"Client code for the various Employee classes.
Writing this first helps us envision the interface we want.
"

from hourly_employee import HourlyEmployee
from salaried_employee import SalariedEmployee

def main():
    """Use the various Employee classes."""
    fred = SalariedEmployee(14, 'Fred Flintstone', 454121883, 'weekly', 5200)
boss = SalariedEmployee(99, 'Mister Slate', 999999999, 'monthly', 12000)
    weekly_employees = [fred, barney]
    monthly_employees = [boss]

    # Week of Jan 7th.
    barney.log_hours(8)
    barney.log_hours(7)
    barney.log_hours(10)
    barney.log_hours(8)
    for e in weekly_employees:
        e.record_pay('Jan 7, 3000 BC')

    # Week of Jan 14th.
    barney.log_hours(42)
    for e in weekly_employees:
        e.record_pay('Jan 14, 3000 BC')

    # Week of Jan 21st.
    barney.log_hours(37)
    for e in weekly_employees:
        e.record_pay('Jan 21, 3000 BC')

    # Week of Jan 28th. End of month so pay the monthly employees too.
    barney.log_hours(40)
    for e in weekly_employees:
        e.record_pay('Jan 28, 3000 BC')
    for e in monthly_employees:
        e.record_pay('Jan 28, 3000 BC')

    # Sum up the company payroll for January.
    grand_total = 0
    for e in weekly_employees + monthly_employees:
        pay = e.total_pay()
        print('{} made ${}'.format(e.name, pay))
        grand_total += pay
    print('In total, employees made ${}'.format(grand_total))

if __name__ == '__main__':
    main()
class Employee:
    """An employee.

    === Public attributes ===
    @type eid: int
        This employee's ID number
    @type name: str
        This employee's name
    @type sin: int
        This employee's social insurance number (SIN)
    @type pay_period: str
        This employee's pay period

    === Private attributes ===
    @type _pay_history: dict[str, int]
        A record of this employee's past payments. Each key is a date and
        its value is the amount this employee was paid on that date.

    === Representation invariants ===
    - pay_period is either 'weekly' or 'monthly'

    This is an abstract class. Only child classes should be instantiated.
    ""
    def __init__(self, eid, name, sin, pay_period):
        """Initialize this employee.
        @type self: Employee
        @type eid: int
        @type name: str
        @type sin: int
        @type pay_period: str
        @rtype None
        Note: This constructor is meant for internal use only;
        Employee is an abstract class and should not be instantiated.
        """
        #type self: Employee
        #type eid: int
        #type name: str
        #type sin: int
        #type pay_period: str
        #rtype None

    def amount_of_pay(self):
        """Return the amount that this Employee should be paid in the next
        pay period.
        @type self: Employee
        @rtype: float
        """

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

    === Public attributes ===
    @type hourly_wage: float
        This employee's hourly rate of pay
    @type not_yet_paid: int
        The number of hours of work this Employee has accumulated
        since their last pay day, and has therefore not yet been
        paid for.

    def __init__(self, eid, name, sin, pay_period, hourly_wage):
        """Initialize this HourlyEmployee
        @type self: HourlyEmployee
        @type eid: int
        @type name: str
        @type sin: int
        @type pay_period: str
        @type hourly_wage: float
        @rtype: None
        >>> e = HourlyEmployee(23, 'Barney Rubble', 333333333, 'weekly', 1.25)
        >>> e.hourly_wage
        1.25
        >>> e.not_yet_paid
        0
        """
    def _init_(self, eid, name, sin, pay_period, hourly_wage):
        """Initialize this HourlyEmployee
        @type self: HourlyEmployee
        @type eid: int
        @type name: str
        @type sin: int
        @type pay_period: str
        @type hourly_wage: float
        @rtype: None
        >>> e = HourlyEmployee(23, 'Barney Rubble', 333333333, 'weekly', 1.25)
        >>> e.hourly_wage
        1.25
        >>> e.not_yet_paid
        0
        """
    def amount_of_pay(self):
        """Return the amount that this Employee should be paid in the next
        pay period.
        @type self: Employee
        @rtype: float
        """

def amount_of_pay(self):
    """Return the amount that this Employee should be paid in the next
    pay period.
    @type self: Employee
    @rtype: float
    """
    >>> e = HourlyEmployee(23, 'Barney Rubble', 333333333, 'weekly', 1.25)
    >>> e.log_hours(15)
    >>> e.amount_of_pay()
    18.75
    """