



Génie Logiciel : Conception Patrons de Conception (chevalier)

Images: Pixabay

Sébastien Mosser
INF-5153, Hiver 2019, Cours #8

UQÀM

1

Menu		Objectif		
		Création	Structure	Comportement
classe	Factory	Adapter	Interpreteur Template Method	
	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor	

3

Crédits

UNIVERSITÉ CÔTE D'AZUR



Philippe Collet

Université Côte d'Azur
Laboratoire I3S, SPARKS

“Prof d’UML” de père en fils depuis 1999

https://www.i3s.unice.fr/Philippe_Collet

2

Catalogue des patrons de conception du GoF

- Auteur : Sébastien Mosser (UQAM)
- Contributeurs: Mireille Blay-Fornarino (UCA), Philippe Collet (UCA)
- Version : 2019.03
- Intégration continue :

Ce dépôt est un support au cours INF-5153 de l'Université du Québec à Montréal. Il recense des implémentations en Java des patrons de conception vus dans ce cours. Il est créé en collaboration avec l'Université Côte d'Azur (IUT & Polytech).

<https://github.com/ace-lectures/pattern-repository>

4

Patrons disponibles

Création

Patron de conception	Description	Code d'exemple	Diapositives
Abstract Factory	✓	✗	✗
Builder	✓	✓	✗
Factory	✓	✗	✗
Prototype	✓	✓	✗
Singleton	✓	✓	✗

Structure

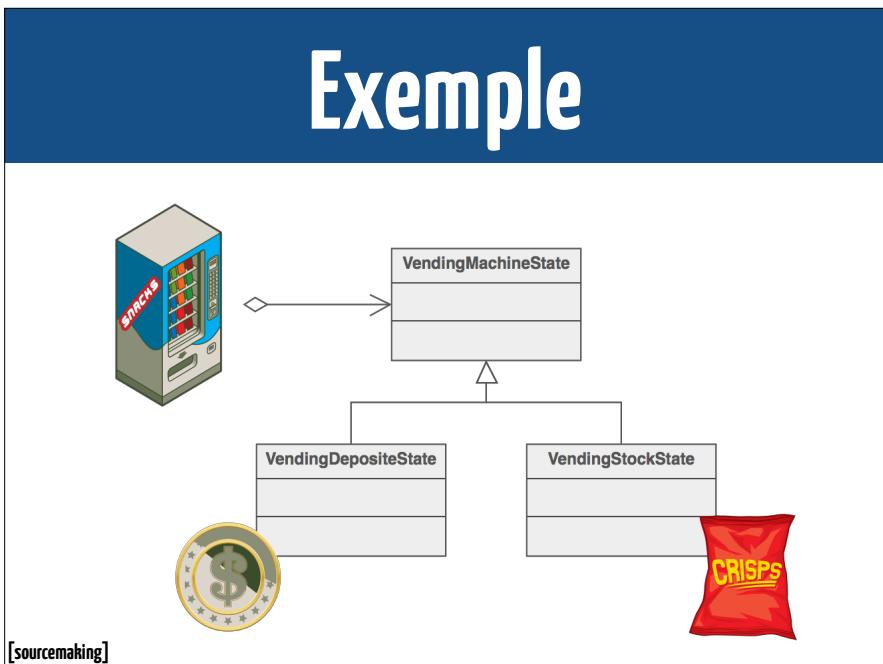
Patron de conception	Description	Code d'exemple	Diapositives
Adapter	✓	✗	✗
Composite	✓	✓	✗
Decorator	✓	✓	✗
Facade	✓	✗	✗
Proxy	✓	✓	✗

Comportement

Patron de conception	Diapositives	Code d'exemple	Description
Command	✗	✗	✗
Observer	✗	✗	✗
State	✗	✗	✗
Strategy	✗	✗	✗
Template Method	✗	✗	✗
Visitor	✗	✗	✗

5

State



7

6

Problème

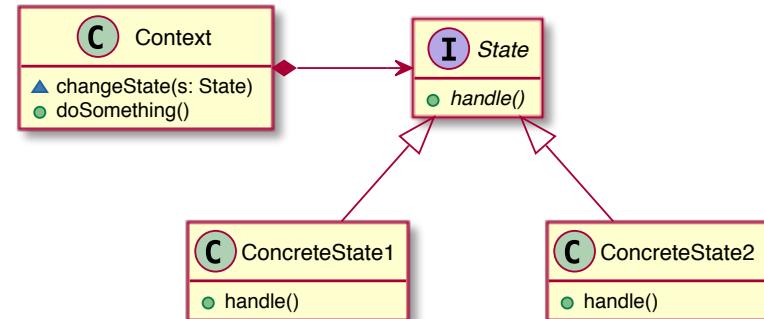
Comment modifier le comportement d'un objet quand son état interne change et obtenir des traitements différents ?

8

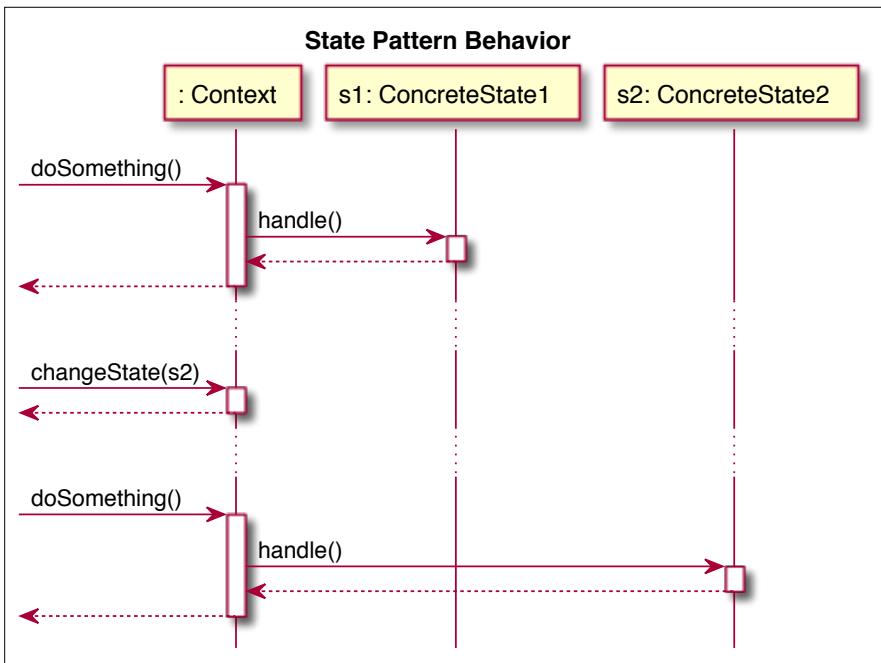
Intention

- Eviter les instructions conditionnelles en masse
- Simplifier l'ajout suppression de nouveaux états
- Donner l'impression que l'objet a été modifié alors que c'est uniquement son état qui a varié

9



10



11

Conséquences

- Séparation des comportements relatifs à chaque état
- Transitions explicite par action sur l'objet
- Transparence pour l'appelant

12

Où le trouver ?

- Gestion des connexion réseaux
- Mise en oeuvre d'une machine à état
- Javax Lifecycle

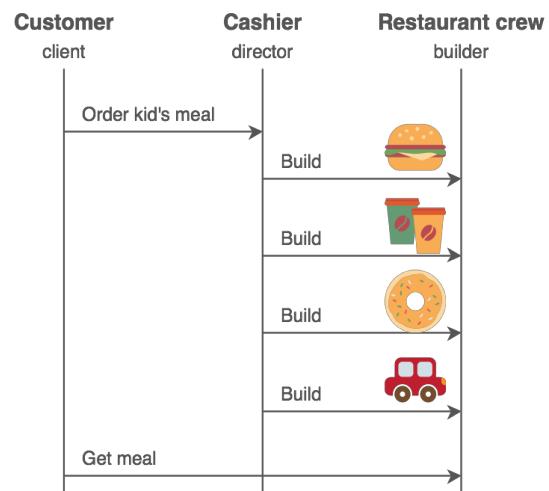
13

Builder



14

Exemple



[sourcemaking]

15

Problème

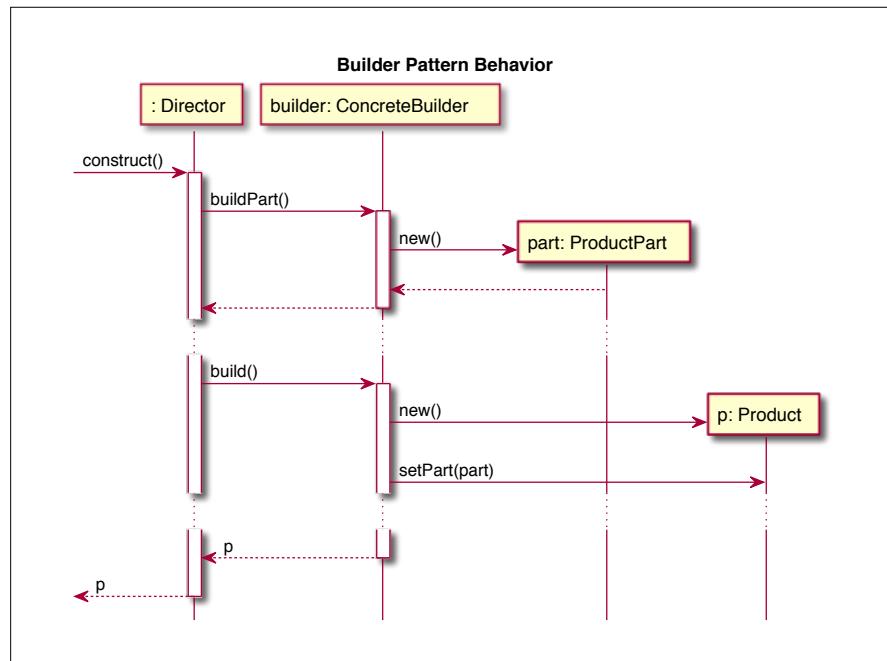
Comment créer par assemblage un objet complexe, indépendamment des parties qui le compose ?

16

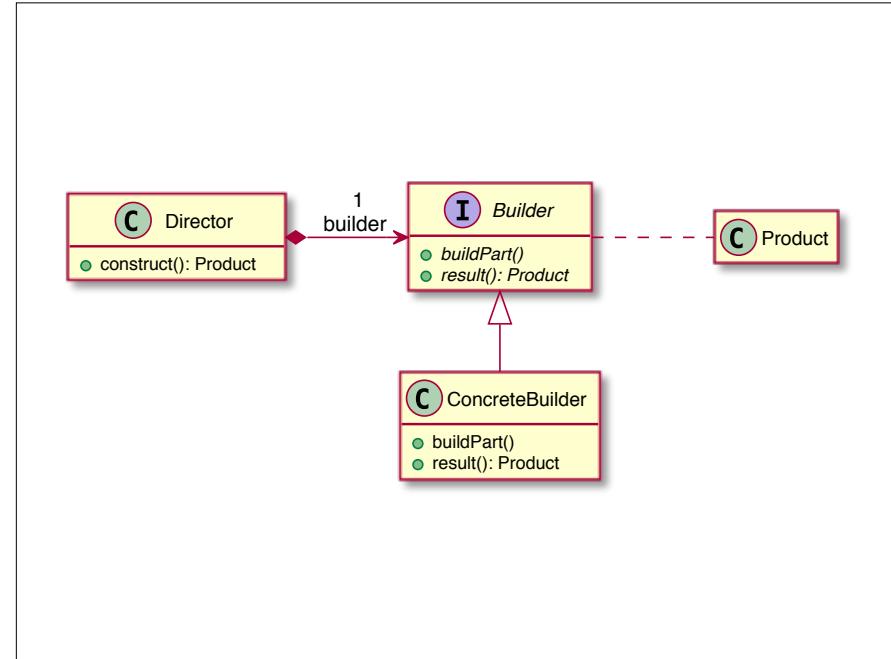
Intention

- Isoler la construction de l'objet de sa représentation
- Contrôler le procédé de construction
- Changer au besoin la représentation interne du produit

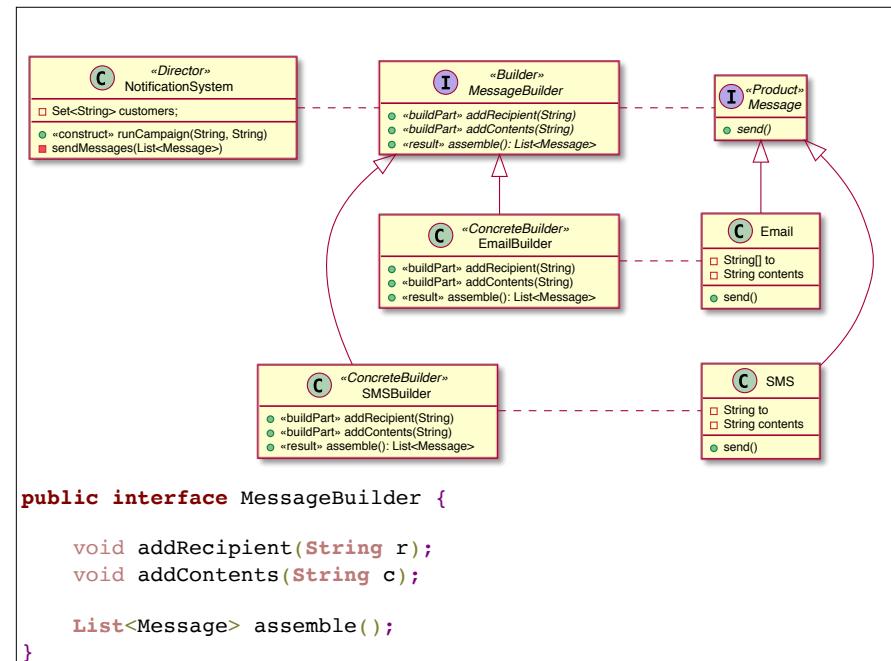
17



19



18



20

```

public class SMSBuilder implements MessageBuilder {
    private Set<String> recipients = new HashSet<>();
    private String contents = null;

    @Override public void addRecipient(String r) { this.recipients.add(r); }

    @Override public void addContents(String c) { this.contents = c; }

    @Override
    public List<Message> assemble() {
        if (recipients.isEmpty() || contents == null)
            throw new IllegalStateException();
        return recipients.stream()
            .map(r -> new SMS(r, contents))
            .collect(Collectors.toList());
    }
}

```

21

Où le trouver ?

- Dès qu'on parle de "Fluent API"
- Mockito
- EntityBuilder (spring, .Net, ...)
- Dans l'API Java (StringBuilder, StringBuffer, ...)
- Interfaces graphiques (composition de composants)

23

Conséquences

- Définition abstraite du processus de construction
 - Réutilisation du process, mais produits différents
- On peut utiliser du polymorphisme pour construire des produits différents
- Le client n'a pas besoin de connaître le processus de fabrication

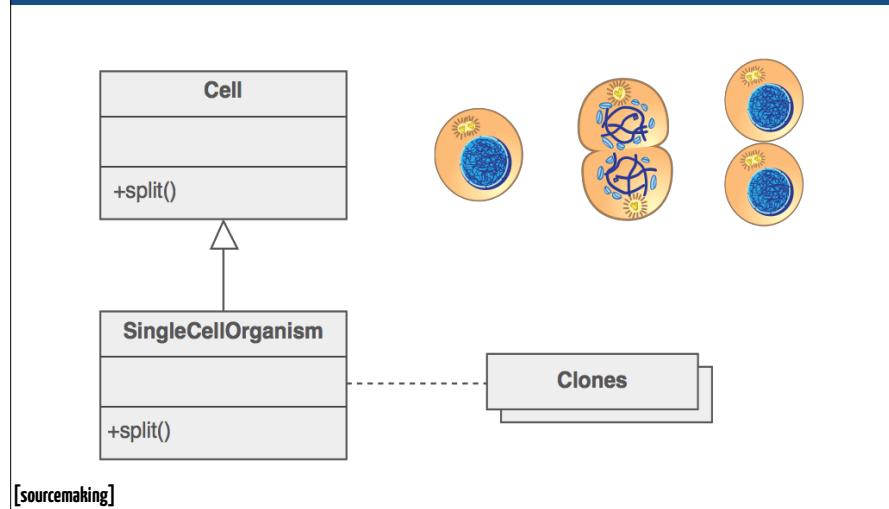
22

Prototype



24

Exemple



25

Intention

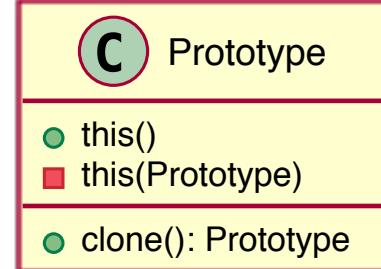
- Spécifier une classe d'objet constructible par clonage
- Permettre de se passer du `new`
- Co-opter une instance comme représentative des suivantes

27

Problème

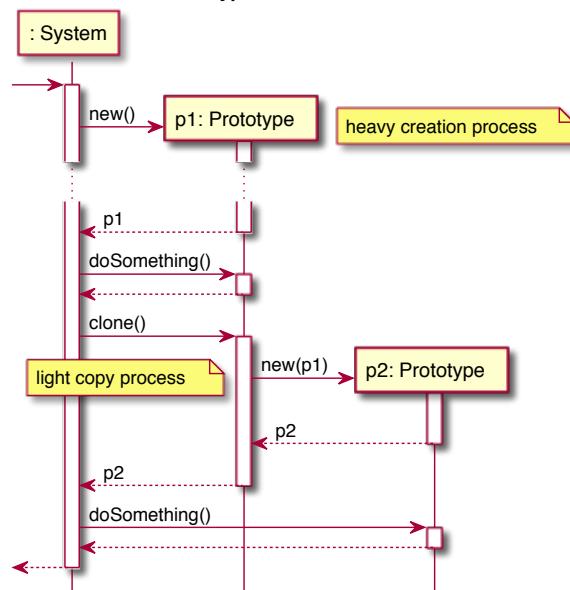
Comment utiliser un objet comme exemple d'instantiation pour les autres, par exemple quand sa création est coûteuse mais sa copie aisée ?

26



28

Prototype Pattern Behavior



29

```

private static Set<Student> usingConstructor(){
    Set<Student> result = new HashSet<>();
    Student bob = new Student("Bob");
    bob.registerTo("INF5151");
    bob.registerTo("INF5153");
    bob.registerTo("OPT6000");
    result.add(bob);

    Student alice = new Student("Alice");
    alice.registerTo("INF5151");
    alice.registerTo("INF5153");
    alice.registerTo("OPT3000");
    result.add(alice);

    Student eve = new Student("Eve");
    eve.registerTo("INF5151");
    eve.registerTo("INF5153");
    eve.registerTo("OPT8000");
    result.add(eve);

    return result;
}

private static Set<Student> usingClones(){
    Set<Student> result = new HashSet<>();
    Student bob = new Student("Bob");
    bob.registerTo("INF5151");
    bob.registerTo("INF5153");
    bob.registerTo("OPT6000");
    result.add(bob);

    Student alice = bob.clone();
    alice.setName("Alice");
    alice.registerTo("INF5151");
    alice.registerTo("INF5153");
    alice.registerTo("OPT3000");
    result.add(alice);

    Student eve = bob.clone();
    eve.setName("Eve");
    eve.registerTo("OPT6000");
    result.add(eve);

    bob.registerTo("OPT3000");
    result.add(bob);

    alice.registerTo("OPT3000");
    result.add(alice);

    eve.registerTo("OPT8000");
    result.add(eve);

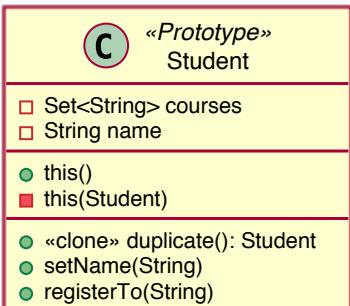
    return result;
}
    
```

30

Conséquences

- On peut factoriser des opérations coûteuses dans le prototype
- Le clonage est une opération difficile :
 - Clonage superficiel ?
 - Clonage en profondeur ?
- Modèle de programmation peu habituel

} Collections / refs



```

lucifer:prototype mosser$ mvn -q exec:java
# Classical instantiation
Registering [Bob] to [INF5151]
=> disconnect ... validate ... register ... disconnect ...
Registering [Bob] to [INF5153]
=> disconnect ... validate ... register ... disconnect ...
Registering [Bob] to [OPT6000]
=> disconnect ... validate ... register ... disconnect ...
Registering [Alice] to [INF5151]
=> disconnect ... validate ... register ... disconnect ...
Registering [Alice] to [INF5153]
=> disconnect ... validate ... register ... disconnect ...
Registering [Alice] to [OPT3000]
=> disconnect ... validate ... register ... disconnect ...
Registering [Eve] to [INF5151]
=> disconnect ... validate ... register ... disconnect ...
Registering [Eve] to [INF5153]
=> disconnect ... validate ... register ... disconnect ...
Registering [Eve] to [OPT8000]
=> disconnect ... validate ... register ... disconnect ...
->> Time consumed: 551ms

## Resulting student set
Student[name='Alice', courses=[OPT3000, INF5151, INF5153]]
Student[name='Bob', courses=[OPT6000, INF5151, INF5153]]
Student[name='Eve', courses=[OPT8000, INF5151, INF5153]]

# Using clones
Registering [Bob] to [INF5151]
=> disconnect ... validate ... register ... disconnect ...
Registering [Bob] to [INF5153]
=> disconnect ... validate ... register ... disconnect ...
Cloning Student{name='Bob', courses=[INF5151, INF5153]}
CloneStudent{name='Bob', courses=[INF5151, INF5153]}
Registering [Bob] to [OPT6000]
=> disconnect ... validate ... register ... disconnect ...
Registering [Alice] to [OPT3000]
=> disconnect ... validate ... register ... disconnect ...
Registering [Eve] to [OPT8000]
=> disconnect ... validate ... register ... disconnect ...
->> Time consumed: 304ms

## Resulting student set
Student[name='Alice', courses=[OPT3000, INF5151, INF5153]]
Student[name='Bob', courses=[OPT6000, INF5151, INF5153]]
Student[name='Eve', courses=[OPT8000, INF5151, INF5153]]

# Are resulting sets equivalents ?
Equivalence
lucifer:prototype mosser$ 
    
```

31

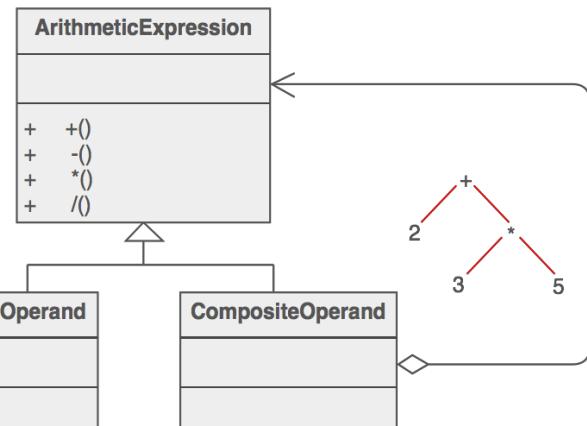
32

Où le trouver ?

- Interface Cloneable de Java
- Modèle objet de Javascript
- Et plus généralement des langages objets à prototype
 - La POO n'inclue pas que les langages à Classe !

33

Exemple



[sourcemaking]

35

Composite



34

Problème

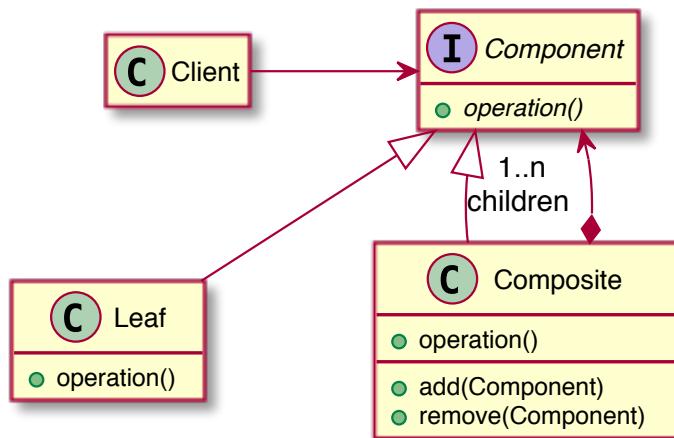
Comment représenter de manière uniforme une hiérarchie d'objets ?

36

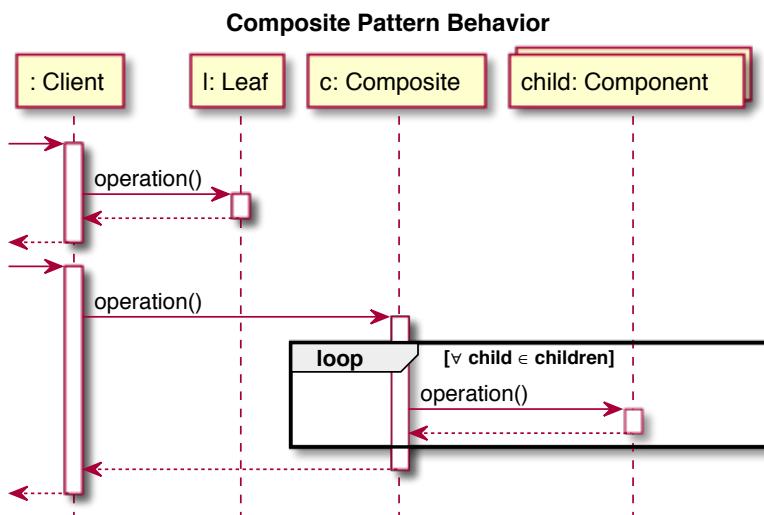
Intention

- Formaliser la structuration arborescente des objets
 - Traiter feuilles et noeuds de manière uniforme
 - Propagation récursive des comportements

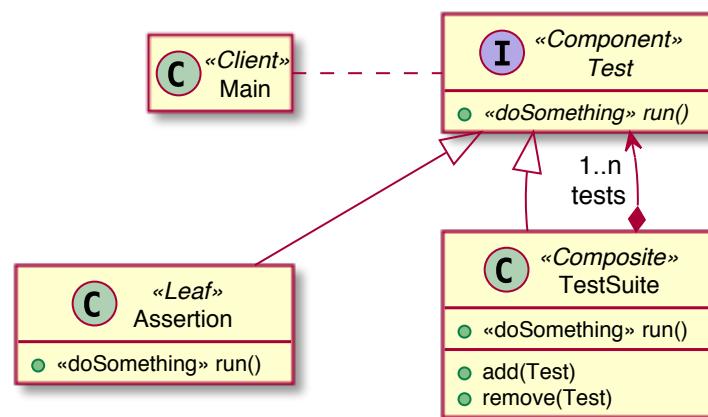
37



38



39



40

```

public static void main(String[] args) {

    TestSuite unit = new TestSuite("Unit tests");
    unit.add(new Assertion("1st unit test"));
    unit.add(new Assertion("2nd unit test"));
    unit.add(new Assertion("3rd unit test"));

    TestSuite accept = new TestSuite("Acceptance tests");
    accept.add(new Assertion("1st scenario"));
    accept.add(new Assertion("2nd scenario"));

    TestSuite ext1 = new TestSuite("Integration with EXT-1");
    ext1.add(new Assertion("Test with external partner #1"));
    TestSuite ext2 = new TestSuite("Integration with EXT-2");
    ext2.add(new Assertion("Test with external partner #2"));
    TestSuite integration = new TestSuite("Integration tests");
    integration.add(ext2);
    integration.add(ext1);

    TestSuite complete = new TestSuite("Complete test suite");
    complete.add(unit);
    complete.add(accept);
    complete.add(integration);

    System.out.println("\n# Running a single assertion");
    (new Assertion("single unit test")).run();

    System.out.println("\n# Running unit tests only");
    unit.run();

    System.out.println("\n# Running the all product suite");
    complete.run();
}

}

```

41

Où le trouver ?

- Dès qu'on fait face à une structure arborescente
 - Système de fichiers, GUI, compilation, documents
- **Il se compose particulièrement bien avec la Commande :**
 - Une macro-commande est un composite de commandes

43

Conséquences

- Structure hiérarchique simple et uniforme
- Facile à étendre pour de nouveaux objets
- Comportement propagable de manière automatique
- Mais ...
 - potentiellement trop rigide
 - Les composants fils sont ils ordonnés ou non ?

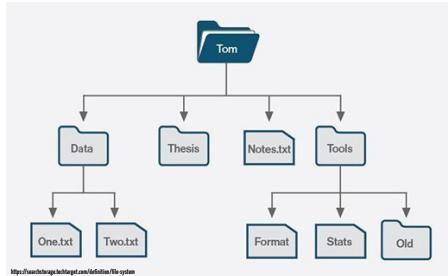
42

Visitor



44

Exemple



Rechercher
un fichier ?

Supprimer
un répertoire ?

Copier un répertoire ?

45

Problème

Comment rendre externe à une hiérarchie d'objets la mise en oeuvre d'un comportement, pour pouvoir changer celui-ci sans avoir à changer les objets ?

46

Intention

- Définir une abstraction pour les opérations à effectuer
- Définir UNE FOIS POUR TOUTES le parcours de la hiérarchie
- “Lancer” une opération sur une hiérarchie,
- Sans avoir à modifier la hiérarchie pour créer de nouvelles opérations

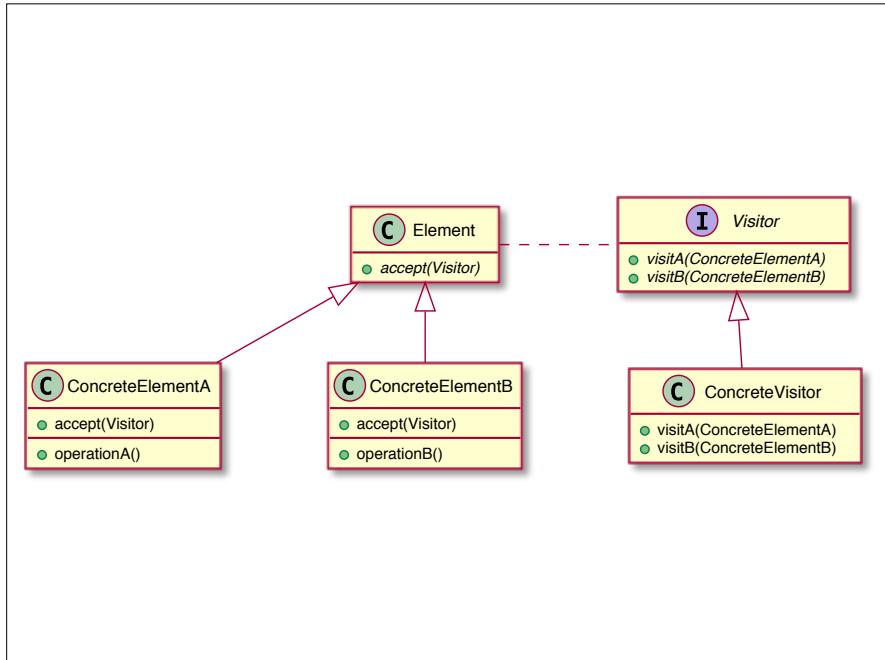
47

“Double Dispatch”

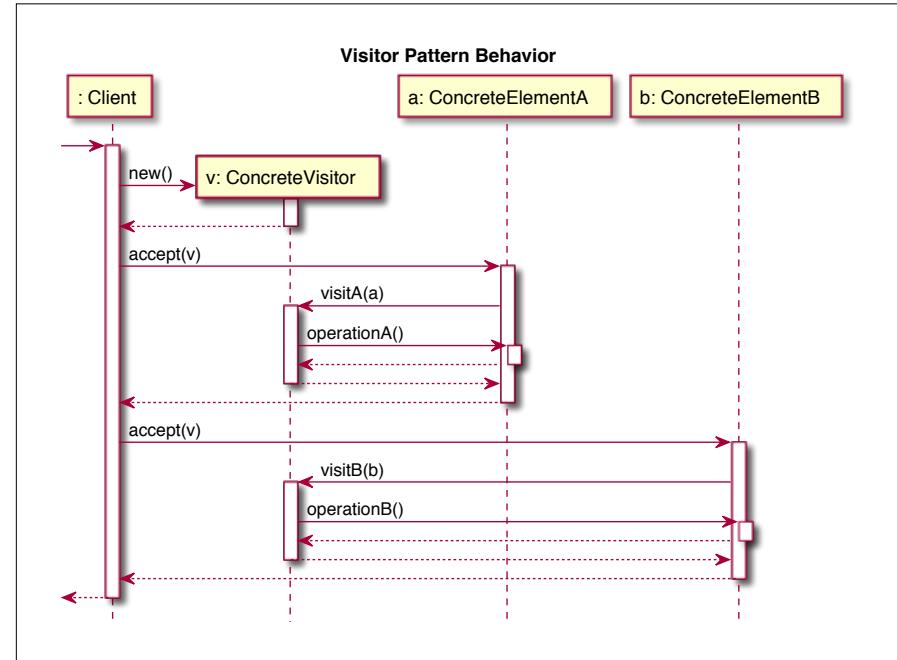
- 1.La hiérarchie “Accepte” l’opération
- 2.Elle délègue immédiatement à l’opération

Un élément accepte, et le visiteur visite.

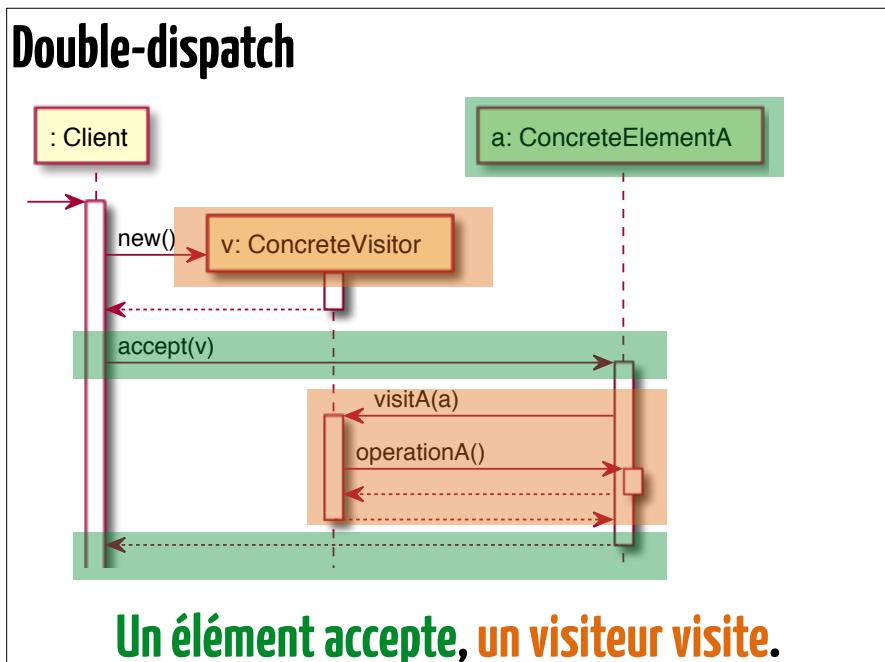
48



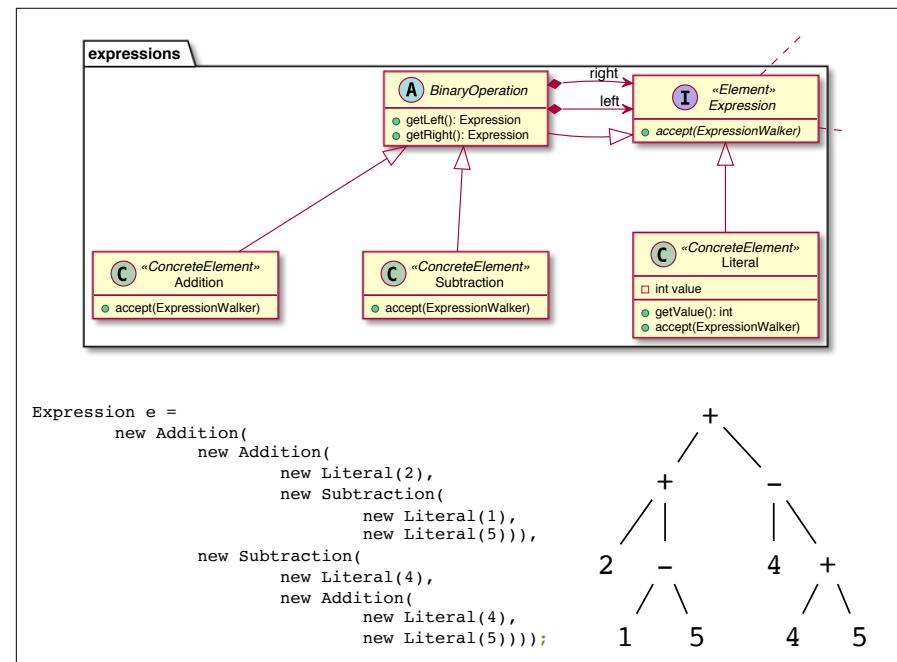
49



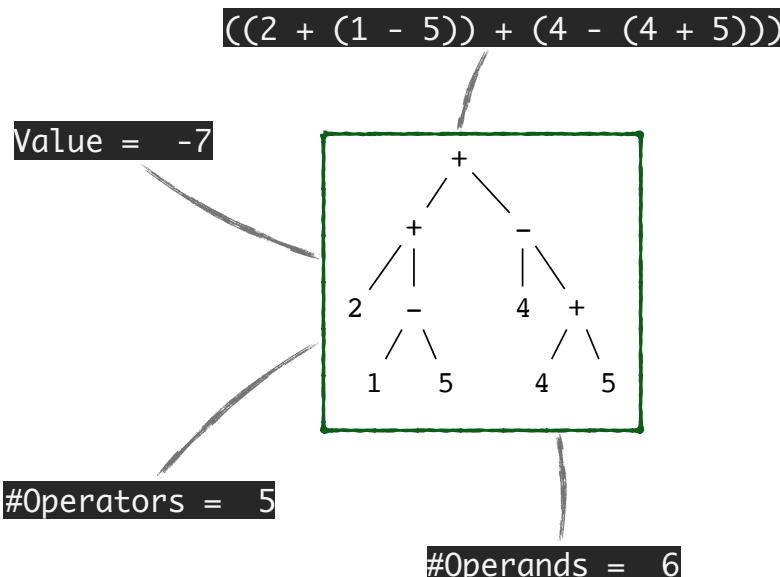
50



51



52



53

```
PrettyPrinter printer = new PrettyPrinter();
e.accept(printer);
System.out.println("Expression: " + printer.getResult());

OperandCounter opc = new OperandCounter();
e.accept(opc);
System.out.println(" #Operands = " + opc.getResult());

OperatorCounter ops = new OperatorCounter();
e.accept(ops);
System.out.println(" #Operators = " + ops.getResult());

Evaluator eval = new Evaluator();
e.accept(eval);
System.out.println("Value = " + eval.getResult());
```

54

```
public class PrettyPrinter implements ExpressionWalker<String> {

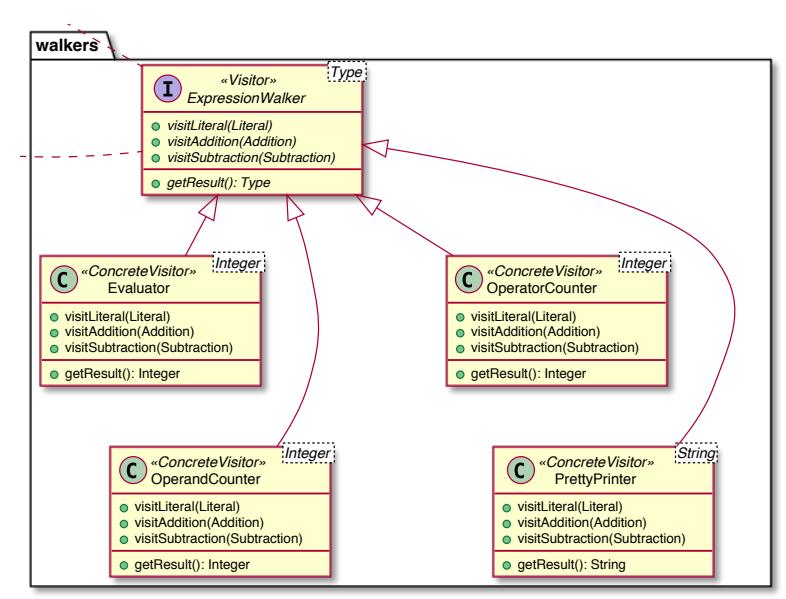
    private StringBuffer buffer = new StringBuffer();

    @Override
    public void visitLiteral(Literal literal) {
        buffer.append(literal.getValue());
    }

    @Override
    public void visitAddition(Addition addition) {
        buffer.append("(");
        addition.getLeft().accept(this);
        buffer.append(" + ");
        addition.getRight().accept(this);
        buffer.append(")");
    }

    @Override
    public void visitSubtraction(Subtraction subtraction) {
        buffer.append("(");
        subtraction.getLeft().accept(this);
        buffer.append(" - ");
        subtraction.getRight().accept(this);
        buffer.append(")");
    }

    @Override
    public String getResult() { return buffer.toString(); }
}
```



55

56

Conséquences

- On peut définir autant de visiteurs ≠ qu'on le souhaite
 - Et on peut le faire avec une approche fonctionnelle
 - (Donc un peu moins 🐷 que cet exemple, sans effets de bord)
- L'introduction du visiteur est invasive dans la hiérarchie
- Le double-dispatch à un coût non négligeable à l'exécution
 - Mais on peut faire plus sioux

57

Template Method



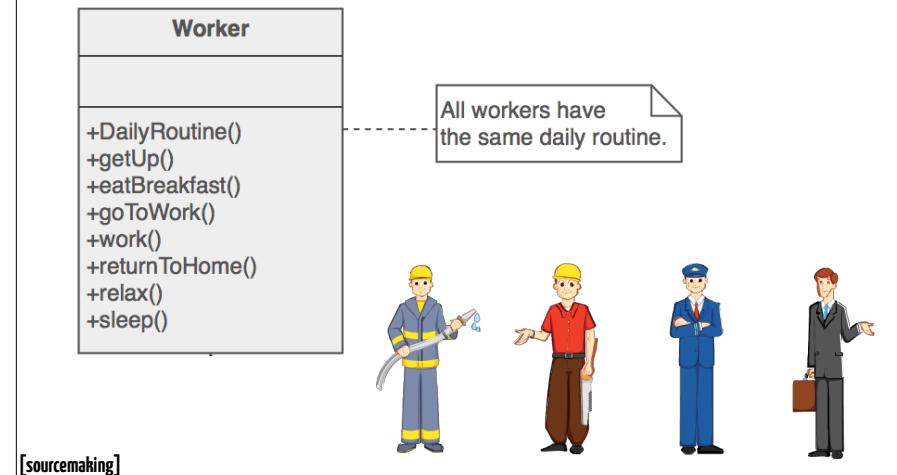
59

Où le trouver ?

- Compilation
 - On visite des arbres de syntaxes abstraits
- Transformation de modèles
 - Norme QVT, langage ATL, ...
- Langage intégrant le “pattern-matching”

58

Exemple

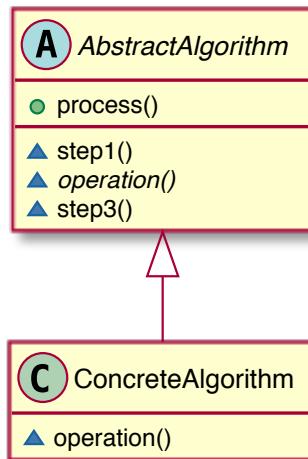


60

Problème

**Comment gérer un algorithme général,
et dont on pourra spécialiser
certaines étapes ?**

61

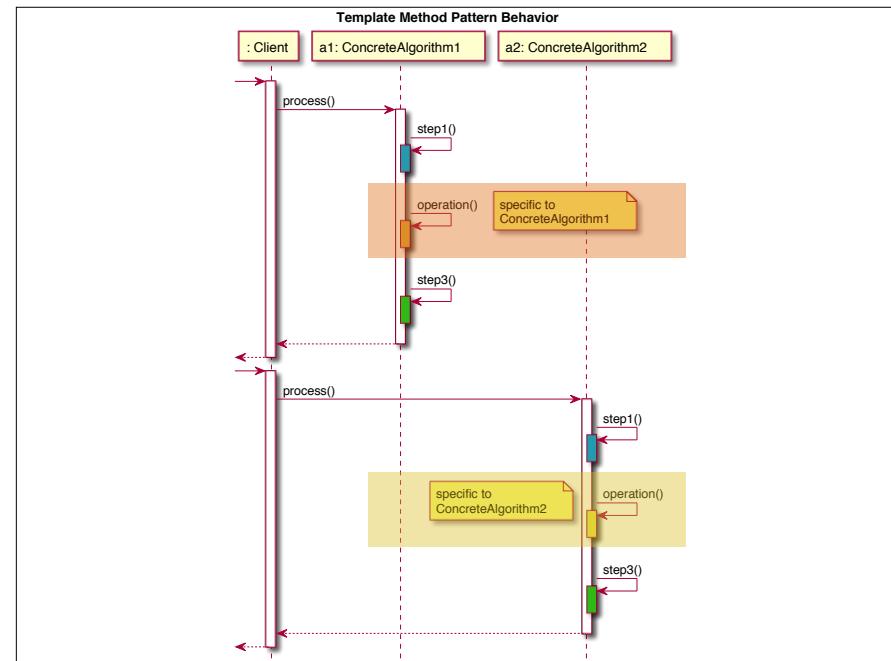


63

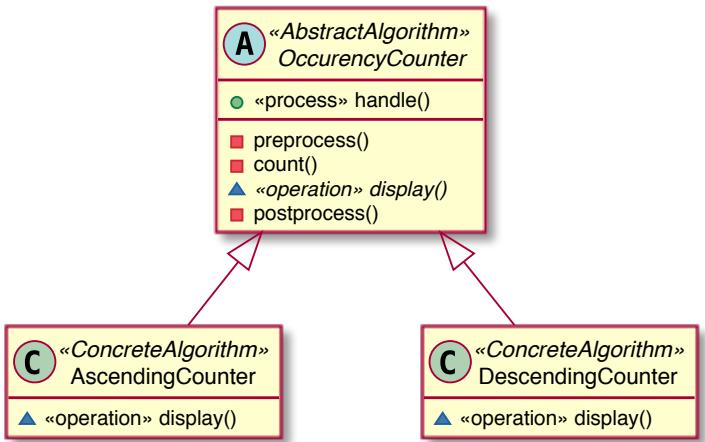
Intention

- Définir un squelette d'algorithme dans une opération
 - L'algorithme référence des étapes bien définies
- Permettre par héritage de redéfinir certaines étapes
 - Modifier l'algorithme sans modifier sa structure générale

62



64



65

```

public class AscendingCounter extends OccurrencyCounter {
    public AscendingCounter(String input) { super(input); }

    @Override protected void display() {
        this.counter.entrySet().stream()
            .sorted(comparingByValue())
            .forEach((e) -> System.out.println(" " + e));
    }
}

System.out.println("\n# Counting word occurencies using an ASC counter");
OccurrencyCounter asc = new AscendingCounter(text);
asc.handle();

System.out.println("\n# Counting word occurencies using a DSC counter");
OccurrencyCounter dsc = new DescendingCounter(text);
dsc.handle();

```

67

```

public abstract class OccurrencyCounter {

    private String data;

    protected Map<String, Integer> counter;

    public OccurrencyCounter(String input) {
        this.data = input;
        this.counter = new HashMap<>();
    }

    public void handle() {
        preprocess();
        count();
        display();
        postprocess();
    }

    private void preprocess() {
        System.out.println("Preprocessing data");
        this.data = this.data.replaceAll("[^a-zA-Z ]", " ").toLowerCase();
    }

    private void count() {
        System.out.println("Counting word occurencies");
        for(String w: data.split("\s+")) {
            counter.put(w, counter.getOrDefault(w, 0)+1);
        }
    }

    protected abstract void display();

    private void postprocess() {
        System.out.println("Found ["+counter.size()+"] distinct words");
    }
}

```

66

Conséquences

- Réutilisation du code
- Structure de contrôle inversé
- La visibilité permet de jouer sur les capacités d'extension
- Mais ...
 - Repose sur des sous-classes (cf stratégie)

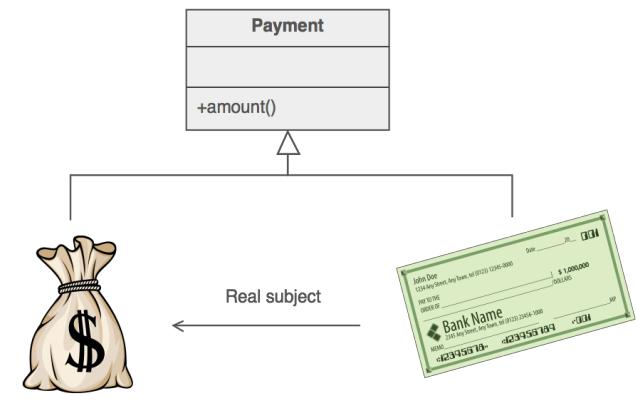
68

Où le trouver ?

- Canevas de tests unitaire
- Dans l'API Java :
 - Entrées / Sorties (InputStream, Writer, ...)
 - Structure de données abstraites (AbstractList, ...)
 - Gestion du protocole HTTP (HTTPServlet.doXXX())

69

Exemple



[sourcemaking]

71

Proxy

70

Problème

**Comment fournir un substitut
à un objet qui n'est pas
accessible facilement ?**

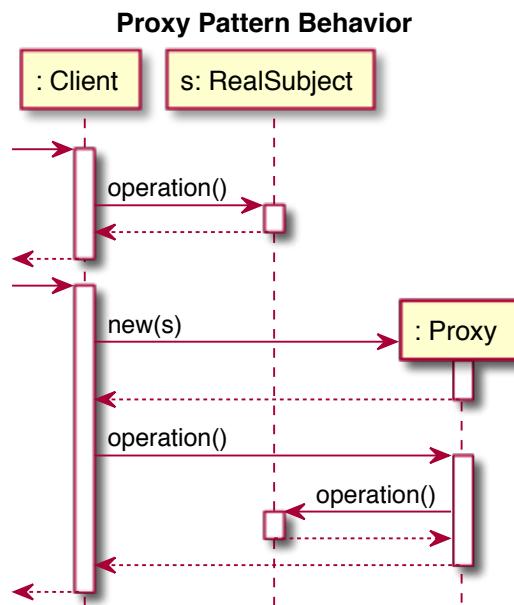
72

080_patrons_chevaliers - March 20, 2019

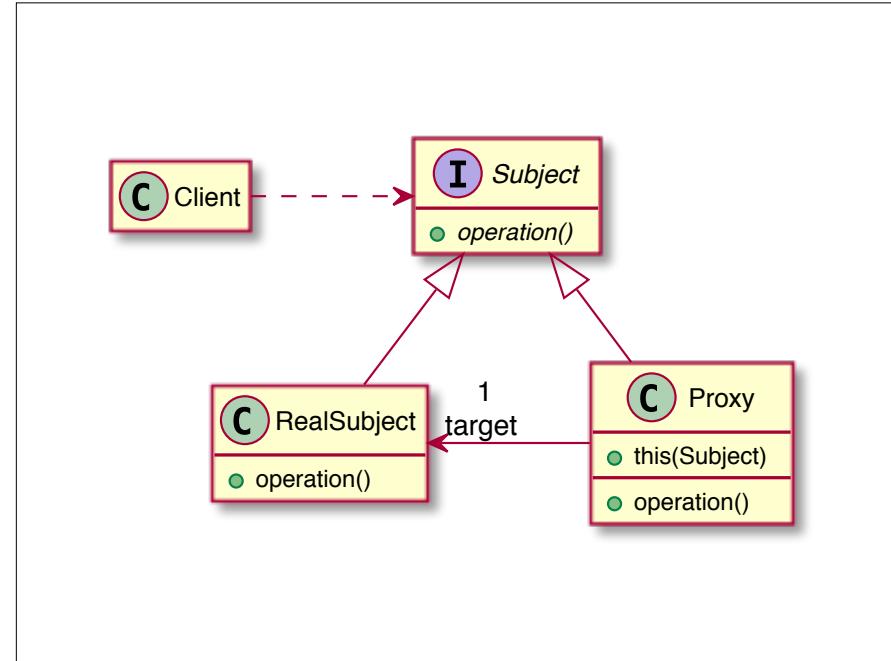
Intention

- Séparer l'interface de l'implémentation
- Réifier le concept de "procuration" pour substituer un objet inaccessible par un autre, qui lui est accessible.
- Travailler avec le substitut et le vrai objet de manière transparente

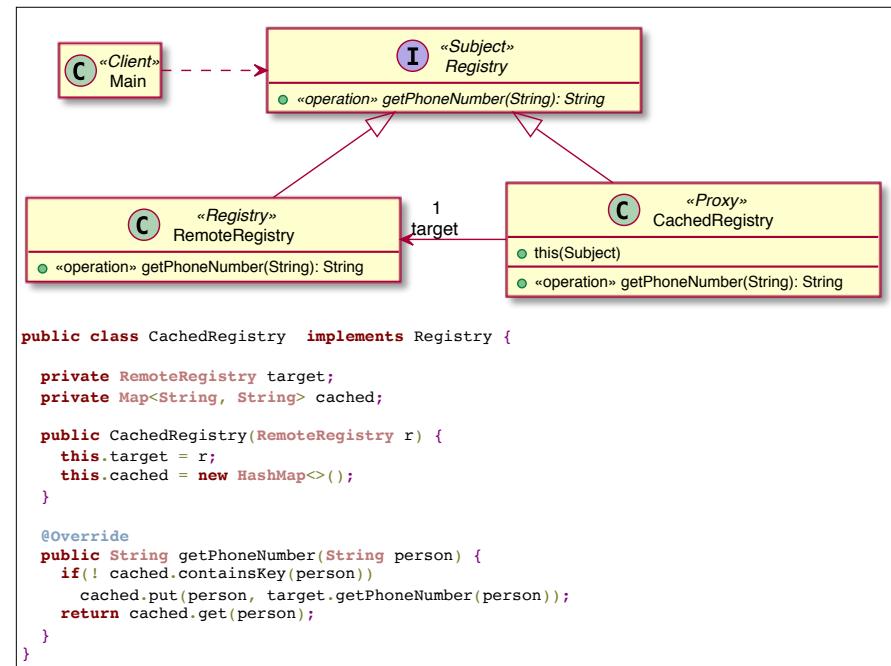
73



75



74



76

Conséquences

- L'inaccessibilité du sujet est transparente
- Proxy et Sujet peuvent être aisément échangés
- Le proxy n'est pas une extension (sous-classe) du sujet, mais simplement une réplique exacte au même niveau dans la hiérarchie

77

Où le trouver ?

- Principe d'échantillonnage de données
 - Image volumineuse à charger
- Gestion des droits d'accès
 - Le proxy gère les droits, l'objet le métier
- Objets distants
 - Java RMI, talons (stubs) d'accès à des WebServices
- Bouchons / Espions pour le test logiciel

78