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)

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

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

Task 2 (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 double getX ()
    {
        return this.x;
    }

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

    // distance returns the distance between this point and a given point
    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) A static method, `nearestPoint`, accepts an array of points (objects of type `Point`) and one point (an object of type `Point`), and returns that point in the array which is closest to the given point. Create that method.

b) A circle in the plane, with radius r and midpoint at the origin, is given by the following equation:

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

A static method, `internalPoints`, accepts an array of points (objects of type `Point`) and the radius of a circle, and returns those points (in an array) that are inside the given circle. Create that method.

c) Create an array of points (objects of type `Point`) and a point (an object of type `Point`). Establish also the radius of the circle.

Use the method `nearestPoint` to determine the point in the array that is closest to the given point. Then use the method `internalPoints` to determine the points that are inside the circle.

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

The class `Course` represents a university course:

```
class Course
{
    // course number and name
    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 returns true if the course is equal
    // to a given course, and false otherwise.
    public boolean equals (Course course)
    {
        return this.number == course.number
           &&  this.name.equals(course.name);
    }
}
```

Tha class `CourseList` represents a list of courses:

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

    // courses
    private Course[]    courses;

    // the number of courses
    private int    courseCount;


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

    // add adds a course to the list
    public void add (Course course)
    {
        courses[courseCount] = course;
        courseCount++;
    }

    // toString returns a string representation of the list
    // code is missing here

    // remove removes a given course from the list.
    // If the course is not in the list, the list is not updated.
    // code is missing here
}
```

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

```
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);
```

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

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

```
<30, Electronics>
<40, Physics>
<50, Programming>

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

a) Implement the method `toString`.

b) Implement the method `remove`.

c) What does the object referred to by reference `list`, look like when the code fragment has been executed? Draw the object.

Task 4 (4 points + 2 points + 3 points)

The interface `SizeComparable`, and the classes `Rectangle`, `CharSequence`, and `Selector`, are defined like this:

```
public interface SizeComparable<T>
{
    // sizeCompare compares this object with a given object
    // according to their size.
    // The method returns -1 if this object is smaller,
    // 0 if the objects are equal, and 1 if this object
    // is bigger.
    int sizeCompare (T o);
}


public class Rectangle implements SizeComparable<Rectangle>
{
    // the sides of the rectangle
    private double    width;
    private double    height;

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

    // sizeCompare compares this rectangle with a given
    // rectangle according to their area.
    // The method returns -1 if this rectangle is smaller,
    // 0 if the rectangles are equal, and 1 if this rectangle
    // is greater.
    // code is missing here
}


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

    // the number of characters in the character sequence
    private int    charCount;

    // characters in the character sequence
    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 compares this character sequence with a given
    // character sequence according to their length (the number of characters).
    // The method returns -1 if this character sequence is shorter,
    // 0 if the character sequences are equal, and 1 if this
    // character sequence is longer.
    // code is missing here
}


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) Make complete the classes `Rectangle` and `CharSequence` – write the missing code.

b) Call the method `oneOfTwo` with objects of type `Rectangle`.

c) Objects of class `CharSequence` are used in the following way:

```
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);
```

Which output is generated when this code fragment is executed?

Task 5 (4 points + 5 points)

An algorithm, that sorts a sequence of elements, can be illustrated as below:

```
[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) Let n denote the number of elements that are being sorted.

Determine the time complexity of the algorithm in terms of the number of element comparisons. Categorize the corresponding complexity function: to which Θ–set does it belong?

Also 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?

b) Create a method `sort` that accepts an array of character strings, and sorts it according to the given algorithm.