Faculteit der Exacte Wetenschappen tentamen Datastructuren
Vrije Universiteit 18 december 2003

Studenten die het tweede deeltentamen Datastructuren maken krijgen 2:00 uur de tijd en maken de opgaven 2 t/m 3.

Studenten die het gehele tentamen Datastructuren maken krijgen 3:00 uur de tijd en maken de opgaven 1 t/m 3.

Opgave 1.

a) Bij overerving (inheritance) van een class A in een andere class B erft (inherit) de class B velden (members) van de class A. Niet alle velden worden echter geerfd. Geef aan welke velden niet geerfd worden door de subclass.

Is het mogelijk om in een subclass te zeggen dat deze subclass een bepaalde methode uit de superclass niet wil erven? Zo ja, hoe dan?

Is het mogelijk in een subclass om een methode, die uit de superclass geerfd wordt, te herdefinieren? Zo ja, hoe dan?

Is het mogelijk in een subclass om een methode, die uit de superclass geerfd wordt, uit te breiden? Zo ja, hoe dan?

Hoe moet in een subclass de default constructor uit de superclass worden aangeroepen?

b) Leg uit hoe het met behulp van inheritance mogelijk is om een array te declareren waarvan de elementen willekeurige objecten kunnen zijn.

Geef de declaratie van een variabele generiekeRij die in de declaratie geinitialiseerd wordt als zo'n array met maximaal 35 elementen. Gebruik constanten indien nodig.

c) Als het de bedoeling is dat de elementen van een array totaal verschillende soorten objecten kunnen zijn, maar niet ieder willekeurig object, dan wordt dit niet met behulp van inheritance maar met behulp van interfaces gerealiseerd.

Leg uit en laat zien hoe je het zou kunnen aanpakken om te zorgen dat de classes

```
class X {          class Y {          class Z {
....               ....               ....
}                  }                  }
```

de enige drie soorten objecten definieren die aan elementen van een array xyzRij, met maximaal 20 elementen, mogen worden toegekend.

Geef ook de declaratie van de variabele xyzRij.

d) Wanneer is er sprake van overloading?
Wanneer is er sprake van polymorfisme (polymorphism)?

Wat is een abstract method?
Wat is een final method?
Wat is een protected method?

Opgave 2.

a) Een gerichte graaf (directed graph), waarbij iedere knoop een hoofdletter bevat, kan zowel met een adjacency matrix als met een adjacency list gerealiseerd worden.
Als gegeven is dat een graaf de onderstaande knopen (nodes) en kanten (edges) bevat, teken dan de adjacency matrix waarmee de gegeven graaf gerealiseerd kan worden.

Er is een kant van knoop 'A' naar knoop 'B'
Er is een kant van knoop 'A' naar knoop 'C'
Er is een kant van knoop 'B' naar knoop 'A'
Er is een kant van knoop 'B' naar knoop 'D'
Er is een kant van knoop 'B' naar knoop 'E'
Er is een kant van knoop 'C' naar knoop 'B'
Er is een kant van knoop 'C' naar knoop 'E'
Er is een kant van knoop 'D' naar knoop 'A'

b) Implementeer de onderstaande minimale specificatie van een lijst in een class Lijst.

```
interface LijstInterface {

/* Elementen: objecten van het type Object
   Structuur: lineair
   Domein:    alle rijen van objecten van het type Object
*/

/* Constructor
   Lijst ();
   PRE  -
   POST - de nieuwe lijst is leeg
*/

Lijst insert (Object obj);
/* PRE  -
   POST - obj is toegevoegd aan de lijst en de lijst is geretourneerd.
*/
```

```
}
```

c) Gegeven is dat de knopen van de class Graaf gerealiseerd worden met de onderstaande class.

```
class GraafKnoop {
   char hoofdletter;
   Lijst buren;

   GraafKnoop (char c) {
            hoofdletter = c;
            buren = new Lijst();
   }
}
```

Verder is gegeven dat de onderstaande class Graaf een ongerichte graaf realiseert met een adjecency list.
Implementeer de methodes voegKnoopToe() en voegKantToe() uit deze class Graaf welke respectievelijk een knoop (node) en een kant (edge) aan de graaf toevoegen.

```
class Graaf {
   Lijst lijst;

   Graaf () {
            lijst = new Lijst();
   }

   GraafKnoop voegKnoopToe (char c) { // c is een hoofdletter
   // Deze methode voegt aan de graaf een knoop toe met inhoud c
   // De toegevoegde knoop wordt geretourneerd.
            ....
   }

   void voegKantToe (GraafKnoop k1, GraafKnoop k2) {
   // Deze methode voegt aan de graaf een kant tussen de knopen k1
   // en k2 toe.
            ....
   }
}
```

Opgave 3.

Gegeven is de onderstaande class voor de implementatie van knopen in een boom:

```
class Knoop {
   int data;
```

```
    Knoop links, rechts;

    Knoop (int data, Knoop links, Knoop rechts) {
            this.data = data;
            this.links = links;
            this.rechts = rechts;
    }

    Knoop () {
            this(0, null, null);
    }
}
```

a) Wanneer is een boom (tree) een binaire boom (binary tree)?

Een binaire boom is een hoop (heap) indien deze binaire boom voldoet aan de hoopvoorwaarden (heap storage rules) en een complete binaire boom (complete binary tree) is.

Welke zijn de hoopvoorwaarden?

Wanneer is een binaire boom een complete binaire boom?

Geef een recursieve implementatie van de onderstaande methode

```
boolean hoop (Knoop k);
/* PRE  - k is een referentie naar de wortel van een
              complete binaire boom
  POST - TRUE : de complete binaire boom met k als wortel
                is een hoop.
         FALSE: de complete binaire boom met k als wortel
                is geen hoop
*/
```

Kan een binaire zoekboom (binary search tree) met meer dan 1 knoop een hoop zijn?

b) Wat is een queue?

Wat is een priority queue?

Beschrijf, voor het geval dat een priority queue geimplementeerd is met een hoop, hoe het toevoegen/verwijderen van een element aan/uit de priority queue uitgevoerd wordt.
N.B. er wordt dus niet gevraagd om iets te programmeren.

c) Gegeven is het volgende stuk van een interface van een rij

```
interface RijInterface {
   /* Elementen: getallen van het type int
```

```
      Structuur: lineair
      Domein   : alle rijen van ints
   */

   /*
   Rij ();
      PRE  -
      POST - Een nieuwe lege rij is aangemaakt.
   */
   void voegToe (int data);
   /* PRE  -
      POST - data is toegevoegd aan de rij als nieuwe laatste element
                achter het laatste element ten tijde van de PRE-conditie
   */
   // De rest van de RijInterface doet er voor deze opgave niet toe
}
```

en een class die deze interface implementeert

```
class Rij implements RijInterface, Iterator {
   // De implementatie doet er voor deze opgave niet toe.
}
```

Implementeer nu de onderstaande methode.

```
Iterator inOrder (Knoop k)
/* PRE  - k refereert naar de wortel van een binaire zoekboom
   POST - Een object van het type Iterator is geretourneerd.
             Dit object bevat de elementen in inorder volgorde.
*/
```

Beoordelingstabel

opgave	a	b	c	d	e	f	totaal
1	10	5	5	10			30
2	5	10	10				25
3	10	10	15				35
TOTAAL							90

Het eindcijfer E van het gehele tentamen wordt als volgt uit het totaal T verkregen : E = T/10 + 1 .
Het eindcijfer E van het tweede deeltentamen wordt als volgt uit het totaal T verkregen : 9*T/60 + 1 .