M2103 (POO)
Bases de la programmation orientée objet

Responsable : Cristina Onete
cristina.onete@gmail.com
https://www.onete.net/teaching.html

Avec les contributions d'Isabelle Blasquez et Marin Bertier

Les variables en Java

Les variables : mode d'emploi

- Comment utiliser les variables dans la programmation Java :
  1. Déclaration : le type et le nom de la variable sont établis

```
private String name;          private Pokemon piplup;
```

  2. Instanciation : ici on fait une première "personnalisation"

```
piplup = new Pokemon("Piplup", "Eau", 5);
```

  3. Assignement (ou initialisation) : une première valeur est assignée

```
name = "Piplup";
```

  4. Modification/ré-assignement : cette valeur peut ensuite être modifiée

```
name = "Rowlet";          piplup = new Pokemon("Rowlet", "Herbe", 10);
```

La déclaration et instanciation/assignement peuvent être réalisés en même temps

```
Pokemon piplup = new Pokemon("Piplup", "eau", 5);
```

Il y a des variables non-modifiables : ex. des constantes, mention "**final**", etc.

Les types de variables

- Des types primitives (8 en total !) :
  - s'écrivent avec une minuscule

    byte, short, int, long - entiers de 8, 16, 32, 64 bits

    float, double - nombres avec des décimales

    char - 1 caractère

    boolean - true/false

- Des types non-primitives :
  - Les **String** - chaine de caractères
  - Les **Arrays** = des tableaux
  - Tout autre objet

Déclaration d'une variable

- Trois types de variables :
  - variable locale à une méthode
  - attribut : variable déclarée dans une classe, non dans une méthode
  - attribut statique : variable déclarée dans une classe, non dans une méthode, mais qui a la même valeur pour tous les objets d'une classe

- Déclaration type primitive : **<type> <nom>;**

```
int x;   byte b;   boolean verite;
```

  - ... et on peut passer directement à l'assignement :

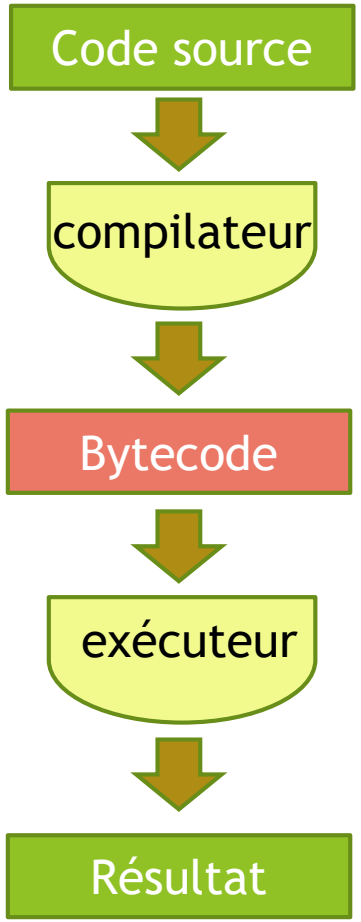
```
x = 6;
```

Attention :

L'assignement d'une variable non-déclarée cause **une erreur de compilation** !

Intermezzo : erreur de compilation

C'est quoi une erreur de compilation ?



- Erreur de compilation :
  - Le compilateur identifie des erreurs de syntaxe dans le code
    - manque de ; à la fin d'une instruction
    - variable utilisée mais non déclarée
    - ...
  - Des éditeurs comme NetBeans ou Eclipse détectent cela et mettent un avertissement

Fin intermezzo

Le type String

- Une classe pré-créée dans Java – voilà pourquoi la majuscule
- Deux façons de manipuler les String :
  - comme un type primitif :

```
String nomPokemon;
nomPokemon = "Piplup";
```

  - comme un objet plus complexe :

```
String nomPokemon;
nomPokemon = new String("Piplup");
```

BP1 : nous allons toujours utiliser la première méthode
toujours en se rappelant que les Strings ne sont pas des types primitifs

Les tableaux

- Un tableau est une collection d'objets, lui-même étant un objet
  - Un attribut principal : sa taille (# d'objets dedans) : **<nom>.length**
- Usage :
  - 1. Déclaration : **<type>[] <nom>**

```
double[] notes;    int[] serie;   Pokemon[] mesPokemons;
```

  - 2. Instanciation : obligatoire (sauf exception sur la page suivante)
    - Initialiser la taille : **<nom> = new <type>[<taille>]**

```
mesPokemons = new Pokemons[6]
```

    - Les tableaux sont indexés par position, de 0 à (<nom>.length - 1) :

Les tableaux

- Un tableau est une collection d'objets, lui-même étant un objet
- Usage :
  - 1. Déclaration : **<type>[] <nom>**
  - 2. Instanciation : obligatoire (sauf exception ci-dessous) !
    - Initialiser la taille : **<nom> = new <type>[<taille>]**
    - Les tableaux sont indexés par position, de 0 à (<nom>.length - 1) :
  - 3. Assignement : trois façons de le faire :
    - Instanciation + assignement : `double[] notes = new double[3];  notes = {12.0, 16.5, 13.0};`
    - Instanciation implicite par assignement : `double[] notes = {12.0, 16.5, 13.0};`
    - Elément par élément :

```
int[] serie = new int[50];
for (int i=0; i<50; i++) {
        serie[i] = i+1;
}
```

Instructions de base avec ces variables

; → obligatoire !

- **Assignement :**

```
String pokemonName = "Piplup";      int level = 5;
```

- **L'affichage** d'une variable de type primitive :

```
System.out.println(<nom Variable>);
```

Exception : utilisable pour les String !

- **Test d'égalité** : renvoie un boolean (attention, pas pour les String !!)

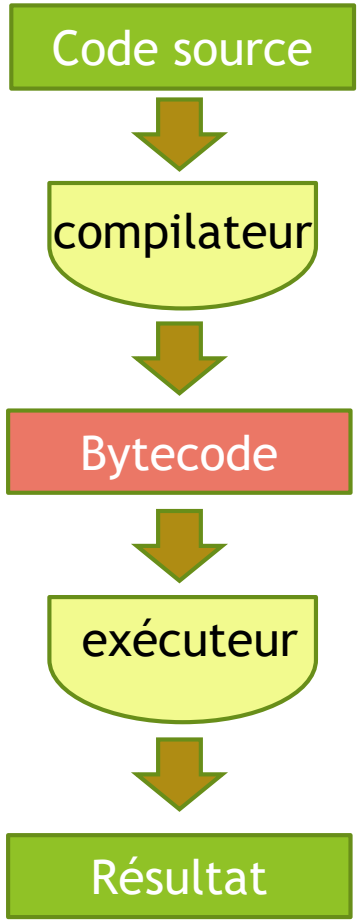
```
int a=2; int b=3; boolean equality = (a == b);
```

== : Teste d'égalité
= : assignement !

Attention : les Strings sont différents !
En String : `String a = "un string";`
`String b = "un string";`
~~`(a == b);`~~
Pour Strings : utiliser **a.equals(b)** !

Intermezzo : erreur d'exécution

Erreur d'exécution



- Erreur d'exécution
  - Aucun problème détecté au niveau de la syntaxe;
  - Par contre, l'exécution renvoie des résultats inattendus (ou aucun résultat)
    - Une boucle infinie;
    - Utiliser "==" pour les String;
    - ...
  - Ceci n'est pas toujours détecté par l'éditeur
  - Voire CM 4 sur les exceptions

Fin intermezzo

L'arithmétique avec les variables

- Le **+ et -** :
  - types numériques : addition, soustraction,
  - type boolean : non-applicable
  - type String : le + fonctionne comme concaténation

```
System.out.println("Pip" + "lup");
```
>> Piplup

Attention : char + char = exception

- Le *** et /** : seulement pour les types numériques !
  - Cas spécial : la division de deux types entiers est par défaut un entier
  - Java arrondit le résultat automatiquement : 7/2 = 3 en Java !
  - Pour avoir le résultat correct il faut "forcer" le type

```
double resultat = (double) 7/2;
```

La logique et les variables

- Les booléens utilisent des opérations logiques :
  - la négation : true → false et false → true;
    - Syntaxe : **!<nomBoolean>** ou **!(<valeur booléenne>)**
    - Du coup, **!(a == b)** est le même que **(a != b)**

```
boolean isEqual;
isEqual = !(2==3);
System.out.println(isEqual);   >> true
System.out.println(5 == 6);    >> false
```

  - Le OU logique : true/false OU true → true; false OU false → false
    - Syntaxe : **<boolean1> || <boolean2>**
    - Les booléens : des variables ou des expressions

```
boolean isEqual = (2!=3) || (5 == 6);
System.out.println(isEqual);   >> true
```

  - Le ET logique : true/false ET false → false; true ET true → true
    - Syntaxe : **<boolean1> && <boolean2>**

```
boolean isEqual = (2!=3) || (5 == 6);
System.out.println(isEqual);   >> false
```

Des opérations avec des tableaux

- Les éléments d'un tableau "empruntent" les opérations de leur type :
  - Ex. : les éléments d'un String[] utilisent des opérations typiques String
    - **.equals()** au lieu de == pour comparer deux valeurs
    - **+** pour la concatenation
    - **=** pour l'assignement (sans oublier les guillemets !)
- Il y a également des opérations sur des tableaux :
  - Mais attention : elles ne font pas ce que vous pensez !

```
double[] mesNotes = {12, 10, 15.6};
double[] tesNotes = mesNotes;
mesNotes[2] = 13;
System.out.println(tesNotes[2]);
```

J'initialise mesNotes
et je mets tesNotes = mesNotes

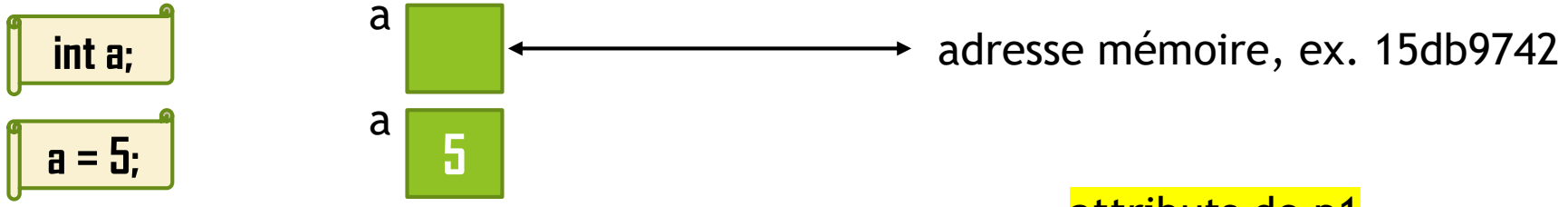
Je modifie mesNotes[2]

>> 13

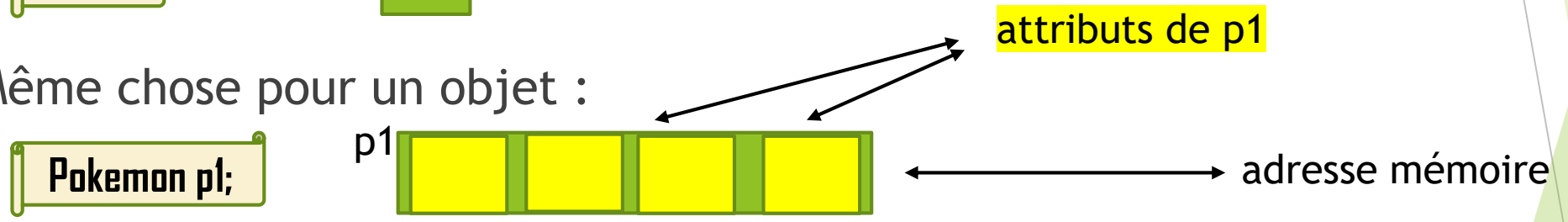
Pourquoi ??

Les attributions en mémoire

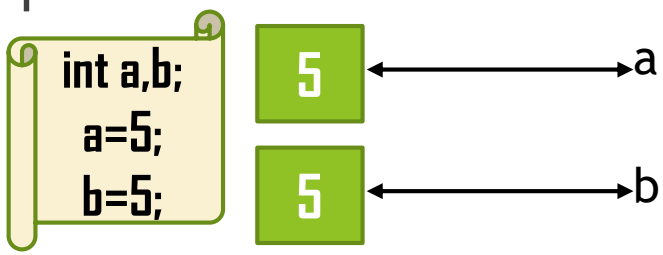
- Toute variable et tout objet a une allocation en mémoire :



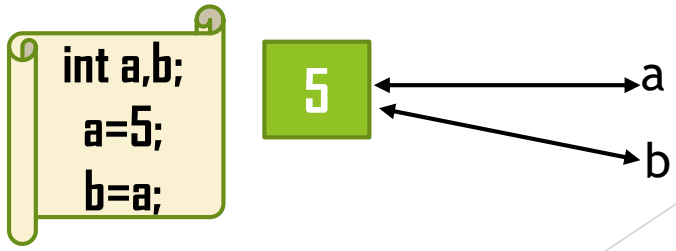
- Même chose pour un objet :



- L'opération d'attribution :



Les variables ont deux adresses différentes



Les variables ont la même adresse
Changer l'une change l'autre

System.out.println, mode d'emploi

- Afficher des résultats sur la console est très utile :
  - Pour voir le résultat final d'une exécution
  - Pour trouver des bugs dans notre code
  - Pour tester le comportement d'un code
  - etc.

- En Java, la méthode System.out.println(<nomString>) est très utile
  - ... mais il faut savoir comment bien l'utiliser

Beaucoup d'exercices sur ce sujet en TD/TP !

System.out.println, mode d'emploi

- Afficher une variable :
  - System.out.println(<var>);
  - Concatenation de String : System.out.println(<var1> + <var2>);
  - Concatenation texte + variable : System.out.println("<texte>" + <var>)

BP2 : toujours mettre un descriptif de ce qu'on affiche !

- Afficher un tableau :
  - Si on essaie System.out.println :

```
int[] monTableau = {2,3,5};
System.out.println(monTableau);
```

>> [I@6d06d69c

System.out.println(monTableau); compile, mais n'affiche pas le résultat souhaité

  - Il faut écrire une méthode qui imprime les éléments du tableau

À faire en TD !

Quelques autres opérateurs

- Comparer deux nombres :
  - == : égalité
  - != : non-égalité
  - < et <= : "inférieur" et "inférieur ou égal à"
  - > et >= : "supérieur" et "supérieur ou égal à"

Mais attention aux String !

- Incrémenter/Décrémenter une variable
  - ++ : augmenter par 1 (e.g. i++ equivalent à i = i+1)
  - -- : décrémenter par 1 (e.g. i-- equivalent à i = i-1)
  - += : i+=x équivalent à i=i+x
  - -= : i-=x équivalent à i=i-x

Des instructions plus complexes

- 1. Les conditions
  - Syntaxe :

```
if (<test>) {
    // instructions, séparées par ;
}
else {
        // instructions, séparées par ;
}
```

commentaire
(texte qui suit non-compilé)

```
If (2==3)
        System.out.println("Je reve.");
else
        System.out.println("C'est la realite.");
```

  - si on n'a qu'une instruction, on peut enlever les accolades

BP3 : toujours utiliser l'indentation (2 à 4 caractères) pour le code entre {}
BP4 : ne pas utiliser des {} si nous n'en avons pas besoin
BP5 : un code moche est un code illisible

Des instructions plus complexes

- 2. Les boucles "while" :
  - Syntaxe :

```
while (condition) {
    // des instructions, séparées par ;
}
```

  - si on n'a qu'une instruction, on peut enlever les accolades

```
// calculer la somme 1+2+...+100

public int somme(){
    int resultat = 0; // le resultat
    int i = 1; //l'iterateur
    while (i <= 100){
        resultat +=i;
        i++;
    }
    return resultat;
}
```

Des instructions plus complexes

- 3. Les boucles "for" :
  - Syntaxe :

```
for (<condition initiale>; <condition finale>; <augmentation>) {
    // instructions, séparées par ;
}
```

  - si on n'a qu'une instruction, on peut enlever les accolades (mais ce n'est pas une bonne pratique)
  - attention : déclarer l'itérateur

```
// calculer la somme 1+2+...+100

public int somme(){
    int resultat = 0; // le resultat
    for (int i = 1; i <= 100; i++){
        resultat +=i;
    }
    return resultat;
}
```

Comprendre la syntaxe Java : anticiper/identifier/corriger les erreurs !

Dans les TDs et TPs

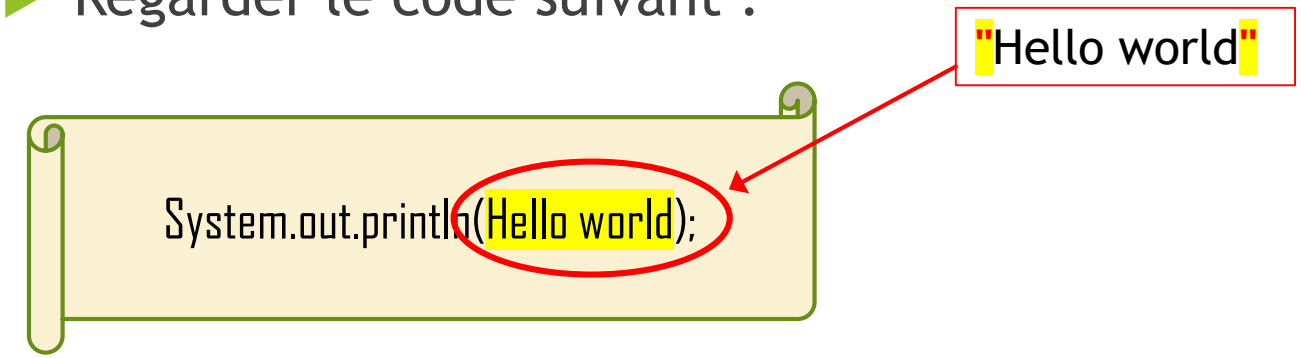
- Regarder le code suivant :

```
System.out.println(Hello world);
```

- Est-ce que ce code compile ?
- Est-ce que ce code s'exécute correctemment ?

Dans les TDs et TPs

- Regarder le code suivant :



- Est-ce que ce code compile ?
- Est-ce que ce code s'exécute correctement ?

- Non, cela ne compile pas
- ... et donc cela ne s'exécute pas correctement

Dans les TDs et TPs

- Regarder le code suivant :

```
for(int i = 0; i<10; i=i+1){
        System.out.println(i);
        i++;
}
```

- Est-ce que ce code compile ?
- Quel est le résultat à l'execution ?

- Oui, la syntaxe est correcte
- i = 0 : Output : 0; i→1 ; i→2

  i = 2 : Output : 2; i→3 ; i→4

  i = 4 : Output : 4; ...

  i = 6 : 6 ...

  i = 8 : 8 ... i → 10

Exécution s'arrête.

Paramètres, attributs

De variables aux paramètres/attributs

- Paramètres & attributs
  - Attribut : variable caractéristique d'une classe

    Ex : les objets du type Pokemon ont un nom

    Attention/Rappel : Pokemon = classe;

    pokemon = instance

  - Paramètre : variable donnée en entrée à une méthode

    Parallèle maths : f(x) -- x est un paramètre

```java
public class Pokemon {
    private String nom;
    private String type;
    private int niveau;

    public Pokemon(String pNom, String pType, int pNiveau) {
        this.nom = pNom;
        this.type = pType;
        this.niveau = pNiveau;
    }

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

    public int getNiveau() {
        return this.niveau;
    }

    public int diffDeNiveau(Pokemon poke) {
        return(poke.getNiveau()-this.getNiveau());
    }

    public String toString() {
        return (this.nom);
    }
}
```

Variables, paramètres, attributs

- Paramètres & attributs : des variables
  - Attribut : variable caractéristique d'une classe

    Ex : les objets du type Pokemon ont un nom

    Attention/Rappel : Pokemon = classe;

    pokemon = instance

  - Paramètre : variable donnée en entrée à une méthode

    Parallèle maths : f(x) -- x est un paramètre

```java
public class Pokemon {
    private String nom;
    private String type;
    private int niveau;

    public Pokemon(String pNom, String pType, int pNiveau) {
        this.nom = pNom;
        this.type = pType;
        this.niveau = pNiveau;
    }

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

    public int getNiveau() {
        return this.niveau;
    }

    public int diffDeNiveau(Pokemon poke) {
        return(poke.getNiveau()-this.getNiveau());
    }

    public String toString() {
        return (this.nom);
    }
}
```

pas de paramètre

Variables, paramètres, attributs

- Paramètres & attributs : des variables
  - Attribut : variable caractéristique d'une classe

    Ex : les objets du type Pokemon ont un nom

    Attention/Rappel : Pokemon = classe;

    pokemon = instance

  - Paramètre : variable donnée en entrée à une méthode

    Parallèle maths : f(x) -- x est un paramètre

```
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 getNom() {
14         return this.nom;
15     }
16
17     public int getNiveau() {
18         return this.niveau;
19     }
20
21     public int diffDeNiveau(Pokemon poke) {
22         return(poke.getNiveau()-this.getNiveau());
23     }
24
25     public String toString() {
26         return (this.nom);
27     }
28 }
```

avec paramètre(s)

Les méthodes en Java

Comment écrire des méthodes

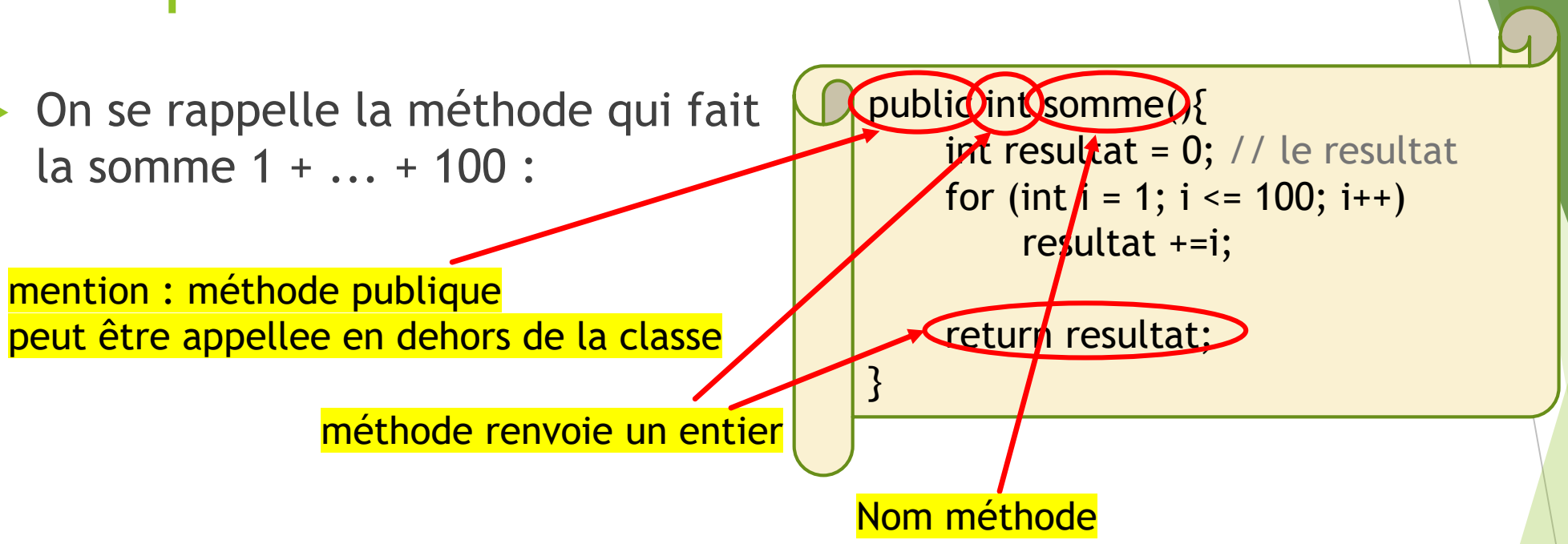
- Une méthode est caractérisée par une signature, contenant :
  - le type de la sortie (ou void sinon)
  - le nom de la méthode
  - le nom + type des paramètres en entrée

- Syntaxe :

```
<mention> <TypeSortie> <Nom> (<typeP1> <nomP1>, <typeP2> <nomP2>, ...) {
        // contenu de la méthode
        // si non-void : return <valeur du type TypeSortie>;
}
```

Exemple : La somme de n nombres

- On se rappelle la méthode qui fait la somme 1 + ... + 100 :



Exemple : La somme de n nombres

- On se rappelle la méthode qui fait la somme 1 + ... + 100 :

```
public int somme(){
        int resultat = 0; // le resultat
        for (int i = 1; i <= 100; i++)
                resultat +=i;

        return resultat;
}
```

- Et si on veut une méthode calculant : m +... + n, avec m<n ?
  - m et n passent en paramètre

```
public int somme(int m, int n){
        int resultat = 0; // le resultat
        for (int i = m; i <= n; i++)
                resultat +=i;

        return resultat;
}
```

Utiliser les méthodes

- En C (et d'autres langages non-orientés objet):
  - Une méthode existe en dehors des classes
  - Exemple : supposons une fonction EstSuperieurA(int a, int b) qui rend true si a>b et false autrement
  - Si on veut l'exécuter on appelle juste EstSuperieurA(x,y)
- En Java :
  - La méthode est dans une classe, disons NombreEntier
  - La méthode aura la signature boolean EstSuperieurA(NombreEntier b)
  - Elle s'exécute pour une instance de NombreEntier (qui joue le rôle de a)
  - Exemple : si x, y sont des instances de NombreEntier, on appelle : x.estSuperieurA(y)

Attributs & constructeurs

- D'habitude on peut nommer les méthodes comment on souhaite

BP3 : Les noms devraient toutefois être intuitifs

- Exception #1 : les constructeurs !
  - Un type spécial de méthode, qui sert à initialiser l'objet lors de sa creation
    - Dans le constructeur, on initialise les attributs de cet objet !
  - Les constructeurs sont en général des méthodes "public"
  - Ils portent obligatoirement le nom de la classe

```
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
```

classe Pokemon

son constructeur

Classe incomplète

Attributs & constructeurs

- D'habitude on peut nommer les méthodes à volonté

BP3 : Les noms devraient toutefois être intuitifs

- Exception #1 : les constructeurs !
  - Un type spécial de méthode, qui sert à initialiser l'objet lors de sa creation
    - Dans le constructeur, on initialise les attributs de cet objet !
  - Les constructeurs sont en général des méthodes "public"
  - Ils portent obligatoirement le nom de la classe

```
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
```

aucune mention retour/void
En fait, le constructeur renvoie un objet

Constructeurs : mode d'emploi

- Il n'est pas obligatoire d'écrire un constructeur à chaque classe
  - Java prevoit déjà un constructeur "par défaut" pour chaque classe
  - Toutefois, celui-ci n'initialise pas les attributs de l'objet

- Nous pouvons créer plusieurs constructeurs dans une classe
  - Impérativement le même nom
  - Impérativement des signatures différentes !

```
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 Pokemon(String pNom, String pType) {
14         this.nom = pNom;
15         this.type = pType;
16         this.niveau = 1;
17     }
```

Les tableaux dans les constructeurs

- Exemple : dans une classe Etudiant on a un attribut de type int[] qui stocke les notes de cet étudiant (jusqu'à un maximum de 20)
  - A l'initialization de chaque objet le tableau sera initialisé comme vide
  - Mais il faut également préciser une taille à ce point-là également

```
public class Etudiant{
        ...
        int[] mesCours;

        public Etudiant (){
                ...
                mesCours = new int[20];
        }
}
```

Les constructeurs abstraits

- Java.lang.Object est une classe publique déjà créée en Java
  - Elle a déjà un constructeur, dit abstrait
- Toute autre classe Java (ex. Pokemon) est construite comme un objet
  - on dit que toute autre classe "hérite" de Java.lang.Object

- Sans constructeur, toute classe utilise le constructeur d'Object
  - Toutefois, ceci ne peut jamais initialiser les attributs spécifiques à notre objet (car Java.lang.Object est beaucoup plus générale !)

- Les constructeurs ne sont pas la seule méthode définie pour Object
  - Une autre méthode très utile est la méthode **toString()** !

Les constructeurs en résumé

- Si pas de constructeur dans une classe :
  - Alors un constructeur par défaut qui n'initialise pas l'objet comme il faut, mais qui fait en sort que l'objet existe
  - Le constructeur par défaut a la signature <nom de classe>()

- Si un constructeur existe :
  - Alors le constructeur par défaut ne peut jamais être utilisé pour cette classe
  - Ce qui veut dire que chaque appel du constructeur doit suivre la signature d'un constructeur valide.

La méthode toString()

- Une méthode du type String
- En général : aucun paramètre
- Est définie implicitement (en Java.lang.Object) ou explicitement (par nous) dans toute classe

- Lorsqu'on utilise :

```
...
public static void main (String[] args){
        Pokemon p1 = new Pokemon("Piplup", "Water", 5);
        System.out.println(p1);
}
```

Lors de l'execution, Java cherche la méthode toString() de Pokemon
Si non-définie, Java execute Java.lang.Object.toString()

>> adresse mémoire de p1

Définir toString()

- Pour définir ce que veut dire "afficher" un objet, on définit la méthode toString() :

```
public class Pokemon {
    private String nom;
    private String type;
    private int niveau;

    public Pokemon(String pNom, String pType, int pNiveau) {
        this.nom = pNom;
        this.type = pType;
        this.niveau = pNiveau;
    }

    public String toString() {
        return (this.nom);
    }
```

```
public static void main (String[] args){
        Pokemon p1 = new Pokemon("Piplup", "Water", 5);
        System.out.println(p1);
}
```

>> Piplup

Classe incomplète

toString(), mode d'emploi

- Nous pouvons définir des méthodes multiples toString(...)
  - ... avec des signatures différentes

- Toutefois : seulement la méthode toString() – sans paramètres va être appelée par System.out.println !

BP4 : Le nom "toString" devrait être réservé pour programmer ce que System.out.println(<objet>) devrait afficher

Les variables et méthodes statiques

La mention "static" ou statique

- Jusqu'au present on a vu plusieures mentions :
  - private : non-utilisé en dehors de la classe
  - public : utilisable en dehors de la classe
  - finale : jamais changé après

- La mention static :
  - peut s'appliquer aux variables et aux méthodes également
  - static = universel à toutes instanciations d'une classe

Qu'est-ce que cela veut dire ?

Static vs non-static

- Prenons une classe Etudiant
- Nous voulons associer chaque étudiant avec un numéro d'ordre unique
  - On commence à 0, puis on augmente par 1 à chaque étudiant
  - Il faut donc savoir combien d'Etudiants ont été crées en total
  - Cette valeur est universelle, elle ne dépend pas de chaque étudiant
    - Au contraire des attributs non-statiques : nom, prenom, annee

```
1
2  public class Etudiant {
3      private String nom;
4      private String prenom;
5      private int annee;
6      private static int numID = 0;
7
8      public Etudiant (String nom, String prenom, String numID, int annee) {
9          this.nom = nom;
10         this.prenom = prenom;
11         this.annee = annee;
12         this.numID++;
13     }
14 }
```

```
public static void main (String[] args){
        Etudiant jeanDupont = new Etudiant("Dupont", "Jean", 2);
        Etudiant anneDesmoulins = new Etudiant("Desmoulins", "Anne", 1);
}
```

Attribut statique
Stocke le numéro actuel d'étudiant
Initié à 0, augmenté pour chaque étudiant

Les méthodes statiques

- Le même principe que les attributs
- Une méthode statique = universelle aux objets d'une classe
  - Nous avons déjà vu la méthode `main`

- Des règles pour les attributs/méthodes statiques/non-statiques :
  - Un attribut non-statique ne peut pas être appellé dans une méthode statique
  - Une méthode non-statique peut être appellee dans une méthode statique...
    - ... mais seulement si la méthode non-statique est appellée pour un objet
      - objet1.nomMethode() Oui
      - nomMethode() NON

Prochains TD/TP

- La programmation de base
- Commencer notre projet suivi de chasse aux Pokemons

- Attention/rappel : quiz au début de chaque TD

Soyez à l'heure !