Patterns of Software Development

Report on a Bohnanza Implementation

Jeroen Meijer (s0166111)

October 4, 2013

1 Design goals

The importance of particular quality criteria depend on in which environment the application is used. This environment also influences the stakeholders; not every categorie stakeholder is affected by the software system. We offer our system open sourced and free of charge online.[1]. We assume the application is a highly popular one and is us used among many people. We identify the most important quality criteria for this environment as *extensibility* and *reusability*. The most affected stakeholders are the *developers*, *maintainers*, *testers* and *users*.

Extensibility is an external criteria; it is most important for the user. Reusability is more important for developers etc. Both criteria are also covered under the term modularity. Modularity allow the user to easily use one of the many extensions, such as High Bohn or spin-offs, such as Al Cabohne. It should be easy to implement such extensions for developers etc.

There is one major design decisions that contribute to modularity. First we have three modules which can be considered separate projects. One module contains basic objects and method related to bohnanza, such as shuffling the discard pile into the draw deck. Another module implements bohnanza without extensions. The last module implements the extension. All projects can of course be tested and compiled individually. The and extension inherit from the base module. The base module provides an abstract class that resolves dependency resolution between modules. Both the and the module implement this class. Making it possible to hide some information in modules and should result in understandable modules.

- Reusability
- Extensibility
- remove inadequacies

[1] https://github.com/Meijuh/psd

- adapt to changing needs
- extend functionality quickly
- integrate with other systems
- use components in other projects
- Modularity
- decomposability
- composability
- understandability
- continuity
- protection
- direct mapping
- few interfaces
- small interfaces
- explicit interfaces
- information hiding
-

2 Design

Our design is split up in three modules. Every module is designed according to the MVC pattern.

bohnanza Mostly an abstract module that contains logic for both the and game.

bohnanza-std A concrete module that uses the *bohnanza* module in order to play the bohnanza game.

bohnanza-hb A concrete module that uses the *bohnanza* module in order to play the bohnanza game.

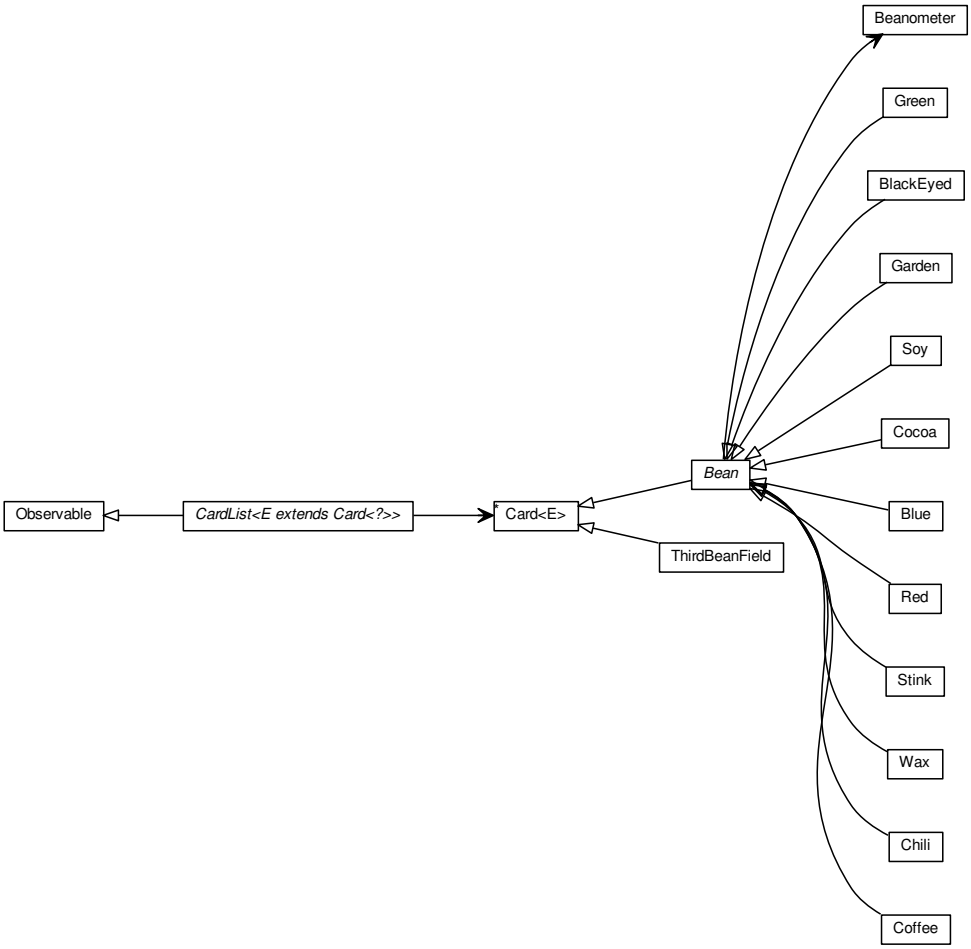

Figure 1: Class diagram of the cards

2.1 Model

2.2 Controller

2.3 View

Currently we support only a textual user interface. The `TUIView` class is an implementation of a `View` interface. Creating a new interface like a graphical or a network would involve implementing the `View` interface.

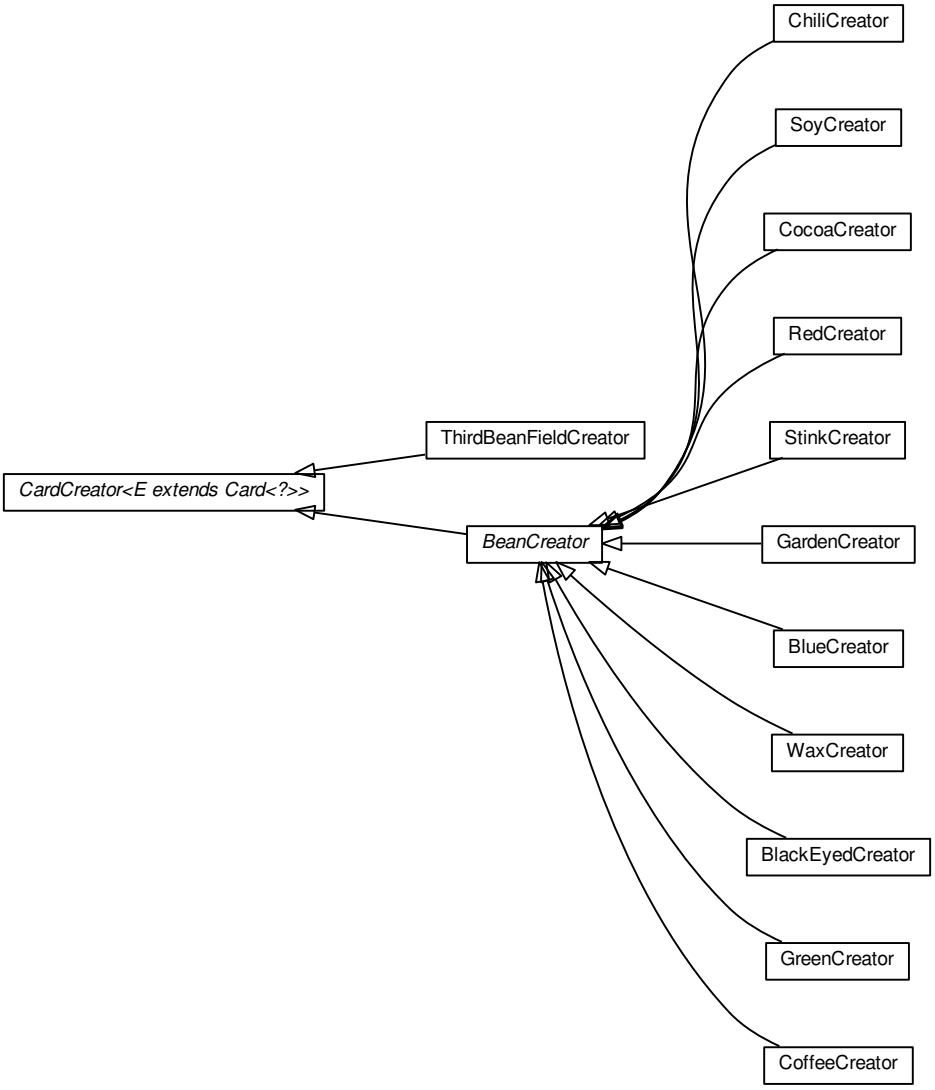

Figure 2: Class diagram for the creation of cards

2.4 Variations

2.4.1 User

The user can specify during the startup of the application how many players there are using a commandline option. For example `java bohnanza.BohnanzaStd -p=3` starts

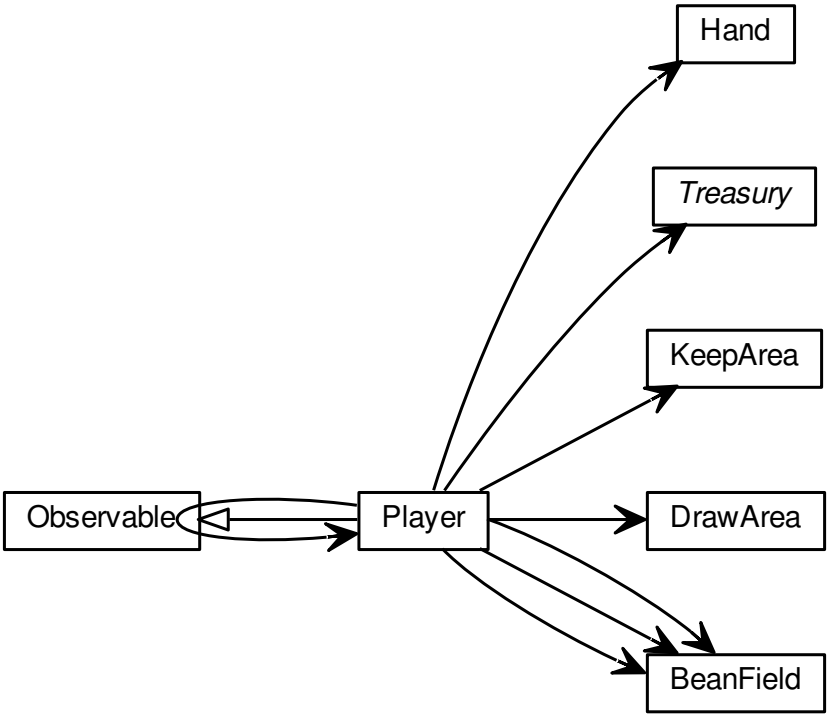

Figure 3: Class diagram for the player and association to particular sets of cards

the BohnanzaStd module with three players. Parsing commandline options is done using the Apache Commons CLI library.

2.4.2 Developer

Variability for the develop depends heavily on notion of dependency injection. Dependency injection basically involves moving dependency resolution from a particular class to a framework dedicated to dependeny injection. We use Guice[2] for this. If a developer wants another variation of the application the developer can simply extend the `BohnanzaModule` class and define other classes to inject. For example if a `Hand` object

[2]https://code.google.com/p/google-guice/

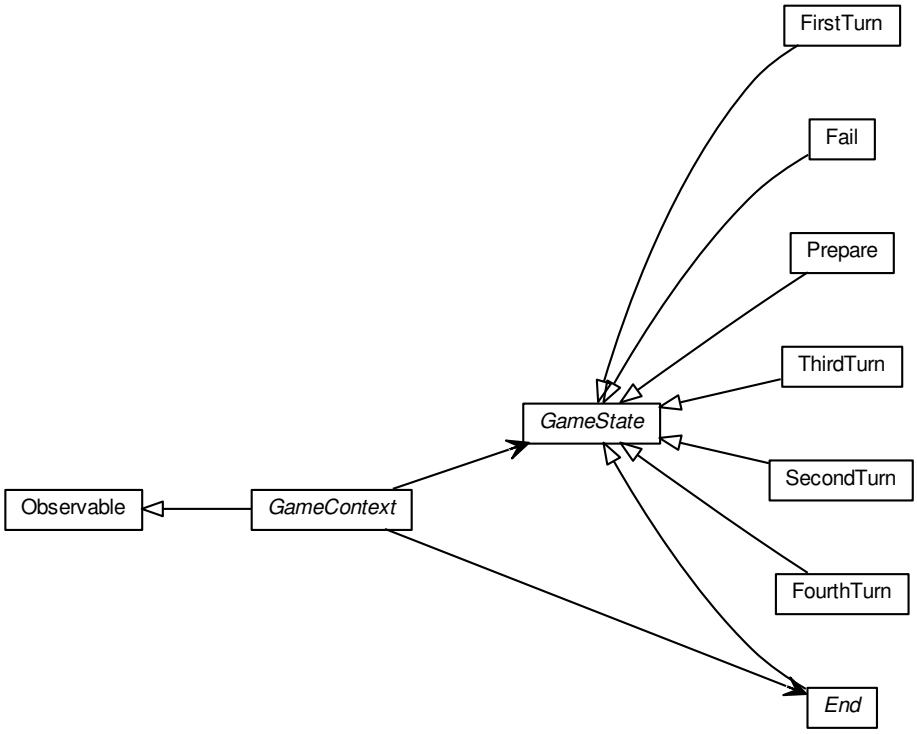

Figure 4: Class diagram for controlling the state of the game

needs to behave differently for a particular extension of Bohnanza one can extend `Hand` and define the extension in extension of `BohnanzaModule`.

3 testing

- The main goal of our testing approach is to test the more complex parts of game such as trading. One of our goals is not to achieve 100 percent test coverage, but to test the complex parts in a a nice way.
- Our design is wel suited for testing. Since our design adheres to the dependency injection pattern injecting mocked objects of classes is easy. For mocking we used the Mockito[3] framework.

[3]https://code.google.com/p/mockito/

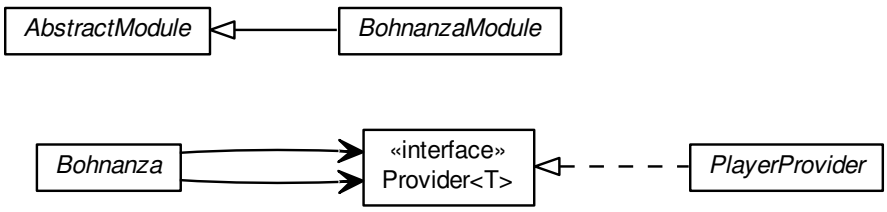

Figure 5: Class diagram of classes involving dependency injection

- Each of the three modules can be compiled and tested separately. However the *bohnanza* module is rather abstract and thus testing some abstract classes do not really make sense. We test more concrete classes in either the module or the module.
- In the bohanza module we mainly tested whether the `Beanometer` or `CardList` works correctly. This is shown in the coverage report in the package `bohnanza.game`. Since a game can not really be played in the bohnanza module the gameplay is tested in a concrete implementation of bohnanza, namely bohnanza-std. This is shown in the package `bohnanza.gameplay`.

4 development process

4.1 Refactor

- can not overwrite non abstract methods because they should by made final by checkstyle.
-

5 Glossary

Notation	**Description**
amount	.
area	.

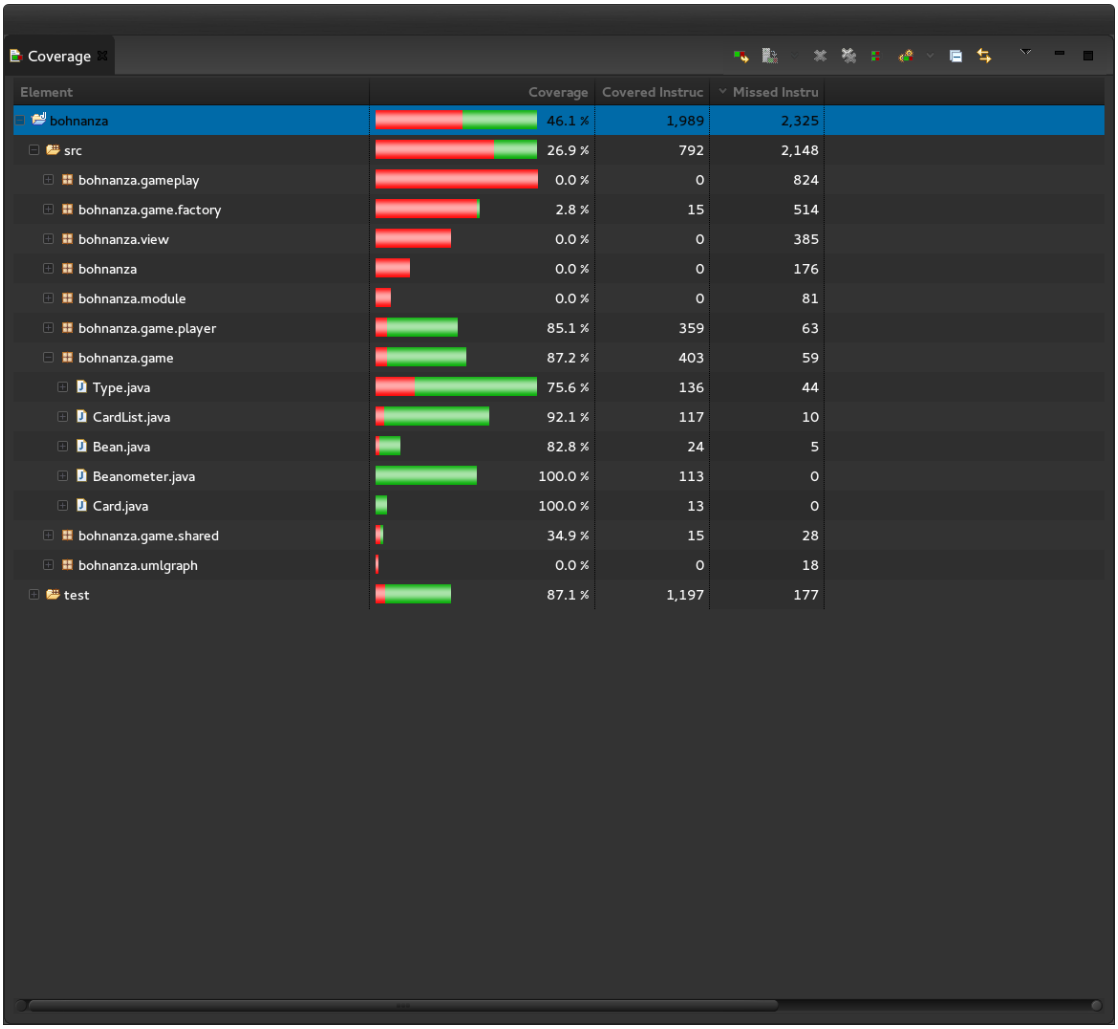

Figure 6: Coverage of the *bohnanza* module

Notation	**Description**
bean	.
bean field	.
beanometer	.
black-eyed	.
blue	.
Bohnanza	.
box	.
buy bean field	.
card	.

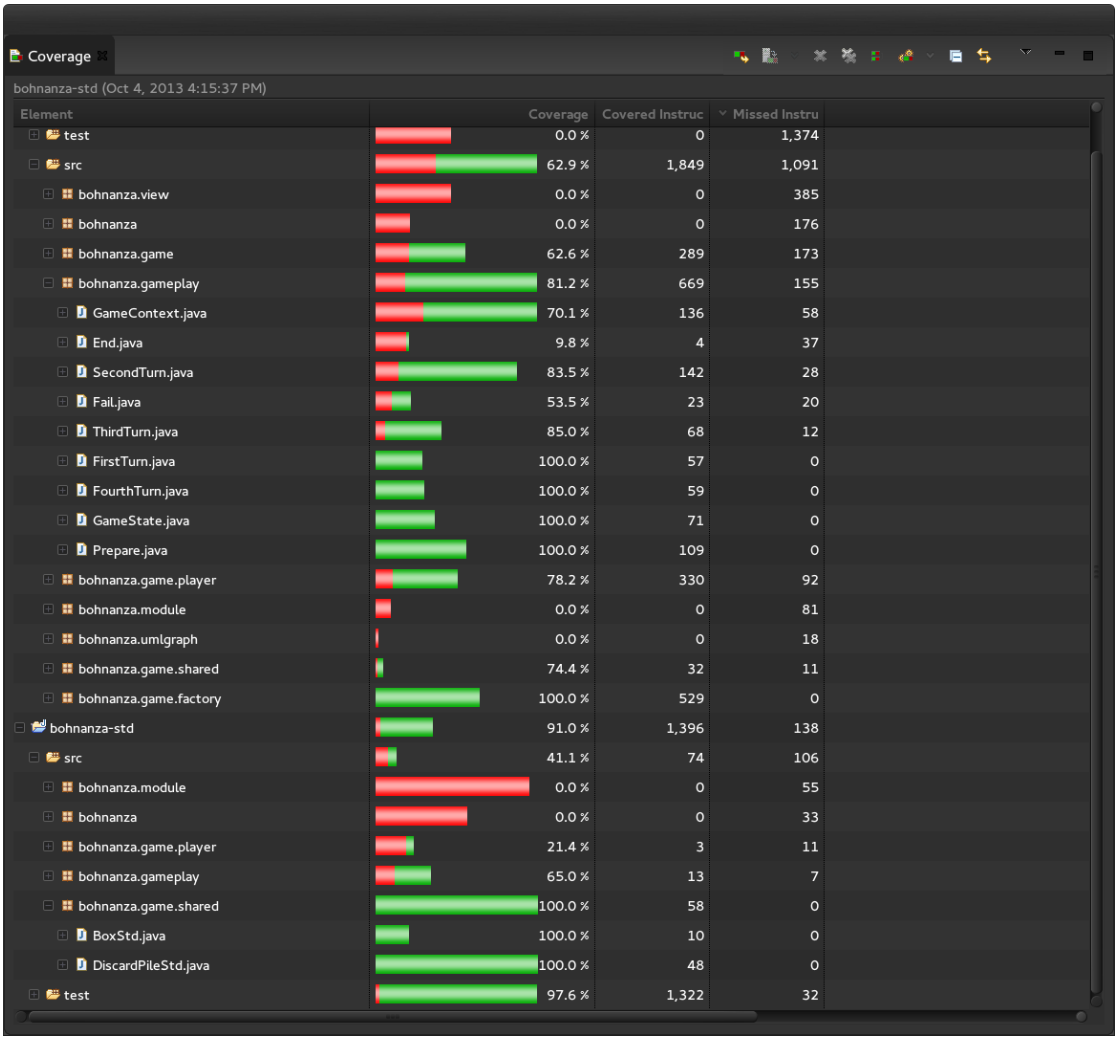

Figure 7: Coverage of the *bohanza-std* module

Notation	**Description**
chili	.
cocoa	.
coffee	.
coin	.
dealer	.
discard pile	.
donate	.
draw	.

Notation	Description
draw area	.
draw deck	.
farm	.
field	.
first draw	.
first turn	.
fourth turn	.
garden	.
green	.
hand	.
harvest and sell	.
keep area	.
last draw	.
left player	.
oldest player	.
plant	.
plant hand	.
plant traded donated	.
player	.
player area	.
red	.
round	.
round one	.
round three	.
round two	.
second turn	.
shuffle	.
soy	.
stink	.

Notation	Description
third turn	.
trade	.
treasury	.
turn	.
wax	.
worth	.

6 appendix

References