|  |  |
| --- | --- |
| 1 | Trình bày cách Spring MVC xử lý request |
| 2 | Trình bày cơ chế Dependency Injection trong Spring |
| 3 | Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày? |
| 4 | Framework là gì ? Framework khác Library chỗ nào ? |
| 5 | Spring Framework là gì ? |
| 6 | Lợi ích của Spring Framework? |
| 7 | Nguyên lý đảo ngược quyền điều khiển (Inversion of Control) là gì ? |
| 8 | Bean là gì? |
| 9 | Trong Spring có bao nhiêu Bean Scope? |
| 10 | @Autowire là gì? |
| 11 | @Component có ý nghĩa gì? |
| 12 | Trình bày ý nghĩa của Controller |
| 13 | Trình bày ý nghĩa của ModelAndView Interface |
| 14 | Trình bày ý nghĩa của ModelMap |
| 15 | Trình bày ý nghĩa của ViewResolver Interface |
| 16 | Phân biệt POST với PUT thường sử dụng để làm gì? |
| 17 | @RequestMapping làm gì? |
| 18 | Thuộc tính consumes trong các Request Mapping là gì ? |
| 19 | Thuộc tính produces trong các Request Mapping là gì ? |
| 20 | Trình bày cơ chế Data Binding trong Spring |
| 21 | So sánh RequestParam và PathVariable ? |
| 22 | Thymeleaf là gì? |
| 23 | Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào? |
| 24 | Formatter là gì ? Converter là gì ? |
| 25 | ORM là gì ? |
| 26 | JPA là gì? Spring Data JPA là gì ? |
| 27 | Hibernate là gì ? Hibernate làm gì với database |
| 28 | Liệt kê một số annotation của hibernate? |
| 29 | Entity là gì? |
| 30 | Persistence Context & Entity Manager là gì? |
| 31 | Câu lệnh truy vấn động là gì? Câu lệnh truy vấn tĩnh là gì? |
| 32 | Trạng thái của Entity bao gồm những gì? |
| 33 | Có bao nhiêu loại mapping trong Hibernate? |
| 34 | Để sử dụng interface JpaRepository cần cung cấp những thông tin nào? |
| 35 | Validation dữ liệu là gì? |
| 36 | Trình bày cách triển khai validate dữ liệu trong Spring |
| 37 | Binding Result là gì? |
| 38 | AOP là gì? AOP gồm những yếu tố nào? |
| 39 | Các loại Advice? |
| 40 | JoinPoint là gì? |
| 41 | Nêu cách triển khai AOP mà bạn dùng trong dự án |
| 42 | Mục đích xử lý ngoại lệ trong Spring Web MVC ? |
| 43 | Các cách xử lý ngoại lệ trong Spring Web MVC ? |
| 44 | Cookie là gì? |
| 45 | Phân biệt Session và Cookie |
| 46 | Session là gì ? Cho ví dụ về session ? |
| 47 | Nêu cách triển khai Session mà bạn dùng trong dự án |
| 48 | Web Service là gì? Lấy ví dụ về web service |
| 49 | Phân biệt Web Service và Web truyền thông |
| 50 | SOAP là gì? RESTful là gì? |
| 51 | jQuery là gì? Ajax là gì? |
| 52 | Nêu ví dụ về cách bạn triển khai Ajax trong dự án của bạn |
| 53 | @RequestHeader và @ResponseHeader có ý nghĩa gì |
| 54 | I18n và L10n là gì? |
| 55 | Nêu cách triển khai I18N và I10N trong dự án Spring |
| 56 | Interceptor là gì? |
| 57 | Bảo mật là gì? Cơ chế bảo mật trong Spring như thế nào? |
| 58 | Authentication là gì ? Authorization là gì ? |
| 59 | Các cơ chế xác thực ? |
| 60 | CSRF là gì |
| 61 | CORS là gì |
| 62 | Spring Boot là gì? |
| 63 | Sự khác nhau giữa SpringBoot và SpringMVC là gì? |
| 64 | Cấu hình cho Spring Boot Tìm kiếm các Bean ở nhiều package khác nhau bằng cách nào? |
| 65 | Trình bày Spring Security? @EnableWebSecurity làm gì? |
| 66 | Trình bày Spring Boot JPA? Cài đặt? |

***1, Trình bày cách Spring MVC xử lý request.***

+ **Luồng thực thi:**

🡪 Người dùng điền tên **website** cần truy cập trên **brower** 🡪 nhấn **Enter**

🡪 gửi **request** lên **server** (nơi triển khai ứng dụng **Spring**)

🡪 **DispatcherServlet** tiếp nhận **request**

🡪 chuyển tới **controller** tương ứng(thông qua các cơ chế **mapping –** ánh xạ đường dẫn)

🡪 **controller** nhận yêu cầu và gọi phương thức thích hợp (**GET** / **POST**)

🡪 **controller** tiến hành **render** dữ liệu và trả về tên **view** cho **DispatcherServlet**.

🡪 **DispatcherServlet** sẽ nhận trợ giúp từ **ViewResolver** để xác định **view** cho các yêu cầu.

🡪 **DispatcherServlet** chuyển dữ liệu tới **view** đó 🡪 chuyển tới trình duyệt

***2,Trình bày cơ chế Dependency Injection trong Spring***

- **DI** (**Dependency Injection**) – là **design pattern** cho phép xóa bỏ sự phụ thuộc, là một cơ chế để triển khai **IoC**, trong đó thao tác tiêm (**inject**) vào các đối tượng khác được thực hiện bởi **container** hoặc **framework**.

(Giống với Nguyên tắc **D** trong **S.O.L.I.D** là **Dependency inversion principle** - **Nguyên lý đảo ngược phụ thuộc**: các **module** cấp cao không nên phụ thuộc vào các **module** cấp thấp, cả 2 nên phụ thuộc vào **abstraction**).

=> Giảm sự phụ thuộc giữa các **module**, **object** giúp cho việc mở rộng **code** sau này được dễ dàng.

=> Giúp cho việc quản lý và kiểm thử ứng dụng dễ dàng hơn.

=> Giúp chúng ta tập trung vào việc viết **logic** nghiệp vụ của ứng dụng, còn việc tạo và quản lý các đối tượng đc giao cho **framework** lo.

🡺 **Có 3 cách triển khai:**

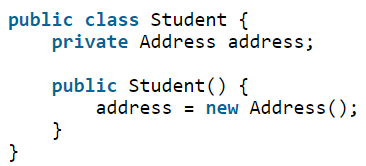
+ **Constructor** - Dùng **inject** (tiêm) các **module** bắt buộc. Các **module** được tiêm nằm trong **constructor**, và được gán lần lượt vào các **field**.

+ **Setter** method - Dùng **inject** các **module** tùy chọn. Mỗi **module** sẽ được **inject** thông qua **setter**, nằm ở tham số và cũng gán cho **field** nào đó.

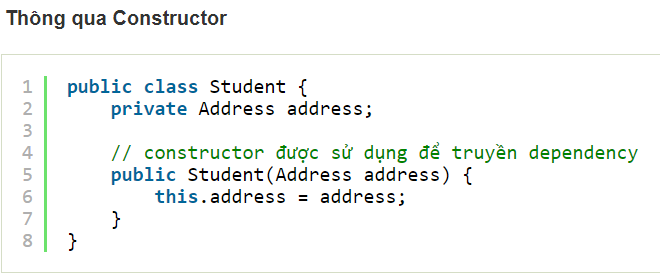
+ **Interface** (field) - Các **client** phải triển khai một **Interface** mà có một **setter** method dành cho việc nhận **dependency**.

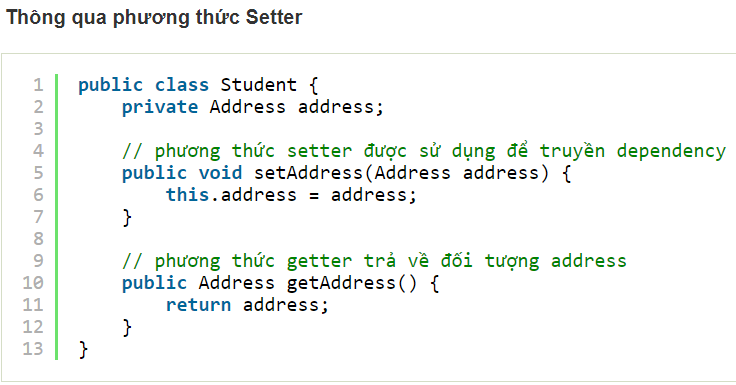
🡺 **Ví dụ:**

+ Khi không sử dụng **DI**: đối tượng **address** sẽ bị phụ thuộc vào **Student**. Mỗi khi muốn thay đổi đối tượng này thì phải thay đổi đối tượng **student**.



+ Khi sử dụng **DI**: để **student** và **address** giảm phụ thuộc vào nhau, chúng ta có thể khởi tạo đối tượng **address** ở bên ngoài lớp **Student** và truyền vào thông qua **Constructor** hoặc phương thức **Setter**.





***3,Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày?***

🡺 **Có 3 cách triển khai:**

+ **Constructor** - Dùng **inject** (tiêm) các **module** bắt buộc. Các **module** được tiêm nằm trong **constructor**, và được gán lần lượt vào các **field**.

+ **Setter** method - Dùng **inject** các **module** tùy chọn. Mỗi **module** sẽ được **inject** thông qua **setter**, nằm ở tham số và cũng gán cho **field** nào đó.

+ **Interface** (field) - Các **client** phải triển khai một **Interface** mà có một **setter** method dành cho việc nhận **dependency**.

***4,Framework là gì ? Framework khác Library chỗ nào ?***

- **Framework** – là 1 **ứng dụng phần mềm** có tính **trừu tượng** cao, cung cấp các tính năng chung thông dụng nhất và có khả năng tùy biến.

(**framework** ràng buộc, ép người dùng phải theo đúng luật chơi của nó, nó nắm quyền điều khiển nên phải code sao cho nó hiểu, ví dụ phải đứng ở trạm xe buýt thì mới xe mới đón khách,…).

- **Phân biệt Framework/Library:**

+ **Giống:** đều cung cấp tính năng viết sẵn để có thể tái sử dụng.

+ **Khác:** Điểm khác biệt chính giữa **framework** và **library** chính là cơ chế **IoC**.

|  |  |
| --- | --- |
| **Library** | **Farmework** |
| - Ứng dụng nắm quyền điều khiển (**code** của mình sẽ gọi **code** của thư viện).  - Là tập hợp các tính năng, các lớp.  - Hoạt động bị động.  - Sử dụng trực tiếp mà không cần thay đổi cấu trúc dự án.  - Không có cơ chế **IoC**. | - **Framework** nắm quyền điều khiển ứng dụng (code của **framework** sẽ gọi code của mình).  - Tập hợp các thư viện.  - Hoạt động chủ động.  - Sử dụng **framework** phải thay đổi cấu trúc dự án.  - Có cơ chế **IoC**. |

***5,Spring Framework là gì ?***

- **Spring Framework** - là một trong những **khung ứng dụng** dựa trên **Java** phổ biến nhất (đc **Rod Johnson** phát triển vào năm **2003**).

Là một **khung mã nguồn mở** có thể được sử dụng để phát triển các ứng dụng **Java** một cách dễ dàng và với tốc độ nhanh chóng.

Nó là một khung công tác nhẹ cũng cung cấp hỗ trợ cơ sở hạ tầng được xác định rõ ràng để phát triển các ứng dụng trong **Java**.

Nói cách khác, có thể nói rằng **Spring** xử lý cơ sở hạ tầng để LTV có thể tập trung hơn vào việc phát triển ứng dụng của mình.

***6,Lợi ích của Spring Framework?***

- **Lợi ích của Spring Framework:**

+ Tính đơn giản và kiểm thử tốt.

+ Đảo ngược quyền điều khiển (**IoC**).

+ Trọng lượng nhẹ (khung cơ bản **Spring** chỉ có kích thước khoảng **1MB** và chi phí xử lý của **Spring** không quá nhiều).

+ Hỗ trợ lập trình hướng khía cạnh (**AOP**).

+ Xây dựng ứng dụng **web** theo mô hình **MVC**.

+ Cấu hình và vòng đời của các đối tượng của ứng dụng được quản lý bởi **Spring framework**.

+ Xử lý ngoại lệ tốt.

***7,Nguyên lý đảo ngược quyền điều khiển (Inversion of Controller) là gì ?***

- **IoC** (**Inversion of Controller**) – “nguyên lý đảo ngược quyền điều khiển”, là **nguyên lý** trong phát triển phần mềm, trong đó việc điểu khiển các đối tượng hoặc thành phần của hệ thống đc thực hiện bởi **framework** hoặc các **container** (LTV không cần tạo đối tượng thay vào đó họ chỉ cần mô tả rằng nó sẽ được tạo như thế nào, **framework** sẽ lo phần còn lại).

***8,Bean là gì?***

- **Bean** – là những **module** **chính** của chương trình, là 1 đối tượng đc quản lý bởi **Spring IoC container** (do **framework** tự khởi tạo, tự quản lý vòng đời của đối tượng).

***9, Trong Spring có bao nhiêu Bean Scope?***

- **Có 5 Bean Scope:**

+ **Singleton** – chỉ duy nhất một thể hiện của **bean** sẽ được tạo cho mỗi **container**. Đây là **scope** mặc định cho **spring bean**.

+ **Prototype** – khác với **Singleton**, 1 thể hiện của **bean** sẽ được tạo cho mỗi lần được yêu cầu (**request**).

+ **Request** – giống với **prototype scope**, tuy nhiên nó dùng cho ứng dụng **web**, 1 thể hiện của **bean** sẽ được tạo cho mỗi **HTTP** **request**.

+ **Session** – mỗi thể hiện của **bean** sẽ được tạo cho mỗi **HTTP Session**.

+ **Global-Session** – được sử dụng để tạo **global sesion** **bean** cho các ứng dụng **Portlet**.

***10, @Autowire là gì?***

- **@Autowired** – tự động nhúng các **bean** được **Spring Container** sinh ra vào **Class** có khai báo **@Autowire**.

***11,@Component có ý nghĩa gì?***

- **@Component** – là một **annotation** của **class**. Nó dùng để đánh dấu **class Java** là một **bean**. Khi **Spring start** thì nó quét qua các **annotation** có đánh dấu là **@Component** thì nó sẽ tạo **bean** cho **class** đó.

***12, Trình bày ý nghĩa của Controller ?***

- **Ý nghĩa của Controller:**

Trong mô hình **MVC**, **controller** là nơi nhận **request** từ người dùng, xử lý **request**, xây dựng dữ liệu cho **view** (**model**) và chọn **view** để trả lại kết quả của cho người dùng.

***13, Trình bày ý nghĩa của ModelAndView Interface?***

**+ ModelAndView** – Là 1 **interface**, là sự kết hợp của 2 khía cạnh truyền **dữ liệu** và **view**.

***14, Trình bày ý nghĩa của ModelMap ?***

**+ ModelMap** – là đối tượng (1 **class**) implement **Map** **interface** trong **Java**, đc **collection** tích hợp thêm các phương thức của **map**.

=> **Model** và **ModelMap** chỉ gửi **dữ liệu**.

***15, Trình bày ý nghĩa của ViewResolver Interface ?***

+ **ViewResolver** (trình phân giải **view**) – là cơ chế ánh xạ “**tên**” của **view** sang đối tượng **view** tương ứng.

***16, Phân biệt POST với PUT thường sử dụng để làm gì?***

Trả lời :

POST: thêm mới dữ liệu

PUT: chỉnh sửa dữ liệu

***17, @RequestMapping làm gì?***

- **@RequestMapping** – đc dùng để ánh xạ các **request** tới các xử lý ở **controller** tương ứng.

***18, Thuộc tính consumes trong các Request Mapping là gì ?***

***19, Thuộc tính produces trong các Request Mapping là gì ?***

Thuộc tính:

+ **value** (URL pattern).

+ **method**: GET, POST, PUT, PATCH, DELETE,…

+ **consumes**: định dạng kiểu dữ liệu khi **client request** tới **server** (tương ứng với cài đặt “**content- type**” của **request header**).

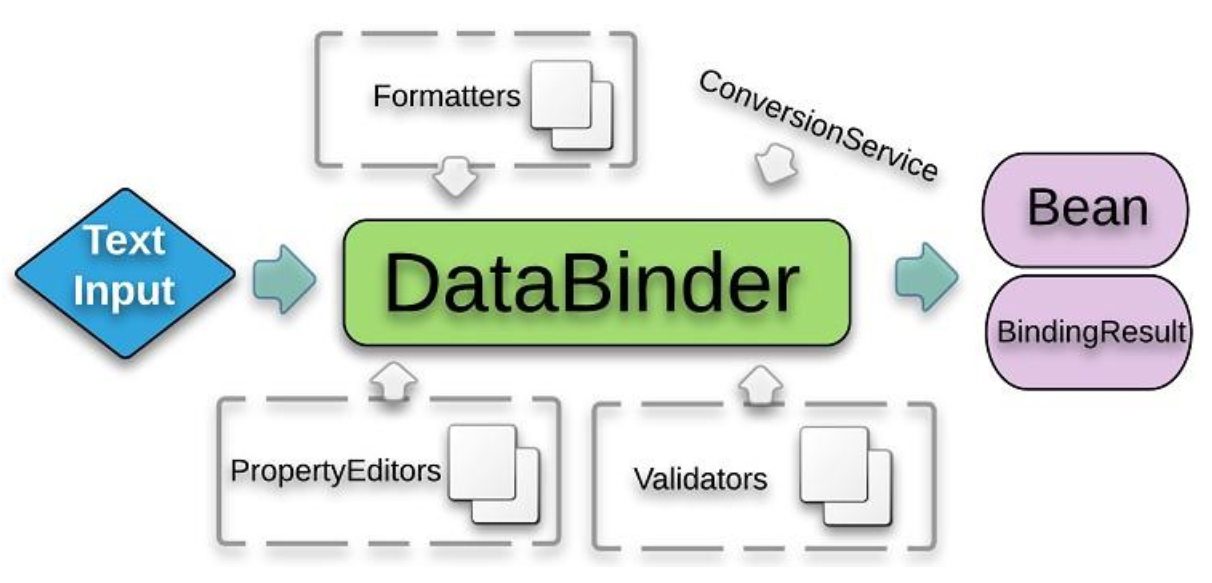
+ **produces**: định nghĩa kiểu dữ liệu trả về cho **client** (tương ứng với cài đặt “**accept**” của **request** **header**).

20, Trình bày cơ chế Data Binding trong Spring

- **Data binding** – là cơ chế liên kết dữ liệu đầu vào / đầu ra với đối tượng **model**.

+ Các **form** đều đc liên kết với 1 đối tượng biểu diễn dữ liệu phía sau 🡪 dữ liệu trên **form** sẽ đc tự động chuyển đổi thành các thuộc tính của đối tượng liên kết **form**.

+ **Data binding** hỗ trợ chuyển đổi dữ liệu và **validate**, hoạt động dựa trên **data binder**.



- **Data binder:**

+ Mỗi đối tượng như là 1 **form**, mỗi thuộc tính tương ứng với 1 ô **input**.

+ **Text input** sẽ đc **mapping** với **data binder** thông qua **propertyEditors** (thông qua **getter/setter**).

+ **ConversionService** giúp **convert** (chuyển đổi) dữ liệu 1 cách tự động.

+ **Formatters** giúp can thiệp sâu về cấu hình (ví dụ như **validate** dữ liệu ngày, giờ).

+ **Validators** giúp **validate** trực tiếp thuộc tính của đối tượng.

+ Đầu ra sẽ là 1 **Bean** 🡪 gán giá trị vào đối tượng (bắt đối tượng thông qua **ModelAttribute**).

***21, So sánh RequestParam và PathVariable ?***

|  |  |
| --- | --- |
| @RequestParam | @PathVariable |
| * Đều có thể dùng để lấy dữ liệu từ client gửi đến server | |
| * Có thể nhận dữ liệu từ cả phương thức Get hoặc Post * Dữ liệu được lấy theo cặp key-value * Dùng nhiều trong web truyền thống | * Chỉ có thể lấy dữ liệu từ phương thức Get * Chỉ cần lấy value * Dùng nhiều trong web service |

***22, Thymeleaf là gì?***

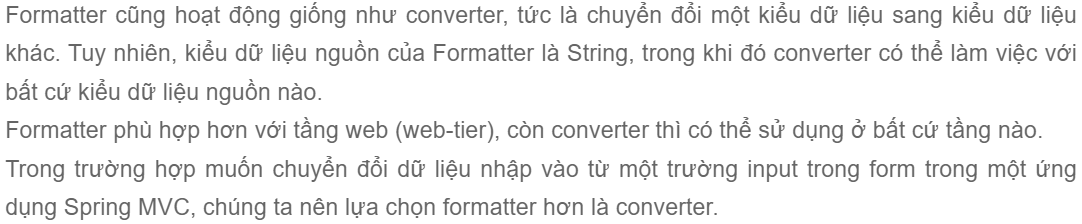
- **Thymeleaf** – là 1 bộ xử lý **view**, đc sử dụng cho các ứng dụng **web** và các ứng dụng độc lập.

+ Được sử dụng cho phù hợp với tiêu chuẩn **web**, hỗ trợ **HTML 5**.

+ **Thymeleaf** cho phép hỗ trợ các **template**: HTML, XML, JS, CSS, TEXT,...

***23, Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào?***

***24, Formatter là gì ? Converter là gì ?***



***25, ORM là gì ?***

- **ORM** (**Object-Relational Mapping**) – là kỹ thuật liên kết giữa đối tượng trong lập trình với đối tượng trong **CSDL** (là quá trình biến đổi giữa 1 **record** trong **DB** và 1 **object** trong ngôn ngữ lập trình **OOP**).

+ **ORM** ra đời để khắc phục nhược điểm của **JDBC**, như:

- Cấu hình thủ công (cần phải chuẩn bị **DB** trước).

- Có kiến thức nền tảng về **SQL**.

- Phải biết kiểu dữ liệu đầu ra / đầu vào của từng **column**.

+ **ORM** có rất nhiều triển khai (Hibernate, Eclipselink, Ibatis,…), trong đó **Hibernate** đc xem là **ORM Framework** nổi tiếng nhất và sử dụng rộng rãi trong **Java**.

***26, JPA là gì? Spring Data JPA là gì ?***

- **JPA** (**Java Persistence API**) – là 1 bộ đặc tả tiêu chuẩn của **Java**, trong đó mô tả các thao tác quản lý dữ liệu trong **Java** cung cấp các mô hình cho phép ánh xạ các bảng trong **CSDL** với các **class**.

(**JPA** chỉ chứa **interface** hướng dẫn triển khai **ORM**, việc **implement** sẽ do **ORM framework** thực hiện).

- **Spring Data Jpa** – là 1 phần của dự án **Spring Data**, có thể sử dụng **Spring Data JPA** với các **ORM** **framework** như Hibernate, Open JPA,…

Là tầng được xây dựng phía trên tầng **JPA**. Có nghĩa là nó sẽ sử dụng tất cả các chức năng của **JPA** như **entity mapping**, **JPA query**, xây dựng một số tính năng như khởi tạo **query** thông qua tên hàm,…

🡺 Giúp cải tiến **JPA** tiêu chuẩn, đơn giản hóa tầng truy xuất dữ liệu, tự tạo **repository**, tạo các truy vấn **JPA** thông quá tên phương thức.

***27,Hibernate là gì ? Hibernate làm gì với database ?***

- **Hibernate** - là một **ORM Framework** mã nguồn mở giúp **LTV** viết ứng dụng **Java** có thể **mapping** các **Object** với hệ **CSDL**, và hỗ trợ thực hiện các khái niệm **OOP** với **CSDL**.

***28, Liệt kê một số annotation của hibernate?***

* @Entity, @Table, @Id, @GenerateValue, @Column, @OneToOne, @OneToMany, @ManyToOne, @ManyToMany

***29, Entity là gì?***

- **Entity** – là đối tượng đại diện cho dữ liệu trong ứng dụng.

+ **Entity** thường là **POJO** (Plain Old Java Object), nghĩa là lớp đơn giản, chỉ mô tả dữ liệu, không có xử lý logic nghiệp vụ.

+ Mỗi **Entity** đc **mapping** với 1 **table** trong **CSDL** (**mapping** 1-1).

+ Mỗi **Entity** cần phải tuân thủ:

- Được gắn **annotation** **@Entity**, **@Id**.

- Có **Constructor** là **public** không tham số.

- Không được khai báo **final**.

- Thuộc tính có **access modifier** là **private**, **protected**, hoặc ở mức **package private** (tức là không phải **public**).