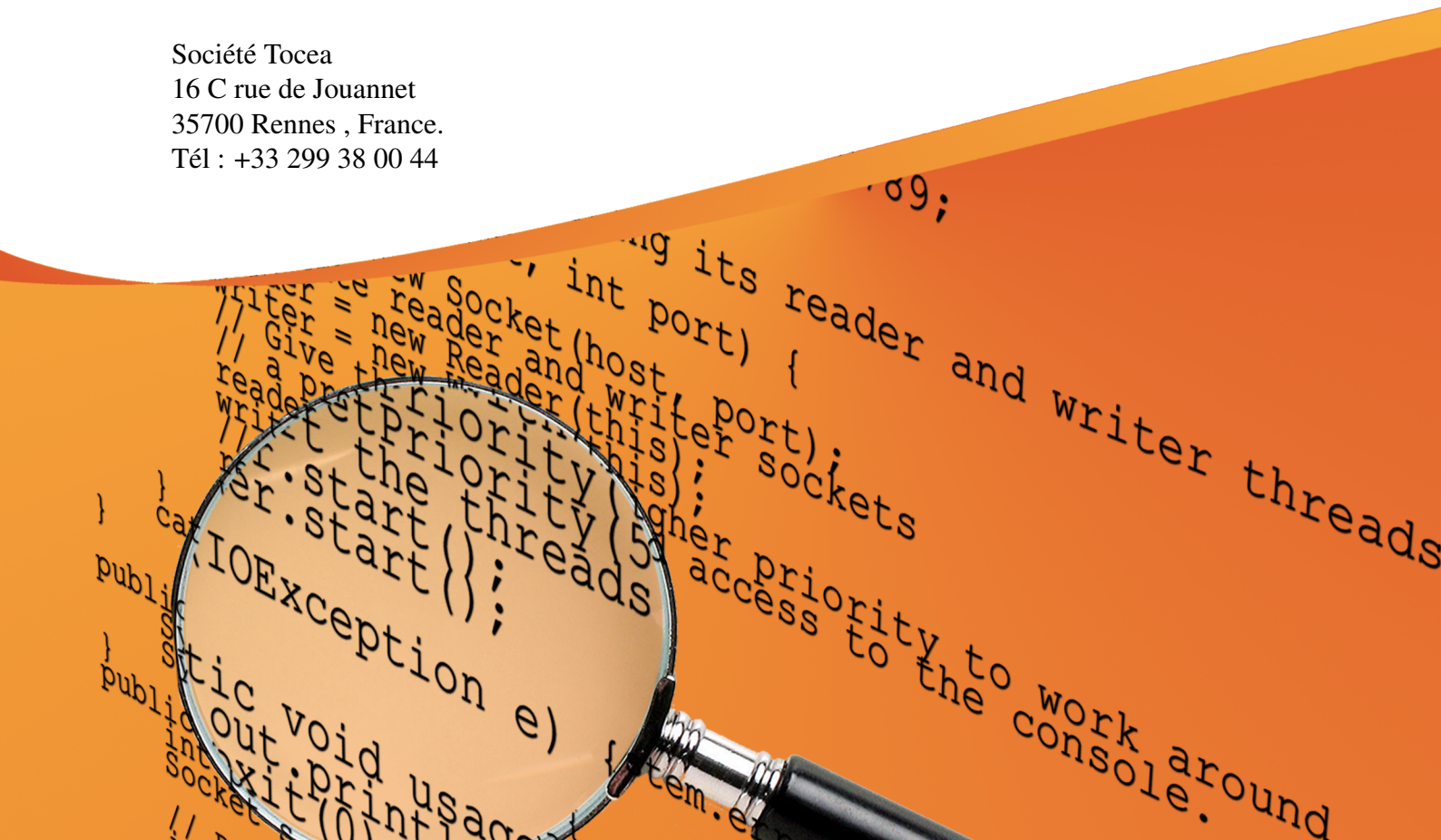


Exercice 1

Formation CleanCode : Module Nommage

Rennes
January 26, 2014

Société Tocea
16 C rue de Jouannet
35700 Rennes , France.
Tél : +33 299 38 00 44

Contents

1 Pré-requis

- Clonez le dépôt GIT à l'adresse suivante : `https://github.com/sleroy/cleancoder-course`
- Ouvrez le programme fourni avec le module 1 (`/module-1`)

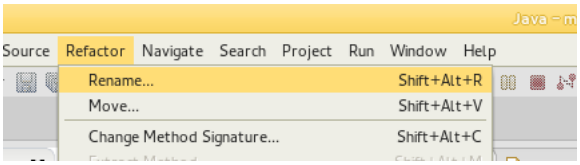
Attention, les plugin Maven et GIT vous sont nécessaires sous Eclipse pour charger le projet correctement.

2 Fonctionnalités de renommage sous Eclipse

2.1 Le refactoring Rename

Cette fonctionnalité est disponible comme d'habitude de plusieurs manières sous Eclipse :

- Par raccourci clavier : `ALT + SHIFT + R`
- Via le menu Refactoring



La fonctionnalité de renommage d'Eclipse est assez évoluée mais suppose que votre code **complètement**. Si votre code ne compile pas, il est possible qu'Eclipse ne puisse renommer l'ensemble des occurences de votre variable.
La fonctionnalité de renommage fonctionne sur tous les identifiants de votre code source.
Exemple d'utilisation de la fonctionnalité :

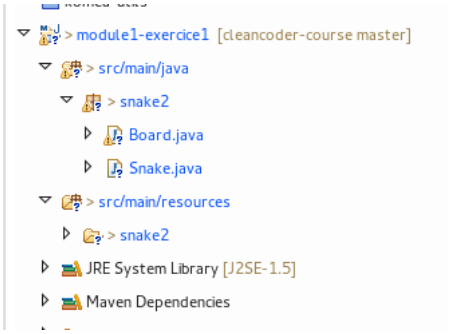


Figure 1: Structure du projet

- Ouvrez la classe **Board.java**. Astuce : utilisez le raccourci ALT + SHIFT + T et entrez le nom de la classe.
- Recherchez la méthode `appleFind` à la ligne 196

  astuce : utilisez le raccourci `CTRL+L` et tapez le numéro de la ligne

  astuce : utilisez le raccourci `CTRL+O` et tapez le nom de la méthode, finissez par `entrée`.
- Sélectionnez l'identifiant de la méthode `appleFind`.

- Activez la fonction de renommage avec le raccourci clavier `ALT + SHIFT + R`
- Vous verrez apparaître un cadre autour de l'identifiant.

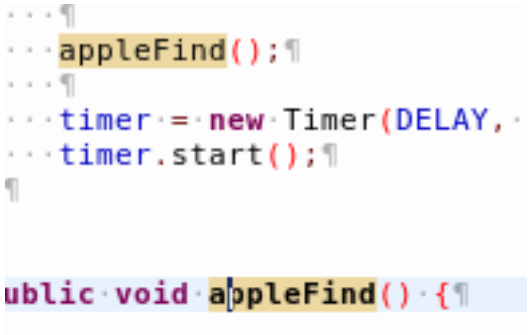


Figure 2: Activation du mode refactoring

- Il est possible de passer dans un mode de renommage avec paramétrage d'options en appuyant une seconde fois sur `ALT + SHIFT + R`.

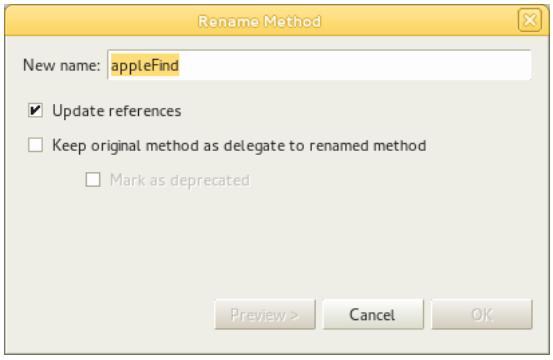


Figure 3: Activation du mode refactoring + options

- Renommez la méthode en `findApple` et pressez entrée.

Questions :

- Quel est le sens des fonctionnalités supplémentaires de configuration du refactoring ? Leur utilité ? (Expérimentez et concluez)
- Les mêmes options sont-elles proposées pour une variable, un champ, ue méthode ? Expérimentez
- Citez quelques exemples d'identifiants mal nommés, justifiez selon les règles violées.

2.2 Utilisation des expressions régulières

Une expression régulière est une chaînes de caractères que l'on appelle parfois un motif et qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise.[1]
Exemples

- `chat|chien` : correspond aux chaînes de caractères "<chat"> ou "<chien"> (et seulement à celles-ci), n'importe où dans le texte (exemple : "<chatte">).

[1] `https://fr.wikipedia.org/wiki/Expression_rationnelle`

- `[cC]hat|[cC]hien` : correspond aux chaônes "<chat">, "<Chat">, "<chien"> ou "<Chien"> (et seulement à celles-ci), n'importe où dans le texte (exemple : "<Chat"> dans "<Chats et chiens">).
- `chu+t` : correspond à "<chut">, "<chuut">, "<chuuut">, etc., n'importe où dans le texte.
- `a[ou]+` : correspond à "<aou">, "<ao">, "<auuu">, "<aououuuoou">, etc., n'importe où dans le texte.
- `peu[xt]?` : correspond à "<peu">, "<peux"> et "<peut"> (et seulement à ces chaînes), n'importe où dans le texte. La recherche retourne le texte le plus long possible en cas d'occurrences multiples à la même position.
- `^[st]ac` : représente les chaînes "<sac"> et "<tac"> en début de ligne.
- `[st]ac$` : représente les chaînes "<sac"> et "<tac"> en fin de ligne ou de texte (par exemple à l'intérieur de "<ressac">).
- `^trax$` : représente la chaîne "<trax"> seule sur une ligne.

2.2.1 Utilisez des expressions régulières dans Eclipse

- Placez vous dans la classe `Board.java`
- Localisez les champs `PREFIX_WIDTH`, `PREFIX_HEIGHT`, nous allons tenter de les renommer en même temps.
- Pressez le raccourci clavier `CTRL + F`

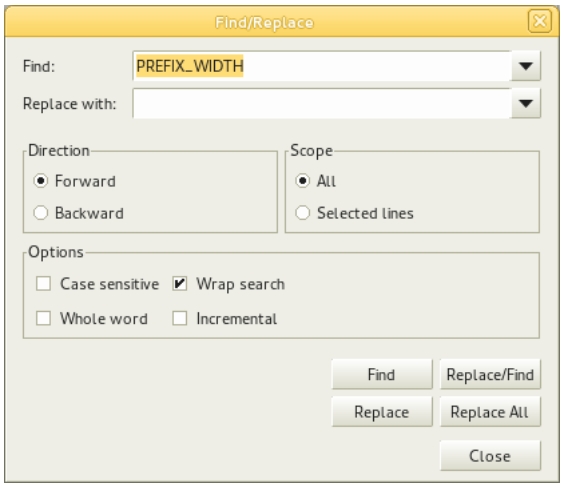


Figure 4: Fenêtre rechercher / remplacer

- Nous allons configurer la fenêtre selon l'illustration suivante :

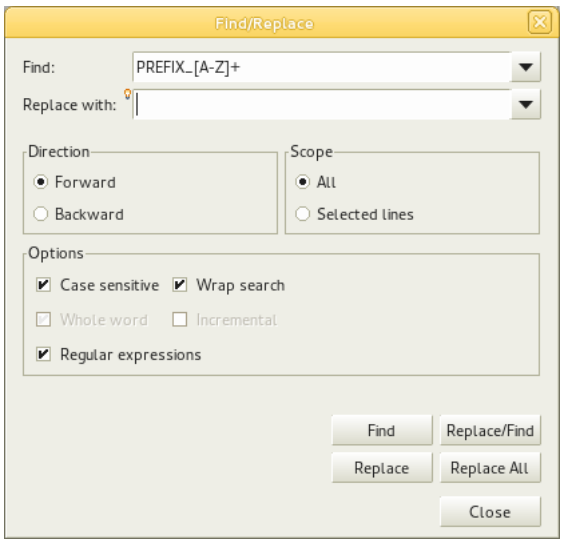


Figure 5: Configuration de l'expression régulière

- Vérifiez que nous trouvons bien (avec le bouton **Find**) les attributs `PREFIX_HEIGHT`, `PREFIX_WIDTH`.
- **Question : Expliquez l'expression régulière**

2.2.2 Remplacement avec les expressions régulières

- Reprenons la classe `Board.java`
- Pressez le raccourci clavier `CTRL + F`

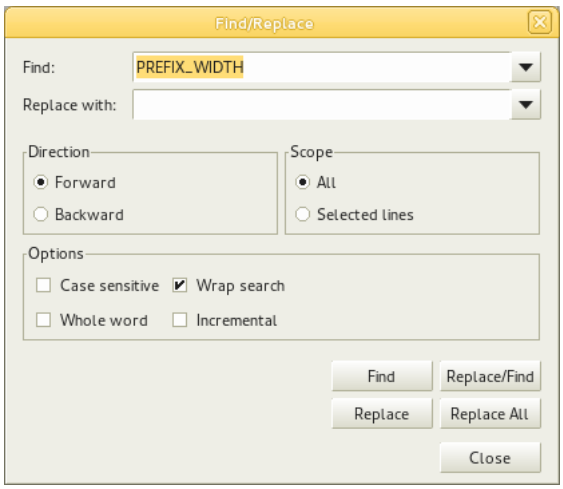


Figure 6: Fenêtre rechercher / remplacer

- Nous allons configurer la fenêtre selon l'illustration suivante :

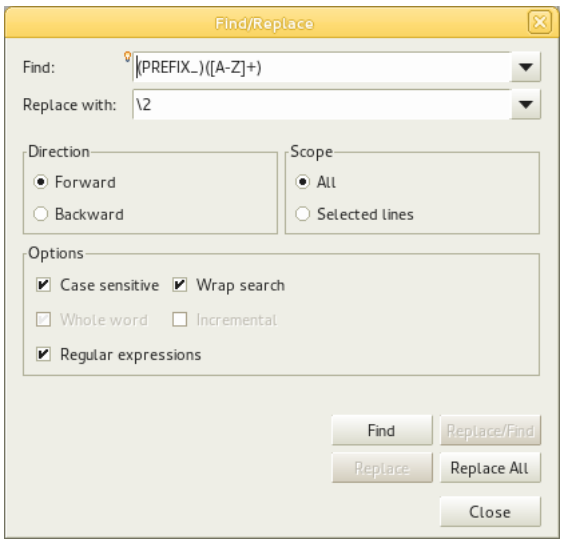


Figure 7: Configuration de l'expression régulière

- **Appuyez-sur Replace All**
- **Question :** Que constatez vous ?
- **Question :** Expliquez l'expression régulière en termes simples
- **Question :** Expliquez le sens des parenthèses
- **Question :** A quoi se réfère l'expression `2` dans la case Remplacer ?
- **Exercice** : Réalisez un remplacement par expression régulière sur les champs et `ALL_DOTS`. Nous souhaitons inverser les suffixes en préfixes.
- Exemple `SIZE_DOTS -> DOTS_SIZE`, `ALL_DOTS -> DOTS_ALL`

3 Configurez quelques règles de nommage dans Eclipse

- Ouvrez le menu **Préférences**
- Ouvrez les préférences **Java/Code style**
- Vous devriez obtenir un affichage proche de celui-ci :

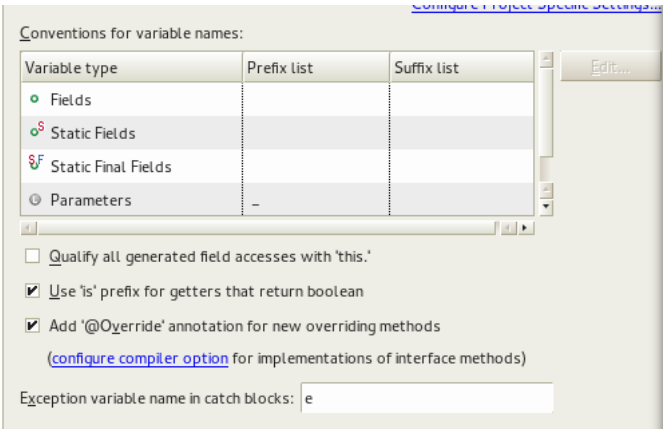


Figure 8: Paramétrage Eclipse pour le nommage

- Dans ce panneau de configuration vous pouvez définir, selon votre norme de codage, les préfixes et suffixes imposés.
- Toutefois rappelez-vous que les préfixes/suffixes pour les attributs sont **fortement déconséillés.**

4 Installer le plugin Checkstyle

Le plugin Checkstyle est un outil d'analyse du code permettant la vérification de bonnes pratiques diverses telles que le nommage, le formattage, certaines règles d'utilisation de JDK, quelques règles d'optimisations, des règles relatives aux frameworks...

> Le site de référence est : `http://checkstyle.sourceforge.net/`

Checkstyle propose entre autres, un plugin pour Checkstyle afin d'aider le développeur à contrôler ces règle de programmation.
L'installation est relativement aisée, soit en suivant les indications de la page soit directement *via le market place.*

> Le site d'installation du plugin est : `http://eclipse-cs.sourceforge.net/downloads.html`

5 Utilisez Checkstyle

5.1 Lancez une analyse

Suivez la démarche décrite ici : `http://eclipse-cs.sourceforge.net/basic_setup_project.html`

5.2 Configuration d'un référentiel

Suivez la démarche décrite ici `http://eclipse-cs.sourceforge.net/basic_creating_config.html`

5.3 Création d'un référentiel de règles

- Listez les règles disponibles via Checkstyle
- Lesquelles vous paraissent intéressantes dans le cadre de la vérification de règles de nommage ?
- Construisez un référentiel adapté et validez le sur l'application Snake.

6 Créer sa propre règle de code via Checkstyle (Expert)

- **Définissez une règle que vous souhaitez vérifier automatique**
- **Faites valider la faisabilité**
- Chargez le projet **module1/exercise2**
- Durant cet exercice, réérez vous à la page `http://checkstyle.sourceforge.net/writingchecks.html` pour plus de détails

- Executez la classe CheckstyleEditorMain, le programme qui se lance vous permet de visualiser la structure d'un fichier Java tel que Checkstyle le manipule.

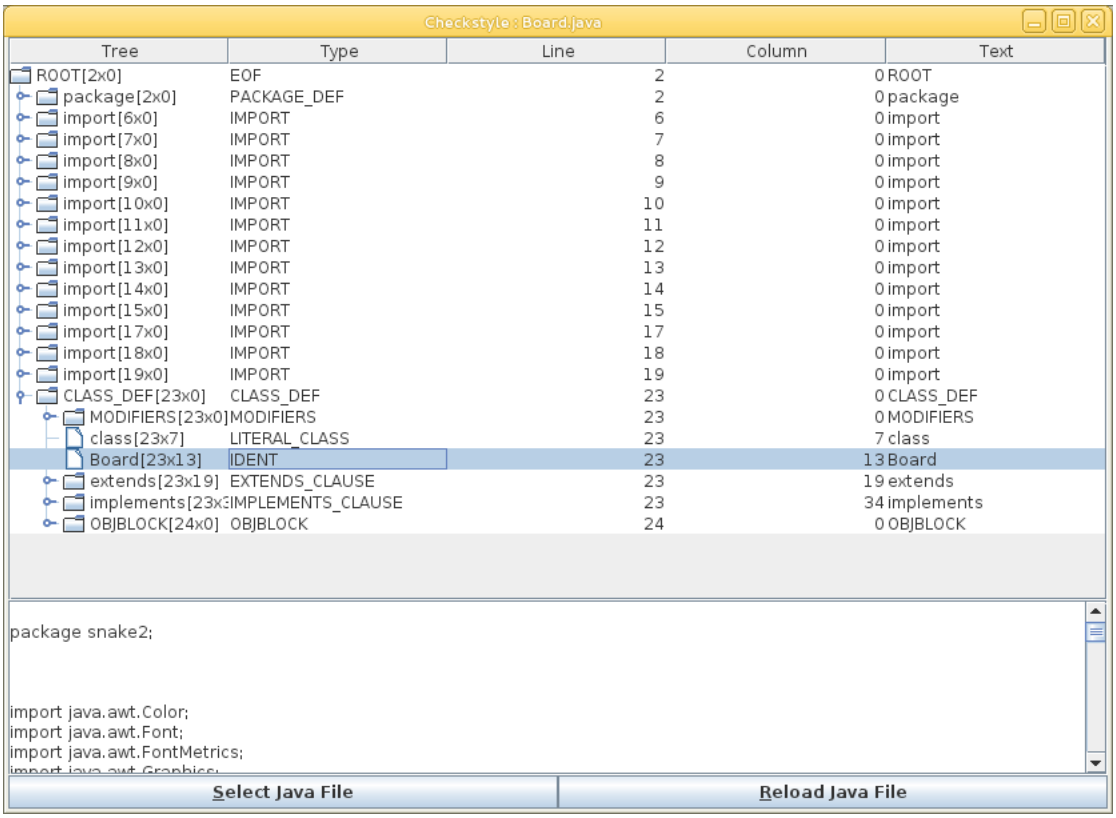


Figure 9: Checkstyle : Visualisateur de fichier Java

- Ouvrez le fichier Board.java avec le bouton se trouvant en bas de la fenêtre.

6.1 Développez votre propre règle.

Pour développer sa propre règle dans Checkstyle, il est néssaire de produire au moins deux fichiers. Nous partirons d'un exemple que nous customiserons pour nos besoins :

- Un fichier Java : `VariableMinimalSize.java` contenant l'algorithme de vérification de votre règle
- Un fichier XML `configRules.xml` contenant la déclaration des règles que vous avez créés.

Pour construire une règle, voici les étapes :

- crééez une nouvelle classe
- faites la hériter de la classe Check de Checkstyle.
- Définissez les noeuds que vous souhaitez vérifier (utilisez l'éditeur pour identifier au moins un cas dans un code source d'exemple)
- implémentez la méthode `getDefaultTokens()`

```
1   public int[] getDefaultTokens() {
2    return new int[] {
3                   TokenTypes.CLASS_DEF, TokenTypes.INTERFACE_DEF↩
                        };
4   }
```

- Pour vérifier un concept, l'algorithme doit être écrit dans la méthode :

```
1     @Override
2      public void visitToken(final DetailAST ast) {
3
4      }
```

- La méthode est appelée pour chaque noeud trouvé dont le type correspond aux valeurs fournies précédemment. L'argument correspond à chacun de ces noeuds.

- Un ensemble de méthodes fournies par la classe `DetailAst` permet de rechercher dans la structure de données, les informations nécessaires.

```
1   @Override
2      public void visitToken(final DetailAST ast) {
3
4           final DetailAST tokenVariableIdentifier =
5 ast.findFirstToken(TokenTypes.IDENT);
6           final String variableIdentifier = ↩
               tokenVariableIdentifier.getText();
7           if (isUnderRequestedSize(variableIdentifier.length()))↩
                {
8            //
9 size.");
10          }
11     }
```

- Pour reporter les erreurs, il faut utiliser la méthode `log()`.

```
1      log(tokenVariableIdentifier, "The identifier of the ↩
          variable '"
2                      + variableIdentifier + "' is under the ↩
                          minimal requested
```

- Le code final :

```
1    @Override
2      public void visitToken(final DetailAST ast) {
3
4
5           final DetailAST tokenVariableIdentifier =
6 ast.findFirstToken(TokenTypes.IDENT);
7           final String variableIdentifier = ↩
               tokenVariableIdentifier.getText();
8           if (isUnderRequestedSize(variableIdentifier.length()))↩
                {
9               log(tokenVariableIdentifier, "The identifier of ↩
                   the variable '"
10                     + variableIdentifier + "' is under the ↩
                          minimal requested
11 size.");
12          }
13
14     }
```

7 Pair-reviewing

Dans cet exercice, nous proposons à un binôme d'essayer de refactorer une méthode et de justifier les améliorations qu'ils proposent devant le reste du groupe. La rotation est ensuite réalisée avec un autre binôme.
Le code se trouve dans `module1/exercise3`