Goed, er kunnen alleen zelf geconstrueerde waarden in omloop zijn.

- Als zelfgedefinieerd interface, bijv. `HuwelijkseStaat`

Mogelijke waarden. Per gewenste waarde een implementerende klasse; bijv. `Gehuwd`, `Ongehuwd`.

Uitbreidbaarheid: Uitstekend, door definiëren nieuwe implementerende klassen.

Efficiëntie van de opslag: Slecht, elke waarde is een object met een referentie er naar toe. Minder slecht als objecten geshared worden.

Efficiëntie van de executie: Slecht, creëren van waarden vergt geheugen-allocatie en gebruik ervan methodenaanroepen.

Efficiëntie in het gebruik: Slecht, het definiëren en creëren van (nieuwe) waarden is omslachtig.

Begrijpelijkheid: Uitstekend, elke implementerende klasse kan een eigen, begrijpelijke naam krijgen.

Flexibiliteit: Uitstekend, door gebruik te maken van nieuwe methodendefinities, incl. overerving en overschrijven.

Type-checking: Uitstekend, de type-ckecker kan alle legale en niet-legale waarden uit elkaar halen.

Opgave 2 (25 punten)

In deze opgave gaat het om een `static` methode `index(List elementen, Object gezocht)`, die de index in `elementen` van `gezocht` oplevert, of `-1` als `gezocht` niet in `elementen` voorkomt. (Van de klasse `List` heeft u uitsluitend de methoden `Object get(int index)` en `int size()` nodeig¿)

a. *15 punten.* Geef een implementatie van de methode, met postconditie en een lusinvariant; maak aannemelijk dat de lusinvariant correct is en dat de postconditie gegarandeerd is.

b. *5 punten.* Stel dat de lijst n elementen bevat. Wat is het gemiddelde aantal stappen (= aantal keren dat de lus doorlopen wordt) die uw methode `index` nodig heeft om de index van een object te bepalen dat in de lijst aanwezig is? Kunt u een met kleiner aantal stappen toe als u weet dat de elementen in de lijst geordend zijn — bijvoorbeeld, als u weet dat de lijst uit geordende `String`s bestaat en `gezocht` ook een `String` is? Zo ja, hoe dan (in woorden omschreven), en wat is het aantal stappen? Zo nee, waarom niet?

c. *5 punten.* Kunt u van een willekeurig `Object` bepalen of het “kleiner is dan” een ander `Object`? Zo ja, wat betekent dit “kleiner dan” zijn? Zo nee, wat is er dan nodig (d.w.z., wat moet u van de `Object`en weten) om dit wél te kunnen bepalen?

Antwoord op Opgave 2

a. *Beoordeling: max.* 5 *punten indien invariant en postconditie afwezig of volledig incorrect.* -5 *als het onderscheid tussen het antwoord* `-1` *en positieve resultaatwaarden niet gemaakt is. Max.* 8 *aks de toelichting geheel ontbreekt.*

De methode, met lusinvariant en postconditie:

```
/**
 * @ensure result >= 0 && elementen.get(result).equals(gezocht) of
 *   result == -1 && voor alle 0 <= i < elementen.length:
 *   !elementen.get(i).equals(gezocht)
 */
static private int index(List elementen, String gezocht) {
    int result = -1;
    int zoek = 0;
    while (result < 0 && zoek < elementen.size()) {
        // zoek >= 0 &&
        // result >= 0 && elementen.get(result).equals(gezocht) of
        //     result == -1 && voor alle 0 <= i < zoek:
        //     !elementen.get(i).equals(gezocht)
        if (elementen.get(zoek).equals(gezocht)) {
            result = zoek;
        }
        zoek++;
    }
    return result;
}
```

De invariant is correct omdat na uitvoering van het `if`-statement geldt dat óf `result == zoek` en `elementen.get(result).equals(gezocht)`, óf `result < 0` en `!elementen.get(zoek).equals(gezocht)`. In het laatste geval geldt de voorwaarde dus nu voor alle `0 <= i <= zoek`. Vervolgens wordt `zoek` opgehoogd en geldt de invariant weer in volle glorie.

De invariant impliceert de posticonditie aangezien de lus pas verlaten wordt als `zoek == elementen.length`; wanneer we dat in de invariant substitueren krijgen we precies de postconditie.

b. *5 punten; max.* -3 *voor fouten in de eerste complexiteitsmaat,* -3 *voor de tweede — waarbij het al heel wat is als er een logarithme wordt genoemd. Merk op dat het antwoord in het boek te vinden is.* Het gemiddelde aantal stappen is $n/2$, want het zoeken houdt op zodra het element gevonden wordt. Als de lijst geordend is kan men *binair* zoeken: begin middenin, zoek in de linkerhelft verder als het gezochte element kleiner is dan het gevonden, en in de rechterhelft als het groter is. Het gemiddelde aantal stappen is nu $\log_2 n$; dit ligt aan het feit dat de resterende zoekruimte bij elke stap gehalveerd wordt.

c. *5 punten;* -3 *als het antwoord goed is maar de uitleg onvoldoende.* Nee, dit kan niet: er is geen basis voor de vergelijking van willekeurige objecten. Om objecten te kunnen vergelijken is een methode nodig die de volgorde bepaalt; dit kan het beste bereikt worden door een interface te definiëren dat de aanwezigheid van zo'n methode afdwingt. Een voorbeeld is de in Java voorhanden interface `Comparable`, met de methode `compareTo`, maar men kan ook een eigen interface met vergelijkbare functionaliteit definiëren.

(Een ander mogelijk antwoord is: *ja*, je kan willekeurige objecten ordenen, nl. met behulp van de `hashCode()`. Dit zullen weinig studenten weten, maar het is wel correct.)

Opgave 3 (50 punten)

U schrijft een programma dat, gebruik makend van beschikbare web services, de snelste verbinding tussen twee steden bepaalt. Alle web services zijn als Java-klassen beschikbaar die de hieronder gedefinieerde

interface `Vervoer` implementeren.

```
public interface Vervoer {
    /**
     * Levert de benodigde tijd, in minuten, om van een
     * gegeven plaats naar een andere te komen. Levert -1 als
     * geen verbinding tussen de gegeven plaatsen gevonden wordt.
     * @param van de beoogde vertrekplaats
     * @param naar de beoogde bestemming
     */
    public int tijd(String van, String naar);
}
```

a. Schrijf een klasse `VervoerMatrix` die `Vervoer` implementeert op basis van twee intern bijgehouden instantievariabelen, met de volgende declaraties:

```
private List plaatsen;  // de plaatsen die dit vervoermiddel aandoet
private int[][] tijden; // de tijden op basis van plaatsindex
```

`plaatsen` is de lijst bekende plaatsnamen; `tijden` is een matrix waarin `tijden[i][j]` de tijd tussen `plaatsen.get(i)` en `plaatsen.get(j)` aangeeft. Een voorbeeld, gebaseerd op treintijden:

`plaatsen`

0	Hengelo
1	Deventer
2	Zutphen

`tijden`

	0	1	2
0	0	30	40
1	30	0	20
2	40	20	0

Hieruit blijkt dat het sneller is direct van Hengelo naar Zutphen te reizen dan via Deventer; en ook dat Boekelo geen treinstation heeft. In dit voorbeeld verwachten we de volgende antwoorden:

- `tijd("Hengelo","Zutphen")` levert `40` op;
- `tijd("Hengelo","Boekelo")` levert `-1` op;
- `tijd("Boekelo","Boekelo")` levert `0` op.

Gebruik in uw implementatie de methode `index` uit Opgave 2.

b. Schrijf een klasse `VervoerKeuze` die intern een `List` van `Vervoer`-objecten bijhoudt, en zelf de interface `Vervoer` implementeert, waarbij de implement van `tijd` de kortste benodigde tijd tussen `van` en `naar` oplevert, volgens de `Vervoer`-objecten in de intern bijgehouden lijst.

c. Schrijf een interface `BetaaldVervoer`, die `Vervoer` uitbreidt met een methode `double kosten(String van, String naar)` die de kosten van een reis tussen `van` en `naar` met dit vervoermiddel oplevert, of `-1` als met dit vervoer zo'n reis niet mogelijk is.

d. Schrijf een klasse `PerMinuurVervoer`, die `BetaaldVervoer` implementeert door twee instantievariabelen bij te houden:

```
private Vervoer vervoer;
private double prijsPerMinuut;
```

`PerMinuutVervoer` dient `tijd` te implementeren door de methode aan `vervoer` door te geven, en `kosten` door de benodigde tijd met de factor `prijsPerMinuut` te vermenigvuldigen. Daarnaast dient `PerMinuutVervoer` een constructor te krijgen waarin `vervoer` van een waarde wordt voorzien, alsmede methoden om `prijsPerMinuut` op te vragen en van waarde te veranderen.

e. Neem aan dat de volgende postcondities gespecificeerd zijn:

- Voor de methode `tijd` in `Vervoer`:

`@ensure result >= -1`

- Voor de methode `kosten` in `BetaaldVervoer`:

  `@ensure result == -1 als tijd(van,naar) == -1; anders result >= 0`

Geef pre- en postcondities voor de constructor en methoden van `PerMinuutVervoer`, en leg uit waarom uw implementaties in `PerMinuutVervoer` de postcondities garanderen.

f. Geef een klassendiagram van de klassen en interfaces in deze opgave, incl. instantievariabelen maar zonder methoden of afhankelijkheden. Neem ook `List` en `Object` als klassen op.

Antwoord op Opgave 3

a. *(8 punten; gemakkelijk verdiend. Beoordeling: -3 als de implementatie van* `vervoer` *niet correct is, tot -4 als het geval dat de* `van` *of* `naar` *niet in de lijst voorkomen niet goed wordt afgehandeld; -3 als .)*

```
public class VervoerMatrix implements Vervoer {
    private List plaatsen;  // de plaatsen die dit vervoermiddel aandoet
    private int[][] tijden; // de tijden op basis van plaatsindex

    /**
     * @ensure result >= 0 && elementen.get(result).equals(gezocht) of
     *   result == -1 && voor alle 0 <= i < elementen.length:
     *   !elementen.get(i).equals(gezocht)
     */
    static private int index(List elementen, String gezocht) {
        int result = -1;
        int zoek = 0;
        while (result < 0 && zoek < elementen.size()) {
            // zoek >= 0 &&
            // result >= 0 && elementen.get(result).equals(gezocht) of
            //     result == -1 && voor alle 0 <= i < zoek:
            //     !elementen.get(i).equals(gezocht)
            if (elementen.get(zoek).equals(gezocht)) {
                result = zoek;
            }
            zoek++;
        }
        return result;
    }

    public int tijd(String van, String naar) {
        int vanIndex = index(plaatsen, van);
        int naarIndex = index(plaatsen, naar);
        if (vanIndex >= 0 && naarIndex >= 0) {
            return tijden[vanIndex][naarIndex];
        } else if (van.equals(naar)) {
            return 0;
        } else {
            return -1;
        }
    }
}
```

b. *9 punten; ook gemakkelijk verdiend. Beoordeling: -3 indien de implementatie van* `Vervoer` *niet correct is, -6 als de type-check en/of de cast niet correct is.*

```
public class VervoerKeuze implements Vervoer {
    private List vervoerList;

    public int tijd(String van, String naar) {
        int result = -1;
        for (int i = 0; i < vervoerList.size(); i++) {
            Vervoer vervoer = (Vervoer) vervoerList.get(i);
            if (result < 0 || result > vervoer.tijd(van, naar)) {
                result = vervoer.tijd(van, naar);
            }
        }
        return result;
    }
}
```

c. *3 punten.*

```
public interface BetaaldVervoer extends Vervoer {
    double kosten(String van, String naar);
}
```

d. *10 punten. Beoordeling:* De klasse kan er als volgt uitzien:

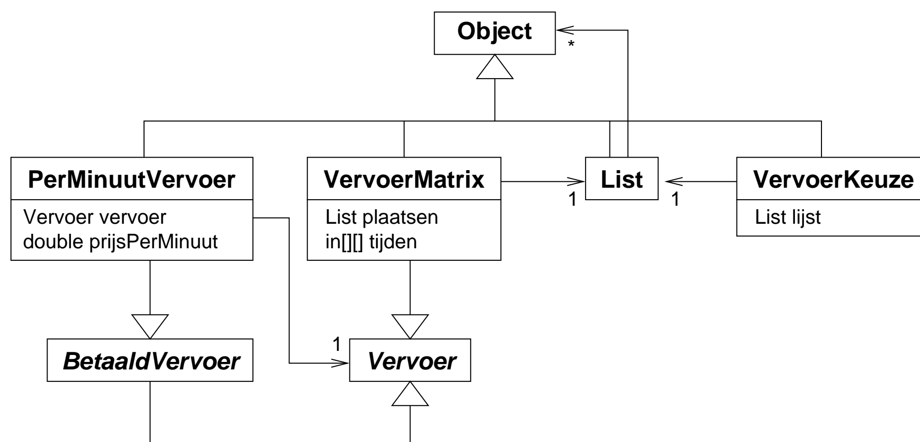
```
public class PerMinuutVervoer implements BetaaldVervoer {
    /** @invariant vervoer != null */
    private Vervoer vervoer;
    /** @invariant prijsPerMinuut >= 0 */
    private double prijsPerMinuut;

    /**
     * @require vervoer != null
     * @ensure getPrijsPerMinuut( ) == 0
     */
    public PerMinuutVervoer(Vervoer vervoer) {
        this.vervoer = vervoer;
    }

    /**
     * @require prijsPerMinuut >= 0
     * @ensure getPrijsPerMinuut( ) == prijsPerMinuut
     */
    public void setPrijsPerMinuut(double prijsPerMinuut) {
        this.prijsPerMinuut = prijsPerMinuut;
    }

    /**
     * @ensure result >= 0
     */
    public double getPrijsPerMinuut() {
        return prijsPerMinuut;
    }

    /**
     * @ensure result >= -1
     */
    public int tijd(String van, String naar) {
        return vervoer.tijd(van, naar);
    }
```

```
    /**
     * @ensure result == -1 als tijd(van,naar) == -1;
     * anders result >= 0
     */
    public double kosten(String van, String naar) {
        int tijd = tijd(van, naar);
        if (tijd < 0) {
            return -1;
        } else {
            return prijsPerMinuut * tijd;
        }
    }
}
```

e. *10 punten. Beoordeling: -5 als er geen klasseninvarianten gebruikt zijn.* De klasseninvarianten en de pre- en postcondities zijn al hierboven aangegeven. Toelichting:

- Bij `vervoer`: De invariant is gegarandeerd door de preconditie van de constructor; dit is de enigeplaats waar aan `vervoer` een waarde toegewezen wordt.
- By `prijsPerMinuut`: De invariant is gegarandeed doordat de standaard beginwarde `0` de conditie vervult, en de preconditie van `setPrijsPerMinuut` hem opnieuw vervult. Dit is de enige plaats waar de variabele van waarde veranderd wordt.
- Bij de constructor `PerMinuutVervoer`: De postconditie is gegarandeerd doordat `0` de standaard-beginwaarde van een `double` variabele is.
- Bij `setPrijsPerMinuut`: De postconditie is gegarandeed door de toewijzing.
- Bij `getPrijsPerMinuut`: De postconditie is gegarandeerd vanwege de klasseninvariant.
- Bij `tijd`: de postconditie is gegarandeerd door de implementatie in de superklasse.
- Bij `kosten`: Als `tijd(van,naar)` een negatief getal oplevert, kan dat alleen `-1` zijn; dat geven we dan door. Anders is `tijd(van,naar)` niet-negatief en `prijsPerMinuut` ook (vanwege de invariant); dus het product van beide ook.

f. *10 punten. Beoordeling: -3 voor elke ontbrekende klasse, -2 voor elke ontbrekende pijl, -1 voor elke ontbrekende variabele.*



Opgave 4 (10 punten)

Bij een matrix zoals in Opgave 3 gebruikt, waarin de getallen de "afstand" aangeven tussen twee dingen[1] hoort het altijd zo te zijn dat de afstand van x naar z hoogstens zo lang is als de som van de afstanden van

[1] in het geval van Opgave 3 is de "afstand" de benodigde tijd en de "dingen" plaatsen

x naar y en van y naar z.

a. *2 punten.* Geef een voorbeeld van een 3×3-matrix waarin deze voorwaarde niet vervuld is.

b. *8 punten.* Schrijf een methode `int[] correct(int[][] a)`, die test of `a` aan deze voorwaarde voldoet; zo ja, dan dient de methode `null` op te leveren, anders een `int`-array met lengte `3`, met daarin de waarden voor x, y, z waarvoor de voorwaarde niet geldt. Geef op basis van uw eigen voorbeeld aan wat de uitkomst van de methode zou moeten zijn.

Antwoord op Opgave 4

a. *2 punten.* Een voorbeeld:

	0	1	2
0	0	10	50
1	10	0	20
2	50	20	0

b. *8 punten.*

```
static public int[] correct(int[][] a) {
    for (int x = 0; x < a.length; x++) {
        for (int y = 0; y < a.length; y++) {
            for (int z = 0; z < a.length; z++) {
                if (a[x][y] + a[y][z] < a[x][z]) {
                    return new int[] { x, y, z };
                }
            }
        }
    }
    return null;
}
```

De vrwachte uitkomst voor bovenstaand voorbeeld is: $[0, 1, 2]$.