TENTAMEN

Programmeren 1

vakcode: 213500
datum: 28 november 2002
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 minimaal aantal punten per opgave bedraagt 0 punten.

Opgave 1 (15 punten)

De volgende opgaven zijn in de stijl van Niño/Hosch, Opgave 9.2. Geef voor elk van de volgende verzamelingen objecten een zinvolle abstractie die voor elk van de objecten geldt, en geef aan in wat voor soort applicatie (programma) dit van pas zou kunnen komen. Geef bovendien zowel voor de concrete objecten als het abstracte niveau een eigenschap of actie die daar bij hoort. Voorbeeld:

Vraag: *Tekens, regels, alinea's.*

Antwoord: Abstract niveau: *tekstelement*. Toepassing: tekstverwerker. Eigenschappen van de verschillende objecten:

- *Tekens*: Font;
- *Regels*: Afstand linker en rechter kantlijn;
- *Alinea's*: Uitlijning (links, reschts, gecentreerd);
- *Tekstelementen*: Positie op beeldscherm.

Beoordeling: –3 per fout geïdentificeerde abstractie of applicatie, –1 voor ontbrekende of foute eigenschap/actie.

a. Paspoort, rijbewijs, studentenkaart;

b. Rol toiletpapier, krat bier, pak koffie;

c. Digitale foto, internet-pagina, MP3-tje;

d. Windkracht, elektrische spanning (voltage), beurskoers;

Opgave 2 (20 punten)

Gegeven zijn de volgende klassendefinities, waarin alle methodendefinities en instantievariabelen vooralsnog ontbreken.

```
public class Bord {
    /**
     * Construeert een nieuw, leeg Bord met gegeven dimensie.
     * @require 0 <= dim
     * @ensure getDim() == dim
     */
    public Bord(int dim)
    /**
     * Levert de dimensie van dit Bord op.
     * @ensure dim >= 0
     */
    public int getDim()
    /**
     * Levert het vakje met gegeven coordinaten op.
     * @ensure als 0 <= i < getDim() en 0 <= j < getDim()
     *     dan result != null, result.getBord() == this,
     *         result.getX() == i, result.getY() == j
     */
    public Vakje getVakje(int i, int j)
}

public class Vakje {
    /**
     * Maakt een nieuw Vakje in een gegeven Bord en met gegeven coordinaten.
     * @require bord != null, 0 <= x < bord.getDim(), 0 <= y < bord.getDim()
     * @ensure getBord() == bord, getX() == x, getY() == y
     */
    public Vakje(Bord bord, int x, int y)
    /**
     * Levert het Bord op waarin dit Vakje zit.
     * @ensure result != null
     */
    public Bord getBord()
    /**
     * Levert de x-coordinaat van dit Vakje op.
     * @ensure 0 <= x < getBord().getDim()
     */
    public int getX()
    /**
     * Levert de y-coordinaat van dit Vakje op.
     * @ensure 0 <= y < getBord().getDim()
     */
    public int getY()
    /**
     * Levert het Stuk op dat op dit Vakje staat.
     * @return het Stuk dat op dit Vakje staat; null als er geen Stuk op staat.
     */
    public Stuk getStuk()
    /**
     * Zet een nieuw Stuk op dit Vakje, mits er geen Stuk op staat.
     * @return true als er nog geen Stuk op dit Vakje stond.
     * @require stuk != null
     * @ensure result == (old.getStuk() == null)
```

```
     *         als (result) dan this.getStuk() == stuk
     *         anders this.getStuk() == old.getStuk()
     */
    public boolean setStuk(Stuk stuk)
    /**
     * Haalt het Stuk van dit vakje af (als er een op staat).
     * @return het Stuk dat op dit Vakje stond
     * @ensure result == old.getStuk()
     *         this.getStuk() == null
     */
    public Stuk haalAf()
}

public class Stuk {
    /**
     * Levert het Vakje op waar dit Stuk op staat.
     * @return het Vakje waar dit Stuk op staat;
     *         null als dit Stuk niet op een Bord staat.
     */
    public Vakje getPos()
    /**
     * Zet dit Stuk op een gegeven (ander) vakje, mits dat leeg is.
     * Haalt het Stuk eerst van het huidige vakje af.
     * @return true als vakje (voor deze zet) leeg was.
     * @require vakje != null
     */
    public boolean zet(Vakje vakje)
    /**
     * Haalt dit Stuk van het bord.
     * @return true als dit Stuk op een bord stond.
     */
    public boolean sla()
}
```

a. *(5 punten.)* Implementeer de klasse `Bord` volgens de gegeven specificatie. Maak hierbij gebruik van een (naar keuze) 1- of 2-dimensionaal array van `Vakje`-instanties. Maak gebruik van klasseninvarianten om de postcondities te garanderen.

b. *(4 punten, –1 per fout antwoord.)* Beantwoord de volgende vragen, uitgaand van variabelen `Vakje v, w` and `Stuk s`. Licht, waar nodig, uw antwoord toe.

   (i) Wat is het effect van `v.setStuk(s)` als `v` de waarde `null` heeft?

   (ii) Wat is het effect van `v.setStuk(s)` als `s` de waarde `null` heeft (maar `v` niet)?

   (iii) Wat is het effect van `v.setStuk(s)` als `v` en `s` beide niet-`null` zijn, en `v.getStuk()` vóór de aanroep de waarde `null` heeft?

   (iv) Als `s`, `v` en `w` allen niet-`null` zijn, en `v.getStuk() == null` en `s.getPos() == w`, wat is dan de waarde van `v.getStuk()` en `s.getPos()` na aanroep van `v.setStuk(s)`?

   (v) Als `s`, `v` en `w` allen niet-`null` zijn, en `v.getStuk() == null` en `s.getPos() == w`, wat is dan de waarde van `v.getStuk()` en `s.getPos()` na aanroep van `s.zet(v)`?

   (vi) `Stuk` bevat geen constructor. Kan er nu wel een instantie van de klasse worden geconstrueerd?

c. *(5 punten.)* Geef postcondities voor de methoden van `Stuk`, die het resultaat van de methode zo volledig mogelijk uitdrukken.

d. *(6 punten.)* Implementeer de klasse `Stuk`. Geef invarianten by de instantievariabelen.

Opgave 3 (25 punten)

We gaan opnieuw uit van de klassen in Opgave 2. Bovendien mag u in onderstaande opgaven gebruik maken van een klasse `List` met functionaliteit zoals in het boek beschreven.

a. *(15 punten, waarvan 10 voor postconditie en lusinvariant.)* Schrijf een methode `List stukken(Bord bord)`, die een lijst oplevert met alle stukken die op het `Bord` staan. Voorzie de methode van een postconditie die uitdrukt wat het resultaat is, en van een bijpassende lusinvariant.

b. *(10 punten, waarvan 5 voor analyse en commentaar.)* Schrijf een methode `List diagonaal(Vakje van, Vakje tot)`, die een lijst oplevert met alle vakjes die een diagonaal vormen van vakje `van` tot en met vakje `tot`. (Dit hoeft dus geen complete dwarsdiagonaal van het bord te zijn.) De methode dient `null` op te leveren als dit geen correcte diagonaal oplevert. Houd rekening met alle (vier) diagonaalrichtingen! Analyseer het probleem aan de hand van één of meer kleine voorbeelden, en voorzie de methode van voldoende commentaar om de werking duidelijk te maken. Pre-/postcondities en invarianten hoeven niet gegeven te worden.

   Aanwijzing: maak hierbij gebruik van de klassenmethode `int abs(int a)` in de klasse `Math`, die de absolute waarde van een getal oplevert.

Opgave 4 (30 punten)

We gaan opnieuw uit van de klassen in Opgave 2.

a. *(3 punten)* Er zijn tenminste drie keuzemogelijkheden om een *speler* d.m.v. een Java-type te representeren: als `boolean`, als `int` of als klasse `Speler`. Weeg deze mogelijkheden tegen elkaar af, door van elke keuze de voor- en nadelen te analyseren.

b. *(7 punten)* Implementeer klassen `Schaakbord` en `Schaakstuk` die uit `Bord` resp. `Stuk` erven, en aan de volgende eisen voldoen. Pre-/postcondities en invarianten mag u achterwege laten.

   - `Schaakstuk` heeft een eigenschap `speler` die de speler aangeeft van wie het stuk is (d.w.z., wit of zwart). Kies zelf een geschikt type voor deze eigenschap. Zorg dat de speler bij constructie van het `Schaakstuk` geïnitialiseerd wordt, en geef een methode om de speler op te vragen.
   - `Schaakstuk` heeft een abstracte methode `waarde`, die een geheel getal levert die de waarde (d.w.z., de kracht) van dat stuk in het spel aangeeft. (De waarde van een pion is 1, van een loper en een paard 3, van een toren 5, van de koningin 9 en van de koning 0.)
   - `Schaakstuk` heeft een abstracte methode `kanZet`, die een `Vakje` als parameter meekrijgt en een `boolean` oplevert die aangeeft of het `Schaakstuk` van zijn huidig positie naar dat vakje gezet mag worden. `Schaakstuk` overschrijft bovendien de methode `zet` in `Stuk`, door de zet alleen uit te voeren als hij wel kán (volgens `kanZet`).
   - Een `Schaakbord` heeft altijd "dimensie" 8.

c. *(7 punten)* Implementeer een subklasse `Loper` van `Schaakstuk`. Een loper beweegt altijd diagonaal, en kan niet "over" andere stukken springen; d.w.z., als er zich tussen het huidige vakje en de nieuwe positie een ander stuk bevindt is de zet niet mogelijk. Als er op de nieuwe positie een eigen stuk staat kan de zet ook niet; als er een stuk van de tegenstander staat wordt dit stuk geslagen.

   Aanwijzing: Maak gebruik van de methode `diagonaal` uit Opgave 3.b, en voorzie uw implementatie waar nodig van commentaar. Pre-/postcondities en invarianten hoeven niet gegeven te worden.

d. *(8 punten)* Schets een klassendiagram met de klassen `Bord`, `Stuk`, `Vakje`, `Schaakbord`, `Schaakstuk` en `Loper`. De elementen (attributen en methoden) kunt u uit het diagram weglaten; geef echter wel de associatie- en abstractie-relaties tussen de klassen volledig weer.

e. *(5 punten)* Schrijf een methode `int evalueer(Bord bord, Speler speler)` (waarbij het type "`Speler`" het door uzelf gekozen representatietype is — zie boven) die de som van de waarden van alle stukken van de betreffende speler oplevert.

   Aanwijzing: Maak hierbij gebruik van de methode `stukken` uit Opgave 3.a, en voorzie uw implementatie waar nodig van commentaar. Pre-/postcondities en invarianten hoeven niet gegeven te worden.