DOSBOX DESIGN OVERVIEW

DOSBox Design Overview

file:///C:/Data/Courses/D10064 - Practice in Scrum .NET Development/Development/Course/Lab - ...

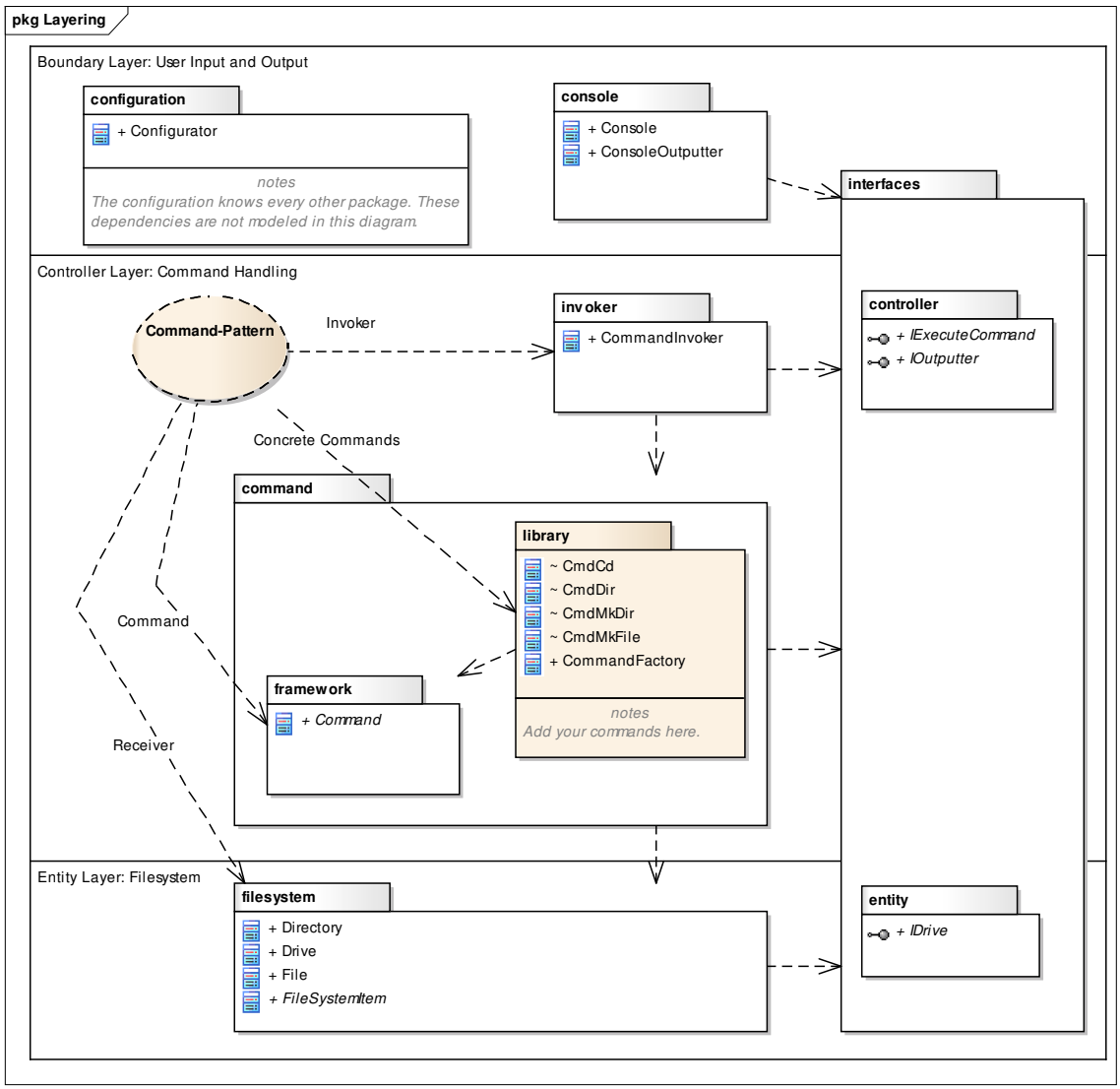
```
Scrum.org PSD Course [Version 01.08.2012]
(C) Copyright 2006-2012 Rainer Grau and Daniel Tobler.

C:> mkdir test1

C:> mkfile mydoc.txt ThisIsContent

C:> dir
Directory of C:

<DIR>    test1
13       mydoc.txt
         1 File(s)
         1 Dir(s)

C:> cd test1

C:\test1> dir
Directory of C:\test1

         0 File(s)
         0 Dir(s)

C:\test1>
```

LAYERING

The DOSBox architecture is divided into three layers:

1. Boundary with user in- and output. Here you do not need to change a lot.
2. Controller with command handling. Here is your main working area.
3. Entity with the filesystem. Here are the classes with which you have to work.

The interfaces package is cross cutting.

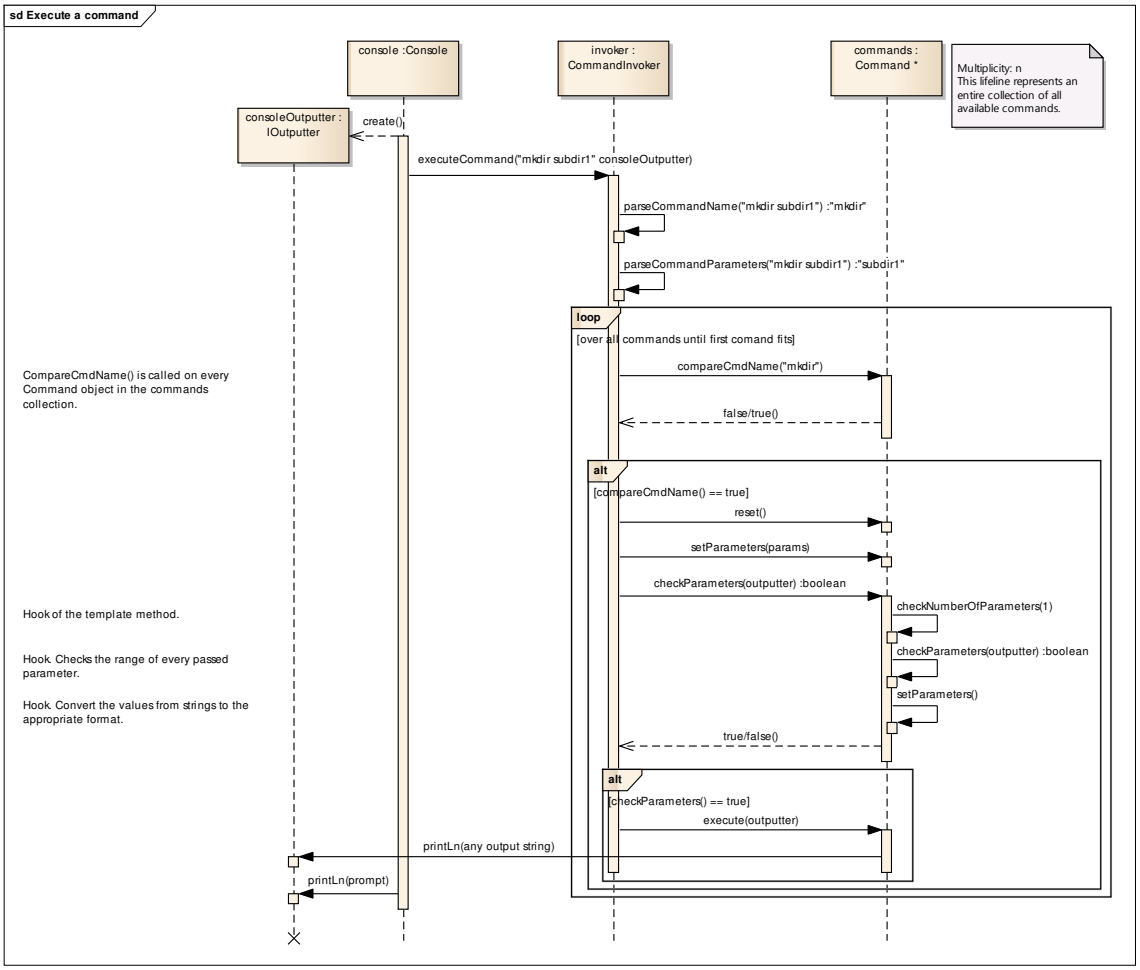


The `library` package is highlighted because this is your main area.

DYNAMIC FLOW, EXECUTE A COMMAND

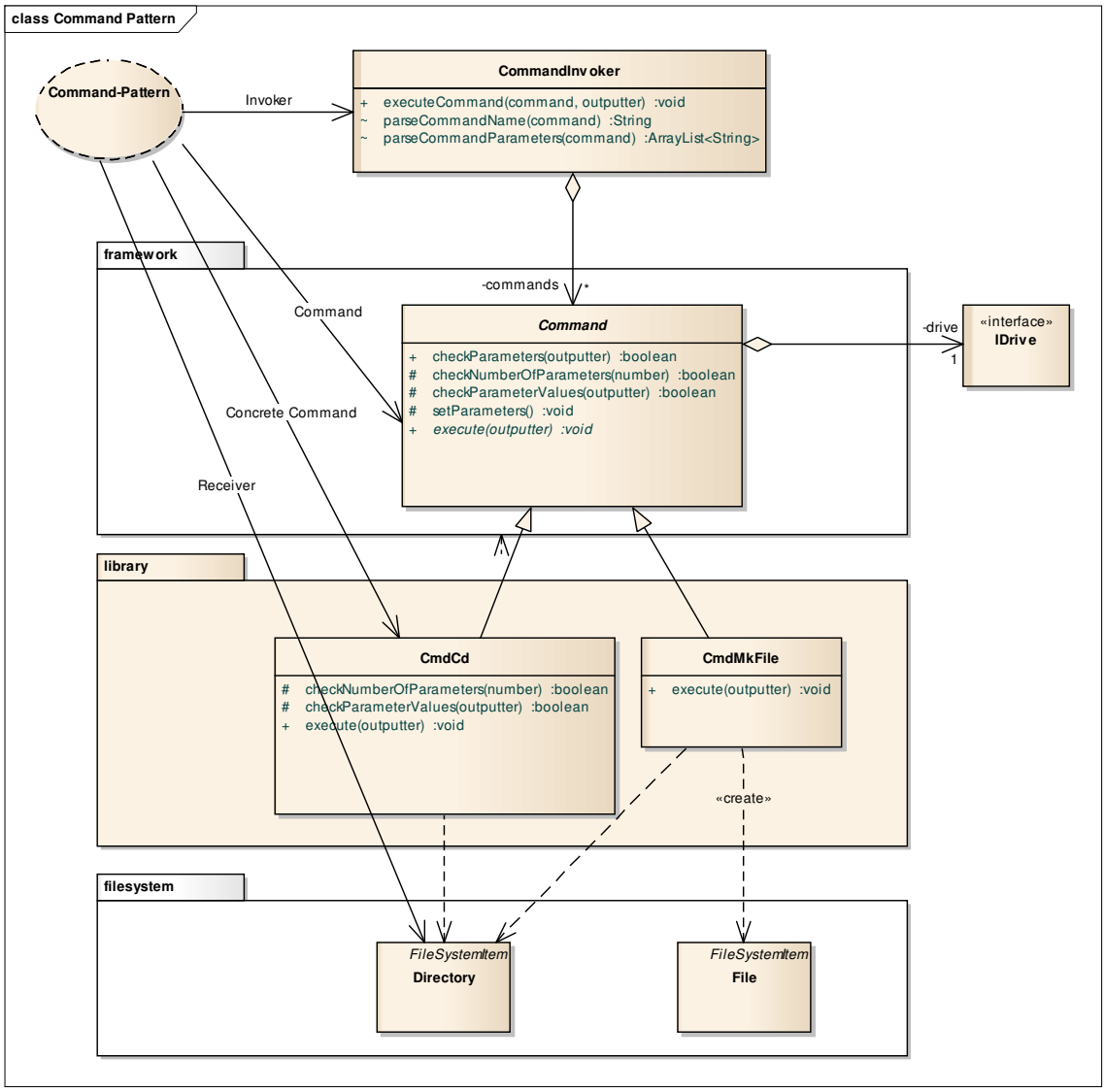
The dynamic flow of the events through the system is very important to know. This diagram models the following scenario:

The user types on the console "mkdir subdir1" and presses enter. A new directory named subdir1 is created under the current directory. Note: The creation of the subdirectory is not modeled in this diagram.



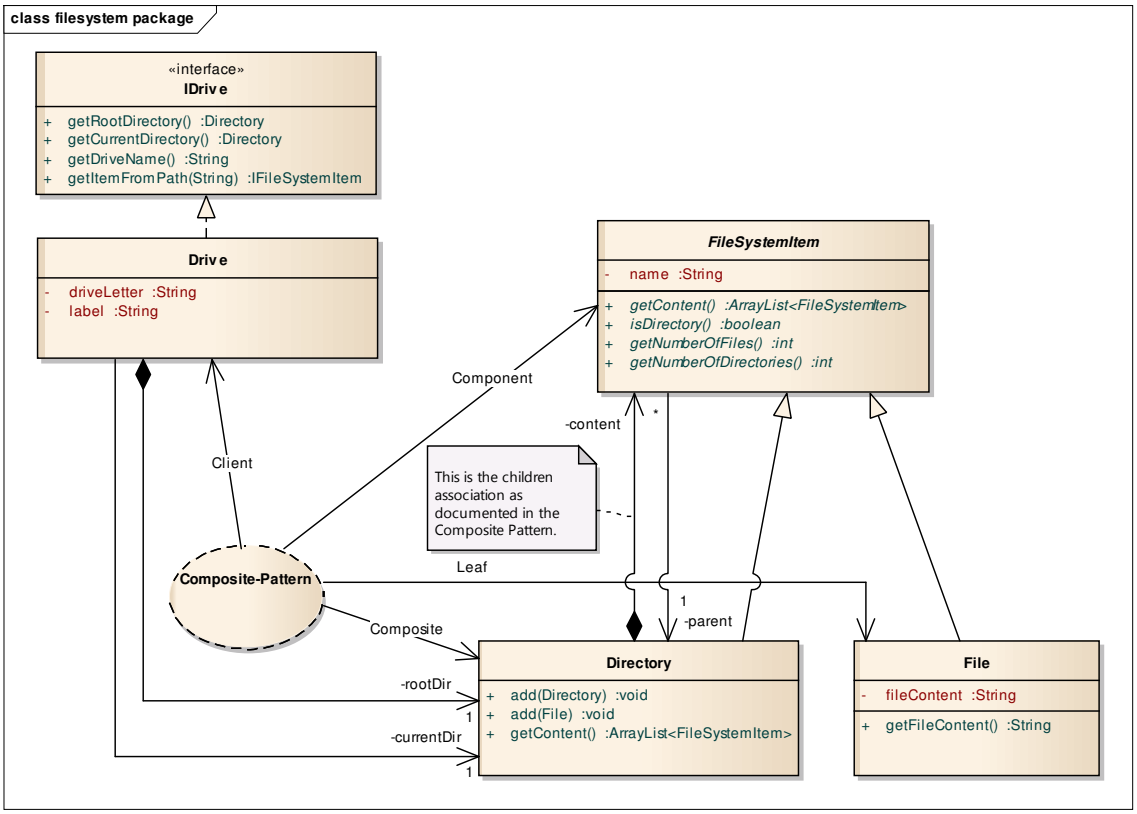
KEY CONCEPT COMMAND PATTERN

A key concept is the Command Pattern for the Controller Layer. To keep the framework of the controller and the expandable part separated, the Controller Layer has two different packages: `framework` and `library`. The `framework` should not be touched meanwhile the `library` is subject for expansion at every sprint.

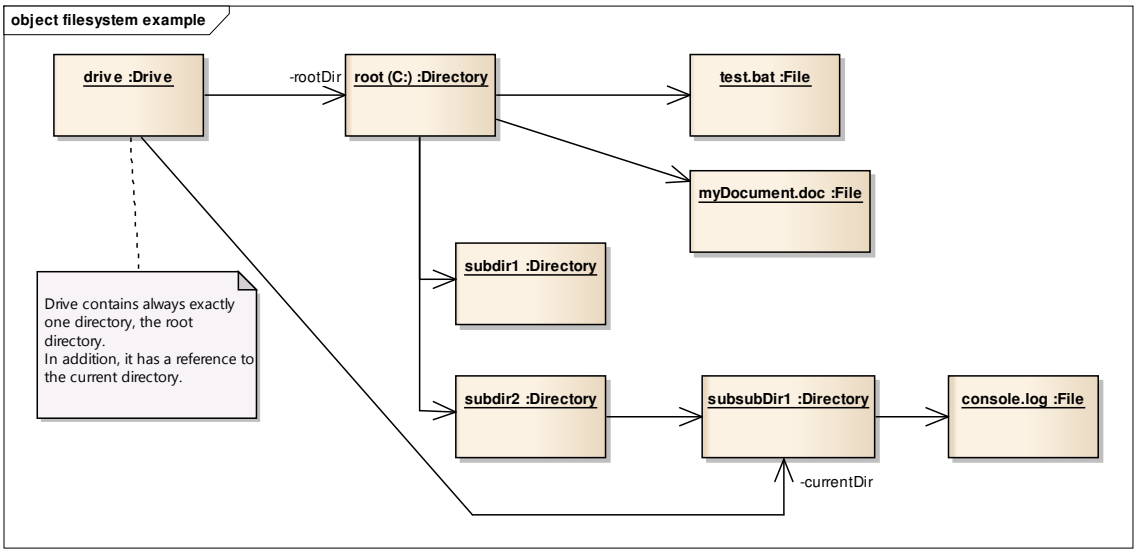


KEY CONCEPT FILESYSTEM

The Filesystem is built according the Composite Pattern: `Directory` is a composite (containing further directories and files) and `File` is a Leaf. The `Drive` acts as your entry point for querying the filesystem. **Here, `getItemFromPath()` is very important!**



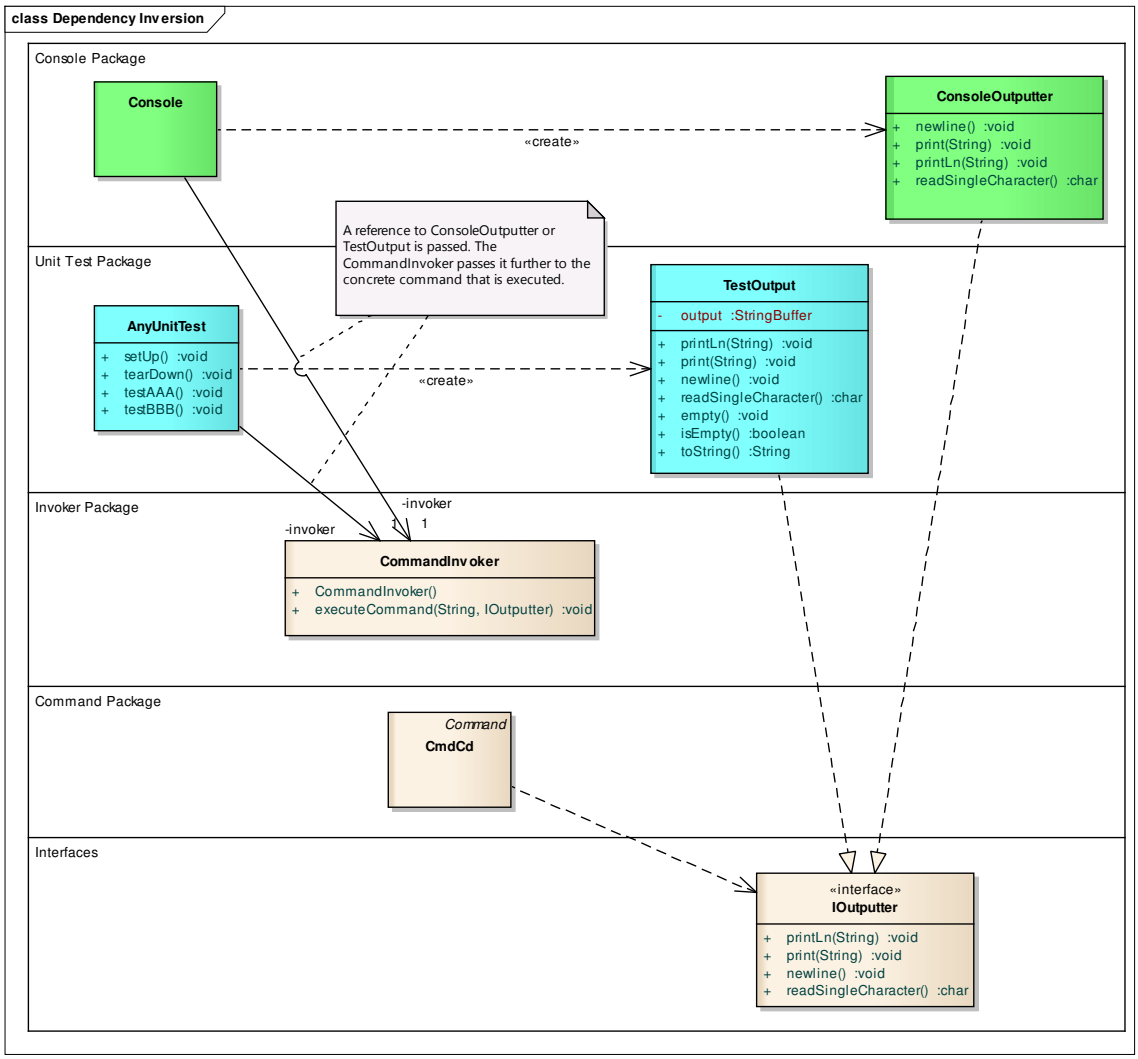
The following object structure can be created using the above classes:



KEY CONCEPT DEPENDENCY INVERSION

The package `console` knows the `invoker` which knows the `command.framwork`. In order to output text to the user over the console, the commands need to know the `console`. This leads to a cycle. This cycle would prevent deep testing among other disadvantages. The unit test would not be able to test the output of commands since it is sent to the console hardcoded.

Therefore, the interface `IOutputter` inverses the dependency so that the commands do not directly know the console. This diagram shows how console and unit tests implement both an own implementation of the interface `IOutputter` and pass this instance to the invoker:



OVERVIEW: ALL CLASSES