CSC148 - Inheritance: Extending the Employee Example

In the space below and on the next page, we've included a simplified version of the code for our Employee example from this week's prep readings. Your task is to extend this code to account for *personal days*, i.e., days when the employee did not come in to work. An employee may take at most *ten* personal days in a single year (you can decide what happens if an employee attempts to take a personal day when this limit is reached). Personal days have no effect on the monthly payment of salaried employees, but hourly employees lose *eight hours* of pay per personal day taken that month, down to a minimum of zero hours. Note that this is true regardless of how many hours per month that employee is scheduled to work.

Tackle this task in two steps, described below.

1. Discuss with your group how to modify the code to keep track of personal days, clearly identifying both attributes and methods to add/modify, and which class each modification should be made in. Once you agree, make the modifications.

2. Discuss with your group how to modify the code to take personal days into account when paying employees, following the same guidelines as the first step.

```python
class Employee:
    """An employee of a company.

    This is an abstract class. Only subclasses should be instantiated.

    === Attributes ===
    id_: This employee's ID number.
    name: This employee's name.
    """
    id_: int
    name: str

    def __init__(self, id_: int, name: str) -> None:
        """Initialize this employee.

        Note: This initializer is meant for internal use only;
        Employee is an abstract class and should not be instantiated directly.
        """
        self.id_ = id_
        self.name = name

    def get_monthly_payment(self) -> float:
        """Return the amount that this Employee should be paid in one month.

        Round the amount to the nearest cent.
        """
        raise NotImplementedError

    def pay(self, pay_date: date) -> None:
        """Pay this Employee on the given date and record the payment.

        (Assume this is called once per month.)
        """
        payment = self.get_monthly_payment()
        print(f'An employee was paid {payment} on {pay_date}.')
```

```python
class SalariedEmployee(Employee):
    """An employee whose pay is computed based on an annual salary.

    === Attributes ===
    salary: This employee's annual salary
    """
    salary: float

    def __init__(self, id_: int, name: str, salary: float) -> None:
        """Initialize this salaried Employee."""
        Employee.__init__(self, id_, name)
        self.salary = salary

    def get_monthly_payment(self) -> float:
        """Return the amount that this Employee should be paid in one month.

        Round the amount to the nearest cent.
        """
        return round(self.salary / 12, 2)
```

```python
class HourlyEmployee(Employee):
    """An employee whose pay is computed based on an hourly rate.

    === Attributes ===
    hourly_wage:
        This employee's hourly rate of pay.
    hours_per_month:
        The number of hours this employee works each month.
    """
    hourly_wage: float
    hours_per_month: float

    def __init__(self, id_: int, name: str, hourly_wage: float,
                 hours_per_month: float) -> None:
        """Initialize this HourlyEmployee.
        """
        Employee.__init__(self, id_, name)
        self.hourly_wage = hourly_wage
        self.hours_per_month = hours_per_month

    def get_monthly_payment(self) -> float:
        """Return the amount that this Employee should be paid in one month.

        Round the amount to the nearest cent.
        """
        return self.hours_per_month * self.hourly_wage
```