Design and Analysis of Information Systems (MAS)

Mariusz Trzaska, Ph. D.
mtrzaska@pjwstk.edu.pl

Lecture 14

Declarative Graphical User Interfaces

Outline

- Introduction
- The key to success
- The proposed approach
- Problems to solve
- Utilization samples
- The summary

The implemented prototype:

https://code.google.com/archive/p/gcl-dsl/

The Declarative Way

- What is it?
- What is the difference between the new approach and the existing one?
- Why we need a new solution?
- Pros
- Cons
- Different interpretations:
  - component declarativeness, e.g. XAML,
  - model declarativeness.

The Key to Success

- On the one hand we would like to computer create a GUI without our involvement.
- On the other hand we expect fulfilment of our requirements regarding:
  - Functionality,
  - Aesthetic,
  - Usability,
  - Performance.

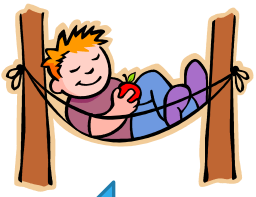
The Key to Success (2)

- In order to fulfilment that a computer would have to read our minds…
- Currently it is not possible…
- Hence we need to provide some hints.
- The point of the declarative approach is that we tell „**what to do**”, rather then how to do it.

The Key to Success (3)

- Hence we need to find:

A happy balance between our involvement in the GUI creation process and a generalization of the solution.



A poor conformation to user's expectations

A perfect conformation to user's expectations

The Key to Success (4)

- If in order to generate a GUI, the system would need too many details (necessary for a perfect customization), then using it will be more complicated then the ordinary GUI creation.
- It seems that in order to lower our involvement, we need to accept some universality of the solution, e.g. the generated GUIs will be similar to each other.
- Nevertheless in typical cases, the GUIs will be useful.

Assumptions of the Proposed Declarative Approach

- A programmer defines which parts of the Java model (classes) should have a GUI:
  - Attributes,
  - Methods.
- The programmer calls a single method which for a given class instance shows the generated form (window).
- The system generates forms allowing creation of the new instances, editing them, etc.

Assumptions of the Proposed Declarative Approach (2)

- Optional customization of the generated window:
  - Labels,
  - Widget class,
- Works with:
  - Numbers and texts,
  - Booleans,
  - Dates,
  - Enums.
- In order to make the implementation easier we are not going to support:
  - Data validation,
  - i18n.

The Ordinary Solution

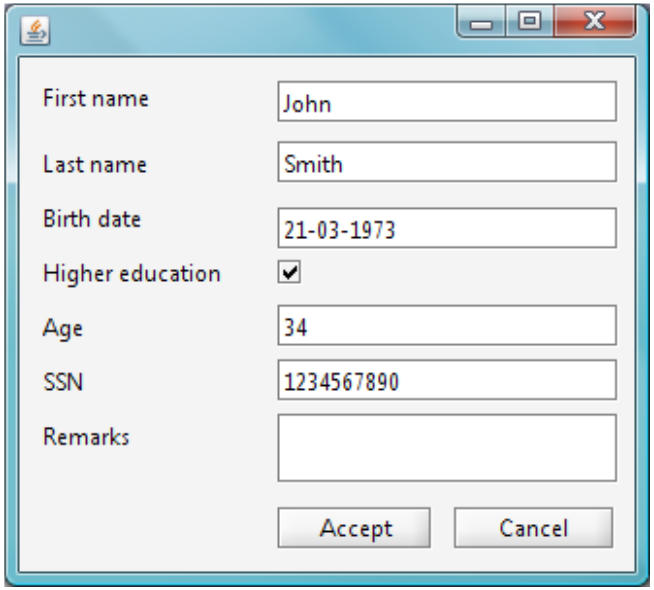
- To achieve the mentioned functionality using an ordinary approach (manual writing of the code) a programmer has to:
  a) Create an empty form,
  b) Add a layout manager,
  c) For each attribute add a widget,
  d) For each method add a widget,

The Ordinary Solution (2)

- To achieve the mentioned functionality - *cont*.
  e) For each widget add a label,
  f) For each widget add a code which will read the value and put it to the widget,
  g) Add control buttons (*Accept*, *Cancel*),
  h) For the "Accept" button add a code which will take all widgets' values and put them to the model and hides the form,
  i) For the "Cancel" button add a code hiding the form.

The Ordinary Solution (3)

- The implementation of the above requirements means writing a few dozens lines of code:
  - 7 attributes multiplied by 5 to 10 lines per widget,
  - Layout management,
  - Control buttons processing.
- The Jigloo GUI Builder generated 105 lines of code (without *f*, *h*, *i* points).



Problems to Solve

- How to read a content of the class (the structure)?
- How to pass the information defining which parts of model should have reflection in the GUI?
- What kind of widgets should be used for particular types of data?
- How to connect particular GUI items (widgets) to data (read, write)?

Reading of the Class Content

- Utilization of the reflection technology.
  - Provides information about classes structure, e.g. attributes, methods.
  - Makes possible
    - Creation of the class instances (objects),
    - Calling methods,
    - Updating/Reading attributes' values.

Reading of the Class Content (2)

- Utilization of the reflection technology - cont.
  - Available for:
    - The Java,
    - The MS C#
    - Partially for the C++ (RTTI - *Run Time Type Information*).
  - The key class (Java): `Class` and methods:
    - `getDeclaredFields()`,
    - `getDeclaredMethods()`.

Which class parts should have GUIs?

- How to define which parts of the model should have dedicated widgets?
  - All of them?
  - Automatically? It is not possible for a general case.
  - A configuration file, e.g. XML, properties?
  - Passing parameters for a method?
  - Convention over configuration,
  - Other possibilities?

Class Annotations

- Annotations defined by a programmer could be related to:
  - Classes,
  - Attributes,
  - Methods.
- They exist in:
  - The Java,
  - The MS C# (as *Attributes*).
- It is possible to use existing ones or create a custom.

Class Annotations (2)

- The Java syntax

```
@Override
public String toString() {
    return „My own implementation of the toString()";
}
```

- The C# syntax

```
[MyOwnAnnotation]
Public override String ToString() {
     return „My own implementation of the toString()";
}
```

- Custom Java annotation syntax

```
public @interface MyOwnAnnotation {
}
```

Class Annotations (3)

- Customization of the annotations
  - Default values,
  - Applies to:
    - Attributes,
    - Constructors,
    - Local variables,
    - Packages,
    - Methods,
    - Types (class, interface, enum types).
  - Available during:
    - Compilation time,
    - Run-time.

Customization of the Generated GUI

- We are going to create dedicated annotations for the GUI customization.
- They need special parameters allowing detailed customization of the particular widgets.
- The number of parameters should be as small as possible.
- The parameters should have default values.

Customization of the Generated GUI (2)

- We divide the set of annotations to achieve better separation of different invariants:
  - Attributes,
  - Methods.
- The set of parameters is similar but the default values are different.

Customization of the Generated GUI (3)

- Annotation parameters:
  - `label`. Describes a label for a widget. If it is an empty string (default) then a name of the attribute or the method (without a prefix get/set) will be used. This parameter was necessary because sometimes we need to change a name, combine it from a few words or we just need to use letters (e.g. special polish letters) which are forbidden in a program's source code.;
  - `widgetClass`. Widget class which will be used to handling editing of the attribute or method. Default value is the `JTextBox`;

Customization of the Generated GUI (4)

- Annotation parameters – *cont.*:
  - `tooltip`. A short text which will be presented to the user when a mouse cursor will be hovered over the widget.
  - `getMethod`. A method to read a value of the attribute. If it is an empty string (default value) a standard setter/getters approach will be used;
  - `setMethod`. A method for writing the value. If it is an empty string (default value) a standard setter/getters approach will be used;

Customization of the Generated GUI (5)

- Annotation parameters – cont.:
  - `showInFields`. A flag telling if this item should be visible in a form with fields;
  - `showInTable`. A flag telling if this item should be visible in a form with a table (grid) view;
  - `showInSearch`. A flag telling if this item should be visible in a form utilized to entering search criterion;
  - `order`. A number defining weight of a widget (order in a form);

Customization of the Generated GUI (6)

- Annotation parameters – cont.:
  - `scaleWidget`. Indicates if this widget should change size during resizing the form.
  - `readOnly`. If this is true then the widget is read only;

The Creation of Custom Annotations

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface GUIGenerateAttribute {
    /**
     * Podaje nazwe  etykiety dla pozycji w GUI. Gdy "" to bedzie uzyta nazwa
    atrybutu (pierwsza litera skonwertowana do duzej).
     * @return
     */
    String label() default "";

    /**
     * Definiuje widget, ktory bedzie stworzony dla obslugi danego atrybutu.
     * Musi dziedziczyc z "java.awt.Component" ("javax.swing.JComponent")
     * W tym elemencie MUSI byc metoda "String getText()" oraz "setText(String)".
     * @return
     */
    String widgetClass() default "javax.swing.JTextField";

    /**
     * Definiuje tooltip wyswietlany dla widgetu.
     * @todo
     * @return
     */
    String tooltip() default "";
    // [...]
```

The Creation of Custom Annotations (2)

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface GUIGenerateAttribute {
    // - c.d.
    /**
     * Definiuje metode do pobierania wartosci dla atrybutu. Jezeli jest ""
    oznacza to wykorzystanie metody "String get<NazwaAtrybutu>()"
     * @return
     */
    String getMethod() default "";

    /**
     * Definiuje metode do ustawiania wartosci dla atrybutu. Jezeli jest ""
    oznacza to wykorzystanie metody "set<NazwaAtrybutu>(String)"
     * @return
     */
    String setMethod() default "";

    /**
     * Definiuje czy generowac widget przy widoku w polach (przegladanie lub
    edycja).
     * @return
     */
    boolean showInFields() default true;
```

The Creation of Custom Annotations (3)

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface GUIGenerateAttribute {
    // - c.d.

    /**
     * Definiuje czy generowac widget przy widoku tabeli.
     * @return
     */
    boolean showInTable() default true;

    /**
     * Definiuje czy generowac widget przy widoku w polach do wyszukiwania.
     * @return
     */
    boolean showInSearch() default true;

    /**
     * Okresla kolejnosc wygenerowanegu widgetu wsrod innych wygenerowanych
    widgetow.
     * @return
     */
    int order() default 0;
    // ...
```

The Creation of Custom Annotations (4)

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface GUIGenerateAttribute {
    // - c.d.

    /**
     * Gdy True to wartosc atrybutu nie moze byc zmieniona.
     * @return
     */
    boolean readOnly() default false;

    /**
     * Gdy true to w czasie zmian wielkosci formy dostosowuje wielkosc widgetu.
     * @return
     */
    boolean scaleWidget() default false;
}
```

Selection of widgets

- The simplest and most general approach utilizes the `JTextBox` control,
- Of course it could be improved:
  - Special `ComboBox`'es for the `enum` type,
  - `CheckBox`'y working with the `boolean` type,
  - Other – defined by a programmer.

Transferring data to/from the Control

- Because our solution is a generic one (able to work with different types of data) we need a common data type: the `String`.
- We require that every widget (control) provided two methods:
  - `void setText(String)` – writing the data into the control (widget),
  - `String getText()` – reading the data from the widget.

Transferring data to/from the Control (2)

- Most of the Swing controls work that way.
- Sometimes we need to implement a simple wrapper for existing control to work with the above methods.
- In some cases there will be a need of own/custom implementation of the control.

Utilization Samples – Simple Annotations

- The implemented prototype:

https://code.google.com/archive/p/gcl-dsl/

- A simple class for which we need a GUI.

```
public class Person {
    private String firstName;
    private String lastName;
    private Date birthDate;
    private boolean higherEducation;
    private String remarks;
    private int SSN;
    private double annualIncome;

    public int getAge() {
        // [...]
    }
}
```

Utilization Samples – Simple Annotations (2)

- We introduce simple annotations with default values.

```
public class PersonAnnotated {
    @GUIGenerateAttribute
    private String firstName;
    @GUIGenerateAttribute
    private String lastName;
    @GUIGenerateAttribute
    private LocalDate birthDate;
    @GUIGenerateAttribute
    private boolean higherEducation;
    @GUIGenerateAttribute
    private String remarks;
    @GUIGenerateAttribute
    private int SSN;
    @GUIGenerateAttribute
    private double annualIncome;
    @GUIGenerateMethod
    public int getAge() {
        // ...
    }
    // Standard getters/setters methods
}
```

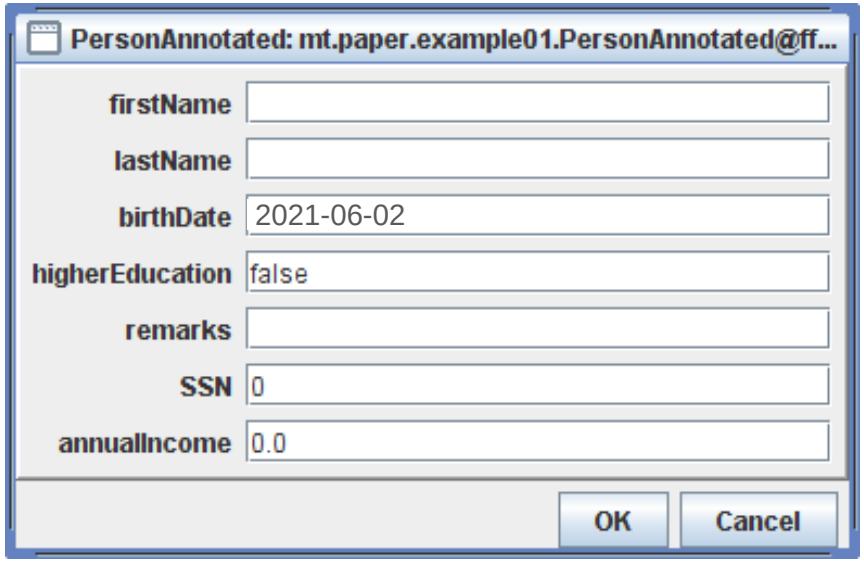
Utilization Samples – Simple Annotations (3)

- We introduce simple annotations with default values - cont.

```
JDesktopPane jdp = new JDesktopPane();

PersonAnnotated person = new PersonAnnotated();
SingleObjectPlusFrame personFrame = GUIFactory.getObjectFrame(person, jdp,
                                                    false, true, true);
```

- And the result of a single method call `GUIFactory.getObjectFrame(...)`
- There are some imperfections which we are about to improve.



Utilization Samples – Customized Annotations

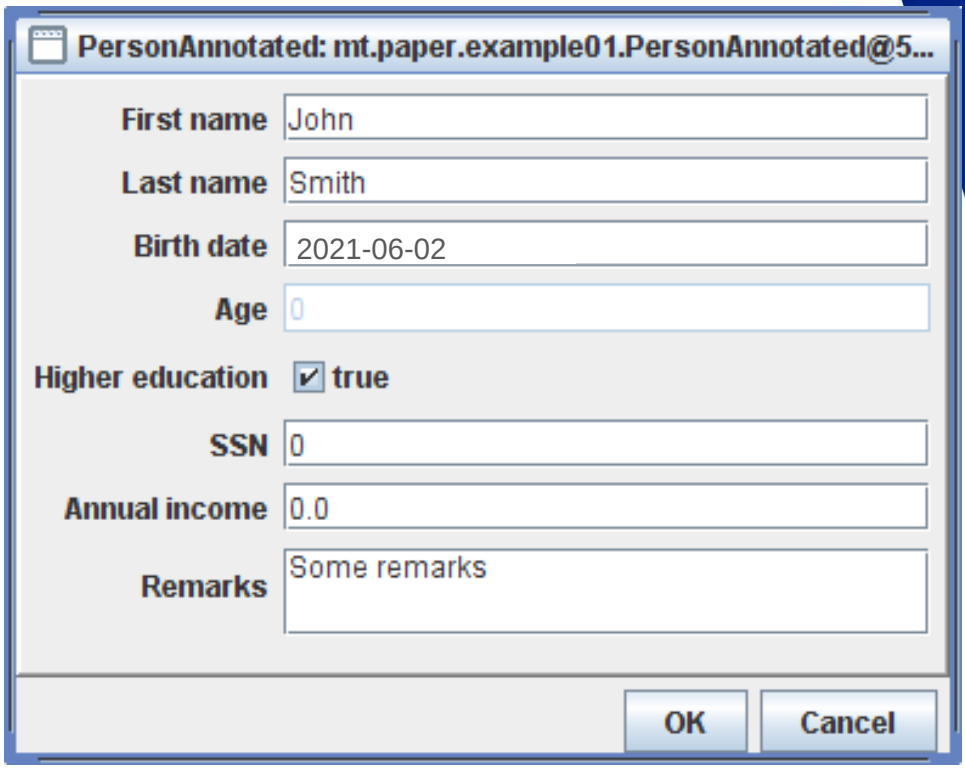
- In this case annotations are parameterized

```
public class PersonAnnotated {
    @GUIGenerateAttribute(label = "First name", order = 1)
    private String firstName;
    @GUIGenerateAttribute(label = "Last name", order = 2)
    private String lastName;
    @GUIGenerateAttribute(label = "Birth date", order = 3)
    private Date birthDate = new Date();
    @GUIGenerateAttribute(label = "Higher education",
    widgetClass="mt.mas.GUI.CheckboxBoolean", order = 5)
    private boolean higherEducation;
    @GUIGenerateAttribute(label = "Remarks", order = 50,
    widgetClass="javax.swing.JTextArea",scaleWidget=false)
    private String remarks;
    @GUIGenerateAttribute(order = 6)
    private int SSN;
    @GUIGenerateAttribute(label= "Annual income", order=7)
    private double annualIncome;
    @GUIGenerateMethod(label="Age", showInFields = true, order = 4)
    public int getAge() { ...}

    // Standard getters/setters methods
}
```

Utilization Samples – Customized Annotations (2)

- The same single call but with parametrized annotations there are:
  - Better labels,
  - Improved order,
  - The age is read only,
  - The *Remarks* field is resized,
  - Dedicated controls (e.g. `JCheckBox`).



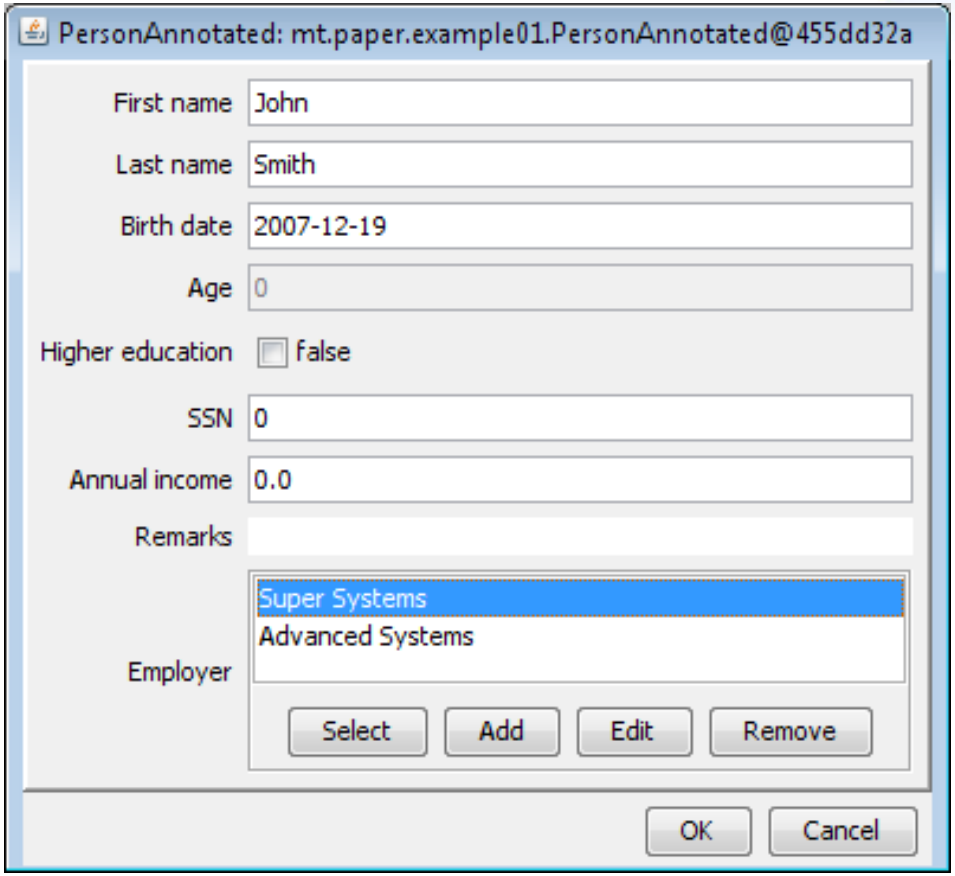
Together with the `ObjectPlus`

- The connection of the senseGUI library together with the `ObjectPlusX`

  provides links

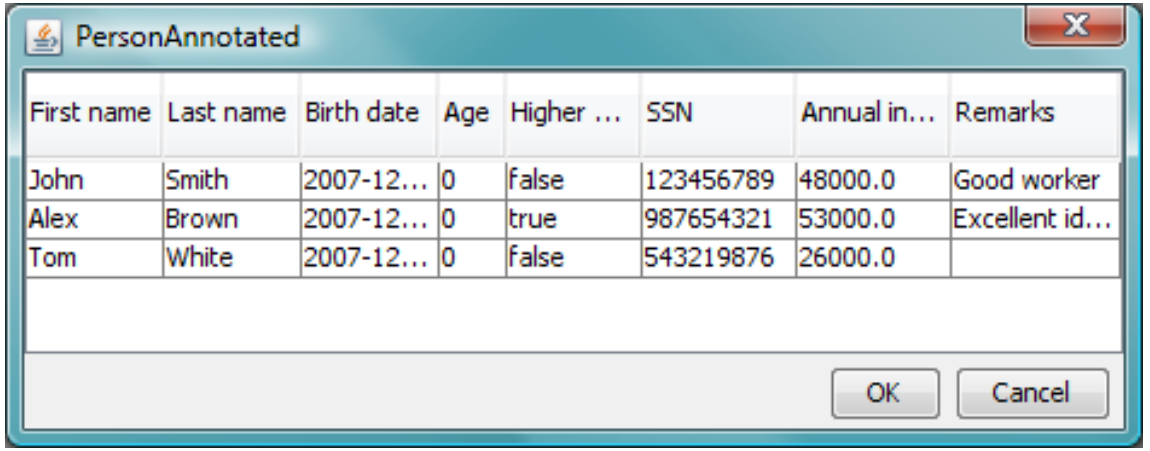
  management,

  extent, etc.



Additional Possibilities

- Object search
  - The criteria are entered using a generated form,
  - Automatic progress information.
- The table view
  - A collection,
  - Possibility of an object (a row) selection from the table.



Fluent API instead of annotations

- Instead of annotations, you can create your own *Domain Specific Language* (DSL).
- One of the possibilities is to use the so-called *Fluent API*.
- Popular solutions, e.g.
  - jQuery,
  - LINQ,
  - model mapping in Entity Framework.

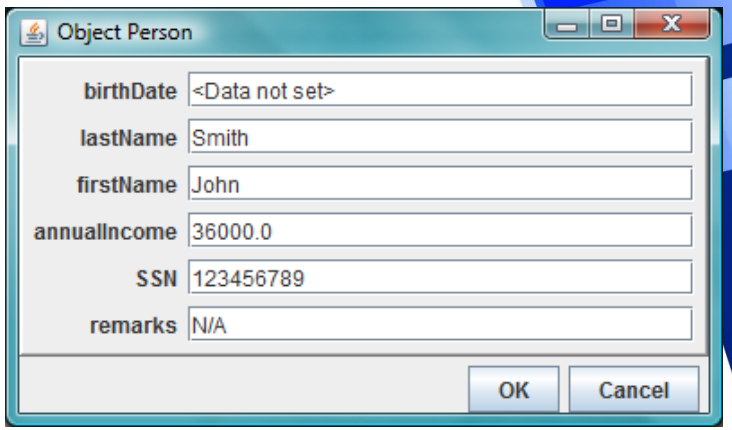
```
using (var ctx = new BloggingContext()) {
  var blogs = ctx.Blogs.Include(blog => blog.Posts).ToList();
}
```

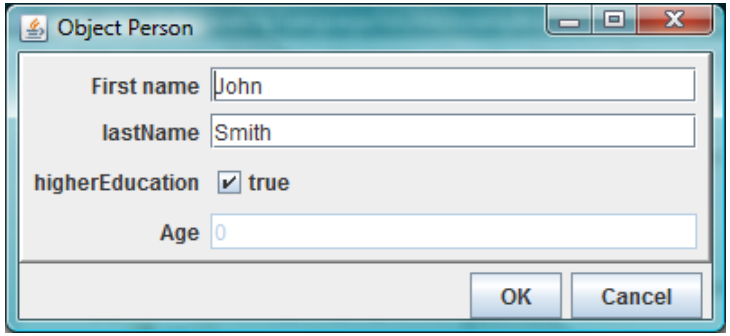
GCL DSL

- New version of the library:
  - Fluent API instead of annotations,
  - Validators,
  - I18n,
  - AdHoc GUI,
  - https://code.google.com/archive/p/gcl-dsl/
  - *Trzaska M.: Data Migration and Validation Using the Smart Persistence Layer 2.0. The 16th IASTED International Conference on Software Engineering and Applications (SEA). Las Vegas, USA. Acta Press. ISBN: 978-0-88986-951-6.* ***Download***
  - *Trzaska M. GCL - An Easy Way for Creating Graphical User Interfaces. Journal of Systemics, Cybernetics and Informatics. ISSN: 1690-4524. pp. 81-88.* ***Download***

GCL DSL (2)

```
JFrame frame1
= create.frame.usingOnly(person);
```

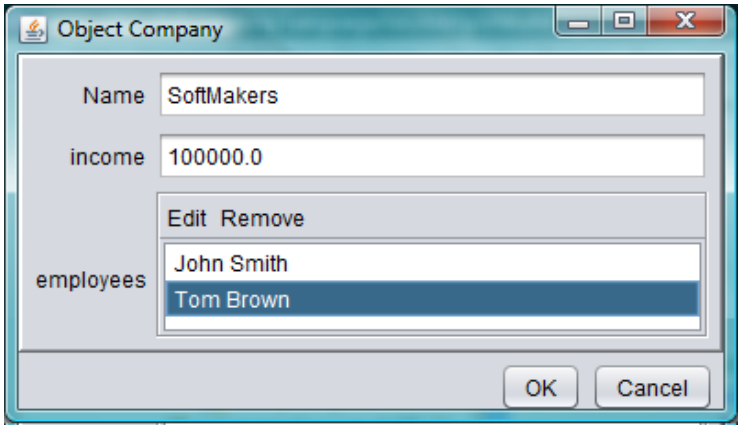




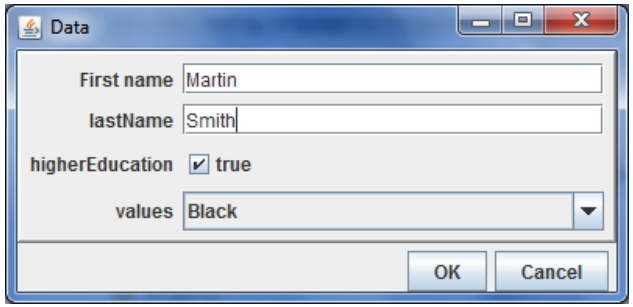

```
JFrame frame =  create.
                frame.
                using(person).
                containing(
                    attribute("firstName").as("First name"),
                    attribute("lastName").validate(new ValidatorNotEmpty()),
                    attribute("higherEducation"),
                    method("getAge").as("Age"));
```

GCL DSL (3)

```
JFrame frame =  create.
                frame.
                using(company).
                containing(
                        attribute("name").as("Name"),
                         attribute("income"),
                         attribute("employees"));
```




```
AdHocActionPerformed processAccept = new AdHocActionPerformed() {
        @Override
        public void Accept(Map<String, String> enteredData) {
                // Do something with the fields...
        }
};
frame =  create.
        frame("Data", processAccept, "OK").
        containing(
                attribute("firstName").as("First name").value("Martin"),
                attribute("lastName").validate(new ValidatorNotEmpty()),
                attribute("higherEducation").type(boolean.class),
                attribute("values").type(Colors.class));
```



The Summary

- It seems that the declarative way for GUI creation is very useful.
- Thank to the solution a programmer is able to save a lot of work – especially in the case of ordinary business application.
- Each such a solution is a compromise between our involvement and the achieved results.
- The implemented library proves that the approach is useful and possible to use.