

Lập trình hướng đối tượng

Mảng và cấu trúc ký tự

GV: ThS. Ngô Tiến Đức

Nội dung chính

- Mảng
- Mảng nhiều chiều
- Các hàm built-in cho mảng
- Xâu ký tự
- StringBuilder
- Regular Expression

Mảng

- Mảng (**array**): Tập hợp các phần tử (element) có cùng kiểu dữ liệu
- Một phần tử được lưu trong mảng sẽ có **index** và **value** (giá trị)
- Ví dụ mảng các số nguyên:

index	0	1	2	3	4	5	6	7	8	9
value	12	49	-2	26	5	17	-6	84	72	3

↑
element 0

↑
element 4

↑
element 9

Mảng

- Khai báo mảng:

`<type>[] <name> = new <type>[<length>];`

- Giá trị mặc định khi khai báo mảng là "zero-equivalent":

- int: 0, double: 0.0, boolean: false, object: null

- VD: `int[] arr = new int[10];` // 10 phần tử 0

- Có thể kết hợp khai báo và khởi tạo giá trị cho mảng:

- VD: `int[] arr = {1, 2, 3, 4};`

Mảng

- Lấy độ dài của mảng: `arr.length`;
- Truy cập các phần tử riêng lẻ của mảng thông qua index
 - VD: Gán giá trị cho phần tử thứ `i` của mảng: `arr[i] = 1`;
 - Lỗi `ArrayIndexOutOfBoundsException` nếu cố ý truy cập tới phần tử với index không hợp lệ
- Thường kết hợp với `for`:

```
for (int i = 0; i < arr.length; i++) {  
    arr[i] = i + 1;  
}
```

- Duyệt các phần tử của mảng:

```
for (int i = 0; i < arr.length; i++) {  
    System.out.print(arr[i] + " ");  
}
```

```
for (int i : arr) {  
    System.out.print(i + " ");  
}
```

```
int[] arr = { 1, 2, 3, 4, 5, 6};  
for (int i = 1; i < arr.length; i++) {  
    arr[i] += arr[i - 1];  
}  
  
for (int i : arr) {  
    System.out.print(arr[i] + " ");  
}  
// output?
```

Luyện tập

- Bài 1: Viết chương trình nhập vào một số nguyên và chèn số đó vào mảng các số nguyên đã được sắp xếp theo thứ tự tăng dần.

VD: Cho mảng $[1, 6, 18, 37, 64]$. Nhập vào 3 sẽ thu được mảng mới là $[1, 3, 6, 18, 37]$

- Bài 2: Viết chương trình dịch các phần tử của mảng sang trái.

VD: $[3, 8, 9, 7, 5] \rightarrow [8, 9, 7, 5, 3]$

Mảng nhiều chiều

- Thường sử dụng mảng 2 chiều (2-dimensional arrays): Mảng các mảng
- Khai báo:

```
<type>[] [] <name> = new <type>[<row_length>] [<column_length>];
```

- Ví dụ:

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

```
// a[0] = [1, 2], a[1] = [3, 4]
```

```
// a[0][0] = 1, a[0][1] = 2, a[1][0] = 3, a[1][1] = 4
```

Mảng nhiều chiều

Ví dụ: Điểm thi 10 môn của 8 sinh viên

```
double [][] quizScores = new double[8][10];
```

`quizScores[0]` →

7.5	4.5	9.5	2	7	8.5	1	8.5	7.5	6
...
...
...
...
...
4.5	2	9	9	5.5	4	7.5	6	5	9

`quizScores[7]` →

`quizScores[0][6]`

`quizScores[7][9]`

Mảng nhiều chiều

- Mảng 2 chiều thường sử dụng để biểu diễn ma trận
- Ví dụ: Tính ma trận chuyển vị

```
public double[][] transpose(double[][] a) {  
    int m = a.length;  
    int n = a[0].length;  
    double[][] b = new double[n][m];  
    for (int i = 0; i < m; i++)  
        for (int j = 0; j < n; j++)  
            b[j][i] = a[i][j];  
    return b;  
}
```

Các hàm built-in cho mảng

Lớp **Arrays** cung cấp một số phương thức được xây dựng sẵn (built-in methods) hỗ trợ thao tác với mảng

- Nằm trong package `java.util`
- Cú pháp: `Arrays.<method_name>(<params>);`

Các hàm built-in cho mảng

- `toString(array)`: In ra mảng dưới dạng chuỗi ký tự (String):

```
int intArr[] = {1, 2, 3};  
System.out.println(Arrays.toString(intArr)); // [1, 2, 3]  
float floatArr[] = new float[2];  
System.out.println(Arrays.toString(floatArr)); // [0.0, 0.0]  
String[] strArr = {"anh", "binh", "chi "};  
System.out.println(Arrays.toString(strArr)); // [an, binh, chi]
```

Các hàm built-in cho mảng

- `equals(array1, array2)`: Kiểm tra 2 mảng có bằng nhau không

```
int[] a1 = {1, 2, 3};  
int[] a2 = {1, 2, 3};  
int[] a3 = {1, 3, 2};  
  
System.out.println(Arrays.equals(a1, a2)); // true  
System.out.println(Arrays.equals(a1, a3)); // false  
System.out.println(Arrays.equals(a2, a2)); // true
```

Các hàm built-in cho mảng

- `compare(array1, array2)`: So sánh 2 mảng, trả về...?

```
int[] a1 = {1, 2, 3};  
int[] a2 = {1, 2, 3};  
int[] a3 = {1, 3, 2};  
int[] a4 = {1, 2};  
  
System.out.println(Arrays.compare(a1, a2)); // 0  
System.out.println(Arrays.compare(a1, a3)); // -1  
System.out.println(Arrays.compare(a3, a2)); // 1  
System.out.println(Arrays.compare(a1, a4)); // 1
```

Các hàm built-in cho mảng

- `compare(array1, array2)`: So sánh 2 mảng, trả về...?

```
String[] a1 = {"anh", "binh", "chi"};  
String[] a2 = {"anh", "binh"};  
String[] a3 = {"nam"};  
  
System.out.println(Arrays.compare(a1, a2)); // 1  
System.out.println(Arrays.compare(a1, a3)); // -13  
System.out.println(Arrays.compare(a3, a1)); // 13  
  
// ascii: a = 97, n = 110
```


Các hàm built-in cho mảng

- `fill(array, <value>)`: Gán giá trị cho tất cả phần tử
- `fill(array, start, end, <value>)` : Gán giá trị cho các phần tử từ vị trí **start** đến trước **end**

```
String[] arr = {"anh", "binh", "chi", "dung"};
Arrays.fill(arr, "nam");
System.out.println(Arrays.toString(arr)); // [nam, nam, nam, nam]
Arrays.fill(arr, 1, 3, "duc");
System.out.println(Arrays.toString(arr)); // [nam, duc, duc, nam]
```

Các hàm built-in cho mảng

- `sort(array)`: Sắp xếp các phần tử trong mảng theo thứ tự tăng dần

```
String[] strArr = { "Volvo", "BMW", "Tesla", "Vinfast" };  
Arrays.sort(strArr);  
System.out.println(Arrays.toString(strArr));  
// [BMW, Tesla, Vinfast, Volvo]  
  
int[] intArr = { 1, 10, 5, 25, 12 };  
Arrays.sort(intArr);  
System.out.println(Arrays.toString(intArr)); // [1, 5, 10, 12, 25]
```

Các hàm built-in cho mảng

- `sort(array, start, end)`: Sắp xếp các phần tử trong mảng từ vị trí **start** đến trước **end** theo thứ tự tăng dần

```
int[] intArr = {50, 10, 25, 1, 99, 17, 33, 12, 25};  
Arrays.sort(intArr, 2, 7);  
System.out.println(Arrays.toString(intArr));  
// [50, 10, 1, 17, 25, 33, 99, 12, 25]
```

Các hàm built-in cho mảng

- `copyOfRange(array, start, end)`: Trả về mảng sao chép từ vị trí **start** đến trước **end** của mảng cũ

```
int[] a = { 1, 2, 3, 4, 5};  
int[] a1 = Arrays.copyOfRange(a, 1, 4);  
System.out.println(Arrays.toString(a1)); // [2, 3, 4]  
int[] a2 = Arrays.copyOfRange(a, 6);  
System.out.println(Arrays.toString(a2)); // error
```

Các hàm built-in cho mảng

- `copyOf(array, length)`: Trả về mảng sao chép từ mảng cũ với độ dài mới

```
int[] a = { 1, 2, 3, 4};  
int[] a1 = Arrays.copyOf(a, 2);  
System.out.println(Arrays.toString(a1)); // [1, 2]  
int[] a2 = Arrays.copyOf(a, a.length);  
System.out.println(Arrays.toString(a2)); // [1, 2, 3, 4]  
int[] a3 = Arrays.copyOf(a, 6);  
System.out.println(Arrays.toString(a3)); [1, 2, 3, 4, 0, 0]
```

Xâu ký tự

- **String**: Kiểu dữ liệu không nguyên thủy biểu diễn một xâu/chuỗi các ký tự (sequence of characters)
- Giá trị nằm trong dấu nháy kép: "<string_value>"
- Khai báo và khởi tạo giá trị:

```
String s = new String();
```

```
String s = null;
```

```
String s = "hello";
```

```
String s = String.valueOf(1);
```

Xâu ký tự

- Có thể sử dụng toán tử + để nối String
 - `s = "abc" + "def";`
 - `s = s + "xyz";`
- Trong một số trường hợp, các kiểu dữ liệu nguyên thủy khi sử dụng cùng String và toán tử + sẽ được tự động chuyển sang kiểu String
 - `System.out.println("abc" + 1 + 2);`
 - `String s = "abc" + 1 + 2;`
 - `String s = 1 + "";`

Xâu ký tự

Một số phương thức của String (VD: `String s = " Abc Def "`):

- `s.toLowerCase(); // " abc def "`
- `s.toUpperCase(); // " ABC DEF "`
- `s.trim(); // "Abc Def"`
- `s.indexOf('b');` // 2 (-1 nếu không tìm thấy)
- `s.length(); // 9`
- `s.charAt(5); // 'D'`
- `s.substring(3); // "c Def "`
- `s.substring(2,6); // "bc D"`

Một số phương thức của String:

- `s.equals(String str)`
- `s.equalsIgnoreCase(String str)`
- `s.startsWith(String str)`
- `s.endsWith(String str)`
- `s.lastIndexOf(Char c)`
- `s.concat(String str)`
- `s.split(String str)`
- `s.replace(String str, String str);`
- ...

Xâu ký tự

Nhập một xâu ký tự từ console với Scanner:

- Phương thức `next()` để đọc một từ
- Phương thức `nextLine()` để đọc một dòng
- Nếu gọi `nextLine` sau `next`, `nextInt`, `nextDouble`,... ?

```
Scanner sc = new Scanner(System.in);
```

```
int n = sc.nextInt();
```

```
String s = sc.nextLine();
```

```
System.out.print("End.");
```

Xử lý tình trạng trôi dòng khi nhập với Scanner:

- Cách 1: Gọi `nextLine()` sau khi nhập số và trước khi nhập dòng

```
int n = sc.nextInt();
```

```
sc.nextLine();
```

```
String s = sc.nextLine();
```

- Cách 2:

```
int n = Integer.parseInt(sc.nextLine());
```

```
String s = sc.nextLine();
```

StringBuilder

- Một đối tượng String đã khởi tạo sẽ không thể thay đổi (immutable)

```
String s = "abc";
```

```
s.concat("def");
```

```
System.out.println(s); // "abc"
```

- **StringBuilder** cho phép tạo chuỗi ký tự có thể thay đổi giá trị
- Khởi tạo StringBuilder:
 - `StringBuilder sb = new StringBuilder();`
 - `StringBuilder sb = new StringBuilder("This is a string builder");`

StringBuilder

Một số phương thức của StringBuilder:

```
sb = "This is a string builder"
```

- `sb.append(String s)` : Nối xâu

VD: `sb.append(" java");` // "This is a string builder java"

- `sb.insert(int offset, String s)` : Chèn xâu **s** vào vị trí trước **offset**

VD: `sb.insert(10, "java ");` // "This is a java string builder"

- `sb.delete(int start, int end)` : Xóa các ký tự có index từ **start** đến trước **end**

VD: `sb.delete(2, 7);` // "Th a string builder"

StringBuilder

Một số phương thức của StringBuilder:

```
sb = "This is a string builder"
```

- `sb.replace(int start, int end, String s)` : Thay thế các ký tự có index từ **start** đến trước **end** bằng **s**

VD: `sb.replace(1, 5, "java");` // "Tjavais a string builder"

- `sb.reverse()` : Đảo ngược chuỗi

VD: `sb.reverse();` // "redliub gnirts a si sihT"

- `sb.toString()` : Chuyển về String

Regular Expression

- Regular Expression (**RegEx**): Biểu thức chính quy là chuỗi ký tự tạo thành các mẫu tìm kiếm (search pattern)
- RegEx được sử dụng để tìm kiếm (và thay thế) những dữ liệu có quy tắc trong văn bản
- RegEx có thể là 1 ký tự hoặc những pattern rất phức tạp
- Ví dụ: Loại bỏ khoảng trắng thừa, tìm kiếm tất cả số điện thoại,...

Regular Expression

- Metacharacter: Các ký tự có ý nghĩa đặc biệt

	Kết quả khớp với một trong các pattern. VD: saturday sunday
.	Ký tự bất kỳ
^	Bắt đầu bằng một xâu ký tự. VD: ^012
\$	Kết thúc bằng một xâu ký tự. VD: 789\$
\d	Chữ số
\s	Ký tự khoảng trắng
...	...

Regular Expression

- RegEx pattern:

[abc]	Kết quả chứa một trong các ký tự trong ngoặc
[^abc]	Kết quả không chứa các ký tự nằm trong ngoặc
[a-b]	Kết quả chứa các ký tự từ a đến b (VD: a-z, A-Z, 0-9)
a+	Kết quả có ít nhất một a
a{x}	Kết quả có số lần xuất hiện của a là x lần
a{x,y}	Kết quả có số lần xuất hiện của a từ x đến y
a{x,}	Kết quả có số lần xuất hiện của a ít nhất là x lần
...	...

Regular Expression

- Java cung cấp package `java.util.regex`
- Các lớp thường được sử dụng:
 - **Pattern**: Định nghĩa một pattern (sử dụng trong quá trình tìm kiếm)
 - **Matcher**: Thực hiện tìm kiếm
 - **PatternSyntaxException**: Phát hiện các lỗi cú pháp trong pattern

Regular Expression

- Ví dụ: Loại bỏ khoảng trắng thừa sử dụng các class của regex:

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

public class RegexDemo {

    public static String removeWhitespace(String str) {

        String reg = "\\s+";

        Pattern pattern = Pattern.compile(reg);

        Matcher matcher = pattern.matcher(str);

        return matcher.replaceAll(" ");

    }

}
```

Regular Expression

- Ví dụ: Loại bỏ khoảng trắng thừa sử dụng các class của regex (tiếp):

```
public static void main(String[] args) throws Exception {  
    String str = "  abbb    ccc  d  ";  
    String result = removeWhitespace(str).trim();  
    System.out.println(result);  
}  
}
```

Regular Expression

- Ví dụ: Loại bỏ khoảng trắng thừa chỉ cần sử dụng String:

```
public static void main(String[] args) {  
    String str = "   abbb   ccc   d ";  
    String result = str.replaceAll("\\s+", " ").trim();  
    System.out.println(result);  
}
```

Regular Expression

Luyện tập: Viết RegEx cho các định dạng sau:

- Số điện thoại Việt Nam (đầu +84 và đầu 0)
- Password: Độ dài 6 – 20 ký tự, trong đó chứa ít nhất 1 chữ thường, 1 chữ hoa, 1 số và 1 ký tự đặc biệt
- Các email



- 39