|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Admin\Desktop\download.jpg | USMAN INSTITUTE OF TECHNOLOGY | | | | | |  |
|  |  |  |  |  |  |  |  |
|  | Department of Computer Science  CS121 Object Oriented Programming | | | | | |  |
|  |  | Lab # 12  Aggregation and Composition | | | |  |  |
|  | Objective:  This experiment introduces the students to the concept of Aggregation and Composition. | | | | | |  |
|  | **Name of Student:**  **Roll No: Sec**  **Date of Experiment:** | | | | | |  |
|  | **Marks Obtained/Remarks:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |  |

**Lab 12: Aggregation and Composition**

Association is a loose relationship in which objects of one class “know” about objects of another class. There are two types of Association Relationships.

1. Aggregation is part-whole relationship (is-part-of)
2. Composition is like aggregation, but is more strict

Aggregation and Composition are subsets of association and are specific cases of association. In both aggregation and composition object of one class "owns" object of another class, with a subtle difference

**Aggregation**

In aggregation, the objects in relationship have their own life-cycle but there exists an ownership as well. Typical whole/part relationship but it may or may not denote physical containment – the objects may or may not be a part of the whole. Aggregation is identified by the phrase “is-part-of”. In Aggregation, parts may be independent of the whole, and parts may be shared between two whole instances.

A department in a university may have many teachers. However, if the department is terminated, the teacher objects wouldn't be destroyed. The objects participating in an aggregation relationship do not have cyclic aggregation relationships i.e., a whole has a part but the reverse is not true.

Aggregation is represented in UML using a line with a hollow diamond. Direction of diamond is towards the ‘whole’



**Composition**

Composition is a specialized / stronger form of Aggregation. In a composition relationship part objects do not have their lifecycle without whole object. If a whole object is deleted, all its part objects will also be deleted. This also represents “death” relationship.

For example, a house has multiple rooms. There is no independent life of an individual room. Any one room cannot belong to two different houses and if the house is deleted, the room will automatically be deleted.

In UML the composition is represented by a solid diamond followed by a line. The class that has the composition symbol (the solid diamond) on its end of the association line is the ‘whole’ and the classes on the other end of the line are the ‘part(s)’



# Student Exercise

A class called Quadrilateral which models a quadrilateral with 4 vertices, is designed as shown as a following UML diagram. The Quadrilateral class uses four Point instances as its four vertices

|  |
| --- |
| **Point** |
| - x : int  - y : int |
| + Point(int, int)  + getX() : int  + getY() : int  + distance(Point) : float |

|  |
| --- |
| **Quadrilateral** |
| - vertex1: Point  - vertex2: Point  - vertex3: Point  - vertex4: Point |
| + Quadrilateral (Point, Point, Point, Point)  + \_\_str\_\_() : String  + getPerimeter() : float  + isSquare() : Boolean |

* \_\_str\_\_() method that returns a string description of the quadrilateral instance in the format "[v1(x1,y1), v2(x2,y2), v3(x3,y3), v4(x4,y4)]"
* getPerimeter() method that returns the length of the perimeter in double. You should use the distance() method of Point to compute the perimeter
* isSquare() method returns a TRUE if all four sides of a quadrilateral are equal, otherwise returns a FALSE

Exercise 1

Implement both the classes in Python

Exercise 2

Write a test application to perform following tasks

* Instantiate four instances of Point class from user input
* Instantiate a quadrilateral instance using the four instances of Point instantiated earlier
* Invoke each method of Quadrilateral class

**Code:**

class Point:

    def \_\_init\_\_(self,x,y):

        self.x=x

        self.y=y

    def getX(self):

        return self.x

    def getY(self):

        return self.y

    def distance(self,Point):

        return ((self.x-Point.x)\*\*2+(self.y-Point.y)\*\*2)\*\*0.5

class Quadrilateral:

    def \_\_init\_\_(self,p1,p2,p3,p4):

        self.vertex1=p1

        self.vertex2=p2

        self.vertex3=p3

        self.vertex4=p4

    def \_\_str\_\_(self):

        return f"{self.vertex1.x},{self.vertex1.y} {self.vertex2.x},{self.vertex2.y} {self.vertex3.x},{self.vertex3.y} {self.vertex4.x},{self.vertex4.y}"

    def getPerimeter(self):

        return self.vertex1.distance(self.vertex2)+self.vertex2.distance(self.vertex3)+self.vertex3.distance(self.vertex4)+self.vertex4.distance(self.vertex1)

    def isSquare(self):

        return self.vertex1.distance(self.vertex2)==self.vertex2.distance(self.vertex3)==self.vertex3.distance(self.vertex4)==self.vertex4.distance(self.vertex1)

p1, p2 = input("Enter p1 and p2: ").split(" ")

p3, p4 = input("Enter p3 and p4: ").split(" ")

p5, p6 = input("Enter p5 and p6: ").split(" ")

p7, p8 = input("Enter p7 and p8: ").split(" ")

vertex1=Point(int(p1),int(p2))

vertex2=Point(int(p3),int(p4))

vertex3=Point(int(p5),int(p6))

vertex4=Point(int(p7),int(p8))

quad=Quadrilateral(vertex1,vertex2,vertex3,vertex4)

print(quad)

print(quad.getPerimeter())

print(quad.isSquare())

**Output:**

**Text

Description automatically generated**

Author class models a book author. A Book class models a book written by a single author. Following UML diagram models their relationship.

|  |
| --- |
| **Author** |
| - name: String  - email: String |
| + Author(String, String)  + \_\_str\_\_() : String |

|  |
| --- |
| **Book** |
| - title : String  - author : Author  - price : double |
| + Book(String, double)  + \_\_str\_\_() : String |

* Book() constructor is used to assign the title and the price of the book instance
* Instantiate author instance from user input in the Book constructor
* \_\_str\_\_() method of Book class returns “Book[name=?,Author[name=?,email=?],price=?”]

Exercise 3

Write class definitions in Python

Exercise 4

Write a test application to test the Book class by instantiating an instance and calling the \_\_str\_\_() method

**Code:**

class Author:

    def \_\_init\_\_(self, name, email):

        self.name = name

        self.email = email

    def \_\_str\_\_(self):

        return f"{self.name}, ({self.email})"

class Book:

    def \_\_init\_\_(self,title,author,price):

        self.title=title

        self.author=author

        self.price=price

    def \_\_str\_\_(self):

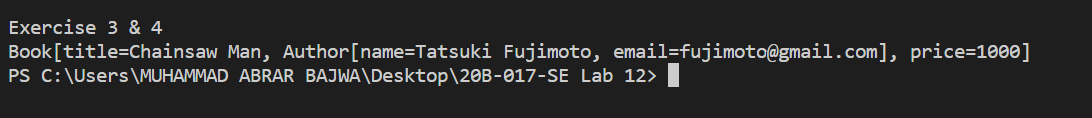
        return f"Book[title={self.title}, Author[name={self.author.name}, email={self.author.email}], price={self.price}]"

author=Author("Tatsuki Fujimoto","fujimoto@gmail.com")

book=Book("Chainsaw Man",author,1000)

print(book)

**Output:**

****