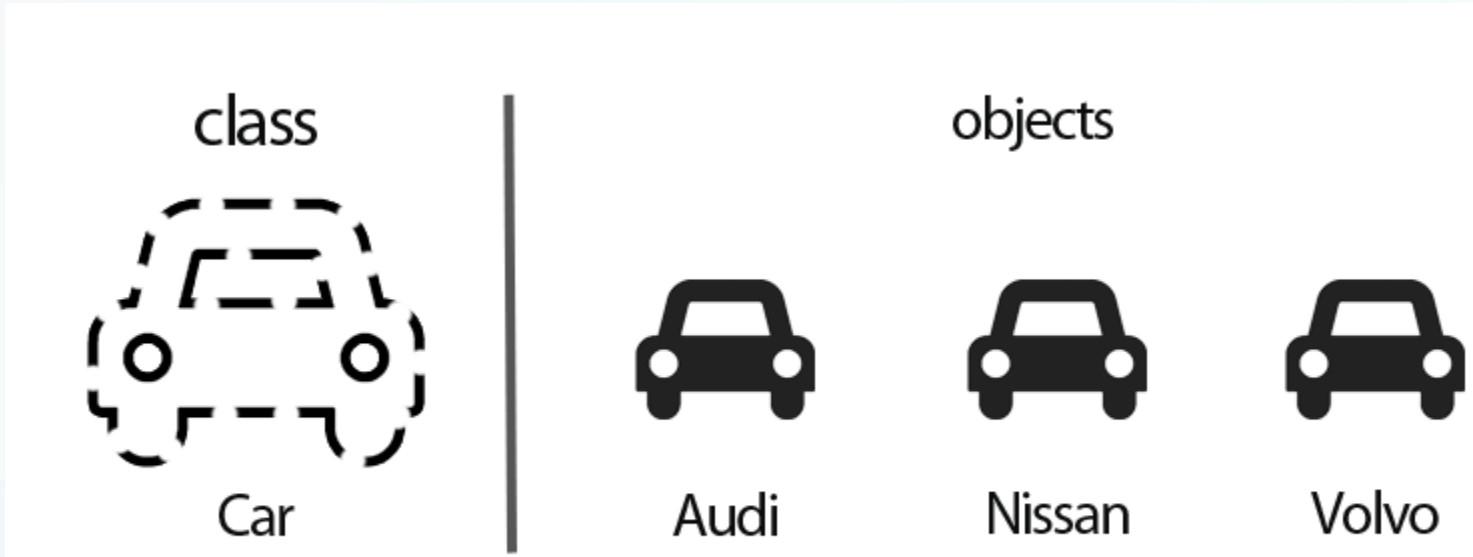
FUNiX
Learn with Mentors

Được tạo bởi Nguyễn Quang Vinh (vinhnq1)
Lần chỉnh sửa cuối - 01/02/2025

Chương 3
Lớp và Đối tượng



Định nghĩa Lớp (class)

- Lớp (Class) là một bản thiết kế (blueprint) để tạo ra các đối tượng. Một lớp bao gồm các thuộc tính (biến) và phương thức (hàm) để thao tác dữ liệu.

```
<access specifier> class class_name {
    // Thuộc tính (biến thành viên)
    <access specifier> <data type> variable;

    // Phương thức (hàm thành viên)
    <access specifier> <return type> method(parameter_list) {
        // Thân phương thức
    }
}
```

- **<access specifier>**: Xác định mức độ truy cập (public, private, protected, default).
- **class_name**: Tên của lớp.
- **variable**: Các biến thành viên của lớp.
- **method**: Phương thức của lớp.

Ví dụ với Lớp (class)

```
public class Student {
    // Thuộc tính
    private String id;
    private String name;
    public float mark;

    // Phương thức
    public void setInfo(String id, String name, float mark) {
        this.id = id;
        this.name = name;
        this.mark = mark;
    }

    public void displayInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", Mark: " + mark);
    }
}
```

Khai báo đối tượng (Object)

- Đối tượng (Object) là một thể hiện cụ thể của một lớp. Để tạo đối tượng, ta sử dụng từ khóa new.

Cú pháp:

```
ClassName objectName = new ClassName();
```

Ví dụ:

```
Student student1 = new Student();
```

Truy xuất các thành phần của đối tượng

Truy xuất thuộc tính:

```
<object_name>.<variable>
```

Truy xuất phương thức:

```
<object_name>.<method>(parameter_list)
```

Ví dụ:

```
Student student1 = new Student();
student1.setInfo("12345", "Alice", 8.5f);
student1.displayInfo();
```

Bài tập 1 - Tạo lớp Person

- Tạo lớp Person với các thuộc tính: **`name`** (tên), **`age`** (tuổi), **`gender`** (giới tính).
- Thêm phương thức **`displayInfo()`** để hiển thị thông tin của đối tượng.
- Tạo đối tượng từ lớp **`Person`** và gọi phương thức **`displayInfo()`**.

Access Specifiers (Phạm vi truy cập)

- `public`: Có thể truy cập từ bất kỳ đâu.
- `private`: Chỉ có thể truy cập từ bên trong lớp.
- `protected`: Có thể truy cập từ bên trong lớp và các lớp con.
- `default` (không khai báo): Có thể truy cập từ các lớp trong cùng package.

Bài tập 2 - Access Specifiers

- Tạo lớp **`BankAccount`** với các thuộc tính:
  - **`accountNumber`** (số tài khoản, **`private`**).
  - **`balance`** (số dư, **`private`**).
- Thêm phương thức **`deposit()`** để nạp tiền và **`withdraw()`** để rút tiền.
- Sử dụng **`getter`** và **`setter`** để truy cập các thuộc tính **`private`**.

Phương thức thiết lập (Constructor)

- **Constructor** là một phương thức đặc biệt được gọi tự động khi tạo đối tượng. Nó dùng để khởi tạo các thuộc tính của đối tượng.
- **Đặc điểm**:
  - Tên constructor trùng với tên lớp.
  - Không có kiểu trả về.
  - Có thể có tham số hoặc không.
- **Các loại constructor**:
  - **Constructor mặc định**: Không có tham số.
  - **Constructor có tham số**: Khởi tạo đối tượng với các giá trị cụ thể.
  - **Constructor sao chép**: Tạo đối tượng mới từ đối tượng hiện có.

Phương thức thiết lập (Constructor) – Ví dụ

```
public class Student {
    private string _id;
    private string _name;
    private float _mark;

    // Constructor mặc định
    public Student() {
        _id = "63131234";
        _name = "Nguyen Thi Lan";
        _mark = 7.2f;
    }

    // Constructor có tham số
    public Student(string id, string name,
float mark) {
        _id = id;
        _name = name;
        _mark = mark;
    }

    // Constructor sao chép
    public Student(Student other) {
        _id = other._id;
        _name = other._name;
        _mark = other._mark;
    }
}
```

Bài tập 3 - Constructor

- Tạo lớp **`Car`** với các thuộc tính: **`brand`** (hãng xe), **`model`** (mẫu xe), **`year`** (năm sản xuất).
- Thêm các constructor:
  - Constructor mặc định.
  - Constructor có tham số để khởi tạo các thuộc tính.
  - Constructor sao chép.
- Tạo đối tượng từ lớp **`Car`** và hiển thị thông tin.

Phương thức hủy bỏ (Destructor)

- Trong Java, không có destructor rõ ràng như trong C++. Thay vào đó, Java sử dụng cơ chế **Garbage Collection** để tự động giải phóng bộ nhớ.
- **Phương thức `finalize()`**: Được gọi trước khi đối tượng bị hủy bỏ.

```java
@Override
protected void finalize() throws Throwable {
    System.out.println("Object is being deleted");
    super.finalize();
}
```

Bài tập 4 - Phương thức hủy bỏ

- Tạo lớp **`Book`** với các thuộc tính: **`title`** (tiêu đề), **`author`** (tác giả), **`year`** (năm xuất bản).
- Thêm phương thức **`finalize()`** để in ra thông báo khi đối tượng bị hủy.
- Tạo đối tượng từ lớp **`Book`** và thử nghiệm phương thức **`finalize()`**.

Con trỏ this

- `this` là một tham chiếu đến đối tượng hiện tại của lớp.
- **Công dụng:**
  - Phân biệt giữa thuộc tính của lớp và tham số của phương thức.
  - Gọi các phương thức hoặc constructor khác trong cùng lớp.

```
public class Student {
    private String name;

    public void setName(String name) {
        this.name = name;  // this.name là thuộc tính của lớp, name là tham số
    }
}
```

Bài tập 5 - Con trỏ this

- Tạo lớp **`Employee`** với các thuộc tính: **`id`** (mã nhân viên), **`name`** (tên), **`salary`** (lương).
- Sử dụng từ khóa this trong phương thức **`setSalary()`** để gán giá trị cho thuộc tính **`salary`**.
- Tạo đối tượng từ lớp **`Employee`** và gọi phương thức **`setSalary()`**.

Thành phần tĩnh (Static)

- Thành phần tĩnh (biến hoặc phương thức) thuộc về lớp chứ không phải đối tượng cụ thể.

Cú pháp:

```
public class ClassName {
    static <data_type> variableName;
    static <return_type> methodName(parameter_list) {
        // Thân phương thức
    }
}
```

Ví dụ:

```
public class Student {
    static int count = 0;  // Biến tĩnh
    public Student() {
        count++;  // Tăng biến đếm mỗi khi tạo đối tượng
    }
    public static void printCount() {  // Phương thức tĩnh
        System.out.println("Total students: " + count);
    }
}
```

Bài tập 6 - Thành phần tĩnh

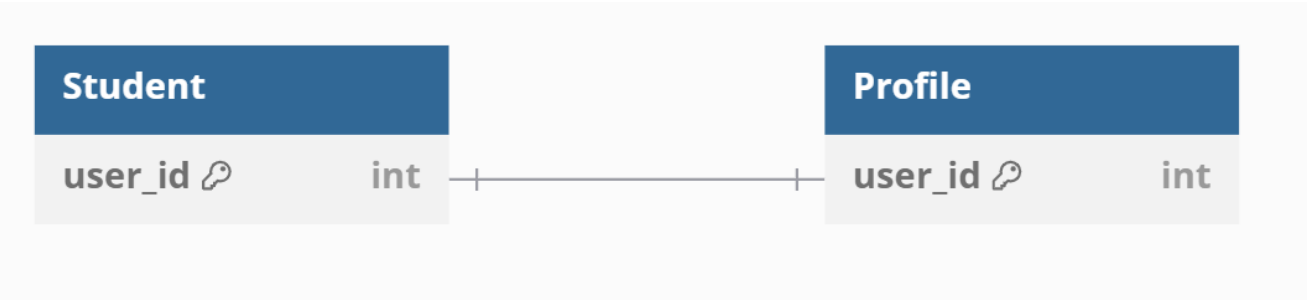
- Tạo lớp **`Counter`** với một biến tĩnh **`count`** để đếm số lượng đối tượng được tạo ra.
- Thêm phương thức tĩnh **`getCount()`** để trả về giá trị của **`count`**.
- Tạo nhiều đối tượng từ lớp **`Counter`** và kiểm tra giá trị của **`count`**.

Quan hệ 1-1

- Một đối tượng của lớp A liên kết với một đối tượng của lớp B.

```
class Student {
    private Profile profile;   // Mỗi sinh viên có một hồ sơ
}

class Profile {
    private Student student;   // Mỗi hồ sơ thuộc về một sinh viên
}
```



Bài tập 7 - Quan hệ 1-1

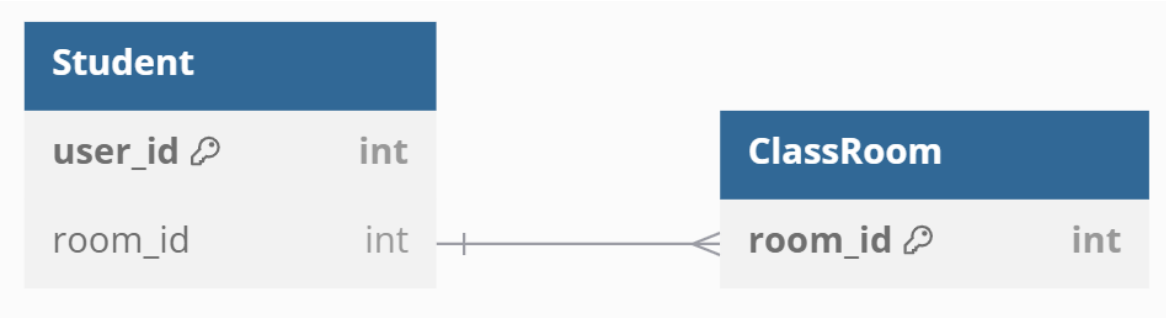
- Tạo lớp **`Student`** và lớp **`StudentProfile`**.
- Mỗi **`Student`** có một **`StudentProfile`** (quan hệ 1-1).
- Thêm phương thức để hiển thị thông tin của **`Student`** và **`StudentProfile`**.

Quan hệ 1-n

- Một đối tượng của lớp A liên kết với nhiều đối tượng của lớp B.

```
class Classroom {
    private Student[] students;   // Một lớp học có nhiều sinh viên
}

class Student {
    private Classroom classroom;   // Mỗi sinh viên thuộc về một lớp học
}
```



Bài tập 8 - Quan hệ 1-n

- Tạo lớp **`Department`** và lớp **`Employee`**.
- Một **`Department`** có nhiều **`Employee`** (quan hệ 1-n).
- Thêm phương thức để hiển thị danh sách nhân viên trong một phòng ban.

Định nghĩa toán tử trên lớp (Operator Overloading)

- Cho phép định nghĩa lại các phép toán (+, -, *, /, ...) cho các đối tượng của lớp.

```
public class Point {
    private int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Định nghĩa toán tử cộng
    public Point add(Point other) {
        return new Point(this.x + other.x, this.y + other.y);
    }
}
```

Bài tập 9 – Toán tử cộng

- Tạo lớp **`ComplexNumber`** để biểu diễn số phức với các thuộc tính: **`real`** (phần thực), **`imaginary`** (phần ảo).
- Định nghĩa toán tử cộng (+) để cộng hai số phức.
- Tạo đối tượng từ lớp **`ComplexNumber`** và thực hiện phép cộng.

Lớp Generics

- **Generics** cho phép bạn viết các lớp, phương thức, và giao diện có thể làm việc với bất kỳ kiểu dữ liệu nào mà không cần phải chỉ định cụ thể kiểu dữ liệu đó.
- **Mục đích**: Tăng tính tái sử dụng mã và an toàn kiểu dữ liệu.

Bài tập 10 – Toán tử so sánh

- Tạo lớp **`Fraction`** để biểu diễn phân số với các thuộc tính: **`numerator`** (tử số), **`denominator`** (mẫu số).
- Định nghĩa toán tử so sánh (==) để so sánh hai phân số.
- Tạo đối tượng từ lớp Fraction và thực hiện phép so sánh.

Khai báo lớp Generics với kiểu T

- Là một kiểu dữ liệu tổng quát, có thể được thay thế bằng bất kỳ kiểu dữ liệu nào khi sử dụng lớp.

```
public class Box<T> {
    private T item;

    public void setItem(T item) {
        this.item = item;
    }

    public T getItem() {
        return item;
    }
}
```

Sử dụng lớp Generics

```
public class Main {
    public static void main(String[] args) {
        Box<String> stringBox = new Box<>();
        stringBox.setItem("Hello, Generics!");
        System.out.println(stringBox.getItem());

        Box<Integer> intBox = new Box<>();
        intBox.setItem(123);
        System.out.println(intBox.getItem());
    }
}
```

Bài tập 11 – Lớp Generics

- Tạo lớp **`Box<T>`** để lưu trữ một phần tử có kiểu dữ liệu bất kỳ.
- Thêm phương thức **`setItem()`** và **`getItem()`** để thiết lập và lấy giá trị của phần tử.
- Tạo đối tượng từ lớp **`Box<T>`** với các kiểu dữ liệu khác nhau (ví dụ: **`String`**, **`Integer`**).

Generics với phương thức

```java
// Định nghĩa
public <T> void printArray(T[] array) {
    for (T element : array) {
        System.out.println(element);
    }
}
```

```java
// Ví dụ
public class Main {
    public static void main(String[] args) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        String[] strArray = {"A", "B", "C"};

        printArray(intArray);
        printArray(strArray);
    }

    public static <T> void printArray(T[] array) {
        for (T element : array) {
            System.out.println(element);
        }
    }
}
```

Bài tập 12 – Phương thức Generics

- Tạo phương thức **`printArray<T>`** để in ra các phần tử của một mảng có kiểu dữ liệu bất kỳ.
- Sử dụng phương thức này để in ra các phần tử của mảng số nguyên và mảng chuỗi.