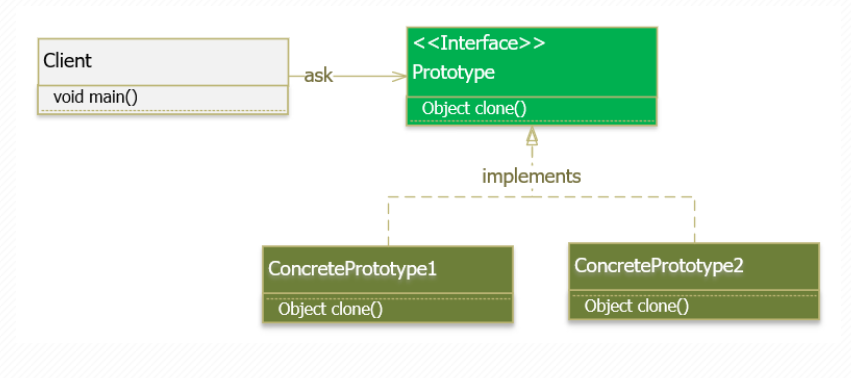
1. Giới thiệu mẫu thiết kế

****Giới thiệu****

Prototype Pattern là một trong những pattern phổ biến trong lập trình hướng đối tượng, là một pattern thuộc nhóm Creational Patterns. Nó có nhiệm vụ khởi tạo một đối tượng bằng cách ****clone**** một đối tượng đã tồn tại thay vì khởi tạo với từ khoá ****new****. Đối tượng mới là một bản sao có thể giống 100% với đối tượng gốc, chúng ta có thể thay đổi dữ liệu của nó mà không ảnh hưởng đến đối tượng gốc.

****Sử dụng****  
Để cài đặt Prototype Pattern, tạo ra một Clone() Method ở lớp cha, và triển khai ở các lớp con.

Prototype pattern sẽ sử dụng những abstract class.  
Những class tham gia vào prototype pattern bao gồm :



* ****Prototype****: Cung cấp Interface hoặc Abstract Class để copy chính bản thân nó – phương thức Clone().
* ****ConcreatePrototype****:  các lớp này thực thi interface (hoặc kế thừa từ lớp abstract) được cung cấp bởi Prototype để copy (nhân bản) chính bản thân nó. Các lớp này chính là thể hiện cụ thể phương thức clone(). Lớp này có thể không cần thiết nếu: Prototype là một class và nó đã implement việc clone chính nó.
* ****Client****: Tạo một object mới bằng cách yêu cầu prototype copy chính bản thân nó.

1. **Mục đích sử dụng**

Prototype Pattern được dùng khi việc tạo một object tốn nhiều chi phí và thời gian trong khi bạn đã có một object tương tự tồn tại.

Prototype ẩn đi những lớp con rời rạc từ phía client, do vậy làm giảm đi số lớp con mà client cần biết. Hơn thế nữa pattern này làm cho client hoạt động với những lớp con cụ thể mà không phải thay đổi gì.  
Sau đây là một vài lợi ích khác của prototype pattern:

* Cải thiện ****performance****: giảm chi phí để tạo ra một đối tượng mới theo chuẩn, điều này sẽ làm tăng hiệu suất so với việc sử dụng từ khóa ****new**** để tạo đối tượng mới.
* Giảm độ phức tạp cho việc khởi tạo đối tượng: do mỗi lớp chỉ implement cách clone của chính nó.
* Giảm việc phân lớp, tránh việc tạo nhiều lớp con cho việc khởi tạo đối tượng như của [Abstract Factory Pattern](https://gpcoder.com/4365-huong-dan-java-design-pattern-abstract-factory/).
* Khởi tạo object mới bằng cách thay đổi một vài thuộc tính của object (các object có ít điểm khác biệt nhau): Một hệ thống linh động sẽ để cho chúng ta tự định nghĩa một hành động nào đó thông qua sự kết hợp với một object (nghĩa là một phương thức của một class) hơn là định nghĩa một class mới.
* Khởi tạo object mới bằng cách thay đổi cấu trúc: Rất nhiều ứng dụng xây dựng hệ thống từ nhiều phần và các phần con. Các phần con lại khởi tạo từ nhiều phần con khác (chia nhỏ bài toán). Prototype pattern cũng hỗ trợ điều này. Nghĩa là các phần đó có thể được khởi tạo từ việc copy một nguyên mẫu từ một “cấu trúc” khác. Miễn là các phần kết hợp đều thể hiện clone() và được sử dụng với cấu trúc khác nhau làm nguyên mẫu.