Chapitre 3

Présentation des concepts de base de la POO appliquée à Java

1. Les éléments de base
2. Le point d’entrée d’une application Java
3. Les mots clés, règles et conventions Java

1 Les éléments de base

2 Le point d’entrée d’une application Java

3 Les mots clés, règles et conventions Java

Les éléments de base de la POO

Les éléments de base de la POO appliqués à Java sont :

- Les classes
- Les attributs
- Les méthodes
- Les instances/objets
- Les membres statiques et invariants
- Les packages
- Les objets et leurs interaction
- Les modificateurs de visibilité
- Les types de données
- L'encapsulation

- (Les concepts complexes tels que l'*héritage* et le *polymorphisme* seront abordés dans des chapitres séparés)

1 Notion de classe (1/2)

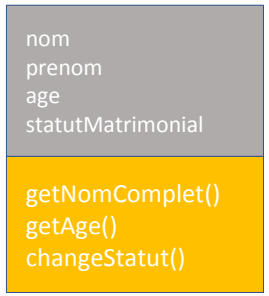
La **classe** est la principale notion de la POO dans et autour de laquelle gravite les autres éléments suscités.
Elle définit d'un point de vue conceptuel, une famille d'***objets*** (*cf. chapitre 2, slides 8 et 9*) ayant une même structure.
Une classe modélise/décrit sous un angle de vue une ***entité*** dans un programme Java.

Exemple : si on vous demande de décrire de façon macroscopique ce qu'est une **Personne**, que diriez vous ?

Réponse :

- ➔ c'est une entité qui possède un ***nom***, un ***prénom***, un ***âge***, un ***statut matrimonial***, par exemple.
- ➔ Sur cette entité en tant que boîte noire, on peut s'interroger à tout moment sur : ***quel est son nom complet ?***, ***quel est son âge ?***, ***quel est son statut matrimonial ?***, ***comment changer son statut matrimonial ?***, etc

Exemple de représentation schématique de la **classe Personne**



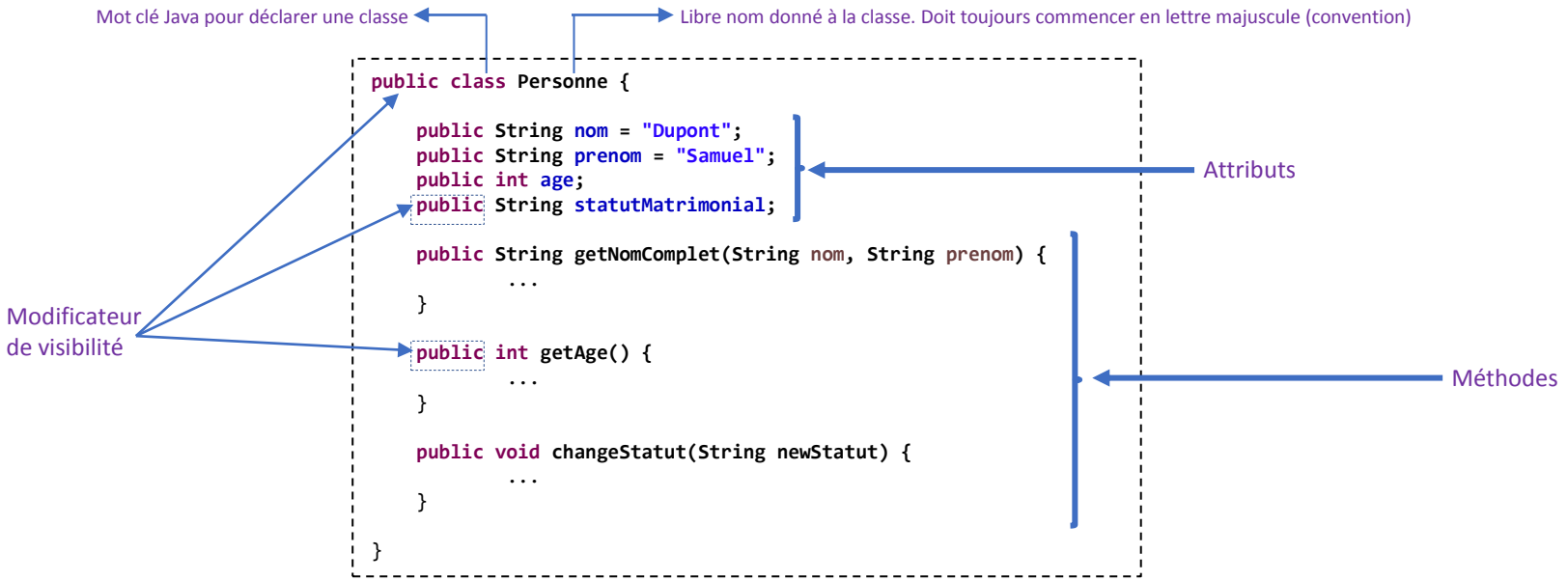
1 Notion de classe (2/2)

Classe **Personne**

nom
prenom
age
statutMatrimonial

getNomComplet()
getAge()
changeStatut()

Exemple de représentation de la classe Personne en langage Java



2 Les attributs

Un **attribut** est une **propriété** qui contribue à la caractérisation d'une classe.
Exemple : *nom*, *prénom*, *âge*, … sont les attributs de la classe Personne

Techniquement, c'est une **variable** qui porte une donnée dans une classe.
Une donnée pouvant être une *chaine de caractères*, une *chaine alphanumérique*, un *entier*, une *date*, un *nombre réel*, etc

```
public class Personne {
  public String nom = "Dupont";
  public String prenom = "Samuel";
  public int age;
  public String statutMatrimonial;

    //méthodes…
}
```

Format de déclaration d'une **variable** en langage Java :

1/ **Déclarer une variable en lui affectant directement une valeur :**

<ModificateurVisibilité> <TypeDeDonnée> **<NomDeLaVariable> = <ValeurDeLaVariable> ;**

Ex : `public String prenom = "Samuel";`

2/ **Déclarer une variable dont la valeur sera affectée plus tard :**

<ModificateurVisibilité> <TypeDeDonnée> **<NomDeLaVariable> ;**

Ex : `public int age;`

Le symbole **=** est utilisé pour affecter une valeur à une variable

3 Les méthodes (1/2)

Une **méthode** est une fonction ou une procédure qui utilise les **propriétés/attributs** de la classe pour réaliser un traitement particulier. Il peut aussi modifier la valeur de ces attributs. C'est pourquoi, elle est considérée comme un élément comportemental de la classe. Une méthode dans une classe se détermine par le questionnement logique des opérations qu'on voudrait qu'elle fasse sur les propriétés de cette classe en cohérence avec les besoins du problème traité. Ce n'est donc pas un questionnement exhaustif, mais ciblé.

Exemple : *obtenir le nom complet à partir du nom et du prénom, changer le statut matrimonial, obtenir l'âge,* peuvent être les seuls questionnements pertinents pour un problème traité. Ils permettent donc d'en déduire les méthodes *getNomComplet(), changeStatut()* et *getAge()*

```
public class Personne {

  //attributs…

  public String getNomComplet (String nom, String prenom) {
          ...
  }

  public int getAge () {
          ...
  }

  public void changeStatut (String newStatut) {
          ...
  }
}
```

Signature d'une méthode :

<ModificateurVisibilité> <TypeDeDonnéesRetour> **<NomMéthode>** (<ListeDesParamètres>) ;

Ex : `public String getNomComplet (String nom, String prenom) ;`

3 Les méthodes (2/2)

Structure d'une méthode qui retourne un résultat

```
public String getNomComplet (String nom, String prenom) {
    //éventuelle instruction 1;
    //éventuelle instruction 2;
    //…
    String nomComplet = prenom+nom;
    return nomComplet;
}
```

Une telle méthode doit toujours avoir l'instruction ***return x*** comme dernière instruction. ***x*** étant la donnée retournée.

Structure d'une méthode qui ne retourne pas de résultat

```
public void changeStatut (String newStatut) {
    //éventuelle instruction 1;
    //éventuelle instruction 2;
    //éventuelle instruction 3;
    //…
    statutMatrimonial = newStatut;
    [return;]
}
```

Une telle méthode doit toujours avoir le mot clé **void** comme type de retour. On peut éventuellement mettre l'instruction ***return;*** comme dernière instruction, mais ce n'est pas obligatoire

Définition : le **corps d'une méthode** correspond au bloc de code (la suite d'instructions) située entre ses accolades ouvrante ({) et fermante (})

4 Notion d'instanciation

```java
public class Personne {

    public String nom = "Dupont";
    public String prenom = "Samuel";
    public int age;
    public String statutMatrimonial;

    public String getNomComplet (String nom, String prenom) {
      String nomComplet = prenom+nom;
      return nomComplet;
    }

    public int getAge () {
      return age;
    }

    public void changeStatut (String newStatut) {
      statutMatrimonial = newStatut;
    }
}
```

- Pour user des fonctionnalités que propose une classe, il faut en général la transformer en **objet**
- L'action de création d'un **objet** à partir d'une classe s'appelle **instanciation**.
- A partir d'une classe, on peut instancier plusieurs objets
- Une **instance** est une occurrence d'une classe en tant qu'objet
- Les instances d'une classe correspondent à la famille d'objets de même structure nés de cette classe
- Chaque instance/objet est autonome et indépendant d'un autre
- Pour instancier une classe en Java, on utilise le mot clé ***new***

Exemple d'instanciation de la classe Personne:

Personne obj1 = *new* Personne();
Personne obj2 = *new* Personne();

Remarques:

- *obj1* et *obj2* sont 2 objets et donc 2 instances de la classe
- D'un point de vue physique, *obj1* et *obj2* sont deux éléments complètement indépendants

- L'accès aux méthodes et attributs se font au moyen de la **notation pointée** sur un objet :

Ex : *obj1.getNomComplet(''Durant'', ''Patrick'')*
obj2.getAge()
obj2.prenom

5 Les membres invariants

On qualifie d'invariant, tout élément qui une fois créé dans un programme ne pourra plus changer (devient non modifiable). Cela concerne les **attributs**, les **méthodes** et les **paramètres de méthodes** d'une classe; et la **classe** elle-même. Java utilise le mot clé **final** pour identifier un tel élément

(➔ Au fur et à mesure dans ce cours, nous aborderons et remarqueront les intérêts d'un tel concept)

La variable **nom** est définie comme un invariant. On dit en d'autres termes que c'est une **Constante**. Toute constante doit être initialisée dès sa déclaration car sa valeur ne changera plus dans le programme

```
public final String nom = "Dupont";
```

La méthode *getAge*() est définie comme invariante. Nous aborderons son intérêt dans le chapitre sur l'héritage. On qualifie une telle méthode de ***méthode finale***

```
public final int getAge () {
   return age;
}
```

```
public String getNomComplet (final String nom, String prenom) {
    String nomComplet = prenom+nom;
    return nomComplet;
}
```

Le paramètre **nom** que la méthode getNomComplet() est marqué **final**. Cela implique que dans le corps de la méthode, sa valeur ne pourra pas être altérée ni modifiée. On qualifie de tels paramètres de ***paramètre final***

```
public final class Personne {
    //attributs
    //méthodes
}
```

La classe Personne est définie comme invariante. Nous aborderons son intérêt dans le chapitre sur l'héritage. On qualifie une telle classe de ***classe finale***

6 Les membres statiques

On qualifie de statique, tout élément définit dans une classe et qui sera commun pour toutes les instances de cette classe. Il peut aussi être accéder sans instancier la classe. Dans le cadre de ce cours, nous limitons ce concept sur les **attributs** et les **méthodes**.
Java utilise le mot clé **static** pour identifier un tel élément

(➔ Au fur et à mesure dans ce cours, nous aborderons et remarqueront les intérêts d'un tel concept)

```
public static String nom = "Dupont";
```

Supposons la variable **nom** définie comme statique dans la classe Personne. Toutes les instances qui y sont issus possèdent une référence unique vers cette variable.

- Exemple :

  Personne obj1 = *new* Personne();
  Personne obj2 = *new* Personne();

  obj1.nom = obj2.nom = **Dupont**

- La modification de cette variable par une instance quelconque se répercutera automatiquement sur les autres instances

  Ex : si obj1 modifie la variable *nom* pour lui affecter la valeur «Durant», alors *obj2* aura automatiquement la valeur «Durant»

  obj1.nom = obj2.nom = **Durant**

- L'accès à une variable statique peut se faire sans instancier la classe on peut directement faire Personne.nom dans le programme

```
public static int getAge () {
   return age;
}
```

Une méthode statique peut être utilisée sans besoin d'instancier un objet de la classe à laquelle elle appartient. Les méthodes ainsi définies peuvent être appelées avec la notation <Maclasse>.<methodeStatique()> au lieu de <objet>.<methodeStatique()>

Exemple, la méthode *getAge*() étant définie comme statique dans la classe Personne, pas besoin de l'instancier pour y avoir accès, on peut directement l'appeler comme suit : Personne.getAge()

7 Les variables de classe et les variables d'instances

- Une *variable d'instance* est un attribut d'une classe dont l'accès n'est possible que via une instance (un objet) de cette dernière. Ce type d'attribut se distingue par le fait qu'il **n'est pas marqué** par le mot clé **static**.

  Exemple : soit la classe Personne ci-contre. Pour accéder aux attributs *nom*, *prenom* et *age*, il faut créer une instance de cette classe : Personne obj = *new* Personne()
  Ensuite on y accède en faisant : obj.nom, obj.prenom, obj.age
  ➔ Conclusion : *nom*, *prenom* et *age* sont des variables d'instances de la classe Personne

- Une *variable de classe* est à l'inverse de la variable d'instance, un attribut statique. Il est donc marqué par le mot clé **static**. On peut dont y accéder de deux manières : avec une instance de la classe (un objet), mais aussi sans besoin d'instancier la classe

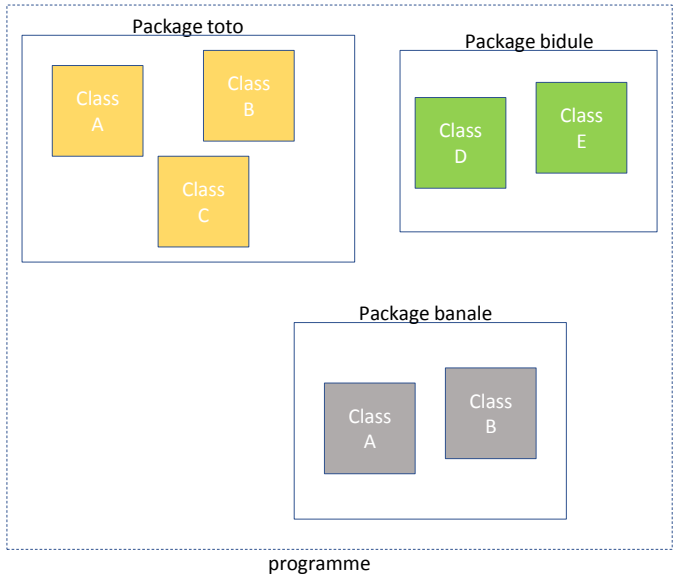
  Exemple : soit la classe Personne ci-en face. L'attribut *statutMatrimonial* est marqué par le mot clé **static**. L'accès se fait donc comme ceci : Personne.statutMatrimonial ou obj.statutMatrimonial
  ➔ Conclusion : *statutMatrimonial* est une *variable de classe* de la classe Personne

```
public class Personne {

    public String nom = "Dupont";
    public String prenom = "Samuel";
    public int age;
    public static String statutMatrimonial;

    public String getNomComplet (String nom, String prenom) {
            ...
    }

    public int getAge () {
            ...
    }

    public void changeStatut () {
            ...
    }

}
```

Note : ce principe est exactement le même pour les méthodes définies dans une classe

8 La notion de package

Dans un programme Java, on sera amené à créer plusieurs classes. La question à laquelle répond la notion de package est liée à l'aspect organisationnel de notre code source. Nous serons amenés à regrouper des classes dans des ensembles différents. Un ensemble ainsi créé s'appelle **package**. Pour permettre à une classe d'indentifier à quel package il appartient, il faut ajouter à l'entête du fichier dans lequel il est défini, la syntaxe *package <nomDuPackage>;*




```
package bidule;

public class D {

    //attributs
    //méthodes
}
```

```
package toto;

public class B {

    //attributs
    //méthodes
}
```

L'avantage majeure de ce concept est qu'elle permet de créer plusieurs classes de même nom dans un programme à la condition qu'elle ne soit pas déclarée dans le même package

9 L'interaction entre objets

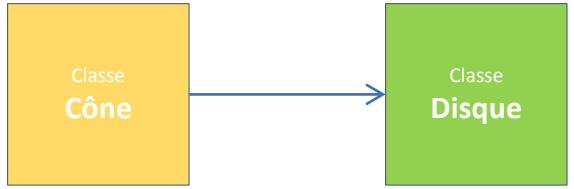
Dans un programme, les objets sont amenés à coopérer ensemble pour réaliser le besoin attendu.

Exemple :

Supposons les classes **Cone** et **Disque** dans un programme.

- La classe **Disque** est chargée de calculer le périmètre et l'aire d'un disque si on lui en fourni le rayon.

- La classe **Cone** est chargée de calculer le volume d'un cône. Sachant qu'un cône a une base en disque, si on veut calculer son volume, on aura par conséquent besoin de calculer l'aire de sa base. Or comme Disque fourni déjà ce service, **comment faire pour l'utiliser dans la classe Cone ?**




```
1  public class Cone {
2
3      public double rayon;
4      public double hauteur;
5
6      public double calculerVolume() {
7
8          Disque objDisk = new Disque();
9
10         double aire = objDisk.calculerAire(rayon);
11         double volume = (aire*hauteur)/3;
12         return volume;
13     }
14 }
```

```
public class Disque {

    public final static double PI = 3.14;

    public double calculerAire (double rayon) {
        return PI*rayon*rayon;
    }

    public double calculerPerimetre (double rayon) {
        return 2*PI*rayon;
    }
}
```

- Pour que deux classes interagissent ensemble, l'une doit pouvoir créer une instance de l'autre.
- Dans l'exemple, cela s'identifie à la ligne 8 par la création de l'objet **objDisk** dans la méthode **calculerVolume()**
- La création d'un objet dépendant peut se faire, soit dans une méthode (cf. ligne 8, classe Cone), soit au niveau de la déclaration des attributs de la classe. Ex : on aurait pu mettre l'instruction de la ligne 8 à la ligne 5
- Pour récupérer le message (le résultat) renvoyé par un objet au travers de l'invocation de sa méthode, on fait comme à la ligne 10. Ainsi, ***aire*** est le message retourné par l'appelle de la méthode ***calculerAire***() de l'objet **objDisk**.

10 Les modificateurs de visibilité

La visibilité désigne le niveau d'accessibilité d'un **attribut** ou d'une **méthode** hiérarchiquement dans (*cf. schéma de la slide 14*) :

1- la classe dans laquelle il/elle est défini(e)
2- les sous-classes de ladite classe (nous aborderons ce point dans le chapitre sur l'héritage)
3- le package auquel appartient ladite classe
4- le monde extérieur au package d'appartenance (les autres packages du programme)

Une **classe** elle-même a un niveau de visibilité et cela se classifie à partir du point 2 ci-dessus au point 4.

Java utilise la notion de **modificateur** identifiée par les mots clés **private**, **protected**, **public** pour identifier, selon la classification ci-dessus édictée, la visibilité d'un élément (attribut/méthode/classe) dans un programme. Lorsqu'un *attribut*, *méthode* ou *classe* est déclaré sans modificateur, Java lui affecte un modificateur par défaut qualifié de « **default package** ».

Question pour comprendre le tableau : si un élément est défini avec le modificateur private/protected/public/(default package), où sera-t-il disponible/accessible ?

Niveau de visibilité ↓

	private	protected	public	default package
classe	✓	✓	✓	✓
Sous-classe	✗	✓	✓	✗
dans le package	✗	✓	✓	✓
Monde extérieur	✗	✗	✓	✗

11 Les types de données

Java est un langage **fortement typé**. C'est-à-dire que lors de la déclaration d'une variable dans un tel langage, on doit explicitement indiquer à quel ensemble de données il appartient. Cette ensemble représente ce qu'on appelle un **type de données**.

- **Types simples ou primitifs**
  - Entier
  - Caractère
  - Point flottant (nombre réel/décimal)
  - Booléen

- **Types de référence**
  - Classe
  - Interface
  - Enumération
  - Chaine de caractères
  - Tableaux/Collections

12 L'encapsulation

Mécanisme qui permet d'une part, d'interdire l'accès direct aux **attributs** d'une classe à l'**extérieure** de celle-ci, et d'autre part, d'autoriser leurs accès/modification uniquement via des méthodes appelées **accesseurs** (on utilise souvent le vocable, *getter/setter*).
Pour ce faire, tous les attributs sont marqués du modificateur private et tous leurs accesseurs du modificateur public

```
public class Cone {

    public double rayon;
    public double hauteur;

    public double getRayon() {
       return rayon;
    }

    public void setRayon(double rayon) {
       this.rayon = rayon;
    }

    public double getHauteur() {
       return hauteur;
    }

    public void setHauteur(double hauteur) {
       this.hauteur = hauteur;
    }
}
```

Ici la classe Cone ne respecte pas le principe d'encapsulation car ses attributs *rayon* et *hauteur* ont le modificateur **public**.
Ainsi, en faisant : Cone objCone = new Cone()

objCone.rayon = 4.2 ✓

objCone.setRayon(4.2) ✓

```
public class Cone {

    private double rayon;
    private double hauteur;

    public double getRayon() {
       return rayon;
    }

    public void setRayon(double rayon) {
       this.rayon = rayon;
    }

    public double getHauteur() {
       return hauteur;
    }

    public void setHauteur(double hauteur) {
       this.hauteur = hauteur;
    }
}
```

Ici la classe Cone respecte le principe d'encapsulation car tous ses attributs sont **private** et leurs accesseurs sont **public**
Ainsi en faisant : Cone objCone = new Cone();

objCone.rayon = 4.2 ✗

objCone.setRayon(4.2) ✓

1 Les éléments de base

2 Le point d’entrée d’une application Java

3 Les mots clés, règles et conventions Java

La méthode main()

Le point d'entrée d'une application Java est une **classe publique** disposant d'une méthode spéciale appelée **main()**. Il ne peut y en avoir qu'une seule classe disposant de la méthode **main()** dans un programme disposant de plusieurs classes. On la qualifie souvent de **classe principale**

Signature :

```
public static void main(String[] args)
```

- dispose toujours d'un modificateur public
- est une méthode statique, marquée par le mot clé static
- ne renvoi aucune donnée résultat, son type de retour est toujours void
- on peut lui passer une liste de paramètres/arguments au démarrage de l'application, via le paramètre *args*

La méthode main()

Exemple :

```
public class Personne {

    public String nom;
    public String prenom;
    public int age;
    public String statutMatrimonial;

    public String getNomComplet (String nom, String prenom) {
             String nomComplet = prenom+nom;
             return nomComplet;
    }

    public int getAge () {
      return age;
    }

    public void changeStatut (String newStatut) {
      statutMatrimonial = newStatut;
    }

    public static void main(String[] args) {
        Personne pers = new Personne();
        String nomComplet = pers.getNomComplet("Durant","Patrick");
        System.out.println("Bonjour Monsieur " + nomComplet);
    }
}
```

Copiez cet exemple dans un fichier Personne.java et appliquez l'exécution ci-dessous pour vérifier que vous avez le bon résultat :

Exécution :

javac Personne.java
java Personne

⇒Résultat :
Bonjour Monsieur Patrick Durant

1 Les éléments de base

2 Le point d’entrée d’une application Java

3 Les mots clés, règles et conventions Java

Les mots clés du langage

Java comme tout autre langage, comporte un ensemble de mots clés (dont nous avons déjà vu certains) qui agrémentent sa syntaxe. Tout au long de ce cours, nous aborderons encore bon nombre d'entre eux.

abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	false	final	finally	float
for	if	implements	import	instanceof
int	interface	long	native	new
null	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	true	try	void	volatile
while				

Quelques règles et conventions

Instruction	Toute instruction Java se termine toujours par un point virgule (;)
Commentaire	Java permet d'ajouter des commentaires dans le code source, soit avec la syntaxe //<monCommentaireSurUneLigne> soit avec la syntaxe /* <monCommentaire SurPlusieursLignes> */
Classe	Le nom d'une classe doit toujours commencer par une lettre majuscule et respecter la notation **camel Case**. Exemple : `class MaClassePersonne`
Classe publique	Toute classe publique doit être écrit dans un fichier qui porte son nom et se termine par l'extension *.java*. Exemple : soit la classe publique **Personne**, alors le fichier contenant cette classe sera nommé **Personne.java**
Classe privée	En général, on définit une classe privée dans le même fichier qu'une classe publique, mais ce fichier ne portera que le nom de la classe publique
Attribut	Le nom d'un attribut doit toujours commencer par une lettre minuscule et respecter la notation **camel Case** ensuite. Exemple : `public int maVariableEntiere`
Méthode	Le nom d'une méthode doit toujours commencer par une lettre minuscule et respecter la notation **camel Case** ensuite. Exemple : `public int getNomComplet(…)`
Constante	Il est préférable de mettre le nom d'une constante en toute majuscule et séparé les mots par des tirets de 8 si le nom de la variable est un mot composé. Exemple : `public final String PRENOM_NOM = ''Samuel DUPONT''`


SUCCESS