Java Program #4

Submit Assignment

Due Feb 11 by 10:59pm **Points** 16 **Submitting** a file upload
File Types doc, docx, and pdf

CS 2265 – Advanced Programming I

Troy eCampus – Prof. Reggie Haseltine

Program #4 - Java Application for Employee's Payroll System with User Interaction

16 points

Must be submitted to the Program #4 Assignment Link by the END OF WEEK #5

(Programs submitted after the deadline will receive a score of zero.)

- Develop a Java application to calculate the monthly paychecks for a number of different types of employees. The employee types are created in a subclass array based on parent base class Employee. Initial code is provided for each class and for a driver class.

You will need to compile each Java source file (provided below) separately, in its own file since they are public classes, to create a .class file for each, ideally keeping them all in the same directory.

You should first compile and execute the original classes and then make your customer modifications for this program.

You should compile Date.java first, then Employee.java, then CommissionEmployee.java, then BasePlusCommission.java, then HourlyEmployee.java, then SalariedEmployee.java, and finally PayrollSystemTest.java. And maintain this compilation order when you make your customized modifications to these classes later.

As part of the modifications you need to make to the supplied code, you are to prompt the user for all the data that is now provided in the code for the class PayrollSystemTest (i.e. instead of using these values from their declaration in the code, get data from prompting the user for it, one value at a time). All program output, including prompts and results and all program input must be done from the PayrollSystemTest class.

In addition you need to add a new private instance field "birthDate" to the Employee class. The “birthDate” field must be a Date data type, not a String! Add get and set methods to Employee class for this new birthDate field.

The birthDate field must be determined from separate integer values, supplied by the user, for the birth month, day, and year. As with all other inputs, these three birth date components must be prompted from and input to class PayrollSystemTest. Once input, these three parameter values must then be supplied to the appropriate modified employee subclass constructor, and then the subclass constructors supply these three values to their parent class, a modified Employee class constructor.

The single birthDate field of a Date data type must be declared and initialized in the Employee class.

In class PayrollSystemTest create an array of Employee variables to store references to the various employee objects. In a loop, calculate the monthly paycheck amount for each Employee (polymorphically), and add a $100.00 bonus to the person's monthly payroll amount if the current month (i.e. November) is the month in which the Employee's birthday occurs.

Your program should input data and create the five employees below on the version of the program you submit for grading. Your program must calculate their monthly salaries for November (assumed to be the "current" month). Assume that the monthly salary is simply four times the weekly salary.

1. one Salaried Employee with a weekly salary of $1000 and a birthday of January 1, 1960,
2. one Salaried Employee with a weekly salary of $1000 and a birthday in this month (November 1, 1961)
3. one Commission Employee Worker with a gross weekly sales of $12,000 and a 5% commission rate and a birthday of February 1, 1962,

4. one Base Plus Commission Employee with a gross weekly sales of $10,000, a 3% commission, and a $500 base weekly salary with a birthday of March 1, 1963, and
5. one Hourly Employee with a wage of $20/hour working for 40 hours and a birthday of April 1, 1964

You should make up your own employee names for the five required employees above. Make sure you create the employees in the order above.

In prompting the user for all input values for the five employees above, your program

essentially replaces the existing main() method of PayrollSystemTest.java. You may user whatever input class you want (e.g. JOptionPane, Scanner). Keep all other lines of the code in PayrollSystemTest as they are originally, including the code that gives a 10% increase to the Base Plus Commission employee.

You will probably find it helpful to prompt for the worker type first. You may

hardcode "November" as the special bonus month.

Since the outputs are to be for monthly paychecks, just multiply the weekly

salaries, weekly output quantities, and weekly hours by 4. Display your results

for each worker's salary after all data for that employee type has been entered

and the salary determined.

Your output should appear like the example below (using the current examples in the code for PayrollSystemTest (the "Employees processed individually" portion isn't shown, and monthly salaries are shown below as required by this program vs. weekly salaries that would be produced from the unmodified code). Note that the test data is shown below. Your five employees will create a different set of outputs. After you verify that the original program produces these results you can remove the output statements for this original data and just keep what you need for your five employees.

Employees process polymorphically:

salaried employee: John Smith

social security number: 111-11-1111

date of birth: 01/02/1960(current code won't do this but yours must)

weekly salary: $800.00

earned $3200.00

hourly employee: Karen Price

social security number: 222-22-2222

date of birth: 01/02/1960(current code won't do this but yours must)

hourly wage: $16.75; hours worked: 40.00

earned $2680.00

commission employee: Sue Jones

social security number: 333-33-3333

date of birth: 01/02/1960(current code won't do this but yours must)

gross sales: $10,000.00; commission rate: 0.06

earned $2400.00

base-salaried commission employee: Bob Lewis

social security number: 444-44-4444

date of birth: 01/02/1960(current code won't do this but yours must)

gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00

new base salary with 10% increase is: $330.00

earned: $2120.00

Employee 0 is a SalariedEmployee

Employee 1 is a HourlyEmployee

Employee 2 is a CommissionEmployee

Employee 3 is a BasePlusCommissionEmployee

Be sure that you include screen snapshots of all user inputs, and all output. Failure to do so will result in lost points.

(This program was taken from Exercise 10.9 on page 508 of Deitel & Deitel's "Java How to Program (Sixth Edition)" (2005 by Pearson Publishing Co.))

- You may use the Windows Command Prompt command line interface or any Java IDE you choose to compile and execute your program.

- You are to submit the following deliverables to the Assignment Link in a single Microsoft Word file:
  1. A screen snapshot of your Java source code (just the beginning is OK) as it appears in your IDE (e.g. Net Beans, Eclipse, etc.) or editor (e.g. a Windows Command Prompt DOS "more" of the .java file's first screen).

2. A listing of your entire Java source code, for all classes, showing your final version of the code, in the same Microsoft Word file as item a), and following item a). You can simply copy and paste the text from your IDE into Word. Be sure to maintain proper code alignment by using Courier font for this item. Source files for all classes must be submitted for full credit.

3. A screen snapshot of your program's execution inputs and output in the same Microsoft Word file, and following item b). **Be sure that you include screen snapshots of all user inputs, and all output. Failure to do so will result in lost points.**

- Your instructor may compile and run your program to verify that it compiles and executes properly.

- You will be evaluated on (in order of importance):

1. Inclusion of all deliverables in the correct order, including all user's inputs and all output
2. Correct execution of your program.
3. Adequate commenting of your code.
4. Good programming style (as specified in the textbook's examples).
5. Neatness in packaging and labeling of your deliverables.

Deficiencies in any of the above areas are subject to up to a 2 (two) point deduction, per area, based on the severity of the deficiency.

- Programs received after the due date/time will receive a score of zero since there are already 15 extra points built into the grading for this class.

Here's the initial code you need to modify!

```
// PayrollSystemTest.java
// Employee hierarchy test program.

public class PayrollSystemTest
{
   public static void main( String args[] )
   {
      // create subclass objects
      SalariedEmployee salariedEmployee =
         new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
      HourlyEmployee hourlyEmployee =
         new HourlyEmployee( "Karen", "Price", "222-22-2222", 16.75, 40 );
```

```
CommissionEmployee commissionEmployee =
  new CommissionEmployee(
  "Sue", "Jones", "333-33-3333", 10000, .06 );
BasePlusCommissionEmployee basePlusCommissionEmployee =
  new BasePlusCommissionEmployee(
  "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

System.out.println( "Employees processed individually:\n" );

System.out.printf( "%s\n%s: $%,.2f\n\n",
  salariedEmployee, "earned", salariedEmployee.earnings() );
System.out.printf( "%s\n%s: $%,.2f\n\n",
  hourlyEmployee, "earned", hourlyEmployee.earnings() );
System.out.printf( "%s\n%s: $%,.2f\n\n",
  commissionEmployee, "earned", commissionEmployee.earnings() );
System.out.printf( "%s\n%s: $%,.2f\n\n",
  basePlusCommissionEmployee,
  "earned", basePlusCommissionEmployee.earnings() );

// create four-element Employee array
Employee employees[] = new Employee[ 4 ];

// initialize array with Employees
employees[ 0 ] = salariedEmployee;
employees[ 1 ] = hourlyEmployee;
```

```
employees[ 2 ] = commissionEmployee;
employees[ 3 ] = basePlusCommissionEmployee;

System.out.println( "Employees processed polymorphically:\n" );

// generically process each element in array employees
for ( Employee currentEmployee : employees )
{
   System.out.println( currentEmployee ); // invokes toString

   // determine whether element is a BasePlusCommissionEmployee
   if ( currentEmployee instanceof BasePlusCommissionEmployee )
   {
      // downcast Employee reference to
      // BasePlusCommissionEmployee reference
      BasePlusCommissionEmployee employee =
         ( BasePlusCommissionEmployee ) currentEmployee;

      double oldBaseSalary = employee.getBaseSalary();
      employee.setBaseSalary( 1.10 * oldBaseSalary );
      System.out.printf(
         "new base salary with 10%% increase is: $%,.2f\n",
         employee.getBaseSalary() );
   } // end if
```

```
         System.out.printf(
            "earned $%,.2f\n\n", currentEmployee.earnings() );
      } // end for

      // get type name of each object in employees array
      for ( int j = 0; j < employees.length; j++ )
         System.out.printf( "Employee %d is a %s\n", j,
            employees[ j ].getClass().getName() );
   } // end main
} // end class PayrollSystemTest
```

```
// Employee.java
// Employee abstract superclass.

public abstract class Employee
{
   private String firstName;
   private String lastName;
   private String socialSecurityNumber;

   // three-argument constructor
   public Employee( String first, String last, String ssn )
   {
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
   } // end three-argument Employee constructor

   // set first name
   public void setFirstName( String first )
   {
      firstName = first;
   } // end method setFirstName
```

```
  // return first name
  public String getFirstName()
  {
     return firstName;
  } // end method getFirstName

  // set last name
  public void setLastName( String last )
  {
     lastName = last;
  } // end method setLastName

  // return last name
  public String getLastName()
  {
     return lastName;
  } // end method getLastName

  // set social security number
  public void setSocialSecurityNumber( String ssn )
  {
     socialSecurityNumber = ssn; // should validate
  } // end method setSocialSecurityNumber

  // return social security number
```

```
   public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

   // return String representation of Employee object
   public String toString()
   {
      return String.format( "%s %s\nsocial security number: %s",
         getFirstName(), getLastName(), getSocialSecurityNumber() );
   } // end method toString

   // abstract method overridden by subclasses
   public abstract double earnings(); // no implementation here
} // end abstract class Employee
```

```
 * no warranty of any kind, expressed or implied, with regard to these    *
 * programs or to the documentation contained in these books. The authors *
 * and publisher shall not be liable in any event for incidental or       *
 * consequential damages in connection with, or arising out of, the       *
 * furnishing, performance, or use of these programs.                     *
 *************************************************************************/

// SalariedEmployee.java
// SalariedEmployee class extends Employee.

public class SalariedEmployee extends Employee
{
   private double weeklySalary;

   // four-argument constructor
   public SalariedEmployee( String first, String last, String ssn,
      double salary )
   {
      super( first, last, ssn ); // pass to Employee constructor
      setWeeklySalary( salary ); // validate and store salary
   } // end four-argument SalariedEmployee constructor

   // set salary
   public void setWeeklySalary( double salary )
   {
```

```
    weeklySalary = salary < 0.0 ? 0.0 : salary;
  } // end method setWeeklySalary

  // return salary
  public double getWeeklySalary()
  {
    return weeklySalary;
  } // end method getWeeklySalary

  // calculate earnings; override abstract method earnings in Employee
  public double earnings()
  {
    return getWeeklySalary();
  } // end method earnings

  // return String representation of SalariedEmployee object
  public String toString()
  {
    return String.format( "salaried employee: %s\n%s: $%,.2f",
      super.toString(), "weekly salary", getWeeklySalary() );
  } // end method toString
} // end class SalariedEmployee


/**************************************************************************
```

```
// HourlyEmployee.java
// HourlyEmployee class extends Employee.

public class HourlyEmployee extends Employee
{
   private double wage; // wage per hour
   private double hours; // hours worked for week

   // five-argument constructor
   public HourlyEmployee( String first, String last, String ssn,
      double hourlyWage, double hoursWorked )
```

```
{
   super( first, last, ssn );
   setWage( hourlyWage ); // validate and store hourly wage
   setHours( hoursWorked ); // validate and store hours worked
} // end five-argument HourlyEmployee constructor

// set wage
public void setWage( double hourlyWage )
{
   wage = ( hourlyWage < 0.0 ) ? 0.0 : hourlyWage;
} // end method setWage

// return wage
public double getWage()
{
   return wage;
} // end method getWage

// set hours worked
public void setHours( double hoursWorked )
{
   hours = ( ( hoursWorked >= 0.0 ) && ( hoursWorked <= 168.0 ) ) ?
      hoursWorked : 0.0;
} // end method setHours
```

```
   // return hours worked
   public double getHours()
   {
      return hours;
   } // end method getHours

   // calculate earnings; override abstract method earnings in Employee
   public double earnings()
   {
      if ( getHours() <= 40 ) // no overtime
         return getWage() * getHours();
      else
         return 40 * getWage() + ( getHours() - 40 ) * getWage() * 1.5;
   } // end method earnings

   // return String representation of HourlyEmployee object
   public String toString()
   {
      return String.format( "hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
         super.toString(), "hourly wage", getWage(),
         "hours worked", getHours() );
   } // end method toString
} // end class HourlyEmployee
```

```
/**************************************************************************
 * (C) Copyright 1992-2005 by Deitel & Associates, Inc. and               *
 * Pearson Education, Inc. All Rights Reserved.                           *
 *                                                                        *
 * DISCLAIMER: The authors and publisher of this book have used their     *
 * best efforts in preparing the book. These efforts include the          *
 * development, research, and testing of the theories and programs        *
 * to determine their effectiveness. The authors and publisher make       *
 * no warranty of any kind, expressed or implied, with regard to these    *
 * programs or to the documentation contained in these books. The authors *
 * and publisher shall not be liable in any event for incidental or       *
 * consequential damages in connection with, or arising out of, the       *
 * furnishing, performance, or use of these programs.                     *
 *************************************************************************/

// CommissionEmployee.java
// CommissionEmployee class extends Employee.

public class CommissionEmployee extends Employee
{
   private double grossSales; // gross weekly sales
   private double commissionRate; // commission percentage

   // five-argument constructor
   public CommissionEmployee( String first, String last, String ssn,
```

```
    double sales, double rate )
  {
    super( first, last, ssn );
    setGrossSales( sales );
    setCommissionRate( rate );
  } // end five-argument CommissionEmployee constructor

  // set commission rate
  public void setCommissionRate( double rate )
  {
    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
  } // end method setCommissionRate

  // return commission rate
  public double getCommissionRate()
  {
    return commissionRate;
  } // end method getCommissionRate

  // set gross sales amount
  public void setGrossSales( double sales )
  {
    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
  } // end method setGrossSales
```

```
   // return gross sales amount
   public double getGrossSales()
   {
      return grossSales;
   } // end method getGrossSales

   // calculate earnings; override abstract method earnings in Employee
   public double earnings()
   {
      return getCommissionRate() * getGrossSales();
   } // end method earnings

   // return String representation of CommissionEmployee object
   public String toString()
   {
      return String.format( "%s: %s\n%s: $%,.2f; %s: %.2f",
         "commission employee", super.toString(),
         "gross sales", getGrossSales(),
         "commission rate", getCommissionRate() );
   } // end method toString
} // end class CommissionEmployee



/**************************************************************************
 * (C) Copyright 1992-2005 by Deitel & Associates, Inc. and               *
```

```
// BasePlusCommissionEmployee.java
// BasePlusCommissionEmployee class extends CommissionEmployee.

public class BasePlusCommissionEmployee extends CommissionEmployee
{
   private double baseSalary; // base salary per week

   // six-argument constructor
   public BasePlusCommissionEmployee( String first, String last,
      String ssn, double sales, double rate, double salary )
   {
      super( first, last, ssn, sales, rate );
```

```
    setBaseSalary( salary ); // validate and store base salary
  } // end six-argument BasePlusCommissionEmployee constructor

  // set base salary
  public void setBaseSalary( double salary )
  {
    baseSalary = ( salary < 0.0 ) ? 0.0 : salary; // non-negative
  } // end method setBaseSalary

  // return base salary
  public double getBaseSalary()
  {
    return baseSalary;
  } // end method getBaseSalary

  // calculate earnings; override method earnings in CommissionEmployee
  public double earnings()
  {
    return getBaseSalary() + super.earnings();
  } // end method earnings

  // return String representation of BasePlusCommissionEmployee object
  public String toString()
  {
    return String.format( "%s %s; %s: $%,.2f",
```

```
        "base-salaried", super.toString(),
        "base salary", getBaseSalary() );
    } // end method toString
} // end class BasePlusCommissionEmployee
```



```
// Date.java
// Date class declaration.

public class Date
```

```
{
   private int month; // 1-12
   private int day;   // 1-31 based on month
   private int year;  // any year

   // constructor: call checkMonth to confirm proper value for month;
   // call checkDay to confirm proper value for day
   public Date( int theMonth, int theDay, int theYear )
   {
      month = checkMonth( theMonth ); // validate month
      year = theYear; // could validate year
      day = checkDay( theDay ); // validate day

      System.out.printf(
         "Date object constructor for date %s\n", this );
   } // end Date constructor

   // utility method to confirm proper month value
   private int checkMonth( int testMonth )
   {
      if ( testMonth > 0 && testMonth <= 12 ) // validate month
         return testMonth;
      else // month is invalid
      {
         System.out.printf(
```

```
        "Invalid month (%d) set to 1.", testMonth );
      return 1; // maintain object in consistent state
    } // end else
  } // end method checkMonth

  // utility method to confirm proper day value based on month and year
  private int checkDay( int testDay )
  {
    int daysPerMonth[] =
      { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    // check if day in range for month
    if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
      return testDay;

    // check for leap year
    if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
        ( year % 4 == 0 && year % 100 != 0 ) ) )
      return testDay;

    System.out.printf( "Invalid day (%d) set to 1.", testDay );
    return 1;  // maintain object in consistent state
  } // end method checkDay

  // return a String of the form month/day/year
```

```
   public String toString()
   {
      return String.format( "%d/%d/%d", month, day, year );
   } // end method toString
} // end class Date
```