**Lab Overview**

**Scenario/Summary**

The purpose of this lab is to practice the concept of inheritance. We will create an Hourly class that *is an* Employee class. In other words, the Hourly class inherits from the Employee class. In addition, we will create a Salary class that inherits from the Employee class. Finally, we will make a Child class work as a Parent class. We will create a Manager class that inherits from the Salary class.

**Deliverables**

The deliverable for this lab includes the following.

* UML Class Diagram
* Word document that contains the project code and screenshots

**Required Software**

Microsoft Office: Word

Use a personal copy, or access the software at [https://lab.devry.edu (Links to an external site.)Links to an external site.](https://lab.devry.edu/).   
Final Step (submitting your lab)

Visio 2013 or newer

Use a personal copy, or access the software at [https://lab.devry.edu (Links to an external site.)Links to an external site.](https://lab.devry.edu/). You can get a personal copy through the Student Software Store.

**Steps: 1**

Visual Studio 2015

Use a personal copy, or access the software at [https://lab.devry.edu (Links to an external site.)Links to an external site.](https://lab.devry.edu/). You can get a personal copy through the Student Software Store.

**Steps: 2–8**

**Lab Steps**

**Step 1**

Create the UML class diagram to show the inheritance relationship between the Parent class, the Children classes, and the Grandchild class.

* Open Visio and create a new diagram using the UML Class template.
* Save the diagram as Week 5—Inheritance UML Class Diagram.
* Drag a class shape to your work area.
  + Change the class name to *Employee.*
  + Add the following attributes, and make them *protected* so that the child classes have direct access to them.
    - fName as a string
    - lName as a string
    - ssn as a string
    - phone as a string
  + Add the following methods, and make them *virtual* so that the Child classes can override them.
    - calculatePay() that returns a float
    - toString() that returns a string
* Now, let's do some inheritance! Drag another class shape to your work area, and place it on the right side of the Employee class.
  + Change the class name to *Hourly.*
  + Add the following attributes and make them *private*because we do not expect to create Child classes off of the Hourly class. Notice that we do not need to add fName, lName, ssn, or phone attributes. We will inherit these attributes from the Parent class!
    - hours as a float
    - rate as a float
  + Add the following methods to override the parent's methods and to allow the child objects to behave differently than the parent objects.
    - calculatePay() that returns a float
    - toString() that returns a string
* Let's do some more inheritance! Drag another class shape to your work area, and place it below the Employee class.
  + Change the name to *Salary.*
  + Add the following attribute, and make it protected because we are going to make a Child class off of the Salary class. Can a child be a parent? You bet! In real life, your mom is someone's child. In software development, the Child class can be a Parent class for another child!
    - annualSalary as a double
  + Add the following methods to override the parent's methods and to allow the child objects to behave differently than the parent objects.
    - calculatePay() that returns a float
    - toString() that returns a string
* Can a child be a parent? Absolutely! Think about it. In real life, your dad is someone's child. Your dad inherits his attributes from his parents. You inherit your attributes from your parents. In software development, a Child class can have a Parent class that can be a Child class of a different Parent class, which, itself, can be a Child class of another Parent class! Drag another class shape to your work area, and place it below the Salary class.
  + Change the name to *Manager.*
  + Add the following attribute, and make it private because we do not expect to create Child classes off of the Manager class.
    - bonus as a double
  + Add the following methods to override the parent's methods and to allow the child objects to behave differently than the parent objects.
    - calculatePay() that returns a float
    - toString() that returns a string
* We have four classes on our Visio diagram. However, they are not related at this point! Let's show the relationship.
  + Drag an inheritance arrow off the template, and drop it on your diagram.
  + Drag the tail of the arrow to the Child class (Hourly) and the head of the arrow to the Parent class (Employee). This inheritance arrow shows everyone that the Hourly class inherits from the Employee class.
  + Drag an inheritance arrow off the template, and connect the Salary class to the Employee class. The direction of the arrow should show that the Salary class inherits from the Employee class.
  + Drag an inheritance arrow off the template, and connect the Manager class to the Salary class. The direction of the arrow should show that the Manager class inherits from the Salary class.
* Save your file!

**Step 2**

Create a C++ project, and call it Week 5—Inheritance. Now, let's realize the UML class diagram, which means let's take the UML class diagram to code.

* Create an Employee class using a separate header file and implementation file.  
  + Review your UML class diagram for the attributes and behaviors
    - The calculatePay() method should return 0.0f.
    - The toString() method should return the attribute values ("state of the object").
* Create an Hourly class using a separate header file and implementation file.
  + The Hourly class needs to inherit from the Employee class
  + The calculatePay() method should return the pay based on the number of hours worked and the pay rate. Remember to calculate overtime!
* Create a Salary class using a separate header file and implementation file.
  + The Salary class needs to inherit from the Employee class.
  + The calculatePay() method should return the annualSalary divided by 52.0f because there are 52 weeks in the year.
* Create a Manager class using a separate header and implementation file.
  + The Manager class needs to inherit from the Salary class (yes, a Child class can be a Parent class).
  + The calculatePay() method should use the Salary's calculatePay() method plus the bonus divided by 52.0f (base pay plus bonus).

**Step 3**

Let's test our classes. Add a Source.cpp file to your project.

* Create a main method for your application.
* In the main method, create three objects—one object using each of the three classes.
* Display the size of the Hourly object. Then, display the size of the first object's memory address (remember, *&* means "address of").
  + The memory address (pointer) is only 4 bytes. However, the Hourly object is huge! If we send the Hourly object across the system bus to the method, it will take rather a lot of time. However, if we pass the pointer across the system bus to the method, it will travel quickly! Pointers make our applications faster!
* Create a method called displayEmployee that accepts an Employee pointer as the parameter. Because Employee is the parent, it can hold child objects. In other words, I can send an Hourly, Salary, or Manager object to this method using the Employee parameter! Here is the prototype:  
    
        void displayEmployee(Employee\* emp);
* In the displayEmployee method, show the object's information one line at a time, including the Weekly Pay (do not use the toString method). Notice that we only have access to the Employee class methods. We do not have access to the specific child methods.
* In the displayEmployee method, convert the Employee object back to the child state. Then, display the specific child information using one line at a time. We use "dynamic\_cast" to convert the object back to the child form. If it is the correct data type, we get an object. If it is the incorrect data type, then we get "NULL". Here is the code for the Manager object.  
    
       Manager\* mgr = dynamic\_cast<Manager\*>(emp);    // try to convert Employee parent object to a Manager child object  
       if( mgr != NULL )                                                        // if the mgr is not NULL, then we have a Manager object!  
       {  
            cout << "Bonus: $" << mgr->getBonus( ) << endl;  
       }
* Go back to the main method. Call the displayEmployee method, and send it the address of the Hourly object. Then, call the displayEmployee method, and send it the address of the Salary object. Finally, call the displayEmployee method, and send it the address of the Manager object.
* Put a breakpoint at the top of your main method and step through your code. Remember to use Step Out if you accidentally step into the C++ code library. Do you see how the child object uses the parent object? The child object builds on the parent object base. Very cool, isn't it?

**Step 4**

Create a Microsoft Word document called Week 5 Lab. At the top, put your information, including your name, course, Week 5 Lab, and the date.

Run your application. Take screenshots while your application is running to demonstrate that it works properly. Remember to hold down the Alt key and the PrtScrn key at the same time (Alt + PrtScrn) to take screenshots of the active window only. Paste these screenshots into your Word document.

Put the file name for each file in the Word document below the screenshots. Bold the file name and increase the font. Copy and paste the code for each file below its file name. You should have nine files for your project.

Submit the Word document.

[Previous](https://devryu.instructure.com/courses/23044/modules/items/2786182)[Next](https://devryu.instructure.com/courses/23044/modules/items/2786186)