Java `static` – Sổ tay đầy đủ (kèm ví dụ dễ hiểu)

> Mục tiêu: nắm vững mọi khía cạnh về `static` trong Java: biến, phương thức, khối khởi tạo, lớp lồng nhau, import tĩnh, kế thừa/đa hình, generic, luồng, bộ nhớ, best practice và lỗi thường gặp — kèm ví dụ ngắn, rõ ràng.

---

1) Khái niệm cốt lõi

- `static` nghĩa là **thuộc về lớp (class)**, không gắn với từng đối tượng (object).
- Thành viên `static` (biến/hàm/khối/lớp lồng) tồn tại **một bản duy nhất** cho mỗi *class loader* trong suốt vòng đời lớp.
- Gọi/đụng đến qua **tên lớp** thay vì qua đối tượng.

Ví dụ tổng quan:

```
class Util {
    static int counter = 0;        // biến tĩnh
    static void tick() { counter++; } // phương thức tĩnh
}

public class Main {
    public static void main(String[] args) {
        Util.tick();
        System.out.println(Util.counter); // 1
    }
}
```

---

2) `static variable` (biến tĩnh)

Tính chất - Dùng chung cho **mọi object** của lớp. - Khởi tạo khi lớp được **initialize** lần đầu; sống đến khi lớp bị dỡ (thường là hết chương trình). - Mặc định về **phạm vi truy cập** (public/private/…) y như biến thường.

Ví dụ 1 – Đếm số instance:

```
class Student {
    static int count = 0; // dùng chung
    Student() { count++; }
}

class Demo {
    public static void main(String[] args) {
        new Student(); new Student(); new Student();
        System.out.println(Student.count); // 3
```

```
    }
}
```

Ví dụ 2 – So sánh `static` vs non-static:

```
class Student {
    static int staticCount = 0; // chung
    int normalCount = 0;        // riêng từng object
    Student() { staticCount++; normalCount++; }
}

class Demo {
    public static void main(String[] args) {
        Student a = new Student();
        Student b = new Student();
        System.out.println(Student.staticCount); // 2
        System.out.println(a.normalCount);       // 1
        System.out.println(b.normalCount);       // 1
    }
}
```

Ví dụ 3 – Hằng số & "inline" (compile-time constant):

```
// A.java
public class A { public static final int VERSION = 1; }

// B.java
class B { int v = A.VERSION; } // 1 được "inline" vào B khi biên dịch
```

> Lưu ý: Với **hằng số biên dịch** (primitive/String, `final` + biểu thức hằng), JVM có thể **chèn trực tiếp** giá trị vào lớp dùng. Nếu đổi `A.VERSION` sang 2 nhưng **không biên dịch lại B**, B vẫn có thể thấy 1.

Lưu ý quan trọng - `static` **không** tự động "tiết kiệm bộ nhớ" cho phương thức; nó chỉ giúp **không cần tạo object** để gọi. - Biến `static` **không được tuần tự hóa** (serialization) theo object. - Dùng `static` cho **trạng thái toàn cục** dễ gây **khó test** và **race condition** nếu đa luồng.

---

3) `static method` (phương thức tĩnh)

Tính chất - Gọi qua **tên lớp**; không cần object. - **Không thể** truy cập trực tiếp biến/phương thức **non-static** (vì thiếu `this`). - Có thể **overload**, nhưng **không override** — chỉ **hide** trong kế thừa (xem §7). - Ví dụ điển hình: `main`, hàm tiện ích (utility), factory.

Ví dụ – Quy tắc truy cập:

```
class Foo {
    int x = 10;
    static int y = 20;
```

```
    static void s() {
        // System.out.println(x); // LỖI: không có this trong ngữ cảnh static
        System.out.println(y);    // OK
    }

    void i() {
        System.out.println(x); // OK
        System.out.println(y); // OK
    }
}
```

Ví dụ – `main` bắt buộc là static:

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

---

4) `static initializer` (khối khởi tạo tĩnh)

Tính chất - Chạy **một lần** khi lớp được **initialize** (lần đầu "được dùng" tích cực: gọi method static, new object, truy cập field không phải hằng số, v.v.). - Có thể có **nhiều khối**, chạy theo **thứ tự xuất hiện** trong mã. - Nếu ném exception → `ExceptionInInitializerError` và lớp coi như hỏng.

Ví dụ – Thứ tự khởi tạo:

```
class Config {
    static int a = initA();
    static { System.out.println("block #1"); }
    static int b = initB();
    static { System.out.println("block #2"); }

    static int initA(){ System.out.println("initA"); return 1; }
    static int initB(){ System.out.println("initB"); return 2; }
}

class Demo {
    public static void main(String[] args) {
        System.out.println(Config.a + Config.b);
        // In:
        // initA
        // block #1
        // initB
        // block #2
        // 3
```

```
    }
}
```

Khi nào lớp được initialize? - Gọi `new`, gọi method `static`, truy cập/ghi field `static` **không phải hằng số biên dịch**, dùng reflection, subclass được init, v.v. - **Truy cập hằng số biên dịch** (static final primitive/String) **không** kích hoạt init.

---

5) Lớp lồng nhau `static` (static nested class)

Tính chất - Là lớp lồng **không** mang tham chiếu ngầm tới outer (`Outer.this`). - Chỉ truy cập trực tiếp **thành viên static** của outer. Muốn đụng non-static → cần instance của outer. - Hữu ích cho nhóm logic/phụ trợ gắn chặt với outer nhưng **không cần** outer instance.

Ví dụ:

```
class Outer {
    static int sx = 1; int ix = 2;

    static class Helper {
        void run() {
            System.out.println(sx); // OK
            // System.out.println(ix); // LỖI
        }
    }
}

class Demo {
    public static void main(String[] args) {
        Outer.Helper h = new Outer.Helper(); // không cần Outer object
        h.run();
    }
}
```

> Ghi nhớ: **Top-level class** không thể là `static`; chỉ lớp **lồng trong** mới dùng `static` được.

---

6) `static import`

Cho phép dùng trực tiếp **tên thành viên static** mà không cần tiền tố lớp.

Ví dụ:

```
import static java.lang.Math.*; // dùng PI, sqrt, pow,... trực tiếp

class Demo {
    public static void main(String[] args) {
```

```
        double r = sqrt(4); // thay vì Math.sqrt(4)
        System.out.println(PI);
    }
}
```

Dùng tiết kiệm để **tránh mơ hồ** tên hàm/hằng số.

---

7) Kế thừa: `static` **hide** chứ không **override**

Nguyên tắc - Phương thức `static` ở subclass **che khuất** (hide) phương thức `static` cùng chữ ký ở superclass. - Gọi qua **tên lớp tham chiếu** quyết định phiên bản nào được dùng (binding **tĩnh**, không đa hình).

Ví dụ:

```
class A { static void who() { System.out.println("A"); } }
class B extends A { static void who() { System.out.println("B"); } }

class Demo {
    public static void main(String[] args) {
        A.who(); // A
        B.who(); // B
        A ref = new B();
        ref.who(); // A  (không đa hình!)
    }
}
```

Với field `static`: cũng là **hide** nếu trùng tên, không đa hình.

---

8) `static` & Generics

- Biến `static` **không** được dùng kiểu tham số của lớp:

```
class Box<T> {
    // static T shared; // LỖI: T gắn với instance, static gắn với class
}
```

- Phương thức `static` có thể có **type parameter riêng**:

```
class Util {
    public static <T> T first(java.util.List<T> list) { return
list.isEmpty()? null : list.get(0); }
}
```

---

9) Đa luồng & an toàn khi dùng `static`

- Biến `static` là **trạng thái dùng chung** → cần đồng bộ hóa khi có ghi/đọc từ nhiều luồng.

- Công cụ: `synchronized`, `volatile`, `Atomic*`, `ConcurrentHashMap`.

Ví dụ – Đếm an toàn:

```
import java.util.concurrent.atomic.AtomicInteger;

class Counter { static final AtomicInteger c = new AtomicInteger(); }

class Demo {
    public static void main(String[] args) {
        Counter.c.incrementAndGet();
    }
}
```

Idiom – Khởi tạo lười (thread-safe) bằng Holder:

```
class Singleton {
    private Singleton() {}
    private static class Holder { static final Singleton I = new Singleton();
}
    public static Singleton get() { return Holder.I; }
}
```

Khuyến nghị - Tránh “biến toàn cục” không cần thiết → khó test, dễ bug. - Đối với config, cache… hãy cân nhắc DI (Dependency Injection) thay vì `static` cứng.

---

10) Bộ nhớ & vòng đời

- Mã lớp/field/method `static` nằm ở **metaspace/method area**; instance nằm ở **heap**.
- `static` sống theo **vòng đời của lớp (class loader)**, không bị GC như object bình thường cho đến khi lớp được dỡ.
- `static` **không làm phương thức “nhẹ” hơn**, chỉ giúp **tránh tạo object** để gọi.

Ví dụ – Tránh new vô ích:

```
class Parser {
    static int parseDigit(char c){ return c - '0'; }
}

class Demo {
    public static void main(String[] args) {
        // Dùng trực tiếp, không cần new Parser()
        System.out.println(Parser.parseDigit('7'));
    }
}
```

---

11) Interface & `static`

- **Field trong interface** mặc định là `public static final`.
- Từ Java 8, **phương thức `static` trong interface** được phép (không thừa kế vào class implement).

Ví dụ:

```
interface MathX {
    double E = 2.718281828; // public static final mặc định
    static int twice(int x) { return 2 * x; }
}

class Demo {
    public static void main(String[] args) {
        System.out.println(MathX.E);
        System.out.println(MathX.twice(5)); // gọi qua tên interface
    }
}
```

---

12) Best practices

- Dùng `static` cho:
  - Hằng số (`public static final`) – đặt TÊN_IN_HOA.
  - Hàm tiện ích không cần trạng thái (`Collections`, `Math`).
  - Bộ đếm/sổ đăng ký chung, cache cục bộ (cẩn trọng thread-safe).
  - Factory method, singleton (ưu tiên **enum singleton** hoặc **Holder** idiom).
- Hạn chế:
  - Tránh **trạng thái toàn cục** khó kiểm soát.
  - Tránh gọi `static` bừa bãi trong code business → khó test/mocking.
  - Tránh `static import *` gây mơ hồ.

---

13) Lỗi thường gặp

- `Cannot make a static reference to the non-static field/method` → bạn đang gọi thành viên non-static từ ngữ cảnh static mà không có object.
- `non-static variable this cannot be referenced from a static context` → không có `this` trong static.
- `Illegal static declaration in inner class` → lớp **inner** (non-static) **không được** có thành viên static (trừ hằng số `static final` compile-time).
- `ExceptionInInitializerError` → lỗi trong khối static init.

Ví dụ lỗi & sửa:

```java
class A {
    int x = 10;
    static void bad() {
        // System.out.println(x); // LỖI
        A obj = new A();
        System.out.println(obj.x); // OK: có object
    }
}
```

---

14) Mini bài tập kiểm tra hiểu bài

1) Viết lớp `Bank` với `static double rate`, tạo 2 bank và chứng minh đổi `rate` một nơi ảnh hưởng cả hai.
2) Tạo `static` block in ra thứ tự init của 3 biến tĩnh xen kẽ 2 khối tĩnh.
3) Viết `static <T> T last(List<T>)` và test với `List<Integer>` & `List<String>`.
4) Tạo `class A` và `class B extends A` mỗi lớp có `static void who()`. Gọi qua biến tham chiếu kiểu `A` trỏ tới `new B()` để thấy **hide**.

---

15) Tóm tắt nhớ nhanh

- `static` = thuộc **class**, một bản duy nhất.
- `static variable` = **dùng chung**; cẩn trọng đa luồng.
- `static method` = **không có `this`**, không override, **hide** trong kế thừa.
- `static block` = chạy **một lần** khi lớp init.
- `static nested class` = lớp lồng **không** mang `Outer.this`.
- Interface: field `public static final`, method `static` gọi qua tên interface.
- Dùng `static` đúng chỗ: **hằng số, tiện ích, holder/singleton**; tránh **state toàn cục**.