CS 116
Week 2: Outline

Reading:

1. Dale, Chapter 7
2. Dale, Lab 2

Objectives:

1. Introduce Inheritance
2. Introduce Polymorphism
3. Introduce Scope

Concepts:

1. Discuss inheritance and its relation to object-oriented design
2. Discuss scope of access
3. Familiarize students with class hierarchies

CS 116
Week 2: Lecture Outline

1. Discuss inheritance and its relation to object-oriented design
    - Introduce basics of inheritance
    - Show examples
    - Inheritance in the OOD process

2. Familiarize students with class hierarchies
    - Introduce basics of class hierarchies

3. Discuss scope of access
    - Discuss scope's purpose in OOD
    - Scope rules
    - Class scope
    - Block scope

4. Introduce polymorphism
    - Discuss polymorphism's basics
    - Polymorphism examples

5. Lab 2 overview

CS 116
Week 2: Lab Outline

Objective:

1. Review coding, compilation and execution of simple programs
2. Be able to create a class
3. Be able to apply and use inheritance.

Student Activities:

1. Create a Program that creates a payroll system using Polymorphism

Let us use abstract classes, abstract methods and polymorphism to perform payroll calculations based on the type of employee. We use an abstract super class Employee. The subclass of Employee are Boss - paid a fixed weekly salary regardless of the number of hours worked, Commission Worker - paid a flat base salary plus a percentage of sales, PieceWorker - paid by the number of items produced and Hourly Worker - paid by the hour and receives overtime pay. Each subclass of Employee has been declared final, because we do not intend to inherit from them again.

An earnings method call certainly applies to all employees. But the way each person's earnings are calculated depends on the class of the employee, and these classes are all derived from the super class Employee. So earnings is declared abstract in super class Employee and appropriate implementations of earnings are provided for each of the subclass. Then, to calculate any employee's object and invokes the earnings method. In a real payroll system, the various Employee objects might be referenced by individual elements in an array of Employee references. The program would simply walk through the array one element at a time, using the Employee references to invoke the earnings method of each object. A Test.java is given and the final result . Good luck!

TEST.JAVA

```
// Test.java
// Driver for Employee hierarchy
// Java core packages
import java.text.DecimalFormat;
// Java extension packages
import javax.swing.JOptionPane;

public class Test {
   // test Employee hierarchy
   public static void main( String args[] )
   {
      Employee employee;  // superclass reference
      String output = "";

      Boss boss = new Boss( "John", "Smith", 800.0 );

      CommissionWorker commissionWorker =
```

```
      new CommissionWorker(
         "Sue", "Jones", 400.0, 3.0, 150 );

   PieceWorker pieceWorker =
      new PieceWorker( "Bob", "Lewis", 2.5, 200 );

   HourlyWorker hourlyWorker =
      new HourlyWorker( "Karen", "Price", 13.75, 40 );

   DecimalFormat precision2 = new DecimalFormat( "0.00" );

   // Employee reference to a Boss
   employee = boss;

   output += employee.toString() + " earned $" +
      precision2.format( employee.earnings() ) + "\n" +
      boss.toString() + " earned $" +
      precision2.format( boss.earnings() ) + "\n";

   // Employee reference to a CommissionWorker
   employee = commissionWorker;

   output += employee.toString() + " earned $" +
      precision2.format( employee.earnings() ) + "\n" +
      commissionWorker.toString() + " earned $" +
      precision2.format(
         commissionWorker.earnings() ) + "\n";

   // Employee reference to a PieceWorker
   employee = pieceWorker;

   output += employee.toString() + " earned $" +
      precision2.format( employee.earnings() ) + "\n" +
      pieceWorker.toString() + " earned $" +
      precision2.format( pieceWorker.earnings() ) + "\n";

   // Employee reference to an HourlyWorker
   employee = hourlyWorker;

   output += employee.toString() + " earned $" +
      precision2.format( employee.earnings() ) + "\n" +
      hourlyWorker.toString() + " earned $" +
      precision2.format( hourlyWorker.earnings() ) + "\n";

   JOptionPane.showMessageDialog( null, output,
      "Demonstrating Polymorphism",
      JOptionPane.INFORMATION_MESSAGE );
```

```
      System.exit( 0 );
   }

  }  // end class Test
```

You are also given the following class:

```
// Employee.java
// Abstract base class Employee.

public abstract class Employee {
   private String firstName;
   private String lastName;
   // constructor
   public Employee( String first, String last )
{
      firstName = first;
      lastName = last;
   }
   // get first name
   public String getFirstName()
{
      return firstName;
   }
   // get last name
   public String getLastName()
   {
      return lastName;
   }
   public String toString()
   {
      return firstName + ' ' + lastName;
   }
//Abstract method that must be implemented for each derived
class of Employee from which
//objects are instantiated.
   public abstract double earnings();
}  // end class Employee
```

a) **Create a Boss.java class.** Class Boss is derived from Employee. The public methods include a constructor that takes a first name, a last name and a weekly salary as arguments and passes the first name and last name to the Employee constructor to initialize the firstName and lastName members of the super class part of the subclass object. Other public methods include a setWeeklySalary method to assign a new value to private instance variable weeklySalary; an earnings method defining how to calculate a Boss's earnings; and a toString

method that forms a String containing the type of the employee (I.e.,"Boss: " followed by the boss's name.

b) **Create a CommissionWorker.java.** Class CommissionWorker is derived from Employee. The public methods include a constructor that takes a first name, a last name, a salary, a commission and a quantity of items sold as arguments and passes the first name and last name to the Employee constructor; set methods to assign new values to instance variables salary, commisssion and quantity; an earnings method to calculate a Commission-Worker's earnings; and a toString method that forms a String containing the employee type (I.e. "Commission worker: ") followed by the worker's name.

c) **Create a PieceWorker.java.** Class PieceWorker is derived from Employee. The public methods include a constructor that takes a first name, a last name, a wage per piece and a quantity of items produced as arguments and passes the first name and last name to the Employee constructor; set methods to assign new values to instance variables wagePerPiece and quantity; an earnings method defining how to calculate a PieceWorker's earnings; and a toString method that forms a String containing the type of the employee (ie. "Piece woker: ") followed by the pieceworker's name.

d) **Create a HourlyWorker.java.** Class HourlyWorker is derived from Employee. The public methods include a constructor that takes a first name, a last name, a wage and the number of hours worked as arguments and passes the first name and last name to the Employee constructor; set methods to assign new values to instance variables wage and hours; an earnings method defining how to calculate an HourlyWorker's earnings; and a toString method that forms a String containing the type of the employee (ie. "Hourly worker: ") followed by the hourly worker's name.

CS 116
Week 2: Lab Solution

a) **Boss.java class**

```
// Boss.java
// Boss class derived from Employee.
public final class Boss extends Employee
{
   private double weeklySalary;
   // constructor for class Boss
   public Boss( String first, String last, double salary )
   {
      super( first, last );  // call superclass constructor
      setWeeklySalary( salary );
   }
    // set Boss's salary
   public void setWeeklySalary( double salary )
   {
      weeklySalary = ( salary > 0 ? salary : 0 );
   }
    // get Boss's pay
   public double earnings()
   {
      return weeklySalary;
   }
    // get String representation of Boss's name
   public String toString()
   {
      return "Boss: " + super.toString();
   }
}  // end class Boss
```

b) **CommissionWorker.java class**

```
// CommissionWorker.java
// CommissionWorker class derived from Employee
public final class CommissionWorker extends Employee {
   private double salary;      // base salary per week
   private double commission;  // amount per item sold
   private int quantity;       // total items sold for week
   // constructor for class CommissionWorker
   public CommissionWorker( String first, String last,
      double salary, double commission, int quantity )
   {
      super( first, last );  // call superclass constructor
```

```
      setSalary( salary );
      setCommission( commission );
      setQuantity( quantity );
   }
      // set CommissionWorker's weekly base salary
   public void setSalary( double weeklySalary )
   {
      salary = ( weeklySalary > 0 ? weeklySalary : 0 );
   }
      // set CommissionWorker's commission
   public void setCommission( double itemCommission )
   {
      commission = ( itemCommission > 0 ? itemCommission : 0 );
   }
    // set CommissionWorker's quantity sold
   public void setQuantity( int totalSold )
   {
      quantity = ( totalSold > 0 ? totalSold : 0 );
   }
     // determine CommissionWorker's earnings
   public double earnings()
   {
      return salary + commission * quantity;
   }
    // get String representation of CommissionWorker's name
   public String toString()
   {
      return "Commission worker: " + super.toString();
   }
}  // end class CommissionWorker
```

c) **PieceWorker.java class**

```
// PieceWorker.java
// PieceWorker class derived from Employee
public final class PieceWorker extends Employee {
   private double wagePerPiece; // wage per piece output
   private int quantity;        // output for week
   // constructor for class PieceWorker
   public PieceWorker( String first, String last,
      double wage, int numberOfItems )
   {
      super( first, last );  // call superclass constructor
      setWage( wage );
      setQuantity( numberOfItems );
   }
    // set PieceWorker's wage
```

```
   public void setWage( double wage )
   {
      wagePerPiece = ( wage > 0 ? wage : 0 );
   }
    // set number of items output
   public void setQuantity( int numberOfItems )
   {
      quantity = ( numberOfItems > 0 ? numberOfItems : 0 );
   }
    // determine PieceWorker's earnings
   public double earnings()
   {
      return quantity * wagePerPiece;
   }
    public String toString()
   {
      return "Piece worker: " + super.toString();
   }
}  // end class PieceWorker
```

d) **HourlyWorker.java class**

```
// HourlyWorker.java
// Definition of class HourlyWorker

public final class HourlyWorker extends Employee {
   private double wage;   // wage per hour
   private double hours;  // hours worked for week
   // constructor for class HourlyWorker
   public HourlyWorker( String first, String last,
      double wagePerHour, double hoursWorked )
   {
      super( first, last );   // call superclass constructor
      setWage( wagePerHour );
      setHours( hoursWorked );
   }
   // Set the wage
   public void setWage( double wagePerHour )
   {
      wage = ( wagePerHour > 0 ? wagePerHour : 0 );
   }
    // Set the hours worked
   public void setHours( double hoursWorked )
   {
      hours = ( hoursWorked >= 0 && hoursWorked < 168 ?
         hoursWorked : 0 );
   }
```

```
    // Get the HourlyWorker's pay
   public double earnings() { return wage * hours; }

   public String toString()
   {
      return "Hourly worker: " + super.toString();
   }
}  // end class HourlyWorker
```