Exercise on FrameWeb – Language School

A language school wants an information system to manage their activities. It offers courses on many different languages and each course is organized into levels (A1, A2, B1, etc.). Because there could be special levels such as "Technical English" or "Italian for Tourists" the system should explicitly register which level is a prerequisite of another (but prerequisites are always of the same language).

Each semester, the school opens classes for each level and language. Each class is identified by a YEAR/SEMESTER code (e.g. 2011/1) and class number because more than one class can be opened in the same semester for the same language and level. Classes take place regularly on the same day(s) of the week at a specified time during the whole school-semester, which has different start/end dates than normal semesters (01/01–30/06, 01/07– 31/12). The school also specifies the minimum and maximum numbers of students that a class can take.

Once a class is open, students can register themselves for classes. A class is considered enabled if it has the minimum number of students and full if it has reached the maximum, at which point new registrations are not accepted unless someone forfeits their place. On the week before the beginning of the school-semester, teachers are allocated to classes that are enabled and at this point the class is considered started. On the other hand, classes that are not enabled at this point are canceled. Once classes are started, no new students can register, even if there are places available and in this state a class is canceled only if all students cancel their registrations. Once the school-semester ends, each class is considered finished.

As stated before, teachers are allocated to classes the week before the school-semester starts. The system should have all teachers registered before that point and, in order to avoid problems of allocation, it should also have the information of which languages a teacher is able to teach. After teachers are allocated to classes they can be replaced and the administration would like to know during which period of time each teacher was responsible for the class.

Once the school-semester is over, teachers should insert into the system the mark for each student, so it will be known if they passed the class and can move on to the next level. For billing purposes, the system should store the full name, address and fiscal code of each student. Proficiency tests (students that want to skip the initial levels of a language) and management of payment of school fees are out of the scope of this system.

Given the above description:

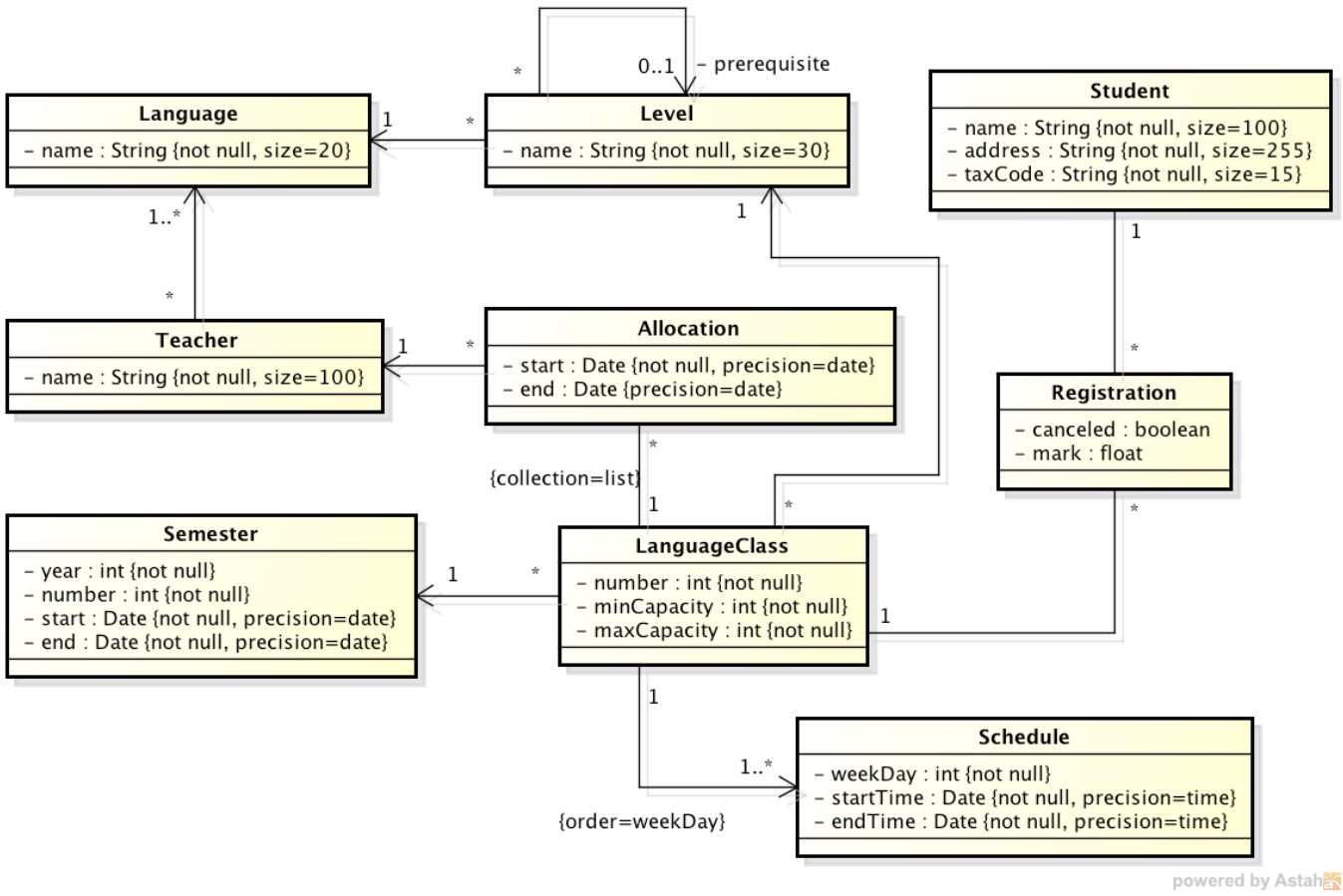
1. Produce a FrameWeb domain model for this system;

Considering a use case for registering the marks of the students:

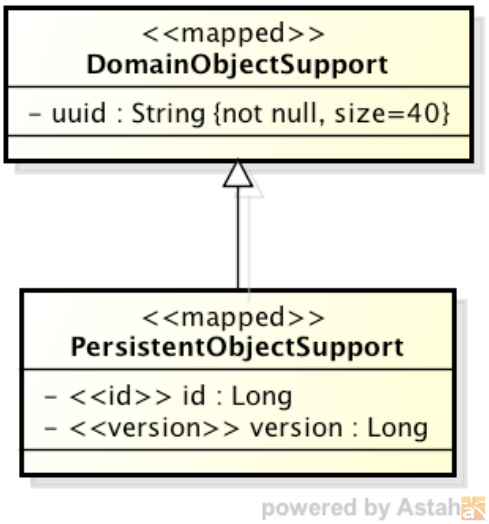
2. Produce a FrameWeb navigation model for this use case;
3. Produce a FrameWeb application model considering only this use case;
4. Produce a FrameWeb persistence model considering only this use case and assuming nemo-utils was used.

Exercise on FrameWeb – Language School – Solution

1. Domain model



Utility package (nemo-utils):

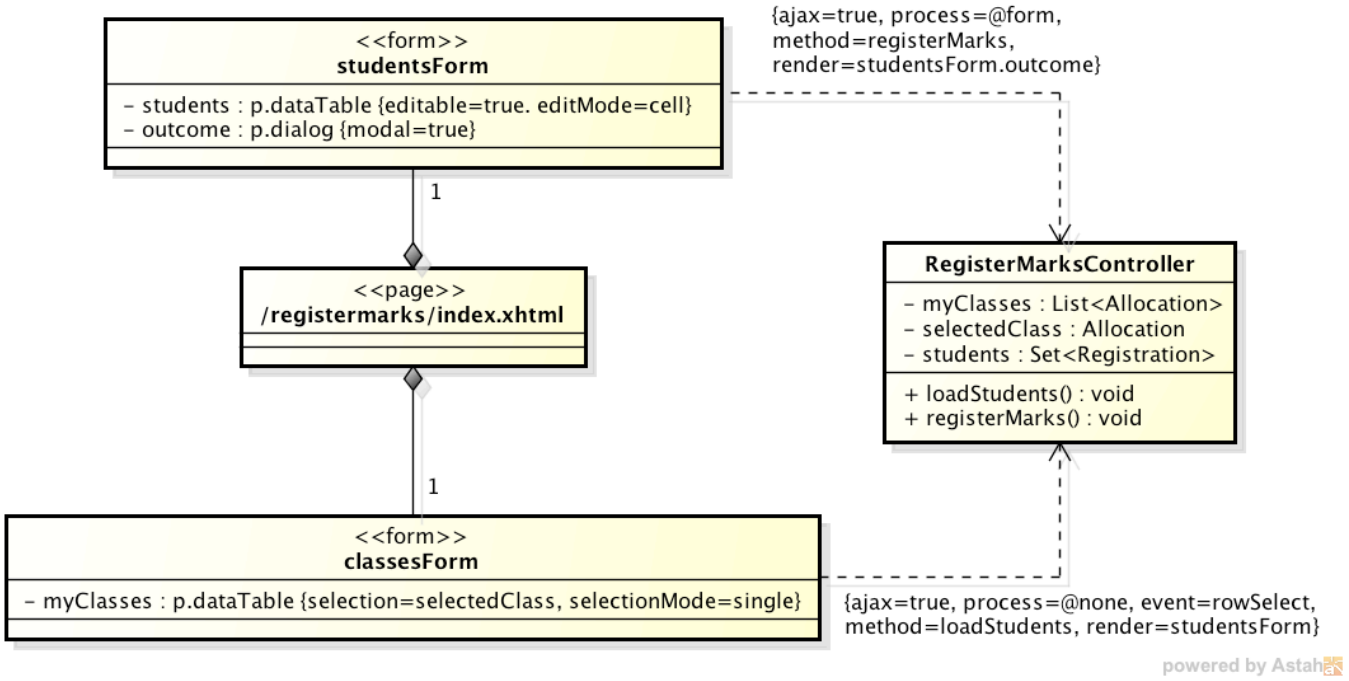


The domain model shows classes from the domain package and their O/R mapping. It uses all default mapping values (all classes are persistent, mapped to tables with the same name as the classes, columns with same names as the attributes, etc.).

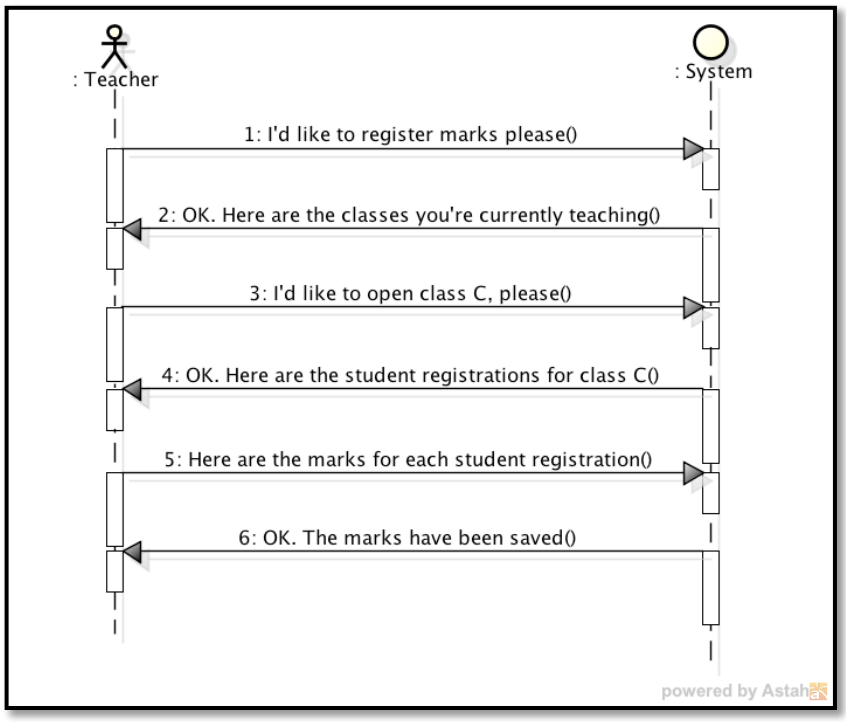
Moreover, all classes extend from `PersistentObjectSupport` from nemo-utils, shown in the diagram to the left. This inheritance is not shown in the above diagram in order not to pollute it with so many inheritance associations.

Finally, the above diagram repeats the `{not null}` constraint to almost every attribute, showing that maybe it should be the default value instead of `{null}`. This change could be proposed to FrameWeb 2.0.

2. Navigation model, using AJAX and PrimeFaces:



The Register Marks use case is implemented with a single `index.xhtml` page that contains 2 forms and a dialog, loaded via AJAX instead of a full request. For this reason, this model is quite different from the original FrameWeb syntax proposed in 2007.



We assume the use case follows the sequence of interactions shown in the diagram to the right. The first difference here, then, is that in 2007, using Struts[2], the use case would begin by calling the controller (action class) and asking it to show the index web page. In JSF, on the other hand, we can access directly `index.faces` in the appropriate path and JSF will take care of the binding. The list of classes the teacher is currently responsible for doesn't have to be pre-loaded by a method like before, in Struts[2]. The developer just needs to make sure that the controller is properly initialized (via CDI) or that the accessor method (`getCollection()`) loads the data lazily.

Hence, step 2 of the sequence diagram consists in showing the `index.xhtml` page and loading the classes in the data table component, which is part of `classesForm`. The p. prefix

in the components' types indicates we are using PrimeFaces[1] as JSF component library. Having the same name as an attribute of the controller (`myClasses`) indicates that this is the binding that should be set at the `value=""` property of the `<p:dataTable />` tag. Other attributes are shown using constraints: `selectionMode=single` indicates that the data table should be configured for row selection, whereas `selection=selectedClass` indicates that the instance represented by the selected row should be bound to the `selectedClass` attribute of the controller. The key/value pairs at the constraints correspond almost exactly to the key/value pairs that should be present at the tag:

```
<p:dataTable value="#{registerMarksController.myClasses}"
     selection="#{registerMarksController.selectedClass}"
     selectionMode="single" />
```

The dependency association between `classesForm` and the controller class represents step 3 of the use case. Constraints on this dependency indicate that: (a) the request should be done using AJAX – `ajax=true`; (b) there's no need to submit data to the controller – `process=@none`; (c) the request should be triggered by the event of selecting a row of the data table – `event=rowSelect`; (d) there's a listener method to be called at the controller – `method=loadStudents`; and (e) as a response for this AJAX request, only the form with the students data table should be rendered – `render=studentsForm`. Such response corresponds to step 4 of the sequence diagram.

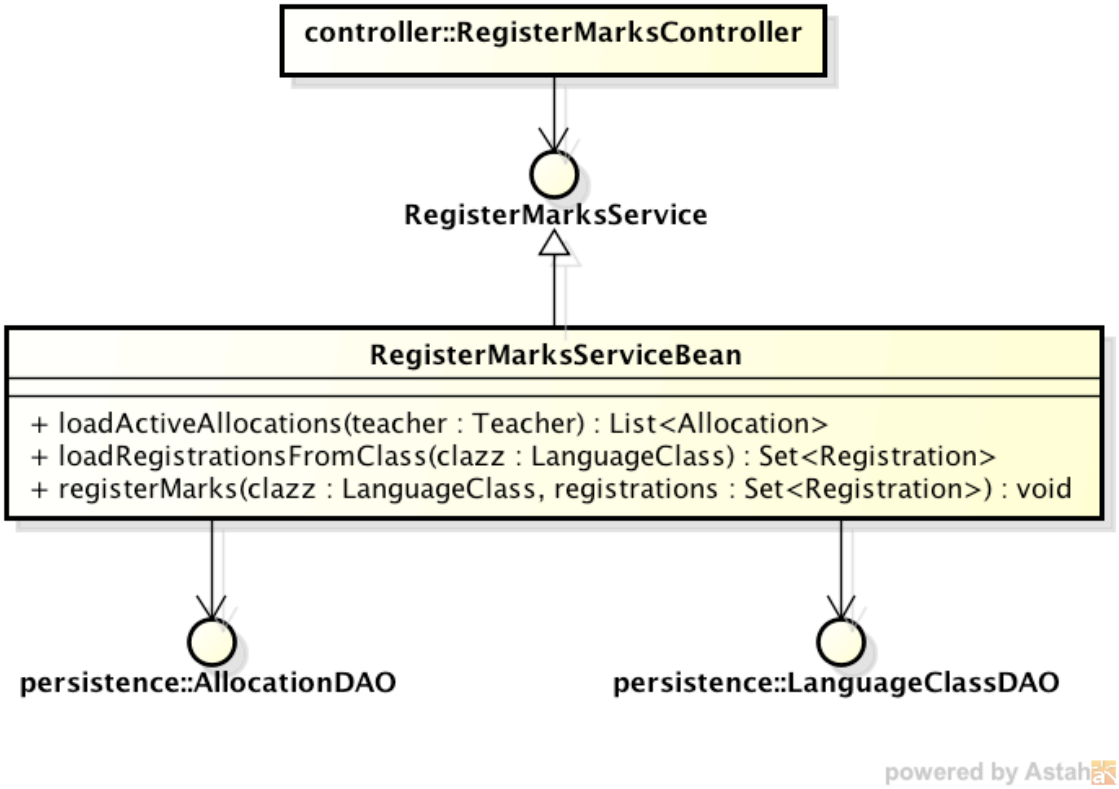
The `studentsForm.students` data table shows the data loaded in the homonymous attribute of the controller and is configured to be an editable table, with editors attached to the cells of the mark column. This is implicit, given what this use case does, but this is a possible point of discussion about FrameWeb: should it be implicit?

The data from the data table is sent to the controller again via AJAX (representing step 5), but this time no event is specified, indicating that there should be a button to trigger this request. Once it's processed, the `studentsForm.outcome` modal dialog is shown with a message indicating the result of the operation. The fact that this dialog was hidden (`rendered="false"`) up to this point is also implicit in the diagram, which is also another point of discussion. The dialog concludes the use case with step 6.

Note that, this way, the teacher can go back to step 3 and select another class, which will re-render the students data table and so on.

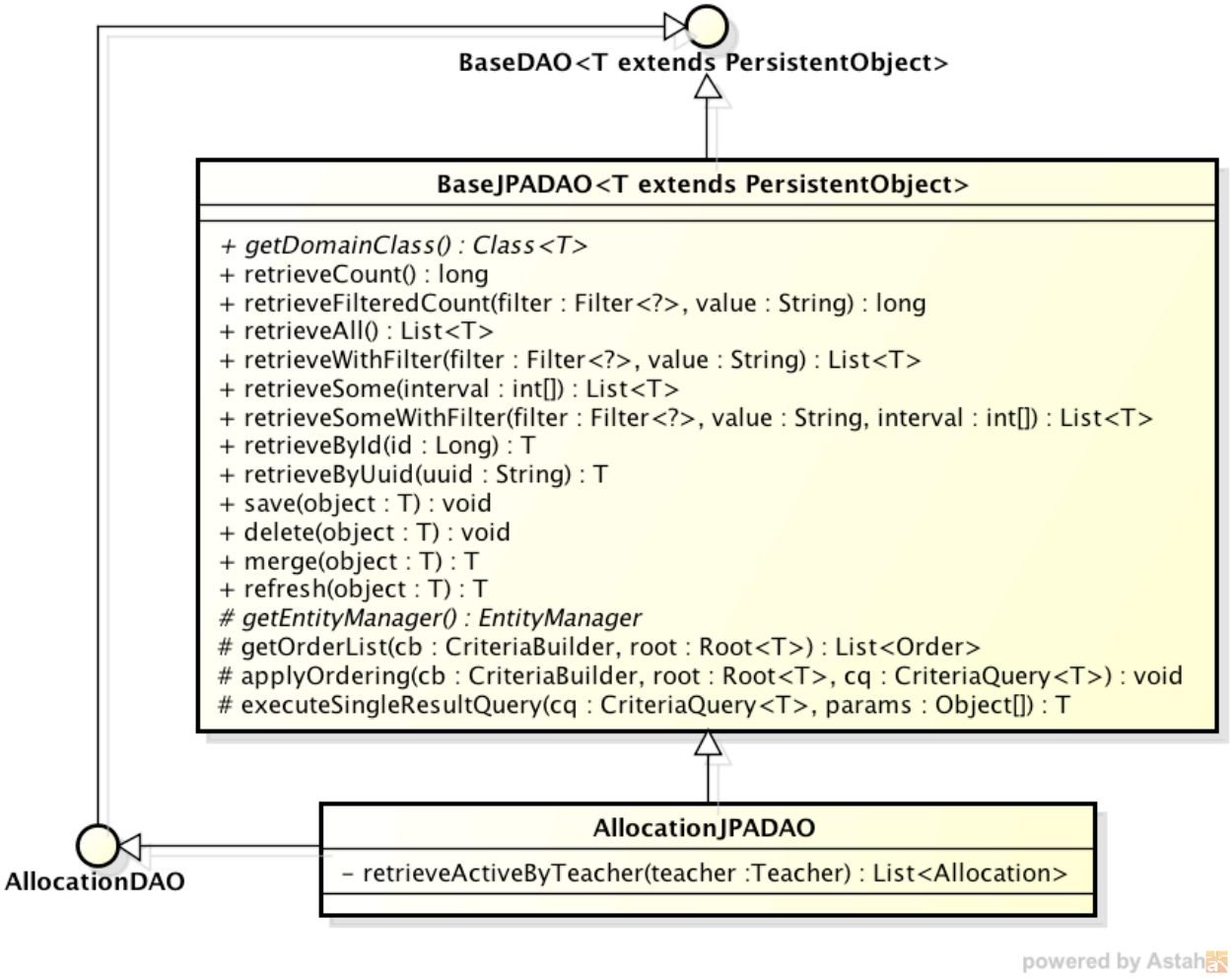
[1] http://www.primefaces.org

3. Application model, considering only the use case "Register Marks"



The application model follows the original FrameWeb 2007 syntax. No changes are needed here, as advanced CDI features have not been used. Considering these features is subject to future work on FrameWeb.

4. Persistence model, considering nemo-utils and only the use case "Register Marks"



This diagram also follows standard FrameWeb 2007 syntax. Note that `LangaugeClassDAO` is not shown, as it has no specific queries related to the "Register Marks" use case (only basic DAO operations are used).