Lab 3: Graphical User Interfaces (GUI)

Task 1: *Vary your domain objects*

- Introduce visual variations of your objects: colours, sizes, and shapes.
  - Interfaces and inheritance hierarchies are helpful concepts.
  - Sometimes it is sufficient to add a number of new methods.
  - Be creative: Use variations specific to your animal or something creative (poses, clothing, accessories)! Don't use variations, other groups use as well.
  - Update your UML class diagram accordingly.
- Depict a couple of objects with different variations. They are stored in the ArrayList in class Scene and should have different positions.

Task 2: *Graphical user interface for controlling variations of the domain objects*

- Besides the panel of your graphics, introduce a panel with buttons
  - to modify the depiction.
  - Depending on the types of variations, other GUI components like sliders or input fields should be employed.
  - Use class JLabel to depict labels near to GUI elements.
- Make a short sketch about your GUI.
- Update your UML according to the GUI.
- Upgrade DrawingTool (or that class which extends JFrame)
  - GUI elements are properties, which are defined in separate classes.
  - Add a number of new methods which initialise these GUI elements.
  - Override public void actionPerformed(ActionEvent e) (see Task 3).

Task 3: *GUI in action*

- The GUI of Task 2 has to be brought to life:
  - The actionPerformed(ActionEvent e) method in DrawingTool is to be extended for this purpose.
  - As a result of a GUI action, e.g. a pushed button, setters of your domain classes are to be used to change the depiction.
  - Your Scene class will be the entry point for any changes:
    - in DrawingArea provide a property which stores your Scene
    - in DrawingArea provide a getter for your Scene
    - the Scene itself should provide appropriate setters to trigger any visual changes
    - with these getters and setters it is possible to control everything as a reaction of GUI usages within actionPerformed
    - That is, from class TestDrawingTool do it like: drawing.getScene().setHouses();
      (drawing fetches DrawingArea,
      drawing.getScene() fetches my Scene, and
      setHouses() sets/defines new houses.)
    - Or, for example: drawing.getScene().switchOnLights();
  - In order to update the screen after the user pushed a button
    - drawing.removeAll(); removes all objects of your graphics
    - drawing.revalidate(); informs the layout manager to recalculate the layout. This is necessary when new components are added.
    - drawing.repaint(); is to tell the system that the graphics needs to be repainted (which triggers the call of paintComponent which one should never call himself).

Software Quality: Code Conventions

a) Identifiers are in English.

b) Identifiers are meaningful, but not too long.

c) Variable identifiers begin with a small letter. Multiple words composed as CamelCase.

d) Identifiers for classes and interfaces begin with a capital letter. Multiple words composed as CamelCase.

e) Identifiers for constants consist only of uppercase letters. Multiple words composed by underline.

f) Left curly braces not in a new line. New line after left curly braces.

g) New line after right curly braces.
Exception: keyword else is in the same line.

h) Logical sections within a method have a comment as a heading.

i) Each block level is horizontally tap-indented by one level.

j) There is a blank line between methods.

k) There is a blank line between classes.

l) Classes and interfaces are separated by a blank line of import and package statements.

m) No more than one blank line in a row.

n) Order within a class or an interface:

1. properties (constants and variables)
2. constructors
3. getter and setter for properties, but only if required
4. other methods