Design Patterns
Prototype, Builder

Sébastien Jean

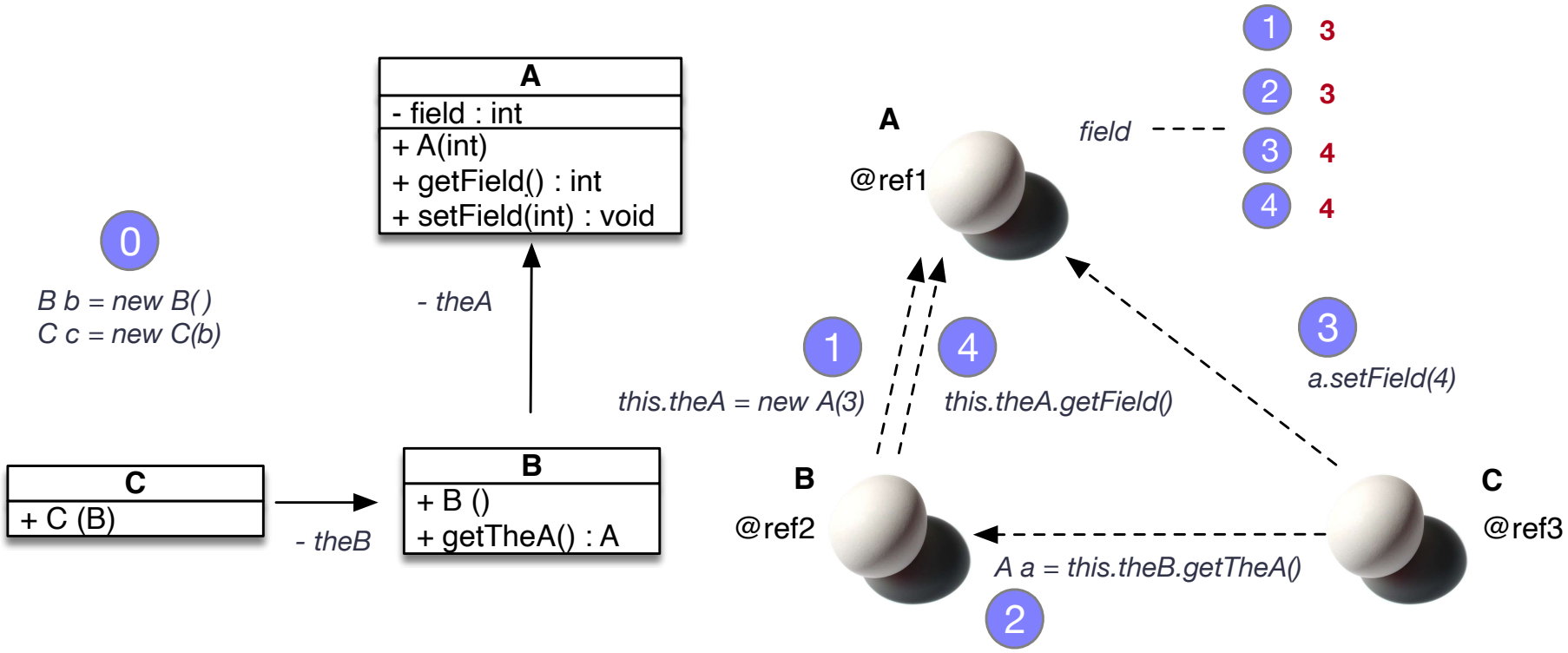
IUT de Valence
Département Informatique

v7.1, 20 octobre 2023

IUT Valence
Université Grenoble Alpes

Partage d'objets

- Les objets sont transmis et manipulés par référence
  - Les modifications sont visibles depuis chaque *client* puisque c'est la même instance qui est utilisée

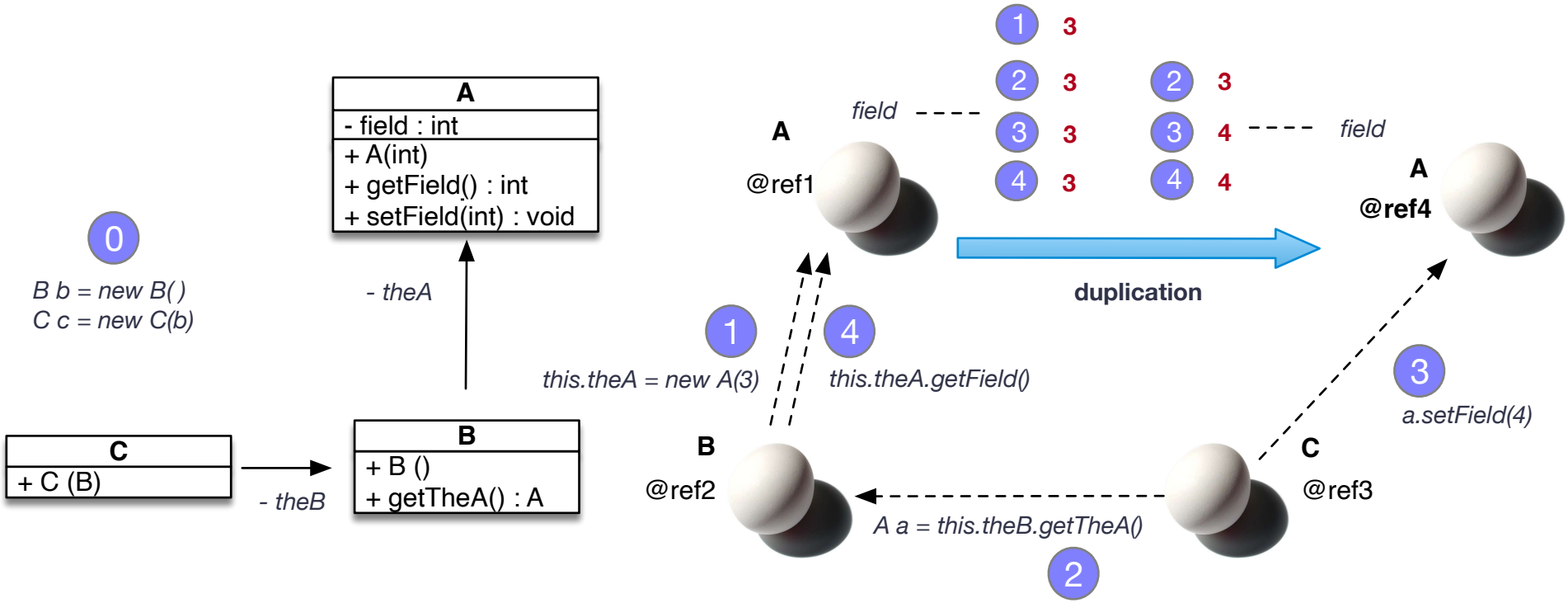


- Il est parfois utile de **protéger les objets des modifications** et d'effectuer **un partage *en lecture seule***

IUT A Valence Université Grenoble Alpes

Duplication

- Solution : **dupliquer les objets** lors du partage

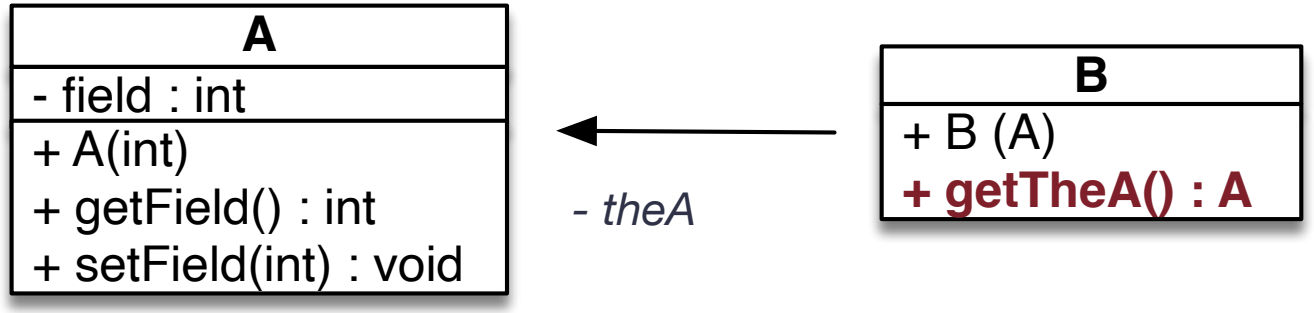


- N.B. : dupliquer est aussi utile pour éviter de reconstruire de zéro des objets dont la **création est très couteuse**

IUT A Valence Université Grenoble Alpes

Comment dupliquer ?

- Solution 1 : **appel aux getters + constructeur**



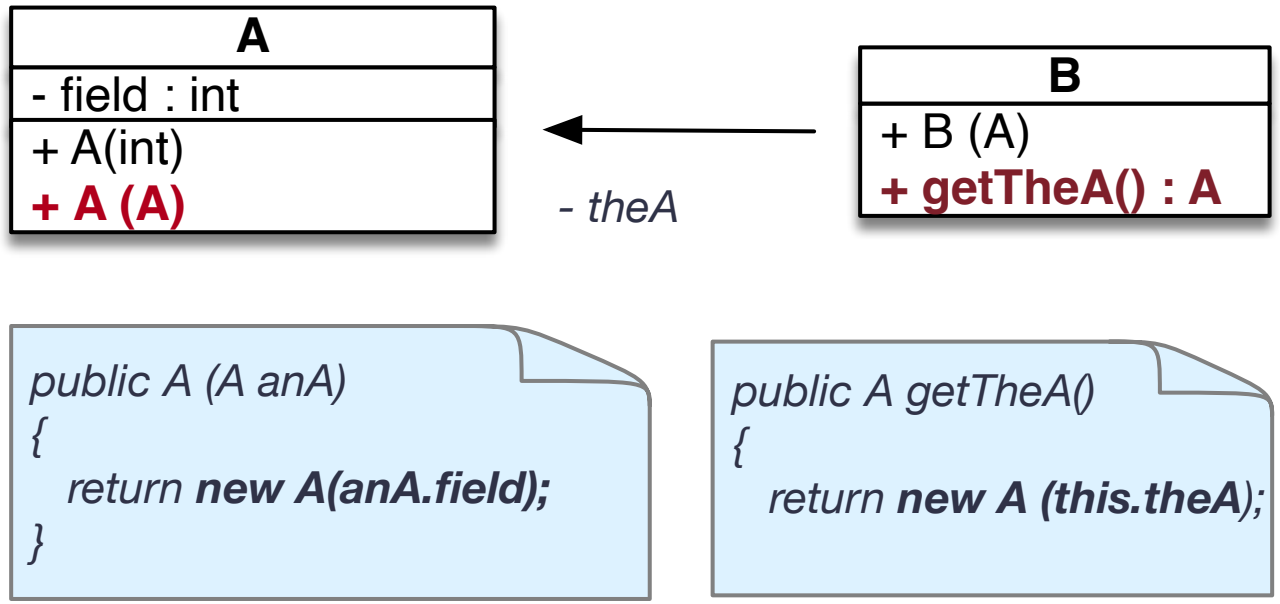

```
public A getTheA()
{
   return new A (this. theA.getField());
}
```

- Mais dans certains cas il n'est pas possible d'extraire depuis l'extérieur tout l'état de l'objet ...

IUT Valence Université Grenoble Alpes

Comment dupliquer ?

- Solution 2 : **constructeur par copie**

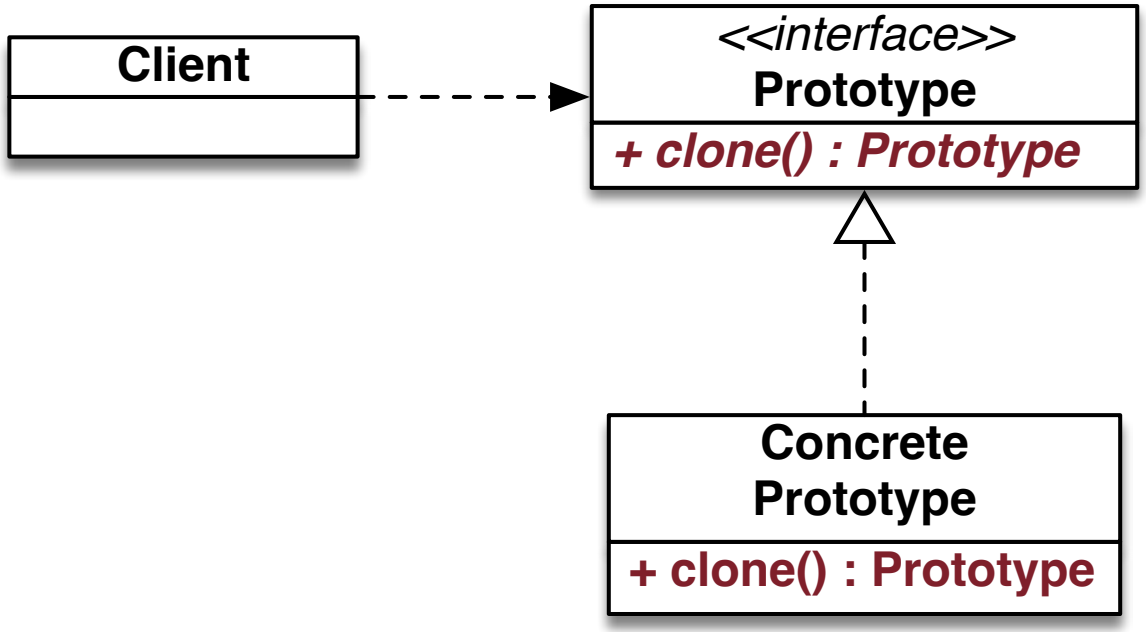


- Mais dans certains cas le type de l'objet à dupliquer n'est pas connu (manipulation via un type abstrait) ...

IUT A Valence Université Grenoble Alpes

Pattern *Prototype*

- Méthode fabrique permettant d'obtenir un clone

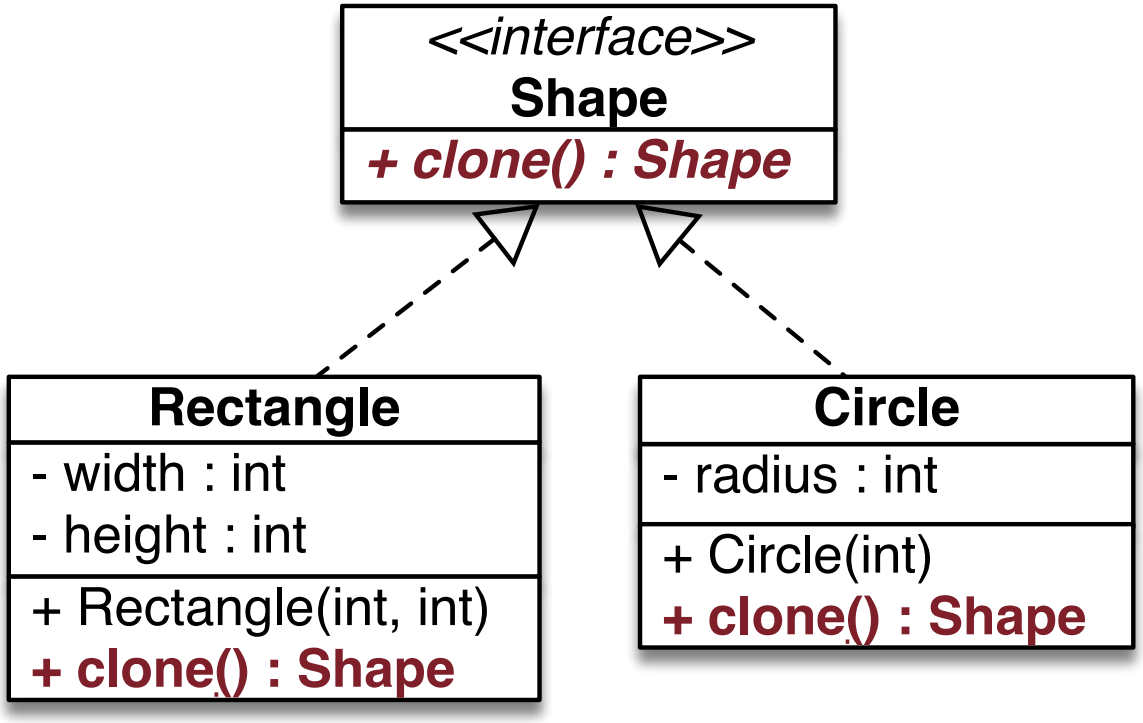


- Chaque implémentation concrète met en oeuvre son propre clonage

IUT Valence
Université Grenoble Alpes

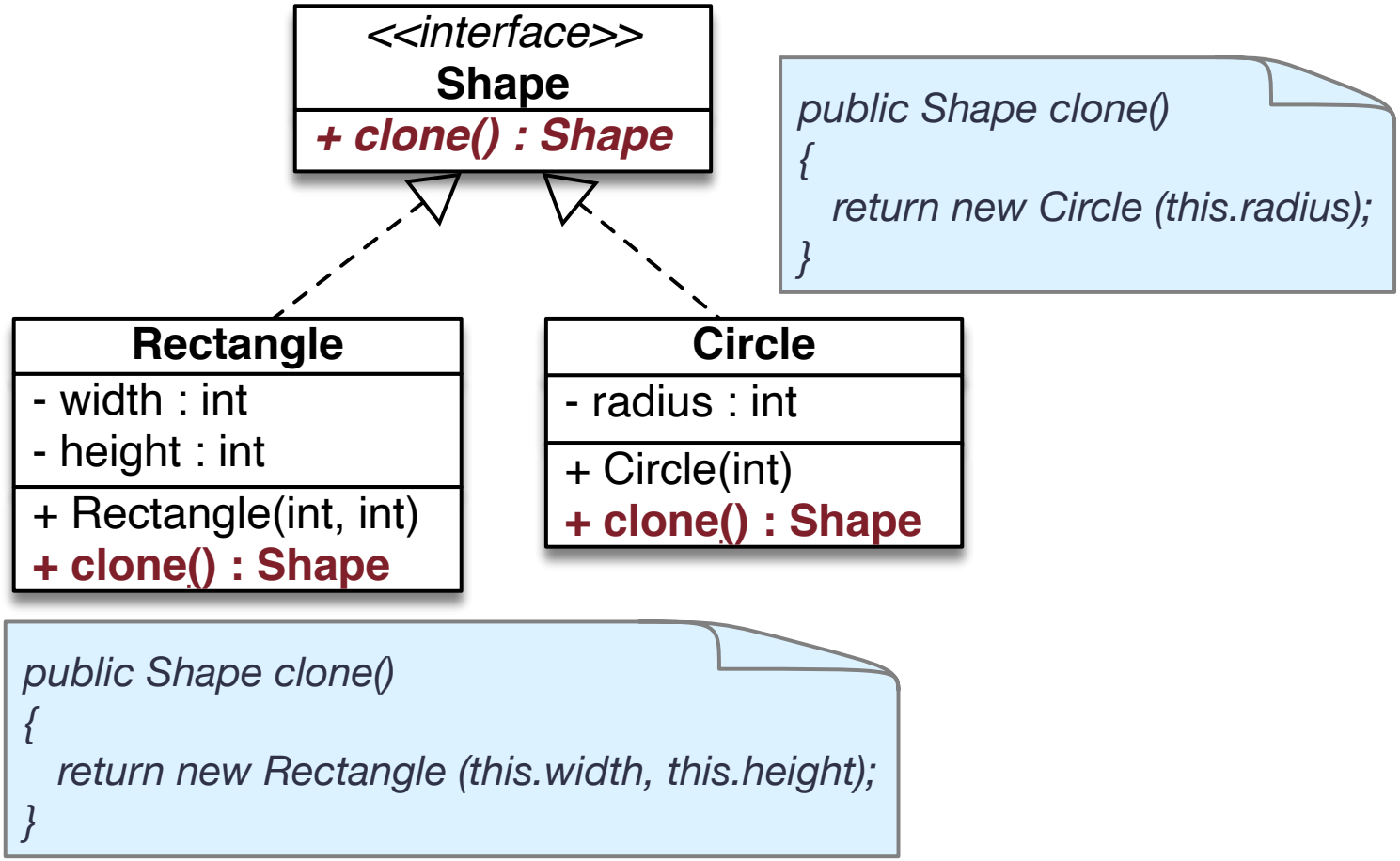
Prototype, exemple

- Exemple :



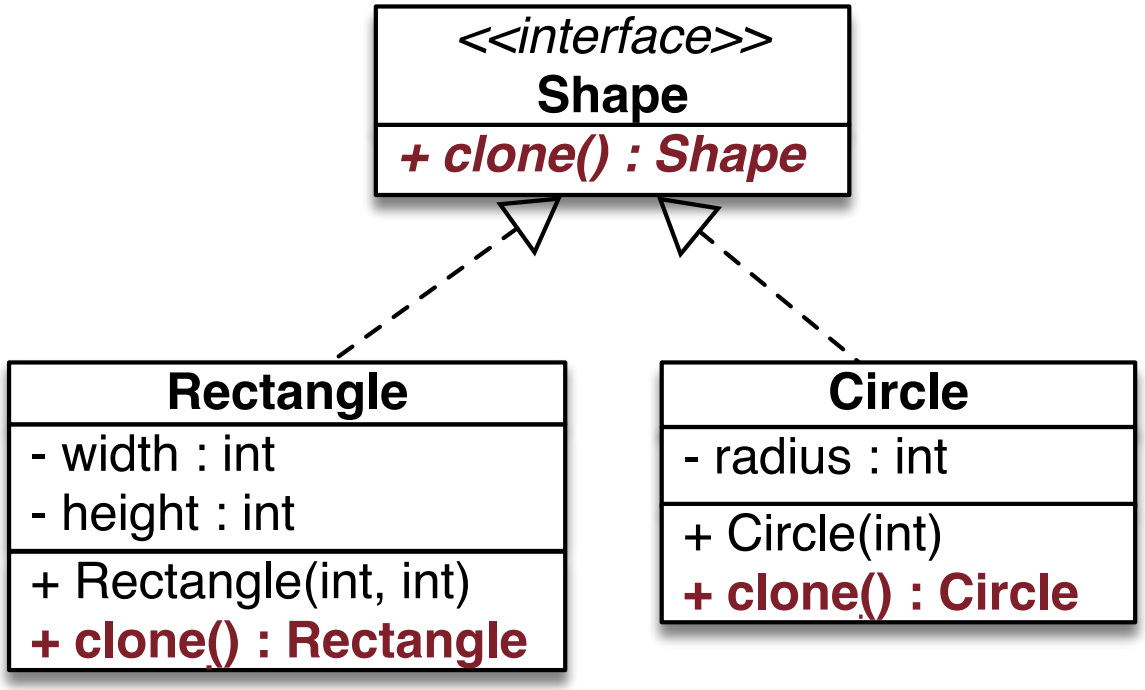
Prototype, exemple

- Solution :



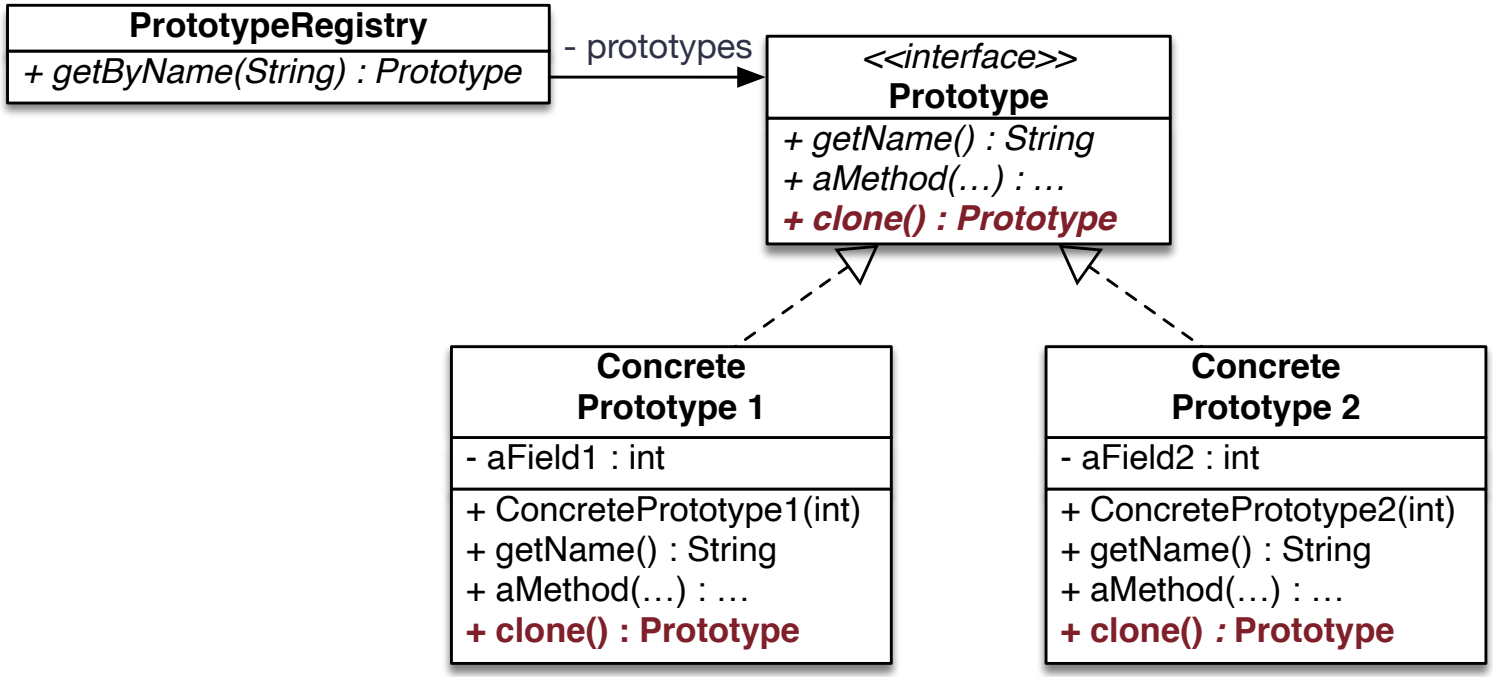
Prototype, exemple

- Remarque : il est possible (en Java) de redéfinir des méthodes en spécifiant un type de retour covariant (sous-type, type compatible)
  - Cela évite les transtypages (*cast*)



Prototype, extension

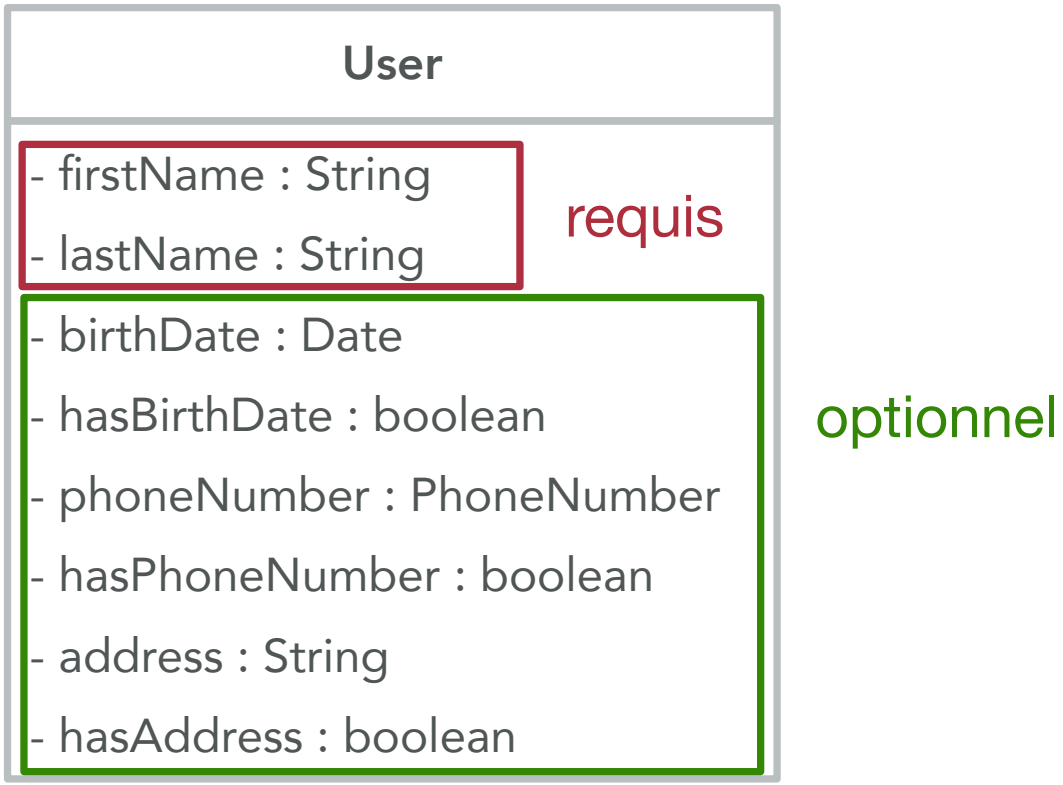
- **Dictionnaire de prototypes**
  - Les prototypes (ensemble d'objets compatibles entre eux mais de types réels ou états différents) sont ajoutés en exemplaire unique dans le dictionnaire
  - Le dictionnaire retourne un duplicata à chaque appel



IUT Valence Université Grenoble Alpes

Builder, le problème

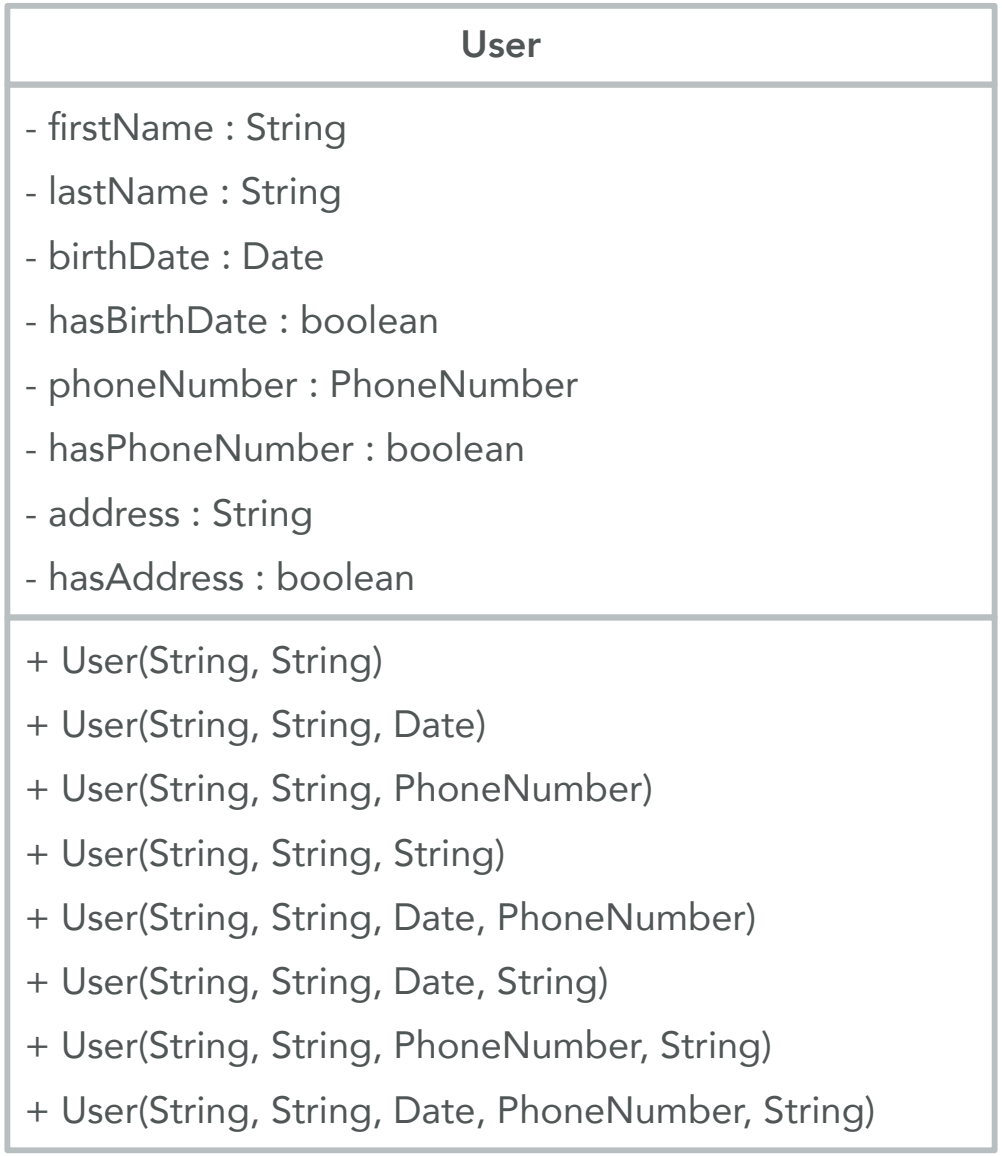
- Comment construire efficacement un objet dont les propriétés sont à *géométrie variable*



- Exemple inspiré de
  - http ://www.javacodegeeks.com/2013/01/the-builder-pattern-in-practice.html
- 8 constructions possibles d'un même objet `User`

IUT Valence Université Grenoble Alpes

Builder, le problème



- Des méthodes fabriques aideraient à s'y retrouver mais bon ...

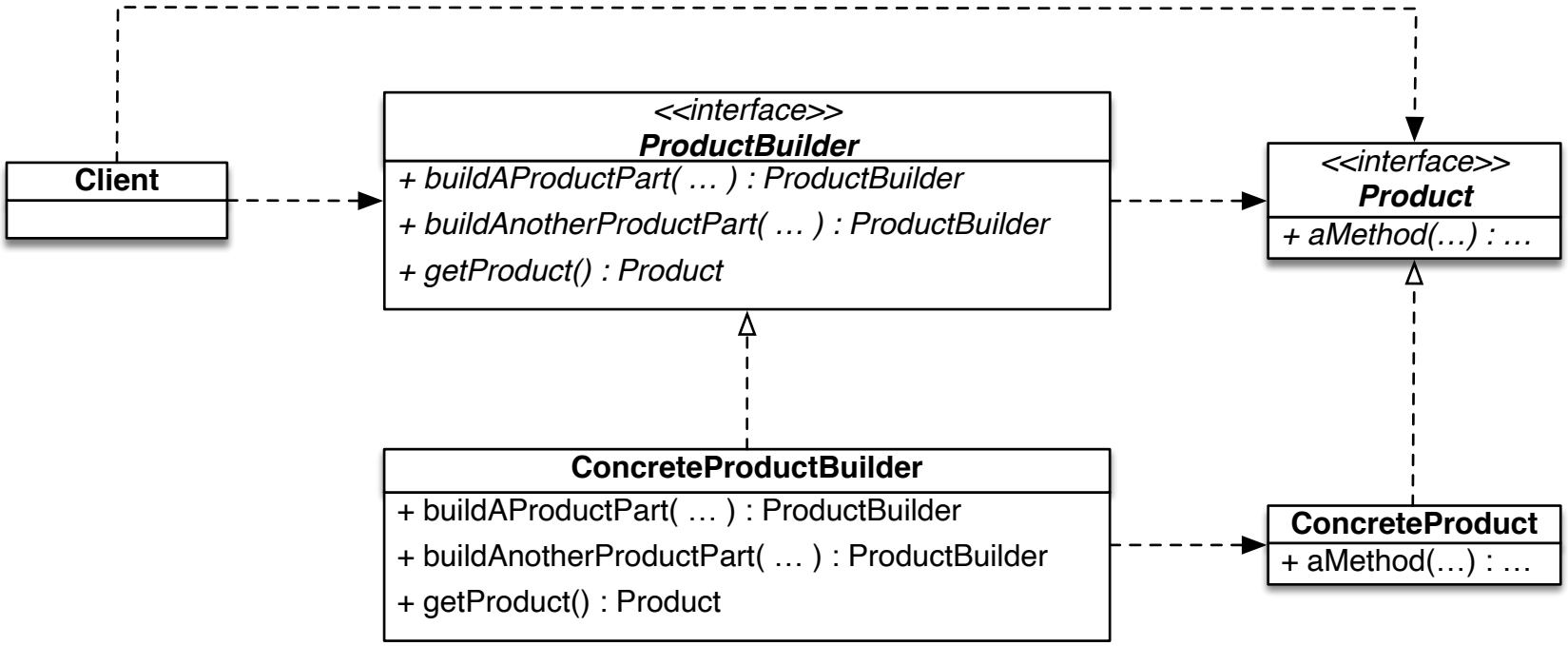
IUT Valence Université Grenoble Alpes

Builder, motivation

- **Créer des objets étape par étape** lorsque leurs structures sont complexes et varient d'un objet à l'autre
  - En masquant au client la multiplicité et la complexité des constructeurs
  - En fournissant au client :
    - des **méthodes de construction incrémentale** permettant de compléter au fur et à mesure l'état initial de l'objet à créer
    - une **méthode permettant d'obtenir l'instance** une fois l'état voulu obtenu
  - En offrant au concepteur la **possibilité d'effectuer des contrôles** sur l'absence ou la redondance d'éléments obligatoires, ...

IUT Valence Université Grenoble Alpes

Builder, définition

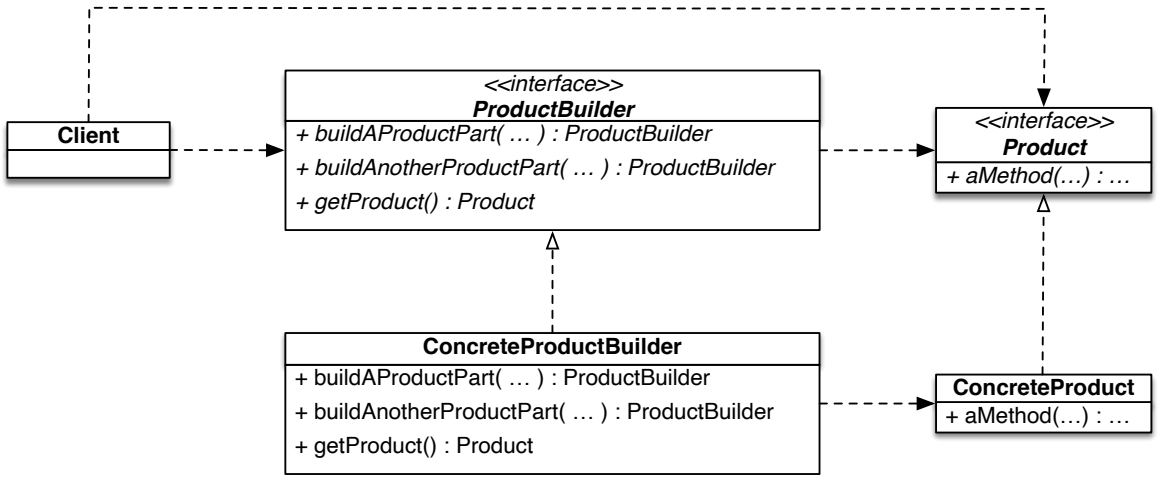


- Méthodes de construction incrémentale
  → `buildAProductPart`, `buildAnotherProductPart`
- Méthode d'obtention du produit
  → `getProduct`

IUT Valence Université Grenoble Alpes

Builder, définition

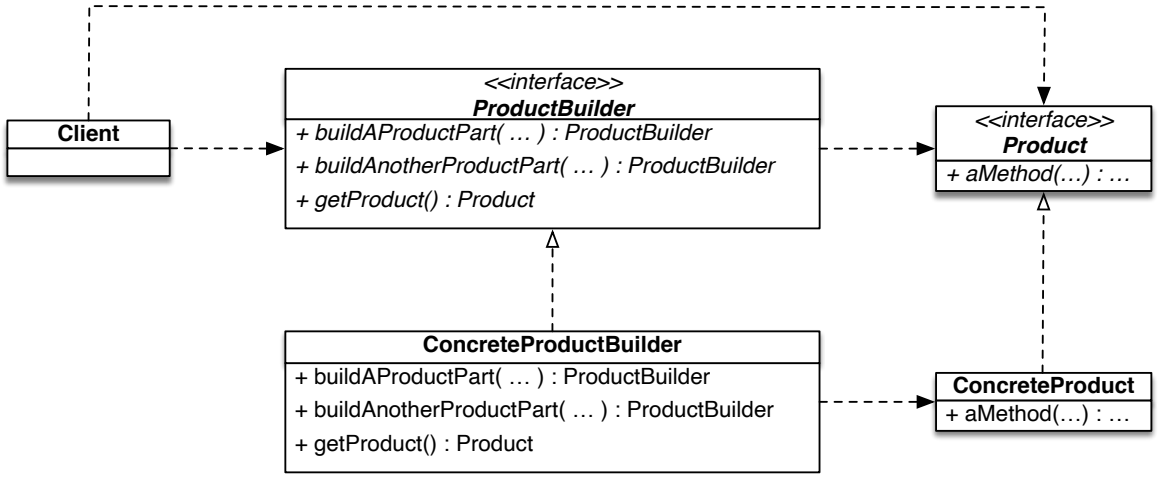
- L'instance à créer est amenée à l'état voulu par une séquence d'appels aux méthodes de construction incrémentale
- Chaque méthode peut être **appelée potentiellement plusieurs fois**, dans **n'importe quel ordre**, avec une sémantique :
  - ajouter une valeur à une propriété de type collection → `add...`
  - renseigner ou écraser la valeur d'une propriété unique → `set...`



IUT Valence Université Grenoble Alpes

Builder, définition

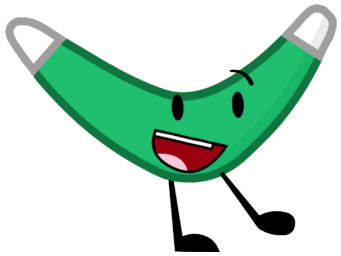
- La méthode d'obtention du produit retourne une **nouvelle instance à chaque appel**
  - Entre 2 appels, liberté (choix de conception) de repartir de l'état de création précédent ou de zéro



- N.B. : la décomposition abstrait / concret n'est pas toujours requise, tout dépend du contexte

IUT Valence Université Grenoble Alpes

Builder, définition (suite)



- La référence du *Builder* est retournée en général à chaque appel d'une méthode de construction incrémentale
  - Cela permet de **rendre plus lisible le processus de création de l'objet**, en proposant une notation élégante

```
Product product = new ProductBuilder()
                      .buildProductPart(...)
                      .buildAnotherProductPart(...)
                      .getProduct();
```

- N.B. : ici `Product` et `ProductBuilder` sont des classes et non des types abstraits

IUT Valence Université Grenoble Alpes

Builder, exemple

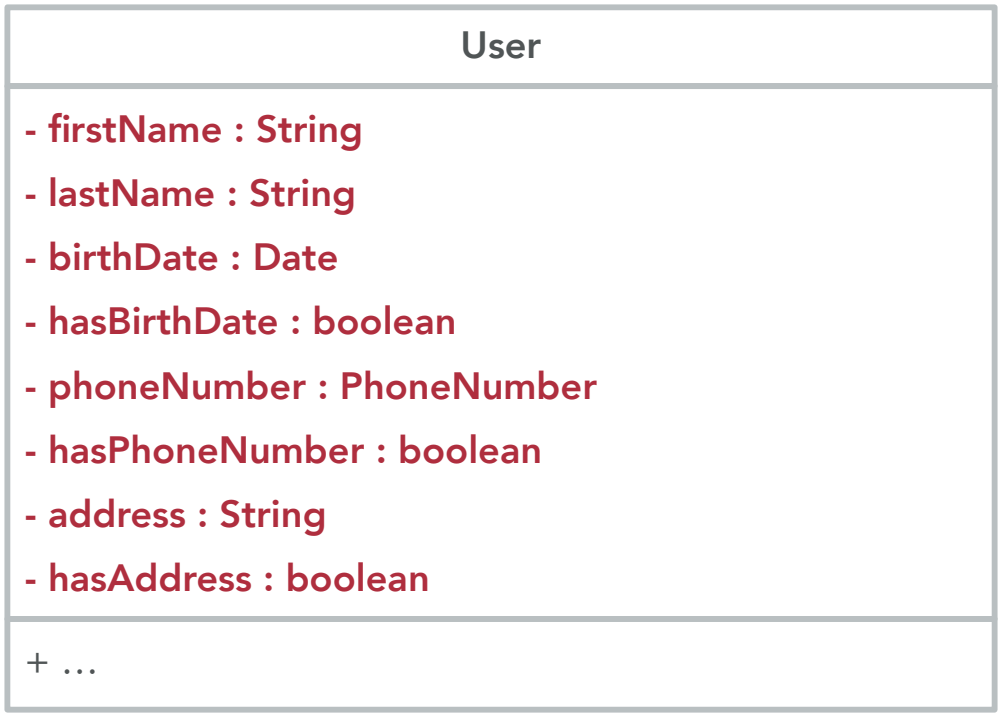
User
- firstName : String - lastName : String - birthDate : Date - hasBirthDate : boolean - phoneNumber : PhoneNumber - hasPhoneNumber : boolean - address : String - hasAddress : boolean
+ User(String, String) + User(String, String, Date) + User(String, String, PhoneNumber) + User(String, String, String) + User(String, String, Date, PhoneNumber) + User(String, String, Date, String) + User(String, String, PhoneNumber, String) + User(String, String, Date, PhoneNumber, String)

- Méthodologie :
  1. Identifier les éléments de structure à initialiser
  2. En déduire les signatures des méthodes de construction incrémentale
  3. Implémenter ces méthodes
  4. Implémenter la méthode d'obtention du produit
  5. Identifier les contrôles à mettre en place
  6. Implémenter les contrôles

IUT A Valence Université Grenoble Alpes

Builder, exemple

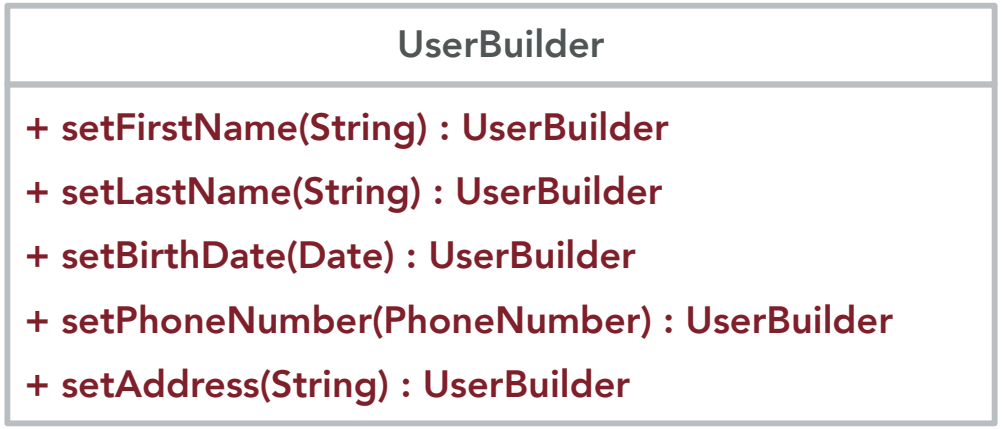
1. Identifier les éléments de structure à initialiser



IUT Valence Université Grenoble Alpes

Builder, exemple

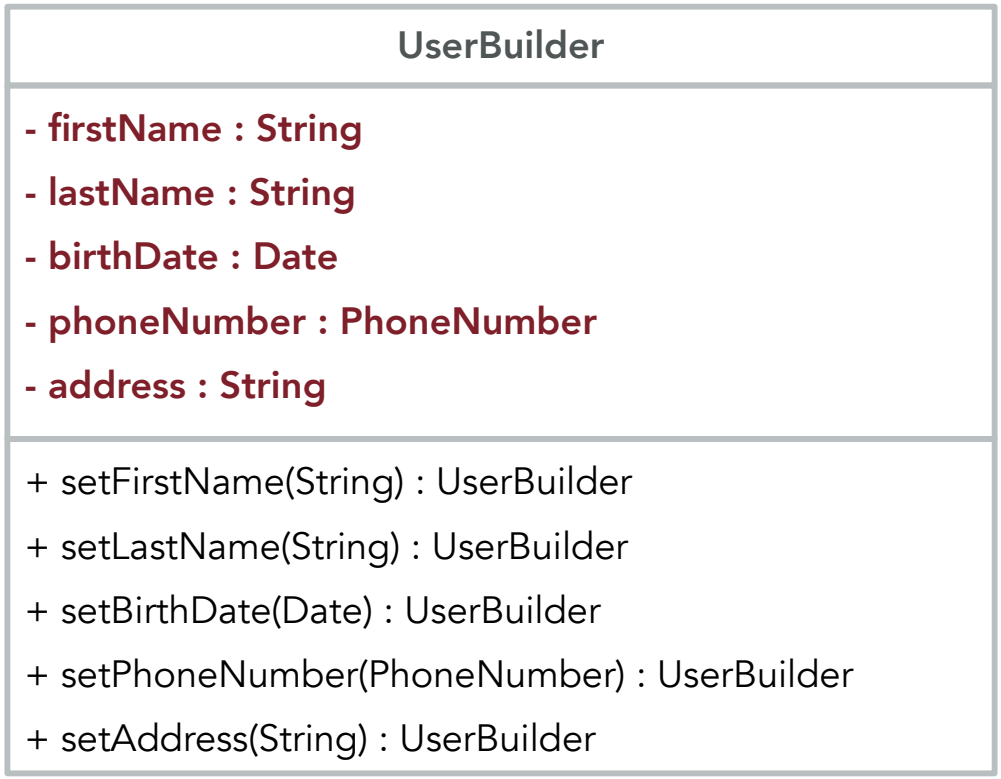
2. En déduire les signatures des méthodes de construction incrémentale



IUT Valence Université Grenoble Alpes

Builder, exemple

3. Implémenter ces méthodes
   - Ici, il n'existe pas de constructeur sans paramètre et de *setters*, la construction de l'objet n'est possible que lorsque le client le demande
   - Il est donc nécessaire de **mémoriser les paramètres du futur appel au constructeur** (correspondants aux éléments fournis)



IUT Valence Université Grenoble Alpes

Builder, exemple

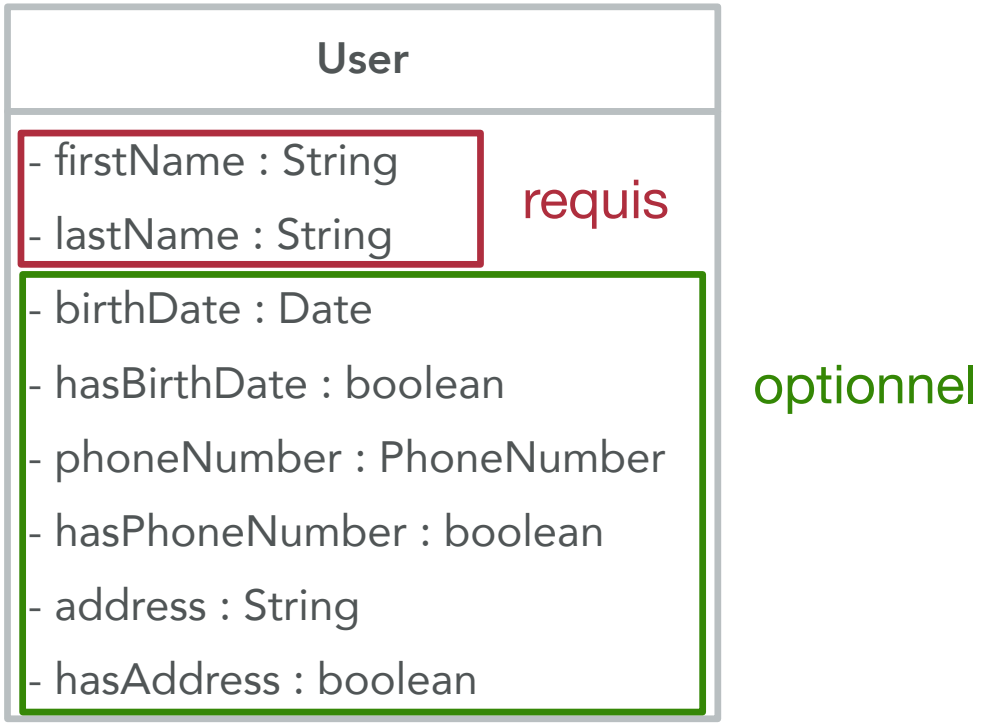
4. Implémenter la méthode d'obtention du produit
   - N.B. : mémorisation des propriétés fournies/non fournies

UserBuilder
- firstName : String **- isFirstNameSet : boolean** - lastName : String **- isLastNameSet : boolean** - birthDate : Date **- isBirthDateSet : boolean** - phoneNumber : PhoneNumber **- isPhoneNumberSet : boolean** - address : String **- isAddressSet : boolean**
+ setFirstName(String) : UserBuilder + setLastName(String) : UserBuilder + setBirthDate(Date) : UserBuilder + setPhoneNumber(PhoneNumber) : UserBuilder + setAddress(String) : UserBuilder **+ getUser() : User**

IUT Valence Université Grenoble Alpes

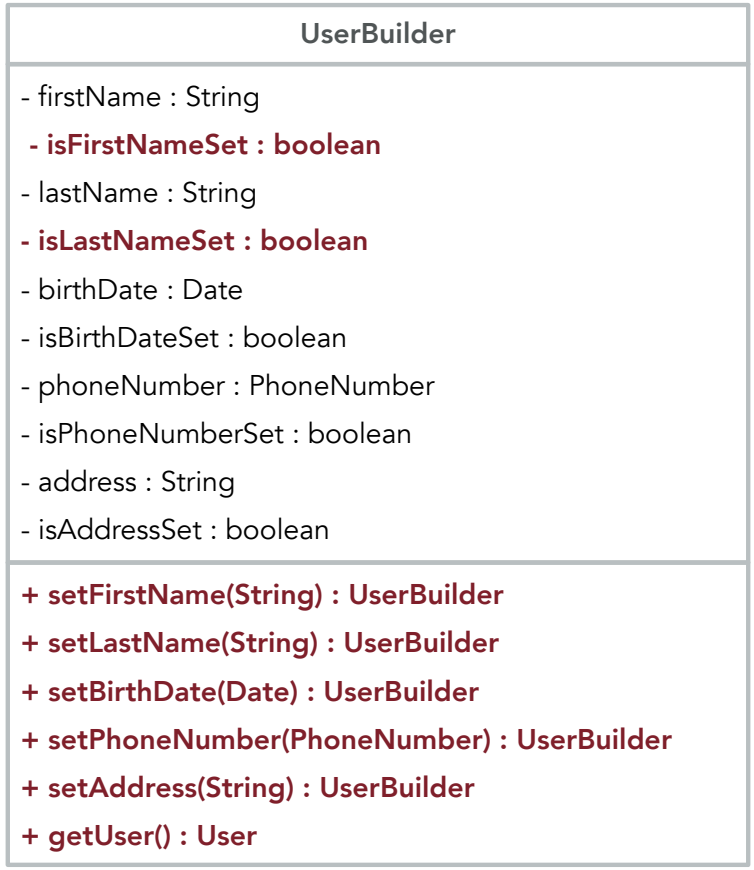
Builder, exemple

5. Identifier les contrôles à mettre en place
   - N.B. : ici, on ne cherchera pas à vérifier que les noms/prénoms/adresses sont cohérents
   - Le caractère requis/optionnel sera contrôlé
   - Les références seront testées à `null`, les chaines de caractères seront testées à vide



Builder, exemple

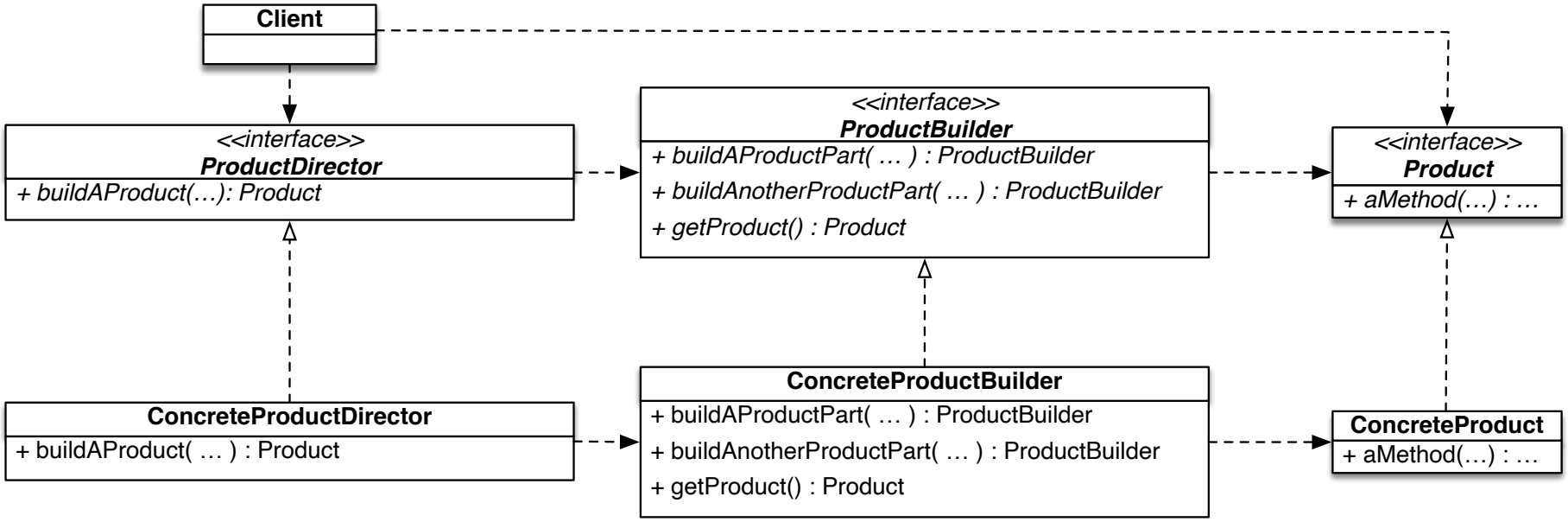
6. Implémenter ces contrôles
   - N.B. : ici, les contrôles s'opèrent à l'appel de `getUser` et des méthodes de construction incrémentales, avec soulèvement d'exception ou retour de la valeur `null`



IUT Valence Université Grenoble Alpes

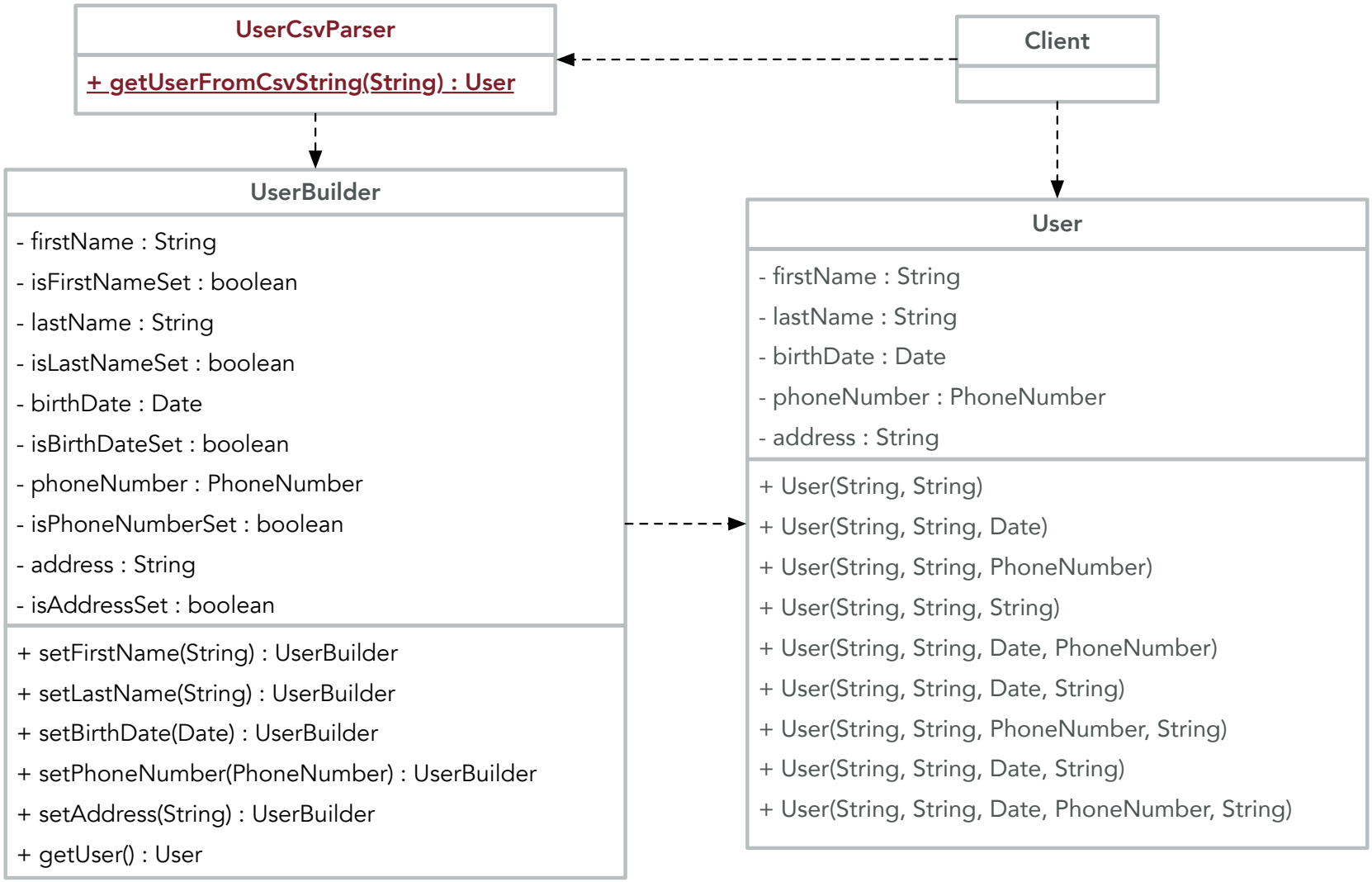
Builder, définition (fin)

- Dans un cas plus englobant, le *builder* peut-être masqué au client, celui-ci utilisant un **mécanisme de plus haut niveau** appelé *director* qui pilote la construction incrémentale



Builder, exemple (fin)

- Création d'un utilisateur à partir d'une représentation texte CSV



Fin !

IUT Valence
Université Grenoble Alpes