The Abstract Factory Pattern

Prof. Dr. Eric Dubuis
Berner Fachhochschule,
Engineering and Information Technology @ Biel

Course "UML and Design Patterns" of module "Software Engineering and Design",
version May 2008
The Problem

Given a framework, and the framework itself uses a set of different, but similar Command objects. You want, however, allow your framework users to provide their concrete implementations of these commands.

How can Client instantiate these classes?

Client

<<interface>>

Command

<<use>>

<<create>>

MyCutCommand

MyMoveCommand

MyResizeCommand

CutCommand

MoveCommand

ResizeCommand

{abstract}

... CutCommand(...)... MoveCommand(...)... ResizeCommand(...)...
General Ideas for a Solution

- Class `Client` does not know the concrete `Command` classes.
- Instead, `Client` uses a (known) `factory class`.
- A concrete instance of a factory knows how to instantiate the concrete `Command` objects.
Adding a Command Factory to the Graphical Editor

```
JdtEditor

<<interface>>
+createMoveCommand(): Command
...

<<create>>
+createMoveCommand(): Command
...

<<use>>
MoveCommand
{abstract}
MoveCommand(....)

MyMoveCommand
YourMoveCommand
...

<<create>>
CutCommand
{abstract}
CutCommand(....)

MyCutCommand
YourCutCommand
...
```
The Classes

We first start with the abstract Command classes:

```java
public abstract class MoveCommand implements Command {
    protected MoveCommand(...) { ... }
    public void execute() { ... }
    ...
}
```

```java
public abstract class CutCommand implements Command {
    protected CutCommand(...) { ... }
    public void execute() { ... }
    ...
}
```

et cetera.
Class **MyCommandFactory** implements a set of factory methods that create **Command** objects:

```java
public class MyCommandFactory implements CommandFactory {

    public MoveCommand createMoveCommand(Shape s, Point origin, Point end) {
        return
    }

    public CutCommand createCutCommand(View v, Shape s) {
        return
    }

} // MyCommandFactory
```

Each factory method returns an concrete `XxCommand` object which the editor can use via the common `XxCommand` abstraction.
Remarks

- The methods of the factory class that create (here: Command) objects are so-called *factory methods* (as with the factory method pattern).

- However, the only responsibility of the factory class is to create objects. No other business methods are supported by this class. This is in contrast to the factory method pattern: There, the class with the factory method has other responsibilities, too.
The Structure of the Abstract Factory Pattern

Client

<<interface>>

<<create>>

Concrete

<<create>>

<<create>>

AbstractProduct1

{abstract}

...could be an interface, too...

AbstractProduct2

{abstract}

...
References


- Vince Huston's illustration of the command pattern:
  http://home.earthlink.net/%7Ehuston2/dp/abstract_factory.html