UNBC Laboratory Assignment
Computer Science 101—Winter 2008

Virtual Functions and Inheritance

Purpose:

To gain familiarity with the use of virtual functions and inheritance.

Due Date:

This assignment is due Wednesday 2008-01-30.

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 attached to your lab assignment. Discuss them with your lab instructor if you are not sure how they work.

Problem Description:

Implement a payroll processing system that processes records for three kinds of `Employee`'s: `FullTime`, `PartTime`, and `Contract`.

The details of input records for each kind of employee are described in the following section. 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 there should be page running totals and grand total for the `Commission`, `Other`, `Deductions` and `Take Home` columns.

Details for the kinds of employees follow:

FullTime 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.

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.

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

```
PT 00613 20 15.00
```

Contract Contract employees are never paid an hourly rate and never earn commissions. Furthermore, no deductions are made from their pay. A typical input record for a contract employee looks like:

```
CO 99001 4512.35
```

Assignment:

⇒ In your main class write methods with signatures:

```
import java.io.*
```

- `public static void processPayroll(Reader in, Writer out)`
- `public static void processPayroll(InputStream in, OutputStream out)`

One or the other of these functions can then be called from the `public static void main` method.

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`.

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, to be explained in a a later assignment.

To begin with, write a program to solve the problem described above that makes heavy use of inheritance and polymorphism.

Here are some guidelines:

```
// various import's

public class Bob {
  public static void processFiles(Reader in, Writer out)
      {
      Scanner sin = new Scanner(in) ;
      Report report = new Report(out);
      report.setLinesPerPage(50) ;
      report.print_banner(out) ;

      EmployeeCode code = new EmployeeCode("--") ;
      Employee current = new UnclassifiedEmployee() ;
      bool employeeFound = false ;

      while (! (sin.ioException() instanceof Object))
          {
          if (employeeFound)
              {
              report.print(current) ;//
              employeeFound = false ;
              }
          String temp ;
          if (!((temp = sin.next()) instanceof String))
              break ;
          // following assumes that EmployeeCode's can == with char*'s!
          code = new EmployeeCode(temp) ;
          if (code.equals("FT"))          current = new Fulltime() ;//
          else if (code.equals("PT"))     current = new Parttime() ;
          else if (code.equals("CO"))     current = new Contract() ;
          else                            current = new UnclassifiedEmployee() ;//
          current.readFrom(sin) ;//<--
          employeeFound = true ;
          }
      return ;
      }
}
```

Figure 1: `processFile` version 1

- The `Employee` super class should have very few member variables, *all* of which should be private. *Do not use* protected member variables. (If you really think that you need them, use protected "get" and "set" functions instead.)
- The `Employee` class should have many non-static methods that subclasses can override. In particular, the `Employee` class should have a attribute ("get") method for each of the colums of the output report. A derived class that does something non-trivial to compute that particular column will override the base class implementation, so the `Employee` class implementations should provide default methods that work for sub-classes that don't need them. For instance the default implementation of `getCommission` might be simply to return `0.00`. This works correctly for employees that aren't paid commissions.
- Each class should have a

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

  method that reads appropriate information from `in` and sets the calling object appropriately. The `EmployeeCode` needs to be read before such a function is called, so the function should process the remainder of an input line. Line 31 shows how this can be used to good effect.
- The `Employee` class should have an overloaded `print` method that calls the appropriate virtual functions of the `Employee` class.
- Employee codes should be stored in a variable of an `EmployeeCode` class. It should have (a) overrides and overloads for the `.equals(_)` method for arguments of type `Object` (to override `Object`'s version), for `EmployeeCodes`, and for `Strings`. and (b) an `int compareTo(EmployeeCode c)`-method for comparing `EmployeeCodes` alphabetically. The code shown in this lab assumes that the `EmployeeCode` class has all of the above.

Figure 1 shows how to put these virtual functions to good effect. Note that the `processFile` function shown in Figure 1 is not complete! (It lacks line counting and page totals and the like.) Also note that with the appropriately defined supporting classes, we are quite close to having a program that needs no modification in order to add a class. As it stands, we need to modify Lines 27–30 each time we add more classes.

Fixing this program so that no modification is necessary may be a future laboratory assignment.