***I confirm that this assignment is my own work.***

***Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.***

**

KAPLAN INTERNATIONAL COLLEGE LONDON – IYO COMPUTER SCIENCE PROGRAMME

Module Title :

object oriented programming

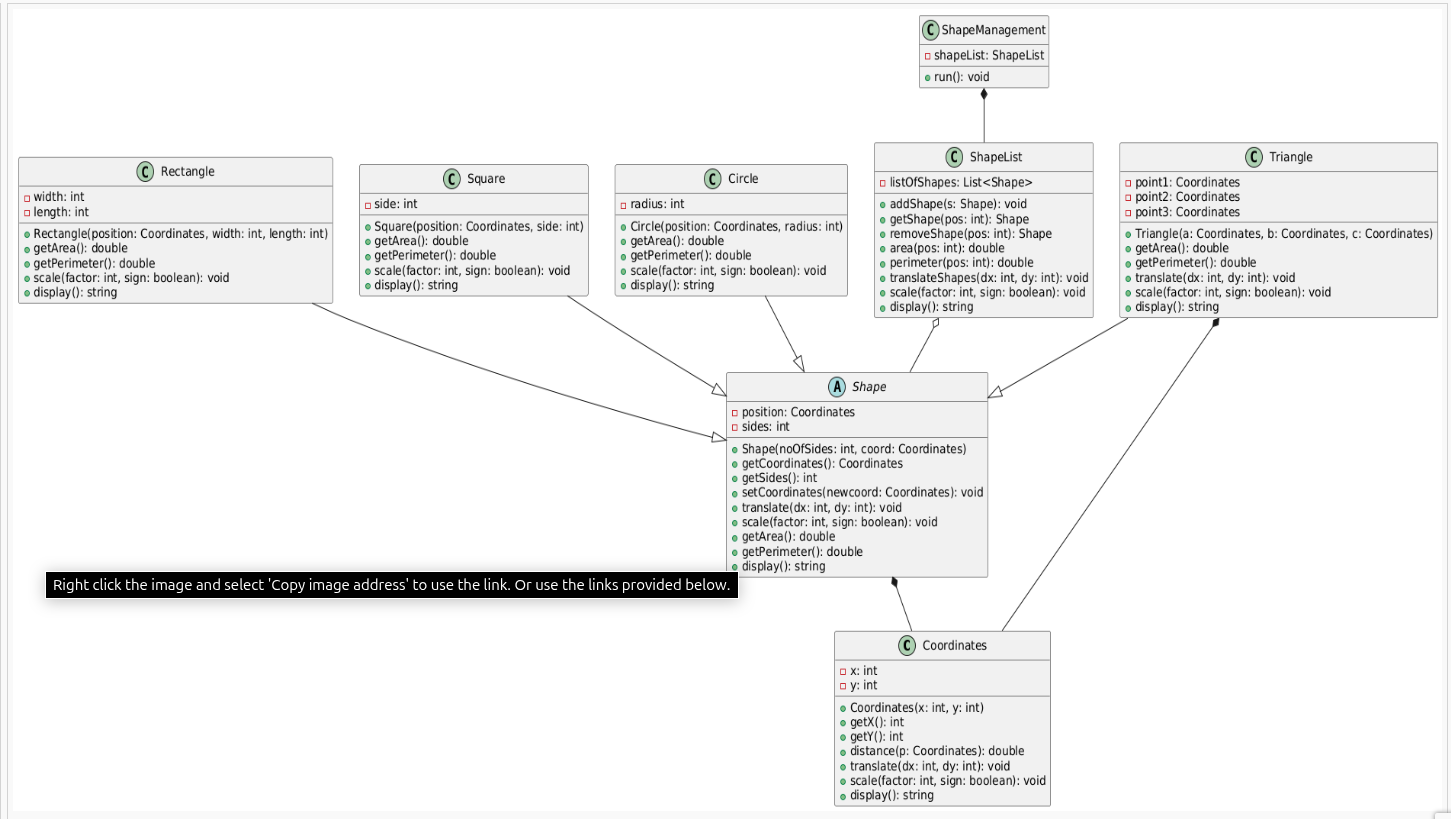
Module Code : IY4101

Pnumber : P482285

Name : Yanis Kaced

# 2 . design of the program :

UML :



### 2.1 UML Class Diagram and Relationships :

This program was built using object-oriented programming. To plan it well, I started with a UML class diagram that shows how all the classes are connected. At the center of the design is an abstract class called Shape. This class gives a common structure to all shapes. It includes methods like getArea(), getPerimeter(), translate(), scale(), and display().

Each shape type such as Rectangle, Square, Circle, and Triangle inherits from the Shape class. This means they all share the same basic functions, but each one calculates things like area and perimeter in its own way. This helps the program use all shapes through one shared interface (Shape\*), which is very flexible.

The program also uses composition. The ShapeList class stores many shapes using a vector of pointers. Then, the ShapeManagement class contains the ShapeList and handles everything the user wants to do like adding, removing, or showing shapes. So the ShapeManagement class "has a" ShapeList, and the ShapeList "has many" Shape objects.

In summary:

* Shapes inherit from Shape (inheritance).
* ShapeList is used inside ShapeManagement (composition).
* The design makes the program easy to extend and keep clean

### 2.2 Design of the Solution

The program was designed to help users create, manage, and test different geometric shapes like rectangles, circles, squares, and triangles. To do this in a clean and flexible way, I used object-oriented programming (OOP). This made it easier to organize the code and avoid repeating the same logic in many places.

At the heart of the solution is the abstract class Shape. This class defines the basic structure for all shapes. Each shape has to know how to calculate its own area, perimeter, and how to move (translate) or grow/shrink (scale). Because of that, each shape class (like Circle or Rectangle) overrides the methods in Shape and adds its own logic.

The Coordinates class is used to store the x and y positions of a shape on the grid. For example, a square just needs one coordinate, but a triangle uses three. The translate() and scale() methods use this class to change the shape’s position or size.

To manage many shapes at once, the ShapeList class stores them in a list using Shape\* pointers. It has methods to add, remove, display, and change all shapes together. This is useful when the user wants to move or scale every shape in one go.

The ShapeManagement class takes care of user interaction. It shows the menu and calls the right functions when the user wants to do something. For testing, I also created a helper method that adds shapes directly without using the menu. This made it easier to run the test plan from the assignment.

Overall, the design is simple but powerful. It follows OOP rules like inheritance and composition, which keeps the code clean, readable, and easy to improve in the future.

### 2.3 Use of AI and Observations

AI was used throughout this project to support my decision-making during the development process. Instead of relying on AI to write the entire program for me, I used it more like a mentor or second opinion. For example, when I had to decide how to structure my classes, I asked the AI whether to use inheritance or composition and why. It helped me compare different design choices and guided me toward solutions that followed good object-oriented programming practices.

AI also helped me make small decisions, like whether to validate input in the constructor or how to avoid memory leaks when using pointers. In each case, I read and understood the suggestions, made my own judgment, and then applied the solution. I also asked for explanations in simple terms to make sure I really understood the concepts, especially when I faced logical bugs or confusion in the test plan.

Using AI this way helped me improve my problem-solving and design skills while still doing all the coding myself. It made my decisions more thoughtful and my project more solid.

### 3.1 Description of Problems Encountered

At the beginning of the project, one of the main challenges I faced was understanding how to organize the classes properly using object-oriented programming. I was confused about which methods should go into which class, especially with inheritance and composition. It took me some time and testing to understand how the abstract Shape class connects to the specific shapes like Circle, Square, and Triangle.

Another problem was with constructor errors — especially when I didn’t provide a default constructor for the Coordinates class. This caused compilation errors in some shape classes like Triangle where I used multiple Coordinates as attributes. I fixed this by learning why a default constructor was needed and adding one that set coordinates to (0, 0).

I also had trouble with memory management when using Shape\* pointers in the ShapeList. At first, I forgot to delete the shapes I removed, which led to memory leaks. I later added a destructor to clean up all shapes and prevent these issues.

Finally, during testing, I realized that floating-point values like area and perimeter couldn’t be compared using == because of rounding errors. I had to learn to use fabs() to compare values with a small tolerance instead.

These challenges helped me learn more deeply about how C++ works and how to think like a programmer.

### 3.2 Testing Strategy

To make sure the program worked correctly, I used a mix of unit testing and functional testing.

For unit tests, I created test functions for each class. These tests checked if the constructors, methods like getArea(), getPerimeter(), translate(), and scale() gave the expected results. I also included input validation tests to make sure that invalid values (like negative coordinates or scale factors) were handled correctly. Instead of using a testing library, I wrote simple if-else statements to check if the results matched the expected values, and printed out whether each test passed or failed.

After testing each class on its own, I moved to functional testing based on the test plan given in the assignment. I created a test scenario that followed all the steps in the plan: adding shapes, displaying them, removing a shape, translating, scaling, and handling errors. I printed the results to the console and compared them with the expected outputs.

This step-by-step strategy helped me catch errors early, understand where problems were coming from, and prove that the full system worked as required.

### 3.3 Sample Output Screenshots and Test Plan Results

To complete the testing process, I ran the full functional test scenario based on the assignment’s test plan. Each step of the test was performed using the actual program, and the results were printed to the console. I captured screenshots of the output to show that each part of the program worked as expected.

The actions included creating different types of shapes, displaying them, removing a shape, translating and scaling all shapes, and checking the area and perimeter of specific shapes. I also tested invalid inputs, such as removing a shape at an index that doesn’t exist. The program handled those cases correctly and displayed warning messages without crashing.

I used these screenshots as evidence in my test plan table. Each test step includes:

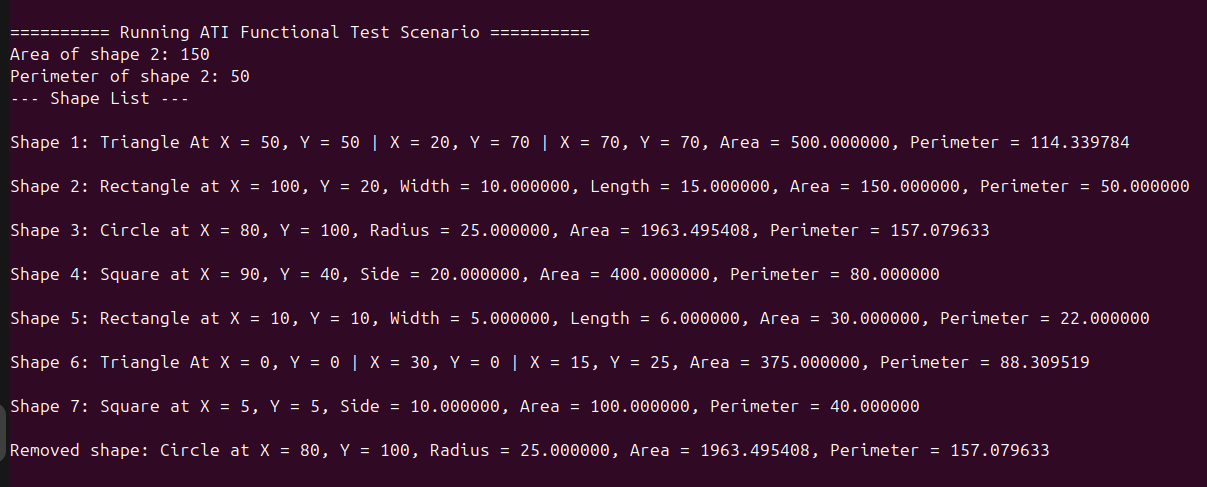
* What action was taken
* What result was expected
* What actually happened
* Whether the test passed or failed

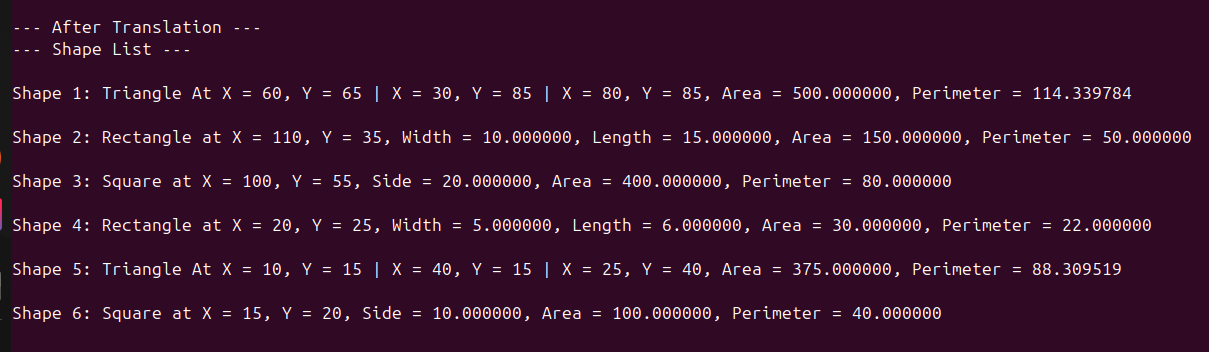
All tests passed successfully, and the program behaved as expected.

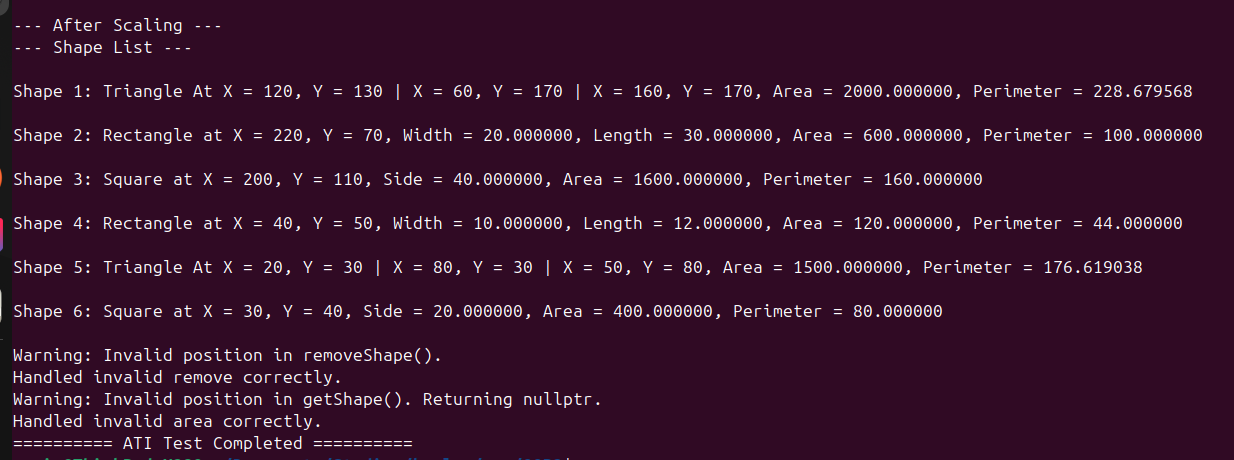
Teste plan :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action Taken | Expected Result | Actual Result | Pass/Fail |
| 1 | Create Triangle (50,50), (20,70), (70,70) | Triangle added successfully with valid coordinates | Triangle at (50,50)... shown in display | Pass |
| 2 | Create Rectangle at (100,20) with width 10, length 15 | Rectangle added with area 150, perimeter 50 | Rectangle data correct in display | Pass |
| 3 | Create Circle at (80,100) with radius 25 | Circle added, area ≈ 1963.49, perimeter ≈ 157.08 | Circle area/perimeter printed correctly | Pass |
| 4 | Create Square at (90,40), side = 20 | Square with area = 400, perimeter = 80 | Display shows correct values | Pass |
| 5 | Add 3 more shapes | All added successfully | Shapes listed in display | Pass |
| 6 | Get area and perimeter of shape at position 1 | Area = 150, Perimeter = 50 (Rectangle) | Output matches expected | Pass |
| 7 | Display all shapes | All 7 shapes displayed with correct info | All shapes printed to console | Pass |
| 8 | Remove shape at index 2 | Circle removed, memory freed | Console shows shape removed | Pass |
| 9 | Translate all shapes by (10,15) | All shapes moved by that offset | Coordinates updated in display | Pass |
| 10 | Display all shapes after translation | Updated positions shown | Display confirms translation | Pass |
| 11 | Scale all shapes by factor 2 | Dimensions doubled | Area and perimeter updated correctly | Pass |
| 12 | Display all shapes after scaling | Larger shapes and updated coordinates | Output confirmed | Pass |
| 13 | Try invalid remove and area (index out of range) | Error handled, no crash, warning message shown | "Handled invalid..." messages printed | Pass |

Test output :







4. Evaluation of the Program :

This project helped me practice and apply object-oriented programming concepts in a real way. I learned how to use inheritance, composition, and polymorphism to build a flexible and organized system. The final program works as expected and meets all the requirements from the assignment.

The design is clear and easy to extend. For example, if I wanted to add a new shape type in the future, I would only need to create a new class that inherits from Shape. The ShapeList and ShapeManagement classes would be able to use it without changing their code, because they rely on the shared interface.

I also made sure that the program handles errors well. If a user tries to remove a shape at an invalid position, or use a negative scale factor, the program shows a warning message instead of crashing. I added these checks after testing the system carefully.

One thing I could improve is separating the user interface from the shape logic. Right now, input and output are mixed with shape operations. In a larger project, I would separate the logic and the interface to make the code easier to maintain and test.

Overall, I’m happy with the results. I wrote clean code, tested everything, and used AI support to help me make better design decisions without skipping the learning process. I now feel more confident working with classes and object-oriented design in C++.

## 5. Conclusion

This project was a valuable experience for me to practice real programming skills using C++. I learned how to apply object-oriented concepts like inheritance, abstraction, and composition to build a full system that manages different shapes. I also gained a better understanding of how to organize code into classes and how to test each part of a program.

One of the things I’m most proud of is that I didn’t just focus on getting the program to work — I also paid attention to writing clean, readable code and making sure every shape behaved correctly. I tested all the features step by step and handled edge cases like invalid input or empty shape lists.

Working on this project also helped me develop better problem-solving skills. I ran into several errors, especially with constructors and pointers, but I took the time to understand the issues and fix them in a proper way. The use of AI helped me make better decisions without doing the work for me.

In the end, I completed a working program that follows the assignment requirements, handles errors safely, and is easy to understand and maintain. This project helped me become a better programmer and gave me more confidence in using object-oriented programming in C++.

Refrences :

* GeeksforGeeks. (n.d.). Object Oriented Programming in C++. [online] Available at: https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/ [Accessed 3 Jul. 2025].
* ChatGPT. (2025). Assistance with OOP Design and C++ Debugging. [online] OpenAI. Available at: [https://chat.openai.com](https://chat.openai.com/) [Accessed 3 Jul. 2025].
* IBM. (n.d.). OOP Concepts: Inheritance, Polymorphism, Abstraction, Encapsulation. [online] IBM Developer. Available at: https://developer.ibm.com/articles/oo-concepts/ [Accessed 3 Jul. 2025].