Tentamen

Förklaringar

Utforma noggrant dina svar, kodavsnitt och bilder

Formulera dina svar kortfattat och noggrant.

Koden ska utformas så att det lätt går att följa och förstå den. I vissa situationer kan lämpliga kommentarer bidra till förståelse. Små syntaktiska fel i koden kan eventuellt tolereras. Om delar i ett kodavsnitt inte kan exakt formuleras, kan möjligen en välutformad pseudokod bidra till lösningen. Man ska inte skriva mer kod än som behövs: om bara en metod krävs, behöver inte en hel klass skapas. All programmeringskod ska skrivas i Java.

När en vektor eller ett objekt ritas, ska det klart framgå vilken referens refererar till denna vektor eller detta objekt, och vilka data som finns inuti denna vektor eller detta objekt. När en vektor eller ett objekt innehåller en referens, ska även den refererade resursen (ett objekt eller en vektor) ritas. Man ska förse alla referenser med relevanta beteckningar.

Antalet poäng och betygsgränser

Totalt: 42 poäng
För betyget E räcker : 21 poäng
För betyget D räcker: 25 poäng
För betyget C räcker: 29 poäng
För betyget B räcker: 33 poäng
För betyget A räcker: 37 poäng
.

Uppgifter

Uppgift 1 (3 poäng + 3 poäng)

```
public static void main (String[] args)
{
    int[][]    b = { {0, 0, 0, 1},
                     {0, 0, 1, 1},
                     {0, 1, 1, 1} };

    int[]    u = new int[b.length];
    for (int i = 0; i < u.length; i++)
        for (int j = 0; j < b[i].length; j++)
            u[i] = u[i] + b[i][j];

    int[][]    v = transform (b);
}

public static int[][] transform (int[][] a)
{
    int[][]   w = new int[a.length][];
    for (int i = 0; i < w.length; i++)
    {
        w[i] = new int[a[i].length - 1];
        for (int j = 0; j < w[i].length; j++)
            w[i][j] = a[i][j + 1];
    }

    return w;
}
```

a) Rita den vektor som refereras med referensen `u`.

b) Rita den vektor som refereras med referensen `v`.

Uppgift 2 (3 poäng + 3 poäng + 3 poäng)

Klassen `Point` representerar en punkt i planet:

```
class Point
{
    // punktens koordinater
    private double    x;
    private double    y;

    public Point (double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public double getX ()
    {
        return this.x;
    }

    public double getY ()
    {
        return this.y;
    }

    // distance returnerar avståndet mellan punkten och en given punkt
    public double distance (Point p)
    {
        return Math.sqrt ((p.x - this.x) * (p.x - this.x) +
                          (p.y - this.y) * (p.y - this.y));
     }
}
```

a) En statisk metod, `nearestPoint`, tar emot en vektor med punkter (objekt av typen `Point`) och en punkt (ett objekt av typen `Point`), och returnerar den punkt i vektorn som ligger närmast den givna punkten. Skapa den metoden.

b) En cirkel i planet, vars radie är r och mittpunkten i origo, är given med följande ekvation:

$$x^2 + y^2 = r^2$$

En statisk metod, `internalPoints`, tar emot en vektor med punkter (objekt av typen `Point`) och en cirkels radie, och returnerar de punkter (som en vektor) som ligger inuti den givna cirkeln. Skapa den metoden.

c) Skapa en vektor med punkter (objekt av typen `Point`) och en punkt (objekt av typen `Point`). Fastställ även cirkelns radie.

Använd metoden `nearestPoint` för att bestämma den punkt i vektorn som ligger närmast till givna punkten. Använd sedan metoden `internalPoints` för att bestämma de punkter som ligger inuti cirkeln.

Uppgift 3 (3 poäng + 3 poäng + 3 poäng)

Klassen `Course` representerar en kurs på en högskola:

```
class Course
{
    // kursens nummer och namn
    private int    number;
    private String    name;

    public Course (int number, String name)
    {
        this.number = number;
        this.name = name;
    }
```

```
    public String toString ()
    {
        return "<" + this.number + ", " + this.name + ">";
    }

    // equals returnerar true om kursen är lika
    // med en given kurs, annars false.
    public boolean equals (Course course)
    {
        return this.number == course.number
           &&  this.name.equals(course.name);
    }
}
```

Klassen `CourseList` representerar en lista med kurser:

```
class CourseList
{
    private static final int    CAPACITY = 5;

    // kurser
    private Course[]    courses;

    // antalet kurser
    private int    courseCount;


    public CourseList ()
    {
        courses = new Course[CAPACITY];
        courseCount = 0;
    }

    // add lägger till en kurs i listan
    public void add (Course course)
    {
        courses[courseCount] = course;
        courseCount++;
    }

    // toString returnerar listans strängrepresentation
    // koden saknas här

    // remove tar bort en given kurs från listan.
    // Om kursen inte finns i listan, uppdateras inte listan.
    // koden saknas här
}
```

En instans av klassen `CourseList` skapas och används så här:

```
CourseList    list = new CourseList ();
list.add (new Course (10, "Algebra"));
list.add (new Course (20, "Algorithms"));
list.add (new Course (30, "Electronics"));
list.add (new Course (40, "Physics"));
list.add (new Course (50, "Programming"));
System.out.println (list);

list.remove (new Course (40, "Physics"));
list.remove (new Course (30, "Electronics"));
System.out.println (list);
```

När detta kodavsnitt exekveras, skapas följande utskrift:

```
<10, Algebra>
<20, Algorithms>
<30, Electronics>
```

```
<40, Physics>
<50, Programming>

<10, Algebra>
<20, Algorithms>
<50, Programming>
```

a) Implementera metoden `toString`.

b) Implementera metoden `remove`.

c) Hur ser ut det objekt som refereras med referensen `list`, när kodavsnittet har exekverats? Rita objektet.

Uppgift 4 (4 poäng + 2 poäng + 3 poäng)

Gränssnittet `SizeComparable`, och klasserna `Rectangle`, `CharSequence` och `Selector`, definieras på följande vis:

```
public interface SizeComparable<T>
{
    // sizeCompare jämför det här objektet med ett givet objekt
    // enligt deras storlek.
    // Metoden returnerar -1 om det här objektet är mindre,
    // 0 om objekten är likadana, och 1 om det här objektet
    // är större.
    int sizeCompare (T o);
}


public class Rectangle implements SizeComparable<Rectangle>
{
    // rektangelns sidor
    private double    width;
    private double    height;

    public Rectangle (double width, double height)
    {
        this.width = width;
        this.height = height;
    }

    // sizeCompare jämför den här rektangeln med en given
    // rektangel enligt deras area.
    // Metoden returnerar -1 om den här rektangeln är mindre,
    // 0 om rektanglarna är likadana, och 1 om den här rektangeln
    // är större.
    // koden saknas här
}


class CharSequence implements SizeComparable<CharSequence>
{
    public static final int    CAPACITY = 1000;

    // antalet tecken i teckensekvensen
    private int    charCount;

    // tecken i teckensekvensen
    private char[]    characters = new char[CAPACITY];

    public CharSequence (char... characters)
    {
        for (int pos = 0; pos < characters.length; pos++)
            this.characters[pos] = characters[pos];
        charCount = characters.length;
    }

    public String toString ()
```

```
    {
        String    s = "";
        for (int pos = 0; pos < charCount; pos++)
            s = s + characters[pos];

        return s;
    }

    // sizeCompare jämför den här teckensekvensen med en given
    // teckensekvens enligt deras längd (antalet tecken).
    // Metoden returnerar -1 om den här teckensekvensen är mindre,
    // 0 om teckensekvenserna är likadana, och 1 om den här
    // teckensekvensen är större.
    // koden saknas här
}


class Selector
{
    public static <T extends SizeComparable<T>> T oneOfTwo (T object1, T object2)
    {
        T    selected = object1;
        if (object2.sizeCompare (object1) > 0)
            selected = object2;

        return selected;
    }

    public static <T extends SizeComparable<T>> T oneOfMany (T[] objects)
    {
        T    selected = objects[0];
        for (int pos = 1; pos < objects.length; pos++)
            if (objects[pos].sizeCompare (selected) > 0)
                selected = objects[pos];

        return selected;
     }
}
```

a) Komplettera klasserna `Rectangle` och `CharSequence` – skriv den kod som saknas.

b) Anropa metoden `oneOfTwo` i samband med objekt av typen `Rectangle`.

c) Objekt av klassen `CharSequence` användas på följande vis:

```
CharSequence[]    sequences = {
                              new CharSequence ('1', '1', '1'),
                              new CharSequence ('1', '+', '2', '=', '3'),
                              new CharSequence ('a', 'b', 'c', 'd'),
                              new CharSequence ('9', '9', '9'),
                              new CharSequence ('0', '0', '0', '0') };
CharSequence    seq = Selector.oneOfMany (sequences);
System.out.println (seq);
```

Vilken utskrift skapas när den här kodsekvensen utförs?

Uppgift 5 (4 poäng + 5 poäng)

En algoritm, som sorterar en sekvens med element, kan illustreras som nedan:

```
[H, B, D, G, F, E, A, C]

[A, B, D, G, F, E, H, C]
[A, B, D, G, F, E, H, C]
[A, B, C, G, F, E, H, D]
[A, B, C, D, F, E, H, G]
[A, B, C, D, E, F, H, G]
[A, B, C, D, E, F, H, G]
[A, B, C, D, E, F, G, H]
```

```
[A, B, C, D, E, F, G, H]
```

a) Låt n beteckna antalet element som sorteras.

Bestäm tidskomplexiteten för algoritmen när det gäller antalet elementjämförelser. Kategorisera motsvarande komplexitetsfunktion: till vilken Θ–mängd tillhör den?

Bestäm även tidskomplexiteten för algoritmen i värsta fall när det gäller antalet elementutbyten. Till vilken Θ–mängd tillhör motsvarande komplexitetsfunktion?

b) Skapa en metod `sort` som tar emot en vektor med teckensträngar, och sorterar den enligt givna algoritmen.