Porting a Swing application to JavaFX & DukeScript using NetBeans

Ioannis Kostaras

NetBeans Day Athens

26 Aug 2016

Agenda

- Prerequisites
- Port to JavaFX
- Port to DukeScript

References:

- **TodoFX** http://wiki.netbeans.org/TodoFX
- **TodoFX2** http://wiki.netbeans.org/TodoFX2
- **TodoDS** http://wiki.netbeans.org/TodoDS

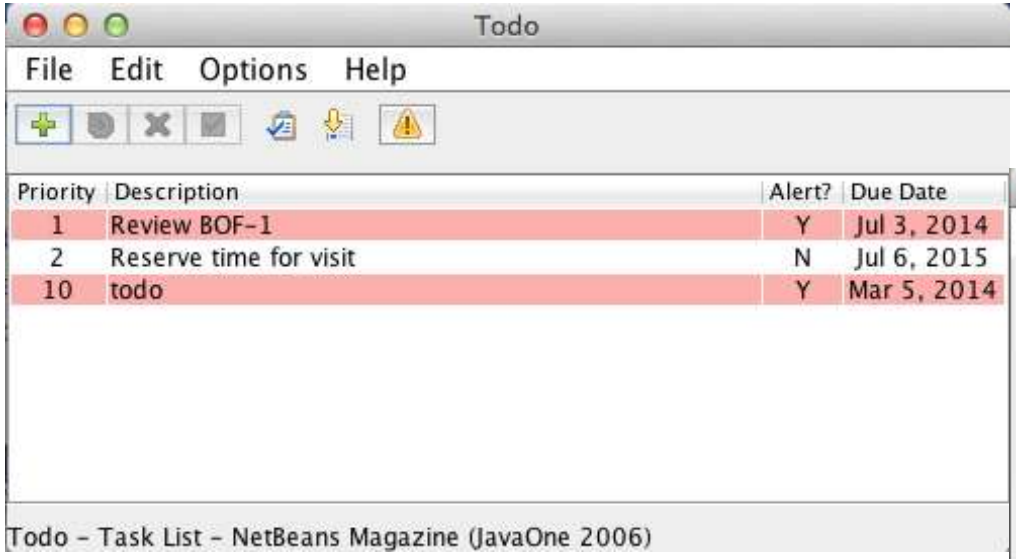
Prerequisites

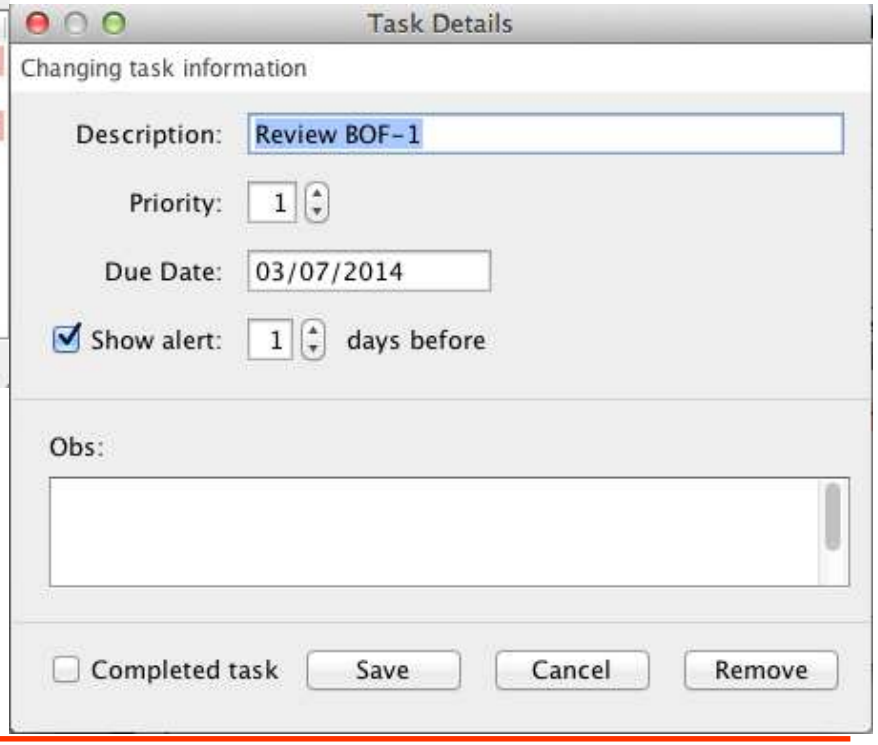
- ToDo Swing application
- NetBeans 8.0.2 or later
- JDK 8
- HSQL DB
- Gluon SceneBuilder
- DukeScript plugin for NetBeans

- Create a Project Group (*Optional*)

To-do Swing application

TodoFX & TodoDS



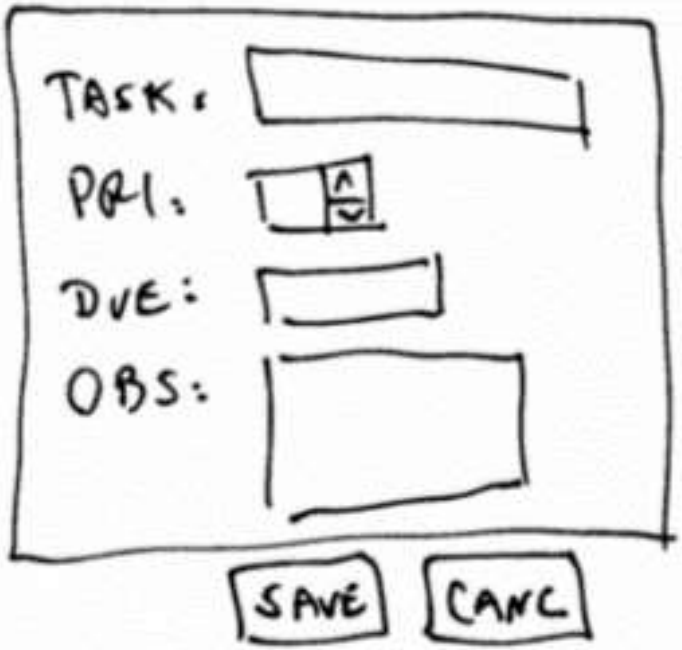


Requirements

- Tasks should have a *priority*, so users can focus on higher-priority tasks first.
- Tasks should have a *due da*te, so users can focus on tasks that are closer to their deadline.
- Tasks that are either late or near their deadlines should have visual cues.
- Tasks can be marked as *completed*, but this doesn't mean they have to be deleted or hidden.

Requirements (cont.)

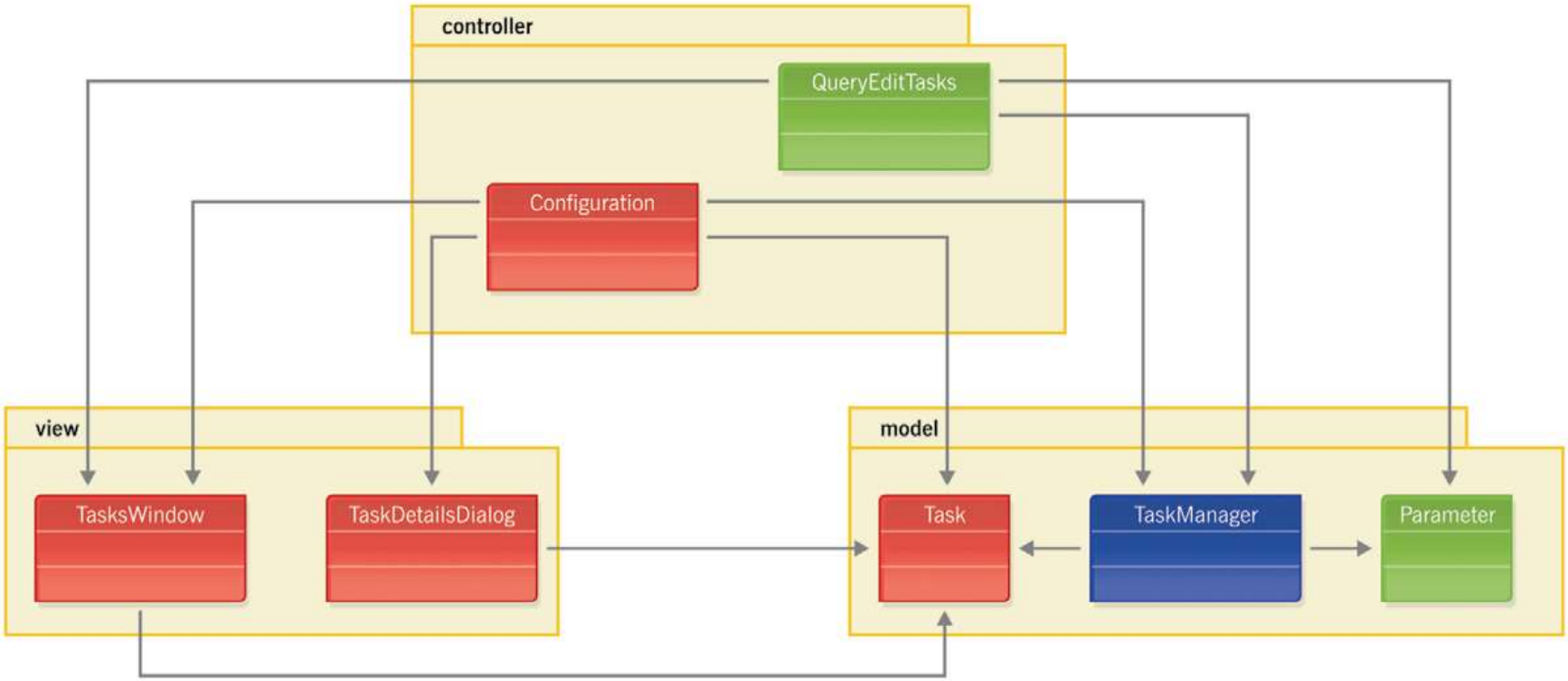
- The to-do application consists of two main windows:
  - A task list window and
  - A task editing form



Steps

1. Build a "static" visual prototype of the GUI, using the Gluon SceneBuilder to build a task list window.
2. Build a "dynamic" prototype of the application, coding user interface events and associated business logic and creating customized GUI components as needed.
3. Code the persistence logic by modeling the classes and the database.
4. Add more features.
5. Repeat the above steps to build it in DukeScript.

Swing ToDo application Architecture



TODOFX

Integrate Gluon SceneBuilder with NetBeans

- In NetBeans, navigate to `Tools` → `Options` → `Java` → `JavaFX` (Windows/Linux) or to `NetBeans` → `Preferences` → `Java` → `JavaFX` (MacOSX) and click **Activate** if JavaFX support is not yet activated.
- After activation is finished, set the *Scene Builder Home* to be the Gluon directory. Some Windows installers install SceneBuilder without asking you for a directory. SceneBuilder can be found in `C:\Users\<YourUser>\AppData\Local\SceneBuilder`.
- Click on **OK** and you are ready to start.

STEP 1

Build a Static Prototype of the GUI

Step 1: Build a Static Prototype of the GUI

There are two ways to develop a JavaFX application:

- programmatically, using Java (JavaFX Application),
- declaratively, declaring the GUI in a special XML format, called FXML (JavaFX FXML Application).

We shall choose the second way here, which has also the advantage that you can use the SceneBuilder to graphically design the GUI.

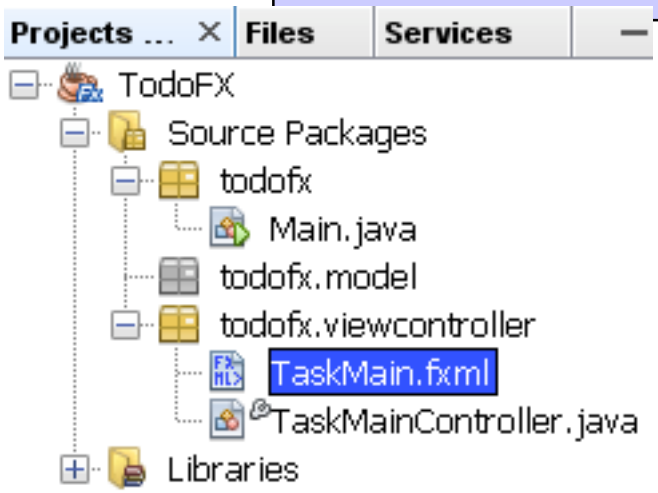
1. **File → New Project → JavaFX → JavaFX FXML Application → Next**
2. **Project Name: TodoFX FXML name: TaskMain**

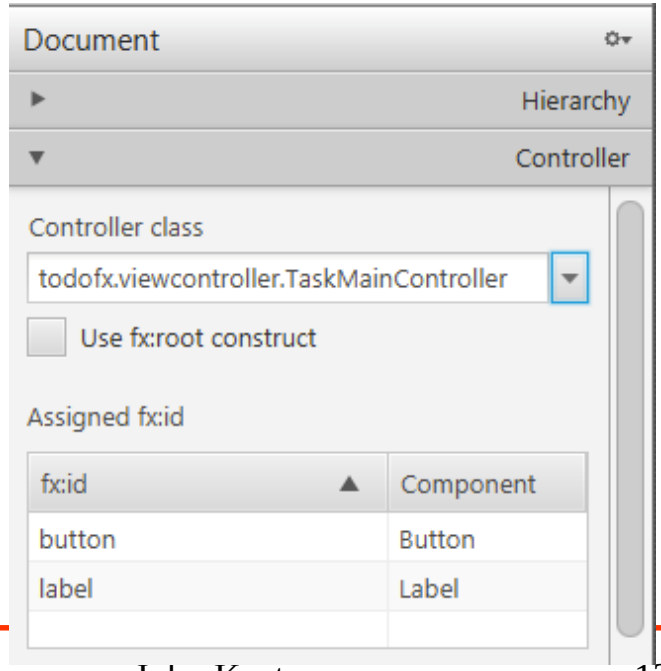
Application Class: todofx.Main

Step 1: Build a Static Prototype of the GUI

TodoFX & TodoDS

- With Scenebuilder you can quickly design your GUI by using simple drag & drop of widgets, in a similar way that Matisse allows you to build Swing GUIs.
- In NetBeans, refactor the project following the Model-View-ViewModel architectural pattern.
- In SceneBuilder, refer to the correct controller path after the refactoring



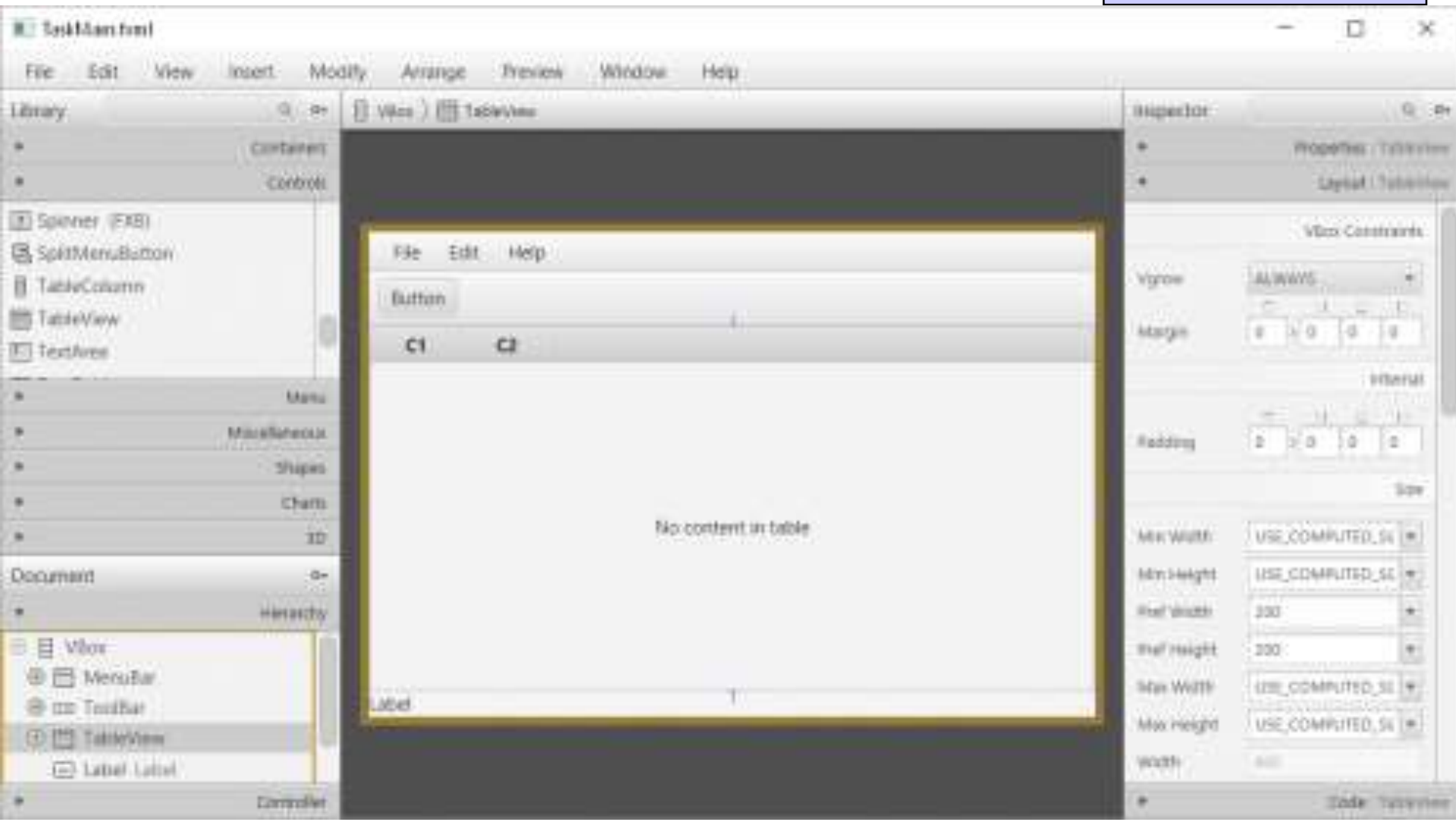


Step 1: Build a Static Prototype of the GUI

1. In Document area of SceneBuilder, click on *Hierarchy*.
2. Select *AnchorPane* and delete it from the popup menu (right-click)
3. On the upper left (Library) area, select the *VBox* from the list of Containers and drag it in the main area or inside the *Document/Hierarchy*.
4. Adjust its preferred size in Inspector/Layout.
5. Add on the VBox a **MenuBar** (Controls), a **ToolBar** (Containers), a **TableView** (Controls) and a **Label** (Controls).
6. Select the TableView and click on Layout.
7. Change Vgrow to **ALWAYS**.
8. Select **Preview → Show Preview** in Window in order to see how the window looks like.

Step 1: Build a Static Prototype of the GUI

TodoFX & TodoDS



Step 1: Build a Static Prototype of the GUI

1. Drag Menu Items, *CheckMenuItems* and *SeparatorMenuItems* on MenuBar and rename them appropriately. (Menu **Edit → Duplicate** or **Ctrl+D** can be useful.)
2. Drag Buttons and CheckButtons on the Toolbar.
3. Drag TableColumns onto the TableView and rename them appropriately.
4. Select the TableView and choose constrained-resize for the Column Resize Policy (under Properties). This ensures that the colums will always take up all available space.

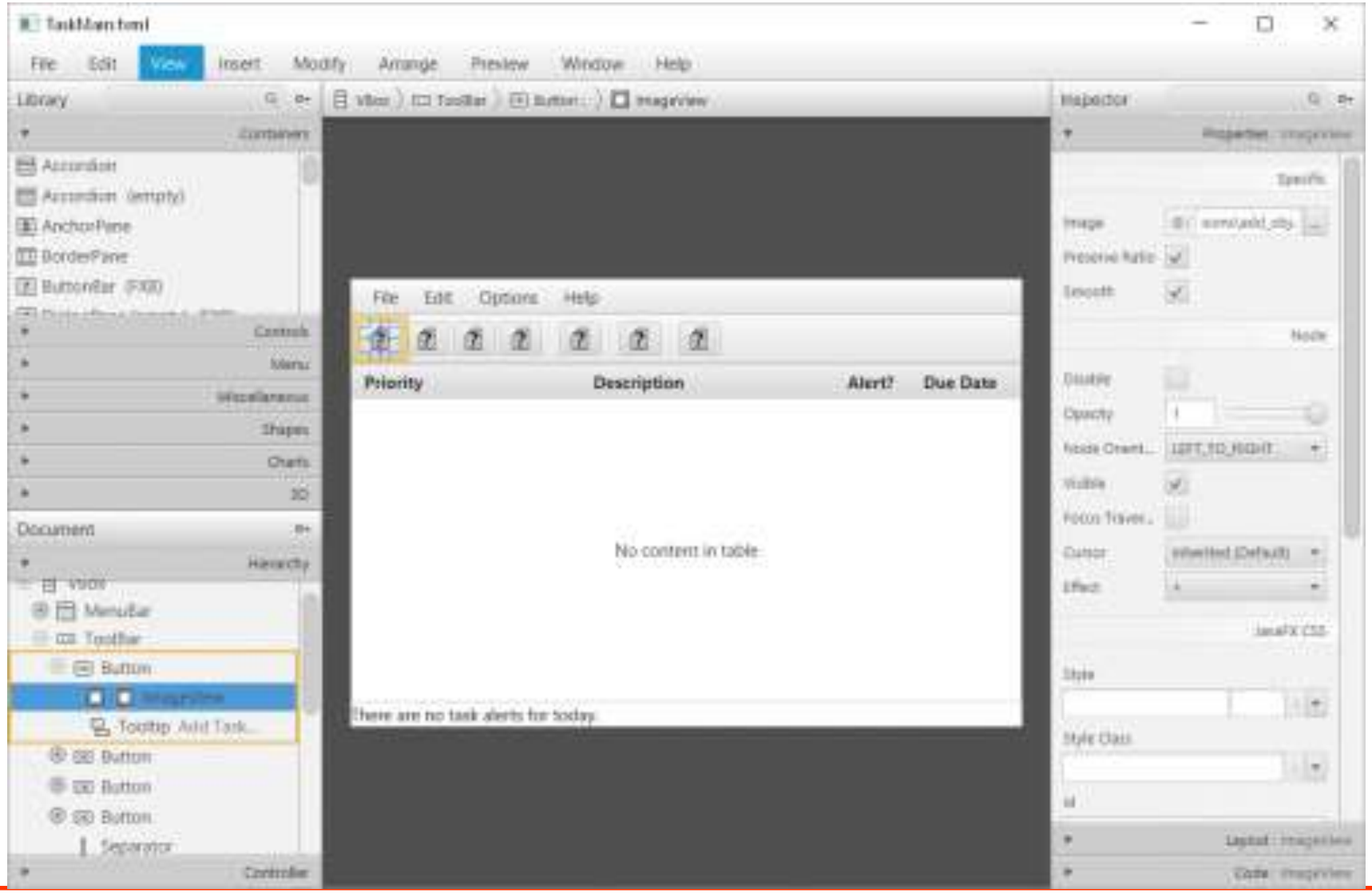
Step 1: Build a Static Prototype of the GUI

To create buttons with only image icons for the toolbar, we need to do some tricks:

1. Copy the icons folder of the original Todo application inside `TodoFX/resources/icons`.
2. Right-click on TodoFX project and select **Properties**.
3. Select category *Sources* and under *Source Package Folders* click on **Add Folder**.
4. Add the resources folder
5. Drag *ImageView* widgets and drop them inside the Buttons.
6. Select an *ImageView* and set its Image property. Click on the small star icon next to the ... button, and select *Switch to classpath relative path*. Then click on the ... button and select the respective icon (e.g. `icons/add_obj.gif`).
7. To verify that you have selected the correct path, in NetBeans, right-click on the `TaskMain.fxml` in NetBeans and select **Edit**. Verify e.g. that `<Image url="@/icons/add_obj.gif">` for the specific button.

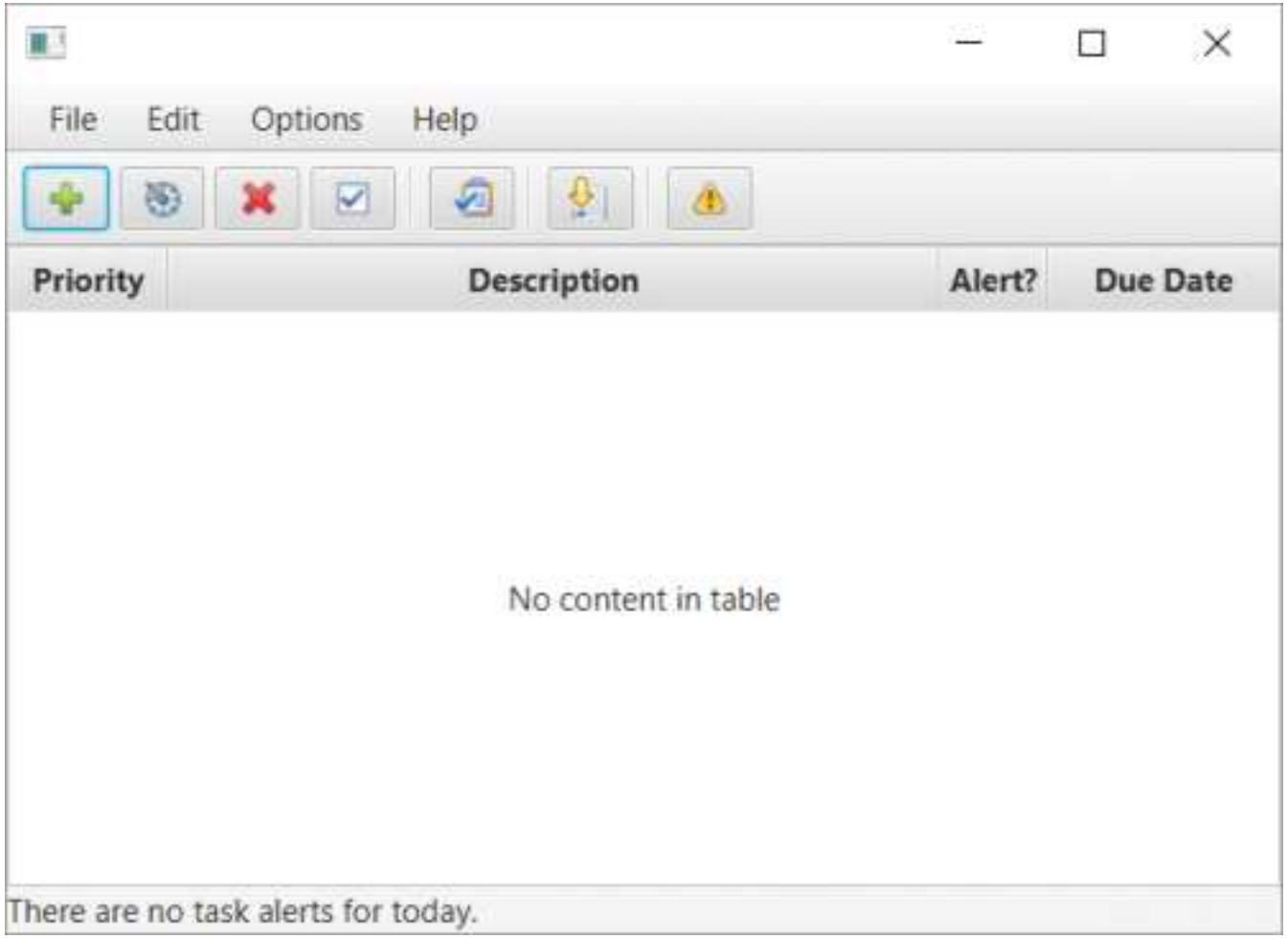
Step 1: Build a Static Prototype of the GUI

TodoFX & TodoDS



Step 1: Build a Static Prototype of the GUI

TodoFX & TodoDS



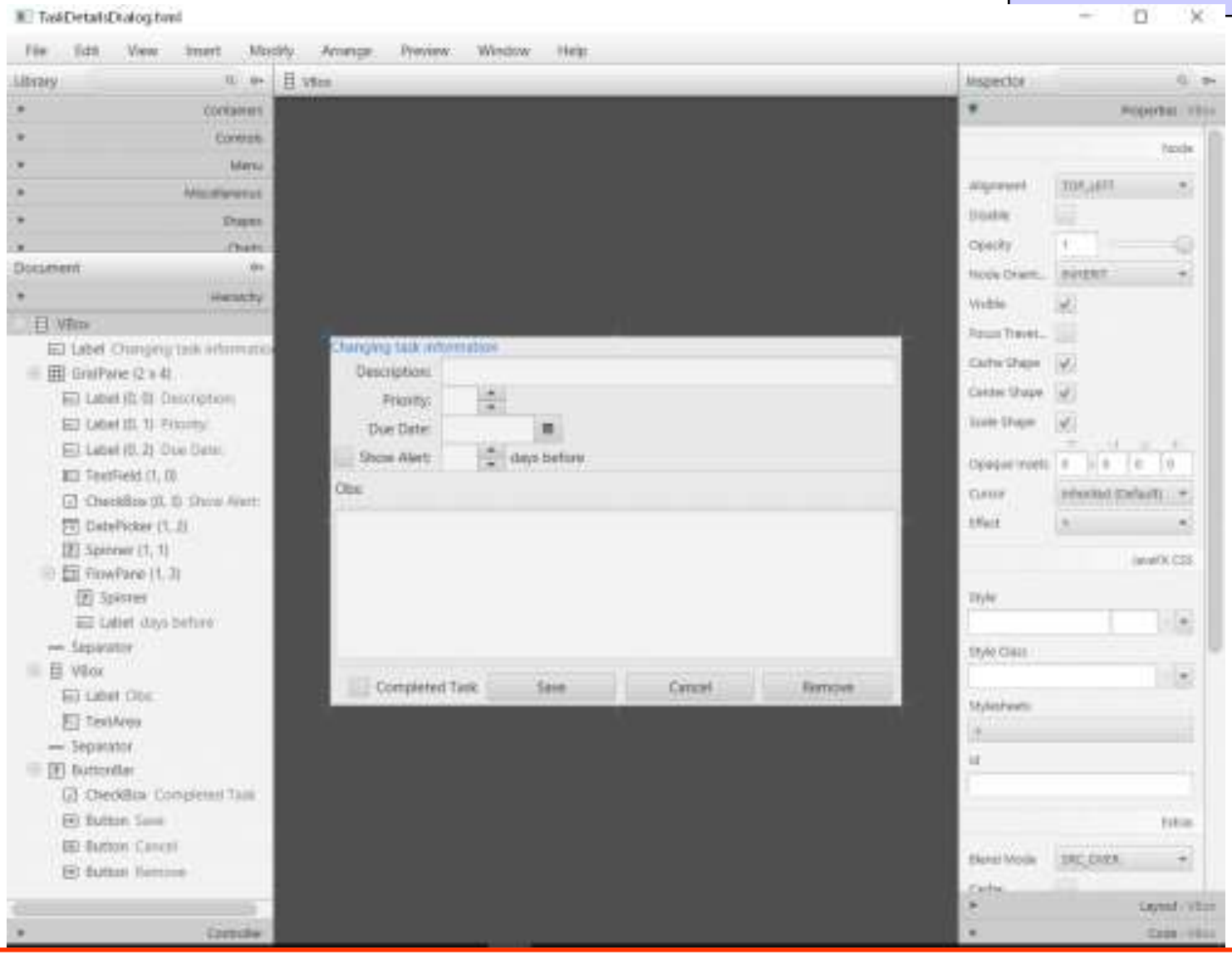
Step 1: Build a Static Prototype of the GUI

Let's design the Task Details dialog box, too.

1. Right-click on the `viewcontroller` package and select **New → Other → JavaFX → Empty FXML**. Click **Next**.
2. Provide `TaskDetailsDialog` as the FXML Name. Click **Next**.
3. Check **Use Java Controll**er and accept the default. Click **Next**.
4. Click **Finish**.

Step 1: Build a Static Prototype of the GUI

TodoFX & TodoDS

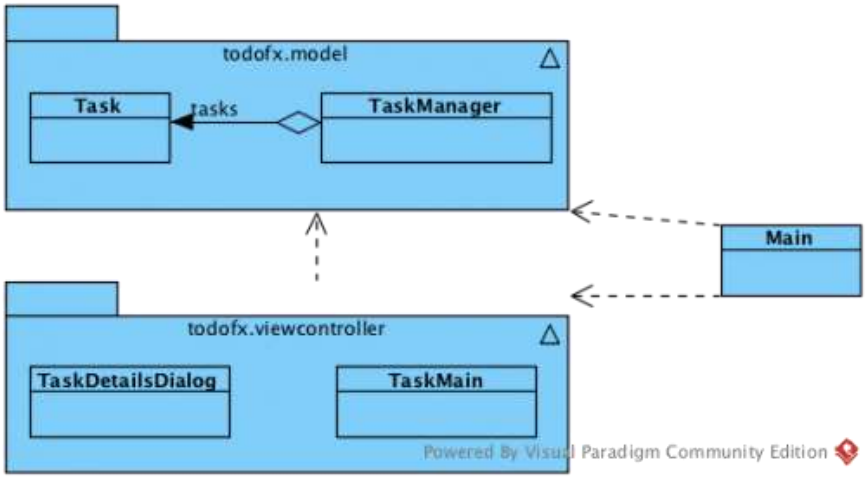


STEP 2

Build a Dynamic Prototype

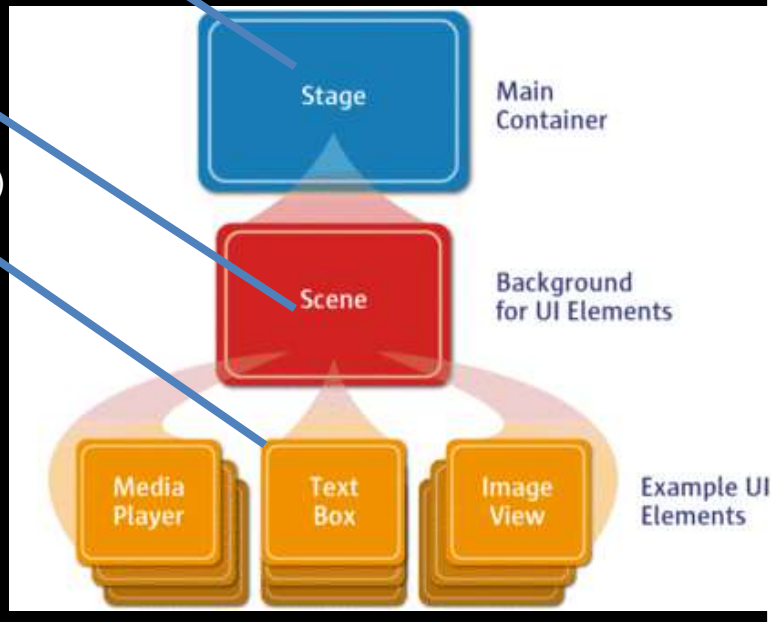
Step 2: Build a Dynamic Prototype

- implement as much user interaction as possible without using persistent storage or implementing complex business logic.
- Value Object (VO): `Task`
- Data Access Object (DAO): `TaskManager`



todofx.Main


```
public class Main extends Application {
  @Override
  public void start(Stage stage) throws Exception {
    Parent root = FXMLLoader.load(
        getClass().getResource("viewcontroller/TaskMain.fxml"));
    Scene scene = new Scene(root);
    stage.setScene(scene);
    stage.show();
  }
  public static void main(String[] args)
  {
    launch(args);
  }
}
```



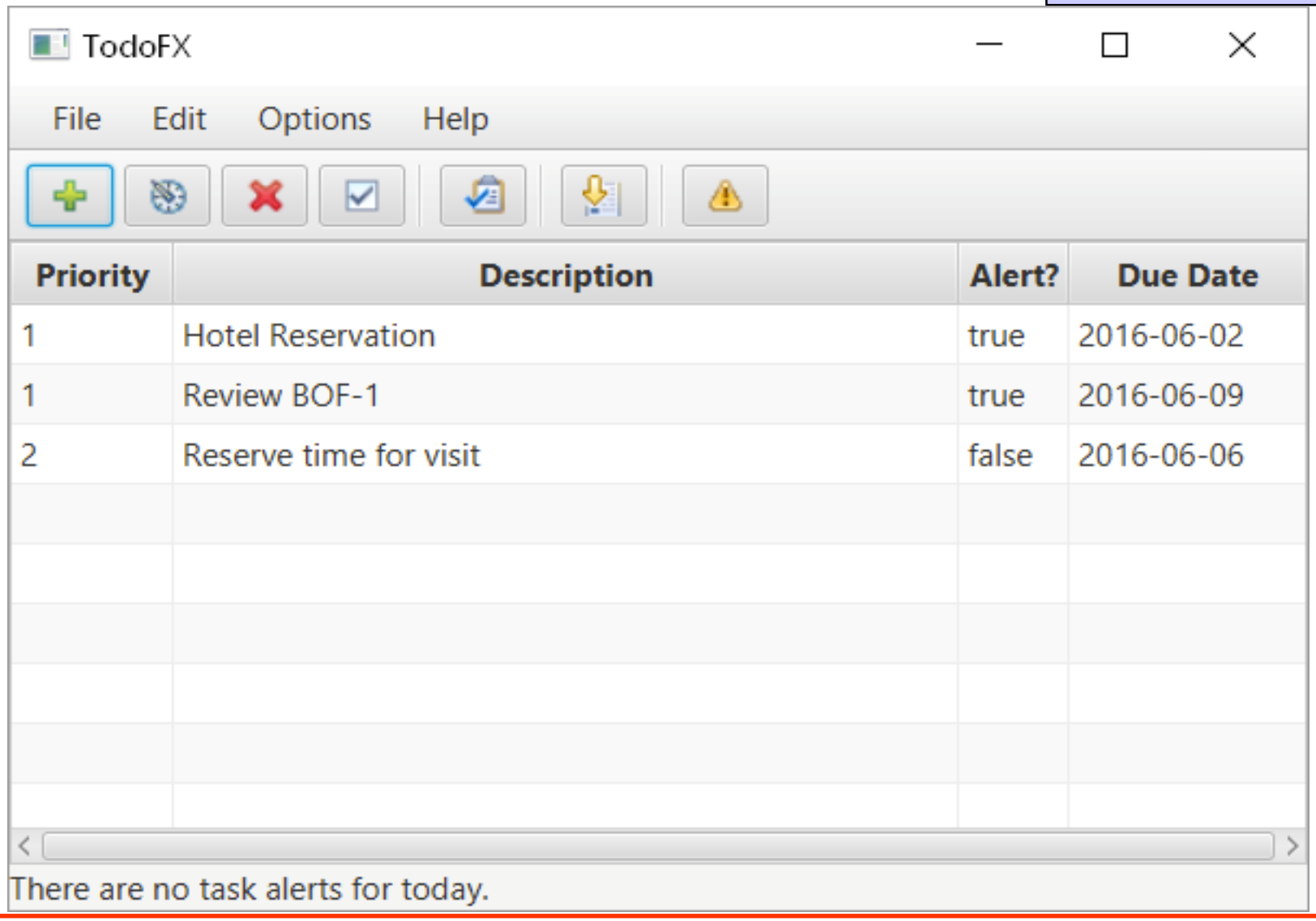
Step 2: Build a Dynamic Prototype

① Display the visual cues for late and completed tasks.
② Handle events:
   a) Handle action events to sort and filter the tasks list.
   b) Handle action events to create, edit, and remove tasks.
③ Add an in-place property editor for dates.

1. Copy `Task` class from **Todo** application
2. Create `TaskWrapper` class to wrap `Task` attributes to JavaFX properties
3. Create `TaskManager` class
4. Create `TaskListWrapper` a wrapper of `TaskManager` that uses `ObservableList<TaskWrapper>`
5. Update `TaskMainController`
6. Build the `TaskDetailsDialogController`

Step 2: Build a Dynamic Prototype

TodoFX & TodoDS

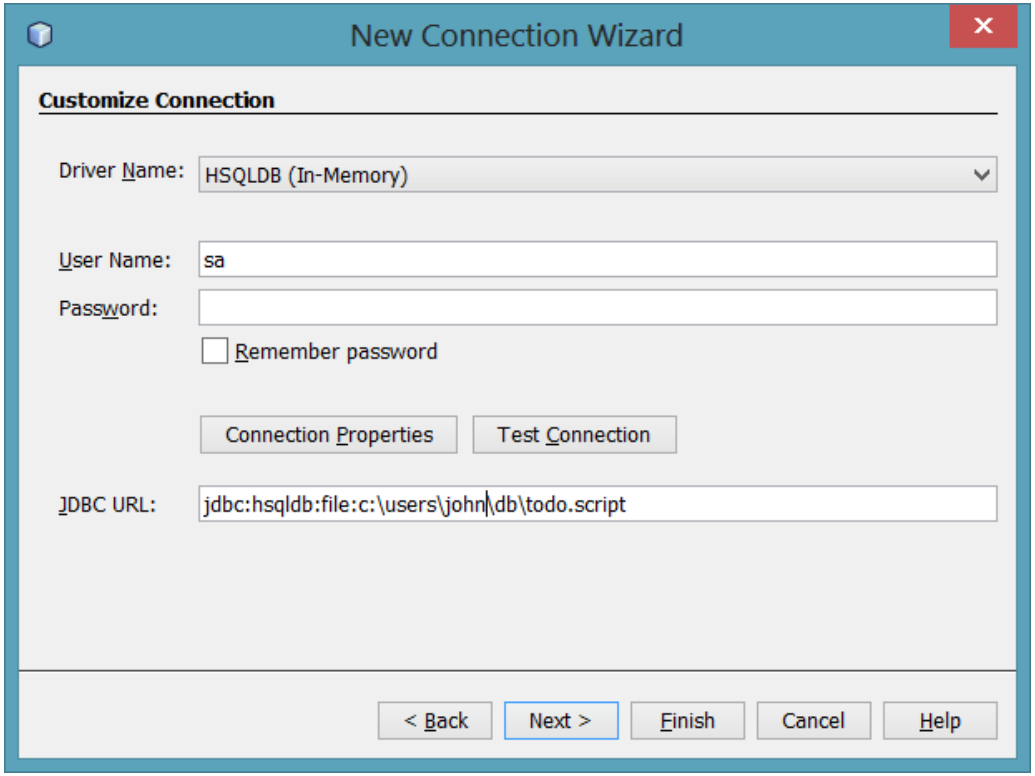


STEP 3

Add Persistence

Step 3: Add Persistence Logic

1. Add `hsqldb.jar` to your project's libraries (`lib`) directory:
2. Right-click on Libraries
3. Add JAR/Folder
4. Navigate to the folder where you downloaded HSQLDB and inside `lib`
5. Choose `hsqldb.jar` and choose **Copy to Libraries Folder**
6. Click **Open**



Step 3: Add Persistence Logic

TodoFX & TodoDS

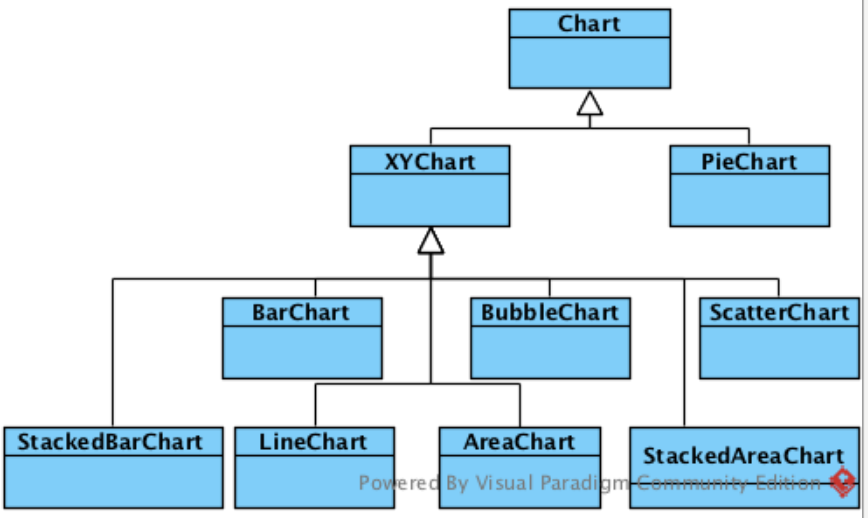
- Copy persistent `TaskManager` from the original Todo application
- Make it a singleton and return `ObservableList<Task>` instead of `List<Task>`
- Copy `Parameters, ModelException` and `DatabaseException`
- Update `TaskMainController`
- Add the two remaining menu items `New Task List` and `Open Task List` under `File` menu

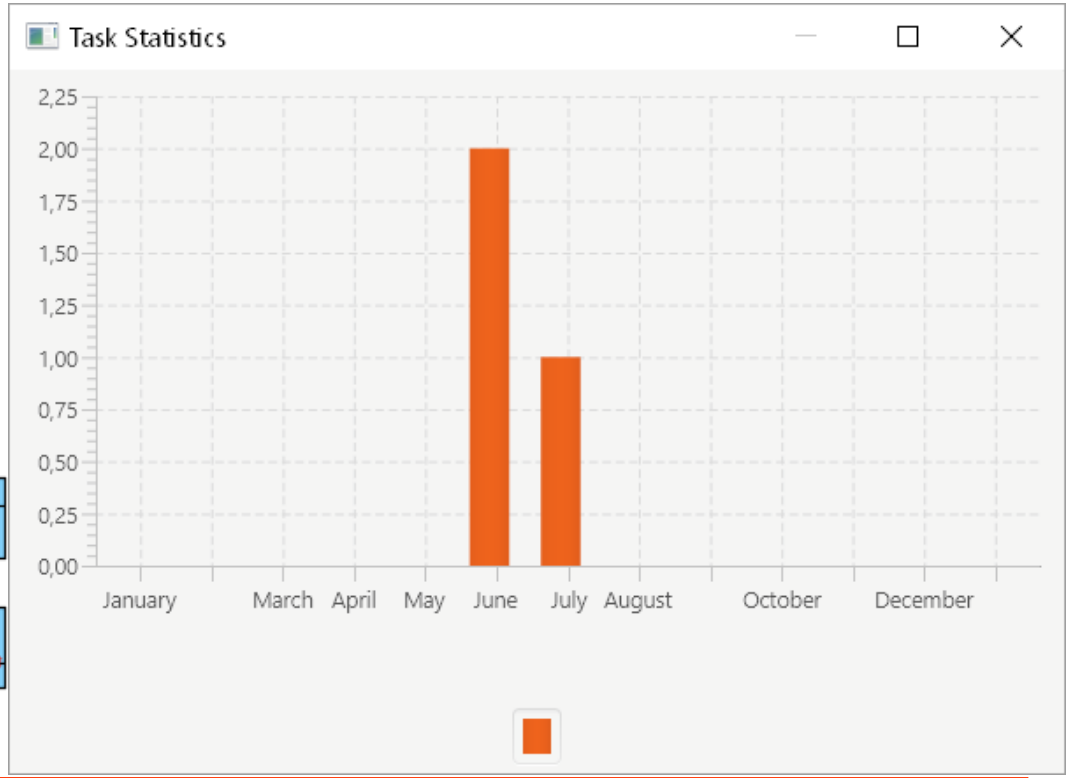
EXTRA STEPS

Graphs and JPA

Step 4: Add Graph

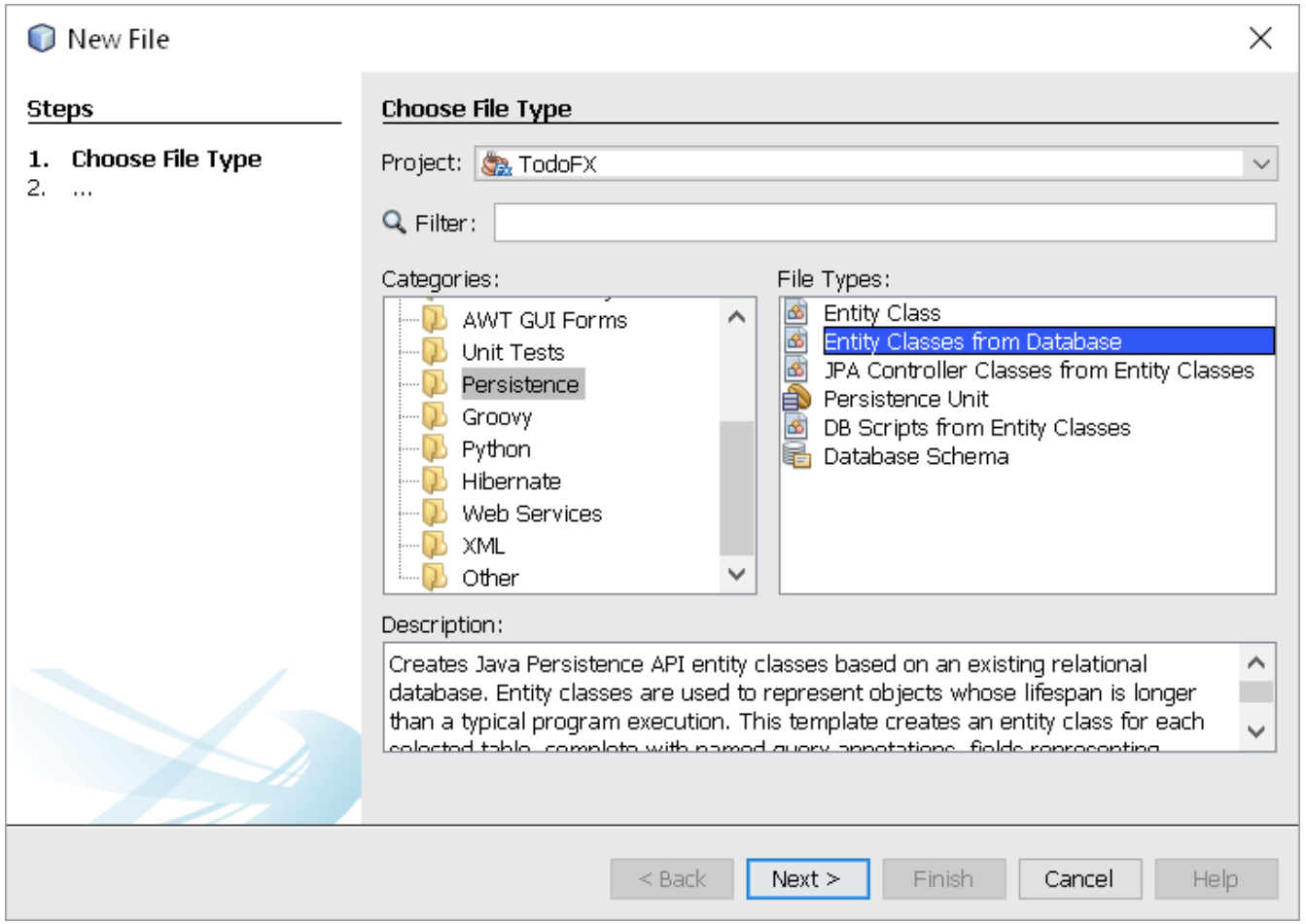
- Add new `TaskStatistics.fxml`
- Select the root **AnchorPane**.
- Add a **BarChart** to the **AnchorPane**.
- Right-click on the **BarChart** and select **Fit to Parent**.





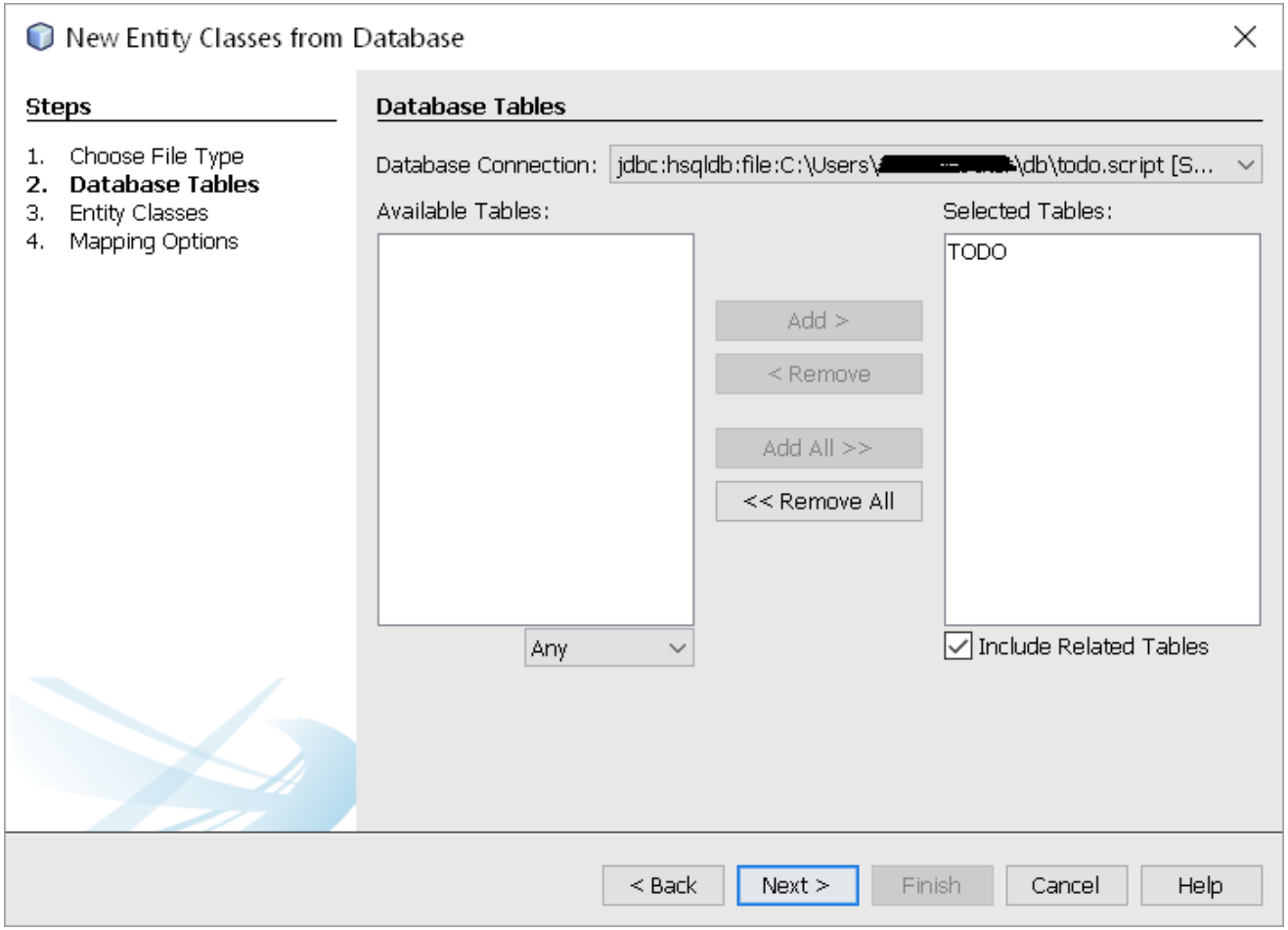
Step 5: Add JPA support

1. Right-click on `todofx package` and select `New` → `Other` → `Persistence` → `Entity Classes from Database...` and click **Next**



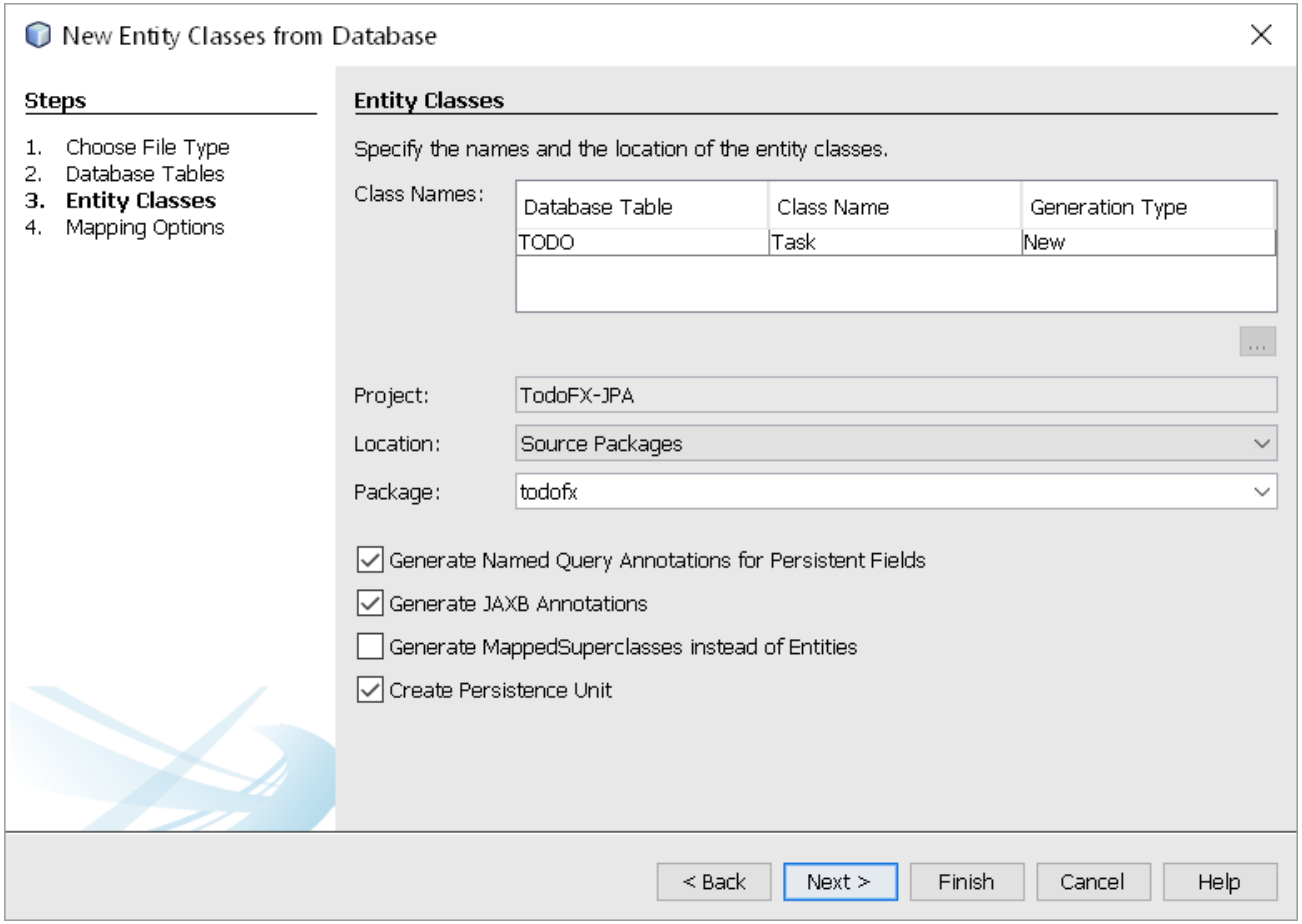
Step 5: Add JPA support

2. Select your *Database Connection*, select the `TODO` table from the list of *Available Tables* and click on **Add** to add it to the list of *Selected Tables*. Click **Next**.



Step 5: Add JPA support

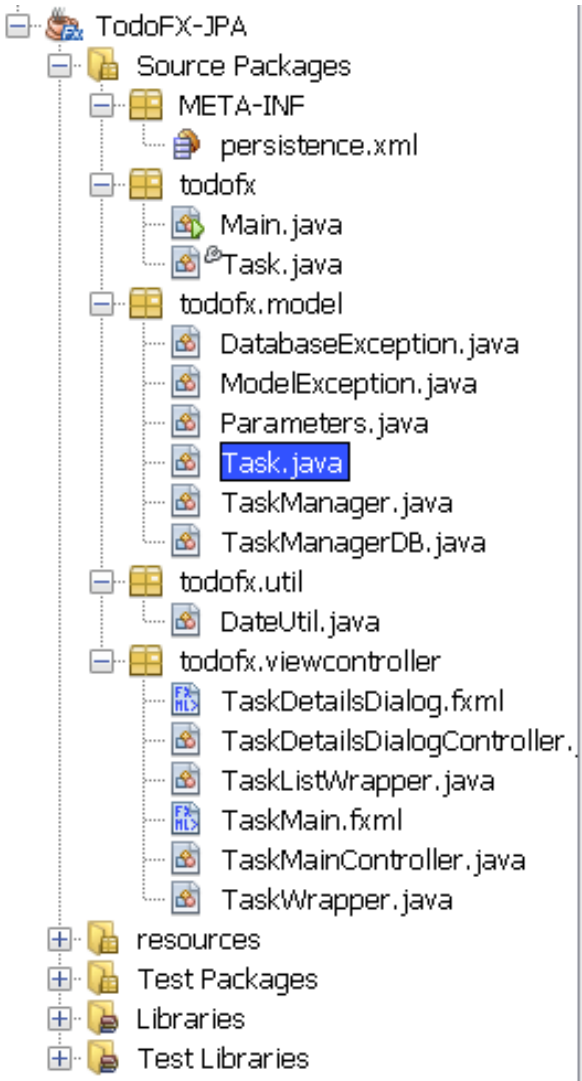
3. Edit the class name from `Todo` to `Task`. Leave the other settings as in the figure and click on **Next**.
4. Leave the default settings in the *Mapping Options* page and click **Finish**.



Step 5: Add JPA support

TodoFX & TodoDS

- The wizard has created the `persistence.xml` file that contains the details about the connection with the database.
- It has also created `todofx.Task` class which is our Entity class.
- Copy the changes from `todofx.Task` to `todofx.model.Task`.
- JPA 2.1 does not support Java 8, so `dueDate` needs special treatment (`LocalDateConverter`).
- Add two new named queries: "`Task.findAllOrderByPriority`" and "`Task.findAllOrderByDueDate`"
- Change named query "`Task.findByAlert`" to `query = "SELECT t FROM Task t WHERE t.alert = true")`
- It is important that there is a parameterless constructor, and that `id` is initialised to 0 and not to -1, because the auto-increment will set the first id value to be 0 and EclipseLink has problems with zero keys (see note).



Step 5: Add JPA support

TodoFX & TodoDS

- Get rid off `DatabaseException.java`, `ModelException.java`, `Parameters.java`, and `todofx.Task.java`.
- Update `TaskManager.java` and `TaskMainController.java` accordingly.
- Add **File → New task list** and **File → Open task list** functionality
- Add `reconnect()` method.

TODO DS

What is DukeScript

- DukeScript is a new technology for creating cross-platform mobile, desktop and web applications. It allows you to write your logic in Java and render the result to a number of clients, which can be web browser, portable devices etc.
- DukeScript applications are plain Java applications that internally use HTML5 technologies and JavaScript for rendering. This way developers only need to write clean Java code and can still leverage the latest developments in modern UI technology.

How does it work

TodoFX & TodoDS

Pros & Cons

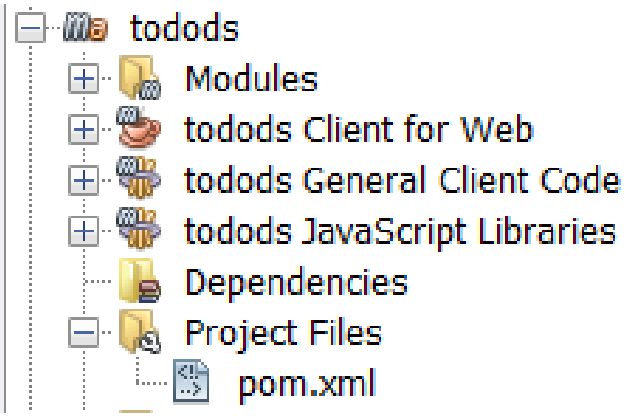
+ Write in Java
+ Write once run everywhere (web, JavaFX, Android, iOS, …)
+ API similar to JavaFX
- not a lot of documentation available
- Need to learn a new API

Installation

- In NetBeans install the DukeScript plugin
  - Tools → Plugins → Available Plugins
    - DukeScript Project Wizard
    - Mine Sweeper

Create a new DukeScript application

- In NetBeans:
  - File → New Project
    - Categories: HTML5
    - Projects: HTML5 with Application Logic
  - Provide an artifact & group id
  - Choose your platform: Browser
  - Select *Knockout 4 Java Maven Archetype* template
  - Right click on it and choose **Build with Dependencies**.
  - Execute *tendrils Client for JavaFX*
  - Execute *tendrils Client for Web*



To be continued

References

- Lozano F. (2006), "A complete App using NetBeans 5", NetBeans Magazine, Issue 1, May, http://netbeans.org/download/magazine/01/nb01_completeapp.pdf
- Kostaras I. (2016), *TodoFX*
- Kostaras I. (2016), *TodoFX2*
- Kostaras I. (2016), *TodoDS*
- Neward T. (2013), “Java 8: Lambdas – Part 1”, *Java Magazine*, Issue 13, July-August, pp. 34-40.
- Neward T. (2013), “Java 8: Lambdas – Part 2”, *Java Magazine*, Issue 14, September-October, pp. 28-34.
- Urma R-G. (2014), “Processing Data with Java SE 8 Streams – Part 1”, *Java Magazine*, Issue 17, March-April, pp. 50-55.
- Urma R-G. (2014), “Processing Data with Java SE 8 Streams – Part 2”, *Java Magazine*, Issue 18, May-June, pp. 49-53.
- Wielenga G., Franklin L., Gydri A. (2014), “Quick and Easy Conversion to Java SE 8 with NetBeans IDE 8”, *Java Magazine*, Issue 18, May-June, pp. 42-45.
- Evans B., Warburton R. (2014), “Java SE 8 Date and Time”, *Java Magazine*, Issue 16, January-February, pp. 56-58.

References (cont.)

- Anderson G., Anderson P. (2014), *JavaFX Rich Client Programming on the NetBeans Platform,* Addison-Wesley.
- Dea C., et. al. (2014), *Java FX 8: Introduction by Example*, 2nd Ed., Apress.
- Epple A. (2016), *Java everywhere: Write Once Run Everywhere with DukeScript,* LeanPub.
- Duodu E. (2015), "How to Create a JavaFX GUI using Scene Builder in NetBeans", IDR Solutions.
- Jacob M. (2014), "JavaFX 8 Tutorial", code.makery.
- Bauer et. al. (2016), Java Persistence with Hibernate, 2nd Ed., Manning.
- Coehlo H., Kiourtzoglou B., Java Persistence API Mini Book, JavaCodeGeeks.
- Goncalves A. (2013), Beginning Java EE 7, Apress.
- Goncalves A. (2013), "Generating Database Schemas with JPA 2.1", Antonio's blog.
- Janssen T. (2016), "Standardized schema generation and data loading with JPA 2.1", Thoughts-on-java.
- Keith M. & Schincariol M. (2013), Pro JPA 2 - Mastering the Java™ Persistence API, 2nd Ed., APress.
- Sharan K. (2015), *Learn Java FX 8,* Apress.
- Taman M. (2016), *JavaFX Essentials,* Apress.
- Vos J., et al. (2014), *Pro JavaFX 8,* Apress.

References (cont.)

- Darwin I. F. (2014), *Java Cookbook*, 3rd Ed., O' Reilly.
- Naftalin M. (2014), *Mastering Lambdas: Java Programming in a Multicore World,* Oracle Press.
- Subramaniam V. (2014), *Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions,* Pragmatic.
- Urma R-G., Fusco M., Mycroft Al. (2014), *Java 8 in Action: Lambdas, Streams, and functional-style programming,* Manning.
- Warburton R. (2014), *Java 8 Lambdas: Pragmatic Functional Programming,* O' Reilly.

Q&A