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.

Tillåtna hjälpmedel: linjal

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)

```
int      n = 30;
int[]    u = new int[6];
int      pos = 0;
for (int k = 2; k <= n / 2; k++)
    if (n % k == 0)
        u[pos++] = k;

int[][]    v = new int[3][4];
for (int i = 0; i < 12; i++)
    v[i / 4][i % 4] = i;
```

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 `Car` representerar en bil.

```
class Car
{
    // typ och pris av bilen
    private String    type;
    private double    price;
```

```
    public Car (String type, double price)
    {
        this.type = type;
        this.price = price;
    }

    public String getType ()
    {
        return type;
    }

    public double getPrice ()
    {
        return price;
    }
}
```

a) En statisk metod, `findCar`, tar emot en vektor med bilar (objekt av typen `Car`) , en biltyp (objekt av typen `String`) och ett pris. Metoden returnerar en av de bilar i vektorn som är av den givna typen, och vars pris inte överstiger givna priset. Om det inte finns sådana bilar i vektorn, returnerar metoden `null`. Skapa den metoden.

b) En statisk metod, `selectCars`, tar emot en vektor med bilar (objekt av typen `Car`) och en biltyp (objekt av typen `String`), och returnerar de bilar (som en vektor) som är av givna typen. Skapa den metoden.

c) Skapa en vektor med bilar (objekt av typen `Car`). Anropa sedan metoderna `findCar` och `selectCars` i samband med den vektorn.

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

Klassen `RationalNumber` representerar ett rationellt tal.

```
class RationalNumber
{
    // täljaren och nämnaren
    private int    p;
    private int    q;

    public RationalNumber (int p, int q)
    {
        if (q == 0) throw new java.lang.IllegalArgumentException ("zero denominator");

        this.p = p;
        this.q = q;
    }


    // toString returnerar strängrepresentation av det här rationella talet
    // koden saknas här


    // equals returnerar true om det här rationella talet är lika med ett givet
    // rationellt tal. Annars returnerar metoden false.
    // koden saknas här


    // add lägger till ett givet rationellt tal till det här rationella talet.
    // koden saknas här
}
```

Klassen `RationalNumber` används på följande vis:

```
RationalNumber[]    numbers = { new RationalNumber (2, 3),
                                new RationalNumber (4, 6),
```

```
                                new RationalNumber (-2, -3),
                                new RationalNumber (2, -3) };
for (int pos = 0; pos < numbers.length; pos++)
    System.out.print (numbers[pos] + "  ");
System.out.println ("\n");

for (int pos = 1; pos < numbers.length; pos++)
    System.out.print (numbers[0].equals (numbers[pos]) + "  ");
System.out.println ();

numbers[0].add (numbers[2]);
System.out.print (numbers[0] + "  ");
numbers[2].add (numbers[3]);
System.out.println (numbers[2]);
```

När detta kodavsnitt exekveras, skapas följande utskrift:

```
2/3  4/6  2/3  -2/3

true  true  false
12/9  0/9
```

a) Implementera metoden `toString`.

b) Implementera metoden `equals`.

c) Implementera metoden `add`.

Matematik

Ett reellt tal är rationellt om och endast om det kan representeras som `a/b`, där `a` och `b` är heltal, `b ≠ 0`.

Likhet mellan två rationella tal: `a/b = c/d` om och endast om `ad = bc`.

Addition av två rationella tal: `a/b + c/d = (ad + bc)/(bd)`.

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

Klassen `IntList` representerar en lista med heltal.

```
class IntList
{
    // Node representerar en nod
    private static class Node
    {
        public int     number;
        public Node    next;

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

    // referens till listans första nod
    private Node    first = null;

    public IntList (int[] numbers)
    {
        if (numbers != null)
        {
            Node    n = new Node (numbers[0]);
            first = n;
```

```
            for (int pos = 1; pos < numbers.length; pos++)
            {
                n.next = new Node (numbers[pos]);
                n = n.next;
            }
        }
    }

    public String toString ()
    {
        StringBuilder    sb = new StringBuilder ("[");
        Node    n = first;
        while (n != null)
        {
            sb.append (n.number);
            if (n.next != null)
                sb.append (", ");
            n = n.next;
        }
        sb.append ("]");

        return sb.toString ();
    }

    // getLast returnerar det sista heltalet i listan.
    // Om listan är tom, kastas ett undantag av typen
    // java.util.NoSuchElementException.
    // koden saknas här

    // removeFirst tar bort det första heltalet i listan.
    // Om listan är tom, kastas ett undantag av typen
    // java.util.NoSuchElementException.
    // koden saknas här
}
```

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

```
int[]      numbers = {10, 20, 30, 40};
IntList    list = new IntList (numbers);  // (1)
System.out.println (list);

System.out.println (list.getLast ());
list.removeFirst ();
System.out.println (list);
```

När detta kodavsnitt exekveras, skapas följande utskrift:

```
[10, 20, 30, 40]
40
[20, 30, 40]
```

a) Hur ser det objekt ut som refereras med referensen `list` när satsen (1) har exekverats? Rita objektet.

b) Implementera metoden `getLast`.

c) Implementera metoden `removeFirst`.

Uppgift 5 (5 poäng + 4 poäng)

En algoritm sorterar en sekvens med element som kan jämföras med operatorn *mindre* (<). Algoritmen används nedan för att sortera en sekvens med heltal.

```
public static void sort (int[] sequence)
{
    int    e = 0;
```

```
    int    holePos = 0;
    for (int pos = 1; pos < sequence.length; pos++)
    {
        e = sequence[pos];
        holePos = pos;
        while (holePos > 0  &&  e < sequence[holePos - 1])
        {
            sequence[holePos] = sequence[holePos - 1];
            holePos--;
        }
        sequence[holePos] = e;

        System.out.println (java.util.Arrays.toString (sequence));
    }
}
```

a) Metoden `sort` anropas så här:

```
int[]   sequence = {8, 2, 7, 1, 5, 6, 3, 4};
System.out.println (java.util.Arrays.toString (sequence));
System.out.println ();
sort (sequence);
```

Vilken utskrift skapas när detta kodavsnitt utförs?

b) Låt n beteckna antalet element som sorteras. Bestäm i så fall tidskomplexiteten för algoritmen när det gäller antalet jämförelser - både i bästa fall och i värsta fall. Kategorisera motsvarande komplexitetsfunktioner: till vilken Θ–mängd tillhör dem?