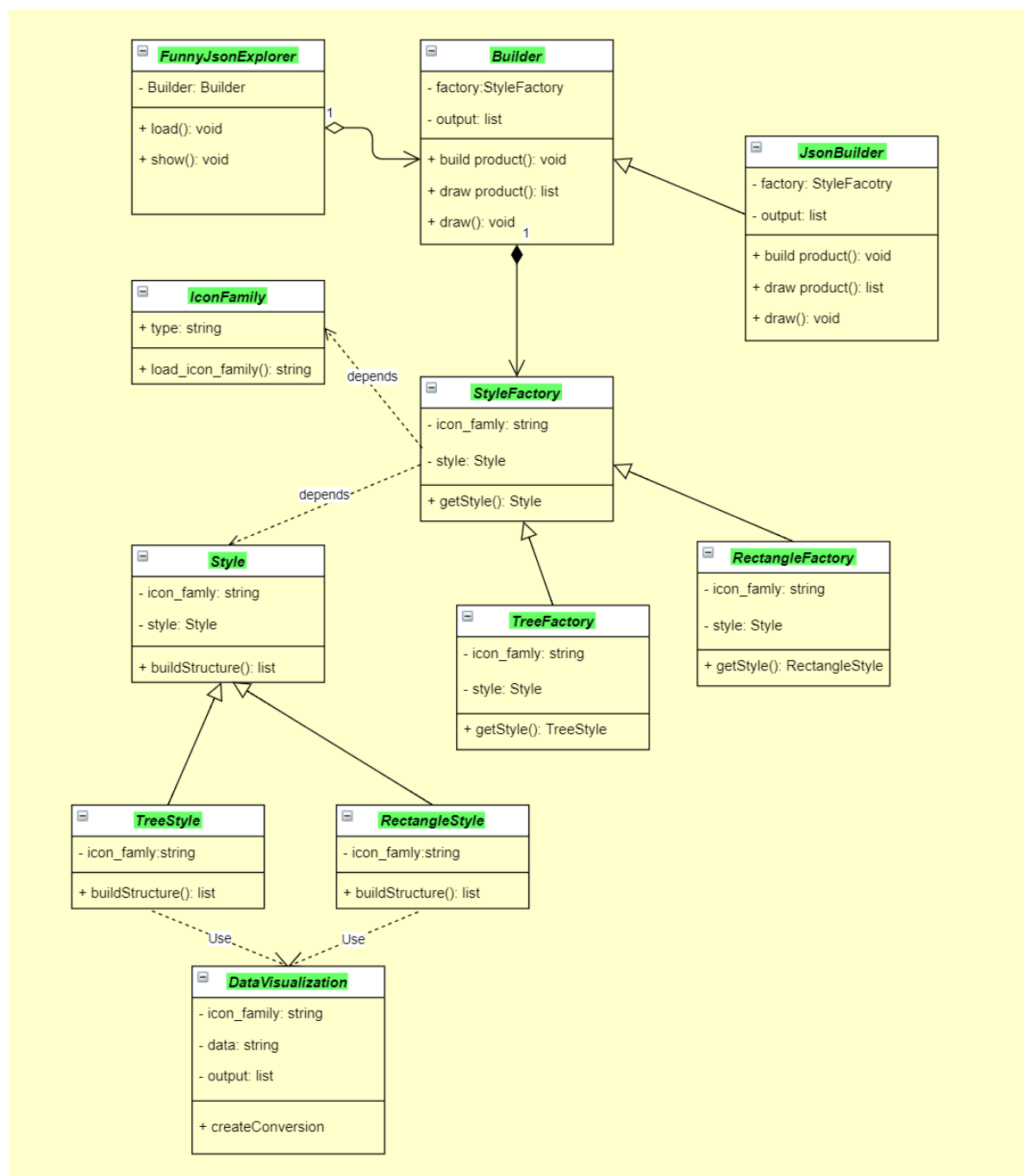


# 1. 设计文档

## 1.1 类图



## 1.2 相关说明

使用到的设计模式和对应的说明如下：

(1) **工厂方法**：在 `factory.py` 文件内，我创建了一个工厂基类 `StyleFactory`，根据传入的参数 `style` 来动态地创建并返回不同的样式实例（例如 `TreeStyle` 或者 `RectangleStyle`），具体工厂 `TreeStyle` 和 `RectangleStyle` 重写了父类中的 `getStyle` 方法分别创建了不同实例；同时 `style` 类也定义了所有风格的接口，具体的风格类可以继承自 `style`。

(2) **抽象工厂**：在 `DataVisualization` 类、`StyleFactory` 类和 `Builder` 类中都能得到体现，只需要创建一个接口，而不需要指定他们具体的类。

**(3) 建造者模式：**在codeBuilder.py和fje.py文件下体现建造者模式，`Builder`类充当抽象建造者，定义了产品构建的基本步骤，然后在具体的建造者类`JsonBuilder`中进行实现。在main函数内，建造者模式代码如下：`JsonBuilder`类作为`Builder`，提供了构建产品的具体实现；`FunnyJsonExplorer`作为`Director`，负责管理建造者，知道建造者按照特定的顺序和规则来构建产品；处理的json数据作为`Product`；main函数作为客户端，负责创建`Builder`和`Director`的实例并构建和展示产品。

```
styleFactory = StyleFactory(args.icon, args.style)
factory = styleFactory.getFactory()

builder = JsonBuilder(factory)
director = FunnyJsonExplorer(builder)
director.load(data)
director.show()
```

**(4) 组合模式：**`StyleFactory`类依赖`Style`类和`IconFamily`类来进行实现，同时`Builder`是由`StyleFactory`来组合实现的，体现了组合模式。

### 1.2.1 添加新的抽象工厂即可实现新风格

在factory.py文件内的`StyleFactory`文件添加判断：

```
class StyleFactory:
    def __init__(self, icon_family, style):
        self.icon_family = icon_family
        self.style = style

    # Factory Pattern
    def getStyle(self):
        if self.style == 'tree':
            return TreeStyle(self.icon_family, self.style)
        elif self.style == 'rectangle':
            return RectangleStyle(self.icon_family, self.style)
        else:
            print('Invalid style')
            return None
```

然后加入自己的新抽象工厂，加入我要实现一个圆形，那么就添加一个`CircularFactory`类和`CircularStyle`类，类似的定义如下：基本上只要进行简单的复制粘贴操作和修改类名称、函数名即可，可以轻松实现扩展。

```
class CircularFactory(StyleFactory):
    def __init__(self, icon_family, style):
        super().__init__(icon_family, style)

    def getStyle(self):
        return CircularFactory(self.icon_family, self.style)

class CircularStyle(Style):
    def __init__(self, icon_family, style):
        super().__init__(icon_family, style)

    def buildStructure(self, codeData):
        # print("buildStructure:", self.icon_family)
```

```

tree = CircularVisualization(icon_family=self.icon_family)
tree.createConversion(codeData)
tree.rebuildOutput()
return tree.output

```

最后在codeVisualization.py文件内，根据你要实现的风格加入CircularVisualization类，然后实现格式化和输出功能即可。总的来说，如果我们要添加新的风格类，实际上要编写的代码只有输出格式的逻辑处理，其他部分都已经完成了，只需要简单添加声明即可。

```

class CircularVisualization(DataVisualization):
    def __init__(self, icon_family, data=None):
        super().__init__(icon_family, data)
        self.output = []

    def createConversion(self, data, level=0, parent_last=[]):
        pass

    def rebuildOutput(self):
        pass

```

### 1.2.2 通过配置文件添加新的图标族

只需要在iconFamily.py文件内添加我们所需要的图标族即可，然后在运行的时候指定所使用的图标。

```

icon_families = {
    'poker-face-icon-family': ("♦", "♠"),

    # 可以在这添加更多的图标族
    'emoji-icon-family': ("😄", "😞"),
    'fruit-icon-family': ("🍊", "🍋"),
    'animal-icon-family': ("🐶", "🐱"),
    # ...
}

```

## 2. 运行截图

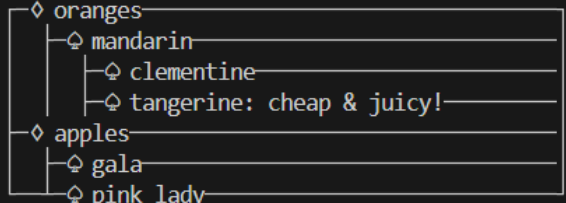
(1) 使用poker-face-icon-family的两种风格展示:

```

(base) PS D:\sE_hw\Design_pattern> fje -f Sample.json -s tree -i poker-face-icon-family
The Middle icon is: ♦
The Leaf icon is: ♠
├─♦ oranges
│   └─♠ mandarin
│       ├──♠ clementine
│       └─♠ tangerine: cheap & juicy!
└─♦ apples
    ├──♠ gala
    └─♠ pink lady

```

```
(base) PS D:\sE_hw\Design_pattern> fje -f Sample.json -s rectangle -i poker-face-icon-family
• The Middle icon is: ♠
The Leaf icon is: ♠
```



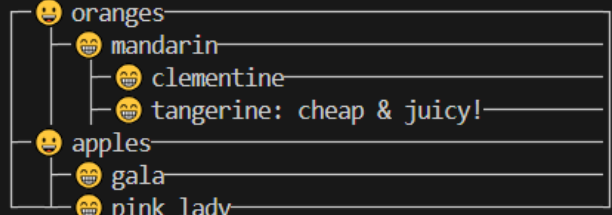
```

graph TD
    oranges[♠ oranges] --- mandarin[♠ mandarin]
    oranges --- apples[♠ apples]
    mandarin --- clementine[♠ clementine]
    mandarin --- tangerine["♠ tangerine: cheap & juicy!"]
    apples --- gala[♠ gala]
    apples --- pink_lady[♠ pink lady]

```

(2) 使用emoji-icon-family的两种风格展示:

```
(base) PS D:\sE_hw\Design_pattern> fje -f Sample.json -s rectangle -i emoji-icon-family
• The Middle icon is: 🍊
The Leaf icon is: 🍊
```

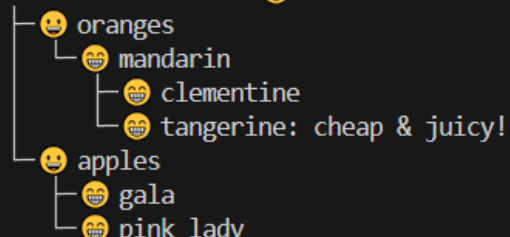


```

graph TD
    oranges[🍊 oranges] --- mandarin[🍊 mandarin]
    oranges --- apples[🍊 apples]
    mandarin --- clementine[🍊 clementine]
    mandarin --- tangerine["🍊 tangerine: cheap & juicy!"]
    apples --- gala[🍊 gala]
    apples --- pink_lady[🍊 pink lady]

```

```
(base) PS D:\sE_hw\Design_pattern> fje -f Sample.json -s tree -i emoji-icon-family
• The Middle icon is: 🍊
The Leaf icon is: 🍊
```



```

graph TD
    oranges[🍊 oranges] --- mandarin[🍊 mandarin]
    oranges --- apples[🍊 apples]
    mandarin --- clementine[🍊 clementine]
    mandarin --- tangerine["🍊 tangerine: cheap & juicy!"]
    apples --- gala[🍊 gala]
    apples --- pink_lady[🍊 pink lady]

```

### 3. 相关源代码库

<https://github.com/shenyun114/Funny-JSON-Explorer>