Informatique 1
L1 Portail IE
L1 Portail MA

Responsables:
pierre-alain.fouque@univ-rennes1.fr
Patrick.derbez@univ-rennes1.fr

Plusieurs slides de F. Morain et M. Christie

Décomposition binaire

```
while (n != 0) {
    System.out.println(n%2);
    n /= 2;
}
```

Pour $n = 13$:

1
0
1
1

Do While

Le programme précédent ne marche que pour $n > 0$. Pour le faire marcher pour $n \geqslant 0$, on le modifie en utilisant l'instruction:

```
do
   I
while(E);
```

qui exécute l'instruction `I` tant que `E` est vraie:

```
do{
   System.out.println(n%2);
   n = n/2;
} while(n != 0);
```

Rem. On passe toujours au moins une fois dans la boucle.

Boucle **for**

Deux formes équivalentes:

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

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

Boucle **for**

Syntaxe:

```
for(<var> = <init>; <cond>; <incr>)
  I
```

où:

<var> est la variable de boucle, qui contrôle les calculs;

<init> est la valeur initiale;

<cond> est le test de continuation, qui est effectué avant de rentrer dans le corps de la boucle;

<incr> est l'incrément;

I est le (bloc d') instruction(s) à exécuter.

Boucle **for**

Autre exemple:

```
int i;
for(i = 0; i <= 10; i += 2)
  System.out.println(i);
```

Forme encore plus compacte avec une variable fraîche:

```
for(int i = 0; i <= 10; i += 2)
  System.out.println(i);
```

Différence: la variable `i` est locale à la boucle: elle n'est pas connue en dehors de cette boucle.

Rem. on utilise d'habitude `for` quand on peut prévoir à l'avance le nombre de passages dans la boucle, `while` quand on ne peut pas.

Imbrication des boucles

Imbrication des boucles: on peut empiler les boucles.

Ex. On veut afficher les lignes suivantes:

```
1
1 2
1 2 3
```

```
for(int i = 1; i <= 3; i++){
  for(int j = 1; j <= i; j++){
    System.out.print(" ");
    System.out.print(j);
  }
  System.out.println();
}
```

Nombre d'itérations

```
public class ex_while {
    public static void main(String[] args){
        int n_iter = 2147483647; // 2^31 - 1
        for (int i = 0; i <= n_iter; ++i) {
            System.out.print("+");
        }
    }
}
```

Condition " i <= n_iter " toujours satisfaite !!

Boucles infinies

```
for (;;)
```

```
while (true)
```

- Boucles infinies
- Sortie en utilisant les mots clés **break** ou **return**

- **Ne pas faire ça en INF1!**

Le problème de Syracuse

$$u_0 = m > 1, \quad u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

```
public class syracuse {
    public static void main(String[] args){
        int n = 154463; // n >= 1;
        while (n > 1) {
            if (n%2 == 0) n /= 2;
            else n = 3*n + 1;
        }
    }
}
```

Conjecture

```
6 -> 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

```
129 -> 388 -> 194 -> 97 -> 292 -> 146 -> 73 -> 220 -> 110 -> 55 -> 166
-> 83 -> 250 -> 125 -> 376 -> 188 -> 94 -> 47 -> 142 -> 71 -> 214 -> 107
-> 322 -> 161 -> 484 -> 242 -> 121 -> 364 -> 182 -> 91 -> 274 -> 137
-> 412 -> 206 -> 103 -> 310 -> 155 -> 466 -> 233 -> 700 -> 350 -> 175
-> 526 -> 263 -> 790 -> 395 -> 1186 -> 593 -> 1780 -> 890 -> 445 -> 1336
-> 668 -> 334 -> 167 -> 502 -> 251 -> 754 -> 377 -> 1132 -> 566 -> 283
-> 850 -> 425 -> 1276 -> 638 -> 319 -> 958 -> 479 -> 1438 -> 719 -> 2158
-> 1079 -> 3238 -> 1619 -> 4858 -> 2429 -> 7288 -> 3644 -> 1822 -> 911
-> 2734 -> 1367 -> 4102 -> 2051 -> 6154 -> 3077 -> 9232 -> 4616 -> 2308
-> 1154 -> 577 -> 1732 -> 866 -> 433 -> 1300 -> 650 -> 325 -> 976 -> 488
-> 244 -> 122 -> 61 -> 184 -> 92 -> 46 -> 23 -> 70 -> 35 -> 106 -> 53
-> 160 -> 80 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

Conjecture: pour tout n, le programme termine.

Exercice

- On veut calculer le 10° terme de la suite:

$$x_{n+1} = 3x_n + 2, x_0 = 1$$

Comment faire?

Premier essai

```
public class Suite {
    public static void main(String[] Args) {
        int x0 = 1;
        int x1 = 3*x0 + 2;
        System.out.println("x1 = " + x1);
    }
}
```

Problème: on ne veut pas utiliser 10 variables différentes!

Solution

```
public class Suite {
    public static void main(String[] Args) {
        int x = 1;
        for (int i = 1; i <= 10; ++i) {
            x = 3*x + 2;
        }
        System.out.println(x);
    }
}
```

- Pour calculer le (n+1)° terme on a seulement besoin de connaître le n° terme.

Fibonacci

- Calculer le 10° terme de la suite de Fibonacci:

$$F_0 = 0, F_1 = 1 \; ; \forall n \geqslant 2, F_n = F_{n-1} + F_{n-2}$$

- Pour calculer un terme de la suite on a besoin des deux précédents.

Fibonacci

```
public class Suite {
    public static void main(String[] Args) {
        int x_2 = 0; // terme n-2
        int x_1 = 1; // terme n-1
        for (int i = 1; i <= 10; ++i) {
            int x = x_1 + x_2; // calcul du terme n
            x_2 = x_1; // le terme n-1 devient le terme n-2
            x_1 = x; // le terme n devient le terme n-1
        }
        System.out.println(x_1);
    }
}
```

Introduction aux fonctions

- Tout langage de programmation impérative offre un moyen de découper un programme en unités fonctionnelles qui peuvent partager des données communes et s'appeler les unes les autres.

- Les données communes à ces unités fonctionnelles sont appelées variables globales.

- Toute unité fonctionnelle a pour vocation d'effectuer certaines opérations à partir de données passées en paramètres (ou arguments) en vue de produire un résultat.

Introduction aux fonctions

- Par convention, l'exécution d'un programme commence par l'exécution du point d'entrée (fonction main).
- C'est ce point d'entrée qui fera appel à d'éventuelles autres unités fonctionnelles.
- Unité fonctionnelle : appelée fonction
- Chacune de ces fonctions peut également faire appel à d'autres fonctions, ou s'appeler elle-même (dans ce dernier cas, on dit que la fonction est récursive).

Lire un entier

```java
import java.util.Scanner;
public class readInt {
    public static int lireEntier() {
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt(); // on lit l'entier et on le stocke
        sc.close();
        return x; // on renvoie l'entier lut
    }

    public static void main(String[] Args) {
        System.out.print("Entrez un entier: ");
        int a = lireEntier();
        System.out.println("Vous avez entré l'entier " + a);
    }
}
```

Syntaxe

```
public static <type_de_retour> <nom>(<paramètres>)
{
   [déclaration de variables]
   [suite d'instructions]
   return <résultat>;
}
```

- **`type_de_retour`**: une fonction calcule un résultat et le retourne à la fonction qui l'a appelée; ce résultat a nécessairement un type (**`int`**, **`double`**, etc.).
- Paramètres: les arguments passés à la fonction et avec lesquels elle peut travailler.

Exemples

```
public static int carre(int n){
  return n * n;
}

public static double milieu(double x, double y){
  return 0.5 * (x+y);
}

public static int fact(int n){
  int f = 1;

  for(int i = 1; i <= n; i++)
    f *= i;
  return f;
}
```

Le type **void**

Si la fonction ne retourne pas de résultat (affichage par ex.), elle a un type de retour spécial **`void`**.

C'est le cas de la fonction **`main`**:

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

Rem. Dans ce cas uniquement, le **`return`** est optionnel, mais il permet de bien voir où finit l'exécution du programme.

Surcharge, signature

```
public static <type_de_retour> <nom>(<paramètres>)
```

est la signature de la fonction **<nom>**. En Java:

```
public static int f(int n){...}
```

et

```
public static int f(int n, int m){...}
```

sont deux fonctions différentes, qu'on peut utiliser simultanément:

```
int a = f(1);
int b = f(2, 3);
```

On parle de surcharge de la fonction **f**.

Utilité: gestion plus facile des variantes de fonction.

La surcharge doit être manipulée avec précaution.

Visibilité des variables

Rappel: dans l'écriture

```
for(int i = 0; i <= 10; i += 2)
  System.out.println(i);
```

la variable **i** ne peut pas être utilisée après le bloc formé par la boucle **for**. La portée de la variable **i** est limitée au bloc.

```
public static int f(int r){
  int n = 3;
  return (r+n);
}
public static void main(String[] args){
  int n = 10, m;
  m = f(n);
  System.out.println(n);
}
```

Qu'est-ce qui est affiché? 10.
Règle: la variable **n** de **f** est locale à **f**. Elle n'a rien à voir avec la variable **n** de **main** (ce sont deux emplacements mémoire différents).

Visibilité des variables

Les paramètres formels ont une portée dite « locale »:

- on ne peut accéder à, ou modifier, leur valeur qu' à l' intérieur de la fonction dans laquelle les paramètres sont déclarés.

```
static int plusgrand(int x, int y) {
    if (x > y) {
        return x;
    } else {
        return y;
    }
}

public static void main(String[] args) {
    x = 12;                    // erreur: la variable x n'est pas déclarée!
    int y = plusgrand(3, 2);// ici, la variable y est différente
                               // de celle déclarée dans
                               // la fonction plusgrand(int x, int y)
}
```

Exemple

```
public static int plusgrand(int x, int y) { // x et y sont des paramètres formels
    int resultat = x;    // Leurs valeurs sont utilisées
    if (y > x) resultat = y;  // dans la fonction
    return resultat;
}

public static void main(String[] Args) {
    int x = plusgrand(3,4); // assignation des valeurs 3 et 4 à x et y
    int y = plusgrand(7,x); //  assignation de 7 et de la valeur de x(=4) à x et y
    //  seul l'ordre de passage des paramètres effectifs est important
    //  (quel que soit leur nom)

    int z = plusgrand(plusgrand(y, 7), plusgrand(x-2*y, 9)); // assignation du
    // résultat des deux expressions à x et y
}
```

Exemple

Autre exemple:

```
public static void f(int n){
   n = n+1;
   return;
}
public static void main(String[] args){
   int n = 10;
   f(n);
   System.out.println(n);
}
```

Qu'affiche-t-il en sortie ? 10.

Règle: une fonction ne peut pas changer la valeur d'une variable d'un type primitif extérieure à elle-même. On dit que les arguments sont passés par valeur (c'est-à-dire recopie).

Structure d'un programme Java

```
public class NomProgramme {
  // déclaration de variables globales reconnues par toutes les fonctions
static type fonction1 (type arg1,…, type argn) {
     // arg1, … , argn sont ici des paramètres “formels”.
     // Leurs valeurs sont utilisées dans le corps de la fonction
     Suite d’instructions ;
     Retourne valeur
}
static void fonction2 (type arg1,…, type argn) {
     Suite d’instructions ;
     // ne retourne pas de valeur (void)
}
....... // autres fonctions

public static void main(String[] args) {
     // déclarations de variables
     appel aux fonctions fonction1, …, fonctionN
     // exemple : int x = fonction1(5)
}
```

Quand décide-t-on d'écrire une fonction?

- Quand on veut réutiliser le même morceau de code;
- quand on a identifié un morceau de code ayant une certaine cohérence;
- si le code dépasse une page.

Quelques conseils de style:

- Convention de nommage (en Java): les noms de fonctions commencent par une minuscule.
- Choisir des noms de fonction adaptés.