Programmation Java

Valeur d'accueil et de reconversion en informatique (VARI1)

Daniel Porumbel (dp.cnam@gmail.com)

http://cedric.cnam.fr/~porumbed/vari1/

Plan

1. Les premiers programmes : les mots clés et la compilation
2. Fonctions avancées, lecture clavier
3. Construire ses propres classes et objets, héritage

Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```

Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```

Il faut **toujours déclarer une classe**, car Java est très orienté objets.

Cependant, on peut utiliser **que des méthodes statiques** (pas associées à des objets)

- La classe sert juste à **donner un nom au programme**, et non pas à créer des objets
- Le fichier doit avoir le même nom que la classe ⊕.java

Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```

On déclare une méthode **publique uniquement si nécessaire**, c. à. d. pour le `main` et quasiment jamais ailleurs

- si on a un seul fichier `.java` toutes les méthodes sont visibles par défaut dans ce fichier
- si une méthode est déclarée publique, elle aussi visibles dans tous les autres fichiers

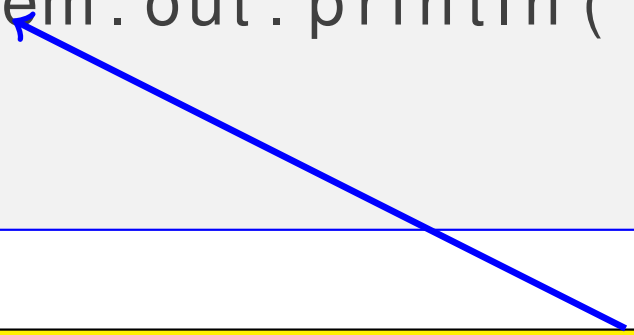
Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```



Remplacer le `println(...)` de `Processing` par `System.out.println(...)`, il n'y a pas d'autre choix!

Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```

Conclusion

Beaucoup de mots clés (langage verbeux), mais le programme n'est pas si complèxe, pas d'intéractions entre les mots clés

Le plus simple programme Java

Solution Processing

```
println("Salut tout le monde");
```

Solution Java

```
class PremierProg{
  public static void main(String[] args){
    System.out.println("Salut tout le monde");
  }
}
```

Il faut deux étapes pour lancer : compilation et exécution

compilation `javac PremierProg.java`

→ un exécutable `bytecode` pour la machine virtuelle `java`

exécution `java PremierProg`

→ lance la machine virtuelle `Java`

Une 2ème fonction : calculer l'indice de masse corporelle IMC $= \frac{\texttt{kg}}{\texttt{mètres}^2}$

```
class CalculerImc{
    static float calcImc(float kg, float m){
        return kg/(m*m);
    }
    public static void main(String[] args){
        float poidsKg  = 90;
        float tailleCM = 2;                  // cm
        float tailleM  = tailleCM/100;       // m
        float imc = calcImc(poidsKg,tailleM);
        System.out.println(imc);
    }
}
```

- Remplacer chaque `float` par un `double` : commande sed
  - un `double` et un `float` 2 fois plus grand (en nb de bits)

Une 3ème fonction : la puissance

```
class CalculerPuissance{
    //renvoyer x^n
    static double puissance(double x, int n){
        double p = x;
        for(int i=0;i<n;i++)//est-ce que
            p = p * x;      //c'est bien
        return p;           //correct?:
    }
    public static void main(String[] args){
        double puissance5  = puissance(2,5);
        System.out.println(puissance5);
    }
}
```

! Remarquez qu'on met `static` devant chaque fonction !

Plan

1. Les premiers programmes : les mots clés et la compilation
2. Fonctions avancées, lecture clavier
3. Construire ses propres classes et objets, héritage

Valeurs minimales et maximales d'un tableau

1. Écrire une fonction `Java` pour déterminer la valeur minimale d'un tableau
2. Faire appel à la fonction ci-dessus pour calculer la valeur maximale :
   - inverser tous les nombres
   - calculer min
   - inverser tous les nombres
3. Écrire une fonction qui calcule la valeur minimale dans une matrice (tableau de tableaux)

Valeurs minimales et maximales d'un tableau

1. Écrire une fonction `Java` pour déterminer la valeur minimale d'un tableau
2. Faire appel à la fonction ci-dessus pour calculer la valeur maximale :
   - inverser tous les nombres
   - calculer min
   - inverser tous les nombres
3. Écrire une fonction qui calcule la valeur minimale dans une matrice (tableau de tableaux)

Valeurs minimales et maximales d'un tableau

1. Écrire une fonction `Java` pour déterminer la valeur minimale d'un tableau
2. Faire appel à la fonction ci-dessus pour calculer la valeur maximale :
   - inverser tous les nombres
   - calculer min
   - inverser tous les nombres
3. Écrire une fonction qui calcule la valeur minimale dans une matrice (tableau de tableaux)

Les variables globales : déclaration `static`

Et si on voulait calculer la valeur `minimale` et `maximale` d'un tableau :

- il ne suffit pas un seul `return`

$\Longrightarrow$ on met le résultat dans des variables globales `min` et `max`

- il suffit de les déclarer `static` au début du code

```
class TabMinEtMax{
    static int min;
    static int max;
    static void calculerMinMax(int[] t){...
...
}
```

- Il est aussi possible de créer une classe `MinMax` avec deux attributs `min` et `max` et faire `calculerMinMax(...)` renvoyer un objet de type `MinMax`

La racine carré

1. Utiliser `Math.sqrt(x)`, c. à. d. la fonction statique `sqrt` de la classe `Math`
2. Écrire notre propre fonction `racine(double x)`
   - Utiliser la suite convergente suivante :

$$r_{n+1} = \frac{r_n + \frac{x}{r_n}}{2}$$

3. Calculer $\sqrt[4]{81}$, implémenter une fonction pour la racine d'ordre 4 !

La racine carré

1. Utiliser `Math.sqrt(x)`, c. à. d. la fonction statique `sqrt` de la classe `Math`

2. Écrire notre propre fonction `racine(double x)`
   - Utiliser la suite convergente suivante :

$$r_{n+1} = \frac{r_n + \frac{x}{r_n}}{2}$$

3. Calculer $\sqrt[4]{81}$, implémenter une fonction pour la racine d'ordre 4 !

La racine carré

1. Utiliser `Math.sqrt(x)`, c. à. d. la fonction statique `sqrt` de la classe `Math`

2. Écrire notre propre fonction `racine(double x)`
   - Utiliser la suite convergente suivante :

$$r_{n+1} = \frac{r_n + \frac{x}{r_n}}{2}$$

3. Calculer $\sqrt[4]{81}$, implémenter une fonction pour la racine d'ordre 4 !

Demander à l'utilisateur de saisir *x*

Il faut utiliser un objet de classe `Scanner`
Ce code permet de lire un entier :

```
java.util.Scanner s ;
s = new java.util.Scanner(System.in);
int x = s.nextInt();
```

On peut lire :

un double `s.nextDouble()`

un mot `s.next()`

une ligne `s.nextLine()`

Tester le scanner

```
class TestScanner{
    public static void main(String[] args){
        java.util.Scanner s ;
        s = new java.util.Scanner(System.in);
        int x = s.nextInt();
        System.out.println(x);
        System.out.println(s.nextDouble());
        System.out.println(s.next());
        System.out.println(s.nextLine());
    }
}
```

Note : `System.out` et `System.in` font référence à l'entrée et la sortie par rapport au terminal

Passer des arguments au programme **dans la console**

Rappel La fonction `main(String[] args)` reçoit un tableau d'objets `String` comme argument

```
public static void main(String[] args){
...
}
```

Ce tableau de `String` représente **les arguments passés à la commande `java`**

- `java PROGNAME unArgument`

Écrire un programme qui fait la somme des entiers passés comme arguments

`Integer.parseInt(...)` fait la conversion `String`→`int`

Passer des arguments au programme **dans la console**

Rappel La fonction `main(String[] args)` reçoit un tableau d'objets `String` comme argument

```
public static void main(String[] args){
...
}
```

Ce tableau de `String` représente **les arguments passés à la commande `java`**

- `java PROGNAME unArgument`

Écrire un programme qui fait la somme des entiers passés comme arguments

`Integer.parseInt(...)` fait la conversion `String`→`int`

Plan

1. Les premiers programmes : les mots clés et la compilation
2. Fonctions avancées, lecture clavier
3. Construire ses propres classes et objets, héritage

Rappel notions théoriques de classe et objet

Une classe est un modèle pour construire des objets

- sauf si tous les attributs/méthodes sont statiques : dans ce cas là, on ne construit jamais d'objet de cette classe

Un objet

- possède un état constitué de valeurs (attributs)
- possède des actions (méthodes) qui peuvent agir sur ce cet état pour le modifier
  - les méthodes définissent le comportement d'un objet

Un objet est une instance (un exemplaire) d'une classe

La classe `Frac` du cours précédent

1. Faire fonctionner la classe sous `Java` et tester
   - Il faut uniquement rendre `toString()` publique, le reste est identique !
2. Améliorer la méthode `simplifier()`
   - Il y a au moins deux solutions
3. Faire `toString()` afficher `num` et non pas `num/den` lorsque `den=1`

La classe `Frac` du cours précédent

1. Faire fonctionner la classe sous `Java` et tester
   - Il faut uniquement rendre `toString()` publique, le reste est identique !
2. Améliorer la méthode `simplifier()`
   - Il y a au moins deux solutions
3. Faire `toString()` afficher `num` et non pas `num/den` lorsque `den=1`

La classe `Frac` du cours précédent

1. Faire fonctionner la classe sous `Java` et tester
   - Il faut uniquement rendre `toString()` publique, le reste est identique !
2. Améliorer la méthode `simplifier()`
   - Il y a au moins deux solutions
3. Faire `toString()` afficher `num` et non pas `num/den` lorsque `den=1`

L'héritage : classe de base

→ Quel est le résultat du code ci-dessous ?

```
class Automobile{
    float vitesseMaxAutoroute(){
        return 130;
    }
    float tempsTrajet (float distance) {
        return distance/vitesseMaxAutoroute();
    }
};
//Et dans le main:
Automobile maVoiture = new Automobile();
float temps = maVoiture.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture:="+ temps+ "heures.");
```

L'héritage : classe dérivée

→ Quel est le résultat du code ci-dessous ?

```
class Bus{
    float vitesseMaxAutoroute(){
        return 90;
    }
}
//Et dans le main:
Bus monBus = new Bus();
float temps = monBus.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture:="+ temps+ "heures.");
```

- Comment peut-on appeler la méthode `tempsTrajet` ?

L'héritage : classe dérivée

→ Quel est le résultat du code ci-dessous ?

```
class Bus extends Automobile{
    float vitesseMaxAutoroute(){
        return 90;
    }
}
//Et dans le main:
Bus monBus = new Bus();
float temps = monBus.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture :="+ temps+ "heures.");
```

- Comment peut-on appeler la méthode `tempsTrajet` ?

  Réponse : Par héritage → la classe `Bus` hérite toutes les méthodes et tous les `attributs` de la classe `Automobile`

Héritage classes Java

- Toutes les classes héritent `Object` par défaut
  - On trouve dans la classe `Object` la méthode `toString()` qui renvoie l'adresse mémoire de l'objet
- On va étudier des classes graphiques. Exemples :
  - `JButton` et `jCheckBox` héritent `AbstractButton`
  - La doc d'une classe standard est disponible si on cherche le nom de la classe sur Internet

Bonnes Pratiques de Programmation Orienté Objet

- 1ère lettre d'un nom de classe : Majuscule
- 1ère lettre d'un nom de méthode/variable/paquetage : minuscule
- Toute accolade fermante occupe une ligne !

Héritage classes Java

- Toutes les classes héritent `Object` par défaut
  - On trouve dans la classe `Object` la méthode `toString()` qui renvoie l'adresse mémoire de l'objet
- On va étudier des classes graphiques. Exemples :
  - `JButton` et `jCheckBox` héritent `AbstractButton`
  - La doc d'une classe standard est disponible si on cherche le nom de la classe sur Internet

Bonnes Pratiques de Programmation Orienté Objet

- 1ère lettre d'un nom de classe : Majuscule
- 1ère lettre d'un nom de méthode/variable/paquetage : minuscule
- Toute accolade fermante occupe une ligne !

Autre Classe : quel est le résultat du code ?

```
class Compte{
    int solde;
    Compte(){           //constructeur
        solde = 0;      //sans arguments
    }
    void ajouter(int montant){
        solde = solde + montant;
    }
}
Classe Exec{
    public static void main(String[] args){
        Compte c = new Compte();
        c.ajouter(10);
        println(c.solde);
    }
}
```

Contineur la classe `Compte` pour :

1. Ajouter un nom de titulaire à la classe `Compte`
   - Ajouter une deuxième constructeur qui reçoit comme argument le nom du titulaire
2. Accorder par défaut un crédit de 10 euros
3. Pouvoir retirer de l'argent
4. Pouvoir verser tout l'argent d'un compte `c1` dans un compte `c2`

Contineur la classe `Compte` pour :

1. Ajouter un nom de titulaire à la classe `Compte`
   - Ajouter une deuxième constructeur qui reçoit comme argument le nom du titulaire
2. Accorder par défaut un crédit de 10 euros
3. Pouvoir retirer de l'argent
4. Pouvoir verser tout l'argent d'un compte `c1` dans un compte `c2`

Contineur la classe `Compte` pour :

1. Ajouter un nom de titulaire à la classe `Compte`
   - Ajouter une deuxième constructeur qui reçoit comme argument le nom du titulaire
2. Accorder par défaut un crédit de 10 euros
3. Pouvoir retirer de l'argent
4. Pouvoir verser tout l'argent d'un compte `c1` dans un compte `c2`

Contineur la classe `Compte` pour :

1. Ajouter un nom de titulaire à la classe `Compte`
   - Ajouter une deuxième constructeur qui reçoit comme argument le nom du titulaire
2. Accorder par défaut un crédit de 10 euros
3. Pouvoir retirer de l'argent
4. Pouvoir verser tout l'argent d'un compte `c1` dans un compte `c2`