Exam ID1018 19 March 2014 — optional part

ID1018

July 9, 2014

Exam — optional part

Explanations

This part of the exam offers the possibility of a grade higher than E

In addition to the required part, the student may also answer this optional part of the exam. This part is only relevant when the student has passed the required part. If so, the student may collect points on this optional part and achieve a grade higher than E.

The number of points and grades

In total: 20 points
For a grade of D: at least 4 points
For a grade of C: at least 8 points
For a grade of B: at least 14 points
For a grade of A: at least 17 points

Tasks

Task 1 (4 points)

An algorithm that sorts a sequence of comparable elements can be described as follows:

> Set a pointer called *first* to the first element in the sequence, and a pointer called *last* to the last element in the sequence. Also prepare pointers *least* and *current*.
>
> As long as the pointer *first* is left of pointer *last*, repeat:
>
> Set the pointer *least* to the element *first* is pointing at. Set the pointer *current* one position after *first*.
>
> *Repeat the following as long as pointer* current *is not right of pointer* last*:*
>
> *Compare the elements pointed to by* current *and* least*. If the element pointed to by* current *is the lesser one, set* least *to point at it.*
>
> *Advance pointer* current *one step.*
>
> Exchange the elements pointed to by *least* and *first*.
>
> Advance pointer *least* one step.

Create a method `sort` that accepts an integer vector and sorts it according to the given algorithm.

Task 2 (2 points + 1 point + 1 point)

The classes `Rectangle2D` and `Ellipse2D` are defined in the package `java.awt.geom`. In the same package there are also several classes called `Double`. In the package `java.awt` there is an interface called `Shape`, and an abstract class called `Graphics2D`.

a) The following statement is correct:

```
Rectangle2D rect = new Rectangle2D.Double(10.0, 20.0, 60.0, 40.0);
```

What can be said about the relation between the class `Rectangle2D` and the class `Double`? Specify two conclusions.

b) The following code fragment is correct:

```
Shape[]    shapes = new Shape[4];
shapes[0] = new Rectangle2D.Double (10.0, 20.0, 60.0, 40.0);
shapes[1] = new Ellipse2D.Double (70.0, 80.0, 40.0, 60.0);
```

What can be said about the relation between class `Rectangle2D.Double` and the interface `Shape`, and between class `Ellipse2D.Double` and the interface `Shape`?

c) In class `Graphics2D` there is a method called `draw`, that can draw figures of various sizes. The method is declared like this:

```
public abstract void draw (Shape s)
```

Let `g` be a reference to an instance of a non-abstract subclass of class `Graphics2D`. Use this object instance and its `draw` method to draw two figures of different classes.

Task 3 (1 point + 3 points)

The class `List` represents an integer list:

```
class List
{
  private static class Node
  {
    public int     value;
    public Node    next;

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

  private Node    first = null;

  // add adds a given integer to the list
  public void add (int value)
  {
    Node    node = new Node (value);

    if (first == null)
      first = node;
    else
    {
      Node    n = first;
      while (n.next != null)
        n = n.next;
      n.next = node;
    }
  }

  // further methods
}
```

a) Create an empty list of type `List`, and draw it.

b) Create a list of type `List` that contains the integers: 1, 4, 5, and 7. Draw that list.

Task 4 (2 points + 2 points)

A static method `countDigits` accepts a character sequence of the type `java.lang.CharSequence` and returns the number of digits in the sequence.

a) Implement the method `countDigits`.

b) Call the method twice: use objects from different classes as arguments.

Task 5 (1 point + 1 point + 2 points)

A method `search` searches for an element in a sequence:

```
public static int search (int[] element, int e)
{
  int    elementPos = -1;

  int    pos = 0;
  while (pos < element.length)
  {
    if (e == element[pos])
    {
      elementPos = pos;
      break;
    }

    pos++;
  }

  return elementPos;
}
```

An element comparison can be viewed as a fundamental operation in the algorithm used in method `search`. Assume that there are $n(n \in N, n > 0)$ elements in the sequence. Determine the algorithm's time complexity in the following cases:

a) best case

b) worst case

c) average case

To determine the complexity of an average case, assume that the following conditions hold: the element searched for is present in the sequence, all elements in the sequence are different, and the probability that the wanted element is found in a particular position is the same for all positions.

The time complexity is to be shown with a complexity function, and it shall be visible how this function was arrived at.