CIS 345 – Business Information Systems Development - II

PE 9: Classes and Polymorphism

Learning Outcomes

1. Implement sub classes and call super initializers
2. Override an inherited member
3. Use name mangling to encapsulate data within an object

Program Overview

This program is focused on implementing inheritance and utilizing super and sub data types using lists of only super types. More specifically, if we create Graduate Students and PhD Students, which have an inheritance or an "is-a" relationship with the class Student, we don't need three separate data structures to store those data types. Since Graduate Students and PhD Students are all students, one list of data type Student will suffice.

This program implements such a scenario to manage the polymorphic types of Students so that attributes from sub classes can be accessed as needed.

Instructions

Create a project using the standard naming conventions. Download the **students.py** file**,** rename to **[ASUrite]_students.py,** and copy to your project folder. Don't forget to a**dd your name, class, and class time to Line 1.**

1. Develop **Class Student**.

   1.1. Within the Student class, create an initializer that adds the instance fields first name and last name. The initializer will only accept one argument for a first name (defaults to blank) and not a last name. We will leave the last name blank and fill it in later using our property.

   1.2. Implement name mangling on all instance fields defined in your initializer by adding 2 leading underscores (see readings to learn more) encapsulating the data in your object.

   1.3. Add Properties and setter for each of the fields. Properties must capitalize the field being returned. Setters check incoming value isalpha(), if true save in field and if false save “Unknown”. Note: isalpha returns a bool and spaces are not alpha characters.

1.4. All objects inherit a method called str (), which returns a string representation of the object. For a Student object, that equates to printing the name of a student. To customize the method for the Student class, do the following

   1.4.1. Define a method called str (), which returns a string.
   1.4.2. Within the body of the method, write one line of code to **return** a string which combines both the first name and last name.

2. Add **Class GraduateStudent**

2.1. Create a Class GradStudent that is **derived from or inherits** from the Student class.

2.2. Within the GradStudent class, create an initializer that adds an instance field for thesis. Accept two arguments into the initializer, one for first name and thesis. Thesis must be a required argument where the first name is optional with a default of blank. Call the, super class, Student’s initializer and pass the first name argument.

2.3. Implement name mangling on all instance fields defined in your initializer by adding 2 leading underscores (see readings to learn more) encapsulating the data in your object.

2.4. Create a Property and setter for the thesis. Property must return the thesis in all capital letters. Setter must add the label “Thesis: “ to string being stored.

2.5. Define a method called str (), which returns a string. This will override the overridden str () from our super class Student. Within the body of the method, get the full name by calling the super class, Student’s str method. Return the below string:

```
“<fullname>\n\t<thesis>”
```

3. Add **Class PhDStudent**

3.1. Create a Class PhDStudent that is **derived from or inherits** from the Student class.

3.2. PhDStudent students have a PhD Dissertation. Create a field for the dissertation. Create an initializer that adds the field. Accept two arguments into the initializer, one for first name and dissertation. Dissertation must be a required argument where the first name is optional with a default of blank. Call the, super class, Student’s initializer and pass the first name argument.

3.3. Implement name mangling on all instance fields defined in your initializer by adding 2 leading underscores (see readings to learn more) encapsulating the data in your object.

3.4. Create a Property and setter for the dissertation. Property must return the field in all capital letters. Setter must add the label "Dissertation: " to string being stored.

3.5. Do **NOT** override __str__() in this class. We will figure this out another way.

4. **Find the add_student method,** it prompts the user for and stores the first and last name, after those lines of code you will add an if statement to process the user input. The studentType parameter is an argument passed in that will be a string of P, G, or S.

The *if* statement compares studentType and based on its value, declare **and** instantiate either a Student, GraduateStudent or a PhDStudent object by calling their respective ***constructors***. You may want to process "Student" in the default case. *You should have 3 code paths in your if, only one of which will be executed based on studentType.*

```
studentType equals "G":
    Ask for and store thesis title
    Create an instance of GradStudent and store it in student.


Case "P":
    Ask for and store dissertation title
    Create an instance of PhDStudent and store it in a student

Default:
    Create an instance of Student and store it in student


Set student last name here by way of the student objects properties
Return the student object
```

5. **Find the main method**, within the loop it asks the user if they want to create a generic student, a graduate student, or a PhD student. After the line of code that stores the entry uppercased, add a decision statement.

```
Entry is in studentTypes:
    Call add_student function passing entry as an argument and save returned object
    Append the returned object to the students list
```

Both GraduateStudent and PhDStudent are **specialized** variations of Student (**derived** or **sub** classes). If someone wants a "Student", you may append a graduate student because all graduate students are students! So, you can append any object of type Student, GraduateStudent, or

PhDStudent as appropriate. *This is polymorphism in action.*

6. **Go to last TODO and add the code to print all objects in students list**.

Loop through all the students in the list and print the student.

However, ***some*** PhDStudents are specialized and do not have an overridden str method. They have a dissertation that we need to display. However, all students do not have a dissertation. That is why we cannot access that information for all Students through the Student class. Do the following:

6.1 First, ***TEST*** accessing the **dissertation** information in the student object. Note that you can access all fields, but an exception will occur if the type of student is not a PhDStudent. This is because that field does not exist in the Student and GradStudent classes.

6.2 **Print the first and last name of the Student to the Console**. Remember that you implemented str () method, which **returns** the full name of the Student. Therefore, you do not need to access the individual properties. We will see their first and last name for all student types and GradStudent objects will show the thesis

In fact, you do not even need to call the str () method explicitly. It's called implicitly if you specify only the object name within the print(objectRef) function call.

6.3 After the name information has been printed, write an *if* statement and test if the current student object from the list is a PhDStudent by using the isinstance(Object, Type) function.

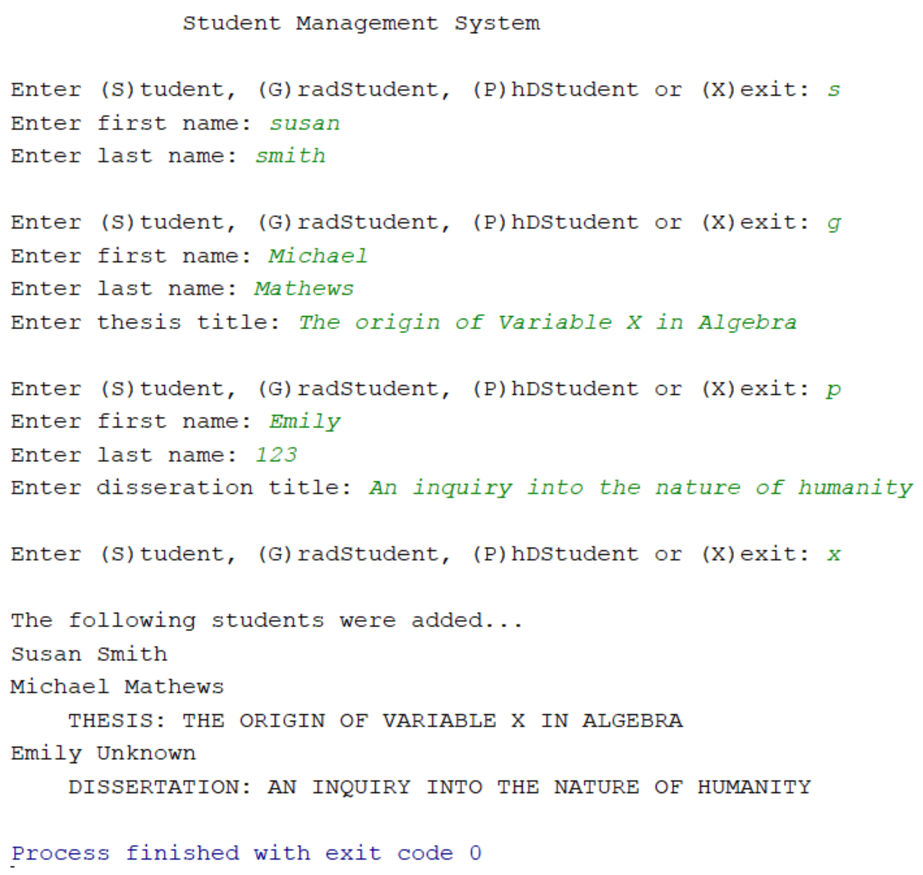
```
If  isinstance(studentObject, PhDStudent)

        Print a tab followed by the dissertation of the PhDStudent.
```

7. Now Test your program!

8. Submit your python module, **[ASUrite]_students.py**, only.

Sample Output



```
            Student Management System

Enter (S)tudent, (G)radStudent, (P)hDStudent or (X)exit: s
Enter first name: susan
Enter last name: smith

Enter (S)tudent, (G)radStudent, (P)hDStudent or (X)exit: g
Enter first name: Michael
Enter last name: Mathews
Enter thesis title: The origin of Variable X in Algebra

Enter (S)tudent, (G)radStudent, (P)hDStudent or (X)exit: p
Enter first name: Emily
Enter last name: 123
Enter disseration title: An inquiry into the nature of humanity

Enter (S)tudent, (G)radStudent, (P)hDStudent or (X)exit: x

The following students were added...
Susan Smith
Michael Mathews
    THESIS: THE ORIGIN OF VARIABLE X IN ALGEBRA
Emily Unknown
    DISSERTATION: AN INQUIRY INTO THE NATURE OF HUMANITY

Process finished with exit code 0
```