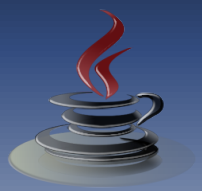
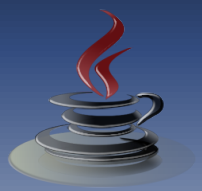
Langage et algorithmique

Rodéric Moitié

ENSTA Bretagne



Sommaire

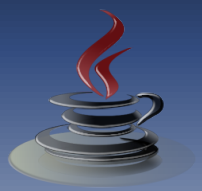


Sommaire

- Découpage du programme en sous-programmes : méthodes
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code

Exemples :

- Initialisation/affichage d'un tableau
- Fonctions de calcul

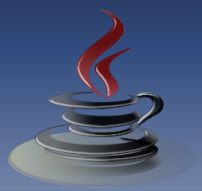


Exemple

Exemple

Fonction $y = x^2$:

```
public double carre(double x) {
      double y = x * x;
      return y;
}
```



Exemple

- Signature de la méthode

Exemple

Fonction $y = x^2$:

```
public double carre(double x) {
      double y = x * x;
      return y;
}
```

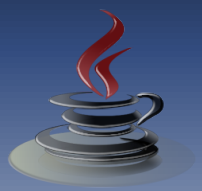
Exemple

- Signature de la méthode
- Bloc de la méthode

Exemple

Fonction $y = x^2$:

```
public double carre(double x) {
      double y = x * x;
      return y;
}
```



Exemple

- Signature de la méthode
- Bloc de la méthode
- Déclaration de variables locales, instructions

Exemple

Fonction $y = x^2$:

```
public double carre(double x) {
      double y = x * x;
      return y;
}
```

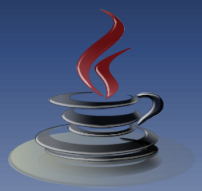
Exemple

- Signature de la méthode
- Bloc de la méthode
- Déclaration de variables locales, instructions
- Retour d'une valeur

Exemple

Fonction $y = x^2$:

```
public double carre(double x) {
      double y = x * x;
      return y;
}
```



Méthode sans valeur de retour

- Type de retour : void
- Pas de return

Exemple

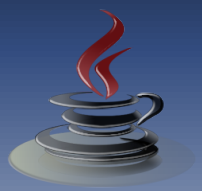
```
public void affiche(int t[]) {
     for (int i=0 ; i<t.length ; i++)
          System.out.println(t[i]);
}
```

Méthode avec plusieurs return

- Possibilité de placer le **return** dans un test

Méthode avec plusieurs return

- Possibilité de placer le **return** dans un test
- Tous les chemins d'exécution doivent conduire à un **return**

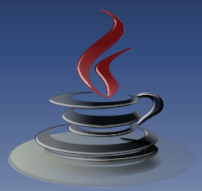


Méthode avec plusieurs return

- Possibilité de placer le **return** dans un test
- Tous les chemins d'exécution doivent conduire à un **return**

Exemple

```
public double valeurAbsolue(double x) {
     if (x<0)
          return -x;
     else
          return x;
}
```



Méthode avec plusieurs return

- Possibilité de placer le **return** dans un test
- Tous les chemins d'exécution doivent conduire à un **return**

Erreur à la compilation

```
public double valeurAbsolue(double x) {
     if (x<0)
          return -x;
     else if (x>=0)
          return x;
}
```

CalcAbs.java :16 : missing return statement

Sommaire

Appel simple

- Appel d'une méthode : indiquer son nom
- Passage de paramètres

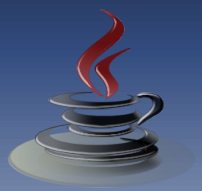
Exemple

```
public void maMethode() {
  double x = -2, x2, x3, y;
  x2 = carre(x);
  x3 = carre(5);
  y = x + x2 + valeurAbsolue(x);
  System.out.println(y);
}
```

Sommaire

Appel depuis le main

```
public int factorielle(int n) {
     int resultat = 1;
     for (int i=2 ; i<=n ; i++) {
          resultat *= i;
     }
     return resultat;
}
```



Appel depuis le main

```
public static void main(String args[]) {
      MaClasse unObjet = new MaClasse();
      int nombre = 5;
      int res = unObjet.factorielle(nombre);
}
```

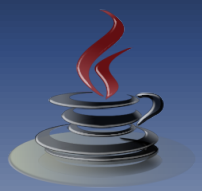
Appel depuis le main

```
public static void main(String args[]) {
      MaClasse unObjet = new MaClasse();
      int nombre = 5;
      int res = unObjet.factorielle(nombre);
}
```

Sinon

```
non-static method factorielle() cannot be
referenced from a static context
```

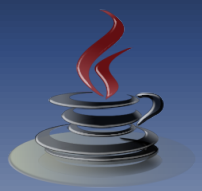
Sommaire



Paramètres formels/paramètres effectifs

Exemple (paramètre formel/paramètre effectif)

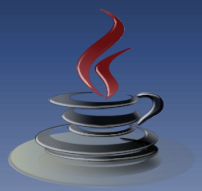
```
public void m1(int parametreFormel) {
  int i = parametreFormel + 22;
  System.out.println(i);
}
public void m2() {
  int parametreEffectif = 20;
  m1(parametreEffectif);
  m1(20);
}
```



Passage de paramètres

Deux types de passages de paramètres :

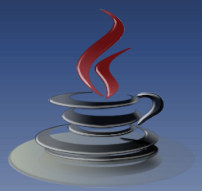
- paramètres de type simple : passage par valeur
- paramètres de type composite : passage par variable



Passage de paramètres

Exemple

```
public void uneMethode(int param) {
      param++;
      System.out.println("dans uneMethode : "+param);
}
public void autreMethode() {
      int i=0;
      System.out.println("autreMethode : "+i);
      uneMethode(i);
      System.out.println("fin de autreMethode : "+i);
}
```

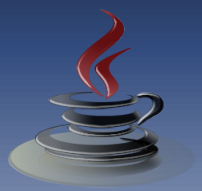


Passage de paramètres

Deux types de passages de paramètres :

- paramètres de type simple : passage par valeur
- paramètres de type composite : passage par variable
- Résultat :

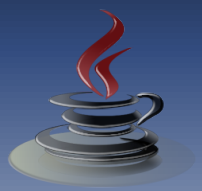
```
autreMethode : 0
dans uneMethode : 1
fin de autreMethode : 0
```



Paramètres de type composite

Exemple (Paramètres de type composite)

```
public void modifieVariable1(int tableau[]) {
     tableau[0]=42;
}
public void modifieVariable2(int tableau[]) {
     tableau = new int[10];
     tableau[0]=3;
}
```



Paramètres de type composite

Exemple (Paramètres de type composite)

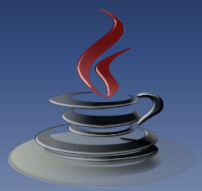
```
public void utiliseMethode() {
      int tab[] = new int[5];
      tab[0] = 1;
      System.out.println(tab.length+" "+tab[0]);
      modifieVariable1(tab);
      System.out.println(tab.length+" "+tab[0]);
      modifieVariable2(tab);
      System.out.println(tab.length+" "+tab[0]);
}
```

Paramètres de type composite

Exemple (Paramètres de type composite)

Résultat de l'exécution :

```
5  1
5  42
5  42
```



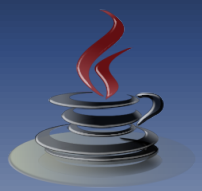
Sommaire

Portée

- Variables locales de la méthode : portée limitée à la méthode
- Paramètres : portée limitée à la méthode

Exemple (Portée)

```
public void methodeTest(int parametre) {
     int nombre = 10;
}
```



Sommaire

Pile d'appel

- Zone de mémoire

Pile d'appel

- Zone de mémoire
- Contient :

Pile d'appel

- Zone de mémoire
- Contient :
    - paramètres

Pile d'appel

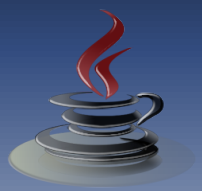
- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour

Pile d'appel

- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales

Pile d'appel

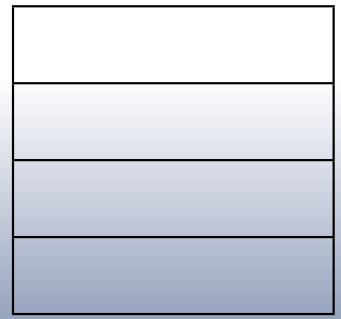
- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

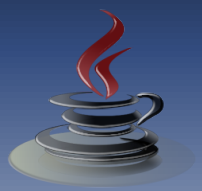


Pile d'appel

- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {
     int i;
}
methode(2, 1.5);
```

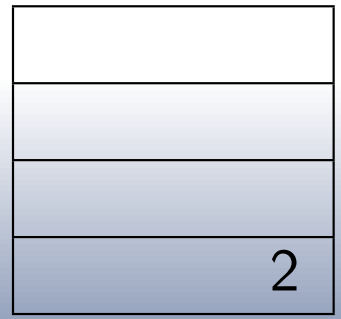


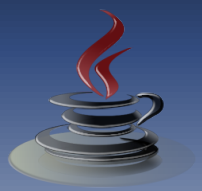


Pile d'appel

- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {
      int i;
}
methode(2, 1.5);
```



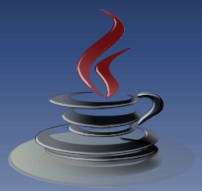


Pile d'appel

- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {
      int i;
}
methode(2, 1.5);
```

1.5
2

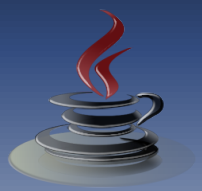


Pile d'appel

- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {
     int i;
}
methode(2, 1.5);
```

0x1E3C85
1.5
2



Pile d'appel

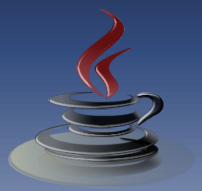
- Zone de mémoire
- Contient :
  - paramètres
  - adresse de retour
  - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {
      int i;
}
methode(2, 1.5);
```

i
0x1E3C85
1.5
2

Sommaire

Sommaire



Recherche par dichotomie

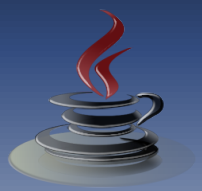
- Exemple : recherche d'un élément dans un tableau trié

1	2	3	4	5

Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane

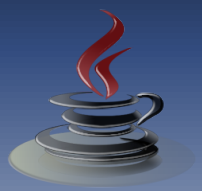
1	2	3	4	5



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

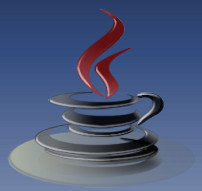
1	2	3	4	5



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5

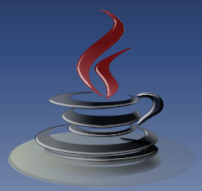


Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5

Complexité :



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5

Complexité : $\Theta(\log n)$

Sommaire

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)

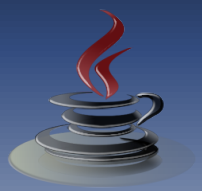
- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :
    - tri par partition/fusion

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :
  - tri par partition/fusion
  - tri rapide



Sommaire

Sommaire

Random

Classe **Random** : génération de nombres aléatoires.
Deux étapes :

Exemple

```
Random alea = new Random();
int i = alea.nextInt(100);
double x = alea.nextDouble();
```

Random

Classe **Random** : génération de nombres aléatoires.
Deux étapes :

- Créer une variable de type **Random**

Exemple

```
Random alea = new Random();
int i = alea.nextInt(100);
double x = alea.nextDouble();
```

Random

Classe **Random** : génération de nombres aléatoires.
Deux étapes :

- Créer une variable de type **Random**
- Utiliser la variable

Exemple

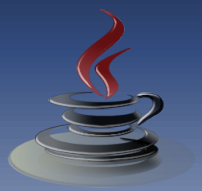
```
Random alea = new Random();
int i = alea.nextInt(100);
double x = alea.nextDouble();
```

Integer/Double/String

```
int i = Integer.MIN_VALUE;
int j = Integer.parseInt("123");
```

Integer/Double/String

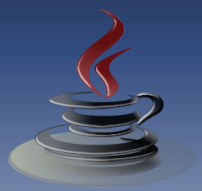
```
int i = Integer.MIN_VALUE;
int j = Integer.parseInt(args[0]);
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;
int j = Integer.parseInt(args[0]);

double x = Double.POSITIVE_INFINITY;
double y = Double.parseDouble(args[1]);
```

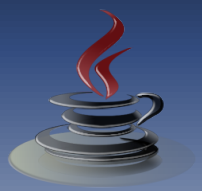


Integer/Double/String

```
int i = Integer.MIN_VALUE;
int j = Integer.parseInt(args[0]);

double x = Double.POSITIVE_INFINITY;
double y = Double.parseDouble(args[1]);

String s = "abcde";
boolean b = s.equals(args[2]);
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;
int j = Integer.parseInt(args[0]);

double x = Double.POSITIVE_INFINITY;
double y = Double.parseDouble(args[1]);

String s = "abcde";
boolean b = s.equals(args[2]);
```

Plus d'informations : consulter la doc de l'API Java.

LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)

Exemple

```
LinkedList l = new LinkedList();
```

LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément

Exemple

```
l.add(42);
```

LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément

Exemple

```
l.remove(0);
```

LinkedList

- Objectif : manipuler des collections de données

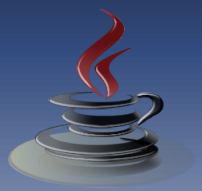
$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
if(l.contains(42)) ...
```



LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
import java.util.LinkedList;
```

LinkedList

- Objectif : manipuler des collections de données

$\simeq$ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
import java.util.*;
```

Stéréotypage des collections

Depuis Java 1.5 : possibilité de stéréotyper les collections :

- Permet de vérifier la cohérence des types
- Évite certains **cast**

Exemple

```
LinkedList<Integer> l = new LinkedList<Integer>();
l.add(42);
l.add(13.2);
int i = l.get(0);
Integer j = l.get(1);
System.out.println(l);
```

Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes

Exemple

```
double pi = Math.PI;
```

Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue

Exemple

```
double x = Math.abs(y);
```

Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue
- Fonctions trigonométriques

Exemple

```
double x = Math.cos(y * Math.PI);
```

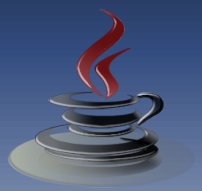
Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue
- Fonctions trigonométriques
- Fonctions d'arrondis

Exemple

```
int i = (int)Math.floor(Math.E);
```



Sommaire

API Java

- Toutes les classes de l'API sont documentées
- Apprendre à consulter la documentation
- Les paquetages les plus utiles :
  - java.lang
  - java.util

Accès depuis moodle : `https://moodle.ensieta.fr/file.php/1/docs/api/index.html`