### **Introduction**

#### Xin chào học viên!

#### Tôi là Mr.Ring. Tôi sẽ là cánh tay giúp đỡ của bạn trong suốt hành trình Java Spring này.

#### Tôi khá chắc chắn rằng khi bạn hoàn thành khóa học này, bạn sẽ có kiến thức tốt về Spring Framework.

#### Spring thường được mô tả là "**Một framework nhẹ để xây dựng các ứng dụng Java**".

#### Điều này có nghĩa là người ta có thể xây dựng bất kỳ loại ứng dụng Java nào với Spring như Java độc lập, Ứng dụng JEE, Ứng dụng web, v.v.

#### Framework nhẹ có nghĩa là bạn chỉ phải thực hiện một vài thay đổi hoặc không có gì đối với mã ứng dụng của mình để có được những lợi ích của Spring. Bạn không cần phải mở rộng các lớp của mình từ bất kỳ thành phần Spring framework cụ thể nào để có được các tính năng của framework.

### **Spring == Easy Development**

#### Java thường được cho là phức tạp để xây dựng các ứng dụng đơn giản.

#### Tuy nhiên, Java cung cấp một nền tảng ổn định với một hệ sinh thái trưởng thành xung quanh nó, điều này làm cho nó trở thành một lựa chọn tuyệt vời để phát triển phần mềm mạnh mẽ.

#### Spring Framework là một trong nhiều framework mạnh mẽ trong hệ sinh thái Java đi kèm với một tập hợp các mô hình lập trình và cấu hình với mục tiêu đơn giản hóa việc phát triển các ứng dụng hiệu suất và có thể kiểm thử trong Java.

#### Hãy tiến về phía trước và khám phá thế giới của mùa xuân.

### **All about Spring**

#### Spring là mã nguồn mở và một khung nhẹ được phát triển bởi **Rod Johnson vào năm** 2003.

#### Theo một cách nào đó, nó là một khung công tác vì nó cung cấp hỗ trợ cho các khung Java khác nhau như Struts, Hibernate, Tapestry, EJB, JSF, v.v.

#### Khung theo nghĩa rộng hơn có thể được định nghĩa là một cấu trúc sử dụng mà bạn có thể giải quyết nhiều vấn đề kỹ thuật.

#### Có thể nói Spring Framework là một công cụ toàn diện để hỗ trợ các ứng dụng sử dụng ngôn ngữ Java.

### **Why Spring?**

#### Mùa xuân là mô-đun trong tự nhiên, có nghĩa là bạn có thể sử dụng các bộ phận mà bạn cần thay vì sử dụng toàn bộ.

#### Sử dụng Spring Framework, bạn có thể xây dựng các ứng dụng Java cũng như các loại ứng dụng web khác (bằng cách sử dụng các phần mở rộng).

#### Spring sử dụng Plain Old Java Object(POJO) và áp dụng các service enterprise cho nó. Spring nhằm mục đích làm cho việc phát triển J2EE dễ dàng hơn và nhanh hơn.

### **Features of Spring Introduction**

#### Mùa xuân trông thật tuyệt vời cho đến nay! Bây giờ bạn đã có một ý tưởng cơ bản về mùa xuân là gì, chúng ta hãy xem xét một số lợi ích của mùa xuân.

#### Hãy tiếp tục và khám phá một số tính năng thú vị làm cho Spring trở nên độc đáo.

### **Benefits of Spring Framework**

#### Spring sử dụng Java POJOs (Plain Old Java Object), giúp việc xây dựng các ứng dụng doanh nghiệp trở nên đơn giản hơn nhiều so với các EJB nặng (Enterprise Java Beans).

#### Spring có nhiều mô-đun theo kiểu mô-đun. Mặc dù số lượng gói và lớp học là đáng kể, bạn chỉ phải lo lắng về những gói bạn cần và bỏ qua phần còn lại.

#### Các ứng dụng được phát triển bằng Spring rất đơn giản vì mã phụ thuộc vào môi trường được chuyển vào framework này.

### **Features**

#### **Trọng lượng nhẹ**: Mùa xuân nhẹ khi nói đến kích thước và độ trong suốt. Phiên bản cơ bản của spring framework là khoảng 1MB và chi phí xử lý cũng rất không đáng kể.

#### **MVC Framework**: Web framework của Spring là một framework MVC được thiết kế tốt, cung cấp một giải pháp thay thế tuyệt vời cho các framework Java khác như Struts hoặc các framework web được thiết kế quá mức hoặc ít phổ biến khác.

#### Nó có thể cấu hình thông qua các giao diện và chứa nhiều công nghệ xem.

### **More Features**

#### **Inversion Of** Control (IoC): Vào mùa xuân, khớp nối lỏng lẻo đạt được bằng cách sử dụng Inversion of Control. Các đối tượng cung cấp các phụ thuộc riêng của chúng thay vì tạo hoặc tìm kiếm các đối tượng phụ thuộc.

#### **Aspect-Oriented Programming** (AOP): Bằng cách tách logic nghiệp vụ khỏi các dịch vụ hệ thống, Spring Framework hỗ trợ Lập trình hướng khía cạnh và cho phép phát triển gắn kết.

#### Quản lý giao dịch: Khung mùa xuân cung cấp một lớp trừu tượng chung để quản lý giao dịch. Điều này cho phép nhà phát triển thêm các trình quản lý giao dịch có thể cắm được và giúp dễ dàng xác định các giao dịch mà không cần xử lý các vấn đề cấp thấp.

### **Spring Installation**

### **Giới thiệu**

#### Chào học viên! Chào mừng trở lại.

#### Bạn đã xóa phiên đầu tiên của mình! Wow!

#### Bây giờ chúng ta hãy tiến lên phía trước và thiết lập môi trường cho sự phát triển mùa xuân.

#### Đi cùng.

#### Trước khi chúng ta tiếp tục và viết code đầu tiên vào Spring, chúng ta cần chuẩn bị môi trường phát triển của mình.

#### Để cài đặt Spring Framework có một số bước mà bạn cần làm theo,

* Setup Java Development Kit (JDK)
* Installing Apache Common Logging API
* Setup Eclipse IDE

#### Hãy làm việc trên từng bước.

### **Setup Java Development Kit (JDK)**

#### Bạn có thể tải xuống phiên bản Java SDK mới nhất từ trang web Java chính thức của Oracle, https://www.oracle.com/java/technologies/javase-downloads.html

#### Bạn sẽ tìm thấy hướng dẫn cài đặt JDK trong các tệp đã tải xuống, hãy làm theo hướng dẫn đã cho để cài đặt và định cấu hình thiết lập.

#### Sau đó thiết lập **PATH** cũng như **JAVA\_HOME** biến môi trường vào thư mục đó có chứa java và javac.

### **Installing Apache Common Logging API**

#### Bạn có thể tải xuống phiên bản mới nhất của API ghi nhật ký Apache Commons từ, https://commons.apache.org/logging/

#### Sau khi bạn tải xuống cài đặt, hãy giải nén bản phân phối nhị phân vào một vị trí thuận tiện.

#### Thư mục này sẽ có các tập tin jar và các tài liệu hỗ trợ khác như thể hiện trong hình trên.

#### Đảm bảo biến CLASSPATH được thiết lập đúng trên thư mục của bạn nếu không nó sẽ gây ra sự cố trong khi chạy ứng dụng của bạn.

### **Thiết lập Eclipse IDE**

#### Để cài đặt Eclipse IDE trên PC của bạn, hãy tải xuống các tệp nhị phân mới nhất từ, https://www.eclipse.org/downloads/

#### Sau khi bạn tải xuống cài đặt, hãy giải nén bản phân phối nhị phân vào một vị trí thuận tiện.

#### Nó sẽ trông giống như hình ảnh trên sau khi khởi động thành công Eclipse.

### **Setting up Spring Framework Libraries**

#### Bây giờ với việc hoàn thành tất cả các bước trên, bước cuối cùng cuối cùng là thiết lập Spring Framework của bạn.

#### Đối với điều này, hãy chắc chắn rằng bạn đưa ra lựa chọn xem khung có nên nằm trên hệ điều hành nào hay không.

#### Sau đó tiến hành tải xuống tệp .zip cho Windows và tệp .tz cho Unix hoặc macOS.

#### Download phiên bản mới nhất của thư viện Spring framework từ, https://repo.spring.io/release/org/springframework/spring

### **Setting up Spring Framework Libraries**

#### Sau khi tệp được giải nén, nó cung cấp cấu trúc tệp như hình trên.

#### Bạn sẽ tìm thấy tất cả các thư viện Spring bên trong thư mục libs.

#### Nếu bạn đang sử dụng Eclipse, thì không bắt buộc phải đặt CLASSPATH vì tất cả các cài đặt sẽ được thực hiện thông qua Eclipse.

#### Khi bạn đã hoàn thành việc này, bạn đã sẵn sàng để tiến hành ứng dụng Spring đầu tiên của mình.

#### Hãy chuyển sang phần tiếp theo, nơi chúng ta sẽ tạo ứng dụng Java Spring đầu tiên của chúng ta.

### **Your First Spring Application Introduction**

### 

#### Sau khi hoàn thành tất cả các quy trình cài đặt cần thiết, chúng tôi đã sẵn sàng để tiếp tục và viết một số mã bằng Java Spring.

#### Chúng tôi sẽ sử dụng Eclipse IDE trong suốt khóa học, vì vậy hãy chắc chắn rằng bạn cảm thấy thoải mái với nó.

#### Điều đó nói rằng, hãy cuộn và bắt đầu với ứng dụng 'Hello World' thường xanh với một chút Spring.

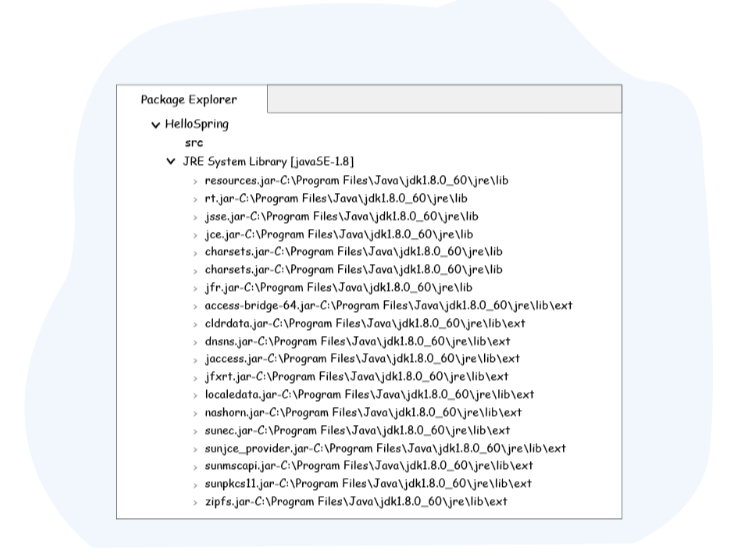
### **Step 1: Create Java project**

#### Bước đầu tiên là tạo một Java Project đơn giản trong Eclipse IDE,

* Mở Eclipse IDE
* File -> New -> Project: Chọn Java Project

#### Thực hiện theo tùy chọn File → New → Project và cuối cùng chọn Java Project wizard từ danh sách wizard. Bây giờ đặt tên cho dự án của bạn là HelloSpring và nhấp vào Kết thúc.

#### Khi dự án của bạn được tạo thành công, bạn sẽ có nội dung sau trong Project Explorer của mình,

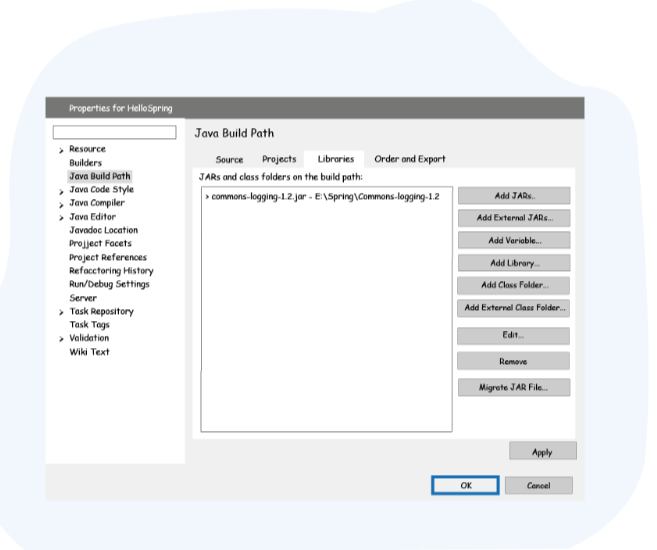


### **Step 2: Add required libraries**

#### Bước tiếp theo là thêm Spring Framework và các thư viện API log phổ biến trong project của chúng ta.

#### Để làm điều này, nhấp chuột phải vào tên dự án của bạn HelloSpring và đi tới Build Path -> Configure Build Path để hiển thị cửa sổ Java Build Path.

#### Bây giờ hãy sử dụng nút Add **External JARs** có sẵn trong tab Libraries để thêm các JAR cốt lõi sau từ các thư mục cài đặt Spring Framework và Common Logging mà chúng ta đã tải xuống trước đó.



### **Step 3: Build Source Files**

#### Bây giờ chúng ta hãy tạo các tệp nguồn thực tế trong dự án HelloSpring.

#### Đầu tiên, chúng ta cần tạo một gói, ví dụ: com.programminghub. Để thực hiện việc này, nhấp chuột phải vào thư mục src trong phần trình khám phá gói và làm theo tùy chọn - Gói → mới.

#### Tiếp theo, chúng ta sẽ tạo các tập tin HelloWorld.java và MainApp.java trong gói com.programminghub.

package com.programminghub;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

### **MainApp.java**

package com.programminghub;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

}

}

### **Note**

#### Bước đầu tiên là tạo một ngữ cảnh ứng dụng nơi chúng ta sử dụng framework **API ClassPathXmlApplicationContext**.

#### API này tải tệp cấu hình beans **Beans.xml** (mà chúng ta sẽ tạo ở bước tiếp theo) để tạo và khởi tạo tất cả các đối tượng.

#### Bước tiếp theo là lấy đậu cần thiết bằng cách sử dụng phương thức **getBean** của ngữ cảnh đã tạo.

#### Phương thức này sử dụng bean ID helloWorld để trả về một đối tượng chung mà cuối cùng có thể được đúc cho đối tượng thực tế. Khi bạn có một đối tượng, bạn có thể sử dụng đối tượng này để gọi bất kỳ phương thức lớp nào.

### **Create Bean Configuration file**

#### Bạn cần tạo một tệp Cấu hình Bean là một tệp XML và hoạt động như một xi măng dán các hạt đậu và các lớp lại với nhau.

#### Bây giờ trong thư mục src, tạo một tệp Beans.xml. (Nhấp chuột phải vào src, chọn New -> Other -> XML: XML File).

#### Beans.xml được sử dụng để gán các ID duy nhất cho các beans khác nhau và để kiểm soát việc tạo các đối tượng với các giá trị khác nhau mà không ảnh hưởng đến bất kỳ tệp nguồn Spring nào.

### **Beans.xml file**

#### Ví dụ: sử dụng tệp bên dưới, chúng tôi có thể chuyển bất kỳ giá trị nào cho biến "tin nhắn" và in các giá trị khác nhau của tin nhắn mà không ảnh hưởng đến các tệp **HelloWorld.java** và **MainApp.java**.

<?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 = "com.programminghub.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

### **Unique ID**

#### Khi ứng dụng Spring được load vào memory, framework sử dụng file cấu hình previous để tạo ra tất cả các beans được định nghĩa và gán cho chúng một ID duy nhất như được định nghĩa trong thẻ .

#### Bạn có thể sử dụng thẻ để truyền các giá trị của các biến khác nhau được sử dụng tại thời điểm tạo đối tượng.

#### Trong trường hợp này, giá trị 'Hello World!' được chuyển đến thông báo field của class **com.programminghub.HelloWorld** cho bean có ID helloWorld.

### **Let’s Execute**

#### Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và đậu, bạn đã sẵn sàng để biên dịch và chạy chương trình của mình. Để thực hiện việc này, hãy giữ tab tệp MainApp.Java hoạt động và nhấp vào Chạy hoặc sử dụng Ctrl + F11 để biên dịch và chạy.

#### Nếu mọi thứ đều ổn với ứng dụng của bạn, thao tác này sẽ in thông báo sau trong bảng điều khiển của Eclipse IDE:

Tin nhắn của bạn: Xin chào thế giới!

#### Hurray! bạn đã tạo thành công Spring Application đầu tiên của mình.

#### Các bạn có thể thấy sự linh hoạt của ứng dụng Spring ở trên bằng cách thay đổi giá trị của thuộc tính "message" và giữ nguyên cả hai file nguồn.

### **The Spring Core Module Introduction**

#### Ê! Chào mừng trở lại, Cadet.

#### Chúc mừng bạn đã tạo ứng dụng Java Spring đầu tiên của bạn.

#### Điều đó thật tuyệt vời, phải không?

#### Bây giờ chúng ta hãy tiếp tục và tăng tốc theo dõi khóa học của chúng ta và khám phá Kiến trúc mùa xuân.

### **Architecture**

#### Như chúng ta đã biết, Spring được tổ chức theo kiểu mô-đun cho phép bạn chỉ lo lắng về các mô-đun bạn cần và bỏ qua phần còn lại.

#### Spring framework cung cấp một số tính năng cần thiết để phát triển một ứng dụng doanh nghiệp. Tuy nhiên, nó không yêu cầu các nhà phát triển tích hợp ứng dụng của họ với khung hoàn chỉnh.

#### Nhà phát triển có thể tự do lựa chọn mô-đun cần thiết. Kiến trúc mô-đun của nó cho phép tích hợp với các framework khác mà không gặp nhiều rắc rối.

### **Core Container**

#### Hãy bắt đầu với Core Container mà phần còn lại của các mô-đun được xây dựng. Nó có các mô-đun ngôn ngữ Beans, Context và Expression.

#### Nó cung cấp tính năng dependency injection, còn được gọi là Inversion of Control.

#### Mô-đun này chứa BeanFactory (một triển khai mô hình nhà máy) tạo và quản lý vòng đời của các đối tượng ứng dụng khác nhau (được gọi là đậu).

### **Context**

#### Context là một mô-đun khác được xây dựng trên cơ sở vững chắc bởi Core và Beans.

#### Mô-đun ngữ cảnh mở rộng BeanFactory và thêm hỗ trợ cho các sự kiện vòng đời ứng dụng và xác thực.

#### Nó cho phép nhiều dịch vụ doanh nghiệp như truy cập JNDI, tích hợp EJB, từ xa và lập lịch, đồng thời tạo điều kiện tích hợp dễ dàng với các khung khác.

#### Nó có một tiêu điểm được gọi là ApplicationContext.

### **Expression Language**

#### Ngôn ngữ được sử dụng để thao tác và truy vấn đồ thị đối tượng trong thời gian chạy được cho là trong Ngôn ngữ biểu thức.

#### Nó hoạt động như một phần mở rộng của ngôn ngữ biểu thức thống nhất trong đặc tả JSP (Java Server Pages) 2.1.

#### Ngôn ngữ này giúp thiết lập và nhận các giá trị thuộc tính, gọi phương thức và truy cập ngữ cảnh trong mảng, gán thuộc tính, toán tử logic và số học và truy xuất các đối tượng.

### 

### **Data Access & Web Modules Introduction**

#### Trong phần trước, chúng ta đã khám phá Spring Core Container và các khái niệm liên quan.

#### Đừng lo lắng nếu bạn không thể chọn một vài khái niệm ở đây và ở đó, từ từ khi bạn cảm thấy thoải mái với mùa xuân, tất cả những điều này sẽ giống như một cuộc dạo chơi.

#### Bây giờ chúng ta hãy đi qua lớp Truy cập / Tích hợp Dữ liệu bao gồm các mô-đun JDBC, ORM, OXM, JMS và Transaction modules.

### **Data Access/Integration**

#### Lớp này bao gồm các mô-đun OXM, Object Relational Mapping (ORM), Java Database Connectivity (JDBC), JMS và Transaction.

#### Mô-đun JDBC cung cấp một lớp trừu tượng JDBC giúp loại bỏ nhu cầu xử lý ngoại lệ lặp đi lặp lại và không cần thiết.

### **ORM and OXM module**

#### Mô-đun ORM cung cấp các lớp tích hợp cho các API ánh xạ quan hệ đối tượng phổ biến, bao gồm JPA, JDO, Hibernate và iBatis.

#### Mô-đun này cung cấp tính nhất quán / tính di động cho mã của chúng tôi bất kể công nghệ truy cập dữ liệu dựa trên các khái niệm ánh xạ hướng đối tượng.

#### OXM là viết tắt của Object XML Mappers. Nó được sử dụng để chuyển đổi các đối tượng sang định dạng XML và ngược lại.

#### Mô-đun OXM cung cấp một lớp trừu tượng hỗ trợ triển khai ánh xạ đối tượng / XML cho JAXB, Castor, XMLBeans, JiBX và XStream.

### **JMS and Transaction modules**

#### Trong gói này, các framework ánh xạ O/R có thể được sử dụng với sự kết hợp của các tính năng được hỗ trợ bởi Spring, như tính năng quản lý giao dịch khai báo đơn giản.

#### Mô-đun JMS chứa các tính năng để sản xuất và tiêu thụ tin nhắn giữa các khách hàng khác nhau.

#### Việc triển khai các giao diện đặc biệt trong quản lý giao dịch lập trình và khai báo cho các lớp chỉ được tìm thấy trong mô-đun Giao dịch của POJO.

#### Tất cả các khái niệm triển khai giao dịch cấp doanh nghiệp có thể được thực hiện trong Spring bằng cách sử dụng mô-đun Giao dịch.

#### 

### **Spring Web Modules**

#### Đó là ngôi nhà của Spring MVC framework. Lớp Web bao gồm các mô-đun Web, Web-MVC, Web-Socket và Web-Portlet.

* Web: Cung cấp các tính năng tích hợp hướng web cơ bản như chức năng tải lên tệp nhiều phần và khởi tạo vùng chứa IoC bằng cách sử dụng trình nghe Servlet và ngữ cảnh ứng dụng hướng web.
* Web-MVC: Mô-đun Web-MVC chứa triển khai Model-View-Controller (MVC) của Spring cho các ứng dụng web.
* Web-Socket: Mô-đun này cung cấp hỗ trợ cho giao tiếp hai chiều và dựa trên WebSocket giữa máy khách và máy chủ trong các ứng dụng web.
* Web-Portlet: Cung cấp triển khai MVC được sử dụng trong môi trường portlet và phản ánh chức năng của mô-đun Web-MVC.

### **Dependency Injection using Spring**

#### Trong thế giới phức tạp này, nơi chúng ta làm việc với các ngôn ngữ lập trình hàng ngày, tất cả chúng ta có xu hướng tìm kiếm các phương pháp và thủ thuật để làm cho cuộc sống và sự phát triển của chúng ta trở nên dễ dàng.

#### Dependency Injection là một trong những kỹ thuật như vậy nhằm mục đích giúp mã nhà phát triển dễ dàng bằng cách cung cấp các phụ thuộc của một đối tượng khác.

#### Trong phần này, chúng ta sẽ khám phá Dependency Injection và các khái niệm liên quan trong Spring.

### **Dependency Injection**

#### Khi bạn nghe thuật ngữ phụ thuộc, điều gì xuất hiện trong tâm trí bạn? Rõ ràng, một cái gì đó dựa vào một cái gì đó khác, phải không? Vâng, điều đó cũng tương tự, trong trường hợp phát triển mùa xuân là tốt.

#### Khái niệm cơ bản của dependency injection là bạn không tạo ra các đối tượng mà mô tả cách chúng nên được tạo.

#### Bạn không trực tiếp kết nối các thành phần và dịch vụ của mình với nhau trong mã, nhưng mô tả dịch vụ nào cần thiết cho các thành phần nào trong tệp cấu hình.

### **Classes A and B**

#### Sự phụ thuộc trong lập trình là một cách tiếp cận trong đó một lớp sử dụng các chức năng cụ thể của một lớp khác.

#### Ví dụ: Nếu bạn xem xét hai lớp A và B, và nói rằng lớp A sử dụng các chức năng của lớp B, thì nó ngụ ý rằng lớp A có sự phụ thuộc của lớp B.

#### Bây giờ, nếu bạn đang viết mã bằng Java thì bạn phải biết rằng bạn phải tạo một thể hiện của lớp B trước khi các đối tượng được sử dụng bởi lớp A.

#### Do đó, nói một cách đơn giản, đó là quá trình tạo một đối tượng cho một lớp khác và để lớp trực tiếp sử dụng sự phụ thuộc.

### **Inversion of Control (IoC)**

#### Bây giờ, bạn đã hiểu, Dependency Injection là gì, hãy hiểu nguyên tắc mà Dependency Injection dựa trên.

#### Như tên cho thấy, Inversion of Control về cơ bản được sử dụng để đảo ngược các loại trách nhiệm bổ sung khác nhau của một lớp chứ không phải là trách nhiệm chính.

#### IoC dựa vào dependency injection vì cần có một cơ chế để kích hoạt các thành phần cung cấp chức năng cụ thể.

### **Inversion of Control**

#### Hãy xem xét một ví dụ, trong đó bạn có khả năng nấu ăn.

#### Theo nguyên tắc IoC, bạn có thể đảo ngược điều khiển. Vì vậy, thay vì bạn nấu thức ăn, bạn chỉ có thể đặt hàng trực tiếp từ bên ngoài, trong đó bạn nhận thức ăn ngay trước cửa nhà.

#### Do đó, quá trình thực phẩm trực tiếp được giao cho bạn trước cửa nhà bạn được gọi là Đảo ngược kiểm soát. Bạn không cần phải tự nấu ăn, thay vào đó, bạn có thể đặt đồ ăn và để một nhân viên giao hàng phục vụ thức ăn cho bạn.

#### Bằng cách này, bạn không phải quan tâm đến các trách nhiệm bổ sung và chỉ tập trung vào công việc chính.

### **Dependency with Constructor Injection**

#### Xem xét một ứng dụng có lớp Vẽ. Trong lớp Vẽ, có một lớp Shape. Hình dạng có thể là Hình tam giác, Hình tròn, Hình chữ nhật, v.v.

public class Drawing{

private Shape shape;

public Drawing(Shape shape) {

this.shape = shape;

}

}

### **Constructor Injection**

#### Ở đây, class Drawing không quan tâm đến việc nó nên vẽ kiểu hình dạng nào.

#### Shape sẽ được triển khai độc lập và sẽ được cung cấp cho lớp Drawing tại thời điểm khởi tạo và toàn bộ quy trình này được kiểm soát bởi Spring Framework.

#### Ở đây, chúng ta đã loại bỏ total control khỏi class Drawing và giữ nó ở một nơi khác (tức là. Tệp cấu hình XML) và sự phụ thuộc (ví .class Shape) đang được đưa vào lớp Shape thông qua Class Constructor.

#### Do đó, luồng kiểm soát đã bị "**đảo ngược**" bởi Dependency Injection (DI) bởi vì bạn đã ủy thác hiệu quả các phụ thuộc cho một số hệ thống bên ngoài.

### **Dependency with Setter Injection**

#### Trong kiểu tiêm này, phương pháp phun tiêm sự phụ thuộc vào phương pháp setter mà khách hàng tiếp xúc.

public class Drawing{

private Shape shape;

public void setShape(Shape shape) {

this.shape = shape;

}

}

#### Phương thức thứ hai của injecting dependency là thông qua các phương thức setter của class Drawing, nơi chúng ta sẽ tạo một instance Shape và instance này sẽ được sử dụng để gọi các phương thức setter để khởi tạo các thuộc tính class Drawing.

### **Understanding Spring AOP Introduction**

#### Chúng ta đã thấy Dependency Injection hoạt động như thế nào, nhưng để tăng tính mô-đun của các ứng dụng Spring, một kỹ thuật mới gọi là AOP (Aspect-Oriented-Programming) đã được phát triển.

#### Bạn có thể đã nghe nói về Lập trình hướng khía cạnh hay còn gọi là AOP, trước đây. Vì vậy, bạn có thể tò mò làm thế nào AOP này hoạt động vào mùa xuân.

#### Hãy cùng tìm hiểu nhé!

### **Description of AOP**

#### "Lập trình hướng khía cạnh" là một cái tên gây tò mò. Nó xuất phát từ thực tế là chúng tôi đang thêm các khía cạnh mới vào các lớp hiện có. Đó là một sự phát triển của mẫu thiết kế trang trí.

#### Một trang trí là một cái gì đó bạn viết mã bằng tay trước khi biên dịch, sử dụng các giao diện hoặc các lớp cơ sở để nâng cao một thành phần hiện có. Đó là tất cả tốt đẹp và tốt, nhưng lập trình định hướng khía cạnh đưa điều này lên một cấp độ khác.

#### AOP cho phép bạn nâng cao các lớp học với tính linh hoạt cao hơn nhiều so với mẫu trang trí truyền thống. Bạn thậm chí có thể làm điều đó với mã của bên thứ ba.

### **What is Spring AOP?**

#### Aspect-Oriented Programming (AOP) khen ngợi OOP vì nó cung cấp tính mô-đun. Nhưng, đơn vị chính của mô-đun ở đây được coi là một khía cạnh hơn là lớp.

#### AOP mùa xuân giúp phá vỡ logic của chương trình thành nhiều phần riêng biệt được gọi là mối quan tâm.

#### Các mối quan tâm xuyên suốt giúp tăng tính mô-đun và tách nó khỏi logic kinh doanh của một ứng dụng.

#### Bạn có thể thêm / loại bỏ các mối quan tâm mà không cần biên dịch lại mã nguồn hoàn chỉnh chỉ bằng cách thay đổi các tệp cấu hình (nếu bạn đang áp dụng các khía cạnh sử dụng cấu hình XML).

### 

### **Why AOP?**

#### Chức năng quan trọng nhất của AOP là cách có thể cắm được để tự động thêm mối quan tâm bổ sung trước, sau hoặc xung quanh logic thực tế.

#### Hãy xem xét có 10 phương thức trong lớp như thể hiện trong hình trên. Đây là 5 phương pháp bắt đầu từ x trong khi 2 phương pháp bắt đầu từ y và 3 phương pháp bắt đầu từ z.

#### Ở đây, chúng ta phải duy trì nhật ký và gửi thông báo sau khi gọi các phương thức bắt đầu từ x.

#### Vì vậy, vấn đề mà không có AOP là gì?

### **x, y & z methods**

#### Ở đây, chúng ta có thể gọi các phương thức (duy trì nhật ký và gửi thông báo) từ các phương thức bắt đầu bằng x. Trong một kịch bản như vậy, chúng ta cần viết mã theo cả 5 phương thức.

#### Nhưng, trong trường hợp nếu khách hàng nói trong tương lai, tôi không phải gửi thông báo, bạn cần thay đổi tất cả các phương thức. Nó dẫn đến một vấn đề bảo trì.

#### Hãy cùng AOP tìm hiểu giải pháp nhé.

### **AOP is here…**

#### Với AOP, chúng ta không phải gọi các phương thức từ phương thức.

#### Chúng ta có thể chỉ cần xác định mối quan tâm bổ sung như duy trì nhật ký, gửi thông báo và hơn thế nữa trong phương thức của một lớp. Mục nhập của nó được đưa ra trong tệp XML.

#### Giả sử trong tương lai, nếu một máy khách nói xóa chức năng thông báo, chúng ta chỉ cần thay đổi trong tệp XML. Vì vậy, bảo trì rất dễ dàng trong AOP.

#### Đó là toàn bộ khái niệm về lý do tại sao chúng ta cần Lập trình hướng khía cạnh trong Springs.

### **Terminologies in AOP**

#### Bây giờ chúng ta hãy thảo luận về các thuật ngữ khác nhau được sử dụng trong lập trình hướng khía cạnh.

#### Lưu ý rằng các thuật ngữ này không chỉ dành riêng cho Spring AOP mà còn được sử dụng chung cho bất kỳ khung AOP nào.

* Aspect: Một khía cạnh là một lớp thực hiện các mối quan tâm về ứng dụng doanh nghiệp cắt ngang nhiều lớp, chẳng hạn như quản lý giao dịch. Các khía cạnh có thể là một lớp bình thường được cấu hình thông qua cấu hình Spring XML. Nó cũng có thể là các lớp thông thường được chú thích bằng cách sử dụng chú thích @Aspect.
* Pointcut: Pointcut là một biểu thức được khớp với các điểm nối để xác định xem lời khuyên có cần được thực hiện hay không.

### **Terminologies in AOP**

* Lời khuyên: Đây là hành động được thực hiện trước hoặc sau khi thực hiện phương pháp. Đây là một đoạn mã thực tế được gọi trong quá trình thực thi chương trình bởi Spring AOP framework.
* Joinpoint: Điều này đại diện cho một điểm trong ứng dụng của bạn, nơi bạn có thể plug-in khía cạnh AOP. Bạn cũng có thể nói, đó là vị trí thực tế trong ứng dụng mà một hành động sẽ được thực hiện bằng cách sử dụng Spring AOP framework.
* Đối tượng mục tiêu: Đây là những đối tượng mà lời khuyên được áp dụng. Trong Spring AOP, một lớp con được tạo ra tại runtime trong đó phương thức target bị ghi đè và lời khuyên được bao gồm dựa trên cấu hình của chúng.
* Proxy: Proxy là đối tượng được tạo ra bởi framework sau khi áp dụng lời khuyên trên đối tượng đích.

### **Architecture of IoC Containers Introduction**

#### Trong Spring framework, container IoC được coi là heart của toàn bộ framework.

#### Nó chịu trách nhiệm tạo ra đối tượng, nối đối tượng lại với nhau, cấu hình các đối tượng này và xử lý toàn bộ vòng đời của các đối tượng này từ khi tạo ra chúng cho đến khi chúng bị phá hủy hoàn toàn.

#### Trong phần này, chúng tôi sẽ làm rõ thêm về cách thức hoạt động của các container này.

### **IoC Container**

#### Spring IoC Container là một phần cốt lõi của Spring framework được sử dụng để quản lý application bean.

#### Nó tiêm các phụ thuộc khi một hạt đậu được tạo ra và quản lý vòng đời của đậu trong quá trình thực hiện.

#### Nó sử dụng Dependency Injection (DI) để quản lý các thành phần và các đối tượng này được gọi là Spring Beans.

#### "Kiểm soát hạt quản lý và vòng đời của nó" là một phần của container chứ không phải các lập trình viên, đó là lý do tại sao nó được đặt tên là "Inversion of Control".

### **Configuration Metadata**

#### Configuration metadata thể hiện cách bạn có thể yêu cầu Spring container khởi tạo, cấu hình và lắp ráp các đối tượng trong ứng dụng của bạn.

#### Siêu dữ liệu cấu hình này không gì khác ngoài các định nghĩa đậu. **BeanDefinition** được cung cấp dưới dạng tệp XML, cấu hình dựa trên chú thích và cấu hình dựa trên Java.

### **Spring Beans**

#### Ngoài việc được quản lý bởi các thùng chứa Spring IoC, không có gì đặc biệt về những hạt đậu này. Trong tất cả các khía cạnh khác, những hạt đậu này chỉ là đối tượng thông thường trong một ứng dụng.

#### Các beans này và các phụ thuộc của chúng với nhau được thể hiện trong siêu dữ liệu cấu hình được sử dụng bởi Spring container.

#### Nó được gọi là bean thay vì được gọi là một đối tượng hoặc thành phần vì thực tế nguồn gốc của Framework - một sự thay thế cho Enterprise JavaBeans phức tạp và nặng.

### **Types of IoC Containers**

#### Spring Framework cung cấp hai loại IoC Container.

* BeanFactory
* ApplicationContext

#### Hãy thực hành với từng container trong các phần tiếp theo.

#### Đi cùng!

### **BeanFactory Spring BeanFactory Container**

#### Spring BeanFactory container là container đơn giản nhất cung cấp hỗ trợ cơ bản cho DI.

#### **org.springframework.beans.BeanFactory** là đại diện thực tế của Spring IoC container chịu trách nhiệm chứa và quản lý Spring beans.

#### Đúng như tên gọi, Bean Factory là nơi sản xuất và tiêm hạt cà phê vào các lớp.

### **Syntax to use BeanFactory**

#### Giao diện BeanFactory là giao diện container IoC trung tâm trong Spring.

#### Trách nhiệm của nó bao gồm khởi tạo hoặc tìm nguồn cung ứng các đối tượng ứng dụng, cấu hình các đối tượng đó và lắp ráp các phụ thuộc giữa các đối tượng này.

#### **Cú pháp:**

Resource resource = new ClassPathResource(“spring configuration file”);

BeanFactory beanFactory = new XmlBeanFactory(resource);

### **BeanFactory Interface Methods**

boolean containsBean(String name): Nó kiểm tra xem nhà máy sản xuất đậu này có chứa định nghĩa đậu hoặc phiên bản singleton được đăng ký bên ngoài với tên đã cho hay không.

Object getBean(String name): Nó trả về một thể hiện, có thể được chia sẻ hoặc độc lập, của bean được chỉ định.

Class<?> getType(String name): Nó xác định loại đậu có tên đã cho.

### **BeanFactory Interface**

#### Có một số triển khai của giao diện BeanFactory đi thẳng ra khỏi hộp với Spring. Một trong những triển khai được sử dụng phổ biến nhất của BeanFactory là XMLBeanFactory.

#### XMLBeanFactory cho phép biểu diễn các đối tượng và các phụ thuộc phong phú của chúng về XML. Phải mất siêu dữ liệu cấu hình XML để tạo ứng dụng được cấu hình đầy đủ.

#### Bộ chứa BeanFactory được ưu tiên, trong đó tài nguyên bị giới hạn ở các thiết bị di động hoặc các ứng dụng dựa trên applet.

### **XmlBeanFactory**

<?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 = "com.programminghub.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

#### Mã XML trên cho thấy nội dung của cấu hình bean xml.

#### Nó có một hạt đậu duy nhất được cấu hình có một thuộc tính duy nhất theo thông báo tên. Giá trị mặc định được đặt cho thuộc tính.

### **HelloWorld.java**

#### Tiếp theo là mã HelloWorld.java được cấu hình trong tệp cấu hình.

package com.programminghub;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Printing the message : " + message);

}

}

#### Cuối cùng, đây là code cho HelloWorldMain.java sử dụng XMLBeanFactory để tạo bean HelloWorld và gọi một method trong Spring bean đã tạo.

### **Spring bean creation**

package com.programminghub;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.core.io.ClassPathResource;

public class HelloWorldMain {

public static void main(String[] args) {

XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("Beans.xml"));

HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");

obj.getMessage();

}

}

#### Trong đoạn code trên, XmlBeanFactory load siêu dữ liệu XML dựa trên biến CLASSPATH.

#### Nó sử dụng siêu dữ liệu cấu hình để tải, lắp ráp và phân phối đậu theo yêu cầu.

#### Bước tiếp theo được sử dụng để lấy bean cần thiết bằng cách sử dụng phương thức getBean() của đối tượng bean factory đã tạo.

### **Spring bean creation**

#### Cuối cùng, một phương thức trên bean (getMessage()) được gọi để có được kết quả mong muốn.

#### Phương thức này sử dụng bean ID để trả về một đối tượng chung, cuối cùng có thể được truyền đến đối tượng thực tế.

#### Khi bạn đã có đối tượng, bạn có thể sử dụng đối tượng này để gọi bất kỳ phương thức lớp nào.

#### Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và đậu, hãy để chúng tôi chạy ứng dụng. Nếu mọi thứ đều ổn với ứng dụng của bạn, nó sẽ in thông báo sau:

In thông điệp: Xin chào thế giới!

### **ApplicationContext Spring ApplicationContext Container**

#### Đây là một container khác có trong spring container bổ sung thêm chức năng dành riêng cho doanh nghiệp.

#### ApplicationContext được định nghĩa bởi interface org.springframework.context.ApplicationContext.

#### Tương tự như BeanFactory, nó có thể tải các định nghĩa đậu, dây đậu với nhau và phân phối đậu theo yêu cầu.

### **Syntax to use ApplicationContext**

#### Ngoài ra, ApplicationContext cũng có thể thực hiện nhiều chức năng doanh nghiệp hơn như giao dịch, AOP, phân giải tin nhắn văn bản từ các tệp thuộc tính và đẩy các sự kiện ứng dụng đến người nghe quan tâm.

#### **Cú pháp**:

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring configuration file");

#### Vì ApplicationContext cung cấp chức năng bổ sung bao gồm tất cả được cung cấp bởi BeanFactory, tốt hơn là sử dụng vùng chứa ApplicationContext.

#### BeanFactory vẫn có thể được sử dụng cho các ứng dụng nhẹ như thiết bị di động hoặc các ứng dụng dựa trên applet.

### **ApplicationContext implementations**

#### Các triển khai phổ biến nhất của ApplicationContext là:

FileSystemXmlApplicationContext: Nó là một loại container tải các định nghĩa của beans từ một tệp XML. Đối với điều đó, bạn sẽ có thể cung cấp đường dẫn đầy đủ của tệp cấu hình hạt XML cho một hàm tạo.

ClassPathXmlApplicationContext: Bộ chứa này tải các định nghĩa của đậu từ tệp XML. Tuy nhiên, không bắt buộc phải cung cấp đường dẫn đầy đủ của tệp XML. Chỉ có CLASSPATH phải được thiết lập đúng vì container này sẽ trông giống như một tập tin XML cấu hình Bean.

WebXmlApplicationContext: Loại container này tải tệp XML với tất cả các định nghĩa bean trong một ứng dụng web.

### **HelloWorld.java**

#### Mẫu sau đây cho thấy mã để tạo ứng dụng Hello World.

package com.programminghub;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Printing the message : " + message);

}

}

### **Understanding HelloWorldMain.java**

#### Như thể hiện trong ví dụ, trong khi sử dụng ClassPathXmlApplicationContext, không bắt buộc phải cung cấp vị trí tệp Xml hoàn chỉnh. Việc triển khai sử dụng biến CLASSPATH để xác định vị trí tệp.

#### Ở đây, bước đầu tiên là tạo một đối tượng factory trong đó framework APIClassPathXmlApplicationContext được sử dụng để tạo factory bean sau khi load file cấu hình bean từ CLASSPATH.

#### API ClassPathXmlApplicationContext() đảm nhận việc tạo và khởi tạo tất cả các đối tượng (tức là. Đậu mùa xuân được đề cập trong tệp cấu hình đậu XML).

### **Understanding HelloWorldMain.java**

#### Bước tiếp theo được sử dụng để lấy bean cần thiết bằng cách sử dụng phương thức getBean() của ngữ cảnh đã tạo.

#### Phương thức này sử dụng bean ID để trả về một đối tượng chung mà cuối cùng có thể được truyền đến HelloWorldobject thực tế.

#### Bean bắt buộc có thể được sử dụng để gọi bất kỳ phương thức lớp nào. Ở đây chúng ta gọi phương thức getMessage in thông báo cần thiết.

### **XML Configuration**

#### Cuối cùng, đây là nội dung của cấu hình bean xml.

<?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 = "com.programminghub.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

### **XML Configuration**

#### Tệp được cấu hình chỉ có một cấu hình đậu có thuộc tính đơn giản theo tên "message".

#### Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và đậu, hãy để chúng tôi chạy ứng dụng. Nếu mọi thứ đều ổn với ứng dụng của bạn, nó sẽ in thông báo sau:

In thông điệp: Xin chào thế giới!

### **Bean Scopes Introduction**

#### Xin chào, chào mừng bạn trở lại Bean. 🥜

#### Trong phần này, chúng ta sẽ tìm hiểu về các scope khác nhau mà các bạn có thể gán cho Bean hoặc một Component trong Spring Framework.

#### Trước khi tiếp tục, tôi chắc chắn rằng bạn đã có ý tưởng về Bean là gì. Nếu không, chúng ta hãy có một đoạn hồi tưởng!

### **Let’s Recall**

#### Spring bean là một đối tượng mà Spring framework quản lý tại runtime. Nó là một khối xây dựng cơ bản của bất kỳ ứng dụng Spring nào.

#### Hầu hết các code logic ứng dụng bạn viết sẽ được đặt trong Spring beans.

#### Điều quan trọng là phải hiểu rằng định nghĩa Bean chỉ là một công thức để tạo ra các thể hiện của một lớp theo định nghĩa của công thức này.

#### Công thức này sau đó có thể được sử dụng một hoặc nhiều lần trong vòng đời của ứng dụng để tạo ra một thể hiện của Bean.

#### Trước tiên chúng ta hãy hiểu vòng đời của Spring Bean.

### **Spring Bean Lifecycle Overview**

#### Vòng đời của bất kỳ đối tượng nào có nghĩa là khi nào và nó được sinh ra như thế nào, nó hoạt động như thế nào trong suốt cuộc đời của nó, và khi nào và nó chết như thế nào.

#### Tương tự, vòng đời của đậu đề cập đến thời điểm và cách hạt đậu được khởi tạo, hành động nào nó thực hiện cho đến khi nó sống, và khi nào và nó bị phá hủy như thế nào.

#### Khi chúng ta chạy chương trình thì trước hết, spring container được start. Sau đó, container tạo instance của bean theo request, và sau đó dependencies được tiêm.

#### Và cuối cùng, hạt đậu bị phá hủy khi thùng chứa lò xo được đóng lại.

### **All about Bean scope**

#### Phạm vi Bean là một trong những đặc điểm chính của cấu hình Bean trong Spring.

#### Phạm vi sẽ chỉ ra khi nào và làm thế nào đối tượng cho định nghĩa Bean sẽ được khởi tạo.

#### Mùa xuân cung cấp một số phạm vi cho đậu của bạn. Cốt lõi của framework đi kèm với hai,

* Singleton – a single instance
* Prototype – multiple instances

#### Hơn nữa, Spring đi kèm với các phạm vi đậu bổ sung dành riêng cho các ứng dụng web:

* **request**
* **session**
* **global session**

### **The Singleton Scope**

#### Phạm vi mặc định cho tất cả các loại đậu là singleton.

#### Khi một bean có phạm vi singleton, Spring chỉ tạo ra một instance và chia sẻ nó trên toàn bộ ứng dụng.

#### Phiên bản đơn này được lưu trữ trong bộ nhớ cache của các hạt đậu singleton như vậy và tất cả các yêu cầu và tham chiếu tiếp theo cho đậu có tên đó sẽ trả về đối tượng được lưu trong bộ nhớ cache.

### **Configuration file**

#### Tuy nhiên, khi bạn cần một và chỉ một thể hiện của đậu, bạn có thể đặt thuộc tính scope thành singleton trong tệp cấu hình đậu:

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

### **A simple example**

@Bean

@Scope("singleton")

public Bird theBird() {

return new Bird();

}

#### Ở đây, container tạo ra một instance duy nhất của bean đó và khi được request (theo tên hoặc type), cùng một object sẽ được trả về.

### **The Prototype Scope**

#### Trong phạm vi Prototype, Spring container sẽ tạo mới một instance của object được bean mô tả mỗi khi nó nhận được request.

#### Một trong những quy tắc cơ bản được nêu trong tài liệu của Spring là sử dụng Prototype scoped beans khi làm việc trên môi trường có quản lý phiên (Stateful) và sử dụng phạm vi Singleton khi làm việc trên các môi trường không có session management (Stateless).

### **Configuration file**

#### Để xác định phạm vi nguyên mẫu, bạn có thể đặt thuộc tính scope thành prototype trong tệp cấu hình bean:

<!-- A bean definition with prototype scope -->

<bean id = "..." class = "..." scope = "prototype">

<!-- collaborators and configuration for this bean go here -->

</bean>

#### Một ví dụ đơn giản:

@Bean

@Scope("prototype")

public Bird protoBird() {

return new Bird();

}

### **Web applications**

* request: Điều này bao gồm một định nghĩa bean cho một yêu cầu HTTP.
* session: Đây là phạm vi định nghĩa bean cho một phiên HTTP.
* global-session: Điều này bao gồm định nghĩa bean cho một phiên HTTP toàn cầu.

#### Các scope này chỉ có giá trị trong context của một ApplicationContext nhận biết web.

### **Annotation Configuration Introduction**

#### Việc tạo ra beans khá quan trọng đối với việc sử dụng Spring, cho phép chúng ta có các lớp Java sống trong bối cảnh ứng dụng.

#### Điều này có thể được sử dụng trong các beans/class khác mà không cần liên tục tạo các instance mới mỗi khi chúng ta cần sử dụng một trong các method của chúng.

#### Trước annotation, hành vi của Spring Framework phần lớn được kiểm soát thông qua cấu hình XML.

#### Ngày nay, việc sử dụng annotations cung cấp cho chúng ta những khả năng to lớn trong cách chúng ta cấu hình các hành vi của Spring Framework.

#### Trong phần này, chúng ta sẽ xem xét các annotations có sẵn trong Spring Framework.

### **Annotation wiring**

#### Annotation wiring không được bật trong Spring container theo mặc định.

#### Vì vậy, trước khi chúng ta có thể sử dụng annotation-based wiring, chúng ta sẽ cần kích hoạt nó trong tập tin cấu hình Spring của chúng ta.

#### Vì vậy hãy xem file cấu hình sau để sử dụng annotation bất kỳ trong ứng dụng Spring của các bạn.

<?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/>

<!-- bean definitions go here -->

</beans>

### **@Required annotation**

#### Chú thích này được áp dụng cho các phương thức bean setter. Hãy xem xét một kịch bản mà bạn cần thực thi một thuộc tính bắt buộc.

#### Chú thích @Required chỉ ra rằng đậu bị ảnh hưởng phải được điền vào thời điểm cấu hình với thuộc tính bắt buộc.

#### Nếu không, một ngoại lệ của loại **BeanInitializationException** được ném.

### **@Autowired annotation**

#### Chú thích này được áp dụng cho các trường, phương thức setter và hàm tạo. Chú thích @Autowired tiêm ngầm phụ thuộc đối tượng.

#### Khi bạn sử dụng @Autowired on fields và truyền các giá trị cho các field bằng cách sử dụng property name, Spring sẽ tự động gán các field với các giá trị pass.

#### Bạn thậm chí có thể sử dụng @Autowired trên các tài sản riêng, như được hiển thị bên dưới.

public class Customer {

@Autowired

private Person person;

private int type;

}

### **Type Autowiring**

#### Khi bạn sử dụng @Autowired trên các phương thức setter, Spring cố gắng thực hiện nó bằng cách gõ auto wiring trên phương thức.

#### Bạn đang hướng dẫn Spring rằng nó nên bắt đầu thuộc tính này bằng cách sử dụng phương thức setter nơi bạn có thể thêm mã tùy chỉnh của mình, như khởi tạo bất kỳ thuộc tính nào khác với thuộc tính này.

public class Customer {

private Address address;

@Autowired

public void setAddress (Address address) {

this.address = address;

}

}

### **The setter method**

#### Hãy xem xét một kịch bản trong đó bạn cần một thể hiện của lớp A, nhưng bạn không lưu trữ A trong trường của lớp. Bạn chỉ cần sử dụng A để lấy một thể hiện của B và bạn đang lưu trữ B trong trường này.

#### Trong trường hợp này, phương pháp setter tự động nối dây sẽ phù hợp hơn với bạn. Bạn sẽ không có các trường không sử dụng ở cấp lớp.

#### Khi bạn sử dụng **@Autowired** trên hàm tạo, thì việc tiêm hàm tạo sẽ xảy ra tại thời điểm tạo đối tượng. Nó nói với người xây dựng tự động dây khi được sử dụng như một hạt đậu.

#### Một điều cần lưu ý ở đây là chỉ có một hàm tạo của bất kỳ lớp đậu nào có thể mang chú thích @Autowired.

### **Person class**

@Component

public class Customer {

private Person person;

@Autowired

public Customer (Person person) {

this.person=person;

}

}

#### Kể từ mùa xuân 4.3, @Autowired trở thành tùy chọn trên các lớp với một hàm tạo duy nhất.

#### Trong ví dụ trên, Spring vẫn sẽ inject một instance của class Person nếu bạn bỏ qua annotation @Autowired.

### **@Qualifier**

#### Có thể có một tình huống khi bạn tạo ra nhiều hơn một hạt đậu cùng loại và chỉ muốn nối một trong số chúng với một thuộc tính.

#### Trong những trường hợp như vậy, bạn có thể sử dụng chú thích @Qualifier cùng với @Autowired để loại bỏ sự nhầm lẫn bằng cách chỉ định hạt đậu chính xác nào sẽ được nối dây.

#### Khi bạn cần kiểm soát nhiều hơn quá trình tiêm phụ thuộc, @Qualifier có thể được sử dụng. @Qualifier có thể được chỉ định trên các đối số xây dựng riêng lẻ hoặc tham số phương thức.

### **Two Beans**

#### Hãy xem xét một ví dụ trong đó một giao diện **BeanInterface** được thực hiện bởi hai đậu, BeanB1 và BeanB2.

@Component

public class BeanB1 implements BeanInterface {

//

}

@Component

public class BeanB2 implements BeanInterface {

//

}

#### Bây giờ, nếu Bean A autowires giao diện này, Spring sẽ không biết nên tiêm một trong hai triển khai nào.

### **@Qualifier annotation**

#### Một giải pháp cho vấn đề này là sử dụng chú thích @Qualifier.

@Component

public class BeanA {

@Autowired

@Qualifier("beanB2")

private IBean dependency;

...

}

#### Với chú thích @Qualifier được thêm vào, Spring bây giờ sẽ biết bean nào để autowire, trong đó beanB2 là tên của BeanB2.

### **What is Event Handling? Introduction**

#### **ApplicationContext** quản lý vòng đời hoàn chỉnh của đậu.

#### Chúng ta đã thấy rằng cốt lõi của Spring là ApplicationContext, chịu trách nhiệm quản lý vòng đời hoàn chỉnh của đậu.

#### Event Handling trong Spring được cung cấp interface ApplicationListener cùng với class ApplicationEvent. Do đó, mỗi khi một ApplicationEvent được xuất bản lên ApplicationContext thông báo sẽ được gửi đến Bean.

### **Built-in Events in Spring**

#### Khi đậu được tải, ApplicationContext sẽ xuất bản một số loại sự kiện nhất định.

#### Trong Spring, chúng ta có một tập hợp các sự kiện được định sẵn được xuất bản bởi ApplicationContext trong suốt vòng đời của context.

#### Ví dụ: ContextStartedEvent được phát hành khi ngữ cảnh được khởi động và ContextStoppedEvent được xuất bản khi ngữ cảnh bị dừng.

### **Working of Spring Built-in Events**

ContextRefreshedEvent: Sự kiện này được đăng khi ApplicationContext được khởi tạo hoặc làm mới. Điều này cũng có thể xảy ra bằng cách sử dụng phương thức refresh() trong giao diện ConfigurableApplicationContext.

ContextStartedEvent: Nó truyền khi ApplicationContext được bắt đầu sử dụng phương thức start() trên giao diện ConfigurableApplicationContext. Ở đây tất cả các hạt đậu trong vòng đời sẽ được khởi động lại.

### **Working of Spring Built-in Events**

ContextStoppedEvent: Sự kiện này được đăng khi ApplicationContext bị dừng sử dụng phương thức stop() trong giao diện ConfigurableApplicationContext. Bạn có thể thực hiện việc dọn dẹp cần thiết sau khi nhận được sự cố này.

ContextClosedEvent: Sự kiện này xuất bản khi ApplicationContext đóng. Đó là do việc sử dụng close() trên interface ConfigurableApplicationContext. Do đó, khi ngữ cảnh kết thúc, nó không thể được làm lại hoặc khởi động lại.

RequestHandledEvent: Đây là một sự kiện dành riêng cho web cho tất cả các đậu biết rằng các yêu cầu HTTP đã được cung cấp dịch vụ.

### **Note**

#### Các sự kiện của mùa xuân là đồng bộ (đơn luồng).

#### Vì vậy, nếu một sự kiện được xuất bản, cho đến khi và trừ khi tất cả người nhận nhận được thông báo, các quy trình sẽ bị chặn và luồng sẽ không tiếp tục.

#### Do đó, nếu xử lý sự kiện được sử dụng, cần cẩn thận khi thiết kế ứng dụng.

### **Listening to ApplicationContext Events Introduction**

#### Để lắng nghe các sự kiện theo ngữ cảnh, một bean nên triển khai giao diện ApplicationListener chỉ với một phương thức, onApplicationEvent (). Nó sẽ được thực hiện khi ứng dụng bị chấm dứt hoặc khởi động.

#### Với ví dụ làm việc trên Eclipse IDE, bạn có thể thấy sự kiện lan truyền như thế nào. Ngoài ra, cách bạn có thể đặt mã của mình để thực hiện tác vụ dựa trên các sự kiện.

### **HelloWorld.java file**

#### Hãy tạo một project với tên SpringExample và tạo một package com.programminghub dưới thư mục src trong project đã tạo.

package com.programminghub;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

### **CStartEventHandler.java file**

package com.programminghub;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStartedEvent;

public class CStartEventHandler

implements ApplicationListener<ContextStartedEvent>{

public void onApplicationEvent(ContextStartedEvent event) {

System.out.println("Done with ContextStartedEvent");

}

}

#### Ở đây class đang deploy ApplicationListener, ContextStartedEvent truyền lại khi ApplicationContext được start sử dụng phương thức start() trên interface ConfigurableApplicationContext.

#### Ở đây tất cả các hạt đậu trong vòng đời sẽ được khởi động lại.

### **CStopEventHandler.java file**

package com.programminghub;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStoppedEvent;

public class CStopEventHandler

implements ApplicationListener<ContextStoppedEvent>{

public void onApplicationEvent(ContextStoppedEvent event) {

System.out.println("Done with ContextStoppedEvent");

}

}

#### Đây là mã cho các sự kiện dừng.

#### Nó chuyển tiếp khi ApplicationContext bị chấm dứt bằng cách sử dụng phương thức stop() trên giao diện ConfigurableApplicationContext.

### **MainApp.java file**

package com.programminghub;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ConfigurableApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

// Let us raise a start event.

context.start();

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

// Let us raise a stop event.

context.stop();

}

}

#### Bối cảnh ứng dụng là tốt nhất của mùa xuân trong người giữ lớp. Giống như BeanFactory, nó có thể xếp chồng các định nghĩa đậu, nối đậu lại với nhau và thực hiện đậu theo yêu cầu.

#### Container này được đặc trưng bởi interface org.springframework.context.ApplicationContext.

### **Beans.xml**

<?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 = "com.programminghub.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

<bean id = "cStartEventHandler" class = "com.programminghub.CStartEventHandler"/>

<bean id = "cStopEventHandler" class = "com.programminghub.CStopEventHandler"/>

</beans>

#### Khi bạn đã hoàn tất việc tạo các tệp cấu hình nguồn và đậu, hãy để chúng tôi chạy ứng dụng.

Thực hiện với ContextStartedEvent Tin nhắn của bạn: Xin chào thế giới! Thực hiện với ContextStoppedEvent

### **Classification of Transaction Management Introduction**

#### Spring là một framework được sử dụng rộng rãi hiện nay, mang lại nhiều tính năng và extension mạnh mẽ cho Java core stack.

#### Tuy nhiên, hầu hết mọi người có xu hướng sử dụng các tính năng này mà không hiểu cơ chế cơ bản của chúng.

#### Trong phần này, chúng ta sẽ đi sâu vào một số tính năng của Spring liên quan đến Quản lý giao dịch.

### **Overview of Transaction**

#### Giao dịch là một đơn vị công việc trong đó tất cả các hoạt động phải thực hiện hoặc không có hoạt động nào trong số đó.

#### Để hiểu tầm quan trọng của giao dịch, hãy nghĩ về một ví dụ áp dụng cho tất cả chúng ta.

#### "**Chuyển số tiền từ tài khoản này sang tài khoản khác**" - thao tác này bao gồm ít nhất dưới hai bước dưới đây

1. Khấu trừ số dư từ tài khoản của người gửi
2. Thêm số tiền vào tài khoản của người nhận.

### **Overview of Transaction**

#### Bây giờ hãy nghĩ về tình huống số tiền được khấu trừ từ tài khoản của người gửi, nhưng không được gửi đến tài khoản người nhận do một số lỗi.

#### Các vấn đề như vậy được quản lý bởi quản lý giao dịch, trong đó cả hai bước được thực hiện trong một đơn vị công việc duy nhất trong đó cả hai bước được thực hiện thành công hoặc trong trường hợp bất kỳ ai bị lỗi, nó nên được khôi phục.

#### Quản lý giao dịch trong Spring Framework là một phần quan trọng trong các ứng dụng doanh nghiệp định hướng RDBMS để đảm bảo tính toàn vẹn và nhất quán của dữ liệu.

### **AXIT**

#### Khái niệm giao dịch có thể được mô tả với bốn thuộc tính chính sau đây được mô tả dưới dạng ACID:

Tính nguyên tử - Nó nói rằng một giao dịch có bản chất nguyên tử, tức là nó đầy đủ hoặc không.

Tính nhất quán- Thuộc tính này đảm bảo rằng dữ liệu phải ở trạng thái nhất quán sau khi giao dịch hoàn tất.

Cách ly- Có thể có nhiều xử lý giao dịch với cùng một tập dữ liệu cùng một lúc. Mỗi giao dịch nên được cách ly với các giao dịch khác để ngăn chặn tham nhũng dữ liệu.

Bền vững - Kết quả của giao dịch phải là vĩnh viễn sau khi giao dịch hoàn thành để tránh mất dữ liệu.

### **Type of Transaction Management: Global Transaction**

#### Có thể có các ứng dụng (rất khó xảy ra) nơi giao dịch có thể xảy ra giữa các cơ sở dữ liệu khác nhau. Đây được gọi là xử lý giao dịch phân tán.

#### Một giao dịch phân tán hoặc toàn cầu được thực hiện trên nhiều hệ thống và việc thực hiện nó đòi hỏi sự phối hợp giữa hệ thống quản lý giao dịch toàn cầu và tất cả các nhà quản lý dữ liệu cục bộ của tất cả các hệ thống liên quan.

#### Giao dịch này được quản lý bởi Máy chủ ứng dụng (WebSphere, Weblogic) sử dụng JTA. Ở đây mã không thể được sử dụng lại vì JTA có sẵn ở cấp máy chủ.

### **Type of Transaction Management: Local Transaction**

#### Giao dịch cục bộ xảy ra giữa ứng dụng và một RDBMS duy nhất, chẳng hạn như kết nối JDBC đơn giản. Với các giao dịch địa phương, tất cả mã giao dịch đều nằm trong mã của chúng tôi.

#### Trong cả giao dịch toàn cầu và địa phương, chúng tôi phải tự quản lý giao dịch.

#### Nếu bạn đang sử dụng JDBC, thì API quản lý giao dịch cho JDBC. Nếu bạn đang sử dụng Hibernate, hãy ngủ đông API giao dịch và JTA tại máy chủ ứng dụng cho các giao dịch toàn cầu.

#### Hầu hết các ứng dụng web chỉ sử dụng các tài nguyên duy nhất do đó nó là tùy chọn tốt nhất để sử dụng trong một ứng dụng bình thường.

### **Spring Transaction Management Support: Programmatic**

#### Spring hỗ trợ hai cách tiếp cận khác nhau để quản lý giao dịch.

#### Programmatic Transaction Management: Spring **Programmatic Transaction Management** cho phép bạn quản lý các giao dịch với lập trình trong source code của bạn. Nó cung cấp cho bạn rất nhiều sự linh hoạt mà có thể khó duy trì.

#### Đối với cách tiếp cận Programmatic, PlatformTransactionManager được sử dụng để triển khai.

### **Spring Transaction Management Support: Declarative**

#### **Declarative** Transaction Management: Spring Declarative Transaction Management cho phép bạn quản lý các giao dịch bằng cách sử dụng configuration, không giống như cách tiếp cận Programmatic.

#### Điều này có nghĩa là bạn tách quản lý giao dịch khỏi mã doanh nghiệp. Bạn chỉ sử dụng chú thích hoặc cấu hình dựa trên XML để quản lý các giao dịch.

#### Quản lý giao dịch khai báo thích hợp hơn quản lý giao dịch có lập trình mặc dù nó kém linh hoạt hơn quản lý giao dịch theo chương trình, cho phép bạn kiểm soát các giao dịch thông qua mã của mình.

#### Spring hỗ trợ quản lý giao dịch khai báo thông qua Spring AOP framework.

### **Spring Transaction Abstractions Introduction**

#### Spring framework khắc phục tất cả các vấn đề của Transaction bằng cách cung cấp một sự trừu tượng trên các API giao dịch khác nhau, do đó cung cấp một mô hình lập trình nhất quán.

#### Sự trừu tượng được định nghĩa bởi giao diện org.springframework.transaction.PlatformTransactionManager.

#### Chúng ta hãy xem đoạn mã của giao diện.

### **Interface**

public interface PlatformTransactionManager {

TransactionStatus getTransaction(TransactionDefinition definition);

throws TransactionException;

void commit(TransactionStatus status) throws TransactionException;

void rollback(TransactionStatus status) throws TransactionException;

}

#### **TransactionStatus getTransaction(TransactionDefinition** def) Phương pháp này là để trả về các giao dịch đang hoạt động hiện tại hoặc tạo một giao dịch mới.

#### **Cam kết vô hiệu (trạng thái TransactionStatus)** Phương thức này cam kết khi giao dịch nhất định được hoàn thành.

#### **Khôi phục vô hiệu (Trạng thái giao dịch)** Phương pháp này thực hiện khôi phục giao dịch nhất định.

### **The TransactionDefinition**

#### Giao diện cốt lõi của transaction support trong Spring là TransactionDefinition.

#### Bạn có thể định nghĩa nó là:

public interface TransactionDefinition {

int getPropagationBehavior();

int getIsolationLevel();

String getName();

int getTimeout();

boolean isReadOnly();

}

### **Methods**

int getPropagationBehavior() - Phương thức này trả về hành vi lan truyền của một số nguyên. Spring cung cấp tất cả các tùy chọn giao dịch tương tự như EJB CMT.

int getIsolationLevel() - Phương thức này trả về mức độ mà giao dịch được tách ra khỏi các giao dịch khác.

String getName() - Phương thức này trả về tên của giao dịch này.

int getTimeout() - Đây là một phương thức số nguyên trả về thời gian mà giao dịch sẽ được hoàn thành.

Bool isReadOnly() - Phương thức này trả về liệu giao dịch có ở chế độ chỉ đọc hay không.

### **Transaction propagation Behavior**

#### Tuyên truyền là một thuộc tính rất quan trọng trong việc thiết lập hành vi giao dịch.

#### Chúng ta hãy xem xét một số chế độ lan truyền được sử dụng phổ biến nhất.

#### TransactionDefinition.PROPAGATION\_MANDATORY Chế độ này hỗ trợ một giao dịch hiện tại. Nếu không có giao dịch hiện tại nào tồn tại, nó sẽ ném một ngoại lệ.

#### TransactionDefinition.PROPAGATION\_REQUIRED Phương thức hiện tại phải chạy trong một giao dịch hiện có và nếu không có giao dịch hiện có, nó sẽ tạo ra một giao dịch mới.

#### TransactionDefinition.PROPAGATION\_SUPPORTS Phương thức hiện tại có thể chạy trong giao dịch hiện tại, nếu nó tồn tại, nếu không thì không cần thiết phải chạy trong một giao dịch.

### **Isolation Levels**

#### Khi nhiều giao dịch trong ứng dụng đang hoạt động đồng thời trên cùng một dữ liệu có thể dẫn đến các vấn đề dưới đây.

Dirty Read: Nếu có hai giao dịch chạy đồng thời và một luồng đã đọc dữ liệu đang được cập nhật nhưng chưa được cam kết bởi một giao dịch khác và thay vì cam kết, nó sẽ khôi phục các thay đổi.

Non Repeatable read: Sự cố này xảy ra khi một giao dịch nhận được dữ liệu khác nhau cho cùng một truy vấn khi thực hiện nhiều lần. Điều này thường xảy ra nếu một giao dịch khác đang cam kết dữ liệu đồng thời.

Phantom Read: Sự cố này xảy ra khi một giao dịch chèn dữ liệu mới trong khi một giao dịch khác đang đọc dữ liệu. Trong trường hợp này, các giao dịch đọc sẽ tìm thấy dữ liệu bổ sung không có trước đó.

### **Values for Isolation Levels**

#### Trong một kịch bản lý tưởng, các giao dịch nên được cách ly hoàn toàn với nhau.

#### TransactionDefinition.ISOLATION\_DEFAULT Đây là mức cách ly mặc định.

#### TransactionDefinition.ISOLATION\_READ\_COMMITTED Mức cách ly này cũng hỗ trợ các giao dịch đến dữ liệu chỉ đọc được cam kết bởi các giao dịch khác. Có cơ hội đọc không lặp lại và đọc ảo.

#### TransactionDefinition.ISOLATION\_READ\_UNCOMMITTED Mức cách ly này cũng hỗ trợ các giao dịch đọc dữ liệu không cam kết bởi các giao dịch khác. Với sự cô lập này, có khả năng đọc màu đỏ bẩn, không thể lặp lại và đọc ảo.

### **TransactionStatus**

#### Giao diện TransactionStatus cung cấp một cách đơn giản để mã giao dịch kiểm soát việc thực hiện giao dịch và truy vấn trạng thái giao dịch.

public interface TransactionStatus extends SavepointManager {

boolean isNewTransaction();

boolean hasSavepoint();

void setRollbackOnly();

boolean isRollbackOnly();

boolean isCompleted();

}

### **Methods**

#### **boolean hasSavepoint()** Phương thức này trả về khi giao dịch này nội bộ mang một điểm lưu.

#### **boolean isCompleted()** Phương thức này trả về khi giao dịch này hoàn tất.

#### **boolean isNewTransaction()** Phương thức này trả về true trong trường hợp giao dịch hiện tại là mới.

#### **boolean làRollbackOnly()** Phương thức này trả về khi giao dịch đã được đánh dấu là chỉ khôi phục.

#### **void setRollbackOnly()** Phương pháp này đặt giao dịch là chỉ rollback.

### **Take a bow**

#### Làm tốt lắm, Cadet!

#### Chúc mừng bạn đã hoàn thành hành trình. Bạn đã làm cho nó có thể! Những nỗ lực tuyệt đối đã khiến bạn làm việc trên Springs.

#### Hãy tiếp tục học hỏi!