TP ODL - programmation Java: notions de base, orienté objet et héritage

Notions théoriques

Classes

Le langage Java se base sur la notion d'objet. Chaque objet est lui-même une instance d'une classe. Une classe définit le comportement de ses objets ainsi que ses données.

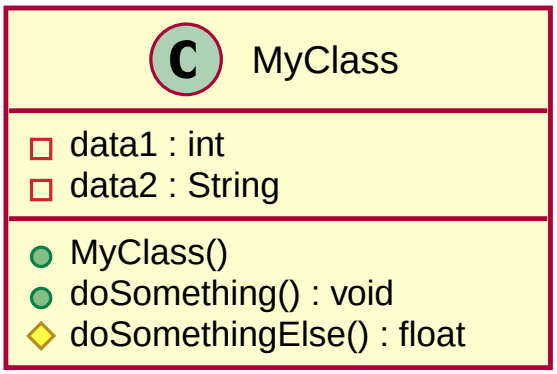

Figure 1. Une classes avec ses champs et ses méthodes

Voici comment la classe décrite ci-dessus en UML se traduit en Java.

src/main/java/examples/MyClass.java

```
class MyClass {
    private int data1;
    private String data2;

    public MyClass() { /* ... */ }

    public void doSomething() { /* ... */ }

    protected float doSomethingElse() {
        /* ... */
        return 1.0F;
    }
}
```

Les mots clés tels que `public`, `protected` et `private` sont utilisés pour définir l'accès aux éléments d'une classe.

private

signifie que l'élément n'est accessible que depuis la classe elle-même.

public

signifie que l'élément est accessible à toutes les autres classes.

protected

donne accès à cet élément depuis les classes qui héritent de celles-ci (cfr. Héritage) ou qui se trouvent dans le même package (cfr. Packages).

Il existe un quatrième niveau d'accès. Il s'agit du niveau "package private" qui n'est identifié par aucun mot-clé car il est considéré comme le niveau d'accès par défaut. Ce niveau permet l'accès des éléments qu'il marque depuis les classes du même package (cfr. Packages).

Une classe publique s'écrit dans un fichier portant le nom de la classe suivi de l'extension `.java`. Il ne peut y avoir qu'une seule classe publique par fichier.

En plus des champs dont la valeur est liée à chaque instance (objet) de la classe, il est possible de définir des éléments partagés par tous les objets de la classe. Ces champs sont identifiés par le mot-clé `static` et ne sont associés à aucun objet.

Packages

Puisqu'une classe publique doit se trouver dans un fichier du même nom, le risque de conflit, l'utilisation du même nom pour désigner deux concepts différents, est important. Afin de prévenir ce problème, Java permet d'isoler un ensemble de classes dans un package. Un package représente donc à la fois un espace de nommage pour les classes, identifié par le mot-clé `package` sur la première ligne du fichier, et un répertoire de fichiers.

L'usage est de regrouper dans un même package des classes qui se rapportent à un même concept métier ou à une même fonction dans l'application.

Une fois les classes isolées, il faut fournir un mécanisme pour leur permettrent de dialoguer. Le mot-clé `import` permet ainsi de rendre accessible une ou plusieurs classes d'un autre package. Les différents imports se placent directement après la déclaration du package.

src/main/java/examples/aPackage/Header.java

```
package examples.aPackage; ①

import examples.anotherPackage.AClassInAnotherPackage; ②
import examples.yetAnotherPackage.*; ③

import static examples.aPackage.Utils.sayHello; ④
```

① Déclare le package

② Importe une unique classe

③ Importe tout un package

④ Importe un élément d'une classe extérieure comme si elle faisait partie de la classe courante

Constructeurs et accesseurs

Pour créer un objet, on utilise une méthode particulière appelée constructeur. Cette méthod n'a aucun type de retour et ne peut être appelée directement : elle doit toujours être précédée du mot-clé `new`. L'usage de cette méthode permet d'initialiser la nouvelle instance d'une classe. Elle peut prendre des paramètres.

S'il n'existe *aucun* constructeur, le compilateur en crée un par défaut. Par extension, on trouve dans la littérature des références au *constructeur par défaut*. Celles-ci se rapporte soit au constructeur écrit par le compilateur, soit à un constructeur sans paramètre écrit par le développeur.

src/main/java/examples/ClassWithConstructor.java

```
    public ClassWithConstructor() { ①
        field1 = 123;
        field2 = "default value";
    }

    protected ClassWithConstructor(int field1, String field2) { ②
        this.field1 = field1;
        this.field2 = field2;
    }
```

① Constructeur *par défaut*

② Constructeur avec paramètres

Une pratique très courante en Java est de ne jamais exposer directement l'état interne d'une classe. Pour ce faire, l'auteur de la classe marque tous les champs comme private. L'auteur crée également des accesseurs en lecture (`getField()`) ou en écriture (`setField(…)`). Cette pratique porte le nom de *convention JavaBean*. Enormément de librairies compte sur cette convention pour leur fonctionnement. Je vous encourage donc à l'adopter, tout en faisant preuve d'esprit critique : rien ne sert d'écrire des accesseurs à la fois en lecture et en écriture pour tous les champs.

src/main/java/examples/ClassWithAccessors.java[]

```
package examples;

public class ClassWithAccessors {
    private int field1;
    private String field2;

    public ClassWithAccessors(int field1) { this.field1 = field1; }

    public int getField1() { return field1; } ①

    public String getField2() { return field2; }

    protected void setField2(String field2) { this.field2 = field2; }
}
```

① Dans cette classe, le champs `field1` ne peut pas être modifié après la création de l'objet.

Héritage

L'héritage est un mécanisme qui permet de créer une classe *enfant* qui augmente ou de redéfinit le comportement de sa classe *parent*.

src/main/java/examples/Child.java

```
package examples;

public class Child extends ClassWithAccessors {
    private String field3;

    public Child(int field1, String field3) {
        super(field1); ①
        this.field3 = field3;
    }

    @Override
    public int getField1() { return super.getField1() * 3; }; ②

    @Override
    public void setField2(String field2) { super.setField2(field2); } ③

    public String getField3() { return field3; } ④
}
```

① Pour appeler le constructeur de la classe parent, on utilise cette construction qui ne fonctionne que sur la première ligne du nouveau constructeur.

② On redéfinit (*override*) le comportement de la méthode de la classe parent. Le champ `field2` étant privé, on utilise l'accesseur du parent.

③ Il est parfois utile de rendre accessible aux autres classes une méthodes du parent.

④ Cette méthode est accessible aux utilisateurs de la classe `Child` mais pas à ceux de son parent.

Interaction avec l'utilisateur

Tout programme a besoin d'un point de départ. En Java, c'est la méthode `public static void main(String[] args)`. Le paramètre de cette méthode est un tableau de chaînes de caractères correspondant aux arguments de l'appel en ligne de commande.

```
java my.package.MyClass arg1 arg2 arg3 ...
```

Application en ligne de commande

Pour gérer l'interface en ligne de commande, vous disposez des flux d'entrée `System.in` et de sortie `System.out`. Ils vous permettent d'écrire un message à l'utilisateur ou de lire ce qu'il tape au clavier. La classe Scanner permet de décoder facilement les entrées de l'utilisateur.

src/main/java/examples/CommandLine.java

```
    public static void main(String[] args) {
        System.out.println("Hello!");
        System.out.print("What's your name? ");
        Scanner scanner = new Scanner(System.in);
        String name = scanner.next();
        System.out.format("Hello %s!\nHave a nice day!\n", name);

    }
```

Interface graphique (GUI)

Il existe un moyen simple d'afficher des boîtes de dialogue pour interagir avec l'utilisateur.

src/main/java/examples/GuiUsingModals.java

```
    public static void main(String[] args) {
        String name = JOptionPane.showInputDialog("What's your name?");
        JOptionPane.showMessageDialog(null, "Hello " + name + "!");
    }
```

Nous verrons les interfaces plus complexes lors d'un prochain labo.

Exercices

Les instructions des exercices se trouvent autant que possible en commentaire dans les fichiers `Exercise<X>.java`.

Exercice 1

src/main/java/exercise1/Exercise1.java

```
package exercise1;

public class Exercise1 {
    static Person createPerson(String name, int age) {
        // Ajoutez les champs name et age à la classe Person.
        // Créez un constructeur public permettant d'initialiser ces valeurs au moment
de la construction.
        // Créez des getters publics pour lire ces valeurs une fois la classe
construite.
        return null;
    }

    public static void main(String[] args) {
        // Considérant que cette classe est démarrée en ligne de commande avec un
premier paramètre donnant le nom
        // et un second donnant l'âge (nombre entier), créez un objet Person sur base
de ceux-ci.
        // Ecrivez ensuite le nom et l'âge de cette personne sur la sortie standard.
    }
}
```

Exercice 2

src/main/java/exercise2/Exercise2.java

```
package exercise2;

public class Exercise2 {

    // Ajoutez un champ age à chaque instance de la classe Person.
    // Créez-y une méthode statique "computePopulationSize" pour retourner la taille
de la population.
    // Créez-y une méthode statique "computeAveragePopulationAge" pour calculer l'âge
moyen de la population.
    // Implémentez-y une méthode statique "resetPopulation" pour remettre les
compteurs à zéro.

    static void createPerson(int age) {
    }

    static int computePopulationSize() {
        // TODO remove comment when implemented
        // return Person.computePopulationSize();
        return 0;
    }

    static float computeAveragePopulationAge() {
        // TODO remove comment when implemented
        // return Person.computeAveragePopulationAge();
        return 0.0F;
    }

    static void resetPopulation() {
        // TODO remove comment when implemented
        // Person.resetPopulation();
    }
}
```

Exercice 3

L'exercice vous demande de créer une famille de classes représentant quelques formes géométriques.

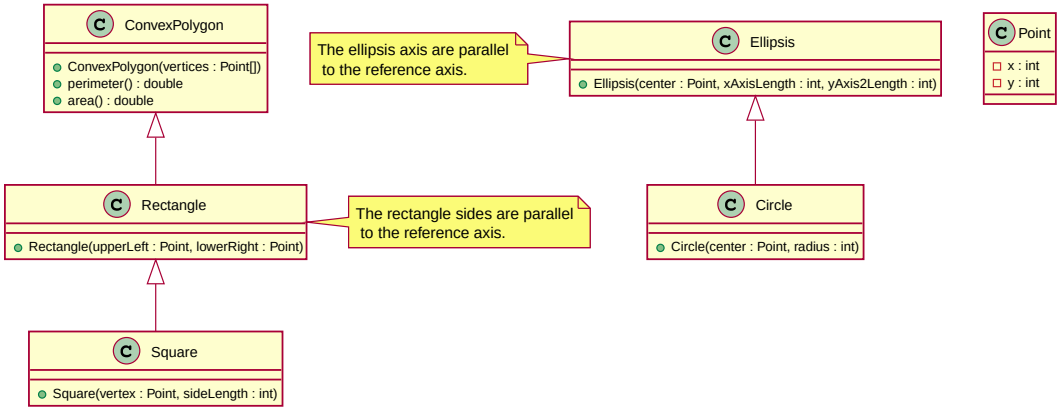

Figure 2. Formes géométriques

Les classes `ConvexPolygon` et `Ellipsis` définissent deux méthodes permettant de calculer le périmètre et l'aire. Ces méthodes, bien que définies uniquement dans les classes parentes, sont héritées par les classes enfants.

Périmètre de l'ellipse

$$2\pi\sqrt{\frac{1}{2}(a^2+b^2)}$$

Aire de l'ellipse

$$\pi \cdot a \cdot b$$

Aire du triangle

$$\left|\frac{(x_1-x_3)(y_2-y_3)-(y_1-y_3)(x_2-x_3)}{2}\right|$$

src/main/java/exercise3/Exercise3.java

```
package exercise3;

public class Exercise3 {
    static ConvexPolygon buildConvexPolygon(Point[] vertices) {
        // TODO build a ConvexPolygon
        return null;
    }

    static ConvexPolygon buildRectangle(Point upperLeft, Point lowerRight) {
        // TODO build a Rectangle
        return null;
    }

    static ConvexPolygon buildSquare(Point upperLeft, int sideLength) {
        // TODO build a square
        return null;
    }

    static Ellipsis buildEllipsis(Point center, int xAxisLength, int yAxisLength) {
        // TODO build an ellipsis
        return null;
    }

    static Ellipsis buildCircle(Point center, int radius) {
        // TODO build a circle
        return null;
    }

    static double computePerimeter(ConvexPolygon convexPolygon) {
        // TODO ask the convexPolygon to compute its perimeter
        return 0.0;
    }

    static double computeArea(ConvexPolygon convexPolygon) {
        // TODO ask the convexPolygon to compute its area
        return 0.0;
    }

    static double computePerimeter(Ellipsis ellipsis) {
        // TODO ask the ellipsis to compute its perimeter
        return 0.0;
    }

    static double computeArea(Ellipsis ellipsis) {
        // TODO ask the ellipsis to compute its area
        return 0.0;
    }
}
```

Exercice 4

Sur base des classes que vous avez créées dans l'exercice 3, implémentez une interface Swing qui permet de définir une forme géométrique par étapes puis d'afficher le périmètre et la surface de cette forme. Chaque question fera l'objet d'une boîte de dialogue et la réponse final celui d'une boîte d'information.

Références

- **Java 8 API documentation**
- **The Java Tutorials**
- Manneback, P., Notes de cours.
- Manneback, P. et Flémal, S. (2014), Notes de TP.
- Eckel, B. (2006). Thinkin in Java (4th ed.). Prentice Hall. Retrieved from http://mindview.net/Books/TIJ4
- Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship (1st ed.).
- Martin, R. C. et al. Clean Coders. https://cleancoders.com/
- Eclipse IDE
- IntelliJ IDE
- Netbeans IDE