

Builder Design Pattern

Encapsulating object creation

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Goals

- Little motivation
- Builder: Power of encapsulating object construction
- Builder uses
 - Settings
 - Microdown
 - Seaside



Creating objects

- Can be cumbersome or complex involving invariants
- Created objects can evolve over time
- Created objects can be changed under the users' feet
- Finding the correct creation API can be daunting



Builder's intent

From the book: Separate the construction of a complex object from its internal representation so that the same the construction process can create different representations



Builder

A builder: an object representing and controlling the creation of other objects

- Encapsulates object creation logic
- Guarantees that the objects are well created
- Decouples object creation from the effectively created objects
 - Supports multiple back-ends



Setting example

BeautifulComments class >> beautifulCommentsSettingsOn: aBuilder

```
<systemsettings>
(aBuilder setting: #rendering)
  parent: #microdownAndcomments;
  label: 'Enable richtext comments';
  default: true;
  target: self;
  description: self renderingDocForSetting.
(aBuilder setting: #captureErrors)
  parent: #microdownAndcomments;
  label: 'Enable rendering error capture';
  default: true;
  target: self;
  description: self captureErrorsDocForSetting
```



Setting Builder API

SettingNodeBuilder selectors sorted

#category:

#default: #description: #dialog:

#domainValues: #getSelector:

#ghostHelp: #icon: #iconName: #label: #name:

#noOrdering #order: #parent: #precondition: #range: #script:

#selector: #shortcutName: #target:

#targetSelector: #type:

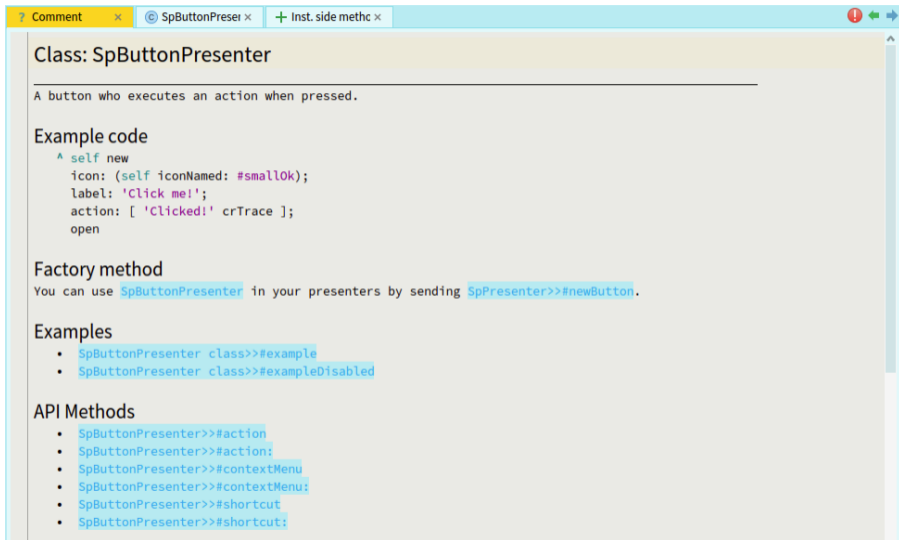


Setting builder analysis

- **Avoid** hardcoding references to Setting objects in the domain
- Act as a DSL
- Guarantee that the objects are well created
- **Encapsulate creation logic**
- Decouple object creation from the effectively created objects



Microdown builder in action



The screenshot shows a code editor window with three tabs: "Comment", "SpButtonPresenter", and "Inst. side methc". The active tab displays the documentation for the `SpButtonPresenter` class. The documentation includes a class title, a description, an example code block, a factory method description, and lists of examples and API methods.

Class: `SpButtonPresenter`

A button who executes an action when pressed.

Example code

```
^ self new
  icon: (self iconNamed: #smallOk);
  label: 'Click me!';
  action: [ 'Clicked!' crTrace ];
  open
```

Factory method

You can use `SpButtonPresenter` in your presenters by sending `SpPresenter>>#newButton`.

Examples

- `SpButtonPresenter class>>#example`
- `SpButtonPresenter class>>#exampleDisabled`

API Methods

- `SpButtonPresenter>>#action`
- `SpButtonPresenter>>#action:`
- `SpButtonPresenter>>#contextMenu`
- `SpButtonPresenter>>#contextMenu:`
- `SpButtonPresenter>>#shortcut`
- `SpButtonPresenter>>#shortcut:`

Microdown builder API example

MicMicrodownTextualBuilder selectors sorted

#anchor: #anchorReference: #bold: ...

#codeblock:firstLineAssociations: #codeblock:firstLineAssociations:withCaption: ...

#comment: ...

#environment:body:arguments: ...

#figureURLString:withCaption:withParameters: ...

#header:withLevel: #horizontalLine #internalLink: ...

#italic: #item: ...

#mathInline: #mathblock: ...

#metaDataFrom: ...

#orderedItem: #orderedItem:startingAt: #orderedListDuring: #paragraph: ...

#raw: #strike: ...



Microdown builder

```
testCodeBlock
```

```
| mictext |
```

```
mictext := builder
```

```
  codeblock:
```

```
'Here is an example of  
code block'
```

```
  firstLineAssociations: { ('language2' -> 'Pharo') };
```

```
  contents.
```

```
  self assert: mictext equals: '` `` language2=Pharo
```

```
Here is an example of  
code block
```

```
`` `
```

```
'
```



Microdown builder analysis

- Provides a high-level API to script Microdown text
- Avoid string manipulation!
- Let Microdown evolves without impacting users!



Seaside builder

```
ScrapBook >> renderContentOn: html
  html heading: 'Hello world'.
  html paragraph: 'Welcome to my Seaside web site. In the
future you will find all sorts of applications here
such as:'.
  html orderedList: [
    html listItem: 'Calendars'.
    html listItem: 'Todo lists'.
    html listItem: 'Shopping carts'.
    html listItem: 'And lots more...' ]
```



When to apply it

- The domain is structured and has some regularity in the object creation
- When we want one single entry point (e.g., refactoring)
- To stabilise an API, while the implementation is evolving



Conclusion

A builder: an object representing and controlling the creation of other objects

- Encapsulates object creation logic
- Guarantees that the objects are well created
- Decouples object creation from the effectively created objects
- Supports evolution



Produced as part of the course on <http://www.fun-mooc.fr>

Advanced Object-Oriented Design and Development with Pharo

A course by

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Inria
LearningLab



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>