TENTAMEN

Programmeren 1

vakcode: 213500
datum: 5 november 2007
tijd: 13:30–17:00 uur

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 90 punten behaald kunnen worden. Het minimale aantal punten per opgave bedraagt 0. Het cijfer is het aantal punten gedeeld door 10, afgerond tot een geheel getal, plus 1.
- 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 (20 punten)

In een agenda-programma is het nodig de mogelijke tijdstippen op een dag tot op de minuut nauwkeurig te kunnen opslaan. Eén manier om tijdstippen te representeren is door middel van niet-negatieve `int`-waarden waarin het totaal aantal minuten vanaf middernacht is opgeslagen. In die representatie is het tijdstip `00:01` (één minuut na middernacht) gerepresenteerd door het getal 1, `12:30` door 750 (nl. $12*60+30$) en `23:59` door 1439 (nl. $23*60+59$).

a. *(6 punten)* Bedenk drie *andere* manieren om zo'n tijdstip te representeren, die allemaal fundamenteel van elkaar en van de hierboven geschetste oplossing verschillen doordat ze op een andere manier gebruik maken van de beschikbare typen in Java. *Let op*: het gebruik van een `int`- of ander getaltype waarin de tijdstippen volgens een andere formule gecodeerd zijn *niet* fundamenteel verschillend van de hierboven geschetste oplossing. Geef voor elk van deze drie manieren aan hoe de tijdstippen `12:30` en `23:59` gerepresenteerd zijn.

b. *(6 punten)* Geef voor elk van de vier oplossingen (dus óók het hierboven geschetste voorbeeld!) een methode `getMinuten` die, gegeven een tijdstip in het bedachte type, het "minuten"-deel als `int`-waarde oplevert (dus 30 bij `12:30` en 59 bij `23:59`).

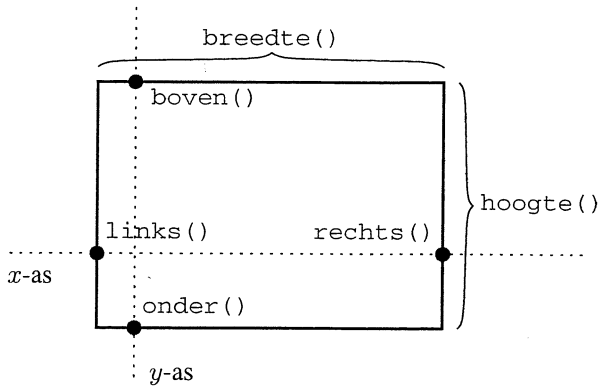
c. *(8 punten)* Geef voor elk van de vier oplossingen (dus óók het hierboven geschetste voorbeeld!) een methode `createTijd(int uren, int minuten)` die, gegeven een geldig aantal uren en minuten, een waarde in het bedachte type oplevert, die dat tijdstip representeert.

Opgave 2 (20 punten)

In hotel Papageno in Wenen gebruikt men een speciale kamernummering. Kamers worden zodanig genummerd dat geen enkel kamernummer (als `String` opgevat) een prefix is van een ander kamernummer. Hierdoor is bij het intoetsen van een kamernummer op de telefooon uniek bepaald welke kamer daarbij hoort, ondanks het feit dat niet alle nummers uit evenveel cijfers bestaan. Een voorbeeld van een geldige nummering is:

```
{"11", "12", "13", "14", "20", "301", "401"} .
```

Aan deze nummering mag (bijvoorbeeld) geen kamer toegevoegd worden met nummer `"110"`, omdat `"11"` een prefix is van `"110"`.



Figuur 1: Voorbeeld van een rechthoek en de bijbehorende coördinaten

a. *(10 punten)* Schrijf een methode `boolean isGeldigeNummering(List<String> lst)` die van een gegeven lijst van kamernummers controleert of geen enkel kamernummer een prefix is van een ander kamernummer. De lijst met nummers is gesorteerd op de stringwaarde.

   Hint: De `String`-methode `boolean startsWith(String prefix)` geeft `true` als de `String` begint met `prefix`.

b. *(10 punten)* Geef passende pre- en postcondities voor de methode en een lus-invariant. Leg kort uit hoe de postconditie volgt uit de lusinvariant.

Opgave 3 (30 punten)

In deze opgave gaat het om implementaties van een interface `Rechthoek`, dat gebruikt kan worden om rechthoekige geometrische figuren te representeren, met positieve breedte en hoogte maar eventueel negatieve coördinaten. Figuur 1 bevat een schematische weergave. Alle coördinaten zijn gegeven als gehele getallen.

a. *(5 punten)* Definieer `Rechthoek` met de volgende methoden:

   - `links` (zonder parameters), die de x-coördinaat van de linkerrand oplevert;
   - `rechts` (zonder parameters), die de x-coördinaat van de rechterrand oplevert;
   - `onder` (zonder parameters), die de y-coördinaat van de onderrand oplevert;
   - `boven` (zonder parameters), die de y-coördinaat van de bovenrand oplevert;
   - `breedte` (zonder parameters), die de breedte van de rechthoek oplevert;
   - `hoogte` (zonder parameters), die de hoogte van de rechthoek oplevert.

   Voorzie de methoden waar mogelijk van postcondities.

b. *(3 punten)* Definieer een abstracte klasse `AbstracteRechthoek`, die de methoden `breedte` en `hoogte` implementeert door gebruik te maken van de andere beschikbare methoden. Gebruik deze klasse in de opgaven hieronder voorzover dit nuttig is.

c. *(6 punten)* Beschouw de volgende klasse.

```
class Punt {
    public final int x, y; // x- en y-coordinaat van dit punt

    public Punt(int x, int y) {
        this.x = x;
        this.y = y;
```

```
        }
    }
```

Definieer een klasse `ConcreteRechthoek`, die `Rechthoek` implementeert door de linkeronderpunt en de rechterbovenpunt in `Punt`-instantievariabelen bij te houden. Voorzie de instantievariabelen van klasseninvarianten en de constructor van pre- en postcondities.

d. *(4 punten)* Implementeer in `ConcreteRechthoek` een methode `doorsnede` met een `Rechthoek` als parameter, die een `Rechthoek` oplevert die de doorsnede (overlap) vormt van de twee rechthoeken (d.w.z., het object waarop de methode wordt aangeroepen en de parameter), of `null` als de twee rechthoeken niet overlappen.

e. *(6 punten)* Definieer een klasse `CombinatieRechthoek`, als subklasse van `ArrayList<ConcreteRechthoek>`. De coördinaten van een `CombinatieRechthoek` worden bepaald door de uiterste coördinaten van de in de lijst bevatte rechthoeken; bijvoorbeeld is de linkerrand van een `Combinatierechthoek` gelijk aan de meest linkse linkerrand van de bevatte rechthoeken. Van de methoden `links`, `rechts`, `onder` en `boven` hoeft u alleen de eerste in zijn geheel te geven; de andere zijn analoog. Zorg bovendien voor een passende constructor.

f. *(6 punten)* `ArrayList<Type>` heeft een methode `boolean add(Type object)` die een object achteraan in de lijst invoegt. Overschrijf deze methode in `CombinatieRechthoek`, zodat een rechthoek alleen wordt toegevoegd als deze niet overlapt met een rechthoek die al in de lijst aanwezig is.

Opgave 4 (20 punten)

In deze opgave zetten we een raamwerk op om rekeningen te kunnen samenstellen en gebruiken. Rekeningen zullen bestaan uit `Items`, die van de volgende soort kunnen zijn:

- Een gegeven aantal van een bepaald `Artikel`, gerepresenteerd door een klasse `Enkel`;
- Een `Combinatie` van andere `Items`;
- `Arbeid` van een gegeven `Werker`; d.w.z. een aantal minuten dat voor deze `Werker` in rekening wordt gebracht.

Daarbij wordt gebuik gemaakt van de volgende klassen:

```
public class Artikel {
    public Artikel(String naam, double prijs) {
        this.naam = naam;
        this.prijs = prijs;
    }

    public String getNaam() {
        return naam;
    }

    public double getPrijs() {
        return prijs;
    }

    private final String naam;
    private double prijs;
}

public class Werker {
    public static final int MINUTEN_PER_UUR = 60;

    public Werker(String naam, double uurloon) {
```

```
        this.naam = naam;
        this.uurloon = uurloon;
    }

    public double getUurloon() {
        return uurloon;
    }

    public String getNaam() {
        return naam;
    }

    public double getLoonPerMinuut() {
        return uurloon / MINUTEN_PER_UUR;
    }

    private final String naam;
    private double uurloon;
}
```

Bijvoorbeeld, als we gebruik maken van de volgende declaraties:

```
Werker bouwer = new Werker("Bob");
Artikel bier = new Artikel("Beugel", 9.70);
Artikel brood = new Artikel("Volkoren", 1.30);
```

dan kan een voorbeeld-rekening bestaan uit

- Een `Enkel` van 2 `bier`
- Een `Arbeid` van 10 minuten werk door `bouwer`
- Een `Combinatie` bestaande uit
  - Een `Enkel` van 2 `brood`;
  - Een `Enkel` van 20 `bier`

a. *(3 punten)* Programmeer een interface `Item`, met een `String`-methode `getBeschrijving` die een tekstuele beschrijving van het item oplevert, en een `double`-methode `getPrijs` die de prijs van het item oplevert.

b. *(3 punten)* Programmeer een klasse `Enkel` die `Item` implementeert. Een `Enkel`-instantie houdt bij dat een aantal exemplaren van een bepaald `Artikel` in rekening gebracht worden. Daartoe worden het aantal en het `Artikel` als instantievarabelen bijgehouden. De prijs van een `Enkel` is het product van het aantal en de prijs van het `Artikel`. De tekstuele beschrijving van een `Enkel` dient er als volgt uit te zien:

artikelnaam: *aantal* stuks

waarbij *artikelnaam* de naam van het aangeschafte `Artikel` is, en `aantal` het aantal in de `Enkel`.

c. *(4 punten)* Programmeer een klasse `Combinatie` die `Item` implementeert. Een `Combinatie`-instantie houdt een lijst `Items` bij. De prijs van de `Combinatie` is de som van de prijzen van de daarin opgenomen `Items`, en de beschrijving dient er als volgt uit te zien:

Combinatie van *aantal-items* items

(waarbij `aantal-items` het aantal items in de bijgehouden lijst is).

d. *(4 punten)* Programmeer een klasse `Arbeid` die `Item` implementeert. Een `Arbeid`-instantie houdt bij dat een bepaalde `Werker` een bepaald aantal minuten heeft gewerkt. De prijs is het product van het aantal minuten en het loon per minuut; de beschrijving dient er als volgt uit te zien

aantal-minuten `werk voor` *loon-per-minuut* `per minuut`

e. *(6 punten)* Geef een klassendiagram waarin de interface en klassen van deze en de vorige opgave in samenhang zijn weergegeven. Geef daarbij *geen* methoden of attributen weer, maar wel de bestaande verbanden tussen de klassen (geen afhankelijkheden) en hun multipliciteiten.