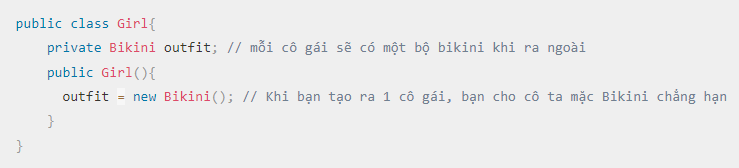
**DI và IoC**

**1, DI (Dependency Injection)**

DI (Dependency Injection) là 1 design pattern được thiết kế để ngăn chặn sự phụ thuộc giữa các Class , để khiến cho code dễ hiểu hơn, trực quan hơn , nhằm phục vụ cho mục đích bảo trì và nâng cấp code

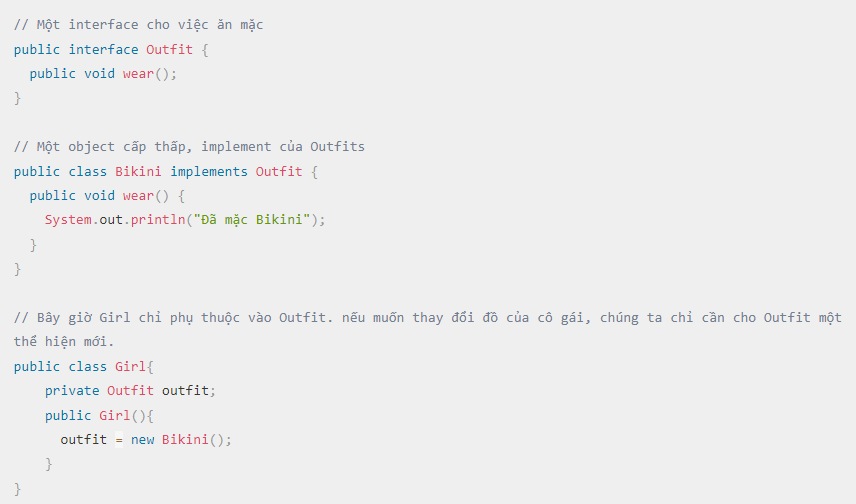
**Ví dụ:**



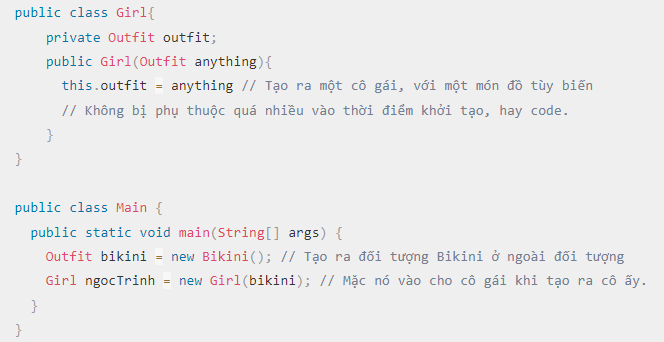
* Tạo ra class Girl đồng thời tạo ra thêm 1 bộ Bikini đi kèm với cô gái này
* **Girl phụ thuộc (dependency) vào Bikini**
* **Nguyên tắc cần ghi nhớ:**

*“Các Class không nên phụ thuộc vào các kế thừa cấp thấp, mà nên phụ thuộc vào Abstraction (lớp trừu tượng)”*

* **Cách làm theo Dependency Injection**

**

Phải sửa ở dòng constructor Girl() vì Girl vẫn đang được gắn với 1 bộ Bikini duy nhất

**

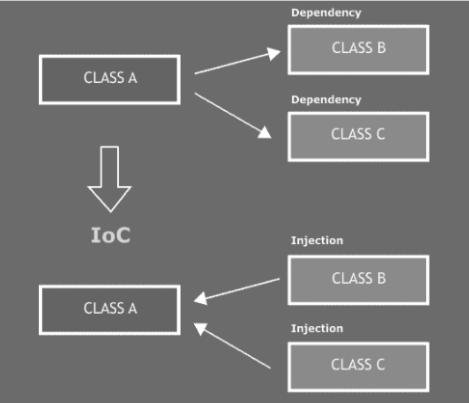
* Chúng ta đã gần như tách khỏi sự phụ thuộc của **Girl** vào **Bikini** rồi mà còn tăng tính tùy biến linh hoạt cho code khi **Girl** có thể mặc bất cứ loại **Outfit** nào tùy thích
* **Girl** muốn mặc outfit khác thì chỉ cần tạo 1 Class kế thừa interface **Outfit** và Inject vào **Girl** là xong!
* **Có 3 cách để DI vào 1 đối tượng:**
* **Constructor Injection:** tiêm trực tiếp dependency vào Constructor (như ví dụ trên)
* **Setter Injection:** các dependency sẽ được truyền thông qua các setter method
* **Interface Injection:** Dependency sẽ cung cấp một Interface, trong đó có chứa hàm có tên là Inject. Các client phải triển khai một Interface mà có một setter method dành cho việc nhận dependency và truyền nó vào class thông qua việc gọi hàm Inject của Interface đó

**2, IoC (Inversion of Control)**

* IoC cho phép các kiến trúc phần mềm có thể đảo ngược quyền điều khiển so với lập trình hướng thủ tục. Là nguyên lý thiết kế được tạo ra để các code tuân thủ theo Dependency Inversion
* 1 vài mô hình để triển khai IoC: Service Locator, Events, DI (Dependency Injection)

Ví dụ: Giả sử 1 class mẹ là A, 2 class con là B và C (B và C được gọi là các dependencies)

* **Không sử dụng IoC:** Class A phải khởi tạo và điều khiển 2 class B và C
* Bất kỳ thay đổi nào ở class A -> Class B và C đều bị thay đổi -> Làm giảm khả năng bảo trì code vì 1 sự thay đổi kéo theo hàng loạt những thay đổi khác
* **Sử dụng IoC:** Class B và C sẽ được đưa đến độc lập so với class A thông qua bên thứ ba
* Class không phụ thuộc lẫn nhau mà chỉ phụ thuộc vào interface -> Bất kỳ sự thay đổi nào ở class A sẽ không làm ảnh hưởng đến class B và C

****

* **Lợi ích của IoC:**
* Có sự tách rời việc thực hiện 1 nhiệm vụ nhất định khỏi quá trình thực thi
* Giúp việc chuyển đổi giữa các implementations trở nên dễ dàng
* Việc thay thế các module không có tác dụng phụ với các modules khác