

M2103 (POO)



Bases de la programmation orientée objet

Responsable : Cristina Onete

cristina.onete@gmail.com

<https://onete.net/teaching.html>

Les règles du jeu

- ▶ La ponctualité : obligatoire et à la minute près

1 minute de retard = absence

- ▶ Le travail : la programmation s'apprend... en programmant

TD, TP : indispensables

Travail au delà des TPs/TDs : fortement conseillé

- ▶ Politesse et courtoisie :

Je tutoie : dites-moi si cela vous dérange

CM : pas d'ordinateur, pas de portable

TD/TP : le travail en premier, Facebook après

Ce cours dans le PPN

Algorithmique, Programmation, Langages

M1102: Intro algos & programmation

M1103: Str. données & algos fondam.

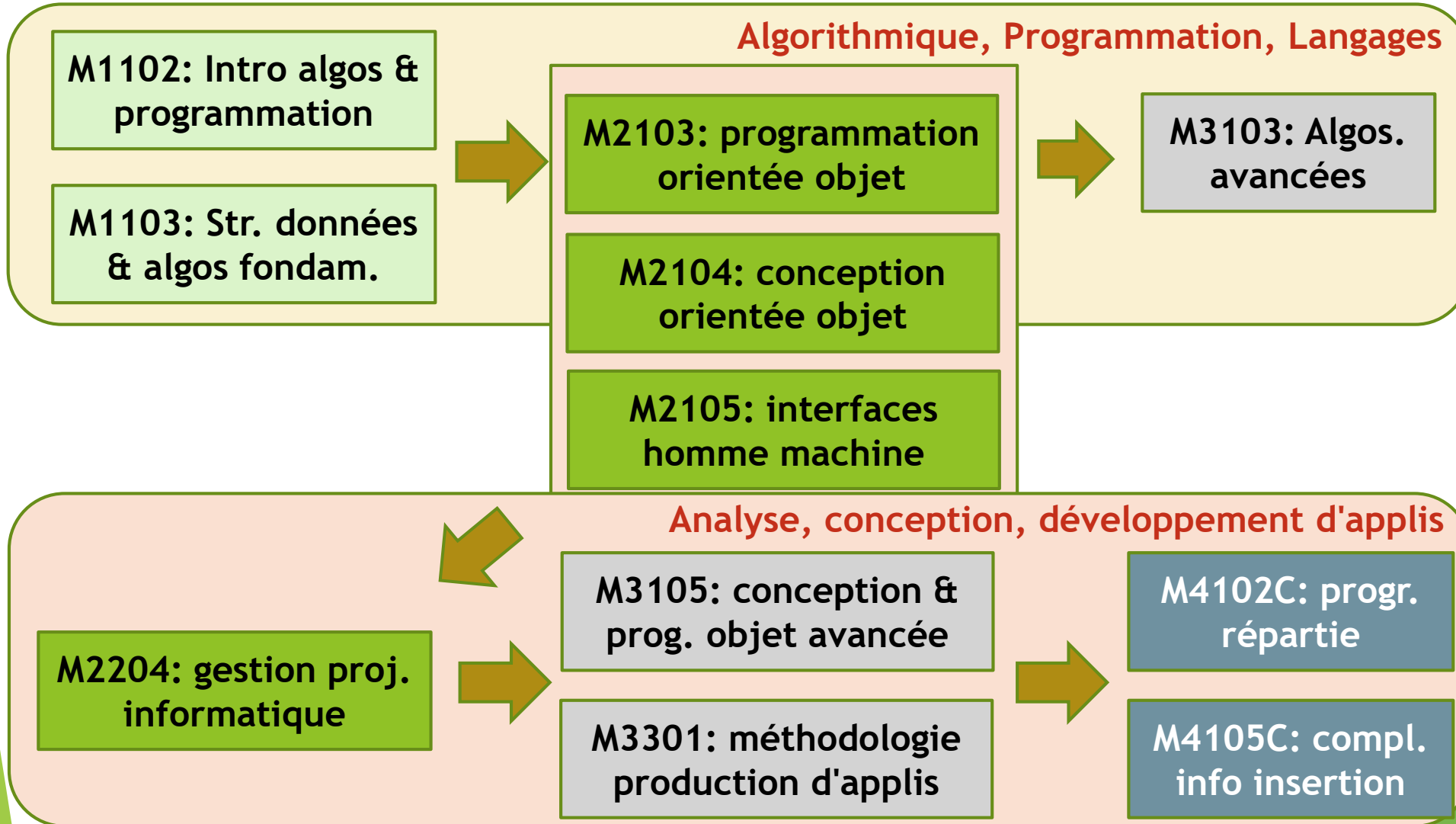


M2103: programmation orientée objet



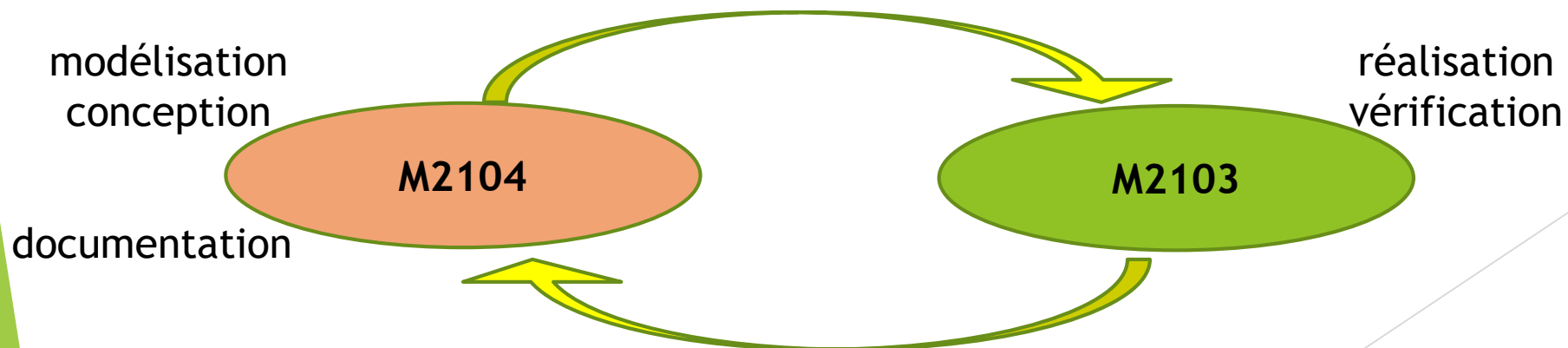
M3103: Algos. avancées

Ce cours dans le PPN



Contenu de ce module

- ▶ Nous **allons apprendre** comment programmer en Java
 - ▶ CM : des concepts et notions de programmation
 - ▶ TD : réflexion et planification de la programmation
 - ▶ TP : la programmation elle-même
- ▶ Prérequis : M1103 - struct. de données & algos. fondamentaux
- ▶ Les cours M2103 et M2104 se font ensemble !



Projet suivi, contrôle, examen

- ▶ Notre but **dans les TPs** : programmer une chasse aux Pokemons
 - ▶ Un projet suivi : on va programmer un peu plus dans chaque TP
 - ▶ Des corrections fournies : c'est **votre obligation** à les étudier et les intégrer dans votre code !
 - ▶ La **créativité** est encouragée : découvrez Java en programmant
- ▶ Contrôle continu :
 - ▶ **2 TPs notés** - la moyenne compte pour 15% de la note finale
 - ▶ **2 TDs notés** - la moyenne compte pour encore 15%
Chaque TD commence avec un quiz
- ▶ Examen finale -70% de la note



Des questions ??

Quelques superstitions...

- ▶ La programmation, c'est une science de geeks

La programmation = cool & utile pour tous les jours !

- ▶ C'est que les garçons qui peuvent programmer

L'inventrice de la programmation = Ada Lovelace

- ▶ Mon ordinateur ne m'aime pas

C'est peut-être l'inverse...

- ▶ Un langage de programmation suffit pour la vie

Développeur avec 1 langage = Linguiste avec 1 langue

Quelques vérités

- ▶ Les langages de programmation **évoluent** tout le temps
 - ▶ Mais suivre l'évolution est facile, tandis que l'apprentissage de tout un langage est difficile !
- ▶ Le langage Java est **modulaire** et **beaucoup utilisé** en pratique
- ▶ Il existent de **bonnes pratiques** et de **mauvaises pratiques** dans la programmation
- ▶ Avec la programmation **on peut changer la vie**/le monde

Convaincu(e)s ? Ben, allons-y !

Java, c'est quoi ?

- ▶ 1991 : James Gosling, Mike Sheridan, Patrick Naughton s'embarquent au **projet du développement de Java**
- ▶ 1995 : Sun Microsystems s'engage : "Write Once Run Anywhere" première **implémentation publique** de Java
- ▶ 1998-1999 : Java 2, inclue J2EE des API pour des **applications serveur**, J2ME pour des **applications mobiles**

J2 SE
(standard)

J2 EE
(entreprise)

J2 ME
(micro)

- ▶ 2006-2007 : Java **publie leur code** sous la licence GNU GPL
- ▶ 2010 : Oracle **achète** Java. Aujourd'hui elle est partout

Les principes de Java

source: Oracle, 1999

Simple, orientée objet et intuitive

Robuste et sécurisée

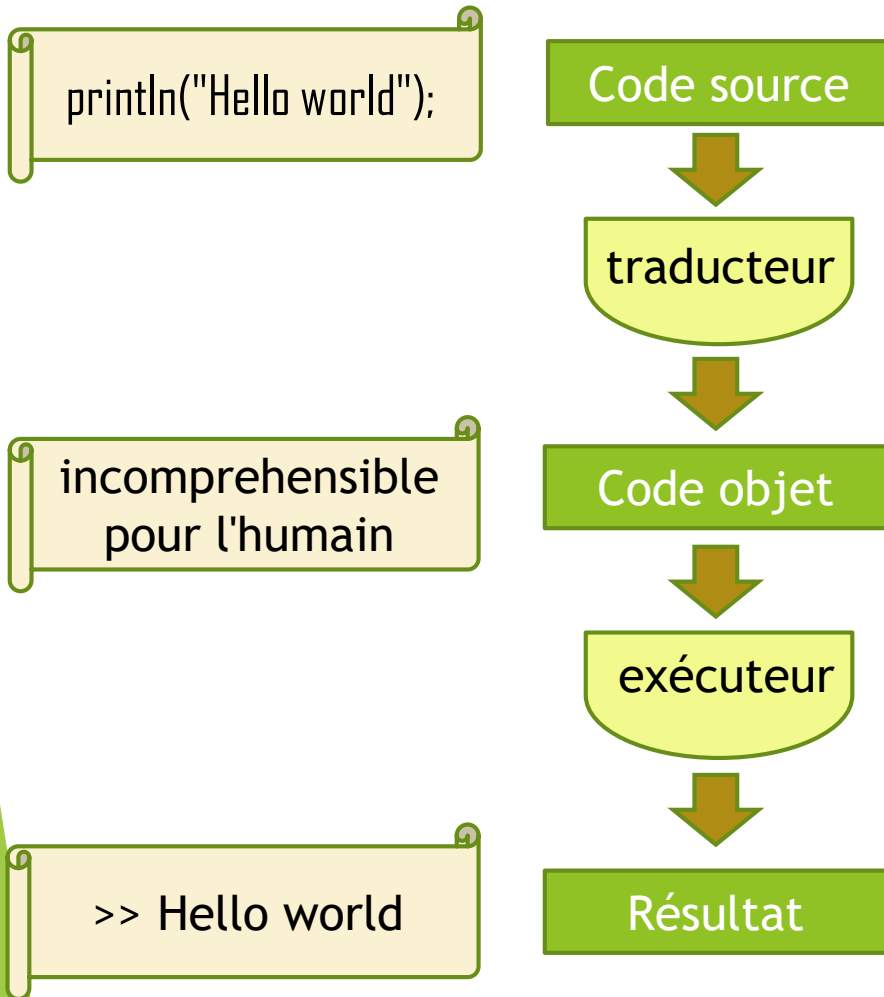
Portable et indépendante de l'architecture

Exécutable à haute performance

Interprétée, dynamique et fileté

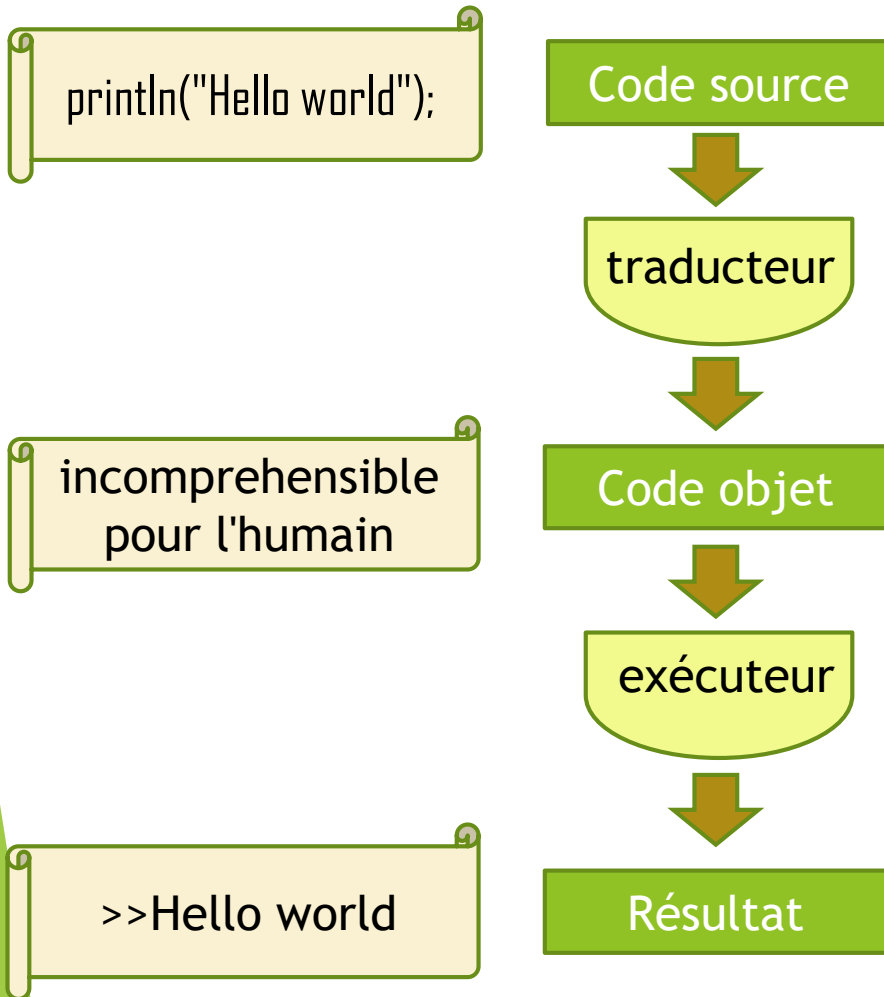
Comment Java diffère-t-elle d'autres langages de programmation?

Compilation à exécution : le cas général



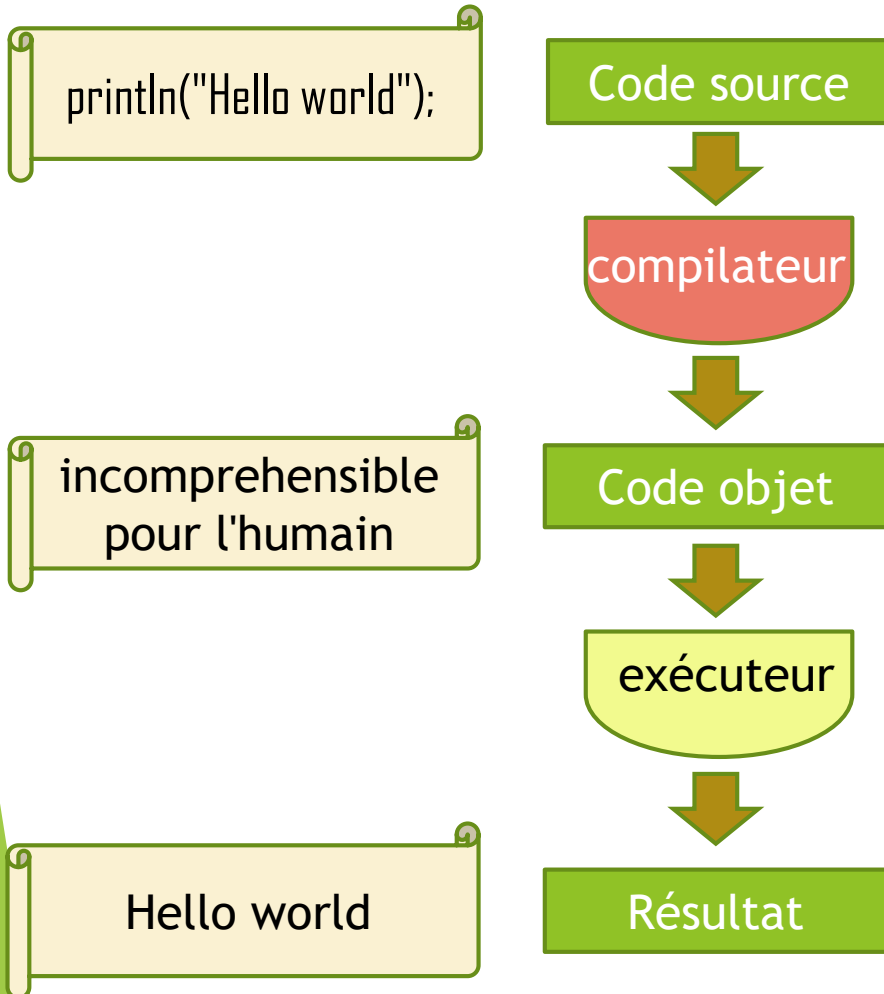
- ▶ 3 façons d'y arriver :

Compilation à exécution



► Façon 1 : Compilation

Compilation à exécution

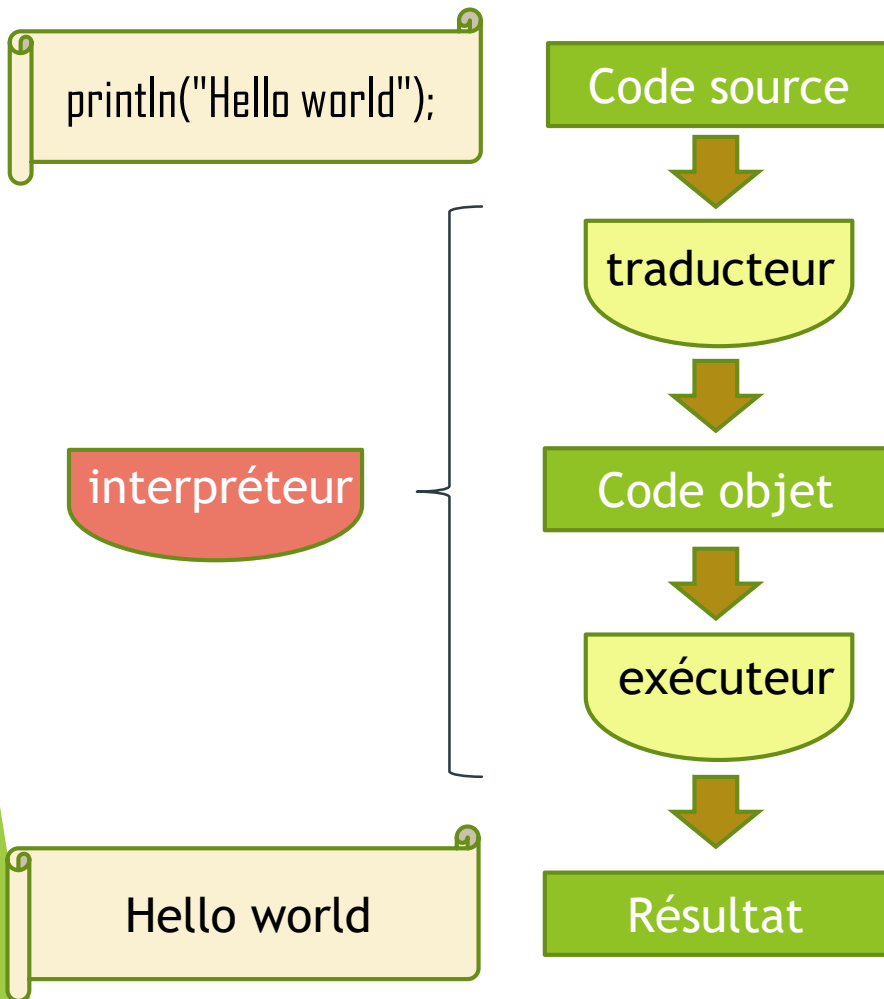


► Façon 1 : Compilation

- Traducteur = compilateur
- Le compilateur **compile** tout le fichier et **génère** un fichier dit "executable"
- Ce dernier peut être exécuté sur la même machine ou une autre
- Présente en : **C**, **C++**

- + Exécution rapide de l'exécutable
- + Plus difficile de désassembler le code

Compilation à exécution

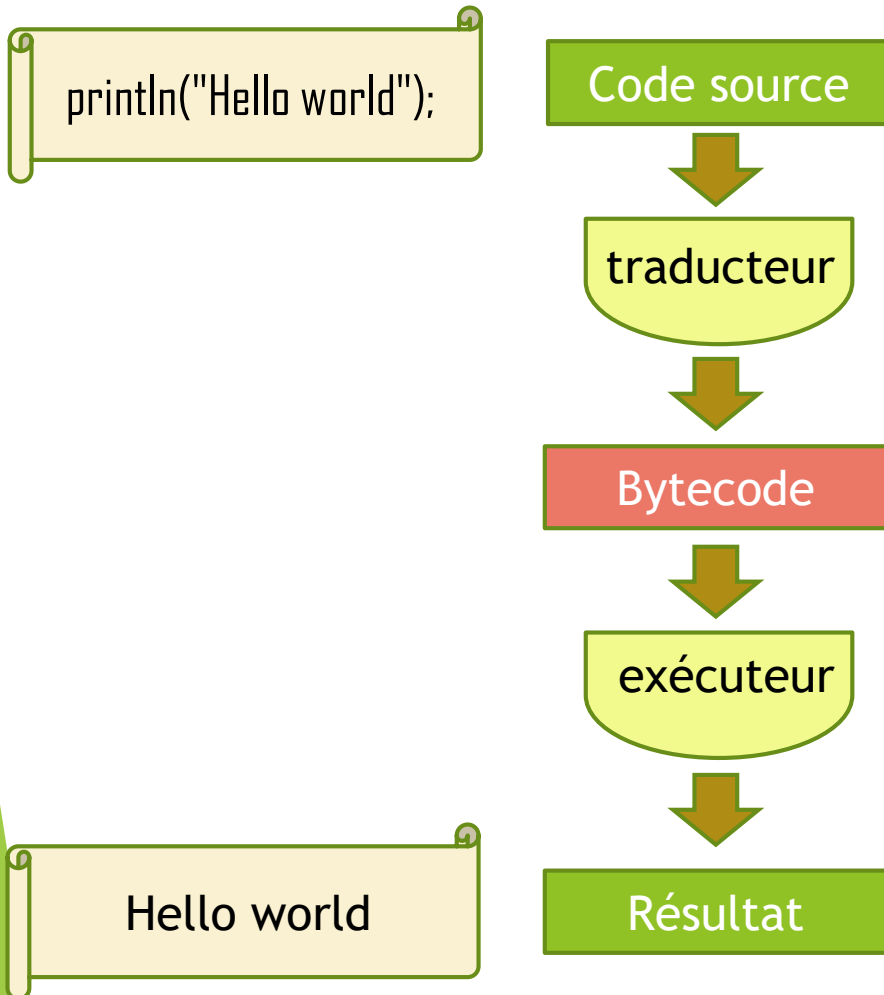


► Façon 2 : Interprétation

- **Compilation** et **exécution** en même temps
- L'exécution se fait forcément sur la même machine
- Présente dans : **Prolog, Basic**

✚ Les tests peuvent s'effectuer plus vite

Compilation à exécution



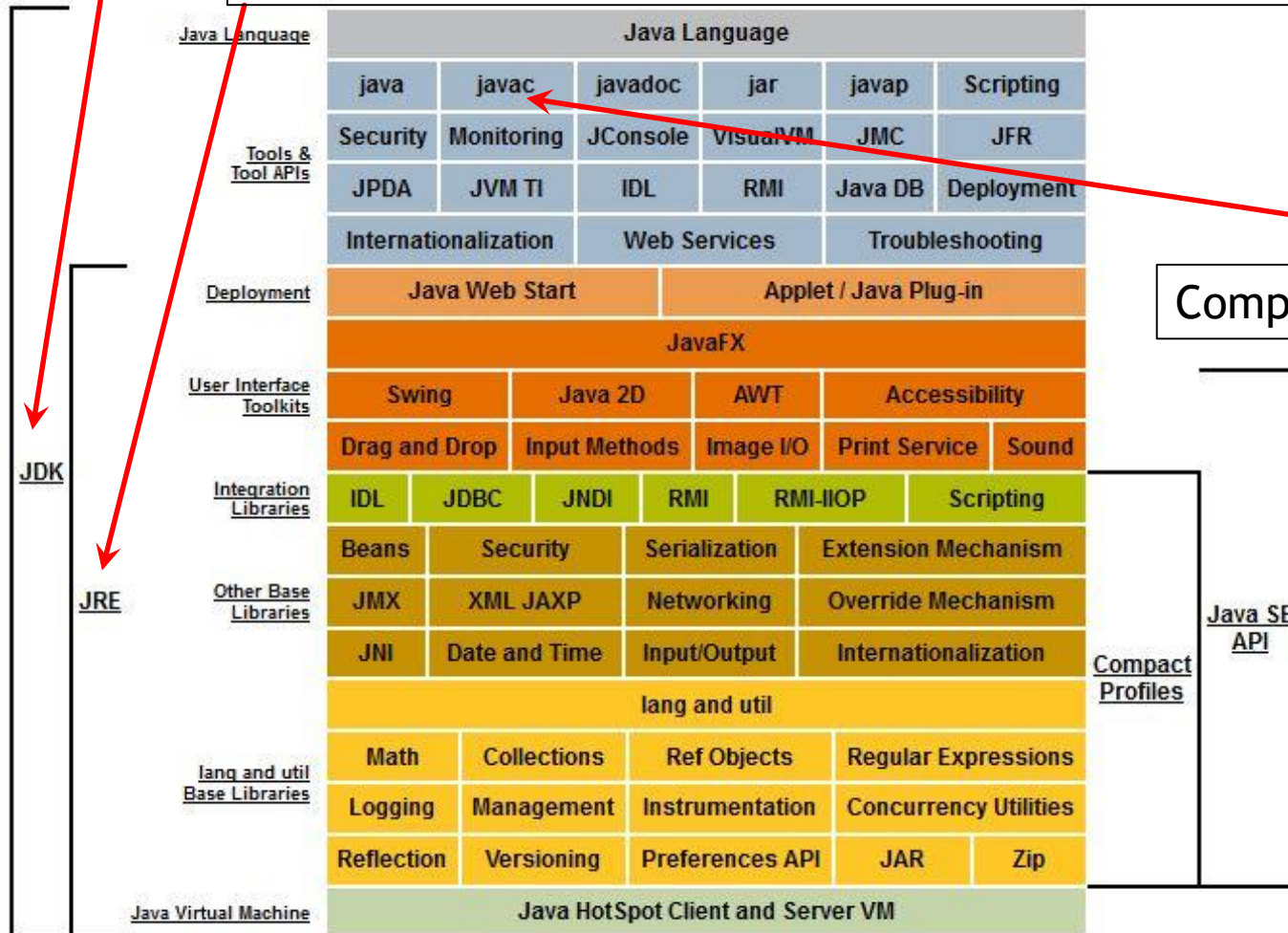
- ▶ Façon 3 : Semi-compilation
 - ▶ **Compilation** dans un code objet = **bytecode**
 - ▶ Le bytecode est exécutable par une **machine virtuelle**
 - ▶ Présente dans : **C#, Java**

✚ Compromis entre compilation et interprétation

Les composantes de Java

Java Development Kit = trousse de développement Java

Java Runtime Environment = environnement d'exécution



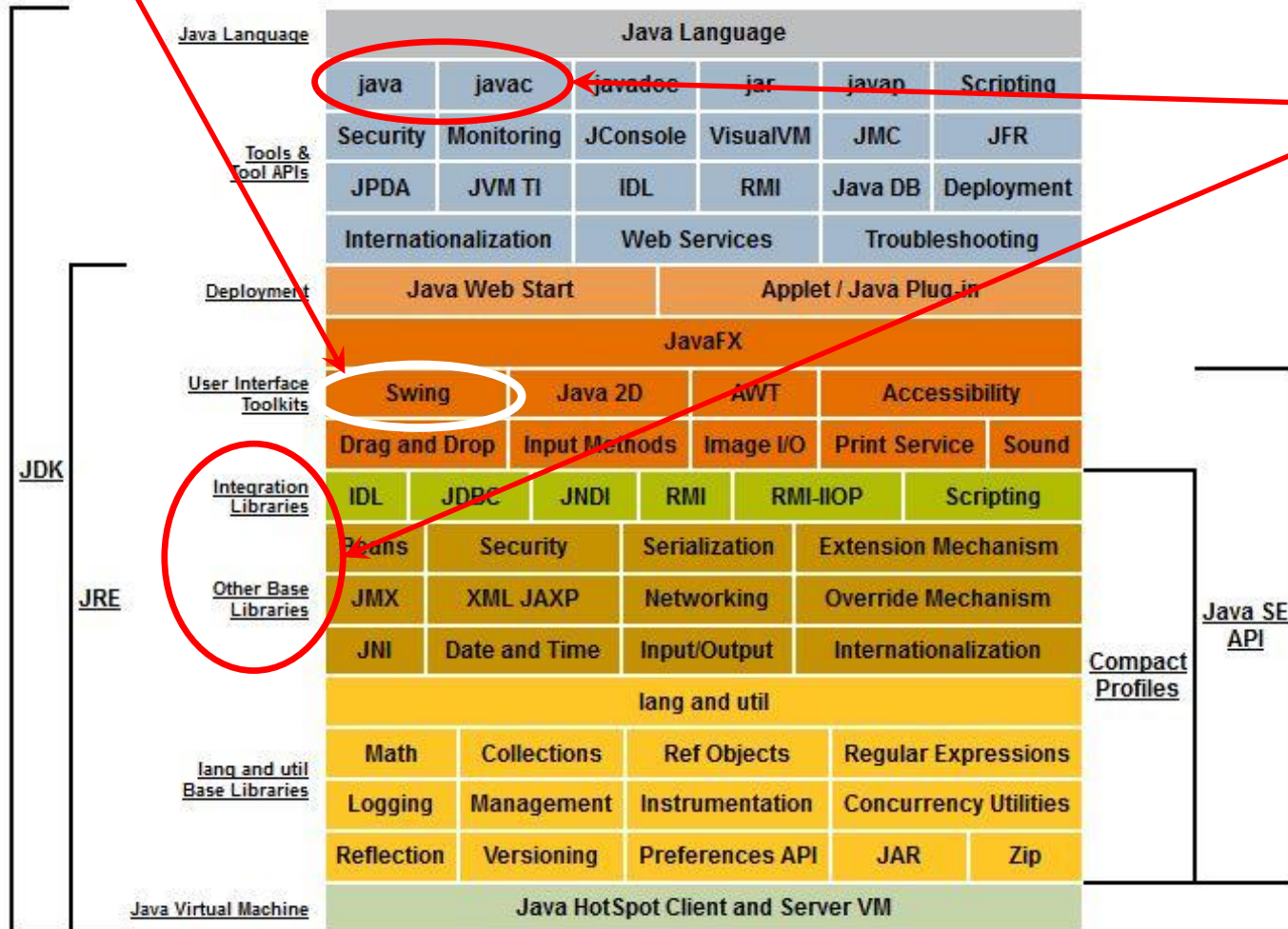
Compilateur : produit du bytecode

Les composantes de Java

IHM

Source: oracle.com

Plutôt dans ce cours



Variables, méthodes, attributs

► Langage Impératif (C)

- Orienté vers **fonctions et procédures**
- Le programme se compose de **descriptifs des fonctions** et d'**appels aux fonctions**
- Chaque fonction est **caractérisée par sa "signature"** : types I/O, nom
- Variables **globales et locales**

► Langage orienté objet (Java)

- Orienté vers **objets** (décrits par **des classes**)
- Chaque objet instancie **une classe**; il a des **attributs et des méthodes** propres
- Chaque méthode est **caractérisée par sa signature** et **associée à une classe**
- Toute variable est **locale** : il n'y a pas de code en dehors des classes

De plus, Java est un langage bavard ! (verbeux)

Exemple de variables et méthodes

```
public class Eleve{
```

```
    public String nom;
```

```
    ...
```

```
    public String getNom(){
```

```
        return this.nom;
```

```
    }
```

```
}
```

← Variable spécifique à
la classe Eleve

Exemple de variables et méthodes

```
public class Eleve{  
    public String nom; ← Type de variable  
    ...  
    public String getNom(){  
        return this.nom;  
    }  
}
```

Exemple de variables et méthodes

```
public class Eleve{  
    public String nom;  
    ...  
    public String getNom(){  
        return this.nom;  
    }  
}
```

Méthode spécifique à
la classe Eleve



Exemple de variables et méthodes

```
public class Eleve{  
    public String nom;  
    ...  
    public String getNom(){  
        return this.nom;  
    }  
}
```

← Type de variable rendue

Exemple de variables et méthodes

```
public class Eleve{  
    public String nom;  
    ...  
    public String getNom()  
        return this.nom;  
}  
}
```

← Ne prend aucun paramètre en entrée

Exemple de variables et méthodes

```
public class Eleve{  
    public String nom;  
    ...  
    public String getNom(){  
        return this.nom;  
    }  
}
```

Rend la valeur stockée par la variable nom de l'instance de classe pour laquelle on appelle la méthode

Appeler une méthode en Java

```
public class Eleve{  
    public String nom;  
    public Eleve(String nom){  
        this.nom = nom;  
    }  
    public String getNom(){  
        return this.nom;  
    }  
}
```

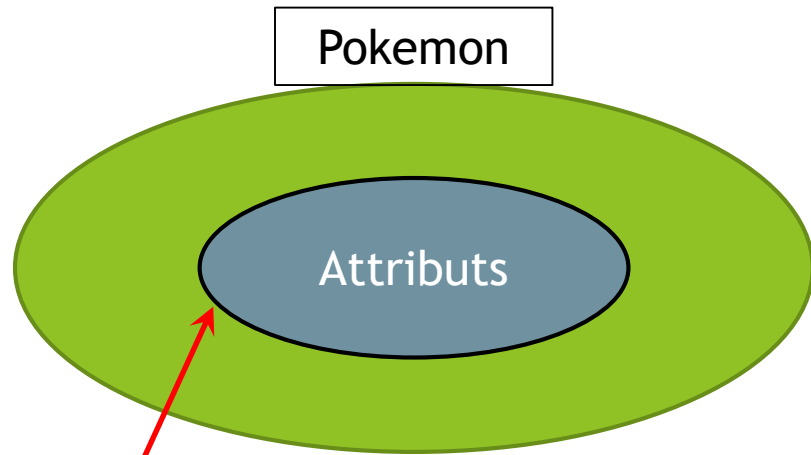
```
public class Appel{  
    public static void main(String[] args){  
        public final jeanDupont = new Eleve("Dupont");  
        System.out.println(jeanDupont.getNom());  
    }  
}
```

Appeler la méthode `getNom()` pour l'objet `jeanDupont`, qui est une instance de la classe `Eleve`

Les bases de Java : objets, classes, execution

C'est quoi une "classe" ?

- ▶ Classe : une représentation abstraite, un modèle de qq. chose
 - ▶ Par exemple : "Etudiant", "Animal", "Ordinateur", "Pokémon"...
 - ▶ A des attributs et des méthodes



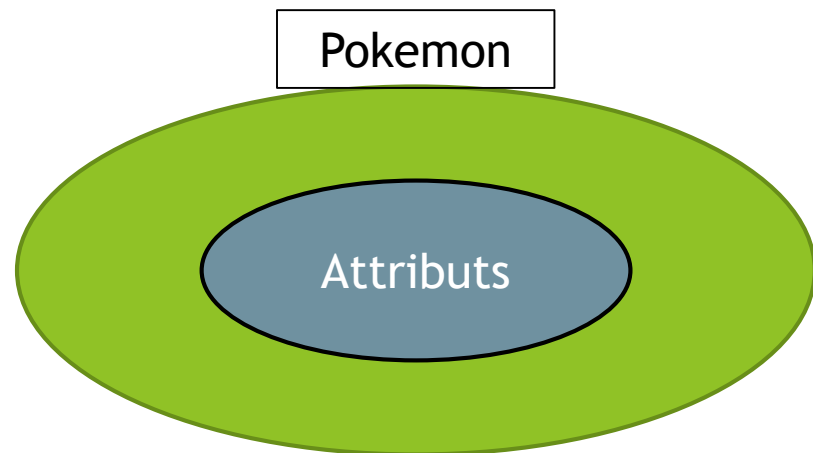
Les variables locales **qui caractérisent la classe**

```
7  /**
8   *
9   * @author crist
10  */
11  public class Pokemon {
12      // Ses attributs
13      private String name;
14      private String type;
15      private int level;
16
17      // Puis les méthodes
18  }
19
```



C'est quoi un "objet" ?

- ▶ Classe : une représentation abstraite, un modèle de qq. chose
- ▶ En Java, chaque objet est issu d'une classe, qui le définit
 - ▶ L'objet est unique et doit être "personnalisé"



p1 = new Pokemon(...);

```
name = "Piplup"  
type = "Eau"  
level = 5
```

```
System.out.println(p1.name);  
>> Piplup
```

```
this.name = Piplup;  
System.out.println(name);  
>> Piplup
```

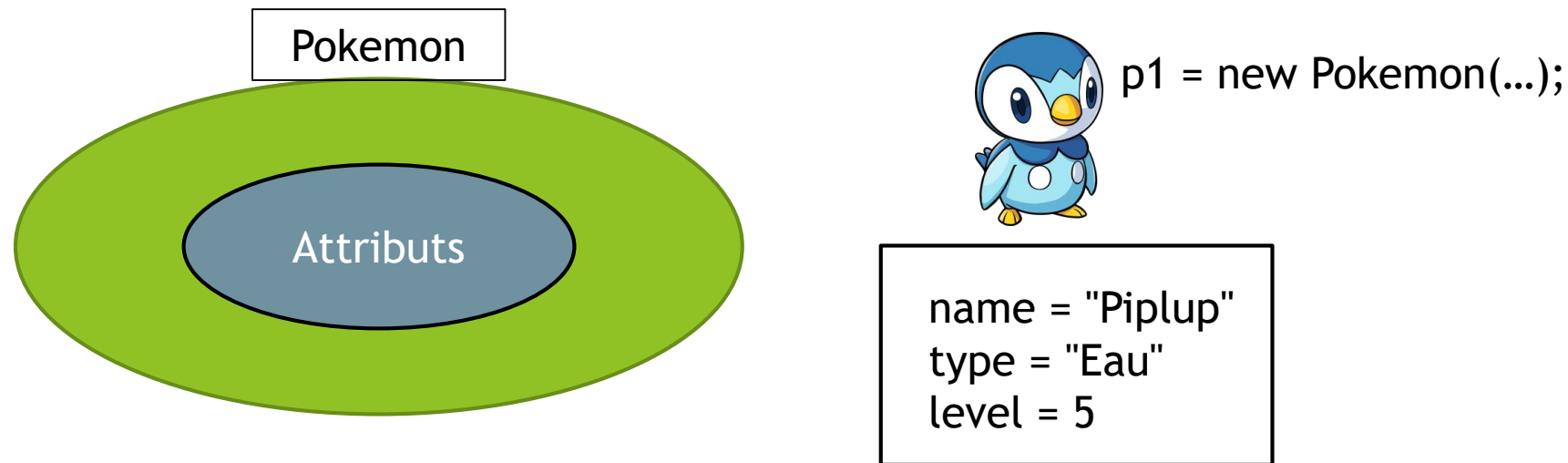
Manipuler un attribut :

- dans la classe : <nom d'attribut> ou `this.<nom d'attribut>`
- en dehors de la classe : <nom de l'objet>.<nom d'attribut>



C'est quoi un "objet" ?

- ▶ Classe : une représentation abstraite, un modèle de qq. chose
- ▶ En Java, chaque objet est issu d'une classe, qui le définit
 - ▶ L'objet est unique et doit être "personnalisé"

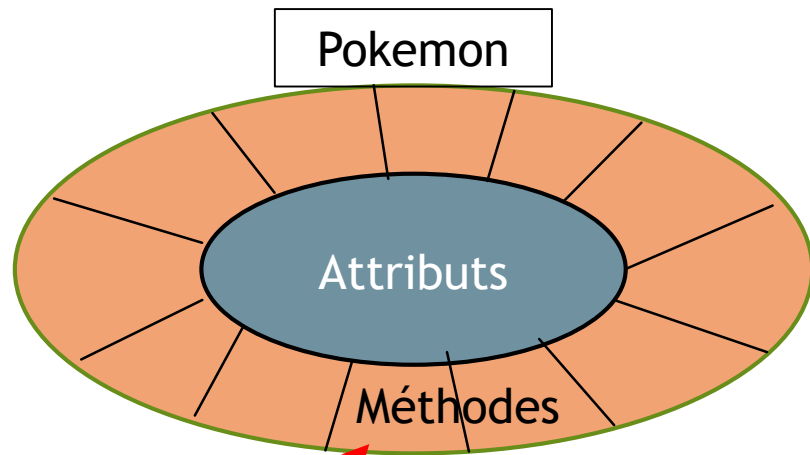


BP1 (Convention) : nom de classe en Majuscule, nom d'objet en minuscule
ex: classe Pokemon vs. un pokemon



C'est quoi une "classe" ?

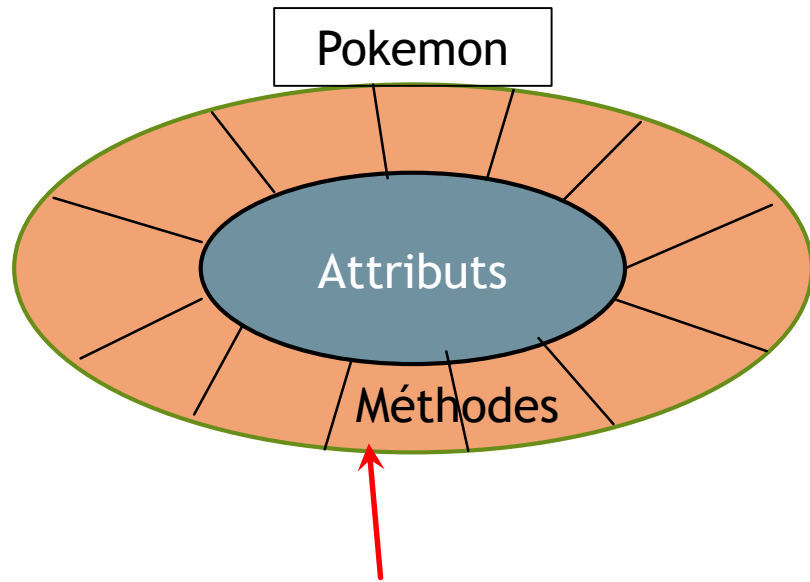
- ▶ Classe : une représentation abstraite, un modèle de qq. chose
 - ▶ Une classe a des attributs et des méthodes



Les méthodes qu'on peut appeller pour les objets de cette classe

C'est quoi une "classe" ?

- ▶ Classe : une représentation abstraite, un modèle de qq. chose
 - ▶ Une classe a des attributs et des méthodes



Les méthodes qu'on peut appeler pour les objets de cette classe
Une méthode peut modifier la valeur des attributs, afficher sur l'écran...

Attributs et méthodes

- ▶ Une méthode peut :
 - ▶ Utiliser les attributs de l'objet même
 - ▶ Changer ces attributs

Mais aussi :

- ▶ Utiliser d'autres variables
- ▶ Accéder à d'autres objets
- ▶ Etc.

2 méthodes spéciales

```
20
21 public class Pokemon {
22     // Ses attributs
23     private String pokemonName;
24     private String type;
25     private int level;
26
27     // Puis les méthodes
28
29     public Pokemon(String nom, String type, int niveau) {
30         this.pokemonName = nom;
31         this.type = type;
32         this.level = niveau;
33     }
34
35     public String getName() {
36         return this.pokemonName;
37     }
38
39     public void avancer() {
40         this.level += this.level;
41     }
42
43     public String toString() {
44         return (this.pokemonName + " est un Pokemon du type " +
45             this.type + " a le niveau " + this.level);
46     }
47 }
48
49
```



Les méthodes en Java

- ▶ Les méthodes Java sont de deux types :
 - ▶ **Procédure (void)**: une méthode sans retour
 - ▶ Ex : une méthode qui attribue des valeurs aux attributs ou qui imprime quelque chose
 - ▶ **Fonction (type)** : une méthode qui renvoie une valeur d'un certain type
 - ▶ Mot dédié : **return**
 - ▶ Ex. : une méthode qui demande la valeur d'un attribut

```
20
21 public class Pokemon {
22     // Ses attributs
23     private String pokemonName;
24     private String type;
25     private int level;
26
27     // Puis les méthodes
28
29     public Pokemon(String nom, String type, int niveau){
30         this.pokemonName = nom;
31         this.type = type;
32         this.level = niveau;
33     }
34
35     public String getName(){
36         return this.pokemonName;
37     }
38
39     public void avancer(){
40         this.level += this.level;
41     }
42
43     public String toString(){
44         return "Je m'appelle " + this.pokemonName + " est un Pokemon du type " +
45             this.type + " et mon niveau est " + this.level;
46     }
47 }
48
49
```

Fonction, retourne un String
-- notamment la valeur d'un attribut



Les méthodes en Java

- ▶ Les méthodes Java sont de deux types :
 - ▶ **Procédure (void)**: une méthode sans retour
 - ▶ Ex : une méthode qui attribue des valeurs aux attributs ou qui imprime quelque chose
 - ▶ **Fonction (type)** : une méthode qui renvoie une valeur d'un certain type
 - ▶ Mot dédié : **return**
 - ▶ Ex. : une méthode qui demande la valeur d'un attribut

```
20 L */
21 public class Pokemon {
22     // Ses attributs
23     private String pokemonName;
24     private String type;
25     private int level;
26
27     // Puis les méthodes
28
29     public Pokemon(String nom, String type, int niveau){
30         this.pokemonName = nom;
31         this.type = type;
32         this.level = niveau;
33     }
34
35     public String getName(){
36         return this.pokemonName;
37     }
38
39     public void printName(){
40         System.out.println("This pokemon's name is " + this.getName());
41     }
42 }
```

Procédure, imprime le nom du pokemon
Utilise la fonction getName



Classe et Instance

- ▶ Une classe modélise un concept (ex: Pokemon)
- ▶ Mais il peut exister différentes instances de la même classe :



name = Piplup
type = Eau
level = 5



name = Rowlet
type = Herbe/Vol
level = 10



name = Totodile
type = Eau
level = 8



Classe et Instance

- ▶ Une classe modélise un concept (ex: Pokemon)
- ▶ Mais il peut exister différentes instances de la même classe :



```
name = Piplup  
type = Eau  
level = 5
```



```
name = Piplup  
type = Eau  
level = 10
```

Sauf dans l'expression "programmation orientée objet",
"objet" = "instance d'une classe"



Plusieurs classes

Dans un programme Java...

- ▶ Un programme Java peut contenir juste une classe
- ▶ Par contre, un programme doit toujours contenir une méthode **main**
 - ▶ C'est ce qui est dans cette méthode qui s'exécute : point d'entrée

```
12
13 public class HelloWorld {
14     /**
15      * @param args the command line arguments
16      */
17
18     public static void main(String[] args) {
19         System.out.println("Hello world !");
20     }
21 }
22
23 }
24
```

Output - JavaApplication1 (run) x

```
run:
Hello world !
BUILD SUCCESSFUL (total time: 0 seconds)
```

"Phrase" standard !

args est un tableau de Strings :
il permet à l'utilisateur de paramétrer l'exécution
pour l'instant on ne l'utilise pas

Dans un programme Java...

- ▶ Un programme Java peut contenir juste une classe
- ▶ Par contre, un programme doit toujours contenir une méthode **main**
 - ▶ C'est ce qui est dans cette méthode qui s'exécute : point d'entrée

```
12
13 public class HelloWorld {
14     /**
15      * @param args the command line arguments
16      */
17
18     public static void main(String[] args) {
19         System.out.println("Hello world !");
20     }
21 }
22
23
24
```

Output - JavaApplication1 (run) ×

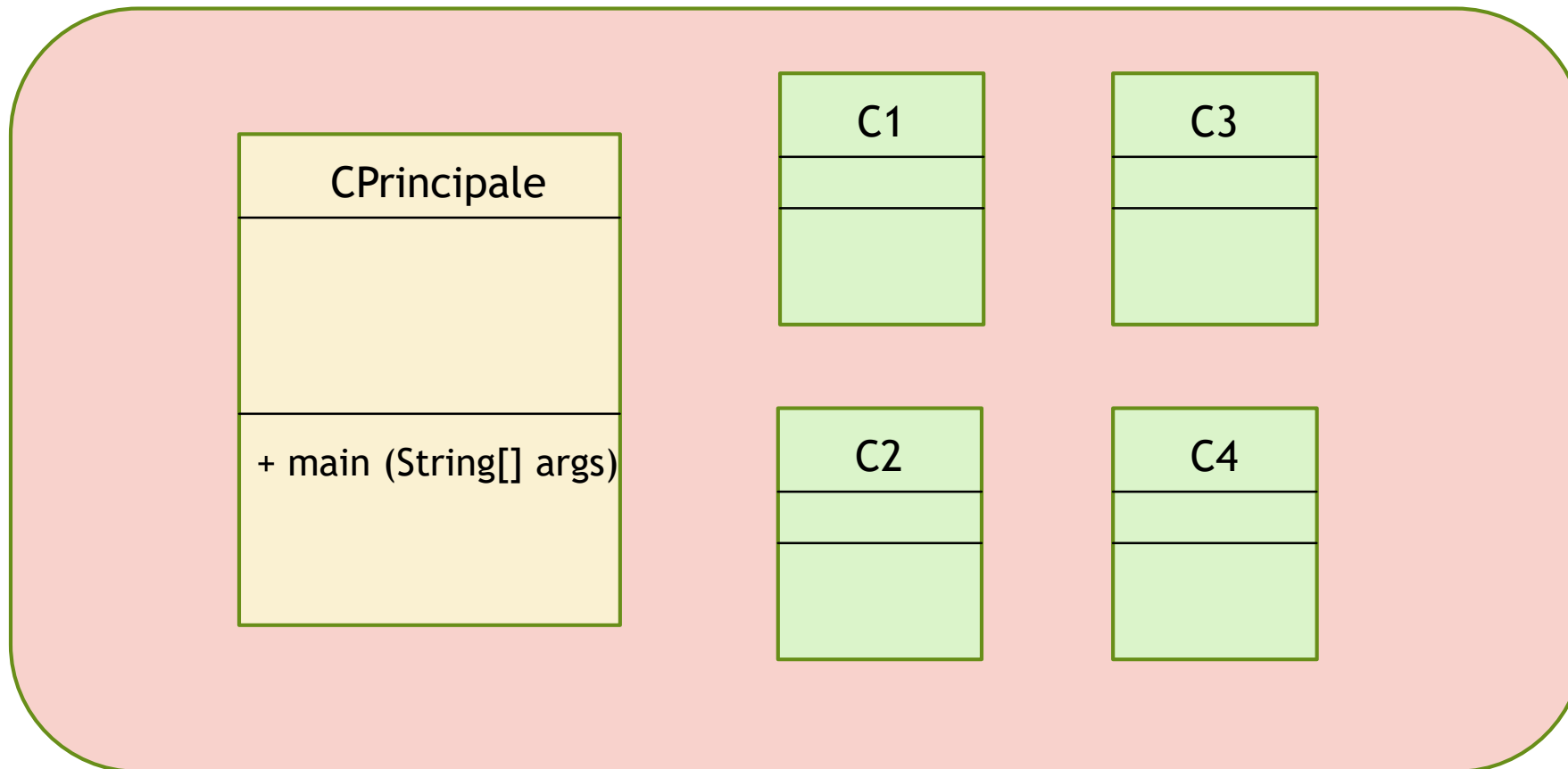
```
run:
Hello world !
BUILD SUCCESSFUL (total time: 0 seconds)
```

void = procédure, ne renvoie rien
static = cette méthode est universelle aux objets du type "HelloWorld"

Plus tard dans ce module

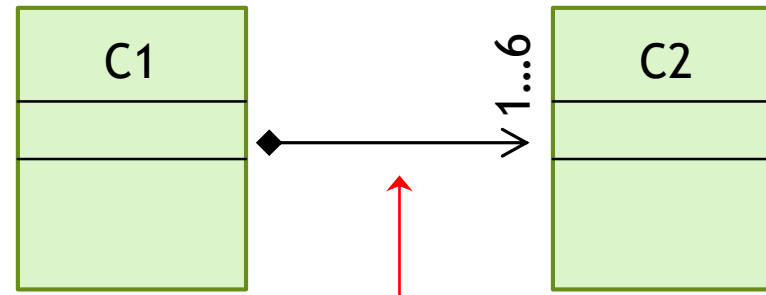
Les programmes à plusieurs classes

- ▶ En général un programme Java aura plus de classes

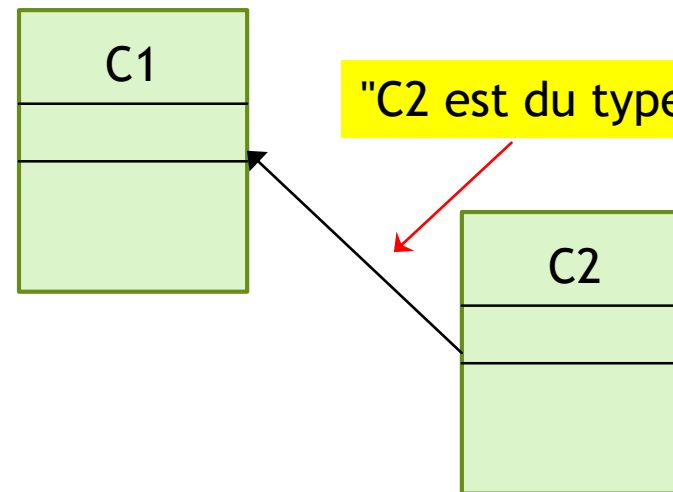


Relations entre classes

- ▶ Une classe peut utiliser d'autres classes : **Composition**
 - ▶ Classe C1 = CollecteurDePokémon;
Classe C2 = Pokémon
- ▶ Une classe peut particulariser une autre classe : **Héritage**
 - ▶ Classe C1 = Pokémon;
Classe C2 = PokémonTypeEau



"C1 a des objets C2"
"C1 a entre 1 et 6 objets C2"



"C2 est du type C1"

Plus tard dans ce module

Comment une classe peut-elle "accéder à" une autre classe ?

Private vs. Public

- ▶ Des classes, des attributs, des variables, des méthodes...
 - ▶ ... peuvent avoir une mention de "public" ou "private"
 - ▶ ... ou d'autres mentions qui seront abordées plus tard (static, final,...)
- ▶ Classe/Attribut/Méthode "public" : accessible en dehors de la classe
 - ▶ On peut créer un Pokemon dans la classe CollecteurDePokemon
 - ▶ On peut accéder au nom du Pokémon dans la classe CollecteurDePokemon



BP2 : Les classes devraient être publiques

BP3 : Par contre, les attributs sont en général privés !

- ▶ Variables définies dans des méthodes : toujours locales

Une vs. plusieurs classes

- Disons qu'on veut que ces trois Pokémon se disent Bonjour :



```
name = Piplup  
type = Eau  
level = 5
```



```
name = Rowlet  
type = Herbe/Vol  
level = 10
```



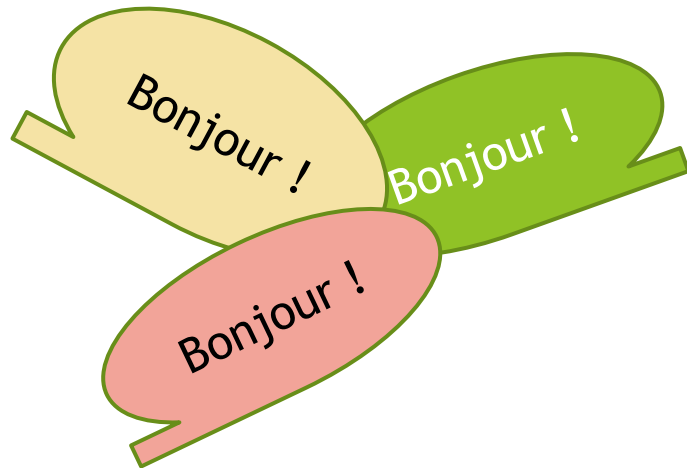
```
name = Totodile  
type = Eau  
level = 8
```

- Plusieurs programmes possibles



Une classe

- **Programme 1** : Une seule classe, avec un **main** "simulant" le dialogue :



- + Facile, rapide
- Pas de flexibilité (Et si on change les consignes ?)
- Pas de versatilité (Cela va toujours rester un dialogue...)

```
13 public class HelloWorld {
14     /**
15      * @param args the command line arguments
16      */
17
18     public static void main(String[] args) {
19         System.out.println("Piplup dit bonjour !");
20         System.out.println("Rowlet dit bonjour !");
21         System.out.println("Totodile dit bonjour !");
22     }
23
24 }
25
26
```

HelloWorld > main >

Output - JavaApplication1 (run) x

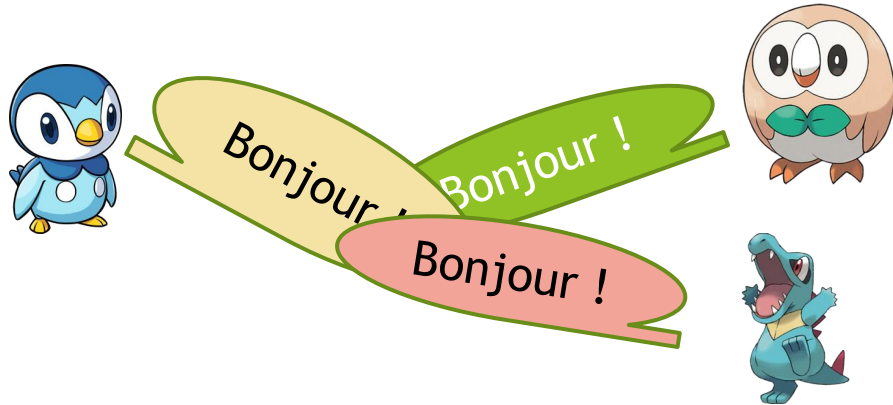
```
run:
Piplup dit bonjour !
Rowlet dit bonjour !
Totodile dit bonjour !
BUILD SUCCESSFUL (total time: 0 seconds)
```

L'exécution du programme



Deux classes

- ▶ **Programme 2** : Une classe Pokémon, une classe principale :



- ▶ Plus de **flexibilité** & de **portabilité**
 - ▶ Mais on peut encore raffiner cela

```
1
2 public class Pokemon {
3     private String nom;
4     private String type;
5     private int niveau;
6
7     public Pokemon(String pNom, String pType, int pNiveau) {
8         this.nom = pNom;
9         this.type = pType;
10        this.niveau = pNiveau;
11    }
12
13    public String toString() {
14        return (this.nom);
15    }
16 }
17
```

```
5 public class HelloWorld {
6
7     public static void main(String[] args) {
8         // créer les pokemons souhaites
9         final Pokemon p1 = new Pokemon("Piplup", "EAU", 5);
10        final Pokemon p2 = new Pokemon("Rowlet", "PLANTE", 10);
11        final Pokemon p3 = new Pokemon("Totodile", "EAU", 8);
12
13        // Les pokemons se disent bonjour
14        System.out.println(p1 + " dit bonjour !");
15        System.out.println(p2 + " dit bonjour !");
16        System.out.println(p3 + " dit bonjour !");
17    }
18 }
19 }
20
```

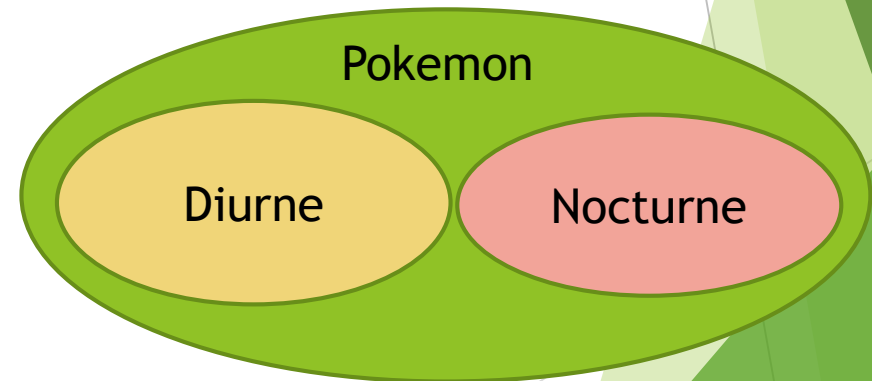
Problems Javadoc Declaration Console

```
<terminated> HelloWorld (2) [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Jan
Piplup dit bonjour !
Rowlet dit bonjour !
Totodile dit bonjour !
```



L'avantage de plusieurs classes

- ▶ Disons qu'on complique la réponse des pokemons : dans la journée :
 - ▶ Les pokemons **diurnes** disent bonjour
 - ▶ Des pokemons **nocturnes** dorment (Zzzzzzz)
- ▶ La nuit, c'est l'inverse !
- ▶ Soit : nouveau attribut diurne/nocturne
- ▶ Soit : 1 classe Pokemon, 1 classe PokemonDiurne, 1 classe PokemonNocturne



BP4 : Utiliser **plusieurs** classes
BP5 : Utiliser une **bonne structure** de classes !



Outils & preparation aux TDs & TPs

▶ TDs :

- ▶ Révisiez le(s) cours avant de vous présenter aux TDs
- ▶ Quiz de 10 à 15 minutes en début de chaque TD, sans matériel de cours
- ▶ Vous êtes toutefois encouragé(e) à utiliser les cours pendant le reste du TD
- ▶ Pas besoin d'une machine pour les TDs (sauf en cas spécial)

▶ TPs :

- ▶ Nous allons installer Java sur votre machine personnelle au premier TP : si vous avez une machine, apportez-la au premier TP
- ▶ Outils :
<http://www.objjis.com/formation-java/tutoriel-java-installation-eclipse.html> (à personnaliser selon la machine)
- ▶ À la fin de chaque TP, tout votre code sera mis à la disposition de l'enseignant (par mail)

