Java
Création de classes en Java

Objectifs

- Créer des classes Java
- Ajouter des attributs d’instance et de classe
- Ecrire des constructeurs
- Définir des méthodes d’instance et des méthodes de classe

Déclaration d’une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe

Déclaration d’une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe
- Le nom d’une classe commence par une majuscule Pour éviter les noms de var et de classe

Déclaration d’une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe
- Le nom d’une classe commence par une majuscule
- On utilise une syntaxe en UpperCamelCase Eviter les "_" et le "-" ne fonctionne pas

Déclaration d’une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe
- Le nom d’une classe commence par une majuscule
- On utilise une syntaxe en UpperCamelCase

- Une classe est définie dans un package

Déclaration d'une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe
- Le nom d'une classe commence par une majuscule
- On utilise une syntaxe en UpperCamelCase

- Une classe est définie dans un package
- Le nommage du package est défini par ce qu'il représente en tant que regroupement logique
  - exemple : java.ateliers.cours1 <=> Répertoire: java/repertoire/ateliers

Déclaration d'une classe

- Une classe est définie dans un fichier portant rigoureusement le même nom que cette classe
- Le nom d'une classe commence par une majuscule
- On utilise une syntaxe en UpperCamelCase

- Une classe est définie dans un package
- Le nommage du package est défini par ce qu'il représente en tant que regroupement logique
  - exemple : java.ateliers.cours1
- Le répertoire de la classe correspond au nom du package
  - exemple : src/java/ateliers/cours1/

Déclaration d'une classe

mot clé java interdit (package mal nommé)

```
package java.demos.cours3;

/*
 * Classe représentant un dé à jouer
 */
public class De {
    /*
     * Pour l'heure, la classe est vide
     */

}
```

Déclaration du package

Déclaration de la classe

Les attributs d’instance

- Ensemble de variables caractérisant une instance de classe

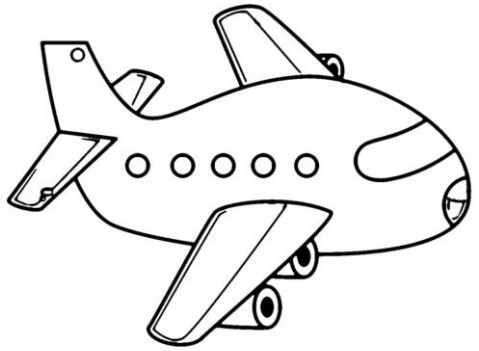
Les attributs d'instance

- Ensemble de variables caractérisant une instance de classe

Avion
+ carburant : String + nbPlaces : int

Les attributs d'instance

- Ensemble de variables caractérisant une instance de classe

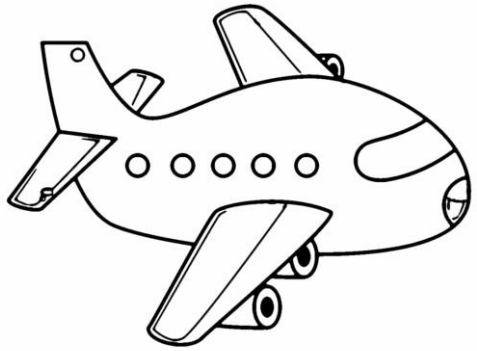


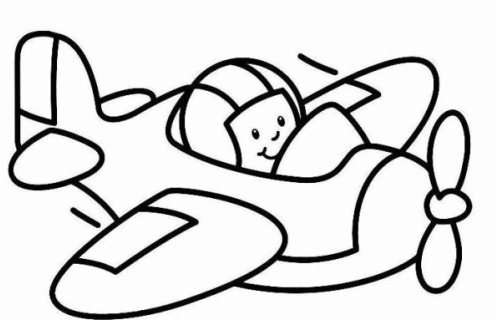
Avion
+ carburant : String + nbPlaces : int

Instance 1 : Boeing	
type	"Kérozène"
nbPlaces	80

Les attributs d'instance

- Ensemble de variables caractérisant une instance de classe





Avion
+ carburant : String + nbPlaces : int

Instance 1 : Boeing	
carburant	"Kérozène"
nbPlaces	80

Instance 2 : Monoplan	
carburant	"Ressort"
nbPlaces	1

Les attributs d'instance

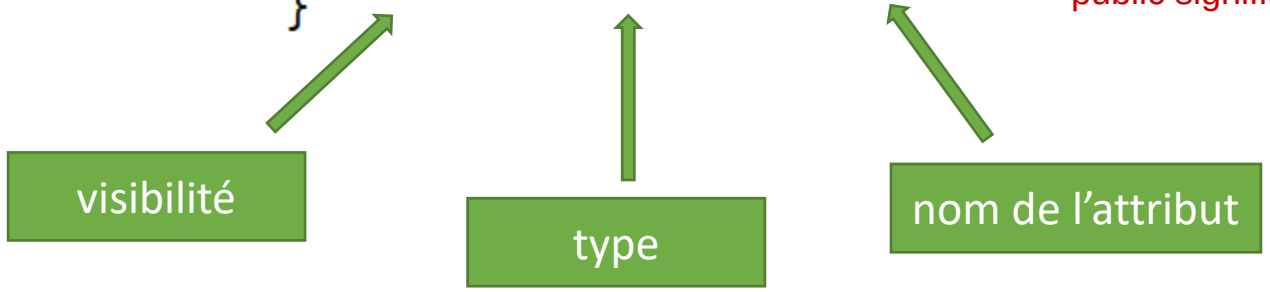
```java
package java.demos.cours3;

/*
 * Classe représentant un dé à jouer
 */
public class De {
    private int nbFaces;
    private int resultat;
}
```

Les attributs d'instance

```java
package java.demos.cours3;

/*
 * Classe représentant un dé à jouer
 */
public class De {
    private int nbFaces;
    private int resultat;
}
```

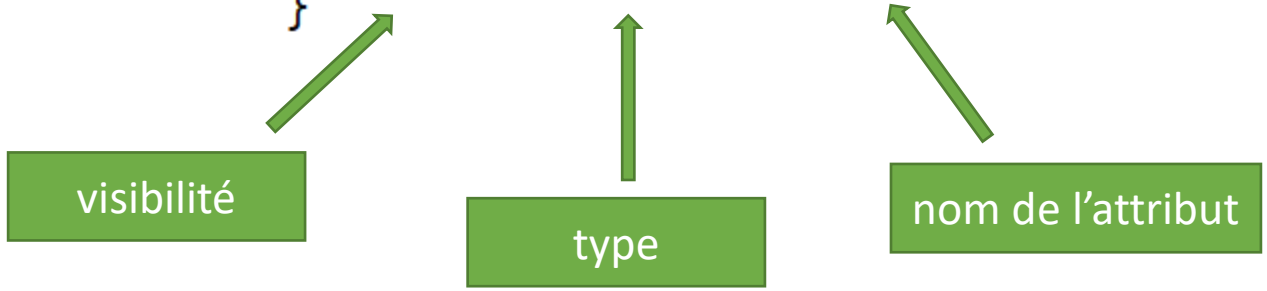


private => visible que dans la class DE
presque toujours private dans une class
public signifie modifiable par n'importe qui en dehors de la class

Les attributs d'instance

```java
package java.demos.cours3;

/*
 * Classe représentant un dé à jouer
 */
public class De {
    private int nbFaces;
    private int resultat;
}
```



De	
- nbFaces : int - resultat : int	le - signifie private

La visibilité

- La visibilité désigne l'accessibilité d'un attribut par les autres classes du projet

Visibilité	Mot clé	Signification
Privée	private	Accessible uniquement au sein de la classe
Publique	public	Accessible de n'importe où
Package	×	Accessible uniquement au sein de ce package
Protégé	protected	Accessible au sein de la classe et de ses héritiers

presque tjrs ds une classe

presque tjrs pour une méthode

peu courrant (par défaut)

un 2eme projet est accessible sur un 1er via un import auquel cas les éléments doivent être public

Le principe d'encapsulation

- Une classe est responsable de ses données
  - De l'extérieur de la classe, il est impossible de manipuler directement les attributs
  - Par contre, il est possible d'appeler une des méthodes de la classe, qui a accès aux attributs

toutes les méthodes sont publics (99% de temps)

De
- nbFaces : int - resultat : int
+ getNbFaces() : int + setNbFace (nbFaces : int) + getResultat() : int + setResultat (resultat : int)

get: lire attribut (getter)

set: affecter un attribut (setter)

Intérêt: permet de contrôler les données (ex.: un dé de 5 n'existe pas)
permet un contrôle de class à ne pas faire partout
Les get et set ne servent presque que à faire des contrôles.

Les méthodes d'instance

```
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public void setNbFaces(int nbFaces) {
        this.nbFaces = nbFaces;
    }

    public void lancer() {
        this.resultat = rand.nextInt(this.nbFaces) + 1;
    }

    public int getResultat() {
        return resultat;
    }
}
```

Les méthodes d'instance

```
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();
    public void setNbFaces(int nbFaces) {
        this.nbFaces = nbFaces;
    }
    public void lancer() {
        this.resultat = rand.nextInt(this.nbFaces) + 1;
    }

    public int getResultat() {
        return resultat;
    }
}
```

parfois on commence par "_" pour ne pas utiliser de this

le mot clé "this" permet d'accéder aux attributs d'instance

static est incompatible avec this

"this" désigne l'attribut d'instance (le private de la class) différent de l'argument (int nbFaces)
ici on modifie une instance et non une méthode

méthode de la class De
on aurait pu faire return resultat dans lancer,
mais le resultat n'aurait été visible qu'une fois

Les méthodes d'instance

Bon reflexe de mettre les getters et setters tjrs

```
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public void setNbFaces(int nbFaces) {
        this.nbFaces = nbFaces;
    }

    public void lancer() {
        this.resultat = rand.nextInt(this.nbFaces) + 1;
    }

    public int getResultat() {
        return resultat;
    }
}
```

Les méthodes permettant de lire des attributs d'instance sont appelés des getters

Les méthodes permettant de modifier des attributs d'instance sont appelés des setters

Les méthodes d'instance

```
public class TestDe {

    public static void main(String[] args) {
        De monDe = new De();
        monDe.setNbFaces(6);
        do {
            monDe.lancer();
            System.out.println("Le dé à fait un : " + monDe.getResultat());
        } while (monDe.getResultat() != 6);
    }

}
```

```
Le dé à fait un : 4
Le dé à fait un : 1
Le dé à fait un : 2
Le dé à fait un : 6
```

Les méthodes d'instance

```
public class TestDe {

    public static void main(String[] args) {
        De monDe = new De();
        monDe.nbFaces = 6;
        do {
            monDe.lancer();
            System.out.println("Le dé à fait un : " + monDe.getResultat());
        } while (monDe.getResultat() != 6);
    }

}
```

Le constructeur

Toute classe dispose d'un constructeur par défaut

- Il est présent lorsqu'aucun autre constructeur n'a été défini dans la classe
- Il ne prend aucun paramètre
- Il est forcément public 99% du temps, sinon on ne peut pas faire de new

```
public class TestDe {

    public static void main(String[] args) {
        De monDe = new De();
        monDe.setNbFaces(6);
        do {
            monDe.lancer();
            System.out.println("Le dé à fait un : " + monDe.getResultat());
        } while (monDe.getResultat() != 6);
    }

}
```

Le constructeur

Il est possible de créer son propre constructeur. Il doit :

- Porter le même nom que la classe
- Ne pas avoir de type de retour

static= commun à toutes ses instances (!= constant)

On crée souvent un constructeur vide lorsqu'on crée un autre (il est souvent utile)

```
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public De(int nbFaces) {
        this.nbFaces = nbFaces;
        this.lancer();
    }
```

Le constructeur est là pour affecter des valeurs aux attributs.

Ajout constructeur:
ici on veut un dé avec un nbFace
en + on fait un jet.
Rem.: le constructeur doit avoir a la virgule près
la syntaxe de la classe

Le constructeur

Il est possible de créer son propre constructeur. Il doit :

- Porter le même nom que la classe
- Ne pas avoir de type de retour

Comme nous avons créé notre constructeur personnalisé, le constructeur par défaut n'existe plus.

```
public static void main(String[] args) {
    De monDe = new De();
    monDe.setNbFaces(6);
    do {
        monDe.lancer();
        System.out.println("Le dé à fait un : " + monDe.getResultat());
    } while (monDe.getResultat() != 6);
}
```

Le constructeur

Il est possible de créer son propre constructeur. Il doit :

- Porter le même nom que la classe
- Ne pas avoir de type de retour

Comme nous avons créé notre constructeur personnalisé, le constructeur par défaut n'existe plus.

Nous pouvons cependant utiliser notre nouveau constructeur :

```
public static void main(String[] args) {
    De monDe = new De(6);
    do {
        monDe.lancer();
        System.out.println("Le dé à fait un : " + monDe.getResultat());
    } while (monDe.getResultat() != 6);
}
```

Le constructeur

Une surcharge consiste à créer une méthode portant le même nom, mais avec des paramètres différents.

Il est possible de créer une (ou plusieurs) surcharge du constructeur.

Le constructeur

Une surcharge consiste à créer une méthode portant le même nom, mais avec des paramètres différents.

Il est possible de créer une (ou plusieurs) surcharge du constructeur.

```
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public De() {
        this.nbFaces = 6;
        this.lancer();
    }

    public De(int nbFaces) {
        this.nbFaces = nbFaces;
        this.lancer();
    }
```

Le constructeur

Une surcharge consiste à créer une méthode portant le même nom, mais avec des paramètres différents.

Il est possible de créer une (ou plusieurs) surcharge du constructeur.

```java
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public De() {
        this.nbFaces = 6;
        this.lancer();
    }

    public De(int nbFaces) {
        this.nbFaces = nbFaces;
        this.lancer();
    }
```

```java
public class De {
    private int nbFaces;
    private int resultat;
    private static Random rand = new Random();

    public De() {
        this(6);
    }

    public De(int nbFaces) {
        this.nbFaces = nbFaces;
        this.lancer();
    }
```

Ce constructeur appelle celui qui suit
(il est identique à celui de sa gauche)

Les attributs de classe

Il s'agit d'une variable collective, commune à toutes les instances.

Elle doit être définie avec le mot clé **static**

```
public class Avion {

    private String carburant;
    private int nbPlace;

    private static int nbAvionsConstruits;

    static {
        nbAvionsConstruits = 0;
    }

    public Avion() {
        nbAvionsConstruits++;
    }
}
```

A mettre dans tous les constructeurs s'il y en a d'autres

Les attributs de classe

Il s'agit d'une variable collective, commune à toutes les instances.

Elle doit être définie avec le mot clé **static**

```
public class Avion {

    private String carburant;
    private int nbPlace;

    private static int nbAvionsConstruits;

    static {
        nbAvionsConstruits = 0;
    }

    public Avion() {
        nbAvionsConstruits++;
    }
}
```

Attribut de classe

Les attributs de classe

Il s'agit d'une variable collective, commune à toutes les instances.

Elle doit être définie avec le mot clé **static**

```
public class Avion {

    private String carburant;
    private int nbPlace;

    private static int nbAvionsConstruits;

    static {
        nbAvionsConstruits = 0;
    }

    public Avion() {
        nbAvionsConstruits++;
    }
}
```

Initialisation statique : le nombre d'avions construits sera initialisé à 0 au lancement du programme.

Attribut de classe

Les attributs de classe

Il s'agit d'une variable collective, commune à toutes les instances.

Elle doit être définie avec le mot clé **static**

```
public class Avion {

    private String carburant;
    private int nbPlace;

    private static int nbAvionsConstruits;

    static {
        nbAvionsConstruits = 0;
    }

    public Avion() {
        nbAvionsConstruits++;
    }
}
```

Ce bout de code est exécutée une seule fois au lancement de l'application (pas prioritaire)

Initialisation statique : le nombre d'avions construits sera initialisé à 0 au lancement du programme.

Attribut de classe

Puis, à chaque nouvel avion, le nombre d'avion construits augmente de 1.

Les méthodes de classe

De même, il s’agit d’une opération qui ne dépend pas d’une instance en particulier.

Elle est également définie avec le mot clé **static**

Une méthode de classe ne peut accéder qu’aux attributs de classe (pas aux attributs d’instance)

Les méthodes de classe

De même, il s'agit d'une opération qui ne dépend pas d'une instance en particulier.

Elle est également définie avec le mot clé **static**

Une méthode de classe ne peut accéder qu'aux attributs de classe (pas aux attributs d'instance)

```
private static void verifNbFaces(int nbFaces) throws Exception {
    if (nbFaces < 2) {
        throw new Exception("Un dé doit avoir au moins deux faces");
    }
}

public void setNbFaces(int nbFaces) throws Exception {
    verifNbFaces(nbFaces);
    this.nbFaces = nbFaces;
}
```

Exception: peut renvoyer des erreurs

Les méthodes de classe

De même, il s'agit d'une opération qui ne dépend pas d'une instance en particulier.

Elle est également définie avec le mot clé **static**

Une méthode de classe ne peut accéder qu'aux attributs de classe (pas aux attributs d'instance)

```
private static void verifNbFaces(int nbFaces) throws Exception {
    if (nbFaces < 2) {
        throw new Exception("Un dé doit avoir au moins deux faces");
    }
}

public void setNbFaces(int nbFaces) throws Exception {
    verifNbFaces(nbFaces);
    this.nbFaces = nbFaces;
}
```

Les méthodes d'instance et méthodes de classe

	Méthode d'instance	Méthode de classe
Mot clé	×	**static** static signifie propre à tous les dés (pas à un dé en particulier)
Accès aux attributs de classe	Oui	Oui
Accès aux attributs d'instance	Oui	Non

Appel depuis la classe	this.nomMethode();
	ex : this.lancer();
		nomClasse.nomMethode();
		ex : De.verifNbFaces(6);
		A noter que nomClasse est facultatif

Appel hors de la classe	nomInstance.nomMethode();
	ex : monDe.lancer();
		nomClasse.nomMethode();
		ex : De.verifNbFaces(6);
		Dans ce cas, nomClasse est obligatoire

Fabrique par méthode de classe

- Constructeur privé
- Méthodes de classe permettant de récupérer une instance

- Cela permet :
  - de nommer explicitement la méthode
  - de n'utiliser le constructeur que si nécessaire
  - de retourner un sous-type le cas échéant

Fabrique par méthode de classe

- Constructeur privé
- Méthodes de classe permettant de récupérer une instance

- Cela permet :
  - de nommer explicitement la méthode
  - de n'utiliser le constructeur que si nécessaire
  - de retourner un sous-type le cas échéant

LocalDate
- LocalDate() + now() : LocalDate + of(year : int, month: int, dayOfMonth : int) : LocalDate + ofEpochDay(epochDay : long) : LocalDate + parse(text : CharSequence, formatter: DateTimeFormatter) : LocalDate

CharSequence : type plus général que les String

Atelier 3
Visite chez le médecin

Java
Création de classes en Java