Shapes blackboard

Purpose of work

Gain practical skills in implementing systems using polymorphism and inheritance principles in mind. The goal of this task is to design and implement a set of common expandable interfaces for objects.

Task

Develop a console-based application which will represent a shapes board with the ability to draw different types of shapes on it. The main idea is that the user has an empty blackboard and can interact with it in a few ways: show the blackboard, place or remove some shape within the blackboard or clear it completely. Additionally, the user should be able to save and load the state of the blackboard from the file.

Command	Parameters	Description	Result	Example
draw	-	Draw blackboard to the console	ASCII-drawn blackboard with all shapes on it, or empty.	> draw
list	-	Print all added shapes with their IDs and parameters	List of all **added** shapes. One shape per line with shape ID and all information related to the shape	> list > id circle radius coordinates > id square width height coordinates
shapes	-	Print a list of all available shapes and parameters for add call	List of all available shapes. One shape per line with shape all information related to the shape (defined by student)	> shapes > circle radius coordinates > square width height coordinates
add	shape parameters	Add shape to the blackboard	Shape is added to the blackboard.	> add circle 10 10 5
undo	-	Remove the last added shape from the blackboard	The last added shape is removed.	> undo
clear	-	Remove all shapes from blackboard	All shapes are removed from the blackboard.	> clear
save	file-path	Save the blackboard to the file	Blackboard is saved to the file. The state of the blackboard is untouched.	> save D:\test.bb

load	file-path	Load a blackboard from the file.	Blackboard is loaded from the file. The previous state of the blackboard is cleared.	> load D:\test.bb

Constrains

There are some constraints and preferences related to that task.

1. Shapes are drawn as frames.
2. Shapes can overlap. In that case, shapes should be correctly drawn.
3. Shapes cannot be placed outside of the board.
4. Shapes bigger than the board cannot be placed on the board.
5. Shapes that are partially outside of the board can be placed, but they are cropped.
6. Figures with the same type and parameters cannot be placed on the same spot.
7. There is no limit to the shape count on the board.
8. Empty boards should also be correctly saved and loaded.
9. A file with an invalid board should not be loadable.
10. There must be shapes with different parameter count (radius, width\height, height\angle).

Task milestones

1. Design your system on paper or by using some online tools, please use squares and arrows to draw classes and their relations, no specific notation is needed in this task.
2. Implement the blackboard drawing part of the system.
3. Implement the single shape\figure class and try to draw it with the board.
4. Try to add more shapes. At that stage, you can evaluate the interface of the shape\figure class and re-design it. Take a note, of what played what, and what did not.
5. Try to add a CLI interface for the developed application and related logic.
6. Thinks how parameters should be passed to the shapes with different parameter count.
7. Test the solution, and develop your input data files

Control questions

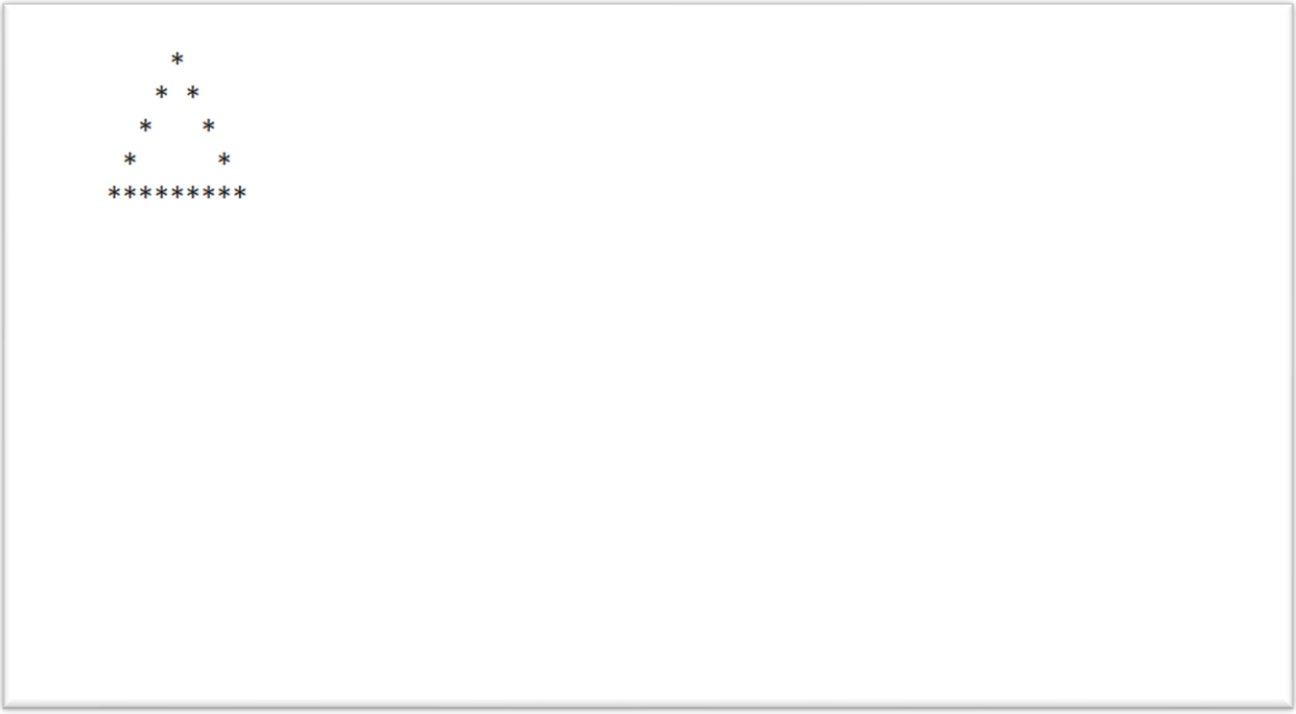
1. What is inheritance?
2. What is polymorphism? How can it be achieved?
3. What is operator overloading?
4. Which types of constructors do you know?
5. What does the **virtual** keyword mean?
6. What does the **override** keyword mean?
7. What does the **final** keyword mean? In which contexts it can be used?
8. What does the **pure** method mean?
9. When do we need virtual functions?
10. What is **dynamic_cast** and how is it implemented under the hood?
11. Explain the difference between unique_ptr and shared_ptr.
12. Explain the difference between shared_ptr and weak_ptr.
13. RAII idiom.

Evaluation

Usage of inheritance and polymorphism principles	4
The shape interface is uniform and strictly defined	2
The draw interface is the same for figures of all shapes and parameter count	2
There are 4 or more shapes available to the user (e.g. triangle, rectangle, circle, line)	1
All constraints are satisfied	1
Total	10

Please note that the evaluation point will be reduced because of incorrect answers to the questions, wrong program output and logic, fatal errors, and crashes during a program run-time. Also, you must use git with 3+ commits to get the highest point.

Example board



(code for this example is provided below)

```cpp
#include <iostream>
#include <vector>

// Define the size of the board
const int BOARD_WIDTH = 80;
const int BOARD_HEIGHT = 25;

// Struct to define the board
struct Board {
    std::vector<std::vector<char>> grid;

    Board() : grid(BOARD_HEIGHT, std::vector<char>(BOARD_WIDTH, ' ')) {}

    void print() {
        for (auto& row : grid) {
            for (char c : row) {
                std::cout << c;
            }
            std::cout << "\n";
        }
    }

    void drawTriangle(int x, int y, int height) {
        if (height <= 0) return; // Ensure the triangle height is positive and sensible

        for (int i = 0; i < height; ++i) {
            int leftMost = x - i; // Calculate the starting position
            int rightMost = x + i; // Calculate the ending position
            int posY = y + i; // Calculate the vertical position

            // Draw only the edges/border of the triangle
            if (posY < BOARD_HEIGHT) {
                if (leftMost >= 0 && leftMost < BOARD_WIDTH) // Check bounds for left
most position
                    grid[posY][leftMost] = '*';
                if (rightMost >= 0 && rightMost < BOARD_WIDTH && leftMost != rightMost)
// Check bounds for right most position
                    grid[posY][rightMost] = '*';
            }
        }

        // Draw the base of the triangle separately
        for (int j = 0; j < 2 * height - 1; ++j) {
            int baseX = x - height + 1 + j;
            int baseY = y + height - 1;
            if (baseX >= 0 && baseX < BOARD_WIDTH && baseY < BOARD_HEIGHT) // Check
bounds for each position on the base
                grid[baseY][baseX] = '*';
        }
    }
};

int main() {
    Board board;
    board.drawTriangle(10, 1, 5);
    board.print();

    return 0;
}
```

Some valid scenarios