Ministry of SMEs and Startups × KOSME Korea SMEs and Startups Agency × LIKELION

StringBuilder, StringBuffer

Nguyễn Anh Tuấn

KTECH COLLEGE

Nội dung bài giảng

1. StringBuilder, StringBuffer
2. Date and Time
3. Mảng 1 chiều, mảng 2 chiều
4. Kiểu dữ liệu tập hợp

Vòng lặp (phần mở rộng)

StringBuider, String Buffer

String

❖ **Định nghĩa:**

➢ Các đối tượng String là bất biến, có nghĩa là một khi một đối tượng String được tạo ra, giá trị của nó không thể thay đổi. Bất kỳ thao tác nào thay đổi chuỗi thực sự sẽ tạo ra một đối tượng mới. Ví dụ:

```
public class HelloWorld {
        public static void main(String[] args) {
                String str = "Hello";
                str = str + " World";          // Tạo ra một đối tượng String mới
                System.out.println(str);  // Output: Hello World
        }
}
```

String

❖ **Lưu ý:**

- ➢ Các đối tượng String là bất biến, nên khi thay đổi giá trị của chuỗi sẽ tạo ra một đối tượng String mới.
- ➢ Đối tượng String cũ sẽ không thể được truy cập và không thể xoá thủ công. Chúng sẽ được Java tự dọn dẹp để giải phóng bộ nhớ bằng cơ chế **garbage collection.**
- ➢ Việc sử dụng String chỉ nên khai báo trong trường hợp không thay đổi giá trị của chuỗi trong tương lai.

StringBuilder

❖ **Định nghĩa:**

➢ Các đối tượng StringBuilder có thể thay đổi được giá trị của chuỗi mà không cần tạo ra đối tượng mới. Ví dụ:

```
public class HelloWorld {
        public static void main(String[] args) {
                StringBuilder sb = new StringBuilder("Hello");
                sb.append(" World");              // Thay đổi trực tiếp đối tượng StringBuilder
                System.out.println(sb.toString()); // Output: Hello World
        }
}
```

StringBuilder

❖ **Lưu ý:**

- ➢ StringBuilder không tạo ra đối tượng mới nên không cần quan tâm đến đối tượng dư thừa như String.
- ➢ StringBuilder không được đồng bộ hoá nên chỉ ưu tiên chạy trong môi trường đơn luồng.
- ➢ StringBuilder được sử dụng khi cần thay đổi giá trị của chuỗi trong tương lai.

StringBuffer

❖ **Định nghĩa:**

➢ Các đối tượng StringBuffer có thể thay đổi được giá trị của chuỗi mà không cần tạo ra đối tượng mới. Ví dụ:

```
public class HelloWorld {
        public static void main(String[] args) {
                StringBuffer sb = new StringBuffer("Hello");
                sb.append(" World");              // Thay đổi trực tiếp đối tượng StringBuffer
                System.out.println(sb.toString()); // Output: Hello World
        }
}
```

StringBuilder

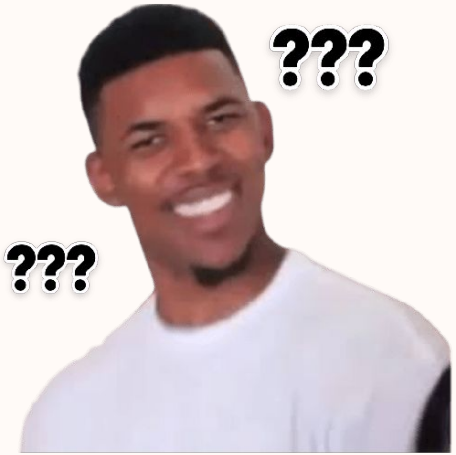
❖ **Lưu ý:**

- ➢ StringBuffer cũng giống như StringBuilder là không tạo ra đối tượng mới nên không cần quan tâm đến đối tượng dư thừa như String.
- ➢ StringBuffer được đồng bộ hoá nên ưu tiên chạy trong môi trường đa luồng.
- ➢ StringBuffer được sử dụng khi cần thay đổi giá trị của chuỗi trong tương lai.

String, StringBuilder, StringBuffer

❖ **Câu hỏi được đặt ra:**

1. **String** và **StringBuilder**, đối tượng nào đảm bảo hiệu suất tốt hơn khi cần giữ nguyên giá trị chuỗi trong môi trường **đa luồng**?
2. Sử dụng **StringBuilder** hoặc **StringBuffer** trong trường hợp nào thì đảm bảo hiệu suất tốt hơn?
3. Sử dụng **StringBuilder** trong môi trường đa luồng thì điều gì sẽ xảy ra? Ngược lại trong môi trường đơn luồng, điều gì sẽ xảy ra khi sử dụng **StringBuffer**?



Date and Time

Date and Time (trước Java 8)

❖ **java.util.Date**

Đại diện cho một điểm thời gian cụ thể với độ chính xác đến milisecond.

```
import java.util.Date;

public class HelloWorld {
	public static void main(String[] args) {
		// Lấy thời gian hiện tại
		Date currentDate = new Date();
		System.out.println("Current Date: " + currentDate);
	}
}
```

Date and Time (trước Java 8)

❖ **java.util.Calendar**

Cung cấp các phương thức để chuyển đổi giữa một thời điểm cụ thể và các trường như năm, tháng, ngày, giờ.

```
import java.util.Calendar;

public class CalendarExample {
        public static void main(String[] args) {
                // Lấy thời gian hiện tại
                Calendar calendar = Calendar.getInstance();
                System.out.println("Current Date and Time: " + calendar.getTime());
        }
}
```

Date and Time (từ Java 8 trở đi)

❖ **java.time.LocalDate**

Đại diện cho một ngày không có thời gian hoặc múi giờ.

```
import java.time.LocalDate;

public class HelloWorld {
	public static void main(String[] args) {
		// Lấy ngày hiện tại
		LocalDate currentDate = LocalDate.now();
		System.out.println("Current Date: " + currentDate);
	}
}
```

Date and Time (từ Java 8 trở đi)

❖ **java.time.LocalTime**

Đại diện cho một thời gian trong ngày mà không có ngày hoặc múi giờ.

```
import java.time.LocalTime;

public class HelloWorld {
	public static void main(String[] args) {
		// Lấy giờ hiện tại
		LocalTime currentTime = LocalTime.now();
		System.out.println("Current Time: " + currentTime);
	}
}
```

Date and Time (từ Java 8 trở đi)

❖ **java.time.LocalDateTime**

Kết hợp LocalDate và LocalTime để đại diện cho cả ngày và thời gian mà không có múi giờ.

```
import java.time.LocalDateTime;

public class HelloWorld {
        public static void main(String[] args) {
                // Lấy ngày giờ hiện tại
                LocalDateTime currentDateTime = LocalDateTime.now();
                System.out.println("Current Date and Time: " + currentDateTime);
}
```

Date and Time (từ Java 8 trở đi)

❖ **java.time.format.DateTimeFormatter**

Cung cấp các phương thức để định dạng và phân tích ngày và giờ, có thể được tùy chỉnh để phù hợp với các định dạng ngày giờ khác nhau.

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class HelloWorld {
        public static void main(String[] args) {
                // Định dạng ngày giờ
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                String formattedDateTime = currentDateTime.format(formatter);
                System.out.println("Formatted Date and Time: " + formattedDateTime);
}
```

Mảng 1 chiều, mảng 2 chiều

Mảng 1 chiều

❖ Mảng 1 chiều là một cấu trúc dữ liệu bao gồm một dãy các phần tử cùng kiểu dữ liệu. Mỗi phần tử trong mảng được truy cập thông qua chỉ số (index) của nó, bắt đầu từ 0.

```
// Cách khai báo mảng
int[] array;

// Khởi tạo mảng với kích thước cố định
array = new int[5];

// Khai báo và khởi tạo mảng cùng lúc
int[] array = new int[5];

// Khai báo và khởi tạo mảng với các giá trị ban đầu
int[] array = {1, 2, 3, 4, 5};
```

Mảng 1 chiều

❖ Cách truy cập và thao tác với mảng 1 chiều:

```
public class HelloWorld {
        public static void main(String[] args) {
                // Khai báo và khởi tạo mảng
                int[] array = {1, 2, 3, 4, 5};

                // Truy cập và in ra các phần tử của mảng
                for (int i = 0; i < array.length; i++) {
                        System.out.println("Element at index " + i + ": " + array[i]);
                }

                // Thay đổi giá trị của một phần tử trong mảng
                array[2] = 10;
                System.out.println("Updated Element at index 2: " + array[2]);
        }
}
```

Mảng 2 chiều

❖ Mảng 2 chiều là một cấu trúc dữ liệu bao gồm một ma trận các phần tử, được tổ chức theo dạng lưới với các hàng và cột. Mỗi phần tử trong mảng hai chiều được truy cập thông qua hai chỉ số: chỉ số hàng và chỉ số cột.

```
// Cách khai báo mảng hai chiều
int[][] matrix;

// Khởi tạo mảng hai chiều với kích thước cố định
matrix = new int[3][4];

// Khai báo và khởi tạo mảng hai chiều cùng lúc
int[][] matrix = new int[3][4];

// Khai báo và khởi tạo mảng hai chiều với các giá trị ban đầu
int[][] matrix = {
        {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}
};
```

Mảng 2 chiều

❖ Cách truy cập và thao tác với mảng 2 chiều:

```
// Khai báo và khởi tạo mảng hai chiều
int[][] matrix = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };

// Truy cập và in ra các phần tử của mảng hai chiều
for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[i].length; j++) {
                System.out.println("Element at row " + i + ", column " + j + ": " + matrix[i][j]);
        }
}

// Thay đổi giá trị của một phần tử trong mảng hai chiều
matrix[1][2] = 20;
System.out.println("Updated Element at row 1, column 2: " + matrix[1][2]);
```

Kiểu dữ liệu tập hợp

Kiểu dữ liệu tập hợp (Collections Framework)

- Collections Framework trong Java là 1 lớp Interface gốc, các đối tượng như danh sách (**List**) và tập hợp (**Set**) đều kế thừa từ lớp interface Collection.
  - **List interface:** Một danh sách có thứ tự cho phép các phần tử trùng lặp.
  - **Set interface:** Một tập hợp không chứa các phần tử trùng lặp.

Kiểu dữ liệu tập hợp - List interface

- Các lớp con kế thừa từ List interface: ArrayList, LinkedList
  - **ArrayList:** Một danh sách động có thể thay đổi kích thước, truy cập phần tử nhanh nhờ vào chỉ số index

```
// Khởi tạo một ArrayList
List<String> arrayList = new ArrayList<>();

// Thêm phần tử vào danh sách
arrayList.add("Apple"); arrayList.add("Banana");

// Thay đổi giá trị của một phần tử
arrayList.set(1, "Blueberry");

// Xóa một phần tử
arrayList.remove("Apple");
```

Kiểu dữ liệu tập hợp - List interface

- Các lớp con kế thừa từ List interface: ArrayList, LinkedList
  - **LinkedList:** Một danh sách động có thể thay đổi kích thước, truy cập phần tử nhanh nhờ vào chỉ số index

```
// Khởi tạo một LinkedList
List<String> linkedList = new LinkedList<>();

// Thêm phần tử vào danh sách
linkedList.add("Apple"); linkedList.add("Banana");

// Thay đổi giá trị của một phần tử
linkedList.set(1, "Blueberry");

// Xóa một phần tử
linkedList.remove("Apple");
```

Kiểu dữ liệu tập hợp - List interface

ArrayList

- Truy cập mảng nhanh nhờ có index
- Thêm/xoá ở cuối mảng nhanh
- Thêm/xoá ở đầu/giữa mảng chậm
- Tăng kích thước mảng gây tốn tài nguyên

LinkedList

- Truy cập mảng chậm do phải duyệt từng node
- Thêm/xoá ở đầu/cuối nhanh
- Thêm/xoá ở giữa tương đối nhanh
- Tốn tài nguyên lưu trữ các node
- Liên kết giữa các node là 1 chiều hoặc 2 chiều

Kiểu dữ liệu tập hợp - Set interface

- Các lớp con kế thừa từ Set interface: HashSet, LinkedHashSet, TreeSet
  - **HashSet:** Một tập hợp không có thứ tự, cho phép truy cập nhanh.

```
// Khởi tạo một HashSet
Set<String> hashSet = new HashSet<>();

// Thêm phần tử vào tập hợp
hashSet.add("Apple"); hashSet.add("Banana");

// Kiểm tra một phần tử có tồn tại
hashSet.contains("Banana")

// Xóa một phần tử
hashSet.remove("Apple");
```

Kiểu dữ liệu tập hợp - Map interface

- Map interface: Một tập hợp các cặp khóa-giá trị (key-value), không cho phép trùng lặp khóa.
- Các lớp con kế thừa từ Map interface: HashMap, LinkedHashMap, TreeMap
  - **HashMap:** Một tập hợp không có thứ tự, cho phép truy cập nhanh.

```
// Khởi tạo một HashMap
Map<String, Integer> hashMap = new HashMap<>();

// Thêm cặp khoá vào map
hashMap.put("Apple", 1); hashMap.put("Banana", 2);

// Thay đổi giá trị của một phần tử
hashMap.put("Banana", 20);

// Xóa một phần tử
hashMap.remove("Apple");
```

Vòng lặp (mở rộng)

Vòng lặp (mở rộng)

❖ **Continue:** Câu lệnh continue được sử dụng để bỏ qua các câu lệnh bên dưới trong vòng lặp và tiếp tục với lần lặp tiếp theo.

```
for (int i = 0; i < 5; i++) {
        if (i == 2 || i == 4) {
                continue; // Bỏ qua in số 2 và số 4
        } System.out.println(i);
}
```

Vòng lặp (mở rộng)

❖ **Break:** Câu lệnh break được sử dụng để kết thúc ngay lập tức vòng lặp hoặc câu lệnh switch bên trong mà nó đang nằm trong

```
for (int i = 0; i < 5; i++) {
        if (i == 3) {
                break; // Kết thúc vòng lặp khi i == 3
        }
        System.out.println(i);
}
```

ROAD TO KOREA

Nếu có bất kỳ thắc mắc nào, hãy đặt câu hỏi qua

mail@mail.com hoặc Zalo 0xxx xxx xxx