IFT2255 - Génie logiciel

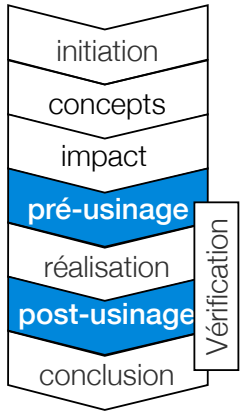
Réusinage

(basé sur le matériel de Václav Rajlich)

Bruno Dufour
dufour@iro.umontreal.ca

Analyse de l'impact

- Pré-usinage
  - Certains changements sont apportés afin de minimiser l'impact des changements
- Post-usinage
  - Permet d'éliminer ou de retarder la dégradation introduite par les changements



Réusinage

- Un réusinage modifie la structure du code sans affecter la fonctionnalité
  - Une activité importante lors de l'évolution et de la maintenance
- Un réusinage consiste en une séquence de transformations qui préservent le comportement
  - Plusieurs de ces transformations sont bien connues et cataloguées
    - http://refactoring.com/catalog/

Exemples de réusinage

- Renommer une entité
- Encapsuler une partie du code dans une fonction (extraire une fonction)
  - inverse : incorporer le code d'une fonction à la place d'un appel
- Déplacer une méthode d'une classe vers une autre
- Combiner ou diviser des classes
  - ex : extraire une classe parente

Extraire une classe parente

- Pour éviter la généralité spéculative, la structure devrait évoluer graduellement
  - La conception initiale est rarement parfaite
  - Introduire la généralité lorsque nécessaire
- Extraire une classe parente prépare le logiciel à l'ajout de nouvelles fonctionnalités par polymorphisme
  - Souvent utile lorsque la nouvelle fonctionnalité et l'ancienne ont beaucoup en commun

Exemple - Matrice dense


```
public class Matrix {
    protected int elements[][];
    protected int columns;
    protected int rows;

    public Matrix(int rows, int columns) { ... }
    public Matrix inverse() { ... }
    public Matrix multiply(Matrix m) { ... }
    public int get (int r, int c) { ... }
    public void set(int r, int c, int value) {...}
}
```


Extraction

- Requête de changement : ajouter le support pour les matrices creuses (*sparse matrix*)
  - Les algorithmes pour `inverse` et `multiply` sont les mêmes que pour les matrices denses (peuvent être réutilisés)
  - Les accès aux éléments sont différents (`get` et `set`)
- Extraire une classe abstraite
  - Deux sous-classes (`DenseMatrix` et `SparseMatrix`)
  - Un exemple de pré-usinage

Étape 1 - Renommer la classe

Nom plus spécifique en prévision des changements à venir (peut affecter les clients de `Matrix`)

```
public class DenseMatrix {
    protected int elements[][];
    protected int columns;
    protected int rows;

    public DenseMatrix(int rows, int columns) { ... }
    public Matrix inverse() { ... }
    public Matrix multiply(Matrix m) { ... }
    public int get (int r, int c) { ... }
    public void set(int r, int c, int value) {...}
}
```

Étape 2 - Extraire la classe parente

- Créer une classe abstraite `Matrix`
  - Faire hériter `DenseMatrix` de `Matrix`
- Remplacer tous les accès au tableau d'éléments par des appels à `get` et `set`
- Déplacer les attributs `columns` et `rows` vers la classe `Matrix`
- Créer des méthodes abstraites `get` et `set` dans `Matrix`
- Déplacer les fonctions `inverse` et `multiply` vers `Matrix`

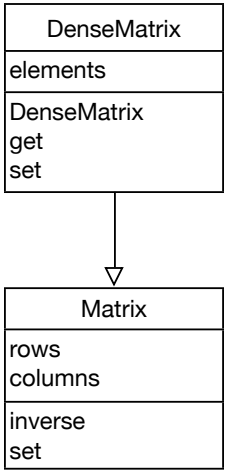
AbstractMatrix


```
public abstract class Matrix {
    protected int columns;
    protected int rows;

    public Matrix inverse() { ... }
    public Matrix multiply(Matrix m) { ... }
    public abstract int get (int r, int c);
    public abstract void set(int r, int c, int value);
}
```

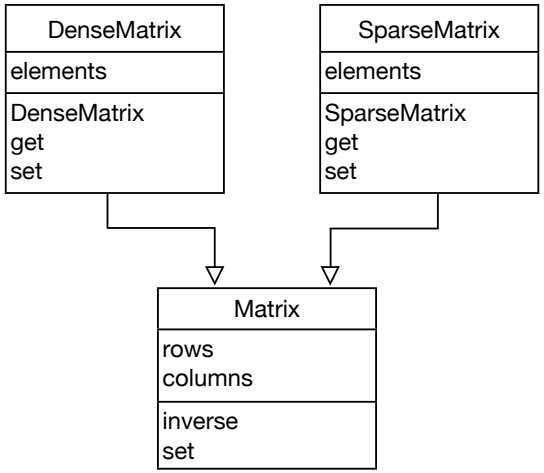

DenseMatrix


```
public class DenseMatrix extends Matrix {
    protected int elements[][];

    public int get (int r, int c) { ... }
    public void set(int r, int c, int value) { ... }
}
```


Résultat de l'extraction



Étape 3 - Ajout de classe



Extraire une méthodes

- Durant le développement, une méthode peut devenir trop grande
  - résultat d'une accumulation de fonctionnalité
- La méthode doit être divisée en plusieurs méthodes plus courtes
  - améliore la lisibilité
  - permet la réutilisation

Exemple


```
public static void countWords(File f) throws IOException {
    InputStream input = new BufferedInputStream(
                            new FileInputStream(f));

    int chr;
    boolean inWord = false;
    while ((chr = input.read()) >= 0) {
        chars++;
        if (chr == '\n')
            lines++;
        if (Character.isWhitespace(chr)) {
            inWord = false;
        } else if (!inWord) {
            words++;
            inWord = true;
        }
    }

    input.close();
}
```


Exemple


```
public static void countWords(File f) throws IOException {
    InputStream input = new BufferedInputStream(
                            new FileInputStream(f));

    int chr;
    boolean inWord = false;
    while ((chr = input.read()) >= 0) {
        chars++;
        if (chr == '\n')
            lines++;
        if (Character.isWhitespace(chr)) {
            inWord = false;
        } else if (!inWord) {
            words++;
            inWord = true;
        }
    }

    input.close();
}
```

2 responsabilités :
ouvrir un fichier et
calculer les statistiques

Procédure d'extraction de méthode

- Sélectionner un bloc de code pour l'extraction
- Identifier les variables et paramètres utilisés
  - S'assurer qu'au plus une variable est modifiée par le code à extraire
- Créer la nouvelle méthode
  - Déclarer en paramètre toutes les variables lues
  - Copier le bloc de code dans la nouvelle méthode
  - Gérer la valeur de retour si nécessaire
- Remplacer le bloc de code par un appel à la nouvelle méthode

Résultat de l'extraction


```
public static void countWords(File f) throws IOException {
    InputStream input = new BufferedInputStream(
                            new FileInputStream(f));
    countWords(input);
    input.close();
}

public static void countWords(InputStream input) {
    int chr;
    boolean inWord = false;
    while ((chr = input.read()) >= 0) {
        chars++;
        if (chr == '\n')
            lines++;
        if (Character.isWhitespace(chr)) {
            inWord = false;
        } else if (!inWord) {
            words++;
            inWord = true;
        }
    }
}
```