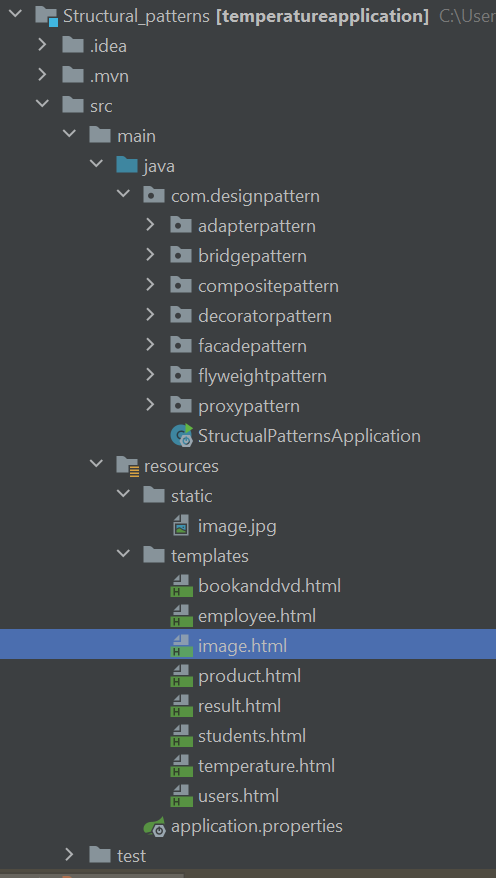
**2. Nhóm cấu trúc (Structural Pattern)**

**Tổ chức thư mục chung**

****

**2.1. Adapter**

**2.1.1. Class Diagram**

****

**2.1.2. Các thành phần chính**

**a. Interface TemperatureSensor**

**A screen shot of a computer program

Description automatically generated with medium confidence**

Đoạn mã trên định nghĩa một giao diện. Giao diện này có một phương thức getTemperature() trả về một giá trị kiểu double.

**b. Interface TemperatureAdapter**

**A screen shot of a computer program

Description automatically generated with low confidence**

Đoạn mã trên định nghĩa một giao diện. Giao diện này có một phương thức getTemperature() trả về một giá trị kiểu double

**c. Class CelsiusSensor**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này triển khai giao diện TemperatureSensor, nghĩa là nó cung cấp một triển khai cụ thể cho phương thức getTemperature() đã được định nghĩa trong giao diện TemperatureSensor.

Trong phương thức getTemperature(), chúng ta giả định rằng giá trị nhiệt độ được lấy từ cảm biến là 23.89 độ Celsius. Do đó, phương thức trả về giá trị này dưới dạng một số thực kiểu double.

Lớp CelsiusSensor có thể được sử dụng để đo nhiệt độ trong đơn vị Celsius và triển khai cách thức cụ thể để lấy giá trị nhiệt độ từ cảm biến nhiệt độ.

**d. Class FahrenheitSensor**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này cũng triển khai giao diện TemperatureSensor, cho phép nó cung cấp một triển khai cụ thể cho phương thức getTemperature() đã được định nghĩa trong giao diện TemperatureSensor.

Trong phương thức getTemperature(), chúng ta giả định rằng giá trị nhiệt độ được lấy từ cảm biến là 75 độ Fahrenheit. Do đó, phương thức trả về giá trị này dưới dạng một số thực kiểu double.

Lớp FahrenheitSensor có thể được sử dụng để đo nhiệt độ trong đơn vị Fahrenheit và triển khai cách thức cụ thể để lấy giá trị nhiệt độ từ cảm biến nhiệt độ.

**e. Class FahrenheitToCelsiusAdapter**

**A screen shot of a computer program

Description automatically generated with medium confidence**

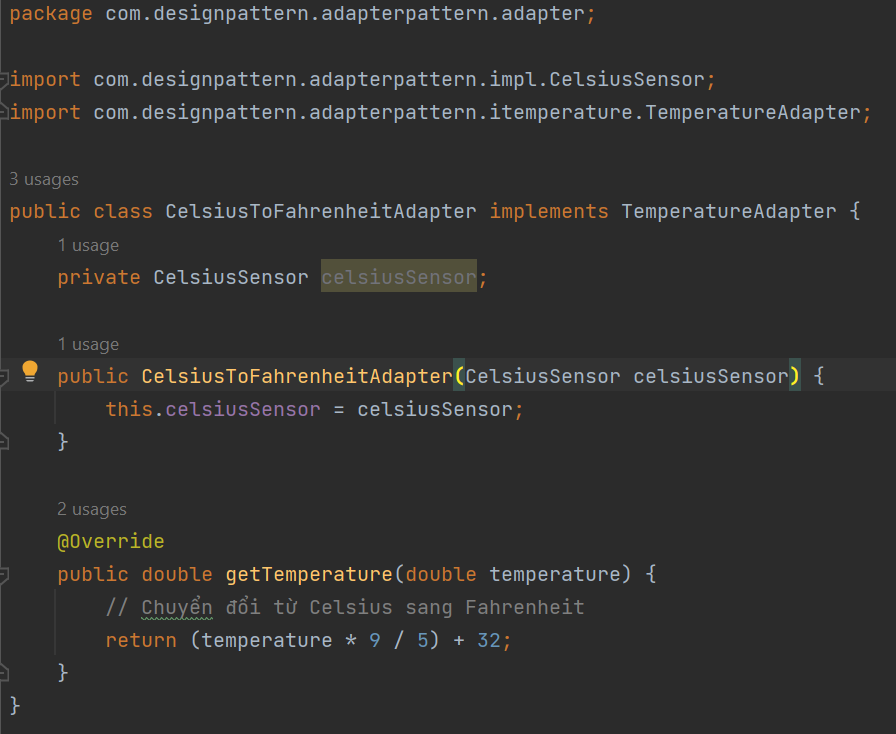
Lớp này triển khai giao diện TemperatureAdapter và chuyển đổi nhiệt độ từ đơn vị Fahrenheit sang Celsius.

Lớp FahrenheitToCelsiusAdapter có một thuộc tính fahrenheitSensor kiểu FahrenheitSensor. Điều này cho phép lớp adapter truy cập đến cảm biến nhiệt độ đo nhiệt độ dưới đơn vị Fahrenheit.

Trong phương thức getTemperature(), chúng ta thực hiện chuyển đổi từ Fahrenheit sang Celsius bằng cách sử dụng công thức (temperature - 32) \* 5 / 9. Đầu vào temperature là giá trị nhiệt độ trong đơn vị Fahrenheit mà chúng ta muốn chuyển đổi. Phương thức này trả về giá trị nhiệt độ tương ứng trong đơn vị Celsius.

Lớp FahrenheitToCelsiusAdapter được sử dụng như một adapter để chuyển đổi các giá trị nhiệt độ từ đơn vị Fahrenheit sang Celsius bằng cách sử dụng cảm biến nhiệt độ đo nhiệt độ trong đơn vị Fahrenheit.

**f. Class CelsiusToFahrenheitAdapter**

****

Lớp này triển khai giao diện TemperatureAdapter và chuyển đổi nhiệt độ từ đơn vị Celsius sang Fahrenheit.

Lớp CelsiusToFahrenheitAdapter có một thuộc tính celsiusSensor kiểu CelsiusSensor. Điều này cho phép lớp adapter truy cập đến cảm biến nhiệt độ đo nhiệt độ dưới đơn vị Celsius.

Trong phương thức getTemperature(), chúng ta thực hiện chuyển đổi từ Celsius sang Fahrenheit bằng cách sử dụng công thức (temperature \* 9 / 5) + 32. Đầu vào temperature là giá trị nhiệt độ trong đơn vị Celsius mà chúng ta muốn chuyển đổi. Phương thức này trả về giá trị nhiệt độ tương ứng trong đơn vị Fahrenheit.

Lớp CelsiusToFahrenheitAdapter được sử dụng như một adapter để chuyển đổi các giá trị nhiệt độ từ đơn vị Celsius sang Fahrenheit bằng cách sử dụng cảm biến nhiệt độ đo nhiệt độ trong đơn vị Celsius.

**g. Class TemperatureForm**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này đại diện cho một đối tượng biểu mẫu nhiệt độ và chứa hai thuộc tính: temperature (giá trị nhiệt độ) và scale (đơn vị nhiệt độ).

Lớp TemperatureForm có các phương thức getter và setter để truy cập và thiết lập giá trị cho thuộc tính temperature và scale.

Thuộc tính temperature là kiểu double và lưu trữ giá trị nhiệt độ. Phương thức getter getTemperature() trả về giá trị nhiệt độ hiện tại, trong khi phương thức setter setTemperature() cho phép thiết lập giá trị nhiệt độ mới.

Thuộc tính scale là kiểu String và lưu trữ đơn vị đo nhiệt độ (ví dụ: Celsius, Fahrenheit). Phương thức getter getScale() trả về đơn vị đo nhiệt độ hiện tại, trong khi phương thức setter setScale() cho phép thiết lập đơn vị đo nhiệt độ mới.

Lớp TemperatureForm được sử dụng để đại diện cho dữ liệu nhiệt độ nhập vào từ biểu mẫu hoặc giao diện người dùng và có thể được sử dụng trong quá trình chuyển đổi nhiệt độ thông qua các adapter.

**h. Class TemperatureController**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này là một Spring MVC Controller và xử lý các yêu cầu liên quan đến chuyển đổi nhiệt độ thông qua các adapter.

Lớp TemperatureController được chú thích bằng @Controller để đánh dấu đây là một bean Controller của Spring.

Phương thức showTemperatureForm() được chú thích bằng @GetMapping("/adapter") để xử lý yêu cầu GET đến URL "/adapter". Phương thức này thêm một đối tượng TemperatureForm vào mô hình (Model) và trả về tên của view "temperature". Điều này cho phép hiển thị một biểu mẫu nhiệt độ cho người dùng nhập dữ liệu.

Phương thức convertTemperature() được chú thích bằng @PostMapping("/adapter/convert") để xử lý yêu cầu POST đến URL "/adapter/convert". Phương thức này nhận một đối tượng TemperatureForm từ form người dùng và một đối tượng Model. Nó trích xuất giá trị nhiệt độ và đơn vị đo từ đối tượng TemperatureForm. Dựa trên đơn vị đo, nó sử dụng các adapter FahrenheitToCelsiusAdapter hoặc CelsiusToFahrenheitAdapter để chuyển đổi nhiệt độ. Kết quả được thêm vào mô hình với các thuộc tính "result", "fromScale" và "toScale" để hiển thị kết quả chuyển đổi và đơn vị đo gốc và đích.

Cuối cùng, phương thức trả về tên của view "result" để hiển thị kết quả chuyển đổi nhiệt độ cho người dùng.

**i. Template**

Bao gồm 2 file html:

temperature.html

A screen shot of a computer program

Description automatically generated with low confidence

Ở giao diện này cho phép người dùng nhập số muốn đổi và chọn loại đơn vị nhiệt độ

result.html

A screen shot of a computer program

Description automatically generated with low confidence

Giao diện này hiển thị kết quả sau khi đổi đơn vị

**2.1.3. Tổ chức thư mục**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.1.4. Kết quả**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.2. Bridge**

**2.2.1. Class Diagram**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.2.2. Các thành phần chính**

**a. Interface Display**

**A screen shot of a computer program

Description automatically generated with low confidence**

Giao diện này định nghĩa một phương thức showDetails() để hiển thị chi tiết về tên, giá và thông tin khác về một sản phẩm.

Phương thức showDetails() có ba tham số:

+ name là tên của sản phẩm.

+ price là giá của sản phẩm.

+ details là thông tin chi tiết về sản phẩm.

Giao diện Display được sử dụng để trừu tượng hóa việc hiển thị thông tin chi tiết của các đối tượng sách (book) và đĩa DVD (DVD) trong một hệ thống. Các lớp triển khai của giao diện Display sẽ cung cấp cách thức cụ thể để hiển thị chi tiết của từng loại sản phẩm (book, DVD) tương ứng.

**b. Class Product**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này đại diện cho một sản phẩm và sử dụng một đối tượng Display để hiển thị thông tin chi tiết của sản phẩm.

Lớp Product có các thuộc tính name, price và display. Thuộc tính name lưu trữ tên của sản phẩm, price lưu trữ giá của sản phẩm, và display lưu trữ một đối tượng Display để hiển thị chi tiết.

Constructor Product() được sử dụng để khởi tạo một đối tượng Product với các thông tin như tên, giá và đối tượng Display. Điều này cho phép kết nối giữa Product và cách thức hiển thị thông tin chi tiết của nó.

Lớp Product có một phương thức trừu tượng getDetails() mà các lớp con phải triển khai. Phương thức này trả về thông tin chi tiết về sản phẩm cụ thể.

Phương thức getName() và getPrice() được sử dụng để truy cập vào thông tin tên và giá của sản phẩm.

Phương thức show() được sử dụng để hiển thị thông tin chi tiết của sản phẩm bằng cách gọi phương thức showDetails() của đối tượng Display. Điều này cho phép việc hiển thị thông tin chi tiết của sản phẩm được thực hiện thông qua cách thức cụ thể mà đối tượng Display triển khai.

Lớp Product đóng vai trò là một phần của cấu trúc Bridge Pattern, nơi nó kết nối một lớp trừu tượng với một giao diện (interface) để thực hiện việc hiển thị thông tin chi tiết của sản phẩm.

**c. Class Book**

A screenshot of a computer program

Description automatically generated with medium confidence

Lớp này là một lớp con của lớp Product và đại diện cho một sản phẩm sách.

Lớp Book có một thuộc tính bổ sung là author (tác giả của sách).

Constructor Book() được sử dụng để khởi tạo một đối tượng Book với các thông tin như tên, giá, tác giả và đối tượng Display. Điều này cho phép kết nối giữa Book và cách thức hiển thị thông tin chi tiết của nó thông qua đối tượng Display.

Phương thức getDetails() được ghi đè từ lớp cha Product. Trong trường hợp của lớp Book, phương thức này trả về tác giả của sách.

Lớp Book là một ví dụ cụ thể của việc sử dụng Bridge Pattern, trong đó nó kết nối giữa lớp Product và giao diện Display để hiển thị thông tin chi tiết về sách.

**d. Class DVD**

A screen shot of a computer program

Description automatically generated with medium confidence

Lớp này là một lớp con của lớp Product và đại diện cho một sản phẩm đĩa DVD.

Lớp DVD có một thuộc tính bổ sung là duration (thời lượng của đĩa DVD).

Constructor DVD() được sử dụng để khởi tạo một đối tượng DVD với các thông tin như tên, giá, thời lượng và đối tượng Display. Điều này cho phép kết nối giữa DVD và cách thức hiển thị thông tin chi tiết của nó thông qua đối tượng Display.

Phương thức getDetails() được ghi đè từ lớp cha Product. Trong trường hợp của lớp DVD, phương thức này trả về thời lượng của đĩa DVD dưới dạng chuỗi, kết hợp với đơn vị "minutes".

Lớp DVD là một ví dụ cụ thể khác của việc sử dụng Bridge Pattern, trong đó nó kết nối giữa lớp Product và giao diện Display để hiển thị thông tin chi tiết về đĩa DVD.

**e. Class ConsoleDisplay**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này triển khai giao diện Display và cung cấp cách thức hiển thị thông tin chi tiết trên console.

Lớp ConsoleDisplay có một phương thức showDetails() được ghi đè từ giao diện Display. Phương thức này nhận ba tham số: name (tên của sản phẩm), price (giá của sản phẩm) và details (thông tin chi tiết về sản phẩm).

Trong phương thức showDetails(), thông tin chi tiết của sản phẩm được hiển thị trên console bằng cách sử dụng phương thức println() của lớp System.out. Tên sản phẩm được hiển thị theo định dạng "Name: [tên sản phẩm]", giá sản phẩm được hiển thị theo định dạng "Price: $[giá sản phẩm]", và thông tin chi tiết được hiển thị theo định dạng "Details: [thông tin chi tiết]".

Lớp ConsoleDisplay là một ví dụ cụ thể của việc triển khai giao diện Display trong Bridge Pattern. Nó cung cấp cách thức cụ thể để hiển thị thông tin chi tiết trên console cho các đối tượng sách (book) và đĩa DVD (DVD).

**f. Class HtmlDisplay**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp này triển khai giao diện Display và cung cấp cách thức hiển thị thông tin chi tiết dưới dạng HTML.

Lớp HtmlDisplay có một phương thức showDetails() được ghi đè từ giao diện Display. Phương thức này nhận ba tham số: name (tên của sản phẩm), price (giá của sản phẩm) và details (thông tin chi tiết về sản phẩm).

Trong phương thức showDetails(), thông tin chi tiết của sản phẩm được hiển thị dưới dạng mã HTML bằng cách sử dụng phương thức println() của lớp System.out. Tên sản phẩm được hiển thị trong thẻ <h2> để tạo tiêu đề cấp 2, giá sản phẩm được hiển thị trong thẻ <p> để tạo đoạn văn bản, và thông tin chi tiết được hiển thị trong thẻ <p>.

Lớp HtmlDisplay là một ví dụ cụ thể khác của việc triển khai giao diện Display trong Bridge Pattern. Nó cung cấp cách thức cụ thể để hiển thị thông tin chi tiết dưới dạng HTML cho các đối tượng sách (book) và đĩa DVD (DVD).

**g. Class AppController**

A screen shot of a computer program

Description automatically generated with medium confidence

Lớp AppController đại diện cho một controller trong mô hình MVC (Model-View-Controller).

Phương thức index() được chú thích bằng @GetMapping("/bridge"), đây là một phương thức xử lý yêu cầu HTTP GET cho đường dẫn "/bridge".

Trong phương thức index(), chúng ta thực hiện các bước sau:

* Tạo một đối tượng consoleDisplay thuộc lớp ConsoleDisplay, đại diện cho cách thức hiển thị thông tin trên console.
* Tạo một đối tượng book thuộc lớp Book, đại diện cho một sản phẩm sách. Chúng ta truyền tên, giá, tác giả và đối tượng consoleDisplay vào constructor của lớp Book.
* Sử dụng đối tượng model để thêm các thuộc tính của đối tượng book vào model. Cụ thể là bookName (tên sách), bookPrice (giá sách) và bookAuthor (tác giả sách). Đối tượng book được ép kiểu thành Book để có thể gọi phương thức getDetails() của lớp Book để lấy thông tin chi tiết về tác giả.
* Tạo một đối tượng htmlDisplay thuộc lớp HtmlDisplay, đại diện cho cách thức hiển thị thông tin dưới dạng HTML.
* Tạo một đối tượng dvd thuộc lớp DVD, đại diện cho một sản phẩm đĩa DVD. Chúng ta truyền tên, giá, thời lượng và đối tượng htmlDisplay vào constructor của lớp DVD.
* Sử dụng đối tượng model để thêm các thuộc tính của đối tượng dvd vào model. Cụ thể là dvdName (tên đĩa DVD), dvdPrice (giá đĩa DVD) và dvdDuration (thời lượng đĩa DVD). Đối tượng dvd được ép kiểu thành DVD để có thể gọi phương thức getDetails() của lớp DVD để lấy thông tin chi tiết về thời lượng.
* Trả về tên view "bookanddvd", để framework hiển thị trang view tương ứng cho người dùng.

Đoạn mã trên là một ví dụ về việc sử dụng Bridge Pattern để kết nối các sản phẩm sách (book) và đĩa DVD (DVD) với các cách thức hiển thị thông tin khác nhau (console, HTML) thông qua việc sử dụng các đối tượng Display và Product.

**h. Template**

Bao gồm 1 trang html: bookanddvd.html

A screen shot of a computer program

Description automatically generated with low confidence

Trang này dùng để hiển thị thông tin lên web bao gồm các thông tin của book và dvd

**2.2.3. Tổ chức thư mục**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.2.4. Kết quả**

**A screenshot of a computer

Description automatically generated**

**2.3. Composite**

**2.3.1. Class Diagram**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.3.2. Các thành phần chính**

**a. Class Employee**

**A screen shot of a computer program

Description automatically generated with medium confidence**

Lớp Employee đại diện cho một nhân viên.

Lớp Employee có hai thuộc tính name và position, đại diện cho tên và vị trí của nhân viên.

Lớp Employee có một constructor nhận hai tham số name và position để khởi tạo đối tượng nhân viên.

Lớp Employee cung cấp các phương thức getName() và getPosition() để truy cập vào thông tin tên và vị trí của nhân viên.

Đây là một ví dụ đơn giản về một lá (leaf) trong mô hình Composite Pattern. Một lá đại diện cho các đối tượng cuối cùng trong cấu trúc cây (composite), trong trường hợp này là đối tượng nhân viên.

**b. Class Department**

A screen shot of a computer program

Description automatically generated with low confidence

Lớp Department đại diện cho một phòng ban.

Lớp Department có hai thuộc tính name và employees. Thuộc tính name đại diện cho tên của phòng ban, và thuộc tính employees là một danh sách (List) chứa các đối tượng nhân viên thuộc phòng ban này.

Lớp Department có một constructor nhận một tham số name để khởi tạo đối tượng phòng ban. Trong constructor, chúng ta khởi tạo danh sách employees bằng một ArrayList.

Lớp Department cung cấp các phương thức addEmployee() và removeEmployee() để thêm và xoá đối tượng nhân viên từ danh sách employees. Phương thức getEmployees() trả về danh sách nhân viên của phòng ban. Phương thức getName() trả về tên của phòng ban.

Lớp Department là một ví dụ về một composite trong mô hình Composite Pattern. Một composite là một đối tượng có thể chứa các đối tượng con (lá hoặc composite khác) trong cấu trúc cây. Trong trường hợp này, Department là composite và các Employee là leaf.

**c. Class EmployeeController**

**A screen shot of a computer program

Description automatically generated with low confidence**

Lớp EmployeeController đại diện cho một controller xử lý các yêu cầu liên quan đến nhân viên.

Lớp EmployeeController được đánh dấu với @Controller để định nghĩa nó là một bean controller trong Spring.

Phương thức index() được đánh dấu với @GetMapping("/composite") để chỉ định nó xử lý yêu cầu GET tới đường dẫn "/composite".

Trong phương thức index(), chúng ta tạo một cấu trúc Composite với hai phòng ban: "Engineering" và "Sales". Mỗi phòng ban chứa một danh sách các đối tượng nhân viên. Chúng ta tạo các đối tượng Employee với tên và vị trí tương ứng, sau đó thêm vào phòng ban tương ứng bằng phương thức addEmployee().

Sau đó, chúng ta tạo một danh sách departments và thêm các phòng ban vào danh sách này.

Cuối cùng, chúng ta truyền danh sách departments vào model bằng phương thức addAttribute() để có thể hiển thị dữ liệu bằng Thymeleaf trong view.

Khi yêu cầu GET tới "/composite" được gửi đi, phương thức index() sẽ được gọi và trả về tên view "employee". Thymeleaf sẽ sử dụng view này để hiển thị dữ liệu phòng ban và nhân viên lên trang web.

**d. Template**

Bao gồm 1 file html: employee.html

**A picture containing text, screenshot, display, software

Description automatically generated**

Hiển thị thông tin nhân viên theo từng phòng ban

**2.3.3. Tổ chức thư mục**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.3.4. Kết quả**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.4. Decorator**

**2.4.1. Class Diagram**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**2.4.2. Các thành phần chính**

**a. Class Student**

A screen shot of a computer

Description automatically generated with medium confidence

Giao diện này định nghĩa một số phương thức để truy xuất thông tin của sinh viên.

* getId(): Phương thức này trả về một số nguyên đại diện cho ID của sinh viên.
* getName(): Phương thức này trả về một chuỗi đại diện cho tên của sinh viên.
* getAge(): Phương thức này trả về một số nguyên đại diện cho tuổi của sinh viên.

Giao diện Student được sử dụng làm cơ sở cho các lớp sinh viên khác để định nghĩa các phương thức cụ thể cho các loại sinh viên cụ thể. Với mẫu thiết kế Decorator, các lớp con có thể thêm chức năng bổ sung vào các phương thức này mà không ảnh hưởng đến giao diện gốc.

**b. Class BasicStudent**

**A screenshot of a computer program

Description automatically generated with medium confidence**

Lớp này đại diện cho một sinh viên cơ bản với các thuộc tính như id, name và age.

* id: Biến lưu trữ ID của sinh viên.
* name: Biến lưu trữ tên của sinh viên.
* age: Biến lưu trữ tuổi của sinh viên.

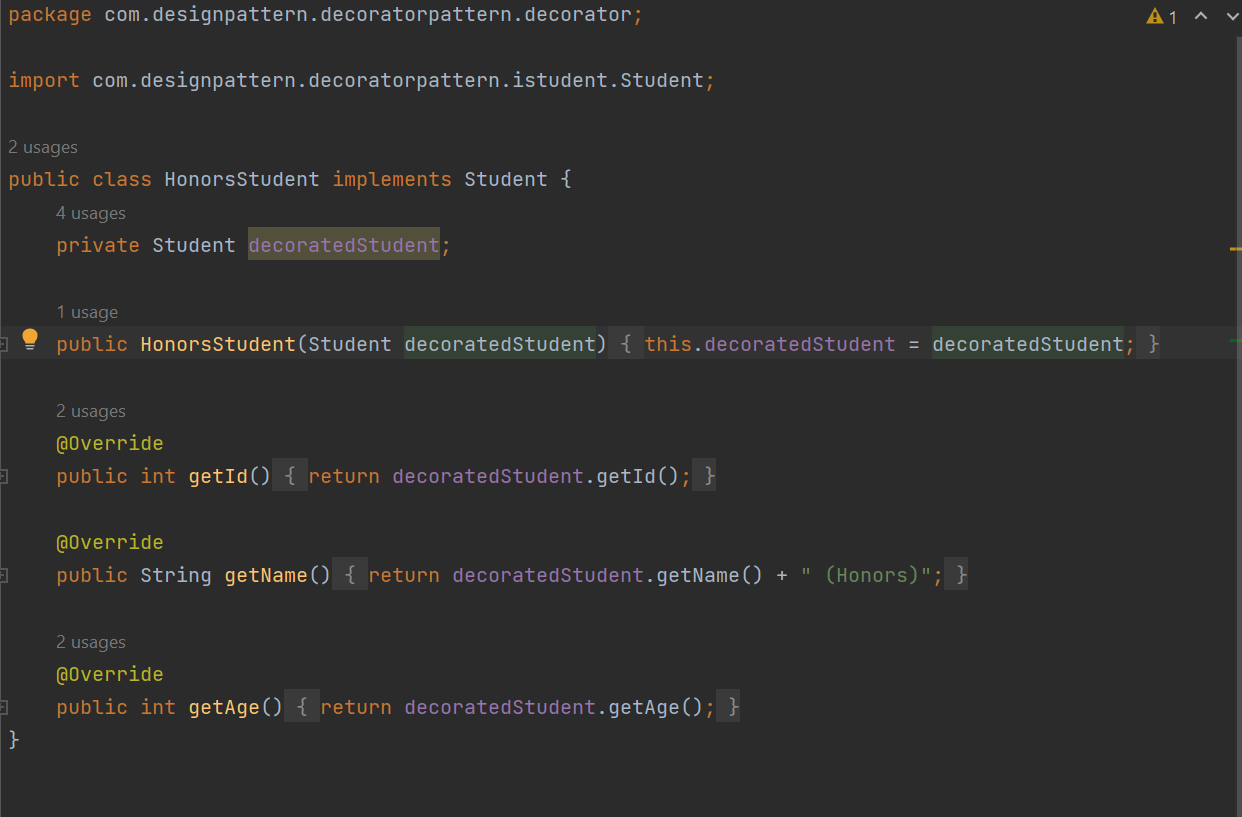
Constructor của lớp BasicStudent được sử dụng để khởi tạo giá trị cho các thuộc tính của sinh viên.

Lớp BasicStudent override các phương thức từ giao diện Student để cung cấp tri thức cụ thể cho mỗi phương thức.

* getId(): Phương thức này trả về giá trị ID của sinh viên.
* getName(): Phương thức này trả về tên của sinh viên.
* getAge(): Phương thức này trả về tuổi của sinh viên.

Lớp BasicStudent được sử dụng làm lớp gốc để xây dựng các lớp Decorator khác để bổ sung chức năng bổ sung hoặc thay đổi hành vi của sinh viên cơ bản.

**c. Class HonorsStudent**



HonorsStudent implement giao diện Student. Lớp này là một Decorator cho sinh viên và cung cấp chức năng bổ sung hoặc thay đổi cho sinh viên gốc.

* decoratedStudent: Biến lưu trữ tham chiếu đến sinh viên gốc mà decorator đang được áp dụng.

Constructor của lớp HonorsStudent nhận một đối tượng sinh viên gốc (decoratedStudent) và lưu trữ nó trong biến decoratedStudent.

Lớp HonorsStudent override các phương thức từ giao diện Student để bổ sung hoặc thay đổi hành vi của sinh viên gốc.

* getId(): Phương thức này trả về giá trị ID của sinh viên gốc.
* getName(): Phương thức này trả về tên của sinh viên gốc kèm theo chuỗi "(Honors)" để biểu thị rằng sinh viên này là sinh viên danh dự.
* getAge(): Phương thức này trả về tuổi của sinh viên gốc.

Lớp HonorsStudent là một ví dụ về việc sử dụng Decorator Pattern để thêm tính năng danh dự cho sinh viên cơ bản.

**d. Class StudentController**

**A screen shot of a computer program

Description automatically generated with low confidence**

StudentController. Đây là một controller trong Spring Boot để quản lý việc hiển thị danh sách sinh viên.

Lớp StudentController được chú thích bằng @Controller để đánh dấu nó là một thành phần điều khiển trong Spring Framework.

Phương thức getStudents() được chú thích bằng @GetMapping("/decorator") để xác định URL mà phương thức này xử lý.

Trong phương thức getStudents(), một danh sách sinh viên được tạo với dữ liệu giả lập. Danh sách này bao gồm các đối tượng BasicStudent và một đối tượng HonorsStudent được tạo bằng cách truyền một đối tượng BasicStudent vào constructor của HonorsStudent. Điều này tạo ra một sinh viên danh dự từ một sinh viên cơ bản.

Danh sách sinh viên được truyền vào model thông qua model.addAttribute("students", students), để có thể truy cập từ giao diện Thymeleaf.

Cuối cùng, phương thức trả về tên của template "students". Điều này cho phép Thymeleaf hiển thị danh sách sinh viên trong template "students.html".

**e. Teamplate**

Bao gồm 1 file html: students.html

A screen shot of a computer program

Description automatically generated with low confidence

Hiển thị id, name, age

**2.4.3. Tổ chức thư mục**

A screenshot of a computer

Description automatically generated\

**2.4.4. Kết quả**

A screen shot of a computer

Description automatically generated with low confidence

**2.5. Façade**

**2.5.1. Class Diagram**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**2.5.2. Các thành phần chính**

**a. Class Product**

**A screen shot of a computer program

Description automatically generated with low confidence**

Đây là một lớp đại diện cho một sản phẩm trong hệ thống.

Lớp Product có các thuộc tính id, name, và price để lưu trữ thông tin về sản phẩm. Các thuộc tính này được khai báo là private để đảm bảo tính đóng gói.

Lớp Product cung cấp các phương thức getter và setter cho các thuộc tính id, name, và price, để cho phép truy cập và thay đổi giá trị của chúng.

Phương thức tạo Product(int id, String name, double price) được sử dụng để khởi tạo một đối tượng Product với các giá trị được chuyển vào qua các tham số. Điều này cho phép tạo ra một sản phẩm với các thông tin cụ thể.

Đây là một lớp đơn giản đại diện cho một sản phẩm trong hệ thống và cung cấp các phương thức để truy cập và thay đổi thông tin của sản phẩm.

**b. Class ProductService**

**A screen shot of a computer program

Description automatically generated with low confidence**

Đây là một lớp dịch vụ (service) được sử dụng để thực hiện các hoạt động liên quan đến sản phẩm trong hệ thống.

Lớp ProductService được đánh dấu với annotation @Service để đánh dấu rằng đây là một thành phần dịch vụ trong Spring Framework.

Phương thức getAllProducts() của ProductService được sử dụng để lấy danh sách tất cả các sản phẩm từ nguồn dữ liệu. Trong ví dụ này, danh sách sản phẩm được tạo ngay trong phương thức để minh họa. Các sản phẩm được tạo bao gồm "iPhone" với giá 1000.0, "Samsung Galaxy" với giá 800.0 và "Google Pixel" với giá 700.0.

Phương thức trả về một danh sách các đối tượng Product, đại diện cho tất cả các sản phẩm trong hệ thống.

Lớp ProductService chịu trách nhiệm thực hiện các hoạt động liên quan đến việc truy xuất dữ liệu về sản phẩm và cung cấp chúng cho các thành phần khác trong hệ thống thông qua giao diện dễ sử dụng.

**c. Class ProductFacade**

**A screen shot of a computer program

Description automatically generated with low confidence**

Đây là một lớp facade, cung cấp một giao diện đơn giản và thuận tiện để tương tác với các dịch vụ liên quan đến sản phẩm trong hệ thống.

Lớp ProductFacade được đánh dấu với annotation @Service để đánh dấu rằng đây là một thành phần dịch vụ trong Spring Framework.

Lớp ProductFacade chứa một tham chiếu đến ProductService, được tiêm qua constructor injection thông qua hàm tạo của lớp. Điều này cho phép ProductFacade truy cập và sử dụng các phương thức được cung cấp bởi ProductService.

Phương thức getAllProducts() của ProductFacade đơn giản là chuyển tiếp yêu cầu lấy danh sách sản phẩm tới ProductService và trả về kết quả trực tiếp từ ProductService. Điều này giúp ẩn đi sự phức tạp của việc truy xuất dữ liệu và cung cấp một giao diện đơn giản và trực tiếp cho việc lấy danh sách sản phẩm.

Lớp ProductFacade có nhiệm vụ đóng gói và ẩn đi chi tiết triển khai của ProductService, giúp đơn giản hóa việc sử dụng và tương tác với dịch vụ liên quan đến sản phẩm trong hệ thống.

**d. Class ProductController**

A screen shot of a computer

Description automatically generated with medium confidence

Đây là một lớp controller trong Spring Framework, được đánh dấu với annotation @Controller để đánh dấu rằng đây là một thành phần xử lý các yêu cầu HTTP.

Lớp ProductController chứa một tham chiếu đến ProductFacade, được tiêm qua constructor injection thông qua hàm tạo của lớp. Điều này cho phép ProductController truy cập và sử dụng các phương thức được cung cấp bởi ProductFacade.

Phương thức home() của ProductController được chú thích với @GetMapping("/facade"), chỉ định rằng phương thức này xử lý các yêu cầu GET tới đường dẫn "/facade".

Trong phương thức home(), danh sách sản phẩm được lấy thông qua productFacade.getAllProducts(). Danh sách sản phẩm này được chuyển vào model thông qua model.addAttribute("products", products) để truyền dữ liệu tới giao diện Thymeleaf.

Cuối cùng, phương thức trả về tên của template, trong trường hợp này là "product". Điều này xác định rằng khi yêu cầu được xử lý thành công, template "product.html" sẽ được hiển thị cho người dùng.

Lớp ProductController có nhiệm vụ xử lý yêu cầu HTTP liên quan đến sản phẩm và sử dụng ProductFacade để truy xuất danh sách sản phẩm. Nó cung cấp một giao diện đơn giản để tương tác với danh sách sản phẩm trong hệ thống và truyền dữ liệu tới giao diện Thymeleaf để hiển thị danh sách sản phẩm cho người dùng.

e. Template

Bao gồm 1 file html: product.html

A screen shot of a computer

Description automatically generated with medium confidence

Hiển thị các thông tin của các sản phẩm bao gồm id, name, price

**2.5.3. Tổ chức thư mục**

A screenshot of a computer

Description automatically generated with medium confidence

**2.5.4. Kết quả**

A screenshot of a computer

Description automatically generated with medium confidence

**2.6. Flyweight**

**2.6.1. Class Diagram**

A screenshot of a computer screen

Description automatically generated with low confidence

**2.6.2. Các thành phần chính**

**a. Class User**



Lớp này đại diện cho một người dùng trong hệ thống.

Lớp User có hai thuộc tính private là username và email, được định nghĩa bằng kiểu dữ liệu String. Để truy cập và thay đổi giá trị của các thuộc tính này, lớp User cung cấp các phương thức getter và setter tương ứng.

Phương thức getter getUsername() được sử dụng để truy xuất giá trị của thuộc tính username. Phương thức tương tự cũng được triển khai cho thuộc tính email.

Phương thức setter setUsername() và setEmail() được sử dụng để thiết lập giá trị cho thuộc tính username và email tương ứng.

Hàm tạo của lớp User nhận vào hai tham số là username và email, và được sử dụng để khởi tạo một đối tượng User với các giá trị được truyền vào.

Lớp User đại diện cho thông tin cơ bản của một người dùng trong hệ thống, bao gồm tên đăng nhập (username) và địa chỉ email (email).

**b. Class UserFactory**

A screen shot of a computer program

Description automatically generated with low confidence

Lớp này đại diện cho một factory (nhà máy) để tạo ra các đối tượng User theo yêu cầu.

Lớp UserFactory được đánh dấu với annotation @Component, cho phép Spring Boot quản lý và tự động cấu hình lớp này.

Lớp UserFactory có một thuộc tính private là userMap, được định nghĩa dưới dạng một Map để lưu trữ các đối tượng User. Key của Map là username và value là đối tượng User tương ứng.

Hàm tạo của lớp UserFactory khởi tạo một HashMap để lưu trữ danh sách các đối tượng User.

Phương thức getUser(String username) được sử dụng để lấy một đối tượng User từ factory. Đầu tiên, phương thức kiểm tra xem có tồn tại đối tượng User trong userMap với username đã cho hay không. Nếu không tồn tại, phương thức sẽ tạo một đối tượng User mới bằng cách gọi phương thức createUser(String username) và thêm đối tượng này vào userMap. Cuối cùng, phương thức trả về đối tượng User.

Phương thức private createUser(String username) được sử dụng để tạo một đối tượng User mới dựa trên username. Trong ví dụ này, hàm tạo đơn giản tạo một địa chỉ email dựa trên username và sử dụng nó để khởi tạo đối tượng User mới.

Factory UserFactory giúp tạo ra các đối tượng User duy nhất theo yêu cầu và giúp tiết kiệm tài nguyên bằng cách tái sử dụng các đối tượng đã được tạo trước đó thông qua việc lưu trữ chúng trong userMap.

**c. Class UserController**

A screen shot of a computer program

Description automatically generated with low confidence

Trong đoạn code trên, chúng ta định nghĩa một UserController trong một ứng dụng Spring Boot để quản lý người dùng. Điểm mấu chốt của code là:

* @Controller: Đây là một chú thích để xác định lớp UserController là một controller trong Spring MVC.
* UserFactory: Lớp UserController sử dụng UserFactory để tạo ra các đối tượng User.
* @GetMapping("/flyweight"): Phương thức users() trong UserController được ánh xạ với yêu cầu GET tới đường dẫn "/flyweight".
* Model: Tham số Model được sử dụng để truyền dữ liệu từ controller đến view.
* userFactory.getUser(): Trong phương thức users(), chúng ta sử dụng UserFactory để tạo ra các đối tượng User dựa trên tên người dùng.
* model.addAttribute(): Danh sách người dùng (userList) được đặt vào model với tên "users".
* return "users": Cuối cùng, controller trả về tên template "users" để hiển thị danh sách người dùng.

Khi truy cập vào đường dẫn "/flyweight", controller sẽ tạo ra danh sách người dùng bằng cách sử dụng UserFactory và truyền danh sách này vào view để hiển thị.

**d. Template**

Bao gồm 1 file html: users.html

A picture containing text, screenshot, software, display

Description automatically generated

Hiển thị thông tin của các user bao gồm username, email

**2.6.3. Tổ chức thư mục**

A screenshot of a computer

Description automatically generated with medium confidence

**2.6.4. Kết quả**

**A screenshot of a computer

Description automatically generated with medium confidence**

**2.7. Proxy**

**2.7.1. Class Diagram**

**A screenshot of a computer program

Description automatically generated with low confidence**

**2.7.2. Các thành phần chính**

**a. Interface Image**

**A screenshot of a computer program

Description automatically generated with medium confidence**

Định nghĩa một interface Image với một phương thức display().

* display(): Phương thức này được sử dụng để hiển thị hình ảnh. Cụ thể cách hiển thị hình ảnh sẽ được triển khai trong các lớp cụ thể implement interface này.

Interface Image đại diện cho một giao diện chung để làm việc với hình ảnh. Nó cho phép các lớp khác triển khai phương thức display() theo cách riêng của chúng, tùy thuộc vào nhu cầu cụ thể của ứng dụng.

**b. Class RealImage**

A screen shot of a computer program

Description automatically generated with medium confidence

Lớp RealImage được implement từ interface Image.

RealImage là một lớp thực hiện giao diện Image, đại diện cho hình ảnh thực tế cần hiển thị.

Trong constructor của lớp, chúng ta nhận một tham số filename đại diện cho tên file hình ảnh. Trong ví dụ này, khi tạo một đối tượng RealImage, chúng ta giả định rằng hình ảnh sẽ được tải từ đĩa.

Phương thức loadFromDisk() được sử dụng để giả lập việc tải hình ảnh từ đĩa.

Phương thức display() được override từ interface Image để hiển thị hình ảnh. Trong trường hợp này, chúng ta đơn giản chỉ in ra thông báo "Displaying image" kèm theo tên file hình ảnh.

**c. Class ImageProxy**

A screen shot of a computer program

Description automatically generated with low confidence

Lớp ImageProxy được implement từ interface Image.

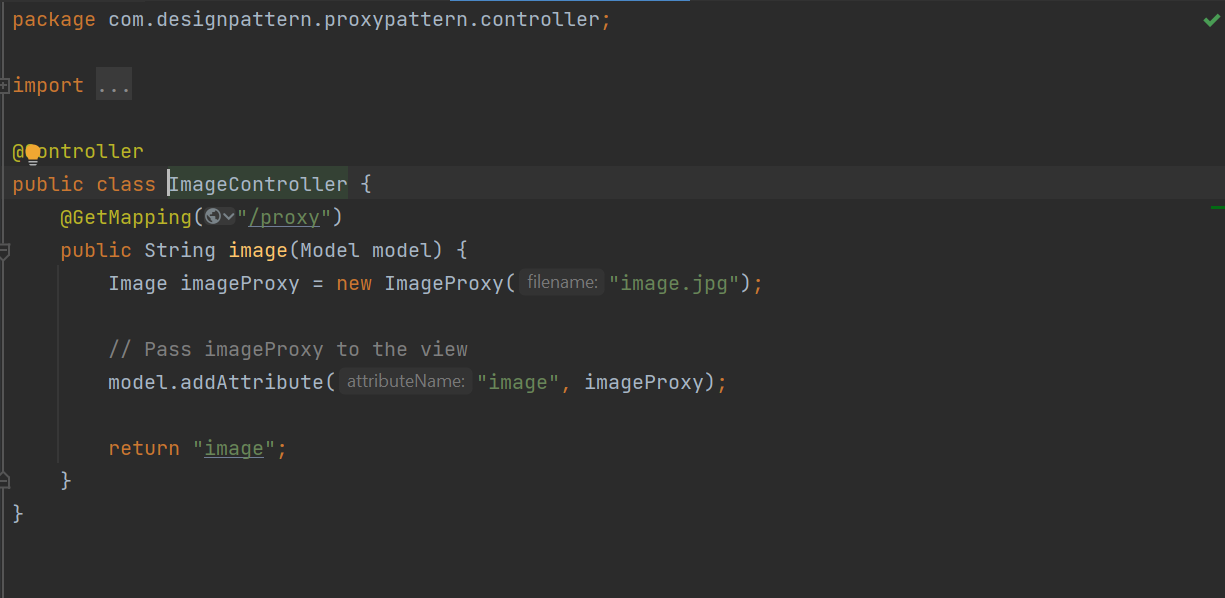
ImageProxy là một lớp proxy cho RealImage, cung cấp một giao diện tương tự như RealImage để hiển thị hình ảnh.

Lớp ImageProxy chứa một đối tượng RealImage và tên file hình ảnh.

Phương thức display() được override từ interface Image. Trong phương thức này, chúng ta kiểm tra xem đối tượng realImage đã được khởi tạo hay chưa. Nếu chưa, chúng ta tạo một đối tượng RealImage mới với tên file và gọi phương thức display() của RealImage. Nếu đã tồn tại đối tượng realImage, chúng ta gọi phương thức display() của nó.

Bằng cách sử dụng ImageProxy, chúng ta có thể trì hoãn việc tạo đối tượng RealImage cho đến khi thực sự cần hiển thị hình ảnh, giúp tối ưu hóa việc tải hình ảnh từ đĩa.

**d. Class ImageController**



Lớp ImageController đóng vai trò là một controller xử lý yêu cầu liên quan đến hình ảnh.

Lớp ImageController được chú thích với @Controller, cho phép Spring Boot nhận dạng và quản lý lớp này.

Phương thức image() được chú thích với @GetMapping("/proxy"), chỉ định rằng phương thức này sẽ được gọi khi có yêu cầu GET được gửi đến đường dẫn "/proxy".

Trong phương thức image(), chúng ta tạo một đối tượng ImageProxy với tên file "image.jpg".

Đối tượng ImageProxy được truyền vào model thông qua phương thức addAttribute() với tên "image".

Cuối cùng, chúng ta trả về tên template "image", để Spring Boot render template và trả về kết quả cho yêu cầu.

Với cấu hình thích hợp, khi người dùng truy cập vào đường dẫn "/proxy", hình ảnh sẽ được hiển thị thông qua ImageProxy.

**e. Template**

Bao gồm 1 file html và 1 ảnh jpg: image.html, image.jpg

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Hiển thị ảnh

**2.7.3. Tổ chức thư mục**

**A screenshot of a computer

Description automatically generated**

**2.7.4. Kết quả**

**A screenshot of a computer

Description automatically generated with low confidence**