Design Pattern / Code Smell

Shotgun Surgery / Prototype

Dian Li, Ouali Cherikh, Ken Mavoungou

Introduction

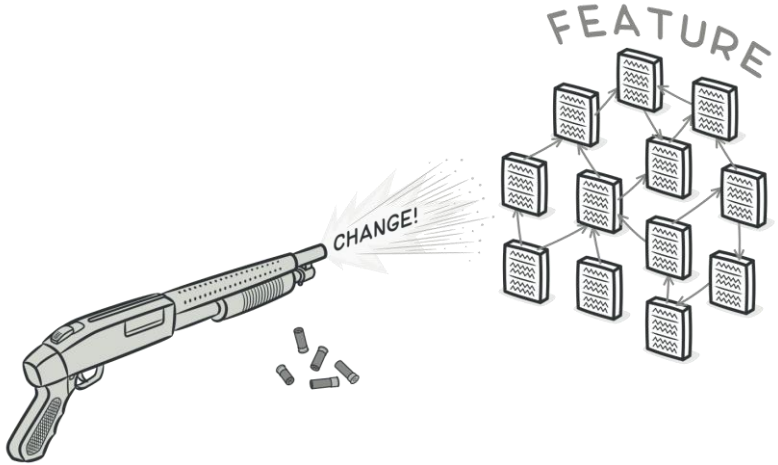
Design pattern:

En génie logiciel, ***un design pattern*** est une solution reproductible générale à un problème courant dans la conception logicielle. Ce n'est pas un design fini qui peut être transformé directement en code. Il s'agit d'une description ou d'un template, utilisé afin de résoudre un problème qui peut être utilisé dans de nombreuses situations différentes.

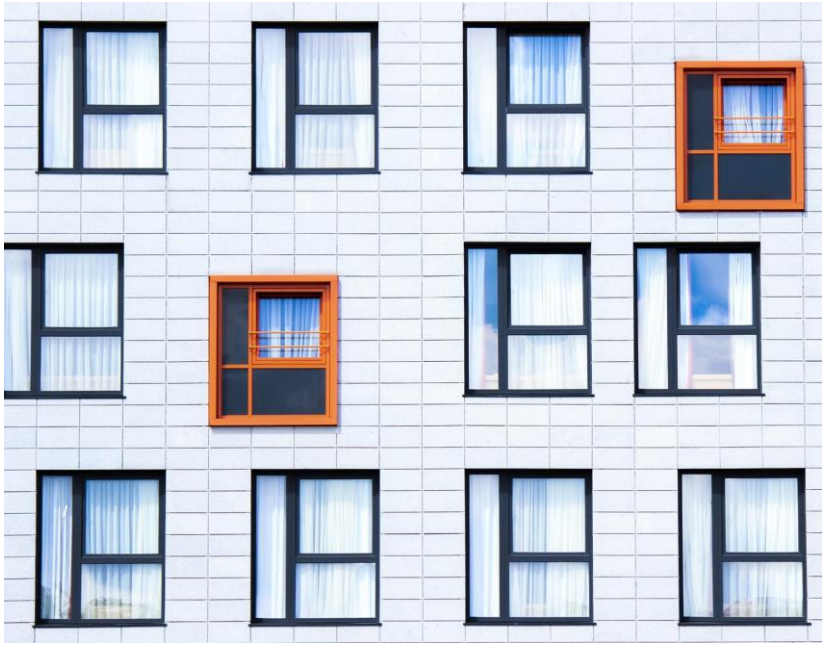
Code smell:

En génie logiciel, les ***code smells*** ou **mauvaises odeurs** sont des mauvaises pratiques de conception logicielle qui conduisent à l'apparition de défauts. Ces défauts sont souvent issus de mauvais choix d'implantation ou de conception et conduisent à une complexification du code source et de la maintenance et évolutivité de celui-ci. Afin de corriger un *code smell*, il est nécessaire de procéder à un réusinage du code source, c'est-à-dire modifier le code sans en altérer son comportement.

Choix et conception



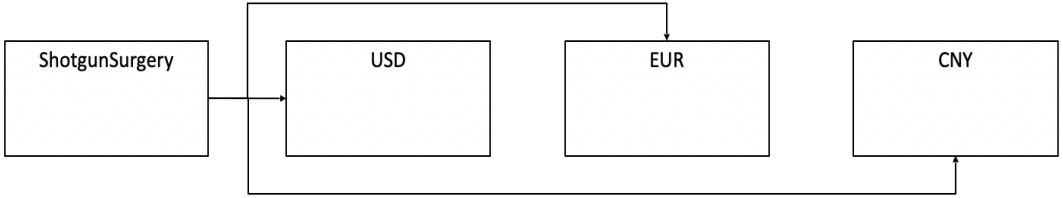
Code smell: Shotgun surgery



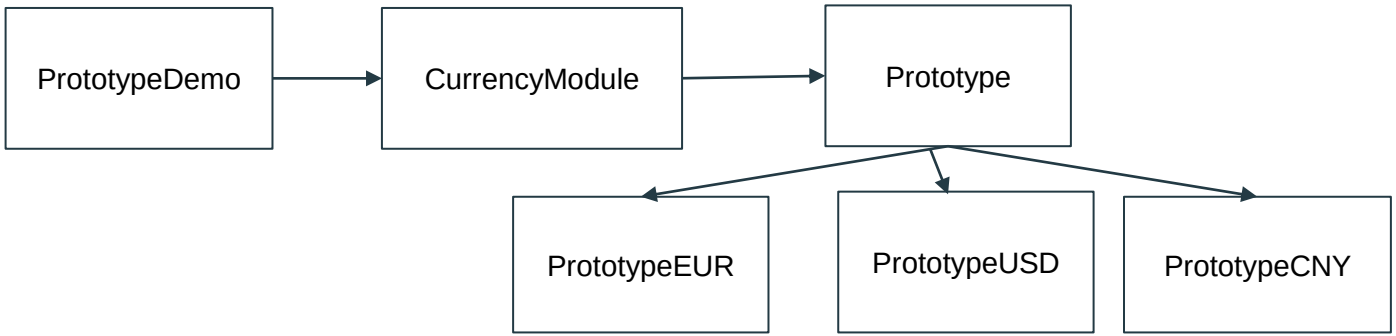
Design pattern: Prototype

Comparaison des deux structures

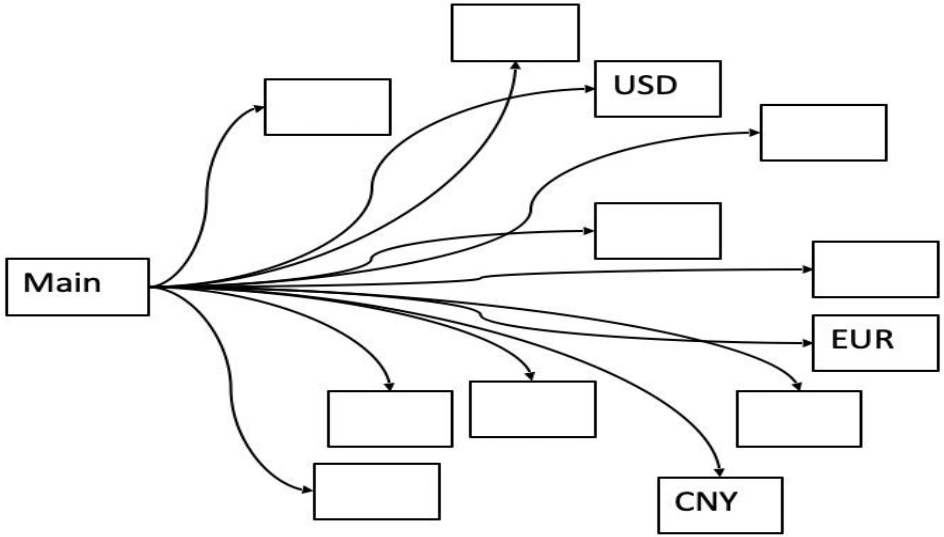
Code smell: Shotgun surgery



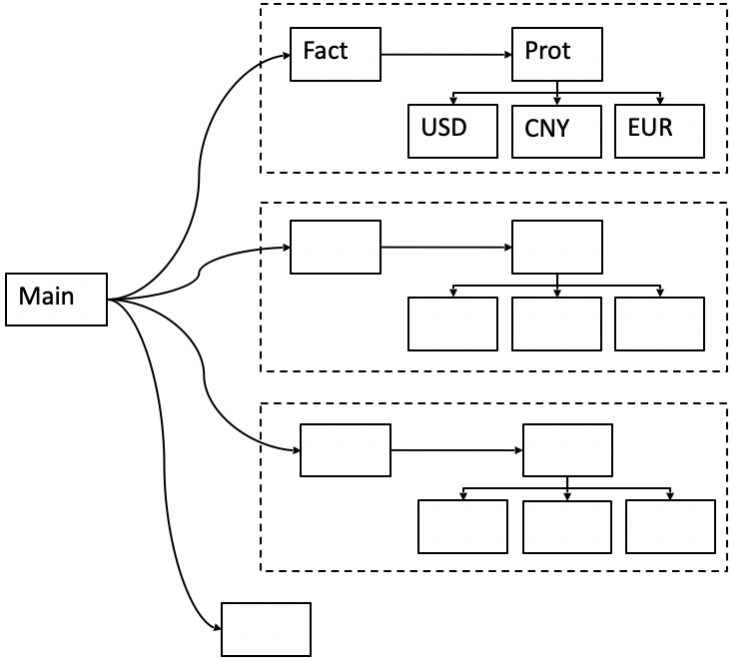
Design pattern: Prototype



Code smell en volume



Design pattern en volume



Conception #1

Un exemple de shotgun surgery

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

        if (args.length > 0) {
            for (String typeCash : args) {
                if (typeCash != null) {
                    if(typeCash.equals("EUR")){
                        EUR E = new EUR();
                        E.execute();
                    }else if(typeCash.equals("USD")){
                        USD E = new USD();
                        E.execute();
                    }else if(typeCash.equals("CNY")){
                        CNY E = new CNY();
                        E.execute();
                    }else if(!typeCash.equals("CNY")&& !typeCash.equals("USD") && !typeCash.equals("CNY")){
                        System.out.println(typeCash + ": doesn't exist");
                    }
                }

            }
        }else{
          System.out.println("Run again with arguments of command string ");
    }

 }

}
```

```java
class USD{
    private String money = "USD";

    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

```java
class EUR {
    private String money = "EUR";

    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

```java
class CNY {
    private String money = "CNY";

    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

Exemples de réalisations des prototypes

Conversion en Prototype

```java
public class PrototypeDemo {
    public static void main(String[] args) {
        if (args.length > 0) {
            initializeCurrencies();
            List<Prototype> currencies = new ArrayList<>();

            for (String currencyName : args) {
                Prototype currency = CurrencyModule.createPrototype(currencyName);
                if (currency != null) {
                    currencies.add(currency);
                }
            }
            for (Prototype c : currencies) {
                c.execute();
            }
        } else {
            System.out.println("Run again with arguments of command string ");
        }
    }


    public static void initializeCurrencies() {
        CurrencyModule.addPrototype(new PrototypeUSD());
        CurrencyModule.addPrototype(new PrototypeEUR());
        CurrencyModule.addPrototype(new PrototypeCNY());
    }
}
```

```java
class CurrencyModule {

    private static List<Prototype> currencies = new ArrayList<>();


    public static void addPrototype(Prototype c) {
        currencies.add(c);
    }

    public static Prototype createPrototype(String money) {
        for (Prototype c : currencies) {
            if (c.getCash().equals(money)) {
                return c.clone();
            }
        }
        System.out.println(money + ": doesn't exist");
        return null;
    }
}
```

```java
import java.util.ArrayList;
import java.util.List;


interface Prototype {
    Prototype clone();
    String getCash();
    void execute();
}
```

```java
class PrototypeUSD implements Prototype {
    private String money = "USD";

    @Override
    public Prototype clone() {
        return new PrototypeUSD();
    }

    @Override
    public String getCash() {
        return money;
    }

    @Override
    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

```java
class PrototypeCNY implements Prototype {
    private String money = "CNY";
    @Override
    public Prototype clone() {
        return new PrototypeCNY();
    }

    @Override
    public String getCash() {
        return money;
    }

    @Override
    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

```java
class PrototypeEUR implements Prototype {
    private String money = "EUR";

    @Override
    public Prototype clone() {
        return new PrototypeEUR();
    }

    @Override
    public String getCash() {
        return money;
    }

    @Override
    public void execute() {
        System.out.println(money + ": buy something");
    }
}
```

Conclusion