Computing 2 Homework Problems

Problem Set 6 (Polymorphism – Payroll System Modification)

Modify the payroll system discussed in class to include additional Employee subclasses. `PieceWorker` and `HourlyWorker`.

A `PieceWorker` represents an `Employee` whose pay is based on the number of pieces of merchandise produced. Class `PieceWorker` should contain private class member variables `wage` (to store the employee's wage per piece) and `pieces` (to store the number of pieces produced). In class `PieceWorker`, provide a concrete implementation of method `earnings` that calculates the employee's earnings by multiplying the number of pieces produced by the wage per piece.

A `HourlyWorker` represents an `Employee` whose pay is based on an hourly wage and the number of hours worked. Hourly workers receive overtime pay (1.5 times the hourly wage) for all hours worked in excess of 40 hours. Class `HourlyWorker` should contain private instance variables `wage` (to store the employee's wage per hour) and `hours` (to store the hours worked). In class `HourlyWorker`, provide a concrete implementation of method `earnings` that calculates the employee's earnings by multiplying the number of hours worked by the wage per hour. If the number of hours worked is over 40, be sure to pay the `HourlyWorker` for overtime hours.

Details about the `Employee` ineheritance hierarchy presented is shown in the UML diagram below.

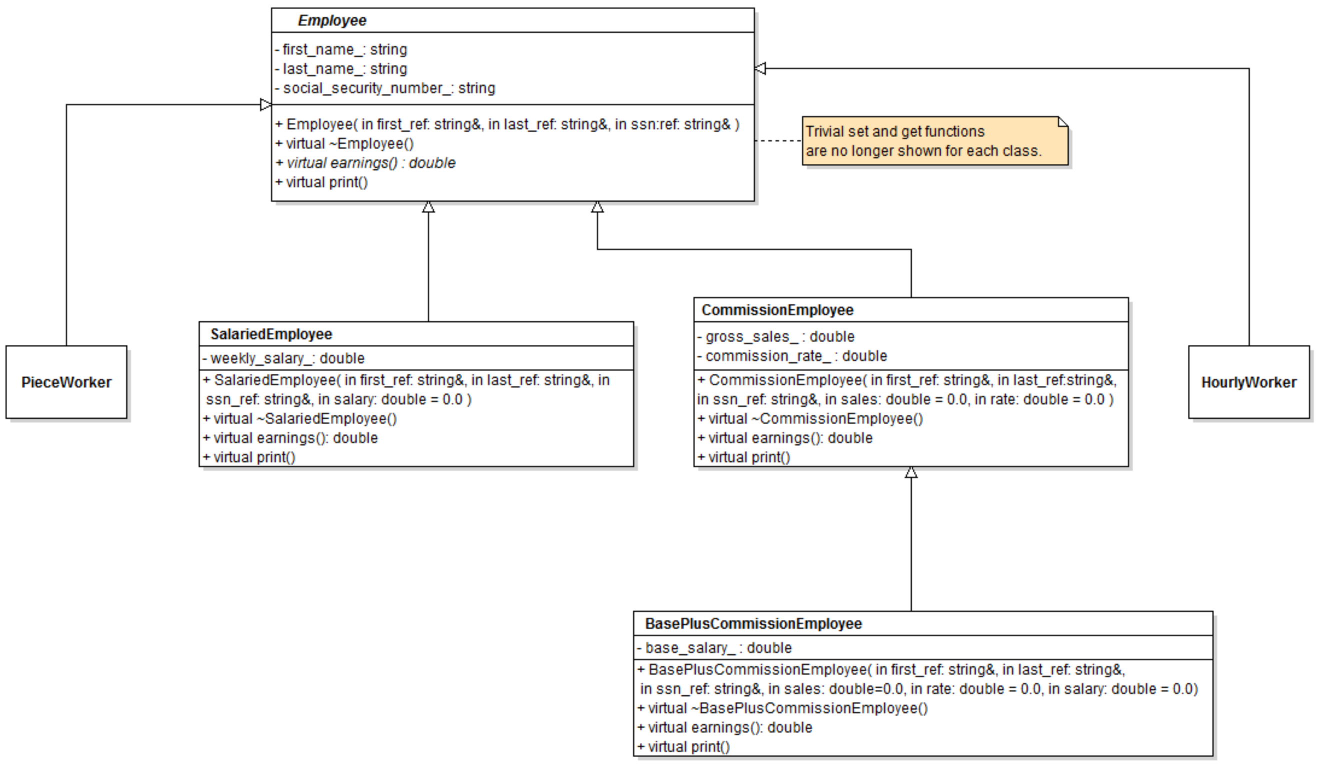

Employee
- first_name_: string
- last_name_: string
- social_security_number_: string
+ Employee(in first_ref: string&, in last_ref: string&, in ssn:ref: string&)
+ virtual ~Employee()
+ virtual earnings() : double
+ virtual print()
Trivial set and get functions are no longer shown for each class.
PieceWorker
SalariedEmployee
- weekly_salary_: double
+ SalariedEmployee(in first_ref: string&, in last_ref: string&, in ssn_ref: string&, in salary: double = 0.0)
+ virtual ~SalariedEmployee()
+ virtual earnings(): double
+ virtual print()
CommissionEmployee
- gross_sales_ : double
- commission_rate_ : double
+ CommissionEmployee(in first_ref: string&, in last_ref:string&, in ssn_ref: string&, in sales: double = 0.0, in rate: double = 0.0)
+ virtual ~CommissionEmployee()
+ virtual earnings(): double
+ virtual print()
HourlyWorker
BasePlusCommissionEmployee
- base_salary_ : double
+ BasePlusCommissionEmployee(in first_ref: string&, in last_ref: string&, in ssn_ref: string&, in sales: double=0.0, in rate: double = 0.0, in salary: double = 0.0)
+ virtual ~BasePlusCommissionEmployee()
+ virtual earnings(): double
+ virtual print()

Figure 1: In the UML diagram, you see the already introduced classes and their (private) data elements. The methods of `SalariedEmployee`, `CommissionEmployee`, and `BasePlusCommissionEmployee` are only partially shown due to space constraints. Use Violet 2.0.1 to extend the UML diagram based on the file PolyBasePlusCommission.violet2.0.class.violet.html

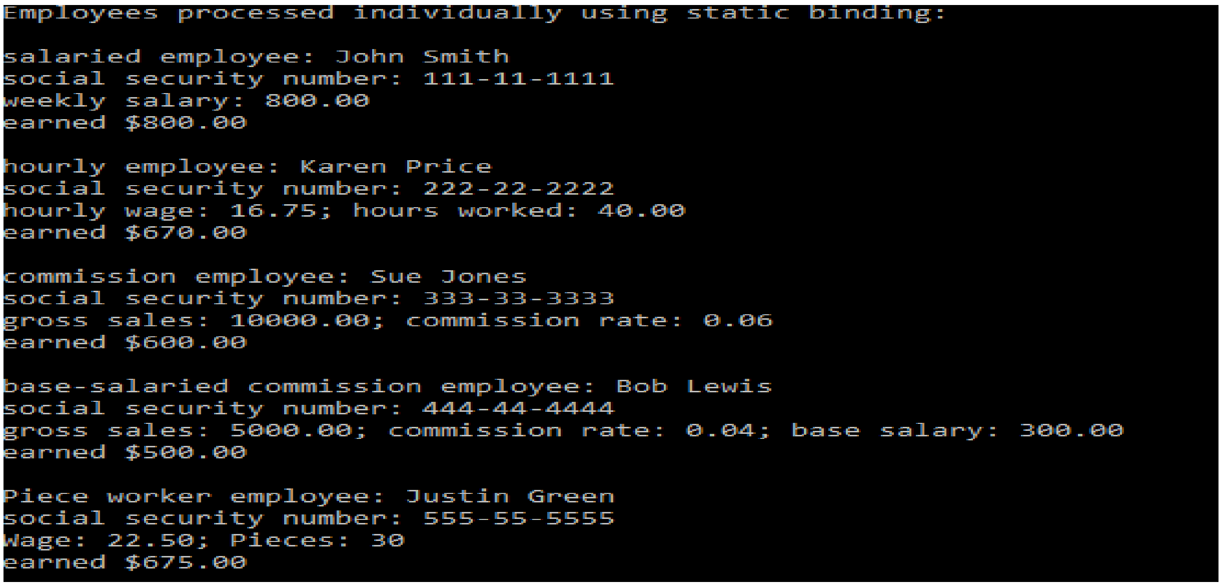
Tasks

1. Extend the UML Diagram for the new classes PieceWorker and HourlyWorker according to the description above.
2. Please extend the class hierarchy by implementing the new classes HourlyWorker and PieceWorker.
3. Please modify the driver program to accommodate the new classes HourlyWorker and PieceWorker. The goals are to investigate static binding and dynamic binding. For dynamic binding, add a pointer to an object of each new class into the vector of Employee pointers in main. For each Employee print its properties (private data members) and earnings using polymorphism.

The source code for the abstract class Employee, the derived classes `SalariedEmployee` and `CommissionEmployee` as well as the indirect derived class `BasePlusCommissionEmployee` is provided. You also find the old driver program.

Hints

Your output for the static binding part of the code may look like this, depending on how you constructed your objects.

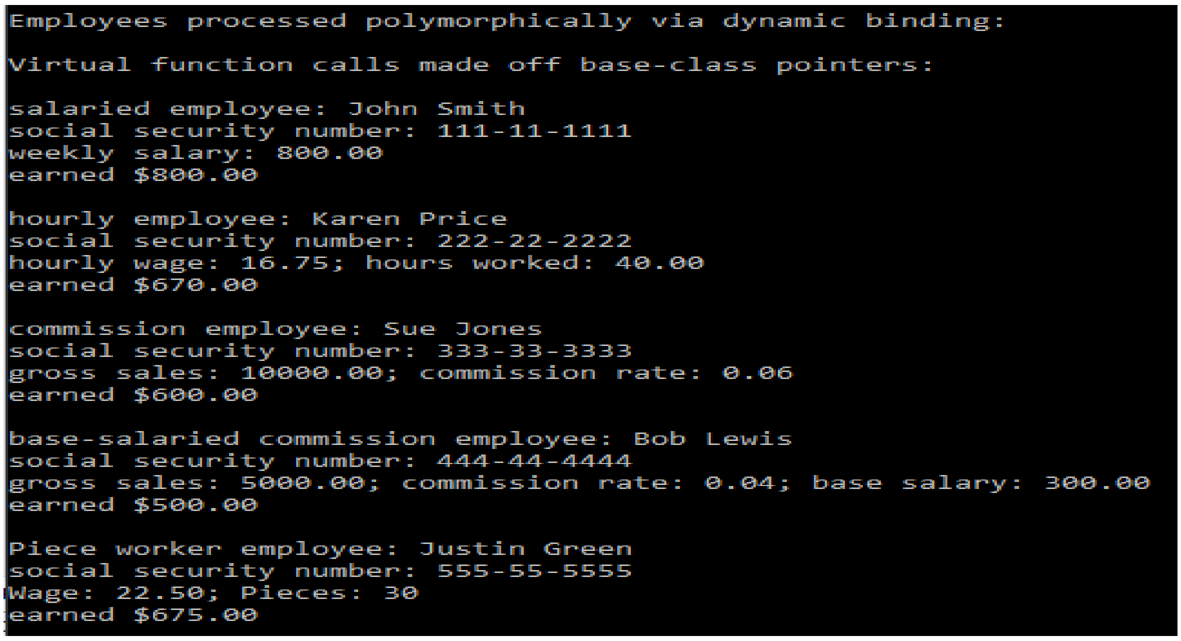


```
Employees processed individually using static binding:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00

Piece worker employee: Justin Green
social security number: 555-55-5555
Wage: 22.50; Pieces: 30
earned $675.00
```

Dynamic binding can be done either using base-class pointers or references. You can implement it using a regular for loop as shown below for a vector `employees` of base-class pointers (for details see Driver.cpp).

```
for ( size_t i = 0; i < employees.size(); ++i )
   virtualViaPointer( employees[ i ] );
```

Here is the result when using polymorphism involving base-class pointers.

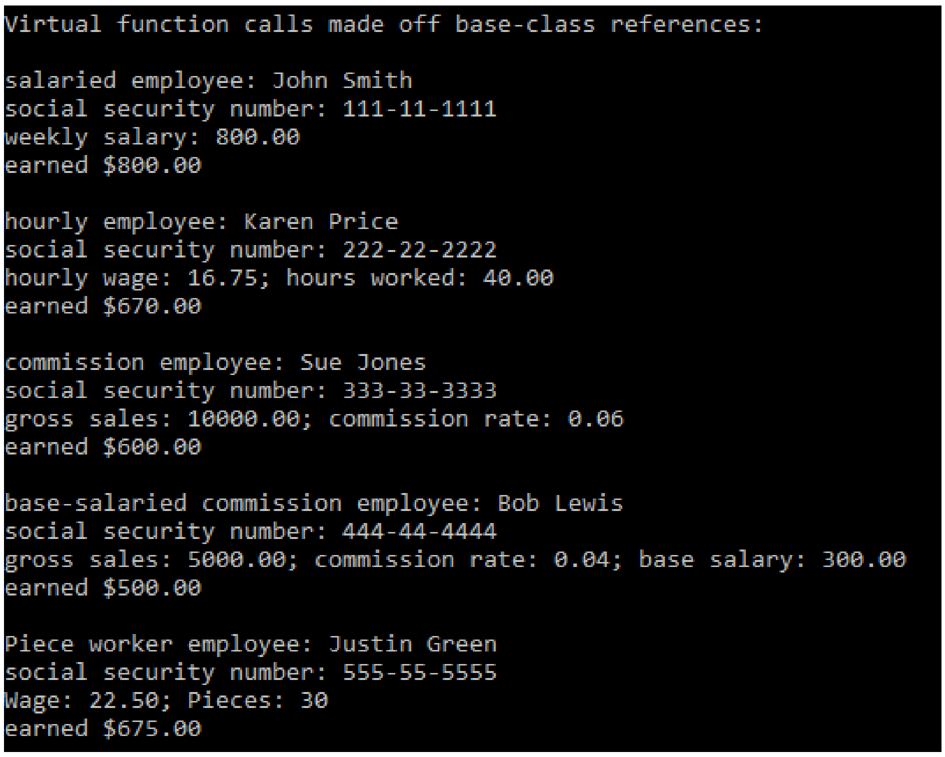


```
Employees processed polymorphically via dynamic binding:

Virtual function calls made off base-class pointers:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00

Piece worker employee: Justin Green
social security number: 555-55-5555
Wage: 22.50; Pieces: 30
earned $675.00
```

Dealing with references is a bit harder, because references have to be initialized when defined. In addition, references are no objects. Consequently, arrays of references do not exist. Still, one can work around this problem, e.g., as shown below.

```
for ( size_t i = 0; i < employees.size(); ++i )    {
    // virtualViaReference( *employees[ i ] ); // note dereferencing (either this
                                               // or what is shown next)
    Employee& employeeRef = *employees[ i ];   // a reference must refer to a valid
                                               // object when inititialized
    virtualViaReference( employeeRef );
}
```

The output is shown below.



```
Virtual function calls made off base-class references:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00

Piece worker employee: Justin Green
social security number: 555-55-5555
Wage: 22.50; Pieces: 30
earned $675.00
```