Universidad Tecnológica del Perú

Docente:

Dilian Anabel HURTADO PONCE

Tarea 7: Patrón de diseño Prototype

Semana: 7

GRUPO 5

Integrantes:

Roberto Agustín Mejía Collazos

Miguel Ángel Velásquez Ysuiza

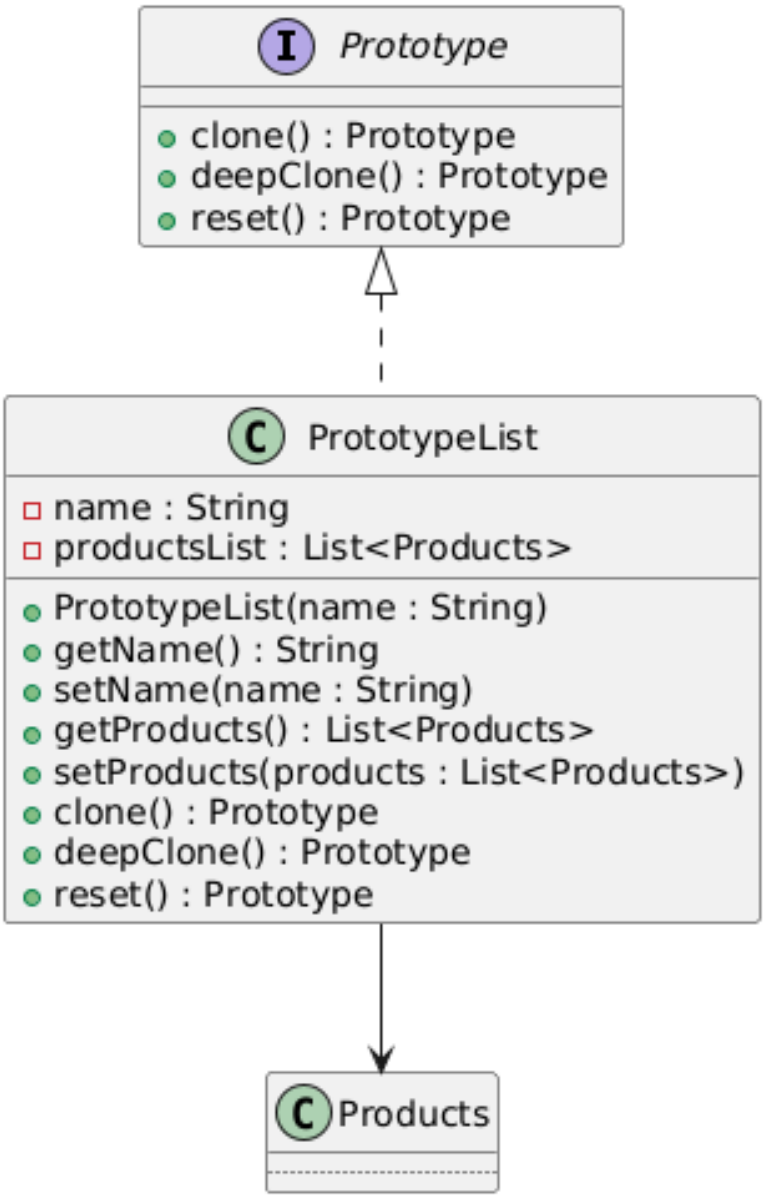
Manuel Ángel Pecho Santos

Daniel Wilfredo Sotomayor Beteta

Cristina Paola Burgos Salazar

26/09/24

DIAGRAMA UML



Código (GitHub)

```
// Definición de la interfaz Prototype que permite la clonación de objetos
public interface Prototype<T extends Prototype<T>> extends Cloneable {
   T clone();  // Clona el objeto
   T deepClone(); // Realiza una clonación profunda (si es necesario)
   T reset();  // // Restablece el objeto a su estado inicial (opcional)
}


// Clase PrototypeList que implementa la interfaz Prototype para listas
public class PrototypeList implements Prototype {
   private String name;
   private List<Products> productsList;

   public PrototypeList(String name) {
      this.name = name;
   }

    @Override
   public Prototype deepClone() {
      PrototypeList clone = new PrototypeList(name);
      List<Products> products = new ArrayList<>();
      List<Prototype> values = productsList.stream()
            .map(Products::clone).collect(Collectors.toList());
      products.add((Products) values);
      clone.setProducts(products);
      return clone;
   }

   @Override
   public Prototype reset() {
      if (productsList.isEmpty()) {
         return null;
      }

      PrototypeList clone = new PrototypeList(name);
      List<Products> products = new ArrayList<>();
      products.addAll(productsList);
      clone.setProducts(products);

      return clone;
   }

   @Override
   public Prototype clone() {
      PrototypeList clone = new PrototypeList(name);
      clone.setProducts(productsList);
      return clone;
   }
}
```

```
// Clase Products que implementa la interfaz Prototype para productos
public class Products implements Prototype {

   private String name;
   private String description;
   private int price;
   private String[] items;

   public Products(String name, String description, int price, String[] items) {
      this.name = name;
      this.description = description;
      this.price = price;
      this.items = items;
   }

   @Override
   public Prototype clone() {
      Products clone = new Products(name, description, price, items);
      return clone;
   }
}

public class Main {
   public static void main(String[] args) {
      // Lista de productos
      List<Products> productsList =
            List.of(new Products("Producto 1", "Este es el primer producto.", 100, new
String[]{"Item 1", "Item 2"}),
                  new Products("Producto 2", "Este es el segundo producto.", 200, new
String[]{"Item 1", "Item 2"}));

      // Crear una lista prototipo de productos
      PrototypeList list = new PrototypeList("Listado");
      list.setProducts(productsList);

      // Clonar la lista prototipo
      PrototypeList clone = (PrototypeList) list.clone();

      // Cambiar el precio de los productos en la lista clonada
      for (Products product : clone.getProducts()) {
         product.setPrice(product.getPrice() * 2);
      }

      // Imprimir la lista original y la lista clonada
      System.out.println(list);
      System.out.println(clone);
   }
}
```

ANÁLISIS DEL CÓDIGO

El código implementa el patrón de diseño Prototipo para crear copias de un objeto PrototypeList sin tener que volver a crearlos desde cero.

La interpretación es:

Imaginemos que se tiene una lista de productos (en el código, productsList). Creaamos una nueva lista con la misma información cada vez sería tedioso. El patrón Prototipo te permite crear una "plantilla" de la lista (el objeto PrototypeList) y luego hacer copias de esa plantilla.

Aplicación del Patrón:

- Interfaz Prototype: Define el contrato para crear copias. Los métodos clone(), deepClone() y reset() garantizan que se puedan crear copias de cualquier objeto que implemente esta interfaz.
- Clase PrototypeList: Implementa la interfaz Prototype. Aquí se define cómo se crean las copias de la lista de productos.

Diferencia un Enfoque Tradicional (POO):

En el enfoque tradicional (POO), para crear una copia, se usaría un constructor o un método de copia manual. Esto implica:

1. Crear un objeto nuevo: Se crea un nuevo objeto de la misma clase.
2. Copiar los datos: Se copian los datos del objeto original al nuevo objeto.

El patrón Prototipo simplifica este proceso. En lugar de crear un objeto nuevo y copiar datos, se utiliza un método clone() o deepClone() para generar una copia del objeto existente.