Übungsgruppe 3; Zweiergruppe 23
274196 Lucas Brutschy & 273784 Philipp Fischer

Übung Programmierung
Blatt 5

Aufgabe 1 – Speicherverwaltung

Teil 1 (Stack und Heap)

a)

Die Strategie wird "Last In First Out" oder auch LIFO genannt, da zuerst das Element vom Stack gelöscht wird, welches als letztes eingefügt wurde.

b)

Diese Referenzen werden im Stack gespeichert und zeigen auf Speicherblöcke im Heap.

c)

Der "garbage collector" verwaltet den Heap und gibt dort Speicherbereiche frei, zu denen keine Referenz mehr im Java-Programm existiert.

d)

Nein, Arrays können nicht verkürzt werden, aber man kann eine Verkürzung nachbilden:

```
int[] lang = {1,2,3,4,5};
int[] kurz = new int[3];
System.arraycopy(lang, 0, kurz, 0, 3);
```

Es wird ein neues Array erstellt, welches kürzer ist. Mit `arraycopy` werden nur die ersten `m` Elemente in das neue, kürzere Array kopiert.

Teil 2 (Deklaration / Initialisierung / Allokation)

a)

Nein, es kann nur Speicher im Heap allokiert werden.

b)

Nein, bei diesem Beispiel greift die automatische Null-Initialisierung:

```
public class TestClass {
   static int test;

   public static void main(String[] args) {
      System.out.print(test);
   }
}
```

c)

Korrekt. Jede Variable, somit auch ein Array muss vor jeglicher Verwendung deklariert werden. Man kann diese beiden Schritte jedoch auch in einer Anweisung zusammenfassen:

```
int[] liste = {1,2,3,4};
```

d)

Falsch, der "garbage collector" sorgt selbst dafür, dass nicht benötigte Objekte deallokiert werden. Es muss nicht extra eine Referenz übergeben werden.

Exercise2.java

```java
/**
 * SHEET 5 - EXERCISE 2
 * substring search algorithm
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 19.11.2006
 * last update: 05.11.06
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
 */
public class Exercise2 {
    /**
     * Finds a substring in a given string
     * @param keywordArg the search keyword
     * @param searchTextArg the search context
     * @return The position of the found substring (0-searchText.length-1). If the string was not found, -1 is returned.
     */
    public static int findText (String keywordArg, String searchTextArg) {
        char keyword[] = keywordArg.toCharArray(), searchText[]=searchTextArg.toCharArray();
        for(int i=0;i+keyword.length<=searchText.length;i++) {
            // Let searchText[i] be the first character of a string with the length keyword.length
            // Compare this substring of searchText and the keyword
            for(int j=0;keyword[j]==searchText[i+j];j++) {
                if(j==keyword.length-1) {
                    return i;
                }
            }
        }
        return -1;
    }

    /**
     * This is the main function of the program
     * @param args Expects 2 arguments. The search keyword and the search context.
     */
    public static void main (String[] args) {
        if(args.length==2) {
            int foundPosition = findText(args[0],args[1]);
            if(foundPosition>=0) {
                System.out.println("found first occurence at position "+foundPosition);
            } else {
                System.out.println("input string does not contain '"+args[0]+"'");
            }
        } else {
            System.out.println("this program expects 2 arguments (the search keyword and the search context)");
        }
    }
}

```

GraphicsContext.java

```java
/**
 * SHEET 5 - EXERCISE 3a,3b,3c
 * GraphicsContext class with Bresenham and polygon support
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 26.11.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class GraphicsContext {

    private int canvasWidth,canvasHeight;
    private char brush,background,frame;
    private char[][] canvas;

    /**
     * Initializes the internal canvas. This method must be run, before
     * other drawing methods may be invoked.
     * @param canvasWidth the width of the canvas
     * @param canvasHeight the height of the canvas
     */
    public void init (int canvasWidth,int canvasHeight) {
        if(canvasWidth<1||canvasHeight<1) return;

        canvas = new char[canvasWidth][canvasHeight];
        this.canvasWidth = canvasWidth;
        this.canvasHeight = canvasHeight;
        brush = '#';
        frame = '*';
        background = ' ';
        clearCanvas();
    }

    /**
     * setBrush specifies the character to be used for drawing into the canvas
     * @param brush the character to be used for drawing into the canvas
     */
    public void setBrush (char brush) {
        this.brush = brush;
    }

    /**
     * setBackground specifies the character to be used for clearing the canvas
     * @param background the character to be used for clearing the canvas
     */
    public void setBackground (char background) {
        this.background = background;
    }

    /**
     * setFrame specifies the character to be used for drawing the frame
     * @param frame the character to be used for drawing the frame
     */
    public void setFrame (char frame) {
        this.frame = frame;
    }

    /**
     * This method clears the canvas and redraws the frame
     */
    public void clearCanvas() {
        if(canvas == null) return;
        for(int y=0;y<canvasHeight;y++) {
            for(int x=0;x<canvasWidth;x++) {
                canvas[x][y] = ((x==0)||(y==0)||(canvasWidth-x==1)||(canvasHeight-y==1))?frame:background;
            }
        }
    }

    /**
     * Draws the brush-character at the given position
     * @param point the position to draw at
     */
    public void draw(Vertex2D point) {
        if(canvas == null) return;
        if(point.getX() < 0 || point.getX() >= canvasWidth) {
            return;
        }
        if(point.getY() < 0 || point.getY() >= canvasHeight) {
            return;
        }
        canvas[point.getX()][point.getY()] = brush;
    }

    /**
     * This method prints out the whole canvas to the console
     */
    public void printCanvas() {
        for(int y=0;y<canvasHeight;y++) {
            for(int x=0;x<canvasWidth;x++) {
```

GraphicsContext.java

```
                System.out.print(canvas[x][y]);
            }
            System.out.print("\n");
        }
    }

    /**
     * Draws a line into the canvas using the Bresenham algorithm
     * @param startPoint Starting point of the line
     * @param endPoint Ending poit of the line
     */
    public void drawLine(Vertex2D startPoint, Vertex2D endPoint) {
        if(startPoint == null || endPoint == null) return;

        // if |gradient| > 90 degree, initialize with swapped x,y values
        boolean swapXY = (Math.abs(((double)(endPoint.getY()-startPoint.getY()))
            /((double)(endPoint.getX()-startPoint.getX())))>1);
        int x0=swapXY?startPoint.getY():startPoint.getX();
        int y0=swapXY?startPoint.getX():startPoint.getY();
        int x1=swapXY?endPoint.getY():endPoint.getX();
        int y1=swapXY?endPoint.getX():endPoint.getY();

        // swap start/end if necessary
        if(x1<x0) {
            int tmp=x0;
            x0=x1;
            x1=tmp;
            tmp=y0;
            y0=y1;
            y1=tmp;
        }

        double gradient = ((double)(y1-y0)/(double)(x1-x0)), error = 0;
        int direction = (gradient<0)?-1:1;
        Vertex2D pointToPass = new Vertex2D();

        for(int y=y0,x=x0;x<=x1;x++) {
            if(error>=0.5) {
                y += direction;
                error--;
            }
            pointToPass.init(swapXY?y:x,swapXY?x:y);
            draw(pointToPass);
            error+=direction*gradient;
        }
    }

    /**
     * This method accepts a {@link Polygon} as parameter and draws
     * all the lines between the vertices onto the canvas using the
     * brush character
     * @param polygonToDraw The Polygon to draw
     */
    public void drawPolygon(Polygon polygonToDraw) {
        Vertex2D vertices[] = polygonToDraw.getCoordinates();
        int numberOfVertices = vertices.length;
        for(int i = 1; i < numberOfVertices; i++) {
            this.drawLine(vertices[i-1], vertices[i]);
        }
        this.drawLine(vertices[numberOfVertices-1], vertices[0]);
    }

}
```

Polygon.java

```java
/**
 * SHEET 5 - Exercise 3c
 * The Polygon class implementation
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 26.11.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class Polygon {
    private Vertex2D[] vertices;

    public Vertex2D[] getCoordinates() {
        return vertices;
    }

    public void setCoordinates(Vertex2D[] coordinates) {
        vertices = coordinates;
    }
}
```

PolygonTestDrive.java

```java
/**
 * SHEET 5 - Exercise 3d
 * a class for testing the Polygon and GraphicsContext classes
 *
 * @author 273784 Philipp Fischer & 274196 Lucas Brutschy
 * @version 26.11.2006
 * platform: J2RE 1.5.0_08-b03, Linux 2.6.17-gentoo
*/
public class PolygonTestDrive {

    /**
     * This static method creates and returns an Object of the type Polygon.
     * It can produce star-like shapes as well as equilateral convex shapes.
     * For equilateral shapes the inner and outer radius have to be set to
     * the same value.
     *
     * @param numberOfSpikes The number of spikes the star should have
     * @param center The center of the shape
     * @param innerRadius The radius for the vertices in the inner of the star
     * @param outerRadius The radius for the vertices at the outer of the star
     * @param rotation This may apply an additional rotation to the shape, given in degree.
     * @return The shape as an object of the type Polygon
     */
    private static Polygon createStar(int numberOfSpikes, Vertex2D center, int innerRadius, int outerRadius, double rotation) {
        int numberOfVertices = numberOfSpikes * 2;

        Polygon myStar = new Polygon();
        Vertex2D[] starVertices = new Vertex2D[numberOfVertices];

        double alphaOffset = rotation * Math.PI / 180;

        for(int i = 0; i < numberOfVertices; i++) {
            starVertices[i] = new Vertex2D();
            double currentRadius = ((i%2==0)?innerRadius:outerRadius);
            double alpha = (Math.PI*2*(double)(i)) / (double)numberOfVertices + alphaOffset;
            int x = (int)Math.round(Math.sin(alpha) * currentRadius);
            int y = (int)Math.round(Math.cos(alpha) * currentRadius);
            starVertices[i].init(x + center.getX(), y + center.getY());
        }

        myStar.setCoordinates(starVertices);
        return myStar;
    }

    /**
     * This is the main class for testing the GraphicsContext class from exercise 3c
     * @param args unused
     */
    public static void main(String[] args) {
        GraphicsContext graphics;
        graphics = new GraphicsContext();
        graphics.init(80, 21);

        // Create a center vertex wich we modify for each creation of a star-polygon
        Vertex2D center = new Vertex2D();

        // Draws a 5-spike-star at (10,10) with the inner Radius of 3 and the outer radius of 8
        center.init(10,10);
        graphics.setBrush('#');
        graphics.drawPolygon(createStar(5, center, 3, 8, 0));

        // Draws a square (inner and outer radius are set to 5)
        // Rotation is set to 45 degree, if it was 0 it would produce a diamond
        graphics.setBrush('@');
        center.init(25,10);
        graphics.drawPolygon(createStar(2, center, 5, 5, 45));

        // This produces an approximation of a circle by creating a 19-segment
        // equilateral convex polygon
        graphics.setBrush('+');
        center.init(40,10);
        graphics.drawPolygon(createStar(10, center, 8, 8, 0));

        // This draws the a shifted copy of the following diamond, to
        // represent a shadow
        graphics.setBrush('-');
        center.init(61,10);
        graphics.drawPolygon(createStar(2, center, 8, 8, 0));

        // This is the diamond, with a radius of 8
        graphics.setBrush('%');
        center.init(60,10);
        graphics.drawPolygon(createStar(2, center, 8, 8, 0));

        // Print out the canvas to the console
        graphics.printCanvas();

    }
}

```