UNBC Laboratory Assignment
Computer Science 101—Winter 2016

Inheritance and Method Overriding

Purpose:

To gain familiarity with the use of inheritance and method overriding.

Due Date:

This assignment is due Wednesday, 2016-02-03 *at the beginning of class*.

Reading Assignment:

⇒ Read, answer, and understand the following questions:

- Can constructors be polymorphic?
- When a subclass method overrides that of a superclass can the signatures be different?
- When a subclass method overrides that of a superclass can the return types be different?

Hand in answers to these questions in a file `answers.txt` in your your lab assignment jar file. Discuss them with your lab instructor if you are not sure how they work.

Problem Description:

To implement a payroll processing system that processes records for three kinds of employees: full time (abbreviated FT), part time (abbreviated PT), and contract (abbreviated CO) — *using inheritance and polymorphism*. The output format is given first; followed by a description of the kinds of employees and the input format; and then by details regarding the internal design of the program.

Output Format

The output should look like the following:

```
Employee  Employee  Hours/    Rate/   Commision   Other  Deductions  Take
Class     Number    Week      Hour                                   Home
-------------------------------------------------------------------------
FT        00613      40.0     65.00        0.00     0.00      700.00  1900.00
FT        00614      40.0     35.00        0.00     0.00      150.00  1250.00
PT        09312      20.0     15.00        0.00     0.00       10.00   290.00
CO        99001       0.0      0.00        0.00  4512.35        0.00  4512.35
```

There should be be

- no more 50 lines per page of output, and
- page running totals for the `Commission`, `Other`, `Deductions` and `Take Home` columns, and
- grand totals for the above columns.

In each line the take-home pay should be equal to the hours-per-week multiplied by rate-per-hour plus any commission plus other minus the deductions.

Kinds of Employees and Input Format

Each employee is represented by a one-line input record that starts with an employee kind code (two capital letters), a space, then five-digit employee number. The format for the rest of the input line depends on the kind of employee.

For each employee kind there are potentially wages paid at a fixed rate per hour (there is no overtime pay in this problem), the number of hours per week that the employee works, commissions that the employee earns, and other earnings. There are also potentially deductions for tax, pensions, EI, and the like.

Details for the kinds of employees follow:

FullTime A typical input record for a full time employee looks like:

```
FT 00613 65.00
```

The first two letter are the employee class designator; the next number is the employee number (always five digits, but the leading ones may be zeroes); and the final number is the hourly pay rate.

Each full time employee works 40 hours per week. Employees in this class never earn commissions or other payments. Deductions are calculated as \$10.00 per week for each dollar per hour earned between \$20.00/hr and \$40.00/hr; and \$20.00 per week for each dollar per hour over forty. Thus someone earning \$65.00/hr pays $20 \times \$10.00 + 25 \times \$20.00 = \$700.00$ per week in deductions. Symbolically:

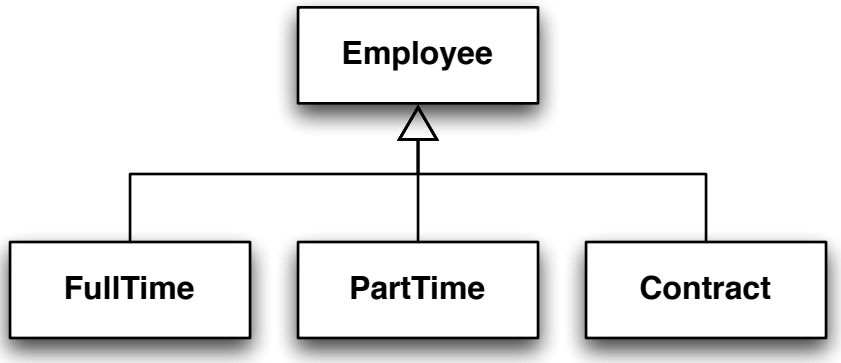

Figure 1: Inheritance Diagram

	WAGES				DEDUCTIONS
	$ 20.00/hr	×	$ 0.00	=	$ 0.00
	$ 20.00/hr	×	$ 10.00	=	$ 200.00
	$ 25.00/hr	×	$ 20.00	=	$ 500.00
TOTAL	$ 65.00/hr		gives		$ 700.00

PartTime A typical input record for a part time employee looks like:

```
PT 00613 20 15.00
```

where the `20` indicates the number of hours per week worked (always integral), and `15.00` is the hourly rate of pay.

Each part time employee works between 10 and 30 hours per week. Employees in this class never earn commissions or other payments. Deductions are calculated as 10% of total earnings between $200 and $500 a week, and 30% on everything in excess of $500 per week. For the example above

	WAGES				DEDUCTIONS
	$ 200.00	×	% 00	=	$ 00.00
	$ 100.00	×	% 10	=	$ 10.00
	$ 000.00	×	% 30	=	$ 00.00
TOTAL	$ 300.00		gives		$ 10.00

Contract A typical input record for a contract employee looks like:

```
CO 99001 4512.35
```

where `4512.35` is the amount of other income earned.

Contract employees are never paid an hourly rate and never earn commissions. Furthermore, no deductions are made from their pay.

Use of Inheritance and Polymorphism

You *must* use inheritance and polymorphism to to receive credit for this assignment. You *must* have an `Employee` base class, and derive a class for each particular kind of `Employee`. The UML diagram showing the inheritance relation between the main classes is shown in Figure 1.

`Employee` Member Variables (fields)

You *must not* use package, protected or public fields in the `Employee` class.

`Employee` Methods

On the other hand, the `Employee` class *must* have public attribute methods for each column of the output. A derived class that does something non-trivial to compute a particular column will override the base class implementation, so the `Employee` class implementations should provide default methods that work for subclasses that don't need them. For instance the default implementation of an `Employee.getCommission` method might be simply to return `0.00`.

The `Employee` class should have a `print` or `toString` method that uses the appropriate attribute methods of the `Employee` class. It is unlikely that this method will need to be overridden in subclasses.

Subclass Methods

Each subclass should have a

```
public void readDetailsFrom(Scanner in) ;
```

method that reads appropriate information from `in` and mutates the calling object appropriately. Note that you cannot know which subclass to create until *after* you have read the employee kind code, so the method should process the remainder of an input line.

Each subclass should also have an overriding method for each attribute whose calculation is non-trivial. For instance, both the `FullTime` and `PartTime` classes should overrride a method called something like `getHours()`.

Subclass Member Variables (Fields)

The fields of each subclass should correspond relatively directly to the information contained on input records for that subclass.

```
4   public static void processReport(Reader in, Writer out)
5       {
6       Scanner employeeScanner = new Scanner(in) ;
7       Report report = new Report(out);
8       report.setLinesPerPage(50) ;
9       report.print_banner(out) ;
10
11      String code = "--" ;
12      Employee current = new UnclassifiedEmployee() ;
13      bool employeeFound = false ;
14      bool done          = false ;
15
16      while (!done)
17          {
18          if (employeeFound)
19              {
20              report.print(current) ;//
21              employeeFound = false ;
22              }
23
24          done = !employeeScanner.hasNext() ;
25          if (done) continue ;
26
27          switch(code = employeeScanner.next())
28              {
29              case "FT":  current = new Fulltime() ; break ; //
30              case "PT":  current = new Parttime() ; break ;
31              case "CO":  current = new Contract() ; break ;
32              default:
33                  current = new UnclassifiedEmployee(code) ;
34                  break; ;//
35              }
36          current.readDetailsFrom(employeeScanner) ;
37          employeeFound = true ;
38          }
39      report.finish() ;
40      return ;
41      }
```

Figure 2: `processFiles` version 1

Example Code

The code shown in Figure 2 on page 5 shows one way to write a `static` report producing method. Note that this code relies on a `Report`-class and an `UnclassifiedEmployee`-class.

Figure 2 shows how to put inheritance to good effect. Aside from the `switch`-logic and constructor calls on lines 29–34, there is nothing specific to particular subclasses in the `processReport` method, which makes it easy to modify the program to add further subclasses of employees.

⇒ Analyze the code in Figure 2 carefully and determine the attributes, behaviours, and collaborations of the `Report` class. Briefly describe these in your `answers.txt` file.

Your goal is to use inheritance and polymorphism as much as possible, so that you can later extend your program to handle additional classes of employees without having to modify *any* of the existing code.

It is possible to write the program in such a way that additional employee classes can be added to the program simply by adding `*.class` files for the new kinds of employees. This requires a certain amount of technical trickery, which may be explained in a later laboratory assignment.

Assignment

⇒ Write code to produce reports as specified above. Your program should be able to accept a command line argument that specifies the input file to read.