

Prototype Pattern

- 모형이 되는 인스턴스를 복사해서 인스턴스 생성
- 그러나 Java/C#에서 제공되는 clone() 을 사용하면 얕은 복사(Shallow Copy)가 되므로 클래스 안에 참조 형식의 데이터가 있으면 제대로 복사가 되지 않음.
 - 직렬화(Serialize)를 이용하여 깊은 복사(Deep Copy) 구현.
- 언제 쓰는게 좋을까?
 - 자주 DB에 접근하여 삽입/삭제/수정하는 데이터인 경우 2Tier가 아닌 중간에 Middleware를 놓고 복사본을 수정하는 게 성능상 이득. (Caching)
 - 원본 데이터는 그대로 보존이 필요하고, 따로 수정할 데이터가 있는 경우 (ex. 파일 불러오기, 되돌리기)

https://en.wikipedia.org/wiki/Design_Patterns#Patterns_by_Type

Architecture, Modeling, Design Pattern, Creational

Java

Prototype

```
/* 깊은 복사를 위해 Serializable 필요 */
public class Manager<T extends Serializable> {
    private HashMap<String, T> showcase = new HashMap<>();
    public void register(String prototypeName, T prototype) {
        showcase.put(prototypeName, prototype);
    }
    public T createClone(String prototypeName) {
        T target = showcase.get(prototypeName);
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(target);
            oos.close();
            baos.close();
            ByteArrayInputStream bais = new
ByteArrayInputStream(baos.toByteArray());
            ObjectInputStream ois = new ObjectInputStream(bais);
            ois.close();
            bais.close();
            return (T) ois.readObject();
        } catch (IOException e) {
            return null;
        } catch (ClassNotFoundException e) {
            return null;
        }
    }
}
```

ConcretePrototype

```
/* 깊은 복사를 위해 Serializable 필요 */  
public interface Character extends Serializable {  
    void attack();  
    void block();  
}
```

```
public class Player implements Character {  
  
    @Override  
    public void attack() {  
        System.out.println("Player - attack!");  
    }  
  
    @Override  
    public void block() {  
        System.out.println("Player - block!");  
    }  
}
```

```
public class Monster implements Character {  
  
    @Override  
    public void attack() {  
        System.out.println("Monster - attack!");  
    }  
  
    @Override  
    public void block() {  
        System.out.println("Monster - block!");  
    }  
}
```

Client

```
public class Main {  
    public static void main(String[] args) {
```

```

        Manager<Character> manager = new Manager<>();
        Player playerOriginal = new Player();
        Monster monsterOriginal = new Monster();
        manager.register("player", playerOriginal);
        manager.register("monster", monsterOriginal);
        Character playerClone = manager.createClone("player");
        Character monsterClone = manager.createClone("monster");
        print(playerOriginal, playerClone);
        print(monsterOriginal, monsterClone);
    }
    private static void print(Character original, Character clone) {
        original.attack();
        original.block();
        clone.attack();
        clone.block();
        System.out.println("같은 참조를 갖는가? " + (original == clone));
        System.out.println();
    }
}

/*
Player - attack!
Player - block!
Player - attack!
Player - block!
같은 참조를 갖는가? false

Monster - attack!
Monster - block!
Monster - attack!
Monster - block!
같은 참조를 갖는가? false
*/

```

C#

```

namespace PrototypePattern
{
    class Program
    {
        static void Main(string[] args)
        {
            Manager manager = new Manager();

            Character player = new Player();

            manager.Register("player", player);
            Character playerClone = manager.CreateClone("player");

```

```
        playerClone.attack();

        Console.WriteLine(player == playerClone);
    }
}

/* Client */
public class Manager
{
    private static Dictionary<string, Character> showcase = new
Dictionary<string, Character>();

    public void Register(string key, Character prototype)
    {
        showcase.Add(key, prototype);
    }

    public Character CreateClone(string key)
    {
        using (var ms = new MemoryStream())
        {
            var formatter = new BinaryFormatter();
            formatter.Serialize(ms, showcase[key]);
            ms.Position = 0;

            return formatter.Deserialize(ms) as Character;
        }
    }
}

/* Prototype */
public interface Character
{
    void attack();
    void block();
}

/* ConcretePrototype */
[Serializable]
public class Player : Character
{
    public void attack()
    {
        Console.WriteLine("Player - attack!");
    }

    public void block()
    {
        Console.WriteLine("Player - block!");
    }
}
```

```
/* ConcretePrototype */  
[Serializable]  
public class Monster : Character  
{  
    public void attack()  
    {  
        Console.WriteLine("Monster - attack!");  
    }  
  
    public void block()  
    {  
        Console.WriteLine("Monster - block!");  
    }  
}  
}
```

From:

<https://ledyx.xyz/> - **Ledyx WIKI**

Permanent link:

https://ledyx.xyz/doku.php?id=design_pattern:prototype_pattern

Last update: **2021/02/07 03:28**

