|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Admin\Desktop\download.jpg | USMAN INSTITUTE OF TECHNOLOGY | | | | | |  |
|  |  |  |  |  |  |  |  |
|  | Department of Computer Science  CS121 Object Oriented Programming | | | | | |  |
|  |  | Lab # 09  Inheritance | | | |  |  |
|  | Objective:  This experiment introduces the students to the fundamental concept of Inheritance. | | | | | |  |
|  | **Name of Student:**  **Roll No: Sec.**  **Date of Experiment:** | | | | | |  |
|  | **Marks Obtained/Remarks:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |  |

**Lab 09: Inheritance**

Classes are written to organize and structure code into meaningful blocks, which can then be used to implement the business logic. These implementations are used in such a way that more complex parts are abstracted away to provide for simpler interfaces which can then be used to build even simpler blocks. While doing this we will find that there are lots of times when we will need to establish relationships between the classes that we build. These relationships can then be established using either association, inheritance, aggregation or composition. In this lab you will get to know how to build relationships between classes using inheritance and the syntax that is needed.

**What is Inheritance?**

As we know already that inheritance an object is based on another object. When inheritance is implemented, the methods and attributes that were defined in the base class will also be present in the inherited class. This is generally done to abstract away similar code in multiple classes. The abstracted code will reside in the base class and the previous classes will now inherit from the base class. Python allows the classes to inherit commonly used attributes and methods from other classes through inheritance. We can define a base class in the following manner:

class DerivedClass (BaseClass):

pass

*Example 1:*

* class Car: #base class or parent class
* def \_\_init\_\_(self, name, mileage):
* self.name = name
* self.mileage = mileage
* def description(self):
* return ("The {self.name} car gives the mileage of {self.mileage}km/l“)
* class BMW(Car): #derived class or child class
* pass
* class Audi(Car): #derived class or child class
* def audi\_desc(self):
* return ("This is the description method of class Audi.“)
* obj1 = BMW("BMW 7-series",39.53)
* print(obj1.description())
* obj2 = Audi("Audi A8 L",14)
* print(obj2.description())
* print(obj2.audi\_desc())

**Initializing Base Class Data Fields**

When instantiating a child class object, it is important to explicitly initialize the date fields of the parent class as well. In Python super() refers to the parent of the calling object. Following is the syntax to call the initializer method of the parent class from the child class

* super().\_\_init\_\_()

**Method Overriding**

Subclass inherits methods from Super class. It may happen that the Subclass may require to modify implementation of a method defined in the Super class. This concept is known as ***method overriding.*** To override a method, the method must be defined in the subclass using the same header as in its super class

# Student Exercise

Following UML diagram represent a Geometric shape class and a Rectangle class being inherited from it. For a rectangle, formula to compute its perimeter and area are as follows

Perimeter = 2 x (width + length)

Area = width x length

|  |
| --- |
| **GeometricShape** |
| - color: String |
| GeometricShape(color: String)  getColor(): String |

|  |
| --- |
| **Rectangle** |
| - width: float  - length: float |
| Rectangle(width: float, length: float, color: String)  getWidth(): float  getLength(): float  getPerimeter(): float  getArea(): float |

Exercise 1

Write Python code for implementing the classes

**Code:**

class GeometricShape:

    def \_\_init\_\_(self,color):

        self.\_\_color = color

    def getColor(self):

        return self.\_\_color

    def getPerimeter(self):

        pass

    def getArea(self):

        pass

class Rectangle(GeometricShape):

    def \_\_init\_\_(self, w=1, l=1, color=None):

        super().\_\_init\_\_(color)

        self.\_\_w = w

        self.\_\_l = l

    def setW(self, w):

        self.\_\_w = w

    def setL(self, l):

        self.\_\_l = l

    def getW(self):

        return self.\_\_w

    def getL(self):

        return self.\_\_l

    def getPerimeter(self):

        return 2 \* (self.\_\_w + self.\_\_l)

    def getArea(self):

        return self.\_\_w \* self.\_\_l

Exercise 2

Implement a program to create three rectangle objects. Instantiate all three based on the user input. Display the color, perimeter, and area of all rectangles.

**Code:**

#Exercise 2

width, height, color = input("Enter width, height and color respectively: ").split(" ")

width = float(width)

height = float(height)

color = str(color)

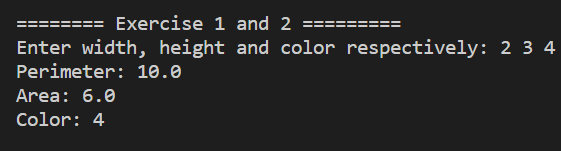
obj = Rectangle(width, height, color)

print("Perimeter:", obj.getPerimeter())

print("Area:", obj.getArea())

print("Color:", obj.getColor())

**Output:**

****

Following UML Class diagram represent classes for an Undergraduate and a Graduate student both being inherited from the Student class. For an undergraduate student, the grade would be ‘Pass’ if the test score is greater than or equals to 60, and ‘Fail’ otherwise. However, for a graduate student the grade would be ‘Pass’ if the test score is greater than or equal to 70, and ‘Fail’ otherwise

|  |
| --- |
| **Student** |
| - name: String  - testScore: int  - grade: String |
| Student (name: String)  getName(): String  setTestScore(int): void  computeGrade(): void  getGrade(): String |

|  |
| --- |
| **UndergraduateStudent** |
|  |
| UndergraduateStudent(name: String)  computeGrade(): void |

|  |
| --- |
| **GraduateStudent** |
|  |
| GraduateStudent(name: String)  computeGrade(): void |

Exercise 3

Write Python code for defining the classes

**Code:**

class Student:

    def \_\_init\_\_(self, name, score,grade):

        self.\_\_name = name

        self.\_\_testScore = score

        self.\_\_grade = grade

    def getName(self):

        return self.\_\_name

    def getScore(self):

        return self.\_score

    def getGrade(self):

        return self.\_\_grade

    def setTestScore(self, score):

        self.\_\_testScore = score

    def getTestScore(self):

        return self.\_\_testScore

    def computeGrade(self):

        pass

class Undergraduate(Student):

    def \_\_init\_\_(self, name, score, grade):

        super().\_\_init\_\_(name, score, grade)

    def computeGrade(self):

        if self.getTestScore() >= 60:

            self.setTestScore('Pass')

        else:

            self.setTestScore('Fail')

    def \_\_str\_\_(self):

        return "Name: {}, Score: {}, Grade: {}".format(self.getName(), self.getTestScore(), self.getTestScore())

class Graduate(Student):

    def \_\_init\_\_(self, name, score, grade):

        super().\_\_init\_\_(name, score, grade)

    def computeGrade(self):

        if self.getTestScore() >= 60:

            self.setTestScore('Pass')

        else:

            self.setTestScore('Fail')

    def \_\_str\_\_(self):

        return "Name: {}, Score: {}, Grade: {}".format(self.getName(), self.getTestScore(), self.getTestScore())

Exercise 4

Implement a program to create three student objects. Depending on the user, instantiate them as either an Undergraduate student or a Graduate student. Set Test Scores for all the three students based on the user input. Assume test scores to be in the range of 0 and 100, inclusive. Compute grades for the three students. Finally display their names along with the test scores and grades.

**Code:**

name, score = input("Enter name and score: ").split(" ")

score = float(score)

obj = Undergraduate(name, score, "")

obj.computeGrade()

print(obj)

name, score = input("Enter name and score: ").split(" ")

score = float(score)

obj = Graduate(name, score, "")

obj.computeGrade()

print(obj)

**Output:**

Text

Description automatically generated