TENTAMEN

Programmeren 1

vakcode: 213500
datum: 15 augustus 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 5 opgaven, waarvoor in het totaal 100 punten behaald kunnen worden. Het minimaal aantal punten per opgave bedraagt 0 punten.

Opgave 1 (15 punten)

Beoordeling: max. 5 punten per deelopgave, -2 voor elk gemiste verschil.

a. Noem drie verschillen tussen een *primitief type* (of *basistype*) en een *referentietype*.

b. Noem drie verschillen tussen een *constructor* en een *methode*.

c. Noem drie verschillen tussen een *interface* en een *abstracte klasse*.

Opgave 2 (15 punten)

Gegeven zijn de volgende klassendefinities:

```
public class Plaats {
    private String plaatsnaam;
    private String land;
}

public class Adres extends Plaats {
    private String straat;
}
```

a. *(4 punten.)* Geef voor `Plaats` en `Adres` constructordefinities, waarin alle attributen van een (begin)waarde worden voorzien.

b. *(5 punten.)* Overschrijf in `Plaats` en `Adres` de methode `equals` van `Object` zodanig dat deze methode `true` oplevert als (en alleen als) alle attributen van de `Plaats`, resp. het `Adres`, inhoudelijk aan elkaar gelijk zijn.

c. *(6 punten.)* Geef klasseninvarianten voor de attributen van `Plaats` en `Adres`, en leg uit waarom deze nodig zijn. Geef tevens precondities voor de zojuist gedefinieerde constructoren en methoden, en leg uit waarom deze nodig zijn.

Opgave 3 (25 punten)

Gegeven zij de volgende interface:

```
package meteo;

public interface MooieDag {
    public Datum getDag();
    public boolean mooierDan(MooieDag ander);
}
```

en de volgende klassen:

```
package meteo;

public class Datum {
    private final int dag;   // dag van de maand;   1 <= dag <= 31
    private final int maand; // maand van het jaar; 1 <= maand <= 12
    private final int jaar;  // 1 <= jaar

    public Datum(int dag, int maand, int jaar) {
        this.dag = dag;
        this.maand = maand;
        this.jaar = jaar;
    }
}

public abstract class Meting implements MooieDag {
    protected int wind;     // windsterkte in Beaufort: 0 <= wind <= 12
    protected int temp;     // temperatuur in graden Celcius: -273 <= temp
    protected int neerslag; // neerslag in mm: 0 <= neerslag
    private Datum dag;      // datum van deze Meting; dag != null

    public Datum getDag() {
        return dag;
    }
}
```

a. *(2 punten.)* Is de declaratie van `Datum` nog correct als aan de attributen (naast `final`) ook de aanduiding `static` toegevoegd wordt? Waarom, of waarom niet?

b. *(2 punten.)* Is de aanduiding `abstract` in de klassendefinitie van `Meting` noodzakelijk? Waarom, of waarom niet?

c. *(2 punten.)* Wat betekent de aanduiding `protected` bij de attributen van `Meting`? Wat zou er anders zijn als deze aanduiding geheel weggelaten was?

d. *(7 punten; -3 als de beperking tot 1 `return`-statement niet in acht genomend wordt, -2 voor ontbrekende precondities.)* Definieer twee subklassen van `Meting`:

- Een klasse `MooieZeilDag`, waarin de methode `mooierDan` zo gedefinieerd wordt dat een dag mooier is dan een andere wanneer de eerste beter zeilweer kent. De kwaliteit van het zeilweer wordt voornamelijk bepaald door de windkracht, die van slecht naar goed gegeven is door:

$$12 - 11 - \cdots - 8 - 0 - 1 - \cdots - 6 - 7$$

  Bij gelijke windkracht telt de temperatuur: hoe warmer hoe beter.

- Een klasse `MooieStrandDag`, waarin de methode `mooierDan` zo gedefinieerd wordt dat een dag mooier is dan een andere wanneer de eerste beter strandweer kent. Als de temperatuur onder de 16 graden is of als het geregend heeft is het strandweer altijd slecht: `mooierDan` levert

dan altijd `false` op. In andere gevallen wordt de kwaliteit van het strandweer bepaald door de temperatuur: hoe warmer hoe beter. Als de temperatuur gelijk is telt de wind: hoe minder wind hoe beter.

Schrijf in beide gevallen `mooierDan` zo dat de methode maar één `return`-statement bevat. Voorzie uw methoden bovendien van precondities.

e. *(10 punten; zonder lusinvariant geen punten.)* Definieer een methode `DatumList mooiereDagen(MooieDagList van, MooieDag grens)`, die als resultaat precies alle data van de elementen in `van` oplevert die mooier zijn dan `grens`. Hierbij zijn `DatumList` en `MooieDagList` analoog aan `StudentList` in het boek (§ 12.2), en kennen dus in ieder geval methoden

- `int size()`
- `MooieDag get(int i)`, resp. `Datum get(int i)`
- `void append(MooieDag dag)`, resp. `append(Datum dag)`

Formeel luidt de postconditie van de gevraagde methode `mooiereDagen`:

```
@ensure res bestaat precies uit alle van.get(j).getDag()
        zodat 0 <= j < van.length en van.get(j).mooierDan(grens)
```

Voorzie de lus(sen) in uw oplossing van een lusinvariant, en beargumenteer dat daarmee wordt aangetoond dat de postconditie inderdaad geldt.

Opgave 4 (25 punten)

U plant een fietsvakantie, waarin u elke dag een etappe fietst en vervolgens in een hotel slaapt, dat u zoveel mogelijk van tevoren boekt. Deze vakantie wilt u modelleren, waarbij in ieder geval de volgende eigenschappen naar voren komen:

- Beginpunt van de vakantie (plaatsnaam en land) en af te leggen etappes;
- Per af te leggen etappe: afstand (in km), te bereiken eindpunt (plaatsnaam en land), en (indien geboekt) het hotel aldaar;
- Per geboekt hotel: naam van het hotel, kosten van de overnachting, adres (straatnaam, plaatsnaam en land).

Maak in uw model gebruik van concepten `Vakantie`, `Etappe` en `Hotel`, en de klassen `Plaats` en `Adres` uit Opgave 2.

a. *(10 punten.)* Geef uw model in de vorm van een klassendiagram, waarin u per klasse de bijbehorende eigenschappen weergeeft (incl. type). Voorzie uw diagram van multipliciteiten. Constructoren en methoden mag u uit het diagram weglaten.

In de *volgende* deelopgaven gaan we ervan uit dat elke eigenschap van elke klasse in uw model door middel van een `get`-methode op te vragen is — bijvoorbeeld, als de klasse `Hotel` een attribuut `naam` heeft dan is de waarde daarvan op te vragen met behulp van een methode `getNaam()`. Bovendien nemen we aan dat de etappes van een `Vakantie` worden bijgehouden in een attribuut van het type `List` (zie § 16.1 van het boek) — hetgeen wil zeggen dat in ieder geval de volgende methoden beschikbaar zijn:

- `public int size()` om de lengte van de lijst op te vragen;
- `public Object get(int i)` om de `i`-de etappe op te vragen (waarbij de eerste etappe nummer 0 heeft).

De in de volgende deelopgaven gevraagde methoden moeten in de klasse `Vakantie` komen te staan. U hoeft geen pre- en postcondities te geven, maar voorzie uw methoden van commentaar waar dat de duidelijkheid ten goede kan komen.

b. *(3 punten.)* Schrijf een methode `isRond`, die `true` oplevert als (en alleen als) de laatste etappe eindigt in dezelfde plaats (d.w.z., met dezelfde plaats- en landnaam) als waar de vakantie begon.

c. *(6 punten.)* Schrijf een methode `etappeZonderHotel`, die een etappe oplevert waarvoor nog geen hotel geboekt is, of `null` als voor alle etappes al een hotel geboekt is.

d. *(6 punten.)* Schrijf een methode `totaleAfstand`, die de som van de afstanden van alle etappes oplevert.

Opgave 5 (20 punten)

Gegeven zijn de volgende klassendefinities: (De aanroep `Console.println("tekst")` heeft tot gevolg dat het woordje `tekst` op de standaarduitvoer wordt geprint.)

```
class A {
    void m1(A par) {
        Console.println("A");
    }

    void m2(A par) {
        Console.println("A");
    }

    void m3(A par) {
        this.m1(par);
    }

    void m4(B par) {
        par.m2(this);
    }
}

class B extends A {
    void m1(A par) {
        Console.println("B");
    }

    void m2(B par) {
        Console.println("B");
    }
}
```

Kruis in onderstaande tabel aan welke uitvoer bij de gegeven methodenaanroepen wordt geprint, ofwel dat de aanroep fout is (d.w.z., in een compilatie- of executiefout resulteert), gegeven de volgende variabelendeclaraties:

```
A aa = new A();
A ab = new B();
B bb = new B();
```

Aanroep	"A"	"B"	fout
aa.m1(ab)			
aa.m1(bb)			
ab.m1(ab)			
ab.m1(bb)			
bb.m1(ab)			
bb.m1(bb)			
aa.m2(ab)			
aa.m2(bb)			
ab.m2(ab)			
ab.m2(bb)			
bb.m2(ab)			
bb.m2(bb)			
aa.m3(ab)			
aa.m3(bb)			
ab.m3(bb)			
bb.m3(bb)			
aa.m4(ab)			
aa.m4(bb)			
ab.m4(bb)			
bb.m4(bb)			

Beoordeling: -2 per fout antwoord.

Zet uw naam op dit blad, en lever het samen met uw uitwerking in.

Naam:
Studentnr.: