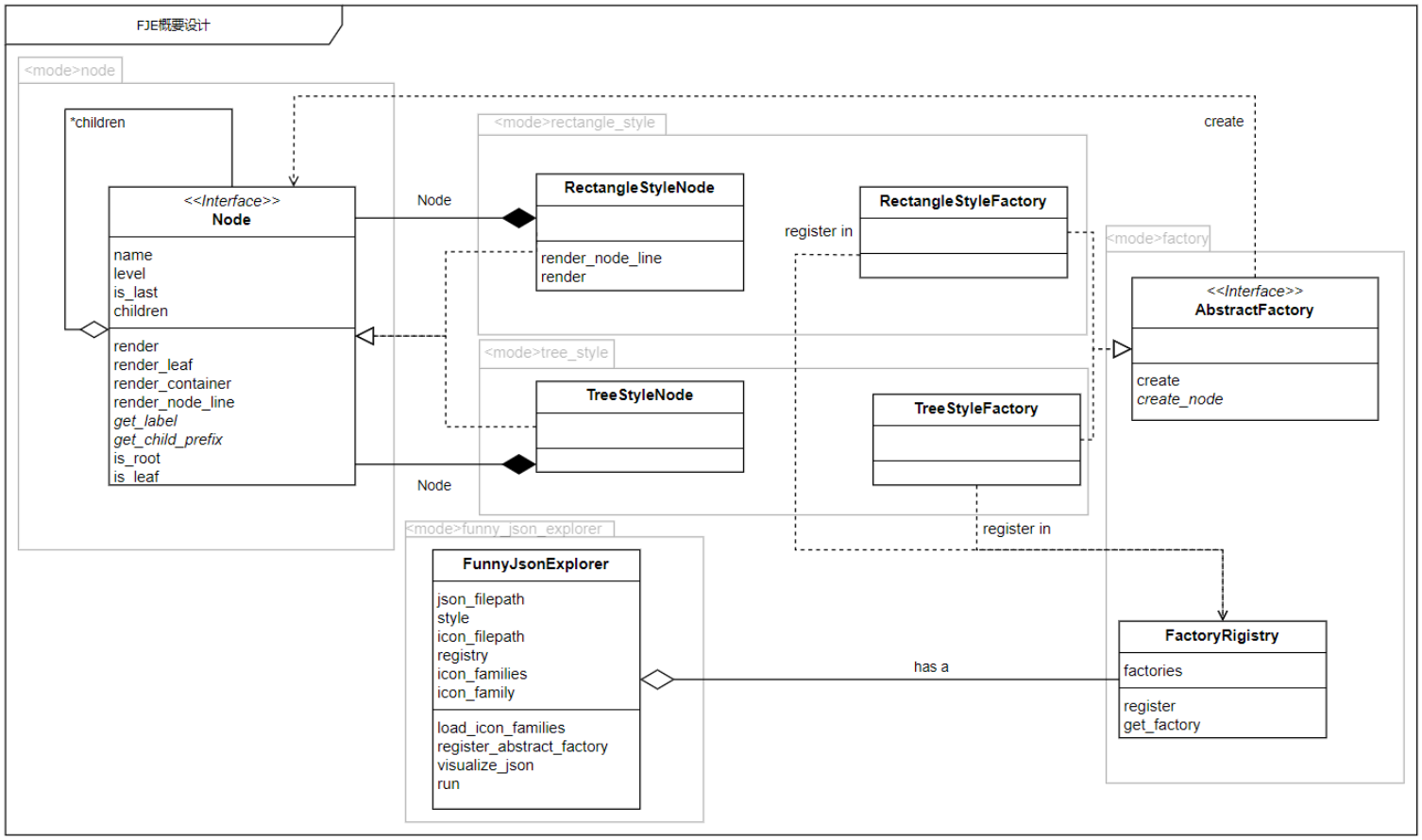
（建议使用 PDF 的书签进行跳转以提升阅读体验，图片可放大）

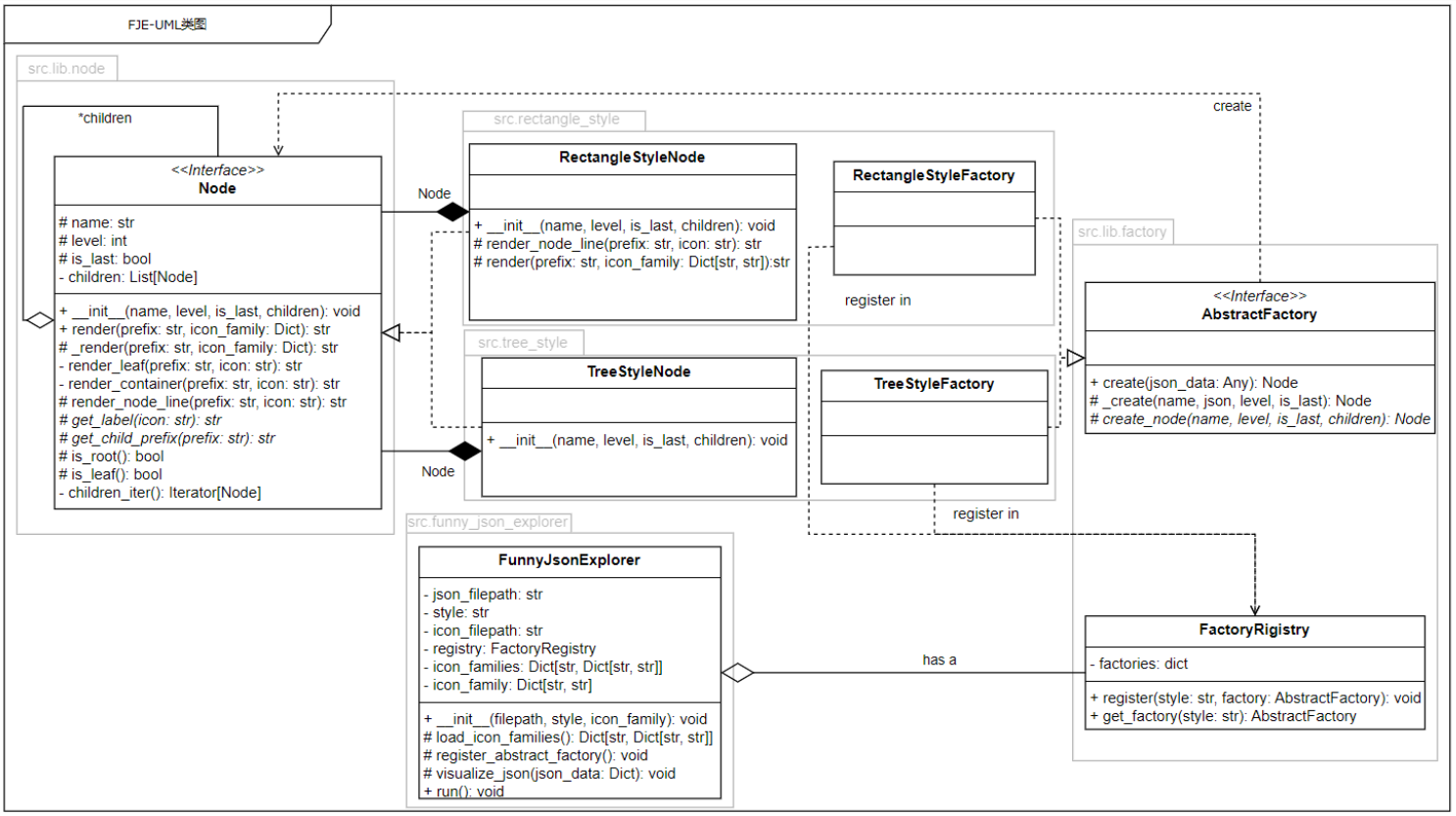
1. 领域模型

1、由于领域模型和 UML 类图是与开发语言无关的，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双/单下划线前缀，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“children_iter”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。
3、由于当父类是抽象类时，子类与父类的关系既有继承关系又有实现关系，为了表现工厂方法的特点，我突出了实现关系。



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。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 和 FactoryRegistry 是工厂方法。这是因为 AbstractFactory 的_create 是 Factory 模式的方法。以及 FactoryRegistry 的 get_factory 是简单工厂模式的方法，而某些情况下可以将简单工厂模式看作工厂方法的一种特例，所以 FactoryRegistry 也是工厂方法。

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。FactoryRegistry 的 get_factory 函数定义的返回值是抽象产品，但运行时根据输入的 style 决定返回的具体产品，也实现了让类的实例化推迟的作用（工厂方法可以返回已有对象而不一定必须创建新的实例）。

在本程序中工厂方法（Factory）的作用是让 Node 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；以及在 FactoryRegistry 的 get_factory 函数让 AbstractFactory 类的实例化由输入决定；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（具体产品）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（建造者）。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 实现了组合模式。这是因为 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。

具体而言，Node 实现了叶子节点和中间节点各自对应的行为/工作，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（叶子节点和中间节点），例如 Node 的渲染都是执行 render 函数，无论这个 Node 是叶子节点还是中间节点。

3.5 registry 模式（非 GoF）

FactoryRegistry 是 Registry 模式的实现。

本程序中 registry 模式的作用是集中管理工厂对象，以此简化代码逻辑。

3.6 Template 模式

Node 和 AbstractJsonFactory 实现了 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，render_leaf 和 render_container 是步骤（step）方法。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，而这个函数是被 render_leaf 和 render_container 两个步骤方法调用的，所以相当于修改了算法的特定步骤。而 AbstractJsonFactory 的 create 函数是 Template 模式中的整体（play）方法，

在本程序中 Template 模式的作用正如上文所述的在超类 Node 和 AbstractJsonFactory 中定义算法的框架（分别对应 Node 的 render 函数和 AbstractJsonFactory 的 create 函数），允许子类在不修改结构的情况下重写算法的特定步骤。

3.7 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、registry 模式（非 GoF）、Template 模式；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以增强代码的结构化程度；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，尤其是抽象工厂，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（具体的）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 内置的命令行终端执行的，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）