Exploration

POO - Comprendre l'encapsulation

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
2024-04-12	Rectification oubli setX() et setY() + effacement pages inutiles

Sommaire

Nomenclature :

- **Assimiler** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découvrir** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Voyager** : Projet en autonomie totale. Environnement ouvert : Vous êtes le capitaine !

A Introduction

La programmation orientée objet utilise des concepts qui facilitent la programmation et la gestion de projets un peu complexes.

Ainsi, l'usage d'une classe doit faciliter la vie du développeur, tout en garantissant que les usages resteront sécurisés. Pour cela, une classe encapsule des membres (les attributs et les méthodes) et les protège des lectures ou écritures en définissant la portée qui convient.

A.1 Présentation

Cette activité vise à mettre en œuvre ce mécanisme : nous allons créer une classe PointGPS qui aura quatre attributs (X, Y, Z et une coordonnée What3words).

A.1.1 Le système What3words

What3words est un système original de coordonnées utilisant trois mots. Ainsi, les coordonnées terrestres ont été découpées en blocs de 3m x 3m et chaque bloc reçoit un ensemble de trois mots uniques.

L'image ci-dessus montre un des blocs devant l'entrée du lycée Bergès : vrai.supposons.salin.

A.1.2 Le système GPS

Vous connaissez bien le système GPS (Global Positioning System) qui utilise trois points, compris entre -180 et +180 pour la coordonnée X (degré), -90 à +90 pour la coordonnée Y (degré) et une altitude pour la coordonnée Z.

Pour la même place que ci-dessus, on trouve environ : Latitude : 45.185081 | Longitude : 5.693577

A.2 Prérequis

- Avoir lu le cours et la vidéo sur les bases de la programmation orientée objet
- Savoir coder de manière basique (variables, itérations, conditions)
- Connaître les fonctions avec paramètres et retour
- Comprendre ce qu'est une API

A.3 Informations complémentaires

Dans cette activité, un objectif secondaire est l'usage d'une API en Java.

B Classe PointGPS

B.1 Création du projet

Nous avons besoin de créer un projet en Java, que nous appelerons "PositionProject" et dont l'objectif est de fournir des informations sur un ensemble d'information de position.

B.2 Création de la classe PointGPS

À l'intérieur du dossier source, nous allons créer un dossier "model" et y créer un fichier Java PointGPS.java

PointGPS.java

```
package model;

public class PointGPS {
   // création de la classe ici...
}
```

B.2.1 Création des attributs

Nous voulons que les manipulations sur les coordonnées soient sécurisées : en effet, nous ne pouvons pas autoriser une saisie directe sur un objet, qui permettrait d'écrire une valeur fausse, par exemple :

```
PointGPS paris = new PointGPS();
paris.x = 3952.254681;  // valeur non comprise entre -180 et 180°
```

Il faut donc rendre les attributs inaccessibles depuis l'extérieur de la classe : pour masquer les attributs aux utilisateurs, il faut utiliser la portée `private` ou `protected`.

D'autre part, nous avons déjà indiqué que nous voulions les attributs x, y, z et what3words mais nous allons ajouter un attribut ville (une chaîne de caractères) que nous rendrons accessible, pour bien comprendre le mécanisme d'encapsulation. Il faudra donc utiliser la portée `public` pour cet attribut.

Sauriez-vous rédiger les attributs en Java ? La solution est visible sur la page suivante.

Voici donc les attributs de notre classe PointGPS :

PositionGPS.java

```
public class PointGPS {
    private double x;
    private double y;
    private double z;
    private String what3words;
    public String ville;

}
```

Nous pouvons tester notre classe dans la partie principale de notre projet (le fichier main.java).

Main.java

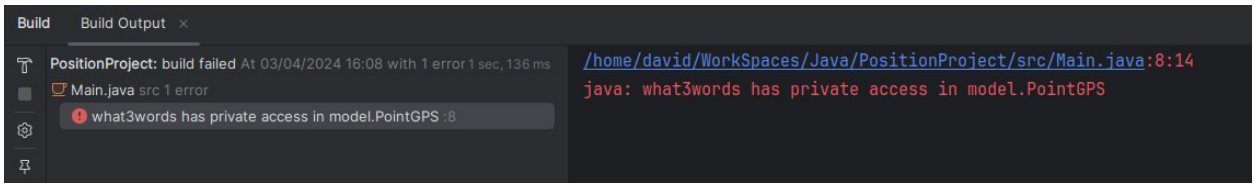
```
import model.PointGPS;

public class Main {
    public static void main(String[] args) {
        // Création d'un point et modification de l'attribut 'ville'
        PointGPS paris = new PointGPS();
        paris.ville = "Paris";
        System.out.println("Ville : "+paris.ville);
    }
}
```

Nous allons maintenant vérifier qu'il n'est pas possible de modifier l'attribut what3words depuis la classe Main (qui est donc utilisatrice de la classe PointGPS).

Ajoutez sous `paris.ville = "Paris" ;` le code suivant :

```
paris.what3words = "goûtons.baisser.tapant";
```

L'IDE doit désormais vous signaler une erreur si vous lancez le programme. Supprimez la ligne fautive.



Pour pouvoir accéder aux attributs privés, il faut créer des méthodes publiques qui auront la capacité de modifier les valeurs, après quelques vérifications.

Par exemple, l'attribut what3words est une chaîne de caractères qui contient trois mots (de deux caractères au minimum) séparés par un point. Une expression régulière peut tester facilement ce critère de validation.

Essayez de trouver l'expression régulière, par vous-même : la solution arrive sur la prochaine page…

B.3 Accesseur et mutateur (getter et setter)

L'expression régulière semble assez simple : `^[a-zA-Z]{2,}\.[a-zA-Z]{2,}\.[a-zA-Z]{2,}$` qui permet d'avoir deux caractères par mot au minimum et la séparation par deux points entre les mots.

Cependant, nous allons utiliser une expression régulière prenant en compte les diacritiques (les caractères normaux recevant un accent, un tréma ou une cédille) : `^\p{L}{2,}\.\p{L}{2,}\.\p{L}{2,}$`.

Voici le code à utiliser dans la classe PointGPS, avec notamment l'usage d'une expression régulière.

PointGPS.java

```
package model;
import java.util.regex.*;

public class PointGPS {
    private double x;
    private double y;
    private double z;
    private String what3words;
    public String ville;

    public String getWhat3words() {
        return this.what3words;
    }
    public void setWhat3words(String what3words) {
        String regex = "^\\p{L}{2,}\\.\\p{L}{2,}\\.\\p{L}{2,}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(what3words);
        if (matcher.matches()) {
            this.what3words = what3words;
            }
    }
}
```

Si vous testez le programme, il doit fonctionner correctement :

```
Ville : Paris
what3words : goûtons.baisser.tapant
```

Si vous tenez de changer la chaîne what3words par "g.baisser.tapant" le programme devrait afficher `null` à la place de la chaîne insérée (l'expression régulière n'étant pas vraie, il n'y aura pas d'affectation).

En utilisant simplement des conditions sur les valeurs, créez les accesseurs et mutateurs pour les attributs x, y. Concernant z, il s'agit d'une altitude, on prendra l'altitude maximal de 8849 mètres pour le mont Everest et -10972 mètres pour la fosse des Mariannes (dans l'océan Pacifique, aussi appelé Challenger Deep).

Le résultat sur la page suivante...

Voici les accesseurs et mutateurs manquants dans la classe PointGPS :

```
    public double getX() {
        return x;
    }

    public void setX(double x) {
        if (x >= -180 && x <= 180) {
            this.x = x;
        }
    }
    public double getY() {
        return y;
    }

    public void setY(double y) {
        if (y >= -90 && y <= 90) {
            this.y = y;
        }
    }

    public double getZ() {
        return z;
    }

    public void setZ(double z) {
        if (z >= -10972 && z <= 8849) {
            this.z = z;
        }
    }
```

On peut associer les accesseurs et les mutateurs à l'idée de permissions sous Linux : RWX

- Read pour un accesseur public
- Write pour un mutateur public

Cependant, la puissance réside dans les contrôles qui sont inclus dans les méthodes, permettant aux utilisateurs de n'utiliser nos classes que de la manière dont nous l'avons prévu.

Attention, un attribut privé au sein de la classe peut être directement modifié, sans aucun contrôle à l'intérieur de celle-ci !

Ainsi, une méthode interne à la classe peut écrire des valeurs hors des limites. Insérez par exemple, la méthode suivante dans la classe PointGPS :

```
    public void setTousLesAttributs() {
        this.x = 2000d;
        this.y = 9999.99999d;
        this.z = 30000.0d;
        this.what3words = "a.b.c.d.e.f";
    }
```

Puis dans le programme principal, ajouter la ligne ci-dessous avant les instructions d'affichage :

```
paris.setTousLesAttributs();
```

Que va afficher le programme ?

Le résultat montre qu'à l'intérieur de la classe, tout est autorisé : c'est au programmeur de gérer la sécurité des informations contenues par la classe.

```
Ville : Paris
what3words : a.b.c.d.e.f
```

B.4 Ajout d'un constructeur

Enfin, notre classe utilise le constructeur par défaut : Java considère qu'il n'y a pas de constructeur déclaré, il affecte donc uniquement la taille de la mémoire nécessaire pour les attributs (les String sont des types spéciaux et sont gérés différemment, voir https://www.baeldung.com/java-string-pool) : les types primitifs et les pointeurs.

Les constructeurs définis par le développeur permettent d'affecter les attributs avec des valeurs transmises en paramètres.

Voici donc un constructeur qui respecte les consignes lors de l'instanciation.

```
public PointGPS(double x, double y, String ville) {
    this.setX(x);
    this.setY(y);
    this.ville = ville;
    // il doit être possible de trouver what3words avec des coordonnées GPS...
}
```

En effet, il est possible de ne pas affecter de valeur directement (`this.x = x;`) et d'utiliser les mutateurs.

Note : plutôt que de choisir d'affecter une valeur uniquement si elle est dans un intervalle défini, il est possible de définir les mutateurs avec un modulo. Ainsi, Quelle que soit la valeur transmise, on affecte bien une valeur dans l'intervalle voulu. Dans une véritable application, c'est un point que les développeurs demanderaient au client.

On peut ajouter autant de constructeur que l'on veut, mais attention : l'ordre des types et le nombre de paramètres ne doivent pas être identiques.

Un second constructeur `PointGPS(double x, double z, String what3words)` ne pourrait exister puisque notre premier constructeur reçoit également trois paramètres avec deux doubles et une chaîne (dans le même ordre que le premier constructeur) : il est impossible pour le compilateur de savoir quel constructeur est appelé, il génère donc une erreur.

Pour que notre code continue à fonctionner, il faut créer un constructeur sans paramètre.

```
public PointGPS() {
}
```

C Héritage avec deux classes

Si la logique de la classe PointGPS est correcte, son utilité va au-delà du simple point GPS, qui pour rappel, ne devrait avoir que deux ou trois attributs : what3words n'est pas un attribut classique d'une coordonnées GPS, de même que le nom de la ville.

Nous allons donc "sortir" les coordonnées GPS dans une classe que nous appelerons GPS.

C.1 Création de la classe GPS

La classe GPS va reprendre une grande partie du code déjà présent dans la classe PointGPS, mais voici la classe entière, pour simplifier votre saisie.

Dans le paquetage "model", créez une classe GPS.java et copiez le code ci-dessous :

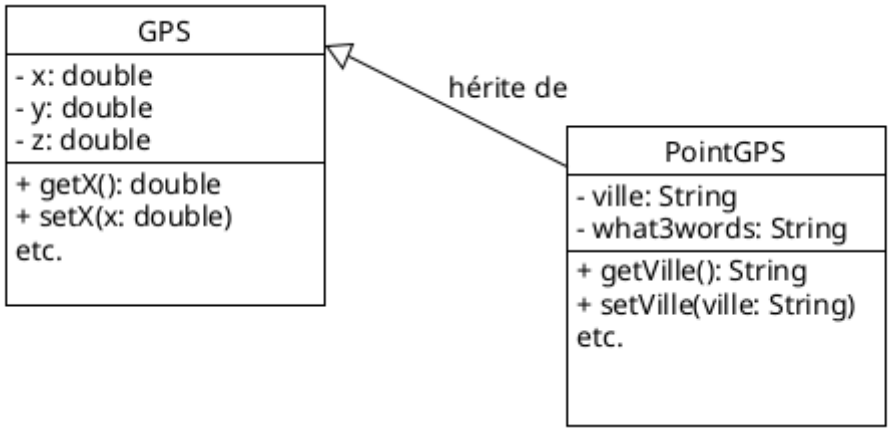
```
package model;

public class GPS {
    protected double x;
    protected double y;
    protected double z;

    public GPS(double x, double y, double alt) {
        this.x = x;
        this.y = y;
        this.z = alt;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        if (x >= -180 && x <= 180) {
            this.x = x;
        }
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        if (y >= -90 && y <= 90) {
            this.y = y;
        }
    }

    public double getZ() {
        return z;
    }

    public void setZ(double z) {
        if (z >= -10972 && z <= 8849) {
            this.z = z;
        }
    }
}
```

Nous devons maintenant supprimer le code inutile dans la classe PointGPS.

C.2 Modification classe PointGPS

C.2.1 héritage

Nous pouvons utiliser la classe GPS qui dispose de 3 points X, Y et Z.



Il suffit d'ajouter `extends` suivi du nom de la classe mère à la déclaration de la classe `PointGPS` :

```
public class PointGPS extends GPS {
```

Cela se lit "*la classe PointGPS étend les fonctionnalités de la classe GPS*".

C.2.2 Suppression des membres redondants

Nous supprimons toutes les références à x, y, z, y compris les accesseurs et mutateurs.

```
package model;
import java.util.regex.*;

public class PointGPS extends GPS {
    private String what3words;
    public String ville;

    // Constructeur
    public PointGPS(double x, double y, String ville) {
        this.setX(x);
        this.setY(y);
        this.ville = ville;
        // il doit être possible de trouver what3words avec des coordonnées GPS...
    }
    public PointGPS() {

    }
    private String getwhat3wordsFromGPS(double x, double y) {
        //What3WordsV3 what3words = What3Words.withApiKey("VOTRE_CLE_API");
        return "";
    }
    public String getWhat3words() {
        return this.what3words;
    }
    public void setWhat3words(String what3words) {
        String regex = "^\\p{L}{2,}\\.\\p{L}{2,}\\.\\p{L}{2,}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(what3words);
        if (matcher.matches()) {
            this.what3words = what3words;
        }
    }

    public void setTousLesAttributs() {
        this.x = 0;
        this.y = 0;
        this.z = 0;
        this.what3words = "";
        this.ville = "";
    }
}
```

Note : pour le moment, le constructeur de la classe PointGPS est souligné en rouge par l'éditeur, car il faut expliquer à Java qu'une partie du constructeur utilisera le constructeur GPS.

C.2.3 Constructeur et héritage

Puisque PointGPS hérite de GPS, il faut dire à PointGPS de remplir les attributs de la classe mère GPS.

Il faut en effet corriger le constructeur de PointGPS pour utiliser le constructeur de GPS dedans, à l'aide de la commande super(). Cette méthode transmet à la classe mère, les paramètres qu'elle contient :

```
public PointGPS(double x, double y, String ville) {
    super(x, y, 0);
    this.ville = ville;
    this.what3words = "";
    // il doit être possible de trouver what3words avec des coordonnées GPS...
}
```

Cette fois, le constructeur de PointGPS doit fonctionner correctement, mais il faut corriger le constructeur vide.

Saurez-vous corriger le constructeur vide ?

Essayez avant de lire la suite…

Un exemple simple de correction :

```
public PointGPS() {
    super(0,0,0);
    this.ville = "";
    this.what3words = "";
}
```

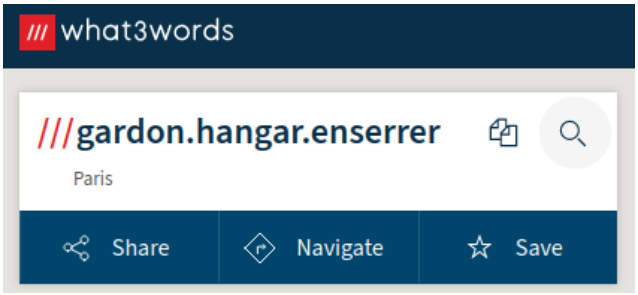
D Utilisation de l'API what3words

Il est intéressant de montrer qu'avec les coordonnées GPS, nous pouvons trouver automatiquement une coordonnée what3words, en utilisant un service sur Internet.

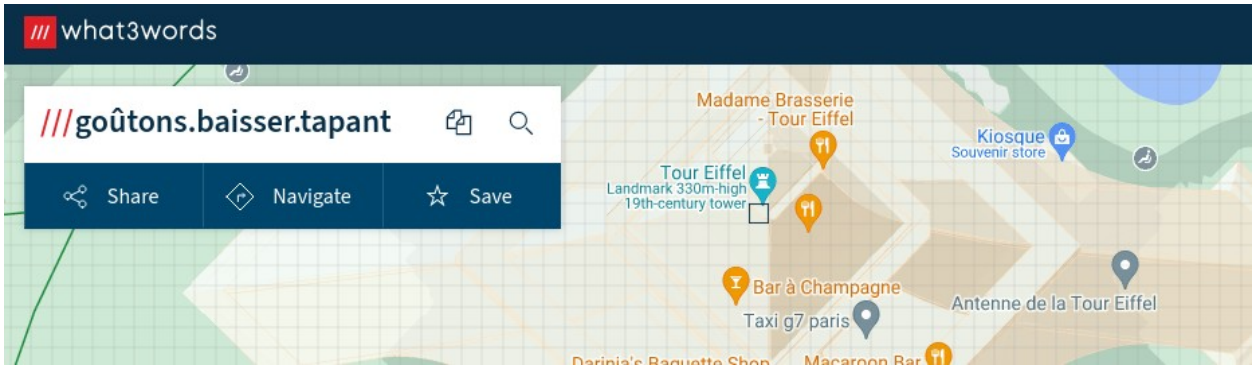
D.1 Méthode manuelle

Voici les coordonnées de la tour Eiffel à Paris : 48.8583706025918, 2.294476876782637

En se connectant sur le site https://what3words.com/ et en cliquant sur la loupe pour rechercher une position, coller les valeurs ci-dessus, juste séparées par une virgule.



Normalement, vous devriez retrouver notre coordonnée **goûtons.baisser.tapant**.



Cependant, la recherche est plutôt fastidieuse et nous pourrions utiliser un service "offert" par what3words, qu'on appelle une API.

Jetez un œil sur le lien https://developer.what3words.com/public-api et déterminez de quoi nous aurons besoin pour la suite de l'activité.

D.2 Méthode par API

L'API de what3words est gratuite pour 100 demandes de conversion en coordonnées GPS par mois, mais le nombre de conversion vers une coordonnée what3words est illimité.

Il faut cependant s'enregistrer sur le site : consulter bien les documentations, les données personnelles sont correctement gérées (2023) bien que l'entreprise soit Britannique.

Lorsque vous obtenez votre clé, il faudra modifier fondamentalement notre code et intégrer l'API dans celui-ci. Pour cela…

D.2.1 Qu'est-ce que maven ?

Notre projet est actuellement en mode Java ou IntelliJ : s'est l'utilisation la plus simple du SDK, mais comme beaucoup de langages, il est devenu fréquent d'avoir besoin d'intégrer des dépendances ou modules. Dès lors, on privilégie l'utilisation d'un outil ayant accès à une grande bibliothèque de dépendances.

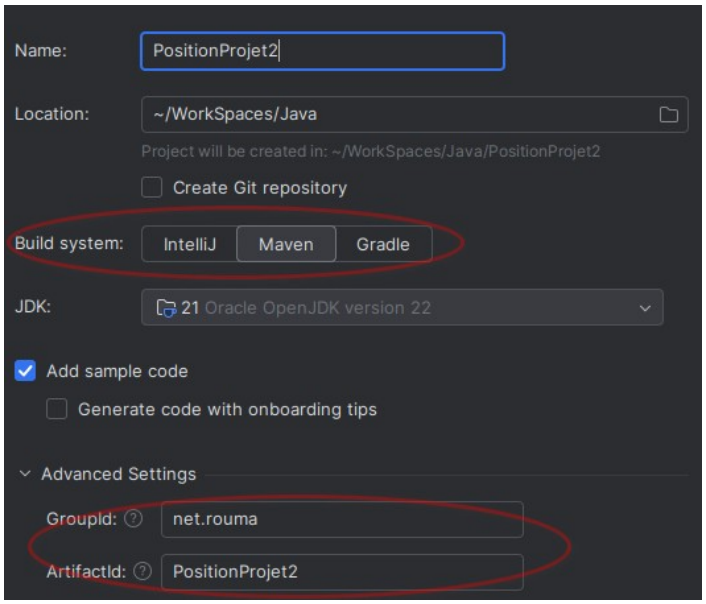
- En Node.JS l'un d'eux s'appelle NPM
- En Java, il existe Gradle ou Maven

Dans cette activité, c'est donc **maven** qui va faciliter notre vie. Ce dont nous avons besoin pour notre projet doit être décrit dans un fichier nommé `pom.xml` à la racine du projet. Avant de compiler le projet de manière classique, il est alors préférable de lancer maven :

Ce dernier analyse le fichier `pom.xml` et va télécharger pour nous les dépendances.

D.2.2 Recréer notre projet sous Maven

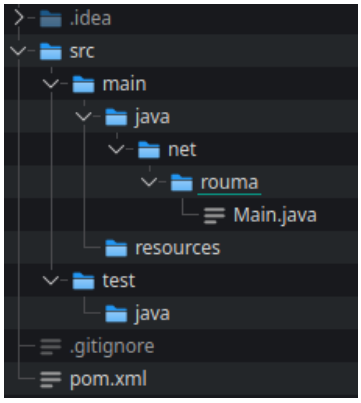
Nous allons créer un nouveau projet que nous appellerons `PositionProject2`, en choisissant le build system Maven.



Pour votre `GroupeId`, il s'agit du domaine inversé de votre entreprise, dans mon cas, `net.rouma`.

La structure imposée est différente des projets habituels :

- Dans le répertoire `main`, on trouve un répertoire `java` et toute l'arborescence du nom de l'entreprise retenue, ainsi qu'un répertoire `ressources`.
- Un répertoire `test` avec un sous répertoire `java`.
- Un fichier `.gitignore` pour permettre les échanges avec GIT.
- Un fichier `pom.xml`.



Dans l'aspect de l'éditeur IntelliJ, un **m** doit apparaître en haut à droite, qui permet d'accéder aux commandes maven.

Si vous double-cliquez sur le fichier Main.java, une seule ligne devrait vous surprendre :

```
package net.rouma;
```

Il s'agit (en quelque sorte) du chemin pour atteindre ce fichier, depuis le répertoire java.

Remplacez le contenu avec le code suivant :

Main.java

```
package net.rouma;

public class Main {
    public static void main(String[] args) {
        // Création d'un point et modification de l'attribut 'ville'
        PointGPS paris = new PointGPS();
        paris.ville = "Paris";
        System.out.println("Ville : "+paris.ville);
        System.out.println("what3words : "+paris.getWhat3words());

        PointGPS grenoble = new PointGPS(45.188529, 5.724524, "Grenoble");
        System.out.println("Grenoble : "+grenoble.getWhat3words());
    }
}
```

Sauvegardez et créez une nouvelle classe GPS dans le même paquetage (répertoire). Copiez le contenu de votre classe GPS du projet initial vers cette classe :

GPS.java

```
package net.rouma;

public class GPS {
    protected double x;
    protected double y;
    protected double z;

    public GPS(double x, double y, double alt) {
        this.x = x;
        this.y = y;
        this.z = alt;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        if (x >= -180 && x <= 180) {
            this.x = x;
        }
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        if (y >= -90 && y <= 90) {
            this.y = y;
        }
    }

    public double getZ() {
        return z;
    }

    public void setZ(double z) {
        if (z >= -10972 && z <= 8849) {
            this.z = z;
        }
    }
}
```

Essayez d'adapter correctement la nouvelle classe PointGPS, avant de vérifier la solution ci-après.

PointGPS.java

```
package net.rouma;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PointGPS extends GPS{
    private String what3words;
    public String ville;

    // Constructeur
    public PointGPS(double x, double y, String ville) {
        super(x, y, 0);
        this.ville = ville;
        this.what3words = what3words;
        // il doit être possible de trouver what3words avec des coordonnées GPS...
    }
    public PointGPS() {
        super(0,0,0);
        this.ville = "";
        this.what3words = "";
    }

    // Méthodes
    public String getWhat3words() {
        return this.what3words;
    }
    public void setWhat3words(String what3words) {
        String regex = "^\\p{L}{2,}\\.\\p{L}{2,}\\.\\p{L}{2,}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(what3words);
        if (matcher.matches()) {
            this.what3words = what3words;
        }
    }

    public void setTousLesAttributs() {
        this.x = 0;
        this.y = 0;
        this.z = 0;
        this.what3words = "";
        this.ville = "";
    }
}
```

Avant, de pouvoir tester votre programme, vous devez récupérer une clé (qui remplacera les caractères surlignés en jaune) pour pouvoir utiliser l'API et ajouter la méthode suivante dans PointGPS :

PointGPS.java

```
import com.what3words.javawrapper.What3WordsV3;
import com.what3words.javawrapper.What3WordsV3Interceptor;
import com.what3words.javawrapper.What3WordsJavaWrapper;
import com.what3words.javawrapper.request.Coordinates;
import com.what3words.javawrapper.response.ConvertTo3WA;
...
private String getWhat3wordsFromGPS(double x, double y) {
        What3WordsV3 api = new What3WordsV3("########");
        ConvertTo3WA words = api.convertTo3wa(new Coordinates(x, y))
                .language("en")
                .execute();
        System.out.println("-----Words: " + words);
        return words.toString();
    }
```

Une dernière modification est d'utiliser la fonction ci-dessus dans le constructeur avec les 3 paramètres (x, y, et ville) :

```
this.what3words = getWhat3wordsFromGPS(x, y);
```

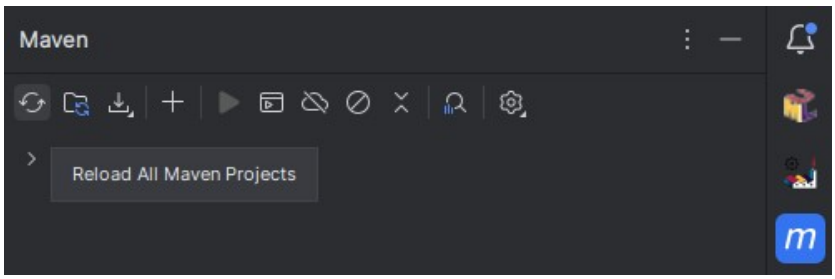
D.2.3 Modification pom.xml

Voici les modifications à apporter à ce fichier de commande pour maven, c'est lui qui décrit comment maven devra compiler notre projet pour qu'il fonctionne correctement.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>net.rouma</groupId>
    <artifactId>PositionProjet2</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>22</maven.compiler.source>
        <maven.compiler.target>22</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.what3words</groupId>
            <artifactId>w3w-java-wrapper</artifactId>
            <version>3.1.19</version>
        </dependency>
    </dependencies>

</project>
```

Vous pouvez lancer votre programme. En cas de problème, cliquez sur le bouton m pour recharger votre projet maven (rafraîchir).



Si tout se passe bien (le build peut prendre beaucoup de temps la première fois), vous obtiendrez alors cet affichage :

```
Ville : Paris
what3words :
-----Words: {
  ...
  "words": "trimmer.driveway.pity"
```

D.2.4 Extraction de données JSON

L'exécution montre donc un retour en JSON (que nous avons dû convertir en chaîne de caractères) : vous pouvez coller le résultat du JSON dans https://jsonformatter.org/json-viewer pour vérifier la structure.

L'objectif pour terminer ce programme est de récupérer directement la valeur correspondante à `words`.

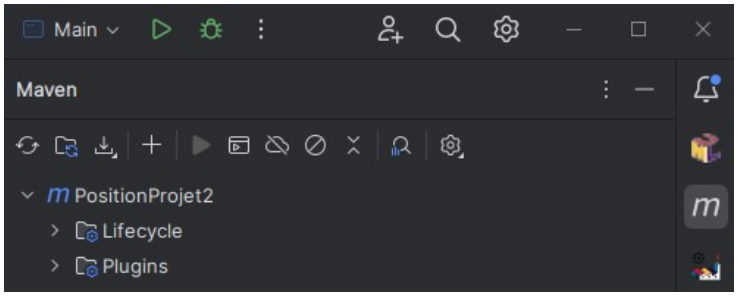
D.2.4.a Ajout de la dépendance org.json

Pour cela, nous allons ajouter une dépendance JSON, car Java n'est pas capable de le lire directement.

Sur le catalogue de dépôt, on trouve https://mvnrepository.com/artifact/org.json/json et en cliquant sur 20240303 (la version sans vulnérabilité au 11 avril 2024), on obtient le code à intégrer dans `pom.xml`.

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20240303</version>
</dependency>
```

Il faut ensuite rafraîchir maven pour qu'il télécharge automatiquement la dépendance :



Normalement, `org.json` passe de rouge à gris clair dans le fichier `pom.xml`.

D.2.4.b Importation et usage dans la classe PointGPS

Il faut ensuite importer les éléments de la dépendance :

```
import org.json.JSONObject;
```

Puis l'utiliser sur l'objet `words` dans la méthode `getWhat3wordsFromGPS(double x, double y)` :

```
    private String getWhat3wordsFromGPS(double x, double y) {
        What3WordsV3 api = new What3WordsV3("AI7AAXW8");
        ConvertTo3WA words = api.convertTo3wa(new Coordinates(x, y))
                .language("en")
                .execute();
        JSONObject json = new JSONObject(words);
        return json.getString("words");
    }
```

L'exécution est maintenant conforme à nos attentes.

E Résultat et améliorations

Le résultat du programme est le suivant :

```
Ville : Paris
what3words :
Grenoble : trimmer.driveway.pity

Process finished with exit code 0
```

Il nous reste une amélioration qui utilisera une méthode privée : la classe Main ou une classe héritant de PointGPS ne pourra pas l'utiliser.

E.1 Amélioration de l'affichage

E.1.1 Demande d'amélioration d'affichage

Tout d'abord, il serait pratique d'avoir une **méthode** qui affiche les attributs d'un objet PointGPS de manière identique, dans la classe Main. **Essayez de générer un tel affichage, ressemblant à ceci :**

```
nomVille GPS        : ######, #####
nomVille what3words : ******.******.*******
```

E.1.2 Proposition d'amélioration d'affichage

Voici une solution possible :

Main.java

```
    public static void afficherPoint(PointGPS point) {
        System.out.println(point.ville+" GPS        : "+ point.getX()+", "+ point.getY());
        System.out.println(point.ville+" what3words : "+point.getWhat3words());
    }

    public static void main(String[] args) {
        // Création d'un point et modification de l'attribut 'ville'
        PointGPS paris = new PointGPS();
        paris.ville = "Paris";
        afficherPoint(paris);

        PointGPS grenoble = new PointGPS(45.188529, 5.724524, "Grenoble");
        afficherPoint(grenoble);
    }
```

C'était plutôt facile...

E.2 Mise à jour what3words

Si un objet est instancié avec le constructeur vide, nous voulons que X et Y n'aient pas de valeur. Par exemple, nous voyons que Paris n'a pas utilisé le constructeur avec des coordonnées et il est donc impossible de générer une coordonnée what3words.

E.2.1 Demande d'amélioration

Nous pourrions cependant lancer une vérification à chaque utilisation d'un accesseur et lancer la fonction getWhat3wordsFromGPS(double x, double y) si x et y sont cohérents. Il faut donc que le constructeur par défaut (celui qui n'a pas de paramètres) n'initialise pas les attributs x et y.

Proposez un code pour

- **modifier le constructeur PointGPS()**
- **modifier les accesseurs X et Y pour vérifier que X et Y sont bien définis.**

La réponse se trouve à la page suivante, mais vous pouvez sûrement y arriver seul… avec un indice :

En **Java**, pour affecter une valeur **null** à un attribut, il faut préciser son type suivi de .NaN comme dans l'exemple ci-après : `Double.NaN`. C'est équivalent à **null** en **Javascript**.

E.2.2 Proposition d'amélioration

Concernant le constructeur, c'est très rapide : il suffit de supprimer les affectations de valeur en utilisant l'astuce proposée sur la méthode `super()` de la classe PointGPS :

```
super(Double.NaN,Double.NaN,Double.NaN);
```

La deuxième partie implique de corriger la méthode privée `getWhat3wordsFromGPS(double x, double y)` en vérifiant si X et Y sont définis, et dans ce cas, lance la recherche d'un what3words :

```
    private String getWhat3wordsFromGPS(double x, double y) {
        if (!Double.isNaN(this.x) && !Double.isNaN(this.y)) {
            What3WordsV3 api = new What3WordsV3("AI7AAXW8");
            ConvertTo3WA words = api.convertTo3wa(new Coordinates(x, y))
                    .language("en")
                    .execute();
            JSONObject json = new JSONObject(words);
            return json.getString("words");
        }
        return "";
    }
```

C'est ici que nous allons devoir redéfinir les méthodes setX() et setY() pour ajouter la fonction `getWhat3wordsFromGPS(double x, double y)` dedans. Pour cela, il faut utiliser `@Override` au-dessus des méthodes redéfinies.

```
    @Override
    public void setX(double x) {
        if (x >= -180 && x <= 180) {
            this.x = x;
        }
        this.what3words = getWhat3wordsFromGPS(this.x, this.y);
    }

    @Override
    public void setY(double y) {
        if (y >= -90 && y <= 90) {
            this.y = y;
        }
        this.what3words = getWhat3wordsFromGPS(this.x, this.y);
    }
```

L'usage de `@Override` indique à Java de privilégier l'utilisation de nos méthodes locales à celles dans la classe héritée (GPS). Nous pouvons ainsi utiliser la méthode privée de notre classe PointGPS qui va vérifier si les attributs existent.

Essayez de tester toutes les combinaisons sur l'objet paris dans la classe Main() :

- Affichage après instanciation
- Affichage après setX(48.866669)
- Affichage après setY(2.33333)

Voici la modification de la méthode main() pour tester que nos modifications fonctionnent.

Main.java

```
    public static void main(String[] args) {
        // Création d'un point et modification de l'attribut 'ville'
        PointGPS paris = new PointGPS();
        paris.ville = "Paris";
        afficherPoint(paris);
        paris.setX(48.866669);
        afficherPoint(paris);
        paris.setY(2.33333);
        afficherPoint(paris);

        PointGPS grenoble = new PointGPS(45.188529, 5.724524, "Grenoble");
        afficherPoint(grenoble);
    }
```

Si vous avez bien suivi l'activité, votre affichage devrait correspondre à celui-ci :

```
Paris GPS         : NaN, NaN
Paris what3words :
Paris GPS         : 48.866669, NaN
Paris what3words :
X et Y existent et ne sont pas NaN
Paris GPS         : 48.866669, 2.33333
Paris what3words :
X et Y existent et ne sont pas NaN
Grenoble GPS         : 45.188529, 5.724524
Grenoble what3words : trimmer.driveway.pity

Process finished with exit code 0
```

Cette activité est terminée, vous avez appris à…

- Utiliser une classe
- Créer et utiliser une classe fille
- Utiliser une dépendance extérieure (what3word et org.json)
- Instancier une classe fille en utilisant le constructeur de la classe mère (super())
- Surcharger une méthode de la classe mère (getX() et getY())

F Annexes

F.1 Sources

Comparaison IntelliJ Ultimate et Community :
https://www.jetbrains.com/idea/features/editions_comparison_matrix.html

F.2 Autres