Trần Quang Huy @tran.quang.huy Theo dõi
559 21 7

Đã đăng vào thg 1 3, 2020 11:06 SA - 10 phút đọc

Từ khóa static trong Java và lập trình hướng đối tượng

Trong Java, static là một từ khóa mà chúng ta rất hay thường gặp khi lập trình.

Vậy static là gì? Sử dụng chúng trong trường hợp nào? Bài hôm nay mình sẽ giới thiệu với mọi người xung quanh từ khóa này.

Biến của lớp và phương thức của lớp

Thông thường, mỗi một phương thức hay một thuộc tính nào đó đều gắn chặt với một đối tượng cụ thể. Muốn truy

Tuy nhiên, trong một số trường hợp, ta muốn có dữ liệu nào đó của lớp được chia sẻ giữa tất cả các đối tượng thuộc một lớp, các phương thức của lớp hoạt động độc lập với các đối tượng của lớp đó, thì giải pháp là các biến lớp và phương thức lớp.

1 - Biến của lớp(biến static)

Đôi khi, ta muốn một lớp có những biến dùng chung cho tất cả các đối tượng thuộc lớp đó. Ta gọi các biến dùng chung này là biến của lớp (class variable), hay gọi tắt là biến lớp. Chúng không gắn với bất cứ một đối tượng nào mà chỉ gắn với lớp đối tượng. Chúng được dùng chung cho tất cả các đối tượng trong lớp đó.

Để phân biệt giữa biến thực thể và biến lớp khi khai báo trong định nghĩa lớp, ta dùng từ khóa `static` cho các biến lớp. Vì từ khóa đó nên biến lớp thường được gọi là *biến static*.

Lấy ví dụ sau, bên cạnh biến thực thể `name`, lớp `Cow` còn có một biến lớp `numOfCows` với mục đích ghi lại số lượng các đối tượng `Cow` đã được tạo.

Mỗi đối tượng `Cow` có một biến `name` của riêng nó, nhưng `numOfCows` thì chỉ có đúng một bản dùng chung cho tất cả các đối tượng `Cow`.

```
public class Cow {
  private String name;

  public static int numOfCows = 0;

  public Cow(String theName) {
    name = theName;

    numOfCows++;

    System.out.println("Cow #"+numOfCows+" created.");
  }
}
```

biến thực thể, không có từ khóa static

biến lớp, được khai báo với từ khóa static

mỗi lần hàm tạo chạy (một đối tượng mới được tạo), bản duy nhất của numOfCows được tăng thêm 1 để ghi nhận đối tượng mới

```
public class CowTestDrive {
  public static void main(String[] args) {
    Cow c1 = new Cow();
    Cow c2 = new Cow();
  }
}
```

```
% java CowTestDrive
Cow #1 created.
Cow #2 created.
```

Từ bên ngoài lớp, ta có thể dùng tên lớp để truy nhập biến `static`. Chẳng hạn, dùng `Cow.numOfCows` để truy nhập `numOfCows`:

2 - Phương thức của lớp(hàm static)

Lại xét ví dụ trong phần 1, giả sử ta muốn `numOfCows` là biến `private` để không cho phép ai đó sửa từ bên ngoài lớp `Cow`.

Nhưng ta vẫn muốn cho phép đọc giá trị của biến này từ bên ngoài, nên ta sẽ bổ sung một phương thức, chẳng hạn `getCount()`, để trả về giá trị của biến đó.

```
public int getCount() {
 return numOfCows;
}
```

Như các phương thức mà ta đã quen dùng, để gọi `getCount()`, người ta sẽ cần đến một tham chiếu kiểu `Cow` và kích hoạt phương thức đó cho một đối tượng `Cow`.

Tuy nhiên, sẽ có những vấn đề xảy ra như sau:

- Cần đến một con bò để biết được có tất cả bao nhiêu con bò? Nghe có vẻ không được tự nhiên lắm.
- `getCount()` không dùng đến một đặc điểm hay dữ liệu đặc thù nào của mỗi đối tượng `Cow`
- Khi còn chưa có một đối tượng `Cow` nào được tạo thì không thể gọi được `getCount()`

Phương thức `getCount()` không nên bị phụ thuộc vào các đối tượng `Cow` cụ thể như vậy.

Để giải quyết vấn đề này, ta có thể cho `getCount()` làm một phương thức của lớp (class method), thường gọi tắt là

tử lớp mà không cần đến một thành chiếu đối tượng nào.

Ta dùng từ khóa `static` khi khai báo phương thức lớp:

Đặc điểm độc lập đối với các đối tượng của phương thức static chính là lí do ta đã luôn luôn phải khai báo phương thức `main()` với từ khóa `static`.

`main()` được kích hoạt để khởi động chương trình - khi chưa có bất cứ đối tượng nào được tạo – nên nó phải được phép chạy mà không gắn với bất cứ đối tượng nào.

3 - Giới hạn của phương thức lớp

Đặc điểm về tính độc lập đó vừa là ưu điểm vừa là giới hạn cho hoạt động của các phương thức lớp.

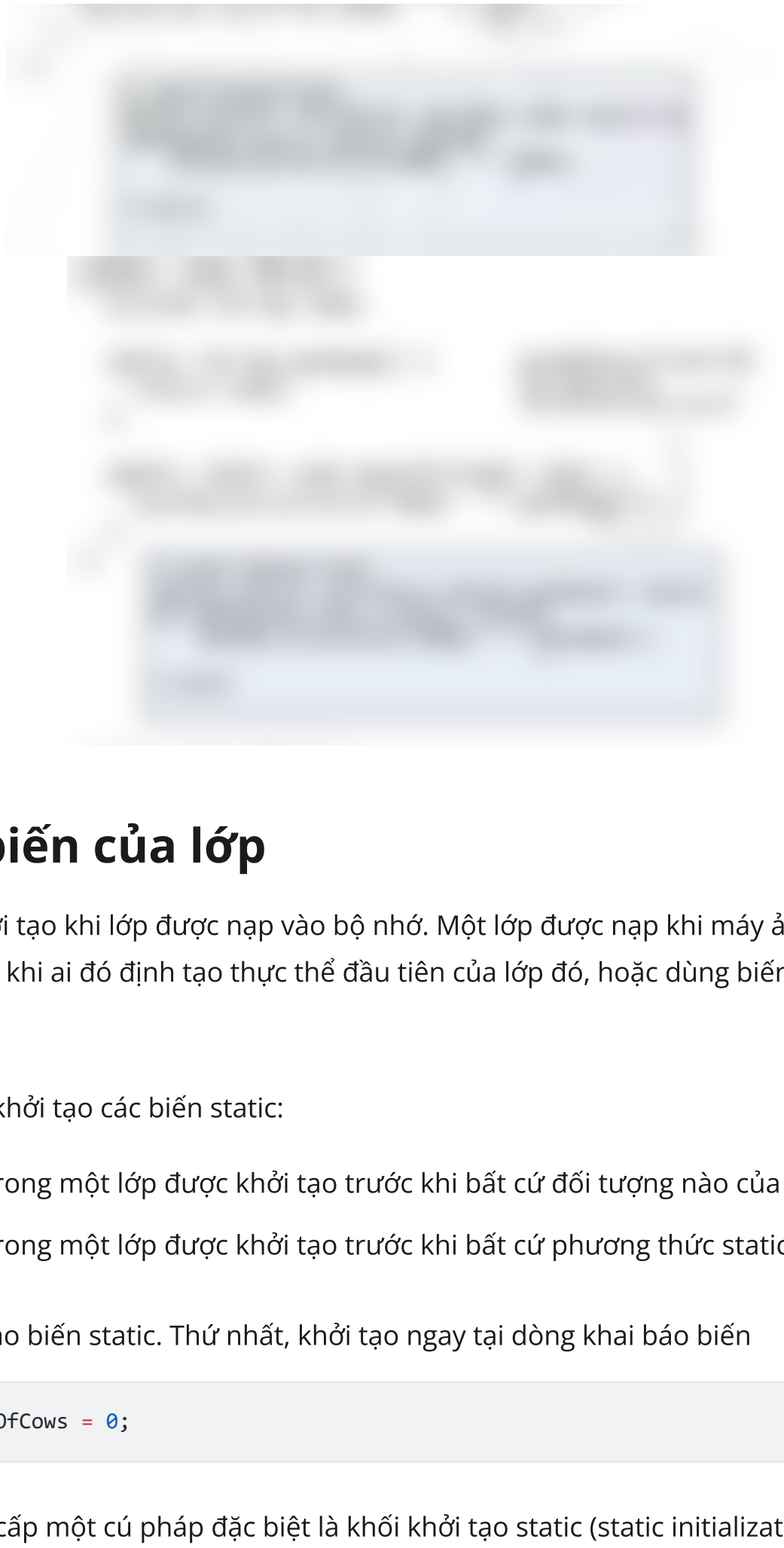
Không được gắn với một đối tượng nào, nên các phương thức static của một lớp chạy mà không biết một chút gì về bất cứ đối tượng cụ thể nào của lớp đó.

Như đã thấy trong ví dụ phần 2, `getCount()` chạy ngay cả khi không tồn tại bất cứ đối tượng `Cow` nào.

Kể cả khi gọi `getCount()` từ một đối tượng cụ thể thì `getCount()` cũng vẫn không biết gì về đối tượng `Cow` của đối tượng đó.

Vì khi đó, trình biên dịch chỉ dùng kiểu khai báo `Cow` để xác định nên chạy `getCount()` của lớp nào, nó không quan tâm tới đối tượng nào.

Nếu một biến thực thể được dùng đến trong một phương thức lớp, trình biên dịch sẽ không hiểu ta đang nói đến biến thực thể của đối tượng nào, bất kể trong `heap` đang có 10 hay chỉ có duy nhất một đối tượng thuộc lớp đó. Tương tự khi gọi các phương thức của thực thể trong các phương thức static



4 - Khởi tạo biến của lớp

Các biến static được khởi tạo khi lớp được nạp vào bộ nhớ. Một lớp được nạp khi máy ảo Java quyết định đến lúc cần nạp, chẳng hạn như khi ai đó định tạo thực thể đầu tiên của lớp đó, hoặc dùng biến static hoặc phương thức static của lớp đó.

Có hai đảm bảo về việc khởi tạo các biến static:

- Các biến `static` trong một lớp được khởi tạo trước khi bất cứ đối tượng nào của lớp đó có thể được tạo
- Các biến `static` trong một lớp được khởi tạo trước khi bất cứ phương thức static nào của lớp đó có thể chạy

Ta có hai cách để khởi tạo biến static. Thứ nhất, khởi tạo ngay tại dòng khai báo biến

```
private static int numOfCows = 0;
```

Cách thứ hai: Java cung cấp một cú pháp đặc biệt là khối khởi tạo static (static initialization block) – một khối mã được bọc trong cặp ngoặc { } và có tiêu đề là từ khóa static.

```
static {
 numOfCows = 0;
}
```

Một lớp có thể có vài khối khởi tạo static đặt ở bất cứ đâu trong định nghĩa lớp. Chúng được đảm bảo sẽ được kích hoạt theo đúng thứ tự xuất hiện trong mã.

Và quan trọng bậc nhất là chúng được đảm bảo sẽ chạy trước khi bất gì biến thành viên nào được truy nhập hay phương thức static nào được chạy.

5 - Tổng kết

- Phương thức lớp hay còn gọi là phương thức static không được gắn với một đối tượng cụ thế nào và không phụ thuộc đối tượng nào, nó chỉ được gắn với lớp
- Nên gọi phương thức static từ tên lớp.
- Phương thức static có thể được gọi mà không cần có đối tượng nào của lớp đó đang ở trong heap.
- Do không được gắn với một đối tượng nào, phương thức static không thể truy nhập biến thực thể hay các phương thức thực thể.
- Biến lớp hay còn gọi là biến static là biến dùng chung cho tất cả các đối tượng của lớp. Chỉ có duy nhất một bản cho cả lớp, chứ không phải mỗi đối tượng có một bản.
- Phương thức static có thể truy nhập biến static.

Nguồn tham khảo

Giáo trình Lập trình hướng đối tượng với Java (2010) Trường Đại học Công nghệ - ĐHQGHN

OOP Java Static Basic Android

Bài viết liên quan

Tự học Automation Testing Cơ Bản với Selenium - Tại...
Tran Thi Tra Giang
9 phút đọc
3314 2 1 0

Cách vận dụng Singleton pattern p2: Singleton trong...
Nguyen Manh Tien
3 phút đọc
1475 1 0 0

Dagger2: những điều cần biết trước khi implement
Nguyen Van Mui
5 phút đọc
864 0 0 0

C言語の文法
Tetsu Kasugai
0 phút đọc
288 0

Magic method và Class aliases trong PHP
Dung Nguyen Quang
11 phút đọc
2098 7 0 11

Multithreading: Java Synchronized Blocks
Cùi Bắp
4 phút đọc
3397 4 0 1

Design Patterns - Singleton pattern
Chương
3 phút đọc
12643 2 0 3

Kiểu biến và kiểu dữ liệu trong Java
Nguyen Van Hung
1 phút đọc
601 2 2 1

Bài viết khác từ Trần Quang Huy

Builder Design Pattern
Trần Quang Huy
5 phút đọc
6484 3 2 7

Ngăn xếp
Trần Quang Huy
6 phút đọc
362 4 1 8

[Một số hướng giải quyết cho bài toán tìm kiếm]

Trần Quang Huy
9 phút đọc
797 3 5 4

[Data Class hay Builder Design Pattern?]

Trần Quang Huy
7 phút đọc
315 2 0 7

[State Design Pattern]

Trần Quang Huy
6 phút đọc
4598 1 3 5

[Data Class trong Kotlin và cách giải quyết cùng bài toán của Builder Pattern]

Trần Quang Huy
7 phút đọc
3847 3 7 6

Bình luận

Đăng nhập để bình luận

Đoàn Phương @phuongtdoan
thg 8 4, 8:50 SA

hay anh

0 | Trả lời Chia sẻ •••

Jarvis Nguyễn @johanforce
thg 8 5, 4:18 CH

Bài viết của bạn rất hay, đi sâu chi tiết về vấn đề mình đang tìm kiếm. Mong bạn sẽ tiếp tục cho ra thêm nhiều bài viết chất lượng hơn nữa :3

0 | Trả lời Chia sẻ •••