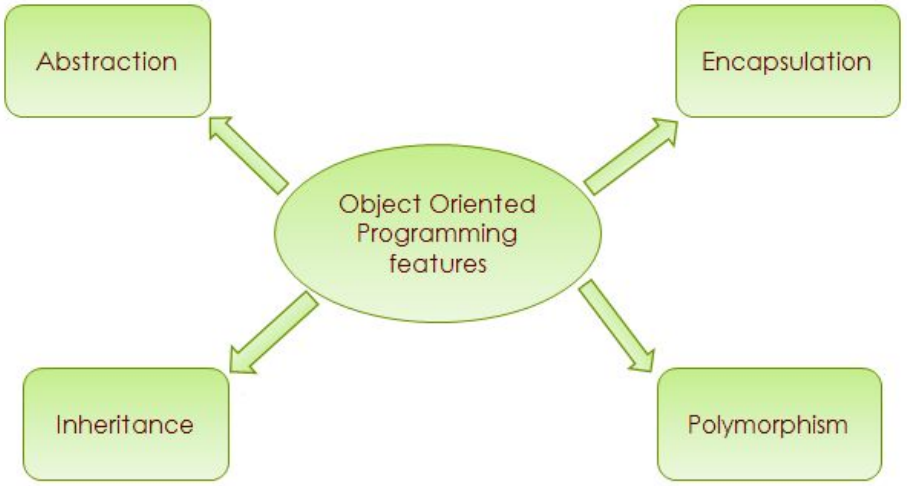
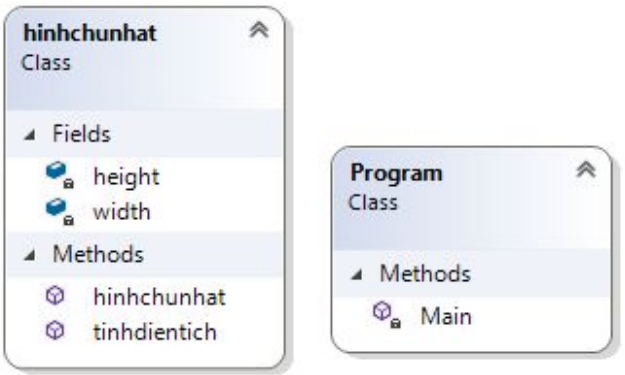
**4 đặc tính của lập trình hướng đối tượng (Object oriented program)**

Lập trình hướng đối tượng quá quen thuộc rồi bạn nào học lập trình đều phải học, đi phỏng vấn cũng vậy hỏi suốt(chắc cái này tùy vào vị trí tuyển dụng chủ yếu junior chắc chắn sẽ hỏi).nó là nền tảng cho hầu hết các design pattern hiện nay.Bài viết này đúc rút kinh nghiệm thực tế và độ hiểu của mình về OOP. Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code để trừu tượng hóa các đối tượng thực tế trong cuộc sống.




**Tính đóng gói (Encapsulation):**

Là cách để che dấu những tính chất xử lý bên trong của đối tượng, những đối tượng khác không thể tác động trực tiếp làm thay đổi trạng thái  chỉ có thể tác động thông qua các method public của đối tượng đó. Mình sẽ tạo ra 2 class để thể hiện điều này:



xem cách thể hiện bằng code dưới đây : class hinhchunhat

```
using System;
namespace oop
{
    class hinhchunhat
    {
        private int height;
        private int width;

        public hinhchunhat(int newHeight, int newWidth) {
            height = newHeight;
            width = newWidth;
        }

        public int tinhdientich() {
            return height * width;
        }
    }
}
```

- height và width ở đây chính là các tính chất (properties) của đối  tượng class hinhchunhat
- tinhdientich() là method được public nhằm mục đích tương tác với các đối tượng khác. Tạo một class Program với method static để run, xem cách tương tác và thay đổi tính chất  của đối tượng thông qua các method public như nào:

```
using System;
namespace oop
{
    class Program
    {
        static void Main(string[] args)
        {
            //thay doi properties (height, width) cua doi tuong thong qua method public
            hinhchunhat hcn = new hinhchunhat(10, 5);
```

```
            //lay du lieu thong qua method public
            Console.WriteLine("Dien tich cua {0} la: " +
hcn.tinhdientich(),hcn);
            Console.ReadLine();
        }
    }
}
```

Output:

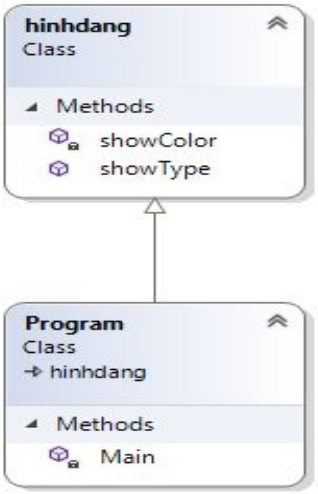

Như vậy khi ta muốn thay đổi các tính chất (properties) không thể tương tác trực tiếp với properties mà phải thông qua các method public được định nghĩa bên trong class

- không thể biết luồng xử lý logic bên trong của đối tượng

**Tính kế thừa (Inheritance):**

Là kỹ thuật cho phép kế thừa lại những tính năng mà một đối tượng khác đã có, giúp tránh việc code lặp dư thừa mà chỉ xử lý công việc tương tự.

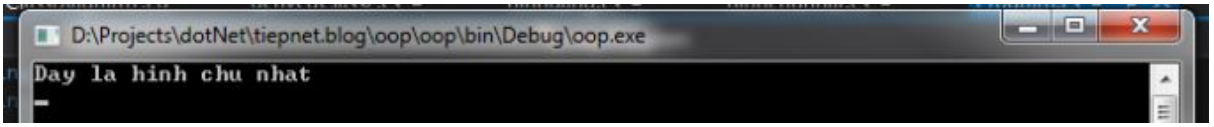
- Kế thừa một cấp (Single level Inheritance): Với một class cha và một class con
-



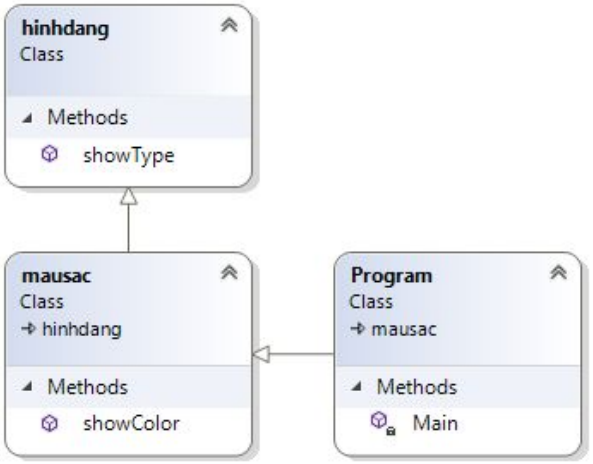

```
using System;
namespace oop
{
    class hinhdang
    {
        private void showColor()
        {
            Console.WriteLine("Mau hong");
        }
        public void showType()
        {
            Console.WriteLine("Day la hinh chu nhat");
        }
    }
}

using System;
namespace oop
{
    class Program : hinhdang
    {
        static void Main(string[] args)
        {
            Program pg = new Program();
```

```
            pg.showType();
            //pg.showColor(); khong the truy cap private method
            Console.ReadLine();
        }
    }
}
```

Output:




Trong class Program không hề có method showType() nhưng vẫn có thể truy cập sử dụng nó bằng cách kế thừa lại method của class hinhdang Kế thừa nhiều cấp (Multiple level Inheritance): Kế thừa  nhiều class.




Ở diagram trên mình viết thêm class mausac và chuyển private  method showColor() sang class mausac  thành public method.

```
using System;
namespace oop
{
    class hinhdang
```

```
    {
        public void showType()
        {
            Console.WriteLine("Day la hinh chu nhat");
        }
    }
}
using System;
namespace oop
{
    class mausac : hinhdang
    {

        public void showColor()
        {
            Console.WriteLine("Mau hong");
        }
    }
}
using System;
namespace oop
{
    class Program : mausac
    {
        static void Main(string[] args)
        {
            Program pg = new Program();
            pg.showType();
            pg.showColor();
            Console.ReadLine();
        }
    }
}
```

Output:


class Program chỉ kế thừa class mausac nhưng vẫn có thể truy cập method showType() được viết trong class hinhdang, đây chính là hình thức kế thừa nhiều cấp,rất tiện đúng không?

**Tính đa hình (Polymorphism ):**

Là một đối tượng thuộc các lớp khác nhau có thể hiểu cùng một thông điệp theo cách khác nhau.

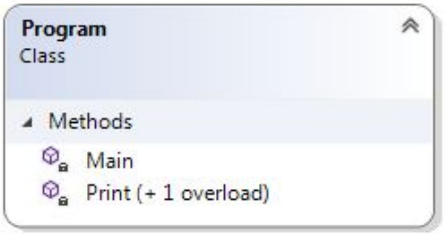
Ví dụ đa hình trong thực tế: Mình có 2 con vật: chó, mèo hai con vật này khi nhận được mệnh lệnh  là "hãy kêu" thì chó kêu "gâu gâu",  mèo kêu "meo meo".

Ví dụ trên cả 2 con vật đều hiểu chung một thông điệp "hãy kêu" và thực hiện theo cách riêng của chúng.

Trong code để thể hiện tính đa hình có 2 cách:

1. Method Overloading (compile time polymorphism)
2. Method Overriding (run time polymorphism)

- Method Overloading : là cách nạp chồng các method có cùng tên nhưng khác tham số




Tạo 1 class Program gồm 2 method Print() có tham số khác nhau (hai method này được gọi là medthod overloading)

- `using System;`
- `namespace oop`
- `{`

```
    class Program
    {
        static void Print(object o)
        {
            Console.WriteLine("Object overload called");
        }
        static void Print(string a)
        {
            Console.WriteLine("String overload called");
        }

        static void Main(string[] args)
        {
            object o = "hello";
            Print(o);
            Console.ReadLine();
        }
    }
```

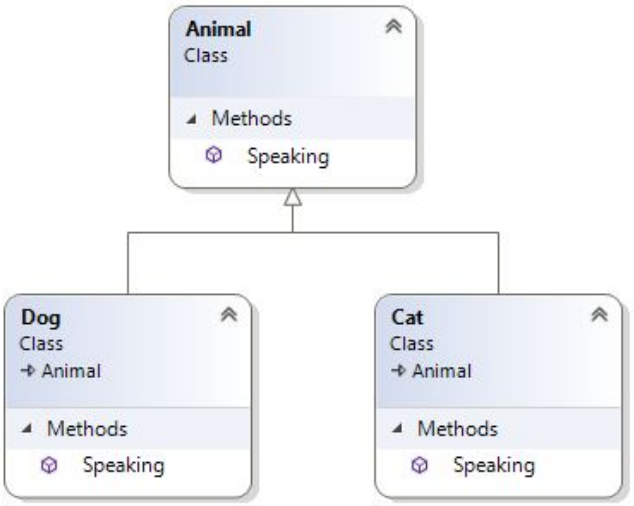
Output:


Ở đây Print(object) được gọi vì o là loại object, để xác định method nào được gọi ra chúng xác định bằng loại param hoặc số lượng param chuyền vào.

- Method Overriding:  Đây là một phương pháp được ghi đè lại các method ảo của một lớp cha nào đó(được khai báo bằng từ khóa virtual).

Để thể hiện phương pháp này cần dùng 2 từ khóa:

- virtual :từ khoá dùng để khai báo 1 phương thức ảo (có thể ghi đè được).
- override: từ khoá dùng để đánh dấu phương thức ghi đè lên phương thức của lớp cha.




Tạo ra 3 class Animal,Dog,Cat

```
using System;
namespace oop
{
    class Animal
    {
        public void Speak()
        {
            Console.WriteLine("Animal is speaking...");
        }
    }
}
using System;
namespace oop
{
    class Dog: Animal
    {

        public new void Speak(){
```

```
            Console.WriteLine("Dog speaks go go");
        }
    }
}
using System;
namespace oop
{
    class Cat:Animal
    {
        public new void Speak()
        {
            Console.WriteLine("Cat speaks meo meo");
        }
    }
}
using System;
namespace oop
{
    class Program
    {

        static void Main(string[] args)
        {
            Animal dog = new Dog();
            dog.Speaking();
            Animal cat = new Cat();
            cat.Speaking();
            Console.ReadLine();
        }
    }
}
```

Output:


Như bạn có thể thấy ở trên mình không dùng đến 2 từ khóa là virtual và override, mặc dù mình khởi tạo lớp Dog và sử dụng method Speak() kết quả in ra dữ liệu của method trong lớp Animal như vậy không thể hiện được tính đa hình.

Sau đây mình sẽ sử dụng 2 từ khóa virtual và override:

```
using System;
namespace oop
{
    class Animal
    {
        public virtual void Speak()
        {
            Console.WriteLine("Animal is speaking...");
        }
    }
}
using System;
namespace oop
{
    class Dog: Animal
    {
        override
        public void Speak(){
            Console.WriteLine("Dog speaks go go");
        }
    }
}
using System;
namespace oop
{
```

```
    class Cat : Animal
    {
        override
        public void Speak()
        {
            Console.WriteLine("Cat speaks meo meo");
        }
    }
}
using System;
namespace oop
{
    class Program
    {
        static void Main(string[] args)
        {
            Animal dog = new Dog();
            dog.Speak();
            Animal cat = new Cat();
            cat.Speak();
            Console.ReadLine();
        }
    }
}
```

Output:


Khi sử dụng virtual và override kết quả đã thể hiện được tính đa hình, lúc này từ khóa override đã được ưu tiên và ghi đè phương thức ảo từ lớp cha, khi các đối tượng gọi chung phương thức Speak() nó sẽ được trỏ tới phương thức tương ứng của mỗi đối tượng được khởi tạo.

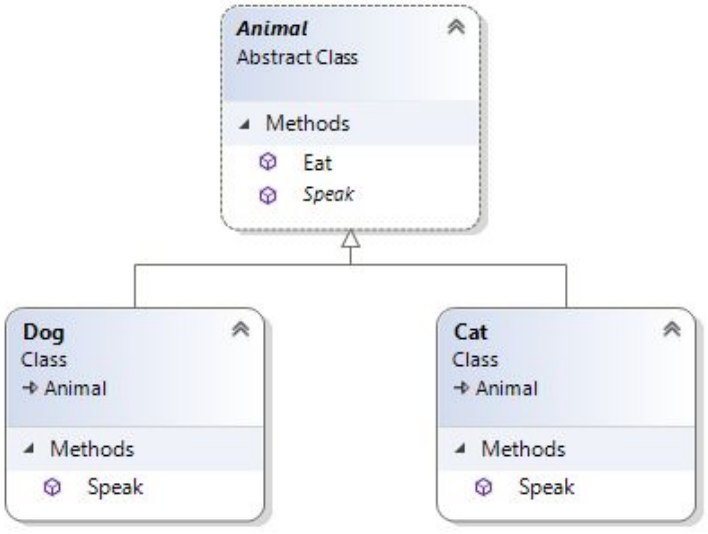
**Tính trừu tượng(Abstraction):**

Là phương pháp trừu tượng hóa định nghĩa lên những hành động, tính chất của loại đối tượng nào đó cần phải có. Ví dụ khi bạn định nghĩa một lớp động vật(Animal), Animal thì có rất nhiều loại, làm sao để xác định đó là một loại động vật? lúc này bạn sẽ hình dung trong đầu động vật có những tính chất hành vi cơ bản nhất định phải có như ăn, nói khi bất kỳ một developer nào định viết một đối tượng thuộc lớp động vật sẽ kế thừa lại lớp Animal có 2 hành vi ăn, nói,đối tượng được tạo ra có thể khác nhau như chó hoặc mèo nhưng đều có những hành vi của động vật là ăn và nói. => Trong ví dụ trên nhìn vào hành vi ăn và nói của chó và mèo ta có thể khẳng định nó thuộc lớp động vật. Vậy chốt lại rõ làng tính trừu tượng ở đây sinh ra chủ yếu để trừu tượng hóa và định nghĩa các tính chất hành vi phải có để xác định đó là đối tượng gì dựa vào tính chất hành vi của đối tượng. => Các method trừu tượng đều rỗng không thực hiện bất kỳ hành vy nào, hành vy sẽ được triển khai cụ thể do các đối tượng kế thừa. => Viết xong đoạn trên không biết các bạn đọc có hiểu không @@ thấy có vẻ lan man quá, vì chưa có kinh nghiệm @@, nói chung định nghĩa một phần phải thực hành nhiều coder mà, thực hành nhiều tự các bạn sẽ hiểu ra ).

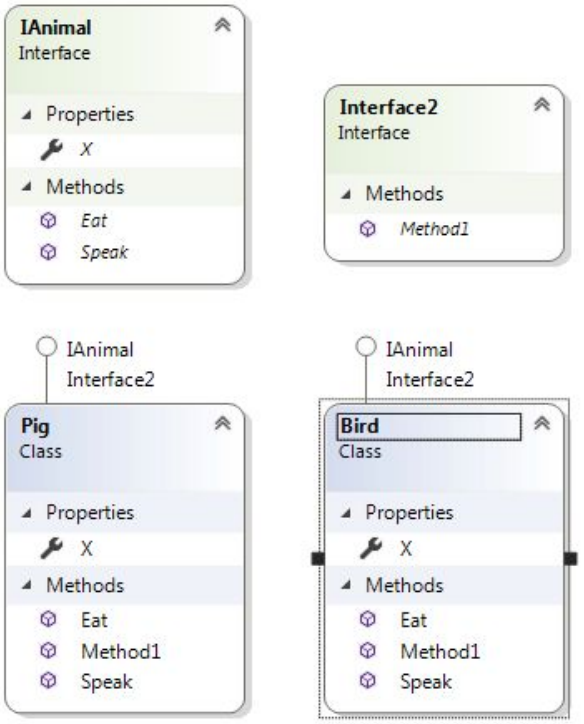
Tiếp tục nhé ^^ trong c# có 2 phương pháp để triển khai tính trừu tượng này:

1. Abstract class

- trong abstract class có 2 loại method:
    - abstract method (là method rỗng không thực hiện gì)
- method thường (là vẫn có logic trả về data hoặc thực thi hành động nào đó, nó được sử dụng cho mục đích dùng chung)



Abstract class Animal và 2 class Dog, Cat kế thừa lại những method được public 2. Interface : Khá giống với abstract class nhưng interface không phải là class, trong interface chỉ  có khai báo những method/properties trống không có thực thi, thực thi sẽ được thể hiện trong các lớp kế thừa, interface giống như một cái khung mẫu để các lớp implement và follow.




Tạo 2 interface IAnimal,Interface2 và 2 class Pig,Bird kế thừa 2 interface

Trước mình học đến phần này thấy rất khó hiểu thằng abstract class và interface không biết khi nào dùng interface hay abstract class sự khác nhau của nó là gì, có abstract class rồi còn sinh ra interface làm gì? chi tiết hơn mình viết trong bài: Khác nhau giữa abstract class và interface khi nào dùng chúng bài này mình chỉ viết về 4 đặc tính của OOP thôi, viết dài quá nhìn lại không muốn đọc rồi .

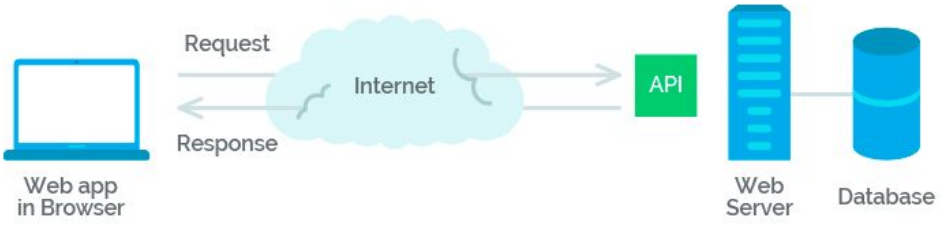
=> Nói chung phải thực hành nhiều làm vài dự án thực tế, sau bài này tạo luôn mấy cái demo về chức năng gì đó(thêm, sửa, xóa sinh viên chẳng hạn làm theo mô hình này) mới hiểu rõ được tự trải nghiệm bao giờ cũng hơn. => Chốt lại bài viết này là đúc rút từ kinh nghiệm thực tế của mình mấy cái hình ảnh diagram hay code đều là tự mình cắt ra trên visual 2017 không phải ảnh mạng nhé trừ cái ảnh đầu tiên, nên sẽ có những sai sót các bạn cứ đóng góp ý kiến gạch đá thoải mái nhé, mình sẽ còn update ^^

# HIỂU RÕ HƠN VỀ RESTFUL API

Trong bài viết này chúng ta sẽ cố gắng đi tìm hiểu và giải thích được **RESTful API** là gì?, bên cạnh đó mình cũng sẽ chia sẻ những kinh nghiệm khi làm việc với REST. Chúng ta sẽ cùng khám phá ý nghĩa của API, về HTTP, học về REST, xem nó hoạt động thế nào, và đưa ra một ví dụ đơn giản về thiết kế kiến trúc của RESTful API. (tham khảo khóa học **xây dựng RESTful API với NodeJS**)

**Một chút lý thuyết**

API (application programming interface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. Chính cái tên đã nói lên ý nghĩa của nó, nhưng chúng ta hãy cùng đi sâu hơn vào chi tiết. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML. Trong bài viết này chúng ta sẽ tập trung xung quanh JSON thôi nhé.




Hãy cùng nhìn vào một ví dụ. Có thể bạn thấy quen thuộc với những ứng dụng web như GitHub, Facebook, Google,… Nó có những API riêng để cho chúng ta sử dụng, một trong số đó giúp chúng ta lấy được thông tin về người dùng, repositores của họ và rất rất nhiều thứ hữu ích khác nữa khi bạn dùng đề xây dựng ứng dụng của mình. Bạn có thể sử dụng và thao tác với dữ liệu này bên trong ứng dụng của mình.

Dưới đây là một ví dụ của một request chuẩn tới một API, nó sẽ trông như thế này:

```
➜ ~ curl https://api.github.com/orgs/MLSDev
```

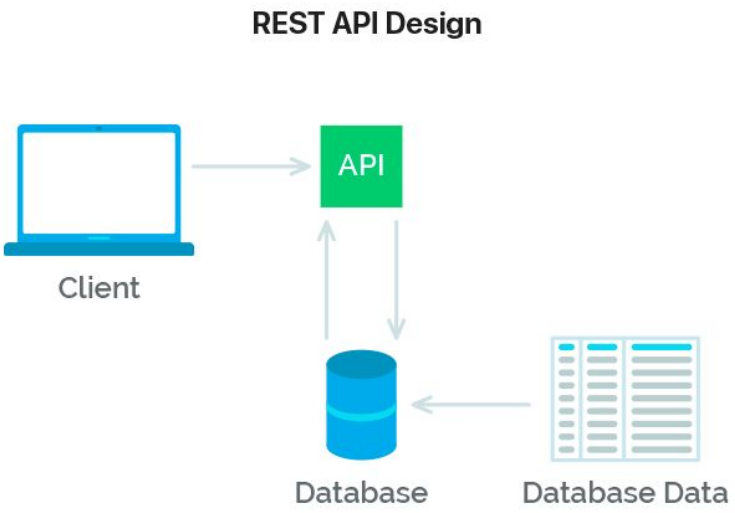
Request sẽ được thực hiện thông qua CURL (xem thêm CURL). Ngoài ra chúng ta còn có các công cụ tiện ích khác giúp chúng ta thực hiện một request như Postman, REST Client,… (Các bạn có thể cài thông qua Chrome web app).

Dưới đây là kết quả trả về mà server trả về sau khi nhận được request, gói response sẽ là:


```
{
  "login": "MLSDev",
  "id": 1436035,
  "url": "https://api.github.com/orgs/MLSDev",
  "repos_url": "https://api.github.com/orgs/MLSDev/repos",
  "events_url": "https://api.github.com/orgs/MLSDev/events",
  "hooks_url": "https://api.github.com/orgs/MLSDev/hooks",
  "issues_url": "https://api.github.com/orgs/MLSDev/issues",
  "members_url": "https://api.github.com/orgs/MLSDev/members{/member}",
  "public_members_url": "https://api.github.com/orgs/MLSDev/public_members{/member}",
  "avatar_url": "https://avatars.githubusercontent.com/u/1436035?v=3",
  "description": "",
  "name": "MLSDev",
  "company": null,
  "blog": "http://mlsdev.com/",
  "location": "Ukraine",
  "email": "hello@mlsdev.com",
  "public_repos": 25,
  "public_gists": 0,
  "followers": 0,
  "following": 0,
  "html_url": "https://github.com/MLSDev",
  "created_at": "2012-02-14T08:35:16Z",
  "updated_at": "2015-10-29T13:48:13Z",
  "type": "Organization"
}
```


**REST là gì?**

Và bây giờ, chúng ta sẽ cùng phân tích về REST. REST là từ viết tắt cho **RE**presentational **S**tate **T**ransfer. Khái niệm này có thể được diễn giải bằng câu từ như sau: dữ liệu sẽ được truyền tải và trình bày cho client side dưới định dạng nào đó (JSON). Có một trong những điểm chính mà bạn cần phải nhớ: REST không phải là một chuẩn hay một giao thức, đây là một cách tiếp cận, một kiểu kiến trúc để viết API.




**Sâu hơn về REST**

Một web service là một tập hợp các giao thức và chuẩn được sử dụng cho mục đích trao đổi giữa ứng dụng và hệ thống. Các ứng dụng phần mềm được viết bởi các ngôn ngữ khác nhau và chạy bởi các nền tảng khác nhau có thể sử dụng web service để trao đổi dữ liệu qua mạng máy tính như internet theo các cách tương tự như trao đổi trên một máy tính.

Web service dựa trên các kiến trúc REST được biết như RESTful webservice . Những webservice này sử dụng phương thức HTTP để triển khai các định nghĩa kiến trúc REST. Một RESTful web service thường được định nghĩa một URI (kiểu như đường dẫn), Uniform Resource Identifier như một service (dịch vụ).

**RESTful hoạt động như thế nào?**

Khi làm việc với server sẽ gồm 4 hoạt động thiết yếu là:

- Lấy dữ liệu ở một định dạng nào đó (JSON)
- Tạo mới dữ liệu
- Cập nhật dữ liệu
- Xóa dữ liệu

REST hoạt động chủ yếu dựa vào giao thức HTTP (xem thêm HTTP). Mỗi trong 4 hoạt động cơ bản trên sẽ sử dụng những phương thức HTTP riêng (HTTP method):

- GET: lấy dữ liệu
- POST: tạo mới
- PUT: cập nhật (thay đổi)
- DELETE: Xóa dữ liệu

Những phương thức (hoạt động) này thường được gọi là CRUD (xem thêm CRUD) tương ứng với Create, Read, Update, Delete – Tạo, Đọc, Sửa, Xóa.

*Dưới đây là bảng tương quan giữa phương thức HTTP, CRUD và các lệnh SQL.*

| HTTP | POST | GET | PUT | DELETE |
|---|---|---|---|---|
| SQL | INSERT | SELECT | UPDATE | DELETE |
| CRUD | CREATE | READ | UPDATE | DELETE |

Tất cả các request mà bạn (client side) thực hiện đều cũng sẽ có mã trạng thái HTTP (HTTP status code). Có rất nhiều status code và sẽ được chia ra thành 5 lớp. Chữ số đầu tiên cho biết một status code thuộc vào lớp nào:

- 1xx: hàm ý mang thông tin.
- 2xx: hàm ý thành công
- 3xx: hàm ý điều hướng
- 4xx: hàm ý là có lỗi từ phía client side
- 5xx: hàm ý là có lỗi phía máy chủ (server)

Các bạn có thể tham khảo thêm về các status code của HTTP tại đây.

**Kiến trúc thiết kế của RESTful**

Tất cả tài nguyên (resources) trong REST là thực thể (entities). Có thể độc lập như:

- GET /users—lấy danh sách các người dùng;
- GET /users/123—lấy thông tin một người dùng có id = 123;
- GET /posts—lấy tất cả bài post.

Cũng sẽ có những thực thể độc lập dựa vào những thực thể khác:

- GET /users/123/projects – Lấy tất cả projects của user với id = 123

Các ví dụ trên cho thấy rằng GET lấy thông tin thực thể mà client side đã request. Một request thành công sẽ được trả về dữ liệu liên quan tới thực thể và kèm theo status code là 200 (OK). Nếu có lỗi, bạn sẽ nhận lại status code 404 (Not Found), 400 (Bad Request) hoặc 5xx (Server Error).

Cùng chuyển qua phương thức POST (tạo mới một thực thể):

- POST /users.

Khi tạo một thực thể mới, bạn sẽ truyền dữ liệu vào trong request body. Ví dụ:

{

“first_name”: “Vasyl”,

“last_name”: “Redka”

}

Sau khi gửi request lên server, bạn sẽ nhận được kết quả trả về có thể là status code 201 (Created), hàm ý tạo mới thành công. Ví dụ, response trả về sẽ là dữ liệu của một thực thể vừa được tạo:

{

“id”: “1”,

“first_name”: “Vasyl”,

“last_name”: “Redka”

}

Request tiếp theo là PUT. Được dùng để cập nhật thực thể. Khi bạn gửi request thì body cũng cần phải bao gồm dữ liệu cần được cập nhật liên qua tới thực thể.

- PUT /users/123 – cập nhật người dùng với id = 123

Sự thay đổi cần phải được chỉ ra là cập nhật cho thực thể nào, được truyền vào thông qua các tham số. Nếu được cập nhật thành công, sẽ trả về mã 200 (OK) và dữ liệu của thực thể vừa được cập nhật.

Request cuối cùng là DELETE. Nó rất là dễ hiểu và được dùng để xóa một thực thể cụ thể được chỉ định thông qua tham số.

- DELETE /users/123 – xóa một user với id = 123

Nếu xóa thành công thì sẽ trả về status **200 (OK)** cùng với response body bao gồm thông tin về trạng thái của thực thể. Ví dụ, khi bạn không xóa thực thể từ database mà chỉ đánh dấu là đã được xóa, status code sẽ luôn trả về là 200 (OK) và response body với trạng thái. DELETE có thể trả về status **204 (No Content)** và không kèm theo response body.

Nếu bạn xóa thực thể trong database luôn thì status code cho request thứ 2 sẽ trả về là **404 (Not Found)** bởi vì thực thể đã được xóa và không thể truy cập được nữa.

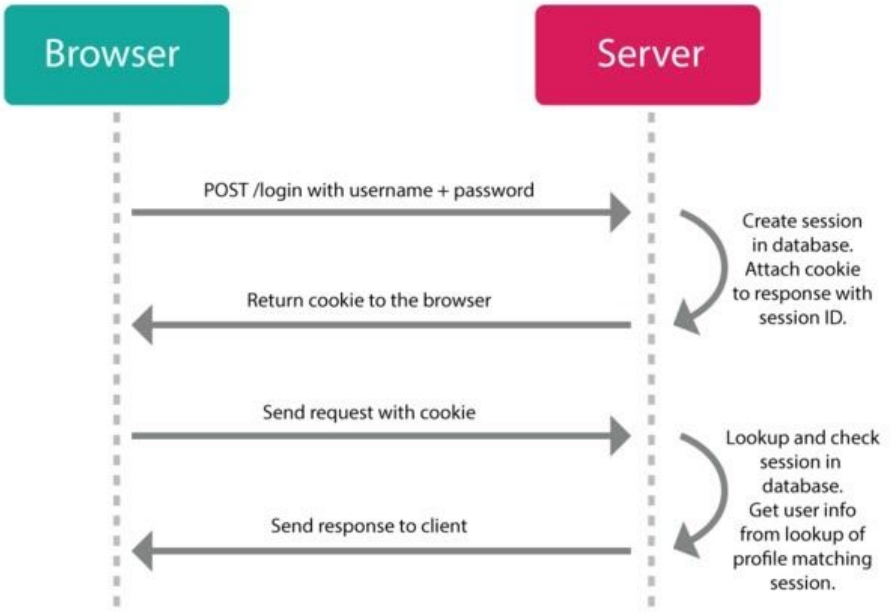
Như vậy là các bạn có thể hiểu được phần nào về **API**, **REST** và **RESTful**. Hi vọng những kiến thức này sẽ giúp ích cho các bạn thật nhiều trong con đường trở thành một lập trình viên chuyên nghiệp. Hãy cùng chia sẻ bài viết này và tiếp tục theo dõi những bài viết bổ ích khác từ **laptrinhvien.io** nhé.

# Session là gì? Cookie là gì?


Session, Cookie là 2 khái niệm được nhắc đến thường xuyên đối với các lập trình web. Khi mới bắt đầu tập làm web, nhiều bạn chắc hẳn rất hay nhầm lẫn giữa các khái niệm này. Sẽ có lúc bạn không biết lúc nào thì mình nên dùng session, lúc nào mình nên dùng cookie.

## Session là gì?

**Session** là một phiên làm việc là một khái niệm phổ biến được dùng trong lập trình web có kết nối với database. Đặc biệt các chức năng như đăng nhập, đăng xuất người dùng sẽ khó có thể thực hiện được nếu không sử dụng **session**.




## Cách sử dụng Session

Một session bắt đầu khi client gửi request đến server, nó tồn tại xuyên suốt từ trang này đến trang khác trong ứng dụng web và chỉ kết thúc khi hết thời gian timeout hoặc khi bạn đóng ứng dụng. Giá trị của session sẽ được lưu trong một file trên server.

Ví dụ khi bạn đăng nhập vào một trang web và đăng nhập với tài khoản đã đăng ký trước đó. Server sau khi xác thực được thông tin bạn cung cấp là đúng thì nó sẽ sinh ra một tập tin chứa dữ liệu cần lưu trữ của người dùng.

Với mỗi session sẽ được cấp phát một định danh duy nhất SessionID. Khi kết thúc một phiên làm việc và bắt đầu một phiên mới, dĩ nhiên bạn sẽ được cấp một SessionID khác với trước đó. Bạn có thể tuỳ ý quyết định xem nên lưu

trữ những thông tin nào vào Session. Nhưng thông thường chúng ta chỉ nên lưu những thông tin tạm thời trong **session.**

## Cookie là gì?

Giống như **session**, **cookie** cũng được dùng để lưu những thông tin tạm thời. Nhưng tập tin cookie sẽ được truyền từ server tới browser và được lưu trữ trên máy tính của bạn khi bạn truy cập vào ứng dụng.

## Cách sử dụng Cookie

**Cookie** thường được tạo ra khi người dùng truy cập một website, **cookie** sẽ ghi nhớ những thông tin như tên đăng nhập, mật khẩu, các tuỳ chọn do người dùng lựa chọn đi kèm. Các thông tin này được lưu trong máy tính để nhận biết người dùng khi truy cập vào một trang web.

Khi người dùng truy cập đến một trang web có sử dụng **cookie**, web server của trang đó sẽ tự động gửi **cookie** đến máy tính của người dùng. Khi truy cập đến các trang web sử dụng được **cookie** đã lưu, những **cookie** này tự động gửi thông tin của người dùng về cho chủ của nó (người tạo ra **cookie**). Tuy nhiên những thông tin do **cookie** ghi nhận không được tiết lộ rộng rãi, chỉ có website chứa **cookie** mới có thể xem được những thông tin này.

Mỗi cookie thường có khoảng thời gian timeout nhất định do lập trình viên xác định trước. Những thông tin được lưu vào cookie ví dụ như thông tin đăng nhập, thao tác người dùng, tần xuất ghé thăm website, thời gian truy

cập… Tất cả chúng đều là những thông tin mang tính tạm thời và được lưu trong 1 khoảng thời gian.

**Cookie** được xem là một thành phần không thể thiếu được với những website có khối lượng dữ liệu lớn, có số lượng người dùng đông, và có những chức năng đi kèm với thành viên đăng ký.

**Đánh cắp Cookie bằng cách lợi dụng lỗi bảo mật XSS**

# So sánh giữa Cookie và Session

| | **Cookie** | **Session** |
|---|---|---|
| | **Cookie** được lưu trữ trên trình duyệt của người dùng. | **Session** không được lưu trữ trên trình duyệt. |
| | Dữ liệu **cookie** được lưu trữ ở phía client. | Dữ liệu **session** được lưu trữ ở phía server. |
| | Dữ liệu **cookie** dễ dàng sửa đổi hoặc đánh cắp khi chúng được lưu trữ ở phía client. | Dữ liệu **session** không dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ. |
| | Dữ liệu **cookie** có sẵn trong trình duyệt đến khi expired. | Sau khi đóng trình duyệt sẽ hết phiên làm việc (session) |

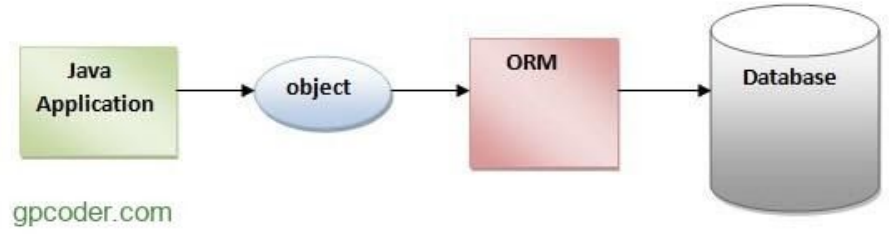
# ○ Hibernate

**Hibernate Framework là gì?**

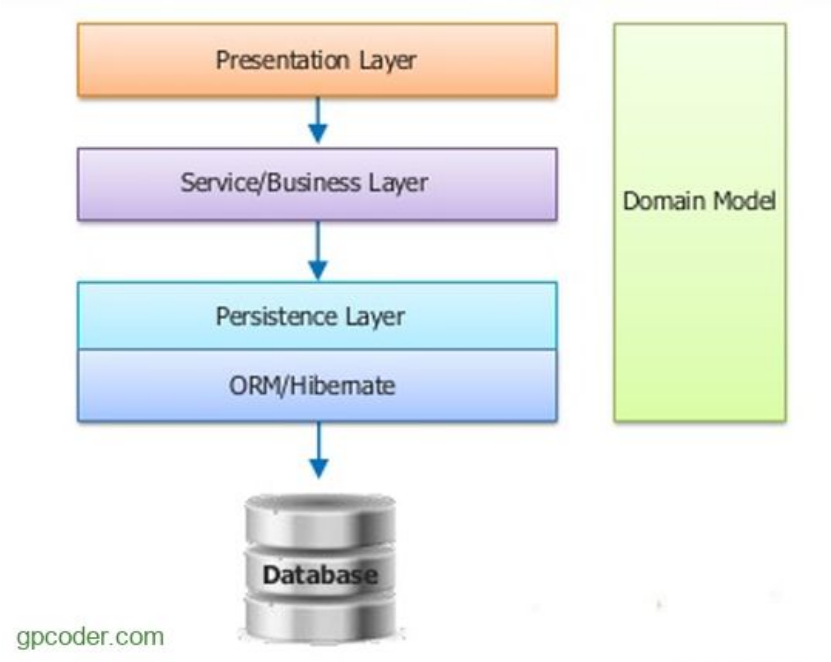
**ORM**

Như chúng ta đã biết, **ORM** (Object Relational Mapping) framework là một cơ chế cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng. Lập trình viên hoàn toàn không quan tâm đến loại database sử dụng SQL Server, MySQL, PostgreSQL, ...

ORM giúp đơn giản hoá việc tạo ra dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đó là một kỹ thuật lập trình để ánh xạ đối tượng vào dữ liệu được lưu trữ trong cơ sở dữ liệu.




**Peristence layer**



Một ứng dụng có thể được chia làm 3 phần như sau: giao diện người dùng (**presentation layer**), phần xử lý nghiệp vụ (**business layer**) và phần chứa dữ liệu (**data layer**). Cụ thể ra, business layer có thể được chia nhỏ thành 2 layer con là business logic layer và persistence layer.

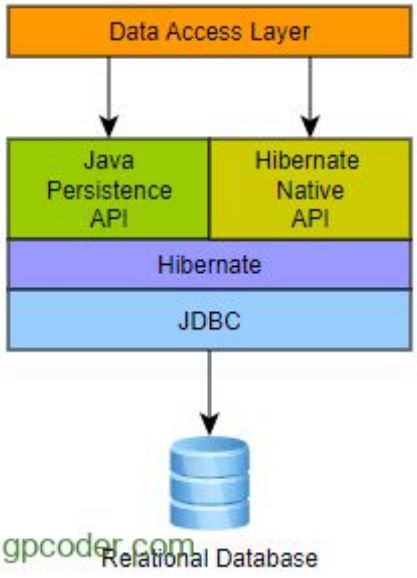
- **Business logic layer**: các tính toán logic nhằm thỏa mãn yêu cầu người dùng.
- **Persistence layer**: chịu trách nhiệm giao tiếp với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ – Relational DBMS). Persistence sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.

#### Hibernate

**Hibernate** là một trong những ORM Framework. Hibernate framework là một framework cho persistence layer. Như vậy, nhờ có Hibernate framework mà giờ đây khi bạn phát triển ứng dụng bạn chỉ còn chú tâm vào những layer khác mà không phải bận tâm nhiều về persistence layer nữa.

Hibernate giúp lập trình viên viết ứng dụng Java có thể map các object (POJO) với hệ quản trị cơ sở dữ liệu quan hệ (database), và hỗ trợ thực hiện các khái niệm lập trình hướng đối tượng với cơ dữ liệu quan hệ.

Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh. Hibernate cho phép bạn truy vấn dữ liệu thông qua Java Persistence API (JPA) hoặc bằng ngôn ngữ SQL mở rộng của Hibernate (HQL) hoặc bằng SQL thuần (Native SQL).



Hibernate vốn là một thư viện sinh ra để làm việc với mọi loại database, nó không phụ thuộc vào bạn chọn loại database nào. Nếu **Java** là "Viết 1 lần chạy mọi nơi" thì **Hibernate** là "Viết 1 lần chạy trên mọi loại database".

### Lợi ích của Hibernate Framework

Hibernate Framework có các lợi ích như dưới đây:

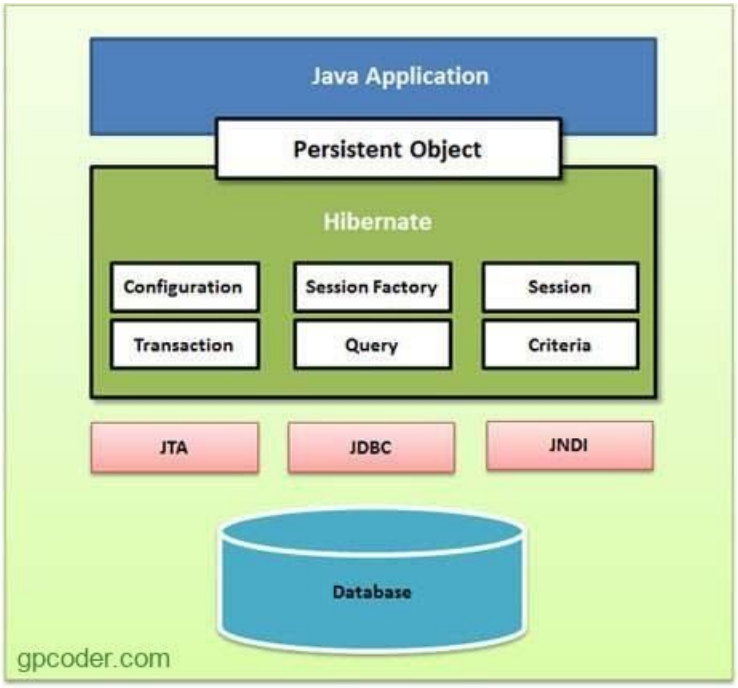
- **Mã nguồn mở và nhẹ**: Hibernate Framework là mã nguồn mở có giấy phép LGPL và nhẹ.
- **Hiệu suất nhanh**: Hiệu suất của Hibernate Framework là nhanh bởi vì bộ nhớ cache được sử dụng trong nội bộ Hibernate Framework. Có hai loại bộ nhớ cache trong Hibernate Framework, gồm bộ nhớ cache cấp một (first level cache) và bộ nhớ cache cấp hai (second level cache). Bộ nhớ cache cấp một được enable mặc định.
- **Truy vấn cơ sở dữ liệu độc lập**: HQL (Hibernate Query Language) là phiên bản hướng đối tượng của SQL. Nó tạo ra các truy vấn cơ sở dữ liệu độc lập. Vì vậy, bạn không cần phải viết các truy vấn cơ sở dữ liệu cụ thể. Trước Hibernate, nếu dự án có cơ sở dữ liệu bị thay đổi, chúng ta cần phải thay đổi truy vấn SQL dẫn đến risk và dễ gây lỗi.
- **Tạo bảng tự động**: Hibernate framework cung cấp phương tiện để tạo ra các bảng cơ sở dữ liệu một cách tự động. Vì vậy, không cần phải tốn công sức tạo ra các bảng trong cơ sở dữ liệu thủ công.
- **Đơn giản lệnh join phức tạp**: Có thể lấy dữ liệu từ nhiều bảng một cách dễ dàng với Hibernate framework.
- **Cung cấp thống kê truy vấn và trạng thái cơ sở dữ liệu**: Hibernate hỗ trợ bộ nhớ cache truy vấn và cung cấp số liệu thống kê về truy vấn và trạng thái cơ sở dữ liệu.

### Database

Hibernate hỗ trợ hầu hết tất cả RDBMS chính, chẳng hạn: Oracle, Microsoft SQL Server, PostgreSQL, MySQL, ...

### Kiến trúc Hibernate

Kiến trúc Hibernate bao gồm nhiều đối tượng như đối tượng persistent, session factory, transaction factory, connection factory, session, transaction, ...




#### Persistence object

Chính là các POJO object map với các table tương ứng của cơ sở dữ liệu quan hệ. Nó như là những container chứa dữ liệu từ ứng dụng để lưu xuống database, hay chứa dữ liệu tải lên ứng dụng từ database.

#### Configuration

Là đối tượng Hibernate đầu tiên bạn tạo trong bất kỳ ứng dụng Hibernate nào và chỉ cần tạo một lần trong quá trình khởi tạo ứng dụng. Nó đại diện cho một tập tin cấu hình hoặc thuộc tính yêu cầu của Hibernate. Đối tượng Configuration cung cấp hai thành phần chính:

- **Database Connection**: Thao tác này được xử lý thông qua một hoặc nhiều tệp cấu hình được Hibernate hỗ trợ. Các tệp này là **hibernate.properties** và **hibernate.cfg.xml**.
- **Class Mapping Setup**: Thành phần này tạo ra kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

#### Session Factory

Là một interface giúp tạo ra session kết nối đến database bằng cách đọc các cấu hình trong Hibernate configuration.

SessionFactory là đối tượng nặng (heavy weight object) nên thường nó được tạo ra trong quá trình khởi động ứng dụng và lưu giữ để sử dụng sau này.

SessionFactory là một đối tượng luồng an toàn (Thread-safe) và được sử dụng bởi tất cả các luồng của một ứng dụng.

Mỗi một database phải có một session factory. Vì vậy, nếu bạn đang sử dụng nhiều cơ sở dữ liệu thì bạn sẽ phải tạo nhiều đối tượng SessionFactory. Giả sử ta sử dụng MySQL và Oracle cho ứng dụng Java của mình thì ta cần có một session factory cho MySQL, và một session factory cho Oracle.

#### Hibernate Session

Một session được sử dụng để có được một kết nối vật lý với một cơ sở dữ liệu. Đối tượng Session là nhẹ và được thiết kế để được tạo ra instance mỗi khi tương tác với cơ sở dữ liệu. Các đối tượng liên tục được lưu và truy xuất thông qua một đối tượng Session.

Các đối tượng Session không nên được mở trong một thời gian dài bởi vì chúng thường không phải là luồng an toàn (thread-unsafe) và chúng cần được tạo ra và được đóng khi cần thiết.

Mỗi một đối tượng session được Session factory tạo ra sẽ tạo một kết nối đến database.

#### Transation

Một Transaction đại diện cho một đơn vị làm việc với cơ sở dữ liệu và hầu hết các RDBMS hỗ trợ chức năng transaction. Các transaction trong Hibernate được xử lý bởi trình quản lý transaction và transaction (từ JDBC hoặc JTA).

Transaction đảm bảo tính toàn vẹn của phiên làm việc với cớ sở dữ liệu. Tức là nếu có một lỗi xảy ra trong transaction thì tất cả các tác vụ thực hiện sẽ thất bại.

Transaction là một đối tượng tùy chọn và các ứng dụng Hibernate có thể chọn không sử dụng interface này, thay vào đó quản lý transaction trong code ứng dụng riêng.

#### Query

Các đối tượng Query sử dụng chuỗi truy vấn SQL (Native SQL) hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu và tạo các đối tượng.

#### Criteria

Đối tượng Criteria được sử dụng để tạo và thực hiện các tiêu chí truy vấn để lấy các đối tượng từ database.

### Tại sao nên dùng Hibernate thay vì JDBC

#### Object Mapping

Với JDBC ta phải map các trường trong bảng với các thuộc tính của Java object một cách "thủ công". Với Hibernate sẽ hỗ trợ ta map một cách "tự động" thông qua các file cấu hình map XML hay sử dụng các anotation.

JDBC sẽ map Java object với table như sau:

```
//rs là ResultSet trả về từ câu query get dữ liệu bảng user.
List<User> users = new ArrayList<>();
while(rs.next()) {
     User user = new User();
     user.setId(rs.getInt("id"));
     user.setUsername(rs.getString("username"));
     user.setPassword(rs.getString("password"));
     user.setCreatedDate(rs.getDate("createdDate"));
     users.add(user);
 }
```

Cũng với table user đó sử dụng các anotaion để Hibernate có thể map một cách “tự động” như sau.

```
@Entity
@Table(name = "user")
@Data
public class UserModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

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

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

    @NotEmpty
    @Column(name = "createdDate")
    private Date createdDate;
}
```

#### HQL

Hibernate cung cấp các câu lệnh truy vấn tương tự SQL, HQL của Hibernate hỗ trợ đầy đủ các truy vấn đa hình như, HQL "hiểu" các khái niệm như kế thừa (inheritance), đa hình (polymorphysm), và liên kết (association)

#### Database Independent

Code sử dụng Hibernate là độc lập với hệ quản trị cơ sở dữ liệu, nghĩa là ta không cần thay đổi câu lệnh HQL khi ta chuyển từ hệ quản trị CSDL MySQL sang Oracle, hay các hệ quản trị CSDL khác... Do đó rất dễ để ta thay đổi CSDL quan hệ, đơn giản bằng cách thay đổi thông tin cấu hình hệ quản trị CSDL trong file cấu hình.

```
//used MySQL

com.mysql.jdbc.Driver



// used Oracle

oracle.jdbc.driver.OracleDriver
```

Ví dụ khi ta muốn lấy 10 bản ghi dữ liệu của một table từ 2 CSDL khác nhau.

Với **JDBC** ta có câu truy vấn như sau.

```
# MySQL

SELECT column_name FROM table_name ORDER BY column_name ASC LIMIT 10;



# SQL Server

SELECT TOP 10 column_name FROM table_name ORDER BY column_name ASC;
```

Với **Hibernate** câu truy vấn không thay đổi với cả 2 CSDL.

```
Session.CreateQuery("SELECT E.id FROM Employee E ORDER BY E.id
ASC").SetMaxResults(10).List();
```

### Minimize Code Changes

Khi ta thay đổi (thêm) cột vào bảng, với **JDBC** ta phải thay đổi những gì:

- Thêm thuộc tính vào POJO class.
- Thay đổi method chứa câu truy vấn "select", "insert", "update" để bổ sung cột mới.
- Có thể có rất nhiều method, nhiều class chứa các câu truy vấn như trên.

Với **Hibernate** ta chỉ cần:

- Thêm thuộc tính vào Entity class.
- Cập nhật Hibernate Annotation để map column – property..

### Lazy Loading

Với những ứng dụng Java làm việc với cơ sở dữ liệu lớn hàng trăm triệu bản ghi, việc có sử dụng Lazy loading trong truy xuất dữ liệu từ database mang lại lợi ích rất lớn.

Ví dụ những file tài liệu do người dùng upload được lưu ở bảng document. Bảng user có quan hệ một-nhiều với bảng document. Trong trường hợp này class User là class cha, class Document là class con. Bảng document nhanh chóng đầy lên theo thời gian. Mỗi khi ta lấy thông tin user và document tương ứng từ database giả sử dữ liệu document là rất lớn, để ứng dụng không bị chậm vì phải mất nhiều bộ nhớ để chứa toàn bộ document của toàn bộ user, ta áp dụng Lazy loading cho từng user như sau.

```
// Declaring fetch type for one to many association in your POJO

@OneToMany(mappedBy = "user", fetch = FetchType.LAZY)

private Set documents = new HashSet();



// To fetch user with document use initialize() method as follows

User user = (User)session.get(User.class, new Integer(100));
```

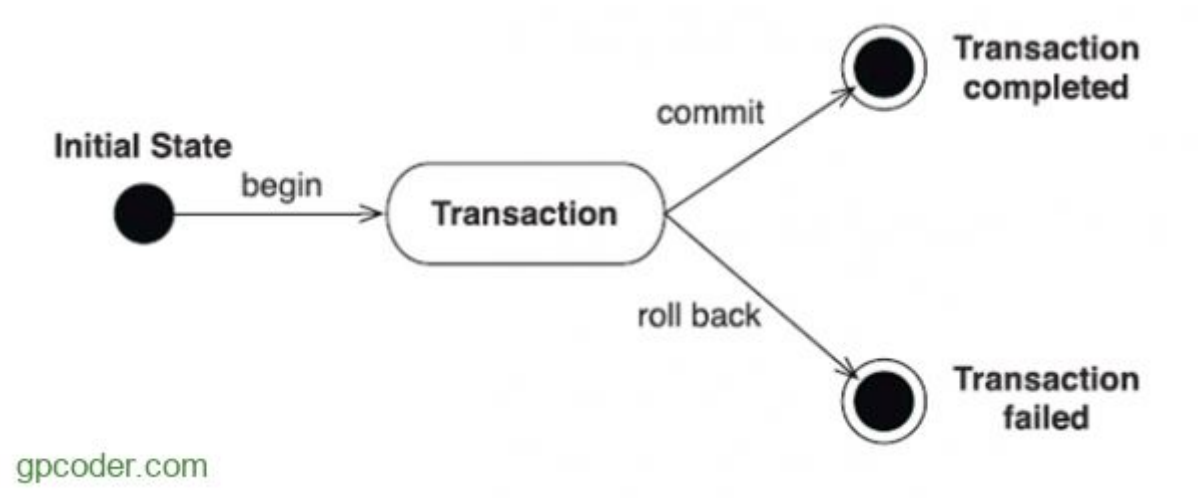
```
// This code will fetch all products for user 100 from database 'NOW'

documents = user.getDocuments();
```

**Loại bỏ Try-Catch Blocks**

Sử dụng JDBC nếu lỗi xảy khi tao tác với database thì sẽ có exception SQLexception ném ra. Bởi vậy ta phải sử dụng try-catch block để xử lý ngoại lệ.

Hibernate xử lý việc này giúp bạn bằng cách nó override toàn bộ JDBC exception thành Unchecked Exception và ta không cần viết try-catch trong code của mình nữa.

**Quản lý commit/ rollback Transaction**

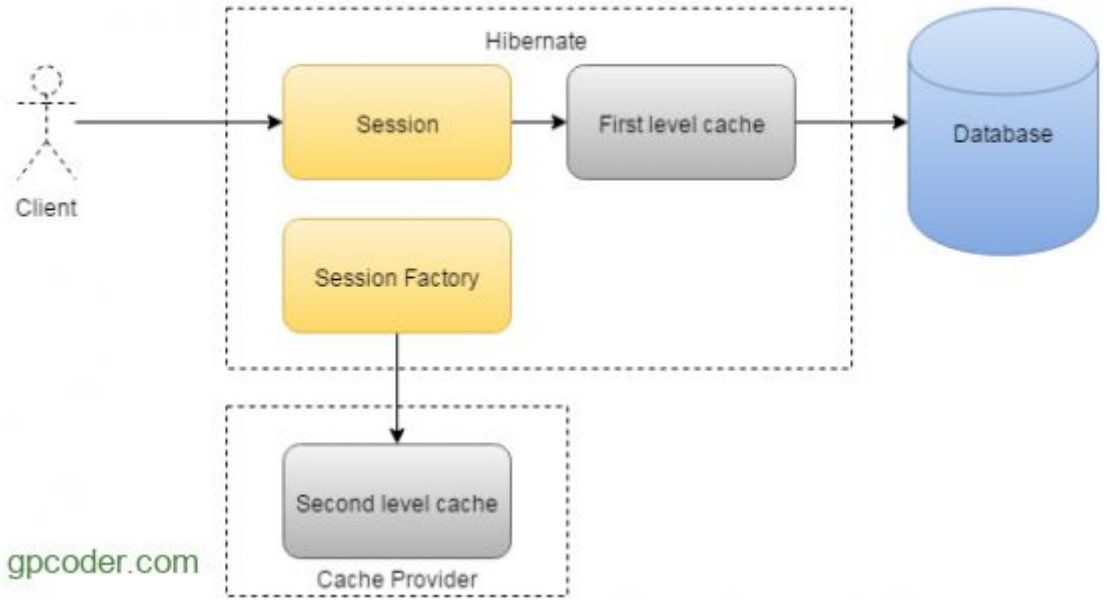



Transaction là nhóm các hoạt động (với database) của một tác vụ. Nếu một hoạt động không thành công thì toàn bộ tác vụ không thành công.

Với **JDBC** lập trình viên phải chủ động thực hiện commit khi toàn bộ hoạt động của tác vụ thành công, hay phải rollback khi có một hoạt động không thành công để kết thúc tác vụ.

Với **Hibernate** thì ta không cần quan tâm đến commit hay rollback, Hibernate đã quản lý nó giúp ta rồi.

### Hibernate Caching




Hibernate cung cấp một cơ chế bộ nhớ đệm, giúp giảm số lần truy cập vào database của ứng dụng càng nhiều càng tốt. Điều này sẽ có tác dụng tăng performance đáng kể cho ứng dụng của bạn.

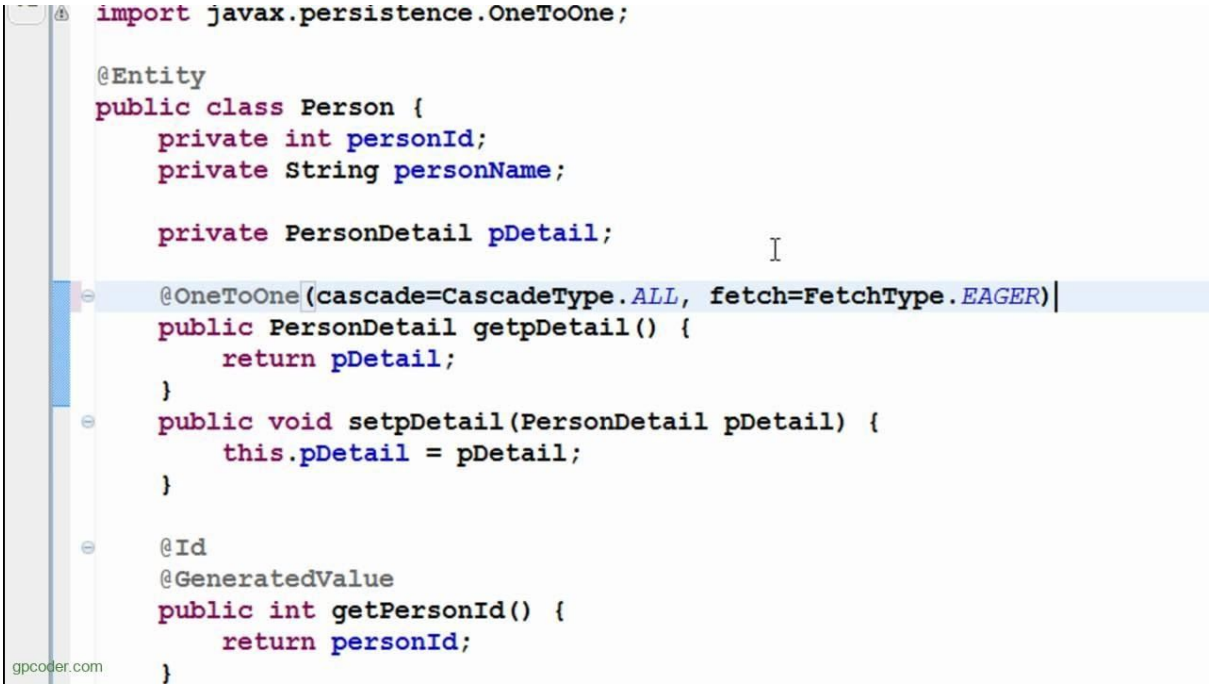
Hibernate lưu trữ các đối tượng trong session khi transation được kích hoạt. Khi một query được thực hiện liên tục, giá trị được lưu trữ trong session được sử dụng lại. Khi một transaction mới bắt đầu, dữ liệu được lấy lại từ database và được lưu trữ session. Hibernate cung cấp hai cấp độ Cache: bộ nhớ cache cấp một (first level cache) và bộ nhớ cache cấp hai (second level cache).

Chúng ta sẽ tìm hiểu chi tiết hơn về Cache trong Hibernate ở các bài viết kế tiếp.

### Associations

Thật dễ dàng để tạo một liên kết giữa các bảng bằng Hibernate như quan hệ một-một, một-nhiều, nhiều-một và nhiều-nhiều trong Hibernate bằng cách sử dụng các Annotation để ánh xạ đối tượng của bảng.

Chẳng hạn, chúng ta có bảng Person quan hệ 1-1 với bảng PersonDetail. Với JDBC, chúng ta phải viết SQL để thực hiện INNER JOIN giữa 2 bảng. Với Hibernate, chỉ đơn giản thêm Annotation @OneToOne như sau:




#### JPA Annotation Support

Hibernate implement đặc tả JPA, do đó chúng ta có thể sử dụng các Annotation của JPA như @Entity, @Table, @Column, ... Nhờ đặc điểm này, chúng ta có thể dễ dàng chuyển đổi giữa các ORM Framework mà không cần phải sử đổi code.

#### Connection Pooling

Như chúng ta đã biết, Connection Pooling giúp tăng performance nhờ vào sử dụng lại các kết nối khi có yêu cầu thay vì việc tạo kết nối mới.

Một số Connection Pooling được hỗ trợ bởi Hibernate: C3p0, Apache DBCP.

Chúng ta có thể dễ dàng tích hợp với các thư viện Connection Pooling có sẵn chỉ với vài dòng cấu hình như sau:

```
<property name="hibernate.c3p0.min_size">5</property>

<property name="hibernate.c3p0.max_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max_statements">50</property>
```

```
<property name="hibernate.c3p0.idle_test_period">3000</property>
```

Trên đây là một số khái niệm cơ bản về Hibernate mà tôi muốn giới thiệu đến các bạn. Trong các bài viết tiếp theo chúng ta sẽ cùng tìm hiểu cách cài đặt và sử dụng các API của Hibernate.