TENTAMEN

Programmeren 1

vakcode: 213500
datum: 28 januari 2003
tijd: 9:00 – 13:30 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 3 opgaven, waarvoor in het totaal 90 punten behaald kunnen worden. Daarnaast is er een bonusvraag (Opgave e). Het minimaal aantal punten per opgave bedraagt 0 punten.

Dit tentamen gaat uit van de volgende (nog onvolledige) klassendefinities:

```
/** Klasse die een sportteam modelleert. */
public class Team {
    public Team(String naam) {
        this.naam = naam;
    }
    public final String naam;
}

/** Klasse die een sportwedstrijd modelleert. */
public class Wedstrijd {
    public Wedstrijd(Team thuis, Team gast) {
        this.thuis = thuis;  this.gast = gast;
    }
    public Team thuisTeam() { return thuis; }
    public Team gastTeam()  { return gast;  }
    public int thuisScore() { return thuisScore; }
    public int gastScore()  { return gastScore;  }
    public boolean isBezig()    { return bezig;    }
    public boolean isGespeeld() { return gespeeld; }

    /** Levert het aantal punten van het thuisteam in deze wedstrijd op. */
    public int thuisPunten() {
        if (thuisScore < gastScore)       return 0;
        else if (thuisScore == gastScore) return 1;
        else                              return 3;
    }
    /** Levert het aantal punten van het gastteam in deze wedstrijd op. */
    public int gastPunten() {
        if (thuisScore < gastScore)       return 3;
        else if (thuisScore == gastScore) return 1;
        else                              return 0;
    }
    private final Team thuis; // Het thuisteam
    private final Team gast;  // Het gastteam
    private int thuisScore;   // Score van het thuisteam
    private int gastScore;    // Score van het gastteam
    private boolean gespeeld; // Geeft aan of de wedstrijd volledig gespeeld is
    private boolean bezig;    // Geeft aan of de wedstrijd momenteel bezig is
}
```

```
/** Competitie waarin een aantal teams twee keer tegen elkaar spelen. */
public class Competitie {
    /** Construeert een lijst wedstrijden met een wedstrijd van elk team
     ** tegen elk ander team. Initialiseert tevens de instantievariabelen. */
    public Competitie(Team[] teams) Zie Opgave 2.c

    /** Verwerkt de eindstand van een gegeven wedstrijd in de punten.
     ** Past tevens de stand aan, als die door deze wedstrijd veranderd is.
     ** @require strijd zit in wedstrijden && strijd.isGespeeld() */
    public void eindeGeweest(Wedstrijd strijd) Zie Opgave 2.c

    /** Levert de wedstrijd tussen twee gegeven teams op, of null als de teams
     ** niet allebei bekend zijn.
     ** @ensure result == null ||
     **        result.thuisTeam() == thuis && result.gastTeam() == gast  */
    public Wedstrijd wedstrijdTussen(Team thuis, Team gast) Zie Opgave 2.c

    /** Levert aantal punten van een team op; -1 als team niet bekend is. */
    public int punten(Team team) {
        int index = teamIndex(team);
        if (index < 0) return -1;
        else           return punten[index];
    }
    /** Levert de lijst teams op, in de volgorde zoals bepaald door het aantal
     ** punten. D.w.z., result[0] is het team met de meeste punten, enz. */
    public Team[] stand() {
        Team[] result = new Team[aantalTeams];
        for (int i = 0; i < aantalTeams; i++)
            result[i] = teams[stand[i]];
        return result;
    }
    /** Levert de index (in "teams") van het als parameter meegegeven team.
     ** @ensure result < 0 || teams[result] == team */
    private int teamIndex(Team team) Zie Opgave 2.b

    /** Levert de index (in wedstrijden) van twee teams met gegeven team-index
     ** @require 0 <= thuisIndex < aantalTeams && 0 <= gastIndex < aantalTeams
     **       && thuisIndex != gastIndex
     ** @ensure wedstrijden[result].thuis() == teams[thuisIndex]
     **      && wedstrijden[result].gast()  == teams[gastIndex]    */
    private int wedstrijdIndex(int thuisIndex, int gastIndex) Zie Opgave 2.a

    /** Het aantal teams in deze competitie. */
    private final int aantalTeams;
    /** De teams van deze competitie, geordend naar dalend aantal punten.
     ** @invariant teams.length == aantalTeams */
    private final Team[] teams;
    /** De punten van de teams (punten[i] is het aantal punten van teams[i]).
     ** @invariant punten.length == aantalTeams */
    private final int[] punten;
    /** De teamindices in volgorde van aflopend aantal punten.
     ** @invariant stand.length == aantalTeams;
     ** voor alle 0 < i < aantalTeams: punten[stand[i]] <= punten[stand[i-1]] */
    private final int[] stand;
    /** Alle wedstrijden van deze competitie.
     ** @invariant wedstrijden bevat geen null-elementen */
    private final Wedstrijd[] wedstrijden;
}
```

Opgave 1 (30 punten)

a. *(6 punten; -2 per fout of ontbrekend antwoord.)* Beschouw de declaratie van `naam` in `Team`.

(i) Wat betekent de aanduiding `public` in deze declaratie?
(ii) Wat betekent de aanduiding `final` in deze declaratie?
(iii) Waarom wordt in het algemeen aangeraden instantievariabelen `private` te maken?
(iv) Waarom kan er in dit geval van dat principe worden afgeweken?
(v) Welke fout zou de compiler melden als `naam` als `static` variabele gedeclareerd zou zijn?
(vi) Welke fout zou de compiler melden als de toewijzing in de constructor "`naam = naam`" zou luiden?

b. *(6 punten.)* Beantwoord de volgende vragen:

(i) Waarom wordt er binnen een methode meestal niet op de precondities van die methode getest?
(ii) In wat voor soort gevallen wordt hier toch wel op getest, en waarom?
(iii) Wat is de relatie tussen precondities, klasseninvarianten en postcondities?

c. *(5 punten)* Breid de klasse `Wedstrijd` uit met:

- Een methode `void start()` die registreert dat de wedstrijd begint. Ga er van uit dat deze methode alleen aangeroepen wordt wanneer de wedstrijd niet al bezig of al gespeeld is (afgaand op de waarde van de instantievariabelen).
- Een methode `void einde()`, die registreert dat de wedstrijd ten einde is. Ga er van uit dat deze methode alleen aangeroepen wordt wanneer de wedstrijd bezig is (afgaand op de waarde van de instantievariabelen).
- Een methode `void scoort(Team scorend)`, die registreert dat een `team` een punt heeft gescoord. Ga er van uit dat deze methode alleen aangeroepen wordt wanneer de wedstrijd bezig is, met als scorende team het thuisteam of het gastteam.

d. *(8 punten)* Geef voor de boven gevraagde methoden middels precondities aan onder welke voorwaarden ze aangeroepen mogen worden, en middels postcondities welke effecten ze dan hebben. Wees hierin zo volledig mogelijk.

e. *(5 punten)* Geef middels klasseninvarianten alle bekende eigenschappen van de instantievariabelen van `Wedstrijd` aan, zodanig dat deze consistent zijn met de pre- en postcondities.

Opgave 2 (30 punten)

a. *(4 punten.)* De wedstrijden van een competitie worden achter elkaar in de array `wedstrijden` (van `Competitie`) opgeslagen. Voor een competitie van 4 teams is het schema bijvoorbeeld als volgt:

	0	1	2	3
0	–	0	1	2
1	3	–	4	5
2	6	7	–	8
3	9	10	11	–

Hierbij zijn de indices 0 t/m 3 in de bovenste rij en linkerkolom indices in de array `teams` waarin de deelnemende teams staan. Programmeer op basis hiervan de methode `wedstrijdIndex`.

b. *(11 punten.)* Programmeer de methode `teamIndex` van `Competitie`, volgens de gegeven specificatie. Voozie de methode van een lusinvariant, en maak aannemelijk dat deze de postconditie garandeert.

c. *(15 punten.)* Programmeer de constructor en de 2 ontbrekende `public` methoden van `Competitie`.

Opgave 3 (30 punten)

Beschouw de volgende declaraties:

```
public interface Fan {
    /** Signaleert dat een bepaalde wedstrijd afgelopen is.  */
    public void eindeGeweest(Wedstrijd strijd);
}

public class WAP {
    /** Zorgt ervoor dat een bericht op het display gezet wordt. */
    public void bericht(String tekst) { } // implementatie hier weggelaten
}
```

a. *(6 punten.)* Definieer een subklasse `MooieWedstrijd` van `Wedstrijd`, die in een lijst een verzameling `Fan`-instanties bijhoudt en hen op de hoogte stelt van het einde van de wedstrijd, door middel van een aanroep van `eindeGeweest`. Concreet dient `MooieWedstrijd`:

   - een methode `meldFanAan` te hebben, die een `Fan`-instantie aan de lijst toevoegt;
   - de methode `einde` van `Wedstrijd` uit te breiden zodat alle aangemelde `Fan`-instanties op de hoogte worden gesteld (zoals boven beschreven) en bovendien de lijst aangemelde fans leeg wordt gemaakt.
   - een passende constructor te hebben.

   Maak hierbij gebruik van de (voorgedefinieerde) klasse `List` met:

   - Een parameterloze constructor, die een lege lijst aanmaakt;
   - Een methode `void size()`, die het aantal lijstelementen oplevert;
   - Een methode `Object get(int i)`, die het element op positie `i` oplevert — waarbij `i` in het juiste bereik moet liggen.
   - Een methode `void remove(int i)`, die het element op positie `i` verwijdert — waarbij `i` in juiste bereik moet liggen.
   - Een method `void add(Object obj)`, die het object `obj` achteraan de lijst zet.

   (Zoals gebruikelijk loopt het "juiste bereik" van `0` tot — en niet met — `size()`.)

b. *(6 punten.)* Definieer een subklasse `FanWAP` van `WAP` met daarin een methode `void meldAanBij(MooieWedstrijd strijd)`, die als effect heeft dat de `FanWAP` zich als fan aanmeldt bij `strijd`. Zorg bovendien dat het einde van `strijd` resulteert in een passende `bericht`-aanroep.

c. *(8 punten.)* Definieer een sub-interface `GroteFan` van `Fan` met een nieuwe methode `void puntGeweest(Wedstrijd strijd, Team scorend)`, die signaleert dat er in `strijd` een punt gescoord is. Definieer een subklasse `DrommelsMooieWedstrijd` van `MooieWedstrijd` waarin het maken van een punt aan alle `GroteFans` in de fan-lijst gemeld wordt. (*Hint*: welke toegankelijkheid moet de lijst fans in `MooieWedstrijd` hebben zodat deze direct in `DrommelsMooieWedstrijd` gebruikt kan worden?)

d. *(10 punten.)* Zet alle klassen die in deze opgave voorkomen (inclusief `List` en `Object`, maar exclusief `Competitie`) in een klassendiagram, met de bijbehorende associaties, multipliciteiten en overervingsrelaties. Variabelen en methoden mag u weglaten.

e. *(Bonusvraag, 5 punten.)* Geef in woorden aan hoe de klasse `Competitie` aangepast zou moeten worden zodat haar methode `eindeGeweest` automatisch wordt aangeroepen wanneer een competitiewedstrijd ten einde is.