NFP121, Cnam/Paris
Design Pattern
Injection de dépendance
Couplage faible & Variabilité

jean-michel Douin, douin au cnam point fr
version : 06 Janvier 2020

Notes de cours

Bibliographie utilisée

- Points de départ
  - https://liris.cnrs.fr/inforsid/sites/default/files/a587c1JF7awD6AU8w.pdf
    - Expression et usage de la variabilité dans les patrons de conception
      Nicolas Arnaud — Agnès Front — Dominique Rieu
  - http://users.polytech.unice.fr/~blay/ENSEIGNEMENT/IDM/presentation-idmLahire.pdf
    - Ligne de Produits Logiciel et Variabilité des modèles
      - Module IDM (Ingénierie Des Modèles) de 5ème année Philippe Lahire

- Feature-Oriented Software Product Lines concepts and implementation
  - Sven Apel · Don Batory Christian Kästner · Gunter Saake
    Principalement les chapitres 4 et 5 de
    http://www.springer.com/la/book/9783642375200

- https://www.cs.cmu.edu/~ckaestne/pdf/JOT09_OverviewFOSD.pdf
    - An Overview of Feature-Oriented Software Development
- http://pagesperso.lip6.fr/Reda.Bendraou/IMG/pdf/lignes_de_produits.pdf

- **Injection de dépendances**
  - **https://martinfowler.com/articles/injection.html**

Sommaire

- **Le Contexte**
  - **Couplage faible ou Variabilité ?**
    - **au niveau conception du logiciel**
- **Les Objectifs**
  - **Ajouter de nouvelles fonctionnalités avec le moins d'impact possible, en supprimer aussi …**
- **Design Pattern == Variabilité** *de fait*
  - **Un exemple classique : Le patron décorateur** permet d'ajouter de nouvelles fonctionnalités(décorateurs)
    - **Un nouveau décorateur engendre une modification du source**
- **Un Conteneur de *beans*,** *beans* prêts à être injectés
  - Vers une séparation configuration/utilisation, la configuration évolue, le code ne change pas
    - Un nouveau décorateur engendre uniquement une modification de la configuration
- **Les patrons Stratégie, Commande, Chaîne de responsabilités, État, …**
  - Bénéfices de l'usage conjoint des patrons et d'un conteneur
- **Le patron ServiceLocator**
  - Un conteneur de conteneurs ?
- **Usage de Publish/Subscribe,**
  - **Vers une généralisation ?**
    - **Vers une architecture logicielle en composants interchangeables Service Component Architecture (SCA) ?**
  - **Une introduction d'une architecture par composants, quelle variabilité ?**

Objectifs, Variabilité au niveau conception

- **Substitution d'une implémentation par une autre**
  - **Un langage à objets, le permet**
    - **interface, sous-classes, classes abstraites, héritage, principe de substitution**
      - **Polymorphisme, liaison dynamique**

- **Ajout de nouvelles fonctionnalités**
  - ***Design pattern*, par exemple le patron décorateur**

- **Modification du code source ou non ?**
  - **Usage d'un *framework*, à injection des dépendances,**
    - **Un conteneur de beans,** le patron décorateur revisité

- **Les patrons Stratégie, Commande, Etat, Visiteur,…** *configurés correctement…*
  - Usage conjoint d'un conteneur de beans et des patrons

- **Variabilité → Design Pattern → couplage faible ou l'inverse ?**

Substitution d'une implémentation par une autre

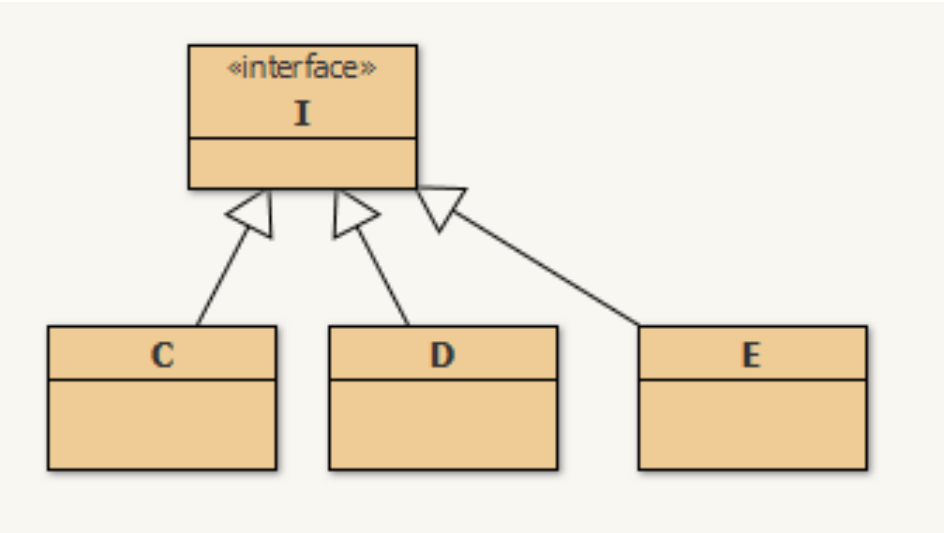
Dans le langage :

- **Interface en java**

- **I i = new C();**

// une nouvelle implémentation pour i

- **I i = new D();**



Modification du source évitée si

Par introspection,

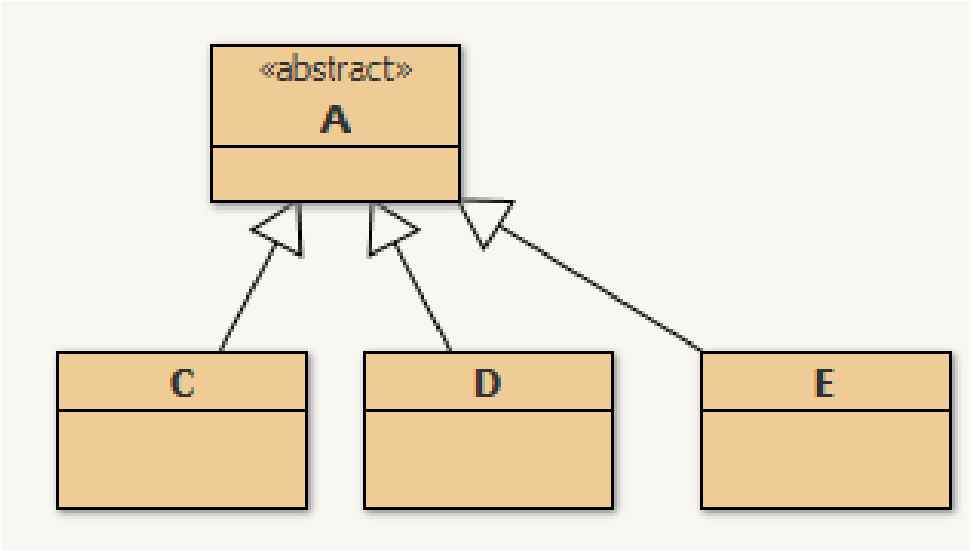
- **I i = (I)Class.forName("E").newInstance();**

Ou bien depuis un conteneur de beans (ici en syntaxe apparentée Spring)

- **I i = (I)container.getBean("E_bean");**
  - *Le nom* **E_bean** *pourrait être dans un fichier texte dit de configuration …*

Substitution une implémentation par une autre

- **Classe abstraite ou interface**
- **A a = new C();**
- **A a = new D();**



Modification du source évitée si

→ *Introspection*

A a = (A)Class.forName("E").newInstance();

→ *Ou bien depuis un conteneur de beans*

A a = (A)container.getBean("E_bean");

Design pattern

- **Modèles de conception réutilisables**

- **Wikipedia : <<Les patrons de conception décrivent des procédés de conception généraux et permettent en conséquence de capitaliser l'expérience appliquée à la conception de logiciel. Ils ont une influence sur l'architecture logicielle d'un système informatique. >>**

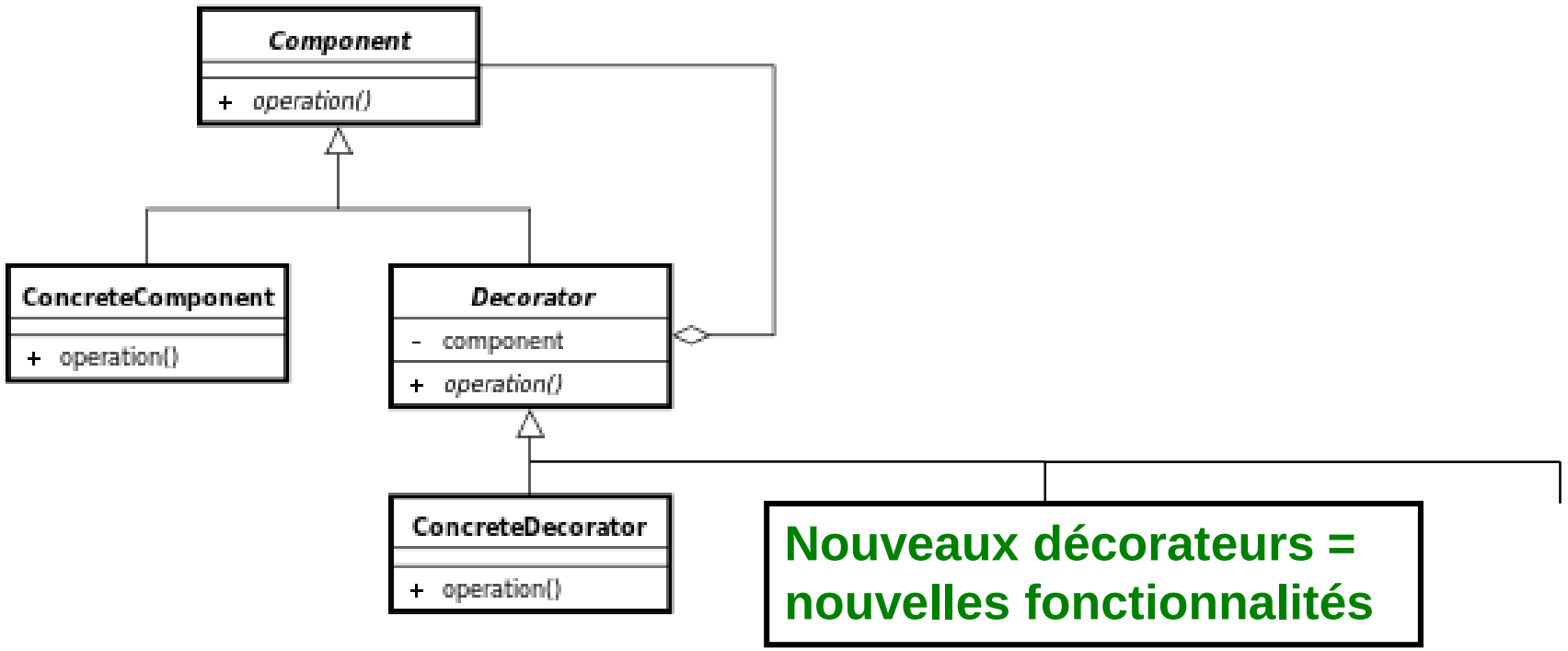
- **Un catalogue existe**
  - ***Design patterns. Catalogue des modèles de conception réutilisables***

  Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (soit « *Gang of Four* », abrégé *GoF*) et publié en 1994 chez Addison-Wesley.

- **Le patron décorateur fait partie de ce catalogue**
  - **(23 patrons au total)**
  - http://jfod.cnam.fr/NFP121/supports/extras_designpatternscard.pdf

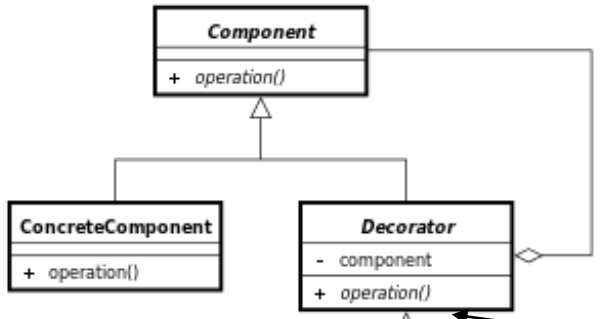
Ajout(s) de nouvelles fonctionnalités ?

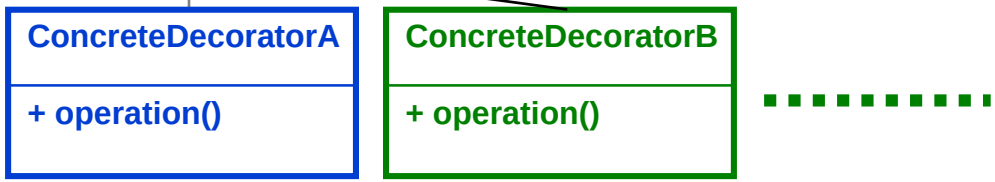
- **wikipedia https://en.wikipedia.org/wiki/Decorator_pattern**
- <<Un décorateur permet d'attacher dynamiquement de nouvelles responsabilités à un objet. Les décorateurs offrent une alternative assez souple à l'héritage pour composer de nouvelles fonctionnalités.>>



Ajout(s) de nouvelles fonctionnalités

- **Component c = new ConcreteComponent();**
- **c.operation();**






```
c = new ConcreteDecoratorA(new ConcreteComponent());
c.operation();

c = new ConcreteDecoratorB(new ConcreteDecoratorA(new ConcreteComponent()));
c.operation();
```

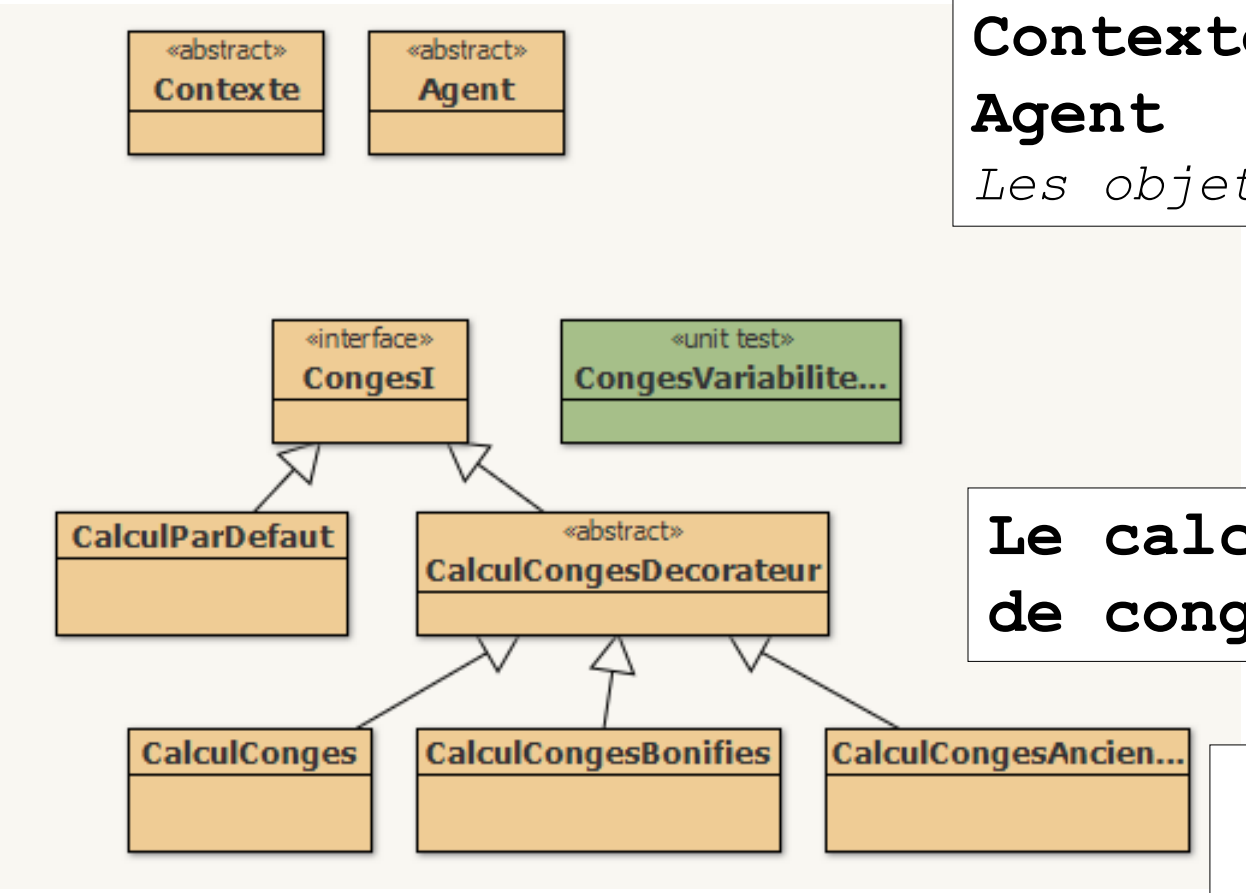
Par introspection

- **c = (Component)Class.forName("ConcreteDecoratorB").newInstance();…;**
- **c.operation();**

Un exemple approprié :le calcul des congés
(Combien de jours me reste-t-il ?)

- **Le contexte : Une mairie, des agents et des congés**
- **Objectif :**
  - **Le calcul des congés pour un agent selon son statut**
- **La réglementation évolue, les statuts, l'ancienneté, …**
  - **Variabilité détectée à l'analyse pour le calcul des congés**
- **Usage du patron décorateur**
  - **Une décoration par fonctionnalité**
    - **Si l'agent est titulaire**
    - **Si l'agent est originaire des DOM-TOM ou de Corse**
    - **Si l'ancienneté de l'agent est supérieure à 3 ans alors un jour de plus par an**
    - **Si la mairie se trouve aux DOM-TOM alors**
    - **…**
      - **Règles fictives, pour l'exemple…**
  - **Une nouvelle fonctionnalité -> une nouvelle décoration**

Exemple : Calcul des congés d'un agent
en mode simplifié…



Contexte (Mairie,…
Agent
Les objets métiers

Le calcul des jours de congés

……variabilité
Nouvelles fonctionnalités
Nouveaux décorateurs

```
public interface CongesI{

    public Integer calculer(Contexte ctxt, Agent agent);

}
```

Usage du patron décorateur

```
public void testCalculConge() throws Exception{
    CongesI conges =
      new CalculConge(
        new CalculCongesBonifies(
          new CalculParDefaut()));

      Agent paul = …
      Contexte contexte = …   // la mairie, …

       int joursDeCongés = conges.calculer(contexte, paul);


      assertEquals(15, joursDeCongés);
      // paul n'a plus que 15 jours de congés !, c'est peu…
}
```

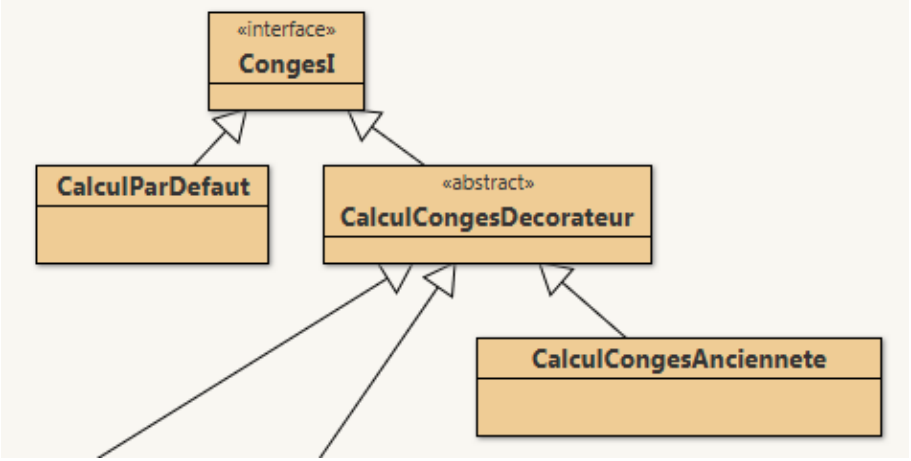
Variabilité : Ajout d'un décorateur

- **Une nouvelle fonctionnalité de congé, l'ancienneté…**
  - **→ Un nouveau décorateur**

```
public void testCalculConge() throws Exception{
   CongesI conges =
    new CalculConge(
      new CalculCongesBonifies(
          new CalculCongesAnciennete(new CalculParDefaut())));

      Agent paul = …
      Contexte contexte = …
      int joursDeCongés = conges.calculer(contexte, paul);

   assertEquals(60, joursDeCongés );
   // il reste à paul  60 jours de congés !,
   // d'autres décorateurs pourraient être ajoutés … encore plus de jours de congés …
 }
```

Patron décorateur, ici le cumul implicite des calculs

Décorateur et calcul des nouveaux congés dus à l'ancienneté




```
public class CalculCongesAnciennete
     extends CalculCongesDecorateur{

    public CalculCongesAnciennete(){super();}

    public CalculCongesAnciennete(CongesI calculette){
      super(calculette);
    }

    public Integer calculer(Contexte ctxt, Agent agent){
      return super.calculer(ctxt,agent) + agent.anciennete()*5;
                                          // 5 jours par an en + ;-)
    }
}
```

La classe CalculCongesDecorateur est en annexe

Ajout d'un nouveau décorateur : discussions

- **Modification du source ?**
  - **À chaque ajout de fonctionnalités ?!?**

```
CongesI conges =
      new CalculConge(
        new CalculCongesBonifies(
          new CalculCongesAnciennete(new CalculParDefaut())));
```

Ici modification du source

variabilité effective : il suffit de créer un décorateur et de modifier le source Java

Pouvons-nous éviter de modifier le source avec les mêmes objectifs ?

- **Avec un conteneur de *Beans ?***
  - **Utilisation de *femtoContainer*, développé au Cnam pour NFP121,**
    - **Apparenté Spring, en syntaxe du moins**
      - **(de tels outils existent et sont largement utilisés Spring, Google Guice, picoContainer…)**
      - **https://en.wikipedia.org/wiki/Dependency_injection**

Conteneur de beans + design pattern

- **Variabilité induite par**
  - **l'utilisation de patrons**
    - **Modèles de conception**
    - **Une nouvelle fonctionnalité, nécessite de modifier le source Java**
  - **l'utilisation d'un conteneur de beans**
    - **Une nouvelle fonctionnalité, nécessite de modifier un fichier texte de configuration**

Patrons + conteneur = variabilité maximale ?

à suivre…

Chemin faisant…

1. **Le patron Décorateur**
   - Un exemple de cumul de fonctionnalités
2. **Le patron Décorateur + conteneur de beans**
   - Les décorateurs sont injectés selon une configuration
3. **Quelques patrons de conception**
   - Stratégie, Commande, Chaîne de responsabilités, État, Composite/Visiteur, …
   - Les fonctionnalités associées à ces patrons sont injectées
4. **Un contexte, une application, plusieurs patrons**
   - Par contexte, les fonctionnalités liées à ces patrons sont injectées
5. **Plusieurs contextes pour une variabilité maximale ?**

Le patron Décorateur, avec un outil d'injection, une séparation nette configuration/utilisation

- **Pas de modifications du source, juste l'usage de femtoContainer**
  - **Avant l'ajout du `calculCongeAnciennete`**
  - **Nous avions le fichier de configuration ci-dessous**

```
bean.id.1=calculParDefaut
calculParDefaut.class=injection_decorateur.CalculParDefaut

bean.id.2=calculConge
calculConge.class=injection_decorateur.CalculConges
calculConge.property.1=calculette
calculConge.property.1.param.1=calculCongeBonifie

#
bean.id.3=calculCongeBonifie
calculCongeBonifie.class=injection_decorateur.CalculCongesBonifies
calculCongeBonifie.property.1=calculette
calculCongeBonifie.property.1.param.1=calculParDefaut
```

Soit la création à la place du programmeur:

```
CongesI conges =
        new CalculConge(
            new CalculCongesBonifies(
                new CalculParDefaut()));
```

femtoContainer : utilisation en quelques lignes

Configuration, un fichier texte, clé=valeur

```
bean.id.2=calculConge
calculConge.class=injection_decorateur.CalculConges
calculConge.property.1=calculette
calculConge.property.1.param.1=calculCongeBonifie
```

utilisation

```
ApplicationContext ctx = Factory.createApplicationContext();
   // en syntaxe apparentée Spring
   CongesI conges = ctx.getBean("calculConge");
   Contexte contexte = …
   Agent paul  = …
   int joursDeCongés = conges.calculer(contexte, paul);

   assertEquals(15, joursDeCongés);

     // 15 jours pour le moment, …
```

Avec un outil d'injection, ajout d'une fonctionnalité

- **Pas de modifications du source !**
  - **juste une nouvelle configuration, ajout du calcul de l'ancienneté**
  - **Le nouveau fichier de configuration …**

en *rouge italique* ce qui a été modifié

```
bean.id.1=calculParDefaut
calculParDefaut.class=injection_decorateur.CalculParDefaut

bean.id.2=calculConge
calculConge.class=injection_decorateur.CalculConges
calculConge.property.1=calculette
calculConge.property.1.param.1=calculCongeBonifie

#
bean.id.3=calculCongeBonifie
calculCongeBonifie.class=injection_decorateur.CalculCongesBonifies
calculCongeBonifie.property.1=calculette
calculCongeBonifie.property.1.param.1=calculCongeAnciennete

bean.id.4=calculCongeAnciennete
calculCongeAnciennete.class=injection_decorateur.CalculCongesAnciennete
calculCongeAnciennete.property.1=calculette
calculCongeAnciennete.property.1.param.1=calculParDefaut
```

Le source est inchangé, mais les jours de congés ont augmenté…

Utilisation : le code n'a pas changé

```
ApplicationContext ctx = Factory.createApplicationContext();
   CongesI conges = ctx.getBean("calculConge");

   Contexte contexte = …
   Agent paul = …
   int joursDeCongés = conges.calculer(contexte, paul);

   assertEquals(60, joursDeCongés);
```

- **Mais Paul a maintenant 60 jours de congés…**
  - **Sans aucune modification du source java**
  - **Seul le fichier de configuration est concerné**

Outil d'injection : une explication sommaire

L'analyse de ce texte*

```
bean.id.2=calculConge
calculConge.class=injection_decorateur.CalculConges
calculConge.property.1=calculette
calculConge.property.1.param.1=calculCongeBonifie
```

Engendre au sein du conteneur

```
calculConge = new injection_decorateur.CalculConges();
calculConge.setCalculette(calculCongeBonifie);
```

L'utilisateur demande une référence à l'aide d'un nom

```
CongesI conges = container.getBean("calculConge");
Contexte contexte = … // Mairie
Agent paul = … // informations pour cet agent

int nombreDeJoursDeCongés = conges.calculer(contexte, paul);
```

Bean : https://fr.wikipedia.org/wiki/JavaBeans

* Fichier de propriétés en java, cf. java.util.Properties

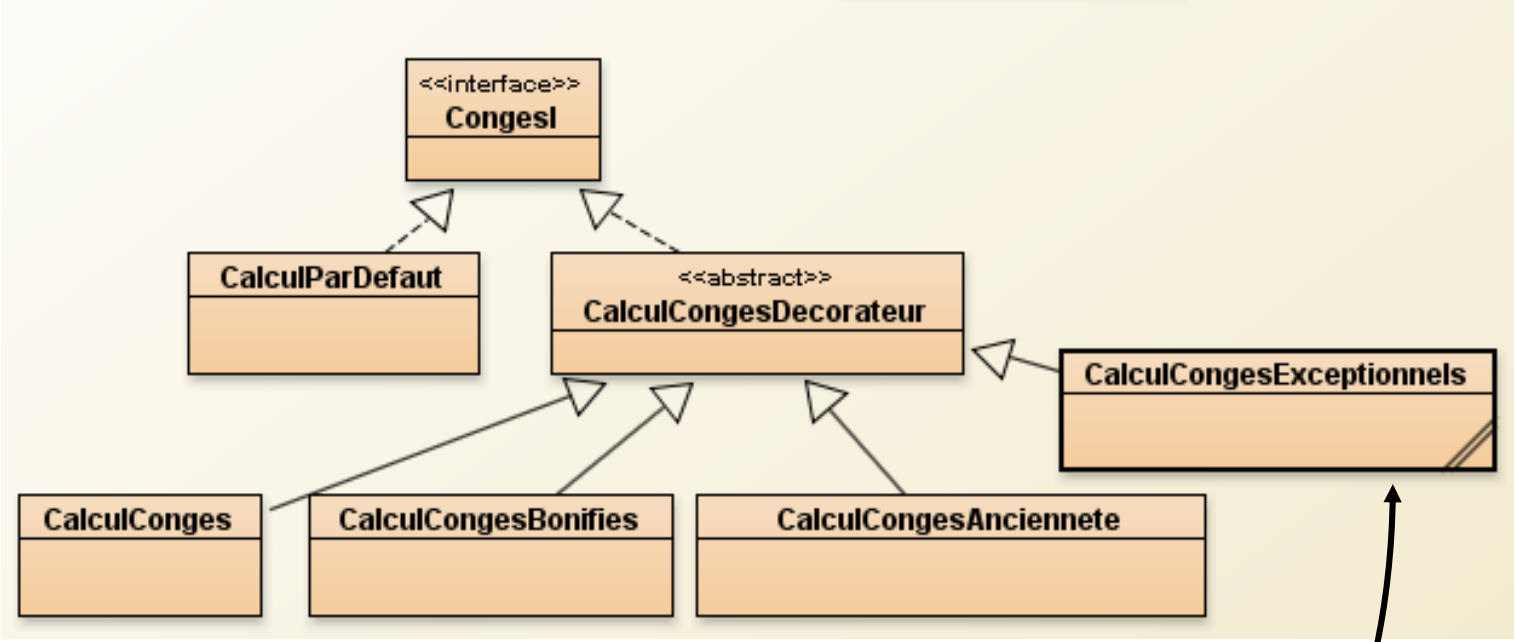
Démonstration/Discussions

- **Ajout d'un nouveau décorateur …**

- **En direct ou avec les 3 diapositives qui suivent**

Démonstration: ajout d'un nouveau décorateur

- **Des congés exceptionnels de 5 jours de plus pour tous**



- **Côté développeur**
  - **Création de la classe CalculCongesExceptionnels**

Démonstration: La classe CalculCongésExceptionnels

```
public class CalculCongesExceptionnels
                    extends CalculCongesDecorateur{

    public CalculCongesExceptionnels(){super();}

    public CalculCongesExceptionnels(CongesI calculette){
      super(calculette);
    }

    public Integer calculer(Contexte ctxt, Agent agent){
      return super.calculer(ctxt,agent) + 5; // + 5 jours pour
                                             //          tous
    }
}
```

1) Une nouvelle classe est créée : une nouvelle décoration
   - Développeur

2) Modification du fichier texte
   - Configurateur

Démonstration: la nouvelle configuration

```
bean.id.4=calculCongeAnciennete
calculCongeAnciennete.class=injection_decorateur.CalculCongesAnciennete
calculCongeAnciennete.property.1=calculette
# modification de la ligne ci-dessous : ajout du nouveau décorateur
#calculCongeAnciennete.property.1.param.1=calculParDefaut
calculCongeAnciennete.property.1.param.1=calculCongesExceptionnels

# Ajout du bean nommé calculCongesExceptionnels
bean.id.5=calculCongesExceptionnels
calculCongesExceptionnels.class=injection_decorateur.CalculCongesAnciennete
calculCongesExceptionnels.property.1=calculette
calculCongesExceptionnels.property.1.param.1=calculParDefaut
```

- **Côté configurateur**
  - **Ajout de la *description du nouveau bean calculCongesExceptionnels***
    - **bean.id.5 *La classe java, ses propriétés***
  - ***Nouveau décorateur du bean.id.4 :*** calculCongeAnciennete
    calculCongeAnciennete.property.1.param.1=**calculCongesExceptionnels**

- **le source java du calcul des congés pour un agent est inchangé**

En résumé

1. **Le développeur se concentre sur la nouvelle fonctionnalité**
   Fonctionnalité incluse dans les variabilités détectées

   Création d'un décorateur

   Tests unitaires … « validation »

2. **Le configurateur modifie le fichier de propriétés**
   Ajout de ce nouveau décorateur en tant que bean

3. **L'appel du calcul est toujours le même**
   La nouvelle fonctionnalité est prise en compte
   Inhérent à l'usage du patron décorateur

Outil d'injection : une explication sommaire

- **Les 3 diapositives qui suivent, présentent**
  - **femtoContainer**
    - **Un outil d'injection minimaliste pour NFP121**
    - **Description sommaire du fonctionnement**
- **Puis**
  - **Usage conjoint d'un conteneur et des patrons**

Outil d'injection : une explication sommaire

L'analyse de ce texte*

```
bean.id.2=calculConge
calculConge.class=injection_decorateur.CalculConges
calculConge.property.1=calculette
calculConge.property.1.param.1=calculCongeBonifie
```

Engendre au sein du conteneur

```
calculConge = new injection_decorateur.CalculConges();
calculConge.setCalculette(calculCongeBonifie);
```

L'*utilisateur* demande une référence à l'aide d'un nom

```
ApplicationContext ctx = Factory.createApplicationContext();
 CongesI conges = ctx.getBean("calculConge");
  Contexte contexte = …
  Agent paul = …

int nombreDeJoursDeCongés = conges.calculer(contexte, paul);
```

Bean : https://fr.wikipedia.org/wiki/JavaBeans

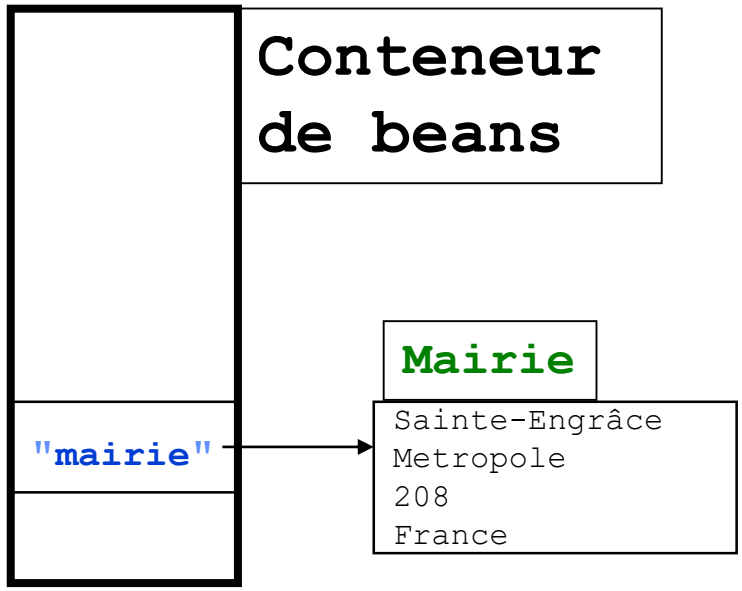
* Ici un fichier de propriétés en java cf. java.util.Properties, Autres formtas possibles XML, JSON …

femtoContainer, la configuration : syntaxe

- **Java :Une classe et des attributs (propriétés)**

```
public class Mairie extends Contexte{
  private String nom;
  private String lieu;
  private String pays;
  private int nombreHabitant
  public Mairie(){}
  public void setNom(String nom){this.nom=nom;}
  public void setLieu(String lieu){this.lieu=lieu;}
  …
}
```

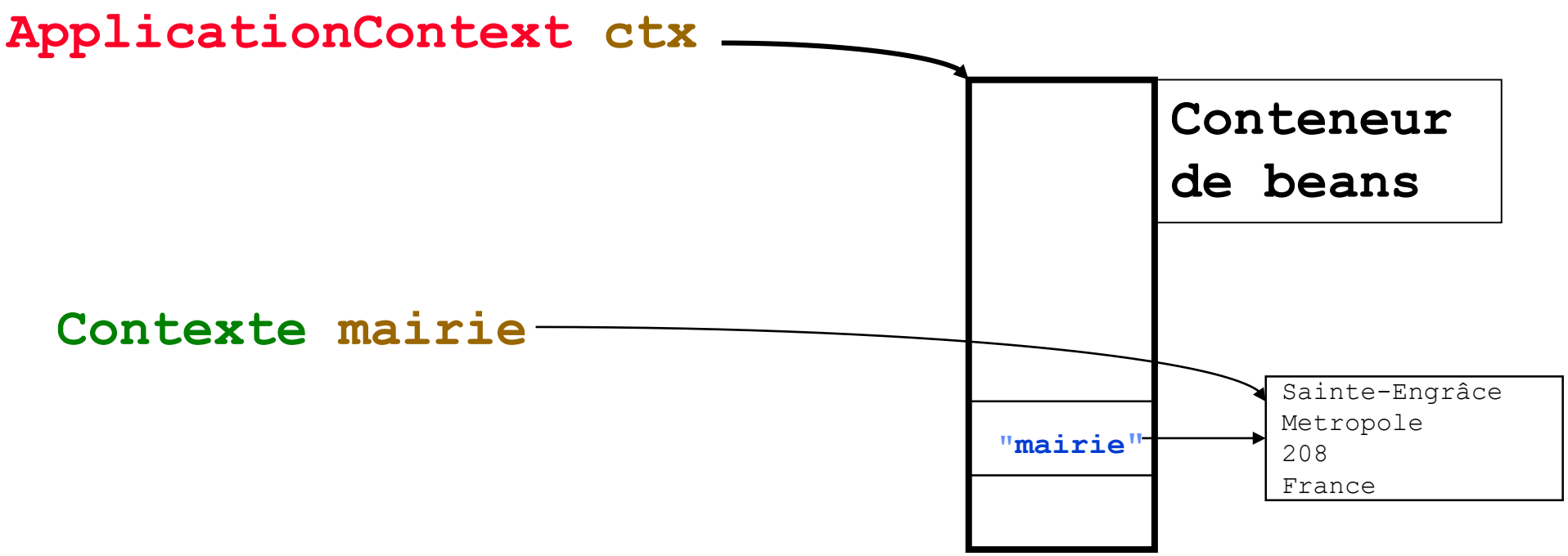
```
# le nom du « bean »
bean.id.1=mairie
# la classe Java
mairie.class=Mairie
# les attributs
mairie.property.1=nom
mairie.property.1.param.1=Sainte-Engrâce
mairie.property.2=lieu
mairie.property.2.param.1=Metropole
mairie.property.3=nombreHabitants
mairie.property.3.param.1=208
mairie.property.4=pays
mairie.property.4.param.1=France
```



femtoContainer, utilisation: syntaxe

- **Java**
  - Utilisation de femtoContainer

```
ApplicationContext ctx = Factory.createApplicationContext();
Contexte mairie = ctx.getBean("mairie");
```



Chemin faisant…

1. Le patron Décorateur
   - Un exemple de cumul de fonctionnalités
2. Le patron Décorateur + conteneur de beans
   - Les décorateurs sont injectées selon une configuration
3. **Quelques patrons de conception**
   - Stratégie, Commande, Chaîne de responsabilités, État, Composite/Visiteur, …
4. Un contexte, une application, plusieurs patrons
5. Plusieurs contextes pour une variabilité maximale ?

Usage conjoint d'un conteneur et des patrons

- **Préambule**
  - Design Pattern, par essence réutilisables.

- **Les patrons étudiés avec un conteneur**
  - Stratégie, Commande, Chaîne de responsabilités, État, Visiteur…

  - Pour chaque Patron
    - L'exemple des congés
    - Ajout d'une nouvelle fonctionnalité

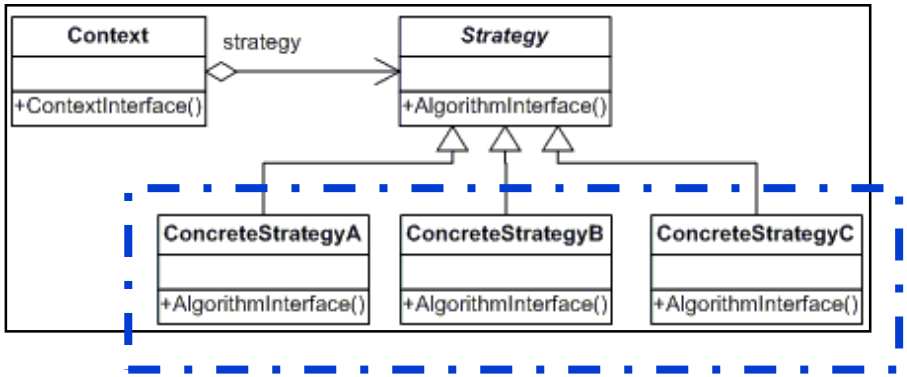
    - Quel impact sur le logiciel ?
      - Nouvelle configuration
      - Modification du source ou non
      - Simple à mettre en œuvre ?

Usage conjoint d'un conteneur et des patrons

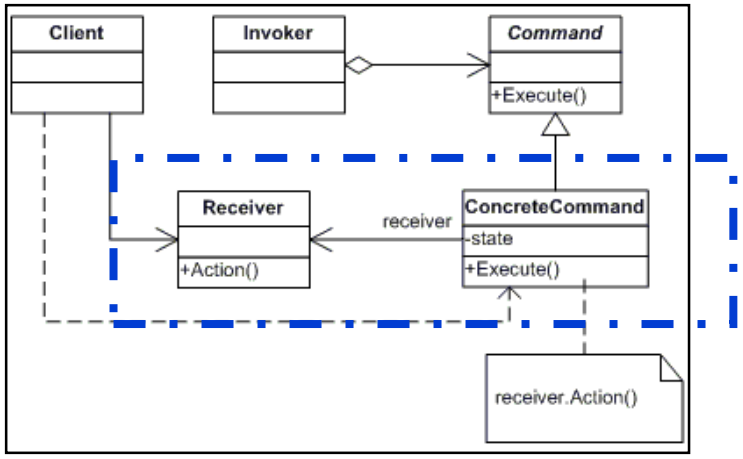
- **Hypothèse**
  - Tout classe possède les propriétés des Bean (constructeur par défaut, set et get …)
- **Patrons étudiés, en préalable:**

Nouvelles fonctionnalités
Variabilité

Stratégie



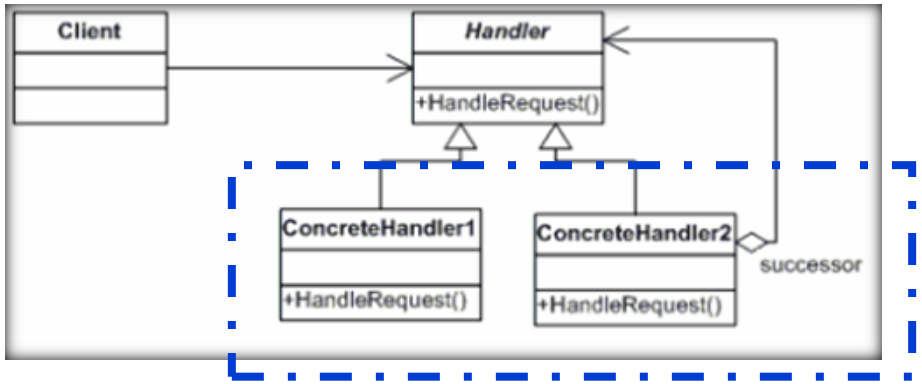
Commande



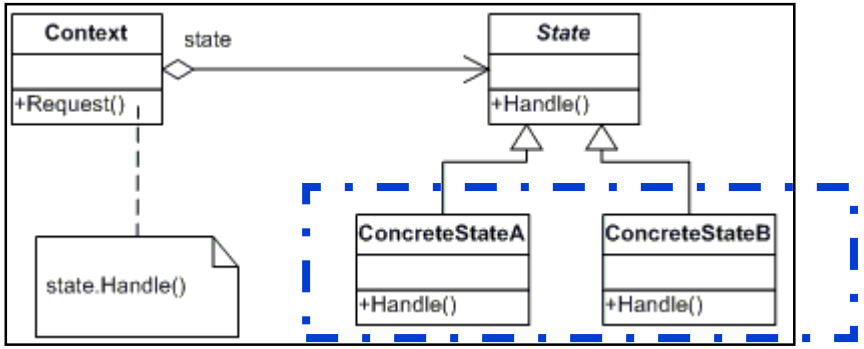
Usage conjoint d'un conteneur et des patrons

- **Hypothèse**
  - Tout classe possède les propriétés des Bean (constructeur par défaut, set et get …)
- **Patrons étudiés, en préalable:**

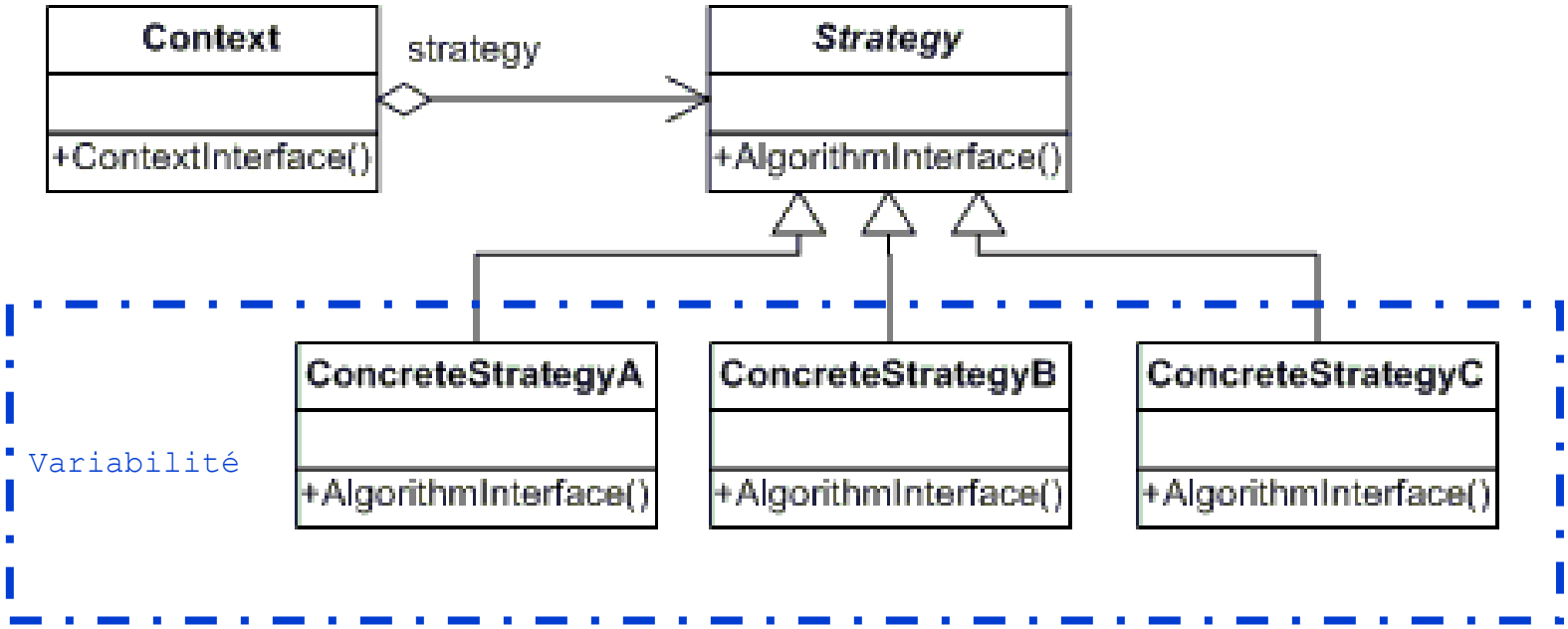
Chaîne de responsabilités



Etat

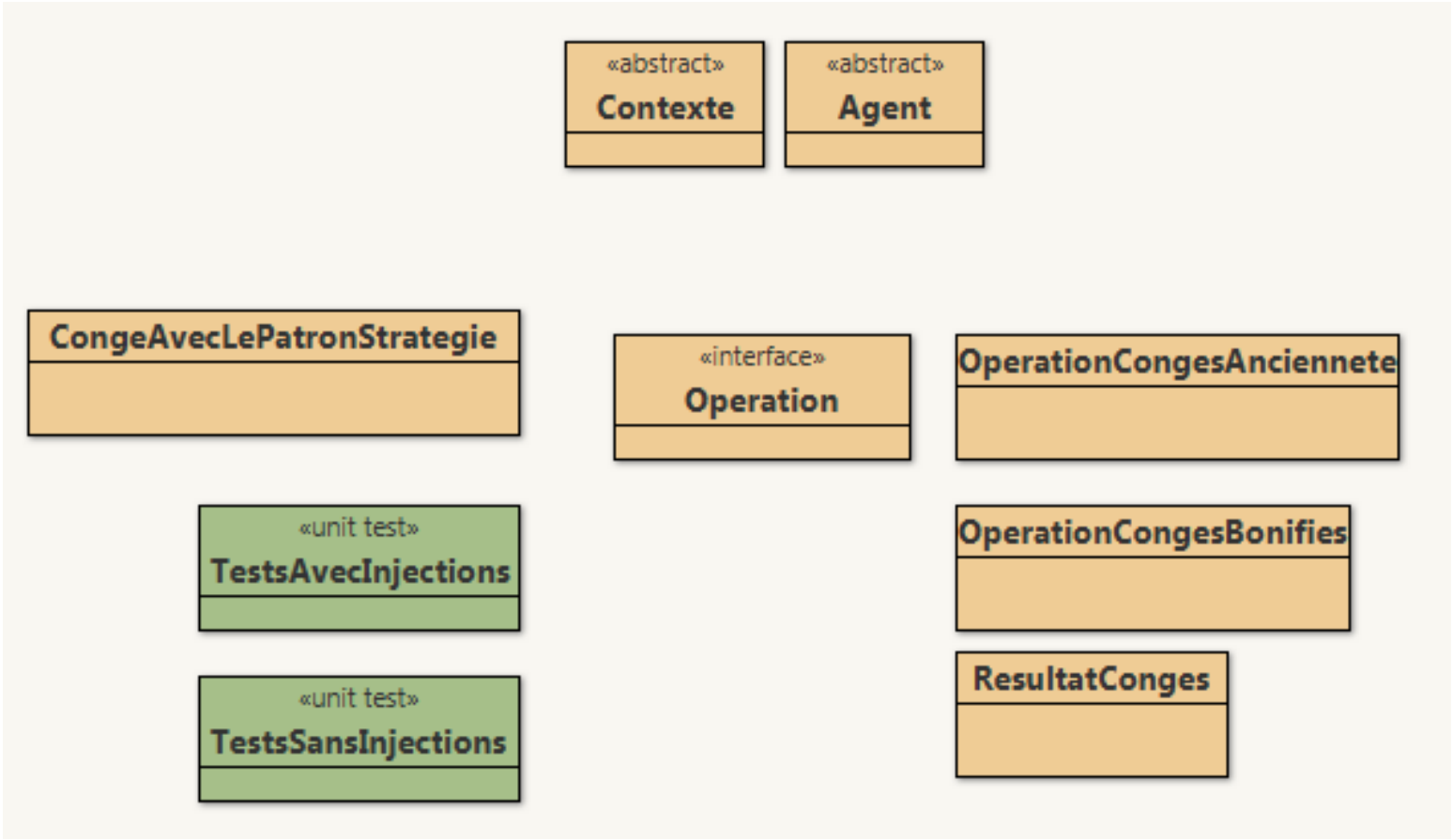


Le patron Stratégie



- **Le calcul des congés correspond à l'une des stratégies**
  - **Nouveau calcul = une nouvelle stratégie**
    -> Soit une modification de la configuration

Les congés et le patron Stratégie



- **Une nouvelle fonctionnalité, une nouvelle opération**
- **Une seule stratégie à la fois, le « client » décide de la stratégie**

L'Operation <C, R, E> est maintenant générique

- **Quelque soit les futures « opérations »** *variabilité…*

- **Un résultat R dans un certain contexte C avec une entité E**

```
public interface Operation<C, R, E>{

  public void apply( C ctxt, R result, E entity)
                                        throws Exception;

}
```

Une opération concrète par fonctionnalité

Configuration, ici deux « stratégies » possibles

```
bean.id.1=congeAnciennete
congeAnciennete.class=injection_strategie.CongeAvecLePatronStrategie
congeAnciennete.property.1=contexte
congeAnciennete.property.1.param.1=null
congeAnciennete.property.2=operation
congeAnciennete.property.2.param.1=operationCongesAnciennete

bean.id.2=operationCongesAnciennete
operationCongesAnciennete.class=injection_strategie.OperationCongesAnciennete

bean.id.3=operationCongesBonifies
operationCongesBonifies.class=injection_strategie.OperationCongesBonifies

bean.id.4=congeBonifie
congeBonifie.class=injection_strategie.CongeAvecLePatronStrategie
congeBonifie.property.1=contexte
congeBonifie.property.1.param.1=null
congeBonifie.property.2=operation
congeBonifie.property.2.param.1=operationCongesBonifies
```

femtoContainer : Usage en java

```
// le fichier de configuration
ApplicationContext container =
 Factory.createApplicationContext("injection_strategie/README.TXT");


  CongeAvecLePatronStrategie conge = container.getBean("congeAnciennete");

  ResultatConges result = new ResultatConges();
  Agent paul = …

  assertEquals(50, conge.calcul(paul).getNombreDeJours());
   // 50 jours de congés


  CongeAvecLePatronStrategie conge2 = container.getBean("congeBonifie");

  assertEquals(15, conge2.calcul(paul).getNombreDeJours());
  // 15 jours de congés
```

selon la stratégie utilisée… cf. le fichier de configuration

Patron stratégie, variabilité ?

- **Une fonctionnalité a évolué:**

1. **Modification de l'Operation existante en java**

```
public class OperationCongesBonifies implements
    Operation<Contexte,ResultatConges,Agent>{

 public void apply( Contexte ctxt, ResultatConges result, Agent agent){

  if( la_condition-supplémentaire()){
     result.setNombreDeJours(result.getNombreDeJours() + 25);
  }

 } // si la_condition-supplémentaire() est satisfaite alors 25 jours de plus
}
```

Pas de modification du fichier de configuration

Patron stratégie

- **Une nouvelle fonctionnalité :**
  1. **Ajout de la nouvelle Operation en java**

```
public class OperationCongesBonifies implements
    Operation<Contexte,ResultatConges,Agent>{

  public void apply( Contexte ctxt, ResultatConges result, Agent agent){
    if(agent.estUltraMarin()){
       result.setNombreDeJours(result.getNombreDeJours() + 15);
    }
  }
}
```

  2. **Modification du fichier de configuration**

```
bean.id.3=nouvelleOperation
```

  3. **Modification du source initial …**

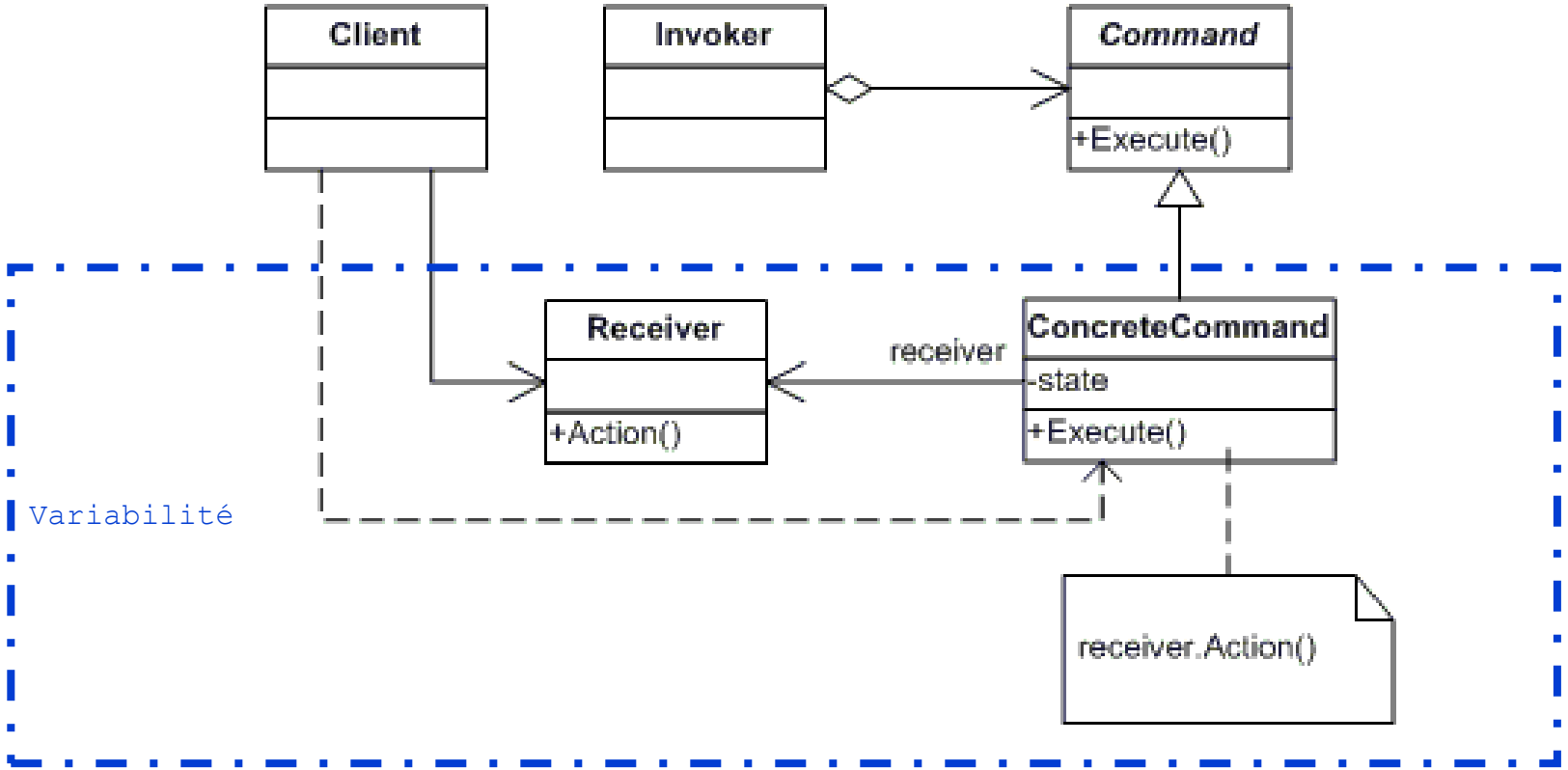
```
Conge conge = container.getBean("nouvelleOperation");
```

Le nom de la nouvelle opération/stratégie pourrait être dans un fichier, cf. le patron Factory

Démonstration / discussions

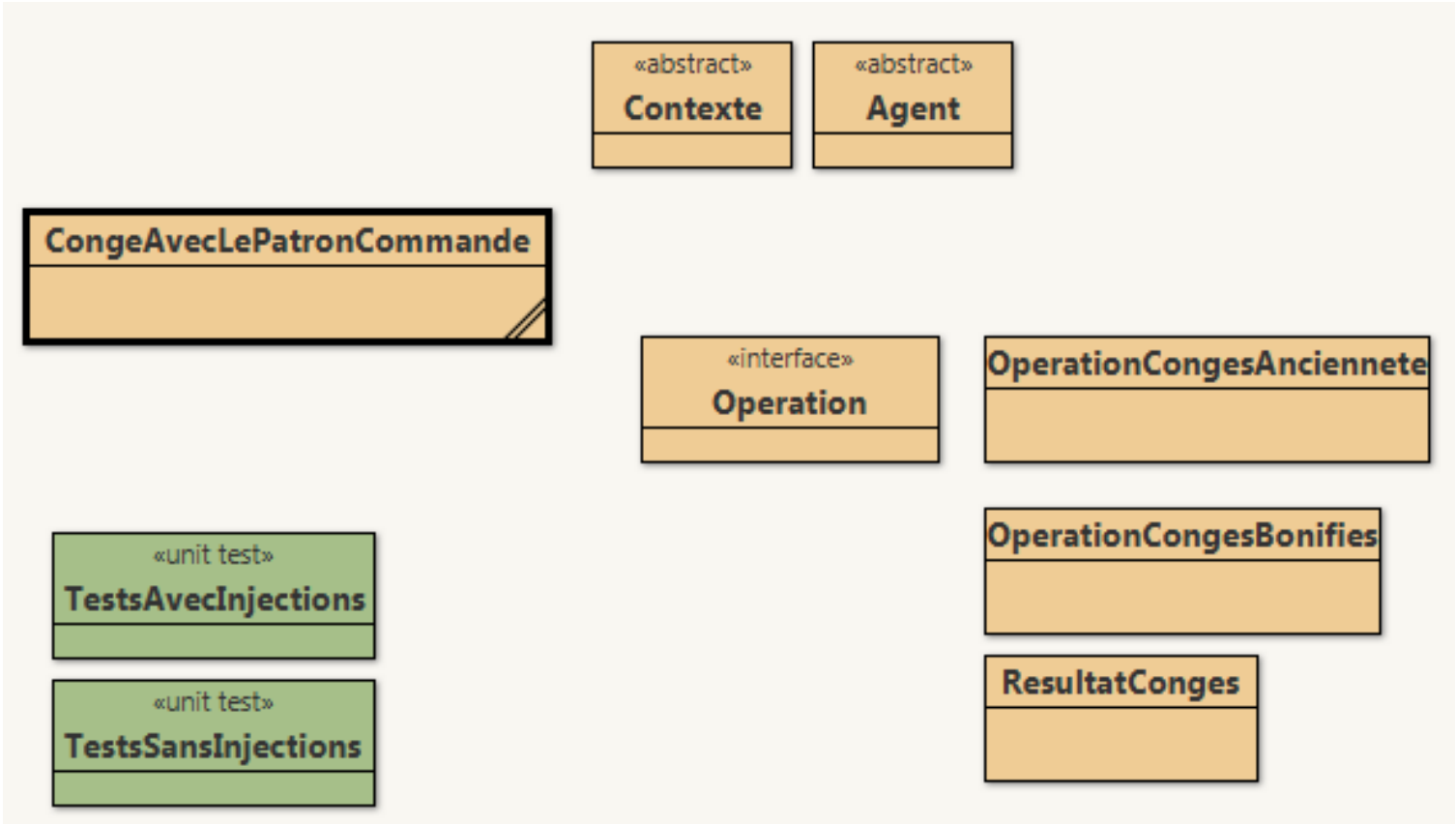
- **Une nouvelle stratégie ?**

Le patron Commande



- **Ici le calcul des congés correspond à l'exécution de toutes les commandes**
  - **Nouveau calcul = une nouvelle commande**
  - **Une modification de la configuration et seulement**

Les congés et le patron Commande



- **Une nouvelle fonctionnalité,**
  - **une nouvelle opération, une nouvelle commande**
- **Ici les commandes sont cumulées**

Les congés avec le patron Commande
MacroCommande ici, cf. la boucle foreach

```
public class CongeAvecLePatronCommande{
  private List<Operation<Contexte,ResultatConges,Agent>> operations;
  private Contexte contexte;

  public void setOperation(Operation<Contexte,ResultatConges,Agent> opr){
     this.operations.add(opr);
   }

   public ResultatConges calcul(Agent agent)throws Exception{
     ResultatConges resultat = new ResultatConges();
    for(Operation<Contexte,ResultatConges,Agent> operation : operations){
       operation.apply(contexte, resultat, agent);
     }
     return resultat;
   }
}
```

Ici les commandes sont ici cumulées…

```
for(Operation<Contexte,ResultatConges,Agent> operation : operations){
  operation.apply(contexte, resultat, agent);
 }
```

Configuration, ici avec deux « commandes »

```
bean.id.1=congeAncienneteEtBonifie
congeAncienneteEtBonifie.class=injection_commande.CongeAvecLePatronCommande
congeAncienneteEtBonifie.property.1=contexte
congeAncienneteEtBonifie.property.1.param.1=null
congeAncienneteEtBonifie.property.2=operation
congeAncienneteEtBonifie.property.2.param.1=operationCongesAnciennete
congeAncienneteEtBonifie.property.3=operation
congeAncienneteEtBonifie.property.3.param.1=operationCongesBonifies

bean.id.2=operationCongesAnciennete
operationCongesAnciennete.class=injection_commande.OperationCongesAnciennete

bean.id.3=operationCongesBonifies
operationCongesBonifies.class=injection_commande.OperationCongesBonifies

bean.id.4=congeBonifie
congeBonifie.class=injection_commande.CongeAvecLePatronCommande
congeBonifie.property.1=contexte
congeBonifie.property.1.param.1=null
congeBonifie.property.2=operation
congeBonifie.property.2.param.1=operationCongesBonifies
```

Un usage en java

```
ApplicationContext container =
  Factory.createApplicationContext("injection_commande/README.TXT");


CongeAvecLePatronCommande conge = container.getBean("congeAncienneteEtBonifie");

    ResultatConges result = new ResultatConges();
    Agent agent = …

    assertEquals(65,conge.calcul(agent).getNombreDeJours());

   CongeAvecLePatronCommande conge2 = container.getBean("congeBonifie");

    assertEquals(15,conge2.calcul(agent).getNombreDeJours());
```

Patron commande

- **Une nouvelle fonctionnalité :**

1. **Ajout de la nouvelle Operation en java**

```
public class OperationCongesBonifies implements
    Operation<Contexte,ResultatConges,Agent>{
  public void apply( Contexte ctxt, ResultatConges result, Agent agent){
    if(agent.estUltraMarin() && nouveau test ){
      result.setNombreDeJours(result.getNombreDeJours() + 15);
    }
  }
}
```

2. **Modification du fichier de configuration**

```
bean.id.3=nouvelleOperation
```

3. **Source inchangé**

Configuration, ici avec trois « commandes »

```
bean.id.1=congeAncienneteEtBonifie
congeAncienneteEtBonifie.class=injection_commande.CongeAvecLePatronCommande
congeAncienneteEtBonifie.property.1=contexte
congeAncienneteEtBonifie.property.1.param.1=null
congeAncienneteEtBonifie.property.2=operation
congeAncienneteEtBonifie.property.2.param.1=operationCongesAnciennete
congeAncienneteEtBonifie.property.3=operation
congeAncienneteEtBonifie.property.3.param.1=operationCongesBonifies
congeAncienneteEtBonifie.property.4=operation
congeAncienneteEtBonifie.property.4.param.1=operationCongesSupplementaires

bean.id.2=operationCongesAnciennete
operationCongesAnciennete.class=injection_commande.OperationCongesAnciennete

bean.id.3=operationCongesBonifies
operationCongesBonifies.class=injection_commande.OperationCongesBonifies


bean.id.5=operationCongesSupplementaires
operationCongesBonifies.class=injection_commande.OperationCongesSupplementaires
```

Usage en java, aucune modification du source mais des jours supplémentaires …

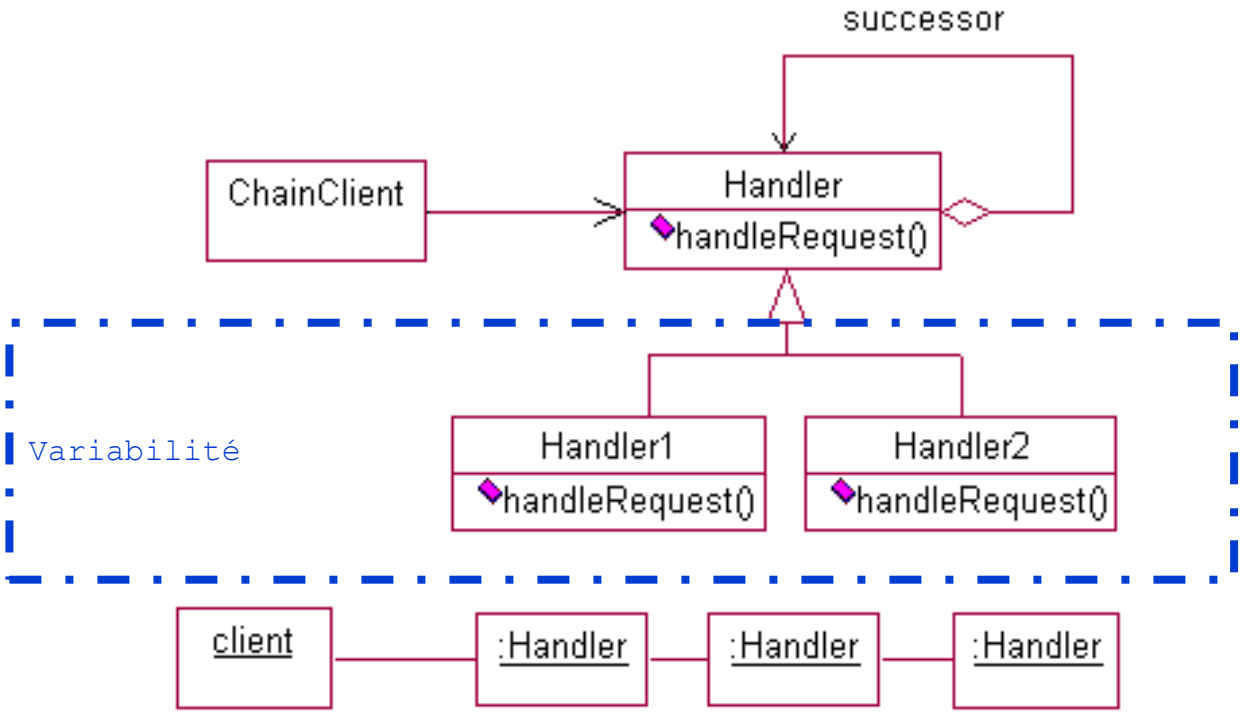
```
ApplicationContext container =
  Factory.createApplicationContext("injection_commande/README.TXT");
    CongeAvecLePatronCommande conge = container.getBean("congeAncienneteEtBonifie");


    ResultatConges result = new ResultatConges();
    Agent agent = …

  assertEquals(75,conge.calcul(agent).getNombreDeJours());


    CongeAvecLePatronCommande conge2 = null;
    conge2 = container.getBean("congeBonifie");

  assertEquals(15,conge2.calcul(agent).getNombreDeJours());
```

Démonstration/discussions

- **Une nouvelle commande ?**

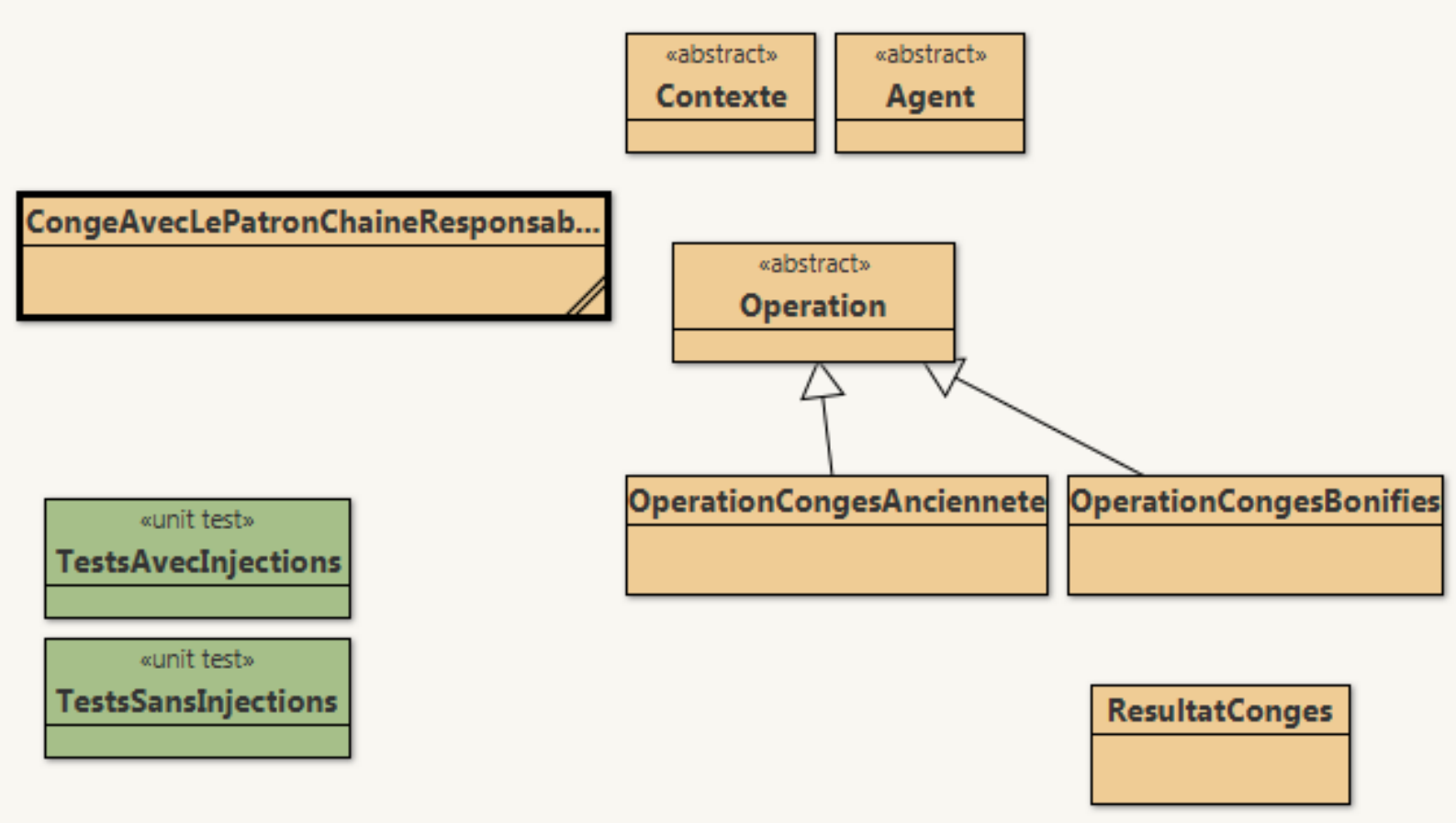
- **Un invocateur ? Avec un Memento ?**

Le patron Chaîne de responsabilités(CoR)



- **Le calcul des congés correspond à l'exécution de certains ou de tous les maillons(Handler) de la chaîne**
  - **Nouveau calcul = un nouveau maillon (Handler)**
    - **La propagation peut être interrompue par l'un des maillons**
  - **Une modification de la configuration et seulement**

Les congés et le patron CoR



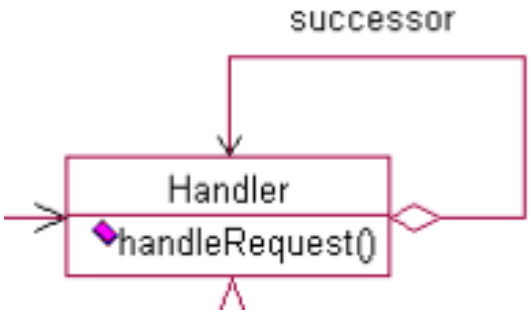
- **Une nouvelle fonctionnalité, une nouvelle opération**
- **Un nouveau maillon**

Les congés avec le patron CoR

```
public class CongeAvecLePatronChaineResponsabilites{
  private Operation<Contexte,ResultatConges,Agent> operation;
  private Contexte contexte;
  public CongeAvecLePatronChaineResponsabilites(){}
  public CongeAvecLePatronChaineResponsabilites(Contexte contexte){
    this.contexte = contexte;
  }

  public void setContexte(Contexte contexte){
    this.contexte = contexte;
  }

  public void setOperation(Operation<Contexte,ResultatConges,Agent> operation){
    this.operation = operation; // la première opération de la chaine
  }

  public ResultatConges calcul(Agent agent)throws Exception{
    ResultatConges resultat = new ResultatConges();
    boolean b = operation.apply(contexte, resultat, agent);
    return resultat;
  }
}
```

L'Operation <C, R, E> a évoluée

- **Quelque soit les futures « opérations »**
- **Un résultat R dans un certain contexte C avec une entité E**




```
public abstract class Operation<C, R, E>{
 public boolean apply( C ctxt, R result, E entity) throws Exception{
   if (successor != null)
     return successor.apply(ctxt, result, entity);
    else
     return false;
  }

  private Operation<C, R, E> successor;

  public void setSuccessor(final Operation<C, R, E> successor){
    this.successor = successor;
  }
}
```

Une opération concrète par fonctionnalité (les ***Handler*** du patron original)

Les congés dus à l'ancienneté comme maillon

```
public class OperationCongesAnciennete extends
  Operation<Contexte,ResultatConges,Agent>{
  public boolean apply( Contexte ctxt, ResultatConges resultat,
  Agent agent) throws Exception{

// les congés dus à l'ancienneté, mais seulement au bout de 3 ans …
    if(agent.anciennete()>=3){
      resultat.setNombreDeJours(resultat.getNombreDeJours()
                                + agent.anciennete()*5);
    }

    return super.apply(ctxt, resultat, agent);
  }
}
```

Propagation vers le successeur dans tous les cas

Configuration, ici avec deux « maillons »

```
bean.id.1=chaineConges
chaineConges.class=injection_chaine_responsabilites.CongeAvecLePatronChaineResponsabilites
chaineConges.property.1=contexte
chaineConges.property.1.param.1=null
chaineConges.property.2=operation
chaineConges.property.2.param.1=operationCongesAnciennete


bean.id.2=operationCongesAnciennete
operationCongesAnciennete.class=injection_chaine_responsabilites.OperationCongesAnciennete
operationCongesAnciennete.property.1=successor
operationCongesAnciennete.property.1.param.1=operationCongesBonifies


bean.id.3=operationCongesBonifies
operationCongesBonifies.class=injection_chaine_responsabilites.OperationCongesBonifies
operationCongesBonifies.property.1=successor
operationCongesBonifies.property.1.param.1=null
```

operationCongesAnciennete → operationCongesBonifies → null

Usage en java

```
 ApplicationContext container =
Factory.createApplicationContext("injection_chaine_responsabilites/README.TXT");
   CongeAvecLePatronChaineResponsabilites conge = null;
   conge = container.getBean("chaineConges");

   ResultatConges result = new ResultatConges();
   Agent agent = new Agent(){
     public boolean estUltraMarin(){return true;}
     public int anciennete(){return 10;}
     public String toString(){return "un agent ultramarin, 10 ans";}
   };

   assertEquals(65,conge.calcul(agent).getNombreDeJours());

   CongeAvecLePatronChaineResponsabilites conge2 = null;
   conge2 = container.getBean("chaineConges2");

   result = new ResultatConges();
   assertEquals(15,conge2.calcul(agent).getNombreDeJours());
```

Patron Chaîne de responsabilités

- **Une nouvelle fonctionnalité :**
  1. **Ajout de la nouvelle Operation en java**

```
public class OperationCongesBonifies extends Operation<Contexte,ResultatConges,Agent>{

 public boolean apply( Contexte ctxt, ResultatConges result, Agent agent) throws Exception{

  if(agent.estUltraMarin()){
   // c'est un forfait de congés ! 15 jours et c'est tout... return true
   result.setNombreDeJours(result.getNombreDeJours() + 15);
   return true;

  }else{
   return super.apply(ctxt, result, agent);
  }
 }
}
```

  2. **Modification du fichier de configuration**

```
bean.id.3=nouvelleOperation
//Renseigner le champ successeur
```

  3. **Source inchangé**

Modification de la configuration

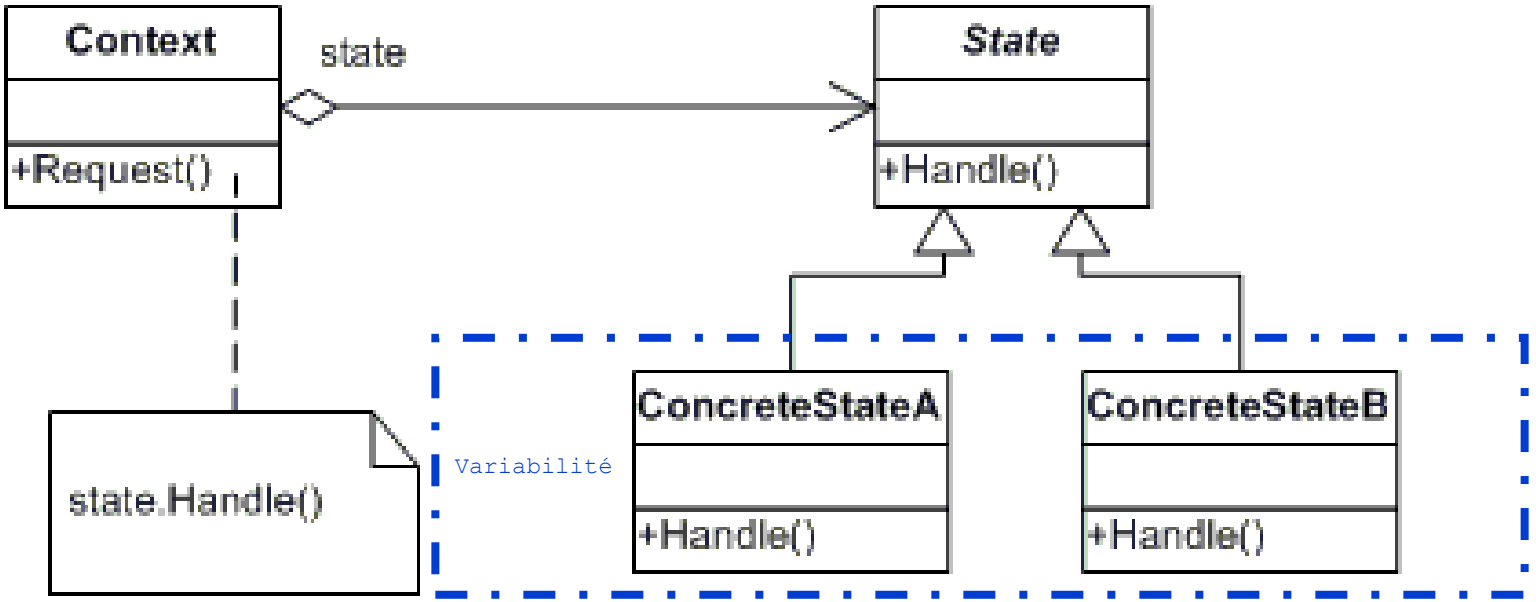
```
bean.id.1=chaineConges
chaineConges.class=injection_chaine_responsabilites.CongeAvecLePatronChaineResponsabilites
chaineConges.property.1=contexte
chaineConges.property.1.param.1=null
chaineConges.property.2=operation
chaineConges.property.2.param.1=operationCongesAnciennete

bean.id.2=operationCongesAnciennete
operationCongesAnciennete.class=injection_chaine_responsabilites.OperationCongesAnciennete
operationCongesAnciennete.property.1=successor
operationCongesAnciennete.property.1.param.1=operationCongesBonifies

bean.id.3=operationCongesBonifies
operationCongesBonifies.class=injection_chaine_responsabilites.OperationCongesBonifies
operationCongesBonifies.property.1=successor
operationCongesBonifies.property.1.param.1=operationCongesExceptionnels

bean.id.4=operationCongesExceptionnels
operationCongesBonifies.class=injection_chaine_responsabilites.OperationCongesExceptionnels
operationCongesBonifies.property.1=successor
operationCongesBonifies.property.1.param.1=null
```

operationCongesAnciennete -> operationCongesBonifies -> **operationCongesExceptionnels** → null

Démonstration / discussions ?

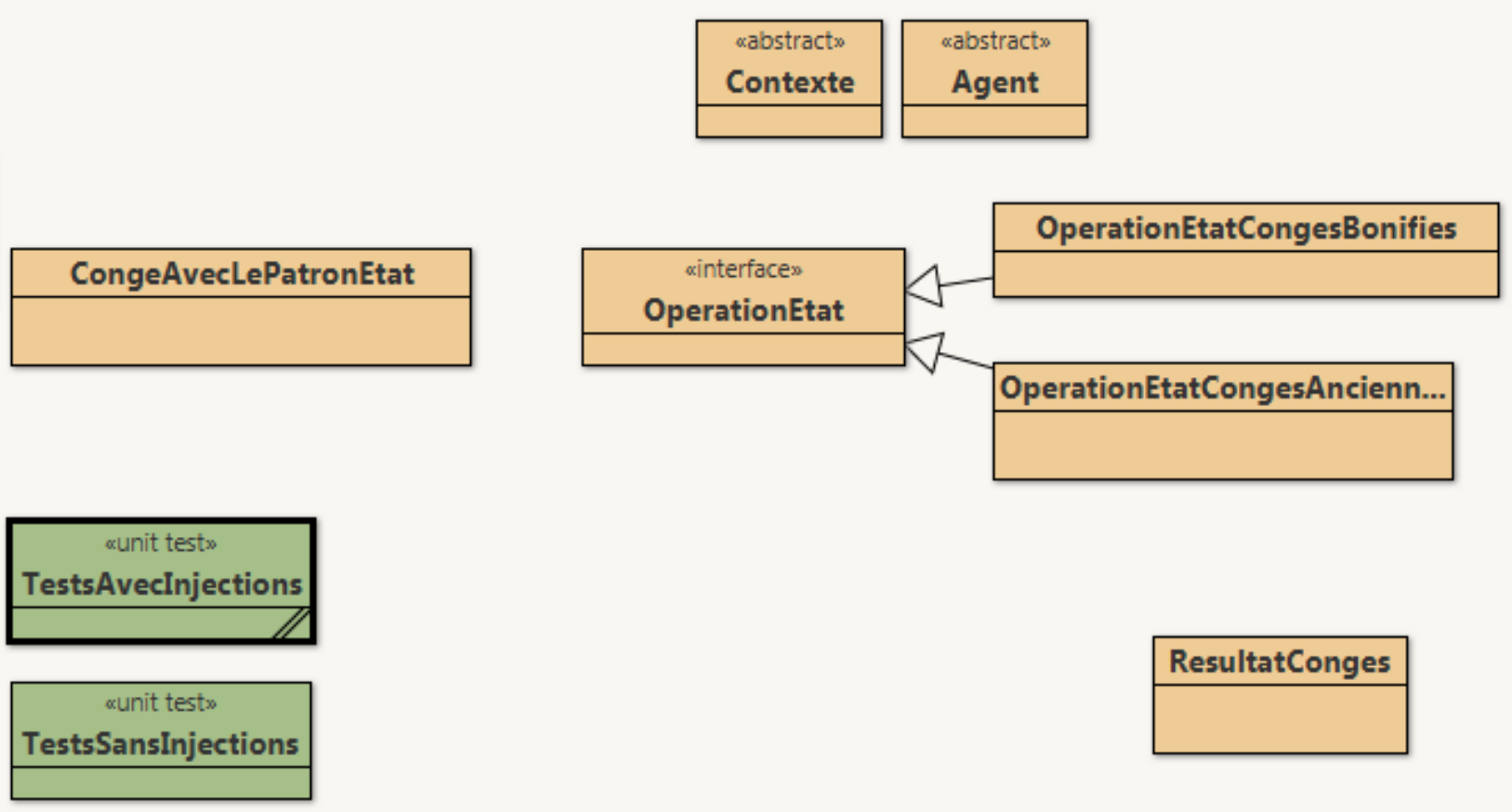
- **Une nouvelle fonctionnalité qui interrompt la chaîne ?**

Le patron Etat



- **Le calcul des congés correspond à l'exécution du diagramme d'états**
  - **Nouveau calcul = un nouvel état**
  - **Une modification de la configuration et seulement**
  - **Chaque état est autonome et décide du prochain état**
    - **Apparenté Stratégie sauf que ce n'est pas le client qui décide**

Les congés et le patron Etat



- **Une nouvelle fonctionnalité, un nouvel état**
- **A chaque appel un changement d'état**
  - **Est-ce approprié pour un calcul ?**

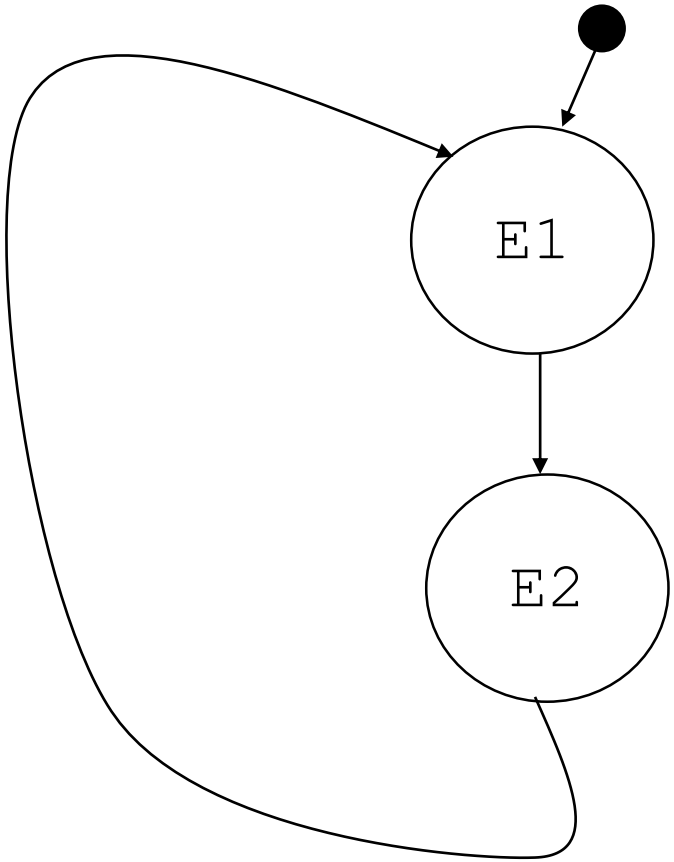
Les congés avec le patron état

```
public class CongeAvecLePatronEtat{
  // état courant (opération courante)
  private OperationEtat<Contexte,ResultatConges,Agent> operationEtat;
  private Contexte contexte;
  public CongeAvecLePatronEtat(){}

  public CongeAvecLePatronEtat(Contexte contexte){
    this.contexte = contexte;
  }

  public void setContexte(Contexte contexte){
    this.contexte = contexte;
  }

  public void setOperationEtat(
  OperationEtat<Contexte,ResultatConges,Agent> operationEtat){
    this.operationEtat = operationEtat;
  }

  public OperationEtat<Contexte,ResultatConges,Agent> getOperationEtat( ){
    return this.operationEtat;
  }
}
```

Configuration, ici avec deux « états »

```
bean.id.1=congeAvecLePatronEtat
congeAvecLePatronEtat.class=injection_etat.CongeAvecLePatronEtat
congeAvecLePatronEtat.property.1=contexte
congeAvecLePatronEtat.property.1.param.1=null
congeAvecLePatronEtat.property.2=operationEtat
congeAvecLePatronEtat.property.2.param.1=operationEtatCongesAnciennete

bean.id.2=operationEtatCongesAnciennete
operationEtatCongesAnciennete.class=injection_etat.OperationEtatCongesAnciennete
operationEtatCongesAnciennete.property.1=conge
operationEtatCongesAnciennete.property.1.param.1=congeAvecLePatronEtat
operationEtatCongesAnciennete.property.2=operationEtat
operationEtatCongesAnciennete.property.2.param.1=operationEtatCongesBonifies

bean.id.3=operationEtatCongesBonifies
operationEtatCongesBonifies.class=injection_etat.OperationEtatCongesBonifies
operationEtatCongesBonifies.property.1=conge
operationEtatCongesBonifies.property.1.param.1=congeAvecLePatronEtat
operationEtatCongesBonifies.property.2=operationEtat
operationEtatCongesBonifies.property.2.param.1=operationEtatCongesAnciennete
```

Automate à états



operationEtatCongesAnciennete

operationEtatCongesBonifies

- **Transition effectuée à chaque appel**
  - **L'état courant est affecté**

Usage en java

```
ApplicationContext ctx =
  Factory.createApplicationContext("injection_etat/README.TXT");
   CongeAvecLePatronCommande conge =  ctx.getBean("congeAvecLePatronEtat");


    Agent agent = …

   ResultatConges result = new ResultatConges();

   // en fonction de l'état courant
   conge.getOperationEtat().apply(contexte, result, agent);
   assertEquals(50,result.getNombreDeJours());

 // en fonction de l'état courant
   result = new ResultatConges();
   conge.getOperationEtat().apply(contexte, result, agent);
   assertEquals(15,result.getNombreDeJours());
```

Patron état

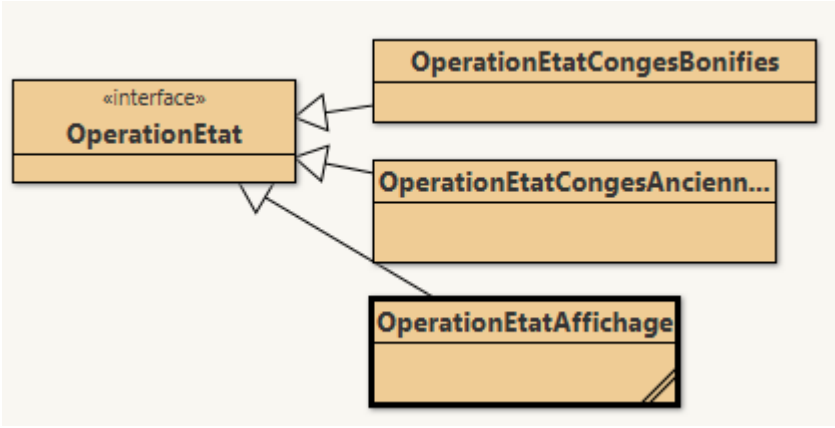
- **Une nouvelle fonctionnalité :**
  1. **Ajout d'un nouvel état en java**

```
public class OperationEtatAffichage implements OperationEtat<Contexte,ResultatConges,Agent>{
  private CongeAvecLePatronEtat conge;
  private OperationEtat<Contexte,ResultatConges,Agent> operationEtat;

  public OperationEtatAffichage(){}
  public void setConge(CongeAvecLePatronEtat conge){
    this.conge = conge;
  }
  public void setOperationEtat(final OperationEtat<Contexte,ResultatConges,Agent> operationEtat){
    this.operationEtat = operationEtat;
  }
  public void apply( Contexte ctxt, ResultatConges resultat, Agent agent) throws Exception{
    System.out.println(this + "resultat: " + resultat);
    conge.setOperationEtat(operationEtat);
  }
```

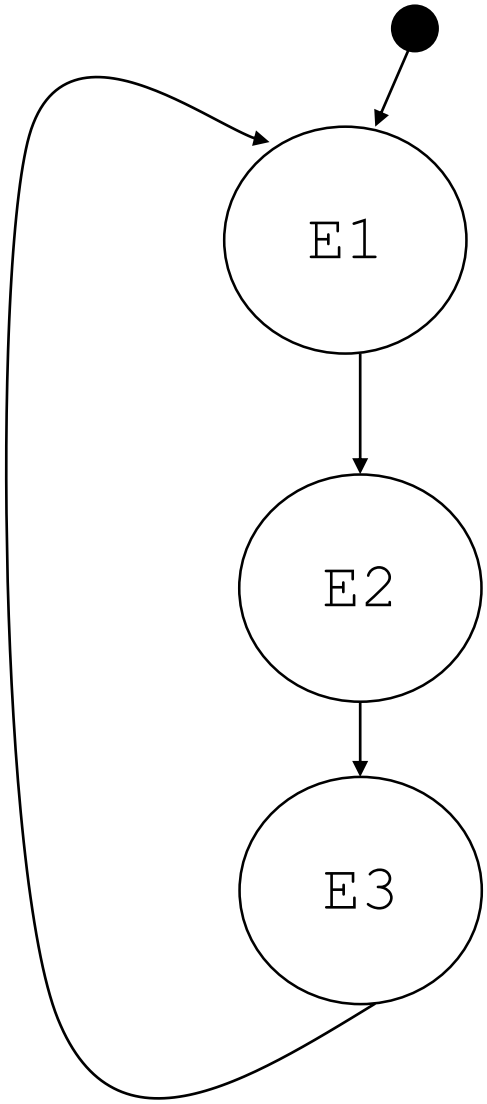
  2. **Modification du fichier de configuration**

```
bean.id.4=operationEtatAffichage
```

  3. **Source inchangé**



Automate à 3 états



operationEtatCongesAnciennete

operationEtatCongesBonifies

operationEtatAffichage

- **Transition effectuée à chaque appel**

Configuration, ici avec trois « états »

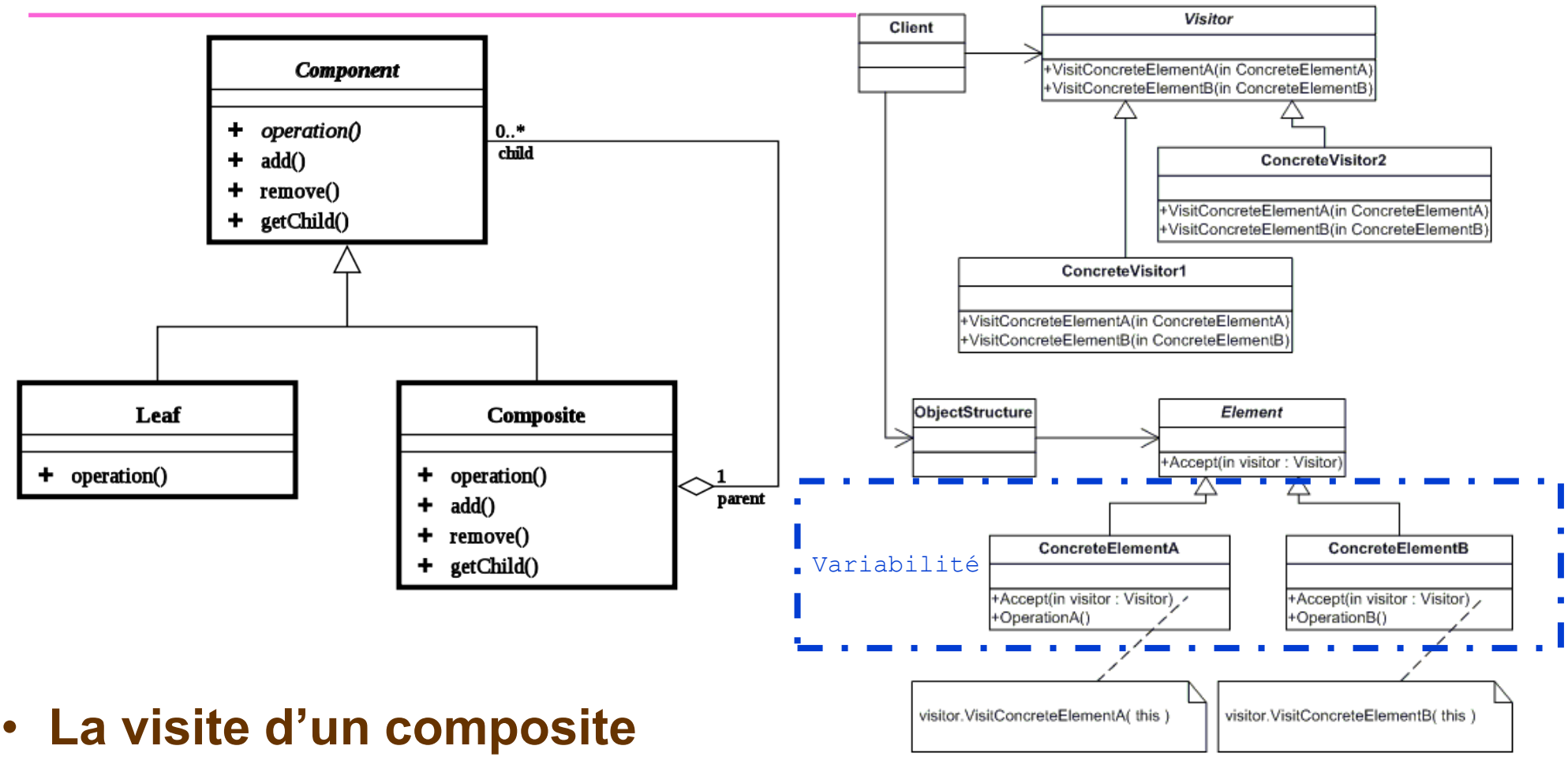
```
bean.id.1=congeAvecLePatronEtat
congeAvecLePatronEtat.class=injection_etat.CongeAvecLePatronEtat
congeAvecLePatronEtat.property.1=contexte
congeAvecLePatronEtat.property.1.param.1=mairie
congeAvecLePatronEtat.property.2=operationEtat
congeAvecLePatronEtat.property.2.param.1=operationEtatCongesAnciennete

bean.id.2=operationEtatCongesAnciennete
operationEtatCongesAnciennete.class=injection_etat.OperationEtatCongesAnciennete
operationEtatCongesAnciennete.property.1=conge
operationEtatCongesAnciennete.property.1.param.1=congeAvecLePatronEtat
operationEtatCongesAnciennete.property.2=operationEtat
operationEtatCongesAnciennete.property.2.param.1=operationEtatCongesBonifies

bean.id.3=operationEtatCongesBonifies
operationEtatCongesBonifies.class=injection_etat.OperationEtatCongesBonifies
operationEtatCongesBonifies.property.1=conge
operationEtatCongesBonifies.property.1.param.1=congeAvecLePatronEtat
operationEtatCongesBonifies.property.2=operationEtat
operationEtatCongesBonifies.property.2.param.1=operationEtatAffichage

bean.id.4=operationEtatAffichage
operationEtatAffichage.class=injection_etat.OperationEtatAffichage
operationEtatAffichage.property.1=conge
operationEtatAffichage.property.1.param.1=congeAvecLePatronEtat
operationEtatAffichage.property.2=operationEtat
operationEtatAffichage.property.2.param.1=operationEtatCongesAnciennete
```

Etat E1

Etat E2

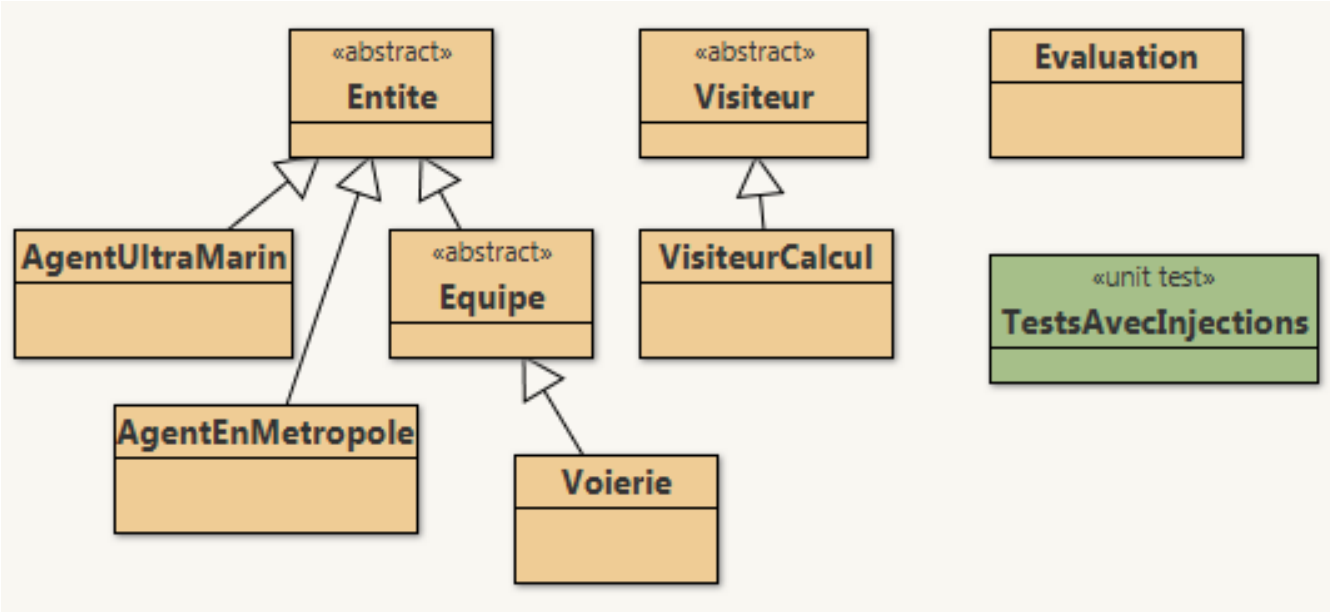
Etat E3

Le patron Composite et ses Visiteurs



- **La visite d'un composite**
  - **Le composite décrit les équipes municipales**
  - **Le parcours, du composite est effectuée par un ou plusieurs visiteurs**
    - **Visiteurs comme Points de variabilité**

Le patron Visiteur et les congés



- **L'équipe de la voierie, le cumul des congés de tous les agents**
  - **Juste pour l'exemple**
  - **Le patron Template Methode est utilisé…**

```
ApplicationContext ctx = null;
ctx =  Factory.createApplicationContext("injection_visiteur/README.TXT");
Evaluation eval =  ctx.getBean("eval");
assertEquals(35,eval.calculer());
```

Configuration

```
bean.id.1=eval
eval.class=injection_visiteur.Evaluation
eval.property.1=visiteur
eval.property.1.param.1=visiteur  // variabilité du visiteur
eval.property.2=entite
eval.property.2.param.1=voierie   // variabilité de l'entité
#eval.property.2.param.1=environnement

bean.id.2=visiteur
visiteur.class=injection_visiteur.VisiteurCalcul // ici un calcul

bean.id.3=voierie
template1.class=injection_visiteur.Voierie

bean.id.4=template2
template2.class=injection_visiteur.Environnement
```

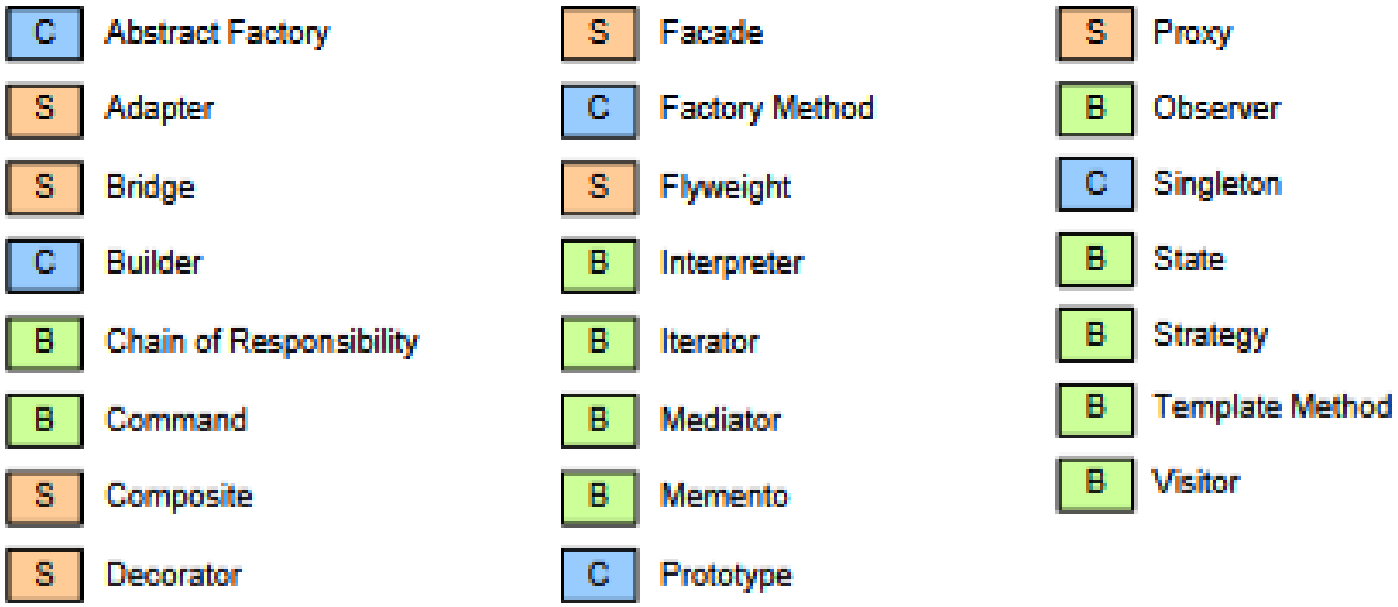
Une nouvelle fonctionnalité

- **Nouvelle fonctionnalité : un nouveau visiteur**
  - **Seule la configuration est concernée**
- **Un nouveau nœud au composite**
  - **Plus délicat, tous les visiteurs doivent être revus…**
  - **Cf. une solution par introspection**
    - **http://www.javaworld.com/article/2077602/learn-java/java-tip-98--reflect-on-the-visitor-design-pattern.html**

Bilan intermédiaire Variabilité

- **Patrons + conteneur : fructueux**
- **Ajout d'une fonctionnalité revient à créer une classe**
  - **Implémentant une interface ou héritant d'une classe existantes**
    - **Suivie d'une modification dans le fichier de configuration**
  - **Pas d'impact sur le source Java**
    - **Les noms des beans pourraient être dans un fichier dédié**
- **Le coût d'exécution dû à l'introspection est reporté dans la phase d'initialisation, une seule fois, négligeable…**

Quels patrons ?



- **Patron + injection**
  - **Tous candidats ?**

- **http://jfod.cnam.fr/NFP121/supports/extras_designpatternscard.pdf**

Pattern + un outil d'injection les bons candidats

- **Décorateur, Commande, Stratégie, Fabriques, TemplateMethode, Chaîne de responsabilités,Template Methode …**
  - **Tout patron utilisant une delégation vers une sous-classe peut utiliser avec profit un outil d'injection**

- **A voir**
  - **Poids Mouche, Singleton ?**
    - **Utile/inutile**

Démonstration en direct

- **Un patron dans la salle ?**
  - **Façade ?**
  - **Observateur ?**
  - **…**

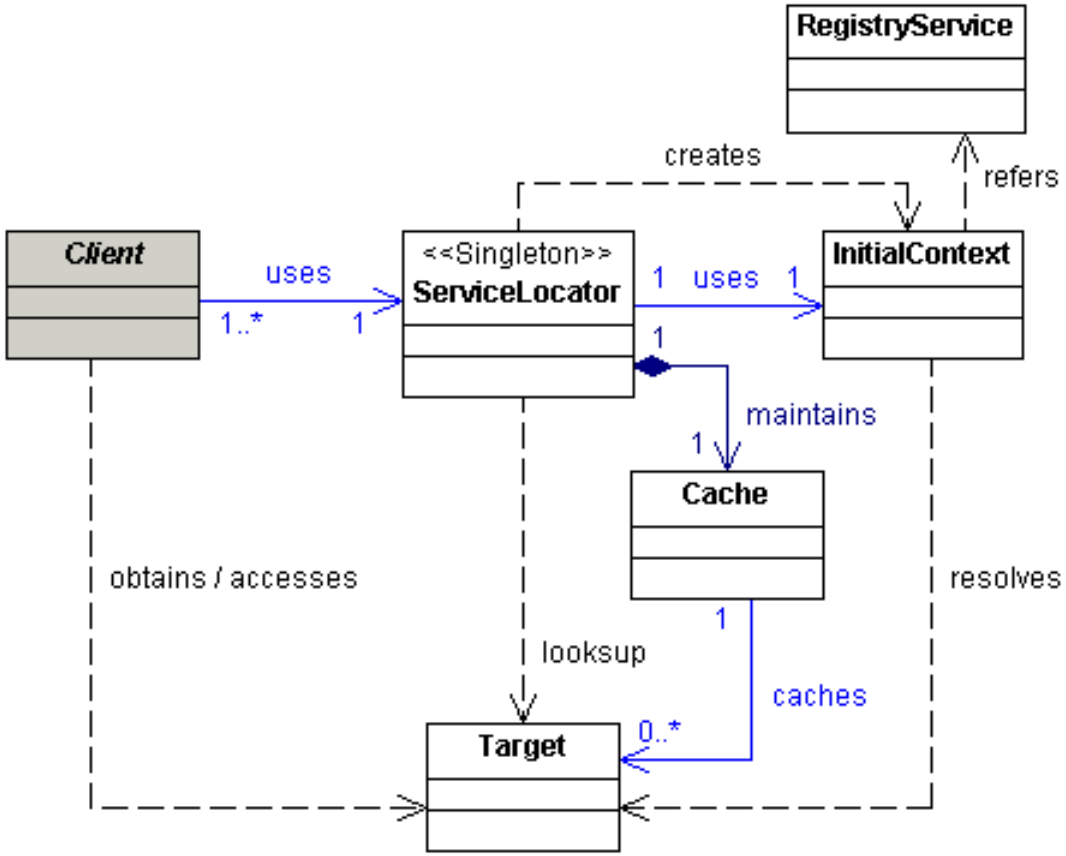
Critiques, toujours possibles

- **Design pattern + Framework**
- Design pattern ? De trop bas niveau ?
- **Vers une généralisation ?**
  - **Le patron Service Locator**
  - **Approche par composants**
    - **Publish/Subscribe**
  - **Composant comme service ? Micro-services ?**

Approche services: un patron tout prêt

- **Le patron Service Locator**
  - Localiser, sélectionner un service selon les besoins d'une application
    - accès de manière uniforme
      - Par exemple (DNS, LDAP, JNDI API)
  - Une table des services, un espace de nommage
    - Avec en interne une optimisation des accès en utilisant un cache,
    - Implémentée par un Singleton en général …

Patron ServiceLocator



- **Source : http://www.corej2eepatterns.com/Patterns2ndEd/ServiceLocator.htm**

Exemple : le service de calcul des congés

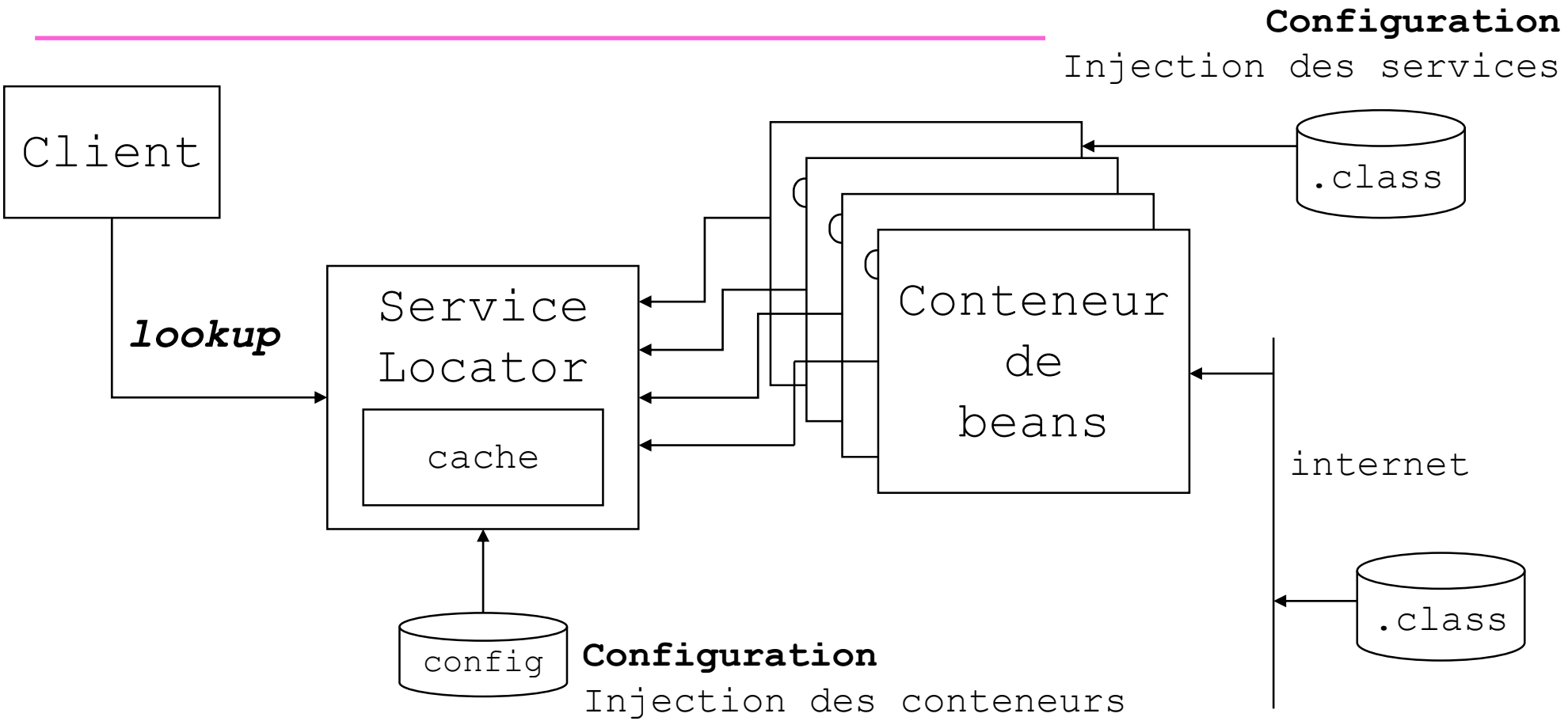
- **ServiceLocator**
  - Nul ne sait où se trouve l'implémentation…
    - En interne une collection de conteneurs
  - Accès simple pour les clients
    - lookup(un nom)

- **Deux méthodes**

```
import container.ApplicationContext;
public interface ServiceLocatorI{

  <T> T lookup(String serviceName) throws Exception;

  boolean addContainer(ApplicationContext container) throws Exception;
}
```

  - Ajout d'un conteneur
    - Les classes peuvent issues du web et/ou intranet
      - Cf. URLClassLoader

Un service locator adapté…



- **Ici, une agrégation de conteneurs**
- **Le client demande un service, peu importe d'où il vient, quel est le conteneur propriétaire du service…**

ServiceLocator : une collection de conteneurs

- **En exemple**

- **Un service locator avec 2 conteneurs**
  - **Calculs des congés**
    1. **Avec le patron stratégie**
    2. **Avec le patron chaîne de responsabilités**

Avons-nous les mêmes jours de congés ? Selon les deux méthodes (…)

- **Une configuration du service locator**

Configuration …cf femtoContainer

```
bean.id.1=serviceLocator
serviceLocator.class=injection_service_locator.ServiceLocator
serviceLocator.property.1=container
serviceLocator.property.1.param.1=strategie
serviceLocator.property.2=container
serviceLocator.property.2.param.1=chaine

bean.id.2=strategie
strategie.class=container.FileSystemPropsApplicationContext
strategie.property.1=fileName
strategie.property.1.param.1=injection_strategie/README.TXT

bean.id.3=chaine
chaine.class=container.FileSystemPropsApplicationContext
chaine.property.1=fileName
chaine.property.1.param.1=injection_chaine_responsabilites/README.TXT

# Deux conteneurs ou plus ..
```

Tests unitaires

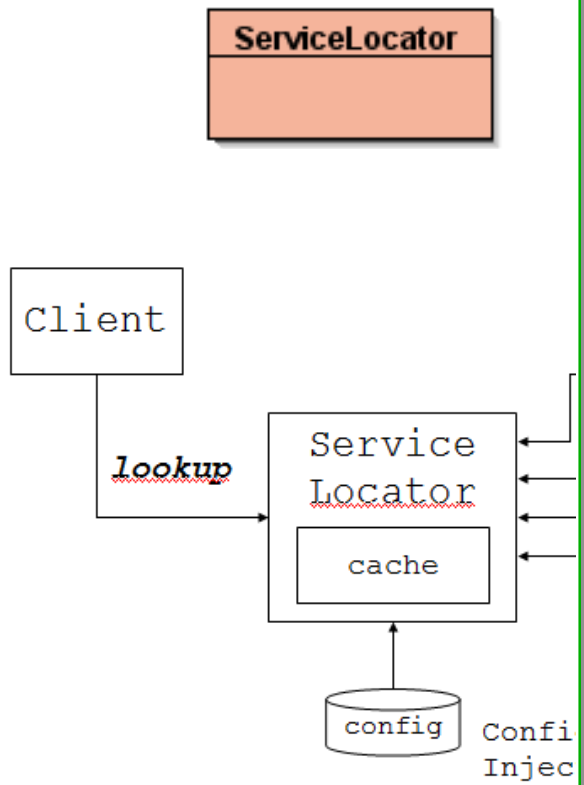
```
public void testServiceLocator() throws Exception{
   ApplicationContext ctx =
  Factory.createApplicationContext("injection_service_locator/README.TXT");
   ServiceLocatorI serviceLocator = (ServiceLocatorI) ctx.getBean("serviceLocator");


   CongeAvecLePatronStrategie conge = serviceLocator.lookup("congeAnciennete");
   assertEquals(50,conge.calcul(agent).getNombreDeJours());

   conge = serviceLocator.lookup("congeBonifie");
   assertEquals(15,conge.calcul(agent).getNombreDeJours());



 CongeAvecLePatronChaineResponsabilites conge2 = serviceLocator.lookup("chaineConges");


 assertEquals(65,conge2.calcul(agent2).getNombreDeJours());
```

65 jours de congés quelque soit le patron utilisé, rassurant…

Modifications, ajout de fonctionnalités ? se reporter au conteneur concerné

Service Locator

```
public class ServiceLocator implements ServiceLocatorI{

  private List<ApplicationContext> containers;
  private Map<String,Object> cache;

  public ServiceLocator(){
    this.containers = new ArrayList<ApplicationContext>();
    this.cache = new HashMap<String,Object>();
  }

  public <T> T lookup(String serviceName) throws Exception{
    T t = (T) cache.get(serviceName);
    if(t==null){
      for(ApplicationContext container : containers){
        for( String bean : container){
          if(serviceName.equals(bean)){
            Object service = container.getBean(bean);
            cache.put(serviceName, service);
            return (T)service;
          }
        }
      }
      throw new Exception(" service not found");
    }
    return (T)cache.get(serviceName);
  }
```



Vers une approche composant

- **Approche par composants logiciels**
  - **Chaque composant est identifié par un nom dans une table**
  - **À ce nom lui est associé des opérations**
    - **Ces opérations acceptent des objets métiers et engendrent un résultat**
- **Clients et fournisseurs de services**
  - **Les services s'inscrivent auprès des fournisseurs**
  - **Les clients recherchent le service et déclenchent les opérations attenantes**
- **Variabilité du service, variabilité des opérations, …**

Vers une approche d'un composant souple

- **Points de variabilité**
  - Chaque point de variabilité est un nom dans la table
  - Ce nom reflète un composant utilisable comme un service
  - Le composant possède des fonctionnalités
    - Fonctionnalités, opérations extensibles par inscriptions/souscriptions
  - La sélection d'un service est faite auprès de la table avec un nom
- **Configuration des points de variabilité**
  - Dans un fichier de configuration
  - Les opérations sont injectées
  - Point besoin de reconfigurer, point besoin de modifier les sources

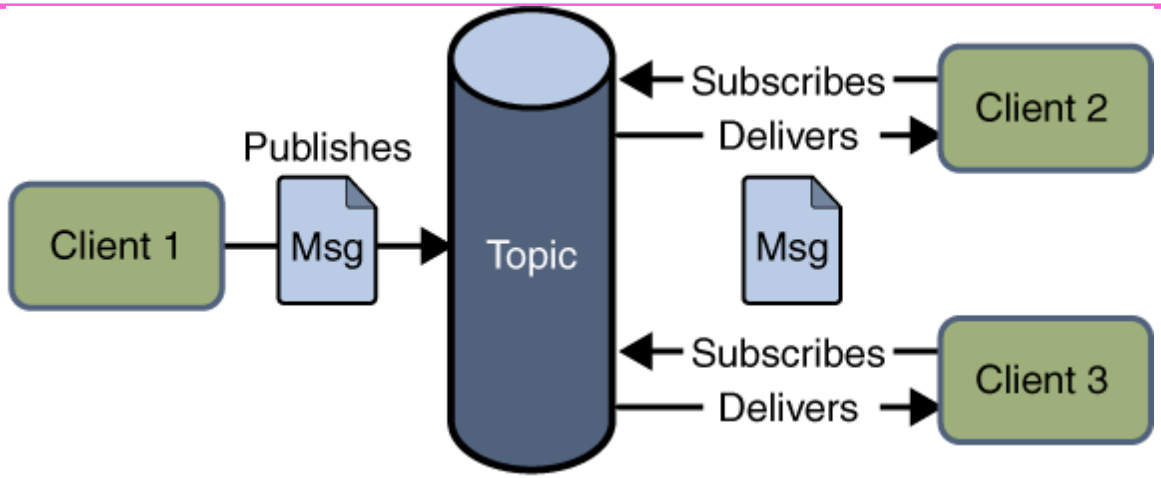
Framework de composants

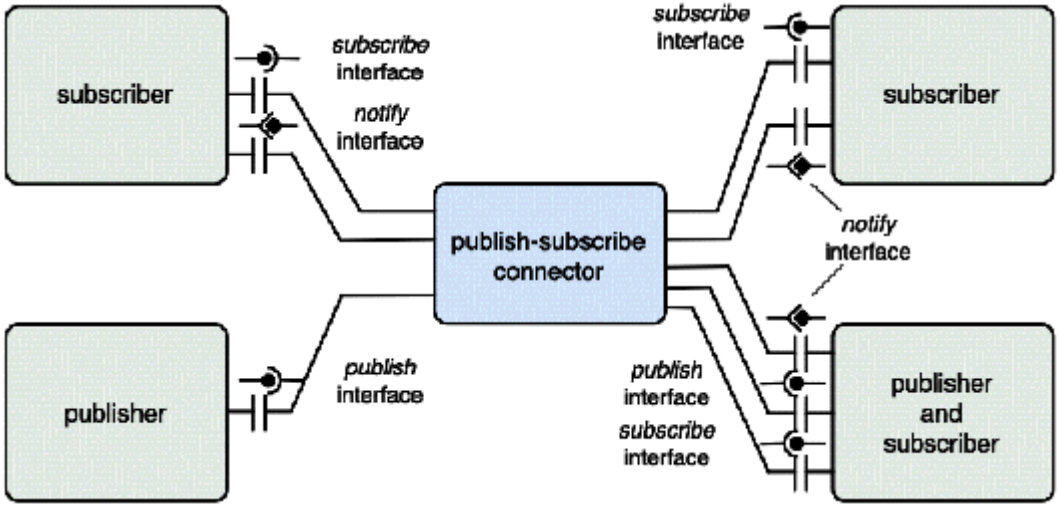
- **Proposition d'implémentation :**

- **Usage du patron publish/subscribe**
  - **Publish pour la recherche et l'exécution d'un service**
  - **Subscribe comme proposer une opération pour un service**

- **Nouveau service -> nouveau nom,**
- **Nouvelles fonctionnalités -> nouvelles souscriptions**

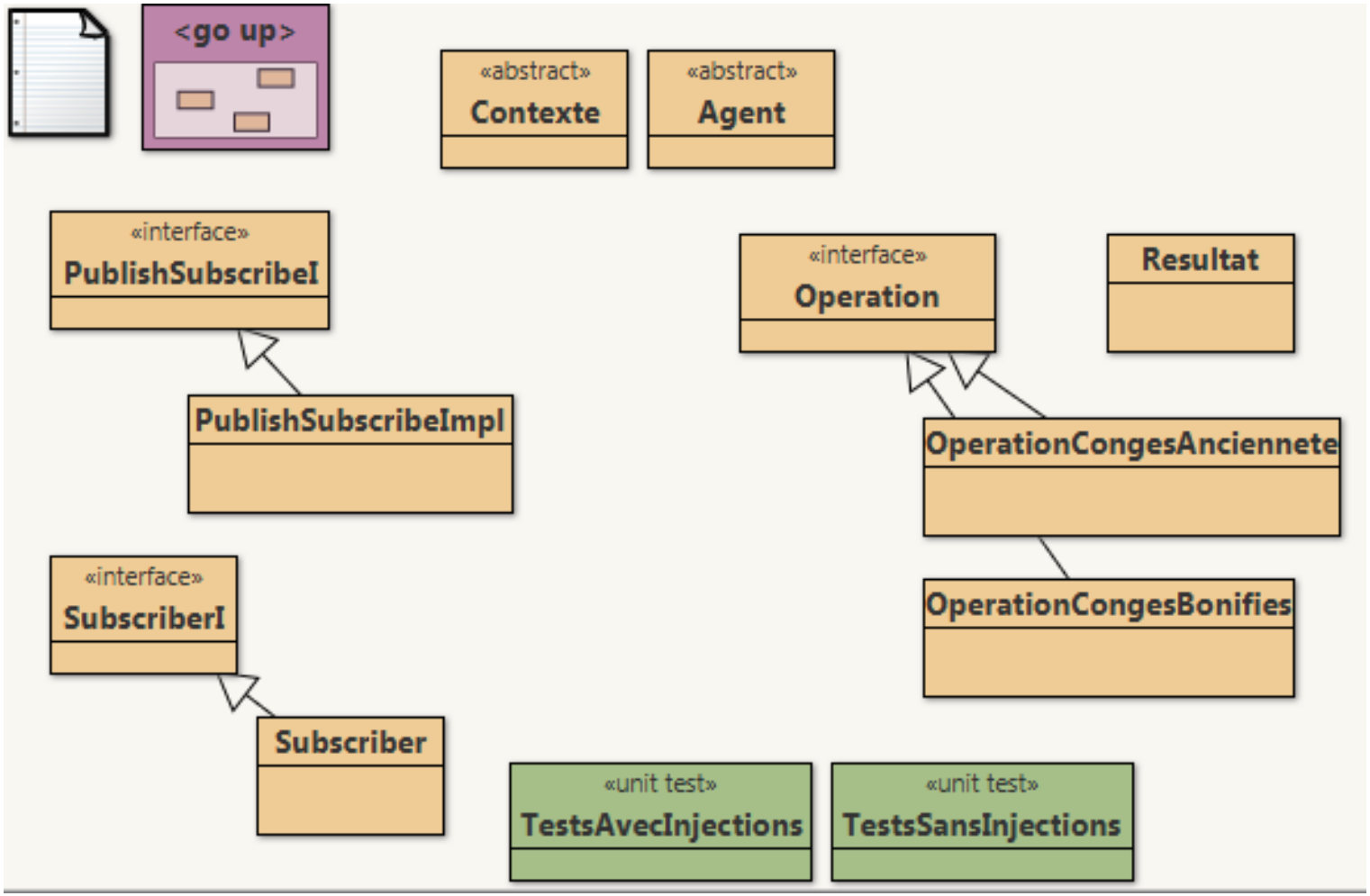
Publish/Subscribe





- http://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html
- http://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Compo/compo-body.html

PubSub + injection



- **Les fonctionnalités, opérations sont ici génériques …**
  - **interface Operation<Context, Result, Entity>{ …}**

Injection de l'ancienneté, la configuration

```
bean.id.1=pubsub
pubsub.class=PublishSubscribeImpl
pubsub.property.1=subcriber
pubsub.property.1.param.1=sub1


bean.id.2=sub1
sub1.class=Subscriber
sub1.property.1=name
sub1.property.1.param.1=anciennete
sub1.property.2=topic
sub1.property.2.param.1=conge
sub1.property.3=operation
sub1.property.3.param.1=operationCongesAnciennete

bean.id.3=operationCongesAnciennete
operationCongesAnciennete.class=OperationCongesAnciennete
```

Une autre fonctionnalité, un autre souscripteur

Calcul des congés ?

```
PublishSubscribeI pubsub;
pubsub = (PublishSubscribeI)container.getBean("pubsub");
Contexte ctxt = …
Agent agent = …
ResultatConges result = new ResultatConges();


pubsub.publish("conge",ctxt,result,agent);
assertEquals(50,result.getNombreDeJours());
// 50 jours pour cet agent
```

Injection des congés bonifiés, la configuration est enrichie

```
Une autre fonctionnalité, un autre souscripteur
bean.id.1=pubsub
pubsub.class=PublishSubscribeImpl
pubsub.property.1=subscriber
pubsub.property.1.param.1=sub1
pubsub.property.2=subscriber
pubsub.property.2.param.1=sub2

bean.id.4=sub2
sub2.class=Subscriber
sub2.property.1=name
sub2.property.1.param.1=bonifie
sub2.property.2=topic
sub2.property.2.param.1=conge
sub2.property.3=operation
sub2.property.3.param.1=operationCongesBonifies

bean.id.5=operationCongesBonifies
operationCongesBonifies.class=OperationCongesBonifies
```

Nouveau calcul des congés, pour un autre agent le même code java

```
PublishSubscribeI pubsub;
pubsub = (PublishSubscribeI)container.getBean("pubsub");
Contexte ctxt = …
Agent agent = …
ResultatConges result = new ResultatConges();


pubsub.publish("conge",ctxt,result,agent);
assertEquals(115,result.getNombreDeJours());
// 115 jours ! Pour cet agent des DOM-TOM


En annexe le même code sans femtoContainer
```

Un autre composant : le compte épargne temps

```
bean.id.1=pubsub
pubsub.class=PublishSubscribeImpl
pubsub.property.1=subcriber
pubsub.property.1.param.1=sub1

…

bean.id.5=sub1
sub1.class=Subscriber
sub1.property.1=name
sub1.property.1.param.1=epargne_temps
sub1.property.2=topic
sub1.property.2.param.1=compte_epargne_temps
sub1.property.3=operation
sub1.property.3.param.1=operationCompteEpargne

bean.id.6=operationCompteEpargne
```

Une autre fonctionnalité, un autre souscripteur

le compte épargne temps nouvelle fonctionnalité

```
bean.id.1=pubsub
pubsub.class=PublishSubscribeImpl
pubsub.property.1=subcriberEpargneTemps
pubsub.property.1.param.1=sub1
pubsub.property.2=subcriberEpargneTemps
pubsub.property.2.param.1=sub2


bean.id.5=sub1
# …
bean.id.7=sub2

bean.id.6=operationCompteEpargne
```

Une autre fonctionnalité, un autre souscripteur

La liste des souscripteurs

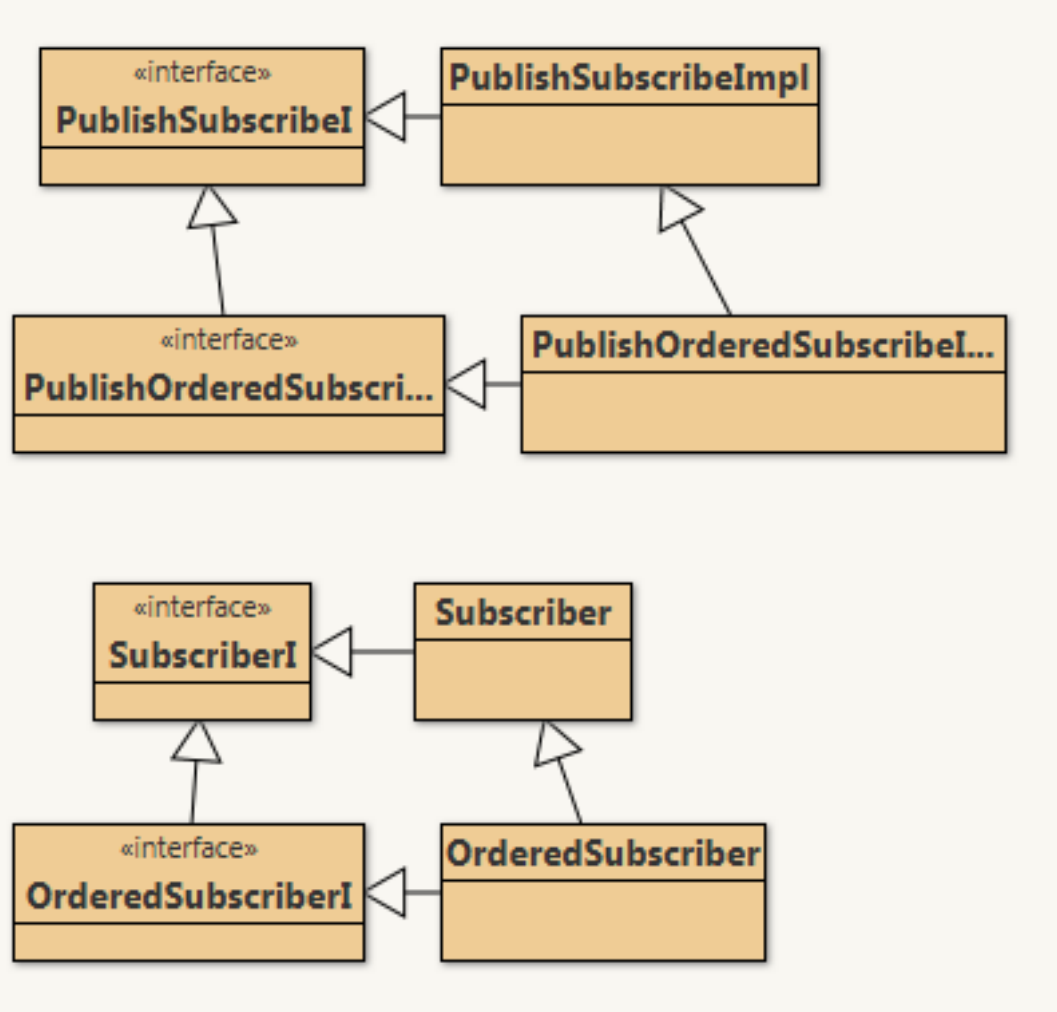
- **Publish/Subscribe, suite**
  - **Les fonctionnalités sont liées entre elles**
    - **Une liste des souscripteurs est en place**

- **Pondération possible des fonctionnalités**
  - **La liste des souscripteurs peut être ordonnée**
  - **Un des souscripteurs peut interrompre le traitement**

  - **Usage du patron Chaîne de responsabilités**
    - **+ liste ordonnée des souscripteurs**

Nouvelle implémentation

- **Un classique en langage à objets**
  - **Une nouvelle sous-classe de PubSubImpl…**



Démonstration

- **Ajout**
  - d'une nouvelle fonctionnalité, d'un nouveau composant

- **Publish/Subscribe**
  - Une abstraction supplémentaire
    - La notification est reportée et effectuée par le *médiateur* (l'environnement)
  - Mais
    - Est-ce une approche par composants interchangeables
    - discussions

Conclusion intermédiaire ?

- **Si interface + mutateur ==** *variabilité possible*

- **Conteneur de beans**
  - **La configuration indique le choix de l'implémentation pour l'interface**
  - **Le conteneur maintient une liste de beans créés**

  -> Apporte le couplage faible nécessaire pour une meilleure variabilité

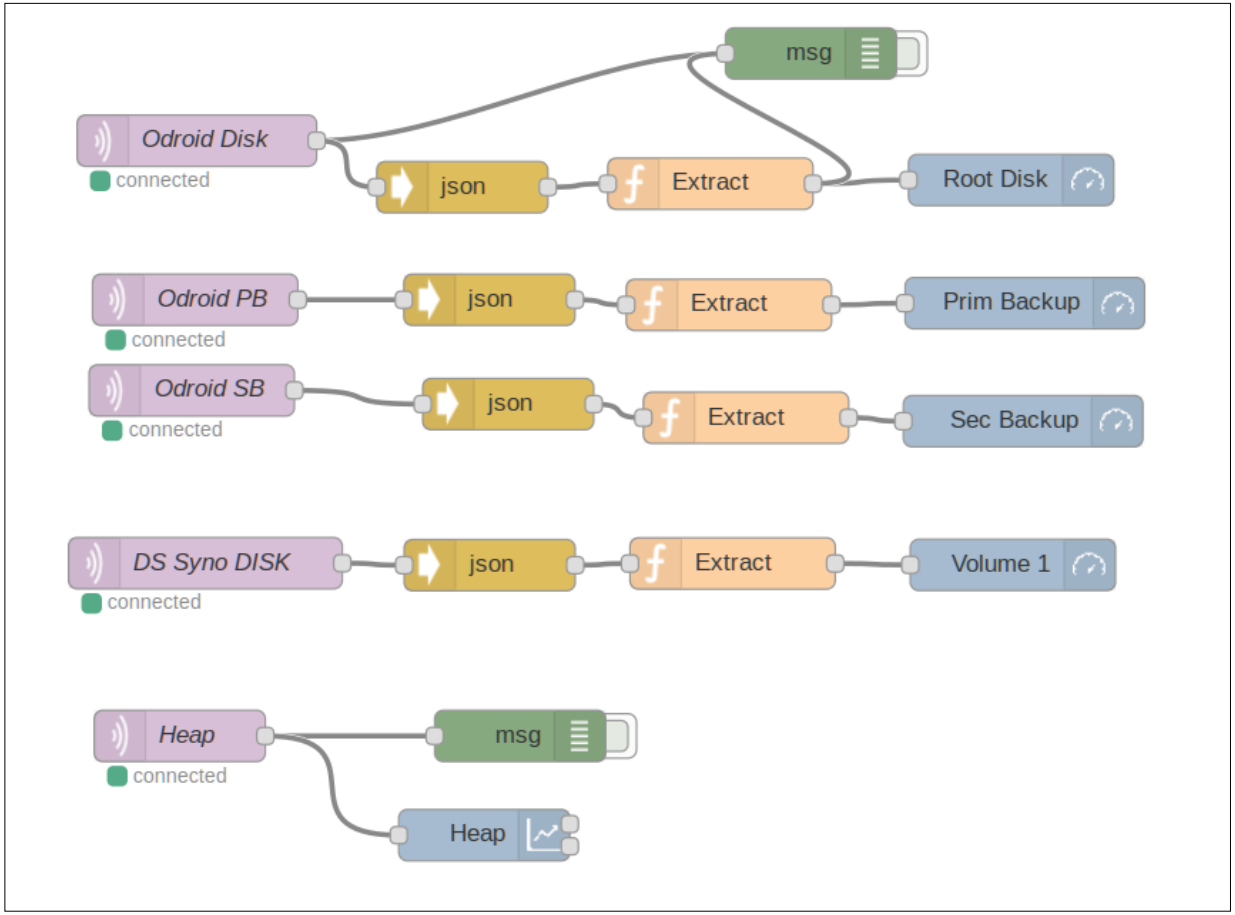
- **Conteneur et patrons, fructueux**
  - **Quels patrons ?**
    - **Tous ou presque**

- **Discussions**

Discussions

- **Est-ce bien une approche de composants/services ?**
  - SCA
    - Service Component Architecture
      - Bluemix ?
  - Abstraction du langage d'implémentation ?
  - Communication : Web Service, JSON …
    - Composants communicants versus appels de méthodes via les beans
  - Variabilité comme gestion de versions des composants …
  - Cf. Java Business Integration ou Bluemix, Azure…

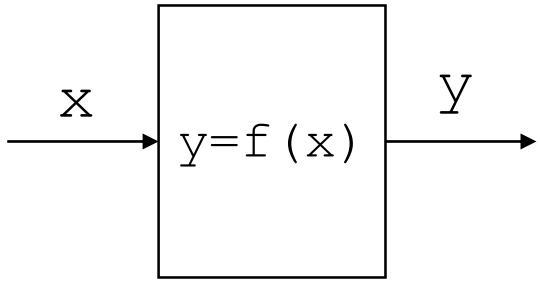
Un essai de composants et de configuration



- Bluemix, assembler des composants …
  - Variabilité : un nouveau composant, nouvelle liaison, gestion de versions ?
- https://primalcortex.wordpress.com/tag/mqtt/

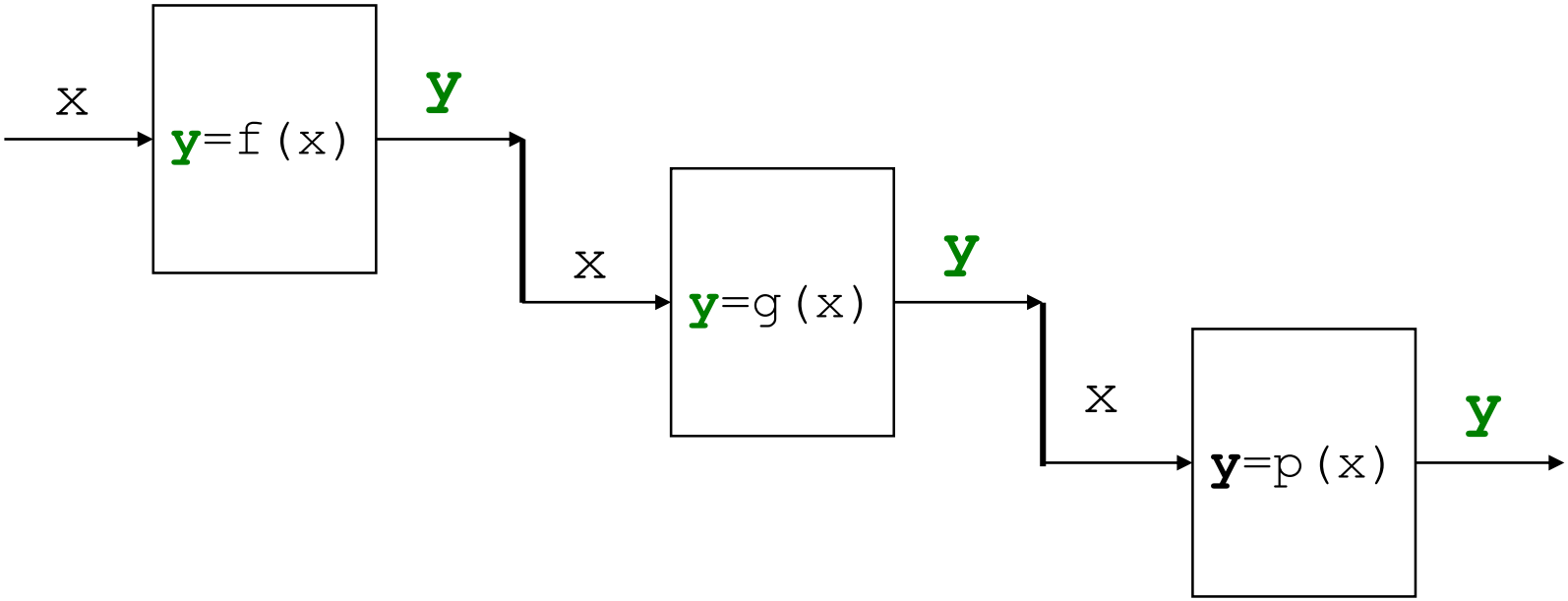
- Les diapositives qui suivent sont présentes pour générer des discussions…

Notion de composant, une tentative



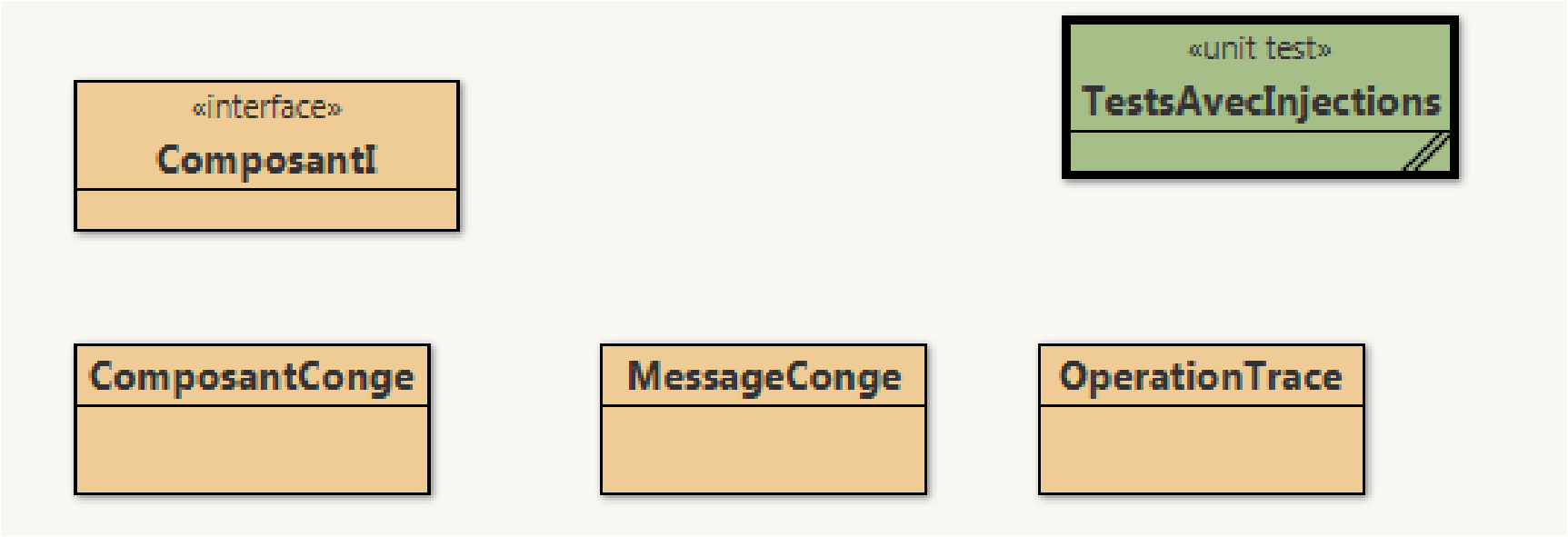
- **Chaque composant est générique**
  - **Une entrée x, une sortie y, une fonction f**
- **Les composants sont configurés dans un fichier texte**
  - **Variabilité : nouvelle configuration**
- **Les composants peuvent être un composite,**
  - **des composants de composants**

Assembler des composants



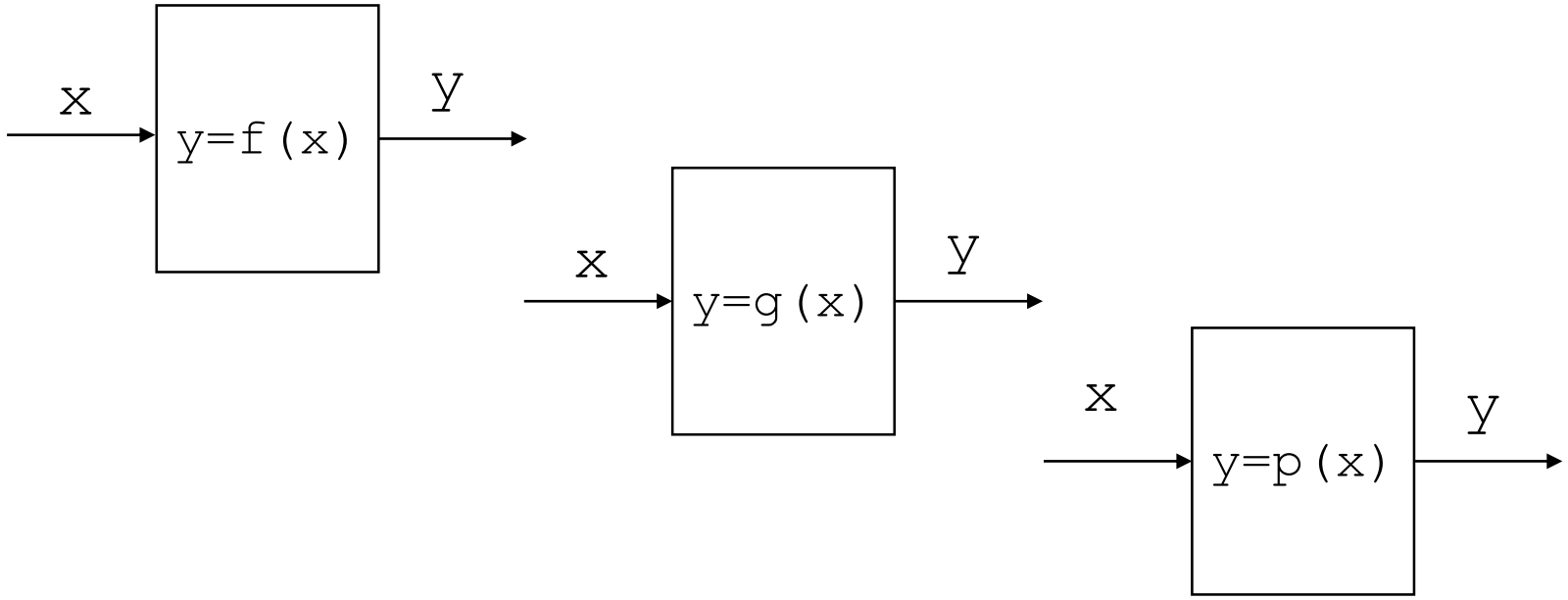
- **Assemblage ou configuration**
- **Variabilité ? Une nouvelle configuration !**
- **En java un exemple**

Une seule classe : Composant



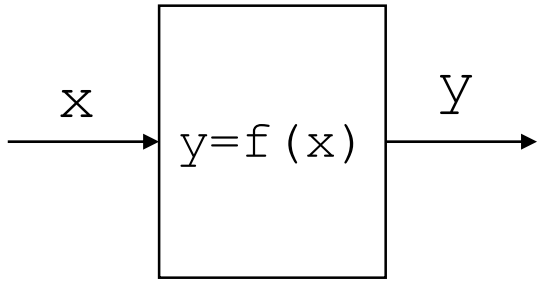
- **Chaque composant a son opération de calcul des congés**
  - **y = operation(x)**

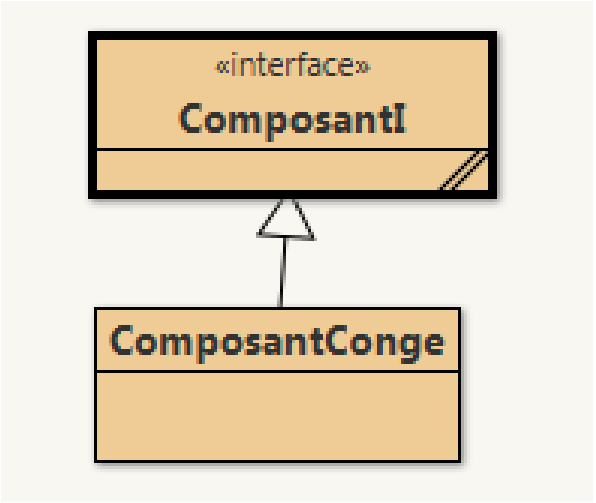
Déclaration



- **Chaque composant est autonome,**
  - **possède un thread en interne qui attend en entrée un message**
    - **Le message reçu depuis l'entré x, contient l'agent, le contexte, un résultat**
    - **le calcul est effectué et est envoyé sur la sortie y**
- **La liaison entre les composants se fera lors de la configuration**

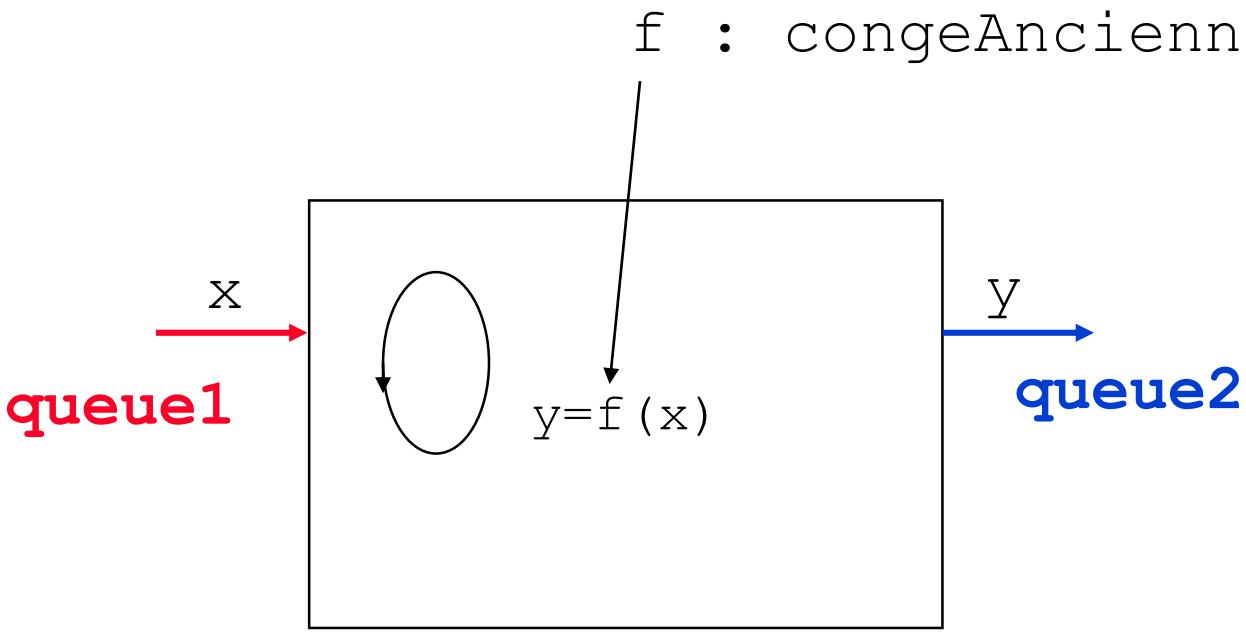
Le composant est générique






```
public interface ComposantI<X, Y, F>{
    public void setX(X x);
    public void setY(Y y);
    public void setF(F f);
}
```

Calcul des congés

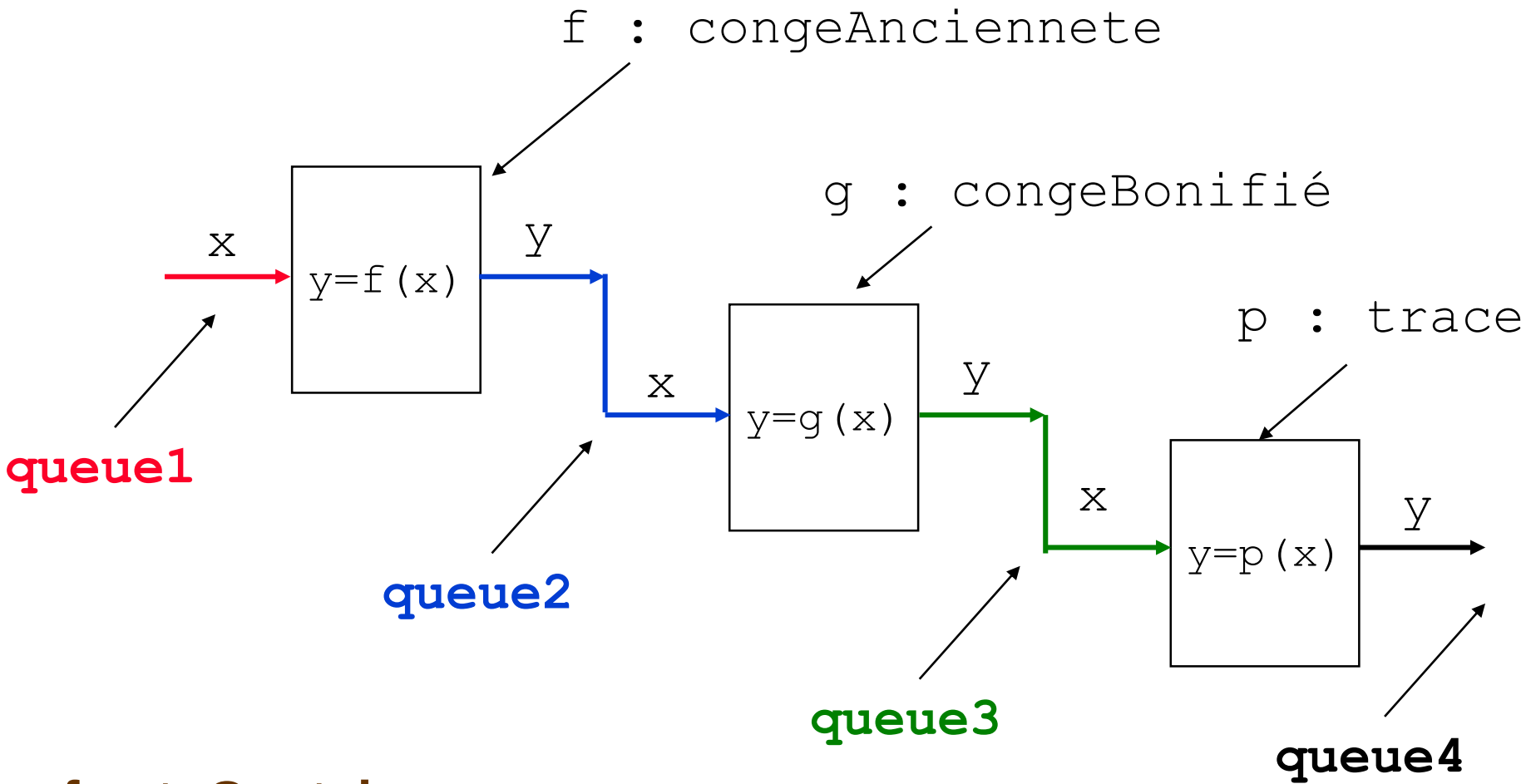


- **Un composant attend x en entrée et, envoie sur la sortie f(x)**

En java le composant des congés

```
public class ComposantConge
implements ComposantI<SynchronousQueue<MessageConge>,              // x
                      SynchronousQueue<MessageConge>,              // y
                      Operation<Contexte,ResultatConges,Agent>>, // f
                      Runnable{

  private SynchronousQueue<MessageConge> x;
  private SynchronousQueue<MessageConge> y;
  private Operation<Contexte,ResultatConges,Agent> f;

  private Thread thread;

  public ComposantConge(){
    this.thread = new Thread(this);
    this.thread.start();
  }
  …
  public void run(){
    while(!thread.interrupted()){
      try{
        MessageConge msg = x.take();
        f.apply(msg.getContexte(), msg.getResultat(), msg.getAgent());
        y.offer(msg);                    // y = f(x)
      }catch(Exception e){}
    }
  }
}
```

Configuration



- **femtoContainer**
  - **Assemblage des composants par le fichier de configuration**
  - **Ici une composition de fonctions f(g(p(x))**

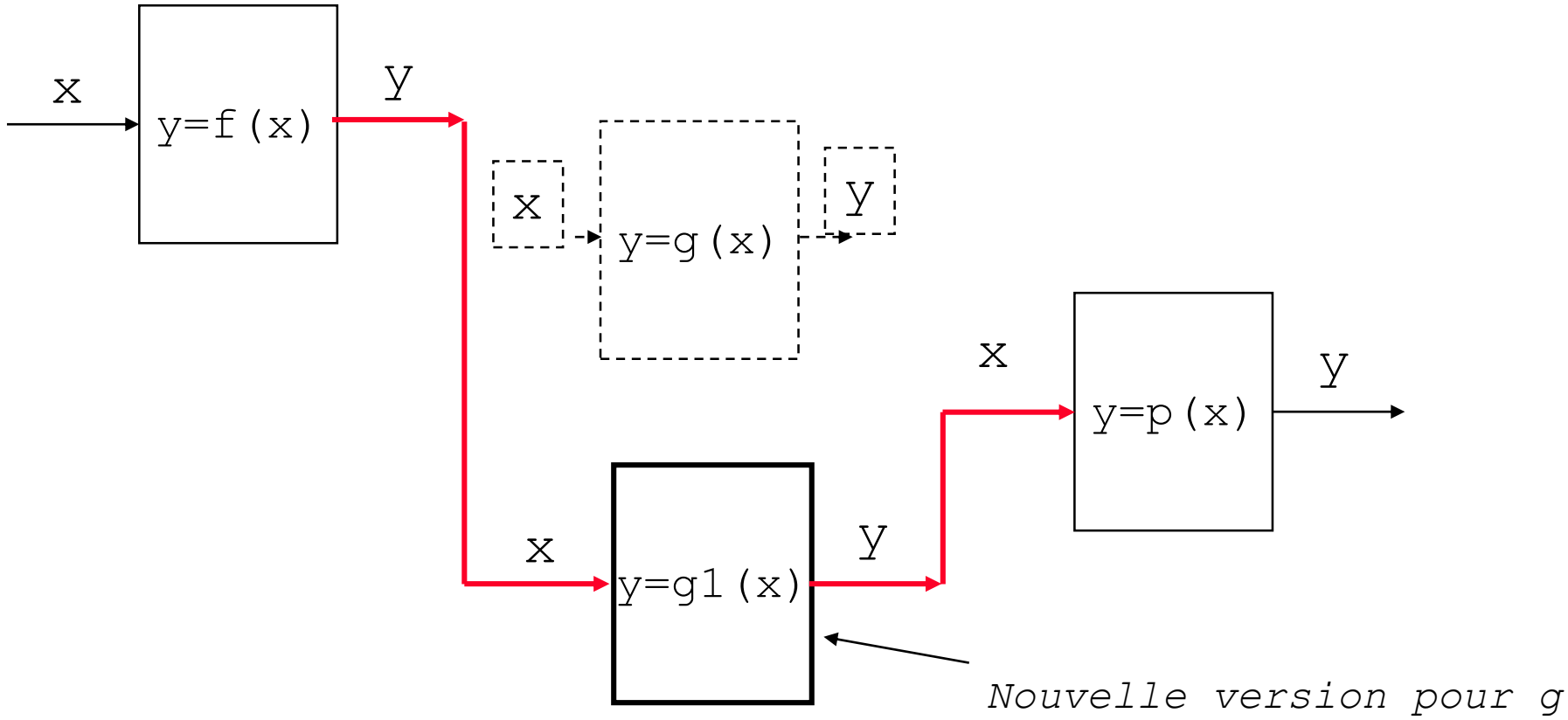
Configuration 1_2, 2 composants et les files de messages

```
bean.id.1=queue1
queue1.class=java.util.concurrent.SynchronousQueue
bean.id.2=queue2
queue2.class=java.util.concurrent.SynchronousQueue
bean.id.3=queue3
queue3.class=java.util.concurrent.SynchronousQueue

bean.id.5=composantConge1
composantConge1.class=injection_service_component.ComposantConge
composantConge1.property.1=x
composantConge1.property.1.param.1=queue1
composantConge1.property.2=y
composantConge1.property.2.param.1=queue2
composantConge1.property.3=f
composantConge1.property.3.param.1=operationCongesAnciennete

bean.id.6=composantConge2
composantConge2.class=injection_service_component.ComposantConge
composantConge2.property.1=x
composantConge2.property.1.param.1=queue2
composantConge2.property.2=y
composantConge2.property.2.param.1=queue3
composantConge2.property.3=f
composantConge2.property.3.param.1=operationCongesBonifies
```

Configuration 2_2
le 3ème composant et le calcul des congés

```
bean.id.4=queue4
queue4.class=java.util.concurrent.SynchronousQueue


bean.id.7=composantCongeTrace
composantCongeTrace.class=injection_service_component.ComposantConge
composantCongeTrace.property.1=x
composantCongeTrace.property.1.param.1=queue3
composantCongeTrace.property.2=y
composantCongeTrace.property.2.param.1=queue4
composantCongeTrace.property.3=f
composantCongeTrace.property.3.param.1=operationTrace


bean.id.8=operationCongesAnciennete
operationCongesAnciennete.class=injection_commande.OperationCongesAnciennete

bean.id.9=operationCongesBonifies
operationCongesBonifies.class=injection_commande.OperationCongesBonifies

bean.id.10=operationTrace
operationTrace.class=injection_service_component.OperationTrace
```

Variabilité, un nouveau composant g remplace g1

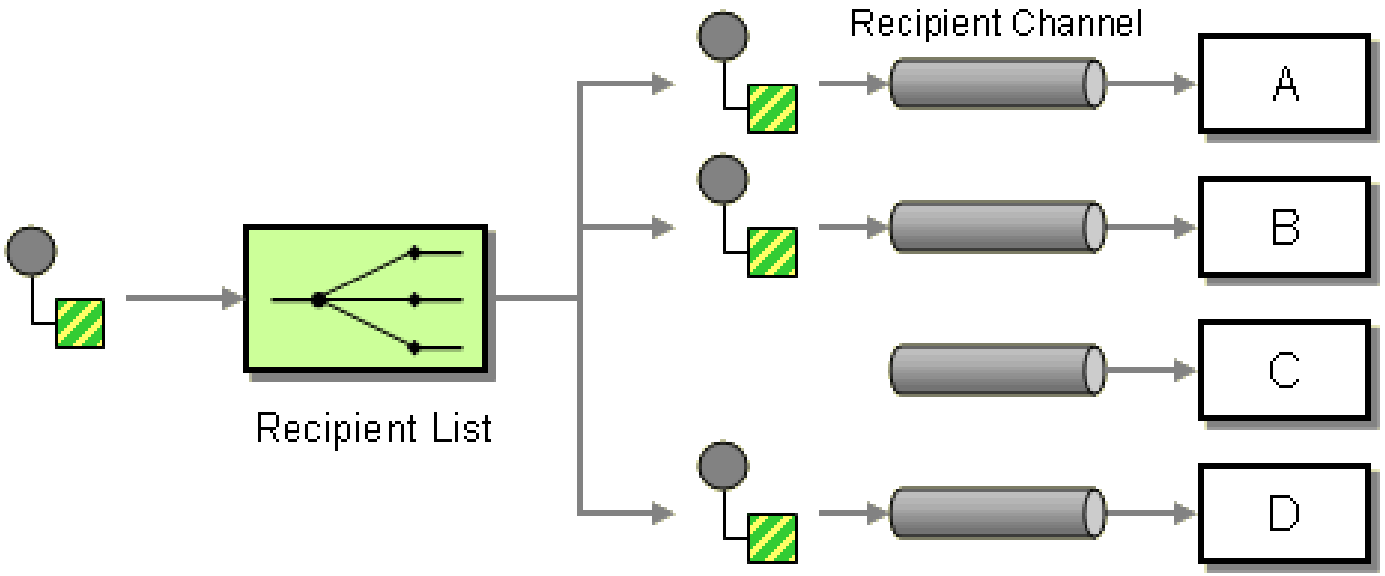


- **Une nouvelle configuration**
  - **Substitution d'un composant par un autre**
  - **Un nouveau composant, seul le fichier de configuration est concerné**
  - **Le « câblage » est différent** ⟶

Chaque composant a accès au ServiceLocator

- **Composant + ServiceLocator**
- **À terminer**
- **SPL, Gestion de versions**
- **Méthodes**

Le Composant RecipientList
www.enterpriseintegrationpatterns.com



- Ajout d'un composant utilitaire, cf. en annexe
- http://www.enterpriseintegrationpatterns.com/patterns/messaging/RecipientList.html

Démonstration/discussions

- **Un nouveau composant ?**
- **Chaque Composant peut utiliser le patron composite**
- **Chaque composant peut être implémenté par un ou des patrons**

Conclusion

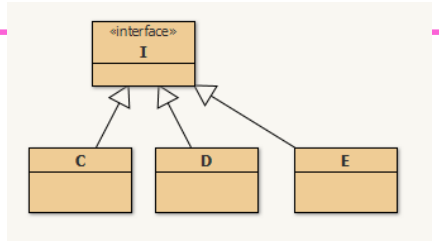
- **Variabilité ?**
- **Quelles directions à étudier … ?**
- **Bluemix ? Azure, Amazon …**

Annexes

Annexe

- **Forme canonique,**
  - **Interface + mutateur + Beans = PGCD des patrons ?**
  - **Conteneur de beans**
  - **Configuration / Utilisation**

Conteneur de Beans : Utilisation/Configuration



Utilisation

```
class M{
  private I i; // variation
  public void setI(I i){…
```

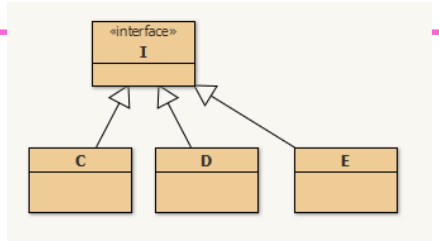
Configuration.txt

```
bean.id1=bean.m
bean.m.class=M.class
bean.m.param.i=bean.i

bean.id2=bean.i
bean.i.class=C.class
```

Analyse du fichier
Création du conteneur

```
M m = conteneur.getBean("bean.m");
      au lieu de
            M m = new M();
            m.setI( new C());
```

Conteneur de Beans : Utilisation/Configuration



Utilisation

```
class M{
  private I i;// variation
  public void setI(I i){…
```

Configuration.txt

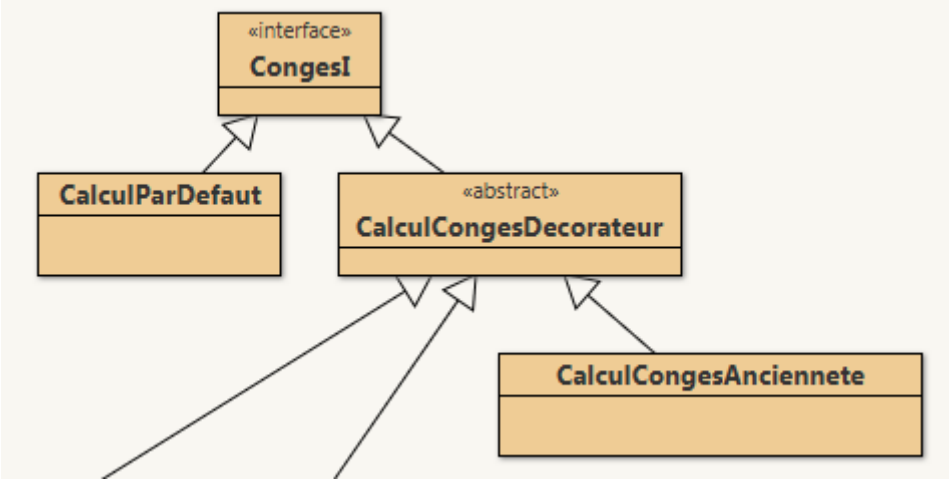
```
bean.id1=bean.m
bean.m.class=M.class
bean.m.param.i=bean.i

bean.id2=bean.i
bean.i.class=D.class
```

```
Le source java est le même
M m = conteneur.getBean("bean.m");
```

Annexes

- **La patron décorateur : un complément**

Un nouveau décorateur et calcul des nouveaux congés dûs à l'ancienneté




```
public abstract class CalculCongesDecorateur implements CongesI{
  private CongesI calculette;

  public CalculCongesDecorateur(CongesI calculette){
    this.calculette = calculette;
  }
  public CalculCongesDecorateur(){super();}

  public void setCalculette(CongesI calculette){
    this.calculette = calculette;
  }

  public Integer calculer(Contexte ctxt, Agent agent){
      return calculette.calculer(ctxt,agent);
  }
}
```

Annexe

- **Le patron interceptor**
  - **Comment prendre en compte « à chaud » un changement de configuration ?**
  - **Une notification asynchrone dès la modification du fichier de configuration …**
  - **Une interception de certains appels de méthodes et vérification de la date du fichier …**
  - **Une notification dédiée …**
  - **À suivre…**

Outil d'injection c'est bien mais

- **Bien :**
  - **Pas de modification du source Java** *ok*
  - **Un fichier texte ou XML de configuration font l'affaire** *ok*
  - **Design pattern + injection** *ok*

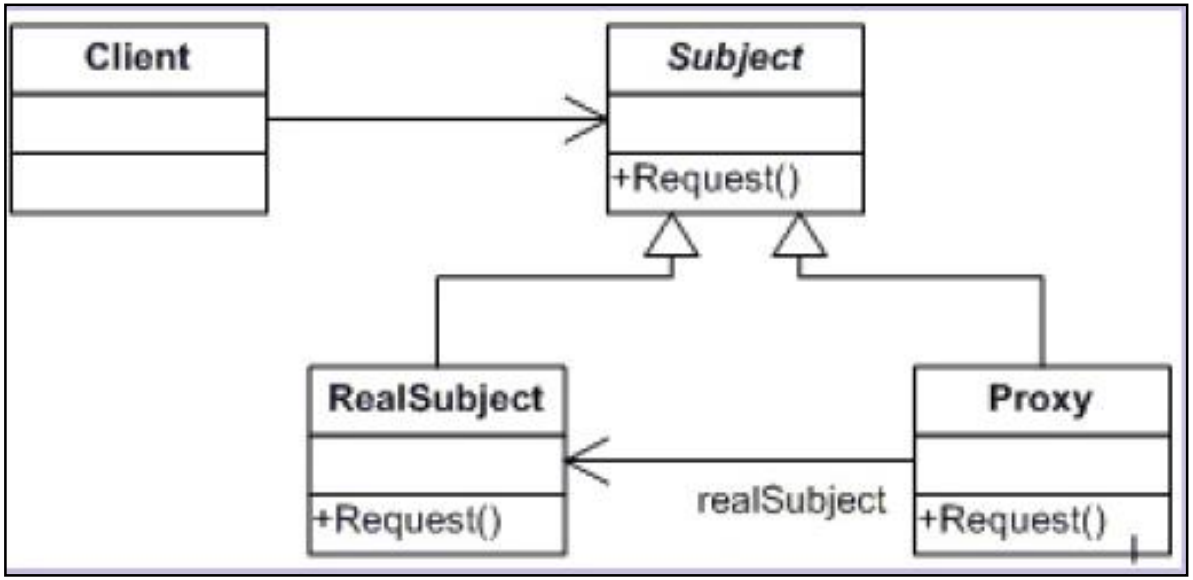
- **mais**
  - **Nécessite un redémarrage de l'application** *…*
    - **Due à la nouvelle configuration**

- **alors**
  - **Un intercepteur pourrait être mis en œuvre**
    - **Intercepteur qui irait s'enquérir de la nouvelle configuration… si celle-ci a changé… Utile/inutile ? Trop coûteux ?**
  - **Notification asynchrone en cas de modification de la configuration ? JMX ?**
    - **Approche système ou architecture ?**
    - **TheFolderSpy ou apparenté**
      - http://www.thewindowsclub.com/thefolderspy-lets-you-track-all-activities-in-any-folder

- **En annexe deux propositions :**
  - **Le patron Interceptor + conteneur**
    - **Interception de tout appel de méthode, trace, coût en temps d'exécution, …**
  - **Notification JMX à la suite du changement de configuration**

En java interception de tout appel de méthode

- **Le patron Proxy est mis en œuvre**



- **Point besoin de l'inventer**
  - **java.lang.reflect.DynamicProxy,**
    http://docs.oracle.com/javase/7/docs/technotes/guides/reflection/proxy.html
    - **C'est tout prêt**
  - Le patron Procuration
    - http://jfod.cnam.fr/NFP121/supports/NFP121_cours_07_introspection.pdf diapositives 66-86

Interceptor + injection

```
bean.id.6=proxy
proxy.class=injection_decorateur.ProxyFactory
proxy.property.1=type
proxy.property.1.param.1=injection_decorateur.CongesI.class
proxy.property.2=interceptor
proxy.property.2.param.1=interceptor
```

Un intercepteur quelque soit le type de données quelque soit la méthode appelée

- **Usage via le conteneur**

```
ProxyFactory proxy = container.getBean("proxy");
CongesI conges = proxy.create();
Integer nombreDeJours = conges.calculer(contexte, agent);
```

ProxyFactory, quelque soit le type, quelque soit la méthode en java

```
public class ProxyFactory{
  private Class<?> type;
  private InvocationHandler interceptor;

  public <T> void setType(Class<T> type){
    this.type = type;
  }

  public void setInterceptor(InvocationHandler interceptor){
      this.interceptor = interceptor;
  }

  public <T> T create(){ // création du mandataire
    Class<T> t = (Class<T>)type;
    return ProxyFactory.create(t,interceptor); // diapo suivante
  }
```

ProxyFactory, la suite…
quelque soit le type, quelque soit la méthode

```
public static <T> T create(final Class<T> type,
                           final InvocationHandler handler){
        return type.cast(
                Proxy.newProxyInstance(
                type.getClassLoader(),  // le chargeur de classes
                new Class<?>[] {type},  // l'interface implémentée
                handler));              // l'intercepteur
    }

} // fin de la classe ProxyFactory
```

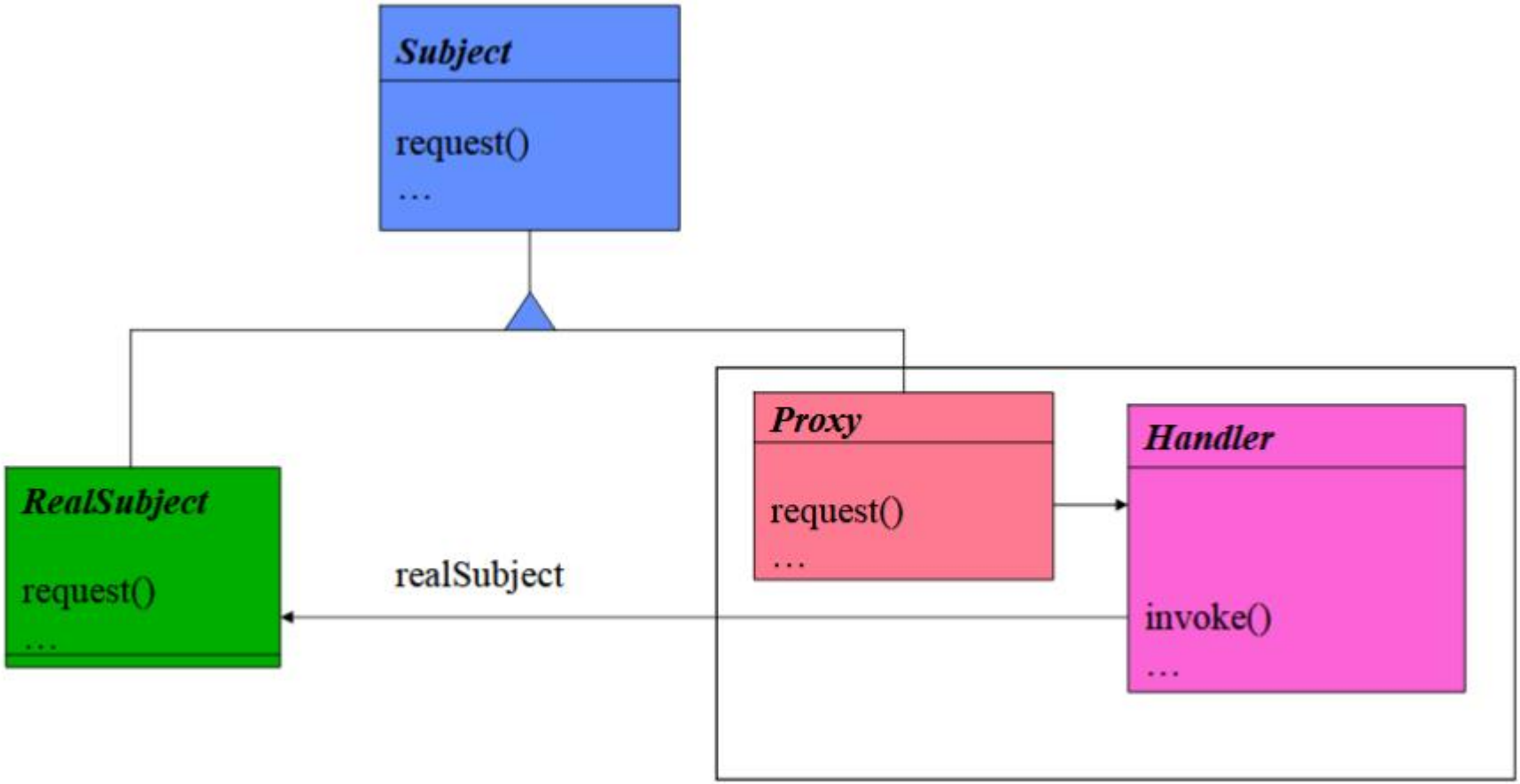
En standard java, la création du Proxy

Proxy.newProxyInstance de java.lang.reflect

Interceptor implémente InvocationHandler

```
public class Interceptor implements java.lang.reflect.InvocationHandler{
  private Object target;
  public Interceptor(){}
  public Interceptor(Object target){this.target = target; }

  public void setTarget(Object target){
    this.target = target;
  }


  public Object invoke(Object proxy, Method m, Object[] args)
                                                  throws Throwable{
    try{
      // si la configuration a changé alors …

      return m.invoke(target,args);

    }catch(InvocationTargetException e){
      throw e.getTargetException();
    }

  }
}
```

DynamicProxy en couleur



- **Invoke comme intercepte … cf. java.lang.InvocationHandler**

Notification d'un changement dans un répertoire

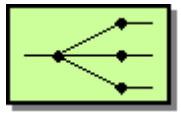
- **Approche par le système d'exploitation**
  - **Un fichier de configuration vient d'être modifié**
- **WatchService**
  - **https://docs.oracle.com/javase/tutorial/essential/io/notification.html**
  - **Un exécutable libre : Folderspy**
  - **Comment être notifié d'une modification sur un fichier**
  - **JMX, c'est tout prêt…**
    - **Rappels, principes, au sein de chaque JVM un serveur JMX est en place**
    - **http://jfod.cnam.fr/NSY102/supports/NSY102_06_JMX.pdf**

Annexe PubSub Sans femtoContainer nous aurions…

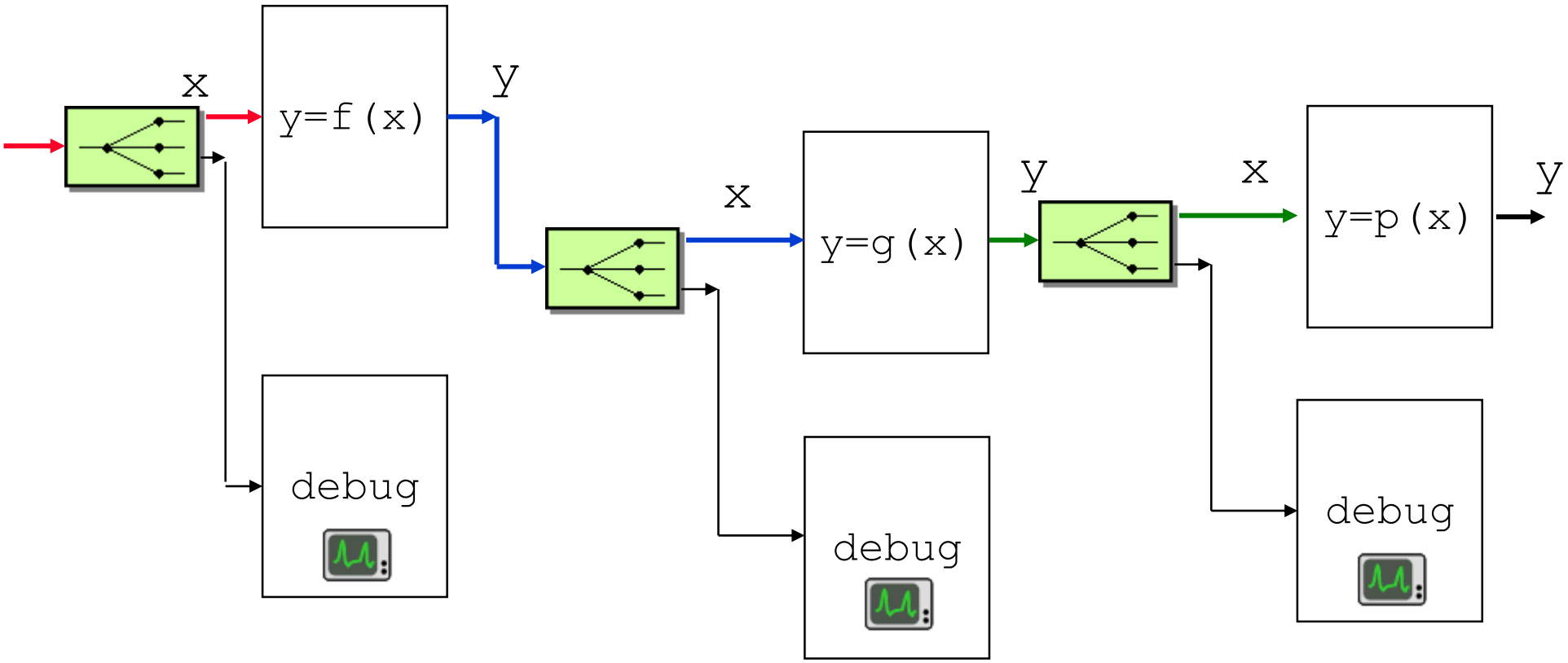
```
PublishSubscribeI pubsub = new PublishSubscribeImpl();
Contexte ctxt = …;
 Agent agent = …;
ResultatConges result = new ResultatConges();

SubscriberI sub1 = new Subscriber();
sub1.setName("anciennete");
sub1.setTopic("conge");
sub1.setOperation(new OperationCongesAnciennete());
pubsub.setSubscriber(sub1);

pubsub.publish("conge",ctxt,result,agent);
assertEquals(50,result.getNombreDeJours());

SubscriberI sub2 = new Subscriber();
sub2.setName("bonifie");
sub2.setTopic("conge");
sub2.setOperation(new OperationCongesBonifies());
pubsub.setSubscriber(sub2);
result = new ResultatConges();
pubsub.publish("conge",ctxt,result,agent);
assertEquals(65, result.getNombreDeJours());
```

SCA

- **Service Component Architecture**
  - **Un composant utilitaire est ajouté …**
- **RecipientList**
  - **www.enterpriseintegrationpatterns.com**



Ajout d'un composant utile



- **Ajout du composant utilitaire : RecipientList**
  - **www.enterpriseintegrationpatterns.com**
- **Envoi du message vers un composant de type *debug***
  - ***Ici un affichage sur la console***

Modification de la configuration, un extrait

```
bean.id.6=composantConge2
#...
#composantConge2.property.2.param.1=queue3
composantConge2.property.2.param.1=queueTrace1
#...


bean.id.11=queueTrace1
queueTrace1.class=java.util.concurrent.SynchronousQueue

bean.id.15=recipientList1
recipientList1.class=injection_service_component.RecipientList
recipientList1.property.1=in
recipientList1.property.1.param.1=queueTrace1
recipientList1.property.2=out
recipientList1.property.2.param.1=queue3
recipientList1.property.3=out
recipientList1.property.3.param.1=debug
```