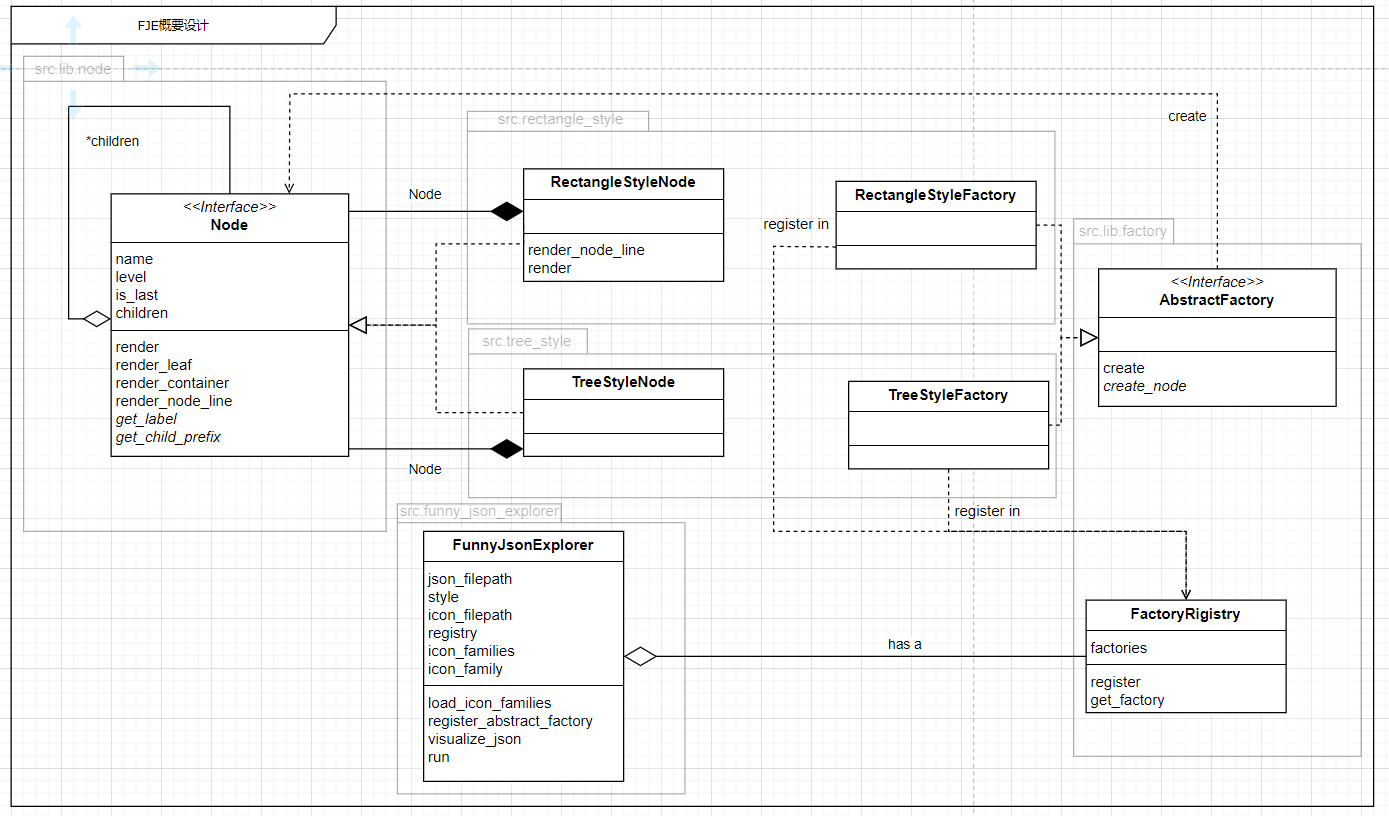
（建议使用PDF的书签进行跳转以提升阅读体验，图片可放大）

# 1.领域模型

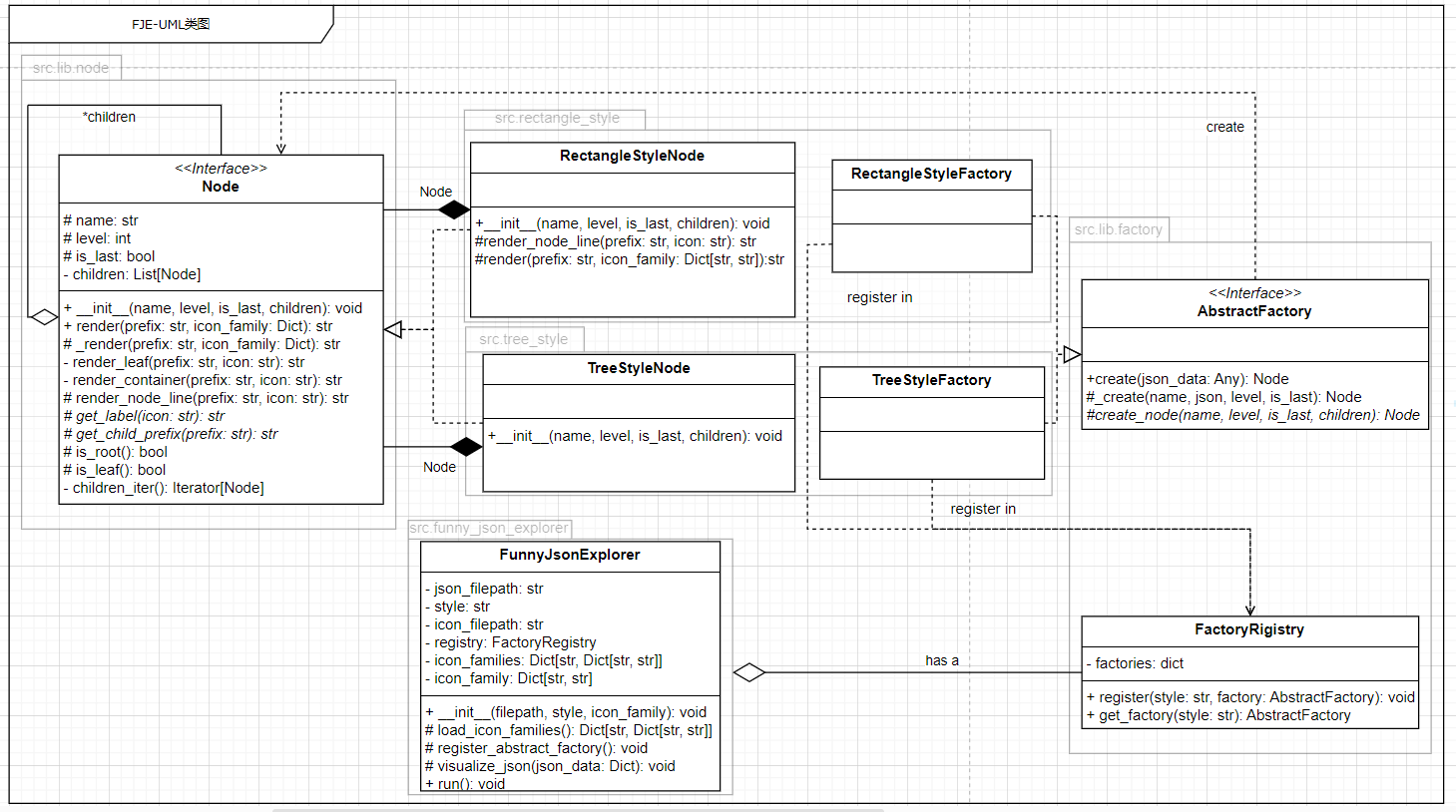
1、由于UML与开发语言无关，隐藏了开发语言的细节，所以尽管代码实现中存在相当大量的用于表示私有的或约定为受保护的变量或函数的单/双下划线前缀，但在领域模型（以及后文展示的UML类图）中隐去了这些用于表示访问权限的前缀。（单下划线前缀约定为protected，双下划线前缀为private）

2、参考老师的课堂示例，隐去了一些不重要的函数例如“is\_root”等，也根据惯例，隐去了构造函数；也根据惯例，隐去了子类实现的行为与父类（抽象类）的虚函数相关的函数。



# 2.UML类图

为了处理一些函数签名过长的情况，我选择将签名过长的函数的形参的数据类型略去。与上文同理隐去了单/双下划线前缀，除非隐去前缀会导致函数名重复。我还隐去了一些不重要的函数例如“is\_root”等，也根据惯例，隐去了构造函数，除非这个构造函数初始化了成员变量；也根据惯例，隐去了子类实现的行为与父类（抽象类）的虚函数相符的函数。



# 3.说明：使用的设计模式及作用

## 3.1工厂方法（Factory）

AbstractFactory的\_create是Factory模式的方法。

具体而言，AbstractFactory的\_create调用\_create\_node，而\_create\_node是抽象函数，这使得子类调用\_create方法时，可以根据自己子类实现的\_create\_node来实例化TreeStyleNode或RectangleStyleNode对象，实现让类的实例化推迟到其子类中进行。

在本程序中工厂方法（Factory）的作用是让类的实例化推迟到其子类中进行，由子类决定创建的对象是TreeStyleNode还是RectangleStyleNode，也封装了创建逻辑，将对象的创建与对象的实现解耦。

## 3.2抽象工厂（Abstract Factory）

AbstractFactory是抽象工厂接口，而TreeStyleFactory和RectangleStyleFactory是具体的抽象工厂，它们一起实现抽象工厂。（而Node就是抽象产品，TreeStyleNode和RectangleStyleNode就是具体产品）

具体而言，在AbstractFactory这一抽象工厂接口的\_create函数中，隐式地包含了两种同一族的产品的生产：Node类型（抽象产品）的叶子节点和Node类型（抽象产品）的中间节点。而对于具体的抽象工厂，不妨以TreeStyleFactory为例进行说明：其继承了抽象类AbstractFactory的\_create函数，于是也类似地隐式地包含了两种同一族的产品：TreeStyleNode类型（具体产品）的叶子节点和TreeStyleNode类型（抽象产品）的中间节点；而且TreeStyleFactory这一具体的抽象工厂确实可以生产这两种同一族的具体产品，因为TreeStyleFactory实现了\_create\_node函数，运行时不会报错。同理可知RectangleStyleFactory的相关分析与上文同理。

在本程序中抽象工厂（Abstract Factory）的作用是AbstractFactory 定义了创建对象的一般方法（或者说是创建抽象产品的方法），而创建具体产品的细节留给了具体的抽象工厂实现，于是实现了接口抽象化。并且抽象工厂接口创建同一族的抽象产品（例如Node类型（抽象产品）的叶子节点和Node类型（抽象产品）的中间节点）、具体的抽象工厂也正如上文所分析的创建同一族的具体产品。

## 3.3建造者（Builder）

AbstractFactory是Builder，其中AbstractFactory的\_create函数是Builder的结果方法，因为它返回的Node是根，相当于返回了完整的结果对象。而AbstractFactory的\_create函数和create\_node函数就是Builder的部分方法（创建产品对象不同部件的方法），其中create\_node创建的部件是节点Node的基本属性，包括name、level、is\_last，以及children列表的最终赋值，而\_create函数创建的部件是children列表（换句话说，children列表这一部件主要是由\_create函数创建的，只是最终的赋值交给create\_node函数执行）。

而且TreeStyleFactory和RectangleStyleFactory就是两个不同的具体的Builder，它们还重写了create\_node函数（提供了构造过程的不同实现，即创建不同类型的产品/部件）。

在本程序中建造者（Builder）的作用是通过上文所述的多个部分方法分步骤创建复杂（存在递归关系的）Node对象，并且可以使用相同的创建代码即\_create函数和create\_node函数生成不同类型的对象（例如TreeStyleNode和RectangleStyleNode）

## 3.4组合模式（Composition）

Node同时是中间节点和叶子节点，或者说Node同时是叶子Leaf和容器Container。

这是因为Node实现了叶子节点和中间节点各自对应的行为/工作（也即业务方法operation），当Node是叶子节点时，其渲染（render）执行的是叶子节点对应的渲染动作（render\_leaf），相应地当Node是中间节点时，其渲染（render）执行的是中间节点对应的渲染动作（render\_container）。并且作为中间节点（容器）的Node可以通过children这一列表包含叶子节点或其他中间节点（容器）。

在本程序中组合模式的作用是实现树状对象结构，并且以相同方式处理简单和复杂元素，例如Node的渲染都是执行render函数。

## 3.5可扩展性和可维护性的相关说明

因为实现了工厂方法、抽象工厂、建造者、组合模式，以及registry模式（因为FactoryRegistry是Registry模式的实现）、Template模式（模板方法模式，因为Node的render函数是Template模式中的整体（play）方法，定义了算法的框架，而Node的子类RectangleStyleNode在不修改算法结构的前提下修改了算法的特定步骤例如render\_node\_line函数）；所以我合理运用了这些设计模式增强了本程序的可扩展性和可维护性，并且变量名和函数名的合理命名增强了代码可读性，并且将不同的类分别放置在不同的文件和目录中实现以代码的结构化程度；所以综上本程序的可扩展性和可维护性是很好的。

由上一自然段所阐述的我所实现的一系列设计模式，可显然推导得到我已经完成了必做任务：不改变现有代码，只需添加新的抽象工厂，即可添加新的风格。

# 4.运行截图（完成了题目所需功能）

详见文件夹“运行截图”，我实现了2种风格（树形、矩形），3种图标族（none、poker\_face、emoji），因此有共计6次运行fje的截图，这完成了题目 “两种风格、两种图标族” 的要求，也实现了选做任务：通过配置文件，可添加新的图标族（例如emoji图标族）。

注意需要依赖库toml等，因此建议在conda命令行中激活了合适的环境再执行

我的截图的指令是利用pycharm这一IDE内置的命令行执行的，这样做是因为

1、unicode制表符在windows内置命令行上的视觉效果不好、对齐不良。

2、考虑到老师在演示示例时，也是直接使用IDE内置的命令行

3、题目要求本程序是命令行界面小工具，而对于截图说明的要求中并未强调不可使用IDE内置的命令行

# 5.源代码库：公开可访问的Github repo URL

https://github.com/Xiebt3/21307352\_design.git

（注：此网站当前为私有仓库，等到提交时间截止后才公开）