

# Chapter 10

## Design Patterns (4)

### Factories

Jean Privat

IT069IU — Object-Oriented Analysis and Design  
2015

# Overview

- 1 Template Method
- 2 Simple Factory
- 3 Factory Method
- 4 Abstract Factory

# Overview

## 1 Template Method

## 2 Simple Factory

## 3 Factory Method

## 4 Abstract Factory

# Template Method

## Context/Problem

- Factorize the skeleton of an algorithm in an super-class
- Defer some steps of an operation to subclasses
- Impose an algorithmic structure to subclasses (e.g. mandatory steps)

# Template Method

## Solution

- Implement the operation skeleton in the superclass
- Define steps as abstract (or noop) sub-methods
- Subclasses implement (or redefine) the steps (not the operation)

## Participant

- AbstractClass: implement a template method that use submethods
- ConcreteClass: redefine the submethod
- Client: call the template method of the concrete class (not the steps)

# Template Method

## Discussion

- How to prevent the concrete class to redefine the whole template method?
- How to prevent the client to directly call the steps?

## Anti-pattern

- Mandatory super call

## Note

- Not to be confused with parametric (generic) methods

# Overview

1 Template Method

2 **Simple Factory**

3 Factory Method

4 Abstract Factory

# Simple Factory (not GoF)

## Context/Problem

- Who should be responsible for creating objects when there are special considerations:
- Complex logic: creation code that we want to factorize or hide
- Memory mechanism: reuse/recycle existing objects
- Control creation: checking some validity rules
- Specialization: return a more specific object
- Code quality: a desire to separate the creation responsibilities for better cohesion

# Simple Factory

## Solution

- Create a Pure Fabrication object called a Factory that handles the creation

## Participants

- Product: the object to create
- Creator: the object responsible of the creation

# Simple Factory

## Advantage

- Separate the responsibility of complex creations
- Hide potentially complex creation logic
- Possible performance-enhancing strategies

## Issues

- Who create the factory?
- A standard new in some owner class
- A singleton

## Tip

- Make the constructor of the product protected (or private) to avoid the hijacking of the factory

# Overview

1 Template Method

2 Simple Factory

**3 Factory Method**

4 Abstract Factory

# Factory Method

## Context/Problem

- A class cannot anticipate how to make an object it must create
- A class wants its subclasses to specify the objects it must create

## Example

- An application framework that produces documents:
- A word processor produces text documents
- A spreadsheet application produces spreadsheets

# Factory Method

## Solution

- Provide an interface for creating objects
- Let subclasses implement the effective object creations

## Participants

- Product: the interface of the type of objects to create
- ConcreteProduct: a specific product that can be created
- Creator: the interface that abstracts the creation service
- ConcreteCreator: a specific creator that implements the creation service

# Factory Method

## Discussion

- The creator interface is defined without knowing about concrete products
- The concrete product is determined by the concrete creator
- It can be combined with Simple Factory so that a concrete creator perform complex creation

## Advantages

- New kinds of creators and products can be added easily

# Overview

1 Template Method

2 Simple Factory

3 Factory Method

4 **Abstract Factory**

# Abstract Factory

## Context/Problem

- Create families of related objects
- Do not want that client has to know the specific families

## Example

- Multi-platform GUIs

# Abstract Factory

## Solution

- Provide an interface for creating a family of objects
- Let subclasses implement the effective object creations

## Participants

- Product: the interfaces of the object to create
- Concrete Product: a specific product in a specific family
- Abstract Factory: the abstract class that declares a create method for each kind of product
- Concrete Factory: the concrete class that implement the create methods

# Abstract Factory

## Discussion

- It is a generalisation of the Factory Method
- Easy to add a new family of products and its factory
- High cohesion because related classes (family) are regrouped

## Issue

- Hard to add a new kind of product (need to update each factory)