THCS 4: LẬP TRÌNH CƠ BẢN VỚI JAVA

MẢNG

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

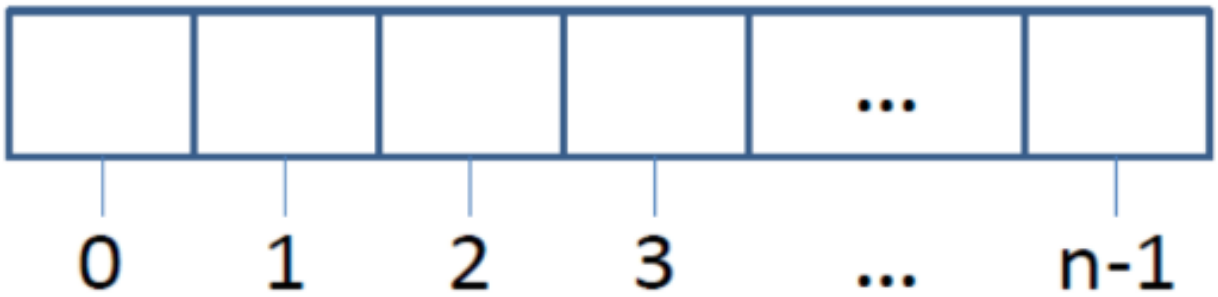
Ngày 28 tháng 10 năm 2024

Dữ liệu kiểu mảng

Dữ liệu kiểu mảng

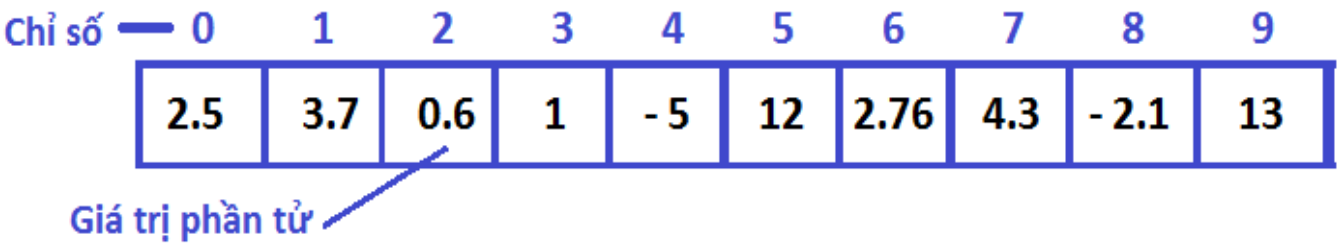
Dữ liệu kiểu mảng

- Một dãy các phần tử có cùng kiểu
  - Kiểu của các phần tử mảng có thể là kiểu bất kỳ, ví dụ các kiểu nguyên thủy char, byte, short, int, long, float, double ... hoặc các kiểu tham chiếu đối tượng như String...
  - Các phần tử mảng luôn được đánh chỉ số lần lượt từ 0.
    Ví dụ mảng có n phần tử (n nguyên dương)



Dữ liệu kiểu mảng

- Ví dụ: Mảng số thực có 10 phần tử



- Biến mảng (khai báo qua tên biến) tham chiếu đến vùng nhớ chứa mảng (một dạng tham chiếu đến đối tượng - Mảng là một đối tượng).
- Vùng nhớ chứa mảng chỉ được cấp phát sau khi biến mảng được khởi tạo.
- Mỗi phần tử của một mảng cũng có thể có kiểu mảng. Lúc đó phần tử mảng tham chiếu đến một đối tượng mảng khác.

Mảng một chiều

- Cú pháp khai báo một biến mảng (một chiều):

  TYPE[ ] NAME; hoặc

  TYPE NAME[ ];

  Ví dụ:

```
double[] arrVector;
int arrIndexes[];
```

- Truy cập đến phần tử mảng:

  NAME[INDEX]

- Chỉ số INDEX bắt đầu từ 0. Với mảng có n phần tử thì chỉ số cuối cùng là $n-1$.
- Ví dụ:

```
arrVector[1]; arrIndexes[0];
```

Khai báo, khởi tạo biến mảng

- Mảng cần được khởi tạo trước khi sử dụng. Có hai cách khởi tạo mảng:
  1. Dùng lệnh new: Với kích thước size nguyên dương

     TYPE[ ] NAME = new TYPE[size]; hoặc

     TYPE NAME[ ] = new TYPE[size];

     Ví dụ:
     ```
     double arrVector[] = new double[5];
     int[] arrIndexes = new int[10];
     ```
  2. Khởi gán giá trị cho các phần tử mảng (kích thước *size*):

     TYPE[ ] NAME = {Value$_0$, Value$_1$, ..., Value$_{size-1}$};

     - **Value$_0$, Value$_1$, ..., Value$_{size-1}$** có cùng kiểu TYPE.
     - Sau lệnh khởi gán, mảng **NAME** sẽ được xác định có size phần tử (bằng số phần tử được gán).
     - Tập hợp các giá trị gán cần đặt giữa { và }, các giá trị cách nhau bởi dấu phẩy.
- Ví dụ: Các lệnh sau đều tạo mảng thực (double) 5 phần tử:

```
double vector1[] = new double[5];
double[] vector2 = {0.5, 4.0, 3.25, 10, 2};
```

Khai báo, khởi tạo biến mảng

- Ví dụ khai báo - khởi tạo và truy cập mảng một chiều

```
int array[] = new int[5];// array of 5 int elements
array[0] = 1;// Correct!
array[4] = 5;// Correct!
array[5] = 5;// WRONG! Compiled but Run-time Exception
```

(Dịch được nhưng khi chạy sẽ báo lỗi: Index 5 out of bounds for length 5).

- Ví dụ khởi tạo sai kiểu:

```
// Define by initializing an array of 5 elements
int array[] = {5, 6.5, 4, -2, 3};
// WRONG! Compiling error: Incompatible types.
```

Báo lỗi ngay khi dịch chương trình.

Khai báo, khởi tạo mảng

- Mảng n chiều có mỗi phần tử mảng tham chiếu đến một mảng $n-1$ chiều khác.
- Khai báo mảng nhiều chiều: Mở rộng theo quy tắc cú pháp cho mảng một chiều.

  TYPE[ ][ ] NAME; hoặc **TYPE NAME[ ][ ];**// 2D - array

  TYPE[ ][ ][ ] NAME; hoặc **TYPE NAME[ ][ ][ ];**// 3D - array

- Khởi tạo mảng nhiều chiều: Mở rộng quy tắc khởi tạo mảng một chiều bằng new hoặc khởi gán giá trị.

Khai báo, khởi tạo mảng

- Ví dụ khởi tạo mảng 2 chiều.

```
int[][] array1 = new int[2][3];
// array1: int 2D-Array of sizes 2x3
```

```
int array2[][] = {{1,2, 3}, {4, 5}};
// array2[0]: 1D - Array of sizes 3;
// array2[1]: 1D - Array of sizes 2;
```

```
int[][] array3 = new int[2][];
array3[0] = new int[3]; //1D - array of sizes 3;
array3[1] = new int[2]; //1D - array of sizes 2;
```

Kích thước mảng - thuộc tính *length*

- Biến mảng có sẵn thuộc tính *length* cho biết số phần tử mảng (kích thước).
- Ví dụ:

```
int[] array1 = new int[10];
int   size1 = array1.length; // size1 = 10

int array2[] = {5, 4, 3, 2, 1};
int   size2 = array2.length; // size2 = 5
```

```
double array3[][] = {{1.0,2, 3}, {4, 5.0}};

int   size3_0 = array3[0].length; // size3_0 = 3
int   size3_1 = array3[1].length; // size3_1 = 2
```

Duyệt phần tử mảng

Có 2 cách duyệt phần tử mảng

- Dùng vòng lặp và thuộc tính *length*. Ví dụ:
  - Vòng lặp for

```
int[] array1 = {3, 4, 5, 7, 9, -10};

for(int i = 0; i < array1.length; i++)
    System.out.println("array1[" + i + "]=" +
                                    array1[i]);
```

  - Vòng lặp while và do ... while ???
- Dùng vòng lặp cho mảng for(type element:array)

```
// Khai báo mảng ARRAY_NAME kiểu TYPE
TYPE[ ] ARRAY_NAME;
//... Khởi tạo mảng ARRAY_NAME...
for(TYPE element: ARRAY_NAME)
        STATEMENTS // lệnh trong vòng FOR
```

Duyệt phần tử mảng

- Vòng lặp for(type element: array) cho phép duyệt **giá trị** các phần tử mảng array thông qua biến element.
- Vòng lặp for(type element: array) còn được gọi là for-each.
- Ví dụ:

```
int  array1[] = {5, 4, 3, 2, 1};

for(int i : array1)
    System.out.print(i + " ");// 5 4 3 2 1
    // print out value of elements of Array1
```

```
String[] array2 = {"One", "Two", "Three"};

for(String s : array2)
    System.out.print(s + " ");// One Two Three
```

Duyệt phần tử mảng

- Ví dụ dùng for-each cho mảng nhiều chiều:

```
//Array2 - 2D array of integers
int array3[][] = {{2, 3, 4}, {2, 3}};

for (int[] arr : array3)
    System.out.println(arr.length + " ");// 3 2
// each element of array3 is an array
// which is considered by arr
// arr.length returns sizes of elements of array3
```

Mảng làm tham đối của phương thức

- Ví dụ: Đối của hàm main()

```
public class Arguments{
    public static void main(String args[])
    {
        int size = args.length;
        System.out.println(size);
        for(int i=0; i<size; i++)
            System.out.println(args[i]);
    }
}
```

- Khai báo:

  [public|private] [static] TYPE NAME(ARG_TYPE Argument[ ])

- Khi gọi phương thức, tại vị trí tham đối phải là biến mảng tương ứng về kiểu.

Mảng làm đối của phương thức

- Giá trị **phần tử mảng** bị thay đổi trong thân hàm/phương thức sẽ được cập nhật sau lệnh gọi.
- Ví dụ:

```
public static void modifyArray(int array[]){
    for(int i = 0; i < array.length ; i++)
        array[i] *= 2;
}
```

Giá trị phần tử của array bị thay đổi sau lệnh gọi.

- So sánh với đối là biến đơn kiểu nguyên thủy:

```
public static void modifyElem(int element){
    element *= 2;
}
```

Trường hợp này, ra khỏi phương thức, element mất hiệu lực.

VD: Mảng làm đối của phương thức(1)

Ví dụ: Sử dụng hai phương thức trên để minh họa sự thay đổi giá trị khi tham đối là kiểu mảng và tham đối là kiểu nguyên thủy.

```
public class EditArray {
    // method with argument is an array
    public static void modifyArray(int array[])
    {
      for(int i=0; i < array.length; i++)
            array[i] *= 2;
      //for(int number:array) number *= 2;
    }

    // method with argument is in primitive type
    public static void modifyElem(int elem)
    {
        elem *=2;
    }
// be continued next page...
```

VD: Mảng làm đối của phương thức(2)

```
// next ....
    public static void main(String[] arguments) {
      int[] array = new int[arguments.length];

      for(int i=0; i < array.length; i++) {
         array[i] = Integer.parseInt(arguments[i]);
      }
      System.out.println("Array's elements:");
      for(int i=0; i < array.length; i++) {
         System.out.print("\t" + array[i]);
      }

      for(int i=0; i < array.length; i++) {
         modifyElem(array[i]);
      }
// be continued next page...
```

VD: Mảng làm đối của phương thức(3)

```
// next ....
        System.out.println("\n Array after modifyElem:");
        for(int i=0; i < array.length; i++) {
            System.out.print("\t" + array[i]);
        }

        modifyArray(array);
        System.out.println("\n Array after modifyArray:");
        for(int i=0; i < array.length; i++) {
            System.out.print("\t" + array[i]);
        }

        int sum = 0;
        for(int number:array) sum += number;
        System.out.println("\n Sum * 2 is " + sum);
    }
}
```

Mảng làm giá trị trả về của phương thức (hàm)

- Mẫu cú pháp phương thức trả lại một mảng

```
[public|private] [static] TYPE[ ] NAME(ARG_TYPE Arguments)
{
        TYPE[ ] ARRAY_RETURN;
        // STATEMENTS Here ...
        return ARRAY_RETURN;
}
```

Mảng làm giá trị trả về của phương thức (hàm)

Ví dụ:

```
public static int[] sumVector(int vector1[], int vector2[])
{
    int sum[];
    if (vector1.length == vector2.length)
    {
        sum = new int[vector1.length];
        for(int i = 0; i < sum.length ; i++)
            sum[i] = vector1[i] + vector2[i];
        return sum;
    }
    return null;
}
```

Luyện tập

Viết chương trình ListOfNumber.java bao gồm các phương thức sau:

1. public static void inputList(int[] elems): nhập các phần tử cho mảng elems.
2. public static void printList(int[] elems): in các phần tử cho mảng elems.
3. public static int indexOf(int[] elems, int value): trả về chỉ số của phần tử đầu tiên trong elems có giá trị bằng value, nếu không có thì trả về giá trị -1.
4. public static void replace(int[] elems, int oldValue, int newValue): thay thế các phần tử trong mảng elems có giá trị oldValue bằng giá trị newValue.
5. public static int[] add(int[] elems, int idx, int value): tạo mảng mới newElems là bản sao của elems, nhưng có bổ sung thêm phần tử có giá trị value vào vị trí có chỉ số idx.
6. public static int[] remove(int[] elems, int value): tạo mảng mới newElems là bản sao của elems, nhưng đã xóa bớt đi một phần tử đầu tiên có giá trị value.

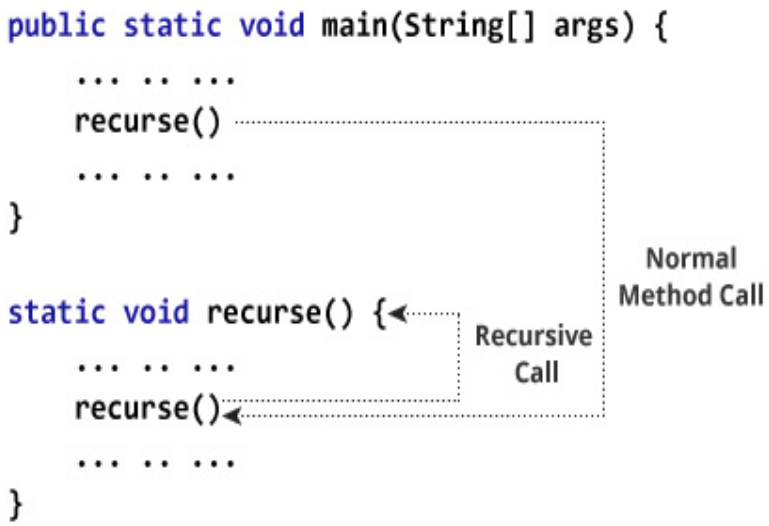
Đệ quy

Nội dung

- Khái niệm đệ quy
- So sánh lặp và đệ quy

Khái niệm Đệ quy

- Cơ chế cho phép một hàm/phương thức gọi đến chính nó.



Ví dụ chương trình đệ quy

- Với $n \in \mathbb{N}$, $n \geq 1$, tính giai thừa

$$n! = n \times (n-1) \ldots \times 2 \times 1.$$

- Sử dụng vòng lặp ???
- Đệ quy

```
public static long fact(int n)
{
    if (n > 1)
        return n * fact(n-1);
    else
        return 1;
}
```

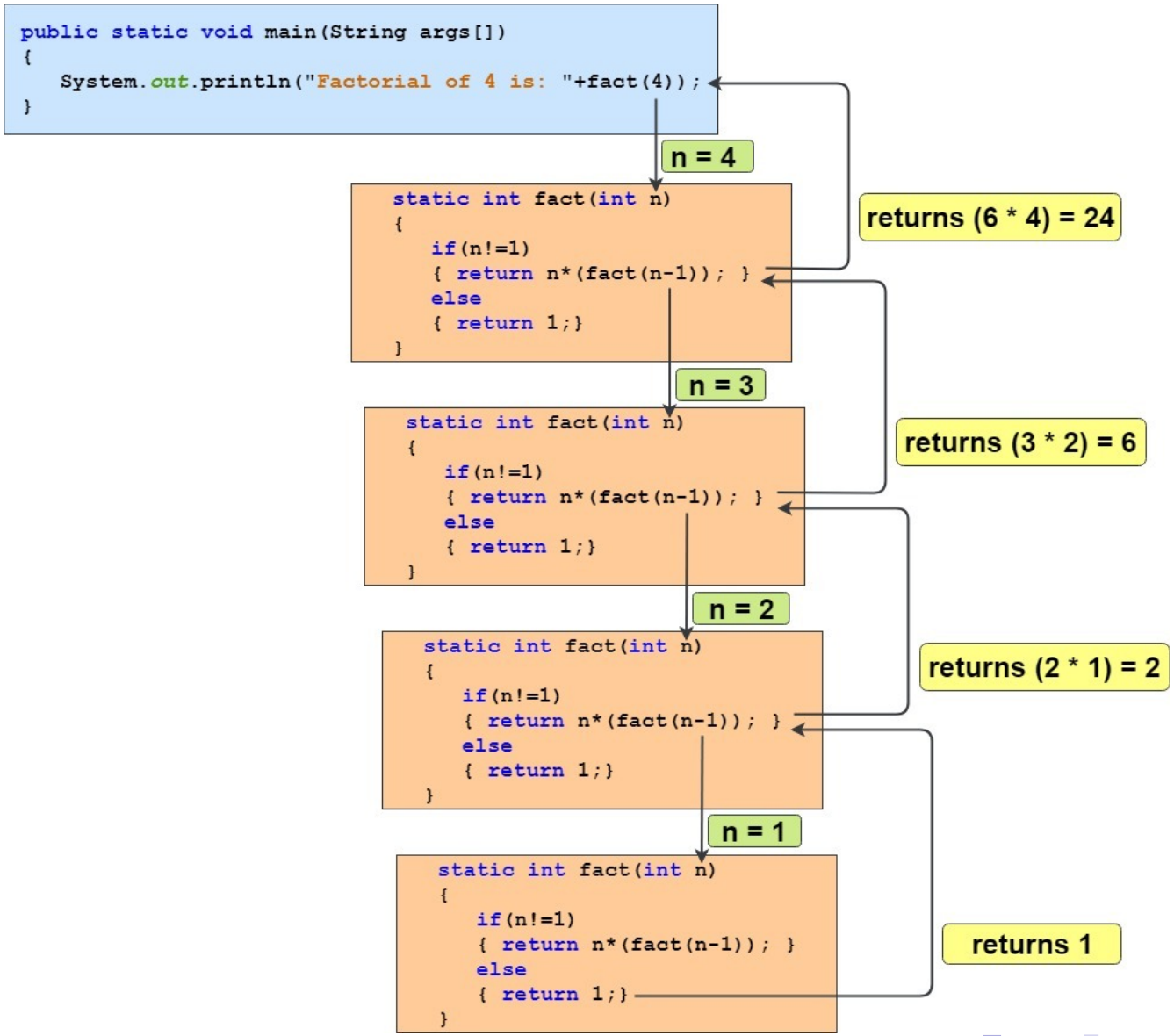
Ví dụ chương trình đệ quy

- Truy ngược vết của chương trình đệ quy

```
fact(5)
    fact(4)
        fact(3)
            fact(2)
                fact(1)
                return 1
            return 2*1 = 1
        return 3*2 = 6
    return 4*6 = 24
return 5*24 = 120
```

.

Ví dụ chương trình đệ quy



Luyện tập 1: Dãy Fibonacci

- Định nghĩa dãy Fibonacci

$$F_0 = 0; \quad F_1 = 1; \quad F_n = F_{n-1} + F_{n-2} \quad \forall n > 1.$$

- Sử dụng vòng lặp (tự thực hiện!)
- Đệ quy

```
public static long Fibonacci(int n)
{
    if (n <= 1)
        return n;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Nhận xét

- Gần như mọi vấn đề giải quyết được bằng đệ quy thì có thể giải quyết được bằng cấu trúc lặp.
- Một số trường hợp, đệ quy cho phép chương trình dễ hình dung - dễ hiểu, dễ thiết lập hơn.
- Vòng lặp hiệu quả thực hiện tốt hơn!

Luyện tập 2: Biểu diễn nhị phân của số tự nhiên

- Với $n \in \mathbb{N}$, hãy in ra xâu ký tự biểu diễn nhị phân của n.
- Phân tích:
  - Nếu $n = 0$ hoặc $n = 1$: Biểu diễn là chữ số n.
  - Nếu $n > 1$: Là xâu biểu diễn của $n/2$ (chia nguyên) ghép với chữ số $n\%2$ (lấy dư). Ví dụ

$$\begin{aligned} 35_{10} = 100011_2 &= "10001" + "1" = [17]_2 + "1" \\ &= [35/2]_2 + [35\%2]_2 \end{aligned}$$

  - Sử dụng String.valueOf(n) để đổi n thành xâu ký tự tương ứng.
- Sử dụng vòng lặp (tự thực hiện!)
- Đệ quy?

Luyện tập 2: Biểu diễn nhị phân của số tự nhiên

```
public static String toBinary(int n) {
    if (n <= 1 ) {
        return String.valueOf(n);
    }
    return toBinary(n / 2) + String.valueOf(n % 2);
}
```

Luyện tập 3: Tìm ước chung lớn nhất

- Tìm ước chung lớn nhất (*greatest common divisor* - gcd) của hai số $p, q \in \mathbb{N}$ dương $\gcd(p, q)$.
  - Nếu $q = 0$: $\gcd(p, q) = p$.
  - Nếu $p > q$: $\gcd(p, q) = \gcd(q, p \mod q)$ và ngược lại.
- Sử dụng vòng lặp (tự thực hiện!)
- Đệ quy?

Luyện tập 3: Tìm ước chung lớn nhất

```
public static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Truy ngược theo vết:

```
gcd(1440, 408)
   gcd(408, 216)
      gcd(216, 192)
         gcd(192, 24)
            gcd(24, 0)
               return 24
            return 24
         return 24
      return 24
   return 24
```

Thận trọng với đệ quy! Tính chất.

Đệ quy cho phép viết chương trình dễ hiểu/giảm công việc của người lập trình. Nhưng:

1. Vấn đề bộ nhớ: Hàm gọi đệ quy quá nhiều lần $\Rightarrow$ Vấn đề bộ nhớ! (Xem lại phần phạm vi hiệu lực của biến và thử giải thích??)
2. Khả năng nhảy ra giữa quá trình tính?
3. Phải lặp lại tính toán. Ví dụ: Tính dãy Fibonacci (Giải thích??).

Thận trọng với đệ quy! Lỗi lập trình.

1. Quên bước khởi tạo (cơ sở). Ví dụ: Tính dãy điều hòa

$$S_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$$

```
public static double harmonic(int n) {
    return harmonic(n-1) + 1.0/n;
}
```

2. Không đảm bảo tính hội tụ của công thức. Ví dụ:

```
public static double harmonic(int n) {
    if (n == 1) return 1.0;
    return harmonic(n) + 1.0/n;
}
```

Kết thúc buổi thứ ba!