Exam

Explanations

Carefully formulate your answers, code, and images

Formulate your answers precise and to the point.

Code shall be written so that it is easy to follow and understand. In some situations suitable comments can contribute to understanding. Small syntactical errors can be tolerated. If some parts of code cannot be exactly produced, it is possible that well-formed pseudocode may provide a solution. Do not write more code than necessary; if just a method is requested there is no need to create a whole class. All program code is to be written in Java.

When an array (vector) or an object is drawn, it must be clearly visible by which reference the array or object is referred to, and what data is located inside it. When an array or object contains a reference, the resource that is referred to (an object or array) shall be drawn. All references shall have relevant labels.

Points and grading

In total: 42 points
For grade E at least: 21 points
For grade D at least: 25 points
For grade C at least: 29 points
For grade B at least: 33 points
For grade A at least: 37 points
.

Tasks

Task 1 (3 points + 3 points)

```
int[][]    b = { {1, 2, 3},
                 {4, 5, 6},
                 {7, 8, 9} };

int[]    u = new int[b.length];
int    k = b[0].length - 1;
for (int i = 0; i < u.length; i++)
{
     u[i] = b[i][k];
     k = k - 1;
}


int[][]    v = new int[b.length][b[0].length];
for (int i = 0; i < v.length; i++)
    for (int j = 0; j < v[i].length; j++)
        v[i][j] = ((i + j) % 2 == 1)? 0 : b[i][j];
```

a) Draw the array referred to by reference `u`.

b) Draw the array referred to by reference `v`.

Task 2 (3 points + 3 points + 3 points)

The class `RationalNumber` represents a rational number:

```
class RationalNumber
{
```

```
    // numerator and denominator
    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;
    }


    public String toString ()
    {
        String    s = "" + Math.abs (this.p) + "/" + Math.abs (this.q);
        if (this.p * this.q < 0)
            s = "-" + s;

        return s;
    }


    // sum returns the sum of this rational number
    // and a given rational number
    public RationalNumber sum (RationalNumber r)
    {
        int    sumP = this.p * r.q + r.p * this.q;
        int    sumQ = this.q * r.q;
        RationalNumber    sum = new RationalNumber (sumP, sumQ);

        return  sum;
    }


    // lessThan returns true if this rational number
    // is less than a given rational number, otherwise
    // it returns false.
    public boolean lessThan (RationalNumber r)
    {
        int    differenceP = this.p * r.q - r.p * this.q;
        int    differenceQ = this.q * r.q;
        boolean    isLessThan = differenceP * differenceQ < 0;

        return isLessThan;
    }
}
```

a) A static method, `totalSum`, accepts an array of rational numbers (objects of type `RationalNumber`), and returns the total sum of these numbers (as an object of type `RationalNumber`). Create that method.

b) A static method, `maxNumber`, accepts an array of rational numbers (objects of type `RationalNumber`), and returns the greatest number in the array. Create that method.

c) Create an array of rational numbers (objects of type `RationalNumber`), and call the methods `totalSum` and `maxNumber` in some way.

Task 3 (3 points + 3 points + 3 points)

The class `Element` represents a chemical element:

```
class Element
{
    private String    name;
    private String    symbol;
```

```
    public Element (String name, String symbol)
    {
        this.name = name;
        this.symbol = symbol;
    }

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

The class `ElementList` represents a list of chemical elements:

```
class ElementList
{
    public static final int    CAPACITY = 4;

    private Element[]    elements;
    private int    numberOfElements = 0;


    public ElementList (Element... elements)
    {
        if (elements.length > CAPACITY)
            throw new java.lang.IllegalArgumentException ("not enough capacity");

        this.elements = new Element[CAPACITY];
        for (int pos = 0; pos < elements.length; pos++)
            this.elements[pos] = elements[pos];
        numberOfElements = elements.length;
    }

    public String toString ()
    {
        StringBuilder    sb = new StringBuilder ("{");
        for (int pos = 0; pos < numberOfElements; pos++)
            sb.append (elements[pos]);
        sb.append ("}");

        return sb.toString ();
    }


    public void addFirst (Element element)
    // the code is missing here

    public Element takeFirst ()
    // the code is missing here
}
```

An instance of class `ElementList` is created and used like this:

```
Element    e1 = new Element ("Zinc", "Zn");
Element    e2 = new Element ("Iron", "Fe");
Element    e3 = new Element ("Magnesium", "Mg");
Element    e4 = new Element ("Calcium", "Ca");


ElementList    list = new ElementList (e1, e2);
System.out.println (list);
System.out.println ();

list.addFirst (e3);
System.out.println (list);
list.addFirst (e4);
System.out.println (list);
```

```
Element    e = list.takeFirst ();
System.out.println (list);
e = list.takeFirst ();
System.out.println (list);
```

When this code fragment is executed, the following output is generated:

```
{<Zinc, Zn><Iron, Fe>}

{<Magnesium, Mg><Zinc, Zn><Iron, Fe>}
{<Calcium, Ca><Magnesium, Mg><Zinc, Zn><Iron, Fe>}
{<Magnesium, Mg><Zinc, Zn><Iron, Fe>}
{<Zinc, Zn><Iron, Fe>}
```

a) What does the object referred to by reference `list` look like when this code sequence has executed? Draw the object.

b) Implement the method `addFirst`.

c) Implement the method `takeFirst`.

Task 4 (3 points + 3 points + 3 points)

The class `Point` represents a planar point:

```
class Point
{
    // the coordinates of the point
    private double    x;
    private double    y;

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

    public String toString ()
    {
        return "(" + x + ", " + y + ")";
    }
}
```

The class `PointN` represents a named planar point:

```
class PointN extends Point
{
    // the name of the point
    private String    name;


    // PointN creates a named point from its
    // name and coordinates
    public PointN (String name, double x, double y)
                                throws NameFormatException
    {
        // a statement is missing here

        char    c = name.charAt (0);
        if (!Character.isLetter (c))
            throw new NameFormatException ("wrong name " + name);

        // a statement is missing here
    }


    public String toString ()
```

```
    // the code is missing here
}
```

The classes `Point` and `PointN` are used like this:

```
Point    p1 = new Point (1, 2);
System.out.println (p1);
Point    p2 = new PointN ("A", 3, 4);
System.out.println (p2);
```

When this code fragment is executed, the following output is generated:

```
(1.0, 2.0)
A (3.0, 4.0)
```

a) Present the two statements missing from the constructor in class `PointN`.

b) Create the method `toString` in class `PointN`.

c) Create the class `NameFormatException`.

Task 5 (3 points + 3 points + 3 points)

An algorithm sorts a sequence of elements that can be compared with the operator *greater* (>). The algorithm is used below to sort a sequence of integers:

```
public static void sort (int[] sequence)
{
    int    lastPos = sequence.length - 1;
    int    maxPos = 0;
    int    i = 0;
    for (int pos = lastPos; pos > 0; pos--)
    {
        maxPos = 0;
        for (int p = 1; p <= pos; p++)
            if (sequence[p] > sequence[maxPos])
                maxPos = p;

        if (maxPos != pos)
        {
            i = sequence[pos];
            sequence[pos] = sequence[maxPos];
            sequence[maxPos] = i;
        }

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

a) The method `sort` is called like this:

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

sort (seq);
System.out.println ();
System.out.println (java.util.Arrays.toString (seq));
```

What output is generated when this code fragment is executed?

b) Specify the preconditions and postconditions of the algorithm. Represent steps in the algorithm in the form of pseudocode.

c) Let n denote the number of elements being sorted.

Determine the time complexity of the algorithm in the worst case, in terms of the number of element exchanges. To which Θ –set does the corresponding complexity function belong?