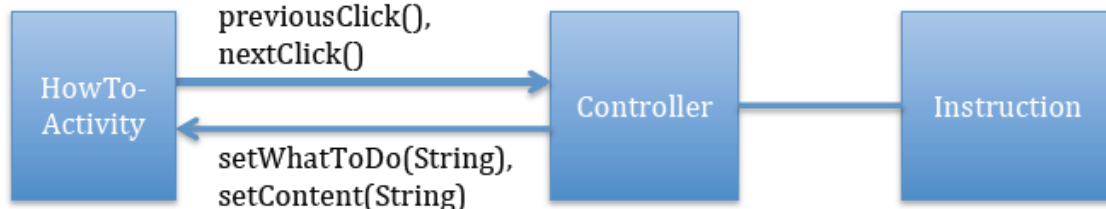


Laboratory 2a

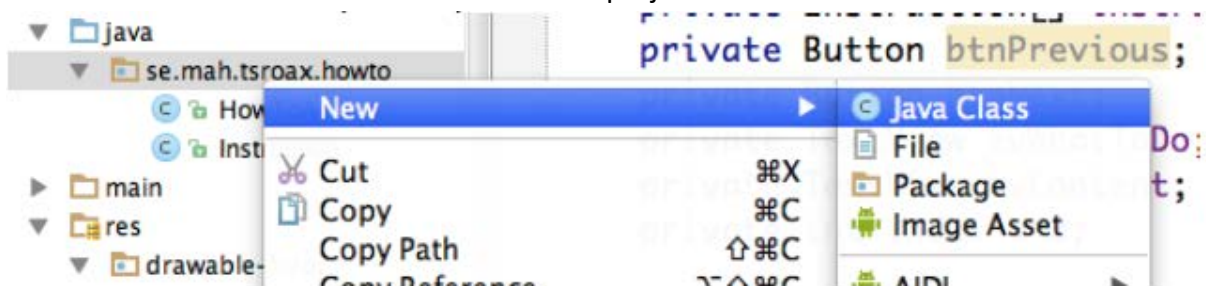
You will develop an app called *HowTo*. This app contains one *Controller* class that will manage the logic of the program. The UI class should only report to the *Controller* class when a button is clicked (calling `click()` and `nextClick()` in the controller class). It should be able to display the results that the user will see (calling `setWhatToDo(String)` and `setContent(String)` in *Controller*).



Below is a detailed description how to solve this. Follow these guidelines and think about the changes you are making.

To-do

- The ***Controller*** class must be created in the project.



- *Controller* should find instructions and items. Therefore, it contains variables *instructions* and *index*, which are now located in *HowToActivity*. Move them to the *Controller* class.

Create a reference to *HowToActivity* (ui) and write a constructor that inputs an instance of *HowToActivity*.

```
package se.mah.tsroax.howto;

public class Controller {
    private Instruction[] instructions = new Instruction[3];
    private int index = 0;
    private HowToActivity ui;

    public Controller(HowToActivity ui) {
        this.ui = ui;
    }
}
```

Save this instance in your reference (ui) variable, so that *HowToActivity* methods can be called from *Controller*.

- Add *previousClick* and *click Next Click* methods in the *Controller*. These methods are very similar to the event handlers in *HowToActivity*:

```
public void previousClick() {  
    index--;  
    if(index<0)  
        index = instructions.length-1;  
    ui.setWhatToDo(instructions[index].getWhatToDo());  
    ui.setContent(instructions[index].getContent());  
}
```

The main difference is that the method ends with two calls to *ui.setWhatToDo* and *ui.setContent*. This way the controller calls UI to display the strings passed as arguments. These methods are marked in red because they have not been created yet in *HowToActivity*.

If you click on the red text and press Alt+Enter, a menu will appear. Click on *Create Method* and it will be declared automatically in *HowToActivity*. Now it is up to you to fill it in.

Now code the method *Next Click* analogously.

- *Controller* must create *Instruction* objects and place them in the array *instructions*. This should be done in the constructor. Add a method call to *initializeResources* in the constructor. Move this method from *HowToActivity* to *Controller*.

```
public Controller(HowToActivity ui) {  
    this.ui = ui;  
    initializeResources();  
}  
  
private void initializeResources() {  
    Resources res = getResources();  
    String whatToDo = res.getString(R.string.what_to_do);  
    String content = res.getString(R.string.content);  
    instructions[0] = new Instruction(whatToDo, content);  
    instructions[1] = new Instruction(res.getString(R.string.what_...  
    instructions[2] = new Instruction(res.getString(R.string.what_...  
}
```

The method *getResources* is still located in *HowToActivity*, but it is reachable from *Controller* using *ui*:

```
Resources res = ui.getResources();
```

Now that *Controller* is finished, some things must be removed from *HowToActivity*.

- Remove instructions and index. Add a reference to a Controller object.

```
public class HowToActivity extends Activity {
    private Controller controller;
    private Button btnNext;
    private TextView tvWhatToDo;
    private TextView tvContent;
```

- Create a *Controller* object in *onCreate* and assign it to the reference. This way you can notify Controller when the user clicks a button.

```
    registerListeners();
    controller = new Controller(this);
}
```

The argument *this* is the reference to *HowToActivity*. Remember that this reference is stored in *Controller.ui*.

- Remove the call to *initializeResources* in *onCreate*. Remember that this is done now in *Controller*. Also remove *initializeResources*.
- The method *previousInstruction* shall notify the *Controller* that the user clicked on *Previous*.

```
public void previousInstruction(View view) {
    controller.previousClick();
}
```

- Manage events when *Next* button is clicked.

```
private class NextListener implements View.OnClickListener {
    public void onClick(View v) {
        controller.nextClick();
    }
}
```

- Make sure that the methods *setWhatToDo* and *setContent* place make their input strings visible in the corresponding Views.

```
public void setWhatToDo(String whatToDo) {
    tvWhatToDo.setText(whatToDo);
}

public void setContent(String content) {
    tvContent.setText(content);
}
```

Laboratory 2b

Change the *HowTo* application so that it contains the *Activity* class, a *Fragment*, and a *Controller*. Do it in a new project but reuse code from Exercise 2a.

Laboratory 2c

You will produce a simple application that consists of an Activity, two fragments, and a Controller. One of the fragments must contain a TextView, and the second fragment must have a Button. The Text View will display the number of clicks made on the Button component.

Follow these guidelines:

Use an inner class for the management. The inner class must be in the fragment with the Button.

Controller will be notified when a click happens. The Fragment with the button must have a reference to the Controller.

Controller must notify the fragment with a TextView the value to be displayed. Turn integers into string as follows:

```
int a = 10;  
String b = String.valueOf( a ); // b = "10"
```

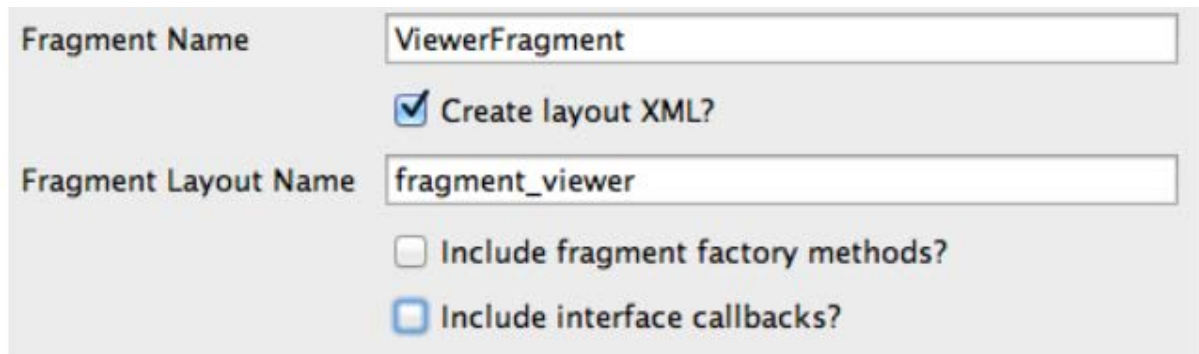
Controller must have a reference to the fragment with a TextView.

In order to test your solution:

1st Create a project - Blanc Activity

2nd Create **fragment** with a Text View.

Right-click on the package with source code and select: New - Fragment - Fragments (Blank). Remove both marks in the bottom checklist and give it a proper name.



Fragment Name	ViewerFragment
<input checked="" type="checkbox"/> Create layout XML?	
Fragment Layout Name	fragment_viewer
<input type="checkbox"/> Include fragment factory methods?	
<input type="checkbox"/> Include interface callbacks?	

Add the Text View in the right layout xml file. Add a method to Controller that can change the content of this Text View.

3rd Create the **Controller**. Add the appropriate instance variables (2) and a suitable constructor. Create a method that will be called upon button clicks, updating the Text View with the number of clicks.

4th Create a **fragment** with a Button. Add the button in the corresponding layout xml and remove the default Text View - component. Add an instance variable r that references to the Controller and add a setController method. Arrange event handling so that the Controller gets notified when a click happens.

5th Add each fragments a string tag variable.

```
public static final String TAG = "FirstFragment";
```

Provide a meaningful *id* to each fragment in its layout xml.

6th Set all object references in Activity.*onCreate*.

Activity might look like:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_lab2b);  
    initializeSystem();  
}  
  
private void initializeSystem() {  
    FragmentManager fm = getFragmentManager();  
    ViewerFragment viewer = (ViewerFragment)fm.findFragmentById(R.id.frViewer);  
    InputFragment input = (InputFragment)fm.findFragmentById(R.id.frInput);  
    Controller controller = new Controller(viewer);  
    input.setController(controller);  
}
```

Laboratory 2d

The RockPaperScissors game: All images, xml files, and source code files for today's lecture are on the course page. Create a project, RSPStaticFragment, gradually building up the app with these files.

Work preferably in groups. These exercises can be quite demanding and many things can go wrong.