Tentamen – extra del

Förklaringar

Den här delen av tentamen gör möjligt att få betyg högre än E

Utöver den obligatoriska delen, kan studenten göra även den här delen av tentamen. Denna del har sin betydelse bara i fall att studenten har klarat den obligatoriska delen. I så fall kan studenten samla tillräckligt antal poäng på den här delen, och uppnå ett betyg som är högre än E.

Antalet poäng och betyg

Totalt: 20 poäng

För betyget D räcker med: 4 poäng
För betyget C räcker med: 8 poäng
För betyget B räcker med: 14 poäng
För betyget A räcker med: 17 poäng

Uppgifter

Uppgift 1 (4 poäng)

En algoritm, som sorterar en sekvens med jämförbara element, kan beskrivas som nedan.

Sätt en pekare som heter `forst` att peka till första elementet i sekvensen och en pekare `sist` att peka till sista element i sekvensen. Även pekarna `minst` och `aktuell` ska finnas.

Upprepa följande så länge pekaren `forst` ligger till vänster om pekaren `sist`.

Sätt pekaren `minst` att peka till det element som pekaren `forst` pekar på. Sätt pekaren `aktuell` en position efter pekaren `forst`.

Upprepa följande, så länge pekaren `aktuell` inte ligger till höger av pekaren `sist`.

Jämför de element som pekas med pekarna `aktuell` och `minst`. Om det element som `aktuell` pekar på är mindre: sätt pekaren `minst` att peka på det.

Flytta pekaren `aktuell` ett steg fram.

Byt plats på de element som pekarna `minst` och `forst` pekar på.

Flytta pekaren `forst` ett steg fram.

Skapa en metod `sort` som tar emot en heltalsvektor, och sorterar den enligt givna algoritmen.

Uppgift 2 (2 poäng + 1 poäng + 1 poäng)

Klassen `Rectangle2D` och klassen `Ellipse2D` är definierade i paketet `java.awt.geom`. I samma paket finns även flera klasser som heter `Double`. I paketet `java.awt` finns ett gränssnitt som heter `Shape`, och en abstrakt klass som heter `Graphics2D`.

a) Följande sats är korrekt:

```
Rectangle2D    rect = new Rectangle2D.Double (10.0, 20.0, 60.0, 40.0);
```

Vad kan man säga om relationen mellan klassen `Rectangle2D` och klassen `Double`? Ange två slutsatser.

b) Följande kodavsnitt är korrekt:

```
Shape[]    shapes = new Shape[4];
shapes[0] = new Rectangle2D.Double (10.0, 20.0, 60.0, 40.0);
shapes[1] = new Ellipse2D.Double (70.0, 80.0, 40.0, 60.0);
```

Vad kan man säga om relationen mellan klassen `Rectangle2D.Double` och gränssnittet `Shape`, och mellan klassen `Ellipse2D.Double` och gränssnittet `Shape`?

c) I klassen `Graphics2D` finns en metod som heter `draw`, som kan rita figurer av olika typer. Metoden deklareras så här:

```
public abstract void draw (Shape s)
```

Låt `g` vara en referens till ett objekt av en icke-abstrakt subklass till klassen `Graphics2D`. Använd detta objekt och dess metod `draw` för att rita två figurer av olika klasser.

Uppgift 3 (1 poäng + 3 poäng)

Klassen `List` representerar en heltalslista:

```
class List
{
    private static class Node
    {
        public int     value;
        public Node    next;

        public Node (int value)
        {
            this.value = value;
            this.next = null;
        }
    }

    private Node    first = null;

    // add lägger till ett givet heltal till listan
    public void add (int value)
    {
        Node    node = new Node (value);

        if (first == null)
            first = node;
        else
        {
            Node    n = first;
            while (n.next != null)
                n = n.next;
            n.next = node;
        }
    }

    // ytterligare metoder
}
```

a) Skapa en tom lista av typen `List`, och rita den.

b) Skapa en lista av typen `List` som innehåller följande heltal: 1, 4, 5 och 7. Rita den listan.

Uppgift 4 (2 poäng + 2 poäng)

En statisk metod `countDigits` tar emot en teckensekvens av typen `java.lang.CharSequence`, och returnerar antalet siffror i den teckensekvensen.

a) Implementera metoden `countDigits`.

b) Anropa metoden två gånger: använd objekt av olika klasser som argument.

Uppgift 5 (1 poäng + 1 poäng + 2 poäng)

En metod `sok` söker ett givet element i en given sekvens:

```
public static int sok (int[] element, int e)
{
    int    elementPos = -1;

    int    pos = 0;
    while (pos < element.length)
    {
        if (e == element[pos])
        {
            elementPos = pos;
            break;
        }

        pos++;
    }

    return elementPos;
}
```

En elementjämförelse kan betraktas som en elementär operation i den algoritm som används i metoden `sok`. Man kan anta att det finns n ($n \in N, n > 0$) element i sekvensen. Bestäm i så fall algoritmens tidskomplexitet i följande fall:

a) i bästa fall

b) i värsta fall

c) i ett genomsnittligt fall.

För att bestämma komplexiteten i ett genomsnittligt fall, anta att följande villkor är uppfyllda: det sökta elementet finns i sekvensen, alla element i sekvensen är olika, och sannolikheten att elementet finns på en given position är lika för all positioner.

Tidskomplexiteten ska anges med en komplexitetsfunktion, och det ska framgå hur denna komplexitetsfunktion erhålls.