refactor tp1

pierre misse

October 2018

1 Création de programmes nécessitant un refactoring

1.1 Push down

Ce refactoring cherche à bien délimiter les classes, en ne s'occupant que de leurs propre contexte.

Ici les comportements de Person s'occupe décrit des comportements de Student, et ne sont du coup pas au bon endroit conceptuellement.

1.2 Introduce Parameter Object

Ce refactor essaye de faire des paquets d'arguments allant bien ensemble, pour une utilisation plus aisée lors du passage d'arguments.

Ici, temps de début et de fin, pourrais peut-être être rassemblés?

2 Application de refactorings sur les programmes d'un autre développeur

2.1 Projet à Théo rogliano : "Move type"

- Passage de la classe interne "roue", à une classe standalone.

- – Clic droit sur la classe "roue"
  – Refactor → Move Type to new file
  – "ok"

- Ceci permet de réutiliser une classe initialement interne dans d'autres parties du programme, ce qui peut arriver lors d'évolution du programme, ou un problème de conception.

- Dans cet exemple simple, la mise en oeuvre se passe sans problème, elle semble donc être efficace.

2.2 Bruno verley : "Extract Interface"

- extraction de l'interface des classes "Link" & "File".
- – Clic droit sur chacune des classes
  – Refactor → Extract Interface
  – Choix du nom de la future interface
  – Choix des méthodes à factoriser
  – Enlever l'option "Use extracted interface type where possible", dans la plupart des cas nous n'avons pas besoin de changer le type de nos instances par un type moins précis.
  – finish
- Ceci permet de réagir plus facilement lorsque l'on se rend compter que nous avons fais deux classes avec des interfaces implicites similaires, ou pour etendre un modèle initial.
- La documentation est générée, les annotations également, et le remplacement du type est possible. L'outil ne gère pas la possibilité d'avoir deux interfaces identiques, mais étant donné la complexité du problème, ce n'est pas surprenant. De plus, il semble manquer un refactor "étendre une interface" pour les classes, qui rajouterais la documentation & les annotations, ainsi que les méthodes présentes dans l'interface mais manquantes dans la classe.

3 Réflexion libre sur les refactorings d'eclipse

J'ai choisis de parler du sous ensemble des refactoring de type Introduce. Je vais commencer par comparer ceux qui sont commun (et qui auront éventuellement un nom différent) puis, je chercherais à voir la faisabilité et la rentabilité des autres.

À première vue, les quatre refactoring d'éclipse sont disponibles sur le catalogue. Ceux ci décrivant des comportements similaires, il n'y à pas grand chose à dire dessus, outre leurs comportements

- "Introduce Parameter (object)" : Son but est d'est d'englober un ensemble d'argument dans un nouvel objet, pour que le passage de nombreux arguments dans les méthodes soit plus lisibles. Eclipse en offre deux types pour des contextes différents.
- "Introduce Factory ("Replace Constructor with Factory Method" dans le catalogue) : Remplace les appels aux constructeurs de classe par une méthode statique "factory" intermédiaire créant des objets à sa place. Ceci peut par exemple permettre de rajouter des comportements qui ne vont conceptuellement pas dans le constructeur, comme la sauvegarde des instances d'une classe.

- Introduce Indirection ("Introduce Foreign Method" dans le catalogue) : Remplace une méthode, ou un bout de code pouvant être reconnu comme une méthode (notamment au niveau de l'environnement lexical)par une méthode classique sur la classe courante. Ceci permet de factoriser des bouts de codes relatifs à une autre classe non modifiable.

- Introduce Assertion : Ajoute une assertion pour vérifier que la méthode se déroule sans erreur. Bien que semblant simple à implémenter coté Eclipse, l'utilité semble être avant tout une question de style. Les gens utilisant les assertions systématiquement lui trouverons peut d'intérêt, alors que ceux qui en utilise peu auront plutôt tendance (en général) à utiliser des exceptions. Il semble donc assez peu intéressant à implémenter.

- Introduce Gateway : Souhaite encapsuler les interactions avec une ressource externe complexe (Gateway est équivalent à proxy ici). Un exemple d'utilisation peut être trouvé en essayant d'utiliser une des légions d'API qui ont été créée depuis plus de 20 ans ! Ceci semble faisable sur Eclipse, mais doit pouvoir se ramener à changer un appel d'une méthode par une autre, sur une classe proxy que l'utilisateur créerais.Par conséquent, son implémentation semble relativement peu rentable.

- Introduce Local Extension : Créer un wrapper ou une sous classe d'une autre classe, pour étendre ses comportements. Également parfaitement faisable, mais se ramène à un refactor similaire.

- Introduce Expression Builder : Dans le même esprit que le Introduce Gateway, il vise à faciliter la communication avec une API complexe, cette fois en créant du code standard par rapport à l'application que l'on développe. Ceci semble relativement complexe, et nécessite probablement une étude approfondie pour voir quand il serait intéressant de l'utiliser, et pour être sûr que le code généré serais sans erreur et proche ou exactement ce que l'on souhaite.

- Introduce Named Parameter : Transformer un appel de méthode en HashMap, pour améliorer la lisibilité du code. Ceci est déjà fait sans la transformation d'arguments sous Android Studio par exemple. En revanche, tant que l'on reste sous un IDE donnant ce service visuel, il n'est utile de l'implémenter que lorsque l'on souhaite partager ce code à des gens utilisant un IDE ne le donnant pas. Il est donc probablement jugé peu utile, bien qu'il pourrait être intéressant.

- Introduce Null Object : Introduit une classe nullMaClasse pour éviter de devoir vérifier si un objet existe. Une fois de plus, Eclipse pourrais l'implémenter facilement, de plus C'est une opération que l'on fait régulièrement, et qui pourrait donc trouver sa place en tant que nouveau refactor.