

TP de Compléments en Programmation Orientée Objet n° 1 : Gradle, objets, classes et encapsulation (Correction)

Attention : partie III) à rendre sur Moodle

À propos

Gradle : « moteur de production », sorte de GNU make évolué. Gradle est en fait plus comparable à Apache Maven. La configuration de Gradle consiste à déclarer un certain nombre de paramètres (plugins de compilation, chemin du kit de développement, dépendances, ...), puis à laisser l'outil gérer les différentes étapes pour compiler et/ou exécuter le projet (téléchargements, exécutions du ou des compilateurs et outils, ...). En cela il est assez différent de make, où l'on doit explicitement donner la suite des étapes.

Gradle est très bien supporté par les principaux environnements de développement pour Java (IntelliJ IDEA, Eclipse, NetBeans, Android Studio, ...).

I) Configurez gradle

a) Sur les machines fixes de l'UFR (pas sur les portables !)

Gradle a souvent besoin de télécharger des fichiers sur le web pour fonctionner (mises à jour de gradle, dépendances de vos projets, etc.).

Pour que cela fonctionne en salle de TP, il faut configurer le proxy HTTP(S) :

1. créez un dossier `$HOME/.gradle/` (s'il n'existe pas déjà)
2. créez un fichier `gradle.properties` dans le dossier `$HOME/.gradle/` (s'il n'existe pas déjà)
3. ajoutez-y les lignes suivantes :

```
1 systemProp.https.proxyHost=cache
2 systemProp.https.proxyPort=3128
3 systemProp.http.proxyHost=cache
4 systemProp.http.proxyPort=3128
```

(si elles n'y sont pas déjà, sinon vérifier que les clés correspondantes contiennent les bonnes valeurs)

b) Sur vos machines

Il faut que Gradle soit installé.

- Si vous avez un système de paquets (comme `apt` sous Debian/Ubuntu/...), installez gradle via le gestionnaire de paquets. Ex :

```
1 $ sudo apt install gradle
```

- Sinon vous pouvez suivre les instructions sur le site de Gradle (<https://gradle.org/install/>).

Remarque : si vous faites ainsi, vous aurez directement la bonne version de Gradle, ce qui rend inutile l'étape II) a) 2. Si vous sautez cette étape, dans la suite, vous remplacerez la commande `./gradlew` par juste `gradle`.

II) Créez un projet gradle

a) Configuration

Il s'agit en réalité de 2 étapes : demander à gradle de télécharger une version récente de lui-même et de configurer le script `gradlew` qui appelle cette dernière version ; puis d'initialiser le projet à proprement parler.

Les étapes :

1. Créez un répertoire pour votre projet et allez dedans :

```
1 $ cd $HOME/chemin/vers/mes/projets
2 $ mkdir monProjet
3 $ cd monProjet
```

2. Configurez le *wrapper* `gradlew` pour qu'il utilise la dernière version de Gradle :

```
1 $ gradle wrapper --gradle-version 5.6.2 --distribution-type BIN
```

3. Lancez le *wizard* d'initialisation de votre projet :

```
1 $ ./gradlew init
```

(`gradlew` téléchargera la version de gradle requise lors de son premier usage).

4. Répondez aux questions. Les réponses apportées dépendront du projet que vous créez.

Probablement, vous répondrez :

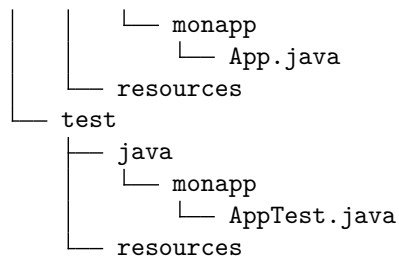
- type de projet → « application » (ce sera le cas dans ce TP) ou « library ».
- « implementation language » (en quel langage programmez vous ?) → « Java »
- « build script DSL » (quel langage utilisera gradle pour sa configuration ?) → « Groovy »¹
- « test framework » (choix du cadriciel de test) → « JUnit 4 » (pas d'importance pour ce TP).
- Nom du projet → ce que vous voulez (choisissez un nom que vous saurez reconnaître !)
- Nom du *package* de sources : il s'agit du *package* principal de ce projet. Respectez les conventions de Java (tout en minuscules!).

b) Vérifications et essais

1. Prenez ensuite le temps de regarder les dossiers et fichiers créés : (par exemple, exécutez `tree`)

```
.
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
```

1. Le DSL Groovy est plus ancien et probablement mieux documenté pour l'instant ; mais il semble que gradle préfère maintenant Kotlin ; ce choix n'est pas très important, mais pour ce TP, les exemples seront donnés en Groovy.



Quelques remarques : le code de votre projet se situera dans `src/main/java/monapp` (`monapp` étant le nom que j'ai donné à mon *package* principal). Le code de test se trouvera dans `src/test/java/monapp`.

- Ouvrez et regardez les fichiers `settings.gradle` (vérifiez que c'est dans ce fichier que le nom de votre projet est configuré) et `build.gradle` (repérez l'endroit où le nom de la classe principale à exécuter est renseigné ; repérez aussi la liste des dépendances et constatez que Gradle a ajouté par défaut une dépendance à Google Guava).
- Vous pouvez ensuite compiler et exécuter le projet par défaut pour vérifier que tout est bon :

```
1 $ ./gradlew build
2 BUILD SUCCESSFUL in 12s
3 7 actionable tasks: 7 executed
4 $ ./gradlew run
5
6 > Task :run
7 Hello world.
8
9 BUILD SUCCESSFUL in 863ms
10 2 actionable tasks: 1 executed, 1 up-to-date
```

c) Importation dans environnement de développement intégré (EDI/IDE)

Importez maintenant votre projet gradle dans votre IDE préféré². Constatez que cela est effectivement bien prévu par celui-ci, que les tâches gradle sont bien à portée de clic et que les paquetages appartenant aux dépendances déclarées dans `build.gradle` sont bien accessibles. Par exemple, ouvrez le fichier `App.java` depuis le projet importé dans votre IDE et insérez-y, au début, la ligne `import com.google.common.collect.*;`. Vérifiez que l'IDE ne signale pas d'erreur.

III) Mini-projet : gestionnaire d'archive zip

Pour illustrer Gradle, programmons maintenant un petit logiciel : un gestionnaire d'archive zip. À la différence des commandes `zip` et `unzip` sous Linux, nous voulons que notre utilitaire ne s'exécute qu'au travers d'une seule commande, avec différentes options (un peu comme les commandes `tar` et `unrar`).

Pour les besoins de l'explication, appelons notre commande `eclair`. La syntaxe d'appel serait la suivante : `eclair <commande> <arguments>`, où `<commande>` serait :

- `"-c"` : pour créer une archive zip. `<arguments>` serait alors une liste de fichiers de taille au moins 2, dont le premier élément serait le nom de l'archive zip à laquelle ajouter les fichiers nommés par les éléments restants (archive à créer si elle n'existe pas).

Exemple :

2. Cherchez comment faire !

```
1 $ eclair -c mon_archive.zip fichier1.txt fichier2.jpg fichier3.wav
```

- "-x" : pour extraire le contenu une archive zip. `<arguments>` serait alors une liste de taille exactement 2, dont le premier élément serait le nom de l'archive zip à décompresser et le second serait le dossier de destination.

Exemple :

```
1 $ eclair -x mon_archive.zip $HOME/Documents
```

Par ailleurs si aucune option n'est donnée ou si les arguments passés en ligne de commande sont invalides, la commande n'écrira aucun fichier et se contentera d'afficher une aide consistant en la liste des commandes avec leur syntaxe et leur description.

Pour réaliser ce petit logiciel, vous allez utiliser deux dépendances :

- Apache commons-cli : cette bibliothèque sert à analyser les arguments de la ligne de commande (le paramètre de la méthode `main`) en fonction d'une liste d'option préconfigurées. Cette bibliothèque sait aussi générer le message d'aide à partir de la liste des options. Documentation ici : <https://commons.apache.org/proper/commons-cli/introduction.html>.
- Zip4j : une bibliothèque pour gérer les fichiers zip. Documentation : <https://github.com/srikanth-lingala/zip4j>.

Étapes :

1. créez un projet comme expliqué précédemment
2. éditez le fichier `build.gradle`
3. allez à la section `dependencies`, retirez³ la dépendance à Google Guava qui avait été mise par défaut (supprimez la ligne)
4. ajoutez les dépendances aux bibliothèques commons-cli et Zip4j. Pour cela, insérez les lignes suivantes à la section `dependencies` :

```
implementation 'net.lingala.zip4j:zip4j:2.1.3'
implementation 'commons-cli:commons-cli:1.4'
```

5. Constatez, à la prochaine exécution de Gradle (essayez avec `./gradlew build`), que Gradle télécharge bien ces deux bibliothèques.
6. Programmez votre logiciel en utilisant ces bibliothèques, c'est à dire en important des classes (`import`) et des méthodes (`import static`) des paquetages `net.lingala.zip4j` et `org.apache.commons.cli` dans vos fichiers `.java` qui en ont besoin.
7. Constatez que, quand vous compilez (`./gradlew build`) ou exécutez votre projet (`./gradlew run`) via gradle, Java trouve bien ces paquetages, ou bien que l'éditeur de votre IDE ne signale pas d'erreur.
8. Recherchez comment générer un fichier `.jar` pour votre logiciel, à l'aide de Gradle.

Indications :

- Pour passer des arguments à `main` quand vous exécutez votre programme via la tâche `run` de Gradle, vous pouvez utiliser le paramètre `--args`⁴. Exemple :

```
1 ./gradlew run --args="-x uneArchive.zip dest"
```

3. Sinon cherchez à quoi sert Guava. Laissez la dépendance si vous pensez que ça vous servira dans ce projet !

4. Attention : il est possible qu'un bug de `gradlew` empêche actuellement d'échapper des guillemets dans le paramètre de `--args`. Évitez donc de tester votre commande sur des fichiers dont les noms contiennent des espaces.

IV) Objets, classes, encapsulation

Exercice 1 : Encapsulation et sûreté

Voici deux classes avec leur spécification. Pour chaque cas :

- soit la spécification est satisfaite par la classe, dans ce cas, justifiez-le ;
- soit la spécification n'est pas satisfaite, dans ce cas écrivez un programme qui la met en défaut (sans modifier la classe fournie), puis proposez une rectification de la classe.

1.

```
public class EvenNumbersGenerator {
    static int MAX = 42;
    public int previous = 0;
    public int next() {
        previous += 2; previous %= MAX;
        return previous;
    }
}
```

Spécification : la méthode `next` ne retourne que des entiers pairs.

2.

```
public class VectAdditioner {
    private Point sum = new Point();

    public void add(Point p) {
        sum.x += p.x; sum.y += p.y;
    }

    public Point getSum() {
        return sum;
    }
}
```

Spécification : la méthode `getSum` retourne la somme de tous les vecteurs qui ont été passés en paramètre par la méthode `add` depuis l'instanciation.

Correction :

1. `EvenNumbersGenerator` fait ce qu'elle promet tant que l'attribut `previous` n'est pas modifié depuis l'extérieur.

Par exemple :

```
1 EvenNumbersGenerator eng = new EvenNumbersGenerator();
2 eng.previous = 1;
3 System.out.println(eng.next()); // affiche 3
```

Une attaque similaire est possible en modifiant `MAX`.

La solution : ajouter `private` et/ou `final` à la ligne 2 et remplacer `public` par `private` à la ligne 3 empêche cela.

Après cette modification, on a la garantie que `MAX` ne sera plus modifiée et que `previous` ne sera modifiée que par la méthode `next`. Or cette dernière semble bien programmée, vu qu'elle ne fait qu'appliquer les opérations $+ 2$ et modulo 42 à l'attribut `previous` (opérations qui préservent la parité).

(Remarques en avance sur le cours :) Cela dit, si cette méthode est exécutée 2 fois en même temps (sur plusieurs *threads*), les différentes opérations peuvent s'entrelacer. En l'occurrence, ici, cela ne casserait pas la garantie de parité. Par ailleurs, des accès en compétition (i.e. : lecture et écriture d'une même variable depuis des *threads* différents, sans synchronisation) auraient lieu, ce qui peut avoir de mauvaises conséquences (cf. cours sur la concurrence, à venir).

Ajouter `volatile` devant la déclaration de `previous` règle ce dernier problème, sans pour autant empêcher les entrelacements bizarres. Pour garantir des propriétés plus fortes que la parité, il faudrait recourir à d'autres primitives de synchronisation, comme `synchronized`, afin d'obtenir d'atomicité de ce calcul.

2. Ici, le problème, c'est que malgré le `private` on peut malgré tout obtenir, par appel à `getSum()` une copie de la référence contenue dans l'attribut `sum`.

Si on fait :

```
1 VectAdditioner va = new VectAdditioner();
2 va.getSum().x = -12;
3 System.out.println(va.getSum());
```

Alors s'affichera `(-12 0)`, alors même qu'on n'a pas encore appelé `add`.

La solution : ne pas partager de référence. Pour cela, modifier la méthode `getSum` afin qu'elle retourne une référence vers un nouvel objet au lieu d'une simple copie de `sum` :

```
1 public Point getSum() {
2     return new Point(sum.x, sum.y);
3 }
```

(Remarques en avance sur le cours :) Là encore, il reste des problèmes liés à la concurrence. Si deux appels à `add` s'entrelaçaient, de l'information pourrait être perdue. Pour corriger, il suffit de rendre `add` atomique en ajoutant `synchronized` à sa déclaration. Il faudrait aussi ajouter `synchronized` à `getSum` pour s'assurer que le dernier appel à `add` est terminé quand on copie `sum`.

Toutes les considérations en rapport à la programmation concurrente seront discutées plus tard dans le cours. Le problème n'est mentionné dans ce corrigé que dans un souci d'exactitude.

Exercice 2 : Nombres complexes

Pour le vocabulaire, référez vous à https://fr.wikipedia.org/wiki/Nombre_complexe.

- Écrivez une classe `Complexe`, avec :
 - les attributs (`double`) : parties réelle et imaginaire du nombre (`static` ou pas ?);
 - le constructeur, prenant comme paramètres les parties réelle et imaginaire du nombre;
 - la méthode `public String toString()`, permettant de convertir un complexe en chaîne de caractères lisible par l'humain;
 - les opérations arithmétiques usuelles (somme, soustraction, multiplication, division);
 - le test d'égalité (méthode `public boolean equals(Object other)`);
 - les fonctions et accesseurs spécifiques aux complexes : partie réelle, partie imaginaire, conjugaison, module, argument...

Vous pouvez vous aider des fonctionnalités de génération de code de votre IDE.

Attention, style demandé : attributs non modifiables (si vous savez le faire, faites une vraie classe immuable), les opérations retournent de nouveaux objets.

- Ajoutez à votre classe :
 - des attributs (constants : vous pouvez ajouter `final`) pour les valeurs les plus courantes du type `Complexe` : à savoir 0, 1 et i (le nombre i tel que $i^2 = -1$). Ces attributs doivent-ils être `static` ou non ?
 - une méthode (fabrique statique)

```
1 public static Complexe fromPolarCoordinates(double rho, double theta)
```

qui construit un complexe depuis son module ρ et son argument θ (on rappelle que la partie réelle vaut alors $\rho \cos \theta$ et la partie imaginaire $\rho \sin \theta$).

Remarquez que cette méthode joue le rôle d'un constructeur. Pourquoi ne pas avoir fait plutôt un autre constructeur alors ? (essayez de compiler avec 2 constructeurs puis expliquez pourquoi ça ne marche pas)

Correction :

```
1
2 public final class Complexe {
3
4     /*
5      * on pourrait presque mettre les attributs en public sans dégâts dans ce
6      * cas (final), mais faire ainsi nous empêcherait dans des évolutions
7      * futures de changer la représentation interne, donc nous allons quand-même
8      * définir les getteurs.
9      */
10    public final double re, im;
11    public final static Complexe I = new Complexe(0, 1);
12
13    public double getRe() {
14        return re;
15    }
16
17    public double getIm() {
18        return im;
19    }
20
21    /*
22     * Constructeur privé pour forcer à utiliser les méthodes fabrique statiques
23     */
24    private Complexe(double re, double im) {
25        this.re = re;
26        this.im = im;
27    }
28
29    @Override
30    public String toString() {
31        return "(" + re + " + " + im + "i)";
32    }
33
34    /*
35     * NB: il faudrait normalement redéfinir en même temps la méthode
36     * hashCode() de façon cohérente (EJ 3 Item 11).
37     */
38    @Override
39    public boolean equals(Object autre) {
40        if (autre == null || !(autre instanceof Complexe)) return false;
41        Complexe autreComplexe = (Complexe) autre;
42        return re == autreComplexe.re
43            && im == autreComplexe.im;
44    }
45
46    public Complexe plus(Complexe autre) {
47        return new Complexe(re + autre.re, im + autre.im);
48    }
49
50    public Complexe moins(Complexe autre) {
51        return new Complexe(re - autre.re, im - autre.im);
52    }
53
54    public Complexe fois(Complexe autre) {
55        return new Complexe(re * autre.re - im * autre.im, re * autre.im + im *
56            autre.re);
57    }
58
59    public Complexe divisePar(Complexe autre) {
60        if (autre.egale(ZERO))
61            throw new ArithmeticException("Division by zero.");
```

```

61         double d = autre.re * autre.re + autre.im * autre.im;
62         double r = re * autre.re;
63         double i = +im * autre.im;
64         return new Complexe((r + i) / d, (r - i) / d);
65     }
66
67     public Complexe conjugue() {
68         return new Complexe(re, -im);
69     }
70
71     public double module() {
72         return Math.sqrt(re * re + im * im);
73     }
74
75     public double argument() {
76         if (re == 0) {
77             if (im == 0) return Double.NaN;
78             else if (im < 0) return -Math.PI / 2;
79             else return Math.PI / 2;
80         }
81         else if (re > 0) return Math.atan(im / re);
82         else if (im >= 0) return Math.atan(im / re) + Math.PI;
83         else return Math.atan(im / re) - Math.PI;
84     }
85
86     private static final Complexe ZERO = new Complexe(0, 0);
87
88     private static final Complexe UN = new Complexe(1, 0);
89
90     private static final Complexe I = new Complexe(0, 1);
91
92     public static Complexe fromPolarCoordinates(double rho, double theta) {
93         return new Complexe(rho * Math.cos(theta), rho * Math.sin(theta));
94     }
95
96     /*
97     * Ne fait rien de plus que le constructeur, mais c'est plus équilibré de
98     * mettre les deux modes de construction sur un pied d'égalité dans l'API.
99     */
100    public static Complexe fromRealAndImaginary(double re, double im) {
101        return new Complexe(re, im);
102    }
103 }

```

- Améliorez l'encapsulation de votre classe, afin de permettre des évolutions ultérieures sans « casser » les clients/dépendants de celle-ci : en l'occurrence, les attributs doivent être privés et des accesseurs publics⁵ doivent être ajoutés pour que la classe reste utilisable.
- Testez en écrivant un programme (méthode `main()`, dans une autre classe), qui fait entrer à l'utilisateur une séquence de nombre complexes et calcule leur somme et leur produit. Améliorez le programme pour permettre la saisie des nombres au choix, via leurs parties réelles et imaginaires ou via leurs coordonnées polaires.
- Écrivez une version « mutable » de cette classe (il faut donc des méthodes `set` pour chacune des propriétés).

Changez la signature⁶ et le comportement des méthodes des opérations arithmétiques

5. Remarquez qu'il n'y a pas de raison de favoriser le couple parties réelle/imaginaire par rapport au couple module/argument (les deux définissent de façon unique un nombre complexe); et qu'il faut donc considérer ce dernier couple comme un couple de propriétés, pour lequel il faudrait utiliser aussi la notation `get`. Ainsi, cette classe aurait 4 propriétés (peu importe si elles sont redondantes : les attributs ne le sont pas; cela limite les risques d'incohérences).

6. Si la méthode déjà programmée est `static` rendre non statique et enlever un paramètre; dans tous les cas

afin que le résultat soit enregistré dans l'objet courant (**this**), plutôt que retourné.