Tentamen OGP, maandag 6 april 2009, 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)

(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 abstracte klasse en interface.
(c) Geef twee access modifiers en omschrijf hun specifieke doel.
(d) Je kunt op twee manieren nieuw klassen maken met behulp van bestaande klassen: door overerving en door compositie (of aggregatie zoals dat ook wel heet). Geef van beide manieren een voorbeeld.
(e) BONUS: Wat is het voordeel van het toepassen van het "dependency inversion principle" (dat wil zeggen dat een hoog-niveau klasse en een laag niveau klasse beiden een gemeenschappelijk interface gebruiken in plaats van dat de hoog-niveau klasse rechtstreeks de laag-niveau klasse gebruikt?

Geef waar nodig voorbeelden om je antwoorden te verduidelijken.

volgende vraag ▶

Opgave 2 [30%] (Datastructuren)

In deze opgave maken we een queue met behulp van een circulair array.

(a) Wat is het verschil tussen een queue en een stack?

Een circulair array is een gewoon array waarbij we net doen alsof het laatste element wordt gevolgd door het eerste. Als het array a 17 elementen heeft dan doen we alsof a[17] == a[0] en a[20] == a[3]. Indexeren gaat dus modulo (in java is dat %) het aantal elementen.

Een begin van de klasse queue is dan:

```
class Queue<T> {
      private T[] a;
      private int kop;
      private int staart;
      private int size;
      public Queue(int maxSize) {....}
      public boolean isFull() {...}
      public boolean isEmpty() {...}
      public void enqueue(<T> value) {....}
      public <T> dequeue() {.....}
}
```

De kop (het eerste element) van de queue staat in a[kop] en de staart (het laatste element) in a[staart]. De queue 3 5 7 kan bijvoorbeeld zo in het array a staan: 5, 7, 4, 2 , 5, 3. Kop is dan 5 en staart 1 (en size 3).

Als je het nodig vindt mag je extra attributen en methoden toevoegen. Maar denk eraan: in wat je niet toevoegt kunnen geen fouten zitten.

(b) Maak de constructor af. De parameter is het aantal elementen dat de queue maximaal kan bevatten.

(c) Wanneer is de queue vol? En wanneer leeg? Maak de methoden isFull en isEmpty af.

(d) Wanneer mag de methode enqueue (die een nieuw element in de queue zet) worden uitgevoerd (ofwel wat is de preconditie van enqueue)? Maak enqueue af.

(e) Wat is de preconditie van de methode dequeue (die een element uit de queue haalt)? Maak dequeue af.

(f) BONUS: aan welke eisen moet het attribuut size voor elk object van de queue in elk geval voldoen? (Ofwel: formuleer een klasse invariant voor Queue.)

volgende vraag ▶

Opgave 3 [30%] (Parsing)

In deze opgave wordt een herkennende parser gevraagd voor de volgende grammatica voor eenvoudige programma's:

```
<program>      ::=  <statements> .
<statements>   ::=  { <statement> ';' } .
<statement>    ::=  <increment> | <decrement> | <assignment> |
                        <repetition> .
<increment>    ::=  'INC' <var> [ <value> ] .
<decrement>    ::=  'DEC' <var> [ <value> ] .
<assignment>   ::=  'LET' <var> '=' <value> .
<repetition>   ::=  'WHILE' <var> 'DO' <statements> 'END' .
<value>        ::=  <var> | <num> .
<var>          ::=  <char> { <char> } .
<num>          ::=  <digit> { <digit> } .
<char>         ::=  'a' | 'b' | ... | 'z' .
<digit>        ::=  '0' | '1' | ... | '9' .
```

De karakters in een variabelenaam (hulpsymbool <var>) en de cijfers in een getal (hulpsymbool <num>) staan aaneengesloten; tussen alle andere elementen mag een willekeurige hoeveelheid 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.

Gesteld dat je een Tokenizer voor bovenstaande grammatica zou moeten gaan maken is het de vraag welke tokens deze moet kunnen opleveren.

(b) Het is in elk geval handig om een InvalidToken en een EOFToken te hebben. Waarom?

(c) Geef de andere tokens die de tokenizer zou moeten kunnen opleveren.

(d) BONUS: Leg uit waarom iets als een identifier wel door een tokenizer kan worden opgeleverd en een statement niet.

Ga er vanuit dat Tokenizer een klasse is die de opgesomde tokens oplevert. Hieronder ge ven we nog kort 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 resultaat */
public Token getCurrent()
/** Schuift de tokenizer door naar het volgende token */
public void moveNext ()
```

lees verder▶

Gevraagd wordt om een herkennende parser te schrijven voor de gegeven grammatica.

We geven alvast een klasse Parser :

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

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

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

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

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

volgende vraag ▶

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

In deze opgave gaat het over een erg ouderwets programma: het heeft geen flitsend user interface, maar het leest zijn invoer van de command line. Het resultaat wordt gewoon afgedrukt met System.out.println.

Het programma heet "zeef", het leest een getal en dan nog een (eventueel leeg) rijtje *positieve* getallen vanaf de command line en drukt af of er een deelrijtje van het rijtje getallen bestaat met als som het eerste getal.

Biivoorbeeld:

```
java zeef -som 125 -rij 17 33 15 31 49 27 31 44 11 19
```

drukt af: "waar" omdat de deelrij 33 15 31 27 19 als som 125 heeft. Een deelrij hoeft dus niet aaneengesloten te zijn.

(a) Schrijf een methode "readInput" die de som in een int variabele som inleest en het rijtje in een int array R. Houd er rekening mee dat gebruikers altijd dingen verkeerd doen. Bijvoorbeeld geen invoer geven of zich niet houden aan de regels over het gebruik van -som en -rij. Zorg daarom dat de methode bij alle mogelijke fouten van een gebruiker een goede exceptie "throwt". Aanwijzing: je hebt vast de methode Integer.parseint(invoer) nodig die de string *invoer* omzet in een int. In de documentatie staat hier bij: throws NumberFormatException.

(b) Schrijf het stukje code dat de methode readInput aanroept en zorgt dat de gebruiker zinvolle meldingen krijgt als er een verkeerde invoer is gegeven.

(c) Java kent checked en unchecked exceptions. Wat is het verschil tussen die twee? Waarom zijn niet alle exception "checked"?

(d) Wat is het doel van een "finally" clause?

(e) BONUS: Waarom is het soms beter om bij het programmeren eventueeel achteraf om vergeving te vragen dan altijd vooraf om toestemming? Leg ook uit wat dat met het exception mechanisme te maken heeft.

einde tentamen ■