SDET Course

Design Patterns - Prototype

3 Types of Design Patterns

VERISOFT

- Creational
  - Singleton
  - Builder
  - **Prototype**
  - Factory Method
  - Abstract Factory
- Structural
  - Adapter
  - Composite
  - Proxy
  - Flyweight
  - Bridge
  - Facade
  - Decorator
- Behavioral
  - Strategy
  - Observer
  - Command
  - Memento
  - State
  - Template Method
  - Mediator
  - Chain of Responsibility
  - Interpreter
  - Visitor
  - Iterator

Agenda

- Description
- Diagram
- Code sample (Java)
- Use cases

VERISOFT

The Proble

VERISOFT

Introduction - The Cloned Teddy Bear

The Problem

Option 1

```
CopyObject co = new CopyObject(1, 2, new Object());
CopyObject co2 = new CopyObject();
co2.a = co.getA();
co2.b = co.getB();
co2.c = co.c; // private member - cannot copy
co2.o = co.getO();
```

Option 2

```
public class CopyObject {

    private int a;
    private int b;
    private Object o;

    public CopyObject(int a, int b, Object o) {
        this.a = a;
        this.b = b;
        this.o = o;
    }
}
```

- **Cannot access private members**
- **The interface issue (does not hold all members)**
- **Tightly coupled**
- **Shallow copy**
- **Concrete class**

Shallow / Deep Copy

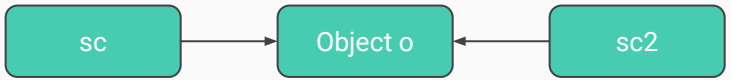
```java
public class ShallowCopy {

   private int a;
   private int b;
   private Object o;

   public ShallowCopy(int a, int b, Object o) {
       this.a = a;
       this.b = b;
       this.o = o;
   }

   public ShallowCopy(ShallowCopy sc) {
       this.a = sc.a;
       this.b = sc.b;
       this.o = sc.o;
   }
}
```

```java
ShallowCopy sc = new ShallowCopy(1, 2, new Object());
ShallowCopy sc2 = new ShallowCopy(sc);
Assertions.assertEquals(sc.o, sc2.o);
```



Shallow / Deep Copy

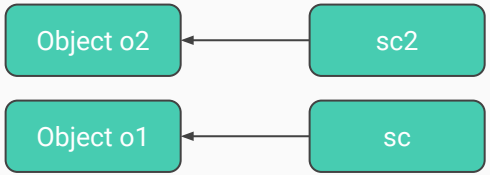
```
public class DeepCopy {

   private int a;
   private int b;
   private Object o;

   public DeepCopy(int a, int b, Object o) {
       this.a = a;
       this.b = b;
       this.o = o;
   }

   public DeepCopy(DeepCopy dc) {
       this.a = dc.a;
       this.b = dc.b;
       this.o = new Object(dc.o);
   }
}
```

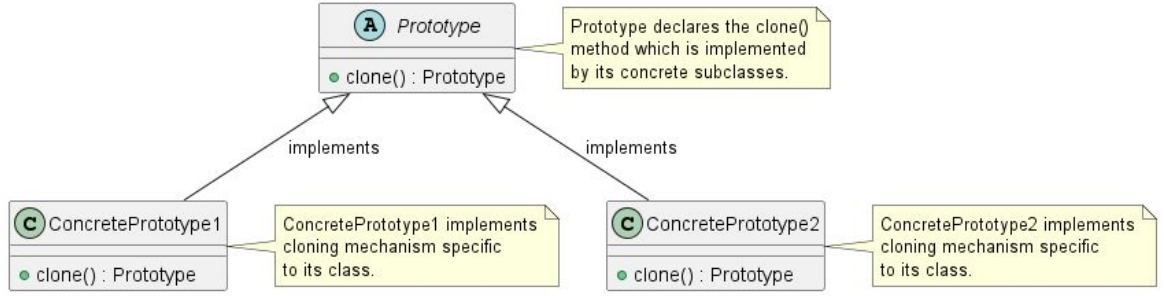
```
DeepCopy dc = new DeepCopy(1, 2, new Object());
DeepCopy dc2 = new DeepCopy(dc);
Assertions.assertNotEquals(dc.o, dc2.o);
```



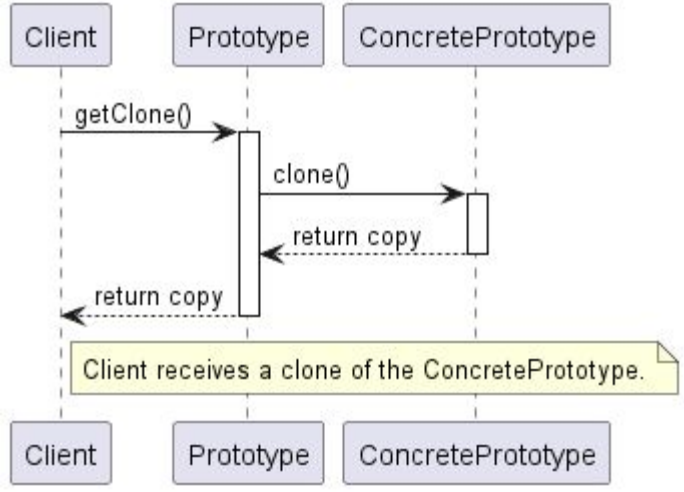
Description

The Prototype pattern is used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects. This pattern is particularly useful when the cost of creating an object is heavier than cloning it

Class Diagram



Sequence Diagram



Code Sample

Use cases

- General
  - Prototype Registry
  - Avoid subclass
  - Preload and caching

- In Test Automation
  - Clone WebDriver
  - Clone Page Objects

Happy Coding