Lab 2: Layout & Model-View separation

Goals

- practice implementing a layout from the prototype
- practice layouting techniques and using gui frameworks
- practice developing a data model
- practice working with provided data and populating the interface
- working in groups on different parts of the same solution

Assignment

In the Prototyping assignement you learned and experienced fast and iterative prototyping. Now it is time to start implementing one of the prototypes. In professional environemtns it often happens that you are assigned to implement prototypes that you haven't developed because they either come directly from the customer or from other department in your working environment. For this reason we will review all of the submitted solutions and choose one that we thought fits best to our requirements and requirments of the course (remaining labs).

Your assignment in this lab is to implement the layout of the chosen prototype and extend the model to fit the needs of the application. You can choose between implementing a **desktop or mobile** application. The process of developing mobile and desktop interfaces is similar in the sense that you usually work with gui frameworks and widgets, your code needs to be compiled before it's executed and in case of Android you can even reuse part of the code between Java desktop and Android application. Web pages/applications typically have a different process and for this reason an exercise of developing a web interface is left for lab 4.

During this assignment it is important to separate the view from the model. All the model code (data and any business rules) should be kept in the model class files. These classes should not know anything about the layout (graphics).

To help you share the work with your team mates we suggest using versioning control.

How (mobile)

For the mobile application prototype we have selected the solution submited by Group 30.

As mentioned in the lectures we provide the instructions for Android development. You are free to use any other platform as long as you follow the key principles (model-view separtion in this lab).

Step 0. Install the Android development environment

To install the Android SDK you can follow the instructions.

After you have installed the SDK you can get the Android startup source code by:

- using git to clone it from the repository
- **OR** download it as a zip archive directly

When you get the startup code you can import it in your Android SDK (Eclipse) as an existing project.

Final setup step is to add a virtual device (on which you will run the code). You can find instructions here. The soruce code project is built for the version 4.3 (Android SDK 18), so be sure to add the virtual device with that version.

When you are done with all this, you should be able to right click on your project and select "Run As->Android Application". When the loading has finished you should see a simple, app with just "Hello world" on the screen.

Step 1. Understaning the code

To understand how files are organized you can read about Android projects. In our source code we have added some extra classes and resources:

- se.kth.csc.iprog.dinnerplanner.model package contains classes for the model (dish, ingredient and general dinner model) as well as interface that you need to implement to support the dinner functionalities (guests, selected dishes, etc.)
- se.kth.csc.iprog.dinnerplanner.android package contains the Activity classes (just one for now) that represent screens of your Android app, and the DinnerPlannerApplication class that is the main application class and has the instance of the model which you will use in each of your activity,
- se.kth.csc.iprog.dinnerplanner.android.view **(corrected link)** package contains classes (just one for now) that repesent the views
- drawable resoruces containing some pictures you can use for your dishes

Step 2. Modify the model

You need to modify the DinnerModel class and implement the IDinnerModel interface. To implement that interface you will need to add fields to the DinnerModel class that will hold the valuse for number of guest and dishes that were selected for the menu. Then you will implement the interface methods (getSelectedDish, getFullMenu, etc.) that will work with those fileds and return the expected values (see the comments in the IDinnerModel interface).

If you need a reminder on how interfaces work in Java look at the notes from Java lecture or the offical documentation.

Step 3. Layout the views

To develop the required screens you will be adding layouts to the project. When you open a layout xml (they are located in a layout folder) they will open in the GUI editor and there you can modify them and add other widgets, you can also switch to XML view to check how the actual declartive layout description looks or to have better control of what is changing.

The application and screens are loaded thorugh activities. Here it's important to understand the difference between the **conceptual view**, **Android View class** (and classes that inherit from it) and the **screen**.

The **screen** is everything that you see in your app at one time. One screen can have several **conceptual veiws** at the same time. For instance one part of the screen lists all the dishes and the other part of the screen gives an overview of the dinner (number of guests, total cost, etc.). You use diffrent layouting options in android to combine several views together. The **Android View class** is the main class from which all the UI components inherit (like java.awt.Component). Thus each Android widget/component (button, input box, etc.) is by definition a View. In the remaining of the lab, whenever we talk about the view we refer to conceptual view and it's classes that you will need to implement and **not** the Android View class (unless specifically stated).

Since some of the conceptual views can be repeated on several screens (for instance navigation) it would be useful to somehow reuse the components. The ideal way is to create each of these views as a separate layout file and then include them in each activitiy's layout file where they are needed. In the provided source code, you can see a activity_main.xml that is the XML layout for the main screen. The activity_main.xml includes example_view.xml that represents the initial example. You can add other views by choosing *File-New-Android XML File...* and choosing the layout for your view. You can now play in the GUI editor to modify the layout to fit your needs. Be sure to give your view an ID (you will see that example_view has this_is_example_view_id set) so you can load it from the code. You can do that by right-click on the top of the view and choosing *Edit ID...*

Each screen in Android application in practice has an Activity. Activites serve at minimum to load the view (we will see in Lab 3 how they deal with interaction as well). You should develop an Activity for each screen from the prototype. You can add an activity by choosing File-New-Other.. and then Android Activity in the wizard. When activity is created it also creates a default layout for it in the layout folder. In the layout of your new Activity you can include all the views you need on that screen.

ExampleView class containes the view code. You should create a simlar class for each of the conceptual views you will have so you separate any potential view modification code from the actual activity that uses the view. In the constructor of the view class you then do all the modification you need. Views are instatiated from the activity that uses them, like in our MainActiviy example:

```
// Creating the view class instance
ExampleView mainView = new ExampleView(findViewById(R.id.this_is_example_view_id));
```

Here comes the importance of defining the ID of your view layout, as mentioned earlier.

If you want to see how the new screen you make looks when running the application you can modify the "AndroidManifest.xml" file and find the following code:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

and move it form the "MainActivity" node to the activity that represents the screen you want to test.

Step 4. Load the demo data

After you have developed your views it is time to link them with the data. To do this you need to access the model. You can do this in each screen activity with this code:

```
DinnerModel model = ((DinnerPlannerApplication) this.getApplication()).getModel();
```

The model variable now holds the instance of the DinnerModel. In the model you can access the methods like model.getDishesOfType(Dish.STARTER) to get all the starter dishes. If you finished Step 2 , you should also be able to access methods for the dinner (getSelectedDish, getFullMenu, etc.). Use these methods in your view classes to fill the layout with the model data. Read more about accessing the view componentes.

You need to pass the model to the view and in the MainActivity example we modify the view instatization code to:

```
// Creating the view class instance and passing the model
 ExampleView mainView = new ExampleView(findViewById(R.id.this_is_example_view_id), model);
```

Finally, you need to modify the ExampleView constructor to accept the passed model and then modify the layout setup code to use the information from the model. Let's show how the modifed constructor of ExampleView would look like

```
DinnerModel model; // we add class variable

public ExampleView(View view, DinnerModel model) {

 // store in the class the reference to the Android View
 this.view = view;

 // and the reference to the model
 this.model = model;

 TextView example = (TextView) view.findViewById(R.id.example_text);
 example.setText("Total price: " + model.getTotalMenuPrice());

 // Setup the rest of the view layout
}
```

And this is it. For now you just need to have layouts as (or close to) shown in prototype and load the actual values from the model. In the next lab we will see how we can make the app interactive (react to changes you make) and play a bit with gestures for navigation.

How (desktop)

For the desktop version you can find the prototype here:

https://kth-csc.mybalsamiq.com/projects/dh2641-vt14-lab2assignment-desktop/grid

As mentioned in the lectures we provide the instructions for Java Swing development. You are free to use any other desktop platform as long as you follow the key principles (model-view separtion in this lab).

Step 0. Install the development environment

You can download Eclipse here. If you want to use some other IDE for Java instead of Eclipse you can do that. Most modern IDEs should not have a problem with reading the Eclipse project source files, but you might need to make some modifications in the settings.

After you have installed the development environment you can get the Swing project startup code by:

- using git to clone it from the repository
- **OR** download it as a zip archive directly

When you get the startup code you can import the code in your Eclipse as an existing project.

When you are done with all this, you should be able to right click on your project and select "Run As->Java Application". When the loading has finished you should see a small window with "Hello world" text.

Step 1. Understaning the code

The code files are located in src folder and then respective packages

- se.kth.csc.iprog.dinnerplanner.model package contains classes for the model (dish, ingredient and general dinner model) as well as interface that you need to implement to support the dinner functionalities (guests, selected dishes, etc.)
- se.kth.csc.iprog.dinnerplanner.swing package contains the DinnerPlanner class that is the main application class which runs the application and containes the instance of the model
- se.kth.csc.iprog.dinnerplanner.swing.view package contains classes (just one for now) that repesent the views
- images folder contains some pictures you can use for your dishes

Step 2. Modify the model

You need to modify the DinnerModel class and implement the IDinnerModel interface. To implement that interface you will need to add fields to the DinnerModel class that will hold the valuse for number of guest and dishes that were selected for the menu. Then you will implement the interface methods (getSelectedDish, getFullMenu, etc.) that will work with those fileds and return the expected values (see the comments in the IDinnerModel interface).

If you need a reminder on how interfaces work in Java look at the notes from Java lecture or the offical documentation.

Step 3. Layout the views

To develop the required screens you will need to develop different views and combine them together.

Each view should be in its own class and extend JPanel or any other appropriate Swing component. In the constructor of your views you should setup the appropiate settings (size, position, look and feel, etc.) and components of the view.

If you want to check how your view looks like when it's run, you can modify the DinnerPlanner class's main method. There you will find the following code:

```
  //Creating the first view

  MainView mainView = new MainView();

  //Adding the view to the main JFrame
```

```
  dinnerPlanner.getContentPane().add(mainView);
```

Here instead of the MainView you initalize your new view and add it to the dinnerPlanner frame. And then run the application.

Step 4. Load the demo data

After you have developed your views it is time to link them with the data. You need to pass the data model to the views. To do this you can modify your view's constructor and add the model as the parametar.

In your view you can then access the methods like model.getDishesOfType(Dish.STARTER) to get all the starter dishes. If you finished Step 2 , you should also be able to access methods for the dinner (getSelectedDish, getFullMenu, etc.). Use these methods in your view to fill it with the model data.

Finally when you are instantiating the view (for now in the DinnerPlanner class) you need to pass the model as well.

To show this on example, let us add the model to the MainView and instead of "Hello world" write the total price of the menu (Step 2 required for this to work). In MainView.java we modify the constructor:

```
DinnerModel model; // we add class variable

public MainView(DinnerModel model){

  // store in the class the reference to the model
  this.model = model;

  label.setText("Total price: " + model.getTotalMenuPrice());

  // Add label to the view
  this.add(label);

  // Setup the rest of the view layout

}
```

and in the DinnerPlanner.java class' main method we modify the view creation:

```
  //Creating the first view

  MainView mainView = new MainView(dinnerPlanner.getModel());
```

And this is it. For now you just need to have the views as (or close to) shown in prototype and load the actual values from the model. In the next lab we will see how we can make the application interactive (react to changes you make on the screens) and loading different screens.

Uploading the results by 9th Feb

When you have finished with developing either mobile or desktop application, just push your changes to GitHub and submit the repository link in the comment of **Bilda assignment** Lab 2: layout and model.

If you didn't use GitHub, **zip** your whole project folder and upload it to the Bilda assignment.

Show earlier events (2) ›

Teacher Filip Kis changed the permissions | 29 January 21:09

Kan därmed läsas av studerande och lärare och ändras av lärare.

... or write a new post

Students, teachers and assistants of this course can read.

Last changed: 2014-02-05 07:29. Show versions

Tags: None so far. Add

Follow this page Report abuse

http://www.kth.se/