University of Zurich[UZH]

s.e.a.l.
software evolution & architecture lab

Software Testing FS 2011

Exercise 1

The goal of this exercise is to implement unit tests in isolation. To achieve this goal you should use a mocking framework (such as, EasyMock, JMock, JMockit, or mockito). The task descriptions mention whether or not a class has to be mocked in order to test another class. If you have any question please contact Beat Fluri (`fluri@ifi.uzh.ch`).

uDoo – Todo List Organization

The goal of this exercise is to unit test the uDoo application. uDoo provides the following features:

- Create and delete todos
- Change existing todos
- Organize todos in projects
- Undo and redo of executed commands
- Store and load projects including their todos

TodoList

Figure 1 shows the UML class diagram of the `TodoList` class with its relations. `TodoList` implements the interface `ITodoListModel`, which defines operations of a todo meta model. A `TodoList` organizes a set of `Project`s. Each `Todo` belongs to a `Project`. If a user-defined `Project` does not exists yet, new `Todo`s are put into the *default*-`Project`.

The `TodoList` is an `IObservable`, but delegates the corresponding calls to the `Dispatcher`. The classes `TodoList`, `Todo`, and `Project` define `EventID`s describing possible model changes. Classes which define `EventID`s implement the interface `IEventProvider`. `IObserver`s may register for certain `EventID`s in the `TodoList` and are notified by the `Dispatcher` whenever the `TodoList` changes. A notification consists of an `Event`, which contains the `EventID` identifying the change, and the changed `Object`.

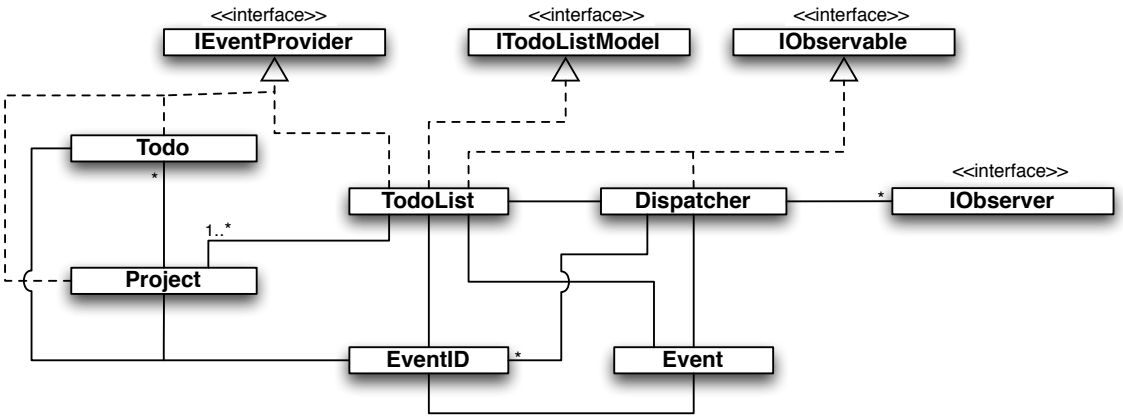

Figure 1: UML diagram of the class `TodoList` and its relations (to reduce complexity, the relations of interfaces are not shown)

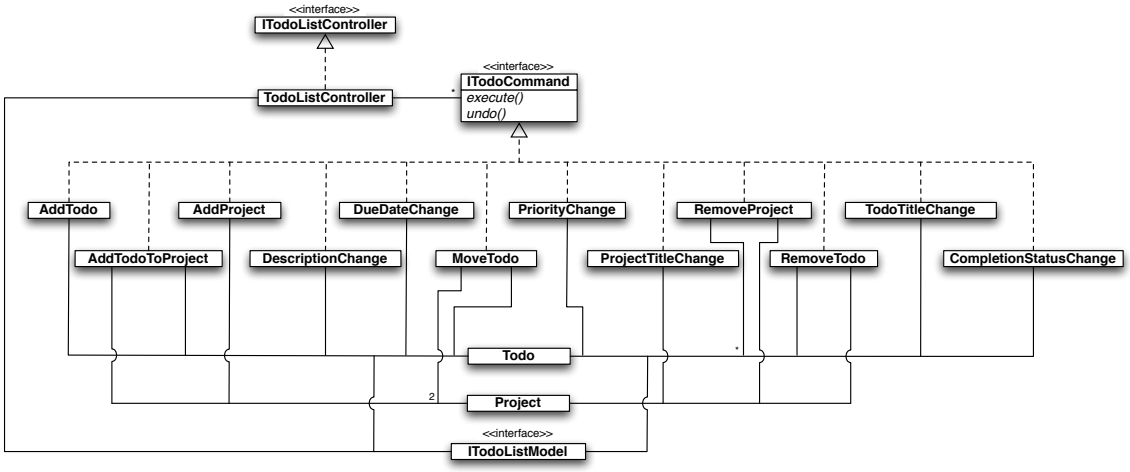

Figure 2: UML diagram of the `ITodoCommand` hierarchy

ITodoCommand

Figure 2 shows the UML class diagram of the `ITodoCommand` hierarchy. The `ITodoCommand` hierarchy implements the *Command Pattern*. For each operation that is specified in `ITodoListModel` a corresponding `ITodoCommand` implementation exists. For instance, the command `AddProject` is responsible for `addProject(Project)`. Such an operation is executed with the method call `execute()` and may be undone with `undo()`.

Before the *controller* executes an operation on a `ITodoListModel` object, it creates and saves a corresponding `ITodoCommand` for *undo/redo*. Then, it executes the operation by using the saved instance. *undo/redo* operations will be triggered via saved `ITodoCommand` instance.

1 Task: Dispatcher test

Write unit tests for the `Dispatcher` class. You can assume that the `EventId` class is already integrated and must not be tested anymore. You can only use `IObserver` to write test cases for the `Dispatcher`. Concrete `IObserver` implementations **are not** available

Are you able to achieve a 100% condition coverage?

2 Task: ITodoCommand hierarchy test

Write unit tests for the `ITodoCommand` hierarchy. You can assume that the `Project` and `Todo` classes are already integrated and must not be tested anymore. The `TodoList` class **cannot** be used for testing the `ITodoCommand` hierarchy.

Are you able to achieve a 100% condition coverage?

3 Task: TodoList test

Write unit tests for the `TodoList` class. You can assume that the `Event`, `EventID`, `Project`, `Todo` classes are already integrated and must not be tested anymore. The `Dispatcher` class **cannot** be used for testing the `TodoList` class.

Try to achieve a 100% branch coverage.