DesignPattern: Prototype

Steckbrief

Name: Prototype
Art: Erzeugungsmuster (Creational Pattern)
Beteiligte Klassen:

- ***Prototype*** (Interface mit clone())
- ***ConcretePrototype*** (konkrete Klasse, implementiert Klonen)
- ***Client*** (fordert Klone an, anstatt konkrete Klassen direkt zu instanziieren)

Ziel: Neue Objekte durch **Klonen vorhandener Prototypen** erzeugen,
anstatt sie mit **new** neu zu instanziieren.

*Auch bekannt als "**Clone**"*

Vor- und Nachteile

- Objekte **klonen, ohne** an ihre **konkreten Klassen gekoppelt** zu sein
- **Wiederholten Initialisierungscode vermeiden**, indem **vorgefertigte Prototypen** geklont werden
- Komplexe **Objekte bequemer** erzeugen
- Eine **Alternative zur Vererbung**, wenn es um **Konfigurationsvorlagen** für komplexe Objekte geht

- Klonen **komplexer Objekte** mit **zirkulären Referenzen** kann **sehr schwierig** sein.

Einsatzgebiete

- Bei Aufwendigen Objektinitialisierungen
- Dynamische Objekterzeugungwenn Klasse unbekannt ist aber Objektvorlagen vorhanden sind
- **Anwendungsbeispiele**: Game Development, CAD, Machine Learning Modelle

Warum existiert das Pattern?

- Kopieren von Objekten ist i.d.R relativ Aufwendig
- **Probleme**: viele Attribute, private Attribute, Klasse muss bekannt sein
- **Lösung:**
- Prototypen implementieren ein Interface, wodurch Kopien erzeugt werden können auch wenn die Klasse unbekannt ist
- Kopie erfolgt über die clone()-Methode

```java
public class RobotPrototypeRegistry<T extends Prototype<T>>{
    private Map<Type, T> prototypes = new HashMap<>();

    public void add(Type key, T prototype){
        prototypes.put(key, prototype);

    }
    public T getClone(Type key){
        T prototype = prototypes.get(key);
        if (prototype == null){
          throw new IllegalArgumentException("No prototype registered for type: " + key.toString().toLowerCase());
        } else {
            return prototype.clone();
        }

    }
}
```

```java
    @Override
    public Robot clone(){
        return new Robot(this.color, this.height,this.type, this.state, this.tasks);
    }


}
```

```java
public interface Prototype <T>  {
        T clone();
    }
```