CHAPTER 1

A Pattern for Widgets working on Domain Objects

In this chapter we want to present a simple pattern that supports the building of UI interface for editing complex domain objects. We will how we can have a simple object displaying facets of an item.

1.1 Setting up the stage

```
Object subclass: #GameItem
  instanceVariableNames: 'console title'
  classVariableNames: ''
  package: 'GameCollector'
```

```
GameItem >> title: aString
  title := aString
```

```
GameItem >> title
  ^ title
```

```
GameItem >> console
  ^ console
```

```
GameItem >> console: aSymbol
  console := aSymbol
```

```
GameItem class>> consoles
  ^ #( #PS4  #PS3 #PS2 )
```

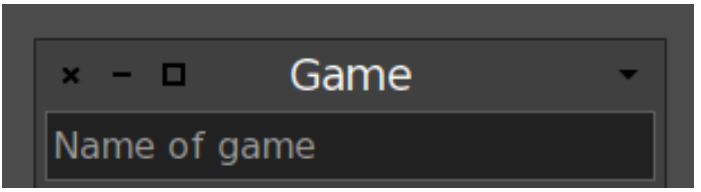

Figure 1.1 First simple input field.

1.2 Simple simple version

```
ComposableModel subclass: #GameItemDisplayer
  instanceVariableNames: 'consoleDropListModel'
  classVariableNames: ''
  package: 'GameCollector-UI'
```

```
GameItemDisplayer >> initializeWidgets
  titleNameTextModel := self newTextInput.
  titleNameTextModel ghostText: 'Name of game'.
```

```
GameItemDisplayer class >> defaultSpec
  ^ SpecLayout composed
    newColumn: [ :col |
      col
        add: #titleNameTextModel height: self inputTextHeight ];
    yourself
```

```
GameItemDisplayer new openWithSpec
```

With these two methods we get a simple widget displaying an input field as shown in Figure 1.1

How to edit an item...

Now we would like to edit a given item (be it given by clicking on a list or specified explicitly). To support this, we will parametrized all the output to pass via a gameItem instance variable holding a valueHolder.

```
ComposableModel subclass: #GameItemDisplayer
  instanceVariableNames: 'gameItem titleNameTextModel'
  classVariableNames: ''
  package: 'GameCollector-UI'
```

We redefine the initialize as follows. It should that there is a hook missing.

```
GameItemDisplayer >> initialize
  gameItem := nil asValueHolder.
  "order important since initialize will invoke initialiseWidgets
    and initializePresenter.
  In the future we should have initializeSubject: "
```

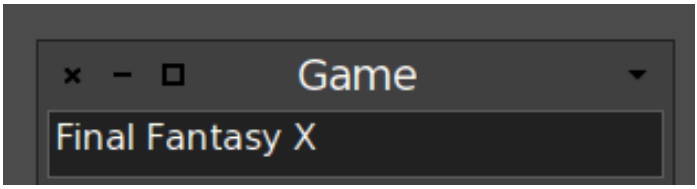

Figure 1.2 Now editing a domain object.

```
  super initialize.
```

We define a simple setter changing the value held by the value holder.

```
GameItemDisplayer >> on: aNewGameItem
  gameItem value: aNewGameItem
```

Now when the game item changes, the name should reflect the change. This is what we do in the `gameItemChanged` method. Here we just change the text of the widget to display the title of the new item.

```
GameItemDisplayer >> gameItemChanged
  titleNameTextModel text: gameItem value title
```

Now in the `initializePresenter` we make sure that when the value of the item is changed, we call the `gametItemChanged` method.

```
GameItemDisplayer >> initializePresenter
  gameItem whenChangedDo: [ self gameItemChanged ]
```

With this simple structure we can now display an item as follows and we should get the result displayed in Figure 1.2.

```
GameItemDisplayer new
  on: GameItem finalFantasyX;
  openWithSpec
```

1.3 Conclusion

We show you the basic architecture to build widgets that work on a model.