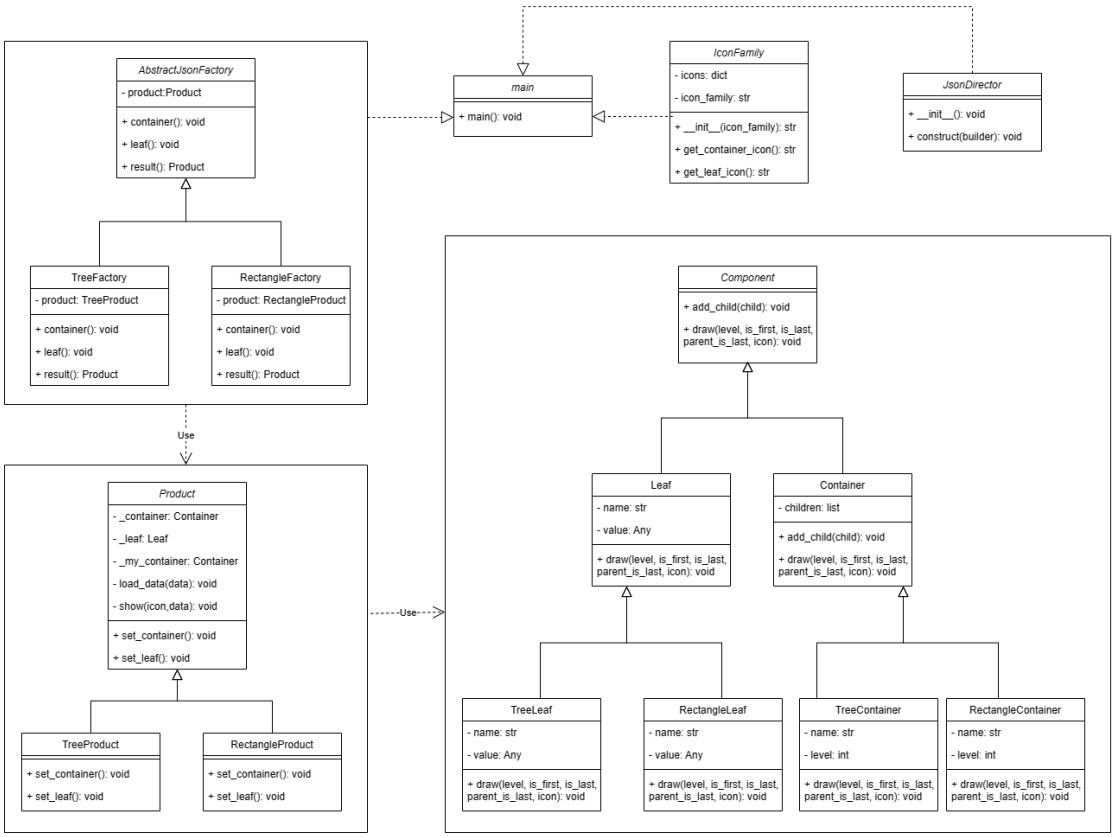
21307112 付芷怡

1.1 类图



类图说明：

Factory

container( ):抽象方法，用于创建 JSON 树中的容器节点

leaf( ):抽象方法，用于创建 JSON 树中的叶子节点

result( ):用于获取 JSON 树

Product

set_container( ): 抽象方法，用于设置容器类

set_leaf( ): 抽象方法，用于设置叶节点类

load_data(data): 加载 JSON 数据，并将其转换为容器和叶节点的树结构

show(icon, data): 显示 JSON 数据的可视化表示

TreeStyleProduct 和 RectangleStyleProduct

set_container( ): 用于设置具体的容器类

set_leaf( ): 用于设置具体的叶节点类

IconFamily

get_container_icon(): 用于获取容器节点的图标

get_leaf_icon(): 用于获取叶节点的图标

JsonDirector

construct(builder): 用于构建 JSON 树结构

main

main( )，主函数，用于解析命令行参数和加载 JSON 文件，并调用相应的工厂和构建器来显示 JSON 数据

1.2 设计模式说明

（1）工厂方法模式是一种创建型设计模式，它的主要目的是在不暴露对象创建逻辑的情况下，允许子类决定实例化哪个类。这种模式通过定义一个创建对象的接口，但是允许子类决定实例化哪个类来创建对象。通过定义抽象的 Product 类，并且在具体的产品类中实现 set_container 和 set_leaf 方法，以通过简单的扩展来支持新的展示风格而不需要修改现有代码，在 Factory.py 文件中，我通过 TreeFactory 和 RectangleFactory 实现了创建不同风格的产品实例，符合工厂方法模式的设计原则。

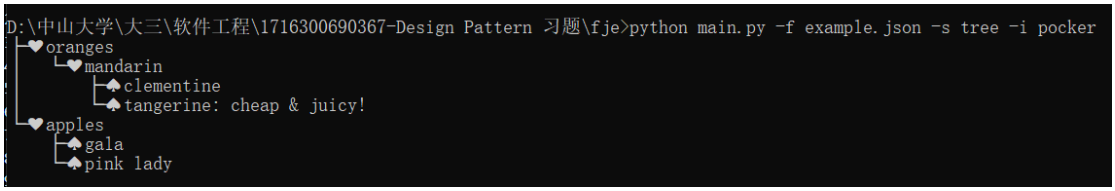
（2）抽象工厂模式，在 Factory.py 文件中，AbstractJsonFactory 作为抽象工厂接口，而 TreeFactory 和 RectangleFactory 则是具体的实现工厂。通过这种设计，可以灵活地展示新的展示风格。在这里添加新的展示风格只需要创建一个新的具体工厂类，就可以实现不需要修改现有代码而添加新的风格，使得程序易于维护和扩展。

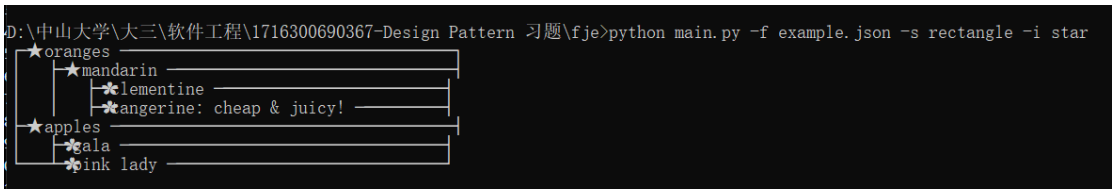
（3）建造者模式是一种创建型设计模式，旨在帮助创建复杂对象。该模式允许你一步一步地构造一个产品，使得构建过程可以独立于实际对象的组装方式，从而使相同的构建过程可以创建不同的表现形式。这有助于隐藏构建的细节，并允许创建不同表示的同一构建过程。在我写的代码中，建造者模式是由 AbstractJsonFactory 作为 Builder 接口，定义了构造 JSON 树结构的步骤方法和获取最终结果的方法。而 TreeFactory 和 RectangleFactory 是具体建造者去实现这些方法，用来创建特定风格的容器和叶子节点。而 JsonDirector 作为指挥者来直到具体建造者按照步骤来构建产品。最后 Product 类用于表示最终的 JSON 树结构，通过具体建造者创建的容器和叶子节点组合成完整的产品。这样的设计方式确保了构建过程的灵活性和可扩展性，因此只需要实现新的具体建造者而无需修改现有的代码逻辑，就可以增加新的展示风格。

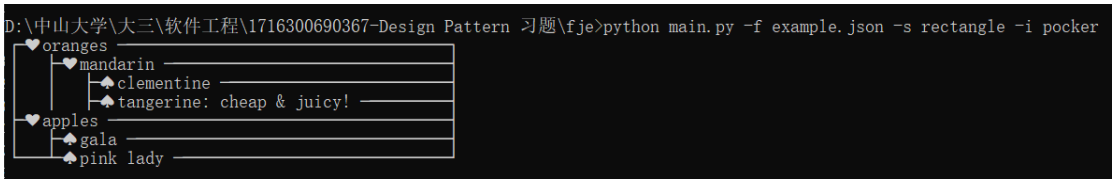
（4）组合模式是一种结构型设计模式，它允许将对象组合成树形结构以表示“部分-整体”的层次结构。通过使用组合模式，客户端可以统一处理单个对象和对象组合，而不需要区分它们之间的差异。组合模式通过三个关键角色实现：抽象构件、叶子构件、容器构件。抽象构件定义了叶子构件和容器构件对象的公共接口，其中包含了所有子类共有的行为。叶子构件表示 JSON 结构中的叶子节点，而容器构件表示 JSON 结构中的容器节点，可以包含多个子节点，这些子节点可以是叶子节点或其他容器节点。

若要添加新的风格，通过抽象工厂模式就可以实现展示风格的灵活扩展。这一设计保证了在不改变现有代码的情况下，只需添加新的抽象工厂，就能够轻松地引入新的展示风格。实现新的展示风格：新增一个具体的抽象工厂并关联新增一个产品。而若要添加新的图标族，只需要在 config.json 中按照格式加入 icon_container 和 icon_leaf 的图标，并将名字放入主函数中即可。

2.运行截图







3.源代码库：https://github.com/chuanyunbaihe/Funny-JSON-Explorer.git