**1. Prototype Pattern là gì?**

Prototype Pattern là một mẫu tạo dựng ([Creation Pattern](https://stackjava.com/design-pattern/design-pattern-la-gi-design-pattern-trong-java.html)).

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.

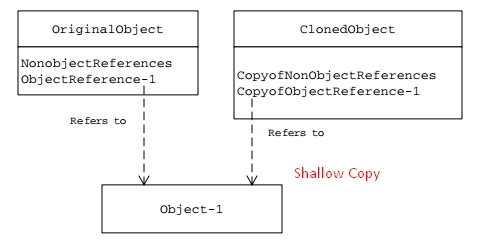
Prototype Pattern cung cấp cơ chế để copy từ object ban đầu sang object mới và thay đổi giá trị một số thuộc tính nếu cần.

Trong Java cung cấp sẵn mẫu prototype pattern này bằng việc implement interface Cloneable và sử dụng method clone() để copy object.

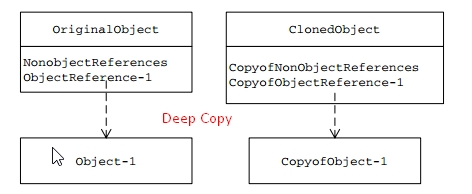
**2. Prototype Pattern UML Diagram**

Vì object cần copy có thể có các object con khác bên trong nên xảy ra 2 trường hợp:

* Trường hợp 1: Các object con bên trong chỉ được copy reference (tức là object ban đầu và object tạo mới đều trỏ tới chung 1 object con bên trong) (Shallow copy)



* Trường hợp 2: Các object con bên trong cũng được copy lại toàn bộ các thuộc tính. (Deep copy)



**3. Ví dụ.**

Ở đây mình có 1 class User với 2 method thực hiện copy là deepCopy và shallow Copy.

* public class User {
* private String firstName;
* private String lastName;
* private String displayName;
* private String email;
* private Address address;
* public User() {
* }
* public User(String firstName, String lastName, String displayName, String email, Address address) {
* this.firstName = firstName;
* this.lastName = lastName;
* this.displayName = displayName;
* this.email = email;
* this.address = address;
* }
* public String getFirstName() {
* return firstName;
* }
* public void setFirstName(String firstName) {
* this.firstName = firstName;
* }
* public String getLastName() {
* return lastName;
* }
* public void setLastName(String lastName) {
* this.lastName = lastName;
* }
* public String getDisplayName() {
* return displayName;
* }
* public void setDisplayName(String displayName) {
* this.displayName = displayName;
* }
* public String getEmail() {
* return email;
* }
* public void setEmail(String email) {
* this.email = email;
* }
* public Address getAddress() {
* return address;
* }
* public void setAddress(Address address) {
* this.address = address;
* }
* public User shallowCopy() {
* User user = new User(this.firstName, this.lastName, this.displayName, this.email, this.address);
* return user;
* }
* public User deepCopy() {
* Address address = new Address(this.getAddress().getProvince(), this.getAddress().getDistrict(),
* this.getAddress().getStreet());
* User user = new User(this.firstName, this.lastName, this.displayName, this.email, address);
* return user;
* }
* @Override
* public String toString() {
* return "User [firstName=" + firstName + ", lastName=" + lastName + ", displayName=" + displayName + ", email="
* + email + ", address=" + address + "]";
* }
* }

method shallowCopy chỉ copy reference object address bên trong

method deepCopy sẽ tạo một đối tượng address mới giống đối tượng address ban đầu vào đối tượng được copy.

**Demo:**

* public static void main(String[] args) {
* Address address = new Address("Ha Noi", "Ha Dong", "Nguyen Trai");
* User originalUser = new User("firstName", "lastName", "stackjava", "abc@gmail.com", address);
* User userShallowCopy = originalUser.shallowCopy();
* User userDeepCopy = originalUser.deepCopy();
* System.out.println("Before change address:");
* System.out.println("original user: " + originalUser);
* System.out.println("shallow user: " + userShallowCopy);
* System.out.println("deep user: " + userDeepCopy);
* System.out.println("-----------------------------------");
* System.out.println("After shallow user change address:");
* userShallowCopy.getAddress().setDistrict("Thanh Xuan");
* System.out.println("original user: " + originalUser);
* System.out.println("shallow user: " + userShallowCopy);
* System.out.println("deep user: " + userDeepCopy);
* System.out.println("-----------------------------------");
* System.out.println("After deep user change address:");
* userDeepCopy.getAddress().setDistrict("Cau Giay");
* System.out.println("original user: " + originalUser);
* System.out.println("shallow user: " + userShallowCopy);
* System.out.println("deep user: " + userDeepCopy);
* }