Design Patterns Introduction, Factories/Singleton

Sébastien Jean

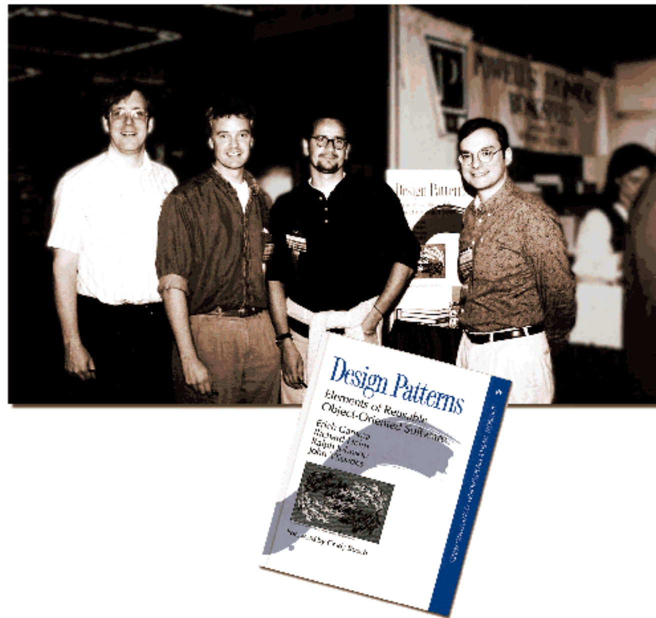
IUT de Valence
Département Informatique

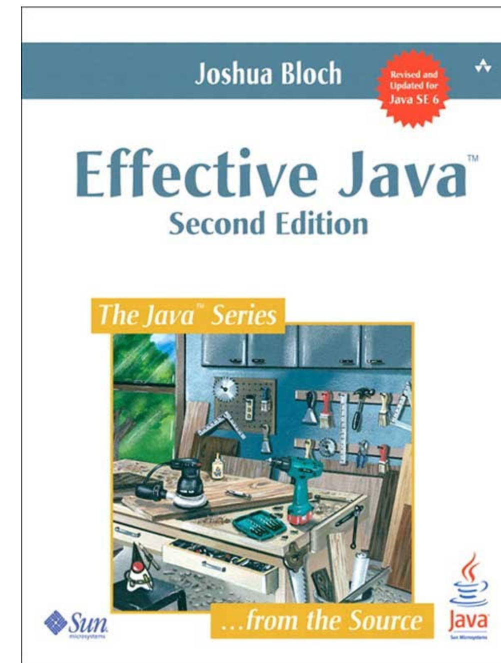
v4.5, 7 octobre 2024

IUT Valence
Université Grenoble Alpes

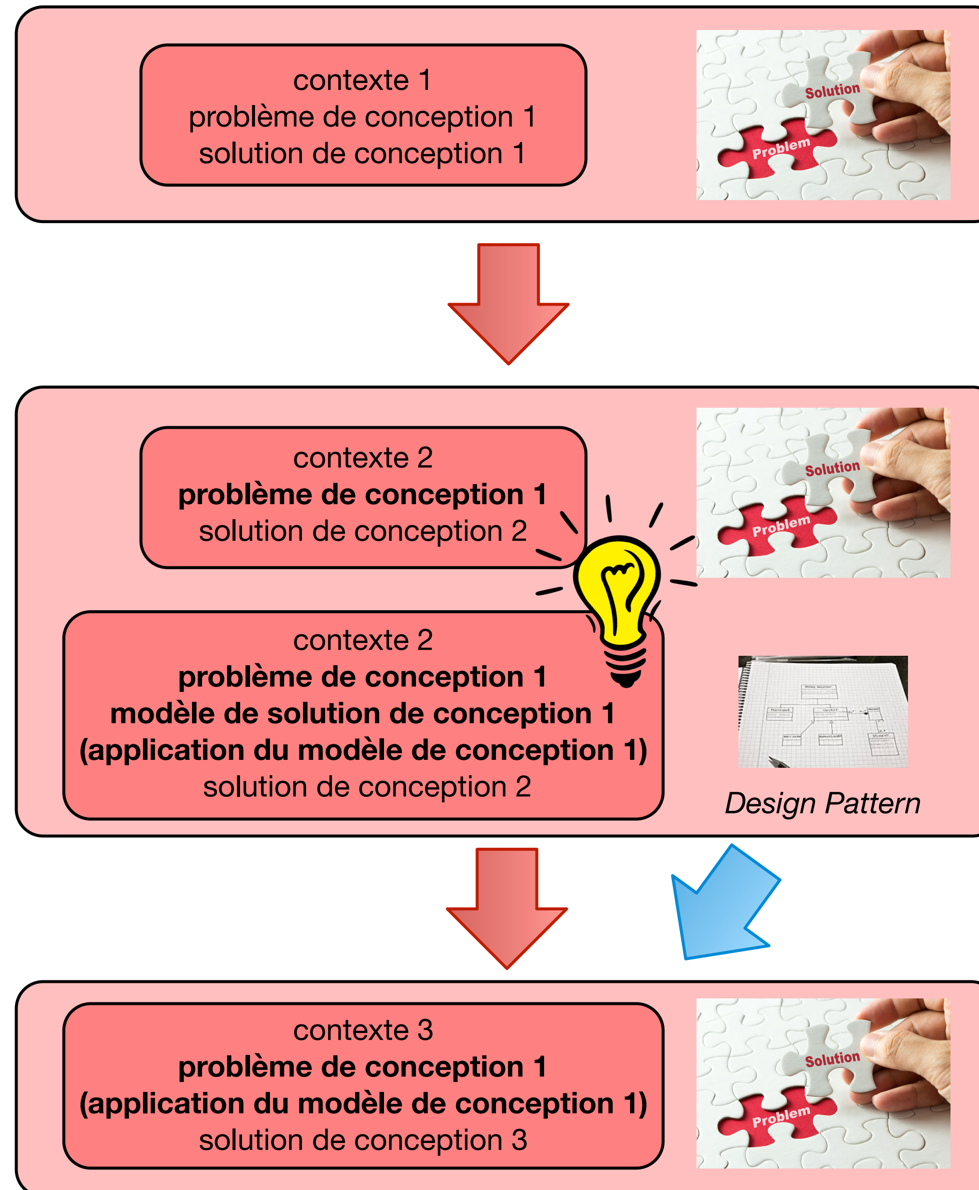
Ouvrages de référence

- Gang of Four (GoF) : Gamma, Helm, Johnson, Vlissides (1995)
  - *Design Patterns : elements of reusable Object-Oriented Software*
- Joshua Bloch
  - *Effective Java*





Patron de conception ?



IUT Valence
Université Grenoble Alpes

Qu'est ce qu'un patron de conception ?

- Un *design pattern*, ou **patron de conception**, est :
  - Une solution élégante et efficace à un problème connu et récurrent de conception/développement par objets
  - Incarné par un ensemble d'objets et de relations entre objets
  - Issu de l'expérience des programmeurs (pratique *Vs* théorie)
  - Basé sur les bonnes pratiques de programmation OO
- Un *design pattern* est défini par :
  - Son **nom**
  - Le **problème** auquel il s'intéresse et sa façon de le résoudre

IUT Valence
Université Grenoble Alpes

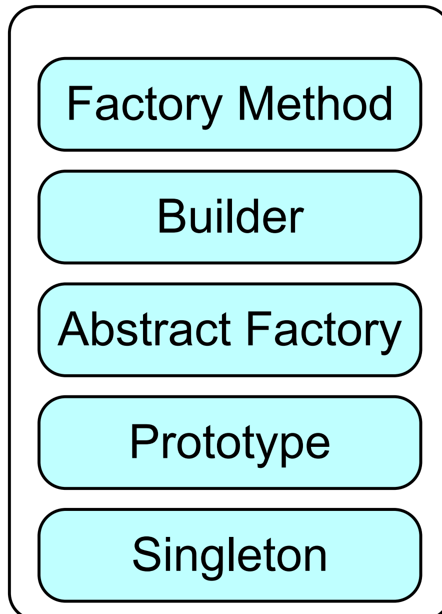
Classification des patrons de conception

- Il existe 23 *design patterns* principaux
- 5 Patterns de **construction** (*Creational*)
  - Organisation de la création d'objets
- 7 Patterns de **structuration** (*Structural*)
  - Organisation de la hiérarchie des classes et de leur utilisation
- 11 Patterns de **comportement** (*Behavioral*)
  - Organisation des interactions et répartition des traitements

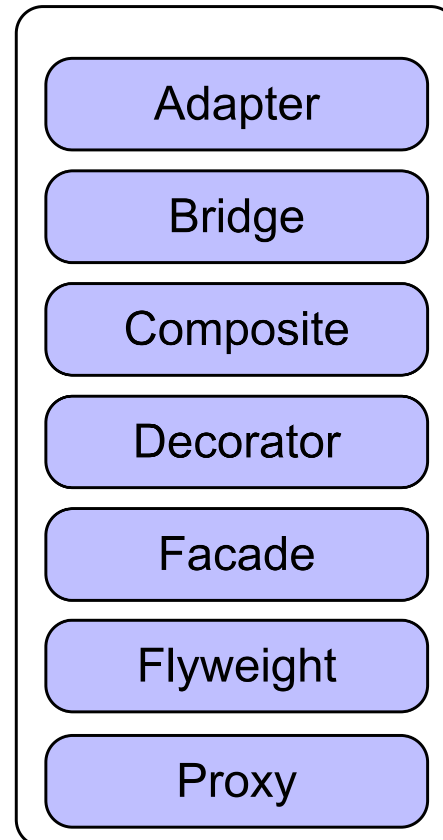
IUTA Valence
Université Grenoble Alpes

Classification des patrons de conception, suite

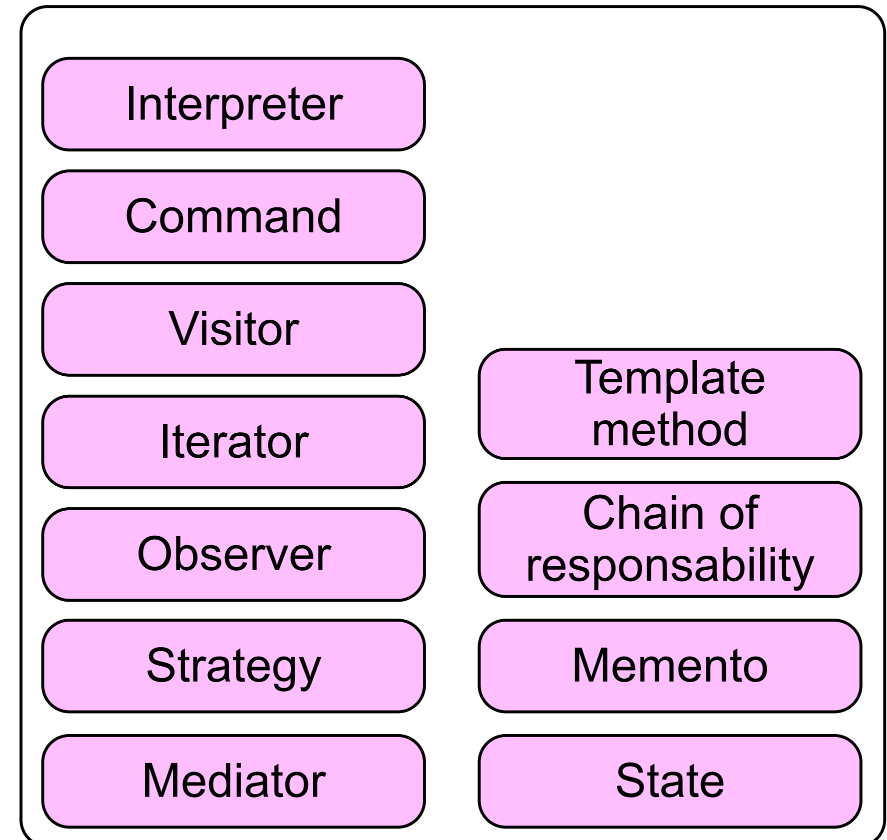
Creational



Structural



Behavioral



Fabriques *Vs* constructeurs : constructeurs

- Le **constructeur** est un moyen simple d'obtenir un objet, de différentes manières (paramètres)
- **mais . . .**

Fabriques *Vs* constructeurs : constructeurs

- **Pas de nom évocateur**, surcharges pas toujours lisibles

Constructor Summary

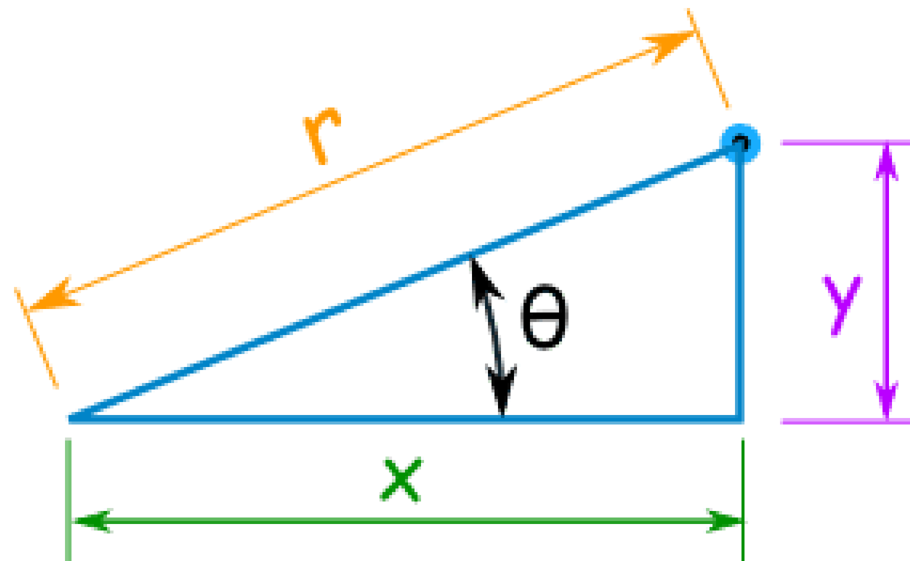
Constructors

Constructor and Description
`DatagramPacket(byte[] buf, int length)` Constructs a `DatagramPacket` for receiving packets of length `length`.
`DatagramPacket(byte[] buf, int length, InetAddress address, int port)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int offset, int length)` Constructs a `DatagramPacket` for receiving packets of length `length`, specifying an offset into the buffer.
`DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)` Constructs a datagram packet for sending packets of length `length` with offset `ioffset`to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)` Constructs a datagram packet for sending packets of length `length` with offset `ioffset`to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int length, SocketAddress address)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.

IUTA Valence
Université Grenoble Alpes

Fabriques *Vs* constructeurs : constructeurs

- Surcharge pas toujours possible
- Exemple : construction d'un point (classe `Point`)
  - à partir de coordonnées cartésiennes : `Point(double x, double y)`
  - à partir de coordonnées polaires : `Point(double r, double teta)`



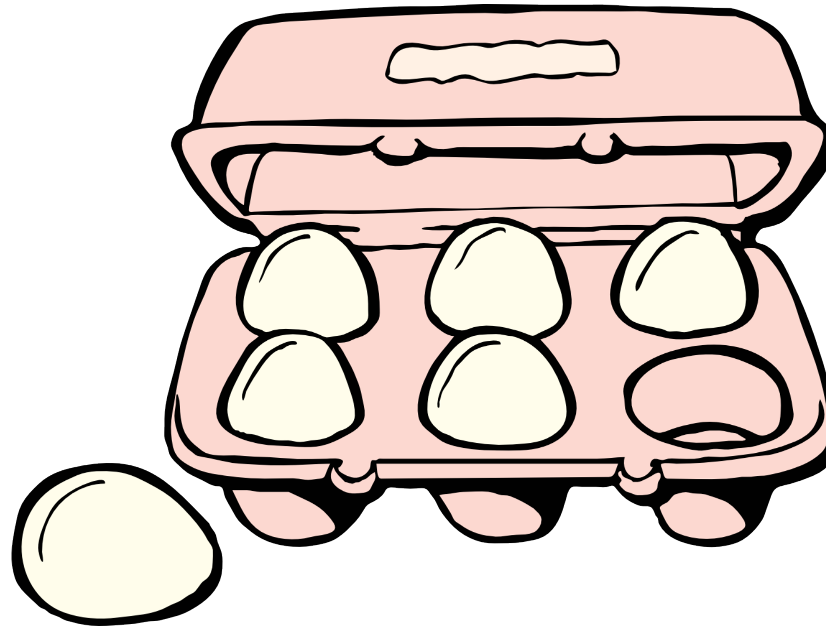
Fabriques *Vs* constructeurs : constructeurs

- Le constructeur retourne toujours un **objet du type de la classe qui le définit**
  - il faut nécessairement connaitre le nom de la classe à instancier
  - on pourrait mieux utiliser le polymorphisme ...

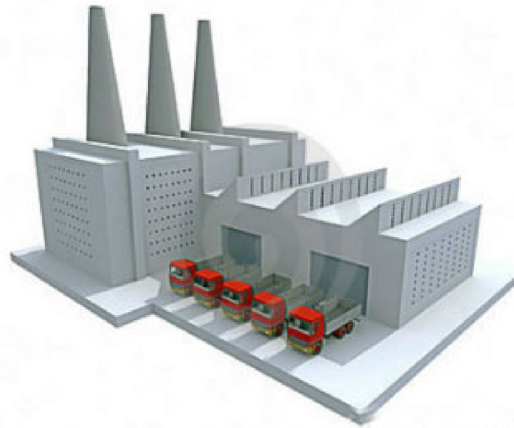
IUTA Valence
Université Grenoble Alpes

Fabriques *Vs* constructeurs : constructeurs

- Le constructeur retourne toujours un **nouvel objet**
  - on pourrait mieux utiliser les ressources ...



Fabriques *Vs* constructeurs : fabriques

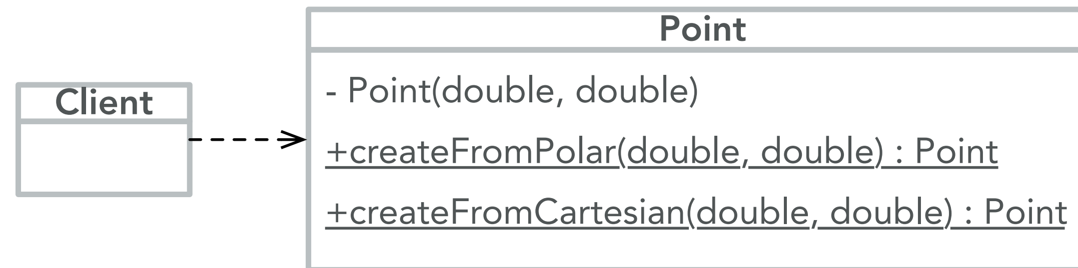


- Les **fabriques** (*factories*) se substituent plus élégamment aux constructeurs . . .et
  - donnent un **nom évocateur** au mécanisme d'obtention d'objet
  - peuvent retourner un objet de n'importe quel **sous-type** du type d'objet qu'elles déclarent retourner
  - ne retournent pas systématiquement un **nouvel objet**
    - contrôle/limitation du nombre d'objets créés, recyclage d'instances inutilisées, . . .

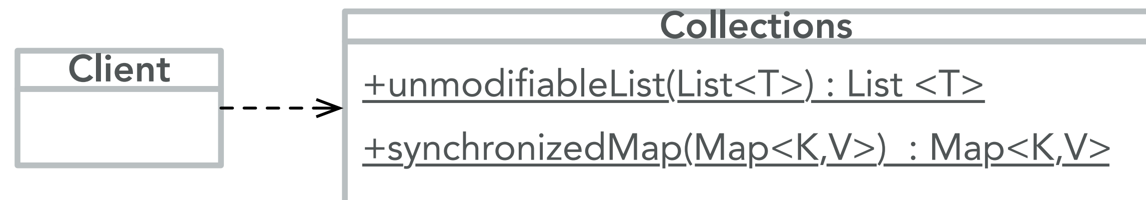
IUTA Valence
Université Grenoble Alpes

Static Factory Method

- **Méthode statique se substituant à un constructeur**
  - nom évocateur, surcharges claires

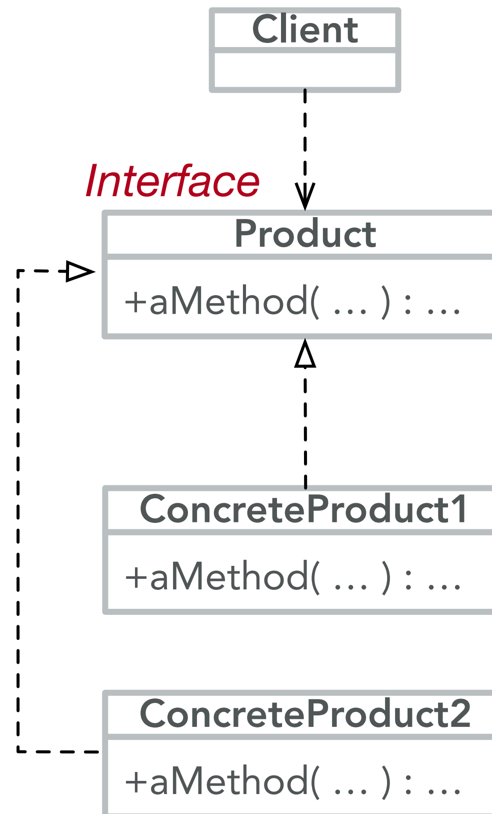


  - création d'objet de type spécifique non connu à priori



  - contrôle/limitation/optimisation du nombre d'instances

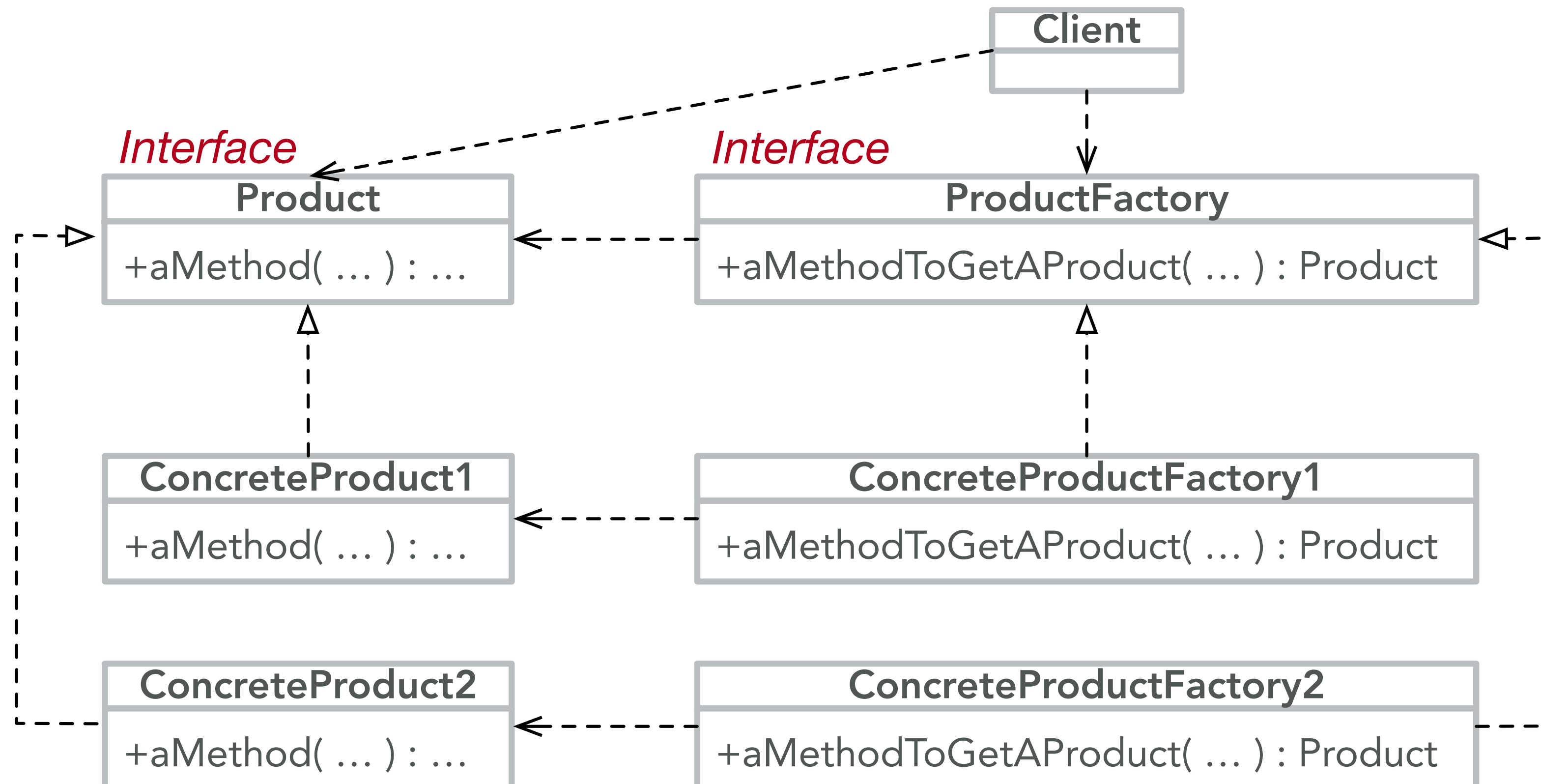
Factory Method : problème



- Le client a besoin d'obtenir et manipuler un produit
- Le produit est vu de manière abstraite,
  le client ne connait pas les implémentations

Factory Method : solution

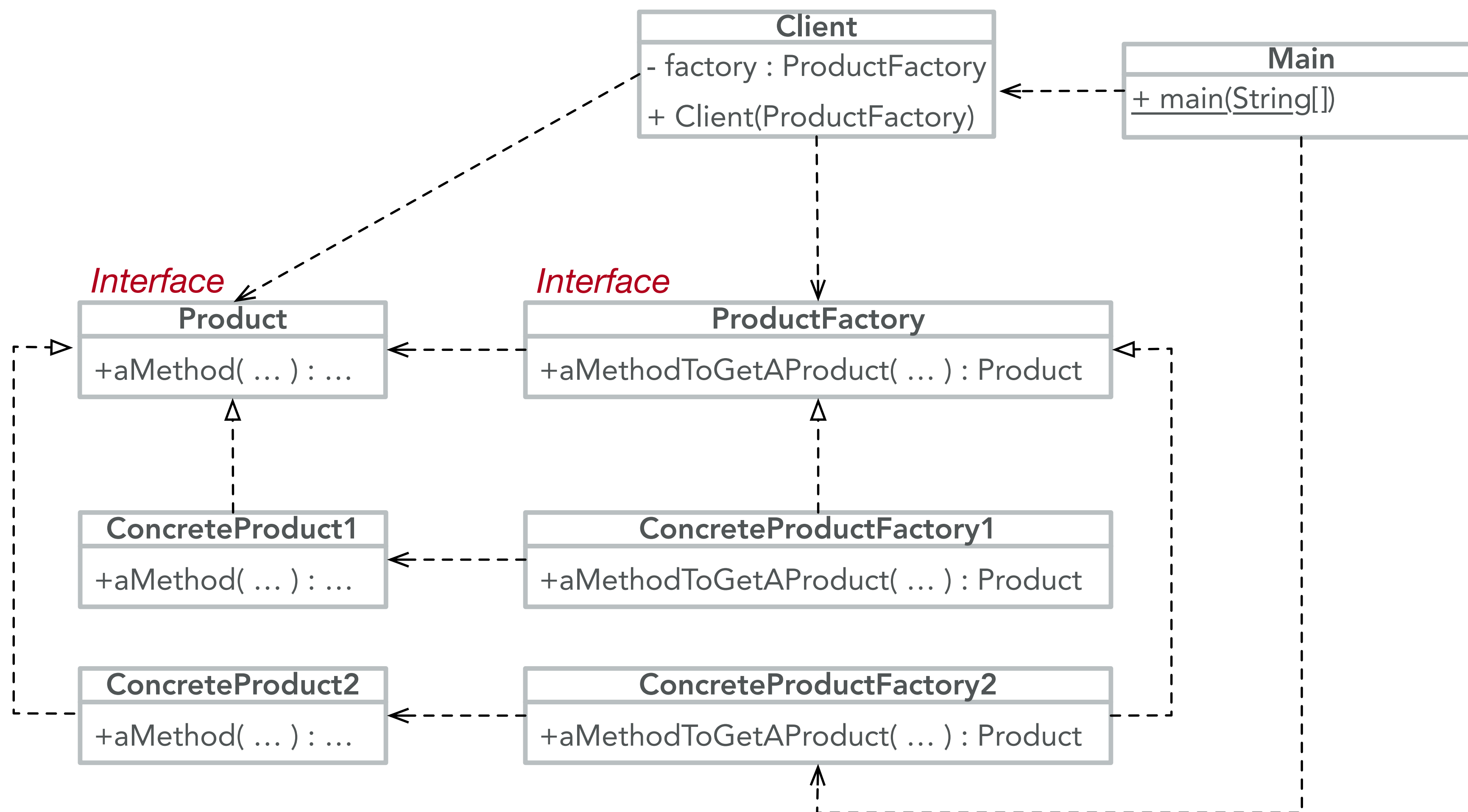
- On donne au client une **fabrique** qui fournit une méthode (méthode fabrique) permettant la création/obtention du produit
- La fabrique est abstraite, la construction d'un nouvel objet (appel au constructeur) est **déléguée** aux sous-classes/implémentations



IUT Valence
Université Grenoble Alpes

Factory Method : association client/fabrique

- Le **choix du produit concrêt**, et donc de la fabrique concrète à utiliser, est effectué an amont, avant l'instantiation du client.
  - dans l'exemple ci-dessous, c'est le `Main` qui s'en charge



Factory Method : association client/fabrique (fin)

- Code du `Main`

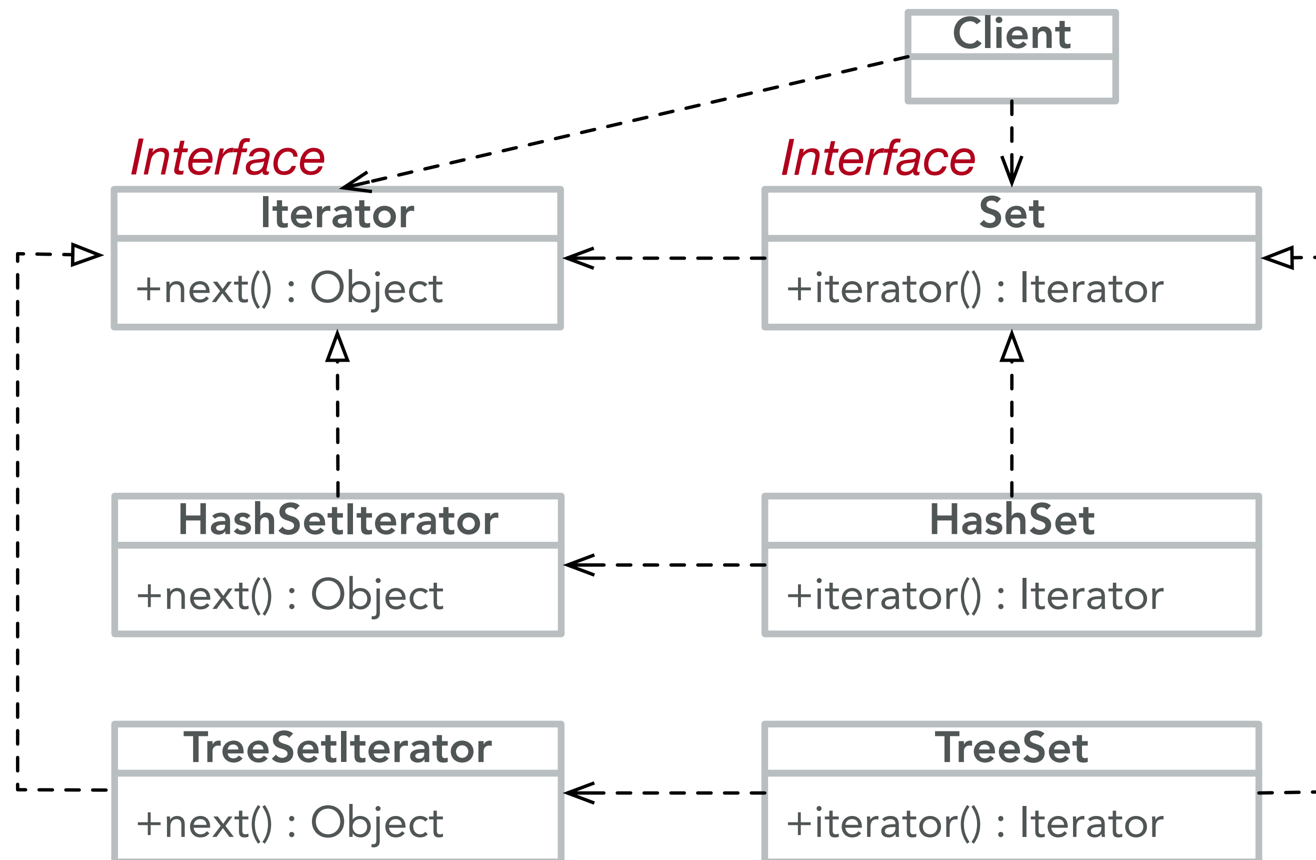
```
public class Main {
    public static void main(String[] args) {
        ProductFactory factory = new ConcreteProductFactory2();
        Client client = new Client(factory);
    }
}
```

- Le code du client n'a pas besoin d'être modifié pour prendre en compte un nouveau type de produit
  - Injection de dépendance
  - Il suffit simplement d'écrire un autre `Main`

IUTA Valence Université Grenoble Alpes

Factory Method : exemple

- Itérateurs de collections (méthode `iterator()`)

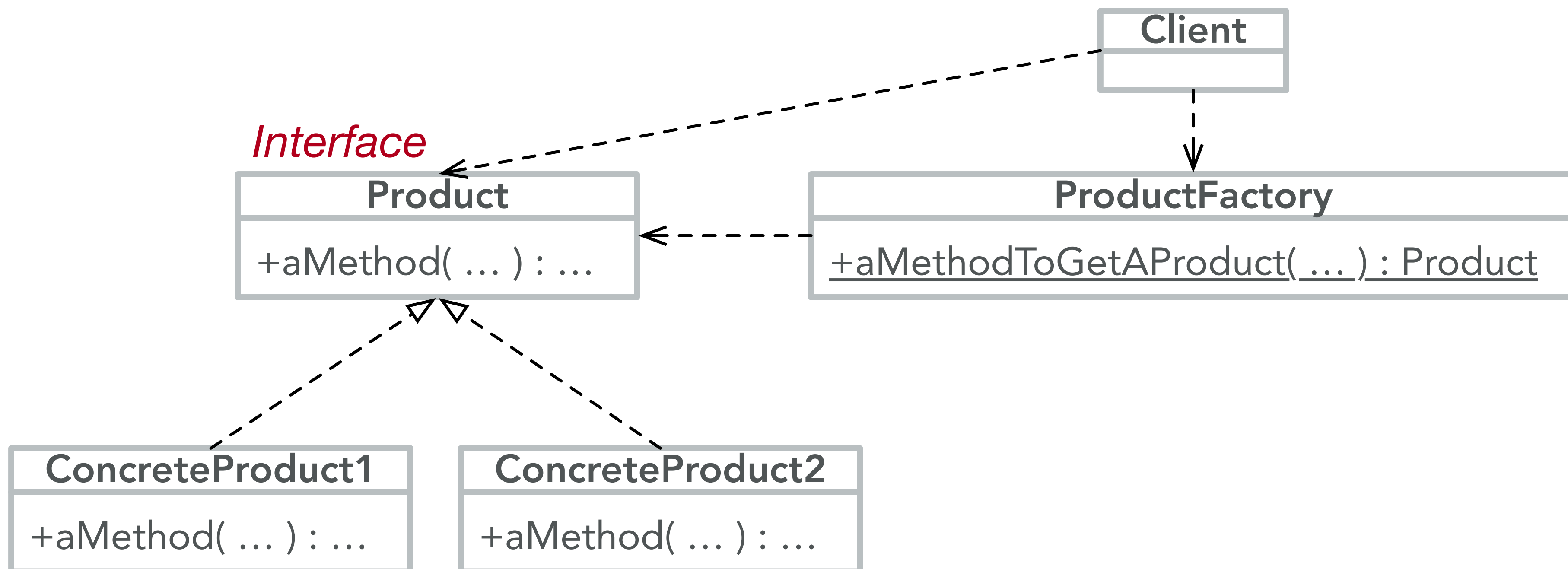


- la collection est la fabrique d'itérateur, chaque implémentation concrète instancie l'objet approprié

IUT Valence Université Grenoble Alpes

Factory Method, version statique

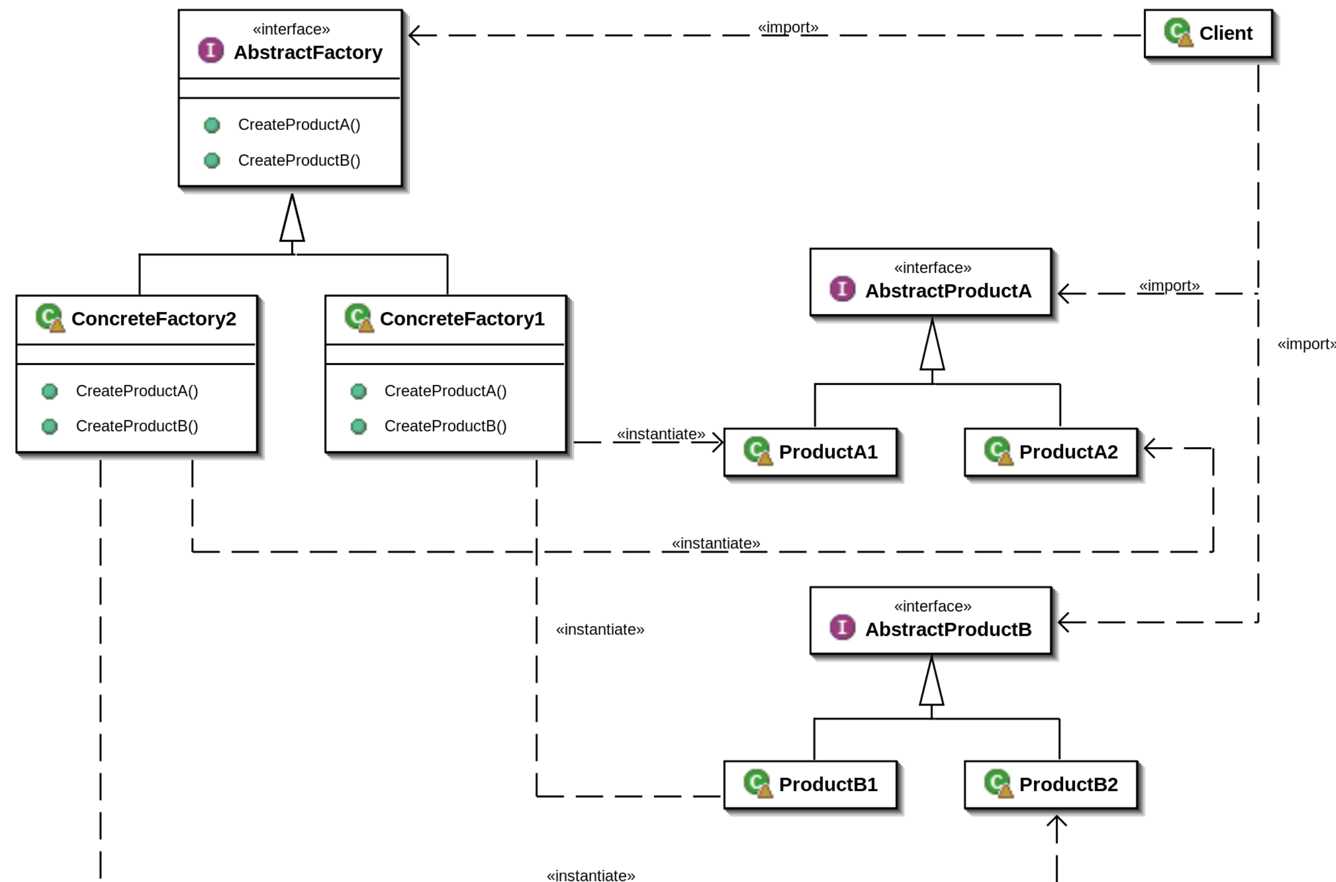
- Cas particulier où la hiérarchie de fabriques est remplacée par une fabrique concrète avec une méthode fabrique statique



- La classe `ProductFactory` est un point d'entrée unique en charge de **fabriquer un objet d'un sous-type de** `Product` choisi en fonction de paramètres passés ou d'un contexte global

Abstract Factory

- Variante de *Factory Method* pour créer des **familles d'objets** sans connaitre leurs classes concrètes



- N.B. : Choix de la fabrique effectué en amont , référence souvent passé au client en paramètre du constructeur

IUTA Valence Université Grenoble Alpes

Singleton

- S'assurer qu'**une ressource est unique** et fournir un point d'accès global à cette ressource
- Mise en œuvre possible
  - Un attribut statique privé contenant la référence de l'instance
  - Un constructeur privé interdisant la création de nouveaux objets
  - Une méthode statique permettant d'obtenir la référence de l'instance

MySingleton
- instance : MySingleton
- mySingleton() + getInstance() : MySingleton + doSomething() : void

IUTA Valence Université Grenoble Alpes

Singleton, en Java

- **Initialisation paresseuse** (*lazy initialization*)
  - Economie de la création de l'objet s'il n'est pas utilisé

```java
public class LazySingleton {

    private static LazySingleton instance = null;

    private LazySingleton() { /* ... */}

    public static LazySingleton getInstance() {
        if (instance == null)
            instance = new LazySingleton();

        return instance;
    }

    public void doSomething() { /* ... */ }
}
```

IUTA Valence Université Grenoble Alpes

Singleton, en Java

- **Initialisation hâtive** (*eager initialization*)

```java
public class EagerSingleton {

    private final static EagerSingleton INSTANCE = new EagerSingleton();

    private EagerSingleton() { /* ... */}

    public static EagerSingleton getInstance() {

        return INSTANCE;
    }

    public void doSomething() { /* ... */ }
}
```

- **Enumération**

```java
public enum EnumSingleton {

    INSTANCE;

    public void doSomething() { /* ... */ }
}
```

Singleton, exemples d'utilisation

- *Toolkits*
  - la classe `java.lang.Math` regroupe des méthodes utilitaires mathématiques (`abs`, `sin`, `pow`, ...)
    - N.B. : l'instance est confondue avec la classe
- *Annuaires*
  - `DriverManager` **JDBC**
    - point d'entrée unique utilisé par les drivers pour s'enregistrer et par les applications pour localiser les drivers
- *Fabriques*

IUT Valence
Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes