

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM



LẬP TRÌNH JAVA

Biên soạn: ThS. Dương Hữu Thành (chủ biên)
ThS. Võ Tân Dũng

Website: <http://www.hutech.edu.vn>

LẬP TRÌNH JAVA

Ãn bản 2021

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN	IV
BÀI 1. LẬP TRÌNH JAVA CĂN BẢN	1
1.1 GIỚI THIỆU NGÔN NGỮ JAVA	1
1.2 ĐẶC ĐIỂM NGÔN NGỮ JAVA	2
1.3 CÀI ĐẶT MÔI TRƯỜNG JAVA	4
1.3.1 Cài đặt môi trường	4
1.3.2 Viết chương trình đầu tiên với Java	4
1.4 BIẾN VÀ KIỂU DỮ LIỆU	7
1.5 CÁC PHÉP TOÁN	9
1.6 CẤU TRÚC ĐIỀU KHIỂN	9
1.7 CHUỖI	12
1.7.1 Giới thiệu chuỗi	12
1.7.2 Các thao tác thông dụng trên chuỗi	12
1.7.3 Sử dụng StringBuilder	15
1.8 MẢNG	16
1.8.1 Giới thiệu mảng	16
1.8.2 Các thao tác trên mảng	16
1.9 DATE	18
1.10 NHẬP XUẤT TRONG CONSOLE	19
1.11 ĐỌC GHI TẬP TIN VĂN BẢN	20
1.11.1 Đọc tập tin văn bản	20
1.11.2 Ghi tập tin văn bản	20
BÀI 2. TỔNG QUAN SPRING FRAMEWORK	22
1.12 TỔNG QUAN SPRING FRAMEWORK	22
1.13 KIẾN TRÚC CỦA SPRING FRAMEWORK	23
1.14 IoC CONTAINER	26
1.14.1 Application Context	28

1.14.2	DEPENDENCY INJECTION	29
1.14.3	SỬ DỤNG XML CẤU HÌNH BEANS	29
1.14.4	SỬ DỤNG ANNOTATION CẤU HÌNH BEANS	36
1.15	VÒNG ĐỜI CỦA SPRING BEAN	39
BÀI 3.	SPRING MVC	41
1.15.1	GIỚI THIỆU SPRING MVC	41
1.15.2	FRONT CONTROLLER DESIGN PATTERN	42
1.15.3	DISPATCHERSERVLET	43
1.15.4	CÀI ĐẶT MÔI TRƯỜNG	46
1.15.5	VIẾT CHƯƠNG TRÌNH ĐẦU TIÊN VỚI SPRING MVC	48
1.16	LÀM VIỆC VỚI CONTROLLER	56
1.17	TAG LIBRARIES	61
1.18	VIEW RESOLVER	69
1.18.1	REDIRECT	69
1.18.2	THUỘC TÍNH FLASK	70
1.18.3	STATIC RESOURCES	71
1.18.4	MULTIPART REQUEST	72
BÀI 4.	JDBC	76
1.19	GIỚI THIỆU MYSQL	76
1.20	GIỚI THIỆU JDBC	81
1.21	SỬ DỤNG JDBC TƯƠNG TÁC CƠ SỞ DỮ LIỆU	83
1.22	GIAO TÁC	86
1.23	BATCH QUERY	87
1.24	SPRING JDBC	88
BÀI 5.	HIBERNATE	91
1.25	GIỚI THIỆU ORM	91
1.26	GIỚI THIỆU HIBERNATE	92
1.27	HQL	94
1.28	SỬ DỤNG ANNOTATION	101

1.29 CRITERIA API	104
1.30 CRITERIA QUERY API	108
BÀI 6. PHÁT TRIỂN ỨNG DỤNG VỚI SPRING MVC VÀ HIBERNATE	112
1.31 MÔ TẢ BÀI TOÁN	112
1.32 KIẾN TRÚC SPRING WEB MVC	120
1.32.1 DOMAIN LAYER	120
1.32.2 PERSISTENCE LAYER	123
1.32.3 SERVICE LAYER	127
1.33 TEMPLATE WITH TILES	130
1.34 BEAN VALIDATION	143
1.34.1 JAVA BEAN VALIDATION	143
1.34.2 SPRING VALIDATION	156
1.35 SPRING SECURITY	161
1.36 TẠO REST API CHO ỨNG DỤNG VỚI AJAX	180
1.37 INTERCEPTORS	194
BÀI 7. JAVA SWING	196
1.38 GIỚI THIỆU JAVA SWING	196
1.39 MỘT SỐ COMPONENT TRONG SWING	200
TÀI LIỆU THAM KHẢO	258

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Môn học trang bị các kiến thức nhập môn về lập trình Java. Nội dung bao gồm:

- Tổng quan Spring Framework.
- Tổng quan phát triển Web với Spring MVC.
- Tương tác với cơ sở dữ liệu MySQL bằng JDBC.
- Tương tác với cơ sở dữ liệu MySQL bằng Hibernate.
- Phát triển ứng dụng Web với Spring MVC và Hibernate.
- Phát triển một ứng dụng bằng Swing kết hợp với JDBC.

NỘI DUNG MÔN HỌC

- Bài 1: Lập trình Java căn bản
- Bài 2: Spring Framework
- Bài 3: Spring MVC
- Bài 4: JDBC
- Bài 5: Hibernate
- Bài 6: Phát triển ứng dụng với Hibernate và JDBC
- Bài 7: Java Swing

KIẾN THỨC TIỀN ĐỀ

Môn Lập trình web cần kiến thức tiền đề của các môn học sau.

- Lập trình hướng đối tượng

- Lập trình Web

YÊU CẦU MÔN HỌC

Người học yêu cầu có kiến thức nền tảng lập trình hướng đối tượng, đặc biệt với Java, đồng thời phải vận dụng được kiến thức nền tảng về lập trình Web để có thể cài đặt các ứng dụng Web bằng Spring Framework.

Kỹ năng lập trình, và phân tích thiết kế ứng dụng thành thạo.

Người học cần đi học đầy đủ, đọc các nội dung sẽ được học trước khi đến lớp, làm các bài tập về nhà và đảm bảo thời gian tự học ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần đọc trước các nội dung chưa được học trên lớp; tham gia đều đặn và tích cực trên lớp; hiểu các khái niệm, tính chất và ví dụ tại lớp học. Sau khi học xong, cần ôn lại bài đã học và làm các bài tập. Tìm đọc thêm các tài liệu khác liên quan đến bài học và làm thêm bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm ba thành phần.

- Điểm thực hành (30%): Hình thức thi thực hành, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.
- Điểm quá trình (20%): Hình thức và cách đánh giá do giảng viên dạy lý thuyết quyết định được phê duyệt của bộ môn.
- Điểm thi cuối kỳ (50%): Hình thức làm đồ án môn học và chấm thi vẫn đáp dựa trên nội dung đồ án môn học kết hợp với lý thuyết trong các bài học. Danh sách đồ án do giảng viên quyết định được bộ môn kiểm duyệt và cung cấp vào đầu khóa học.

BÀI 1. LẬP TRÌNH JAVA CĂN BẢN

Học xong bài này người học sẽ nắm được các nội dung sau:

- *Hiểu đặc trưng quan trọng và cơ chế hoạt động Java.*
- *Hiểu các cấu trúc cơ bản của Java và lập trình hướng đối tượng trong Java.*
- *Lập trình giải quyết các bài toán cơ bản bằng Java SE.*

1.1 GIỚI THIỆU NGÔN NGỮ JAVA

Java là ngôn ngữ lập trình cấp cao mã nguồn mở được xây dựng trên nền tảng ngôn ngữ C/C++ khởi đầu bởi James Gosling và các đồng nghiệp ở công ty Sun Microsystems vào năm 1991 và là một phần của dự án Xanh (Green). Ngôn ngữ này ban đầu có tên là Oak, chính thức được đổi tên thành Java năm 1995. Từ đó, nó được giữ bản quyền và phát triển bởi công ty Sun Microsystems. Năm 2010, Oracle thâu tóm Sun Microsystems và tiếp tục đẩy mạnh phát triển nền tảng Java cho đến ngày nay.

Java có thể tương thích với nhiều dòng máy PC, Macintosh, với nhiều hệ điều hành như Windows, Linux, OSX... và là ngôn ngữ vừa biên dịch, vừa thông dịch: đầu tiên mã nguồn Java được biên dịch thành mã bytecode và chạy trên từng nền tảng nhờ trình thông dịch. Java ban đầu được sử dụng chủ yếu lập trình trong môi trường mạng và internet với khẩu hiệu “viết một lần, chạy mọi nơi” (write once, run anywhere).

Write Once, Run Anywhere

Phiên bản	Tên mã	Ngày phát hành
Java SE 1.0	Oak	21-01-1996
Java SE 1.1	-	19-02-1997
Java SE 1.2	Playground	08-12-1998
Java SE 1.3	Kestrel	08-05-2000
Java SE 1.4	Merlin	06-02-2002

Java SE 5	Tiger	30-09-2004
Java SE 6	Mustang	11-12-2006
Java SE 7	Dolphin	28-07-2011
Java SE 8	Spider	18-03-2014
Java SE 9	-	21-09-2017
Java SE 10	-	20-03-2018
Java SE 11	-	25-09-2018
Java SE 12	-	19-03-2019
Java SE 13	-	10-09-2019

Bảng 1. Các phiên bản phát hành của Java.

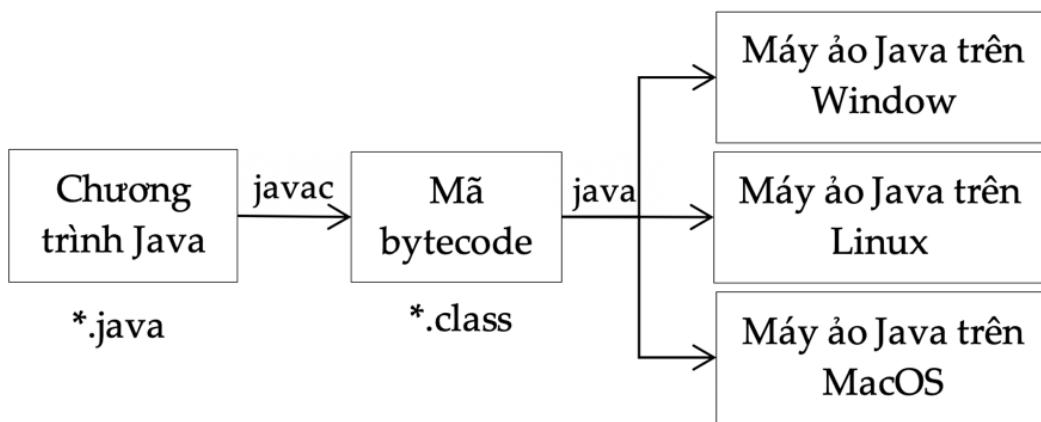
1.2 ĐẶC ĐIỂM NGÔN NGỮ JAVA

Đơn giản (simple): ngôn ngữ Java được xây dựng trên nền tảng ngôn ngữ C++, nhưng đơn giản và có nhiều cải thiện, chẳng hạn:

- Loại bỏ các thao tác con trỏ, không sử dụng struct, union.
- Không sử dụng các tập tin header.
- Không cho phép đa kế thừa trong các lớp và thay vào đó là sử dụng giao diện (interface).

Hướng đối tượng (object-oriented): tất cả mã nguồn trong Java đều được viết trong các lớp (class) và mọi thực thể hoạt động trong hệ thống đều là đối tượng (object). Đặc trưng hướng đối tượng trong Java gần như tương tự ngôn ngữ C++, điểm khác nhau chính là Java không hỗ trợ đa kế thừa và có mô hình metaclass (tức là một lớp có các thể hiện cũng là một lớp).

Độc lập nền tảng (platform independent): Java có thể thực thi trên bất kỳ nền tảng nào có hỗ trợ máy ảo Java. Máy ảo Java là phần mềm mô phỏng máy tính để thực thi chương trình, nó trong suốt với hệ điều hành và phần cứng khi chương trình tương tác với nó. Java là ngôn ngữ vừa biên dịch, vừa thông dịch: mã nguồn Java đầu tiên sẽ được trình biên dịch javac dịch thành mã bytecode độc lập với phần cứng, sau đó máy ảo Java sẽ dùng trình thông dịch java để chuyển mã bytecode thành dạng chạy được trên từng nền tảng khác nhau.



Hình 1. Quá trình thực thi chương trình Java.

Khả chuyển (portable): do Java độc lập nền tảng nên có tính khả chuyển, không giống như mã máy phụ thuộc trên từng nền tảng khác nhau, mã bytecode là độc lập nền tảng, nó có thể thực thi trên bất kỳ nền tảng nào có hỗ trợ máy ảo Java mà không cần biên dịch lại mã nguồn chương trình.

Mạnh mẽ (robust): trong Java kiểu dữ liệu phải được khai báo tường minh, bắt buộc lập trình viên phải giải quyết các ngoại lệ không kiểm soát, cơ chế bắt ngoại lệ try..catch dễ dàng xử lý lỗi và phục hồi sau lỗi. Quá trình cấp phát và giải phóng vùng nhớ được thực hiện tự động bằng bộ thu gom rác (garbage collector). Trình biên dịch của Java có thể phát hiện được nhiều vấn đề của chương trình, mà ở nhiều ngôn ngữ khác chỉ phát hiện khi thực thi chương trình.

Phân tán (distributed): phân tán có nghĩa là nhiều máy tính cùng làm việc trên môi trường mạng. Java được thiết kế để lập trình trên môi trường mạng và internet dễ dàng.

An toàn (secure): Java có nhiều cơ chế bảo vệ hệ thống trước các chương trình độc hại.

Đa luồng (multithreaded): đa luồng là khả năng một chương trình có thể thực hiện nhiều nhiệm vụ đồng thời. Đa luồng đặc biệt quan trọng trong các ứng dụng giao diện người dùng và lập trình mạng, chẳng hạn trong lập trình giao diện có những chức năng người dùng vừa nghe nhạc, vừa đọc sách, vừa chơi game, hoặc trong lập trình

mạng hệ thống phải phục vụ rất nhiều người dùng cùng lúc. Việc lập trình đa luồng được Java hỗ trợ thực hiện một cách dễ dàng.

Linh động (dynamic): Java được thiết kế dễ dàng cho việc tiến hóa và mở rộng chương trình.

1.3 CÀI ĐẶT MÔI TRƯỜNG JAVA

1.3.1 Cài đặt môi trường

Để phát triển và chạy chương trình với Java đầu tiên cần cài đặt bộ công cụ JDK (Java Development Toolkit), là bộ công cụ dành cho lập trình viên phát triển và thực thi ứng dụng với Java. Tải phiên bản JDK (Java Development Toolkit) mới nhất tại:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

double-click vào file cài đặt và thực hiện theo các hướng dẫn.

Sau đó cài đặt biến môi trường, mục đích thiết lập biến môi trường để thực thi các chương trình Java được thuận tiện mà không cần chỉ định đường dẫn đầy đủ đến chương trình đó.

<https://docs.oracle.com/javase/tutorial/essential/environment/paths.html>

Sau khi cài đặt Java có thể command prompt gõ lệnh “java -version” kết quả sẽ giống như sau:

```
Command Prompt
Microsoft Windows [Version 10.0.17134.191]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\dhthanh>java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) Client VM (build 25.181-b13, mixed mode, sharing)

C:\Users\dhthanh>javac -version
javac 1.8.0_101
```

1.3.2 Viết chương trình đầu tiên với Java

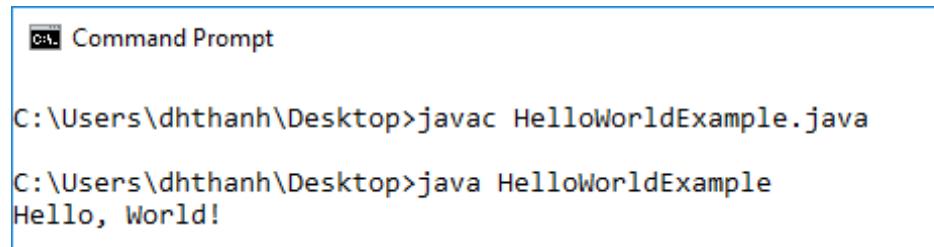
Minh họa bằng Notepad

Mở Notepad và gõ đoạn mã nguồn như bên dưới, sau đó lưu lại với tên HelloWorldExample.java.

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

Chú ý quy tắc là một tập tin mã nguồn *.java chỉ chứa duy nhất một lớp có từ khóa public và tên lớp phải trùng với tên tập tin chứa mã nguồn Java đó.

Để chạy chương trình trên Windows, ta mở Command Prompt (cmd), chuyển đến thư mục chứa tập tin mã nguồn trên và dùng lệnh javac để biên dịch HelloWorldExample.java thành mã bytecode. Sau lệnh này một tập tin HelloWorldExample.class chứa mã bytecode sẽ được tạo ra cùng thư mục, tiếp tục dùng lệnh java để thực thi mã bytecode vừa được biên dịch. Kết quả thực thi chương trình.



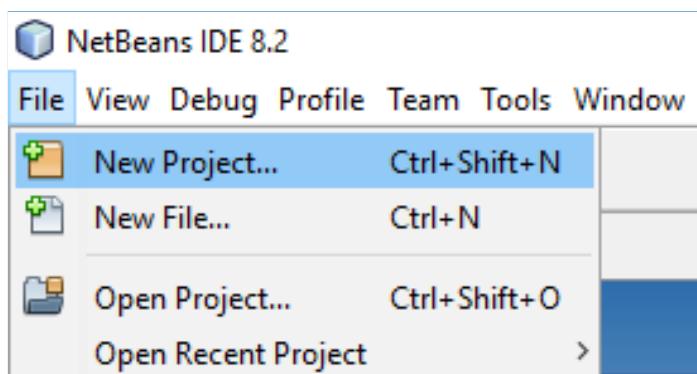
```
Command Prompt

C:\Users\dhthanh\Desktop>javac HelloWorldExample.java
C:\Users\dhthanh\Desktop>java HelloWorldExample
Hello, World!
```

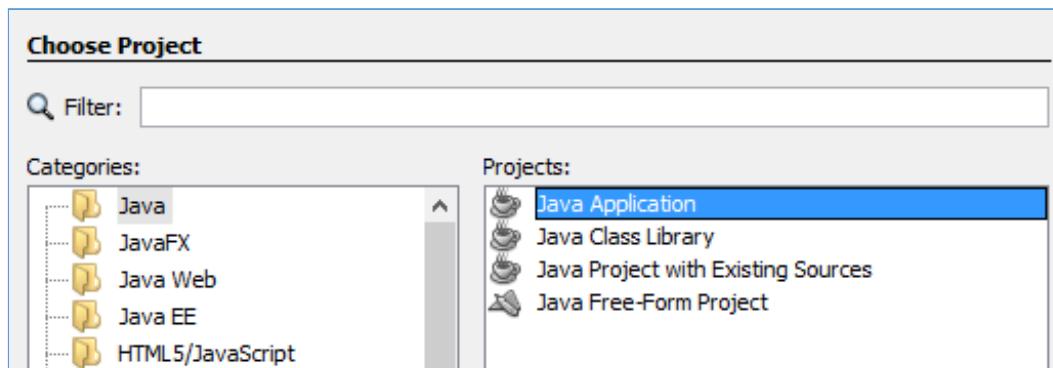
Minh họa NetBean IDE

Đầu tiên cần tải NetBeans tại <https://netbeans.org/downloads/>.

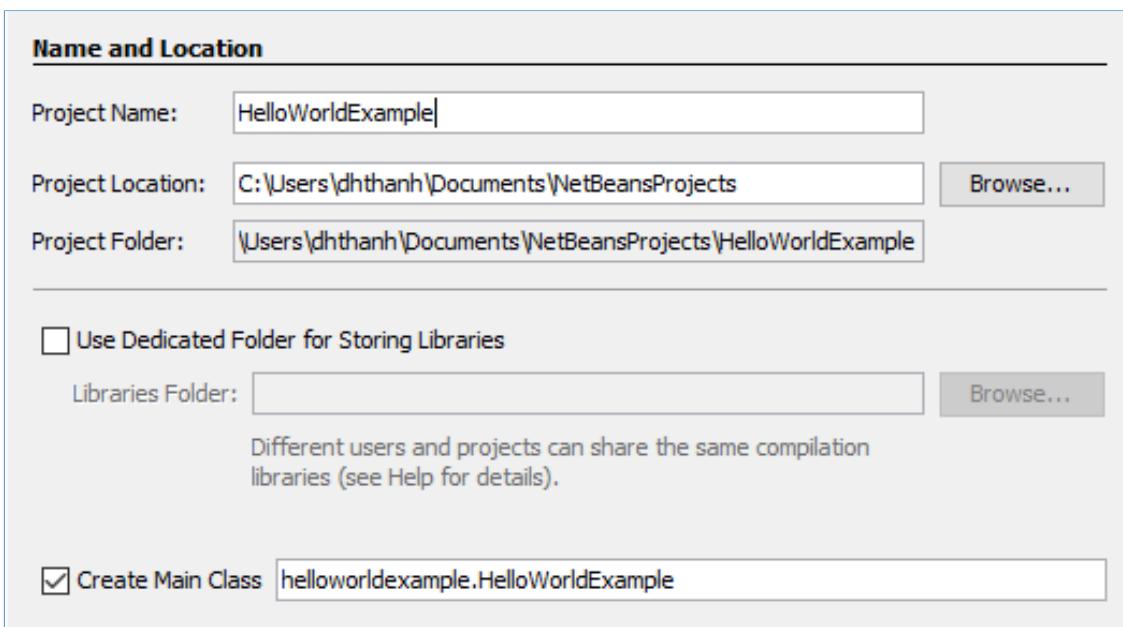
Tạo project trong NetBeans: tại cửa sổ NetBeans, click vào menu File/New Projects... hoặc dùng tổ hợp phím Ctrl + Shift + N.



Một hộp thoại tạo project hiện ra, trong mục Categories chọn Java và mục Projects chọn Java Application và click Next.

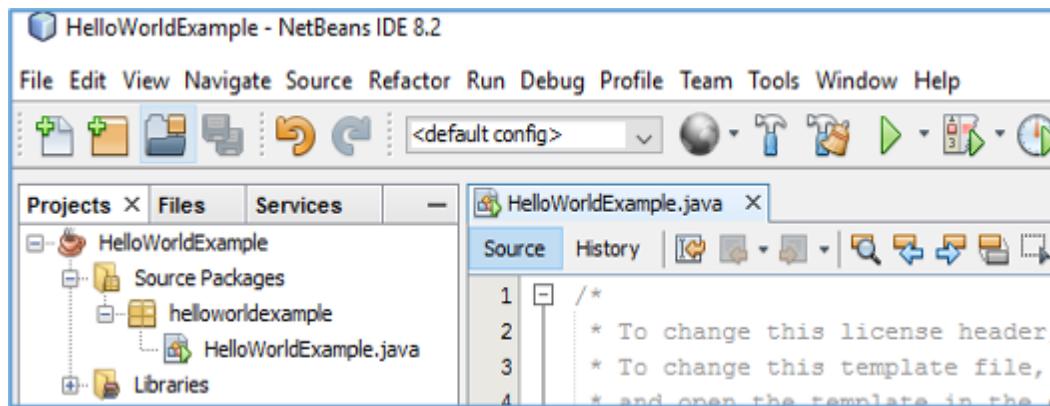


Đặt tên project là HelloWorldExample và click Finish.

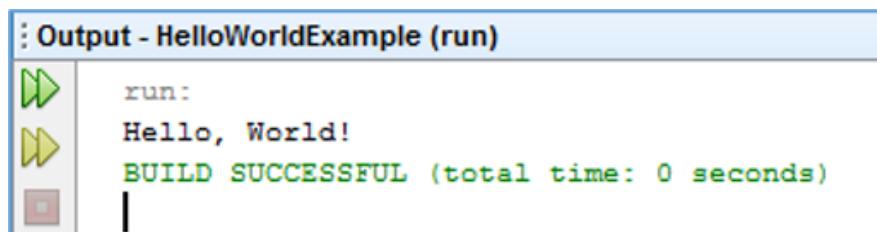


Tập tin HelloWorldExample.java sẽ tự động được tạo trong project. Trong phương thức main() thêm dòng lệnh:

```
System.out.println("Hello, World!");
```

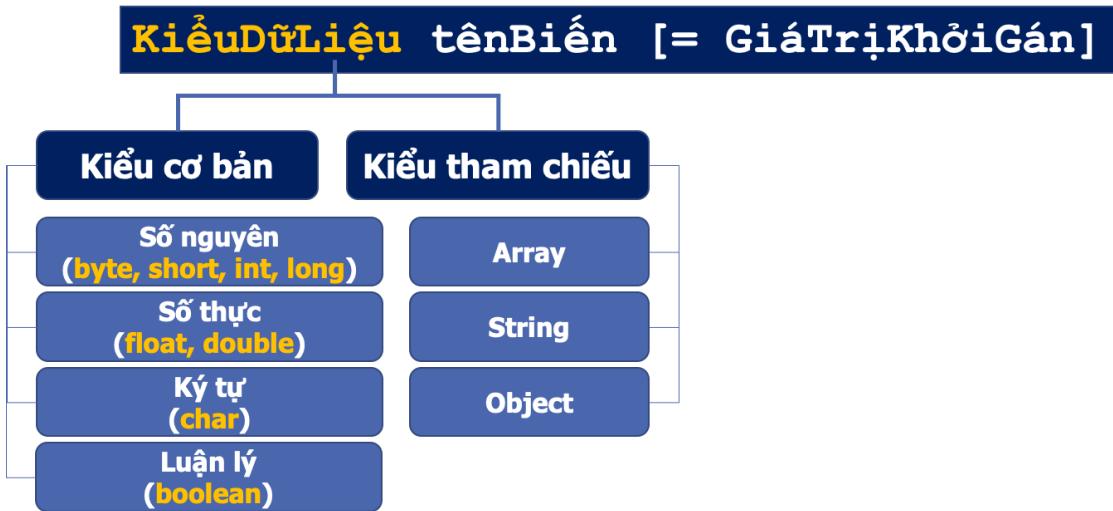


Để thực thi chương trình dùng tổ hợp phím Shift+F6 hoặc click chuột phải vào tập tin cần thực thi chọn “Run File” hoặc nút tam giác xanh trên thanh công cụ. Kết quả sẽ xuất dòng chữ “Hello, World!” trong cửa sổ console.



1.4 BIẾN VÀ KIỂU DỮ LIỆU

Biến là vùng nhớ để lưu trữ một giá trị trong chương trình, mỗi biến phải được khai báo một kiểu dữ liệu tường minh và một định danh duy nhất gọi là tên biến. Tên biến là chuỗi ký tự liên tiếp nhau bao gồm ký tự (a..z, A..Z), số (0..9), dấu gạch chân (_), dấu đôla (\$). Tên biến không thể bắt đầu bằng ký tự số (0..9), không chứa các ký tự đặc biệt và không trùng với các từ khóa như class, int, private, interface.

**Hình 2. Kiểu dữ liệu trong Java.**

Java chia các kiểu dữ liệu thành hai loại: kiểu dữ liệu cơ bản (primitive types) và kiểu dữ liệu tham chiếu (reference types).

- Các biến có kiểu dữ liệu cơ bản sẽ lưu các giá trị biến đó. Java cung cấp 8 kiểu dữ liệu cơ bản đại diện cho số nguyên, số thực, ký tự và giá trị luận lý (true/false).
- Các biến có kiểu dữ liệu tham chiếu lưu trữ địa chỉ của đối tượng trong bộ nhớ máy tính.

Ví dụ

```

int soLuong;
int dem = 1;
double heSo = 1.67;
String ketQua;
  
```

Java cũng cung cấp cách thức khai báo các biến khởi gán giá trị một lần để sử dụng cho tất cả, không được phép thay đổi giá trị trong suốt quá trình chạy chương trình, những biến như vậy gọi là hằng có tên (named constant) hay đơn giản gọi là hằng.

```
final KiểuDữLiệu TÊN BIẾN = GiáTrịKhởiGán
```

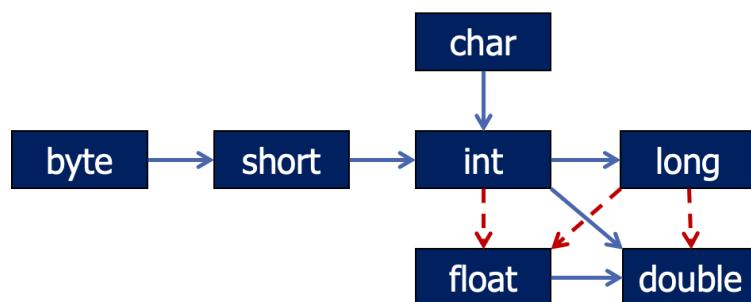
Ví dụ

```
final int SO_LUONG = 30;
```

```
final double PI = 3.14;
```

Chuyển đổi giữa các kiểu dữ liệu

Khi thực hiện các phép toán hai ngôi (như +, -, *, /) các toán hạng sẽ được chuyển đổi cùng kiểu dữ liệu trước khi phép toán được thực thi. Nếu thực hiện phép toán hai ngôi giữa một số nguyên và một số thực thì số nguyên sẽ được chuyển đổi thành số thực. Nếu thực hiện phép toán hai ngôi giữa các số cùng kiểu thì số có kiểu dữ liệu nhỏ hơn sẽ được chuyển đổi thành số có kiểu dữ liệu lớn hơn.



→ Phép chuyển đổi không mất thông tin

- - > Phép chuyển đổi có thể mất tính chính xác

Hình 2.3. Chuyển đổi kiểu dữ liệu.

Ép kiểu (casting)

Khi thực hiện các phép chuyển đổi kiểu dữ liệu có khả năng mất thông tin thì cần phải chỉ định một cách tường minh, gọi là ép kiểu (casting). Cú pháp ép kiểu trong Java:

```
(<kiểu-dữ-liệu>) <tên-biến>;
```

1.5 CÁC PHÉP TOÁN

Nhóm phép toán	Ký hiệu	Ví dụ
Phép toán số học	+,-,*,/,%	5 % 3 2 5.0 / 2 2.5

Phép gán	= +=, -=, *=, /=, %=	int a = 1; a++ + ++a 4
Phép tăng, giảm	++, --	
Phép toán quan hệ	==, !=, >, >=, <, <=	5 != 5 false “5” != 5 true
Phép luận lý	&&, , !	
Các hàm toán học	Math.<tên-hàm>()	Math.abs(-5) 5 Math.sqrt(4) 2 Math.pow(2, 3) 8

Bảng 2. Các phép toán trong Java.

1.6 CẤU TRÚC ĐIỀU KHIỂN

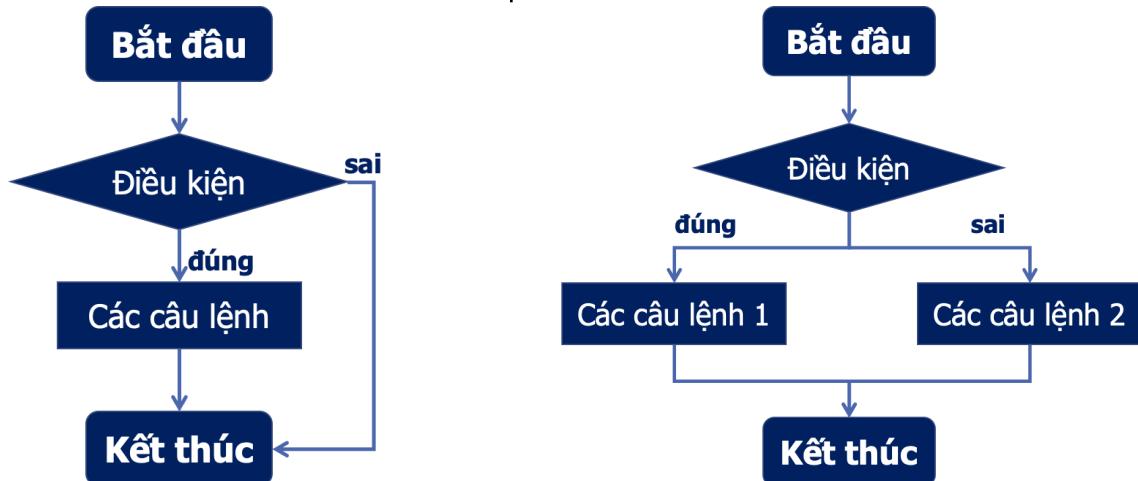
Lệnh if và if/else

Lệnh if

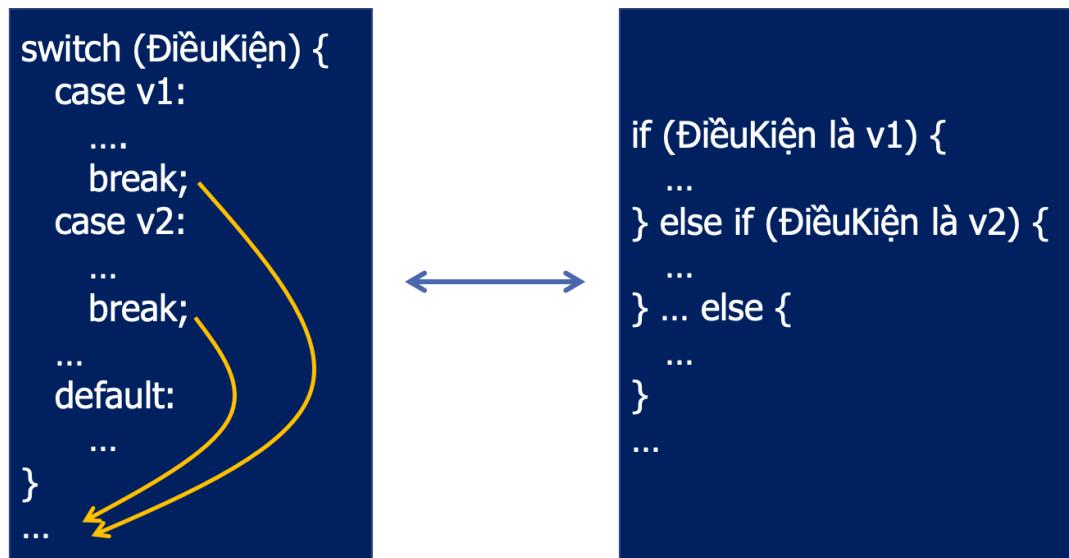
```
if (<Điều kiện>) {
    <Các câu lệnh>;
}
```

Lệnh if/else

```
if (<Điều kiện>) {
    <Các câu lệnh 1>;
} else {
    <Các câu lệnh 2>;
}
```

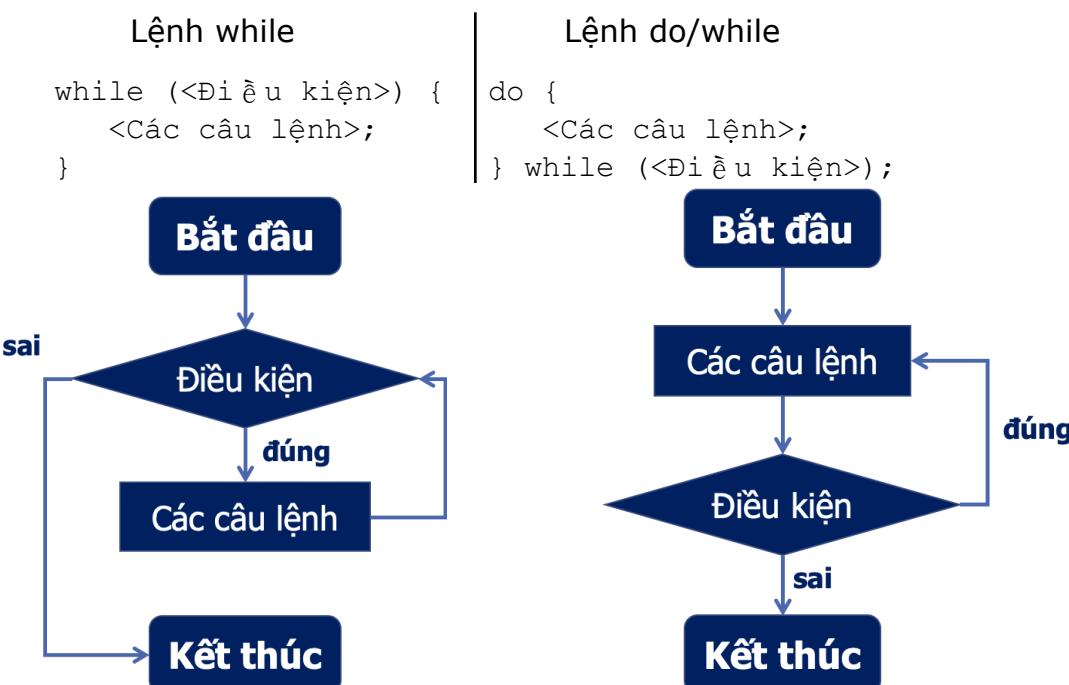


Lệnh switch/case



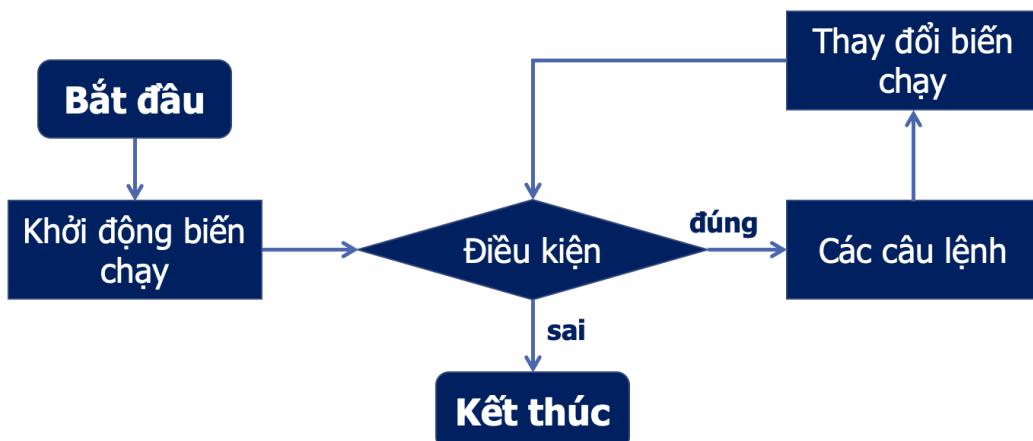
Khi thực thi, giá trị <Biểu thức điều kiện> sẽ được tính toán và so sánh với các giá trị hằng vi ($1 \leq i \leq n$) trong các case, nếu khớp giá trị của case nào thì câu lệnh khối i của case đó được thực thi cho đến khi hết lệnh switch hoặc gặp `break` thì thoát khỏi switch, trong trường hợp không có giá trị của case nào khớp với giá trị của <ĐiềuKiện> thì khối lệnh của `default` sẽ được thực thi.

Lệnh while và do/while

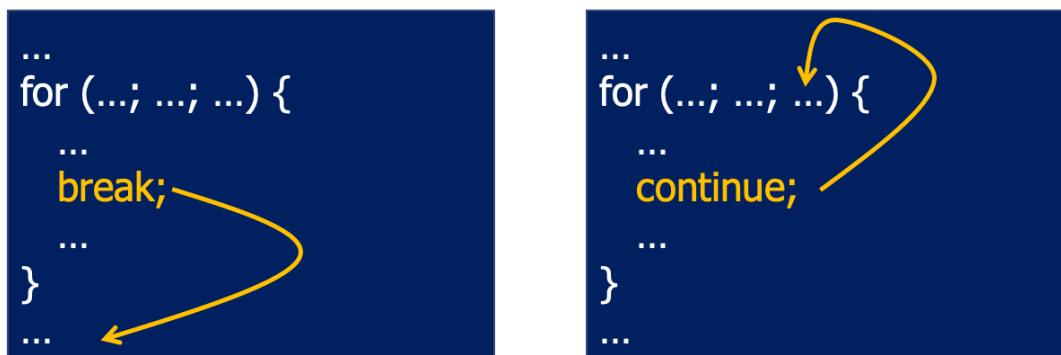


Điểm khác nhau chính giữa 2 lệnh lặp này là phần thân của vòng lặp do/while sẽ được thực hiện ít nhất một lần, còn vòng lặp while có thể sẽ không thực hiện lần nào.

Lệnh for



Lệnh break và continue



Lệnh continue chỉ được sử dụng trong các lệnh lặp, còn lệnh break ngoài dùng trong các lệnh lặp, còn có thể dùng trong lệnh switch để thoát khỏi switch.

1.7 CHUỖI

1.7.1 Giới thiệu chuỗi

Chuỗi trong Java là một dãy các ký tự unicode, chẳng hạn chuỗi "Java\u2122" chứa 5 ký tự J, a, v, a, TM. Java không cung cấp kiểu dữ liệu string, nhưng trong thư viện chuẩn của Java cung cấp lớp String để thao tác trên chuỗi.

Lớp String có rất nhiều phương thức khởi tạo và hơn 40 phương thức khác để thao tác với chuỗi. Ta có thể tạo ra đối tượng chuỗi từ một string literal hoặc từ một mảng các ký tự thông qua hai phương thức khởi tạo sau:

```
String(String <chuỗi-ký-tự>)
String(char[] <mảng-ký-tự>)
```

Ví dụ

```
String s1 = new String("Java"); // (1)
char[] c = {'J', 'a', 'v', 'a'};
String s2 = new String(c); // (2)
```

1.7.2 Các thao tác thông dụng trên chuỗi

Nối hai chuỗi: để nối hai chuỗi có thể sử dụng một trong các cách

- Sử dụng phép toán +
- Sử dụng phương thức concat() của lớp String.

Ví dụ

```
String sub1 = "Hello";
String sub2 = "World";

// Nối hai chuỗi bằng phép toán +
String concat = sub1 + " " + sub2;
System.out.println(concat); // Hello World

// Sử dụng +=
sub1 += ", Java!";
System.out.print (sub1); // Hello, Java!
// Sử dụng concat
sub1 = sub1.concat(" Good.");
System.out.print (sub1); // Hello, Java! Good.
```

So sánh chuỗi:

- s1.equals(s2): trả về true nếu nội dung hai chuỗi s1 và s2 giống nhau.

- `s1.equalsIgnoreCase(s2)`: giống `equals()` nhưng không phân biệt hoa, thường.
- `s1.compareTo(s2)`: trả về giá trị 0, lớn hơn 0, nhỏ hơn 0 tương ứng nếu chuỗi `s1` bằng, lớn hơn, nhỏ hơn chuỗi `s2`.
- `s1.compareToIgnoreCase(s2)`: giống `compareTo()` nhưng không phân biệt hoa, thường.
- `s1.startsWith(s2)`: trả về true nếu chuỗi `s1` bắt đầu bằng chuỗi `s2`.
- `s1.endsWith(s2)`: trả về true nếu chuỗi `s1` kết thúc bằng chuỗi `s2`.
- `s1.contains(s2)`: trả về true nếu chuỗi `s1` chứa chuỗi `s2`.

Ví dụ

```
String str1 = "Hello";
String str2 = "HELLO";
str1.equals(str2); // -> false
str1.equals("Hello"); // -> true
str1.equalsIgnoreCase(str2); // -> true
str1.startsWith("Hel"); // -> true
str1.startsWith("HeL"); // -> false
str2.endsWith("LLO"); // -> false
str1.compareTo(str2); // 32, do 'e' - 'E' = 32
str1.compareToIgnoreCase(str2); // 0
```

Tìm chuỗi:

- `s.indexOf(<char>/<string> [, <fromIndex>])`: trả về vị trí đầu tiên tìm thấy ký tự/chuỗi chỉ định trong chuỗi `s` tính từ vị trí `fromIndex` (mặc định là 0), và trả về -1 nếu không tìm thấy.
- `s.lastIndexOf(<char>/<string> [, <endIndex>])`: trả về vị trí cuối cùng tìm thấy ký tự/chuỗi chỉ định trong chuỗi `s` từ đầu chuỗi cho đến `endIndex - 1` (mặc định `endIndex` là chiều dài chuỗi) và trả về -1 nếu không tìm thấy.

Ví dụ

```

String s = "Welcome to Java! Java is great!";
System.out.print(s.indexOf("Java")); // 11
System.out.print(s.indexOf("Java", 15)); // 17
System.out.print(s.indexOf("Java", 25)); // -1
System.out.print(s.lastIndexOf("Java")); // 17
System.out.print(s.lastIndexOf("Java", 14)); // 11

```

Lấy chuỗi con:

- `s.substring(beginIndex [, endIndex]):` phương thức này lấy một chuỗi con của chuỗi s, bắt đầu từ vị trí beginIndex đến endIndex – 1 (mặc định endIndex là chiều dài chuỗi). Nếu beginIndex hoặc endIndex lớn hơn chiều dài chuỗi, hoặc có giá trị âm, hoặc beginIndex > endIndex thì ngoại lệ `StringIndexOutOfBoundsException` được ném ra.

```

String hello = "Hello, World!";
String s = hello.substring(0, 5); // (1)
s = hello.substring(7, 12); // (2)
s = hello.substring(7); // (3)

```

Định dạng chuỗi: phương thức tĩnh `format()` được sử dụng để định dạng chuỗi về một định dạng cho trước.

```
String.format(<chuỗi-định-dạng>, <gt1>, <gt2>, ...);
```

Ví dụ

```

// In chuỗi với định dạng chỉ định
String s = String.format("%7.2f", 45.3333);
System.out.println(s); // 45.33
// Lệnh sau tương đương hai lệnh trên
System.out.printf("%7.2f", 45.3333); // 45.33

```

Chuyển kiểu dữ liệu khác sang chuỗi và ngược lại

- Lớp String cung cấp phương thức `valueOf()` chuyển đổi kiểu ký tự, kiểu số, boolean hay đối tượng thành một chuỗi.
- Mỗi kiểu dữ liệu cơ bản sử dụng phương thức `parseT()` của lớp T để chuyển chuỗi thành kiểu dữ liệu cơ bản tương ứng, trong đó T là wrapper class.

```
// Đổi mảng ký tự thành chuỗi  
char[] c = {'4', '5'};  
String s = String.valueOf(c);  
// Đổi chuỗi thành số  
System.out.println(Integer.parseInt(s) + 15); // 60  
// Đổi chuỗi thành boolean  
if (Boolean.parseBoolean("true") == true)  
    System.out.println("T");
```

1.7.3 Sử dụng StringBuilde

Khi xây dựng chuỗi từ các chuỗi ngắn hơn có thể sử dụng phép nối chuỗi, nhưng cách này không hiệu quả vì mỗi lần nối chuỗi thì đối tượng chuỗi mới được tạo ra tốn thời gian và lãng phí bộ nhớ. Để khắc phục điều này, Java cung cấp lớp StringBuilder cho mục đích, lớp này tương tự như lớp String, nhưng có thể thay đổi nội dung của các đối tượng của StringBuilder.

Ví dụ

```
StringBuilder builder = new StringBuilder();  
  
// Nối một ký tự  
builder.append('A');  
// Nối chuỗi  
builder.append(" great method!");  
// Lấy về chuỗi  
String result = builder.toString();
```

1.8 MẢNG

1.8.1 Giới thiệu mảng

Mảng là một cấu trúc dữ liệu lưu tập các giá trị cùng kiểu, mảng một khi được tạo ra có kích thước cố định. Sử dụng một trong hai cú pháp khai báo mảng:

```
<kiểu-dữ-liệu>[] <tên-biến-mảng>;
```

```
<kiểu-dữ-liệu> <tên-biến-mảng>[];
```

Ví dụ

```
// Khai báo mảng, rồi cấp phát vùng nhớ
int[] a;
a = new int[10];

// Vừa khai báo, vừa cấp phát vùng nhớ
int[] b = new int[10];
```

Khởi tạo các giá trị cho mảng

```
int[] a = {6, 5, 7, 8, 9};
double[] b = new double[] {1.0, 2.0, 5.0};
```

1.8.2 Các thao tác trên mảng

Lấy chiều dài mảng

Sử dụng thuộc tính length của mảng.

Truy cập vào một phần tử mảng

Để truy cập vào một phần tử của mảng sử dụng chỉ số mảng, mảng được đánh chỉ số từ 0.

Duyệt các phần tử của mảng

Để duyệt qua các phần tử mảng có thể dùng vòng lặp for với chỉ số chạy qua các phần tử, hoặc sử dụng vòng lặp for..in với cú pháp như sau:

```
for (<biến-chạy>: <biến-mảng>) {
    <các-câu-lệnh>;
}
```

Sao chép mảng

- Sử dụng phương thức copyOf() của lớp Arrays.

```
int[] d = {6, 5, 7, 8, 9, 2};

// sao chép d.length phần tử từ mảng d -> c
```

```
int[] c = Arrays.copyOf(d, d.length);
c[3] = 100; // d[3] không ảnh hưởng
System.out.println(d[3]); // -> 8
```

- Sử dụng phương thức `arraycopy()` của lớp `System`.

```
int[] d = {6, 5, 7, 8, 9, 2};
int[] c = new int[6];
System.arraycopy(d, 0, c, 0, d.length);
c[3] = 100; // d[3] không ảnh hưởng
System.out.println(d[3]); // -> 8
```

Sắp xếp mảng

Để sắp xếp mảng sử dụng phương thức `sort()` của lớp `Arrays`.

```
int[] arr = {5, 2, 9, 8, 4};

//Sắp xếp tăng một phần mảng: từ vị trí 2->4
Arrays.sort(arr, 2, 5);
for (int v: arr) // 5 2 4 8 9
    System.out.print(v + "\t");

//Sắp xếp tăng toàn mảng
Arrays.sort(arr);
for (int v: arr) // 2 4 5 8 9
    System.out.print(v + "\t");
```

Mặc định phương thức `sort()` sẽ sắp xếp tăng danh sách, để thực hiện sắp xếp giảm như sau:

```
Integer[] arr = {5, 2, 9, 8, 4};
Arrays.sort(arr, Collections.reverseOrder());
for (int v: arr) // 9 8 5 4 2
    System.out.print(v + "\t");
```

1.9 DATE

Java cung cấp lớp Date (java.util.Date) để thao tác với dữ liệu ngày, giờ. Lớp này cung cấp hai phương thức khởi tạo:

- Date(): tạo thể hiện Date là ngày, giờ hiện tại.
- Date(long millisecond): tạo thể hiện Date với đối số là số mili giây tính từ ngày 01/01/1970.

```
Date d1 = new Date();

// Số mili giây từ 01/01/1970 00:00:00 -> hiện tại
System.out.println("Số mili giây: " + d1.getTime());

Date d2 = new Date(System.currentTimeMillis()-500);
System.out.println(d2.toString());
```

Định dạng ngày tháng dùng SimpleDateFormat

```
Date d = new Date();
SimpleDateFormat f
= new SimpleDateFormat("dd/mm/yyyy 'at' hh:mm:ss");
System.out.println(f.format(d));
```

Chuyển chuỗi thành Date

```
SimpleDateFormat f
= new SimpleDateFormat("dd/mm/yyyy");
Date d = f.parse("09/04/2019");
System.out.printf("Số mili giây: %d\n", d.getTime());
```

Sử dụng lớp GregorianCalendar tương tác dữ liệu ngày tháng

```
GregorianCalendar c = new GregorianCalendar();
System.out.println(c.get(Calendar.YEAR));
System.out.println(c.get(Calendar.MONTH));
System.out.println(c.get(Calendar.DAY_OF_MONTH));
System.out.println(c.get(Calendar.HOUR));
```

```
System.out.println(c.get(Calendar.MINUTE));
System.out.println(c.get(Calendar.SECOND));
System.out.println(c.get(Calendar.AM_PM));
if (c.isLeapYear(2020))
    System.out.println("Năm nhuận");
c.add(Calendar.DAY_OF_MONTH, 1);
System.out.println("Ngày sau: " + c.getTime());
```

1.10 NHẬP XUẤT TRONG CONSOLE

Java dùng System.out và System.in cho thiết bị nhập/xuất chuẩn. Để xuất ra console dùng phương thức print hoặc println của System.out.

Java không trực tiếp hỗ trợ đọc từ console, nhưng dùng lớp java.util.Scanner tạo đối tượng đọc dữ liệu từ System.in.

Ví dụ

```
System.out.print("Nhập n = ");
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();
int dem = 0;

while (n != 0) {
    if ((n % 10) % 2 != 0)
        dem++;
    n /= 10;
}

System.out.println("So chu so le cua n: " + dem);
```

1.11 ĐỌC GHI TẬP TIN VĂN BẢN

1.11.1 Đọc tập tin văn bản

Sử dụng `java.util.Scanner` để đọc tập tin văn bản.

```
File f = new File(<filename>);
Scanner scanner = new Scanner(f);
```

Ngoại lệ `FileNotFoundException` được ném ra nếu tập tin không tồn tại.

Sử dụng các phương thức `next()`, `nextInt()`, `nextDouble()` để đọc dữ liệu tương ứng.

Sử dụng phương thức `close()` đóng tập tin khi không dùng.

Ví dụ ta có tập tin `data.txt` như sau



Đọc tập tin này

```
File f = new File("data.txt");

try (Scanner scanner = new Scanner(f)) {
    String hoTen = scanner.nextLine();
    int mssv = scanner.nextInt();
    double diemTB = scanner.nextDouble();

    System.out.printf("MSSV: %d\nHo ten: %s\n" +
        "Diem TB: %.2f\n", mssv, hoTen, diemTB);
}
```

1.11.2 Ghi tập tin văn bản

Sử dụng lớp `java.io.PrintWriter` để ghi tập tin văn bản.

```
File f = new File(<filename>);  
PrintWriter p = new PrintWriter(f);
```

Nếu tập tin không tồn tại thì nó sẽ được tạo ra, ngược lại thì nội dung tập tin sẽ bị huỷ.

Sử dụng các phương thức print(), println() để ghi dữ liệu.

Sử dụng phương thức close() đóng tập tin khi không dùng.

Ví dụ

```
File f = new File("data.txt");  
try (PrintWriter writer = new PrintWriter(f)) {  
    writer.println("Duong Huu Thanh");  
    writer.println(1);  
    writer.println(8.9);  
}
```

Để mở tập tin ghi tiếp sử dụng FileWriter

```
File f = new File("data.txt");  
FileWriter w = new FileWriter(f, true);  
  
try (PrintWriter writer = new PrintWriter(w)) {  
    writer.println(Math.random());  
}
```

BÀI 2. TỔNG QUAN SPRING FRAMEWORK

Học xong bài này người học sẽ nắm được các nội dung sau:

- *Hiểu tổng quan Spring Framework cho phát triển ứng dụng Java EE.*
- *Hiểu tổng quan các khái niệm, kiến trúc của Spring MVC.*
- *Cài đặt môi trường phát triển ứng dụng Web với Spring MVC.*
- *Viết chương trình đầu tiên với Spring MVC.*

1.12 TỔNG QUAN SPRING FRAMEWORK

Spring là một framework phát triển ứng dụng phổ biến cho Java EE dễ dàng và nhanh chóng, nó là một nền tảng mã nguồn mở được phát triển bởi Rod Johnson phát hành đầu tiên vào trong Apache 2.0 vào năm tháng 06/2003. Sử dụng Spring, ta có thể phát triển các ứng dụng độc lập, ứng dụng desktop, ứng dụng Web, ứng dụng phân tán, các ứng dụng thương mại, v.v. Spring cung cấp các hạ tầng toàn diện để phát triển các ứng dụng Java EE, nó xử lý tất cả các vấn đề hạ tầng và lập trình viên chỉ tập trung vào việc phát triển ứng dụng.

Các đặc trưng quan trọng của Spring

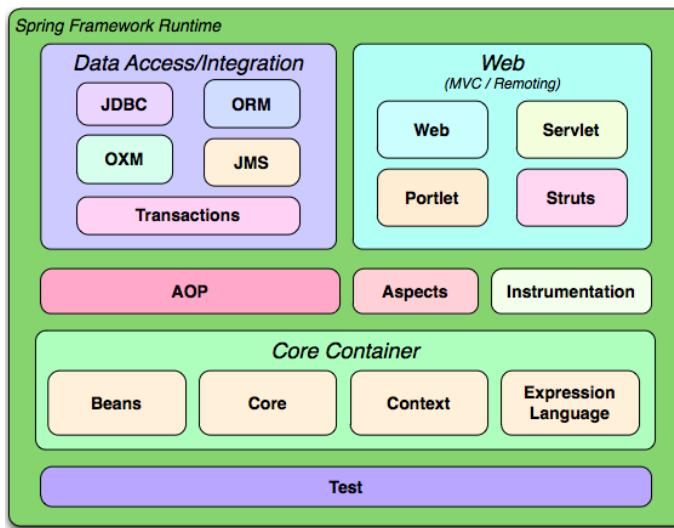
- **Lightweight:** Spring là một framework nhẹ về mặt kích thước (size) và độ trong suốt (transparent) khi phát triển ứng dụng. Điều này giúp giảm bớt độ phức tạp mã nguồn chương trình, chương trình bắt đầu nhanh hơn và chạy được trong nhiều môi trường.
- **Non-intrusive:** các đối tượng trong ứng dụng Spring không phụ thuộc các giao diện (interface) hay các lớp định nghĩa sẵn trong Spring API, nên Spring có thể cấu hình các đối tượng mà không cần import Spring API.
- **Inversion of Control (IoC):** IoC là một kỹ thuật lập trình mà việc mốc nối đối tượng (object coupling) được kết buộc với đối tượng ráp (assembler object) vào lúc thực thi (runtime) chương trình, thường không được biết trong lúc biên dịch (compile

time). IoC giúp chương trình dễ quản lý, dễ kiểm thử, và khả chuyển. Spring framework hiện thực IoC hỗ trợ Injection.

- Aspect-oriented Programming (AOP): AOP là mô hình lập trình cō lập các chức năng hỗ trợ từ logic xử lý của chương trình chính, nó cho phép lập trình viên xây dựng các chức năng của hệ thống mà không cần quan tâm sự tồn tại của các yêu cầu bổ sung. AOP sử dụng trong Spring Framework để cung cấp các khai báo theo khía cạnh như transaction hay security. Các đối tượng thực hiện các logic nghiệp vụ, không quan tâm đến các xử lý liên quan khác như logging, locking, hay event handling.
- JDBC exception handling: tăng trừu tượng JDBC cung cấp xử lý ngoại lệ trong JDBC, điều này giúp giảm bớt rất nhiều mã nguồn ta cần viết xử lý ngoại lệ trong JDBC.
- Spring MVC Framework: cung cấp khả năng xây dựng ứng dụng Web mạnh mẽ và dễ bảo trì, nó sử dụng IoC cung cấp sự tách biệt trong xử lý logic của controller.
- Spring Security: cung cấp cơ chế khai báo security cho các ứng dụng Spring, đây là phần rất quan trọng trong các ứng dụng.

1.13 KIẾN TRÚC CỦA SPRING FRAMEWORK

Trong phần này trình bày chi tiết các module khác nhau trong Spring Framework. Hình 2.4 là tổng quan các module có trong Spring Framework.



Hình 2.4. Kiến trúc Spring Framework (nguồn: <https://spring.io>)

❖ Core Container

- Core module: đây là thành phần quan trọng nhất của Spring Framework cung cấp các đặc trưng như IoC, DI.
- Bean module: cung cấp BeanFactory là mẫu thiết kế Factory tổng quát phân tách sự phụ thuộc (dependencies) như khởi động, tạo đối tượng và việc truy cập vào các đối tượng từ logic chương trình. BeanFactory hỗ trợ hai phạm vi cho đối tượng:
 - o Singleton: chỉ tạo một thể hiện dùng chung của đối tượng, mỗi khi gọi phương thức `getBean()` trên ApplicationContext, Spring Singleton sẽ trả về cùng đối tượng Bean.
 - o Prototype (non-singleton): mỗi kết quả truy vấn sẽ tạo đối tượng mới, mỗi khi gọi phương thức `getBean()` của ApplicationContext, Spring Prototype sẽ tạo ra một thể hiện độc lập.
- Context module: ApplicationContext container sẽ nạp các định nghĩa Spring Bean và mốc nối (wire) chúng với nhau,

- SpEL (Spring Expression Language): là một ngôn ngữ biểu diễn (expression language) mạnh mẽ hỗ trợ các đặc trưng truy vấn và thao tác với một đối tượng lúc thực thi. SpEL hỗ trợ gọi các phương thức và truy vấn các đối tượng bằng tên từ IoC container trong Spring.

❖ **AOP (Aspect-oriented Programming) module**

AOP là một cơ chế cho phép ta thêm chức năng mới vào mã nguồn đã tồn tại không cần thay đổi thiết kế.

Module này cung cấp hiện thực AOP, là một framework dựa trên proxy (proxy-based framework) trong Java. Spring Framework sử dụng AOP cho hầu hết các logic hạ tầng (infrastructure logic).

❖ **Data Access/Integration**

- JDBC module: cung cấp JDBC abstraction framework giảm bớt một số mã nguồn lặp lại nhầm chán khi sử dụng JDBC. Lớp chính của framework này là JdbcTemplate chứa các logic chung trong sử dụng JDBC API để truy cập dữ liệu như tạo kết nối, tạo Statement, thực thi Statement và giải phóng tài nguyên.
- ORM (Object-relational mapping) module: cung cấp các API ORM, bao gồm JPA, Hibernate.
- OXM (Object XML Mapping) module: cung cấp tầng trừu tượng hỗ trợ thực thi ánh xạ Object/XML cho JAXB, Castor, XMLBeans, JiBX và XStream.
- JMS: cung cấp các đặc trưng để tạo (produce) và sử dụng (consume) các thông điệp (message).
- Transaction: hỗ trợ khai báo và lập trình quản lý giao tác cho các lớp hiện thực các interface đặc biệt và cho tất cả các lớp POJO.

❖ **Web**

- Web module: module này cho phép xây dựng Application Context module, bao gồm tất cả đặc trưng phát triển ứng dụng Web mạnh mẽ và dễ bảo trì.

- Servlet module: module này chứa hiện thực MVC giúp xây dựng các ứng dụng Web thương mại.
- Struts module: module này hỗ trợ sự tương thích tầng Strut Web trong ứng dụng Spring, nó cũng hỗ trợ cấu hình Struts Action sử dụng DI.
- Web-Portlet: module hỗ trợ phát triển các ứng dụng Web dễ dàng hơn, Portlet quản lý các Portlet container tương tự như Web container. Portlet được sử dụng trong tầng giao diện người dùng để hiển thị nội dung từ data source cho người dùng.

❖ Test module

Module này giúp kiểm thử các ứng dụng đang phát triển sử dụng jUnit hoặc TestNG, nó cũng giúp tạo các mock object để thực hiện Unit Test độc lập và hỗ trợ chạy kiểm thử tích hợp ngoài application server.

Ví dụ tạo project Java Application >> tạo gói springdemo >> tạo HelloWorld.java

```
public class HelloWorld {
    private String message;
    // Các phương thức getter/setter của message
}
```

Trong thư mục src tạo Beans.xml có nội dung như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id = "helloWorld" class = "springdemo.HelloWorld">
        <property name = "message" value = "Hello World!"/>
    </bean>
</beans>
```

Trong phương thức main()

```
ApplicationContext context
```

```

        = new ClassPathXmlApplicationContext("Beans.xml");
HelloWorld obj1 = (HelloWorld) context.getBean("helloWorld");
System.out.println(obj1.getMessage());

```

Kết quả thực thi chương trình

```
Hello World!
```

1.14 IoC CONTAINER

Trong công nghệ phần mềm, container dùng mô tả các thành phần có thể chứa các thành phần khác. Spring Container là thành phần quan trọng nhất trong Spring Framework, nó tạo các đối tượng, cấu hình, quản lý vòng đời của chúng. Container chính của Spring hiện thực IoC (Inversion of Control) hỗ trợ Injection.

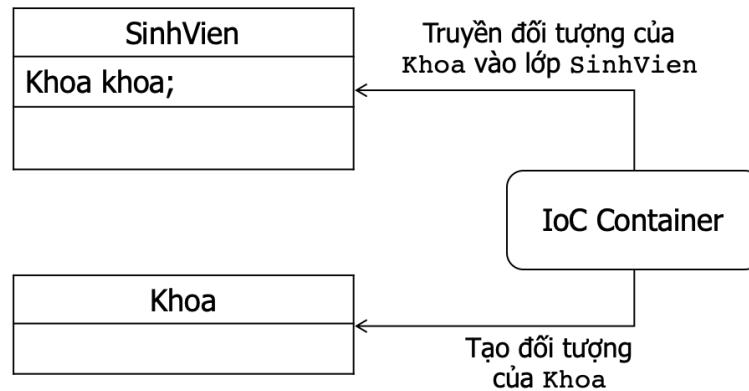
IoC là một kỹ thuật lập trình mà việc mốc nối đối tượng (object coupling) được kết buộc với đối tượng ráp (assembler object) vào lúc thực thi (runtime) chương trình, thường không được biết trong lúc biên dịch (compile time). IoC giúp chương trình dễ quản lý, dễ kiểm thử, và khả chuyển.

Xét ví dụ có hai lớp SinhVien và Khoa, lớp SinhVien có thuộc tính kiểu lớp Khoa để biết sinh viên thuộc khoa nào, như vậy lớp SinhVien phụ thuộc lớp Khoa. Vấn đề gặp phải khi viết chương trình cho các lớp là làm sao giữa được tính độc lập giữa các lớp (tight coupling) vì theo quan hệ phụ thuộc này thì khi thay đổi lớp Khoa thì lớp SinhVien cũng phải thay đổi theo.

Thông thường cách thức chung để tạo ra một đối tượng là dùng toán tử `new` giống như bên dưới, theo cách này nếu vì lý do nào đó đối tượng Khoa tạo không thành công thì việc tạo đối tượng SinhVien cũng thất bại.

SinhVien
Khoa khoa;
public SinhVien() { khoa = new Khoa(); }

Nguyên tắc cơ bản của IoC dựa trên nguyên tắc Hollywood: “Đừng gọi chúng tôi, chúng tôi sẽ gọi bạn” (Do not call us, we'll call you). Trong trường hợp này, lớp Khoa sẽ nói với lớp SinhVien là: “Đừng tạo tôi, tôi sẽ tự tạo cho một số người khác dùng nữa”. IoC container có trách nhiệm tạo đối tượng Khoa và truyền tham chiếu của nó vào lớp SinhVien.



Trong trường hợp SinhVien cần đối tượng Khoa để hoạt động, nó sẽ giao cho bên thứ ba quyết định thời điểm cần tạo và loại sử dụng để tạo thể hiện. Quá trình này gọi là Dependency Injection.

Spring Container sử dụng DI (Dependency Injection) (xem mục 1.14.2) để quản lý thành phần tạo nên ứng dụng, gọi là các đối tượng Beans. Spring Container có nhiệm vụ mốc nối (wire) các bean của ứng dụng bằng cách kết hợp các bean khác với nhau. Các siêu dữ liệu (metadata) cấu hình có thể sử dụng XML, Java annotation hoặc mã nguồn Java, siêu dữ liệu này sẽ giúp cho Spring Container quyết định đối tượng để khởi động, cấu hình và lắp ráp. Spring có hai loại Container phân biệt:

- Spring BeanFactory Container: đây là container cơ sở, tất cả các container khác đều phải hiện thực BeanFactory.
- Spring ApplicationContext Container: đây là một giao diện con của BeanFactory và được sử dụng phổ biến nhất trong các ứng dụng thương mại.

1.14.1 Application Context

org.springframework.context.ApplicationContext định nghĩa ApplicationContext và cung cấp các đặc trưng nâng cao để ứng dụng Spring thành các ứng dụng thương mại, còn BeanFactory chỉ cung cấp vài chức năng cơ bản. Một số lớp hiện thực ApplicationContext

- ClassPathXmlApplicationContext: định nghĩa bean được nạp bởi container từ tập tin XML trong classpath.
- FileSystemXmlApplicationContext: định nghĩa bean được nạp bởi container từ tập tin XML, nó yêu cầu đường dẫn tuyệt đối của tập tin cấu hình XML.
- XmlWebApplicationContext: được sử dụng tạo context cho ứng dụng Web bằng cách nạp tập tin cấu hình XML với các định nghĩa của các bean từ vị trí chuẩn trong thư mục webapp, mặc định tập tin cấu hình XML ở /WEB-INF/applicationContext.xml.
- AnnotationConfigApplicationContext: được sử dụng tạo context bằng cách nạp các lớp Java được gắn annotation @Configuration thay vì sử dụng các tập tin XML. Lớp này được sử dụng khi định nghĩa các cấu hình bean dựa trên mã nguồn Java.
- AnnotationConfigWebApplicationContext: được sử dụng tạo web application context bằng cách nạp các lớp Java gắn annotation @Configuration.

1.14.2 Dependency Injection

Dependency Injection (DI) là một mẫu thiết kế mà sự phụ thuộc (dependency) của đối tượng sẽ được truyền vào (injected) bởi framework mà không cần phải tạo đối tượng đó. Điều này giúp giảm bớt sự phụ thuộc (loose coupling) giữa nhiều đối tượng. Trong DI, framework chịu trách nhiệm đọc thông tin cấu hình, cấu hình và mã nguồn tác biệt, điều này giúp cho nhiều thực thi khác nhau có thể được cung cấp thông qua cấu hình mà không cần thay đổi mã nguồn phụ thuộc. Với DI giúp ứng dụng được phát triển có thể dễ thực hiện kiểm thử (testability), DI cho phép thay thế đối tượng

thực sự bằng mock object, điều này giúp cải thiện khả năng kiểm thử bằng cách viết unit test bằng jUnit sử dụng mock object.

Trong Spring Framework, DI được sử dụng xử lý sự phụ thuộc giữa các đối tượng, có hai loại được sử dụng:

- Setter Injection: sử dụng các phương thức setter trên bean, các phương thức setter được định nghĩa trong tập tin cấu hình của Spring. Để chỉ định theo cách này sử dụng thẻ <property> (xem ví dụ bên dưới) trong cấu hình bean.
- Constructor Injection: sử dụng phương thức khởi tạo với vài tham số, các tham số này được truyền vào (injected) tự động tại thời điểm tạo thể hiện. Để chỉ định theo cách này sử dụng thẻ <constructor-arg> (xem trong ví dụ bên dưới) trong cấu hình bean.

1.14.3 Sử dụng XML cấu hình beans

Tạo project Java Application (chạy console), trong project tạo gói springdemo, trong gói này tạo tập tin Beans.xml như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id = "helloWorld" class = "springdemo.HelloWorld">
        <property name = "message" value = "Hello World!" />
    </bean>
</beans>
```

Trong gói này tạo lớp HelloWorld.java như sau:

```
package springdemo;
public class HelloWorld {
    private String message;
    private String name;
```

```

public HelloWorld() {

}

public HelloWorld(String msg, String name) {
    this.message = msg;
    this.name = name;
}
// Các phương thức getter/setter
}

```

❖ Thẻ <property>

Đây là thẻ con của <beans> được sử dụng cho Setter Injection, thẻ này có ba thuộc tính quan trọng:

- name: tên thuộc tính Java Bean.
- value: giá trị thiết lập cho thuộc tính Java Bean, giá trị dạng chuỗi và sẽ được chuyển thành kiểu dữ liệu tương ứng sử dụng PropertyEditors của Java Bean.
- ref: tham chiếu tới đối tượng Bean khác.

❖ Một số thuộc tính quan trọng trong thẻ bean.

- class: thuộc tính bắt buộc dùng để chỉ định tên lớp dùng để tạo bean.
- name: chỉ định tên duy nhất cho bean, ngoài ra ta có thể sử dụng thuộc tính id với ý nghĩa tương tự khi sử dụng XML cấu hình.
- scope: thuộc tính này dùng chỉ định phạm vi hoạt động các đối tượng bean, mặc định thuộc tính có giá trị là singleton. Các giá trị của scope
 - o singleton: tạo một thể hiện beans duy nhất cho các lần sử dụng nó.
 - o prototype: tạo các thể hiện mới của beans mỗi khi cần sử dụng nó.
 - o request: tạo một thể hiện beans mới trong mỗi HTTP request, giá trị này chỉ có hiệu lực trong Spring Web ApplicationContext.

- o session: tạo một thể hiện beans mới trong mỗi HTTP session, giá trị này chỉ có hiệu lực trong Spring Web ApplicationContext.
- o global-session: chỉ làm việc trong môi trường portlet sử dụng Spring, nó sẽ tạo bean mỗi khi có portlet session mới.

Ví dụ cấu hình thông tin beans như sau:

```
<bean id="helloWorld" class="springdemo.HelloWorld" scope="singleton">
    <property name = "message" value = "Hello World!" />
</bean>
```

Trong phương thức main() sử dụng đối tượng beans.

```
ApplicationContext context
        = new ClassPathXmlApplicationContext("Beans.xml");
HelloWorld obj1 = (HelloWorld) context.getBean("helloWorld");
System.out.println(obj1.getMessage());
obj1.setMessage("Welcome!!!");

HelloWorld obj2 = (HelloWorld) context.getBean("helloWorld");
System.out.println(obj2.getMessage());
```

Kết quả chương trình

```
Hello World!
Welcome!!!
```

Tương tự ví dụ này, nhưng ta đổi scope của Beans thành prototype thì kết quả thực thi chương trình như sau:

```
Hello World!
Hello World!
```

- init-method: thuộc tính này chỉ định phương thức được gọi khi khởi tạo bean.
- destroy-method: thuộc tính chỉ định phương thức được gọi trước khi bean bị huỷ.

Ví dụ cấu hình thông tin beans như sau:

```
<bean id = "helloWorld" class = "springdemo.HelloWorld"
```

```

    init-method="init" destroy-method="destroy" >
<property name = "message" value = "Hello World!">
</bean>
```

Phương thức main() tạo beans. Chú ý nếu sử dụng Spring Container không phải trong môi trường các ứng dụng Web, ta cần đăng ký shutdown hook với JVM để đảm bảo phương thức destroy được gọi và tất cả các tài nguyên được giải phóng, trong ví ta gọi phương thức registerShutdownHook() của lớp AbstractApplicationContext.

```

AbstractApplicationContext context
                    = new ClassPathXmlApplicationContext("Beans.xml");
HelloWorld obj1 = (HelloWorld) context.getBean("helloWorld");
System.out.println(obj1.getMessage());
context.registerShutdownHook();
```

Kết quả thực thi chương trình

```

Welcome!!!
Hello World!
Goodbye!!!
```

Ngoài ra, ta có thể khai báo phương thức init và destroy mặc định cho tất cả các đối tượng beans trong thẻ beans như sau:

```

<beans ...
        default-destroy-method="destroy" default-init-method="init">
        ...
</beans>
```

❖ Thẻ <constructor_arg>

Đây là thẻ con của <beans> được sử dụng cho Constructor Injection, thẻ này có bốn thuộc tính quan trọng:

- index: chỉ định chỉ số trong danh sách đối số của constructor, điều này có ích trong trường hợp constructor có nhiều đối số cùng kiểu.
- type: chỉ định kiểu của đối số trong constructor.

- value: giá trị là chuỗi chỉ định giá trị cho bean, nó sẽ được chuyển thành kiểu dữ liệu tương ứng của đối số sử dụng PropertyEditors Java Bean.
- ref: tham chiếu tới bean khác.

Trong ví dụ trên sửa lại cấu hình Bean của helloWorld như sau:

```
<bean id = "helloWorld" class = "springdemo.HelloWorld">
    <constructor-arg value="Welcome " />
    <constructor-arg value="Thanh!" />
</bean>
```

Chạy thử một đoạn chương trình trong main() như bên dưới, khi đó đối tượng helloWorld được inject thông qua phương thức khởi tạo.

```
public static void main(String[] args) {
    ApplicationContext context
        = new ClassPathXmlApplicationContext("springdemo/Beans.xml");

    HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
    System.out.println(obj.getMessage() + obj.getName());
}
```

Kết quả thực thi chương trình

```
Welcome Thanh!
```

❖ Autowiring

Autowiring là một đặc trưng được hỗ trợ trong Spring Framework, nó cho phép ta không cần cung cấp chi tiết bean injection tường minh, giúp giảm bớt cấu hình Spring Bean và dự đoán một cách thông minh đối tượng nào được tham chiếu đến. Spring container có thể autowire quan hệ giữa các bean mà không cần sử dụng các thẻ <property> hay <constructor-arg>.

Đặc trưng này được thể hiện thông qua thuộc tính `autowire` của thẻ `<beans>`, mặc định thuộc tính này bị vô hiệu (disabled).

```
<bean id = "helloWorld" class = "springdemo.HelloWorld"
      autowire = "autowire-type">
```

Spring container có năm loại (autowire-type) chế độ autowiring:

- No: chế độ autowiring bị vô hiệu (mặc định).
- byname: autowiring dựa trên name, tức là tên của thuộc tính bean giống tên bean khác. Phương thức setter được sử dụng cho loại autowire này để inject một dependency.
- byType: autowiring dựa trên kiểu dữ liệu, tức là kiểu dữ liệu thuộc tính bean tương thích với kiểu dữ liệu của bean khác. Chú ý chỉ một bean được phép cấu hình theo kiểu này trong tập tin cấu hình, ngược lại sẽ có ngoại lệ ném ra.
- constructor: tương tự byType nhưng sử dụng constructor để inject một dependency.
- autodetect: Spring sẽ autowire bằng constructor, nếu không được Spring sẽ thử autowire bằng byType.

❖ Cấu hình cho Java Collection

Spring cung cấp 4 loại cấu hình Collections:

- <list>: kết nối với thuộc tính kiểu danh sách (list).
- <set>: kết nối với thuộc tính kiểu tập hợp (set), tức là không cho phép hai phần tử trùng nhau trong danh sách.
- <map>: kết nối thuộc tính kiểu Collection mà mỗi phần tử là một cặp key/value, trong đó key và value có thể có kiểu dữ liệu bất kỳ.
- <props>: tương tự map nhưng key và value có kiểu String.

Ví dụ tạo tập tin CollectionDemo.java có nội dung như sau:

```
public class CollectionDemo {
    private List demoList;
    private Set demoSet;
    private Map demoMap;
    private Properties demoProperties;
```

```
// Các phương thức getter/setter của các thuộc tính
}
```

Cấu hình đối tượng beans

```
<bean id="collectionDemo" class="springdemo.CollectionDemo">
    <property name="demoList">
        <list>
            <value>Apple</value>
            <value>Banana</value>
            <value>Apple</value>
        </list>
    </property>
    <property name="demoSet">
        <set>
            <value>Apple</value>
            <value>Banana</value>
            <value>Apple</value>
        </set>
    </property>
    <property name="demoMap">
        <map>
            <entry key="orange" value="Orange" />
            <entry key="lemon" value="Lemon" />
        </map>
    </property>
    <property name="demoProperties">
        <props>
            <prop key="blackberry">Blackberry</prop>
            <prop key="mango">Mango</prop>
        </props>
    </property>
</bean>
```

Trong phương thức main() sử dụng đối tượng bean

```

ApplicationContext context
        = new ClassPathXmlApplicationContext("Beans.xml");
CollectionDemo colDemo
        = (CollectionDemo) context.getBean("collectionDemo");
System.out.println("List: " + colDemo.getDemoList());
System.out.println("Set: " + colDemo.getDemoMap());
System.out.println("Map: " + colDemo.getDemoSet());
System.out.println("Properties: " + colDemo.getDemoProperties());

```

Kết quả thực thi chương trình

```

List: [Apple, Banana, Apple]
Set: {orange=Orange, lemon=Lemon}
Map: [Apple, Banana]
Properties: {blackberry=Blackberry, mango=Mango}

```

1.14.4 Sử dụng Annotation cấu hình beans

Thay vì sử dụng tập tin XML để cấu hình đối tượng beans, từ phiên bản Spring 2.5 cho phép sử dụng annotation để cấu hình trực tiếp trong các lớp liên quan.

Để sử dụng annotation, đầu tiên ta cần bắt nó lên trong tập tin cấu hình như sau:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context = "http://www.springframework.org/schema/context"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <context:annotation-config />
</beans>

```

Một số annotation quan trọng

- @Required: áp dụng cho các phương thức setter của các thuộc tính.

- **@Autowired:** áp dụng cho thuộc tính, phương thức khởi tạo, phương thức setter.
- **@Qualifier:** khi có nhiều đối tượng beans cùng kiểu, thì sử dụng annotation kết hợp với @Autowired để chỉ định mốc nối (wired) cụ thể đến đối tượng beans nào.

Ví dụ tạo lớp Category:

```
import org.springframework.beans.factory.annotation.Required;
public class Category {
    private String name;
    public String getName() {
        return name;
    }
    @Required
    public void setName(String name) {
        this.name = name;
    }
}
```

Lớp Product

```
import org.springframework.beans.factory.annotation.Autowired;
public class Product {
    private String name;
    @Autowired
    private Category category;
}
```

Cấu hình trong Beans.xml như sau:

```
<bean id="category" class="springdemo.Category">
    <property name="name" value="Mobile" />
</bean>
<bean id="product" class="springdemo.Product">
    <property name="name" value="iPhone 7 Plus" />
</bean>
```

Trong phương thức main() tạo đối tượng beans

```

ApplicationContext context
    = new ClassPathXmlApplicationContext("Beans.xml");
Product p = (Product) context.getBean("product");
System.out.printf("%s - %s\n", p.getName(), p.getCategory().getName());

```

Kết quả thực thi chương trình

iPhone 7 Plus - Mobile

Chú ý: nếu cấu hình đối tượng beans của Category không cấu hình giá trị cho thuộc tính name thì ngoại lệ BeanInitializationException sẽ được ném ra do annotation @Required trước phương thức setter của thuộc tính name trong lớp Category.

Ta cũng có thể khai báo @Autowired phương thức khởi tạo như sau:

```

public class Product {
    private String name;
    private Category category;

    @Autowired
    public Product(Category category) {
        this.category = category;
    }
    // Các phương thức getter/setter của các thuộc tính
}

```

Trong trường hợp có nhiều đối tượng beans cùng kiểu, ta cần sử dụng @Qualifier kết hợp @Autowired để chỉ định chính xác đối tượng bean móc nối (wired). Chẳng hạn trong Beans.xml cấu hình hai đối tượng beans cho Category như sau:

```

<bean id="category" class="springdemo.Category">
    <property name="name" value="Mobile" />
</bean>
<bean id="cate" class="springdemo.Category">
    <property name="name" value="Tablet" />
</bean>

```

Khi đó sử dụng @Autowired kết hợp @Qualifier trong lớp Product như sau:

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Product {
    private String name;
    @Autowired
    @Qualifier("cate")
    private Category category;
}

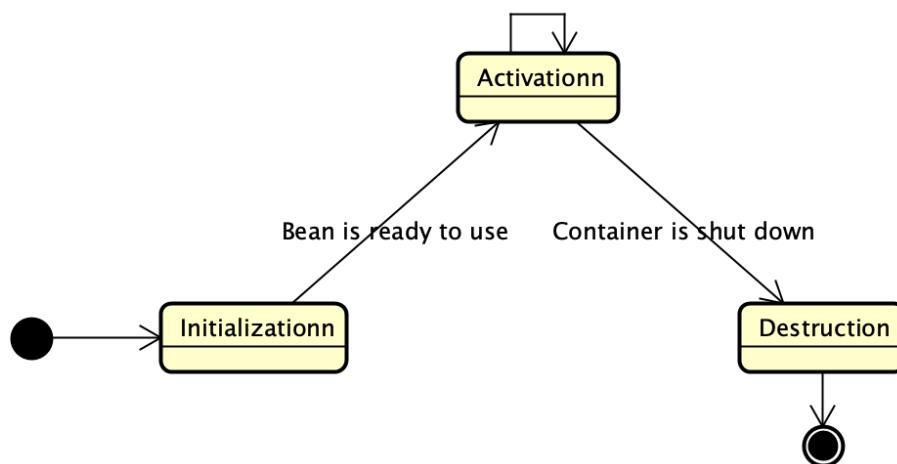
```

Thực thi lại đoạn chương trình trong phương thức main(), kết quả sẽ là:

iPhone 7 Plus - Tablet

1.15 VÒNG ĐỜI CỦA SPRING BEAN

BeanFactory quản lý vòng đời của các beans được tạo thông qua Spring IoC Container, vòng đời hoạt động của bean bao gồm nhiều hàm callback thực hiện trong hai thời điểm: sau khi khởi động (post-initialization) và trước khi huỷ (pre-destruction) bean.



Hình 5. Vòng đời của Spring Bean.

❖ Khởi tạo bean (Initialization)

- Đầu tiên, bean container tìm thấy định nghĩa bean trong tập tin cấu hình và tạo thể hiện của bean, nếu các thuộc tính được đề cập thì nó sẽ truyền các giá trị vào các thuộc tính bằng phương thức setter.

- Nếu lớp Bean hiện thực giao diện BeanNameAware thì sẽ gọi phương thức setBeanName().
- Nếu lớp Bean hiện thực giao diện BeanFactoryAware thì sẽ gọi phương thức setBeanFactory().
- Nếu lớp Bean hiện thực giao diện ApplicationContextAware thì sẽ gọi phương thức setApplicationContext().
- Nếu có đối tượng BeanPostProcessors kết hợp với giao diện BeanFactory, Spring sẽ gọi phương thức postProcessBeforeInitialization() trước khi các thuộc tính của bean được inject.
- Nếu lớp Bean hiện thực giao diện InitializingBean thì sẽ gọi phương thức afterPropertiesSet() một khi tất cả các thuộc tính của bean định nghĩa trong tập tin cấu hình được inject.
- Nếu định nghĩa bean trong tập tin cấu hình có khai báo thuộc tính init-method, thì phương thức chỉ định sẽ được gọi sau khi phân giải giá trị của thuộc tính với tên phương thức trong lớp Bean.
- Phương thức postProcessAfterInitialization() sẽ được gọi có vài post processor của bean kèm theo giao diện BeanFactory nạp bean.

❖ **Sử dụng bean (Activation)**

- Bean đã được khởi tạo và dependency đã được inject, bean sẵn sàng được sử dụng trong ứng dụng.

❖ **Huỷ bean (Destruction)**

- Nếu lớp Bean hiện thực giao diện DisposableBean thì sẽ gọi phương thức destroy() khi ứng dụng không còn tham chiếu nào đến bean nữa.
- Nếu định nghĩa của bean trong tập tin cấu hình có khai báo thuộc tính destroy-method thì phương thức sẽ được gọi sau khi phân giải giá trị của thuộc tính với tên phương thức trong lớp bean.

BÀI 3. SPRING MVC

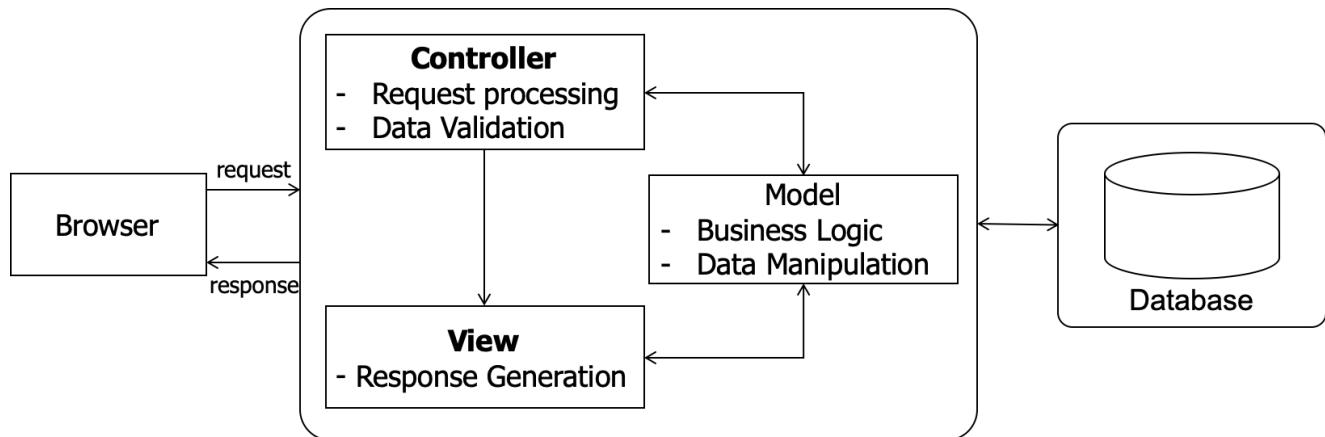
Học xong bài này người học sẽ nắm được các nội dung sau:

- *Hiểu tổng quan Spring MVC.*
- *Biết cơ chế hoạt động của ứng dụng Web phát triển bằng Spring MVC.*
- *Biết cách cài đặt môi trường phát triển.*
- *Biết cách tạo project, cấu trúc cơ bản của một project phát triển ứng dụng Web bằng Spring MVC.*

1.15.1 Giới thiệu Spring MVC

Spring MVC là một framework mã nguồn mở dùng phát triển các ứng dụng Web theo mô hình MVC (Model-View-Controller). Đây là một framework có tính linh hoạt, mạnh mẽ, thiết kế tốt, hỗ trợ một cách toàn diện và chuyên nghiệp trong việc phát triển ứng dụng Web. MVC là mẫu kiến trúc phát triển ứng dụng, nó tách biệt dữ liệu (Model) với kiến trúc người dùng (View), Controller chịu trách nhiệm kết hợp tương tác giữa View và Model. Controller xử lý các yêu cầu (action) từ người dùng, tương tác với Model và cập nhật dữ liệu hiển thị lên View. Ý tưởng chính của mẫu MVC là tách biệt phần xử lý nghiệp vụ với giao diện người dùng cho phép thay đổi một cách độc lập không ảnh hưởng nhau.

Spring MVC hiện thực tất cả các đặc trưng nổi bật của Spring Core như Inversion of Control, Dependency Injection. Phát triển các ứng dụng theo Spring MVC, models sẽ bao gồm các đối tượng domain được xử lý bởi tầng service và được lưu trữ bởi tầng persistence, view sử dụng JSP template được viết với JSTL (Java Standard Tag Library), ta cũng có thể định nghĩa các view là các tập tin pdf, excel hoặc các RESTful Web Service.



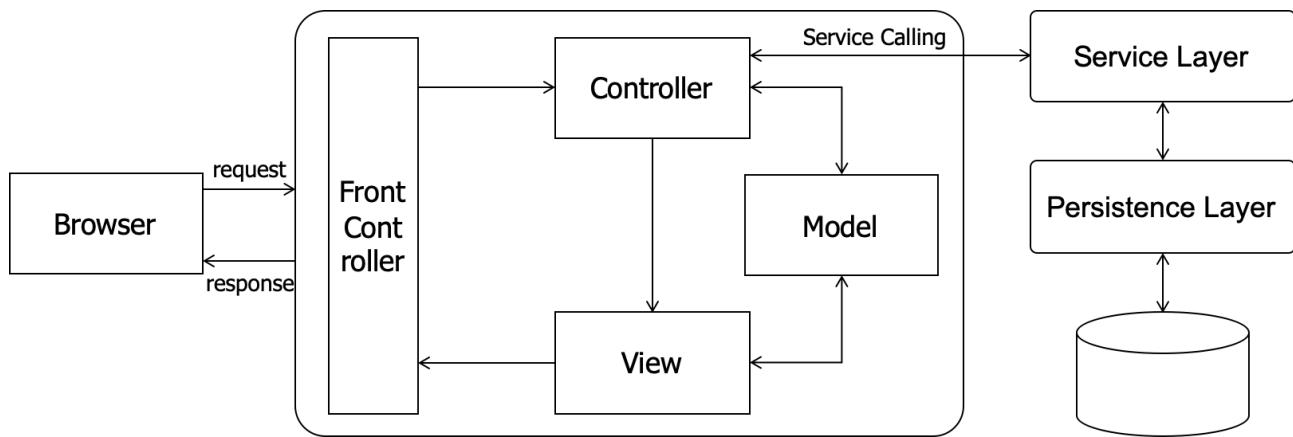
Hình 6. Mô hình MVC trong ứng dụng Web.

Các đặc trưng của Spring MVC

- Spring MVC cung cấp khả năng cấu hình mạnh mẽ, cũng như các lớp ứng dụng.
- Spring MVC giúp dễ dàng kiểm thử ứng dụng, hầu hết các lớp ứng dụng JavaBean cung cấp đều cho phép truyền dữ liệu kiểm thử vào sử dụng phương thức setter của các lớp JavaBean.
- Spring MVC cung cấp các thành phần xử lý chuyên biệt, mỗi thành phần trong Spring MVC có vai trò khác nhau trong xử lý request. Một request có thể được xử lý bởi các thành phần như Controller, Validator, Model Object, View, Resolver và HandlerMapping.
- Tránh tình trạng mã nguồn lặp lại (code duplication).
- Cho phép thực hiện kiểm tra dữ liệu (Data Form Validation) dễ dàng.

1.15.2 Front Controller Design Pattern

Front Controller là điểm bắt đầu xử lý của tất cả các HTTP request, nó cũng là nơi khởi động vài thành phần quan trọng của framework.

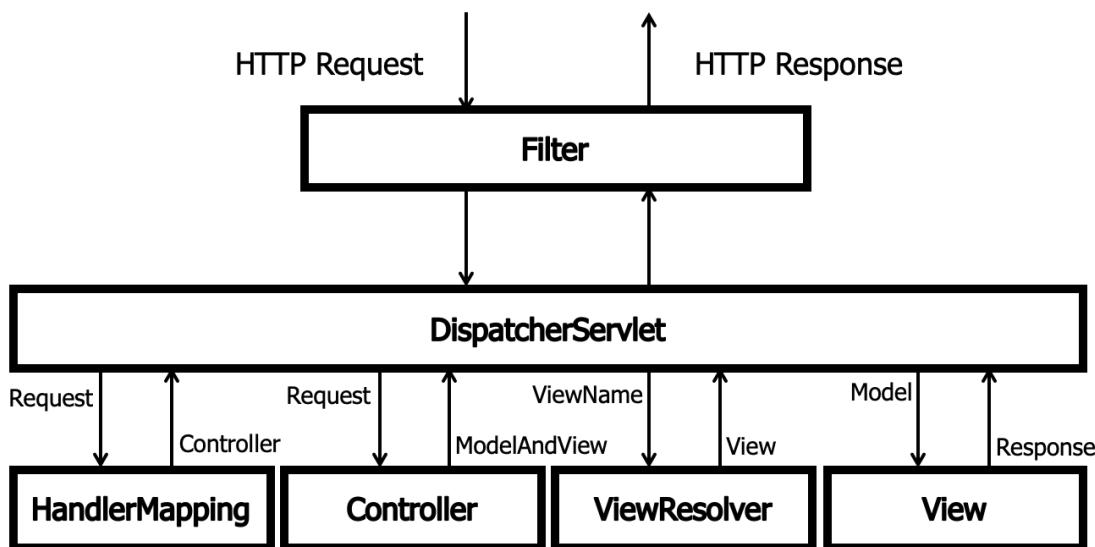


Hình 7. Minh họa Front Controller trong các ứng dụng Web.

Hình 7 cho thấy người dùng chỉ tương tác với một controller, đó là Front Controller. Front Controller nhận (intercept) request của người dùng, thực hiện các chức năng chung, chuyển (dispatch) request đến controller tương ứng dựa trên cấu hình của ứng dụng Web và thông tin của HTTP request. Controller tương tác với tầng dịch vụ (Service Layer) thực hiện các logic nghiệp vụ (business logic) và lưu trữ (persistence logic), sau đó cập nhật model và view sẽ kết xuất dữ liệu của model cho View hiển thị và trả View đó về cho người dùng. Cuối cùng, Front Controller phản hồi đến client dưới dạng một View. Trong Spring MVC, DispatcherServlet làm việc như Front Controller.

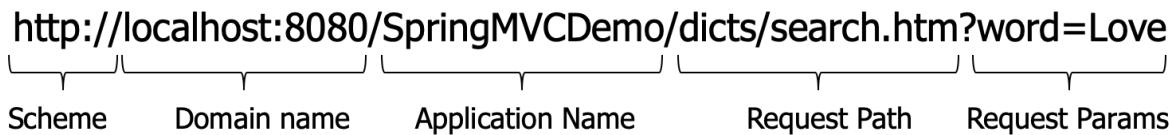
1.15.3 DispatcherServlet

DispatcherServlet (`org.springframework.web.servlet.`) xử lý tất cả các HTTP Request và HTTP Response, nó sẽ quyết định phương thức nào của controller sẽ được thực thi khi nhận một HTTP Request. Khi nhận được HTTP request, DispatcherServlet sẽ gọi Controller thích hợp dựa trên HandlerMapping. Controller nhận được request sẽ gọi phương thức thích hợp dựa trên phương thức request là POST hay GET. DispatcherServlet tìm view có sẵn cho request dựa trên ViewResolver, sau đó gửi dữ liệu đến view để kết xuất, hiển thị lên trình duyệt.



Hình 8. Cơ chế hoạt động của ứng dụng Spring MVC.

Trong Spring MVC, URL được chia làm 5 phần như bên dưới, khi người dùng thực hiện một request thì DispatcherServlet sẽ tìm kiếm phương thức trong controller phù hợp với phần Request Path.



Hình 9. Các thành phần trên URL trong ứng dụng Spring MVC.

Trong Spring MVC để chỉ định một lớp controller trong Spring sử dụng annotation `@Controller` hoặc `@RestController`. Khi một lớp gắn annotation là `@Controller` nhận một request, nó sẽ tìm kiếm phương thức xử lý thích hợp cho request đó thông qua annotation `@RequestMapping` chỉ định ánh xạ (mappings) giữa request với phương thức được gắn annotation này. Phương thức xử lý request có thể chứa tuỳ ý các loại tham số sau:

- `HttpServletRequest` hoặc `HttpServletResponse`.
- Các tham số trên URL với annotation `@RequestParam`.
- Các thuộc tính model với annotation `@ModelAttribute`.
- Các giá trị cookie đính kèm trong request với annotation `@CookieValue`.

- Map hoặc ModelMap để thêm các thuộc tính vào model.
- Errors hoặc BindingResult để truy cập vào các kết buộc và kết quả kiểm tra (validation) cho đối tượng command.
- SessionStatus để thông báo hoàn tất xử lý session.

Sau khi xử lý xong, phương thức sẽ giao quyền điều khiển cho View thông qua giá trị trả về của phương thức, phương thức xử lý có thể trả về giá trị kiểu String đại diện cho tên View hoặc void, trong trường hợp này View được chọn dựa trên tên phương thức hoặc tên controller.

❖ View Resolver

View Resolver dùng xác định các view được render để response cho một request từ client. View Resolver sẽ trả về URL cho DispatcherServlet, DispatcherServlet sẽ lấy tập tin view từ server để kết xuất hiển thị trên trình duyệt của client. Nếu trong thời gian này, DispatcherServlet không tìm thấy tập tin view hoặc URL bị lỗi thì sẽ trả về HTTP status 404.

Spring MVC cung cấp nhiều view resolver khác nhau để xác định view phù hợp, chẳng hạn InternalResourceViewResolver.

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

❖ Web Application Context

Trong các ứng dụng Spring, các đối tượng của ứng dụng tồn tại trong một container. Container dùng để tạo các đối tượng, kết hợp giữa các đối tượng và quản lý vòng đời các đối tượng, những đối tượng trong container gọi là Spring Managed beans (hay đơn giản gọi là các đối tượng b) và container gọi là Application Context.

Container sử dụng Dependency Injection (DI) quản lý các đối tượng beans, một thể hiện Application Context (`org.springframework.context.ApplicationContext`) dùng

tạo beans, kết hợp các beans thông qua cấu hình bean, và cung cấp beans khi có request từ client. Cấu hình bean được định nghĩa hoặc trong tập tin XML, hoặc annotation hoặc thông qua các lớp Java.

❖ Model View Controller

Trong các ứng dụng theo mô hình MVC (Model View Controller) chia thành 3 phần như sau:

- Model: quản lý dữ liệu.
- View: tạo giao diện người dùng.
- Controller: xử lý tương tác giữa người dùng, giao diện người dùng và dữ liệu.

Khi người dùng tương tác trên View, View sẽ gửi một sự kiện đến Controller và Controller thông báo lệnh đến Model để cập nhật dữ liệu. Trong mô hình MVC truyền thống, bất cứ khi nào dữ liệu có thay đổi, Model sẽ thông báo với View, tức là Model và View có thể tương tác trực tiếp với nhau. Trong Spring Web MVC, sự tương tác giữa Model và View phải thông qua controller do một số giới hạn trong giao thức HTTP.

1.15.4 Cài đặt môi trường

Cài bộ JDK (Java Development Toolkit)

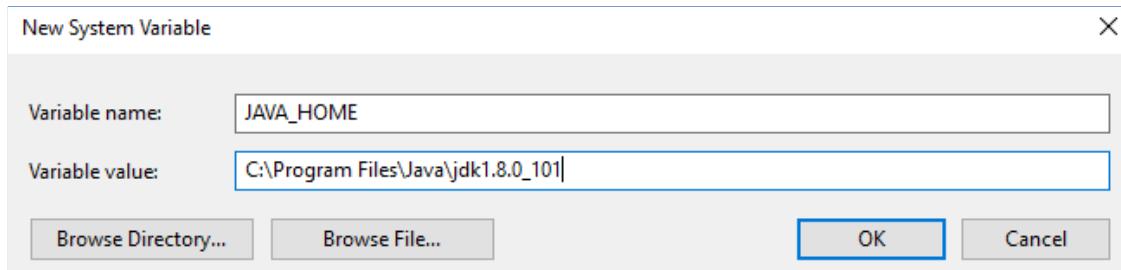
JDK (Java Development Toolkit) là bộ công cụ dùng để lập trình phát triển và thực thi các ứng dụng Java. Tải bộ JDK mới nhất tại trang web sau (thời điểm thực hiện tài liệu này bộ JDK 13 là phiên bản mới nhất).

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

Cài đặt biến môi trường

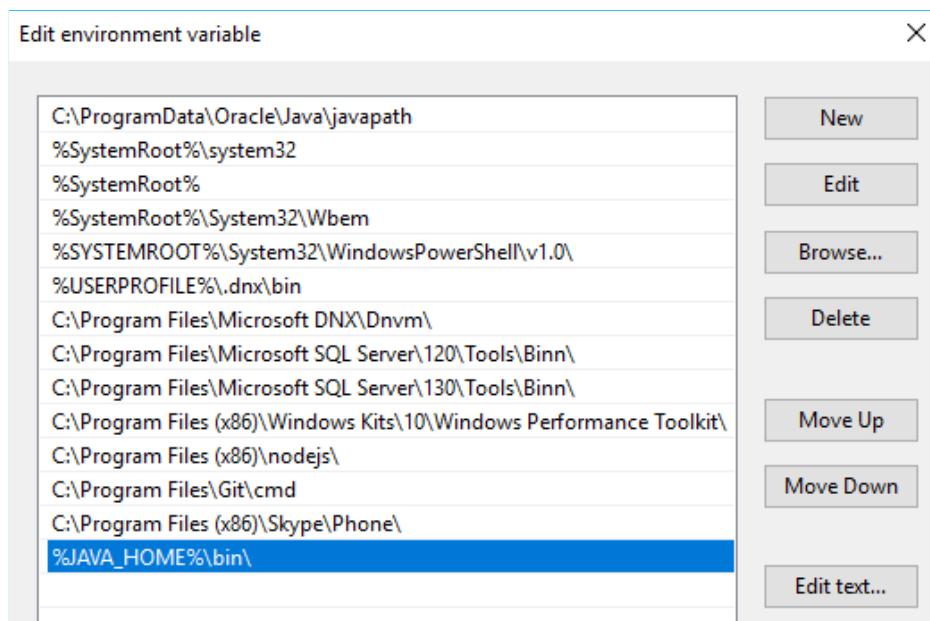
Tại màn hình Desktop, click phải vào biểu tượng This PC > Properties, cửa sổ System hiện ra, click “Advanced system settings”. Tại cửa sổ System Properties > tab Advanced và click vào nút Environment Variables..., cửa sổ các biến môi trường hiện ra, trong vùng system properties, click nút New... để tạo biến môi trường, đặt tên là

JAVA_HOME (tên này đặt tùy ý sao cho gợi nhớ được) với giá trị là đường dẫn trỏ đến JDK vừa được cài đặt trên máy và click OK.



Hình 10. Tạo biến môi trường JAVA_HOME.

Cũng trong vùng system properties, chọn biến Path và click nút Edit.... Một cửa sổ hiện ra, click New để thêm biến môi trường mới với giá trị "%JAVA_HOME%\bin\" trỏ đến thư mục bin của JDK.

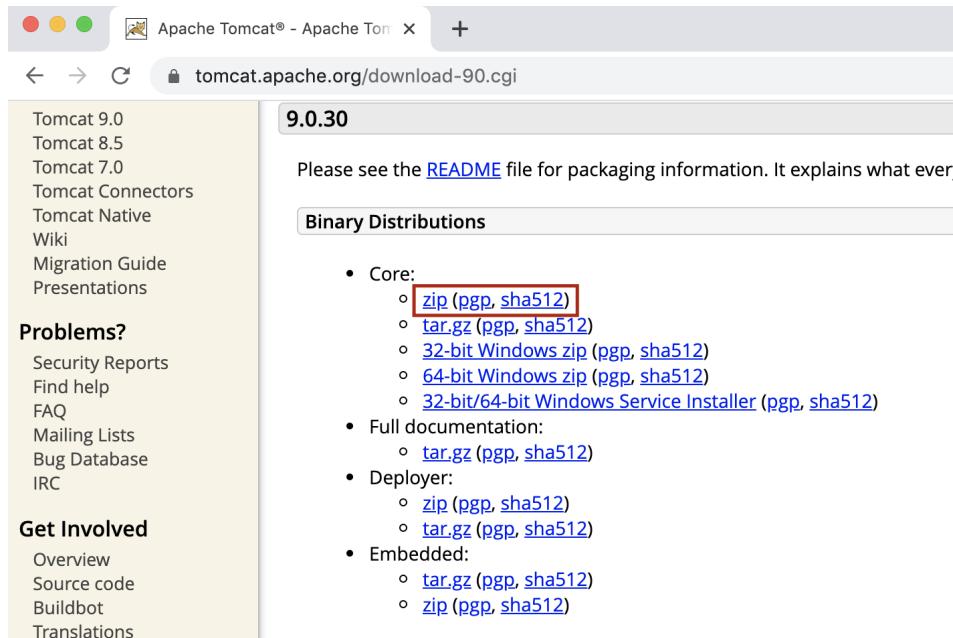


Hình 11. Cài đặt biến môi trường cho Java.

Để kiểm tra biến môi trường đã cài đặt thành công, ta có thể mở Command Prompt và gõ các lệnh "java -version" và "javac -version".

Cài đặt Apache Tomcat Server.

Apache Tomcat là một Java Web Server phổ biến, tài liệu này sử dụng Apache Tomcat 9, truy cập vào trang <http://tomcat.apache.org/> và click vào Tomcat 9 để đến trang tải về.



Hình 12. Tải Apache Tomcat Server.

Cài đặt NetBeans IDE

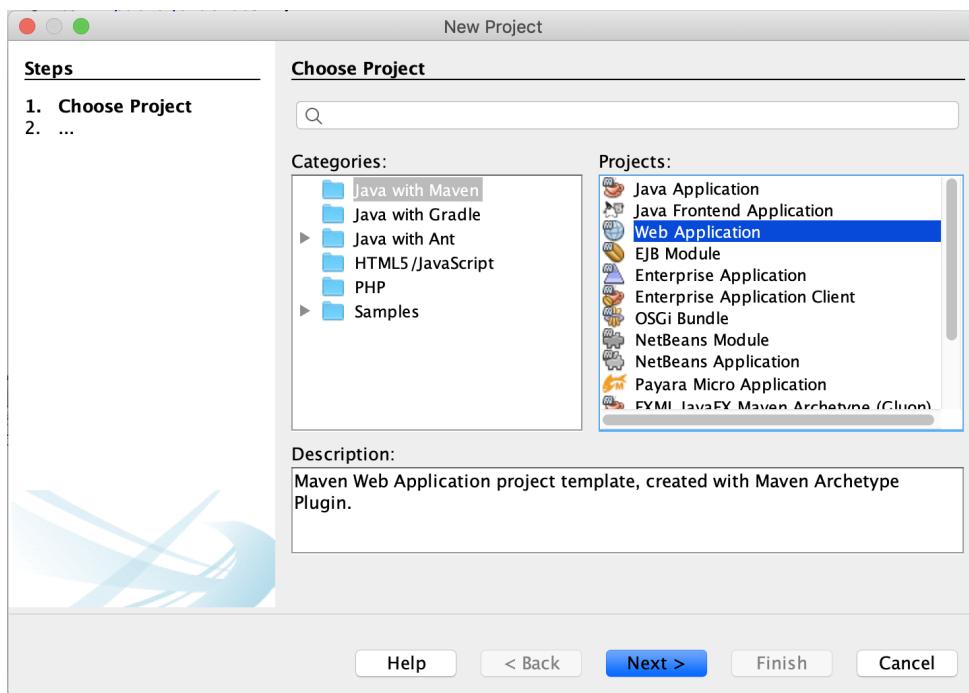
Có nhiều công cụ IDE (Integrated Development Environment) để lập trình phát triển ứng dụng Java EE, phổ biến nhất hiện nay là IDEA NetBeans IDE, IntelliJ IDE, Eclipse IDE. Trong tất cả các ví dụ của tài liệu này được minh họa trên NetBeans IDE, thời điểm thực hiện tài liệu này phiên bản mới nhất là 11.2.

Ta có thể tải NetBeans IDE tại <https://netbeans.apache.org/download/>

1.15.5 Viết chương trình đầu tiên với Spring MVC

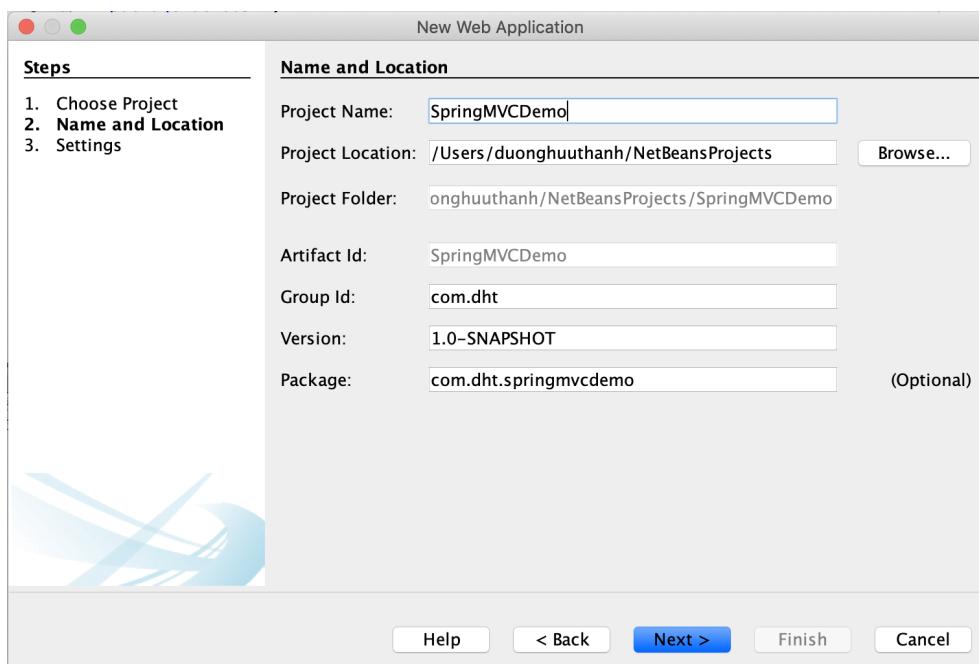
❖ Tạo project phát triển Web với Maven

Trong NetBeans tạo project Java Maven phát triển Web.



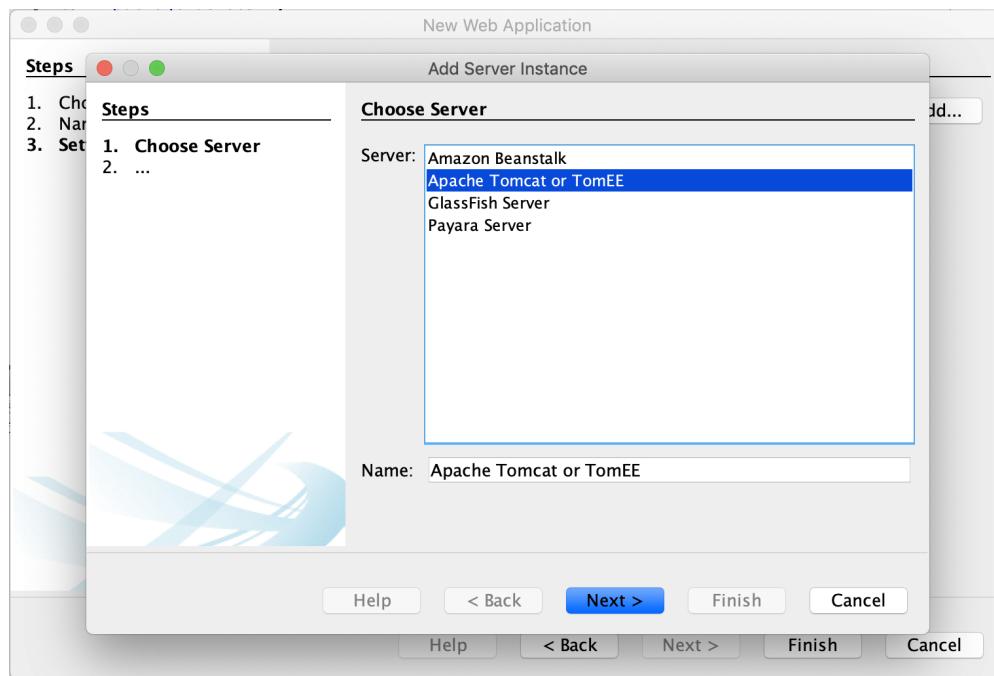
Hình 13. Tạo project Java Maven cho ứng dụng Web với NetBeans IDE 11.2.

Điền thông tin tên project và các trường Artifact Id, Group Id.



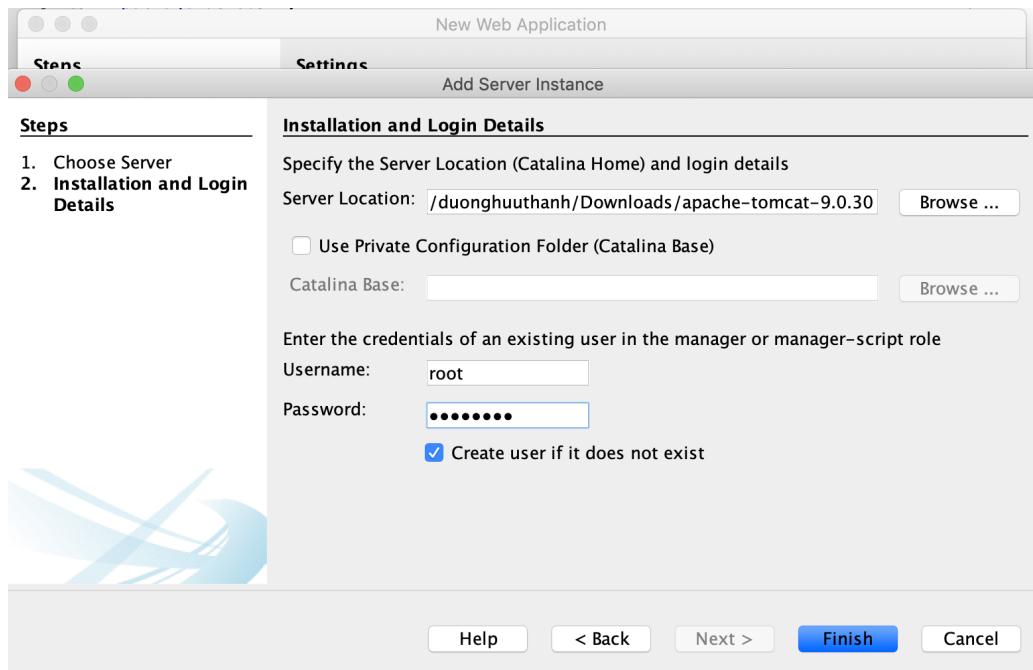
Hình 14. Thông tin cho project Java Maven cho ứng dụng Web.

Tiếp theo là màn hình cấu hình Server, click vào button Add... để thêm Apache Tomcat Server.

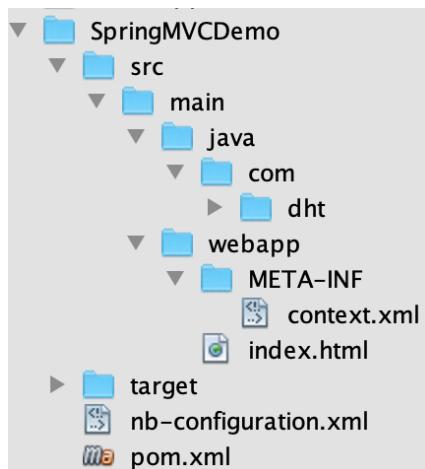


Hình 15. Thêm Tomcat Server vào project ứng dụng Web.

Sau khi tải Apache Tomcat Server thì giải nén, chỉ định vị trí thư mục giải nén này cho trường Server Location cho màn hình tiếp theo, đồng thời chỉ định thông tin username, password khi chạy server và click nút Finish.

**Hình 16. Cấu hình Tomcat Server cho ứng dụng Web**

Một project được tạo ra có cấu trúc như sau:

**Hình 17. Cấu trúc cơ bản của project ứng dụng Web.**

Trong Maven, pom.xml là tập tin cấu hình định nghĩa các dependencies cần thiết, khi build project, Maven sẽ đọc tập tin này và tải các tập tin jar từ thùng chứa trung tâm của Maven. Thêm dependency cho Spring MVC và JSTL vào pom.xml như sau:

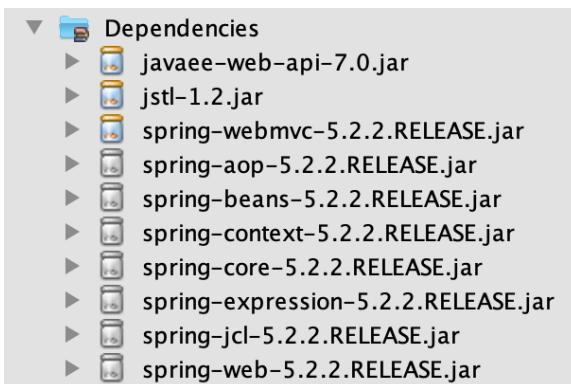
```
<dependencies>
    <dependency>
```

```

<groupId>javax</groupId>
<artifactId>javaee-web-api</artifactId>
<version>7.0</version>
<scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
</dependencies>

```

Sau khi hoàn tất thêm các dependency vào pom.xml và lưu lại thì các thư viện sẽ được tải vào project như bên dưới. Chú ý rằng spring-mvc.jar phụ thuộc vào spring-core, spring-context, spring-aop, nhưng ta không cần chỉ định các jar này trong pom.xml, Maven sẽ tự động tải tất cả các jar phụ thuộc cần thiết, ta chỉ cần cung cấp cho Maven thông tin dependency cấp một.



Hình 18. Các dependency cần thiết phát triển Web với Spring MVC.

Cấu hình servlet mapping để chỉ định các HTTP request muốn DispatcherServlet xử lý. Đầu tiên, tạo lớp WebApplicationContextConfig.java trong gói com.dht.config.

```
package com.dht.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.dht.springmvcdemo")
public class WebApplicationContextConfig implements WebMvcConfigurer {
    @Override
    public void configureDefaultServletHandling(
            DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
    @Bean
    public InternalResourceViewResolver getInternalResourceViewResolver() {
        InternalResourceViewResolver resolver
                = new InternalResourceViewResolver();
        resolver.setViewClass(JstlView.class);
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");

        return resolver;
    }
}
```

Các annotation sử dụng trong lớp trên:

- @Configuration: chỉ định lớp khai báo các phương thức @Bean
- @EnableWebMvc: Spring MVC cấu hình các bean cần thiết truyền các request đến controller: DefaultAnnotationHandlerMapping, AnnotationMethodHandlerAdapter và ExceptionHandlerExceptionResolver. Ngoài ra, annotation này cho phép hỗ trợ nhiều annotation thuận tiện khác để định dạng các trường trong bean, các annotation để kiểm tra các tham số của các phương thức trong controller hay annotation @RequestBody chuyển từ XML sang JSON và ngược lại.
- @ComponentScan: annotation này cho phép tự động xác định các lớp được gắn annotation @Controller (xem phần sau). DispatcherServlet sẽ tìm kiếm phương thức thích hợp trong lớp @Controller dựa trên annotation @RequestMapping, thuộc tính chỉ định các gói cơ sở để tìm kiếm các lớp controller.

Tiếp theo, ta tạo lớp DispatcherInitializer.java trong gói com.dht.config, lớp này giống như web.xml kế thừa AbstractAnnotationConfigDispatcherServletInitializer và cần ghi đè ba phương thức quan trọng:

- Phương thức getRootConfigClasses(): chỉ định các lớp cấu hình cho Root Application Context.
- Phương thức getServletConfigClasses(): chỉ định các lớp cấu hình cho Dispatcher Servlet Application Context.
- Phương thức getServletMappings(): chỉ định các Servlet Mappings cho DispatcherServlet.

Sau bước này DispatcherServlet được cấu hình và sẵn sàng xử lý các HTTP Request và chuyển nó đến phương thức phù hợp.

```
package com.dht.config;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcher
ServletInitializer;
```

```

public class DispatcherServletInitializer
    extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] {
            WebApplicationContextConfig.class
        };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {"/"};
    }
}

```

Tiếp theo tạo JSP View, trong ví dụ này cấu hình các tập tin JSP View nằm trong thư mục WEB-INF/jsp và sử dụng InternalResourceViewResolver để phân giải tên View từ phương thức xử lý thành hiện thực View kết xuất hiển thị cho người dùng (xem trong tập tin WebApplicationContextConfig.java). Đầu tiên ta tạo thư mục WEB-INF/jsp trong thư mục webapp, trong thư mục jsp tạo tập tin welcome.jsp như sau:

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome Page</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>

```

```
</body>
</html>
```

Tại dòng \${message} thì message là một biến và giá trị của nó sẽ hiển thị trên trang web JSP, giá trị của biến này sẽ được gán trong một phương thức nào đó của controller. Tạo HomeController.java trong com.dht.springmvcdemo.controller, biến message sẽ được gán trong phương thức index của HomeController.

```
package com.dht.springmvcdemo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {
    @RequestMapping(value = "/")
    public String index(Model model) {
        model.addAttribute("message", "Welcome to our Website!!!!");
        return "welcome";
    }
}
```

Giá trị của biến message sẽ được gán thông qua phương thức addAttribute() của Model, bất kỳ giá trị nào đặt vào trong Model đều có thể được truy vấn sử dụng trong trang JSP bằng biểu thức \${}.

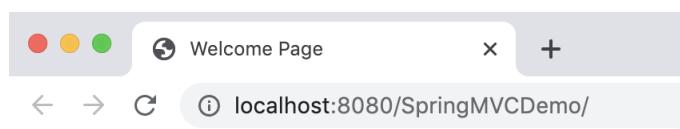
Một số Annotation sử dụng trong controller

- @Controller chỉ định một lớp có vai trò là controller.
- @RequestMapping chỉ định một ánh xạ URL tới một lớp hay một phương thức, trong đó thuộc tính value chỉ định đường dẫn, thuộc tính method chỉ định loại HTTP request sẽ được xử lý (mặc định là HTTP GET), trong ví dụ này sẽ xử lý các HTTP GET request, phương thức hello trả về một chuỗi là tên của view sẽ được kết xuất (render) hiển thị cho người dùng. Ngoài ra, Spring cung cấp các annotation tiện ích cho HTTP

GET, POST, PUT, DELETE tương ứng @GetMapping, @PostMapping, @PutMapping và @DeleteMapping.

- @ResponseBody: khi một phương thức được gắn annotation là @ResponseBody thì giá trị trả về của phương thức sẽ được ghi vào HTTP Response.

Click phải chuột vào project > Run để chạy server, sau đó truy cập vào đường dẫn <http://localhost:8080/SpringMVCdemo/> thì kết quả trang web như sau:



Welcome to our Website!!!

1.16 LÀM VIỆC VỚI CONTROLLER

Để định nghĩa controller, ta chỉ cần có các lớp Java kết hợp annotation **@Controller** (org.springframework.stereotype.Controller).

Các phương thức trong controller thường được gắn annotation **@RequestMapping** (org.springframework.web.bind.annotation.RequestMapping) chỉ định đường dẫn URL sẽ ánh xạ với phương thức đang viết. Ta cũng có thể sử dụng @RequestMapping cho lớp controller, khi đó Spring MVC sẽ xét các giá trị của @RequestMapping ở cấp lớp trước khi ánh xạ phần còn lại của đường URL vào phương thức xử lý.

Mỗi lớp controller được phép chỉ định một **phương thức ánh xạ mặc định** (default mapping method), nó đơn giản là phương thức không cần chỉ định đường dẫn URL cho thuộc tính value của @RequestMapping, phương thức này được xem là phương thức ánh xạ mặc định cho lớp controller đó. Chú ý nếu lớp controller có hơn một phương thức mặc định thì ngoại lệ IllegalStateException sẽ được ném ra với lỗi "Ambiguous mapping found".

Tạo DictionaryController.java gói com.dht.springmvcdemo.controller của project SpringMVCdemo như sau:

```

import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping(value = "/dicts")
public class DictionaryController {
    private final static Map<String, String> dicts;
    static {
        dicts = new HashMap<>();
        dicts.put("Hello", "Xin chào");
        dicts.put("Love", "Yêu");
        dicts.put("Computer", "Máy tính");
        dicts.put("Remember", "Nhớ");
        dicts.put("Artificial", "Nhân tạo");
    }
    @RequestMapping
    public String index(ModelMap model) {
        model.addAttribute("message", "Welcome to Our Dictionary!!!!");
        return "dicts";
    }
    @RequestMapping(value = "/list")
    public String list(ModelMap model) {
        model.addAttribute("words", dicts);
        return "dicts-list";
    }
}

```

Phương thức index() là phương thức mặc định được thực thi khi request truy cập đường dẫn /dicts định nghĩa trong @RequestMapping của lớp controller. Phương thức list() sẽ được thực thi khi request truy cập đường dẫn /dicts/list.

Tiếp theo tạo các view tương ứng cho hai request của hai phương thức này. Trong thư mục webapp/WEB-INF/jsp tạo hai tập tin jsp đặt tên là dicts.jsp và dicts-list.jsp.

Tập tin dicts.jsp

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>My Dictionary</title>
  </head>
  <body>
    <h1>${message}</h1>
  </body>
</html>
```

Tập tin dicts-list.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@ pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

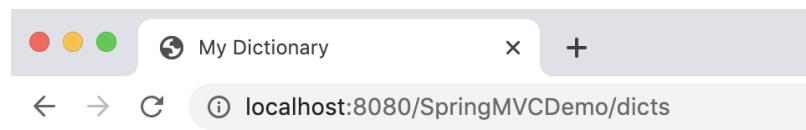
<html>
    <head>
        <meta charset="UTF-8">
        <title>My Dictionary</title>
    </head>
    <body>
        <table width="80%" border="1">
            <tr>
                <th width="30%">Word</th>
                <th width="70%">Description</th>
            </tr>
            <c:forEach items="${words.entrySet()}" var="w">
                <tr>
                    <td>${w.getKey()}</td>
```

```

        <td>${w.getValue()}</td>
    </tr>
</c:forEach>
</table>
</body>
</html>

```

Truy cập <http://localhost:8080/SpringMVCDemo/dicts> kết quả như sau:



Welcome to Our Dictionary!!!

Truy cập <http://localhost:8080/SpringMVCDemo/dicts/list> kết quả như sau:

Word	Description
Artificial	Nhân tạo
Hello	Xin chào
Remember	Nhớ
Love	Yêu
Computer	Máy tính

❖ @PathVariable (org.springframework.web.bind.annotation.PathVariable)

Để lấy giá trị tham số truyền trên đường dẫn URL của request sử dụng annotation @PathVariable. Nó dùng kết buộc một tham số của phương thức trong controller với các tham số trên đường dẫn URL.

Ví dụ trong controller DictionaryController định nghĩa thêm phương thức tìm nghĩa của một từ cụ thể như sau:

```

@RequestMapping(value = "/list/{word}")
public String details(ModelMap model,
    @PathVariable(value = "word") String word) {

```

```

String message = dicts.get(word);
if (message == null)
    message = "Không có từ này!!!";
model.addAttribute("message", message);
return "dicts-detail";
}

```

Tạo tập tin dicts-detail.jsp trong WEB-INF/jsp như sau:

```

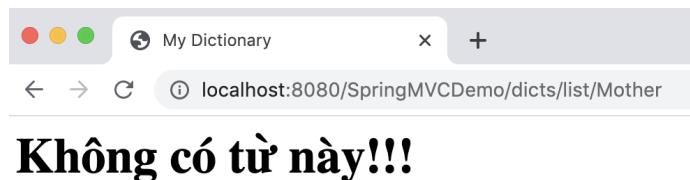
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
    <head>
        <meta charset="UTF-8">
        <title>My Dictionary</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>
</html>

```

Truy cập các URL <http://localhost:8080/SpringMVCDemo/dicts/list/Computer> và <http://localhost:8080/SpringMVCDemo/dicts/list/Love> kết quả như sau:



Truy cập <http://localhost:8080/SpringMVCDemo/dicts/list/Mother> kết quả như sau



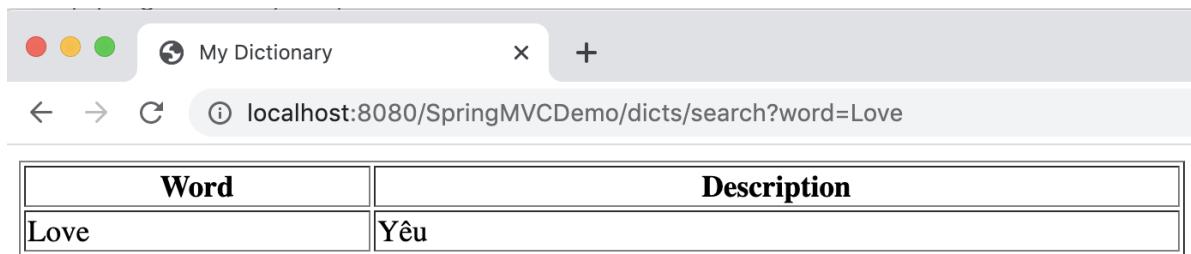
❖ @RequestParam

Để lấy giá trị các tham số được truyền thông qua các tham số của HTTP GET sử dụng annotation `@RequestParam`. `@RequestParam` dùng kết buộc một tham số của request và một tham số của phương thức trong controller.

Ví dụ cũng vẫn để cho phép tra từ, nhưng cho phép người dùng truyền từ cần tìm dưới dạng các tham số truy vấn của HTTP GET. Định nghĩa nạp chồng thêm phương thức list trong controller DictionaryController như sau:

```
@RequestMapping(value = "/search")
public String list(ModelMap model,
    @RequestParam(value = "word") String word) {
    Map<String, String> res = new HashMap<>();
    String des = dicts.get(word);
    if (des != null)
        res.put(word, des);
    model.addAttribute("words", res);
    return "dicts-list";
}
```

Truy cập <http://localhost:8080/SpringMVCDemo/dicts/search?word=Love> kết quả như sau:



The screenshot shows a web browser window titled 'My Dictionary'. The address bar displays the URL 'localhost:8080/SpringMVCDemo/dicts/search?word=Love'. The main content area shows a table with two columns: 'Word' and 'Description'. There is one row with the word 'Love' in the 'Word' column and the description 'Yêu' in the 'Description' column.

Word	Description
Love	Yêu

1.17 TAG LIBRARIES

JavaServer Page (JSP) là công nghệ cho phép nhúng mã nguồn Java vào các trang HTML, mã nguồn Java được chèn giữa cặp dấu `<% %>` hoặc thông qua các thẻ của JSTL (JavaServer Pages Standard Tag Library).

JSTL là thư viện các thẻ chuẩn được cung cấp bởi Oracle. Để sử dụng thư viện JSTL trong các trang JSP cần chỉ định nó thông qua taglib (taglib directives). Taglib khai

báo các trang JSP sử dụng tập các thẻ thư viện của JSTL và chỉ định vị trí của thư viện bằng thuộc tính uri, thuộc tính prefix chỉ tiền tố khi sử dụng các thẻ trong thư viện chỉ định.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

Spring MVC cũng cung cấp thư viện thể riêng giúp cho việc phát triển các view JSP được dễ dàng hơn, để sử dụng các thư viện này ta cần chỉ định nó trong các trang JSP như sau:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

Các thẻ trong thư viện Spring sẽ giúp ta dễ dàng kết buộc (binding) các giá trị của các thành phần HTML với bean trong Model, sau đó controller truy vấn bean trong Model bằng cách sử dụng @ModelAttribute.

Ví dụ thực hiện chức năng thêm một từ vào từ điển

Tạo gói com.dht.pojo và tập tin Word.java trong gói này như sau:

```
public class Word {
    private String word;
    private String description;
    // Các phương thức getter/setter của các thuộc tính
}
```

Trong DictionaryController tạo hai phương thức bên dưới, cả hai phương thức này có cùng URL nhưng khác nhau thuộc tính method của request. Khi ta truy cập <http://localhost:8080/SpringMVCDemo/dicts/add>, nó sẽ xem là HTTP request dạng GET nên phương thức addWordView() sẽ được gọi, phương thức addWordProcess() sẽ được gọi khi submit một form thêm từ mới thông qua HTTP request dạng POST (xem view dicts-add-word.jsp bên dưới)

```
@RequestMapping(value = "/add")
public String addWordView(ModelMap model) {
    Word w = new Word();
    model.addAttribute("word", w);
```

```

        return "dicts-add-word";
    }

@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
                             @ModelAttribute(value = "word") Word newWord) {
    if (dicts.get(newWord.getWord()) == null) {
        dicts.put(newWord.getWord(), newWord.getDescription());
        return "redirect:/dicts/list";
    } else {
        model.addAttribute("message", "Từ đã tồn tại!!!");
        return "dicts-add-word";
    }
}

```

Trong WEB-INF/jsp tạo view đặt tên dicts-add-word.jsp như bên dưới, trang này khai báo sử dụng thư viện thẻ form của Spring.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
    <head>
        <meta charset="UTF-8">
        <title>My Dictionary</title>
    </head>
    <body>
        <form:form method="POST" modelAttribute="word">
            <h2>THÊM TỪ MỚI</h2>
            <h3 style="color:red">${message}</h3>
            <table>
                <tr>
                    <td>Từ mới</td>
                    <td><form:input id="wordId" path="word" /></td>
                </tr>
                <tr>

```

```

        <td>Ý nghĩa</td>
        <td><form:input id="desId" path="description" /></td>
    </tr>
    <tr style="text-align: right">
        <td colspan="2">
            <input type="submit" value="Thêm từ" />
        </td>
    </tr>
</table>
</form:form>
</body>
</html>
```

Trong thẻ `<form:form>` ngoài việc khai báo thuộc tính method là POST, thì một thuộc tính quan trọng khác được khai báo là **modelAttribute** có giá trị là "word", đây là tên thuộc tính dùng lưu trữ đối tượng Word mới được tạo trong phương thức `addWordView()` (đối tượng này được gọi là **Backing Bean** trong Spring MVC). Trong các thẻ `<form:input>` bên trong `<form:form>` có thuộc tính quan trọng là **path**, giá trị của thuộc tính này là tên trường của đối tượng Backing Bean, nên giá trị được nhập vào form này sẽ được kết buộc vào trường tương ứng trong Bean.

Khi form này được submit thì phương thức `addWordProcess()`, đối tượng word sẽ được truyền vào với các giá trị đã nhập trong form, trước tham số `newWord` sử dụng annotation `@modelAttribute`, để giá trị thuộc tính `value` của `@modelAttribute` chính là giá trị đã khai báo trong `<form:form>`. Chú ý chuỗi trả về của phương thức này là "redirect:/dicts/list", trong đó tiền tố `redirect` chỉ định Spring biết gọi một View đặc biệt `RedirectView` (`org.springframework.web.servlet.view.RedirectView`).

Trong lớp `DispatcherServletInitializer` bổ sung phương thức sau:

```

@Override
protected Filter[] getServletFilters() {
    CharacterEncodingFilter filter = new CharacterEncodingFilter();
    filter.setEncoding("UTF-8");
```

```

        filter.setForceEncoding(true);

    return new Filter[] { filter };

}

```

Truy cập <http://localhost:8080/SpringMVCdemo/dicts/add> và nhập thêm từ “Love”, chương trình sẽ báo lỗi “Từ đã tồn tại” như sau:

Thêm Từ Mới

Từ mới

Ý nghĩa

Thêm Từ Mới

Từ đã tồn tại!!!

Từ mới

Ý nghĩa

Nhập thêm từ “Miss”, nghĩa “Nhớ” như sau:

Thêm Từ Mới

Từ mới

Ý nghĩa

Sau khi bấm nút “Thêm từ” kết quả sẽ chuyển sang trang danh sách các từ, bao gồm từ mới thêm vào.

The screenshot shows a web browser window with the title 'My Dictionary'. The address bar shows the URL 'localhost:8080/SpringMVCdemo/dicts/list'. The main content is a table with two columns: 'Word' and 'Description'. The data in the table is:

Word	Description
Artificial	Nhân tạo
Hello	Xin chào
Remember	Nhớ
Love	Yêu
Computer	Máy tính
Miss	Nhớ

WebDataBinder dùng lấy dữ liệu từ đối tượng HttpServletRequest, chuyển nó thành định dạng dữ liệu thích hợp, nạp nó vào đối tượng Backing Bean và kiểm tra dữ liệu (validate). Để điều chỉnh cách thức kết buộc dữ liệu (data binding), ta cần khởi động và cấu hình đối tượng WebDataBinder trong Controller. Annotation @InitBinder (org.springframework.web.bind.annotation.InitBinder) dùng để chỉ định phương thức khởi động WebDataBinder.

Giả sử trong lớp com.dht.pojo.Word thêm thuộc tính note

```
public class Word {
    private String word;
    private String description;
    private String note;
    // Các phương thức getter/setter
}
```

Trong form thêm câu hỏi của dicts-add-word.jsp thêm đoạn <tr> trước <tr> chứa nút submit.

```
<tr>
    <td>Ghi chú</td>
    <td><form:input id="noteId" path="note" /></td>
</tr>
```

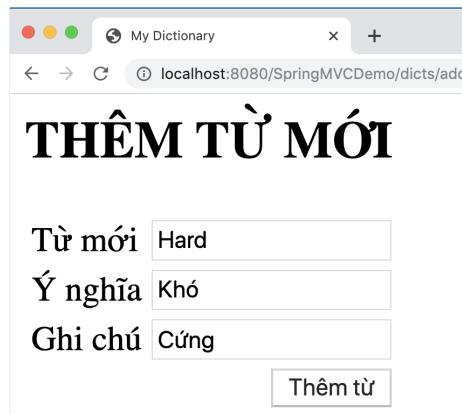
Trong lớp DictionaryController thêm phương thức initBinder() để chỉ định các trường được phép kết buộc.

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.setAllowedFields("word", "description");
}
```

Đầu của phương thức addWordProcess() thêm tham số kiểu BindingResult và thêm các dòng lệnh kiểm tra dữ liệu kết buộc như sau:

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
    @ModelAttribute(value = "word") Word newWord,
    BindingResult result) {
    if (result.getSuppressedFields().length > 0)
        throw new RuntimeException("Attempting to disallowed fields!!!!");
    ...
}
```

Khi đó ta thử thêm một từ mới chương trình sẽ báo lỗi có trường không được phép kết buộc vào (trường note), gỡ bỏ đoạn HTML của trường note, chương trình hoạt động lại bình thường.



HTTP Status 500 - Internal Server Error	
type	Exception report
message	Internal Server Error
description	The server encountered an internal error that prevented it from fulfilling this request.
exception	<pre>org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.RuntimeException: Attempting to disallowed fields!!! root cause java.lang.RuntimeException: Attempting to disallowed fields!!! note The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 5.0 logs.</pre>
GlassFish Server Open Source Edition 5.0	

Trong ví dụ trên các nhãn hiển thị trên trang web đều là hard-code trực tiếp từ trong tập tin .jsp, điều này thiếu linh hoạt khi ta cần chỉnh sửa nội dung hiển thị trang web, cũng như khi cần phát triển trang web đa ngôn ngữ.

Trong tập tin dicts-add-word.jsp thêm dòng khai báo directives sau và chỉnh sửa đoạn HTML hiển thị các văn bản của các input.

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<form:form method="post" modelAttribute="word">
    <h2>THÊM TỪ MỚI</h2>
    <h3 style="color:red">${message}</h3>
    <table>
        <tr>
            <td><spring:message code="label.word" /></td>
            <td>
                <form:input id="wordId" path="word" />
            </td>
        </tr>
        <tr>
            <td><spring:message code="label.description" /></td>
            <td>
                <form:input id="desId" path="description" />
            </td>
        </tr>
        <tr style="text-align: right">
            <td colspan="2">
```

```

        <input type="submit" value="Thêm từ" />
    </td>
</tr>
</table>
</form:form>
```

Thẻ `<spring:message>` chỉ định văn bản từ ngoài được điền vào khi chương trình thực thi, để sử dụng thẻ này ta cần thêm thư viện Spring Tag. Trong thẻ này có thuộc tính quan trọng là code để chỉ định key và giá trị của nó sẽ được đọc khi chương trình thực thi, Spring sẽ đọc giá trị tập tin property. Trong thư mục src/main/resources tạo tập tin messages.properties có nội dung:

```

label.word=Từ mới (tiếng Anh)
label.description=Nghĩa của từ (tiếng Việt)
```

Để kết nối thông tin từ tập tin properties và trên JSP view, ta cần cấu hình Bean trong WebApplicationContextConfig cho lớp ResourceBundleMessageSource với tên messageSource như bên dưới, trong đó thuộc tính basename chỉ định giá trị là tên của tập tin property.

```

@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource resource
        = new ResourceBundleMessageSource();
    resource.setBasename("messages");
    return resource;
}
```

Truy cập <http://localhost:8080/SpringMVCDemo/dicts/add> được trang kết quả:



1.18 VIEW RESOLVER

1.18.1 Redirect

Redirect là kỹ thuật chuyển người dùng (visitor) đến một trang khác với trang web đang request, kỹ thuật này thường được sử dụng sau khi submit một web form để hạn chế người dùng submit lại form tương tự khi bấm nút Back hoặc Refresh trên trình duyệt. Để sử dụng org.springframework.web.servlet.view.RedirectView để xử lý chuyển trang trong controller, ta chỉ cần trả về chuỗi URL với phần tiền tố (prefix) chuyển trang, có hai tiền tố được sử dụng để chuyển trang: forward và redirect.

Tạo com.dht.controller.springmvcdemo. HelloController hai phương thức như sau:

```
@RequestMapping(value = "/hello2")
public String hello2(ModelMap model) {
    return "hello";
}

@RequestMapping(value = "/hello1")
public String hello1(Model model) {
    model.addAttribute("message", "Welcome to our Website!!!");
    return "forward:/hello2";
}
```

Trang WEB-INF/jsp/hello.jsp có phần body như sau:

```
<body>
```

```
<h1>${message}</h1>
</body>
```

Truy cập vào trang <http://localhost:8080/SpringMVCDemo/hello1> thì ta được trang web hiển thị dòng chữ "Welcome to our Website !!!!". Sau đó sửa dòng return của phương thức hello1 thành "redirect:/hello2" và truy cập lại vào đường dẫn trên thì ta được một trang web rỗng. Điều này cho thấy sự khác nhau sử dụng forward và redirect như sau:

- forward: Spring chuyển request hiện tại đến một phương thức request mapping khác dựa trên đường dẫn sau tiền tố forward, request được chuyển tới vẫn là request gốc ban đầu, nên những giá trị được đặt vào model khi bắt đầu request vẫn còn giá trị.
- redirect: Spring sẽ tạo một request mới, nên những giá trị đặt vào model khi bắt đầu request hiện tại sẽ mất đi.

1.18.2 Thuộc tính Flask

Như đã nói ở trên, thông thường để tránh tình trạng người dùng submit nhiều lần khi submit form với HTTP Post, thì trong phương thức xử lý HTTP Post thường redirect đến một trang khác để trình duyệt thực hiện một request mới dạng HTTP Get và nạp trang Get. Tuy nhiên khi thực hiện redirect để chuyển trang thì dữ liệu trong HTTP request ban đầu sẽ bị mất đi và không thể sử dụng cho request mới sau khi redirect. Thuộc tính Flask cung cấp cho ta cách thức lưu trữ thông tin tạm thời trong session để sử dụng trong một request khác.

Để sử dụng thuộc tính Flask ta cần thêm tham số `RedirectAttributes` cho các phương thức xử lý trong controller như sau:

```
@RequestMapping(value = "/hello1")
public String hello1(Model model, RedirectAttributes attr) {
    attr.addFlashAttribute("message", "Welcome to our Website!!!!");
    return "redirect:/hello2";
}
```

Bây giờ truy cập vào trang <http://localhost:8080/SpringMVCDemo/hello1> nó sẽ chuyển sang trang hello2.htm và hiển thị dòng chữ "Welcome to our Website!!!".

1.18.3 Static Resources

Tạo thư mục src/main/webapp/resources/images và sao chép tập tin ảnh nào đó có tên java.jpg vào thư mục này.

Ghi đè phương thức addResourceHandlers() trong WebApplicationContextConfig để chỉ định vị trí chứa các tài nguyên. Trong đó

- Phương thức addResourceHandler() chỉ định request path trả về thư mục chứa tài nguyên. Trong ví dụ này ta gán giá trị "/img/**" cho mapping, ký hiệu ** chỉ định tìm kiếm đệ quy các tài nguyên trong thư mục tài nguyên cơ sở chỉ định.
- Phương thức addResourceLocations() của ResourceHandlerRegistry chỉ định đường dẫn cơ sở của các tài nguyên tĩnh (static resources).

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/img/**")
        .addResourceLocations("/resources/images/");
}
```

Truy cập <http://localhost:8080/SpringMVCDemo/img/java.png> sẽ thấy hình ảnh hiển thị trên trình duyệt.

Chỉnh sửa trang welcome.jsp như sau:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
    <head>
        <title>Welcome Page</title>
    </head>
```

```

<body>
    <h1>${message}</h1>
    <img src=<c:url value="/img/java.png" />" alt="Trang chủ" />
</body>
</html>

```

Truy cập <http://localhost:8080/SpringMVCdemo> kết quả như sau:



Welcome to our Website!!!



1.18.4 Multipart Request

Multipart request là một loại HTTP Request để gửi các tập tin và dữ liệu đến server. Multipart request được Spring MVC hỗ trợ tốt.

Ví dụ bổ sung thêm thông tin ảnh minh họa cho từ mới thêm vào từ điển.

Thêm Beans trong WebApplicationContextConfig, lớp CommonsMultipartResolver quyết định một request có được phép chứa nội dung multipart và chuyển HTTP request thành các tham số và tập tin multipart hay không.

```

@Bean
public CommonsMultipartResolver multipartResolver() {
    CommonsMultipartResolver resolver = new CommonsMultipartResolver();
    resolver.setDefaultEncoding("UTF-8");
}

```

```
    return resolver;
}
```

Thêm 2 dependency vào tập tin pom.xml hỗ trợ xử lý upload tập tin.

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>
```

Bổ sung dòng sau vào tập tin messages.properties

```
label.image=Ảnh minh họa (tùy chọn)
```

Trong lớp com.dht.pojo.Word bổ sung thêm thuộc tính kiểu MultipartFile (org.springframework.web.multipart.MultipartFile) và các phương thức getter và setter của nó, thuộc tính này sẽ giữ một tập tin thực sự được upload.

```
private MultipartFile img;
```

Cập nhật tập tin dicts-add-word.jsp như bên dưới, chú ý là <form:form> khai báo thêm enctype="multipart/form-data" để chỉ định dữ liệu trong form sẽ được mã hoá dạng multipart request khi submit form và <form:input> có thuộc tính type="file" để tạo button hiển thị cửa sổ lựa chọn tập tin.

```
<form:form method="POST" modelAttribute="word"
    enctype="multipart/form-data"
```

```

<tr>
    <td><spring:message code="label.image" /></td>
    <td><form:input id="imageId" path="img" type="file" /></td>
</tr>
...
</table>
</form:form>

```

Chỉnh sửa phương thức addWordProcess trong DictionaryController như bên dưới. Trong đó để lấy thư mục gốc của ứng dụng Web lúc chạy ứng dụng, ta cần bổ sung tham số HttpServletRequest.

```

@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
        @ModelAttribute(value = "word") Word newWord,
        HttpServletRequest request) {
    if (dicts.get(newWord.getWord()) == null) {
        MultipartFile img = newWord.getImg();
        String rootDir = request.getSession()
                .getServletContext().getRealPath("/");
        if (img != null && !img.isEmpty()) {
            try {
                img.transferTo(new File(rootDir + "resources/images/"
                        + newWord.getWord() + ".png"));
            } catch (IOException | IllegalStateException ex) {
                System.err.println(ex.getMessage());
            }
        }
    }
    dict.put(newWord.getWord(), newWord.getDescription());
    return "redirect:/dicts/list";
} else {
    model.addAttribute("message", "Từ đã tồn tại!!!!");
    return "dicts-add-word";
}

```

```
}
```

Truy cập vào <http://localhost:8080/SpringMVCDemo/dicts/add>

My Dictionary

localhost:8080/SpringMVCDemo/dicts/add

THÊM TỪ MỚI

Từ mới (tiếng Anh)

Nghĩa của từ (tiếng Việt)

Ảnh minh họa (tuỳ chọn) apple.png

Thêm từ

Truy cập vào <http://localhost:8080/SpringMVCDemo/img/Apple.png> sẽ thấy hình tương ứng vừa được upload.

BÀI 4. JDBC

Học xong bài này người học sẽ nắm được các nội dung sau:

- *Hiểu tổng quan MySQL, các cú pháp cơ bản trong MySQL.*
- *Hiểu được cơ bản về giao tác và tầm quan trọng của nó trong phát triển ứng dụng Web.*
- *Biết sử dụng JDBC tương tác cơ sở dữ liệu quan hệ, cụ thể là MySQL.*
- *Biết lợi ích sử dụng Spring JDBC và cách sử dụng nó.*

1.19 GIỚI THIỆU MYSQL

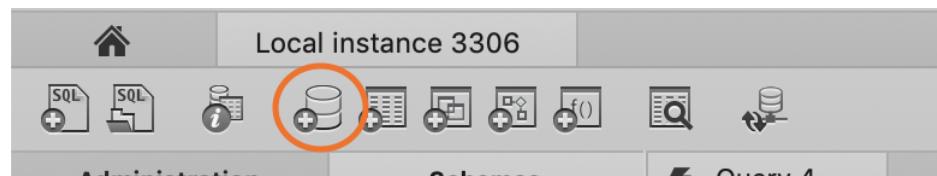
MySQL là một cơ sở dữ liệu quan hệ mã nguồn mở được sử dụng phổ biến trong phát triển các ứng dụng từ nhỏ đến lớn. MySQL sử dụng hình thức chuẩn của ngôn ngữ truy vấn dữ liệu SQL phổ biến nên cú pháp truy vấn thông dụng của nó cũng gần giống với các truy vấn trong MS SQL Server.

Tải bản MySQL Community Server tại <https://dev.mysql.com/downloads/mysql/>, sau đó double-click vào file cài đặt và thực hiện theo các hướng dẫn để cài. Trong quá trình cài đặt nên chọn cài luôn công cụ MySQL Workbench hỗ trợ tương tác với cơ sở dữ liệu MySQL dễ dàng hơn. Sau đây minh họa các thao tác cơ bản sử dụng với MySQL Workbench thông qua việc tạo cơ sở dữ liệu endb cho ứng dụng trắc nghiệm ngữ pháp tiếng Anh đơn giản, nó được sử dụng minh họa trong xuyên suốt tài liệu này. Cơ sở dữ liệu gồm 3 bảng và các quan hệ ràng buộc như sau:

- Bảng question: id, content
- Bảng choice: id, content, is_correct
- Bảng category: id, name
- Bảng level: id, name

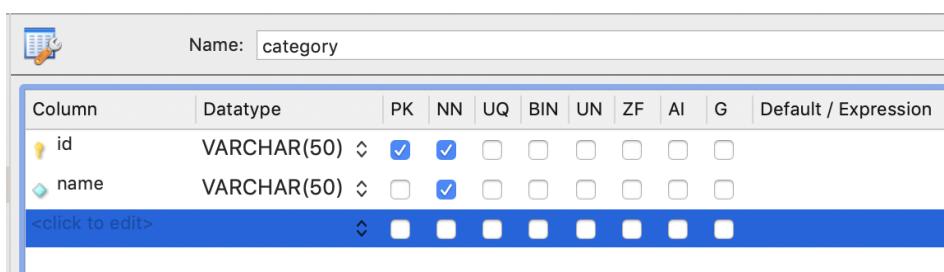
- Một câu hỏi (question) có thể thuộc 1 hoặc nhiều danh mục, một danh mục có thể có nhiều câu hỏi.
- Một câu hỏi (question) có thể có nhiều lựa chọn (choice) và chỉ có một lựa chọn chính xác, một lựa chọn chỉ thuộc vào một câu hỏi.
- Mỗi câu hỏi thuộc 1 cấp độ khó, mỗi cấp độ khó có thể có nhiều câu hỏi.

Tạo cơ sở dữ liệu: click vào icon như trong hình, sau đó đặt tên cho cơ sở dữ liệu và bấm Apply.



Hình 19. Tạo cơ sở dữ liệu trong MySQL Workbench.

Tạo các bảng: click phải vào Tables trong cơ sở dữ liệu endb > Create Table..., thiết lập thông tin các cột cho bảng và bấm Apply.



Hình 20. Tạo bảng category.

Table: level

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
id	VARCHAR(50)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					

Table: question

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
id	VARCHAR(50)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					
content	TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					
level_id	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					

Hình 21. Tạo bảng question

Bảng question có một khoá ngoại là level_id tham chiếu đến bảng Level để chỉ định mức độ khó của câu hỏi. Để thiết lập khoá ngoại click vào tab Foreign Keys và thiết lập như hình.

Table: question

Foreign Key: fk_question_level **Referenced Table:** `endb`.`level`

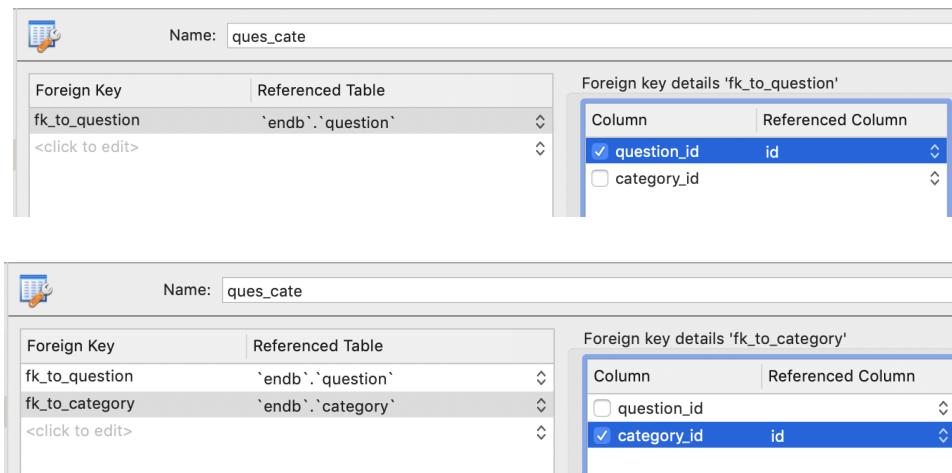
Column	Referenced Column
<input type="checkbox"/> id	<input type="checkbox"/>
<input type="checkbox"/> content	<input type="checkbox"/>
<input checked="" type="checkbox"/> level_id	<input type="checkbox"/> id

Tạo bảng ques_cate là bảng trung gian trong quan hệ nhiều-nhiều giữa hai bảng category và question. Bảng này có hai cột tương ứng là hai khoá ngoại đến hai bảng question và category.

Table: ques_cate

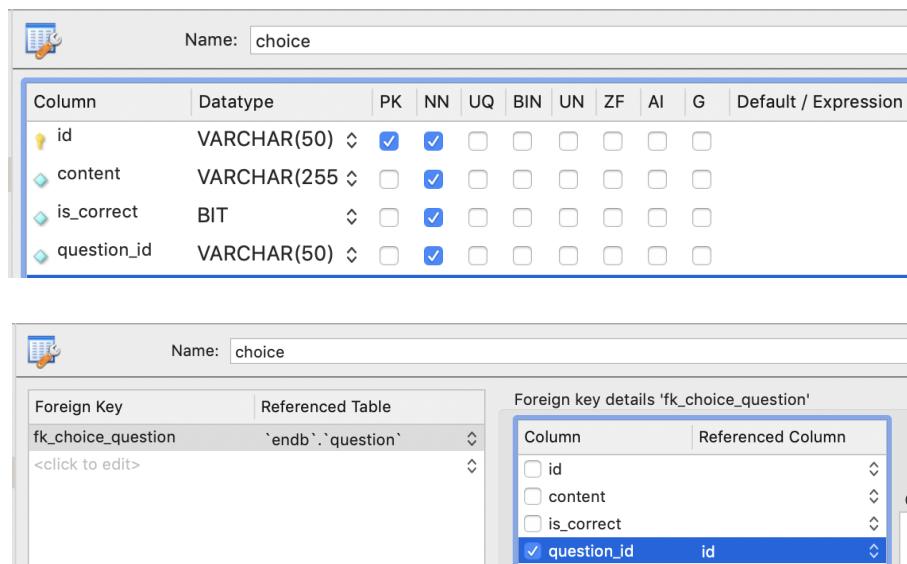
Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
question_id	VARCHAR(50)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					
category_id	VARCHAR(50)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<click to edit>					

Hình 22. Tạo bảng trung gian ques_cate trong quan hệ ManyToMany.



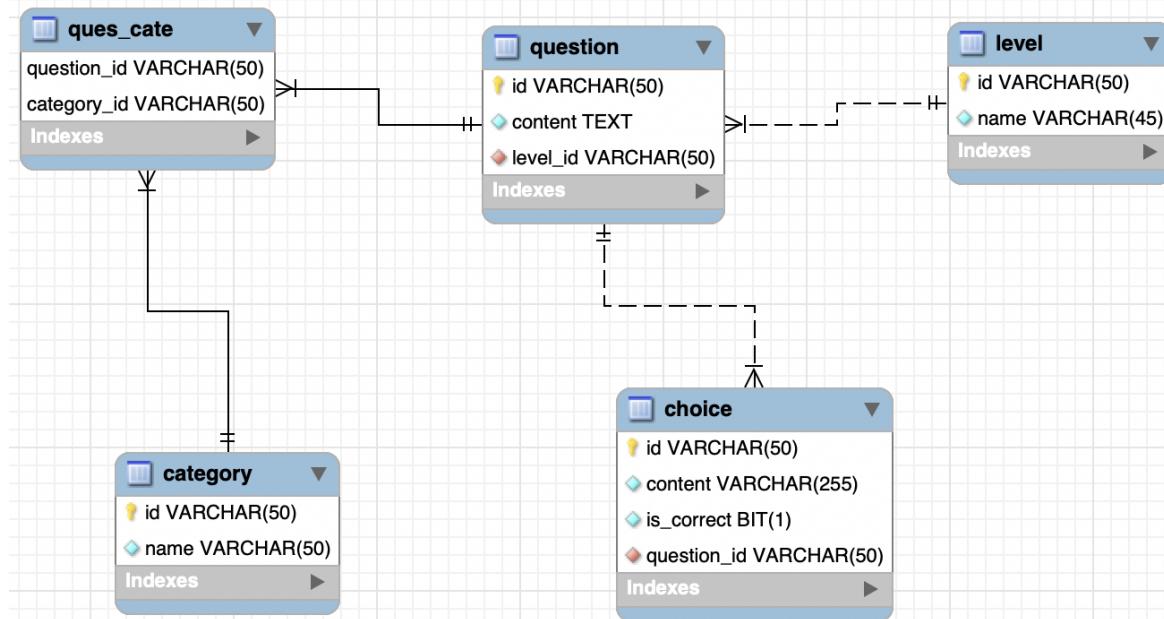
Hình 23. Thiết lập quan hệ khoá ngoại trong MySQL Workbench.

Tương tự tạo bảng choice, trong đó question_id là khoá ngoại tham chiếu đến bảng question.



Hình 24. Tạo bảng choice.

Để xuất lược đồ cơ sở dữ liệu quan hệ chọn menu Database > Reverse Engineer... và làm theo các hướng dẫn để xuất lược đồ. Kết quả ta được lược đồ cơ sở dữ liệu quan hệ như sau:



Hình 25. Lực đồ cơ sở dữ liệu quan hệ endb.

Một số cú pháp truy vấn quan trọng.

Chèn dữ liệu

```
INSERT INTO <tên-bảng> [ (cột-1, cột-2, ..., cột-n) ] VALUES (giá-trị-1, giá-trị-2, ..., giá-trị-n);
```

Ví dụ chèn một số dòng dữ liệu vào bảng category.

```
INSERT INTO category(id, name) VALUES ('Cate01', 'Noun');
INSERT INTO category(id, name) VALUES ('Cate02', 'Verb');
INSERT INTO category(id, name) VALUES ('Cate03', 'Adj');
```

Cập nhật dữ liệu

```
UPDATE <tên-bảng>
SET <cột-1> = <giá-trị-1> [, ..., <cột-n> = <giá-trị-n>]
[WHERE <điều-kiện>];
```

Ví dụ cập nhật danh mục Adj thành Adjective

```
UPDATE category
SET name = 'Adjective'
WHERE id='Cate03';
```

Xoá dữ liệu

```
DELETE FROM <tên-bảng>
[WHERE <điều kiện>]
```

Truy vấn dữ liệu đơn giản

```
SELECT <cột-1> [<cột-2>, ..., <cột-n>]
FROM <tên-bảng>
[WHERE <điều kiện>]
[GROUP BY <cột-i>]
[HAVING <điều kiện>]
[ORDER BY <cột-i> [ASC/DESC]]
[OFFSET M] [LIMIT N]
```

- SELECT chỉ định các trường dữ liệu được lấy ra, dùng dấu * nếu muốn lấy tất cả.
- FROM chỉ định các bảng dùng lấy dữ liệu.
- WHERE chỉ định điều kiện lọc dữ liệu.
- GROUP BY dùng gom nhóm dữ liệu theo các trường chỉ định.
- HAVING chỉ định điều kiện lọc dữ liệu trong gom nhóm.
- ORDER BY dùng sắp xếp dữ liệu theo các trường chỉ định, ASC là tăng dần và DESC là giảm dần.
- LIMIT chỉ định số lượng dòng lấy ra.
- OFFSET là vị trí bắt đầu lấy số dòng đó.

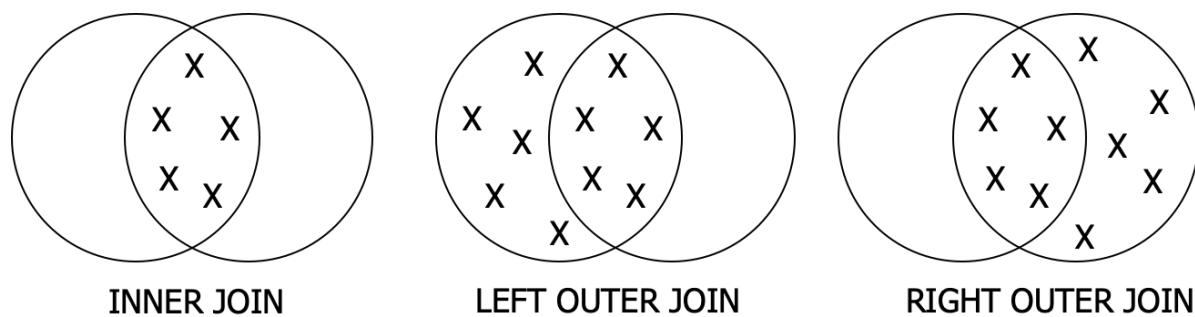
Ví dụ lấy hai dòng dữ liệu trong category xuất phát từ vị trí 1, kết quả sắp xếp giảm dần theo id.

```
SELECT *
FROM category
ORDER BY id DESC
LIMIT 2 OFFSET 1;
```

Phép kết: dùng khi truy vấn dữ liệu cần kết hợp nhiều bảng.

Trong MySQL có 3 loại kết:

- INNER JOIN: lấy các dòng từ hai bảng thỏa điều kiện kết.
- LEFT OUTER JOIN: lấy các dòng bảng bên trái và chỉ lấy những dòng thỏa điều kiện kết của các bảng khác.
- RIGHT OUTER JOIN: lấy các dòng bên phải và chỉ lấy những dòng thỏa điều kiện kết của các bảng khác.



Hình 26. Các phép kết trong MySQL.

1.20 GIỚI THIỆU JDBC

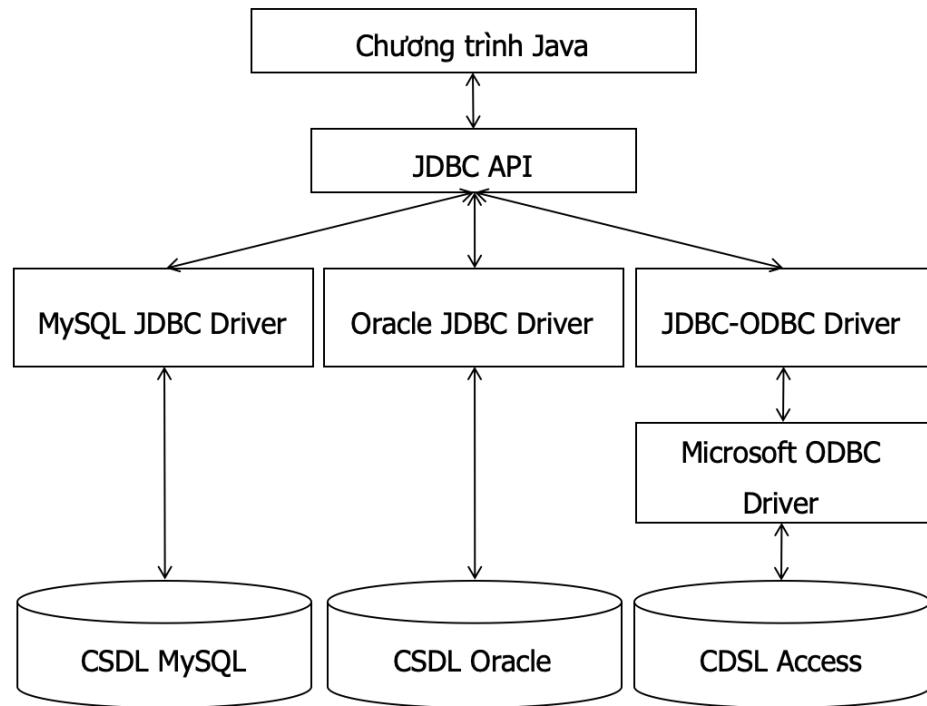
JDBC được xem là từ viết tắt của Java Database Connectivity, nó cung cấp các API để phát triển các ứng dụng tương tác với cơ sở dữ liệu quan hệ bằng Java.

JDBC API có thể tương tác với nhiều hệ quản trị cơ sở dữ liệu quan hệ khác nhau như MySQL, SQL Server, Oracle. Để tương tác với từng cơ sở dữ liệu cần có một driver tương ứng của nó, thường được cung cấp bởi nhà cung cấp cơ sở dữ liệu đó. Các driver này hiện thực lại JDBC Driver – một giao diện trung gian để giao tiếp giữa JDBC và một cơ sở dữ liệu cụ thể.

Một số giao diện quan trọng trong JDBC API

- DriverManager: quản lý các driver của cơ sở dữ liệu.
- Driver: xử lý giao tiếp với cơ sở dữ liệu.
- Connection: kết nối đến cơ sở dữ liệu.
- Statement: thực thi các câu truy vấn, stored procedures.

- ResultSet: dữ liệu trả về từ câu truy vấn.
- SQLException: ngoại lệ có thể được ném ra trong quá trình tương tác với cơ sở dữ liệu.



Hình 27. Cơ chế hoạt động của JDBC.

1.21 SỬ DỤNG JDBC TƯƠNG TÁC CƠ SỞ DỮ LIỆU

Các bước tương tác với cơ sở dữ liệu bằng JDBC

- Bước 1: Nạp Driver
- Bước 2: Thiết lập kết nối đến cơ sở dữ liệu.
- Bước 3: Thực thi câu truy vấn.
- Bước 4. Xử lý kết quả trả về.

❖ **Nạp driver**

Cú pháp nạp Driver như bên dưới. Nếu lớp Driver chỉ định không tồn tại thì ngoại lệ ClassNotFoundException sẽ được ném ra.

```
Class.forName("<Lớp-driver>")
```

Đối với MySQL, các đường dẫn driver như sau:

- MySQL 5.*

```
Class.forName("com.mysql.jdbc.Driver");
```

- MySQL 8.*

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

❖ Thiết lập kết nối

Cú pháp

```
import java.sql.Connection  
DriverManager.getConnection("databaseUrl", username, password);
```

Đối với MySQL giá trị của databaseUrl là

```
jdbc:mysql://hostname:port/databaseName
```

Ví dụ kết nối cơ sở dữ liệu có tên là endb, với username và password của MySQL lần lượt là root và 12345678.

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://localhost/endb", "root", "12345678"  
) ;
```

Chú ý luôn đóng kết nối khi không còn sử dụng bằng lệnh conn.close().

❖ Xử lý kết quả trả về

Sử dụng Statement: để tạo thể hiện của Statement sử dụng phương thức createStatement() của đối tượng kết nối. Tuỳ vào loại câu truy vấn sẽ gọi phương thức thực thi tương ứng của Statement.

- Đối với các truy vấn thao tác dữ liệu sử dụng phương thức executeQuery().
- Đối với các truy vấn định nghĩa dữ liệu sử dụng phương thức executeUpdate().

Ví dụ

```
Statement statement = conn.createStatement();
```

```

String query = "INSERT INTO category (id, name) VALUES ('Cate05',
'Conjunction');";
int result = statement.executeUpdate(query);

```

Sử dụng PreparedStatement: dùng để tạo các câu lệnh SQL cần truyền đối số, để tạo thể hiện của PreparedStatement sử dụng phương thức prepareStatement() của đối tượng kết nối. Tuỳ vào kiểu dữ liệu của đối số cần truyền vào sử dụng phương thức setX() của PreparedStatement để truyền đối số vào câu truy vấn, trong đó X là tên kiểu dữ liệu tương ứng.

Ví dụ

```

String query = "INSERT INTO category (id, name) VALUES (?, ?);";
PreparedStatement preparedStatement
    = conn.prepareStatement(query);
preparedStatement.setString(1, "Cate06");
preparedStatement.setString(2, "Tag Question");
preparedStatement.executeUpdate();

```

Sử dụng CallableStatement: dùng thực thi các stored procedure, để tạo thể hiện của CallableStatement sử dụng phương thức prepareCall() của đối tượng kết nối.

Ví dụ ta có một stored procedure để đếm số lượng câu hỏi trong bảng question.

```

CREATE DEFINER=`root`@`localhost` FUNCTION `count_question`() RETURNS
int(11)
BEGIN
    declare numberOfQuestion int;
    select COUNT(*) into numberOfQuestion
    from question;
RETURN numberOfQuestion;
END

```

Truy vấn thực thi stored procedure này.

```

CallableStatement stm = connection.prepareCall("{? = call
count_question()}");
stm.registerOutParameter(1, Types.INTEGER);
stm.execute();

```

```
System.out.println(stm.getInt(1));
```

❖ Xử lý kết quả trả về các câu truy vấn

Phương thức executeQuery() của Statement trả về thể hiện của ResultSet chứa tất cả các dòng dữ liệu tìm thấy. Để lấy dữ liệu trong ResultSet, ta có thể sử dụng chỉ số cột hoặc tên cột trong cơ sở dữ liệu bằng phương thức getX() tương ứng với từng kiểu dữ liệu của cột muốn lấy, trong đó X là tên kiểu dữ liệu.

Ví dụ lấy câu hỏi có id là Q001.

```
Statement statement = conn.createStatement();
String query = "SELECT * FROM question WHERE id='Q001'";
ResultSet result = statement.executeQuery(query);
while (result.next()) {
    System.out.printf("Id: %s\n", result.getString("id"));
    System.out.printf("Content: %s\n", result.getString("name"));
}
```

Việc di chuyển cursor trong ResultSet phụ thuộc vào tham số RSType khi tạo Statement trong các phương thức sau:

```
createStatement(int RSType, int RSCurrency)
prepareStatement(String q, int RSType, int RSCurrency)
prepareCall(String q, int RSType, int RSCurrency)
```

Trong đó RSType có thể nhận giá trị:

- ResultSet.TYPE_FORWARD_ONLY: cursor chỉ di chuyển tới (mặc định).
- ResultSet.TYPE_SCROLL_INSENSITIVE: cursor có thể di chuyển tới hoặc lùi, không quan tâm sự thay đổi dưới cơ sở dữ liệu bởi thread khác.
- ResultSet.TYPE_SCROLL_SENSITIVE: cursor có thể di chuyển tới hoặc lùi, phản ảnh những thay đổi dưới cơ sở dữ liệu bởi thread khác.

Một số phương thức di chuyển cursor trong ResultSet

- first(): di chuyển cursor đến dòng đầu.
- last(): di chuyển cursor đến dòng cuối.

- previous(): di chuyển cursor đến dòng trước, trả về false nếu đang ở dòng đầu.
- next(): di chuyển cursor đến dòng sau, trả về false nếu đang ở dòng cuối.
- absolute(int row): di chuyển cursor đến dòng chỉ định.
- relative(int row): di chuyển cursor $|row|$ dòng chỉ định từ vị trí hiện tại, nếu $row > 0$ thì di chuyển tới, còn $row < 0$ thì di chuyển lùi.

1.22 GIAO TÁC

Giao tác (transaction) đại diện cho một đơn vị xử lý có nhiều thao tác, được hỗ trợ bởi hầu hết các hệ quản trị cơ sở dữ liệu quan hệ. Quản lý giao tác là một phần quan trọng trong việc đảm bảo tính nhất quán và toàn vẹn dữ liệu. Hệ quản trị cơ sở dữ liệu bảo đảm bốn tính chất sau của giao tác, gọi là tính chất ACID:

- Tính nguyên tố (Atomicity): một giao tác là một đơn vị xử lý nguyên tố, không thể chia nhỏ, tức là một giao tác thành công nếu tất cả các thao tác của nó thành công, ngược lại tồn tại một thao tác thất bại thì toàn bộ giao tác thất bại.
- Tính nhất quán (Consistency): một giao tác phải chuyển cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác.
- Tính cô lập (Isolation): có thể có nhiều giao tác xử lý cùng một đơn vị dữ liệu ở cùng một thời điểm, nhưng một giao tác phải được cô lập với các giao tác khác, tức là giao tác này không thể thấy được tác động của giao tác khác nếu nó chưa hoàn thành.
- Tính bền vững (Durability): một giao tác khi hoàn tất, kết quả của giao tác sẽ tác động bền vững và không thể xoá đi dưới cơ sở dữ liệu từ những vấn đề của hệ thống.

Giao diện Connection mặc định sử dụng chế độ autocommit để làm việc với giao tác, tức là mỗi câu lệnh (Statement) trong kết nối sẽ được thực thi và commit lưu trữ ngay xuống cơ sở dữ liệu. Dùng phương thức setAutoCommit(false) để tắt chế độ

autocommit, khi đó các câu lệnh trong một kết nối sẽ được nhóm lại và xử lý như một đơn vị đến khi lệnh gặp

- commit() được thực thi để lưu những thay đổi xuống cơ sở dữ liệu.
- rollback() để phục hồi lại những thay đổi đã thực hiện trong giao tác.

Ví dụ

```
String query = "INSERT INTO category (id, name)"
    + " VALUES ('%s', '%s');";

conn.setAutoCommit(false);

Statement stm = conn.createStatement();
stm.executeUpdate(String.format(query, "Cate06", "Relative Pronoun"));
stm.executeUpdate(String.format(query, "Cate07", "Passive Voice"));
conn.commit();
```

1.23 BATCH QUERY

Ta có thể gom các lệnh truy vấn liên quan thành một lô (batch) để thực thi cùng một lúc để giảm bớt được số lần tương tác với cơ sở dữ liệu và có thể cải thiện được hiệu năng thực thi của chương trình.

Kiểm tra hệ quản trị cơ sở dữ liệu có hỗ trợ khả năng này hay không.

```
DatabaseMetaData.supportsBatchUpdates()
```

Phương thức addBatch() của Statement dùng thêm một câu lệnh vào batch.

Phương thức executeBatch() thực thi batch, nó trả về mảng các số nguyên, mỗi phần tử đại diện số dòng bị ảnh hưởng thực thi bởi lệnh truy vấn tương ứng.

Xóa các lệnh đã được thêm vào batch bằng clearBatch().

Ví dụ chèn hai dòng dữ liệu vào bảng category.

```
DatabaseMetaData metadata = conn.getMetaData();
if (metadata.supportsBatchUpdates()) {
    conn.setAutoCommit(false);
```

```

Statement stm = conn.createStatement();
stm.addBatch(String.format(query, "Cate08", "Vocabulary"));
stm.addBatch(String.format(query, "Cate09", "Article"));
int[] kq = stm.executeBatch();
conn.commit();
}

```

1.24 SPRING JDBC

Sử dụng Spring JDBC tương tác cơ sở dữ liệu giúp giảm bớt một số mã nguồn không cần thiết, giúp phát triển ứng dụng nhanh và hiệu quả hơn. Nó hỗ trợ trong tất cả các công đoạn tương tác cơ sở dữ liệu từ mở kết nối, thực thi truy vấn, xử lý kết quả trả về, xử lý ngoại lệ cho đến khi đóng kết nối.

Lớp JdbcTemplate là lớp quan trọng nhất trong Spring JDBC quản lý tất cả các tương tác với cơ sở dữ liệu bao gồm truy vấn dữ liệu, cập nhật dữ liệu, thực thi stored procedure, duyệt qua ResultSet, v.v., cũng như xử lý ngoại lệ xảy ra.

Cấu hình thông tin kết nối cơ sở dữ liệu

```

<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/endb" />
    <property name="username" value="root" />
    <property name="password" value="12345678" />
</bean>

```

Tạo đối tượng của JdbcTemplate để tương tác với cơ sở dữ liệu.

```

ApplicationContext context
        = new ClassPathXmlApplicationContext("Beans.xml");
DataSource dataSource = (DataSource) context.getBean("dataSource");
JdbcTemplate t = new JdbcTemplate(dataSource);

```

Cập nhật thông tin tên danh mục có id là "Cate01"

```
String sql = "UPDATE category SET name=? WHERE id=?";
t.update(sql, "TEST", "Cate01");
```

Lấy danh sách danh mục từ bảng category

```
List cats = t.queryForList("SELECT * FROM category");
System.out.println(cats);
```

Sử dụng phương thức query() để thực thi câu truy vấn, kết hợp tạo lớp Mapper để xử lý kết quả trả về. Phương thức này có thể thực thi các dạng câu truy vấn dữ liệu và định nghĩa dữ liệu.

Tạo lớp CategoryMapper như sau:

```
class CategoryMapper implements RowMapper<Category> {
    @Override
    public Category mapRow(ResultSet rs, int i) throws SQLException {
        Category c = new Category();
        c.setId(rs.getString("id"));
        c.setName(rs.getString("name"));
        return c;
    }
}
```

Truy vấn lấy danh sách category và duyệt qua các category lấy được

```
List<Category> cats = t.query("SELECT * FROM category",
                               new CategoryMapper());
cats.stream().forEach(c -> System.out.println(c.getName()));
```

Hoặc ngắn gọn, ta sử dụng biểu thức lambda như sau:

```
List<Category> cats = t.query("SELECT * FROM category", (rs, rowNum) ->{
    Category c = new Category();
    c.setId(rs.getString("id"));
    c.setName(rs.getString("name"));
    return c;
});
cats.stream().forEach(c -> System.out.println(c.getName()));
```

--

BÀI 5. HIBERNATE

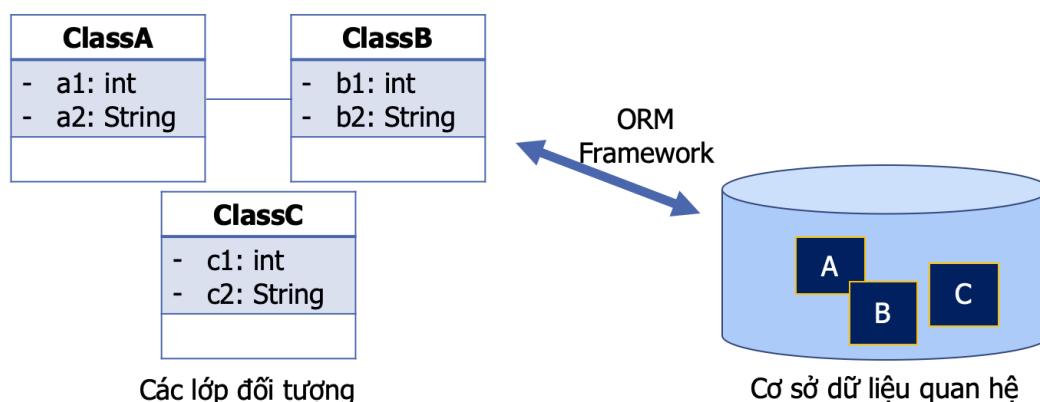
Học xong bài này người học sẽ nắm được các nội dung sau:

- Hiểu được tổng quan ORM và tầm quan trọng của nó.
- Hiểu tổng quan Hibernate, kiến trúc Hibernate và cách thức hoạt động.
- Biết cách cài đặt tương tác cơ sở dữ liệu bằng Hibernate.
- Biết tạo và thực thi truy vấn bằng HQL.
- Biết tạo và thực thi truy vấn bằng Criteria API.

1.25 Giới thiệu ORM

ORM (Object Relational Mapping) là kỹ thuật lập trình chuyển đổi dữ liệu giữa các cơ sở dữ liệu quan hệ và các ngôn ngữ lập trình hướng đối tượng như Java, C#, Python. ORM mang đến nhiều lợi ích cho việc phát triển ứng dụng tương tác với cơ sở dữ liệu quan hệ:

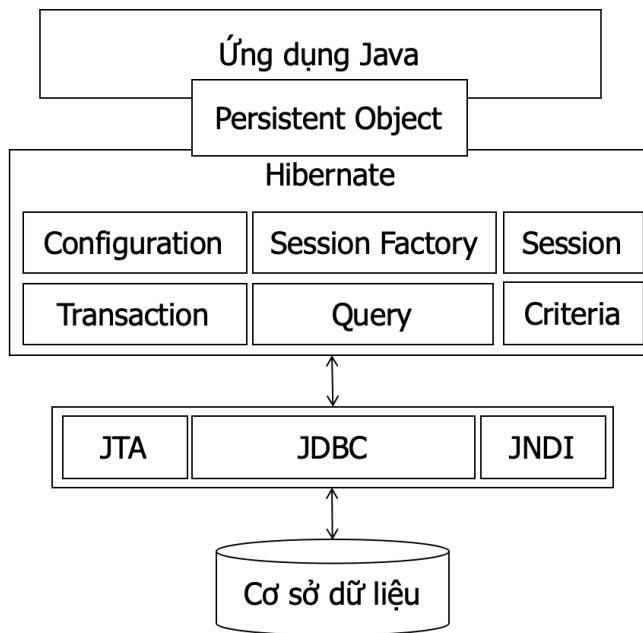
- Giúp phát triển ứng dụng nhanh.
- Tập trung vào logic nghiệp vụ cần xử lý.
- Tương tác với đối tượng nhiều hơn tương tác với bảng trong cơ sở dữ liệu.
- Truy vấn thông qua logic đối tượng, không cần trực tiếp viết các truy vấn SQL.



Hình 28. Giới thiệu ORM.

1.26 GIỚI THIỆU HIBERNATE

Hibernate là một framework mã nguồn mở được phát triển bởi Gavin King năm 2001, và là một giải pháp ORM mạnh mẽ, hiệu năng cao của Java, hỗ trợ nhiều cơ sở dữ liệu quan hệ như MySQL, PostgreSQL, MS SQL Server, DB2. Hibernate sử dụng tập tin XML để ánh xạ các lớp Java với các bảng cơ sở dữ liệu quan hệ và cung cấp các API để lưu trữ dữ liệu trên các đối tượng Java vào cơ sở dữ liệu, cũng như truy vấn dữ liệu trên các đối tượng Java từ cơ sở dữ liệu.



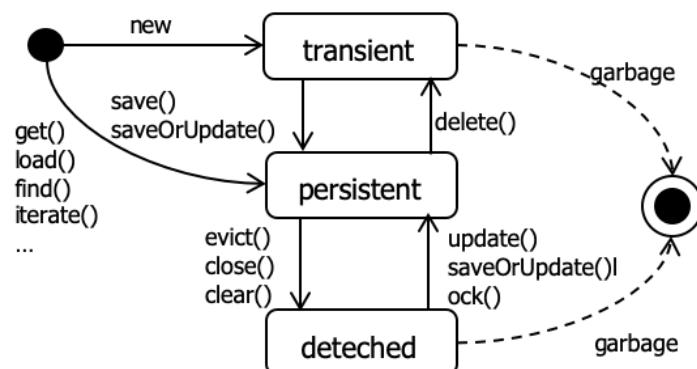
Hình 29. Kiến trúc Hibernate

Configuration: hibernate thiết lập cấu hình thông tin cơ sở dữ liệu và các thông tin liên quan khác trong tập tin hibernate.cfg.xml. Configuration là đối tượng đại diện cho các cấu hình và nó được tạo chỉ một lần lúc khởi động ứng dụng hibernate.

Session Factory: đối tượng SessionFactory dùng tạo các đối tượng Session tương tác với cơ sở dữ liệu. Đối tượng này được tạo thông qua đối tượng Configuration và tồn tại trong suốt quá trình hoạt động của chương trình. Nếu ứng dụng có tương tác với nhiều cơ sở dữ liệu, thì mỗi cơ sở dữ liệu cần một thể hiện của SessionFactory riêng.

Session: đối tượng Session được sử dụng để lấy kết nối vật lý đến cơ sở dữ liệu, có thời gian tồn tại ngắn, thể hiện của Session được tạo mỗi lúc tương tác với cơ sở dữ liệu. Nhiệm vụ chính của đối tượng Session thực hiện các thao tác tạo CRUD (Create, Read, Update, Delete) trên các đối tượng của Persistent Class, các đối tượng này có các trạng thái sau:

- transient: đối tượng chưa kết hợp với một Session nào và chưa đại diện cho record nào trong cơ sở dữ liệu.
- persistent: đối tượng kết hợp với một Session và đại diện cho một record trong cơ sở dữ liệu.
- detached: khi đóng Session.



Hình 30. Các trạng thái của đối tượng Persistent.

Persistent Class: các lớp mà các đối tượng của nó được lưu trữ xuống cơ sở dữ liệu gọi là Persistent Class. Một Persistent Class yêu cầu:

- Có phương thức khởi tạo không tham số.
- Các thuộc tính nên được khai báo private và có các phương thức getter, setter.
- Không được kế thừa từ lớp nào khác.

Mapping File: hibernate sử dụng các tập tin XML để ánh xạ (mapping) các lớp đối tượng với các bảng trong cơ sở dữ liệu.

Transaction: đối tượng Transaction dùng làm việc với giao tác.

Query: đối tượng Query sử dụng HQL (Hibernate Query Language) để truy vấn cơ sở dữ liệu và tạo các đối tượng.

Criteria: đối tượng Criteria dùng để tạo và thực thi các truy vấn có điều kiện để tìm các đối tượng.

Hibernate sử dụng một số Java API có sẵn như JDBC, JTA (Java Transaction API), JNDI (Java Naming and Directory Interface):

- JDBC cung cấp mức thấp các chức năng thông dụng thao tác với cơ sở dữ liệu quan hệ, được hỗ trợ bởi hầu hết cơ sở dữ liệu thông qua JDBC driver.
- JNDI và JTA cho phép Hibernate tương thích các ứng dụng J2EE.

1.27 HQL

HQL (Hibernate Query Language) là ngôn ngữ truy vấn hướng đối tượng, nó làm việc với các đối tượng Persistent, mà không tương tác với các cột và bảng của cơ sở dữ liệu. Câu truy vấn viết bằng HQL sẽ được hibernate dịch thành câu truy vấn SQL.

Một số mệnh đề thông dụng trong HQL:

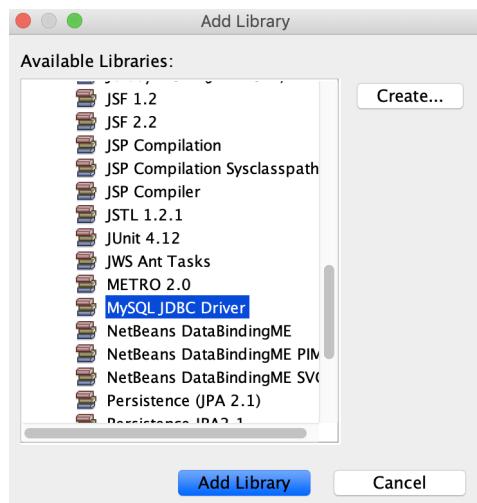
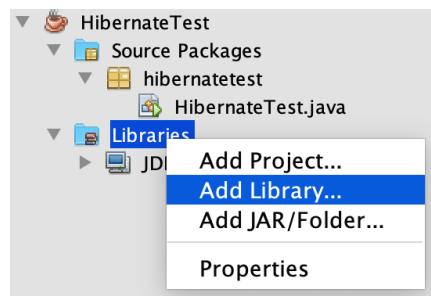
- **Mệnh đề FROM:** lấy các đối tượng lớp Persistent.
- **Mệnh đề SELECT:** lấy vài thuộc tính của đối tượng Persistent.
- **Mệnh đề WHERE:** chỉ định điều kiện lọc trong truy vấn.
- **Mệnh đề ORDER BY:** chỉ định cách sắp xếp kết quả truy vấn.
- **Mệnh đề GROUP BY:** gom nhóm kết quả truy vấn theo một thuộc tính nào đó, mệnh đề này thường kết hợp avg (tính trung bình), count (đếm số lần xuất hiện một giá trị nào đó), max (tính giá trị lớn nhất), min (tính giá trị nhỏ nhất), sum (tính tổng giá trị của một thuộc tính nào đó).
- **Mệnh đề INSERT, UPDATE và DELETE:** lần lượt dùng để chèn, cập nhật và xoá thông tin của một đối tượng trong cơ sở dữ liệu.

- **Phân trang kết quả truy vấn:** hai phương thức sau của Query dùng để phân trang kết quả truy vấn:

public Query setFirstResult(int startPosition): chỉ số dòng bắt đầu lấy kết quả.

public Query setMaxResults(int maxResult): số lượng dòng trên một trang kết quả.

Ví dụ tương tác với cơ sở dữ liệu endb. Tạo project Java Application, đặt tên HibernateTest, sau đó chọn thêm thư viện MySQL connector vào project.



Tạo tập tin cấu hình hibernate.cfg.xml đặt trong thư mục src.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
    </session-factory>
</hibernate-configuration>
```

```
</property>
<property name="hibernate.connection.driver_class">
    com.mysql.cj.jdbc.Driver
</property>
<property name="hibernate.connection.url">
    jdbc:mysql://localhost:3306/endb
</property>
<property name="hibernate.connection.username">
    [your-username]
</property>
<property name="hibernate.connection.password">
    [your-password]
</property>
</session-factory>
</hibernate-configuration>
```

Tạo tập tin `HibernateUtil.java` chưa lớp `HibernateUtil` dùng tạo `SessionFactory`.

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            Configuration configure = new Configuration();
            configure.configure("hibernate.cfg.xml");
            StandardServiceRegistryBuilder builder
                = new StandardServiceRegistryBuilder()
                    .applySettings(configure.getProperties());
            sessionFactory = configure
                .buildSessionFactory(builder.build());
        } catch (HibernateException ex) {
            System.err.println("Initial SessionFactory failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
}
```

```

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}

```

Tạo các Persistent Class: tạo gói pojo và lần lượt tạo các lớp Category, Question và Choice trong gói pojo.

```

public class Category {
    private String id;
    private String name;
    public Category() {

    }

    public Category(String id, String name) {
        this.id = id;
        this.name = name;
        this.categories = new HashSet<>();
    }

    // Các phương thức getter/setter của các thuộc tính
}

@Entity
@Table(name = "level")
public class Level implements Serializable {
    @Id
    @Column(name = "id")
    private String id;
    @Column(name = "name", nullable = false)
    private String name;

    public Level() {

    }

    public Level(String id, String name) {

```

```
        this.id = id;
        this.name = name;
    }
    // Các phương thức getter/setter của các thuộc tính
}

public class Question {
    private String id;
    private String content;
    private Level level;
    private Set<Category> categories;
    public Question() {

    }
    public Question(String id, String content) {
        this.id = id;
        this.content = content;
    }
    // Các phương thức getter/setter của các thuộc tính
}

public class Choice {
    private String id;
    private String content;
    private boolean correct;
    private Question question;
    public Choice() {

    }
    public Choice(String id, String content, boolean correct,
                  Question question) {
        this.id = id;
        this.content = content;
        this.correct = correct;
    }
}
```

```

        this.question = question;
    } // Các phương thức getter/setter của các thuộc tính
}

```

Tạo các tập tin ánh xạ.

- Category.hbm.xml

```

<hibernate-mapping>
    <class name="pojo.Category" table="category">
        <id name="id" column="id" type="string" />
        <property name="name" column="name" type="string" />
    </class>
</hibernate-mapping>

```

- Level.hbm.xml

```

<hibernate-mapping>
    <class name="pojo.Level" table="level">
        <id name="id" column="id" type="string" />
        <property name="name" column="name" type="string" />
    </class>
</hibernate-mapping>

```

- Question.hbm.xml

```

<hibernate-mapping>
    <class name="pojo.Question" table="question">
        <id name="id" column="id" type="string" />
        <property name="content" column="content" type="string" />
        <many-to-one name="level" column="level_id"
                     class="pojo.Level" />
        <set name="categories" table="ques_cate">
            <key column="question_id" />
            <many-to-many column="category_id" class="pojo.Category" />
        </set>
    </class>
</hibernate-mapping>

```

- Choice.hbm.xml

```
<hibernate-mapping>
    <class name="pojo.Choice" table="choice">
        <id name="id" column="id" type="string" />
        <property name="content" column="content" type="string" />
        <property name="correct" column="is_correct" type="boolean" />
        <many-to-one name="question" column="question_id"
                      class="pojo.Question" />
    </class>
</hibernate-mapping>
```

Trong tập tin cấu hình hibernate.cfg.xml bổ sung cấu hình thông tin tập tin ánh xạ.

```
<hibernate-configuration>
    <session-factory>
        ...
        <mapping resource="hibernatetest/Category.hbm.xml"/>
        <mapping resource="hibernatetest/Level.hbm.xml"/>
        <mapping resource="hibernatetest/Question.hbm.xml"/>
        <mapping resource="hibernatetest/Choice.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Tương tác với các bảng trong cơ sở dữ liệu.

- Lấy danh sách danh mục

```
Session session = HibernateUtil.getSessionFactory().openSession();
Query q = session.createQuery("FROM Category");
List<Category> categories = q.list();
categories.forEach(c -> {
    System.out.println(c.getName());
});
session.close();
```

- Thêm danh mục

```

Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
Category cate = new Category("Cate04", "Pronoun");
session.save(cate);
session.getTransaction().commit();
session.close();

```

- Cập nhật danh mục

```

Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
// Lấy danh mục theo khoá chính
Category cate = (Category) session.get(Category.class, "Cate02");
// Cập nhật danh mục
cate.setName("Verb Tenses");
session.update(cate);
session.getTransaction().commit();
session.close();

```

- Xoá danh mục: gọi phương thức session.delete(cate).
- Thêm và cập nhật câu hỏi

```

Session session = HibernateUtil.getSessionFactory().openSession();

Category c1 = (Category) session.get(Category.class, "Cate01");
Category c2 = (Category) session.get(Category.class, "Cate02");

session.beginTransaction();
Question q = new Question("Q0001", "This is a ... book");
q.getCategories().addAll(Arrays.asList(c1, c2));
session.save(q);

Choice a = new Choice("C0001", "well", false, q);
Choice b = new Choice("C0002", "good", true, q);
Choice c = new Choice("C0003", "lovely", false, q);
Choice d = new Choice("C0004", "beauty", false, q);

```

```
session.save(a);  
session.save(b);  
session.save(c);  
session.save(d);  
session.getTransaction().commit();  
  
session.close();
```

1.28 SỬ DỤNG ANNOTATION

Ngoài việc sử dụng các tập tin XML để ánh xạ giữa lớp đối tượng và bảng trong cơ sở dữ liệu, ta có thể sử dụng một giải pháp đơn giản và hiệu quả hơn là Annotation. Với giải pháp này, việc ánh xạ được viết trực tiếp trong các Persistent Class, nó giúp chương trình tường minh hơn, linh hoạt hơn và dễ theo dõi cấu trúc bảng cơ sở dữ liệu và cấu trúc của Persistent Class.

Một số annotation quan trọng

- @Entity: chỉ định một lớp là Persistent Class.
- @Table: chỉ định bảng tương ứng trong cơ sở dữ liệu cho Persistent Class.
- @Id: chỉ định thuộc tính tương ứng là khoá chính.
- @Column: chỉ định một trường tương ứng trong bảng trong cơ sở dữ liệu, trong đó thuộc tính name chỉ định tên trường của bảng trong cơ sở dữ liệu.
- @ManyToOne: thiết lập quan hệ Many To One giữa hai Persistent Class.
- @JoinColumn: chỉ định tên trường khoá ngoại trong cơ sở dữ liệu, thường sử dụng kết hợp các quan hệ như @ManyToOne, @ManyToMany.
- @ManyToMany: thiết lập quan hệ Many To Many giữa hai Persistent Class.
- @JoinTable: chỉ định thông tin ánh xạ bảng trung gian trong quan hệ Many To Many.

Ví dụ chỉnh sửa các lớp Persistent Class với các annotation thay cho các tập tin ánh xạ trong ví dụ phần 2.5.

Đầu tiên cần dòng import sau

```
import java.io.Serializable;
import javax.persistence.*;
```

Lớp Category

```
@Entity
@Table(name = "category")
public class Category implements Serializable {
    @Id
    @Column(name = "id")
    private String id;
    @Column(name = "name", nullable = false)
    private String name;
    // Các phương thức khởi tạo
    // Các phương thức getter/setter của các thuộc tính
}
```

Lớp Level

```
@Entity
@Table(name = "level")
public class Level implements Serializable {
    @Id
    @Column(name = "id")
    private String id;
    @Column(name = "name", nullable = false)
    private String name;
    // Các phương thức khởi tạo
    // Các phương thức getter/setter của các thuộc tính
}
```

Lớp Question

```
@Entity
```

```

@Table(name = "question")
public class Question implements Serializable {
    @Id
    @Column(name = "id")
    private String id;
    @Column(name="content", nullable = false)
    private String content;
    @OneToMany(mappedBy = "question")
    private List<Choice> choices;
    @ManyToMany
    @JoinTable(
        name = "ques_cate",
        joinColumns = { @JoinColumn(name = "question_id") },
        inverseJoinColumns = { @JoinColumn(name = "category_id") }
    )
    private Set<Category> categories;
    // Các phương thức khởi tạo
    // Các phương thức getter/setter của các thuộc tính
}

```

Lớp Choice

```

@Entity
@Table(name = "choice")
public class Choice implements Serializable {
    @Id
    @Column(name = "id")
    private String id;
    @Column(name = "content", length = 255, nullable = false)
    private String content;
    @Column(name = "is_correct")
    private boolean correct;
    @ManyToOne
    @JoinColumn(name = "question_id")

```

```

private Question question;
// Các phương thức khởi tạo
// Các phương thức getter/setter của các thuộc
}

```

Trong tập tin cấu hình hibernate.cfg.xml sửa các dòng mapping lại như sau:

```

<hibernate-configuration>
  <session-factory>
    ...
    <mapping class="pojo.Category"/>
    <mapping class="pojo.Level" />
    <mapping class="pojo.Question"/>
    <mapping class="pojo.Choice"/>
  </session-factory>
</hibernate-configuration>

```

1.29 CRITERIA API

Criteria API cho phép thực hiện các thao tác truy vấn, tương tác với cơ sở dữ liệu dễ dàng, nhanh chóng, mà không cần biết quá nhiều cú pháp truy vấn SQL. Criteria API cho phép xây dựng các đối tượng truy vấn Criteria bằng cách sử dụng phương thức `createCriteria()` của Session, với đối tượng này ta có thể.

- Thêm các điều kiện lọc dữ liệu sử dụng phương thức `add()`, mỗi điều kiện lọc là một thể hiện của Criterion được tạo thông qua các phương thức của Restrictions.
- Thêm sắp xếp sử dụng phương thức `addOrder()`, lớp Order cung cấp phương thức `asc()` và `desc()` chỉ định thứ tự sắp xếp tăng hay giảm.
- Thực hiện các phép tính toán thống kê sử dụng phương thức `setProjection()`, lớp Projections cung cấp các phương thức thống kê.

Trước khi thực hiện các truy vấn với Criteria API cần import các lớp sau:

```

import org.hibernate.Criteria;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Restrictions;

```

```
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projections;
```

❖ Truy vấn duyệt các phần tử danh sách

Ví dụ lấy danh sách câu hỏi, yêu cầu hiển thị mã câu hỏi, nội dung câu hỏi và mức độ khó của câu hỏi.

```
Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Question.class);
List<Question> rs = cr.list();
rs.forEach((q) -> {
    System.out.printf("%s - %s - %s\n", q.getId(),
                      q.getContent(), q.getLevel().getName());
});
session.close();
```

❖ Một số phương thức của Restrictions để thêm điều kiện lọc cho câu truy

- Restrictions.eq() equals
- Restrictions.ne() not equals to
- Restrictions.like() và Restrictions.ilike()
- Restrictions.isNull() và Restrictions.isNotNull()
- Restrictions.gt(), Restrictions.ge(), Restrictions.lt(), Restrictions.le()
- Restrictions.isEmpty(), Restrictions.isNotEmpty()
- Restrictions.sqlRestriction(): cho phép chỉ định trực tiếp lệnh SQL trong Criteria API, nó hữu dụng cho các mệnh đề SQL không được hỗ trợ trong Criteria API.
- Restrictions.and(): kết hợp các điều kiện lọc bằng mệnh đề AND.
- Restrictions.or(): kết hợp các điều kiện lọc bằng mệnh đề OR.

Ví dụ lấy danh sách câu hỏi có chứa từ “book” và sắp xếp kết quả giảm dần theo id

```
Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Question.class);
```

```

cr.add(Restrictions.ilike("content", "%book%"));
cr.addOrder(Order.desc("id"));

List<Question> questions = cr.list();
questions.forEach(q -> {
    System.out.printf("%s: %s\n", q.getId(), q.getContent());
});
session.close();

```

❖ Truy vấn kết (join)

- Lấy danh sách câu hỏi thuộc mức độ có tên bắt đầu bằng chữ "e".

```

Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Question.class);
Criteria cr2 = cr.createCriteria("level");
cr2.add(Restrictions.ilike("name", "e%"));

List<Question> rs = cr.list();
rs.forEach((q) -> {
    System.out.printf("%s - %s - %s\n", q.getId(),
                      q.getContent(), q.getLevel().getName());
});
session.close();

```

- Lấy danh sách câu hỏi thuộc mức độ có mã "L01" hoặc nội dung câu hỏi chứa từ "book" và chỉ lấy hai kết quả đầu tiên.

```

Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Question.class);
cr.createAlias("level", "lvl");
Criterion c1 = Restrictions.ilike("content", "%book%");
Criterion c2 = Restrictions.eq("lvl.id", "L01");
cr.add(Restrictions.or(c1, c2));
cr.setMaxResults(2);

List<Question> questions = cr.list();

```

```

questions.forEach(q -> {
    System.out.printf("%s: %s\n", q.getId(), q.getContent());
});
session.close();

```

❖ Thống kê

Ví dụ thống kê số lượng câu hỏi của từng danh mục.

```

Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Category.class);
cr.createAlias("questions", "ques");
cr.setProjection(Projections.projectionList()
                    .add(Projections.groupProperty("id"))
                    .add(Projections.count("ques.id")));
List rs = cr.list();
for (int i = 0; i < rs.size(); i++) {
    Object[] k = (Object[]) rs.get(i);
    System.out.printf("%s: %d\n", k[0], k[1]);
}
session.close();

```

❖ Phân trang kết quả truy vấn:

Tương tự đối tượng Query, đối tượng Criteria cũng có hai phương thức để xử lý phân trang:

```

public Criteria setFirstResult(int firstResult)
public Criteria setMaxResults(int maxResults)

```

Ví dụ

```

Session session = HibernateUtil.getSessionFactory().openSession();
Criteria cr = session.createCriteria(Question.class);
cr.setFirstResult(1); // lấy từ dòng 1
cr.setMaxResults(2); // lấy 2 dòng
List<Question> questions = cr.list();
session.close();

```

1.30 CRITERIA QUERY API

Từ Hibernate 5.2 các phương thức của org.hibernate.Criteria không còn dùng nữa, phát triển mới tập trung vào javax.persistence.criteria.CriteriaQuery API của JPA.

Các bước sử dụng CriteriaQuery

- Tạo thể hiện CriteriaBuilder

```
CriteriaBuilder builder = session.getCriteriaBuilder();
```

- Tạo đối tượng truy vấn là thể hiện của CriteriaQuery (T là một POJO class)

```
CriteriaQuery<T> query = builder.createQuery(T.class)
```

- Thiết lập truy vấn Root bằng cách gọi phương thức from() của đối tượng CriteriaQuery.

```
Root<T> root = query.from(T.class);
```

- Chỉ định loại kết quả truy vấn bằng cách gọi phương thức select() của đối tượng CriteriaQuery.

```
query.select(root);
```

- Tạo thể hiện org.hibernate.query.Query để thực thi truy vấn bằng cách gọi phương thức createQuery() của đối tượng Session với đối số là loại kết quả truy vấn.

```
Query<T> q = session.createQuery(query);
```

- Thực thi câu truy vấn bằng cách gọi phương thức getResultList() hoặc getSingleResult() của đối tượng org.hibernate.query.Query.

```
List<T> list = q.getResultList();
```

Ví dụ 1: lấy danh sách sản phẩm cho phép lọc dữ liệu theo tên hoặc theo mô tả hoặc theo tên nhà sản xuất.

```
public List<Product> getProducts(String kw) {
    Session session = sessionFactory.openSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Product> cr = builder.createQuery(Product.class);
    Root<Product> root = cr.from(Product.class);
```

```

CriteriaQuery<Product> query = cr.select(root);
if (!kw.isEmpty()) {
    String pattern = String.format("%%%s%%", kw);
    Predicate p1 = builder.like(root.get("name")
        .as(String.class), pattern);
    Predicate p2 = builder.like(root.get("description")
        .as(String.class), pattern);
    Predicate p3 = builder.like(root.get("manufacturer")
        .as(String.class), pattern);
    query = query.where(builder.or(p1, p2, p3));
}
List<Product> products = session.createQuery(query).getResultList();
session.close();
return products;
}

```

Ví dụ 2: lấy danh sách sản phẩm theo giá

```

public List<Product> getProductsByPrice(BigDecimal fromPrice,
                                         BigDecimal toPrice) {
    Session session = sessionFactory.openSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Product> cr = builder.createQuery(Product.class);
    Root<Product> root = cr.from(Product.class);
    CriteriaQuery query = cr.select(root);
    Predicate p1 = builder.greaterThanOrEqualTo(root.get("price")
        .as(BigDecimal.class), fromPrice);
    Predicate p2 = builder.lessThanOrEqualTo(root.get("price")
        .as(BigDecimal.class), toPrice);
    query = query.where(builder.and(p1, p2));
    List<Product> products = session.createQuery(query).getResultList();
    session.close();
    return products;
}

```

Ví dụ 3: đếm số lượng, giá cao nhất, giá thấp nhất, giá trung bình các sản phẩm.

```
public void productStats() {
    Session session = sessionFactory.openSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Object[]> cr = builder.createQuery(Object[].class);
    Root<Product> root = cr.from(Product.class);
    cr.multiselect(builder.count(root.get("id")),
                    builder.max(root.get("price")).as(BigDecimal.class)),
    builder.min(root.get("price")).as(BigDecimal.class)),
    builder.avg(root.get("price")).as(BigDecimal.class)));

    Query<Object[]> query = session.createQuery(cr);
    Object[] k = query.getSingleResult();
    System.out.println("Số lượng sản phẩm: " + k[0]);
    System.out.println("Giá cao nhất: " + k[1]);
    System.out.println("Giá thấp nhất: " + k[2]);
    System.out.println("Giá trung bình: " + k[3]);
    session.close();
}
```

Ví dụ 4: tương tự ví dụ 3 nhưng gom nhóm theo từng danh mục. Yêu cầu hiển thị tên danh mục và các thông tin thống kê tương ứng.

```
public void productStats() {
    Session session = sessionFactory.openSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Object[]> cr = builder.createQuery(Object[].class);

    Root<Product> rootPro = cr.from(Product.class);
    Root<Category> rootCat = cr.from(Category.class);
    cr.where(builder.equal(rootPro.get("category"), rootCat.get("id")));
    cr.multiselect(rootCat.get("name"),
                    builder.count(rootPro.get("id")),
                    builder.max(rootPro.get("price")).as(BigDecimal.class)),
```

```
        builder.min(rootPro.get("price").as(BigDecimal.class)),
        builder.avg(rootPro.get("price").as(BigDecimal.class)));
cr.groupBy(rootCat.get("name"));
cr.orderBy(builder.asc(rootCat.get("name")));

Query<Object[]> query = session.createQuery(cr);
List<Object[]> rs = query.getResultList();
for (Object[] obj: rs) {
    System.out.println("Danh mục: " + obj[0]);
    System.out.println("Số lượng sản phẩm: " + obj[1]);
    System.out.println("Giá cao nhất: " + obj[2]);
    System.out.println("Giá thấp nhất: " + obj[3]);
    System.out.println("Giá trung bình: " + obj[4]);
    System.out.println("=====");
}
session.close();
}
```

BÀI 6. PHÁT TRIỂN ỨNG DỤNG VỚI SPRING MVC VÀ HIBERNATE

Học xong bài này người học sẽ nắm được các nội dung sau:

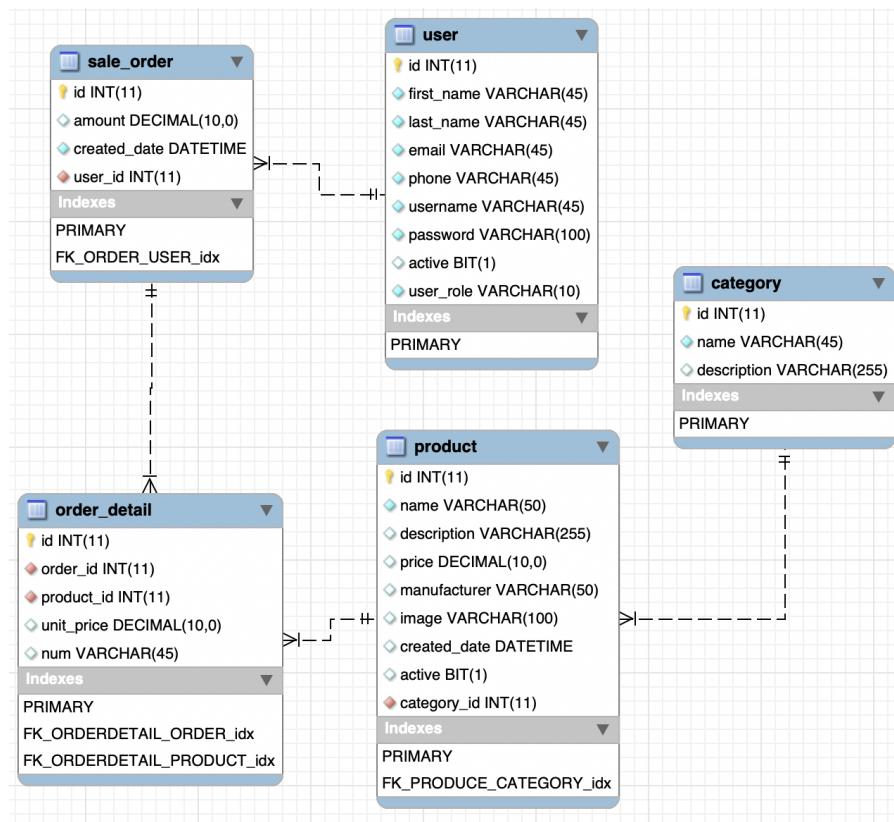
- *Biết cách tạo project phát triển ứng dụng Web bằng Spring MVC.*
- *Biết cấu hình Hibernate trong ứng dụng Spring MVC.*
- *Biết kiến trúc xây dựng một ứng dụng Web bằng Spring MVC kết hợp Hibernate.*
- *Biết sử dụng các kỹ thuật quan trọng Spring MVC hỗ trợ: Bean Validation, Spring Tiles, Spring Security, Restful API.*

1.31 MÔ TẢ BÀI TOÁN

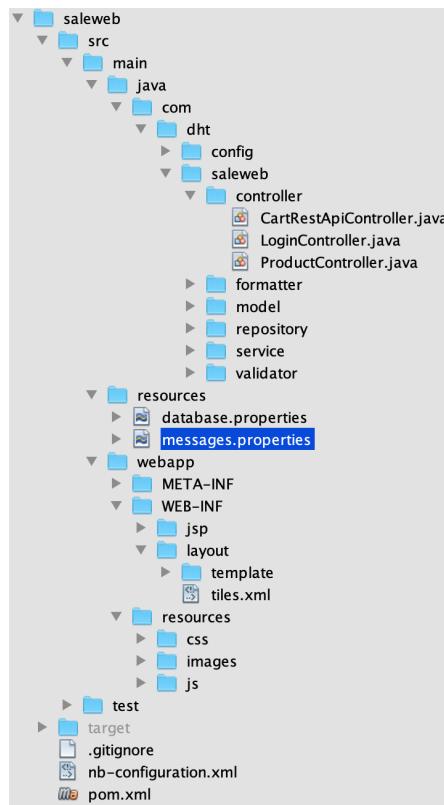
Trong chương này xây dựng ứng dụng Web cơ bản sử dụng Spring MVC kết hợp tương tác với cơ sở dữ liệu quan hệ MySQL thông qua Hibernate, hệ thống bán hàng đơn giản có hai phân hệ người dùng admin và người dùng thường, bao gồm các chức năng cơ bản sau:

- Đăng nhập, đăng ký
- Thêm sản phẩm
- Xem thông tin sản phẩm, chi tiết sản phẩm và tra cứu sản phẩm
- Giỏ hàng và thanh toán

Các chức năng người dùng đều có thể thực hiện, riêng chức năng thêm sản phẩm chỉ có người dùng là admin mới được thực hiện. Ngoài ra chức năng thêm sản phẩm và thanh toán bắt buộc người dùng phải đăng nhập. Trước tiên, tạo cơ sở dữ liệu đơn giản với tên saledb theo lược đồ quan hệ Hình 31. Tạo project maven phát triển ứng dụng Web với tên saleweb. Cấu trúc project có dạng Hình 32.



Hình 31. Lược đồ cơ sở dữ liệu quan hệ của project saleweb.



Hình 32. Cấu trúc project saleweb.

Tập tin pom.xml có các dependency sau:

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
```

```
<artifactId>spring-orm</artifactId>
<version>5.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.10.Final</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.18</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
```

```
</dependency>
</dependencies>
```

Đầu tiên, ta cần tạo tập tin properties chứa thông tin các thuộc tính cấu hình kết nối cơ sở dữ liệu. Tạo thư mục resources trong thư mục src/main, trong thư mục này tạo tập tin database.properties có nội dung như sau:

```
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.showSql=true
hibernate.connection.driverClass=com.mysql.cj.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/saledb
hibernate.connection.username=root
hibernate.connection.password=12345678
```

Trong thư mục src/main/java tạo gói com.dht.config chứa các tập tin cấu hình bằng mã nguồn Java.

- Tạo tập tin HibernateConfig.java như sau:

```
package com.dht.config;

import java.util.Properties;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.context.annotation.PropertySource;
import static org.hibernate.cfg.Environment.*;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate5.HibernateTransactionManager;

@Configuration
@PropertySource("classpath:database.properties")
public class HibernateConfig {
```

```
@Autowired
private Environment env;
@Bean
public LocalSessionFactoryBean getSessionFactory() {
    LocalSessionFactoryBean sessionFactory
        = new LocalSessionFactoryBean();
    sessionFactory.setPackagesToScan(new String[] {
        "com.dht.saleweb.model"
    });
    sessionFactory.setDataSource(dataSource());
    sessionFactory.setHibernateProperties(hibernateProperties());
    return sessionFactory;
}
@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource
        = new DriverManagerDataSource();
    dataSource.setDriverClassName(
        env.getProperty("hibernate.connection.driverClass"));
    dataSource.setUrl(env.getProperty("hibernate.connection.url"));
    dataSource.setUsername(
        env.getProperty("hibernate.connection.username"));
    dataSource.setPassword(
        env.getProperty("hibernate.connection.password"));
    return dataSource;
}
private Properties hibernateProperties() {
    Properties props = new Properties();
    props.put(DIALECT, env.getProperty("hibernate.dialect"));
    props.put(SHOW_SQL, env.getProperty("hibernate.showSql"));
    return props;
}
```

```

@Bean
public HibernateTransactionManager transactionManager() {
    HibernateTransactionManager transactionManager
        = new HibernateTransactionManager();
    transactionManager.setSessionFactory(
        getSessionFactory().getObject());
    return transactionManager;
}
}

```

`@EnableTransactionManagement` cho phép khả năng sử dụng quản lý giao tác thông qua annotation của Spring. Quản lý giao tác là kỹ thuật lập trình quan trọng trong phát triển các ứng dụng thương mại để đảm bảo tính nhất quán và toàn vẹn dữ liệu. `HibernateTransactionManager` kết buộc Session từ một SessionFactory vào một thread, cho phép một Session cho mỗi SessionFactory.

`@Bean dataSource()`: việc tạo kết nối đến cơ sở dữ liệu tốn nhiều thời gian, đặc biệt trong môi trường mạng, nên rất cần thiết cho việc tái sử dụng, cũng như chia sẻ sử dụng các kết nối đã mở (connection pool). Việc tạo Bean `dataSource` có nhiệm vụ tối ưu việc sử dụng các kết nối này.

`@Bean getSessionFactory`: sử dụng `LocalSessionFactoryBean` tạo `SessionFactory`.

- Tạo tập tin `WebApplicationContextConfig.java` như sau:

```

package com.dht.config;

import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.web.multipart.commons.CommonsMultipartResolver;
import
org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;

```

```
import
org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@EnableWebMvc
@EnableTransactionManagement
@ComponentScan(basePackages = "com.dht.saleweb")
public class WebApplicationContextConfig implements WebMvcConfigurer {

    @Override
    public void configureDefaultServletHandling(
            DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/images/**")
                .addResourceLocations("/resources/images/");
        registry.addResourceHandler("/js/**")
                .addResourceLocations("/resources/js/");
        registry.addResourceHandler("/css/**")
                .addResourceLocations("/resources/css/");
    }

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource resource
                = new ResourceBundleMessageSource();
        resource.setBasename("messages");
    }
}
```

```

        return resource;
    }

    @Bean
    public CommonsMultipartResolver multipartResolver() {
        CommonsMultipartResolver resolver
            = new CommonsMultipartResolver();
        resolver.setDefaultEncoding("UTF-8");
        return resolver;
    }
}

```

- Tạo tập tin DispatcherServletInitializer.java như sau:

```

package com.dht.config;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcher
ServletInitializer;

public class DispatcherServletInitializer
    extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {
            HibernateConfig.class
        };
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] {
            WebApplicationContextConfig.class
        };
    }
    @Override
    protected String[] getServletMappings() {

```

```

        return new String[] {"/"};
    }
}

```

1.32 KIẾN TRÚC SPRING WEB MVC

1.32.1 Domain Layer

Domain layer chứa các domain model đại diện cho các loại lưu trữ dữ liệu dựa trên các yêu cầu logic nghiệp vụ. Ví dụ tạo gói com.dht.saleweb.model trong thư mục src/main/java, trong gói này lần lượt tạo các lớp Category.java, Product.java.

Category.java

```

package com.dht.saleweb.model;

import java.io.Serializable;
import java.util.Set;
import javax.enterprise.inject.spi.Producer;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "category")
public class Category implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id")

```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;

@Column(name = "name")
private String name;

@Column(name = "description")
private String description;

@OneToMany(mappedBy = "category")
private Set<Product> products;

// Các phương thức getter/setter
}
```

Product.java

```
package com.dht.saleweb.model;

import java.math.BigDecimal;
import java.util.Date;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;

@Entity
@Table(name = "product")
public class Product implements Serializable {
```

```
private static final long serialVersionUID = 1L;

@Id
@Column(name = "id")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;

@Column(name = "name")
private String name;

@Column(name = "description")
private String description;

@Column(name = "price")
private BigDecimal price;

@Column(name = "manufacturer")
private String manufacturer;

@Temporal(javax.persistence.TemporalType.DATE)
@Column(name = "created_date")
private Date createdDate;

@Column(name = "active")
private boolean active;

@ManyToOne
@JoinColumn(name = "category_id")
private Category category;

{
    createdDate = new Date();
```

```

        active = true;
    }

    // Các phương thức getter/setter
}

```

1.32.2 Persistence Layer

Việc xử lý truy vấn dữ liệu từ cơ sở dữ liệu cần được tách thành một tầng riêng giúp việc tái sử dụng logic xử lý tương tác dữ liệu hiệu quả hơn ở các controller và các tầng khác. Các công việc này được thực hiện ở tầng Persistence.

Đối tượng repository có nhiệm vụ thực hiện các thao tác thêm, xoá, cập nhật và truy vấn dữ liệu (CRUD: create, read, update, delete) trên các đối tượng domain (domain object). `@Repository` (`org.springframework.stereotype.Repository`) là chỉ định lớp Repository.

Persistence layer chứa các đối tượng repository để truy cập vào các đối tượng domain, đối tượng repository gửi các câu truy vấn tới data source của dữ liệu, ánh xạ (map) dữ liệu từ data source đến đối tượng domain, và cuối cùng nó lưu trữ bền vững (persist) sự thay đổi của đối tượng domain xuống data source.

Ví dụ tạo gói `com.dht.saleweb.repository` trong `src/main/java`, trong gói này lần lượt tạo các interface sau:

```

public interface CategoryRepository {
    List<Category> getCategories(String kw);
    Category getCategoryById(int id);
    List<Product> getProductsByCategory(int cateId);
}

public interface ProductRepository {
    List<Product> getProducts(String kw);
    List<Product> getProductsByPrice(BigDecimal fromPrice,
                                      BigDecimal toPrice);
    Product getProductById(int id);
}

```

```
    void addProduct(Product product);  
}
```

Tạo gói com.dht.saleweb.repository.impl, trong gói này lần lượt tạo các lớp hiện thực các giao diện như sau:

CategoryRepositoryImpl.java

```
package com.dht.saleweb.repository.impl;  
  
import com.dht.saleweb.model.Category;  
import com.dht.saleweb.model.Product;  
import com.dht.saleweb.repository.CategoryRepository;  
import java.util.List;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import org.hibernate.Hibernate;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public class CategoryRepositoryImpl implements CategoryRepository {  
    @Autowired  
    private SessionFactory sessionFactory;  
  
    @Override  
    public List<Category> getCategories(String kw) {  
        List<Category> cates;  
        Session session = sessionFactory.getCurrentSession();  
        CriteriaBuilder builder = session.getCriteriaBuilder();  
        CriteriaQuery<Category> cr
```

```

        = builder.createQuery(Category.class);

Root<Category> root = cr.from(Category.class);

CriteriaQuery query = cr.select(root);
if (!kw.isEmpty())
    query = query.where(builder.like(
        root.get("name").as(String.class),
        "%" + kw + "%"));

cates = session.createQuery(query).getResultList();
return cates;
}

@Override
public Category getCategoryById(int id) {
    Category cat;
    Session session = sessionFactory.getCurrentSession();
    cat = session.get(Category.class, id);
    return cat;
}

@Override
public List<Product> getProductsByCategory(int id) {
    List<Product> products = null;
    Session session = sessionFactory.getCurrentSession();
    Category cate = session.get(Category.class, id);
    if (cate != null) {
        Hibernate.initialize(cate.getProducts());
        products = cate.getProducts();
    }
    return products;
}
}

```

ProductRepositoryImpl.java

```
package com.dht.saleweb.repository.impl;
```

```
import com.dht.saleweb.model.Product;
import com.dht.saleweb.repository.ProductRepository;
import java.math.BigDecimal;
import java.util.List;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class ProductRepositoryImpl implements ProductRepository {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public List<Product> getProducts(String kw) {
        List<Product> products;
        Session session = sessionFactory.getCurrentSession();
        CriteriaBuilder builder = session.getCriteriaBuilder();
        CriteriaQuery<Product> cr = builder.createQuery(Product.class);
        Root<Product> root = cr.from(Product.class);

        CriteriaQuery<Product> query = cr.select(root);
        if (!kw.isEmpty()) {
            String pattern = String.format("%%%s%%", kw);
            Predicate p1 = builder.like(
                root.get("name").as(String.class), pattern);
            Predicate p2 = builder.like(

```

```
        root.get("description").as(String.class), pattern);
    Predicate p3 = builder.like(
        root.get("manufacturer").as(String.class), pattern);
    query = query.where(builder.or(p1, p2, p3));
}
products = session.createQuery(query).getResultList();
return products;
}

@Override
public List<Product> getProductsByPrice(BigDecimal fromPrice,
                                             BigDecimal toPrice) {
    List<Product> products;
    Session session = sessionFactory.getCurrentSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Product> cr = builder.createQuery(Product.class);
    Root<Product> root = cr.from(Product.class);

    CriteriaQuery query = cr.select(root);
    Predicate p1 = builder.greaterThanOrEqualTo(
        root.get("price").as(BigDecimal.class), fromPrice);
    Predicate p2 = builder.lessThanOrEqualTo(
        root.get("price").as(BigDecimal.class), toPrice);
    query = query.where(builder.and(p1, p2));
    products = session.createQuery(query).getResultList();
    return products;
}

@Override
public Product getProductById(int id) {
    return sessionFactory.getCurrentSession().get(Product.class, id);
}

@Override
public void addProduct(Product product) {
    sessionFactory.getCurrentSession().save(product);
```

```
}
```

1.32.3 Service Layer

Service Layer chứa các xử lý nghiệp vụ phức tạp tương tác với cơ sở dữ liệu, bao gồm nhiều thao tác CRUD, thực hiện trên nhiều đối tượng repository.

Tạo gói com.dht.saleweb.service và lần lượt tạo các giao diện CategoryService, ProductService trong gói này.

```
public interface CategoryService {
    List<Category> getCategories();
    List<Product> getProductsByCategory(int cateId);
}

public interface ProductService {
    List<Product> getProducts(String kw);
    Product getProductById(int productId);
    void addProduct(Product product, String rootDir);
}
```

Tạo gói com.dht.saleweb.service.impl, trong gói này tạo các lớp hiện thực các giao diện trong gói service.

CategoryServiceImpl.java

```
package com.dht.saleweb.service.impl;

import com.dht.saleweb.model.Category;
import com.dht.saleweb.model.Product;
import com.dht.saleweb.repository.CategoryRepository;
import com.dht.saleweb.service.CategoryService;
import java.util.List;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;

@Service
public class CategoryServiceImpl implements CategoryService {
    @Autowired
    private CategoryRepository categoryRepository;

    @Override
    @Transactional(readOnly = true)
    public List<Category> getCategories() {
        return categoryRepository.getCategories("");
    }

    @Override
    @Transactional(readOnly = true)
    public List<Product> getProductsByCategory(int cateId) {
        return categoryRepository.getProductsByCategory(cateId);
    }
}
```

ProductServiceImpl.java

```
package com.dht.saleweb.service.impl;

import com.dht.saleweb.model.Product;
import com.dht.saleweb.repository.ProductRepository;
import com.dht.saleweb.service.ProductService;
import java.io.File;
import java.io.IOException;
import java.util.List;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

@Service
```

```
public class ProductServiceImpl implements ProductService {
    @Autowired
    private ProductRepository productRepository;

    @Override
    @Transactional(readOnly = true)
    public List<Product> getProducts(String kw) {
        return productRepository.getProducts(kw);
    }

    @Override
    @Transactional(readOnly = true)
    public Product getProductById(int productId) {
        return productRepository.getProductById(productId);
    }

    @Override
    @Transactional
    public void addProduct(Product product, String rootDir) {
        MultipartFile img = product.getImgFile();
        if (img != null && !img.isEmpty()) {
            product.setImage("/images/uploads/" +
                img.getOriginalFilename());
            try {
                img.transferTo(new File(rootDir + "resources" +
                    product.getImage()));
                productRepository.addProduct(product);
            } catch (IOException | IllegalStateException ex) {
                System.err.println(ex.getMessage());
            }
        }
    }
}
```

@Transactional có thể được sử dụng cho phương thức để định nghĩa phương thức public là giao tác hoặc cho lớp, khi nó được dùng cho lớp thì tất cả các phương thức public của lớp đó sẽ được định nghĩa là giao tác. Chú ý để annotation này làm việc thì thêm @EnableTransactionManagement cho lớp cấu hình Java.

1.33 TEMPLATE WITH TILES

Apache Tiles là một framework mã nguồn mở giúp tái sử dụng tối đa khi xây dựng các front-end template. Tiles cho phép lập trình viên định nghĩa các phần con (tiles) để lắp ráp thành một trang web hoàn chỉnh khi ứng dụng thực thi, những phần con này có các tham số với giá trị có thể thay đổi khi chương trình thực thi. Điều này giúp tăng khả năng tái sử dụng template và giảm số lượng mã nguồn trùng lắp.

Dựa trên template <https://freshdesignweb.com/demo/template/ustora/>, ta chỉnh sửa cho project saleweb. Đầu tiên bổ sung dependency org.apache.tiles vào tập tin pom.xml.

```
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-extras</artifactId>
    <version>3.0.8</version>
</dependency>
```

Trong ví dụ này ta tạo cấu trúc một trang Web trong Website gồm 3 phần chính là header, content và footer. Trong đó phần header và footer của các trang Web không đổi, chỉ thay đổi phần content. Bây giờ tạo thư mục WEB-INF/, trong thư mục này tạo tập tin tiles.xml như bên dưới. Tập tin tiles.xml là tập tin rất quan trọng trong phát triển ứng dụng dựa trên Apache Tiles.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC "-//Apache
Software Foundation//DTD Tiles Configuration 3.0//EN"
"http://tiles.apache.org/dtds/tiles-config_3_0.dtd">

<tiles-definitions>
```

```

<definition name="baseLayout"
            template="/WEB-INF/layout/template/base.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header"
                  value="/WEB-INF/layout/template/header.jsp" />
    <put-attribute name="content" value="" />
    <put-attribute name="footer"
                  value="/WEB-INF/layout/template/footer.jsp" />
</definition>
<definition name="index" extends="baseLayout">
    <put-attribute name="title" value="Trang chủ" />
    <put-attribute name="content" value="/WEB-INF/jsp/index.jsp" />
</definition>
<definition name="detail" extends="baseLayout">
    <put-attribute name="title" value="Chi tiết sản phẩm" />
    <put-attribute name="content" value="/WEB-INF/jsp/detail.jsp" />
</definition>
<definition name="add-product" extends="baseLayout">
    <put-attribute name="title" value="Thêm sản phẩm" />
    <put-attribute name="content"
                  value="/WEB-INF/jsp/add-product.jsp" />
</definition>
<definition name="cart" extends="baseLayout">
    <put-attribute name="title" value="Giỏ hàng" />
    <put-attribute name="content" value="/WEB-INF/jsp/cart.jsp" />
</definition>
</tiles-definitions>

```

Tập tin tiles.xml chứa các định nghĩa kết hợp với các template, mỗi định nghĩa được chỉ định bằng thẻ `<definition>`, mỗi định nghĩa có nhiều thuộc tính được chỉ định bằng thẻ `<put-attribute>` bên trong `<definition>`, giá trị của các thuộc tính này có thể chèn vào template thông qua thẻ `<tiles:insertAttribute name="">`. Apache Tiles cũng cho phép một định nghĩa kế thừa một định nghĩa khác thông qua thuộc tính `extends`

trong thẻ <definition>, định nghĩa con có thể kế thừa các thuộc tính từ định nghĩa cha và cũng có thể ghi đè giá trị các thuộc tính khi cần. Chẳng hạn định nghĩa index kế thừa định nghĩa baseLayout, ghi đè hai thuộc tính title và content, còn các thuộc tính header và footer nhận từ định nghĩa baseLayout.

Tiếp theo định nghĩa các template cho các định nghĩa trong tiles.xml. Trong thư mục template trong thư mục layout, trong thư mục này lần lượt tạo các trang base.jsp, header.jsp và footer.jsp như sau:

Tập tin base.jsp

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title><tiles:insertAttribute name="title" /></title>
        <link rel="stylesheet"
            href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
        <link rel="stylesheet"
            href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
        <link rel="stylesheet" href="

```

```

<script src="https://code.jquery.com/jquery.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
</script>
<script src="

```

Tập tin header.jsp

```

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<div class="header-area">
    <div class="container">
        <div class="row">
            <div class="col-md-7">
                <div class="user-menu">
                    <ul>
                        <li>
                            <a href="#"><i class="fa fa-check"></i>
                            Đăng ký</a>
                        </li>
                        <li>
                            <a href="#"><i class="fa fa-user"></i>
                            Đăng nhập</a>
                        </li>
                    </ul>
                </div>
            </div>
            <div class="col-md-5">
                <form action="

```

Tập tin footer.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<div class="footer-bottom-area">
```

```

<div class="container">
    <div class="row">
        <div class="col-md-8">
            <div class="copyright">
                <p>Spring Web MVC © 2020.</p>
            </div>
        </div>
    </div>
</div>

```

Tiếp theo thiết lập cấu hình cho Spring Tiles, trong gói com.dht.config tạo TilesConfig.java như sau:

```

package com.dht.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.view.UrlBasedViewResolver;
import org.springframework.web.servlet.view.tiles3.TilesConfigurer;
import org.springframework.web.servlet.view.tiles3.TilesView;

@Configuration
public class TilesConfig {
    @Bean
    public UrlBasedViewResolver viewResolver() {
        UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();
        viewResolver.setViewClass(TilesView.class);
        viewResolver.setOrder(-2);
        return viewResolver;
    }

    @Bean
    public TilesConfigurer tilesConfigurer() {

```

```

        TilesConfigurer configurer = new TilesConfigurer();
        configurer.setDefinitions("/WEB-INF/layout/tiles.xml");
        configurer.setCheckRefresh(true);
        return configurer;
    }
}

```

Ta định nghĩa UrlBasedViewResolver để phân giải tên các View thành tên các Tiles View và cấu hình TilesConfigurer để chỉ định tập tin chứa các định nghĩa cho Apache Tiles, chỉnh sửa phương thức getRootConfigClasses() của DispatcherServletInitializer như sau:

```

@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] {
        HibernateConfig.class, TilesConfig.class
    };
}

```

Tạo gói com.dht.saleweb.controller, trong gói này tạo ProductController như sau:

```

package com.dht.saleweb.controller;

import com.dht.saleweb.service.CategoryService;
import com.dht.saleweb.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class ProductController {
    @Autowired
    private CategoryService categoryService;
}

```

```
@Autowired  
private ProductService productService;  
  
@GetMapping(value = "/")  
public ModelAndView index(  
    @RequestParam(value = "cat_id", defaultValue = "") String cateId,  
    @RequestParam(value = "kw", defaultValue = "") String kw) {  
    ModelAndView view = new ModelAndView();  
    view.setViewName("index");  
    view.addObject("categories", categoryService.getCategories());  
    if (cateId.isEmpty())  
        view.addObject("products", productService.getProducts(kw));  
    else  
        view.addObject("products",  
categoryService.getProductsByCategory(Integer.parseInt(cateId)));  
  
    return view;  
}  
  
@GetMapping(value = "/products/{product_id}")  
public ModelAndView detail(  
    @PathVariable(value = "product_id") int productId) {  
    ModelAndView view = new ModelAndView();  
    view.setViewName("detail");  
    view.addObject("product",  
        productService.getProductById(productId));  
    return view;  
}  
}
```

Tạo tập tin WEB-INF/jsp/index.jsp và xoá rỗng các nội dung tập tin này. Chạy ứng dụng và truy cập vào trang chủ <http://localhost:8080/saleweb/> kết quả như sau:



Chỉnh sửa tập tin index.jsp, nội dung của trang này sẽ hiển thị trong phần content khi sử dụng định nghĩa index để hiển thị.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<div class="mainmenu-area">
    <div class="container">
        <div class="row">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle"
                    data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li class="active"><a href="#">Trang chủ</a></li>
                    <c:forEach items="${categories}" var="category">
                        <li>
                            <a href="

```

```

        </c:forEach>

        <li><a href="#">Liên hệ</a></li>
    </ul>
</div>
</div>
</div>
</div>

<div class="container">
    <h2 class="section-title">Danh sách sản phẩm</h2>
    <div class="row">
        <c:forEach items="${products}" var="product">
            <div class="col-md-3 single-product">
                <div class="product-f-image">
                    " />
                    <div class="product-hover">
                        <a href="#" class="add-to-cart-link"><i class="fa fa-shopping-cart"></i> Thêm vào giỏ</a>
                        <a href="<spring:url value="/products/${product.id}" />" class="view-details-link"><i class="fa fa-link"></i> Xem chi tiết</a>
                    </div>
                </div>
                <h2><a href="#">${product.name}</a></h2>
                <div class="product-carousel-price">
                    <ins>${product.price} VNĐ</ins>
                </div>
            </div>
        </c:forEach>
    </div>
</div>

```

Chạy ứng dụng và truy cập vào trang chủ <http://localhost:8080/saleweb/> kết quả:

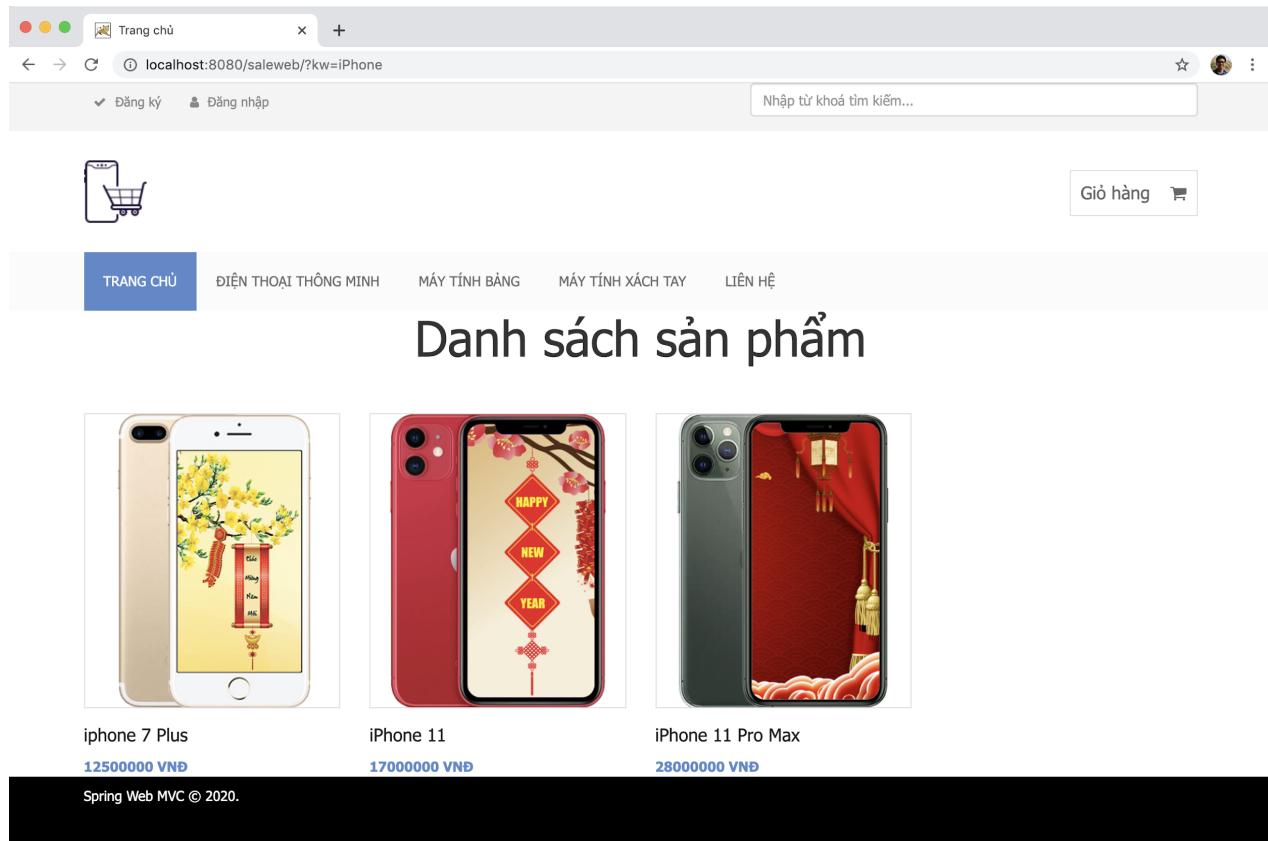
Danh sách sản phẩm

iphone 7 Plus 12500000 VND	Galaxy Note 10 21000000 VND	iPhone 11 17000000 VND	iPhone 11 Pro Max 28000000 VND
Galaxy Tab S6 18600000 VND	iPad Mini 7.9 12990000 VND		

Spring Web MVC © 2020.

Click trên liên kết các danh mục xem sản phẩm theo danh mục, click vào “MÁY TÍNH BẢNG” sẽ truy cập vào http://localhost:8080/saleweb/?cat_id=1.

Nhập từ khoá tìm kiếm vào thanh tìm kiếm gốc trên phải và enter, chẳng hạn tìm sản phẩm “iPhone” kết quả như sau:



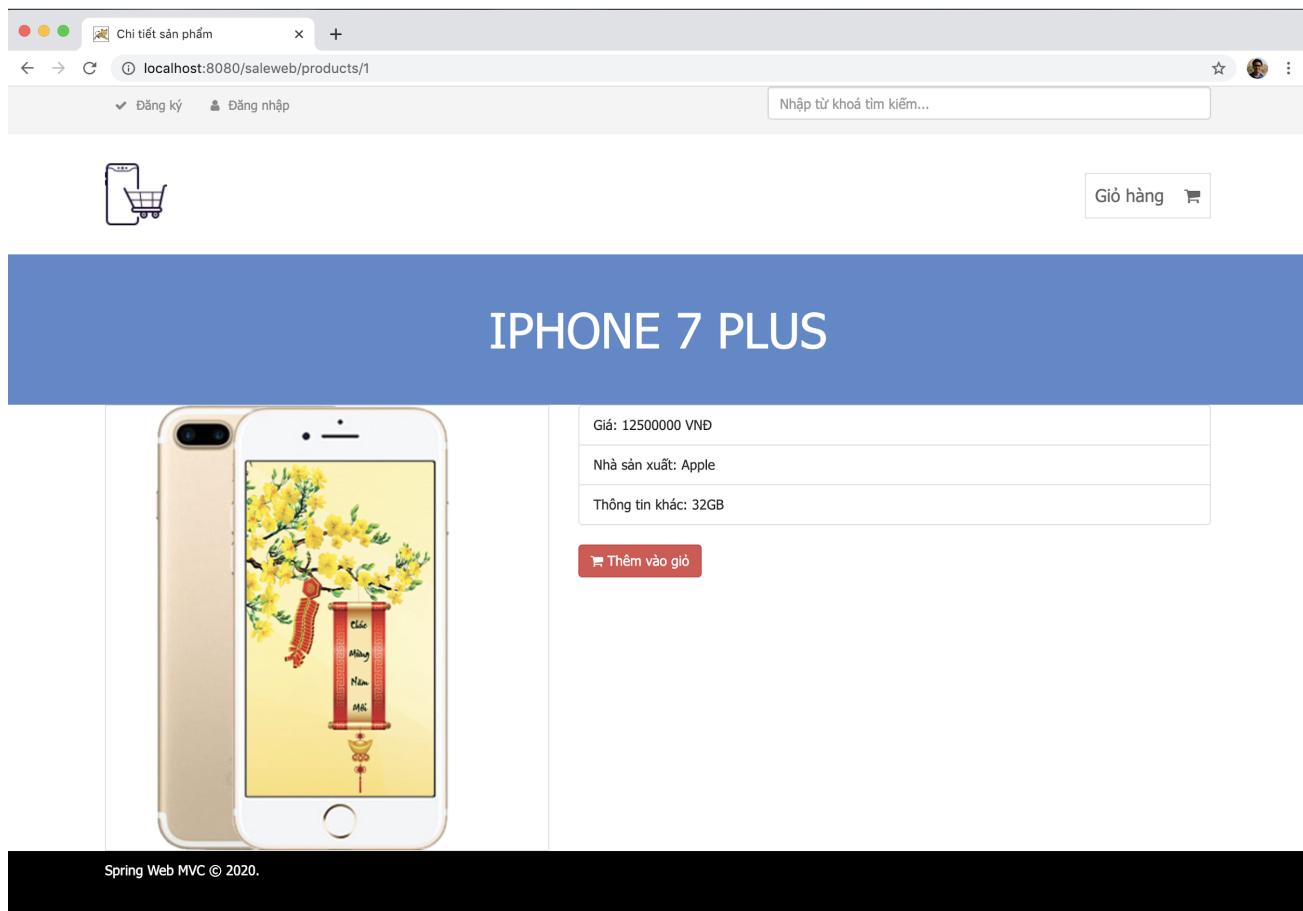
Tương tự rê chuột lên từng sản phẩm vào click vào "XEM CHI TIẾT" để xem chi tiết thông tin sản phẩm, chẳng hạn click vào sản phẩm "iPhone 7 Plus" sẽ truy cập vào trang <http://localhost:8080/saleweb/products/1>. Tuy nhiên trước tiên ta cần tạo template detail.jsp trong WEB-INF/jsp như sau:

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<div class="product-big-title-area">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <div class="product-bit-title text-center">
                    <h2 class="text-uppercase">${product.name}</h2>
                </div>
            </div>
        </div>
    </div>
```

```
</div>
</div>
<div class="container">
    <div class="row">
        <div class="col-md-5">
            <div class="product-f-image">
                " />
                <div class="product-hover">
                    <a href="#" class="add-to-cart-link">
                        <i class="fa fa-shopping-cart"></i>
                        Thêm vào giỏ
                    </a>
                </div>
            </div>
        </div>
        <div class="col-md-7">
            <ul class="list-group">
                <li class="list-group-item">
                    Giá:  ${product.price} VNĐ
                </li>
                <li class="list-group-item">
                    Nhà sản xuất:  ${product.manufacturer}
                </li>
                <li class="list-group-item">
                    Thông tin khác:  ${product.description}
                </li>
            </ul>
            <div>
                <a href="#" class="btn btn-danger">
                    <i class="fa fa-shopping-cart"></i>
                    Thêm vào giỏ
                </a>
            </div>
        </div>
    </div>
</div>
```

```
</div>  
</div>  
</div>
```

Truy cập lại vào chi tiết sản phẩm “iPhone 7 Plus” kết quả như sau:



1.34 BEAN VALIDATION

Một trong các xử lý quan trọng trong các ứng dụng Web là kiểm tra dữ liệu của người dùng mỗi khi gửi dữ liệu vào ứng dụng Web để thực hiện nhiệm vụ nào đó. Trong phần này trình bày cách sử dụng Bean Validation và Spring Validation để kiểm tra dữ liệu người dùng.

1.34.1 Java Bean Validation

Java Bean Validation cho phép mô tả các ràng buộc trên các đối tượng thông qua các annotation, trong đó Hibernate Validator hiện thực các đặc trưng của Bean Validation, ta sử dụng Hibernate Validator để kiểm tra một số dữ liệu đầu vào của chức năng thêm mới sản phẩm.

Bổ sung dependency hibernate-validator vào pom.xml

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.1.Final</version>
</dependency>
```

Trong ProductController bổ sung phương thức sau.

```
@GetMapping(value = "/products/add")
public String addProductView(Model model) {
    model.addAttribute("product", new Product());
    model.addAttribute("categories",
                      categoryService.getCategories());
    return "add-product";
}
```

Trong lớp com.dht.saleweb.model.Product bổ sung thuộc tính MultipartFile và các annotation để kiểm tra dữ liệu cho các thuộc tính cần thiết như sau:

- Chiều dài trường name tối thiểu là 10 và tối đa 50 ký tự.
- Chiều dài tối đa của trường description là 255 ký tự.
- Bắt buộc thông tin price và giá trị tối thiểu là 100 nghìn và tối đa 1 tỷ.
- Bắt buộc người dùng phải chọn category.

```
@Entity
@Table(name = "product")
public class Product implements Serializable {
```

```

// ...

@Column(name = "name")
@Size(min = 10, max = 50, message = "{product.name.error.sizeMsg}")
private String name;

@Column(name = "description")
@Size(max = 255, message = "{product.description.error.sizeMsg}")
private String description;

@Column(name = "price")
@NotNull(message = "{product.price.error.notNullMsg}")
@Min(value = 100000, message = "{product.price.error.minMsg}")
@Max(value = 1000000000, message = "{product.price.error.maxMsg}")
private BigDecimal price;

@Column(name = "manufacturer")
@Size(max = 50, message = "{product.manufacturer.error.sizeMsg}")
private String manufacturer;
// ...

@ManyToOne
@JoinColumn(name = "category_id")
@NotNull(message = "{product.category.error.notNullMsg}")
private Category category;

@Transient
private MultipartFile imgFile;
// ...
}

```

Gói javax.validation.constraints chứa các annotation dùng để thiết lập kiểm tra dữ liệu như @Size, @Max, @Min, @NotNull. Thuộc tính message của mỗi annotation chỉ định nội dung lỗi sẽ được hiển thị khi dữ liệu vi phạm ràng buộc chỉ định và các nội

dung này được cấu hình lấy từ messages.properties. Thêm các khai báo sau trong tập tin messages.properties trong src/main/resources

```
product.price.error.notNullMsg=Phải có giá sản phẩm
product.price.error.minMsg=Giá sản phẩm tối thiểu 100.000 VNĐ
product.price.error.maxMsg=Giá sản phẩm tối đa 1 tỷ VNĐ
product.name.error.sizeMsg=Tên sản phẩm tối thiểu 10 và tối đa 50 ký tự
product.description.error.sizeMsg=Mô tả chi tiết tối đa 255 ký tự
product.manufacturer.error.sizeMsg=Nhà sản xuất tối đa 50 ký tự
product.category.error.notNullMsg=Phải chọn danh mục cho sản phẩm
product.image.error.notNullMsg=Phải có ảnh đại diện sản phẩm
```

Bổ sung phương thức xử lý thêm sản phẩm trong ProductController. Để ý annotation @Valid để cho controller biết cần kiểm tra dữ liệu request được submit. Sau khi kiểm tra xong, Spring MVC sẽ lưu kết quả trong đối tượng BindingResult, nên đối tượng result sẽ chứa các thông tin lỗi nếu có.

```
import javax.validation.Valid;
...
@PostMapping(value = "/products/add")
public String addProductProcess(Model model,
        @ModelAttribute(value = "product") @Valid Product product,
        BindingResult result, HttpServletRequest request) {
    if(result.hasErrors()) {
        model.addAttribute("categories",
                categoryService.getCategories());
        return "add-product";
    }
    String rootDir = request.getSession()
            .getServletContext().getRealPath("/");
    productService.addProduct(product, rootDir);
    return "redirect:/";
}
```

Trong WEB-INF/jsp tạo add-product.jsp.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>

<div class="container">
    <h1 class="text-center text-uppercase">
        <spring:message code="product.submit" />
    </h1>
    <spring:url value="/products/add" var="action" />
    <form:form action="${action}" modelAttribute="product"
               method="post" enctype="multipart/form-data" >
        <form:errors path="*"
            cssClass="alert alert-danger" element="div" />
        <div class="form-group">
            <label for="categoryId">
                <spring:message code="product.category" />
            </label>
            <form:select path="category" cssClass="form-control">
                <form:options items="${categories}"
                              itemLabel="name"
                              itemValue="id" />
            </form:select>
            <form:errors path="category" cssClass="text-danger" />
        </div>
        <div class="form-group">
            <label for="nameId">
                <spring:message code="product.name" />
            </label>
            <form:input id="nameId" path="name"
                       cssClass="form-control" />
            <form:errors path="name" cssClass="text-danger" />
        </div>
    </form:form>
</div>
```

```
</div>

<div class="form-group">
    <label for="priceId">
        <spring:message code="product.price" />
    </label>
    <form:input id="priceId" path="price"
                cssClass="form-control" />
    <form:errors path="price" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="manufacturerId">
        <spring:message code="product.manufacturer" />
    </label>
    <form:input id="manufacturerId" path="manufacturer"
                cssClass="form-control" />
    <form:errors path="manufacturer" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="descriptionId">
        <spring:message code="product.description" />
    </label>
    <form:textarea id="descriptionId" path="description"
                  cssClass="form-control" />
    <form:errors path="description" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="imgId">
        <spring:message code="product.image" />
    </label>
    <form:input type="file" id="imgId" path="imgFile"
                cssClass="form-control" />
    <form:errors path="image" cssClass="text-danger" />
</div>
```

```

<div class="form-group">
    <form:button class="pull-right">
        <spring:message code="product.submit" />
    </form:button>
</div>
</form:form>
</div>

```

Các thẻ `<form:errors>`, trong đó thuộc tính path chỉ định tên thuộc tính của đối tượng model khi dữ liệu cung cấp cho nó vi phạm ràng buộc. Để dòng `<form:errors>` đầu tiên thuộc tính path có giá trị là * chỉ định hiển thị lỗi tất cả các trường nếu có.

Tiếp theo, ta cần thiết lập cấu hình cho phép kiểm tra dữ liệu, bổ sung cấu hình `LocalValidatorFactoryBean` trong `WebApplicationContextConfig`.

```

@Bean(name = "validator")
public LocalValidatorFactoryBean validator() {
    LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();
    bean.setValidationMessageSource(messageSource());
    return bean;
}

@Override
public Validator getValidator() {
    return validator();
}

```

Tạo gói `com.dht.saleweb.formatter`, trong gói này tạo `CategoryFormatter.java`

```

package com.dht.saleweb.formatter;

import com.dht.saleweb.model.Category;
import java.text.ParseException;
import java.util.Locale;
import org.springframework.format.Formatter;

```

```

public class CategoryFormatter implements Formatter<Category> {
    @Override
    public String print(Category obj, Locale locale) {
        return String.valueOf(obj.getId());
    }

    @Override
    public Category parse(String text, Locale locale)
            throws ParseException {
        Category cat = new Category();
        cat.setId(Integer.parseInt(text));

        return cat;
    }
}

```

Bổ sung phương thức trong WebApplicationContextConfig

```

@Override
public void addFormatters(FormatterRegistry registry) {
    registry.addFormatter(new CategoryFormatter());
}

```

Truy cập <http://localhost:8080/saleweb/products/add> thêm một sản phẩm mới

Nhập một số thông tin vi phạm ràng buộc và click “THÊM SẢN PHẨM”

BÀI 4. JDBC

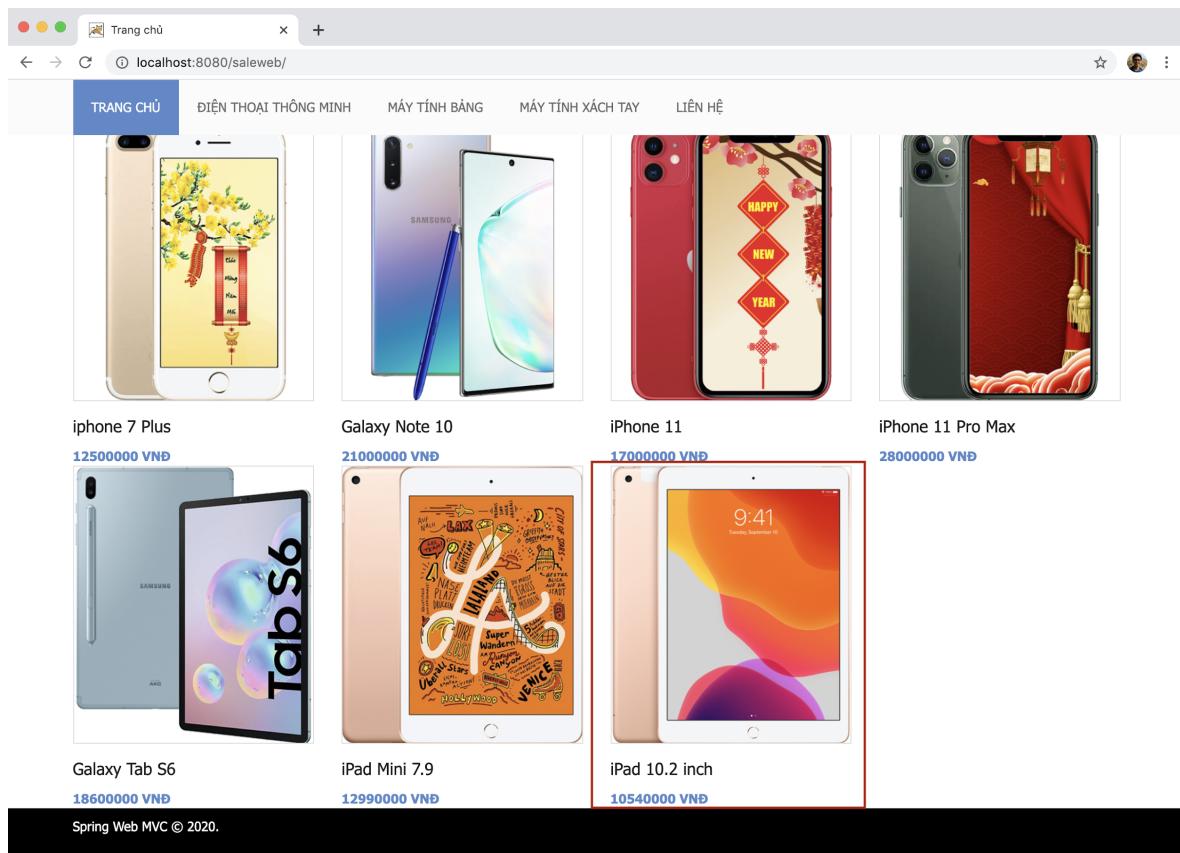
Nhập đúng thông tin một sản phẩm mới.

The screenshot shows a web application titled "Thêm sản phẩm" (Add Product) at the URL "localhost:8080/saleweb/products/add". The page includes a header with navigation links for "Đăng ký" and "Đăng nhập", a search bar, and a shopping cart icon. The main content area is titled "THÊM SẢN PHẨM" and contains the following form fields:

- Danh mục sản phẩm:** A dropdown menu currently set to "Máy tính bảng".
- Tên sản phẩm:** A text input field containing "iPad 10.2 inch".
- Giá sản phẩm:** A text input field containing "10540000".
- Nhà sản xuất:** A text input field containing "Apple".
- Mô tả chi tiết sản phẩm:** A text input field containing "32GB, Wifi Cellular".
- Ảnh sản phẩm:** A file upload input field with the path "Choose File: | Ipad-10-2-inch-wifi-cellular-32gb.png".

A blue "THÊM SẢN PHẨM" button is located at the bottom right of the form. At the very bottom of the page, there is a footer note: "Spring Web MVC © 2020."

Sau khi thêm thành công nó sẽ chuyển về trang chủ với sản phẩm mới thêm.



Trong ví dụ này, ta sử dụng các annotation kiểm tra có sẵn của Bean Validation, nhưng thực tế có những nhiều trường hợp cần kiểm tra phức tạp hơn, không có sẵn trong các annotation chuẩn, khi đó ta cần định nghĩa một validator riêng cho nó. Chẳng hạn ta sẽ tạo một validator cho quy định tên sản phẩm không được phép trùng nhau.

Thêm phương thức kiểm tra tên sản phẩm trong giao diện ProductRepository và hiện thực trong lớp ProductRepository.

```
public interface ProductRepository {
    // ...
    boolean checkProductName(String name);
}

@Repository
public class ProductRepositoryImpl implements ProductRepository {
```

```

// ...

@Override
public boolean checkProductName(String productName) {
    Session session = sessionFactory.getCurrentSession();
    CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<Product> cr = builder.createQuery(Product.class);

    Root<Product> root = cr.from(Product.class);
    CriteriaQuery query = cr.select(root);
    query.where(builder.equal(
        builder.upper(root.get("name")).as(String.class)),
        productName.toUpperCase()));
}

return session.createQuery(query).getSingleResult() == null;
}
}

```

Thêm phương thức kiểm tra trong giao diện ProductService và hiện thực trong lớp ProductServiceImpl.

```

public interface ProductService {
    // ...
    boolean checkProductName(String productName);
}

@Service
public class ProductServiceImpl implements ProductService {
    // ...
    @Override
    @Transactional(readOnly = true)
    public boolean checkProductName(String productName) {
        return productRepository.checkProductName(productName);
    }
}

```

Tạo gói com.dht.saleweb.validator, trong gói này tạo giao diện ProductName.java làm annotation và lớp ProductNameValidator.java. Mỗi giao diện annotation mới cần có @Constraint với thuộc tính validateBy chỉ định lớp sẽ thực hiện kiểm tra dữ liệu thực sự, trong ví dụ này là lớp ProductNameValidator, lớp này phải hiện thực giao diện ConstraintValidator.

- Giao diện ProductName

```
package com.dht.saleweb.validator;

import java.lang.annotation.Documented;
import static java.lang.annotation.ElementType.*;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;

@Target({METHOD, FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = ProductNameValidator.class)
@Documented
public @interface ProductName {
    String message() default "";
    Class<?>[] groups() default {};
    public abstract Class<? extends Payload>[] payload() default {};
}
```

- Lớp ProductNameValidator

```
package com.dht.saleweb.validator;

import com.dht.saleweb.service.ProductService;
import javax.persistence.NoResultException;
import javax.validation.ConstraintValidator;
```

```

import javax.validation.ConstraintValidatorContext;
import org.springframework.beans.factory.annotation.Autowired;

public class ProductNameValidator
        implements ConstraintValidator<ProductName, String> {
    @Autowired
    private ProductService productService;
    @Override
    public void initialize(ProductName constraintAnnotation) {
    }
    @Override
    public boolean isValid(String value,
                          ConstraintValidatorContext context) {
        try {
            return productService.checkProductName(value);
        } catch (NoResultException ex) {
            return false;
        }
    }
}

```

Thêm annotation vừa tạo vào trước thuộc tính name của lớp Product

```

@Entity
@Table(name = "product")
public class Product implements Serializable {
    // ...
    @Column(name = "name")
    @Size(min = 10, max = 50, message = "{product.name.error.sizeMsg}")
    @ProductName(message = "{product.name.error.productNameMsg}")
    private String name;
    // ...
}

```

Bổ sung khai báo message vào tập tin messages.properties.

product.name.error.productNameMsg=Tên sản phẩm này đã tồn tại

Chạy thử chương trình

Tên sản phẩm này đã tồn tại

Danh mục sản phẩm
Điện thoại thông minh

Tên sản phẩm
Galaxy Note 10
Tên sản phẩm này đã tồn tại

Giá sản phẩm
10000000

Nhà sản xuất
Samsung

Mô tả chi tiết sản phẩm
256GB

Ảnh sản phẩm
Choose File No file chosen

THÊM SẢN PHẨM

Spring Web MVC © 2020.

1.34.2 Spring Validation

Spring cũng cung cấp một cơ chế cổ điển cho phép kiểm tra các dữ liệu đầu vào gọi là Spring Validation, so với Bean Validation thì Spring Validation linh hoạt và dễ mở rộng hơn.

Ta minh họa sử dụng Spring Validation kiểm tra dữ liệu giá bán (price) của các sản phẩm phải có giá ít nhất 5 triệu đối với các sản phẩm máy tính xách tay.

Trong gói com.dht.saleweb.validator tạo lớp PriceValidator, lớp này phải hiện thực giao diện Validator (org.springframework.validation.Validator), giao diện này có hai phương thức quan trọng là supports() và validate().

- supports() cho biết validator có được phép kiểm tra lớp chỉ định không.
- validate() là phương thức được gọi để kiểm tra dữ liệu của đối tượng lớp.

```
package com.dht.saleweb.validator;

import com.dht.saleweb.model.Product;
import java.math.BigDecimal;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component
public class PriceValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return Product.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        Product product = (Product) target;

        // Máy tính xách tay trong CSDL hiện tại đang có id là 3
        if (product.getCategory().getId() == 3 &&
            product.getPrice().compareTo(new BigDecimal(5000000)) < 0)
            errors.rejectValue("price",
                               "product.price.error.priceValidatorMsg");
    }
}
```

Bổ sung khai báo trong messages.properties

```
product.price.error.priceValidatorMsg=Máy tính xách tay phải có giá ít  
nhất là 5 triệu
```

Trong lớp com.dht.saleweb.controller.ProductController khai báo thuộc tính tham
chiếu đến đối tượng của PriceValidator và khai báo phương thức @InitBinder như sau:

```
@Controller  
public class ProductController {  
    @Autowired  
    private CategoryService categoryService;  
    @Autowired  
    private ProductService productService;  
    @Autowired  
    private PriceValidator priceValidator;  
  
    @InitBinder  
    public void initBinder(WebDataBinder binder) {  
        binder.setValidator(priceValidator);  
    }  
    // ...  
}
```

Chạy thử chương trình kết quả như sau:

Máy tính xách tay phải có giá ít nhất là 5 triệu

Danh mục sản phẩm

Tên sản phẩm

Giá sản phẩm

Nhà sản xuất

Mô tả chi tiết sản phẩm

Ảnh sản phẩm

THÊM SẢN PHẨM

Spring Web MVC © 2020.

Vì ta đã sử dụng Spring Validator để kiểm tra dữ liệu, nên những validator đã thiết lập trước đó dựa trên Bean Validator sẽ không còn tác dụng, Spring MVC sẽ bỏ qua các annotation như @Min, @Max, @Size. Để có thể kết hợp Spring Validation và Bean Validation ta có thể làm như sau:

Tạo lớp WebAppValidator trong gói com.dht.saleweb.validator

```
package com.dht.saleweb.validator;

import com.dht.saleweb.model.Product;
import java.util.HashSet;
import java.util.Set;
import javax.validation.ConstraintViolation;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component
public class WebAppValidator implements Validator {
    @Autowired
    private javax.validation.Validator beanValidator;

    private Set<Validator> springValidators = new HashSet<>();

    @Override
    public boolean supports(Class<?> clazz) {
        return Product.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        Set<ConstraintViolation<Object>> constraintViolations
            = beanValidator.validate(target);

        for (ConstraintViolation<Object> obj: constraintViolations)
            errors.rejectValue(obj.getPropertyPath().toString(),
                obj.getMessageTemplate(), obj.getMessage());

        for (Validator obj: springValidators)
            obj.validate(target, errors);
    }

    public void setSpringValidators(Set<Validator> springValidators) {
        this.springValidators = springValidators;
    }
}
```

Trong lớp WebApplicationContextConfig tạo một Beans cho WebAppValidator như sau:

```
@Bean
public WebAppValidator productValidator() {
    Set<Validator> springValidators = new HashSet<>();
    springValidators.add(new PriceValidator());

    WebAppValidator validator = new WebAppValidator ();
    validator.setSpringValidators(springValidators);

    return validator;
}
```

Trong ProductController thay thuộc tính priceValidator bằng productValidator:

```
@Controller
public class ProductController {
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private ProductService productService;
    @Autowired
    private WebAppValidator productValidator;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.setValidator(productValidator);
    }
    // ...
}
```

Chạy thử lại chương trình

1.35 SPRING SECURITY

Spring Security quan tâm đến các đối tượng HttpServletRequest và HttpServletResponse, một request có thể thực hiện thông qua trình duyệt Web, Web Service, HTTP client hoặc thực hiện bằng Ajax. Spring Security cung cấp các Servlet Filter xây dựng sẵn và chỉ cần cấu hình các filter thích hợp cho ứng dụng Web để kiểm tra các HTTP request trước khi thực hiện công việc nào đó.

Thêm dependency vào pom.xml

```
<dependency>
    <groupId>org.springframework.security</groupId>
```

```

<artifactId>spring-security-web</artifactId>
<version>5.2.1.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.2.1.RELEASE</version>
</dependency>

```

Tạo lớp com.dht.saleweb.model.User

```

package com.dht.saleweb.model;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

@Entity
@Table(name = "user")
public class User implements Serializable {
    private static final long serialVersionUID = 3L;
    public static final String USER = "ROLE_USER";
    public static final String ADMIN = "ROLE_ADMIN";

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)

```

```
private int id;

@Column(name = "first_name")
@Size(min = 1, max=45, message = "{user.firstName.error.sizeMsg}")
private String firstName;

@Column(name = "last_name")
@Size(min = 1, max = 45, message = "{user.lastName.error.sizeMsg}")
private String lastName;

@Column(name = "email")
@Pattern(regexp = "^[A-Za-z0-9+_.-]+@[.]+\\.$",
          message = "{user.email.error.invalidMsg}")
private String email;

@Column(name = "phone")
@Pattern(regexp = "\\\d{10}",
          message = "{user.phone.error.invalidMsg}")
private String phone;

@Column(name = "active")
private boolean active;

@Column(name = "user_role")
private String userRole;

@Column(name = "username")
@Size(min = 1, max = 45, message = "{user.username.error.sizeMsg}")
private String username;

@Column(name = "password")
@NotEmpty(message = "{user.password.error.sizeMsg}")
```

```

private String password;

@Transient
private String confirmPassword;

{
    active = true;
    userRole = USER;
}

// Các phương thức getter/setter
}

```

Tạo PassValidator trong com.dht.saleweb.validator

```

package com.dht.saleweb.validator;

import com.dht.saleweb.model.User;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component
public class PassValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return User.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        User u = (User) target;
        if (!u.getPassword().trim().equals(
                u.getConfirmPassword().trim()))
            errors.rejectValue("password",
                "user.password.error.notMatchMsg");
    }
}

```

}

Sửa phương thức trong supports() lớp com.dht.saleweb.validator.WebAppValidator như sau:

```
@Override  
public boolean supports(Class<?> clazz) {  
    return Product.class.isAssignableFrom(clazz) ||  
        User.class.isAssignableFrom(clazz);  
}
```

Khai báo thêm Bean userValidator trong WebApplicationContextConfig như sau:

```
@Bean  
public WebAppValidator userValidator() {  
    Set<Validator> springValidators = new HashSet<>();  
    springValidators.add(new PassValidator());  
  
    WebAppValidator validator = new WebAppValidator();  
    validator.setSpringValidators(springValidators);  
  
    return validator;  
}
```

Bổ sung các khai báo trong messages.properties

```
user.firstName=Tên  
user.lastName=Họ  
user.email=Email  
user.phone=Điện thoại  
user.username=Tên đăng nhập  
user.password=Mật khẩu  
user.confirmPassword=Xác nhận mật khẩu  
user.login=Đăng nhập  
user.register=Đăng ký  
user.firstName.error.sizeMsg=Phải nhập tên và tối đa 45 ký tự  
user.lastName.error.sizeMsg=Phải nhập họ và tối đa 45 ký tự
```

```

user.username.error.sizeMsg=Phải nhập tên đăng nhập
user.password.error.sizeMsg=Phải nhập mật khẩu
user.password.error.notMatchMsg=Các mật khẩu nhập không khớp
user.email.error.invalidMsg=Email không hợp lệ
user.phone.error.invalidMsg=Điện thoại không hợp lệ

```

Tạo trang register.jsp trong WEB-INF/jsp

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@ taglib prefix="form"
           uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<div class="container">
    <h1 class="text-center text-uppercase">
        <spring:message code="user.register" />
    </h1>
    <spring:url value="/register" var="action" />
    <form:form action="${action}" modelAttribute="user" method="post" >
        <form:errors path="*" cssClass="alert alert-danger"
                      element="div" />

        <div class="form-group">
            <label for="firstNameId">
                <spring:message code="user.firstName" />
            </label>
            <form:input path="firstName" id="firstNameId"
                        cssClass="form-control" />
            <form:errors path="firstName" cssClass="text-danger" />
        </div>
        <div class="form-group">
            <label for="lastNameId">
                <spring:message code="user.lastName" />

```

```
</label>

<form:input path="lastName" id="lastNameId"
            cssClass="form-control" />
<form:errors path="lastName" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="emailId">
        <spring:message code="user.email" />
    </label>
    <form:input path="email" id="emailId"
                cssClass="form-control" />
    <form:errors path="email" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="phoneId">
        <spring:message code="user.phone" />
    </label>
    <form:input path="phone" id="phoneId"
                cssClass="form-control" />
    <form:errors path="phone" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="usernameId">
        <spring:message code="user.username" />
    </label>
    <form:input path="username" id="usernameId"
                cssClass="form-control" />
    <form:errors path="username" cssClass="text-danger" />
</div>

<div class="form-group">
    <label for="passwordId">
        <spring:message code="user.password" />
    </label>
```

```

        </label>
        <form:input id="passwordId" path="password"
                    cssClass="form-control" type="password" />
        <form:errors path="password" cssClass="text-danger" />
    </div>
    <div class="form-group">
        <label for="confirmPasswordId">
            <spring:message code="user.confirmPassword" />
        </label>
        <form:input id="confirmPasswordId" path="confirmPassword"
                    cssClass="form-control" type="password" />
        <form:errors path="confirmPassword"
                    cssClass="text-danger" />
    </div>
    <div class="form-group">
        <form:button class="pull-right">
            <spring:message code="user.register" />
        </form:button>
    </div>
</form:form>
</div>

```

Tạo giao diện com.dht.saleweb.repository.UserRepository.

```

public interface UserRepository {
    void addUser(User user);
    List<User> getUsers(String username);
}

```

Tạo lớp com.dht.saleweb.repository.impl.UserRepositoryImpl hiện thực giao diện

```

package com.dht.saleweb.repository.impl;

import com.dht.saleweb.model.User;
import com.dht.saleweb.repository.UserRepository;
import java.util.List;

```

```
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserRepositoryImpl implements UserRepository {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void addUser(User user) {
        sessionFactory.getCurrentSession().save(user);
    }

    @Override
    public List<User> getUsers(String username) {
        List<User> users;
        Session session = sessionFactory.getCurrentSession();
        CriteriaBuilder builder = session.getCriteriaBuilder();
        CriteriaQuery<User> cr = builder.createQuery(User.class);

        Root<User> root = cr.from(User.class);
        CriteriaQuery<User> query = cr.select(root);
        if (!username.isEmpty())
            query.where(builder.equal(root.get("username"), username));
        users = session.createQuery(query).getResultList();
        return users;
    }
}
```

Tương tự tạo giao diện com.dht.saleweb.service.UserService

```
package com.dht.saleweb.service;

import com.dht.saleweb.model.User;
import org.springframework.security.core.userdetails.UserDetailsService;

public interface UserService extends UserDetailsService {
    void addUser(User user);
    User getUserByUsername(String username);
}
```

Lớp com.dht.saleweb.service.impl.UserServiceImpl hiện thực giao diện

```
package com.dht.saleweb.service.impl;

import com.dht.saleweb.model.User;
import com.dht.saleweb.repository.UserRepository;
import com.dht.saleweb.service.UserService;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
```

```
private UserRepository userRepository;
@Autowired
private BCryptPasswordEncoder bCryptPasswordEncoder;

@Override
@Transactional
public void addUser(User user) {
    user.setPassword(
        bCryptPasswordEncoder.encode(user.getPassword()));
    userRepository.addUser(user);
}

@Override
@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username)
    throws UsernameNotFoundException {
    List<User> users = userRepository.getUsers(username);
    if (users.isEmpty())
        throw new UsernameNotFoundException("User không tồn tại!");
    User u = users.get(0);
    Set<GrantedAuthority> authorities = new HashSet<>();
    authorities.add(new SimpleGrantedAuthority(u.getUserRole()));

    return new org.springframework.security.core.userdetails.User(
        u.getUsername(), u.getPassword(), authorities);
}

@Override
@Transactional(readOnly = true)
public User getUserByUsername(String username) {
    return userRepository.getUsers(username).get(0);
}
}
```

Tạo controller com.dht.saleweb.controller.LoginController như sau:

```
package com.dht.saleweb.controller;

import com.dht.saleweb.model.User;
import com.dht.saleweb.service.UserService;
import com.dht.saleweb.validator.WebAppValidator;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class LoginController {

    @Autowired
    private UserService userService;
    @Autowired
    private WebAppValidator userValidator;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.setValidator(userValidator);
    }
    @GetMapping(value = "/register")
    public String registerView(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }
    @PostMapping(value = "/register")
```

```
public String registerProcess(  
    @ModelAttribute(name = "user") @Valid User user,  
    BindingResult result) {  
    if (result.hasErrors())  
        return "register";  
    userService.addUser(user);  
    return "redirect:/login";  
}  
}
```

Chạy thử chương trình

The screenshot shows a web browser window with the title 'Đăng ký' and the URL 'localhost:8080/saleweb/register'. The page displays a registration form with the following fields:

- Tên: Thanh
- Họ: Duong
- Email: dhthanhqa@gmail.com
- Điện thoại: 0984461461
- Tên đăng nhập: thanhduong
- Mật khẩu: (redacted)
- Xác nhận mật khẩu: (redacted)

On the right side of the form, there is a 'Giỏ hàng' button with a shopping cart icon. At the bottom right of the form area is a blue 'ĐĂNG KÝ' button. The footer of the page contains the text 'Spring Web MVC © 2020.'

Tạo lớp SpringSecurityConfig.java trong gói com.dht.config

```
package com.dht.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWeb
Security;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.config.annotation.web.configuration.WebSecuri
tyConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;

@EnableWebSecurity
@ComponentScan(basePackages = "com.dht")
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
        auth.userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder());
    }
}
```

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.formLogin().loginPage("/login")
        .usernameParameter("username")
        .passwordParameter("password");
    http.formLogin().defaultSuccessUrl("/")
        .failureUrl("/login?error");
    http.logout().logoutSuccessUrl("/login");
    http.exceptionHandling()
        .accessDeniedPage("/login?accessDenied");
    http.authorizeRequests().antMatchers("/").permitAll()
        .antMatchers("/**/add").access("hasRole('ROLE_ADMIN')");
        .antMatchers("/**/pay")
        .access("hasAnyRole('ROLE_USER', 'ROLE_ADMIN')");
    http.csrf().disable();
}
}

```

Sửa phương thức getRootConfigClasses() trong DispatcherServletInitializer.java

```

@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] {
        HibernateConfig.class, TilesConfig.class,
        SpringSecurityConfig.class
    };
}

```

Tạo lớp SecurityWebApplicationInitializer.java trong gói com.dht.config

```

package com.dht.config;

import org.springframework.security.web.context
    .AbstractSecurityWebApplicationInitializer;

```

```
public class SecurityWebApplicationInitializer
    extends AbstractSecurityWebApplicationInitializer {
}
```

Bổ sung phương thức login trong LoginController như sau:

```
@GetMapping(value = "/login")
public String loginView(Model model) {
    model.addAttribute("user", new User());
    return "login";
}
```

Tạo trang login.jsp trong WEB-INF/jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<div class="container">
    <h1 class="text-center text-uppercase">
        <spring:message code="user.login" />
    </h1>
    <c:if test="${param.error != null}">
        <div class="alert alert-danger">
            <p><spring:message code="user.login.error1" /></p>
        </div>
    </c:if>
    <c:if test="${param.accessDenied != null}">
        <div class="alert alert-danger">
            <p><spring:message code="user.login.error2" /></p>
        </div>
    </c:if>
    <spring:url value="/login" var="action" />
    <form action="${action}" method="post" >
```

```

<div class="form-group">
    <label for="usernameId">
        <spring:message code="user.username" />
    </label>
    <input name="username" id="usernameId"
           class="form-control" />
</div>
<div class="form-group">
    <label for="passwordId">
        <spring:message code="user.password" />
    </label>
    <input id="passwordId" name="password"
           class="form-control" type="password" />
</div>
<div class="form-group">
    <button class="pull-right" type="submit">
        <spring:message code="user.login" />
    </button>
</div>
</form>
</div>

```

Mở header.jsp sửa hai siêu liên kết của Đăng nhập và Đăng ký như sau:

```

<ul>
    <c:choose>
        <c:when test="#{pageContext.request.userPrincipal.name == null}">
            <li>
                <a href=<spring:url value="/register" />>
                    <i class="fa fa-check"></i>
                    <spring:message code="message.register" />
                </a>
            </li>
            <li>

```

```

<a href="

```

Bổ sung các khai báo trong messages.properties

```

message.welcome=Xin chào
message.login=Đăng nhập
message.register=Đăng ký
message.logout=Đăng xuất
user.login.error1=Username hoặc password không hợp lệ
user.login.error2=Bạn không có quyền truy cập trang này

```

Giả sử ta đã có hai user như sau dưới cơ sở dữ liệu, mật khẩu đều là 123456 (chú ý vai trò admin, tạm thời ta chỉnh trực tiếp dưới cơ sở dữ liệu)

id	first_name	last_name	email	phone	username	password	active	user_role
6	Thanh	Duong	thanh.dh@...	0984461467	dhthanh	\$2a\$10\$...	1	ROLE_ADMIN
7	Thanh	Duong	dhthanhqa...	0984461461	thanhduong	\$2a\$10\$...	1	ROLE_USER
HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

Chạy thử ứng dụng và truy cập vào trang thêm sản phẩm, ta sẽ thấy nó chuyển sang trang đăng nhập. Bây giờ đăng nhập với người dùng thanhduong và truy cập lại vào trang thêm sản phẩm, nó sẽ chuyển về trang login và thông báo không có quyền truy cập.

The screenshot shows a web browser window with the following details:

- Title Bar:** ĐĂNG NHẬP
- Address Bar:** localhost:8080/saleweb/login
- Content Area:**
 - Two input fields:
 - Tên đăng nhập: thanhduong
 - Mật khẩu: *****
 - A blue "ĐĂNG NHẬP" button.
 - A shopping cart icon labeled "Giỏ hàng 0".
 - A footer bar with the text "Spring Web MVC © 2020."

The screenshot shows a web browser window with the following details:

- Title Bar:** Đăng nhập
- Address Bar:** localhost:8080/saleweb/products/add
- User Information:** Xin chào thanhduong (User) and Đăng xuất (Logout).
- Search Bar:** Nhập từ khóa tìm kiếm...
- Cart Icon:** Giỏ hàng (Cart) with 0 items.

The main content area displays the following:

ĐĂNG NHẬP

Bạn không có quyền truy cập trang này

Tên đăng nhập

Mật khẩu

ĐĂNG NHẬP Spring Web MVC © 2020.

Bây giờ đăng xuất và đăng nhập với người dùng dhthanh và truy cập lại vào trang thêm sản phẩm bình thường.

The screenshot shows a web application interface for adding a new product. At the top, there's a header bar with a back button, forward button, refresh button, and a search bar labeled "Nhập từ khoá tìm kiếm...". Below the header, there's a user menu with "Xin chào dhthanh" and "Đăng xuất". On the right side of the header, there's a shopping cart icon labeled "Giỏ hàng" with a value of "0". The main content area has a title "THÊM SẢN PHẨM". Below it, there are several input fields: "Danh mục sản phẩm" (dropdown menu showing "Điện thoại thông minh"), "Tên sản phẩm" (text input), "Giá sản phẩm" (text input), "Nhà sản xuất" (text input), "Mô tả chi tiết sản phẩm" (text input), and "Ảnh sản phẩm" (file input field showing "Choose File No file chosen"). At the bottom right, there's a blue button labeled "THÊM SẢN PHẨM". The footer of the page contains the text "Spring Web MVC © 2020".

1.36 TẠO REST API CHO ỨNG DỤNG VỚI AJAX

REST (REpresentational State Transfer) được đề xuất vào năm 2000 bởi Roy Fielding, nó có ảnh hưởng quan trọng trong phát triển các ứng dụng Web hiện đại. RESTful Web Service là giải pháp thông dụng nhất để xây dựng Web Service trong các ứng dụng Web.

Mọi thứ trong REST được xem là một tài nguyên được xác định bởi URI, URI được sử dụng kết nối client và server để trao đổi tài nguyên theo định dạng HTML, JSON, XML, ... để có thể trao đổi dữ liệu, REST dựa trên các phương thức của giao thức HTTP

như GET, POST, PUT và DELETE. Chẳng hạn chức năng thêm sản phẩm được thực hiện bằng phương thức POST của HTTP request bằng URI

<http://localhost:8080/saleweb/products/add>,

chi tiết thông tin của sản phẩm cần thêm được truyền vào URL dưới dạng XML hoặc JSON.

Thông thường, các ứng dụng Web Services dựa trên REST sẽ trả về dữ liệu theo hai định dạng chính là JSON hoặc XML.

Ví dụ ta sẽ tạo các REST API cho phép người dùng thêm sản phẩm, cũng như cập nhật số lượng sản phẩm trong giỏ hàng. Sau đó ở phía client, ta sử dụng jQuery để gọi thực thi các REST API.

Đầu tiên ta xây dựng trang giỏ hàng, trong HomeController bổ sung phương thức:

```
@GetMapping(value = "/cart")
public String cart(Model model, HttpSession session) {
    if (session.getAttribute("cart") != null) {
        Map<Integer, Cart> carts
            = (Map<Integer, Cart>) session.getAttribute("cart");

        BigDecimal sum = new BigDecimal(0);
        for (Cart c: carts.values())
            sum = sum.add(c.getProduct()
                .getPrice()
                .multiply(new BigDecimal(c.getNum())));
    }

    model.addAttribute("carts", carts.values());
    model.addAttribute("sum", sum);
}

return "cart";
}
```

Tạo trang cart.jsp trong thư mục WEB-INF/jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>

<div class="product-big-title-area">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <div class="product-bit-title text-center">
                    <h2 class="text-uppercase">GIỎ HÀNG</h2>
                </div>
            </div>
        </div>
    </div>
</div>

<div class="container">
    <h2 class="text-center">CÁC SẢN PHẨM ĐÃ CHỌN</h2>
    <table class="table table-bordered">
        <tr>
            <th>Mã sản phẩm</th>
            <th>Tên sản phẩm</th>
            <th>Đơn giá</th>
            <th>Số lượng</th>
        </tr>
        <c:forEach items="${carts}" var="cart">
            <tr>
                <td>${cart.product.id}</td>
                <td>${cart.product.name}</td>
                <td><fmt:formatNumber type="number" maxFractionDigits="3"
                        value="${cart.product.price}" /> VNĐ</td>
                <td>${cart.num}</td>
            </tr>
        </c:forEach>
    </table>
</div>
```

```

</c:forEach>

<tr>
    <td colspan="2" class="text-right">Tổng cộng</td>
    <td colspan="2"><fmt:formatNumber type="number"
        maxFractionDigits="3" value="${sum}" /> VNĐ</td>
</tr>
</table>
<div class="pull-right">
    <input type="button" class="btn btn-danger"
        value=">" />
</div>
</div>

```

Bổ sung khai báo trong messages.properties

```
cart.pay=Thanh toán
```

Chỉnh sửa siêu liên kết của giỏ hàng trong header.jsp như sau:

```

<a href="

```

Tiếp theo ta xây dựng các REST API cho phép người dùng thêm sản phẩm vào giỏ hàng, mỗi khi click vào nút “THÊM VÀO GIỎ” khi hover trên từng sản phẩm. Trong gói com.dht.saleweb.controller tạo controller CartRestController như sau:

```

package com.dht.saleweb.controller;

import com.dht.saleweb.model.Cart;
import com.dht.saleweb.service.ProductService;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpSession;
import javax.websocket.serverPathParam;

```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api/cart")
public class CartRestApiController {
    @Autowired
    private ProductService productService;

    @PostMapping
    @ResponseStatus(value = HttpStatus.CREATED)
    public void addOrUpdate(HttpSession session,
        @RequestParam(value = "productId") int productId,
        @RequestParam(value = "num") int num) {
        Map<Integer, Cart> d;
        if (session.getAttribute("cart") == null)
            d = new HashMap<>();
        else
            d = (Map<Integer, Cart>) session.getAttribute("cart");
        Cart c;
        if (d.containsKey(productId) == true) {
            c = d.get(productId);
            c.setNum(c.getNum() + 1);
        } else {
            c = new Cart(productService.getProductById(productId), num);
        }
        d.put(productId, c);
    }
}
```

```

        session.setAttribute("cart", d);
    }
}

```

Tạo tập tin cart.js trong thư mục webapp/resources/js

```

function addCart(productId) {
    $.ajax({
        url: "/saleweb/api/cart",
        type: "POST",
        data: {
            productId: productId,
            num: 1
        },
        success: function (data) {
            var a = $(".product-count").text();
            a = a === "" ? 0 : parseInt(a);
            $(".product-count").text(a + 1);
        },
        error: function (jqXHR) {
            console.info(jqXHR);
        }
    });
}

```

Mở tập tin WEB-INF/layout/template/base.jsp thêm dòng import sử dụng tập tin Javascript này trước dòng </body>

```
<script src="

```

Trong trang WEB-INF/jsp/index.jsp sửa lại siêu liên kết của nút “THÊM VÀO GIỎ”

```

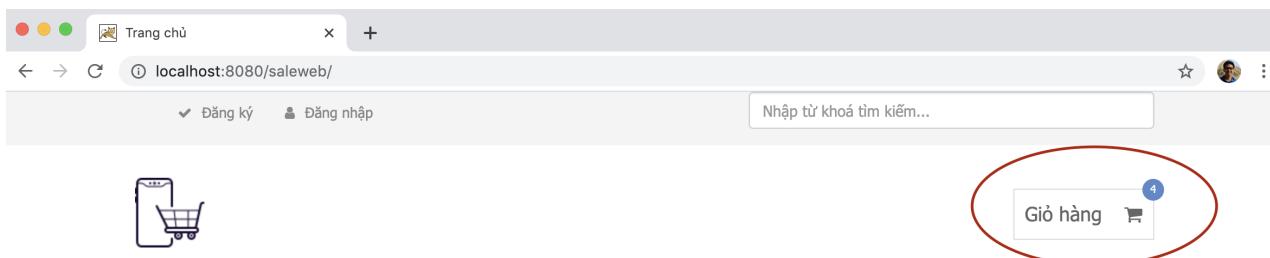
<a href="javascript:;" class="add-to-cart-link"
    onclick="addCart(${product.id})" >
    <i class="fa fa-shopping-cart"></i> Thêm vào giỏ
</a>

```

Tương tự sửa lại siêu liên kết trong trang WEB-INF/jsp/detail.jsp

```
<a href="javascript:;" onclick="addCart(${product.id})"
    class="btn btn-danger">
    <i class="fa fa-shopping-cart"></i> Thêm vào giỏ
</a>
```

Chạy thử chương trình và truy cập vào trang sản phẩm click vào các nút “THÊM VÀO GIỎ” trên các sản phẩm, ta sẽ thấy số lượng trên icon giỏ hàng ở góc trên phải của trang sẽ thay đổi, kết quả giống như sau:



Sau đó bấm vào liên kết này truy cập trang giỏ hàng, kết quả như sau:

Mã sản phẩm	Tên sản phẩm	Đơn giá	Số lượng
1	iPhone 7 Plus	12.500.000 VNĐ	1
3	iPhone 11	17.000.000 VNĐ	1
4	iPhone 11 Pro Max	28.000.000 VNĐ	2
	Tổng cộng	85.500.000 VNĐ	

Thanh toán

Spring Web MVC © 2020.

Tuy nhiên khi chuyển sang trang chi tiết một sản phẩm nào đó thì số lượng sản phẩm đã chọn trên giỏ hàng hiển thị không đúng nữa. Do đó để số lượng sản phẩm đã chọn đều hiển thị đúng trên giỏ hàng khi di chuyển qua lại các trang thì ta bổ sung phương thức sau và gắn thêm annotation `@ControllerAdvice` cho `ProductController`.

```

@Controller
@ControllerAdvice
public class ProductController {
    // ...

    @ModelAttribute
    public void addAttributes(HttpSession session, Model model) {
        int count = 0;
        if (session.getAttribute("cart") != null) {
            Map<Integer, Cart> carts
                = (Map<Integer, Cart>) session.getAttribute("cart");
            for (Cart c: carts.values())
                count += c.getNum();
        }
        model.addAttribute("cartCount", count);
    }
    // ...
}

```

Trong các phần trước, ta đã sử dụng `@ModelAttribute` để kết buộc tham số của phương thức trong controller sử dụng trong các Web View. Trong minh họa này sử dụng `@ModelAttribute` cho phương thức trong controller, các phương thức được gắn annotation này sẽ được gọi trước tất cả các phương thức `RequestMapping` khác, mục đích của ta là thêm các thuộc tính chung vào model trước khi gọi các phương thức `RequestMapping`, trong trường hợp này là thuộc tính `cartCount` để đếm số lượng sản phẩm đã chọn trong giỏ hàng. Một điều quan trọng nữa cần chú ý là lớp tương ứng chưa phương thức `@ModelAttribute` cần gắn annotation `@ControllerAdvice`.

Tiếp theo, Mở WEB-INF/layout/template/header.jsp sửa HTML của giỏ hàng như sau. Sau đó ta chạy lại chương trình theo kịch bản trên sẽ thấy số lượng sản phẩm chọn trong giỏ hàng chuyển giữa trang chủ và trang chi tiết sản phẩm vẫn hiển thị đúng.

```
<div class="shopping-item">
    <a href=">">
        Giỏ hàng <span class="cart-amunt"></span>
        <i class="fa fa-shopping-cart"></i>
        <span class="product-count">${cartCount}</span>
    </a>
</div>
```

Bây giờ ta phát triển chức năng thanh toán, ta tạo các lớp SaleOrder và OrderDetail trong gói com.dht.saleweb.model như sau:

Tập tin SaleOrder.java

```
@Entity
@Table(name = "sale_order")
public class SaleOrder implements Serializable {
    private static final long serialVersionUID = 5L;

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "amount")
    private BigDecimal amount;

    @Column(name = "created_date")
    @Temporal(javax.persistence.TemporalType.DATE)
    private Date createdDate;
```

```

@ManyToOne
@JoinColumn(name = "user_id")
private User user;

{
    createdDate = new Date();
}

public SaleOrder() {

}

public SaleOrder(BigDecimal amount, User user) {
    this.amount = amount;
    this.user = user;
}
// Các phương thức getter/setter
}

```

Tập tin OrderDetail.java

```

@Entity
@Table(name = "order_detail")
public class OrderDetail implements Serializable {
    private static final long serialVersionUID = 6L;

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "order_id")
    private SaleOrder order;
}

```

```
@ManyToOne  
@JoinColumn(name = "product_id")  
private Product product;  
  
@Column(name = "unit_price")  
private BigDecimal unitPrice;  
  
@Column(name = "num")  
private int num;  
  
public OrderDetail() {  
  
}  
  
public OrderDetail(SaleOrder order, Product product, int num) {  
    this.order = order;  
    this.product = product;  
    this.unitPrice = product.getPrice();  
    this.num = num;  
}  
// Các phương thức getter/setter  
}
```

Trong gói com.dht.saleweb.repository tạo các giao diện SaleOrderRepository và OrderDetailRepository.

```
public interface SaleOrderRepository {  
    void addSaleOrder(SaleOrder order);  
}  
public interface OrderDetailRepository {  
    void addOrderDetail(OrderDetail orderDetail);  
}
```

Tạo các lớp hiện thực các giao diện này trong com.dht.saleweb.repository.impl

```

@Repository
public class SaleOrderRepositoryImpl implements SaleOrderRepository {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void addSaleOrder(SaleOrder order) {
        Session session = sessionFactory.getCurrentSession();
        session.save(order);
    }
}

@Repository
public class OrderDetailRepositoryImpl
    implements OrderDetailRepository {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public void addOrderDetail(OrderDetail orderDetail) {
        Session session = sessionFactory.getCurrentSession();
        session.save(orderDetail);
    }
}

```

Tạo giao diện OrderService trong gói com.dht.saleweb.service

```

public interface OrderService {
    void addOrder(Collection<Cart> carts);
}

```

Tạo lớp OrderServiceImpl trong gói com.dht.saleweb.service.impl

```

package com.dht.saleweb.service.impl;

import com.dht.saleweb.model.Cart;
import com.dht.saleweb.model.OrderDetail;

```

```
import com.dht.saleweb.model.SaleOrder;
import com.dht.saleweb.model.User;
import com.dht.saleweb.repository.OrderDetailRepository;
import com.dht.saleweb.repository.SaleOrderRepository;
import com.dht.saleweb.service.OrderService;
import com.dht.saleweb.service.UserService;
import java.math.BigDecimal;
import java.util.Collection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;

@Service
public class OrderServiceImpl implements OrderService {
    @Autowired
    private SaleOrderRepository saleOrderRepository;
    @Autowired
    private OrderDetailRepository orderDetailRepository;
    @Autowired
    private UserService userService;

    @Override
    @Transactional(propagation = Propagation.REQUIRED)
    public void addOrder(Collection<Cart> carts) {
        BigDecimal sum = new BigDecimal(0);
        for (Cart c: carts)
            sum = sum.add(c.getProduct()
                    .getPrice()
                    .multiply(new BigDecimal(c.getNum())));
        String username = SecurityContextHolder.getContext()
```

```

        .getAuthentication().getName();

User user = userService.getUserByUsername(username);

SaleOrder order = new SaleOrder(sum, user);
saleOrderRepository.addSaleOrder(order);

for (Cart c: carts) {
    OrderDetail detail
        = new OrderDetail(order, c.getProduct(), c.getNum());
    orderDetailRepository.addOrderDetail(detail);
}
}
}
}

```

Chú ý khái nát annotation @Transactional(propagation = Propagation.REQUIRED) với thuộc tính propagation chỉ định sự lan truyền giao tác (transaction propagation). Trong các phương thức giao tác có thể sẽ gọi các phương thức khác, khi đó ta cần chỉ định cách thức giao tác được sử dụng giữa các phương thức. Spring hỗ trợ 7 cách thức truyền giao tác. Chú ý không phải tất cả các trình quản lý giao tác đều hỗ trợ các loại lan truyền, mà còn tùy thuộc tài nguyên cơ sở, chẳng hạn các cơ sở dữ liệu có thể hỗ trợ các mức cô lập khác nhau chỉ định cách thức truyền giao tác nào sẽ được hỗ trợ.

- REQUIRED: nếu giao tác đang tồn tại, phương thức sẽ chạy trong giao tác này, ngược lại nó sẽ bắt đầu một giao tác mới và chạy trong giao tác này.
- REQUIRES_NEW: phương thức bắt đầu một giao tác mới và chạy trong giao tác này, nếu có một giao tác khác tồn tại sẽ bị hoãn lại (suspend).
- SUPPORTS: nếu có giao tác tồn tại thì phương thức sẽ chạy trong giao tác này, ngược lại nó không cần hoạt động trong giao tác.
- NOT_SUPPORTED: phương thức không chạy trong giao tác, nếu có giao tác tồn tại sẽ bị hoãn lại (suspend).
- MANDATORY: phương thức bắt buộc chạy trong giao tác, nếu không có giao tác nào tồn tại nó sẽ ném ra ngoại lệ.

- NEVER: phương thức không chạy trong giao tác, nếu giao tác tồn tại nó sẽ ném ra ngoại lệ.
- NESTED: nếu có giao tác tồn tại, phương thức sẽ chạy trong giao tác lồng của nó (nested transaction) (giao tác lồng là giao tác được tạo bên trong giao tác khác), ngược lại nó sẽ bắt đầu một giao tác mới và chạy trong giao tác này.

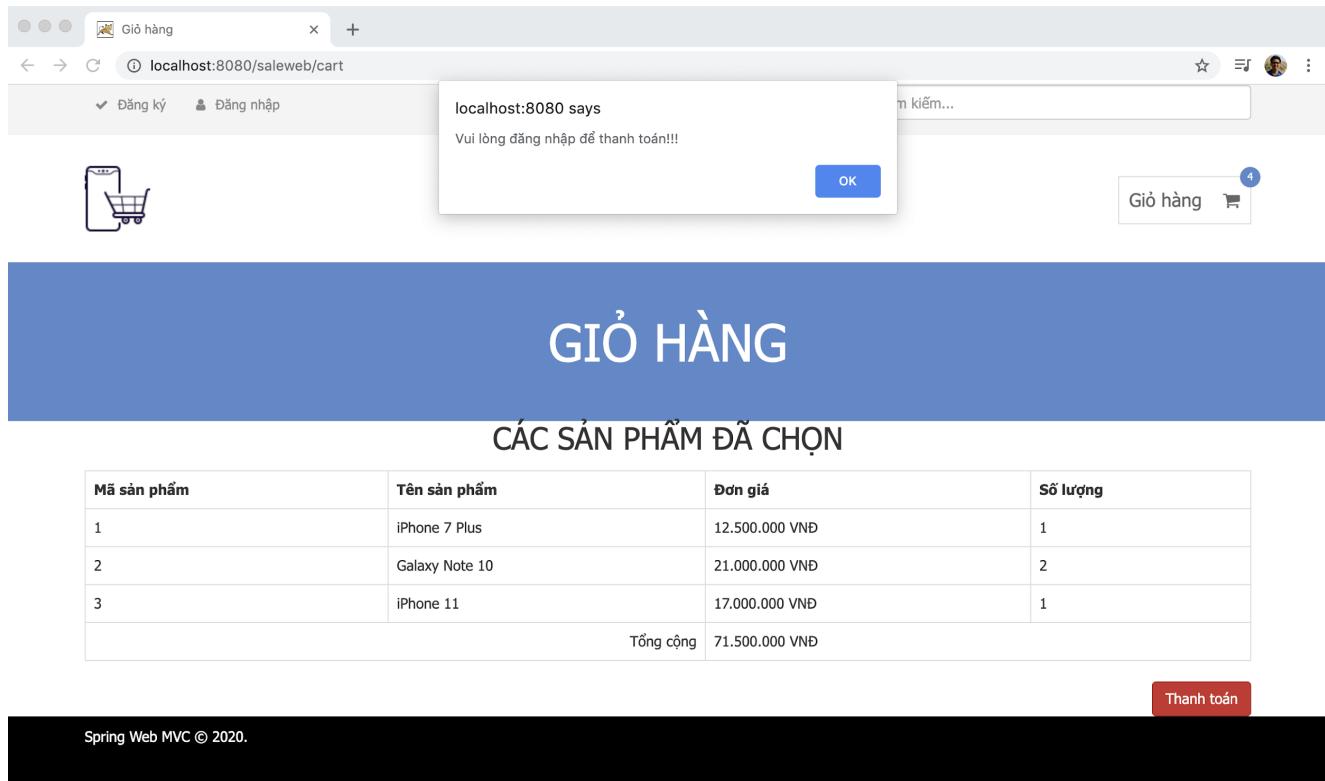
Tiếp theo thêm đoạn Javascript xử lý nút thanh toán cuối trang cart.jsp.

```
<script>
function pay() {
    if (confirm("Bạn muốn thanh toán?")) {
        $.ajax({
            url: "/saleweb/api/cart/pay",
            type: "POST",
            success: function (data, textStatus, jqXHR) {
                if (data === "")
                    $(".selected-products").hide("slow");
                else
                    alert("Vui lòng đăng nhập để thanh toán!!!");
            },
            error: function (jqXHR, textStatus, errorThrown) {
                alert("Đã có lỗi khi thanh toán!!!");
            }
        });
    }
}
</script>
```

Sửa lại HTML của nút “Thanh toán” trong trang cart.jsp như sau:

```
<input type="button" class="btn btn-danger" onclick="pay()"
value=<spring:message code="cart.pay" />" />
```

Chạy thử chương trình thêm vài sản phẩm vào giỏ hàng ở trạng thái chưa đăng nhập, sau đó truy cập vào trang giỏ hàng và click nút “Thanh toán” sẽ nhận thông báo lỗi đăng nhập để thanh toán.



1.37 INTERCEPTORS

Interceptor là một kỹ thuật lập trình Web đặc biệt để thực thi một xử lý nào đó trước và sau xử lý request. Trong Spring MVC, interceptor là các lớp đặc biệt hiện thực giao diện `org.springframework.web.servlet.HandlerInterceptor`, giao diện này có ba phương thức quan trọng:

- `preHandle()`: phương thức này được gọi trước khi request được xử lý trong một phương thức của controller.
- `postHandle()`: phương thức này được gọi sau khi request được xử lý trong một phương thức của controller.

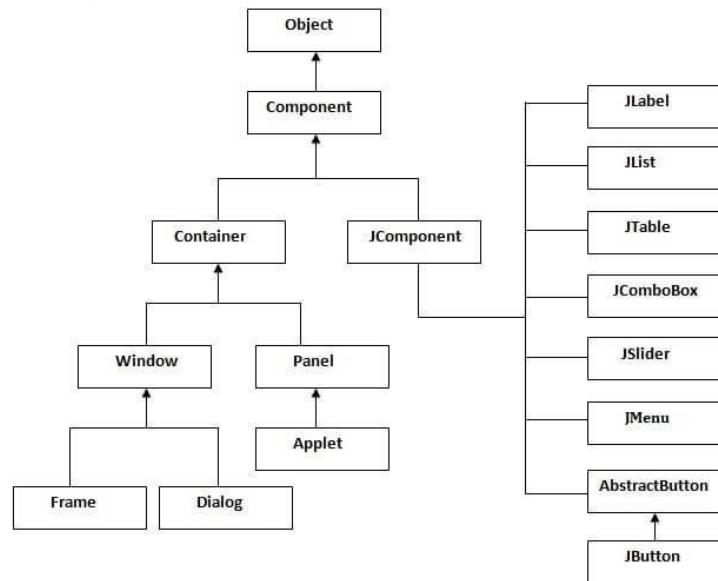
- afterCompletion(): phương thức được gọi sau khi hoàn tất chu trình thực hiện request.

Bài 7. Java Swing

Java Swing là một phần của Java Foundation Classes (JFC) được sử dụng để tạo các ứng dụng window-based. Nó được xây dựng trên API AWT (Abstract Windowing Toolkit) và được viết hoàn toàn bằng Java. Không giống như AWT, Java Swing cung cấp các thành phần không phụ thuộc vào nền tảng và nhẹ hơn. Gói javax.swing cung cấp các lớp cho java swing API như JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser, v...v...

1.38 GIỚI THIỆU JAVA SWING

Hệ thống phân cấp của API java swing được đưa ra dưới đây.



❖ Các phương thức thường dùng của lớp Component:

Các phương thức của lớp Component được sử dụng rộng rãi trong java swing được đưa ra dưới đây.

Phương thức	Mô tả
public void add(Component c)	thêm một thành phần vào thành phần khác.
public void setSize(int width, int height)	thiết lập kích thước của thành phần.
public void setLayout(LayoutManager m)	thiết lập trình quản lý bố cục (layout) cho thành phần.
public void setVisible(boolean b)	thiết lập khả năng hiển thị của thành phần. Nó theo mặc định là false (ẩn)

❖ Ví dụ về Java Swing

Có hai cách để tạo khung (Frame):

- Bằng cách tạo đối tượng của lớp JFrame.
- Bằng cách kế thừa lớp JFrame.

Chúng ta có thể viết code của Swing bên trong hàm main(), constructor hoặc bất kỳ phương thức nào khác.

❖ Ví dụ Swing Java đơn giản

Chúng ta hãy xem một ví dụ swing đơn giản, nơi chúng ta đang tạo một button và thêm nó vào đối tượng JFrame bên trong phương thức main().

File: FirstSwingExample.java

```
package chapter.swing;

import javax.swing.JButton;
import javax.swing.JFrame;

public class FirstSwingExample {

    public static void main(String[] args) {
        JFrame f = new JFrame();// tạo thẻ hiện của JFrame
        JButton b = new JButton("click");// tạo thẻ hiện của JButton
    }
}
```

```
b.setBounds(130, 50, 100, 40); // trục x , y , width, height
```

```
f.setTitle("Ví dụ Java Swing");
```

```
f.add(b); // thêm button vào JFrame
```

```
f.setSize(400, 200); // thiết lập kích thước cho cửa sổ
```

```
f.setLayout(null); // không sử dụng trình quản lý bố cục
```

```
f.setVisible(true); // hiển thị cửa sổ
```

```
}
```

```
}
```

Kết quả:



❖ Ví dụ Java Swing - tạo đối tượng của lớp JFrame

Chúng ta cũng có thể viết tất cả các mã tạo JFrame, JButton bên trong constructor.

```
package chapter.swing;

import javax.swing.JButton;
import javax.swing.JFrame;
public class JavaSwingExample2 {
```

```
    JFrame f;
```

```
    public JavaSwingExample2() {
```

```
        f = new JFrame(); // tạo thê hiện của JFrame
```

```
 JButton b = new JButton("click");// tạo thểm hiện của JButton  
 b.setBounds(130, 50, 100, 40);  
 f.add(b);// thêm button vào JFrame  
  
 f.setSize(400, 200);// thiết lập kích thước cho cửa sổ  
 f.setLayout(null);// không sử dụng trình quản lý bố cục  
 f.setVisible(true);// hiển thị cửa sổ  
}  
}
```

❖ **Ví dụ Java Swing - kế thừa lớp JFrame**

Chúng ta cũng có thể kế thừa lớp JFrame, vì vậy không cần phải tạo thểm hiện của lớp JFrame.

```
 package chapter.swing;  
  
 import javax.swing.JButton;  
  
 import javax.swing.JFrame;  
  
 public class JavaSwingExample3 extends JFrame {// kế thừa lớp JFrame  
 public JavaSwingExample3() {  
 JButton b = new JButton("click");// tạo button  
 b.setBounds(130, 50, 100, 40);  
 add(b);// thêm button vào JFrame  
 setSize(400, 200);  
 setLayout(null);  
 setVisible(true);
```

```

    }

    public static void main(String[] args) {
        new JavaSwingExample3();
    }
}

```

1.39 Một số COMPONENT TRONG SWING

❖ Lớp JLabel trong Java Swing

Lớp JLabel trong Java Swing có thể hiển thị text, hoặc hình ảnh hoặc cả hai. Các nội dung của Label được gán bởi thiết lập căn chỉnh ngang và dọc trong khu vực hiển thị của nó. Theo mặc định, các label được căn chỉnh theo chiều dọc trong khu vực hiển thị. Theo mặc định, text-only label là căn chỉnh theo cạnh, image-only label là căn chỉnh theo chiều ngang.

- Constructor của lớp JLabel trong Java Swing**

- JLabel(): Tạo một instance của JLabel, không có hình ảnh, và với một chuỗi trống cho title.
- JLabel(Icon image): Tạo một instance của JLabel với hình ảnh đã cho.
- JLabel(Icon image, int horizontalAlignment): Tạo một instance của JLabel với hình ảnh và căn chỉnh ngang đã cho.
- JLabel(String text): Tạo một instance của JLabel với text đã cho.
- JLabel(String text, Icon icon, int horizontalAlignment): Tạo một instance của JLabel với text, hình ảnh, và căn chỉnh ngang đã cho.
- JLabel(String text, int horizontalAlignment): Tạo một instance của JLabel với text và căn chỉnh ngang đã cho.

- Một số phương thức thường được sử dụng của lớp JLabel:**

STT	Phương thức & Mô tả
1	void setDisabledIcon(Icon disabledIcon) Thiết lập icon để được hiển thị nếu JLabel này là "disabled" (JLabel.setEnabled(false))

2	void setDisplayedMnemonic(char aChar) Xác định displayedMnemonic như là một giá trị char
3	void setDisplayedMnemonic(int key) Xác định keycode mà chỉ dẫn một mnemonic key
4	void setDisplayedMnemonicIndex(int index) Cung cấp một hint cho L&F, từ đó ký tự trong text nên được trang trí để biểu diễn mnemonic
5	void setHorizontalAlignment(int alignment) Thiết lập sự căn chỉnh nội dung của label theo trục X
6	void setHorizontalTextPosition(int textPosition) Thiết lập vị trí theo chiều ngang của phần text của label, cân xứng với hình ảnh của nó
7	void setIcon(Icon icon) Định nghĩa icon mà thành phần này sẽ hiển thị
8	void setIconTextGap(int iconTextGap) Nếu cả hai thuộc tính icon và text được thiết lập, thuộc tính này xác định khoảng trống giữa chúng
9	void setLabelFor(Component c) Thiết lập thành phần đang gán nhãn cho

10	void setText(String text) Định nghĩa trường text dòng đơn mà thành phần này sẽ hiển thị
11	void setUI(LabelUI ui) Thiết lập đối tượng L&F mà biểu diễn thành phần này
12	void setVerticalAlignment(int alignment) Thiết lập sự căn chỉnh nội dung của label theo trục Y
13	void setVerticalTextPosition(int textPosition) Thiết lập vị trí theo chiều dọc của phần text của label, cân xứng với hình ảnh của nó
14	void updateUI() Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại

- **Chương trình ví dụ lớp JLabel**

SwingJlabelExam1.java

```
package chapter.swing;
```

```
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
import javax.swing.JFrame;  
  
import javax.swing.JLabel;  
  
import javax.swing.JPanel;  
  
  
public class SwingJlabelExam1 {  
  
    private JFrame mainFrame;  
  
    private JLabel headerLabel;  
  
    private JLabel statusLabel;  
  
    private JPanel controlPanel;  
  
    public SwingJlabelExam1() {  
  
        prepareGUI();  
  
    }  
  
    public static void main(String[] args) {  
  
        SwingJlabelExam1 swingJlabelExam1 = new SwingJlabelExam1();  
  
        swingJlabelExam1.showLabelDemo();  
  
    }  
  
    private void prepareGUI() {  
  
        mainFrame = new JFrame("Vi du Java Swing");  
  
        mainFrame.setSize(400, 300);  
  
        mainFrame.setLayout(new GridLayout(3, 1));  
  
        mainFrame.addWindowListener(new WindowAdapter() {  
  
            public void windowClosing(WindowEvent windowEvent) {  
  
                System.exit(0);  
            }  
        });  
    }  
}
```

```
    }

});

headerLabel = new JLabel("", JLabel.CENTER);

statusLabel = new JLabel("", JLabel.CENTER);

statusLabel.setSize(350, 100);

controlPanel = new JPanel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showLabelDemo() {

    headerLabel.setText("Control in action: JLabel");

    JLabel label = new JLabel("", JLabel.CENTER);

    label.setText("Chao mung ban den voi bai huong dan Java Swing.");

    label.setOpaque(true);

    label.setBackground(Color.GRAY);

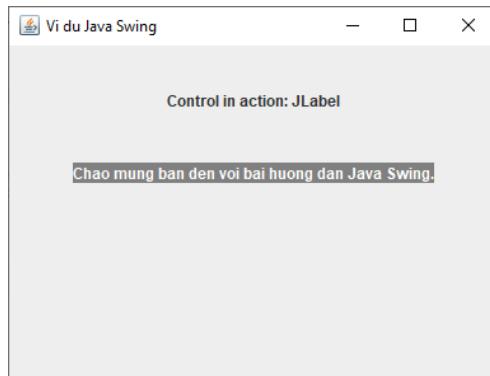
    label.setForeground(Color.WHITE);

    controlPanel.add(label);

    mainFrame.setVisible(true);

}
```

Chạy chương trình Java trên cho kết quả như sau:



❖ Lớp JButton trong Java Swing

Lớp JButton trong Java Swing được sử dụng để tạo một nút có thể click. Thành phần này có một label và tạo một sự kiện (event) khi được nhấn. Bạn cũng có thể chèn ảnh vào button.

- **Lớp JButton có các constructor sau:**

- JButton(): Tạo một button mà không thiết lập text hoặc icon.
- JButton(Action a): Tạo một button tại đây các thuộc tính được nhận từ Action đã cung cấp.
- JButton(Icon icon): Tạo một button với một icon.
- JButton(String text): Tạo một button với text.
- JButton(String text, Icon icon): Tạo một button với text ban đầu và một icon.

- **Các phương thức được sử dụng phổ biến của lớp JButton**

- void removeNotify(): Ghi đè JComponent.removeNotify để kiểm tra xem button này hiện tại được thiết lập như là button mặc định trên RootPane hay không, và nếu có, thiết lập button mặc định của RootPane về null để bảo đảm rằng Rootpane không giữ một tham chiếu button không hợp lệ.
- void setDefaultCapable(boolean defaultCapable): Thiết lập thuộc tính defaultCapable, mà quyết định xem có hay không button này có thể được tạo là button mặc định cho Root Pane của nó
- void updateUI(): Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại.

- **Một số phương thức được sử dụng phổ biến của lớp AbstractButton**

- public void setText(String s): được sử dụng để thiết lập text đã cho trên button.

- public String getText(): được sử dụng để trả về text của button.
- public void setEnabled(boolean b): được sử dụng để kích hoạt hoặc vô hiệu hóa button.
- public void setIcon(Icon b): được sử dụng để thiết lập Icon đã cho trên button.
- public Icon getIcon(): được sử dụng để lấy Icon của button.
- public void setMnemonic(int a): được sử dụng để thiết lập thuộc tính mnemonic trên button.
- public void addActionListener(ActionListener a): được sử dụng để thêm action listener tới đối tượng này.

- **Chương trình đơn giản để hiển thị hình ảnh trên button**

SwingJButtonExam1.java

```
package chapter.swing;
```

```
import javax.swing.ImageIcon;
```

```
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
```

```
public class SwingJButtonExam1 {
```

```
    SwingJButtonExam1() {
```

```
        JFrame f = new JFrame();
```

```
        JButton b = new JButton(new ImageIcon("b.jpg"));
```

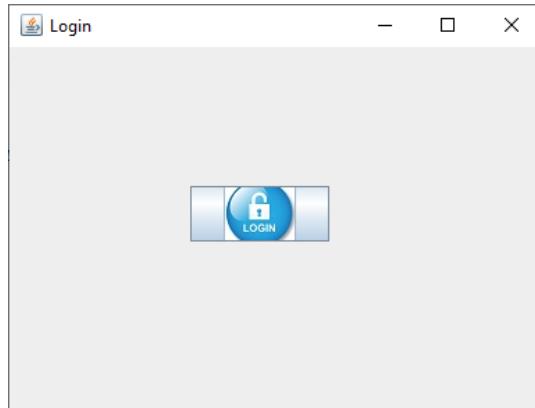
```
        b.setBounds(130, 100, 100, 40);
```

```
        f.add(b);
```

```
        f.setSize(400, 300);
```

```
f.setLayout(null);  
f.setVisible(true);  
  
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
  
public static void main(String[] args) {  
    new SwingJButtonExam1();  
}  
}
```

Chạy chương trình Java trên cho kết quả như sau:



- **Chương trình ví dụ khác về lớp JButton**

SwingJButtonExam2.java

```
package chapter.swing;
```

```
import java.awt.FlowLayout;  
  
import java.awt.GridLayout;  
  
import java.awt.event.ActionEvent;  
  
import java.awt.event.ActionListener;  
  
import java.awt.event.WindowAdapter;  
  
import java.awt.event.WindowEvent;  
  
  
import javax.swing.JButton;  
  
import javax.swing.JFrame;  
  
import javax.swing.JLabel;  
  
import javax.swing.JPanel;  
  
import javax.swing.SwingConstants;  
  
  
public class SwingJButtonExam2 {  
  
    private JFrame mainFrame;  
  
    private JLabel headerLabel;  
  
    private JLabel statusBar;  
  
    private JPanel controlPanel;  
  
  
    public SwingJButtonExam2() {  
  
        prepareGUI();  
  
    }  
  
  
    public static void main(String[] args) {
```

```
SwingJButtonExam2 swingJButtonExam2 = new SwingJButtonExam2();

swingJButtonExam2.showButtonDemo();

}

private void prepareGUI() {

    mainFrame = new JFrame("Vi du Java Swing - JButton");

    mainFrame.setSize(400, 300);

    mainFrame.setLayout(new GridLayout(3, 1));

    mainFrame.addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent windowEvent) {

            System.exit(0);

        }

    });

    headerLabel = new JLabel("", JLabel.CENTER);

    statusLabel = new JLabel("", JLabel.CENTER);

    statusLabel.setSize(350, 100);

    controlPanel = new JPanel();

    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);

    mainFrame.add(controlPanel);

    mainFrame.add(statusLabel);

    mainFrame.setVisible(true);

}
```

```
private void showButtonDemo() {  
    headerLabel.setText("Control in action: Button");  
  
    JButton okButton = new JButton("OK");  
  
    JButton javaButton = new JButton("Submit");  
  
    JButton cancelButton = new JButton("Cancel");  
  
    cancelButton.setHorizontalTextPosition(SwingConstants.LEFT);  
  
    okButton.addActionListener(new ActionListener() {  
  
        public void actionPerformed(ActionEvent e) {  
  
            statusLabel.setText("Ok Button clicked.");  
  
        }  
  
    });  
  
    javaButton.addActionListener(new ActionListener() {  
  
        public void actionPerformed(ActionEvent e) {  
  
            statusLabel.setText("Submit Button clicked.");  
  
        }  
  
    });  
  
    cancelButton.addActionListener(new ActionListener() {  
  
        public void actionPerformed(ActionEvent e) {  
  
            statusLabel.setText("Cancel Button clicked.");  
  
        }  
  
    });  
  
    controlPanel.add(okButton);  
  
    controlPanel.add(javaButton);  
}
```

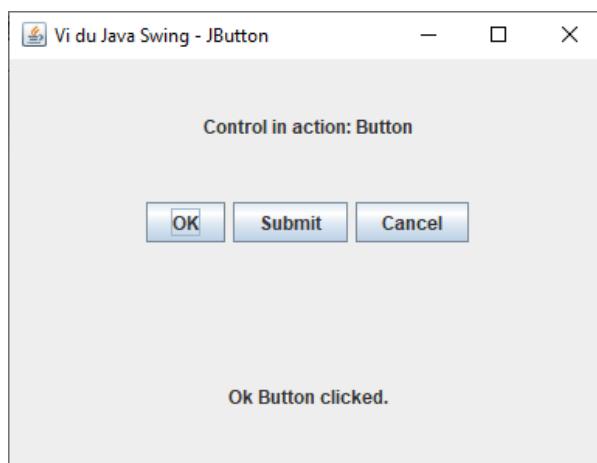
```
controlPanel.add(cancelButton);

mainFrame.setVisible(true);

}

}
```

Chạy chương trình Java trên cho kết quả như sau:



❖ Lớp JCheckBox trong Java Swing

Lớp JCheckBox trong Java Swing là một trình triển khai của một checkbox, là một item mà có thể được lựa chọn (selected) hoặc không được lựa chọn (unselected), và hiển thị trạng thái

• Các constructor của lớp JCheckBox

- JCheckBox(): Tạo một unselected checkbox ban đầu không có text và icon.
- JCheckBox(Action a): Tạo một checkbox, với các thuộc tính được lấy từ Action đã cho.
- JCheckBox(Icon icon): Tạo một unselected checkbox với một icon.
- JCheckBox(Icon icon, boolean selected): Tạo một checkbox với một icon và xác định rằng ban đầu nó là selected hoặc không .
- JCheckBox(String text): Tạo một unselected checkbox ban đầu với text.
- JCheckBox(String text, boolean selected): Tạo một checkbox với text và xác định rằng ban đầu nó là selected hoặc không.
- JCheckBox(String text, Icon icon): Tạo một unselected checkbox ban đầu với text và icon đã cho.

- JCheckBox(String text, Icon icon, boolean selected): Tạo một checkbox với text và icon, và xác định rằng ban đầu nó là selected hoặc không.
- **Các phương thức của lớp JCheckBox trong Java Swing**
 - AccessibleContext getAccessibleContext(): Lấy AccessibleContext được liên kết với JCheckBox này.
 - String getUIClassID(): Trả về một chuỗi xác định tên của lớp L&F mà truyền thành phần này.
 - boolean isBorderPaintedFlat(): Lấy giá trị của thuộc tính borderPaintedFlat.
 - protected String paramString(): Trả về một biểu diễn chuỗi của JCheckBox này.
 - void setBorderPaintedFlat(boolean b): Thiết lập thuộc tính borderPaintedFlat, mà cung cấp một hint tới L&F tới bề mặt của đường viền của checkbox.
 - void updateUI(): Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại.

- **Chương trình ví dụ về lớp JCheckBox**

JCheckBoxExam1.java

```
package chapter.swing;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.awt.event.KeyEvent;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import javax.swing.JCheckBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class JCheckBoxExam1 {

    private JFrame mainFrame;
```

```
private JLabel headerLabel;  
  
private JLabel statusLabel;  
  
private JPanel controlPanel;  
  
public JCheckBoxExam1() {  
  
    prepareGUI();  
  
}  
  
public static void main(String[] args) {  
  
    JCheckBoxExam1 swingControlDemo = new JCheckBoxExam1();  
  
    swingControlDemo.showCheckBoxDemo();  
  
}  
  
private void prepareGUI() {  
  
    mainFrame = new JFrame("Vi du JCheckBox trong Java Swing");  
  
    mainFrame.setSize(400, 300);  
  
    mainFrame.setLayout(new GridLayout(3, 1));  
  
    mainFrame.addWindowListener(new WindowAdapter() {  
  
        public void windowClosing(WindowEvent windowEvent) {  
  
            System.exit(0);  
  
        }  
  
    });  
  
    headerLabel = new JLabel("", JLabel.CENTER);  
  
    statusLabel = new JLabel("", JLabel.CENTER);  
  
    statusLabel.setSize(350, 100);  
  
    controlPanel = new JPanel();
```

```
controlPanel.setLayout(new FlowLayout());  
  
mainFrame.add(headerLabel);  
  
mainFrame.add(controlPanel);  
  
mainFrame.add(statusLabel);  
  
mainFrame.setVisible(true);  
  
}  
  
private void showCheckBoxDemo() {  
  
    headerLabel.setText("Control in action: CheckBox");  
  
    final JCheckBox chkApple = new JCheckBox("Green");  
  
    final JCheckBox chkMango = new JCheckBox("Red");  
  
    final JCheckBox chkPeer = new JCheckBox("Yellow");  
  
    chkApple.setMnemonic(KeyEvent.VK_C);  
  
    chkMango.setMnemonic(KeyEvent.VK_M);  
  
    chkPeer.setMnemonic(KeyEvent.VK_P);  
  
    chkApple.addItemListener(new ItemListener() {  
  
        public void itemStateChanged(ItemEvent e) {  
  
            statusLabel.setText("Green Checkbox: "  
                + (e.getStateChange() == 1 ? "checked" : "unchecked"));  
  
        }  
  
    });  
  
    chkMango.addItemListener(new ItemListener() {  
  
        public void itemStateChanged(ItemEvent e) {  
  
            statusLabel.setText("Red Checkbox: "  
                + (e.getStateChange() == 1 ? "checked" : "unchecked"));  
  
        }  
  
    });  
}
```

```
    }

});

chkPeer.addItemListener(new ItemListener() {

    public void itemStateChanged(ItemEvent e) {

        statusLabel.setText("Yellow Checkbox: "

            + (e.getStateChange() == 1 ? "checked" : "unchecked"));

    }

});

controlPanel.add(chkApple);

controlPanel.add(chkMango);

controlPanel.add(chkPeer);

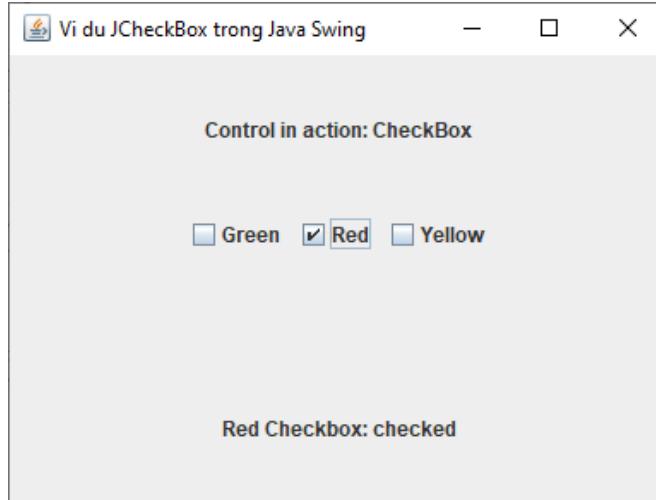
mainFrame.setVisible(true);

}

}

}

Chạy chương trình Java trên cho kết quả như sau:
```



❖ Lớp JRadioButton trong Java Swing

Lớp JRadioButton trong Java Swing là một trình triển khai của một radio button, một item mà có thể được lựa chọn hoặc không, và hiển thị trạng thái của nó tới người dùng. Lớp này nên được thêm vào trong ButtonGroup để chỉ lựa chọn một radio button.

- **Các constructor được sử dụng phổ biến của lớp JRadioButton:**

- JRadioButton(): Tạo một unselected radiobutton không có text.
- JRadioButton(String s): Tạo một unselected radiobutton với text đã cho.
- JRadioButton(String s, boolean selected): Tạo một radiobutton với text đã cho và trạng thái là selected.

- **Các phương thức được sử dụng phổ biến của lớp JRadioButton**

- AccessibleContext getAccessibleContext(): Lấy AccessibleContext được liên kết với JRadioButton này.
- String getUIClassID(): Trả về tên của lớp L&F mà truyền thành phần này.
- protected String paramString(): Trả về biểu diễn chuỗi của JRadioButton này.
- void updateUI(): Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại.

- **Chương trình ví dụ về lớp JRadioButton**

JRadioButtonExam1.java

```
package chapter.swing;
```

```
import javax.swing.ButtonGroup;
```

```
import javax.swing.JFrame;
```

```
import javax.swing.JRadioButton;
```

```
public class JRadioButtonExam1 {
```

```
    JFrame frame;
```

```
    JRadioButtonExam1() {
```

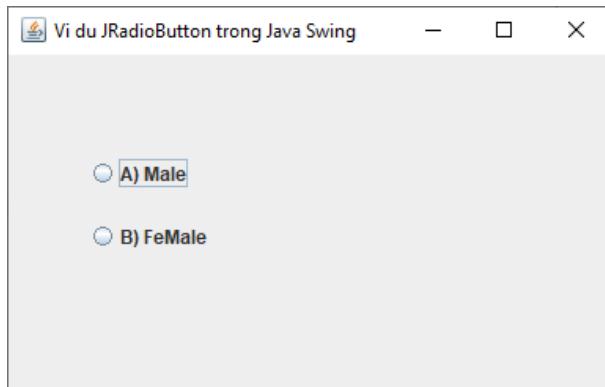
```
        frame = new JFrame();
```

```
JRadioButton radioBtn1 = new JRadioButton("A) Male");
JRadioButton radioBtn2 = new JRadioButton("B) FeMale");
radioBtn1.setBounds(50, 60, 170, 30);
radioBtn2.setBounds(50, 100, 170, 30);

ButtonGroup bg = new ButtonGroup();
bg.add(radioBtn1);
bg.add(radioBtn2);
frame.add(radioBtn1);
frame.add(radioBtn2);
frame.setTitle("Vi du JRadioButton trong Java Swing");
frame.setSize(400, 250);
frame.setLayout(null);
frame.setVisible(true);
}

public static void main(String[] args) {
    new JRadioButtonExam1();
}
}
```

Kết quả:



- **Chương trình ví dụ khác về lớp JRadioButton**

JRadioButtonExam2.java

```
package chapter.swing;

import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
```

```
public class JRadioButtonExam2 {  
  
    private JFrame mainFrame;  
  
    private JLabel headerLabel;  
  
    private JLabel statusLabel;  
  
    private JPanel controlPanel;  
  
  
    public JRadioButtonExam2() {  
        prepareGUI();  
    }  
  
  
    public static void main(String[] args) {  
        JRadioButtonExam2 swingControlDemo = new JRadioButtonExam2();  
        swingControlDemo.showRadioButtonDemo();  
    }  
  
  
    private void prepareGUI() {  
        mainFrame = new JFrame("Vi du JRadioButton trong Java Swing");  
        mainFrame.setSize(400, 250);  
        mainFrame.setLayout(new GridLayout(3, 1));  
        mainFrame.addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent windowEvent) {  
                System.out.println("Window is closing...");  
            }  
        });  
        mainFrame.setVisible(true);  
    }  
}
```

```
        System.exit(0);

    }

});

headerLabel = new JLabel("", JLabel.CENTER);

statusLabel = new JLabel("", JLabel.CENTER);

statusLabel.setSize(350, 100);

controlPanel = new JPanel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}
```

```
private void showRadioButtonDemo() {

    headerLabel.setText("Control in action: RadioButton");

    final JRadioButton radApple = new JRadioButton("Green", true);

    final JRadioButton radMango = new JRadioButton("Red");

    final JRadioButton radPeer = new JRadioButton("Yellow");

    radApple.setMnemonic(KeyEvent.VK_C);

    radMango.setMnemonic(KeyEvent.VK_M);

    radPeer.setMnemonic(KeyEvent.VK_P);

    radApple.addItemListener(new ItemListener() {

        public void itemStateChanged(ItemEvent e) {
```

```
statusLabel.setText("Green RadioButton: "
+ (e.getStateChange() == 1 ? "checked" : "unchecked"));
}

});

radMango.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        statusLabel.setText("Red RadioButton: "
+ (e.getStateChange() == 1 ? "checked" : "unchecked"));
    }
});

radPeer.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        statusLabel.setText("Yellow RadioButton: "
+ (e.getStateChange() == 1 ? "checked" : "unchecked"));
    }
}); // Group the radio buttons.

ButtonGroup group = new ButtonGroup();
group.add(radApple);
group.add(radMango);
group.add(radPeer);
controlPanel.add(radApple);
controlPanel.add(radMango);
controlPanel.add(radPeer);
```

```

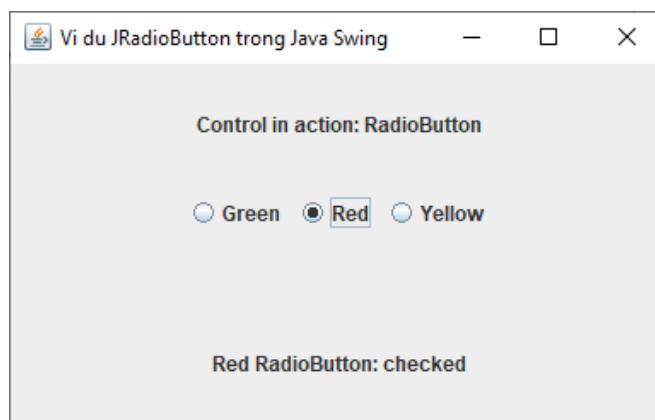
        mainFrame.setVisible(true);

    }

}

```

Kết quả:



❖ Lớp ButtonGroup trong Java Swing

Lớp ButtonGroup có thể được sử dụng để nhóm nhiều lớp lại với nhau, để mà tại một thời điểm, chỉ có một nút được lựa chọn.

- **Ví dụ:**

```

package chapter.swing;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

```

```
import javax.swing.JRadioButton;

public class JRadioButtonExam3 extends JFrame implements ActionListener {

    JRadioButton rb1, rb2;
    JButton b;

    JRadioButtonExam3() {

        rb1 = new JRadioButton("Green");
        rb1.setBounds(100, 50, 100, 30);

        rb2 = new JRadioButton("Red");
        rb2.setBounds(100, 100, 100, 30);

        ButtonGroup bg = new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);

        b = new JButton("click");
        b.setBounds(100, 150, 80, 30);
        b.addActionListener(this);

        add(rb1);
```

```
add(rb2);

add(b);

setTitle("Vi du JRadioButton trong Java Swing");

setSize(400, 250);

setLayout(null);

setVisible(true);

}

public void actionPerformed(ActionEvent e) {

if (rb1.isSelected()) {

JOptionPane.showMessageDialog(this, "Ban la Green");

}

if (rb2.isSelected()) {

JOptionPane.showMessageDialog(this, "Ban la Red");

}

}

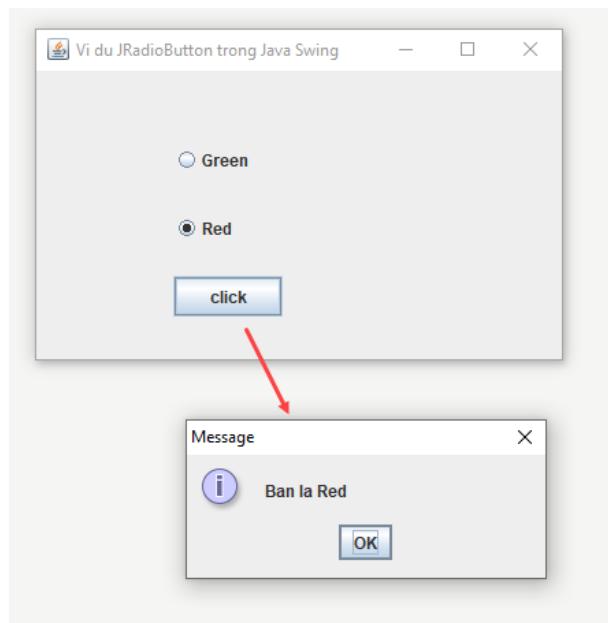
public static void main(String args[]) {

new JRadioButtonExam3();

}

}

Chạy chương trình Java trên cho kết quả như sau:
```



❖ Lớp JList trong Java Swing

Lớp JList trong Java Swing là một thành phần mà hiển thị một danh sách các đối tượng và cho phép người dùng lựa chọn một hoặc nhiều item. Một Model riêng rẽ, ListModel, duy trì các nội dung của list.

- **Lớp JList có các trường sau:**

- static int HORIZONTAL_WRAP: Trình bày một layout theo phong cách như một tờ báo ("newspaper style") với các ô tràn theo chiều ngang và sau đó là chiều dọc.
- static int VERTICAL: Chỉ một cách bố trí các ô theo chiều dọc, trong một cột đơn; đây là layout mặc định.
- static int VERTICAL_WRAP: Chỉ một layout theo phong cách như một tờ báo "newspaper style" với các ô tràn theo chiều dọc và sau đó là chiều ngang.

- **Các constructor của lớp JList trong Java Swing**

- JList(): Xây dựng một JList với một model là empty, read-only.
- JList(ListModel dataModel): Xây dựng một JList mà hiển thị các phần tử từ model đã cho và non-null.
- JList(Object[] listData): Xây dựng một JList mà hiển thị các phần tử trong mảng đã cho.
- JList(Vector listData): Xây dựng một JList mà hiển thị các phần tử trong Vector đã cho.

- **Các phương thức của lớp JList trong Java Swing**

STT	Phương thức & Mô tả
1	void addListSelectionListener(ListSelectionListener listener) <p>Thêm một Listener tới list, để được thông báo mỗi khi xuất hiện một thay đổi tới selection; đây là cách ưu tiên để nghe các trạng thái của thay đổi</p>
2	void addSelectionInterval(int anchor, int lead) <p>Thiết lập selection thành sự kết hợp của khoảng interval đã cho với selection hiện tại</p>
3	void clearSelection() <p>Xóa selection sau khi gọi phương thức này, isEmpty sẽ trả về true</p>
4	protected ListSelectionModel createSelectionModel() <p>Trả về một instance của DefaultListSelectionModel; được gọi trong khi xây dựng để khởi tạo thuộc tính selection model của list</p>
5	void ensureIndexIsVisible(int index) <p>Cuộn danh sách bên trong một viewport đang bao quanh để làm cho ô đã cho là hoàn toàn nhìn thấy</p>
6	protected void fireSelectionValueChanged(int firstIndex, int lastIndex, boolean isAdjusting) <p>Thông báo cho ListSelectionListeners để thêm trực tiếp tới danh sách các thay đổi của selection được tạo tới selection model</p>
7	AccessibleContext getAccessibleContext() <p>Lấy AccessibleContext được liên kết với JList này</p>

8	int getFirstVisibleIndex() Trả về chỉ mục nhỏ nhất của list mà có thể nhìn thấy hiện tại
9	int getLastVisibleIndex() Trả về chỉ mục lớn nhất của list mà có thể nhìn thấy hiện tại
10	Dimension getPreferredScrollableViewportSize() Tính toán kích cỡ của viewport cần thiết để hiển thị các hàng visibleRowCount
11	int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction) Trả về khoảng cách để cuộn để trung bày hàng trước đó hoặc khối tiếp theo (với cuộn theo chiều dọc) hoặc cột (với cuộn theo chiều ngang)
12	void removeListSelectionListener(ListSelectionListener listener) Xóa một selection listener từ list này
13	void removeSelectionInterval(int index0, int index1) Thiết lập selection để thiết lập sự khác nhau của interval đã cho và selection hiện tại
14	void setDragEnabled(boolean b) Tắt hoặc bật bộ xử lý hoạt động drag mặc định
15	void setDropMode(DropMode dropMode) Thiết lập drop mode cho thành phần này
16	void setFixedCellHeight(int height)

	Thiết lập một giá trị cố định để được sử dụng cho chiều cao của mỗi ô trong list
17	void setLayoutOrientation(int layoutOrientation) Định nghĩa cách các ô trong list được bố trí
18	void setListData(Object[] listData) Xây dựng một read-only ListModel từ một mảng các đối tượng và gọi setModel với model này
19	void setListData(Vector listData) Xây dựng một read-only ListModel từ một Vector và gọi setModel với model này
20	void setModel(ListModel model) Thiết lập model mà biểu diễn các nội dung hoặc "value" của list, thông báo sự thay đổi thuộc tính tới listener, và sau đó xóa selection của list
21	void setPrototypeCellValue(Object prototypeCellValue) Thiết lập thuộc tính prototypeCellValue, và sau đó (nếu giá trị mới là non-null) thì tính toán các thuộc tính fixedCellWidth và fixedCellHeight bởi yêu cầu thành phần cell renderer cho giá trị đã cho (và chỉ mục 0) từ cell renderer đó, và sử dụng kích cỡ của thành phần đó
22	void setSelectedIndex(int index) Lựa chọn một ô đơn
23	void setSelectedIndices(int[] indices) Thay đổi selection thành là tập hợp các chỉ mục được xác định bởi mảng đã cho

24	void setSelectedValue(Object anObject, boolean shouldScroll) Thiết lập đối tượng đã cho từ list
25	void setSelectionBackground(Color selectionBackground) Thiết lập màu được sử dụng để vẽ màu nền background của item được chọn, mà Call Renderer có thể sử dụng để điền vào ô đã chọn
26	void setSelectionForeground(Color selectionForeground) Thiết lập màu được sử dụng để vẽ foreground của item được chọn, mà Call Renderer có thể sử dụng để truyền text và đồ họa
27	void setSelectionInterval(int anchor, int lead) Thiết lập interval đã cho
28	void setSelectionMode(int selectionMode) Thiết lập selection mode cho list
29	void setSelectionModel(ListSelectionModel selectionModel) Thiết lập selectionModel cho list tới một trình triển khai non-null ListSelectionModel
30	void setUI(ListUI ui) Thiết lập ListUI, đối tượng L&F mà truyền đối tượng này
31	void setValueIsAdjusting(boolean b) Thiết lập thuộc tính valueIsAdjusting của selection model
32	void setVisibleRowCount(int visibleRowCount)

	Thiết lập thuộc tính visibleRowCount, mà có ý nghĩa khác nhau phụ thuộc vào hướng bố trí layout orientation: Với hướng bố trí VERTICAL, phương thức này thiết lập số hàng ưu tiên để hiển thị (không yêu cầu cuộn); với các hướng khác, phương thức này tác động đến việc bao các ô
33	void updateUI() Phục hồi thuộc tính ListUI bởi thiết lập nó tới giá trị được cung cấp bởi L&F hiện tại

- **Chương trình ví dụ lớp JList**

```
package chapter.swing;
```

```
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
```

```
import javax.swing.ListSelectionModel;

public class JListExam1 {

    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel statusLabel;
    private JPanel controlPanel;

    public JListExam1(){
        prepareGUI();
    }

    public static void main(String[] args) {
        JListExam1 jListExam1 = new JListExam1();
        jListExam1.showListDemo();
    }

    private void prepareGUI() {
        mainFrame = new JFrame("Vi du JList trong Java Swing");
        mainFrame.setSize(500, 300);
        mainFrame.setLayout(new GridLayout(3, 1));
        mainFrame.addWindowListener(new WindowAdapter() {
```

```
public void windowClosing(WindowEvent windowEvent) {  
    System.exit(0);  
}  
});  
  
headerLabel = new JLabel("", JLabel.CENTER);  
  
statusLabel = new JLabel("", JLabel.CENTER);  
statusLabel.setSize(350, 100);  
  
controlPanel = new JPanel();  
controlPanel.setLayout(new FlowLayout());  
  
mainFrame.add(headerLabel);  
mainFrame.add(controlPanel);  
  
mainFrame.add(statusLabel);  
  
mainFrame.setVisible(true);  
}  
  
  
private void showListDemo() {  
    headerLabel.setText("Control in action: JList");  
  
    final DefaultListModel colorsName = new DefaultListModel();  
  
    colorsName.addElement("Green");  
  
    colorsName.addElement("Red");  
  
    colorsName.addElement("Yello");  
  
    colorsName.addElement("Black");  
  
    final JList fruitList = new JList(colorsName);  
  
    fruitList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
fruitList.setSelectedIndex(0);

fruitList.setVisibleRowCount(3);

JScrollPane fruitListScrollPane = new JScrollPane(fruitList);

final DefaultListModel vegName = new DefaultListModel();

vegName.addElement("Broccoli");

vegName.addElement("Onion");

vegName.addElement("Potato");

vegName.addElement("Tomato");

final JList vegList = new JList(vegName);

vegList.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);

vegList.setSelectedIndex(0);

vegList.setVisibleRowCount(3);

JScrollPane vegListScrollPane = new JScrollPane(vegList);

JButton showButton = new JButton("Show");

showButton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        String data = "";

        if (fruitList.getSelectedIndex() != -1) {

            data = "Colors Selected: " + fruitList.getSelectedValue();

            statusLabel.setText(data);

        }

        if (vegList.getSelectedIndex() != -1) {

            data += ", Vegetables selected: ";

        }

    }

});
```

```

        for (Object vegetable : vegList.getSelectedValues()) {
            data += vegetable + ", ";
        }
    }

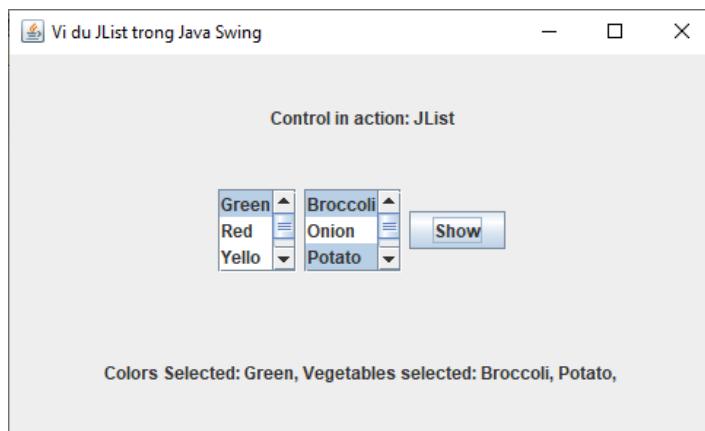
    statusLabel.setText(data);
}

controlPanel.add(fruitListScrollPane);
controlPanel.add(vegListScrollPane);
controlPanel.add(showButton);

mainFrame.setVisible(true);
}
}

```

Chạy chương trình Java trên cho kết quả như sau:



❖ Lớp JComboBox trong Java Swing

Lớp JComboBox trong Java Swing là một thành phần mà kết hợp một button, một trường có thể chỉnh sửa và một drop-down list. Tại một thời điểm chỉ có một item có thể được lựa chọn từ list.

- **Các constructor được sử dụng phổ biến của lớp JComboBox**

- JComboBox(): Tạo một JComboBox với data model mặc định.
- JComboBox(Object[] items): Tạo một JComboBox mà chứa các phần tử trong mảng đã cho.
- JComboBox(Vector items): Tạo một JComboBox mà chứa các phần tử trong Vector đã cho.

- **Các phương thức được sử dụng phổ biến của lớp JComboBox**

- public void addItem(Object anObject): được sử dụng để thêm một item tới list.
- public void removeItem(Object anObject): được sử dụng để xóa một item từ list.
- public void removeAllItems(): được sử dụng để xóa tất cả item từ list.
- public void setEditable(boolean b): được sử dụng để xác định xem có hay không JComboBox là editable.
- public void addActionListener(ActionListener a): được sử dụng để thêm ActionListener.
- public void addItemListener(ItemListener i): được sử dụng để thêm ItemListener.

- **Chương trình ví dụ đơn giản đầu tiên về lớp JComboBox trong Java Swing**

```
package chapter.swing;
```

```
import javax.swing.JComboBox;
```

```
import javax.swing.JFrame;
```

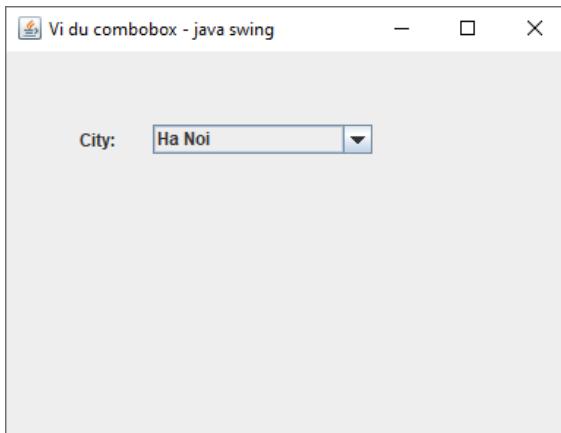
```
import javax.swing.JLabel;
```

```
public class JComboBoxExam1 {
```

```
    JFrame f;
```

```
JComboBoxExam1() {  
    f = new JFrame("Vi du combobox - java swing");  
  
    String city[] = { "Ha Noi", "Vinh Phuc", "Da Nang",  
                      "TP. Ho Chi Minh", "Nha Trang" };  
  
    JComboBox cb = new JComboBox(city);  
    cb.setBounds(100, 50, 150, 20);  
    f.add(cb);  
  
    JLabel cityLabel = new JLabel("City: ");  
    cityLabel.setBounds(50, 50, 80, 20);  
    f.add(cityLabel);  
  
    f.setLayout(null);  
    f.setSize(400, 300);  
    f.setVisible(true);  
}  
  
public static void main(String[] args) {  
    new JComboBoxExam1();  
}  
}
```

Chạy chương trình trên cho kết quả như sau:



- **Một chương trình khác về lớp JComboBox**

```
import java.awt.FlowLayout;  
  
import java.awt.GridLayout;  
  
import java.awt.event.ActionEvent;  
  
import java.awt.event.ActionListener;  
  
import java.awt.event.WindowAdapter;  
  
import java.awt.event.WindowEvent;  
  
  
import javax.swing.DefaultComboBoxModel;  
  
import javax.swing.JButton;  
  
import javax.swing.JComboBox;  
  
import javax.swing.JFrame;  
  
import javax.swing.JLabel;  
  
import javax.swing.JPanel;  
  
import javax.swing.JScrollPane;
```

```
public class JComboBoxExam2 {  
  
    private JFrame mainFrame;  
    private JLabel headerLabel;  
    private JLabel statusLabel;  
    private JPanel controlPanel;  
  
    public JComboBoxExam2(){  
        prepareGUI();  
    }  
  
    public static void main(String[] args) {  
        JComboBoxExam2 swingDemo = new JComboBoxExam2();  
        swingDemo.showComboboxDemo();  
    }  
  
    private void prepareGUI() {  
        mainFrame = new JFrame("Vi du combobox - java swing");  
        mainFrame.setSize(400, 400);  
        mainFrame.setLayout(new GridLayout(3, 1));  
        mainFrame.addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent windowEvent) {  
                System.exit(0);  
            }  
        });  
    }  
}
```

```
    }

});

headerLabel = new JLabel("", JLabel.CENTER);

statusLabel = new JLabel("", JLabel.CENTER);

statusLabel.setSize(350, 100);

controlPanel = new JPanel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showComboboxDemo() {

    headerLabel.setText("Control in action: JComboBox");

    final DefaultComboBoxModel cityName = new DefaultComboBoxModel();

    cityName.addElement("Ha Noi");

    cityName.addElement("TP. HCM");

    cityName.addElement("Da Nang");

    cityName.addElement("Hai Phong");

    final JComboBox fruitCombo = new JComboBox(cityName);

    fruitCombo.setSelectedIndex(0);

    JScrollPane fruitListScrollPane = new JScrollPane(fruitCombo);
```

```
JButton showButton = new JButton("Show");

showButton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        String data = "";
        if (fruitCombo.getSelectedIndex() != -1) {
            data = "City Selected: " + fruitCombo.getItemAt(
                fruitCombo.getSelectedIndex());
        }
        statusLabel.setText(data);
    }
});;

controlPanel.add(fruitListScrollPane);

controlPanel.add(showButton);

mainFrame.setVisible(true);

}
}
```

Chạy chương trình trên cho kết quả như sau:



❖ Lớp JTextField trong Java Swing

Lớp JTextField trong Java Swing là một thành phần cho phép sửa đổi một dòng text đơn.

- **Các constructor của lớp JTextField trong Java Swing**

- JTextField(): Xây dựng một TextField mới.
- JTextField(Document doc, String text, int columns): Xây dựng một JTextField mới mà sử dụng mô hình lưu trữ text đã cho và số cột đã cho.
- JTextField(int columns): Xây dựng một TextField mới và trông với số cột đã cho.
- JTextField(String text): Xây dựng một TextField mới được khởi tạo với text đã cho.
- JTextField(String text, int columns): Xây dựng một TextField mới được khởi tạo với text và các cột đã cho.

- **Các phương thức được sử dụng phổ biến của lớp JTextField trong Java Swing**

STT	Phương thức & Mô tả
1	void setActionCommand(String command) Thiết lập chuỗi lệnh được sử dụng cho action event
2	void setColumns(int columns) Thiết lập số cột trong TextField này, và sau đó làm mất hiệu ứng layout đó
3	void setDocument(Document doc) Liên kết editor với một tài liệu text
4	void setFont(Font f) Thiết lập font hiện tại
5	void setHorizontalAlignment(int alignment)

	Thiết lập căn chỉnh ngang cho text
6	void setScrollOffset(int scrollOffset) Thiết lập scroll offset, giá trị pixel
7	protected void actionPerformed(ActionEvent action, String propertyName) Cập nhật trạng thái của textfield trong phản hồi các thay đổi của thuộc tính trong action liên kết với
8	void addActionListener(ActionListener l) Thêm action listener đã cho để nhận các action event từ textfield này
9	protected void configurePropertiesFromAction(ActionEvent a) Thiết lập các thuộc tính của textfield này để kết nối chúng trong Action đã cho
10	protected PropertyChangeListener createActionPropertyChangeListener(ActionEvent a) Tạo và trả về PropertyChangeListener mà chịu trách nhiệm nghe các thay đổi từ Action đã cho và cập nhật các thuộc tính thích hợp
11	protected Document createDefaultModel() Tạo trình triển khai mặc định của model để được sử dụng tại sự xây dựng nếu không được cung cấp tường minh
12	Action[] getActions() Gọi danh sách lệnh cho trình soạn thảo Editor
13	void postActionEvent()

	Xử lý action event xảy ra trên textfield này bằng cách gửi chúng tới bất cứ đối tượng ActionListener đã được đăng ký nào
14	void removeActionListener(ActionListener l) Xóa action listener đã cho để nó không bao giờ nhận action event từ textfield này nữa
15	void scrollRectToVisible(Rectangle r) Cuộn trường này sang trái hoặc phải
16	void setAction(Action a) Thiết lập Action cho ActionEvent source

- **Chương trình ví dụ lớp JTextField**

```

package chapter.swing;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;
  
```

```
import javax.swing.JPasswordField;  
  
import javax.swing.JTextField;  
  
  
public class JTextFieldExam {  
  
  
    private JFrame mainFrame;  
  
    private JLabel headerLabel;  
  
    private JLabel statusLabel;  
  
    private JPanel controlPanel;  
  
    public JTextFieldExam(){  
  
        prepareGUI();  
  
    }  
  
  
    public static void main(String[] args) {  
  
        JTextFieldExam swingDemo = new JTextFieldExam();  
  
        swingDemo.showTextFieldDemo();  
  
    }  
  
    private void prepareGUI() {  
  
        mainFrame = new JFrame("Vi du JTextField - Java Swing");  
  
        mainFrame.setSize(400, 300);  
  
        mainFrame.setLayout(new GridLayout(3, 1));  
  
        mainFrame.addWindowListener(new WindowAdapter() {  
  
            public void windowClosing(WindowEvent windowEvent) {  
  
                System.exit(0);  
            }  
        });  
    }  
}
```

```
    }

});

headerLabel = new JLabel("", JLabel.CENTER);

statusLabel = new JLabel("", JLabel.CENTER);

statusLabel.setSize(350, 100);

controlPanel = new JPanel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showTextFieldDemo() {

    headerLabel.setText("Control in action: JTextField");

    JLabel namelabel = new JLabel("User ID: ", JLabel.RIGHT);

    JLabel passwordLabel = new JLabel("Password: ", JLabel.CENTER);

    final JTextField userText = new JTextField(6);

    final JPasswordField passwordText = new JPasswordField(6);

    JButton loginButton = new JButton("Login");

    loginButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            String data = "Username " + userText.getText();
```

```

        data += ", Password: " + new String(passwordText.getPassword());

        statusLabel.setText(data);

    }

});

controlPanel.add(namelabel);

controlPanel.add(userText);

controlPanel.add(passwordLabel);

controlPanel.add(passwordText);

controlPanel.add(loginButton);

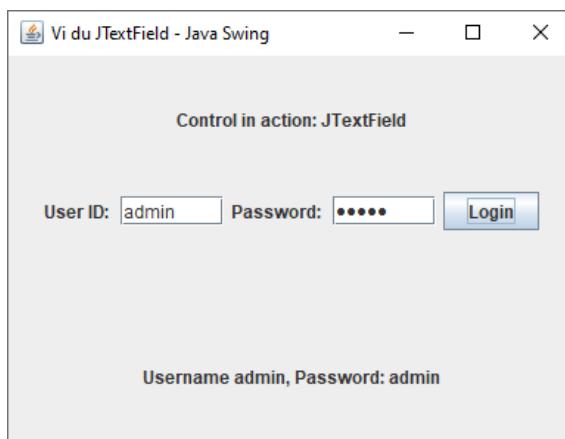
mainFrame.setVisible(true);

}

}

```

Chạy chương trình trên cho kết quả như sau:



❖ Lớp JOptionPane trong Java Swing

Lớp JOptionPane trong Java Swing là một thành phần cung cấp các phương thức chuẩn để gọi một hộp thoại dialog chuẩn cho một giá trị hoặc thông báo người dùng về một cái gì đó.

- **Các constructor của lớp JOptionPane trong Java Swing**

- JOptionPane(): Tạo một JOptionPane với một thông điệp kiểm tra (test message).
- JOptionPane(Object message): Tạo một instance của JOptionPane để hiển thị một message bởi sử dụng kiểu thông điệp thuận message và các tùy chọn option mặc định được phân phối bởi UI.
- JOptionPane(Object message, int messageType): Tạo một instance của JOptionPane để hiển thị một thông điệp với kiểu thông điệp đã cho và các tùy chọn mặc định.
- JOptionPane(Object message, int messageType, int optionType): Tạo một instance của JOptionPane để hiển thị một thông điệp với kiểu thông điệp đã cho và các tùy chọn mặc định.
- JOptionPane(Object message, int messageType, int optionType, Icon icon): Tạo một instance của JOptionPane để hiển thị một thông điệp với kiểu thông điệp, tùy chọn và icon đã cho.
- JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options): Tạo một instance của JOptionPane để hiển thị một thông điệp với kiểu thông điệp, tùy chọn và icon đã cho.
- JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue): Tạo một instance của JOptionPane để hiển thị một thông điệp với kiểu thông điệp, tùy chọn và icon đã cho với giá trị tùy chọn được lựa chọn ban đầu đã được xác định.

- **Các phương thức được sử dụng phổ biến của lớp JOptionPane**

STT	Phương thức & Mô tả
1	void selectInitialValue() Yêu cầu giá trị khởi tạo ban đầu để được lựa chọn, mà sẽ thiết lập focus tới giá trị đó
2	void setIcon(Icon newIcon) Thiết lập icon để hiển thị
3	void setInitialSelectionValue(Object newValue)

	Thiết lập giá trị input mà được hiển thị ban đầu như là selected tới người dùng
4	void setInitialValue(Object newValue) Thiết lập giá trị ban đầu để được kích hoạt. Đây là thành phần mà có focus khi pane được hiển thị ban đầu
5	void setInputValue(Object newValue) Thiết lập giá trị input mà là selected hoặc input bởi người dùng
6	void setMessage(Object newMessage) Thiết lập đối tượng message của option pane
7	void setMessageType(int newType) Thiết lập kiểu thông điệp của option pane
8	void setOptions(Object[] newOptions) Thiết lập các tùy chọn mà pane này hiển thị
9	void setOptionType(int newType) Thiết lập các tùy chọn để hiển thị
10	static void setRootFrame(Frame newRootFrame) Thiết lập frame để sử dụng cho các phương thức lớp mà chưa được cung cấp frame nào
11	void setSelectionValues(Object[] newValues) Thiết lập các giá trị selection cho một pane mà cung cấp cho người dùng một danh sách item để lựa chọn từ đó
12	void setUI(OptionPaneUI ui)

	Thiết lập đối tượng UI mà triển khai L&F cho thành phần này
13	void setValue(Object newValue) Thiết lập giá trị mà người dùng đã lựa chọn
14	void setWantsInput(boolean newValue) Thiết lập thuộc tính wantsInput
15	static int showConfirmDialog(Component parentComponent, Object message) Hiển thị một hộp thoại với các tùy chọn Yes, No và Cancel với title là Select an Option
16	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType) Hiển thị một hộp thoại, với số tùy chọn được xác định bởi tham số optionType
17	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) Hiển thị một hộp thoại, với số tùy chọn được xác định bởi tham số optionType, và tham số messageType xác định icon để hiển thị
18	static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) Hiển thị một hộp thoại với icon đã cho, với số tùy chọn được xác định bởi tham số optionType

19	static String showInputDialog(Component parentComponent, Object message, Object initialValue)
	Hiển thị một hộp thoại dạng question-message yêu cầu input từ người dùng được tạo ra từ parentComponent
20	static String showInputDialog(Component parentComponent, Object message, String title, int messageType)
	Hiển thị một hộp thoại dạng question-message yêu cầu input từ người dùng được tạo ra từ parentComponent với hộp thoại có title và messageType
21	static Object showInternalInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialValue)
	Gợi ý người dùng nhập input trong một hộp thoại nội tại, ở đây sự lựa chọn ban đầu, sự lựa chọn có thể có, và tất cả tùy chọn khác có thể được xác định
22	JDialog createDialog(Component parentComponent, String title)
	Tạo và trả về một JDialog mới mà bao quanh optionpane này được căn chỉnh vào giữa parentComponent trong frame của parentComponent
23	JDialog createDialog(String title)
	Tạo và trả về một JDialog mới (không phải là cha) với title đã cho
24	JInternalFrame createInternalFrame(Component parentComponent, String title)
	Tạo và trả về một instance của JInternalFrame

- **Chương trình ví dụ lớp JOptionPane**

```
package chapter.swing;

import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class JOptionPaneExam1 {

    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel statusLabel;
    private JPanel controlPanel;
```

```
public JOptionPaneExam1() {  
    prepareGUI();  
}  
  
public static void main(String[] args) {  
    JOptionPaneExam1 demo = new JOptionPaneExam1();  
    demo.showDialogDemo();  
}  
  
private void prepareGUI() {  
    mainFrame = new JFrame("Vi du JOptionPane trong Java Swing");  
    mainFrame.setSize(500, 400);  
    mainFrame.setLayout(new GridLayout(3, 1));  
    mainFrame.addWindowListener(new WindowAdapter() {  
        public void windowClosing(WindowEvent windowEvent) {  
            System.exit(0);  
        }  
    });  
    headerLabel = new JLabel("", JLabel.CENTER);  
    statusLabel = new JLabel("", JLabel.CENTER);  
    statusLabel.setSize(350, 100);  
    controlPanel = new JPanel();  
    controlPanel.setLayout(new FlowLayout());  
    mainFrame.add(headerLabel);
```

```
mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showDialogDemo() {

    headerLabel.setText("Control in action: JOptionPane");

    JButton okButton = new JButton("OK");

    JButton javaButton = new JButton("Yes/No");

    JButton cancelButton = new JButton("Yes/No/Cancel");

    okButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            JOptionPane.showMessageDialog(mainFrame,

                "Welcome to VietTuts.Vn");

        }

    });

    javaButton.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            int output = JOptionPane.showConfirmDialog(mainFrame,

                "Click any button", "VietTuts.Vn",

                JOptionPane.YES_NO_OPTION);

            if (output == JOptionPane.YES_OPTION) {

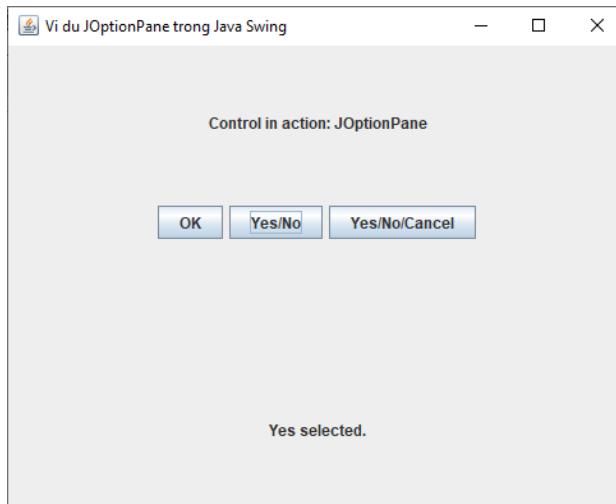
                statusLabel.setText("Yes selected.");
            }
        }
    });
}
```

```
    } else if (output == JOptionPane.NO_OPTION) {  
  
        statusLabel.setText("No selected.");  
  
    }  
  
});  
  
cancelButton.addActionListener(new ActionListener() {  
  
    public void actionPerformed(ActionEvent e) {  
  
        int output = JOptionPane.showConfirmDialog(mainFrame,  
                "Click any button", "VietTuts.Vn",  
                JOptionPane.YES_NO_CANCEL_OPTION,  
                JOptionPane.INFORMATION_MESSAGE);  
  
        if (output == JOptionPane.YES_OPTION) {  
  
            statusLabel.setText("Yes selected.");  
  
        } else if (output == JOptionPane.NO_OPTION) {  
  
            statusLabel.setText("No selected.");  
  
        } else if (output == JOptionPane.CANCEL_OPTION) {  
  
            statusLabel.setText("Cancel selected.");  
  
        }  
  
    }  
  
});  
  
controlPanel.add(okButton);  
  
controlPanel.add(javaButton);  
  
controlPanel.add(cancelButton);  
  
mainFrame.setVisible(true);
```

```
}
```

```
}
```

Chạy chương trình trên cho kết quả như sau:



❖ Lớp JMenuBar trong Java Swing

Mỗi cửa sổ window có một thanh trình đơn (menu bar) được liên kết với nó. Thanh trình đơn này gồm các lựa chọn có sẵn tới người dùng cuối cùng. Các điều khiển Menu và MenuItem là lớp con của lớp MenuComponent.

- **Constructor của lớp JMenuBar**

JMenuBar(): Tạo một thanh trình đơn mới.

- **Chương trình ví dụ về lớp JMenuBar**

SwingMenuDemo.java

```
package chapter.swing;
```

```
import java.awt.FlowLayout;  
  
import java.awt.GridLayout;  
  
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.WindowConstants;

public class JMenuBarExam1 {
    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel statusLabel;
    private JPanel controlPanel;

    public JMenuBarExam1() {
```

```
prepareGUI();

}

public static void main(String[] args) {
    JMenuBarExam1 demo = new JMenuBarExam1();
    demo.showMenuDemo();
}

private void prepareGUI() {
    mainFrame = new JFrame("Vi du JMenuBar trong Java Swing");
    mainFrame.setSize(400, 300);
    mainFrame.setLayout(new GridLayout(3, 1));
    headerLabel = new JLabel("", JLabel.CENTER);
    statusLabel = new JLabel("", JLabel.CENTER);
    mainFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    statusLabel.setSize(350, 100);
    controlPanel = new JPanel();
    controlPanel.setLayout(new FlowLayout());
    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
}

private void showMenuDemo() {
```

```
// Tao mot menu bar

final JMenuBar menuBar = new JMenuBar(); // Tao cac menu

JMenu fileMenu = new JMenu("File");

JMenu editMenu = new JMenu("Edit");

final JMenu aboutMenu = new JMenu("About");

final JMenu linkMenu = new JMenu("Links");

// tao cac item

JMenuItem newItem = new JMenuItem("New");

newItem.setMnemonic(KeyEvent.VK_N);

newItem.setActionCommand("New");

JMenuItem openMenuItem = new JMenuItem("Open");

openMenuItem.setActionCommand("Open");

JMenuItem saveMenuItem = new JMenuItem("Save");

saveMenuItem.setActionCommand("Save");

JMenuItem exitMenuItem = new JMenuItem("Exit");

exitMenuItem.setActionCommand("Exit");

JMenuItem cutMenuItem = new JMenuItem("Cut");

cutMenuItem.setActionCommand("Cut");

JMenuItem copyMenuItem = new JMenuItem("Copy");

copyMenuItem.setActionCommand("Copy");

JMenuItem pasteMenuItem = new JMenuItem("Paste");

pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
```

```
openMenuItem.addActionListener(menuItemListener);

saveMenuItem.addActionListener(menuItemListener);

exitMenuItem.addActionListener(menuItemListener);

cutMenuItem.addActionListener(menuItemListener);

copyMenuItem.addActionListener(menuItemListener);

pasteMenuItem.addActionListener(menuItemListener);

final JCheckBoxMenuItem showWindowMenu = new JCheckBoxMenuItem("Show About", true);

showWindowMenu.addItemListener(new ItemListener() {

    public void itemStateChanged(ItemEvent e) {

        if (showWindowMenu.getState()) {

            menuBar.add(aboutMenu);

        } else {

            menuBar.remove(aboutMenu);

        }

    }

});

final JRadioButtonMenuItem showLinksMenu = new JRadioButtonMenuItem("Show Links", true);

showLinksMenu.addItemListener(new ItemListener() {

    public void itemStateChanged(ItemEvent e) {

        if (menuBar.getMenu(3) != null) {

            menuBar.remove(linkMenu);

            mainFrame.repaint();

        } else {

    }

}};
```

```
menuBar.add(linkMenu);

mainFrame.repaint();

}

}

}); // Them item toi cac menu

fileMenu.add(newMenuItem);

fileMenu.add(openMenuItem);

fileMenu.add(saveMenuItem);

fileMenu.addSeparator();

fileMenu.add(showWindowMenu);

fileMenu.addSeparator();

fileMenu.add(showLinksMenu);

fileMenu.addSeparator();

fileMenu.add(exitMenuItem);

editMenu.add(cutMenuItem);

editMenu.add(copyMenuItem);

editMenu.add(pasteMenuItem); // Them menu toi menubar

menuBar.add(fileMenu);

menuBar.add(editMenu);

menuBar.add(aboutMenu);

menuBar.add(linkMenu); // Them menubar toi frame

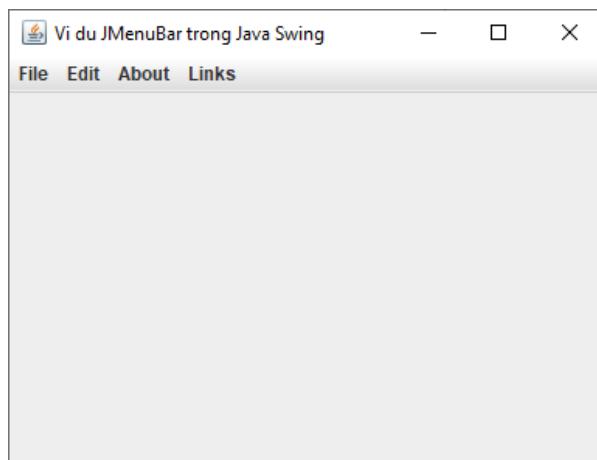
mainFrame.setJMenuBar(menuBar);

mainFrame.setVisible(true);

}
```

```
class MenuItemListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        statusLabel.setText(e.getActionCommand() + " JMenuItem clicked.");  
    }  
}  
}
```

Chạy chương trình trên cho kết quả như sau:



TÀI LIỆU THAM KHẢO

- [1] Ravi Kant Soni. *Learning Spring Application Development*. Packt Publishing Ltd. 2015.
- [2] Amuthan Ganeshan. *Spring MVC Beginner's Guide*. Packt Publishing Ltd. 2014.
- [3] Dương Hữu Thành. *Lập trình Java*. NXB Thông tin & Truyền thông. 2019.
- [4] <https://www.tutorialspoint.com/spring/index.htm> (truy cập 28/11/2019)
- [5] <https://www.journaldev.com/2888/spring-tutorial-spring-core-tutorial> (truy cập 28/11/2019)
- [6] <https://www.javatpoint.com/java-swing>