Evaluation

All the material of the course is allowed, including the solution of TPs. Internet access is allowed, excluding communication programs (e-mail, instant messaging, …).
At the end of the evaluation, export all the Eclipse projects in a single zip file and upload it on Campus.

A DSL for GUI windows

We want to create a DSL to represent GUI windows. Each window is structured in sections. Each section can contain several labels, buttons, or other sections.

Here is an example of program in this language (**Example 1**):

```
frame A {
  title: "Frame A"
  size: 200 x 100
  section content {
    section header {
       label: "Hello"
    }
    section body {
       button: "World!"
    }
  }
}
```

The program execution is equivalent to the following Java code. Note that sections are only used to organize the code and have no effect on the actual window structure.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.SwingUtilities;

import java.awt.FlowLayout;

public class FrameApplication {
   public static void main(String[] args) {
     SwingUtilities.invokeLater(new Runnable() {
       public void run() {
         JFrame A = new JFrame();
         A.setLayout(new FlowLayout());
         A.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         A.setTitle("Frame A");
         A.setSize(200, 100);
         JLabel label0 = new JLabel();
         A.add(label0);
         label0.setText("Hello");
         JButton button1 = new JButton();
         A.add(button1);
         button1.setText("World!");
         A.setVisible(true);
       }
     });
  }
}
```

Buttons can optionally point to a frame to visualize when the button is clicked. Here is another example with its java equivalent (**Example 2**):

```
frame A {
  title: "Frame A"
  size: 200 x 100
  section content {
    label: "Hello"
    button: "..." -> B
  }
}

frame B {
  title: "Frame B"
  size: 300 x 100
  section content {
    label: "...World!"
    button: "Restart" -> A
  }
}
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.SwingUtilities;

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class FrameApplication {
   public static void main(String[] args) {
     SwingUtilities.invokeLater(new Runnable() {
       public void run() {

               JFrame A = new JFrame();
               JFrame B = new JFrame();

               A.setLayout(new FlowLayout());
               A.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               A.setTitle("Frame A");
               A.setSize(200, 100);
               JLabel labelA0 = new JLabel();
               A.add(labelA0);
               labelA0.setText("Hello");
               JButton buttonA1 = new JButton();
               buttonA1.addActionListener(new ActionListener(){
                       @Override
                       public void actionPerformed(ActionEvent e)
                       {
                               A.setVisible(false);
                               B.setVisible(true);
                       }
               });
               A.add(buttonA1);
               buttonA1.setText("...");

               B.setLayout(new FlowLayout());
               B.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               B.setTitle("Frame B");
               B.setSize(300, 100);
```

```
                JLabel labelB0 = new JLabel();
                B.add(labelB0);
                labelB0.setText("...World!");
                JButton buttonB1 = new JButton();
                buttonB1.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e)
                        {
                                B.setVisible(false);
                                A.setVisible(true);
                        }
                });
                B.add(buttonB1);
                buttonB1.setText("Restart");

                A.setVisible(true);
        }
    });
  }
}
```

Exercise 1 (6pts)

Write a metamodel for this DSL, expressive enough to represent **Example 1** and **Example 2**.

Exercise 2 (6pts)

Develop a textual editor for this DSL using Xtext. The editor must correctly recognize the DSL programs of **Example 1** and **Example 2**. (Note that Xtext will generate another metamodel for this DSL, that can be different from the one you created in Exercise 1)

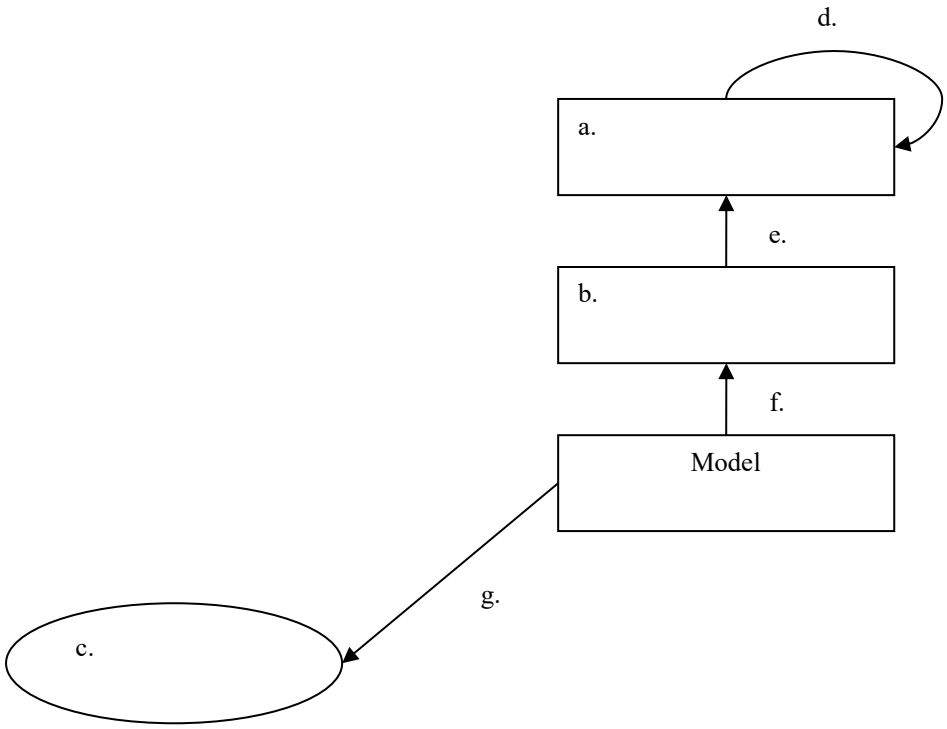
Exercise 3 (6pts)

Write a fluent API for this DSL, following the pattern and naming conventions explained during the course. The fluent API must be expressive enough to represent **Example 1**.

(+2pts) Add an “execute” method that interprets a complete program written using the fluent API. Running “execute” on the code in Example 1 should have the same effect as the equivalent java code.

Exercise 4 (2pts)

What are the names of the entities, relations and levels indicated by the letters a…g in the following figure?



Overview of the main concepts and relations in Model Engineering