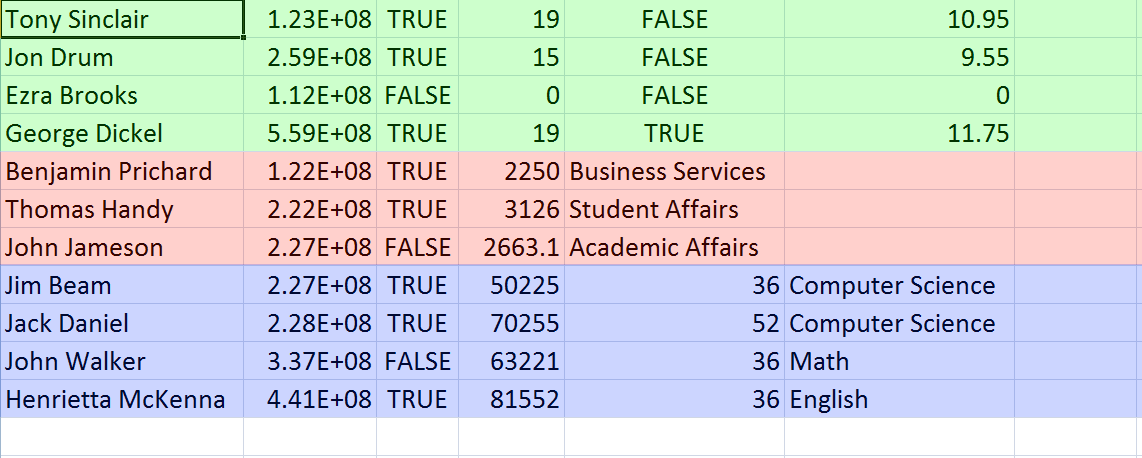
# CptS 122 Assignment #3

This assignment builds off of Lab 5 and uses a similar class structure. In order to complete this lab, you will probably need the StringSplitter used in the lab.

## Objective

In this assignment, you read a list of employees from a CSV file and convert them into corresponding objects. Note that, unlike Lab 4, the CSV file contains more than one type of Employee. In addition to a StudentEmployee, we also have ClassifiedStaff and Faculty that extend our base Employee. Each employee has different information (described more in the UML diagram). Let's take a look at the CSV as it appears in Excel (note: color added to distinguish between Employee types):



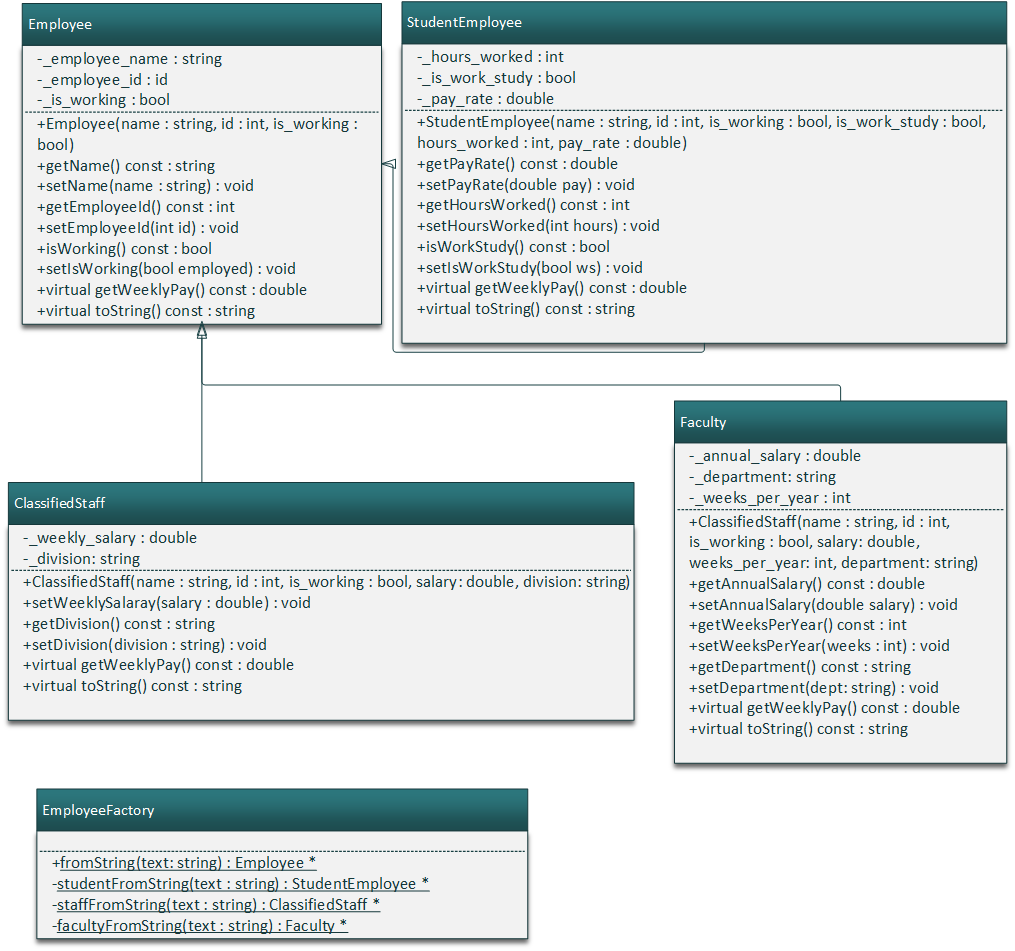
In green, we have StudentEmployees. In red are examples of ClassifiedStaff. Finally, the blue contains Faculty. Note the subtle differences between each Employee type. ClassifiedStaff have one less column. StudentEmployee's last column contains a double whereas Faculty's last column contains a string. You will need to use these differences when converting from CSV into the appropriate object-type.

## Obtaining the Correct Object-type

Speaking of converting into the correct object-type, this assignment also introduces you to the classic design pattern known as the Factory (see <http://en.wikipedia.org/wiki/Factory_method_pattern>). The goal of the factory is to abstract away object creation. Doing so allows us to create more generic code that can be used in a wider variety of circumstances. In our situation, our factory will accept a line from the CSV, convert that line into a pointer of the appropriate class (StudentEmployee, ClassifiedStaff, or Faculty), and return that instance of that class back to the main function.

## UML Diagram

Here's a complete UML diagram of all the classes that you'll need to create:



### Employee and its Derivatives

Most of the Employee and its derivative class' methods should be self-explanatory as they're mostly just basic getters and setters. However, I will comment on the class constructors and the two virtual function.

#### Class Constructors

In order to promote code reuse, derivatives of Employee should call Employee's base constructor. We've talked about how to do this in class, but here's a basic example:

|  |
| --- |
| StudentEmployee( <paramters here>)  : Employee( <pass parameters to employee> )  {  //initialize variables specific to StudentEmployee here.  } |

#### The getWeeklyPay Method

the getWeeklyPay() method is calculated differently based on the derived class:

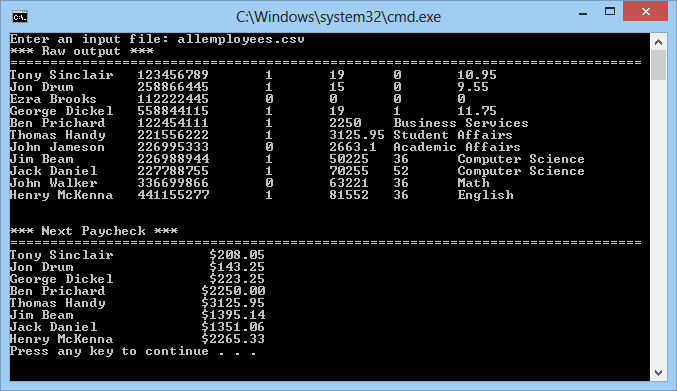
* For StudentEmployees, you get the weekly pay by multiplying their hours worked by their hourly wage
* For ClassifiedStaff, simply return the weekly salary
* For Faculty, return their yearly wage divided by the number of weeks that they work

#### The toString Method

Like reusing the base class' constructor, you should also aim to reuse the base class' toString() method. To do so, simply call Employee::toString() in your overridden method. The format for each toString() is shown in the sample output section.

## Sample Output

Below is sample output from my program:



Note that only people presently working are displayed in the "Next Paycheck" area. In my case, Ezra Brooks isn't working so he isn't displayed in the paycheck section.

## Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your script to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. A basic list of rules can be found on OSBIDE.

## Deliverables

You must upload your program store through OSBIDE no later than midnight on Wednesday, March 5, 2014. For more information on how to submit an assignment, [read this page](https://osbide.codeplex.com/wikipage?title=Submitting an Assignment&referringTitle=Documentation).

## Grading Criteria

Your assignment will be judged by the following criteria:

### Error Free Compile (weight: 5%)

* [0] Your program contains compiler errors.
* [1] Your program compiles without issue.

### Error Free Runtime (weight: 10%)

* [0] Your program throws a runtime exception.
* [1] Your program does not encounter any runtime exceptions.

### Pointer Cleanup (weight: 5%)

* [0] Your program does not delete any dynamically-created pointers
* [1] Your program remembers to delete some, but not all dynamically-created pointers
* [2] Your program deletes all dynamically-created pointers

### User Interface (weight: 10%)

* [0] Your program does not even attempt to follow the UI guidelines
* [1] Your program's UI has major inconsistencies when compared to the sample output
* [2] Your program's UI has minor inconsistencies when compared to the sample output
* [3] Your program completely matches the specified user interface guidelines

### CSV Conversion (weight: 10%)

* [0] Your program does not convert an arbitrary CSV file into an array of Employees
* [1] Your program converts a list of employees in CSV format into an array of Employees

### EmployeeFactory (weight: 20%)

* [0] You do not implement the EmployeeFactory
* [1] The EmployeeFactory is implemented, but is not used
* [2] The EmployeeFactory is implemented and used. However, there exist issues that prevent the factory from correctly converting a line from the CSV file into one or more Employee types.
* [3] Your factory accepts a line from the CSV file and correctly converts it into the corresponding Employee type.

### Wage Calculations (weight: 10%)

* [0] Your program does not compute wages correctly
* [1] Your program correctly computes wages for some employees
* [2] Your program correctly computes the wages for all relevant employees

### Wage Visibility (weight: 5%)

* [0] Your program displays wages for all employees
* [1] Your program displays wages only for employees that are currently working

### Style (weight: 5%)

* [0] Your code contains more than 7 distinct stylistic errors
* [1] Your code contains between 2 and 7 distinct stylistic errors
* [2] Your code contains 1 or 0 stylistic errors

### Employee Class (weight: 5%)

* [0] You are missing three or more of the items below
* [1] You are missing at least one of the items below
* [2] All required functionality, as defined in the list below, has been implemented

#### Employee Requirements

* All methods are fully implemented
* All getters are declared const
* getWeeklyPay() is declared as pure virtual
* toString() is virtual and contains a default implementation

### StudentEmployee Class (weight: 5%)

* [0] You are missing four or more of the items below
* [1] You are missing at least three of the items below
* [2] You are missing at least one of the items below
* [3] All required functionality, as defined in the list below, has been implemented

#### StudentEmployee Requirements

* StudentEmployee extends Employee
* StudentEmployee's constructor calls Employee's constructor to initialize name, id, and is\_working
* StudentEmployee adds the additional functionality as specified in the UML class diagram
* All getters are declared const
* StudentEmployee implement's Employee's pure virtual getWeeklyPay() method
* StudentEmployee overrides Employee's toString() method
* StudentEmployee calls Employee's toString() method to output the employee's name, id, and work status

### ClassifiedStaff Class (weight: 5%)

* [0] You are missing four or more of the items below
* [1] You are missing at least three of the items below
* [2] You are missing at least one of the items below
* [3] All required functionality, as defined in the list below, has been implemented

#### ClassifiedStaff Requirements

* ClassifiedStaff extends Employee
* ClassifiedStaff constructor calls Employee's constructor to initialize name, id, and is\_working
* ClassifiedStaff adds the additional functionality as specified in the UML class diagram
* All getters are declared const
* ClassifiedStaff implement's Employee's pure virtual getWeeklyPay() method
* ClassifiedStaff overrides Employee's toString() method
* ClassifiedStaff calls Employee's toString() method to output the employee's name, id, and work status

### Faculty Class (weight: 5%)

* [0] You are missing four or more of the items below
* [1] You are missing at least three of the items below
* [2] You are missing at least one of the items below
* [3] All required functionality, as defined in the list below, has been implemented

#### Faculty Requirements

* Faculty extends Employee
* Faculty constructor calls Employee's constructor to initialize name, id, and is\_working
* Faculty adds the additional functionality as specified in the UML class diagram
* All getters are declared const
* Faculty implement's Employee's pure virtual getWeeklyPay() method
* Faculty overrides Employee's toString() method
* Faculty calls Employee's toString() method to output the employee's name, id, and work status