Extended Shapes Blackboard

Purpose of work

Gain practical skills in extending an already designed system. Improve confidence with already learned tools and principles of the OOP.

Task

In this assignment, you should extend the blackboard that was already developed from the previous assignment. This extension will be presented in the form of new commands, and the appearance of new stricter requirements to the already developed parts.

CMD	Parameters	Description	Result	Example
draw	-	Draw blackboard to the console	ASCII-drawn blackboard with all shapes on it in regards to their colour and fill mode or empty.	> draw < [board]
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][shape][params] < [id][shape][params]
shapes	-	Print a list of all available shapes and parameters for add call	List of all available shapes. One shape per line with a shape all information related to the shape (defined by the student)	> shapes < circle [params] < box [params]
add	shape fill|frame red|green|… parameters	Add shape with specified colour and fill mode to the blackboard	A shape with a specified colour is added to the blackboard.	> add fill red box 10 5 < [id] box red 10 5
select	ID or coordinates	Select a shape with an ID or foreground shape by the selected coordinates.	The shape was selected. Print info about the selected shape to the output. Selection by coordinate works not only by the “centre” of the figure but also by selecting any shape that occupies the “point”. Frame figures do not occupy space inside them.	> select 0001 < [shape][parameters] > select 0 5 < shape was not found

remove	-	Remove the selected shape from the blackboard	The selected shape was removed	> select 0001 < [shape] [parameters] > remove < [id] [shape] removed
edit	new params	Allows to modify the parameters of the selected figure.	The selected shape is modified. You cannot change the shape type. You cannot make a valid shape an invalid.	> select 0001 < 0001 box red 2 3 10 > edit 5 1 < error: invalid argument count > edit 10 < size of box changed > edit 1000 < error: shape will go out of the board
paint	colour	Change the colour of the selected figure	The selected shape has a new colour	> select 0001 > paint red < [id][shape] red
move	coordinates	Move the selected shape to the coordinates.	The selected shape is moved and becomes a foreground. Shape can have an identity move, in that case, it just became foreground.	> select 0001 > move 10 10 < [id][shape] moved
clear	-	Remove all shapes from blackboard	All shapes are removed from the blackboard.	> clear < board is 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

Constraints

In addition to previous constraints there are a new one:

1. Command must be read from the console as a whole. In other words, you should input the command with all its arguments as a single string.
2. Now shapes can be drawn not only as a frame but also as a whole-filled figure.
3. Shapes on the board have an associated colour with them, which should be respected. Shapes must be "drawn" with the first letter of their colour as a symbol.
4. A file with an invalid board should not be loadable.

5. All objects must be stored in a single array. This was not an explicit, but an implicit requirement from the previous work. Here we just want to state it explicitly.
6. You can draw figures using real colours. In that case, there will be extra points for you. You can use external libraries or your platform API for that one.

Task milestones

1. Carefully read the table and think about the corner case of the designed system.
2. Update the system design if needed.
3. Implement figures selecting logic, consider figure level (background, foreground, middle).
4. Add figure removing functionality.
5. Implement figures filling with a colour (painting).
6. Add ability to edit figures parameters.
7. Implement moving logic of the figure.
8. Test your solution and prepare a report document.

Evaluation

Usage of inheritance and polymorphism principles including dynamic_cast usage	4
Extended functionality is fully implemented	5
All constraints are satisfied	1
Extra points: real colours in the terminal.	1
Total	10 + 1

Please note that to get the evaluation point will be reduced because of incorrect answers on the questions, wrong program output and logic, in case of Git project absence etc.

Code example (dynamic_cast)

```cpp
#include <iostream>
#include <typeinfo>

class Base {
public:
    virtual void show() {
        std::cout << "Base class show function" << std::endl;
    }
    virtual ~Base() = default; // Ensure Base class has a virtual destructor
};

class Derived : public Base {
public:
    void show() override {
        std::cout << "Derived class show function" << std::endl;
    }
    void derivedFunction() {
        std::cout << "Derived class specific function" << std::endl;
    }
};

int main() {
    Base* basePtr = new Derived(); // Base class pointer pointing to a Derived object

    // Use dynamic_cast to attempt downcasting
    Derived* derivedPtr = dynamic_cast<Derived*>(basePtr);

    if (derivedPtr) {
        // If cast succeeds, we can use Derived class specific functions
        derivedPtr->show();
        derivedPtr->derivedFunction();
    } else {
        std::cout << "dynamic_cast failed" << std::endl;
    }

    delete basePtr; // Clean up memory
    return 0;
}
```