

### Cours 6

- ♦ Intention de construction : Builder
- ♦ Intention de responsabilité : Proxy
- ♦ Intention d'extension : Visitor

## Intention de construction

- ♦ Construction ordinaire
  - ♦ Constructeur Java
  - ♦ Aspect important : collaboration que l'on peut orchestrer parmi les constructeurs
- ♦ Collaboration de superclasse
  - ♦ Chainage des constructeurs
- ♦ Collaboration dans une classe
  - ♦ Interaction des constructeurs d'une classe

## Classification des patterns selon leur intention

Intention	Patterns
Interfaces	<a href="#">Adapter</a> , Facade, <a href="#">Composite</a> , <a href="#">Bridge</a>
Responsability	<a href="#">Singleton</a> , <a href="#">Observer</a> , Mediator, <a href="#">Proxy</a> , Chain of Responsibility, Flyweight
Construction	<a href="#">Builder</a> , <a href="#">Factory Method</a> , <a href="#">Abstract Factory</a> , Prototype, memento
Operations	<a href="#">Template Method</a> , <a href="#">State</a> , <a href="#">Strategy</a> , Command, Interpreter
Extensions	Decorator, Iterator, Visitor

## Le patron Builder

### Intention

Séparer la construction d'un objet complexe de sa représentation afin qu'un même processus de création puisse créer différentes représentations

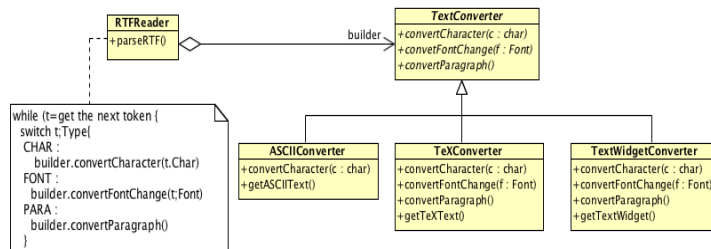
### Motivation

Un lecteur pour le format RTF doit être capable de convertir le RTF en de nombreux formats de texte : ASCII, Text, TextWidget.

Le nombre de conversion possible est sans fin

Solution : configurer le lecteur RTF avec un objet TextConverter qui réalisera les conversions. Les sous-classes de TextConverter sont spécialisées dans les différentes conversions et formats

## Exemple de motivation

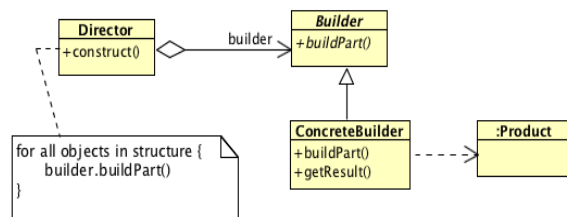


Quand le RTFReader reconnaît le token, il fait une requête au TextConverter pour convertir le token.  
Les sousclasses de TextConverter sont spécialisées pour les différentes Conversion et formats

## Quand utiliser le patron Builder

- ♦ Quand une classe a une structure interne complexe, par exemple avec un ensemble variable d'objets reliés.
- ♦ Quand l'algorithme, pour créer un objet complexe, doit être indépendant des parties qui forment l'objet et comment elles sont assemblées.
- ♦ Quand une classe a des attributs qui dépendent les uns des autres, par exemple un ordre de construction.
- ♦ Quand le processus de construction doit permettre différentes représentations de l'objet qui est construit.

## Structure du pattern Builder



Le builder est responsable de la construction.  
Il coordonne l'assemblage de l'objet product : créer des ressources, stocker des résultats intermédiaires et fournir la structure fonctionnelle pour la création.

## Les rôles des classes

- ♦ **Builder**: l'interface abstraite qui définit les méthodes disponibles pour créer les parties de l'objet Product
- ♦ **ConcreteBuilder** :
  - ♦ Implémente toutes les méthodes requises pour créer un produit réel.
  - ♦ Construit et assemble les parties du produit
  - ♦ Définit et garde trace de la représentation qu'il crée
  - ♦ Fournit une interface pour retrouver le produit
- ♦ **Director** : construit un objet en utilisant une instance de Builder. Il appelle les méthodes de création sur son instance de builder.
- ♦ **Product** : représente l'objet complexe en construction ; il inclut les classes qui définissent les parties constituantes

## Bénéfices et défauts de Builder

### Bénéfice

- ♦ Rend plus facile la gestion du flot général pendant la création de l'objet

### Défauts/conséquences

- ♦ Principal défaut : il y a un couplage fort parmi le builder, son produit et tout autre délégué utilisé

IB- R5.A.08

9/30

## Variantes et Patterns reliés

### ♦ Variantes

- ♦ Builder est soit une classe abstraite, soit une interface
- ♦ Définir plusieurs méthodes de création pour le Builder pour fournir une variété de façons d'initialiser la ressource construite.

### ♦ Patterns reliés :

- ♦ Abstract Factory : cible plus les familles de produits.
- ♦ Composite : le pattern Builder est souvent utilisé pour produire des objets composites

IB- R5.A.08

10/30

## Intention de responsabilité

If your intent to	Apply the pattern
♦ Centralize responsibility in a single instance of a class	=> Singleton
♦ Decouple an object from awareness of which other objects depend on it	=> Observer
♦ Centralize responsibility in a class that oversees how a set of other objects interact	=> Mediator
♦ Let a object act on behalf of another objet	=> Proxy
♦ Allow a request to escalate up a chain of objects until one handles it	=> Chain of Responsibility
♦ Centralize responsibility to shared fine-grained objects	=> Flyweight

IB- R5.A.08

11/30

## Le patron Proxy

### ♦ Intention

- ♦ Fournir un substitut afin d'accéder à un autre objet souvent inaccessible

### ♦ Motivation

- ♦ Si un objet comme une image volumineuse met beaucoup de temps à se charger
- ♦ Si un objet est situé sur une machine distante
- ♦ Si un objet a des droits d'accès spécifiques

IB- R5.A.08

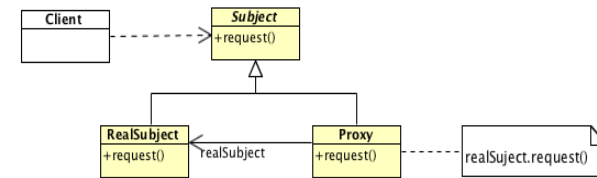
12/30

## Les champs d'application

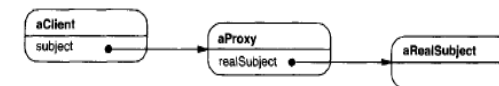
Dès qu'il y a un besoin de référencement sophistiqué ou polyvalent autre qu'une simple pointeur

- ♦ **Remote Proxy** est un représentant d'un objet situé dans un autre espace d'adressage
- ♦ **Virtual Proxy** crée des objets « couteux » à la demande
- ♦ **Access Proxy** pour contrôler l'accès à un objet
- ♦ **Smart reference** effectue un travail supplémentaire lors de l'accès à l'objet
  - ♦ Comptage de références (smart pointers)
  - ♦ Chargement d'objets persistants
  - ♦ Vérification de non verrouillage

## Proxy : structure



Un diagramme d'objets possible d'une structure proxy pendant l'exécution



## Proxy : Participants

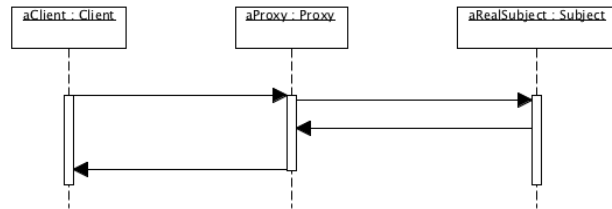
- ♦ **Proxy**
  - ♦ maintient une référence qui permet au Proxy d'accéder à l'objet RealSubject.
  - ♦ fournit une interface identique à celle de Subject, pour pouvoir se substituer au RealSubject.
  - ♦ contrôle les accès au RealSubject et peut être responsable de sa création ou destruction.
- ♦ **Subject**
  - ♦ définit une interface commune pour RealSubject et Proxy. Proxy peut ainsi être utilisé partout où le RealSubject devrait être utilisé.
- ♦ **RealSubject**
  - ♦ définit l'objet réel que le Proxy représente

## Les différents type de proxy : autres responsabilités

- ♦ Les remote proxies sont responsables de l'encodage d'une requête et de ses arguments afin d'envoyer la requête encodée au sujet réel dans un espace d'adresse différent.
- ♦ Les virtual proxies peuvent utiliser un cache pour l'information supplémentaire sur le sujet réel
- ♦ Les proxies de protection vérifient que l'appelant a les permissions d'accès requises pour exécuter la requête.

## Proxy : collaborations

Le **Proxy** retransmet les requêtes au **RealSubject** lorsque c'est nécessaire, selon la forme du proxy



IB- R5.A.08

17/30

## Conséquences

Le pattern Proxy introduit un niveau d'indirection lors de l'accès à un objet. L'indirection supplémentaire a beaucoup d'utilisations selon le type de proxy :

- ♦ Un *remote proxy* peut cacher le fait qu'un objet réside dans un espace d'adressage différent
- ♦ Un *virtual proxy* peut réaliser des optimisations telles que créer un objet sur demande
- ♦ Un *protection proxy* et les *smart references* permettent des tâches où il n'y a pas un objet réel mais simplement la réplique exacte de son sujet

IB- R5.A.08

18/30

## Intention d'extensions(1)

- ♦ Réutilisation comme une alternative à l'extension
  - ♦ Moins de code à maintenir
- ♦ Etendre en sousclassant
  - ♦ A la création d'une classe : garantir que c'est une extension logique et cohérente. Bien nommer les classes
- ♦ Le principe de substitution de Liskov
  - ♦ Une instance d'une classe doit fonctionner comme une instance de sa superclasse
- ♦ Extension par délégation
  - ♦ Quand une classe a des méthodes qui transmettent les appels d'opérations identiques fournies par une autre objet

IB- R5.A.08

19/30

## Intention d'extensions(2)

- | If your intent to  | Apply the pattern  |
|--|--------------------|
| ♦ Allow a client to hook in an operation at a step in an algorithm                   | => Template Method |
| ♦ Let a client outfit your code with an operation to execute in response to an event | => Command         |
| ♦ Attach additional responsibilities to an object dynamically                        | => Decorator       |
| ♦ Provide a way to access a collection of instances of a class that you create       | => Iterator        |
| ♦ Allow for the addition of new operations to a class without changing the class     | => Visitor         |

IB- R5.A.08

20/30

# Le pattern Visitor

Le pattern Visitor construit une opération à réaliser sur les éléments d'un ensemble d'objets. De nouvelles opérations peuvent ainsi être ajoutées sans modifier les classes de ces objets.

## Motivation

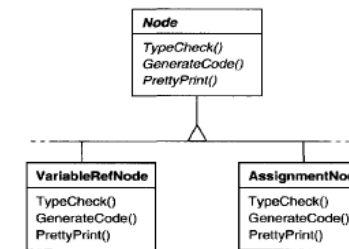
- ♦ Un arbre de syntaxe abstraite (AST) pour un compilateur . Le compilateur utilise l'AST pour les analyses sémantiques statiques. Il doit définir des opérations pour : vérifier les types, optimiser le code, vérifier l'affectation des variables à des valeurs.
- ♦ De plus, les AST seront utilisés pour du pretty-print, restructuration de programme, instrumentation, calcul de métrique ..

IB- R5.A.08

21/30

## Motivation (suite)

La plupart des opérations auront besoin de traiter des nœuds qui représentent de façon différentes les affectations, les variables ou les expressions arithmétiques .  
Ce diagramme montre une partie de la hiérarchie de la classe Noeud.



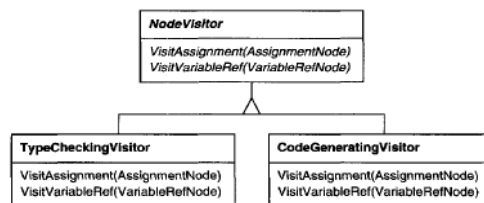
IB- R5.A.08

22/30

## Visitor

Pour faire marcher les visiteurs pour faire plus que la vérification de type, on a besoin d'une classe abstraite parente *NodeVisitor* pour tous les visiteurs d'un AST. *NodeVisitor* doit déclarer une opération pour chaque classe *node*.

Le pattern Visitor encapsule les opérations pour chaque phase de la compilation dans un visitor associé avec cette phase.

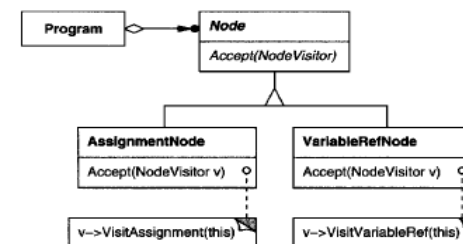


IB- R5.A.08

23/30

## Description du pattern

Avec le pattern Visitor on définit deux hiérarchies de classes : une pour les éléments sur lesquels on opère (hiérarchie Node) et une pour les visiteurs qui définissent les opérations sur les éléments (hiérarchie NodeVisitor). On crée une nouvelle opération en ajoutant une sousclasse à la hiérarchie des visitor



IB- R5.A.08

24/30

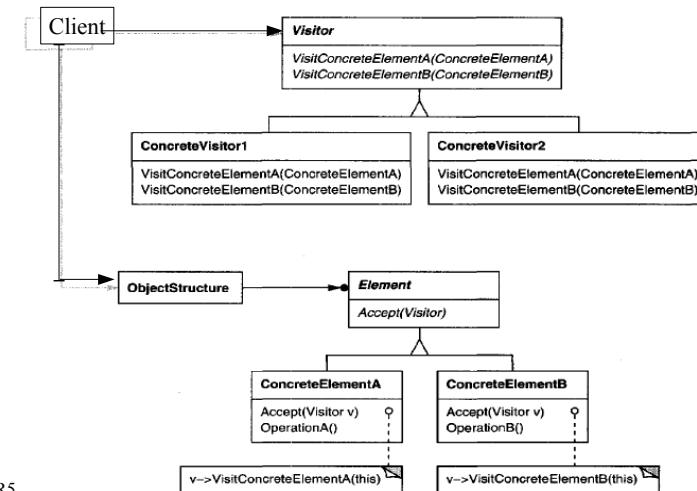
## Quand utiliser Visitor ?

- ♦ Une structure d'objet contient beaucoup de classes d'objets avec des interfaces différentes, et on veut faire des opérations sur ces objets qui dépendent de leur classes concrètes.
- ♦ Beaucoup d'opérations distinctes et non reliées ont besoin d'être réalisées sur des objets dans une structure et on veut éviter de polluer leurs classes avec des opérations. Visitor permet de garder les opérations ensemble en les définissant dans une classe.
- ♦ Les classes définissant la structure objet change rarement, mais on veut définir souvent de nouvelles opérations dans la structure. Changer les classes de la structure demande de redéfinir l'interface de tous les visitors, ce qui coûte cher. Dans ce cas il est préférable de définir les opérations dans ces classes.

IB- R5.A.08

25/30

## Structure de la solution de Visitor



IB- R5

26/30

## Participants

### Visitor (NodeVisitor)

- ♦ Déclare une opération `Visit` pour chaque classe de `ConcreteElement`. Le nom de l'opération et la signature identifient la classe qui envoie la requête `Visit` au visitor.

### ConcreteVisitor (TypeCheckingVisitor)

- ♦ Implémente chaque opération déclarée par `Visitor`. Chaque opération implémente un fragment de l'algorithme défini par la classe d'objets correspondante dans la structure.
- ♦ Fournit le contexte pour l'algorithme et stocke l'état local qui souvent accumule les résultats durant la traversée de la structure.

IB- R5.A.08

27/30

### Element (Node)

- ♦ Définit une opération `Accept` qui prend un visiteur en argument

### ConcreteElement (Assignment Node, VariableRefNode)

- ♦ implémente une opération `Accept` qui prend un visiteur en argument

### ObjectStructure (Program)

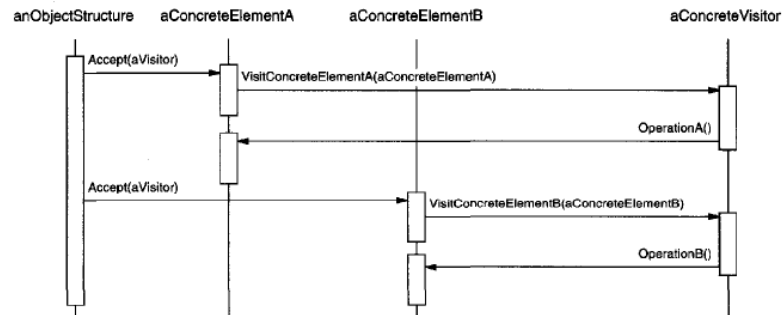
- ♦ Peut énumérer ses éléments
- ♦ Peut fournir une interface de haut-niveau pour permettre au visitor de visiter ses éléments
- ♦ Peut être soit un composite (voir pattern Composite) ou une collection telle qu'une liste ou un ensemble

IB- R5.A.08

28/30

## Collaborations

Le client doit créer le ConcreteVisitor  
Quand un élément est visité, il appelle l'opération Visitor qui correspond à sa classe  
Il se passe lui-même en paramètre pour laisser le visitor accéder à son état.



Collaboration entre une structure d'objet, un visitor et deux éléments

## Conséquences

- ♦ Ajout de nouvelles opérations : facile
- ♦ Ajout de nouvelles classes ConcreteElement : difficile
- ♦ Visitor traverse des structures où les éléments sont de types complètement différents. Ce qui n'est pas possible avec un itérateur.
- ♦ Accumulation d'états dans le visiteur plutôt que dans des arguments
- ♦ Suppose que l'interface de `ConcreteElement` est assez riche pour que le visiteur fasse son travail. Souvent la conséquence brise l'encapsulation car le pattern nous force à fournir des opérations publiques qui accèdent à l'état interne d'un élément.