**评估(对设计原则中的模块化、接口、信息隐蔽、增量式开发、抽象与通用性这几个方面进行描述)**

负责人：孟小凡

负责模块：求职者（应聘者）模块

**模块化：**

|  |  |
| --- | --- |
|  | 概况描述 |
| 耦合度 | 企业模块实现了较低耦合度  Company 类和 Job 类之间的耦合度较低，它们通过 Company 类中的 jobs 列表进行连接。这种松散的耦合度使得 Company 类和 Job 类可以相对独立地进行修改和扩展。  Job 类和 Applicant 类之间的耦合度较低，Job 类通过 applicants 列表与 Applicant 类进行连接。这种设计允许多个求职者申请同一个职位，而不需要对 Job 类进行修改。  Company 类和 Applicant 类之间没有直接的耦合，它们之间的交互主要是通过 Job 类作为中介。这种设计使得 Company 类和 Applicant 类可以相对独立地操作。 |
| 内聚度 | 内聚度较高  Company 类负责管理企业的基本信息和工作职位列表，并提供添加、删除、更新职位的功能。这种职责相关性使得 Company 类具有较高的内聚度。  Job 类负责管理工作职位的详细信息和申请者列表，并提供添加、删除申请者的功能。这种职责相关性使得 Job 类具有较高的内聚度。  Applicant 类负责管理求职者的基本信息和申请职位的操作，包括申请职位和撤回申请。这种职责相关性使得 Applicant 类具有较高的内聚度。 |

**接口：**

Company 类接口：

\_\_init\_\_(self, name, address, industry): 初始化一个企业对象。

add\_job(self, job): 将一个 Job 对象添加到企业的职位列表中。

remove\_job(self, job): 从企业的职位列表中移除一个 Job 对象。

update\_job(self, job, new\_details): 更新指定工作职位的信息。

get\_jobs(self) -> List[Job]: 获取企业的工作职位列表。

Job 类接口：

\_\_init\_\_(self, title, description, requirements, salary): 初始化一个工作职位对象。

add\_applicant(self, applicant): 将一个 Applicant 对象添加到工作职位的申请者列表中。

remove\_applicant(self, applicant): 从工作职位的申请者列表中移除一个 Applicant 对象。

get\_applicants(self) -> List[Applicant]: 获取工作职位的申请者列表。

Applicant 类接口：

\_\_init\_\_(self, name, email, phone): 初始化一个求职者对象。

apply\_for\_job(self, job): 申请一个工作职位。

withdraw\_application(self, job): 撤回对一个工作职位的申请。

通过这些接口，企业模块提供了对企业、工作职位和求职者的基本操作。例如，可以使用 Company 类的 add\_job() 方法添加工作职位，使用 Job 类的 add\_applicant() 方法将求职者添加到特定职位的申请者列表中。还可以使用 Applicant 类的 apply\_for\_job() 方法来申请一个工作职位，或使用 withdraw\_application() 方法撤回申请。

**信息隐蔽：**

优点：

封装性较好，代码结构使用了类和方法的封装，将企业、工作职位和求职者等相关信息封装在各自的类中。这样可以减少直接访问和修改数据的可能性，提高了信息的隐蔽性和安全性。

实现了访问控制，通过类的方法来操作数据，可以对数据访问进行控制和限制。例如，在企业对象中，添加了 add\_job、remove\_job 和 update\_job 等方法，可以对工作职位的管理进行限制，避免直接对数据进行修改。

缺点：

数据访问限制不完善：当前的代码结构并没有提供严格的数据访问限制，导致外部仍然可以直接访问和修改数据。可以考虑在类的属性前加上私有属性标识符并提供相关的访问方法，但又增加了传输困难。

缺乏验证和授权：代码结构中没有明确的验证和授权机制来确保只有授权的用户才能访问和修改数据。对于招聘系统来说，确保只有企业用户才能对工作职位进行管理和修改是非常重要的，这一点在当前代码结构中并未体现。

安全性考虑不足：当前的代码没有考虑到数据的安全性问题，如密码加密、防止 SQL 注入等。在招聘系统中，保护用户的敏感信息和系统的安全是至关重要的。

**增量式开发：**

Company\_otherinf（）

使用图：

company\_creat（）

company（）

company\_job\_select（）

company\_control（）

applicant\_list（）

job（）

job\_control（）

job\_inform（）

**抽象：**

代码使用类来表示company、job具有清晰的关系和交互模式，易于理解和扩展。

每个类都有自己的属性和方法，对外部代码隐藏了内部实现细节，提高了代码的安全性和可维护性。但缺乏输入验证和错误处理，需要对输入数据进行验证，以确保数据的有效性和完整性，并处理可能出现的错误和异常情况。

**通用性：**

该部分通用性较为良好

模块化设计，将企业、工作职位和求职者分别封装在各自的类中，实现了模块化设计。这样可以提高代码的可维护性和可扩展性，使得系统更易于理解和修改。

通过类的方式，可以灵活地创建和管理多个企业、工作职位和求职者实例。这种灵活性使得系统能够适应不同规模和类型的企业招聘需求。且容易扩展，可以很容易地添加新的功能或属性。例如，可以添加更多的方法来处理企业的招聘流程，同时，可以为企业、工作职位和求职者添加更多的属性以满足特定需求。