Les structures de données hétérogènes

Structures de données hétérogènes

- On souhaite parfois regrouper des informations de types différents à l'intérieur d'une même variable.
- Pour cela, on doit créer un nouveau type sous la forme d'une structure composée de différents champs.
- Chaque champs possède un type qui lui est propre et est identifié par un nom.
- On peut ainsi définir des « Personne », « Voiture »..

Structures de données hétérogènes Exemple

On peut ainsi définir des « Personnes » ayant une chaine de caractères comme « nom », un réel comme « taille » et un entier comme « age ».

On pourra ensuite définir des variables de type « Personnes » qui auront toutes la même structuration correspondante aux 3 champs.

Structures de données hétérogènes
En java

On doit créer une nouvelle « class » (donc un nouveau fichier de description de class) qui contiendra juste la liste des champs, mais aucun algorithme (*donc aucune méthode ni « public static main(…) » ni autre*).

Chaque déclaration de champs est précédée du mot « **public** ».

Structures de données hétérogènes
Exemple en java

```
public class Personnes {
    public String nom;
    public double taille;
    public int age;
}
```

Structures de données hétérogènes
En java

- Pour déclarer ensuite une variable « p1 » de type « Personnes », on procède comme suit :

```
Personnes p1 = new Personnes();
```

- On peut ensuite accéder à chacun de ces champs en précisant le nom de variable **point** le nom du champs :

```
p1.nom = "Jean";
p1.taille = 1.80;
p1.age = 31;
```

Structures de données hétérogènes System.out.print

Attention comme dans le cas d'un tableau les instructions « System.out.print » (ou println) ne savent pas afficher une structure de données hétérogènes.

Les sous-programmes ou les méthodes

Notion de Sous-programme

- Sous-programme : petit programme qui s'exécute à l'intérieur d'un autre programme et qui réalise un traitement spécifique.

- Deux cas principaux justifiants leurs utilisations:
  - ✓ Répétition d'un même traitement dans le programme principal
  - ✓Organisation et lisibilité du code
- Sous-programme = méthode en Java/langage objet

Découpage prog /sous-prog. Insretion des sous-programmes

- Découpage en programme principal et sous programmes spécialisés
- On appelle **programme principal**, le programme qui appelle les autres sous programmes

```
public class Cours42 {
 //Code des sous-programmes possibles
   public static void main(String[] args) {
      //programme principal ici
      //appel(s) de sous-programmes possibles
   }
   //Code des sous-programmes possibles
}
```

Du programme au sous programme Exemple de x^n

programme

```
public class Cours3 {
   //calcul x^n
   public static void main(String[] args) {
      // variables
      int x, n, res = 1;
java.util.Scanner entree = new java.util.Scanner(System.in);
//code
       System.out.println("Nombre ?");
       x = entree.nextInt();
       System.out.println("Puissance ?");
       n = entree.nextInt();

       for (int i=0;i<n;i++){
          res = res*x;
       }
       System.out.println("Résultat "+ res);
   }
}
```

Programme complet qui va calculer x puissance n (x et n sont des entiers) à partir des saisies de l'utilisateurs.

Du programme au sous programme Exemple de x^n

programme

```
public class Cours4 {
   //calcul x^n
   public static void main(String[] args) {
      // variables
java.util.Scanner entree = new java.util.Scanner(System.in);
       int x, n, res = 1;
       //code
       System.out.println("Nombre ?");
       x = entree.nextInt();
       System.out.println("Puissance ?");
       n = entree.nextInt();

       for (int i=0;i<n;i++){
          res = res*x;
       }
       System.out.println("Résultat "+ res);
   }
}
```

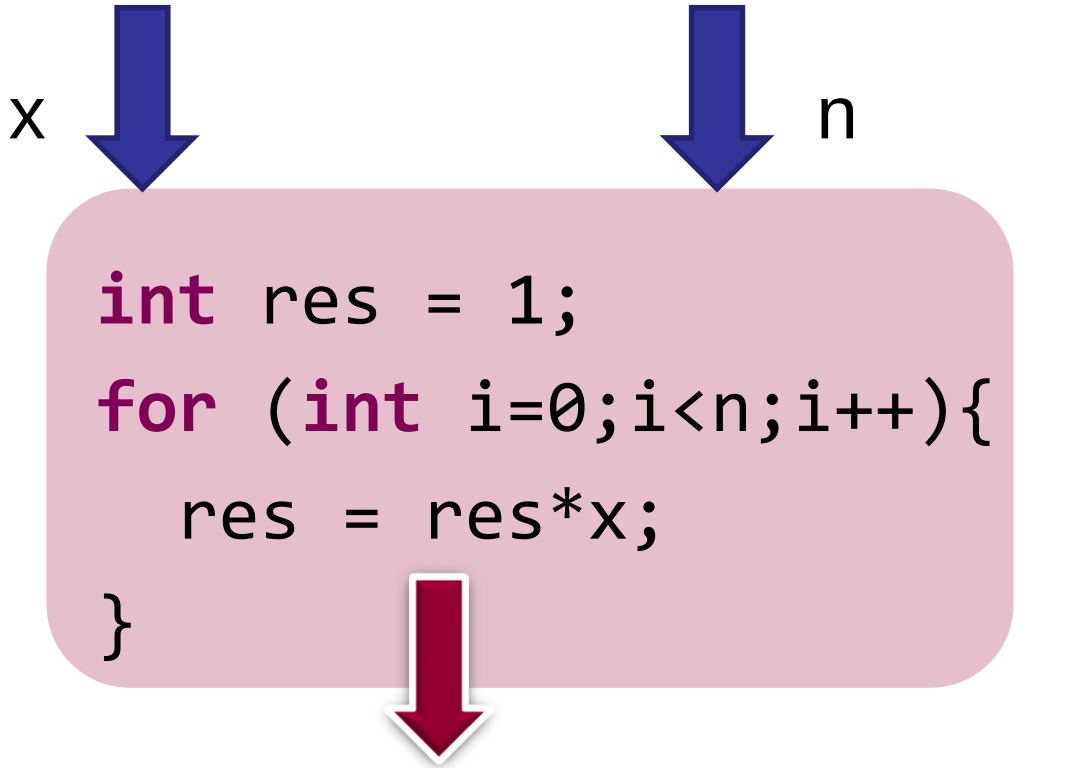
Programme complet qui va calculer **x** puissance **n** à partir des saisies de l'utilisateurs

Bout de code qu'on souhaite ré-utiliser

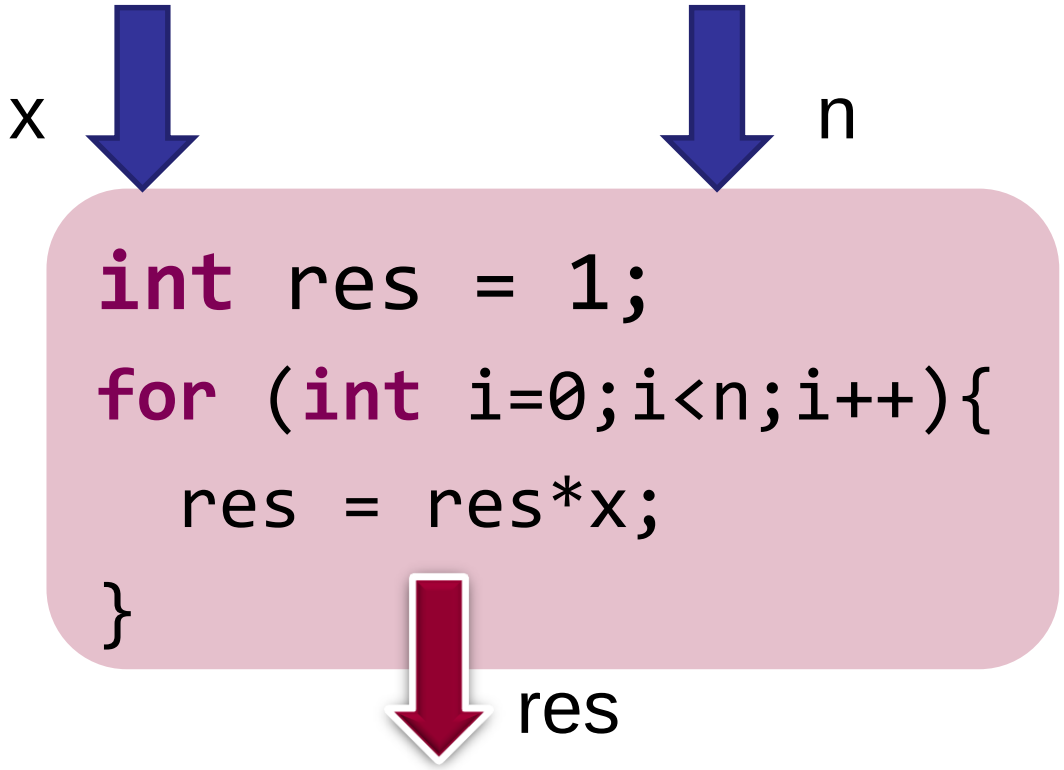
Du programme au sous programme Exemple de x^n

```
for (int i=0;i<n;i++){
   res = res*x;
}
```

Du programme au sous programme Exemple de x^n



Du programme au sous programme Exemple de x^n

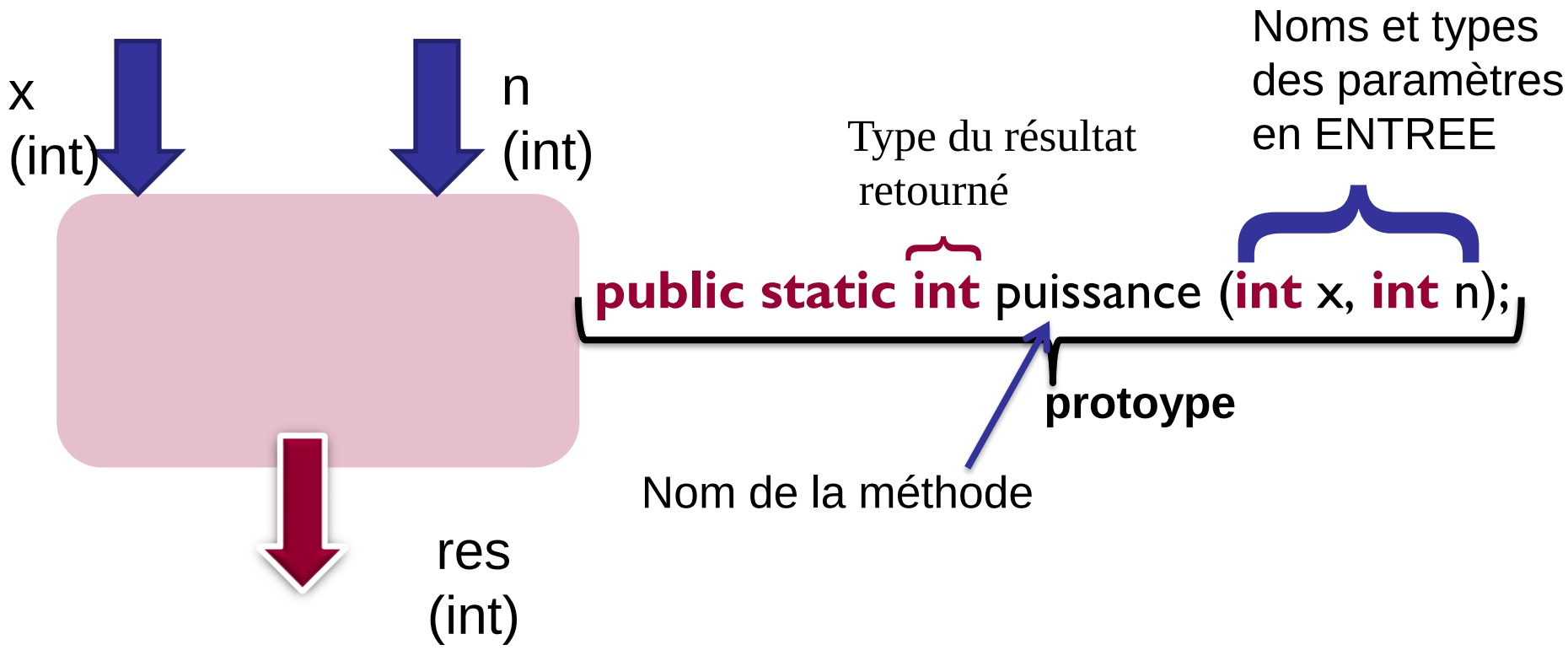


3 paramètres:

2 paramètres *en entrée* : n et x (type int)

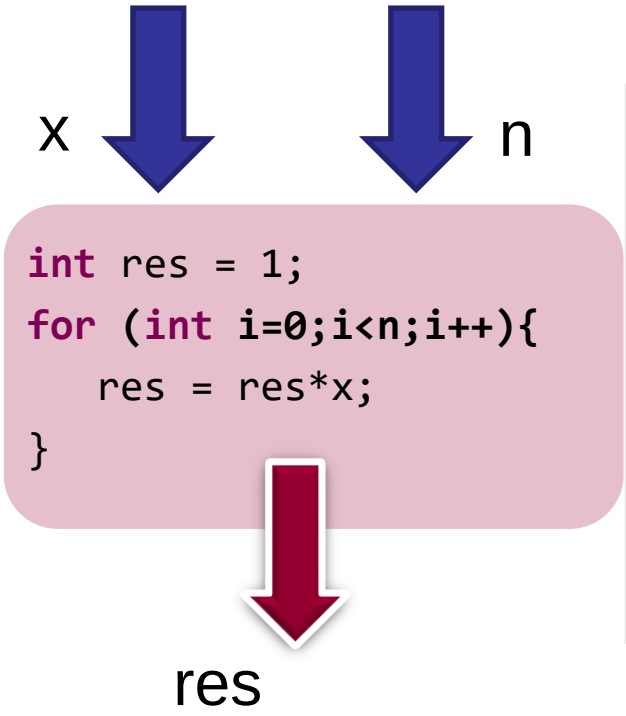
1 paramètre en *sortie*: res (type int)

Paramètres e/s et prototype d'une méthode



Prototype d'une méthode = entête de la méthode suivi du symbole de fin d'instruction.

Du prototype à la méthode




```
public static int puissance (int x, int n){
    int res = 1;
    for (int i=0;i<n;i++){
        res = res*x;
    }
    return res;
}
```

Mot clef du langage pour signifier que la méthode **retourne une copie de la valeur de res**

Méthode qui ne retourne rien

- Une méthode peut ne pas retourner une valeur
- Cette méthode provoque un effet de bord (modification de l'environnement: écriture dans un fichier…).

- Syntaxe:
  - **A la place du type de retour mettre un *void* et enlever le mot clé *return***

public static void nomMethode (types et noms des paramètres)

Méthode qui ne retourne rien
Exemple

- Exemple méthode sans retour:

```
public void helloWorld(){
      System.out.println("Hello World");
}
```

- Le mot clef ***void*** veut dire pas de type de retour, donc par extension pas de retour.

Appel d' une méthode dans le programme principal

- Une fois la méthode définie, celle-ci pour être utilisée, doit être appelée dans un programme principal.

- On distingue 2 types d'appels:
  a) Quand la méthode retourne un résultat, celui doit être stocké dans une variable:
     - double resultat = puissance(4,5);

  b) Quand la méthode ne retourne pas de résultat, pas besoin de stocker le résultat dans une variable:
     - helloWorld();

Portée de variables – Exemple

```
public class Cours4 {
   static String texte1 = " puissance ";

   public static int puissance(int x, int n){
      int variableLocale = 1;
      for (int i=0;i<n;i++) {variableLocale = variableLocale*x};
      return variableLocale;
   }

   static  String texte2 = " = ";

   //calcul x^n
   public static void main(String[] args) {
       int x1 = 2, n1 = 3;
       System.out.println(x1+texte1+n1+texte2+ puissance(x1,n1));
   }
}
```

Portée de variables –Solution

```
public class Cours4 {
   static String texte1 = " puissance ";

   public static int puissance(int x, int n){
      int variableLocale = 1;
      for (int i=0;i<n;i++) {variableLocale = variableLocale*x};
      return variableLocale;
   }

   static  String texte2 = " = ";

   //calcul x^n
   public static void main(String[] args) {
       int x1 = 2, n1 = 3;
       System.out.println(x1+texte1+n1+texte2+ puissance(x1,n1));
   }
}
```

Portée de variables

- Une variable est caractérisée par un nom, un type, mais aussi une portée.

- On distingue :
  - ✓les **variables globales :** utilisables partout dans le programme et ses différentes méthodes,
  - ✓les **variables locales:** utilisables que dans la méthode où elles ont été déclarées.

Conclusion

- Une méthode est nommée, elle a de 0 à n paramètres en entré qui sont nommés et typés.
- Une méthode ***return*** quelque chose et son résultat est utilisé
- Eclipse indique un *warning* si on fait appel à une méthode et qu'on n'utilise pas son résultat.
- Une méthode qui ne ***return*** rien ? => mettre *void et enlever le return*

Exercice:

1. Ecrivez une **méthode** qui retourne le minimum de 2 entiers passés en paramètres
2. Ecrivez une **méthode** qui réalise le minimum de 3 entiers passés en paramètres (utilisez la méthode précédente)
3. Ecrivez un **programme principale** qui permet de saisir 3 entiers et qui les affiche dans l'ordre croissant pour cela utilisez les méthodes définies en 1 et 2