Question 1: MiniCAD (Operations on Geometric Shapes)

a) The idea of my solution

We create the class Geometry with next fields: a static unique id to accumulate it each time when object of this or nested class were created, id to store new id for the geometry object, color, scale factor, position of X and a position of Y. The constructor initializes X and Y positions, color and an id. Also this abstract class we create with the move, scale and the toString methods because they use fields only inside current class. The only abstract method here is the paint method.

In the class Circle we add one field (radius) and override the method paint from the Geometry class. To draw a circle we rotate the turtle every time on the same angle and move it forward on the calculated distance depends on radius and scale factor. When the radius is very small we decrease the angle to prevent massive drawing with pixels in the small area.

In the class Rectangle we add three fields (weight, height and isFilled Boolean variable). In the method paint we easily draw borders when isFilled is false and when it is true, in the loop draw many lines one after each other till reach the height.

In the class Triangle we add a field length. In the method paint just draw three lines rotating turtle two times left on 120 degrees angles.

In the Model class when command is working with an id we just searching for it and do scale, move or remove. When it is working with creation, we just add a new geometry to the vector of geometries. Refreshing method calls the paint method for every a geometry in the vector.

In the ConsolePanel class we add method which parses the array of Strings and for the each command calls correspondent method to parse it further. If an input will be incorrect we show a popup message using JOptionPane.showMessageDialog method.

b) Source code

- Geometry.java

```
package uebung9.question1.geometry;

import java.awt.Color;

/** Geometry.java
 *
 * This abstract class provides useful methods for an arbitrary geometry object
 * like painting, moving and scaling.
 *
 * Institute for Pervasive Computing Johannes Kepler University Linz, Austria
 * http://www.pervasive.jku.at
 *
 * Copyright (c) 2013 IPC
 *
 * @author Thomas Schmittner, Andrii Dzhyrma */
public abstract class Geometry {

  /** Paints current geometry on the screen. */
  public abstract void paint();

  private static final String TO_STRING_FORMAT_STRING = "[%d] %s: posX=%d, posY=%d";
  // static id parameter which increases when new geometry was initialized
  private static int uniqueId = 100;
  // id of the current geometry
```

```
  private final int id;
  // color of the current geometry
  protected final Color color;
  // coordinates
  protected int posX;
  protected int posY;
  // scale factor for the scaling
  protected double scaleFactor = 1;

  /** Initializes a geometry.
   *
   * @param posX - the minimal X coordinate of the geometry
   * @param posY - the minimal Y coordinate of the geometry
   * @param color - the color of the geometry */
  public Geometry(int posX, int posY, Color color) {
    this.id = Geometry.uniqueId++;
    this.color = color;
    this.posX = posX;
    this.posY = posY;
  }

  /** Returns the id of the current geometry.
   *
   * @return the identification number */
  public final int getId() {
    return id;
  }

  /** Moves the current geometry by given coordinates.
   *
   * @param x - the X offset
   * @param y - the Y offset */
  public void move(int x, int y) {
    posX += x;
    posY += y;
  }

  /** Scales the current geometry by the given scale factor.
   *
   * @param factor - the scale factor */
  public void scale(double factor) {
    scaleFactor *= factor;
  }

  @Override
  public String toString() {
    return String.format(TO_STRING_FORMAT_STRING, id, this.getClass().getSimpleName(), posX, posY);
  }
}
```

- Circle.java

```
package uebung9.question1.geometry;

import java.awt.Color;

import uebung9.question1.turtle.Turtle;

/** Circle.java
 *
 * This class provides the basic functionality to draw a circle.
 *
 * Institute for Pervasive Computing Johannes Kepler University Linz, Austria
 * http://www.pervasive.jku.at
 *
 * Copyright (c) 2013 IPC
 *
 * @author Thomas Schmittner, Andrii Dzhyrma */
public class Circle extends Geometry {

  private double radius;

  /** Initializes a circle.
   *
   * @param posX - the minimal X coordinate of the circle
   * @param posY - the minimal Y coordinate of the circle
   * @param radius - the radius of the circle
```

```
   * @param color - the color of the circle */
  public Circle(int posX, int posY, double radius, Color color) {
    super(posX, posY, color);
    this.radius = Math.abs(radius);
  }

  @Override
  public void paint() {
    Turtle.setColor(color);
    // calculate scaled radius
    double scaledRadius = radius * scaleFactor;
    // if radius is less then 0.5, put just a dot
    if (scaledRadius < 0.5) {
      Turtle.setPos(posX, posY);
      Turtle.forward(0);
      return;
    }
    int max = 180;
    double side = 2.0 * Math.PI * scaledRadius / max;
    int angle = 360 / max;
    // if side is less than 2, choos bigger angle
    if (side < 2) {
      angle = (int) Math.round(2 * angle / side);
      while (90 % angle != 0)
        angle--;
      max = 360 / angle;
      side = side / 2 * angle;
    }
    // put turtle on the first position
    Turtle.setPos((int) Math.round(posX - side / 2.0 + scaledRadius), posY);
    Turtle.setAngle(0);
    // draw a circle
    for (int i = 0; i < max; i++) {
      Turtle.forward(side);
      Turtle.left(angle);
    }
  }
}
```

- Rectangle.java

```
package uebung9.question1.geometry;

import java.awt.Color;

import uebung9.question1.turtle.Turtle;

/** Rectangle.java
 *
 * This class provides the basic functionality to draw a rectangle.
 *
 * Institute for Pervasive Computing Johannes Kepler University Linz, Austria
 * http://www.pervasive.jku.at
 *
 * Copyright (c) 2013 IPC
 *
 * @author Thomas Schmittner, Andrii Dzhyrma */
public class Rectangle extends Geometry {

  private double width;
  private double height;
  private boolean isFilled;

  /** Initializes a rectangle.
   *
   * @param posX - the minimal X coordinate of the rectangle
   * @param posY - the minimal Y coordinate of the rectangle
   * @param width - the width of the rectangle
   * @param height - the height of the rectangle
   * @param color - the color of the rectangle
   * @param isFilled - if true the rectangle fills */
  public Rectangle(int posX, int posY, double width, double height,
      Color color, boolean isFilled) {
    super(posX, posY, color);
    this.width = width;
    this.height = height;
    this.isFilled = isFilled;
```

```
  }

  @Override
  public void paint() {
    // calculate scaled sides
    double newWidth = width * scaleFactor;
    double newHeight = height * scaleFactor;
    Turtle.setColor(color);
    Turtle.setAngle(0);
    // if isFilled is true, in loop draw height amount of lines
    if (isFilled) {
      int maxPosY = posY + (int) newHeight;
      for (int y = posY; y < maxPosY; y++) {
        Turtle.setPos(posX, y);
        Turtle.forward(newWidth);
      }
      return;
    }
    // otherwise just draw borders of the rectangle
    Turtle.setPos(posX, posY);
    Turtle.forward(newWidth);
    Turtle.left(90);
    Turtle.forward(newHeight);
    Turtle.left(90);
    Turtle.forward(newWidth);
    Turtle.left(90);
    Turtle.forward(newHeight);
  }
}
```

- Triangle.java

```
package uebung9.question1.geometry;

import java.awt.Color;

import uebung9.question1.turtle.Turtle;

/** Triangle.java
 *
 * This class provides the basic functionality to draw a equilateral triangle.
 *
 * Institute for Pervasive Computing Johannes Kepler University Linz, Austria
 * http://www.pervasive.jku.at
 *
 * Copyright (c) 2013 IPC
 *
 * @author Thomas Schmittner, Andrii Dzhyrma */
public class Triangle extends Geometry {

  private double length;

  /** Initializes a triangle.
   *
   * @param posX - the minimal X coordinate of the triangle
   * @param posY - the minimal Y coordinate of the triangle
   * @param length - the side size of the triangle
   * @param color - the color of the triangle */
  public Triangle(int posX, int posY, double length, Color color) {
    super(posX, posY, color);
    this.length = length;
  }

  @Override
  public void paint() {
    // calculate a scaled side length
    double newLength = length * scaleFactor;
    // draw the triangle
    Turtle.setColor(color);
    Turtle.setAngle(0);
    Turtle.setPos(posX, posY);
    Turtle.forward(newLength);
    Turtle.left(120);
    Turtle.forward(newLength);
    Turtle.left(120);
    Turtle.forward(newLength);
  }
```

```
}
```

- Model.java

```
      …
  public void remove(int id) {
    // search for the geometry with the given id and remove it
    int i = 0;
    for (; i < geometries.size(); i++)
      if (geometries.get(i).getId() == id) {
        geometries.remove(i);
        break;
      }
    // do not refresh panel if nothing was removed
    if (i == geometries.size())
      return;
    // --- do not change source code below this line ---
    refreshPanel(); // refresh panel only if scaling was successful
    fireModelChanged(); // refresh tree view only if moving was successful
  }

  private void refreshPanel() {
    Turtle.erase();
    // --- do not change source code above this line ---
    // paint all the geometries
    for (Geometry g : geometries)
      g.paint();
    // --- do not change source code below this line ---
    Turtle.showGraphics("Drawing Panel");
  }

  public void scale(int id, double factor) {
    // search for the geometry with the given id and scale it
    int i = 0;
    for (; i < geometries.size(); i++)
      if (geometries.get(i).getId() == id) {
        geometries.get(i).scale(factor);
        break;
      }
    // do not refresh panel if nothing was scaled
    if (i == geometries.size())
      return;
    // --- do not change source code below this line ---
    refreshPanel(); // refresh panel only if scaling was successful
  }

  public void move(int id, int moveX, int moveY) {
    int i = 0;
  // search for the geometry with the given id and move it
    for (; i < geometries.size(); i++)
      if (geometries.get(i).getId() == id) {
        geometries.get(i).move(moveX, moveY);
        break;
      }
    // do not refresh panel if nothing was moved
    if (i == geometries.size())
      return;
    // --- do not change source code below this line ---
    refreshPanel(); // refresh panel only if scaling was successful
    fireModelChanged(); // refresh tree view only if moving was successful
  } …
```

- ConsolePanel.java

```
package uebung9.question1.gui;

import java.awt.Button;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
```

```java
import uebung9.question1.geometry.Circle;
import uebung9.question1.geometry.Rectangle;
import uebung9.question1.geometry.Triangle;
import uebung9.question1.model.Model;
import uebung9.question1.util.Text;

/** ConsolePanel.java
 *
 * This class represents the console window at the bottom of the frame to enter
 * user commands.
 *
 * Institute for Pervasive Computing Johannes Kepler University Linz, Austria
 * http://www.pervasive.jku.at
 *
 * Copyright (c) 2013 IPC
 *
 * @author Thomas Schmittner, Andrii Dzhyrma */
public class ConsolePanel extends JPanel {

  private static final long serialVersionUID = 5788032037315903578L;
  private JTextField commandWindow;
  private JLabel label;

  public ConsolePanel() {
    label = new JLabel("Enter command:");
    commandWindow = new JTextField();
    commandWindow.setPreferredSize(new Dimension(550, 25));
    add(label);
    add(commandWindow);
    Button buttonRefresh = new Button("Execute");
    buttonRefresh.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        String[] args = commandWindow.getText().split(" ");
        // call read command method
        readCommand(args);
      }
    });
    add(buttonRefresh);
  }

  /* Checks whether amount of parameter is enough */
  private boolean checkParametersAmount(String[] args, int num) {
    if (args == null || args.length < num) {
      JOptionPane.showMessageDialog(getParent(), "Not enough parameters.",
          "Error", JOptionPane.ERROR_MESSAGE);
      return false;
    }
    return true;
  }

  /* Parses args for the command "create" */
  private void createGeometry(String[] args) {
    // if it is a circle
    if (args[1].equalsIgnoreCase(Text.CIRCLE)) {
      if (checkParametersAmount(args, 6)) {
        try {
          int posX = Integer.parseInt(args[2]);
          int posY = Integer.parseInt(args[3]);
          double radius = Double.parseDouble(args[4]);
          if (radius <= 0) {
            JOptionPane.showMessageDialog(getParent(),
                "Radius of the circle should be a postivie number.", "Error",
                JOptionPane.ERROR_MESSAGE);
            return;
          }
          Color color = parseColor(args[5]);
          if (color == null)
            return;
          Model.instance().add(new Circle(posX, posY, radius, color));
        } catch (NumberFormatException e) {
          showParseErrorMessage();
        }
      }
    } else
    // if it is a rectangle
    if (args[1].equalsIgnoreCase(Text.RECTANGLE)) {
```

```
    if (checkParametersAmount(args, 8)) {
      try {
        int posX = Integer.parseInt(args[2]);
        int posY = Integer.parseInt(args[3]);
        double width = Double.parseDouble(args[4]);
        double height = Double.parseDouble(args[5]);
        if (width <= 0 || height <= 0) {
          JOptionPane
              .showMessageDialog(
                  getParent(),
                  "Width and height of the rectangle should be positive numbers.",
                  "Error", JOptionPane.ERROR_MESSAGE);
          return;
        }
        Color color = parseColor(args[6]);
        if (color == null)
          return;
        boolean isFilled = Boolean.parseBoolean(args[7]);
        Model.instance().add(
            new Rectangle(posX, posY, width, height, color, isFilled));
      } catch (NumberFormatException e) {
        showParseErrorMessage();
      }
    }
  } else
  // if it is a triangle
  if (args[1].equalsIgnoreCase(Text.TRIANGLE)) {
    if (checkParametersAmount(args, 6)) {
      try {
        int posX = Integer.parseInt(args[2]);
        int posY = Integer.parseInt(args[3]);
        double length = Double.parseDouble(args[4]);
        if (length <= 0) {
          JOptionPane.showMessageDialog(getParent(),
              "Length of the triangle should be a postivie number.", "Error",
              JOptionPane.ERROR_MESSAGE);
          return;
        }
        Color color = parseColor(args[5]);
        if (color == null)
          return;
        Model.instance().add(new Triangle(posX, posY, length, color));
      } catch (NumberFormatException e) {
        showParseErrorMessage();
      }
    }
  } else
    // print an error otherwise
    JOptionPane.showMessageDialog(getParent(),
        "Unknown name of the geometry to crate.\nPlease use one from the follows: ["
            + Text.CIRCLE + ", " + Text.RECTANGLE + ", " + Text.TRIANGLE
            + "].", "Error", JOptionPane.INFORMATION_MESSAGE);
}

/* Parses args for the command "move" */
private void moveGeometry(String[] args) {
  if (checkParametersAmount(args, 4)) {
    try {
      int id = Integer.parseInt(args[1]);
      int moveX = Integer.parseInt(args[2]);
      int moveY = Integer.parseInt(args[3]);
      Model.instance().move(id, moveX, moveY);
    } catch (NumberFormatException e) {
      showParseErrorMessage();
    }
  }
}

/* Parses color to return a object of type Color */
private Color parseColor(String colorString) {
  if (colorString.equalsIgnoreCase(Text.BLUE))
    return Color.BLUE;
  if (colorString.equalsIgnoreCase(Text.GREEN))
    return Color.GREEN;
  if (colorString.equalsIgnoreCase(Text.RED))
    return Color.RED;
  if (colorString.equalsIgnoreCase(Text.YELLOW))
    return Color.YELLOW;
```

```
      JOptionPane.showMessageDialog(getParent(),
          "Unknown color.\nPlease use one from the follows: [" + Text.BLUE + ", "
              + Text.RED + ", " + Text.GREEN + ", " + Text.YELLOW + "].",
          "Error", JOptionPane.INFORMATION_MESSAGE);
      return null; // return null if the color is not blue, green, red or yellow
    }

    /* Parses args for the commands and call the correspondent methods */
    private void readCommand(String[] args) {
      if (!checkParametersAmount(args, 2))
        return;
      if (args[0].equalsIgnoreCase(Text.CREATE))
        createGeometry(args);
      else if (args[0].equalsIgnoreCase(Text.SCALE))
        scaleGeometry(args);
      else if (args[0].equalsIgnoreCase(Text.MOVE))
        moveGeometry(args);
      else if (args[0].equalsIgnoreCase(Text.REMOVE))
        removeGeometry(args);
      else
        JOptionPane.showMessageDialog(getParent(), "Unknown command.", "Error",
            JOptionPane.ERROR_MESSAGE);
    }

    /* Parses args for the command "remove" */
    private void removeGeometry(String[] args) {
      if (checkParametersAmount(args, 2)) {
        try {
          int id = Integer.parseInt(args[1]);
          Model.instance().remove(id);
        } catch (NumberFormatException e) {
          showParseErrorMessage();
        }
      }
    }

    /* Parses args for the command "scale" */
    private void scaleGeometry(String[] args) {
      if (checkParametersAmount(args, 3)) {
        try {
          int id = Integer.parseInt(args[1]);
          double scaleFactor = Double.parseDouble(args[2]);
          if (scaleFactor <= 0) {
            JOptionPane.showMessageDialog(getParent(),
                "Scale factor should be a postivie number.", "Error",
                JOptionPane.ERROR_MESSAGE);
            return;
          }
          Model.instance().scale(id, scaleFactor);
        } catch (NumberFormatException e) {
          showParseErrorMessage();
        }
      }
    }

    /* Shows an error message about parsing */
    private void showParseErrorMessage() {
      JOptionPane.showMessageDialog(getParent(),
          "Some numbers were not parsed correctly.", "Error",
          JOptionPane.ERROR_MESSAGE);
    }
  }
```

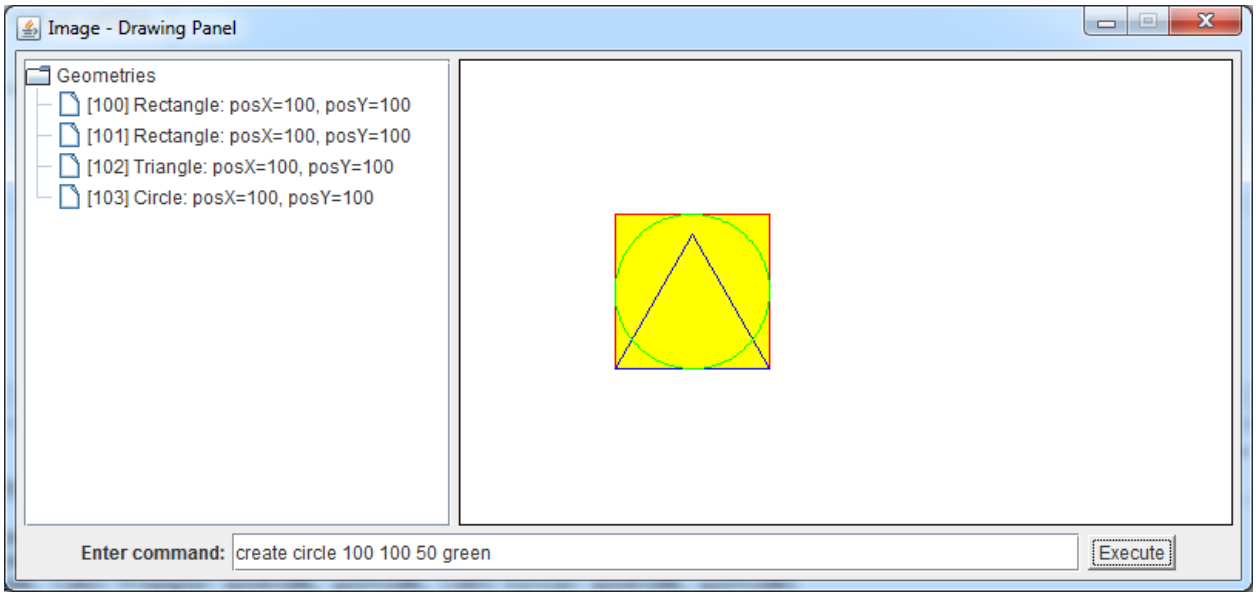
c) Test plan

#	Description	Input	Expected output
1	Test the figures drawing them inside each other.	**create rectangle 100 100 100 100 yellow true** **create rectangle 100 100 100 100 red false** **create triangle 100 100 100 blue** **create circle 100 100 50 green**	**Correct drawn geometries.**
2	Test the move, scale and the	**remove 100** **scale 101 1.5**	**A big red square, green circle inside and a blue**

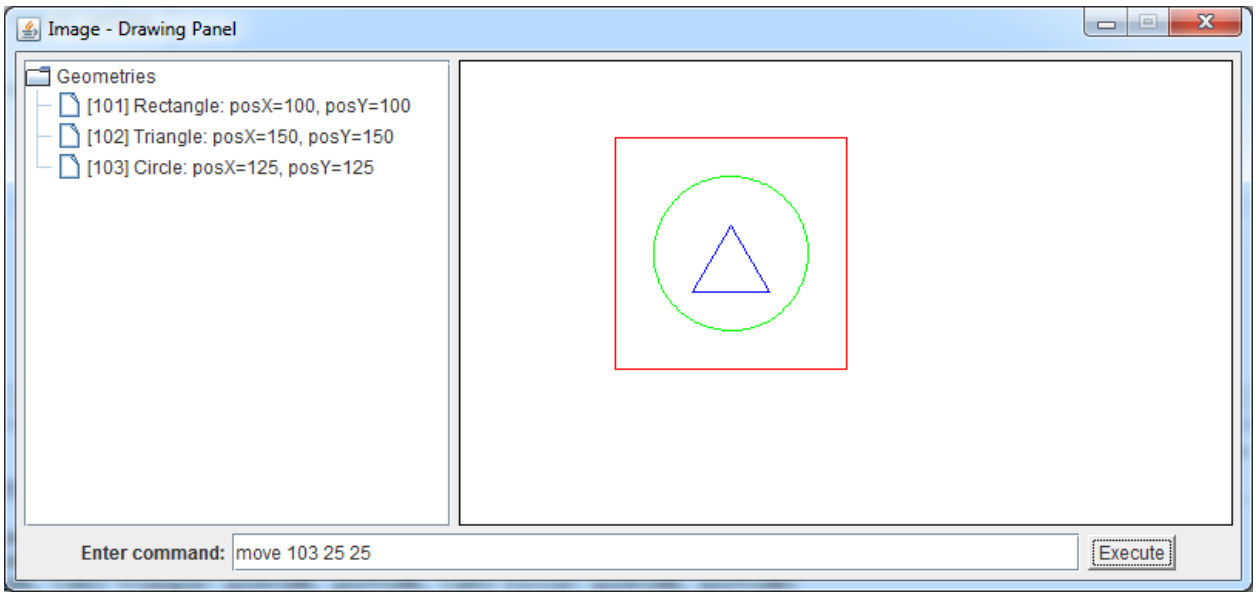
	remove commands.	**scale 102 0.5** **move 102 50 50** **move 103 25 25**	**triangle in the circle.**
3	Test the input commands for the incorrect number of parameters (less than required).	**“”, “create”, “move”, “remove”, “scale”, “create circle”, “create rectangle”, “create triangle”, “move 101”, “scale 101”**	**Get the error message about parameters.**
4	Test the input commands for the negative or equal to zero distance values.	**create circle 100 100 -50 green** **create rectangle 100 100 -100 100 red false** **create rectangle 100 100 100 -100 red false** **create triangle 100 100 -100 blue** **scale 101 -2** **scale 101 0**	**Get the error message about the numbers.**
5	Test incorrect integer values (Strings instead of integers, etc.).	**create circle one two three four** **create rectangle one two three four five six** **create circle one two three four** **scale one two** **move one two three** **remove one** **create circle 100.5 100.5 100 red** **move 101 0.5 0.5**	**Get the error message about parsing.**
6	Test incorrect color values.	**create circle 100 100 50 black** **create triangle 100 100 50 100** **create rectangle 100 100 100 100 BLUU false**	**Get the error message about colors.**
7	Test incorrect commands.	**draw circle** **delete 101** **cut 101 100 horizontal**	**Get the error message about unknown commands.**
8	Test incorrect geometries.	**create ellipse 100 100 100 50 red**	**Get the error message about an unknown geometry.**

d) The output of the program

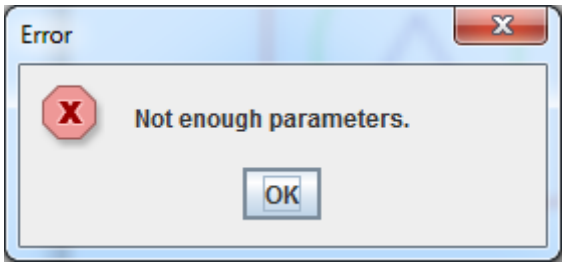
1.



2.

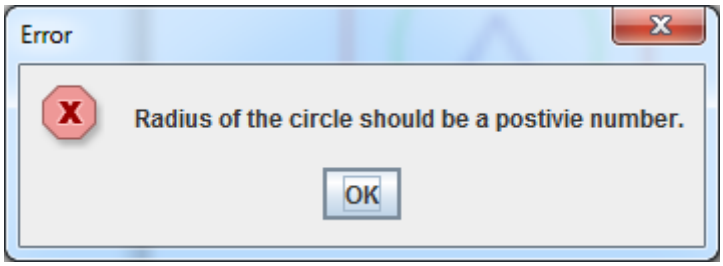


3.



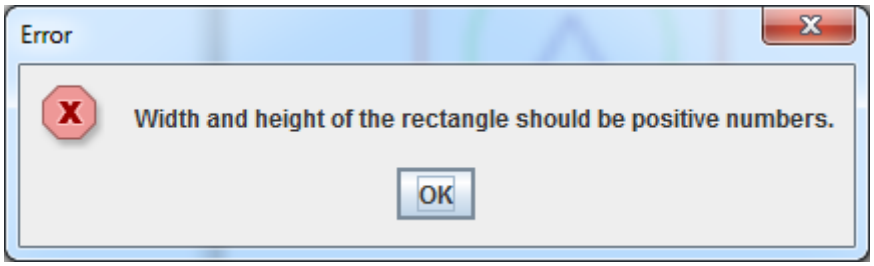
4.

create circle 100 100 -50 green

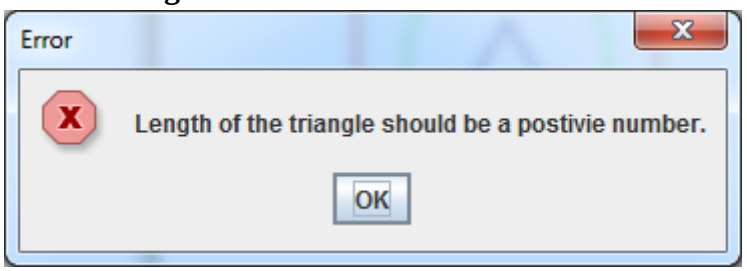


create rectangle 100 100 -100 100 red false

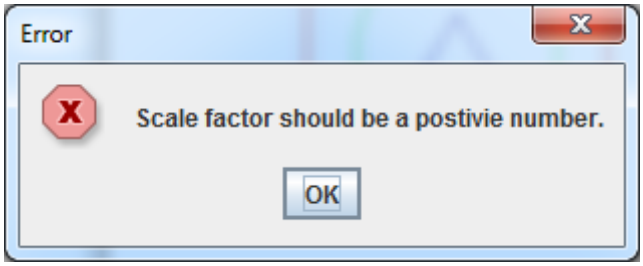
create rectangle 100 100 100 -100 red false



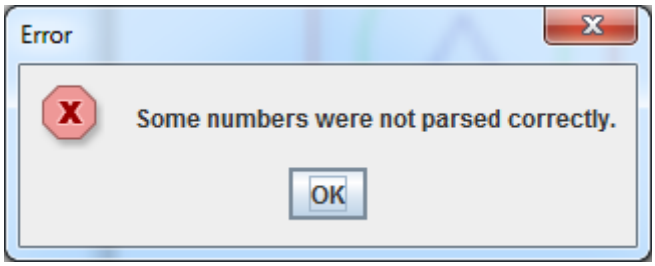
create triangle 100 100 -100 blue



scale 101 -2
scale 101 0

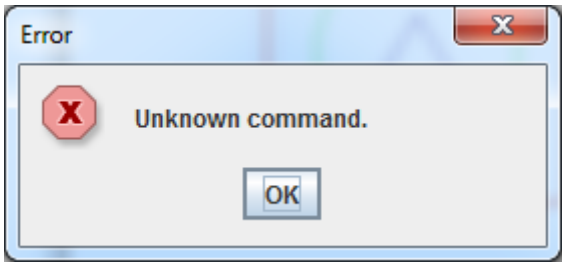


5.



6.

7.



8.