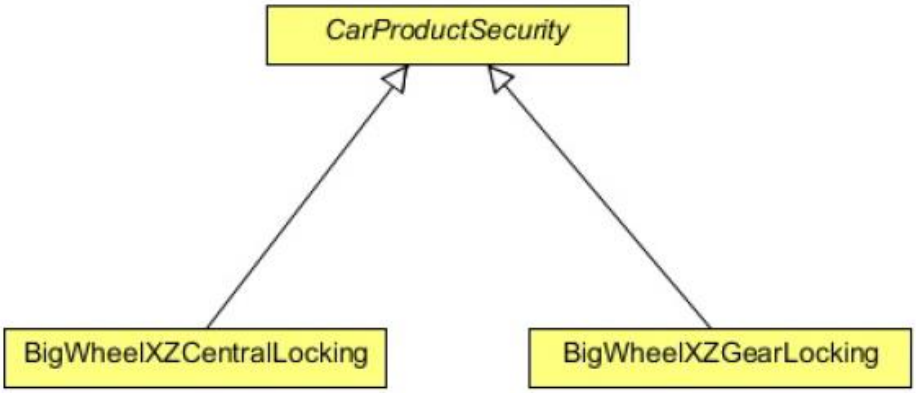
6.5 Mẫu thiết kế BRIDGE

6.5.1 Mở đầu

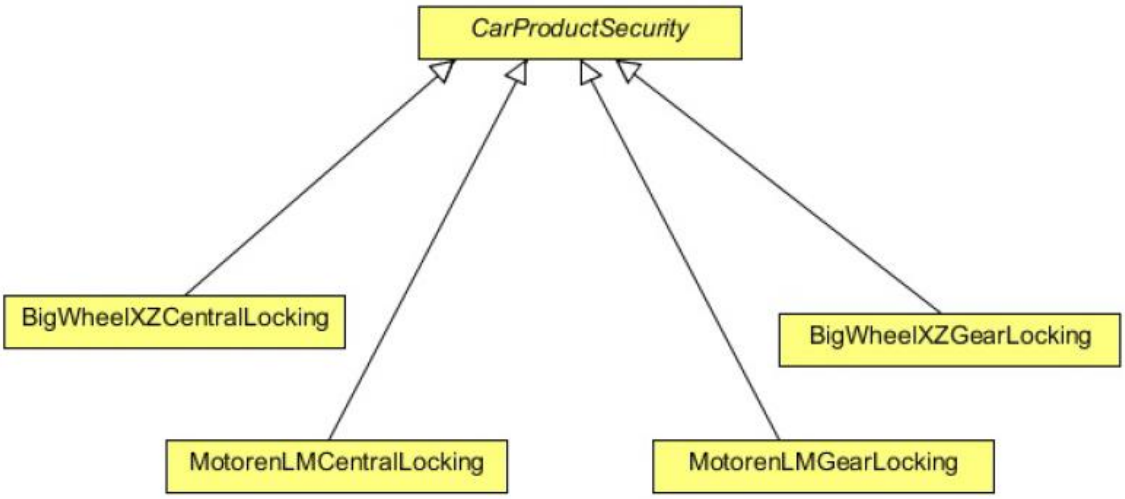
Hệ thống an ninh Sec là sản phẩm của công ty điện tử và an ninh, mà sản xuất và lặp đặt các sản phẩm cho ôtô. Nó phân phối bất cứ hệ thống điên tử và an ninh cho ôtô mà bạn muốn, từ túi khí cho đến hệ thống giám sát GPS, hệ thống đỗ lùi. Các công ty ôtô lớn sử dụng các sản phẩm của nó trong ôtô của họ. Công ty sử dụng cách tiếp cận hướng đối tượng được xác định tốt để theo dõi các sản phẩm của họ sử dụng phần mềm mà được phát triển và bảo trì chỉ bởi họ. Họ nhận ôtô và tạo ra các sản phẩm cho nó và lắp đặt chúng vào ôtô.

Hiện tại, họ nhận được các hợp đồng mới từ BigWheel (một công ty ôtô) để tạo ra hệ thống khóa bánh răng và khóa trung tâm cho mẫu ôtô mới xz. Để bảo trì nó, họ tạo ra hệ thống phần mềm mới. Họ bắt đầu từ việc tạo lớp abstract mới CarProductSecurity, mà ở đó họ giữ một số phương thức đặc thù cho ôtô và một số đặc trưng mà họ nghĩ là dùng chung cho mọi sản phẩm an ninh. Sau đó họ kế thừa lớp này để tạo ra hai lớp con khác nhau đặt tên là BigWheelXZCentral Locking và BigWheelXZGear Locking. Lược đồ lớp trông như sau:



Thời gian sau, một công ty ôtô khác Motoren yêu cầu họ sản xuất hệ thống khóa trung tâm và hệ thống khóa bánh răng mới cho mẫu ôtô mới lm. Do đó, một hệ thống an ninh cùng loại không thể được sử dụng cho cả hai mô hình ôtô khác nhau, Hệ thống an ninh Sec buộc phải sản xuất hệ thống mới cho họ, và buộc phải tạo các lớp mới MotorenLMCentralLocking và MotorenMLGearLocking mà cũng mở rộng lớp CarProductSecurity.

Bây giờ lược đồ lớp mới trông như sau:



Như vậy cũng được, nhưng điều gì xảy ra, nếu một công ty ôtô nữa lại yêu cầu một hệ thống mới khác khóa trung tâm và khóa bánh răng? Người ta cần phải tạo hai lớp mới cho nó. Thiết kế này sẽ tạo mỗi lớp cho một hệ thống, và sẽ tồi tệ hơn, nếu hệ thống đỗ lùi lại được sản xuất cho mỗi trong số hai công ty ôtô, hai hoặc nhiều hơn lớp mới sẽ được tạo cho mỗi một trong số đó.

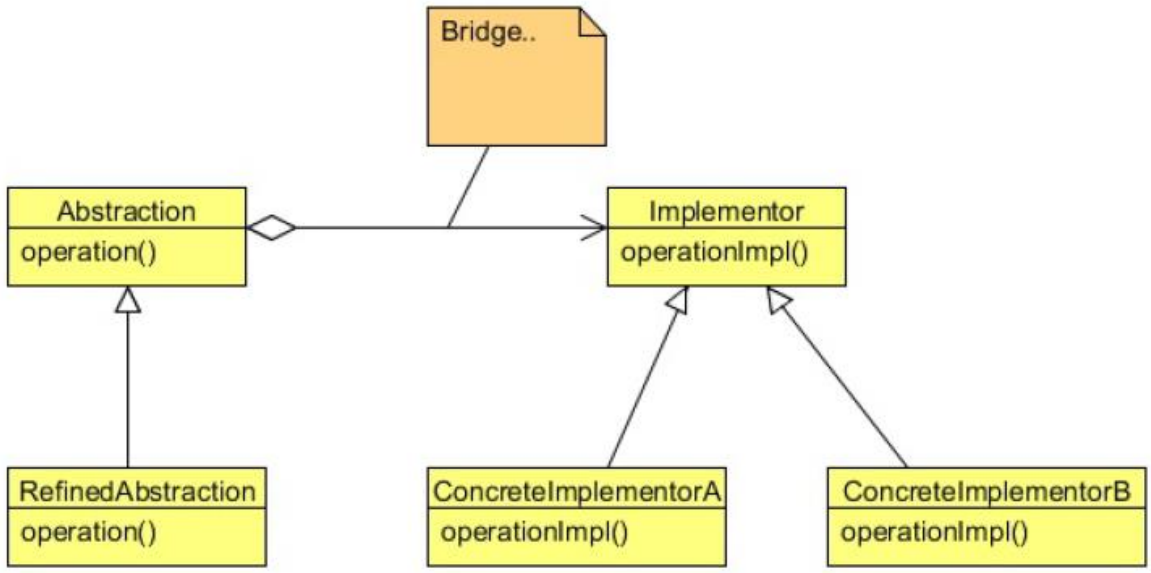
Một thiết kế với quá nhiều lớp con sẽ không linh hoạt và khó bảo trì. Kế thừa sẽ trói cài đặt với lớp trừu tượng thường xuyên, mà làm cho khó thay đổi, mở rộng và tái sử dụng lớp trừu tượng và cài đặt một cách độc lập.

Lưu ý rằng, ôtô và sản phẩm cần phải được đa dạng độc lập để tạo nên phần mềm dễ dàng mở rộng và tái sử dụng.

Mẫu thiết kế Bridge có thể giải quyết bài toán này, nhưng trước hết chúng ta xem xét chi tiết mẫu Bridge.

6.5.2 Mẫu thiết kế Bridge là gì

Mẫu thiết kế Bridge hướng tới tách gắn kết trừu tượng ra khỏi cài đặt sao cho hai cái đó có thể đa dạng một cách độc lập. Nó đặt trừu tượng và cài đặt thành hai phân cấp khác nhau sao cho cả hai có thể mở rộng độc lập.



Các thành phần của mẫu Bridge được tạo từ trừu tượng, trừu tượng làm mịn, cài đặt và cài đặt cụ thể.

Trừu tượng xác định giao diện trừu tượng và cũng bảo trì tham chiếu đến đối tượng kiểu cài đặt và kết nối giữa trừu tượng và cài đặt được gọi là cái cầu Bridge.

Trừu tượng làm mịn mở rộng giao diện trừu tượng.

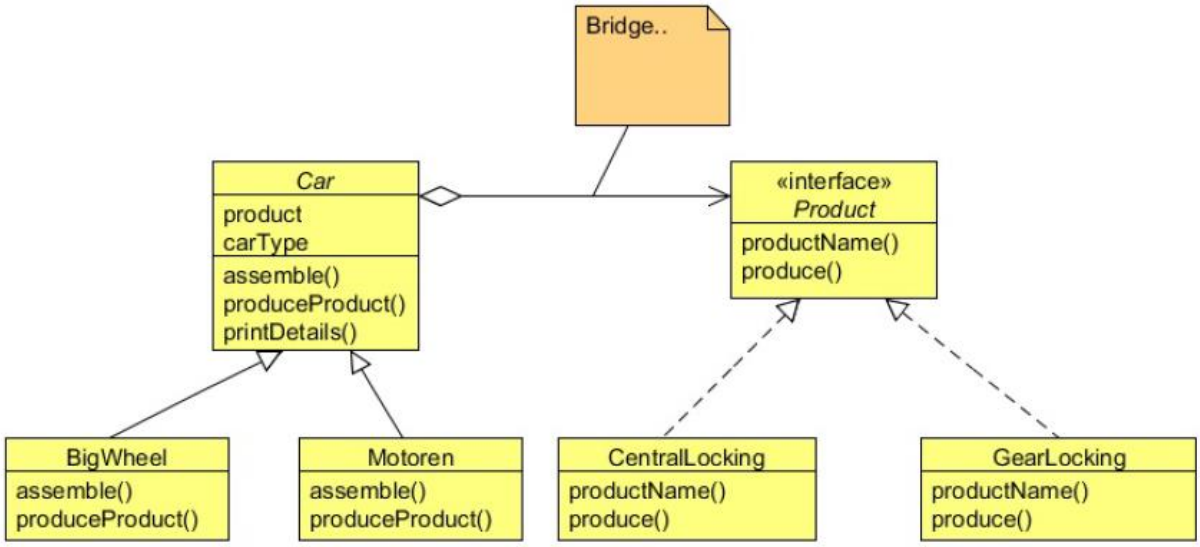
Cài đặt cung cấp giao diện cho lớp cài đặt, tức là cài đặt cụ thể.

Và cài đặt cụ thể triển khai giao diện Implementer và xác định cài đặt cụ thể.

Mẫu Bridge tách giao diện khỏi cài đặt. Kết quả là, cài đặt không gắn thường xuyên với giao diện. Cài đặt của trừu tượng có thể được cấu hình trong thời gian chạy. Nó cũng loại bỏ sự phụ thuộc thời gian dịch vào cài đặt. Thay đổi lớp cài đặt không đòi hỏi dịch lại lớp trừu tượng và client của nó. Client chỉ cần biết về trừu tượng và bạn có thể che giấu cài đặt khỏi chúng.

6.5.3 Giải pháp cho bài toán

Thay vì tạo các lớp con cho mỗi sản phẩm cho mỗi mẫu xe trong bài toán đang bàn luận ở trên, chúng ta có thể tách thiết kế thành hai phân cấp khác nhau. Một giao diện cho sản phẩm mà sẽ được dùng như một cài đặt và một cái khác sẽ là một trừu tượng của một kiểu ôtô.




```
package com.javacodegeeks.patterns.bridgepattern;

public interface Product {

        public String productName();
        public void produce();
}
```

Bộ cài Implementer có phương thức *produce ()*, mà sẽ được sử dụng bởi bộ cài cụ thể concrete implementer để cung cấp chức năng cụ thể cho nó. Phương thức này sẽ tạo ra mẫu cơ bản của sản phẩm mà sẽ được sử dụng với bất cứ mẫu ôtô nào sau một số điều chỉnh chuyên biệt cho mẫu ôtô đó.

```
package com.javacodegeeks.patterns.bridgepattern;

public class CentralLocking implements Product{

        private final String productName;

        public CentralLocking(String productName){
                this.productName = productName;
        }

        @Override
        public String productName() {
                return productName;
        }

        @Override
        public void produce() {
                System.out.println("Producing Central Locking System");
        }

}

package com.javacodegeeks.patterns.bridgepattern;
```

```
public class GearLocking implements Product{

        private final String productName;

        public GearLocking(String productName){
                this.productName = productName;
        }

        @Override
        public String productName() {
                return productName;
        }

        @Override
        public void produce() {
                System.out.println("Producing Gear Locking System");
        }

}
```

Hai bộ cài cụ thể cung cấp cài đặt cho bộ cài Product.

Bây giờ nói đến trừu tượng là lớp Car, mà giữ tham chiếu đến kiểu sản phẩm và cung cấp hai phương thức abstract *produceProduct ()* và *assemble ().*

```
package com.javacodegeeks.patterns.bridgepattern;

public abstract class Car {

        private final Product product;
        private final String carType;

        public Car(Product product,String carType){
                this.product = product;
                this.carType = carType;
        }

        public abstract void assemble();
        public abstract void produceProduct();

        public void printDetails(){
                System.out.println("Car: "+carType+", Product:"+product.productName());
        }
}
```

Các lớp con của Car sẽ cung cấp các cài đặt cụ thể và chuyên biệt cho các phương thức *assemble* và *produceProduct().*

```
package com.javacodegeeks.patterns.bridgepattern;

public class BigWheel extends Car{

        private final Product product;
        private final String carType;

        public BigWheel(Product product, String carType) {
                super(product, carType);
                this.product = product;
                this.carType = carType;
        }

        @Override
        public void assemble() {
                System.out.println("Assembling "+product.productName()+" for "+carType);
        }

        @Override
        public void produceProduct() {
                product.produce();
                System.out.println("Modifing product "+product.productName()+" according to ↩
                    "+carType);
        }

}
```

```
package com.javacodegeeks.patterns.bridgepattern;

public class Motoren extends Car{

        private final Product product;
        private final String carType;

        public Motoren(Product product, String carType) {
                super(product, carType);
                this.product = product;
                this.carType = carType;
        }

        @Override
        public void assemble() {
                System.out.println("Assembling "+product.productName()+" for "+carType);
        }

        @Override
        public void produceProduct() {
                product.produce();
                System.out.println("Modifing product "+product.productName()+" according to ↩
                    "+carType);
        }

}
```

Bây giờ ta sẽ kiểm tra ví dụ này

```
package com.javacodegeeks.patterns.bridgepattern;

public class TestBridgePattern {

        public static void main(String[] args) {
                Product product = new CentralLocking("Central Locking System");
                Product product2 = new GearLocking("Gear Locking System");
                Car car = new BigWheel(product , "BigWheel xz model");
                car.produceProduct();
                car.assemble();
                car.printDetails();

                System.out.println();

                car = new BigWheel(product2 , "BigWheel xz model");
                car.produceProduct();
                car.assemble();
                car.printDetails();

                car = new Motoren(product, "Motoren lm model");
```

```
                car.produceProduct();
                car.assemble();
                car.printDetails();

                System.out.println();

                car = new Motoren(product2, "Motoren lm model");
                car.produceProduct();
                car.assemble();
                car.printDetails();

        }

}
```

Ví dụ trên sẽ in ra kết quả sau:

```
Producing Central Locking System
Modifing product Central Locking System according to BigWheel xz model
Assembling Central Locking System for BigWheel xz model
Car: BigWheel xz model, Product:Central Locking System

Producing Gear Locking System
Modifing product Gear Locking System according to BigWheel xz model
Assembling Gear Locking System for BigWheel xz model
Car: BigWheel xz model, Product:Gear Locking System

Producing Central Locking System
Modifing product Central Locking System according to Motoren lm model
Assembling Central Locking System for Motoren lm model
Car: Motoren lm model, Product:Central Locking System

Producing Gear Locking System
Modifing product Gear Locking System according to Motoren lm model
Assembling Gear Locking System for Motoren lm model
Car: Motoren lm model, Product:Gear Locking System
```

6.5.4 Sử dụng mẫu Bridge

Bạn sẽ sử dụng mẫu Bridge khi:

- Bạn muốn tránh trói buộc thường xuyên giữa trừu tượng và các cài đặt của nó. Điều này có thể là trường hợp, chẳng hạn cài đặt cần được lựa chọn hoặc chuyển đổi trong thời gian chạy.
- Cả các trừu tượng và các cài đặt cần được mở rộng bằng các lớp con. Trong trường hợp này, mẫu Bridge cho phép kết hợp các trừu tượng và cài đặt khác nhau và mở rộng chúng một cách độc lập.
- Các thay đổi trong cài đặt của trừu tượng không có tác động đến client; như vậy code của họ sẽ không phải dịch lại.
- Bạn muốn chia sẻ một cài đặt giữa nhiều đối tượng (chẳng hạn, sử dụng đếm tham chiếu) và sự việc này cần được che giấu khỏi client.