Tentamen OOP, maandag 23 april 2007, 09:00-12:00.

LEES DIT EERST !

- Dit tentamen behoort bij het 5-EC vak Object-georiënteerd Programmeren.
- Vul de kop van het eerste in te leveren blad **volledig** in.
- Nummer de bladen en zet bovenaan het eerste blad het totaal aantal ingeleverde bladen; voorzie elk blad van je naam.
- Werk zorgvuldig en schrijf netjes met blauwe of zwarte pen; **geen** potlood!
- Het werk inleveren: leg je bladen op volgorde, vouw ze dubbel en doe ze in de envelop. Plak de envelop **niet** dicht. Voorzie de envelop van je naam en je studentnummer.
- Lees elke opgave eerst volledig door.
- De BONUSVRAGEN in het tentamen zijn niet verplicht, maar kunnen extra punten kunnen opleveren. Er is in totaal een score van 115% mogelijk is.
- Het gecorrigeerde werk is na ongeveer drie weken af te halen bij de studieadministratie Wiskunde en Informatica. Je wordt verzocht dit ook daadwerkelijk te doen!
- Indien niet aan de gestelde practicumeis is voldaan zal geen tentamenbriefje worden uitgereikt.
- Het hertentamen is op 25 juni 2007.

VEEL SUCCES !

■ **begin tentamen**

Opgave 1 [25%] (OO-theorie)

Belangrijke begrippen in object-georiërenteerd programmeren zijn: *klasse*, *object* en *inheritance* (= overerving).

(a) Beschrijf kort en duidelijk het verschil tussen de begrippen *klasse* en *object*.
(b) Beschrijf kort en duidelijk het begrip *inheritance*.
(c) Leg uit welke relatie aangeeft dat er inheritance toegepast kan worden.
(d) BONUSVRAAG: Leg uit wat *dynamic binding* is.

Geef waar nodig voorbeelden om je antwoorden te verduidelijken.

lees verder ▶

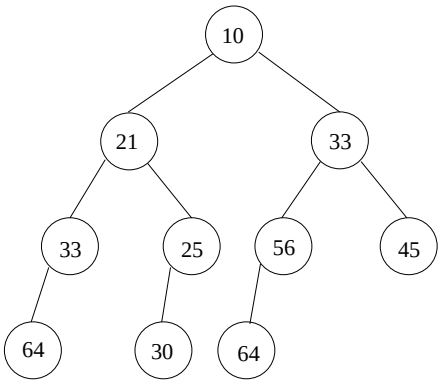
Opgave 2 [35%] (Datastructuren)

In deze opgave wordt gevraagd een zogenaamde *heap*-datastructuur te maken. Een heap is een binaire boom waarbij elke knoop in die boom voldoet aan twee eigenschappen:

de heap-eigenschap de waarden in zowel de linker- als de rechtersubboom zijn groter dan of gelijk aan de waarde in de knoop zelf.

de balans-eigenschap het aantal kinderen in de linkersubboom is gelijk aan, óf precies één groter, dan het aantal kinderen in de rechtersubboom.

In een heap kunnen waarden worden opgeslagen. Uit de heap-eigenschap volgt dat het kleinste element dat in de heap opgeslagen is in de wortel-knoop staat. Met de balans-eigenschap wordt voorkomen dat de boom onnodig hoog wordt. De balans-eigenschap bepaalt waar (links of rechts) een nieuwe knoop toegevoegd moet worden.



Figuur 1: Voorbeeld van een heap

We representeren een knoop in de heap als volgt:

```
class HeapNode {
  int val;

  /** balance : true  = links en rechts evenveel knopen
                false = links heeft 1 knoop meer dan rechts */
  boolean balance;

  HeapNode left  = null;
  HeapNode right = null;

  HeapNode() { // default constructor
    balance = true; // links & rechts geen kinderen, dus gelijk
  }
}
```

lees verder ▶

Net als bomen representeren we de heap zelf met een eigen klasse. Hier is alvast een aanzet:

```
public class Heap {
  private HeapNode root;

  public Heap() { // default constructor
    root = null;
  }
}
```

In de volgende opgaven is het de bedoeling dat je de gegeven code uitbreidt. Het staat je hierbij vrij om hulp-methodes te introduceren.

(a) Voorzie de klasse HeapNode van een constructor met een **int** als parameter: de waarde voor de nieuwe knoop.

(b) Voeg toe aan de klasse Heap een methode

```
public int size()
```

die het aantal opgeslagen elementen in de heap oplevert.

(c) Voeg aan de klasse Heap een methode

```
public boolean contains(int v)
```

toe, die oplevert of waarde v in de heap is opgeslagen. Zorg ervoor dat deze niet nodeloos inefficiënt is.

(d) Implementeer in de klasse Heap een methode

```
public boolean checkConsistency()
```

die oplevert of de heap aan zowel de heap- als balans-eigenschap voldoet. Hint: maak gebruik van verschillende hulpmethoden; eentje om de heap-eigenschap te controleren, en eentje om de balans-eigenschap te controleren.

(e) Voorzie de klasse Heap van een methode

```
public void add(int v)
```

die een waarde v aan de heap toevoegt. Zorg dat na uitvoering van de methode de heap nog steeds aan de twee gestelde eigenschappen voldoet. Hint: voeg eerst de waarde toe aan de heap, op een manier dat de balans-eigenschap niet wordt geschonden. Herstel vervolgens de heap-eigenschap, indien nodig.

(f) BONUSVRAAG: Heaps worden vaak als *priority queue* gebruikt, waarbij het opvragen en/of verwijderen van het kleinste (of voorste) element een basisoperatie is. Voorzie daarom de klasse Heap van een methode

```
public int deleteMinimum()
```

die de kleinste waarde uit de heap verwijdert en teruggeeft. Zorg ervoor dat na uitvoering van de methode de heap nog steeds aan de twee gestelde eigenschappen voldoet.

lees verder ▶

Opgave 3 [30%] (Recursive-descent parsing)

In deze opgave wordt een herkende parser gevraagd voor de volgende grammatica voor zeer simpele programma's:

```
<program>    ::= <statements> .
<statements> ::= { <statement> ';' } .
<statement>  ::= <assignment> | <query> .
<assignment> ::= 'SET' <var> '=' <expr> .
<query>      ::= 'GET' <var> .
<expr>       ::= <value> [ <op> <value> ] .
<op>         ::= '+' | '-' | '*' | ':' .
<value>      ::= <var> | <num> .
<var>        ::= <char> { <char> } .
<num>        ::= <digit> { <digit> } .
<char>       ::= 'a' | 'b' | ... | 'z' .
<digit>      ::= '0' | '1' | ... | '9' .
```

De karakters in een variabele-naam (hulpsymbool `<var>`) en de cijfers in een getal (hulpsymbool `<num>`) staan aaneengesloten; tussen alle andere elementen mag witruimte staan. Onder witruimte verstaan we spaties, tab- en regelovergangs-symbolen. Een programma dient gevolgd te worden door een einde-bestand (ook wel end-of-file) symbool.

(a) Beschrijf wat een tokenizer doet.

(b) Gesteld dat je een Tokenizer voor bovenstaande grammatica zou moeten gaan maken, geef aan welke tokens deze kunnen opleveren. Je kan dit doen door een klasse-hiërachie van alle aanwezige tokens te tekenen. Je mag uit gaan van een *abstracte* klasse Token met de volgende publieke constructoren en methoden:

```
/** creeert een Token met stringrepresentatie s */
public Token(String s)

/** creert een Token met characterrepresentatie ch */
public Token(char ch)

/** retourneert de stringrepresentatie van het Token */
public String toString()
```

Ga er vanuit dat Tokenizer een klasse is die de opgesomde tokens oplevert. Hieronder geven kort nog even de beschikbare constructoren en methoden van de klasse Tokenizer zoals deze op college en het practicum aan de orde zijn geweest:

```
public Tokenizer(InputStream is) // constructor
public Tokenizer(Reader r) // constructor

/** Levert het huidige token als resultaat */
public Token getCurrent()

/** Schuift de tokenizer door naar het volgende token */
public void moveNext()
```

lees verder ▶

Gevraagd wordt een herkennende recursive-descent parser te schrijven voor de gegeven grammatica. We geven alvast een klasse `Parser`:

```
absract class Parser {
  protected static void reportError(String message) {
    System.out.println("FOUT: " + message);
    System.exit(0);
  }
}
```

(c) Beschrijf voor welke hulpsymbolen er subklassen van `Parser` gemaakt moeten worden.

(d) Geef voor elk van deze klassen de bijbehorende implementatie van de volgende methode

```
public static boolean tryParse(Tokenizer tok)
```

Merk op dat er niet gevraagd wordt een parse-tree op te bouwen, of excepties op te gooien. Er wordt slechts een *herkennende* parser gevraagd. Bij het constateren van een fout is het voldoende om de reportError methode aan te roepen.

Opgave 4 [25%] (Excepties en I/O)

In de grammatica van opgave 3 kun je zeer simpele programma's schrijven. Bijvoorbeeld:

```
SET aap = 3;
GET aap;
SET noot = aap * 9;
SET mies = noot - 20;
GET mies;
```

Deze programma's bestaan uit een aantal simpele statements zoals toekenningen van simpele expressies aan variabelen, en de waarde van variabelen opvragen. Een grammaticaal correct programma is echter geen garantie voor een zinvol programma; delen door nul en de waarde van ongeïnitialiseerde (= nog niet een waarde aan toegekend) variabelen opvragen, worden niet uitgesloten. Deze kunnen middels *excepties* worden gesignaleerd.

(a) Geef je eigen klasse definities in Java voor beide hierboven genoemde excepties

Een toestand registreert welke waarde een variabele momenteel heeft. Een toekenning in het programma verandert de toestand, en de waarde van de simpele expressie wordt bepaald in een gegeven, huidige, toestand. Net als op college is een toestand een interface, en beschrijft deze welke methoden een concrete implementatie tenminste moet definiëren.

(b) Hieronder staat het grootste deel van het interface State gegeven. Maak deze beschrijving af (op de open puntjes) zodat deze vastlegt dat de bij onderdeel (a) gegegeven ongeïnitialiseerde-variabele-exceptie opgegooid kan worden bij de methode `query`.

```
public interface State {
  /** kent aan variabele var de waarde value toe */
  public void assign(String var, double value);
  /** geeft de aan variabele var toegekende waarde terug */
  public double query(String var) .. .. ..
}
```

lees verder ▶

(c) BONUSVRAAG: Geef een implementatie van State m.b.v. `java.util.Hashtable`

Voor het vervolg gaan we er vanuit dat er een parser is voor de grammatica van opgave 3 die wel een parse-tree opbouwt als de invoer aan de grammatica voldoet, en anders een `ParseException` geeft. Een Program-object vormt de wortel van de parse-tree en representeert het gehele programma. Een programma kan worden uitgevoerd gegeven een toestand. Tijdens executie kunnen de bij onderdeel (a) genoemde excepties optreden.

(d) Hieronder is een fragment van de klasse Program gegeven. Gevraagd is de methode-headings aan te vullen (op de open puntjes) zodat deze de correcte excepties vermelden:

```
public class Program extends ParseTreeNode {

  // prive velden en de constructor

  public static Program tryParse(Tokenizer tok) .. .. .. {
    // code
  }

  public void execute(State s) .. .. .. {
    // code
  }

  // rest van de methoden
}
```

Een interpreter is een programma dat een ander programma inleest en uitvoert. Gegeven het hierboven aangeleverde werk is een interpreter niet veel meer dan een programma dat dient te worden aangeroepen met één parameter, welke de naam van het bestand is waarin een programma staat. De interpreter controleert of de inhoud van het bestand aan de bovenstaande grammatica voldoet. Als dit niet zo is, dan stopt de interpreter en sluit af met een passende foutmelding. Voldoet de inhoud wel aan de grammatica, dan wordt het programma uitgevoerd vanuit de initiële toestand (een toestand zonder geïnitialiseerde variabelen).

(e) Gevraagd wordt het interpreter hoofdprogramma te schrijven. Zorg hierbij voor een nette afhandeling van eventuele fouten (excepties enz.) die kunnen optreden.

Aanwijzingen:

- Voor het inlezen kent Java een FileReader en BufferedReader
- In opgave 3 staan de constructoren van de Tokenizer

einde tentamen ■