Faculteit der Exacte Wetenschappen — tentamen Datastructuren
Vrije Universiteit — 22 december 2000

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

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

Opgave 1

a)
```
Werknemer (int dag, int maand, int jaar, double salaris) {
    geboortedatum = new Datum(dag, maand, jaar);
    this.salaris = salaris;
}
```

b) in de class Datum

```
public boolean equals (Object obj) {
    if (! obj instanceof Datum) {
        return false;
    }

    Datum rhs = (Datum) obj;
    return dag == rhs.dag && maand = rhs.maand && jaar = rhs.jaar;
}
// ontbreken equals Datum: 3 punten
```

in de class Werknemer

```
public boolean equals (Object obj) {
    if (! obj instanceof Werknemer) {
        return false;
    }

    Werknemer rhs = (Werknemer) obj;
    return geboortedatum.equals(rhs.geboortedatum) && salaris ==
rhs.salaris;
}

/*
alle verdere fouten: b.v. geen controle type object,
geen cast, geen aanroep van equals : 1 punt
Ook in het geval dat iemand methodes geefDag(), geefMaand en
geefJaar() o.i.d. toevoegt aan de class Datum, en deze gebruikt om in de
methode equals() van de class Werknemer het vergelijken van twee data
ter plekke uit te programmeren: 1 punt
*/
```

c) in de class Datum

```
Datum (Datum datum) {
    dag = datum.dag;
    maand = datum.maand;
    jaar = datum.jaar;
}
// ontbreken copy constructor Datum: 2 punten
```

in de class Werknemer

```
        Werknemer (Werknemer werknemer) {
            // aanroep copy constructor Datum: 2 punten
            geboortedatum = new Datum(werknemer.geboortedatum);
            salaris = werknemer.salaris;
        }

        // iedere verdere fout: 1 punt
```

d)
```
        Onderzoeker (int d, int m, int j, Specialisme specialisme) {
            super(d, m, j, 0.0);        3 punten
            this.specialisme = new Specialisme(specialisme); 3 punten
        }

        // verkeerde volgorde: 1 punt
        // fouten met salaris niet aanrekenen vanwege fout in tentamen.
```

e)
```
        overloading:  verschillende methodes met dezelfde naam
        polymorfisme: verschillende types voor hetzelfde object
```

Opgave 2.

a)
```
        /* De class Knoop hoeft alleen de default constructor te bevatten.
        class Knoop {
            char data;
            Knoop links,
                  rechts;

            Knoop () {
                char = '';
                links = rechts = null;
            }

        }


        class Boom {
            private Knoop wortel;

            Boom () {
                wortel = null;
            }


            private String toString (Knoop k) {
                if (k == null) {
                    return "";
                }

                return toString(k.links) + k.data + toString(k.rechts);
            }


            String toString () {
                return toString(wortel);
            }

        }
```

b)
```
        minimale aantal:         n
        maximale aantal:         2n
        minimale aantal wortel: 1 (0 wordt ook goedgerekend)
```

```
maximale aantal wortel: 2n
ordening:               monotoon stijgend
aantal subbomen:        x+1
waarden subboom i:      alle kleiner dan x
waarden subboom i+1:    alle groter dan x

                                  6

                 0 3                             9 12
        -2 -1    1 2     4 5             7 8     10 11    13 14
```

Opgave 3.

a) Het probleem dat men met hash-tabellen wil oplossen is een O(c) operatie om data te vinden, waarbij c een constante is.
Dit wordt bereikt door data op een array-index in een array (de hash table) op te slaan. Deze array-index wordt met de hashfunctie berekent uit de key. De key is een unieke identificatie van de data. Een hashfunctie die voor iedere key een unieke array-index geeft heet perfect., Als de hashfunctie niet perfect is leidt dit tot hashen op dezelfde array-index, een zgn. collision. Algoritmes die hier een oplossing voor geven heten een colision-resolution strategies.

```
/* ieder onderwerp dat ontbreekt: -2
   onzin, fouten: -1
*/
```

b)

```
interface HashtableInterface {

    /* Elementen:       entries bestaande uit een key van het type
                        String en data van het type Cloneable
       Structuur:       lineair
       Domein:          willekurig veel entries die iedere String als
                        key en iedere waarde van het type Cloneable als
                        data kunnen hebben.
    */

    /*          N.B. de constructor moet in commentaar staan.
    Hashtable ();
    PRE  -
    POST - De hashtable is geinitialiseerd op de lege hashtable.
    */

    void voegToe (String key, Cloneable data);
    /* PRE - Er is nog data opgeslagen onder key.
       POST- Een kopie van data is onder key in de hashtable opgeslagen.
    */

    Cloneable get (String key) throws Exception;
    /* PRE -
       POST- SUCCES:  Een kopie van de onder key in de hashtable
                      opgeslagen data is geretourneerd.
             FAILURE: Er is geen data onder key opgeslagen in de
                      hashtable.

    */

    boolean isEmpty ();
    /* PRE -
       POST- TRUE : De hashtable is leeg.
             FALSE: De hashtable is niet leeg.
    */
```

```
}

/* puntenverdeling:     elementen, structuur, domein:  ieder 1 punt
                        methode isEmpty(): 1 punt
                        andere methodes: 2 punten
*/
```

c)

```
class Knoop {
    String Key;
    Cloneable data;
    Knoop next;

    Knoop (String s, Cloneable c, Knoop k) {
        key = new String(s);     // sla een kopie op
        data = c.clone();        // sla een kopie op
        next = k;
    }

    // N.B. het opslaan van kopieen kan ook gerealiseerd worden door
    //      die kopieen in de methode voegToe te maken.

}

class Hashtable implements HashtableInterface {

    Knoop[] tabel;
    int aantalEntries;

    static final int AANTAL_INDICES = ......;
    // De exacte waarde doet er niet toe.

    Hashtable () {
        tabel = new Knoop[AANTAL_INDICES]
        aantalEntries = 0;
    }

    boolean isEmpty () {
        return aantalentries == 0;
    }

    void voegToe (String key, Cloneable data) {
        int index = String.hashCode() % AANTAL_INDICES;
        tabel[index] = new Knoop(key, data, tabel[index]);
    }

    Cloneable get (String key) throws Exception {
        int index = String.hashCode() % AANTAL_INDICES;

        for (k = tabel[index]; k != null; k = k.next) {
            if (k.key.equals(key)) {
                return k.data.clone();
            }
        }

        throw new Exception("geen entry met key: '" + key + "'");

    }
}
```