Refactoring

Module HAI913I

Marianne Huchard

Refactoring

[Fowler et al, 1999] Refactoring : Improving the Design of Existing Code
by Martin Fowler, Kent Beck, John Brant, William Opdyke, don Roberts

Révision pour Java en 2002. Ce cours est principalement tiré du livre.

site web : https ://refactoring.com/catalog/

> *Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs.*

Refactoring

Principes

- Lutter contre la dégradation du logiciel
- Améliorer la compréhension
- Diminuer le coût des modifications et le temps de programmation
- Aide à la découverte des bugs
- Des étapes simples dont l'effet cumulé améliore le design

Temporalité

- Etape typique de l'Extreme Programming et des processus Agile : petites étapes de conception suivies de refactoring, diminue le stress de la conception
- Lors des évolutions
- Lors des corrections de bugs
- Lors des revues de code
- Lors des étapes d'optimisation des temps de calcul

Refactoring

Obstacles stratégiques

- L'équipe de développement est payée pour produire de "nouvelles fonctions"
- Les bénéfices ne sont pas immédiats
- Il est nécessaire d'effectuer des tests de non régression : avoir des tests solides et automatisés
- Il faut de bons outils

Obstacles techniques

- Déterminer ce qui doit être retravaillé et comment
- La liaison avec les bases de données peut limiter la flexibilité des modifications
- Changer les interfaces d'une API est coûteux pour les programmes clients

Catalogue de refactoring (Fowler et al. 2002)

Format de la description

- nom
- résumé (situation, effet)
- motivation
- mécanique
- exemples

Catalogue de refactoring (Fowler et al. 2002)

Principales catégories

- Composing methods
- Moving Features Between Objects
- Organizing Data
- Simplifying Conditional Expressions
- Making Method Calls Simpler
- Dealing with Generalization
- Big Refactorings

Composing methods

- Inline method (remplacer l'appel d'une méthode très simple par son code)
- Remove assignments to parameters (utiliser plutôt des variables locales)
- Replace Temp with query (utiliser une méthode plutôt qu'une variable locale contenant une expression complexe)
- Form Template Method (regrouper des instructions apparaissant dans plusieurs méthodes et correspondant à des étapes de ces méthodes, mais appelées dans un ordre différent)
- ...

Composing methods : un exemple

- Extract Method [Fowler et al. 2002]
- Diviser une méthode trop longue ou regrouper des parties de méthodes qui ont une fonction commune

```
1 void printOwing(double amount) {
2   printBanner();
3   //print details
4   System.out.println ("name:" + _name);
5   System.out.println ("amount" + amount);
6 }
```

```
1 void printOwing(double amount) {
2   printBanner();
3   printDetails(amount);
4 }
5 void printDetails (double amount) {
6   System.out.println ("name:" + _name);
7   System.out.println ("amount" + amount);
8 }
```

Moving features

- Move field
- Move method
- Extract Class
- Hide delegate
- ...

Moving Features Between Objects : un exemple

- Move Method [Fowler et al. 2002] "Moving methods is the bread and butter of refactoring."
- Lorsque les responsabilités sont mal distribuées, les classes trop couplées

```
1 public class Bibliotheque {
2 ...
3 }
4 public class Livre {
5   public void inscritAdherent(){...}
6 }
```

```
1 public class Bibliotheque {
2   public void inscritAdherent(){...}
3 }
4 public class Livre {
5 }
```

Moving Features Between Objects : un exemple

- Extract class [Fowler et al. 2002]
- Lorsque la classe a trop de responsabilités, parfois mal cernées

```
1 public class Animal {
2   private String nomEspece;
3   private double tailleMaxAdulte;
4   private double age;
5   private double genre;
6 }
```

```
1 public class Animal {
2   private double age;
3   private double genre;
4   private Espece espece;
5 }
6   public class Espece {
7   public String nom;
8   public double tailleMaxAdulte;
9 }
```

Organizing Data

- Change bidirectional association to unidirectional
- Change value to reference (et l'inverse)
- Encapsulate a collection (en donner une vue immuable et les opérations pour la manipuler)
- ...

Organizing Data : un exemple

- Replace Type Code with Class (or enum) [Fowler et al. 2002]
- Exemple transposé en Java 8

```java
public class Person {
  public static final int O=0, A=1, B=2, AB=3;
  public int groupeSanguin;
}
```

```java
public class Person {
  public GroupesSanguin groupeSanguin;
}
  public enum GroupesSanguin {
   O, A, B, AB;
}
```

Simplifying conditional expressions

- Remove control flag (remplacer par break ou return)
- Introduce assertion
- Consolidate duplicate conditional fragments
- ...

Simplifying Conditional Expressions : un exemple

- Replace Conditional with Polymorphism

```
1  double getSpeed() { // Dans Bird class
2   switch (_type) {
3    case EUROPEAN: return getBaseSpeed();
4    case AFRICAN: return getBaseSpeed() - getLoadFactor() *↩
         _nbCoconuts;
5    case NORWEGIAN_BLUE: return (_isNailed) ? 0 : getBaseSpeed(↩
         _voltage);
6   }
7   throw new RuntimeException ("Should be unreachable");
8  }
```

```
1  public class Bird { public abstract double getSpeed(); }
2  public class European extends Bird{
3    public double getSpeed(){return getBaseSpeed();}
4  }
5  public class African extends Bird{
6    public double getSpeed()
7      {return getBaseSpeed() - getLoadFactor() *_nbCoconuts;}
8  }
9  public class Norwegian_Blue extends Bird{
10   public double getSpeed()
11     {return (_isNailed) ? 0 : getBaseSpeed(_voltage);}
12 }
```

Making method calls simpler

- Encapsulate downcast (spécialiser un type de retour par ex.)
- Remove setting methods (pour les attributs qui ne doivent pas être modifiés)
- Introduce Named Parameter (nommer —correctement les paramètres)
- ...

Making Method Calls Simpler : un exemple

- Replace Error Code with Exception

```
1 int withdraw(int amount) {
2   if (amount > _balance)
3     return -1;
4   else {
5     _balance -= amount;
6     return 0;
7   }
8 }
```

```
1 void withdraw(int amount) throws BalanceException {
2   if (amount > _balance) throw new BalanceException();
3   _balance -= amount;
4 }
```

Dealing with Generalization

- Pull up / push down Method/field/constructor body
- Replace Constructor with Factory
- Form Template Method
- Extract Subclass, Extract Superclass, Extract Interface / collapse hierarchy
- Replace Inheritance with Delegation (or reversely)

Dealing with Generalization : un exemple

- Replace Inheritance with Delegation

```
1 public class Vector{
2 }
3
4 public class Stack extends Vector {
5 }
```

```
1 public class Vector{
2 }
3
4 public class Stack {
5   private Vector content;
6 }
```

Big refactorings

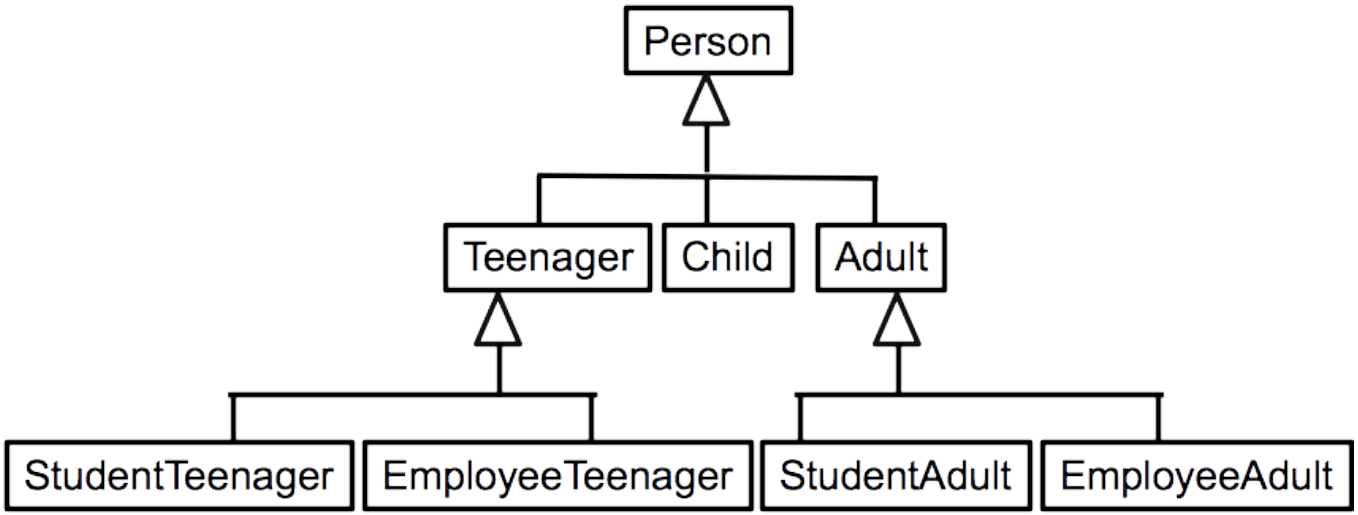
- "The preceding chapters present the individual "moves" of refactoring. What does the whole game look like ? (...) All the (small) refactorings can be accomplished in a few minutes or an hour at most. We have worked at some of the big refactorings for months or years on running systems. (...) Because they can take such a long time, the big refactorings also don't have the instant gratification of the refactorings in the other chapters. You will have to have a little faith that you are making the world a little safer for your program each day." [Kent Beck and Martin Fowler 2002]

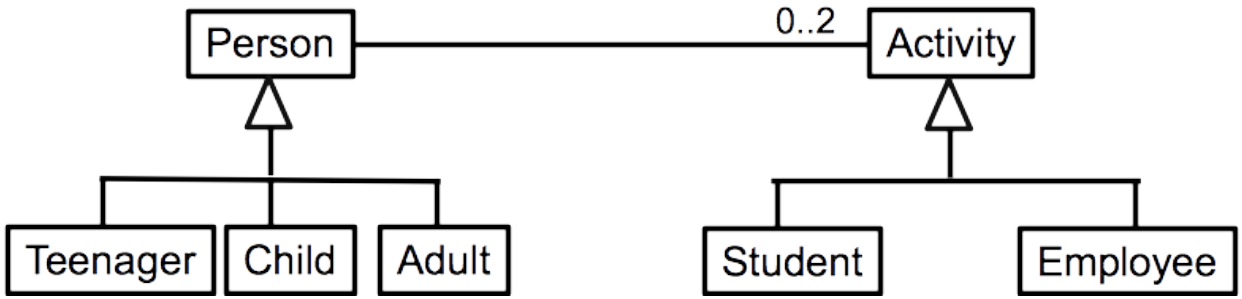
Exemples [Kent Beck and Martin Fowler 2002] :

- Tease Apart Inheritance
- Convert Procedural Design to Objects
- Separate Domain from Presentation
- Extract Hierarchy

Big refactoring : Tease Apart Inheritance

- "You have an inheritance hierarchy that is doing two jobs at once. Create two hierarchies and use delegation to invoke one from the other." [Kent Beck and Martin Fowler 2002]





Bad Smells / Code smells

Ce sont des structures dans le code qui "sentent mauvais" et encouragent à effectuer des refactorings. Quelques exemples :

- Code dupliqué $\Longrightarrow$ Extract Method, Substitute algorithm, Extract class
- Longue méthode $\Longrightarrow$ Extract Method
- Grande classe $\Longrightarrow$ Extract class, Extract Subclass, Extract interface
- Lazy class $\Longrightarrow$ Collapse Hierarchy, Inline Class
- Longue liste de paramètres $\Longrightarrow$ Replace Parameter with Method, Preserve Whole Object, Introduce Parameter Object
- Changement divergent (une classe est régulièrement modifiée de diverses manières) $\Longrightarrow$ Extract class
- "Shotgun Surgery" (les changements demandent de modifier de nombreuses classes) $\Longrightarrow$ Move Method, Move Field, Inline Class

Bad Smells / Code smells

- Feature Envy (une méthode s'intéresse beaucoup aux attributs d'une autre classe) $\Longrightarrow$ Move Method, Extract Method
- Data clump (des groupes de données sont souvent accédées ensemble) $\Longrightarrow$ Extract class
- Switch Statements (tests de type régulier pour déterminer un traitement) $\Longrightarrow$ Extract Method, Move Method, Replace Type Code with Subclasses, Replace Conditional with Polymorphism
- Message Chains $\Longrightarrow$ Hide Delegate, Extract Method

Refactorings et design patterns

- [Gamma et al. 1994] Design Patterns : Elements of Reusable Object-Oriented Software. Erich Gamma Richard Helm, Ralph Johnson and John Vlissides

  Many of the refactorings, such as Replace Type Code with State/Strategy and Form Template Method are about introducing patterns into a system. As the essential Gang of Four book says, "Design Patterns . . . provide targets for your refactorings." There is a natural relation between patterns and refactorings. Patterns are where you want to be ; refactorings are ways to get there from somewhere else. I don't have refactorings for all known patterns in this book, not even for all the Gang of Four patterns [Gang of Four]. This is another aspect of the incompleteness of this catalog. I hope someday the gap will be closed. [Fowler et al. 2002]

Anti-patterns

[Brown et al. 1998] AntiPatterns : Refactoring Software, Architectures, and Projects in Crisis. William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick and Thomas J. Mowbray
"An AntiPattern is a pattern that tells how to go from a problem to a bad solution"

https ://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Architecture/Anti-Patterns

Quelques exemples :

- Functional Decomposition : Classes contenant une opération (comme AfficherSolde). C'était utile en Java avant l'existence des lambdas pour passer des fonctionnalités en paramètres comme pour le tri.
- Spaghetti code : De très longues méthodes.
- Blob, God class, couteau suisse : Une classe avec beaucoup d'attributs et de méthodes, qu'il faudra redécomposer.
- Object orgy : les champs d'une classe n'ont pas été encapsulés et de nombreuses classes y font accès. Si la structure interne de la classe change, toutes les classes clientes vont être impactées.
- Cargo cult programming : Utiliser les patterns sans savoir pourquoi.

Quelques problématiques

- Systèmes de détection/correction : La méthode Decor (Moha et al. 2008)
- Connexion avec les métriques logicielles (Simon et al. 2001)
- Application aux nouveaux paradigmes : SOA (Nayrolles et al. 2015), composants (Gautier/Seriai et al. 2006)
- Application à des problématiques ciblées : sécurité, performance, parallélisation du code, lambdas-expressions, etc. (Gyori et al. 2013)
- Ingénierie Dirigée par les Modèles pour le refactoring (Batory 2007)
- Search-based software engineering (Ouni et al. 2012, Mohan et al. 2018)