Exam – required part

Explanations

This part of the exam is required and sufficient for the grade E

To pass the exam, the student must pass this part of the exam. If only this part is passed, the exam grade is E. To achieve a higher grade, the student must earn a sufficient number of points from the extra part of the exam. Depending on this extra number of points, the exam grade can be D, C, B, or A.

To pass this part of the exam

To pass this required part of the exam the student must achieve at least two thirds of the total number of points in this part. These points do not contribute towards the higher grades. The highest grade in this required part is E.

Number of points

Total: 33 points
For the grade E, the requirement is at least: 22 points

Carefully formulate your answers, code, and drawings

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 what data is located inside the array or object. 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.

Tasks

Task 1 (2 points + 2 points)

```
int[]    u = new int[6];
int    m = 1;
for (int pos = 0; pos < u.length; pos++)
{
    u[pos] = m;
    m = (m + 1) % 5;
}

int[][]    v = new int[2][4];
int    n = 1;
for (int row = 0; row < v.length; row++)
{
    for (int column = 0; column < v[row].length; column++)
        v[row][column] = n++;
}
```

a) Draw the array referred to by reference u.

b) Draw the array referred to by reference v.

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

The following code fragment is executed:

```
int[]    u = new int[4];
// u[0] = 12 / (u.length - 4);  // (1)
// u[u.length] = 12;            // (2)

StringBuilder[]    v = new StringBuilder[5];
// v[0].append ("a");           // (3)
```

a) What happens if statement `(1)`is included? Why?

b) What happens if statement `(2)`is included? Why?

c) What happens if statement `(3)`is included? Why?

Task 3 (4 points + 3 points)

A class `Integer` represents an integer number:

```
public class Integer
{
    // the value of the integer
    private int    value;

    public Integer (int value)
    {
        this.value = value;
    }

    // lessThan returns true if this integer is smaller than
    // the given integer, and false otherwise.
    public boolean lessThan (Integer p)
    {
        return this.value < p.value;
    }

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

A static method, `filter`, accepts an array of integers and one integer. The method returns another array, which only contains those integers from the given array that are less than the given integer.

```
public static Integer[] filter (Integer[] v, Integer p)
{
    // code is missing here
}
```

a) Create the method `filter`.

b) Create an array of integers and one integer, and call the method `filter` with them. Show the array that is returned. The printout shall have the following form:

```
<40><50><10>
```

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

The class`Point` represents a planar point:

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

    public Point (double x, double y)
    {
```

```
        this.x = x;
        this.y = y;
    }

    public double distance (Point p)
    {
        return Math.sqrt ((p.x - this.x) * (p.x - this.x)
                        + (p.y - this.y) * (p.y - this.y));
    }

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

The class `Line` represents a planar line segment:

```
class Line
{
    // the start and end points of the line segment
    private Point    startPoint;
    private Point    endPoint;

    // Line creates a line segment: the start and end points are given.
    public Line (Point startPoint, Point endPoint)
    {
        this.startPoint = new Point (startPoint.x, startPoint.y);
        this.endPoint = new Point (endPoint.x, endPoint.y);
     }

    // toString returns the string representation of the line segment
    // code is missing here

    // length returns the length of the line segment
    // code is missing here

    // midPoint returns the point in the middle of the line segment
    // code is missing here
```

A line segment is created and used like this:

```
Point    p1 = new Point (1, 1);
Point    p2 = new Point (4, 5);
Line     line = new Line (p1, p2);

System.out.println (line);
System.out.println (line.length ());
System.out.println (line.midPoint ());
```

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

```
{(1.0, 1.0), (4.0, 5.0)}
5.0
(2.5, 3.0)
```

a) Implement the method `toString`.

b) Implement the method `length`.

c) Implement the method `midPoint`.

d) Draw the object referred to by the reference `line`.

Task 5 (4 points + 2 points + 2 points)

The interface `CharSequence` represents a character sequence:

```
public interface CharSequence
{
    // charAt returns the character located at a given index in the sequence
    char charAt (int index);

    // length returns the length of the character sequence
    int length ();
}
```

The class `ImmutableCharSequence` represents an immutable character sequence:

```
class ImmutableCharSequence implements CharSequence
{
    // characters in the character sequence
    private char[]    chars;

    public ImmutableCharSequence (char[] chars)
    {
        this.chars = new char[chars.length];
        for (int pos = 0; pos < chars.length; pos++)
            this.chars[pos] = chars[pos];
    }

    // startsWith returns true if the character sequence begins with a given character,
    // and false otherwise
    public boolean startsWith (char c)
    {
        boolean    b = false;
        if (this.chars.length > 0  &&  c == this.chars[0])
            b = true;

        return b;
    }

    public String toString ()
    {
        return new String (this.chars);
    }

    // code is missing here
}
```

A character sequence is created and used like this:

```
char[]    letters = {'a', 'b', 'c', 'd', 'e'};
CharSequence    cs = new ImmutableCharSequence (letters);
System.out.println (cs);

char   c = cs.charAt (4);
int    len = cs.length ();
System.out.println (c + " " + len);
// boolean    b = cs.startsWith ('a');   // (1)
```

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

```
abcde
e 5
```

a) Make complete the class `ImmutableCharSequence`: write the missing code.

b) What happens when statement `(1)` is included? Why?

c) Draw the object referred to by reference `cs`