Lab 2: Layout & Model-View separation

Goals

- practice implementing a layout from the prototype
- practice layout 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 assignment you learned and experienced fast and iterative prototyping. Now it is time to start implementing one of the prototypes. In professional environments 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 prepared a prototype that we thought fits best to our requirements and requirements of the course (remaining labs).

Your assignment in this lab is to implement the layout of the this prototype and extend the model to fit the needs of the application. You can choose between implementing a **desktop, mobile and web** application.

For **DH2642** students it is recommended that they do web application as Lab 4 and 5 will be only web based.

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. This code should not know anything about the layout (graphics).

One of the requirement for this and all the future labs and project is to use version control system. This should help you in sharing your work among the team mates. It will also help us to evaluate the contribution of each team member. Finally, code sharing, especially using git and GitHub is a common practice in the industry and valuable skill to develop early on.

How (web)

For the web version of the Dinner Planner we have prepared the following solution.

You can use the commenting option on the balsamiq site to ask questions and clarifications on the prototype.

Step 0. Installing the environment

First get started by using git to clone the base code from the repository.

To work with the code you can use any plain text editor (my favorite is SublimeText, but there also examples like Kate for Linux, TextMate for Mac and Notepad++ for Windows) or HTML editor.

To run HTML code you just need your web browser.

Step 1. Understanding the code

When you look at the code you will find following content:

- index.html - the only HTML page you will have in this project (though while testing you can create more to make it easier). You will need to implement the skeleton of the layout there and then through code (JavaScript) create the rest.
- js/model/dinnerModel.js - is JavaScript file that contains the model code.
  The file contains the model (dish, ingredient and general dinner model) as well as methods and comments that you need to fully implement to support the dinner functionalities (guests, selected dishes, etc.)
- js/view/ - here you can find a JavaScript code of a example view. The view sets up some initial components and their values.
- js/app.js - this is the overall code of the application. It is responsible for initial setup of the app (when the page loads for the first time).
- images folder contains some pictures you can use for your dishes

You can test the source code by running the index.html file in your browser. You should see a "Hello world" text in the place where number of guests and two buttons for increasing and decreasing the number of guest (though they do not do anything yet).

Step 2. Modify the model

You need to modify the dinnerModel.js and add the code to implement the needed functionality. To do that you will need to add fields that will hold the values for *number of guest* and *dishes that were selected for the menu*. Then you will implement the methods (getSelectedDish, getFullMenu, etc.) that will work with those fields and return the expected values (see the comments in the code).

If you need a reminder on how JavaScript works check the lecture materials and/or literature for resources on JavaScript.

Step 3. Layout the views

Your next task is to implement the 5 different screens from the prototype as closely as possible. We suggest taking the advantage of Bootstrap framework for good layout and styling solutions. The framework is already included in the index.html, so you just need to check the documentation and start using the examples.

In the prototype you will notice that there are a minimum of 6 different views (more if you consider single dish in the lists as a separate view) in 5 different screens. All the HTML

code for the views will after Lab 3 need to be in one HTML file and you will just use JavaScript to populate and generate the code that you need. However, for now you can, for easier development and division of the work, create each screen in a separate file.

If you do use Bootstrap, we require that you change some of the default values. For instance, you can change the default color of the button, use different font, etc. To achieve this, you will need to create your own stylesheet, include it in index.html and in it override bootstrap classes you want to change.

Step 4. Load the demo data

You can take a exampleView.js as an example of how the view JavaScript code should look like. In the beginning of the view constructor function you need to get the relevant objects (by their respective IDs) or you create them.

Typically, the components that are static (their quantity doesn't change) you create declaratively in the HTML and just select them in your code using container.find() method. The dynamic components (like dishes or ingredients) in different lists you will need to create procedurally through code.

After you create or get the elements, you need to load the data from the model. First you need make the model available to the view. To achieve this, you need to add it to the view constructor method and pass it in the view creation. Afterwards you access the model in the view and populate the view with the values from the model.

To show this on example, let us add the model to the exampleView.js and instead of "Hello world" write the number of guests (Step 2 required for this to work). In exampleView.js we modify the constructor:

```
//ExampleView Object constructor
var ExampleView = function (container,model) {

 // Get all the relevant elements of the view (ones that show data
 // and/or ones that responded to interaction)
 this.numberOfGuests = container.find("#numberOfGuests");
 this.plusButton = container.find("#plusGuest");
 this.minusButton = container.find("#minusGuest");

 this.numberOfGuests.html(model.getNumberOfGuests());

}
```

in the app.js we modify the view creation:

```
var exampleView = new ExampleView($("#exampleView"), model);
```

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.

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 separation 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.

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

When you are done with all this, you should be able to run project by selecting "Run->Run 'app'". When the loading has finished (it takes quite some time to load for the first time) you should see a simple, app with just "Hello world" on the screen.

Step 1. Understanding 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 package contains classes (just one for now) that represent the views
- drawable resources 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 values 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 fields 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 declarative layout description looks or to have better control of what is changing.

The application and screens are loaded through 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 views** 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 different layout 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 activity'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. Activities 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 contains the view code. You should create a similar 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 instantiated 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 components.

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-vt15-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.

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 values 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 fields 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.

What to deliver on 13th Feb

When you have finished with developing, just push your changes to GitHub and present your results to assistants in one of the Lab sessions.

Visa tidigare händelser (4) ›

Assistent **Filip Kis** kommenterade | 2 februari 09:13

Hi Sydney,

It has been fixed and you should be able to access it now.

... eller skriv ett nytt inlägg

Alla användare med KTH-konto får läsa.

Senast ändrad: 2015-02-06 15:45. Visa versioner

Taggar: Saknas än så länge.

Följ denna sida Anmäl missbruk

http://www.kth.se/