**题目：**

开发一个系统帮助业务部门实现灵活的奖金计算。对于普通员工，主要有个人当月业务奖金、个人当月回款奖金等，对于部门经理，除了有普通员工的奖金外，还有团队当月业务奖金等。目前各奖金类别的计算规则如下：

个人当月业务奖金 = 个人当月销售额 \* 3%

个人当月回款奖金 = 个人当月回款额 \* 0.1%

团队当月业务奖金 = 团队当月销售额 \* 1%

考虑到业务部门要通过调整奖金计算方式来激励士气，系统应灵活地适应各种需求变化，例如，将来可能会增加个人业务增长奖金、团队当月回款奖金、团队业务增长奖金、团队盈利奖金等奖金类别，也可能增加新的员工岗位，或变更奖金计算规则等。请写出你所选择的设计模式，画出类图，并给出核心代码。

**解答：**

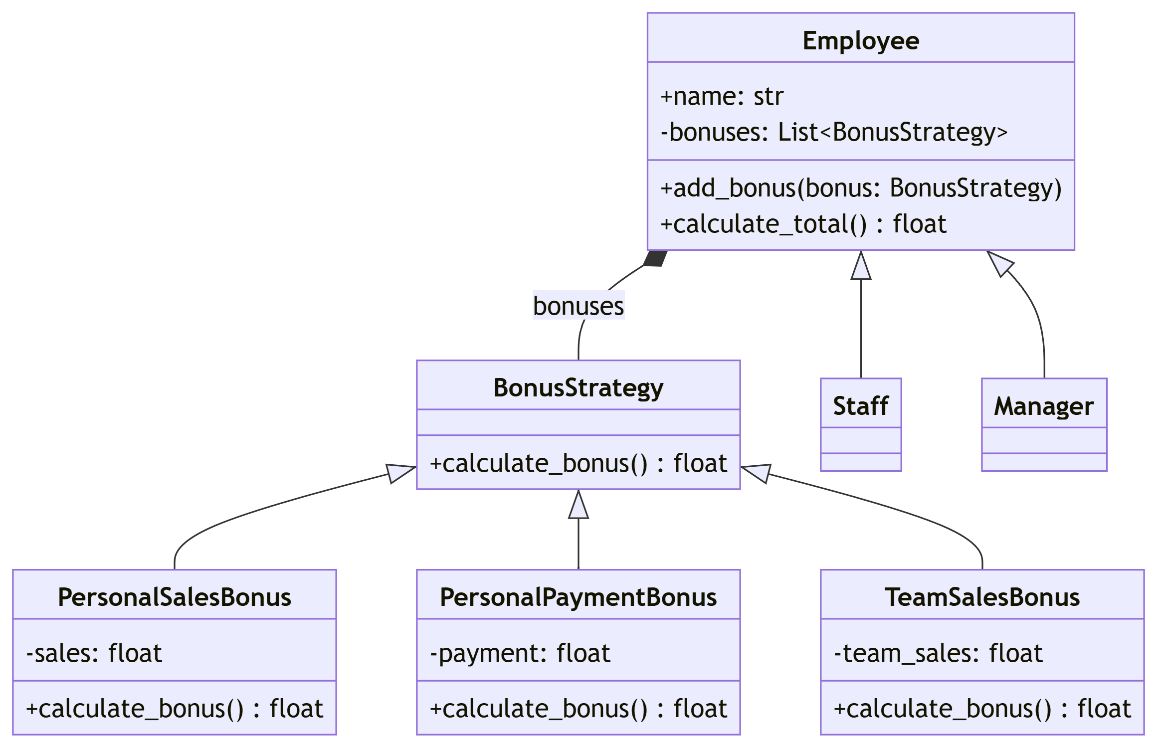
* **设计思路：**

1. 使用策略模式和组合模式的组合。
2. 策略模式可以将不同的奖金计算逻辑分离，定义一系列的算法，并让它们可以互换。这样可以轻松替换或增加新的奖金计算规则，而不会影响系统的其他部分。
3. 组合模式可以用于处理不同员工类型（如普通员工、部门经理）在奖金计算上的组合逻辑。例如，部门经理除了个人奖金，还包括团队奖金。组合模式可以处理奖金的层次结构，支持对个体和团队奖金的组合计算。

* **模式：**

策略模式和组合模式

**类图：**



|  |
| --- |
| * **核心代码：** |
| from abc import ABC, abstractmethod  *# 奖金策略的抽象基类*  class BonusStrategy(ABC):      @abstractmethod      def calculate\_bonus(self) -> float:          pass  *# 个人业务奖金*  class PersonalSalesBonus(BonusStrategy):      def \_\_init\_\_(self, sales: float):          self.sales = sales      def calculate\_bonus(self) -> float:          return self.sales \* 0.03  *# 个人回款奖金*  class PersonalPaymentBonus(BonusStrategy):      def \_\_init\_\_(self, payment: float):          self.payment = payment      def calculate\_bonus(self) -> float:          return self.payment \* 0.001  *# 团队业务奖金*  class TeamSalesBonus(BonusStrategy):      def \_\_init\_\_(self, team\_sales: float):          self.team\_sales = team\_sales      def calculate\_bonus(self) -> float:          return self.team\_sales \* 0.01  *# 员工类*  class Employee:      def \_\_init\_\_(self, name: str):          self.name = name          self.bonuses = []      def add\_bonus(self, bonus: BonusStrategy):          self.bonuses.append(bonus)      def calculate\_total(self) -> float:          total\_bonus = 0.0          for bonus in self.bonuses:              total\_bonus += bonus.calculate\_bonus()          return total\_bonus  *# 普通员工*  class Staff(Employee):      pass  *# 部门经理*  class Manager(Employee):      pass |