Tentamen OGP, donderdag 26 juni 2008, 09:00-12:00.

LEES DIT EERST !

- Dit tentamen behoort bij het 5-EC vak Object-georiënteerd Programmeren.
- Vul op het eerste in te leveren blad je naam, student nummer en e-mail adres 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!
- Lees elke opgave eerst volledig door.
- De BONUSVRAGEN in het tentamen zijn niet verplicht, maar leveren extra punten op. Er is in totaal een score van 110% mogelijk.
- Het gecorrigeerde werk is na ongeveer twee tot drie weken in te zien op 5161.576.
- Indien niet aan de gestelde practicumeis is voldaan zal geen eindcijfer worden uitgereikt.

VEEL SUCCES !

■ **begin tentamen**

Opgave 1 [25%] (OO-theorie)

Belangrijke begrippen in object-georiënteerd 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 verschil en de overeenkomst tussen de begrippen *abstract class* en *interface*.
(c) Wat is data-hiding?
(d) Wat is een klasse variabele? Met welk keyword wordt dit aangegeven in Java?
(e) Wat betekent het *overloaden* en *overriden* van een methode?

Geef waar nodig voorbeelden om je antwoorden te verduidelijken.

lees verder ►

Opgave 2 [10%] (UML)

Een manier om klasses en hun relaties weer te geven is het maken van een diagram. Maak een UML class diagram over het onderwerp "de universiteit". Zorg dat je diagram de werkelijkheid natuurgetrouw modelleerd en de volgende elementen bevat:

- 10 verschillende klasses
- ten minste 1 multiple inheritance relatie
- 5 inheritance relaties
- 2 aggegratie relaties

Geef voor elke klasse een korte omschrijving en een motivate van de gemodelleerde relaties.

Opgave 3 [15%] (Zoek de fout)

Goed programeren is niet altijd eenvoudig. Het vorige tentamen bewees dit wel waar gevraagd werd een implementatie te geven van binaire zoekbomen.
De volgende klasses implementeren een zo'n binaire zoekboom. Helaas bevat het een aantal fouten, zowel syntaxtisch, semantisch, als conventie overtredingen. Identificeer ten minste 10 van deze fouten. Geef voor elke fout die je vindt het regelnummer en stukje code dat de fout voorstelt, een korte omschrijving wat er mis mee is en een stukje code om de fout te herstellen. BONUS: identificeer nog twee extra fouten.

lees verder ▶

```
1  class BinSearchTree<T extends Comparable<T>> extends BinTree {
2      public BinSearchTree() {super(T);}
3      protected boolean contains(T x, BinTreeNode<T> t) {
4          if (t == null || t.data == null) { return false; }
5          if (t.data.compareTo(x) == 0) { return false; }
6          if (t.data.compareTo(x) > 0) {
7              return contains(x, t.left);
8          }
9          return contains(x, t.right);
10     }
11     protected BinTreeNode<T> insert(T x, BinTreeNode<T> t) {
12         if (t = null) { return new BinTreeNode(x); }
13         if (t.data.compareTo(x) >= 0) {
14             t.left = insert(x, t.left);
15         } else {
16             t.right = insert(x, t.right);
17         }
18         return t;
19     }
20     public void insert(T x) {
21         root = insert(x, root);
22     }
23 }
24 class BinTree<T> {
25     BinTreeNode<T> root;
26     public BinTree() {
27         root = null;
28     }
29     private int Size(BinTreeNode<T> t) {
30         if (t != null) { return 1 + Size(t.left) + Size(t.right); }
31         return 0;
32     }
33     public int size() {
34         return size(root);
35     }
36     protected int height(BinTreeNode<T> t) {
37         if (t == null) { return height; }
38         return Math.max(height(t.left),height(t.right));
39     }
40     public int height() {
41         return height(root);
42     }
43
44     private boolean contains(T x, BinTreeNode<T> t) {
45         if (t == null || t.data == null) { return false; }
46         return (t.data == x) || contains(x, t.left) || contains(x, t.right);
47     }
48     public boolean contains(T x) {
49         return contains(x, root);
50     }
51 }
52 class BinTreeNode<T> {
53     BinTreeNode<T> data = null;
54     BinTreeNode<T> left  = null;
55     BinTreeNode<T> right = null;
56     public BinTreeNode() {}
57     public BinTreeNode(T x) {
58         data = x
59     }
60 }
```


3

Opgave 4 [30%] (Verzamelingen)

In de vorige opgave werd een implementatie gegeven van een binaire zoekboom. Ga voor deze opgave ervan uit dat deze klasses inmiddels verbeterd zijn. Een binaire zoekboom kan gebruikt worden om een verzameling te representeren. Aangezien duplicaten niet voorkomen in verzamelingen moeten we er dus voor zorgen dat dit ook niet gebeurd in een binaire zoekboom die een verzameling voorstelt. Zorg ervoor dat bij operaties die je op de boom uitvoert de zoek-eigenschap in stand blijft. Implementeer nu de klasse:

```
class Set<T extends Comparable<T>> extends BinSearchTree<T> {
  ......
}
```

Voorzie deze klasse van:

(a) De juiste constructor(en)

(b) Een methode om het aantal elementen op te leveren.

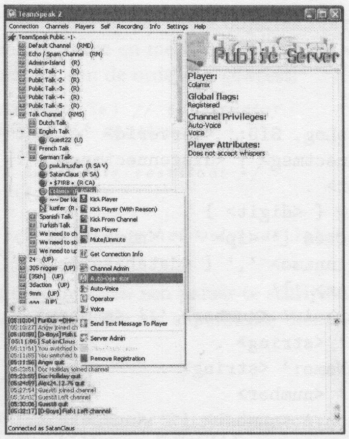
(c) Een methode om een element toe te voegen.

(d) Een methode om de verzameling te verenigen met een andere verzameling.

(e) Een methode om een element te verwijderen.

(f) BONUS: Een methode om de verzameling te doorsnijden met een andere verzameling.

lees verder ▶



Figuur 1: Teamspeak client

1 Opgave 5 [30%] (Recursive-descent parsing)

In deze opgave proberen we met behulp van parsing en enkele datastructuren een concreet probleem op te lossen. Een bekend en veel gebruikt voice chat programma is Teamspeak. Het programma bestaat uit een client (zie figuur 1), die gebruikers in staat stelt om verschillende voice chat kanalen op een voice chat server te joinen. Nu zouden we graag willen weten, hoe actief elke gebruiker van een teamspeak server is.

Gelukkig heeft de teamspeak server een optie om in een log file bepaalde acties van gebruikers bij te houden. Twee van de acties die qworden opgeslagen zijn het moment dat de gebruiker verbinding maakt met een server en wanneer hij of zij deze verbinding weer verbreekt.

De volgende (vereenvoudigde) grammatica beschrijft hoe de relevante delen van een Teamspeak log file eruit zien:

```
<tslog>       ::= { <tslogentry> } .
<tslogentry>  ::= <datetime> ',ALL,Info,' (<servermsg> | <accessmsg>).)
<datetime>    ::= <date> <time> .
<date>        ::= <day> '-' <month> '-' <year> .
<day>         ::= <twodigit> .
<month>       ::= <twodigit> .
<year>        ::= <twodigit> .
<time>        ::= <hour> ':' <minute> ':' <second> .
<hour>        ::= <twodigit> .
<minute>      ::= <twodigit> .
<second>      ::= <twodigit> .
<twodigit>    ::= <digit><digit> .
<digit>       ::= '0' | '1' | ... | '9' .
<servermsg>   ::= 'server,Server' (<shutdownmsg> | <initmsg>) .
<shutdownmsg>::= 'shutdown' ('initialized' | 'finished') .
<initmsg>     ::= 'init' ('initialized' | 'finished') .
```


5

```
<accessmsg>   ::= 'AccessLog, SID:' <serverid> 'client'
                  ( <connectmsg> | <disconnectmsg> ) .
<serverid>    ::= <number> .
<number>      ::= <digit> { <digit> } .
<connectmsg>  ::= 'connected [' <ip> ',' <nickname> ','
                  [ <loginname> ',' [ <databaseid> ',' ] ]
                  <version> ']' .
<ip>          ::= <number> '.' <number> '.' <number> '.' <number> .
<nickname>    ::= 'Nick:' <string> .
<loginname>   ::= 'LoginName:' <string> .
<databaseid>  ::= 'DBID:' <number>
<version>     ::= 'Version:' <number> '.' <number> '.' <number> '.'
                  <number> .
<string>      ::= { <char> } .
<char>        ::= 'a' | 'b' | ... | 'z' | 'A' | ... |'Z' .
<disconnectmsg>::= 'disconnected.[' <nickname>
                     [ ',' <loginname> ',' <databaseid> ] ']' .
```

De karakters in een string (hulpsymbool `<string>`) en de cijfers in een getal (hulp-symbool `<number>`) staan aaneengesloten; tussen alle andere elementen mag witruimte staan. Onder witruimte verstaan we spaties, tab- en regelovergangs-symbolen. Een log file eindigt met een end-of-file symbool.

Wanneer de teamspeak server gestart of gestopt wordt, dan wordt een `<servermsg>` in de log file geplaatst. Hetzelfde gebeurt als een client een verbinding maakt of verbreekt (`<connectmsg>`, `<disconnectmsg>`).

(a) Geef een voorbeeld van een client disconnect log entry, die voldoet aan de beschreven grammatica.

(b) Geef een voorbeeld van een log entry, die voldoet aan de grammatica, als de server gestart/geinitialiseerd wordt.

(c) Wat is parseren?

(d) Wat zijn de tokens van de gegeven grammatica? Hint: dit zijn *niet* de token klasses!

lees verder ►

Ga er vanuit dat Tokenizer een klasse is die token objecten oplevert. Hieronder wordt kort nog even de beschikbare constructoren en methoden gegegeven 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()
```

Doel is om met behulp van deze Tokenizer een parser te schrijven voor deze grammatica. We geven alvast de klasse ParseTreeNode, die als superklasse dient van de parse klasses en hun resultaat:

```
abstract class ParseTreeNode {

  protected void reportError() throws ParseException {
    throw new ParseException(''Parse Error'');
  }

  public abstract ParseTreeNode tryParse(Tokenizer tok) throws ParseException;

  public String toString() {
    return getClass().getName().substring(getClass().getName().lastIndexOf("."));
  }
}
```

(e) Schrijf een recursive descent parser klasse DatabaseID, welke de productie `<databaseid>` parseert en het database nummer opslaat.

einde tentamen ■