Prototype パターン

Seeeeee:D　夏休み勉強会

デザインパターンの巻

Ishijima Nanaka

Prototype パターンとは

- 生成が複雑なインスタンスをコピーして初期化する手法。
- 最初に生成したインスタンスをプロトタイプとして登録し、
  これをコピーすることで楽に速く同じものを作る。
- クラスからインスタンスをつくるのではなく、インスタンスをコピーすることで、インスタンスから別のインスタンスをつくるというもの。

ダメな例

```
public class Teacher{
    public Paper[] createManyCrystals(){
        Paper[] papers = new Paper[100];
        for(int n=0; n<papers.length ; n++){
            drawCrystal(papers[n]);  // 時間がかかる
            cutAccordanceWithLine(papers[n]);  // 時間がかかる
        }
        return papers;
    }
    private void drawCrystal(Paper paper){
        // きれいに描くため時間がかかる
    }
    private void cutAccordanceWithLine(Paper paper){
        // 描かれた線に沿ってきれいに切るには時間がかかる
    }
}
```

紙を雪の結晶の形に切り抜いたものを100枚作りたい。

100枚ひとつひとつに

・形を書く（drwaCrystal）
・切る（cutAccordance~）

とやっていると、時間がかかる

→最初の１枚をなぞって作ったほうがはやそう

パターンを適用

```
public class Paper implements Cloneable{
    private String name;
    public Paper(){}
    public Paper(String name){
        this.name = name;
    }
    public Cloneable createClone(){
        Paper newPaper = new Paper();
        newPaper.name = this.name;
        // newPaper と this を重ねて、this の形に切り抜く
        return newPaper;
    }
}
```

cloneメソッドをPaperクラスに作る

新しい紙を用意する・書く・切るの部分をcloneメソッドに実装

パターンを適用

```
public class Teacher{
    public Paper[] createManyCrystals(){
        Paper prototype = new Paper("雪の結晶");
        drawCrystal(prototype);
        cutAccordanceWithLine(prototype);

        Paper[] papers = new Paper[100];

        for(int n=0; n<papers.length ; n++){
            papers[n] = (Paper)prototype.createClone();
        }
        return papers;
    }
                                .
                                .
```

最初の一枚だけ作ったら、
あとは、100枚の紙をcloneメソッドにかけるだけ

速いね！楽だね！

でも・・・

よくよく見るとちゃんと複製できてなくない？

```
public class Teacher{
    public Paper[] createManyCrystals(){
        Paper prototype = new Paper("雪の結晶");
        drawCrystal(prototype);
        cutAccordanceWithLine(prototype);

        for(int n=0; n<papers.length ; n++){
            papers[n] = (Paper)prototype.createClone();
        }
```

```
    public Cloneable createClone(){
        Paper newPaper = new Paper();
        newPaper.name = this.name;
        // newPaper と this を重ねて、this の形に切り抜く
        return newPaper;
    }
```

もっと簡単なコードで確認してみる

そもそも…

コピーしたいなら代入すればいいじゃないか説

- `a = b` みたいな代入だけだと、オリジナルのオブジェクトを参照するだけ
- コピーのオブジェクトに変更を加えると、オリジナルのオブジェクトまで変わってしまう
- 浅いコピー(shallow copy)と深いコピー(deep copy)

詳しくはこのあとの実行結果で

実装方針：C#

ーーー

- Icloneableというインターフェースを持ったクラスを作る
- Icloneableなクラスの中にcloneメソッドを作る
- cloneメソッドにインスタンスをコピーする実装を書く

インターフェースとは：
抽象メソッドだけを持つクラスのようなもの。なんか、クラスの仕様規約みたいな。Icloneableをつけると、cloneメソッドを持たなければならないという制約がつく=cloneメソッドを持っていることが明示的になる。
（C#の機能で、Rubyにはないっぽい）

```
public class Paper : ICloneable
    {
        //コンストラクタ
        public Paper(string name)
        {
            Name = name;
            Vertexes = new List<Tuple<float, float>>();
        }

        //頂点座標のタプルをリストにする
        public List<Tuple<float, float>> Vertexes { get; set; }

        public string Name { get; set; }

        //cloneメソッド
        public object Clone()
        {
            return new Paper(Name) { Vertexes = new List<Tuple<float, float>>(Vertexes) };
        }
    }
```

このパターンのポイントは
clone()のところ

あとはmainでcloneメソッド
を使うだけ

Visual Studioでコードと実行結果を確認

まとめ

- prototypeパターンは深いコピーがしたい時の話
- 深いコピーは実装が難しい
- 状態を持つような(生成からその状態に持っていくのに手間がかかる)オブジェクトに対して有効
- ただ複製したいだけであればFactoryMethodパターンのほうがふさわしい