Práctica 2

Util

```
/** Clase de funcionalidad variada que proporciona una ayuda al alumno */
public class Util
{
  /**
    Detiene el programa el tiempo especificado
    @param segundos número de segundos a esperar
  */
  public static void wait(int segundos)
  {
    try
    {
      Thread.sleep(segundos*1000);
    }
    catch(Exception e)
    {
      System.out.println(e);
    }
  }

  /**
   * Detiene el programa el tiempo especificado, en milisegundos
   * @param milliseconds
   */
  public static void waitMilli(int milliseconds)
  {
    try
    {
      Thread.sleep(milliseconds);
    }
    catch(Exception e)
    {
      System.out.println(e);
    }
  }
}
```

Lienzo

```
import java.awt.*;
import javax.swing.JFrame;

public class Lienzo extends Canvas
{
  private Cuadrado cuadrados[];
  private boolean guardarCuadrados;

  public void setGuardarCuadrados(boolean guardarCuadrados) {
    this.guardarCuadrados = guardarCuadrados;
  }

  public boolean getGuardarCuadrados() {
    return this.guardarCuadrados;
```

```
  }

  /**
   * Draws one square.
   * @param cuadrado
   */
  public void pintar(Cuadrado cuadrado)
  {
    Cuadrado cuadrados[] = new Cuadrado[]{cuadrado};
    // Call more generic method
    this.pintar(cuadrados);
  }

  /**
   * Paints multiple squares at once
   * @param cuadrados
  */
  public void pintar(Cuadrado cuadrados[]) {
    if (this.getGuardarCuadrados()) {
      // If flag guardarCuadrados is activated, add cuadrado to buffer
      this.addCuadrados(cuadrados);
    } else {
      this.cuadrados = cuadrados;
    }
  }

  /**
   * Adds cuadrados to the cuadrados buffer
   * @param cuadrados
   * @return
   */
  public Cuadrado[] addCuadrados(Cuadrado cuadrados[]) {
    // Copy old buffer and clear
    Cuadrado oldBuffer[] = this.clearCuadrados();
    // Create new buffer with extra spaces
    this.cuadrados = new Cuadrado[oldBuffer.length + cuadrados.length];
    // New buffer is the sum of old buffer plus new items
    this.concatenateToCuadrados(oldBuffer);
    this.concatenateToCuadrados(cuadrados);

    // Return just in case user wants to use them
    return this.cuadrados;
  }

  /**
   * Concatenate an array of cuadrados to the cuadrados buffer.
   * @param cuadrados
   */
  public void concatenateToCuadrados(Cuadrado cuadrados[]) {
    int position = 0;
    for (int i = 0; i < this.cuadrados.length; i++) {
      if (this.cuadrados[i] == null && position < cuadrados.length) {
        this.cuadrados[i] = cuadrados[position];
        position++;
      }
    }
  }

  /**
   * Return cuadrados array and clear
   * @return
   */
  public Cuadrado[] clearCuadrados() {
    Cuadrado oldBuffer[];
    if (this.cuadrados == null) {
      oldBuffer = new Cuadrado[0];
    } else {
      oldBuffer = this.cuadrados;
    }

    this.cuadrados = new Cuadrado[0];
    return oldBuffer;
  }

  /**
   * Paints cuadrados in cuadrados buffer
```

```
   * @param graphics
   */
  public void paint(Graphics graphics)
  {
    for (Cuadrado cuadrado:this.cuadrados) {
      if (cuadrado != null) {
        // Draw cuadrado
        graphics.setColor(Color.BLUE);
        graphics.drawRect(cuadrado.getX(), cuadrado.getY(), cuadrado.getLado(), cuadrado.getLado());

        if (cuadrado.getFILL()) {
          graphics.fillRect(cuadrado.getX(), cuadrado.getY(), cuadrado.getLado(), cuadrado.getLado());
        }
      }
    }
  }
}
```

Dibujo

```
import java.awt.*;
import javax.swing.JFrame;

/**
  Facilita la representación gráfica de objetos creados por el alumno mediante una ventana gráfica y un lienzo
*/
public class Dibujo extends JFrame
{
  private Lienzo lienzo;

  private int sizeX;
  private int sizeY;

  public final int MAX_SIZE_X = 1920;
  public final int MAX_SIZE_Y = 1080;
  public final int DEFAULT_SIZE = 600;

  public Dibujo(int sizeX, int sizeY, boolean guardarCuadrados)
  {
    super("Dibujo");

    // Set canvas size
    this.setSizeX(sizeX);
    this.setSizeY(sizeY);

    // Create canvas
    lienzo = new Lienzo();
    lienzo.setSize(this.getSizeX(), this.getSizeY());

    // Add new Cuadrados to buffer or ov
    lienzo.setGuardarCuadrados(guardarCuadrados);

    this.add(lienzo);
    this.pack();
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setVisible(true);
  }

  public Dibujo() {
    this(0,0, true); // Create canvas of default size
  }

  public void setSizeX(int sizeX) {
    if (sizeX > 0 && sizeX < MAX_SIZE_X) {
      this.sizeX = sizeX;
    } else {
      this.sizeX = DEFAULT_SIZE;
    }
  }

  public int getSizeX() {
    return this.sizeX;
```

```
  }

  public void setSizeY(int sizeY) {
    if (sizeY > 0 && sizeY < MAX_SIZE_Y) {
      this.sizeY = sizeY;
    } else {
      this.sizeY = DEFAULT_SIZE;
    }
  }

  public int getSizeY() {
    return this.sizeY;
  }

  /**
   * Clears cuadrado buffer
  */
  public void clearCuadrados() {
    this.lienzo.clearCuadrados();
  }

  /**
    Pinta el cuadrado recibido por el App y actualiza el lienzo (canvas)
    @param cuadrado cuadrado a pintar
  */
  public void pintar(Cuadrado cuadrado)
  {
    lienzo.pintar(cuadrado);
    lienzo.repaint();
  }

  /**
   * Pinta múltiples cuadrados a la vez
   * @param cuadrados
  */
  public void pintar(Cuadrado cuadrados[]) {
    lienzo.pintar(cuadrados);
    lienzo.repaint();
  }

  /**
   * Draws the square in the middle of the canvas, regardless
   * of the size of the square and the size of the square
   * @param cuadrado
   */
  public void setInTheMiddle(Cuadrado cuadrado) {
    int sizeX = this.getSizeX();
    int sizeY = this.getSizeY();
    int lado = cuadrado.getLado();

    int x = (sizeX - lado) / 2;
    int y = (sizeY - lado) / 2;

    cuadrado.setX(x);
    cuadrado.setY(y);
  }

  /**
   * Renders the square and makes it go smoothly through the points passed.
   * From point a to point b it will generate n steps.
   * Framerate will determine how fast it will complete each step.
   * @param cuadrado
   * @param points
   * @param steps
   * @param frameRate
   */
  public void pintarPath(Cuadrado cuadrado, Point[] points, int steps, int frameRate) {
    // Move to initial position
    cuadrado.moveTo(points[0].getX(), points[0].getY());

    // Waiting times
    int waitTime = steps / frameRate;

    for (int i = 0; i < points.length; i++) {
      Point currentPoint = points[i];
      Point nextPoint;
```

```
      if ((i+1) == points.length) {
        nextPoint = points[i];
      } else {
        nextPoint = points[i+1];
      }

      // Calculate size of step to get from currentPoint to nextPoint
      // To increase smoothness use float/double
      int stepX = (nextPoint.getX() - currentPoint.getX()) / steps;
      int stepY = (nextPoint.getY() - currentPoint.getY()) / steps;

      for (int j = 0; j < steps; j++) {
        int nextPositionX = this.getNextPos(cuadrado.getX(), nextPoint.getX(), stepX);
        int nextPositionY = this.getNextPos(cuadrado.getY(), nextPoint.getY(), stepY);
        cuadrado.moveTo(nextPositionX, nextPositionY);

        this.pintar(cuadrado);
        Util.waitMilli(waitTime);
      }
    }
  }

  /**
   * Calculates the next position for pintarPath and avoids
   * overstepping. Since we are using ints, it is possible that
   * the step is rounded to a an higher value integer.
   * @param current
   * @param next
   * @param step
   * @return
   */
  public int getNextPos(int current, int next, int step) {
    int nextPosition = current + step;
    if (step > 0) {
      if (nextPosition > next) {
        nextPosition = next;
      }
    } else {
      if (nextPosition < next) {
        nextPosition = next;
      }
    }

    return nextPosition;
  }

}
```

Cuadrado

```
public class Cuadrado {
  private int x;
  private int y;
  private int lado;
  private static boolean FILL;

  Cuadrado(int x, int y, int lado) {
    this.setX(x);
    this.setY(y);
    this.setLado(lado);
  }

  // If lado is not given, generate cuadrado of lado 1
  Cuadrado(int x, int y) {
    this(x, y, 1);
  }

  public static void setFILL(boolean fill) {
    FILL = fill;
  }

  public static boolean getFILL() {
```

```
    return FILL;
  }

  public void setX(int x) {
    this.x = x;
  }

  public void setY(int y) {
    this.y = y;
  }

  public void setLado(int lado) {
    if (lado > 0) {
      this.lado = lado;
    } else {
      this.lado = 1; // Defualt value
    }
  }

  // Wrapper for easy moving
  public void moveTo(int x, int y) {
    this.setX(x);
    this.setY(y);
  }

  public int getX() {
    return this.x;
  }

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

  public int getLado() {
    return this.lado;
  }

  public void print() {
    System.out.println("Cuadrado: (" + this.getX() + ", " + this.getY() + "); " + this.getLado() + "; " + getFILL());
  }

}
```

AppDibujo01

```
public class AppDibujo01 {
  public static void main(String args[]) {
    Dibujo dibujo = new Dibujo();
    Cuadrado cuadrados[] = new Cuadrado[3];

    Cuadrado.setFILL(true);
    for (int i = 0; i < 3; i++) {
      Cuadrado cuadrado = new Cuadrado(100*i, 100*i, 100);
      cuadrados[i] = cuadrado;
    }

    dibujo.pintar(cuadrados);
  }
}
```

AppDibujo02

```
public class AppDibujo02 {
  public static void main(String args[]) {
    Dibujo dibujo = new Dibujo();
    Cuadrado cuadrados[] = new Cuadrado[3];

    Cuadrado.setFILL(true);
    for (int i = 0; i < 3; i++) {
```

```
      int lado = 100;
      Cuadrado cuadrado = new Cuadrado((dibujo.getSizeX() - 100*i - lado), 100*i, lado);
      dibujo.pintar(cuadrado);
      Util.wait(1);
    }
  }
}
```

AppDibujo03

```
public class AppDibujo03 {
  /**
   * When calling the app, you have two options, pass no
   * args which will render a square following a defautl path
   * or you can enter pair of points, that the square will follow.
   * For example:
   * java AppDibujo03 0 0 400 0 400 400 0 400
   * Will draw a square which goes to all corners of the canvas
   * @param args
   */
  public static void main(String[] args) {
    Dibujo dibujo = new Dibujo();

    // Create the square
    Cuadrado cuadrado = new Cuadrado(100, 100, 200);

    if (args.length < 2 || args.length % 2 != 0) {
      doDefaultPath(cuadrado, dibujo);
    } else {
      int points = args.length / 2;
      Point[] path = new Point[points];

      for (int i = 0; i < points; i++) {
        int x = Integer.parseInt(args[i*2]);
        int y = Integer.parseInt(args[i*2 + 1]);

        Point newPoint = new Point(x,y);
        path[i] = newPoint;
      }

      dibujo.pintarPath(cuadrado, path, 50, 5);
    }

  }

  public static void doDefaultPath(Cuadrado cuadrado, Dibujo dibujo) {
    Point point1 = new Point(0,0);
    Point point2 = new Point(400, 0);
    Point point3 = new Point(400,400);
    Point point4 = new Point(0, 400);
    Point point5 = new Point(300, 300);

    Point[] path = new Point[]{point1, point2, point3, point4, point5};
    dibujo.pintarPath(cuadrado, path, 50, 5);
    dibujo.setInTheMiddle(cuadrado);
    dibujo.pintar(cuadrado);
  }
}
```