Session 07:

TỪ KHÓA STATIC, FINAL

NỘI DUNG

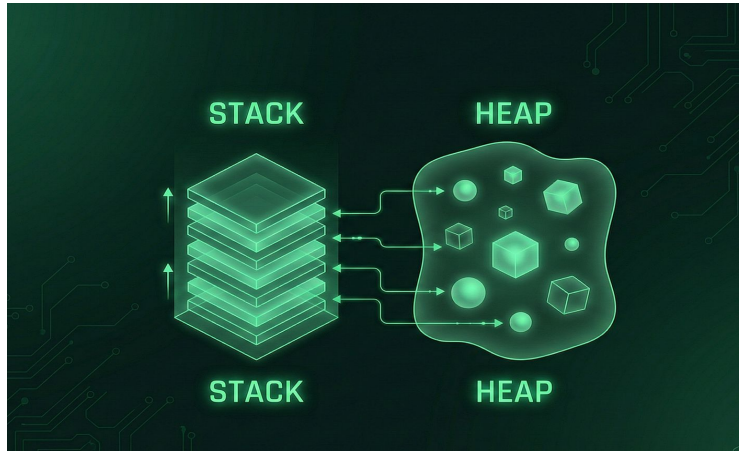
1. Biến kiểu dữ liệu nguyên thủy
2. Biến kiểu dữ liệu tham chiếu
3. Biến của lớp, biến của đối tượng
4. Phương thức của lớp, phương thức của đối tượng
5. Cú pháp khai báo package
6. Từ khóa final

1. Biến kiểu dữ liệu nguyên thủy

Biến kiểu dữ liệu nguyên thủy là gì?

- Là các kiểu dữ liệu cơ bản, được định nghĩa sẵn trong Java.
- **8 kiểu nguyên thủy:**
  - Số nguyên: byte, short, int, long
  - Số thực: float, double
  - Ký tự: char
  - Logic: boolean
- **Đặc điểm:**
  - Lưu trữ trực tiếp giá trị trong bộ nhớ Stack.
  - Kích thước cố định.
  - Không phải là đối tượng, không có phương thức.

```
1  int age = 20;
2  double salary = 1000.5;
3  char grade = 'A';
4  boolean isPassed = true;
```



1. Biến kiểu dữ liệu nguyên thủy

Giá trị mặc định & Phạm vi lưu trữ của biến nguyên thủy

- **Giá trị mặc định** khi khai báo (trong class, không gán giá trị):
  - byte, short, int, long: 0
  - float, double: 0.0
  - char: '\u0000' (ký tự rỗng)
  - boolean: false
- **Lưu ý:** Biến cục bộ (trong phương thức) bắt buộc phải khởi tạo trước khi sử dụng.
- **Phạm vi lưu trữ:** Biến nguyên thủy lưu trực tiếp giá trị trong bộ nhớ Stack.

2. Biến Kiểu Dữ Liệu Tham Chiếu

Biến kiểu dữ liệu tham chiếu (Reference Data Types)

- Là biến lưu trữ **địa chỉ tham chiếu (reference)** đến một đối tượng trong bộ nhớ Heap.
- **Bao gồm:**
  - Các class (do người dùng định nghĩa, hoặc class thư viện như String, Scanner).
  - Các interface, array.
- **Đặc điểm:**
  - Biến tham chiếu lưu trong Stack, đối tượng thực tế lưu trong Heap.
  - Có thể tham chiếu đến null (không trỏ đến đối tượng nào).
  - Có thể gọi phương thức, truy cập thuộc tính của đối tượng thông qua biến tham chiếu.

```
String name = "John"; // name là biến tham chiếu, trỏ đến đối tượng String
Account acc = new Account(); // acc tham chiếu đến đối tượng Account
int[] numbers = new int[5]; // numbers tham chiếu đến mảng
```

2. Biến Kiểu Dữ Liệu Tham Chiếu

So sánh biến nguyên thủy và tham chiếu

Đặc điểm	Biến nguyên thủy (Primitive)	Biến tham chiếu (Reference)
Lưu trữ	Giá trị trực tiếp	Địa chỉ tham chiếu đến đối tượng
Vị trí bộ nhớ	Stack	Stack (biến) & Heap (đối tượng)
Kích thước	Cố định (theo kiểu dữ liệu)	Không cố định (phụ thuộc đối tượng)
Giá trị mặc định	Có (trong class)	null
Gán giá trị	Sao chép giá trị	Sao chép địa chỉ (2 biến có thể cùng trỏ 1 đối tượng)
Ví dụ	int a = 5;	Account acc = new Account();

3. Biến Của Lớp, Biến Của Đối Tượng

Biến của đối tượng (Instance Variable)

- Là biến **không** có từ khóa **static**.
- **Thuộc về từng đối tượng riêng biệt.** Mỗi đối tượng có một bản sao riêng của biến instance.
- Được khởi tạo khi đối tượng được tạo (gọi constructor).
- Truy cập thông qua đối tượng: **objectName.variableName**

```java
public class Student {
    // Instance variables
    private int id;
    private String name;
    // Mỗi Student sẽ có id và name riêng
}
```

3. Biến Của Lớp, Biến Của Đối Tượng

Biến của lớp (Static Variable)

- Là biến có từ khóa static.
- **Thuộc về lớp**, được chia sẻ bởi **tất cả các đối tượng** của lớp đó.
- Chỉ có **một bản sao duy nhất** trong bộ nhớ (vùng Class Area), tiết kiệm bộ nhớ.
- Được khởi tạo khi lớp được tải (load) vào bộ nhớ (trước khi tạo bất kỳ đối tượng nào).
- Truy cập thông qua tên lớp: ClassName.variableName (khuyến khích) hoặc qua đối tượng.
- **Dùng khi nào?** Khi m ụ: tên trường, quỹ lớp, biến đếm).

```java
public class Student {
    // Static variable - thuộc về lớp Student
    public static String schoolName = "Rikkei Academy";
    public static int totalStudents = 0;

    // Instance variables
    private int id;
    private String name;
}
```

3. Biến Của Lớp, Biến Của Đối Tượng

Ví dụ: Biến static và instance trong class Student

```java
public class Student {
    // Static variables
    public static String className = "RK13";
    public static int count = 0;
    public static int fund = 0;

    // Instance variables
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
        count++; // Tăng biến static mỗi khi tạo đối tượng mới
    }

    // Getter, setter, toString...
}
```

```java
public class Demo {
    public static void main(String[] args) {
        Student st1 = new Student(1, "Nam");
        Student st2 = new Student(2, "Lan");

        // Truy cập static variable qua tên lớp
        System.out.println("Lớp học: " + Student.className); // RK13
        System.out.println("Số sinh viên: " + Student.count); // 2

        // Thay đổi static variable
        Student.className = "RK13_Advanced";
        System.out.println("Lớp của st1: " + st1.className); // RK13_Advanced
        System.out.println("Lớp của st2: " + st2.className); // RK13_Advanced

        // Đóng quỹ lớp
        st1.fund += 100; // Không nên truy cập static qua instance
        Student.fund += 200;
        System.out.println("Quỹ lớp: " + Student.fund); // 300
    }
}
```

4. Phương Thức Của Lớp, Phương Thức Của Đối Tượng

Phương thức của đối tượng (Instance Method)

- Là phương thức **không** có từ khóa static.
- **Thuộc về từng đối tượng**, có thể truy cập và thao tác trên dữ liệu của đối tượng đó.
- Có thể truy cập **cả biến instance và biến static**.
- Phải được gọi thông

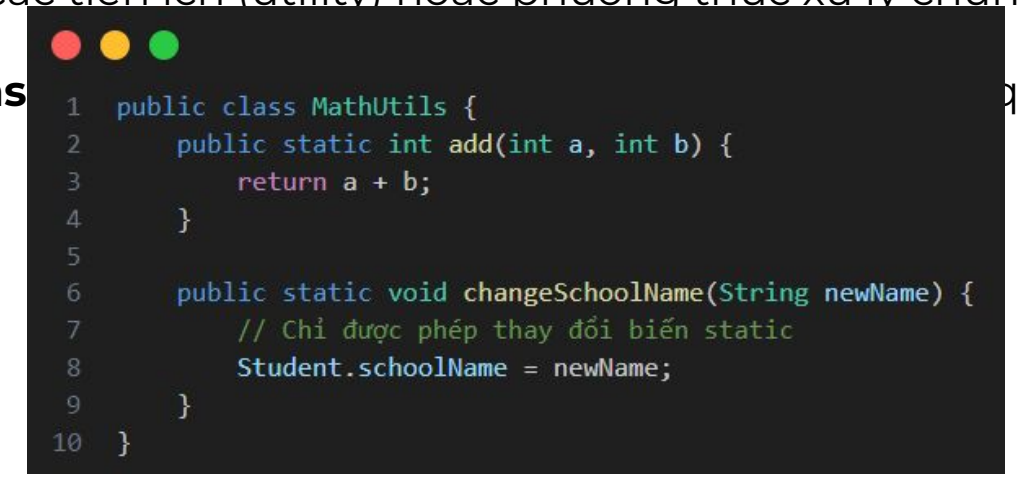
```java
public class Student {
    private String name;

    public void displayInfo() {
        // Truy cập biến instance
        System.out.println("Tên: " + this.name);
        // Truy cập biến static
        System.out.println("Lớp: " + className);
    }
}
```

4. Phương Thức Của Lớp, Phương Thức Của Đối Tượng

Phương thức của lớp (Static Method)

- Là phương thức có từ khóa static.
- **Thuộc về lớp**, có thể gọi mà **không cần tạo đối tượng**.
- **Chỉ có thể truy cập trực tiếp đến các biến static và gọi các phương thức static khác.**
- **Không thể** sử dụng từ khóa this (vì không thuộc đối tượng cụ thể).
- Thường dùng cho các tiện ích (utility) hoặc phương thức xử lý chung.
- Gọi qua tên lớp: **Clas** qua đối tượng.



```java
public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }

    public static void changeSchoolName(String newName) {
        // Chỉ được phép thay đổi biến static
        Student.schoolName = newName;
    }
}
```

4. Phương Thức Của Lớp, Phương Thức Của Đối Tượng

Hạn chế của phương thức static

- **Không thể truy cập trực tiếp biến instance (non-static)** hoặc gọi phương thức instance.
- **Ví dụ sai:**

```java
public class Example {
    private int id; // instance variable

    public static void staticMethod() {
        System.out.println(id); // LỖI: Cannot make a static reference to non-static field
        instanceMethod(); // LỖI: Cannot make a static reference to non-static method
    }

    public void instanceMethod() { }
}
```

Vì phương thức static có thể được gọi khi chưa có đối tượng nào tồn tại, nên không thể biết id của đối tượng nào để truy cập.

4. Phương Thức Của Lớp, Phương Thức Của Đối Tượng

Ví dụ: Phương thức static trong class Student

Nội dung chính (Code):

```java
public class Student {
    private int id;
    private String name;
    public static String schoolName = "Rikkei";

    // Instance method
    public void showInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", School: " + schoolName);
    }

    // Static method
    public static void changeSchool(String newSchool) {
        schoolName = newSchool;
        // System.out.println(id); // Lỗi! Không thể truy cập biến instance
    }

    // Static method dùng để tạo đối tượng với logic đặc biệt
    public static Student createStudent(int id, String name) {
        Student st = new Student();
        st.id = id;
        st.name = name;
        return st;
    }
}
```

```java
public class Demo {
    public static void main(String[] args) {
        // Gọi static method không cần đối tượng
        Student.changeSchool("Rikkei Academy Advanced");

        // Tạo đối tượng bằng static method
        Student st = Student.createStudent(1, "Nam");
        st.showInfo();
    }
}
```

5. Cú pháp khai báo package

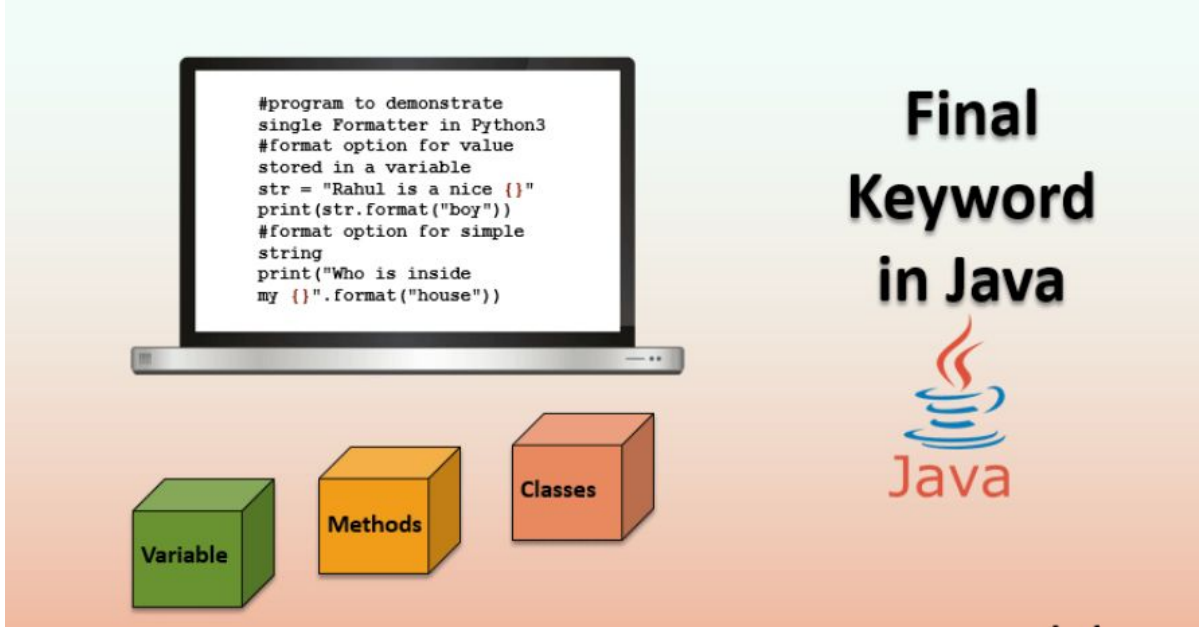
Package trong Java

- Nhóm các lớp liên quan vào cùng một thư mục.
- **Mục đích:** Quản lý code, tránh xung đột tên, dễ tìm kiếm.
- Trong Java, tên package phải **viết thường hoàn toàn** và được đặt theo **quy ước domain đảo ngược** để tránh trùng lặp. Cú pháp chung là <**domain đảo ngược**>.<**tên dự án**>.<**chức năng**>. Package thường được chia theo **chức năng hoặc tầng kiến trúc** như controller, service, repository, entity. Không nên dùng chữ hoa, ký tự đặc biệt hoặc tên không có ý nghĩa.
- Ví dụ: com.rikkei.ac;

6. Từ khóa final

Từ khóa final là gì?

- Dùng để **hạn chế/hạn chế** người dùng, đánh dấu một thứ gì đó là "không thể thay đổi".
- Có thể áp dụng cho:
  - **Biến (variable):** Biến trở thành hằng số (constant), không thể thay đổi giá trị.
  - **Phương thức (method):** Không thể ghi đè (override) trong lớp con.
  - **Lớp (class):** Không thể kế thừa (extends).



6. Từ khóa final

Biến final (Constant)

- Khi khai báo biến là final, biến đó **không thể thay đổi giá trị** sau khi đã được gán.
- **Quy tắc:**
  - Phải được khởi tạo giá trị (có thể khởi tạo ngay khi khai báo, hoặc trong constructor nếu là biến instance).
  - Nếu là biến static final, thường được dùng làm **hằng số** và đặt tên theo quy ước UPPER_CASE.

```java
public class Constants {
    // Khai báo và khởi tạo ngay
    public static final double PI = 3.14159;
    public static final String COMPANY_NAME = "Rikkei Academy";

    // Biến final (không static) có thể khởi tạo trong constructor
    private final int id;

    public Constants(int id) {
        this.id = id; // Chỉ được gán giá trị một lần
    }

    // this.id = 100; // Lỗi! Không thể thay đổi
}
```

6. Từ khóa final

Phương thức final

- Phương thức được khai báo với final **không thể bị ghi đè (override)** bởi lớp con.
- **Mục đích:** Đảm bảo hành vi của phương thức không bị thay đổi trong các lớp kế thừa.
- **Ví dụ:**

```java
public class Vehicle {
    // Phương thức final, không thể override
    public final void startEngine() {
        System.out.println("Engine started");
    }
}

public class Car extends Vehicle {
    // @Override
    // public void startEngine() { } // Lỗi! Không thể override phương thức final
}
```

6. Từ khóa final

Lớp final

- Lớp được khai báo với final **không thể được kế thừa** (không thể có lớp con).
- **Mục đích:** Ngăn chặn việc mở rộng lớp, đảm bảo tính toàn vẹn của lớp (ví dụ: lớp String trong Java là final).
- **Ví dụ:**

```java
public final class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }
}

// public class AdvancedMath extends MathUtils { } // Lỗi! Không thể kế thừa lớp final
```

TỔNG KẾT

- Phân biệt được biến nguyên thủy và biến tham chiếu, hiểu cách lưu trữ và đặc điểm của từng loại.
- Hiểu và sử dụng được từ khóa static cho biến và phương thức trong lớp.
- Nắm được đặc điểm của biến static (dùng chung cho mọi đối tượng) và phương thức static (gọi không cần đối tượng).
- Hiểu và sử dụng được từ khóa final cho biến, phương thức và lớp.
- Nắm được ý nghĩa của biến final, phương thức final và lớp final trong thiết kế chương trình.
- Hiểu khái niệm package, cú pháp khai báo, import và quy ước đặt tên package trong Java.

RIKEI Academy®
where the dream begins

KẾT THÚC

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN