Tên và số JEP

Tên đầy đủ: Local-Variable Type Inference

Số hiệu: 286

Trạng thái: Feature

Ngày cập nhật gần nhất: 2022/09/28 16:49

Mục tiêu

Mục tiêu chính của tính năng này là cải thiện trải nghiệm của lập trình viên bằng cách **giảm bớt sự rườm rà (boilerplate)** của các cú pháp dài dòng, không cần thiết, cho phép bỏ qua việc khai báo kiểu tường minh cho các biến cục bộ.Điều này không chỉ giúp mã nguồn trở nên gọn gàng và **tăng tính dễ đọc** khi tên biến đã đủ rõ ràng, mà còn **khuyến khích việc khai báo biến tốt hơn** bằng cách giảm bớt "gánh nặng" khi cần chia nhỏ các biểu thức phức tạp thành những biến trung gian đơn giản. Quan trọng nhất, tính năng này vẫn hoàn toàn đảm bảo cam kết của Java về **an toàn kiểu tĩnh (static type safety)**, vì kiểu của biến vẫn được trình biên dịch xác định và kiểm tra chặt chẽ tại thời điểm biên dịch.

Chi tiết kỹ thuật

Về mặt kỹ thuật, tính năng này cho phép sử dụng tên kiểu dành riêng **var** để thay thế cho việc khai báo kiểu tường minh, giúp mã nguồn gọn hơn đáng kể. Một điểm quan trọng là var **không phải là một từ khóa** (keyword) mà là một tên kiểu dành riêng. Điều này đảm bảo tính tương thích ngược, nghĩa là các mã nguồn cũ đã sử dụng var làm tên biến hoặc tên phương thức sẽ không bị lỗi.

Tuy nhiên, phạm vi áp dụng của var khá chặt chẽ: nó chỉ có thể được sử dụng cho **biến cục bộ có giá trị khởi tạo ngay lập tức** và cho các biến trong vòng lặp for. Ngược lại, var **không được phép** sử dụng cho các trường của lớp (fields), tham số hay kiểu trả về của phương thức. Ngoài ra, bạn cũng không thể dùng var để khai báo một biến mà không khởi tạo giá trị, gán giá trị null ban đầu, hoặc với các biểu thức cần kiểu mục tiêu rõ ràng như biểu thức lambda.

Ảnh hưởng

Về mặt ảnh hưởng, tính năng var có **mức độ áp dụng rất cao**; một cuộc khảo sát trên mã nguồn của OpenJDK đã chỉ ra rằng có tới 87% các khai báo biến cục bộ có thể được chuyển đổi để sử dụng var, cho thấy tác động rộng rãi và tính hữu dụng của nó

trong thực tế. Tuy nhiên, việc sử dụng var cũng đi kèm với một số rủi ro. Lớn nhất là **rủi ro về tính dễ đọc**: nếu lạm dụng hoặc kết hợp với tên biến không rõ ràng, mã nguồn có thể trở nên khó hiểu, và trách nhiệm viết mã sạch cuối cùng vẫn thuộc về người lập trình. Ngoài ra, còn có một **rủi ro rất nhỏ về tương thích ngược**, đó là khả năng gây xung đột nếu mã nguồn cũ đã từng sử dụng var làm tên của một lớp (class) hoặc giao diện (interface), dù trường hợp này cực kỳ hiếm vì nó vi phạm quy ước đặt tên trong Java.

DEMO

```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        System.out.println("=== JEP 286: Local-Variable Type Inference ===\n");

        // 1. Thay đổi của cách viết truyền thống với var
        traditionalVsVar();

        // 2. Ví dụ minh họa
        examples();

        // 3. Các tình huống lỗi
        // errorCases(); // Uncomment để xem lỗi compile
    }
```

```java
    // 1. Thay đổi của cách viết truyền thống với var
    private static void traditionalVsVar() {  1 usage
        System.out.println("1. TRUYỀN THỐNG vs VAR:");

        // Truyền thống
        String message = "Hello";
        int number = 42;
        List<String> list = new ArrayList<>();
        Map<String, Integer> map = new HashMap<>();

        // Với var
        var message2 = "Hello";
        var number2 = 42;
        var list2 = new ArrayList<String>();
        var map2 = new HashMap<String, Integer>();

        System.out.println("  Cả hai cách đều tương đương!\n");
    }
```

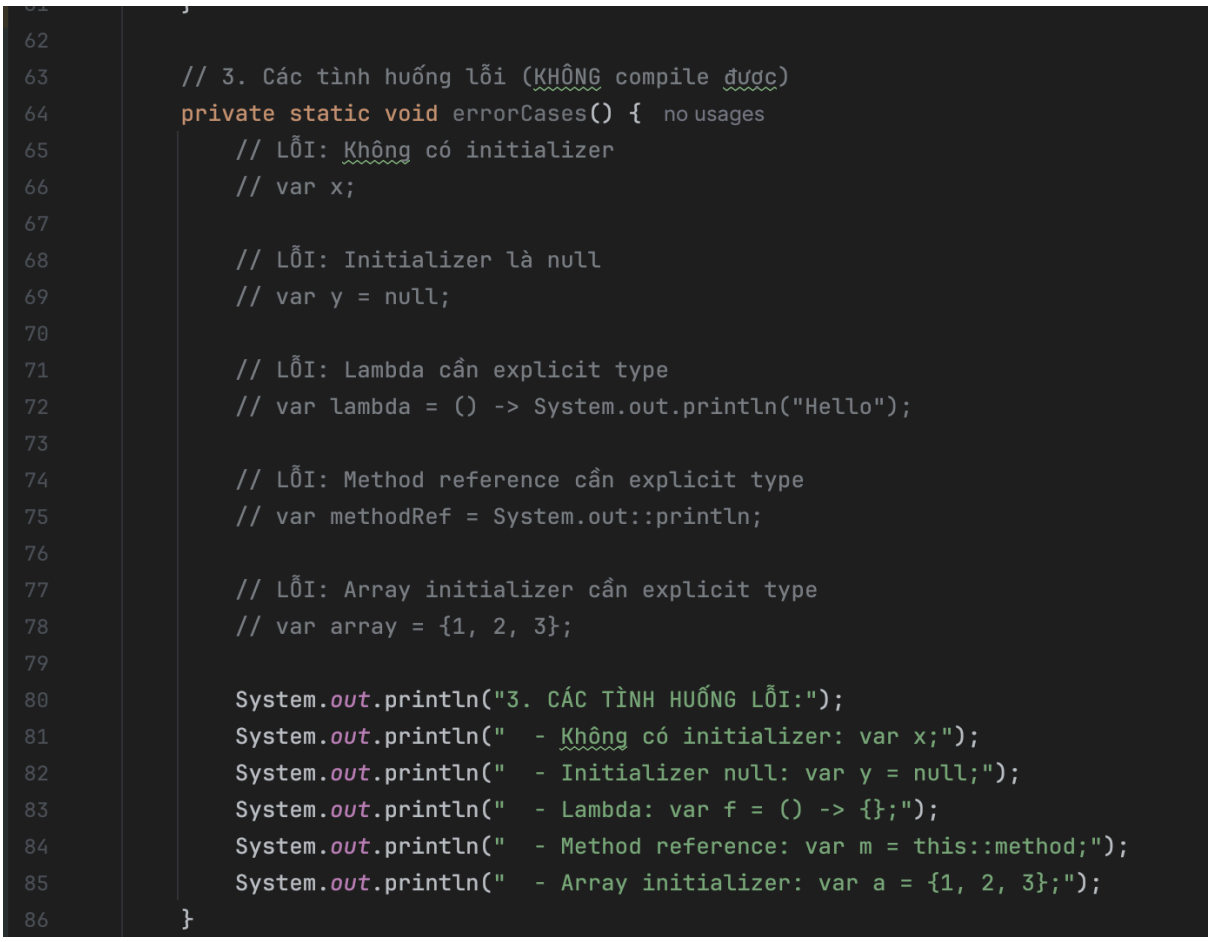
```java
    // 2. Ví dụ minh họa
    private static void examples() {  1 usage
        System.out.println("2. VÍ DỤ MINH HỌA:");

        // Collections
        var names = List.of("Alice", "Bob", "Charlie");
        System.out.println("  Names: " + names);

        // Streams
        var evenNumbers = List.of(1, 2, 3, 4, 5, 6) List<Integer>
                .stream() Stream<Integer>
                .filter( Integer n -> n % 2 == 0)
                .toList();
        System.out.println("  Even: " + evenNumbers);

        // For loops
        for (var name : names) {
            System.out.println("  - " + name);
        }

        // Complex types
        var userMap = new HashMap<String, List<Integer>>();
        userMap.put("scores", List.of(90, 85, 95));
        System.out.println("  User map: " + userMap + "\n");
    }
```



```
    }

    // 3. Các tình huống lỗi (KHÔNG compile được)
    private static void errorCases() {  no usages
        // LỖI: Không có initializer
        // var x;

        // LỖI: Initializer là null
        // var y = null;

        // LỖI: Lambda cần explicit type
        // var lambda = () -> System.out.println("Hello");

        // LỖI: Method reference cần explicit type
        // var methodRef = System.out::println;

        // LỖI: Array initializer cần explicit type
        // var array = {1, 2, 3};

        System.out.println("3. CÁC TÌNH HUỐNG LỖI:");
        System.out.println("  - Không có initializer: var x;");
        System.out.println("  - Initializer null: var y = null;");
        System.out.println("  - Lambda: var f = () -> {};");
        System.out.println("  - Method reference: var m = this::method;");
        System.out.println("  - Array initializer: var a = {1, 2, 3};");
    }
```

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        System.out.println("=== JEP 286: Local-Variable Type Inference ===\n");

        // 1. Thay đổi của cách viết truyền thống với var
        traditionalVsVar();

        // 2. Ví dụ minh họa
        examples();

        // 3. Các tình huống lỗi
        // errorCases(); // Uncomment để xem lỗi compile
    }

    // 1. Thay đổi của cách viết truyền thống với var
    private static void traditionalVsVar() {
        System.out.println("1. TRUYỀN THỐNG vs VAR:");

        // Truyền thống
        String message = "Hello";
```

```
    int number = 42;
    List<String> list = new ArrayList<>();
    Map<String, Integer> map = new HashMap<>();

    // Với var
    var message2 = "Hello";
    var number2 = 42;
    var list2 = new ArrayList<String>();
    var map2 = new HashMap<String, Integer>();

    System.out.println("  Cả hai cách đều tương đương!\n");
}

// 2. Ví dụ minh họa
private static void examples() {
    System.out.println("2. VÍ DỤ MINH HỌA:");

    // Collections
    var names = List.of("Alice", "Bob", "Charlie");
    System.out.println("  Names: " + names);

    // Streams
    var evenNumbers = List.of(1, 2, 3, 4, 5, 6)
            .stream()
            .filter(n -> n % 2 == 0)
            .toList();
    System.out.println("  Even: " + evenNumbers);

    // For loops
    for (var name : names) {
        System.out.println("  - " + name);
    }

    // Complex types
    var userMap = new HashMap<String, List<Integer>>();
    userMap.put("scores", List.of(90, 85, 95));
    System.out.println("  User map: " + userMap + "\n");
}

// 3. Các tình huống lỗi (KHÔNG compile được)
private static void errorCases() {
    // LỖI: Không có initializer
    // var x;

    // LỖI: Initializer là null
    // var y = null;

    // LỖI: Lambda cần explicit type
```

```
        // var lambda = () -> System.out.println("Hello");

        // LỖI: Method reference cần explicit type
        // var methodRef = System.out::println;

        // LỖI: Array initializer cần explicit type
        // var array = {1, 2, 3};

        System.out.println("3. CÁC TÌNH HUỐNG LỖI:");
        System.out.println("  - Không có initializer: var x;");
        System.out.println("  - Initializer null: var y = null;");
        System.out.println("  - Lambda: var f = () -> {};");
        System.out.println("  - Method reference: var m = this::method;");
        System.out.println("  - Array initializer: var a = {1, 2, 3};");
    }
}
```