Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2

з дисципліни
«Об’єктно-орієнтоване програмування»

Виконав:
студент групи ІМ-42
Туров Андрій Володимирович
номер у списку групи: 26

Перевірив:
Порєв В. М.

Київ 2025

Варіант 26

Варіант Ж = 26:

1. Статичний масив `*shapes[126]`.
2. Слід — суцільна лінія синього кольору.
3. Увід прямокутника — від кута до кута.
4. Прямокутник — блакитний, з чорним контуром.
5. Увід еліпса — від центру до кута прямокутника.
6. Еліпс — білий, з чорним контуром.
7. В меню має позначатись поточний об'єкт, що вводиться.

Текст програм

```
#include "canvas.h"
#include "editor.h"
#include <QActionGroup>
#include <QApplication>
#include <QMainWindow>
#include <QMenuBar>

class MainWindow : public QMainWindow {
  Q_OBJECT
private:
  Canvas *canvas = nullptr;

  PointEditor point_editor;
  LineEditor line_editor;
  RectEditor rect_editor;
  EllipseEditor ellipse_editor;

public:
  MainWindow() {
    canvas = new Canvas(this);
    setCentralWidget(canvas);

    menuBar()->addMenu("Файл");
    auto *menu = menuBar()->addMenu("&Об'єкти");
    menuBar()->addMenu("Довідка");

    auto *action_point = new QAction("&Точка", this);
    auto *action_line = new QAction("Л&інія", this);
    auto *action_rect = new QAction("П&рямокутник", this);
    auto *action_ellipse = new QAction("&Еліпс", this);
```

```
    action_point->setCheckable(true);
    action_line->setCheckable(true);
    action_rect->setCheckable(true);
    action_ellipse->setCheckable(true);

    auto *group = new QActionGroup(this);
    group->setExclusive(true);
    group->addAction(action_point);
    group->addAction(action_line);
    group->addAction(action_rect);
    group->addAction(action_ellipse);

    menu->addAction(action_point);
    menu->addAction(action_line);
    menu->addAction(action_rect);
    menu->addAction(action_ellipse);

    connect(action_point, &QAction::triggered, this,
            [this] { canvas->set_editor(&point_editor); });
    connect(action_line, &QAction::triggered, this,
            [this] { canvas->set_editor(&line_editor); });
    connect(action_rect, &QAction::triggered, this,
            [this] { canvas->set_editor(&rect_editor); });
    connect(action_ellipse, &QAction::triggered, this,
            [this] { canvas->set_editor(&ellipse_editor); });

    action_point->setChecked(true);
    canvas->set_editor(&point_editor);

    resize(1000, 700);
    setWindowTitle("Лабораторна робота №2");
  }
};

#include "main.moc"

auto main(int argc, char *argv[]) -> int {
  QApplication app(argc, argv);
  MainWindow window;
  window.show();
  return QApplication::exec();
}

// vim: sw=2 ts=2
```

Файл 1: src/main.cpp

```
#pragma once
#include "editor.h"
```

```
#include <QWidget>

class Canvas : public QWidget {
  Q_OBJECT
private:
  Editor *editor = nullptr;

protected:
  void paintEvent(QPaintEvent *event) override;
  void mousePressEvent(QMouseEvent *event) override;
  void mouseMoveEvent(QMouseEvent *event) override;
  void mouseReleaseEvent(QMouseEvent *event) override;

public:
  explicit Canvas(QWidget *parent = nullptr);
  void set_editor(Editor *editor);
};
```

Файл 2: src/canvas.h

```
#include "canvas.h"
#include "shape.h"
#include <QPaintEvent>
#include <QPainter>
#include <qcolor.h>

Canvas::Canvas(QWidget *parent) : QWidget(parent) {
  setMouseTracking(true);
  setMinimumSize(800, 600);
}

void Canvas::set_editor(Editor *editor_) {
  editor = editor_;
  if (editor != nullptr) {
    editor->set_canvas(this);
  }
}

void Canvas::paintEvent(QPaintEvent *event) {
  QPainter painter(this);
  painter.setRenderHint(QPainter::Antialiasing, true);

  painter.fillRect(rect(), QColor(255, 255, 255));

  Shape::render_all(painter);
  if (editor != nullptr) {
    editor->paint_preview(painter);
  }
}

void Canvas::mousePressEvent(QMouseEvent *event) {
```

```
  if (editor != nullptr) {
    editor->on_mouse_down(event);
  }
}

void Canvas::mouseMoveEvent(QMouseEvent *event) {
  if (editor != nullptr) {
    editor->on_mouse_move(event);
  }
}

void Canvas::mouseReleaseEvent(QMouseEvent *event) {
  if (editor != nullptr) {
    editor->on_mouse_up(event);
  }
}
```

Файл 3: src/canvas.cpp

```
#pragma once
#include <QMouseEvent>
#include <QPainter>

class Canvas;

class Editor {
protected:
  Canvas *canvas = nullptr;

public:
  virtual ~Editor() = default;
  virtual void on_mouse_down(QMouseEvent *event) = 0;
  virtual void on_mouse_move(QMouseEvent *event) = 0;
  virtual void on_mouse_up(QMouseEvent *event) = 0;
  virtual void paint_preview(QPainter &painter) = 0;
  void set_canvas(Canvas *canvas_) { canvas = canvas_; }
};

class PointEditor : public Editor {
public:
  void on_mouse_down(QMouseEvent *event) override;
  void on_mouse_move(QMouseEvent *event) override;
  void on_mouse_up(QMouseEvent *event) override;
  void paint_preview(QPainter &painter) override;

private:
  bool drawing = false;
  QPoint cursor;
};

class LineEditor : public Editor {
```

```
public:
  void on_mouse_down(QMouseEvent *event) override;
  void on_mouse_move(QMouseEvent *event) override;
  void on_mouse_up(QMouseEvent *event) override;
  void paint_preview(QPainter &painter) override;

private:
  bool drawing = false;
  QPoint start;
  QPoint end;
};

class RectEditor : public Editor {
public:
  void on_mouse_down(QMouseEvent *event) override;
  void on_mouse_move(QMouseEvent *event) override;
  void on_mouse_up(QMouseEvent *event) override;
  void paint_preview(QPainter &painter) override;

private:
  bool drawing = false;
  QPoint start;
  QPoint end;
};

class EllipseEditor : public Editor {
public:
  void on_mouse_down(QMouseEvent *event) override;
  void on_mouse_move(QMouseEvent *event) override;
  void on_mouse_up(QMouseEvent *event) override;
  void paint_preview(QPainter &painter) override;

private:
  bool drawing = false;
  QPoint center;
  QPoint corner;
};
```

Файл 4: src/editor.h

```
#include "editor.h"
#include "canvas.h"
#include "shape.h"

static inline auto to_point(QMouseEvent *event) -> QPoint {
  return event->position().toPoint();
}

void PointEditor::on_mouse_down(QMouseEvent *event) {
  if (event->button() != Qt::LeftButton) {
    return;
```

```
  }
  drawing = true;
  cursor = to_point(event);
  if (canvas != nullptr) {
    canvas->update();
  }
}

void PointEditor::on_mouse_move(QMouseEvent *event) {
  if (!drawing) {
    return;
  }
  cursor = to_point(event);
  if (canvas != nullptr) {
    canvas->update();
  }
}

void PointEditor::on_mouse_up(QMouseEvent *event) {
  if (!drawing || event->button() != Qt::LeftButton) {
    return;
  }
  drawing = false;
  QPoint point = to_point(event);
  Shape::add_shape(new PointShape(point, point));
  if (canvas != nullptr) {
    canvas->update();
  }
}

void PointEditor::paint_preview(QPainter &painter) {
  if (!drawing) {
    return;
  }
  painter.setPen(QPen(Qt::blue, 1, Qt::SolidLine));
  painter.setBrush(Qt::blue);
  const int r = 2;
  painter.drawEllipse(QRect(cursor.x() - r, cursor.y() - r, 2 *
  ↪  r, 2 * r));
}

void LineEditor::on_mouse_down(QMouseEvent *event) {
  if (event->button() != Qt::LeftButton) {
    return;
  }
  drawing = true;
  start = to_point(event);
  end = start;
  if (canvas != nullptr) {
    canvas->update();
  }
```

```
}

void LineEditor::on_mouse_move(QMouseEvent *event) {
  if (!drawing) {
    return;
  }
  end = to_point(event);
  if (canvas != nullptr) {
    canvas->update();
  }
}

void LineEditor::on_mouse_up(QMouseEvent *event) {
  if (!drawing || event->button() != Qt::LeftButton) {
    return;
  }
  drawing = false;
  end = to_point(event);
  Shape::add_shape(new LineShape(start, end));
  if (canvas != nullptr) {
    canvas->update();
  }
}

void LineEditor::paint_preview(QPainter &painter) {
  if (!drawing) {
    return;
  }
  painter.setPen(QPen(Qt::blue, 1, Qt::SolidLine));
  painter.setBrush(Qt::NoBrush);
  painter.drawLine(start, end);
}

void RectEditor::on_mouse_down(QMouseEvent *event) {
  if (event->button() != Qt::LeftButton) {
    return;
  }
  drawing = true;
  start = to_point(event);
  end = start;
  if (canvas != nullptr) {
    canvas->update();
  }
}

void RectEditor::on_mouse_move(QMouseEvent *event) {
  if (!drawing) {
    return;
  }
  end = to_point(event);
  if (canvas != nullptr) {
```

```
    canvas->update();
  }
}

void RectEditor::on_mouse_up(QMouseEvent *event) {
  if (!drawing || event->button() != Qt::LeftButton) {
    return;
  }
  drawing = false;
  end = to_point(event);
  Shape::add_shape(new RectShape(start, end));
  if (canvas != nullptr) {
    canvas->update();
  }
}

void RectEditor::paint_preview(QPainter &painter) {
  if (!drawing) {
    return;
  }
  painter.setPen(QPen(Qt::blue, 1, Qt::SolidLine));
  painter.setBrush(Qt::NoBrush);
  QRect rect = QRect(start, end).normalized();
  painter.drawRect(rect);
}

void EllipseEditor::on_mouse_down(QMouseEvent *event) {
  if (event->button() != Qt::LeftButton) {
    return;
  }
  drawing = true;
  center = to_point(event);
  corner = center;
  if (canvas != nullptr) {
    canvas->update();
  }
}

void EllipseEditor::on_mouse_move(QMouseEvent *event) {
  if (!drawing) {
    return;
  }
  corner = to_point(event);
  if (canvas != nullptr) {
    canvas->update();
  }
}

void EllipseEditor::on_mouse_up(QMouseEvent *event) {
  if (!drawing || event->button() != Qt::LeftButton) {
    return;
```

```
  }
  drawing = false;
  corner = to_point(event);
  Shape::add_shape(new EllipseShape(center, corner));
  if (canvas != nullptr) {
    canvas->update();
  }
}

void EllipseEditor::paint_preview(QPainter &painter) {
  if (!drawing) {
    return;
  }
  painter.setPen(QPen(Qt::blue, 1, Qt::SolidLine));
  painter.setBrush(Qt::NoBrush);
  int rect_x = std::abs(corner.x() - center.x());
  int rect_y = std::abs(corner.y() - center.y());
  QRect rect(center.x() - rect_x, center.y() - rect_y, 2 *
  ↪  rect_x, 2 * rect_y);
  painter.drawEllipse(rect);
}
```

Файл 5: src/editor.cpp

```
#pragma once
#include <QPainter>
#include <QPoint>
#include <QRect>
#include <array>

class Shape {
protected:
  QPoint start;
  QPoint end;

public:
  static inline std::array<Shape *, 126> shapes{};
  static inline int shapes_len = 0;

  Shape(QPoint start, QPoint end) : start(start), end(end) {}
  virtual ~Shape() = default;
  virtual void show(QPainter &painter) = 0;

  static void add_shape(Shape *shape) {
    if (shapes_len < 126) {
      shapes[shapes_len++] = shape;
    }
  }

  static void render_all(QPainter &painter) {
    for (int i = 0; i < shapes_len; ++i) {
```

```
      if (shapes[i] != nullptr) {
        shapes[i]->show(painter);
      }
    }
  }
};

class PointShape : public Shape {
public:
  using Shape::Shape;
  void show(QPainter &painter) override;
};

class LineShape : public Shape {
public:
  using Shape::Shape;
  void show(QPainter &painter) override;
};

class RectShape : public Shape {
public:
  using Shape::Shape;
  void show(QPainter &painter) override;
};

class EllipseShape : public Shape {
public:
  using Shape::Shape;
  void show(QPainter &painter) override;
};
```

Файл 6: src/shape.h

```
#include "shape.h"

void PointShape::show(QPainter &painter) {
  QPen pen(Qt::black);
  painter.setPen(pen);
  painter.setBrush(Qt::black);
  const int r = 2;
  painter.drawEllipse(QRect(start.x() - r, start.y() - r, 2 * r,
   ↪  2 * r));
}

void LineShape::show(QPainter &painter) {
  QPen pen(Qt::black);
  painter.setPen(pen);
  painter.setBrush(Qt::NoBrush);
  painter.drawLine(start, end);
}
```

```cpp
void RectShape::show(QPainter &painter) {
  QPen pen(Qt::black);
  painter.setPen(pen);
  painter.setBrush(QColor(173, 216, 230));
  QRect rect = QRect(start, end).normalized();
  painter.drawRect(rect);
}

void EllipseShape::show(QPainter &painter) {
  QPen pen(Qt::black);
  painter.setPen(pen);
  painter.setBrush(Qt::white);
  int rect_x = std::abs(end.x() - start.x());
  int rect_y = std::abs(end.y() - start.y());
  QRect rect(start.x() - rect_x, start.y() - rect_y, 2 * rect_x,
  ↪  2 * rect_y);
  painter.drawEllipse(rect);
}
```

Файл 7: src/shape.cpp

```cmake
cmake_minimum_required(VERSION 3.16)
project(lab2 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Core Widgets)

qt_standard_project_setup()
add_executable(lab2
    src/main.cpp
    src/shape.h
    src/shape.cpp
    src/editor.h
    src/editor.cpp
    src/canvas.h
    src/canvas.cpp
)

target_link_libraries(lab2 PRIVATE Qt6::Core Qt6::Widgets)

add_custom_target(run
  COMMAND $<TARGET_FILE:lab2>
  DEPENDS lab2
  WORKING_DIRECTORY $<TARGET_FILE_DIR:lab2>
)
```

Файл 8: CMakeLists.txt

Зображення

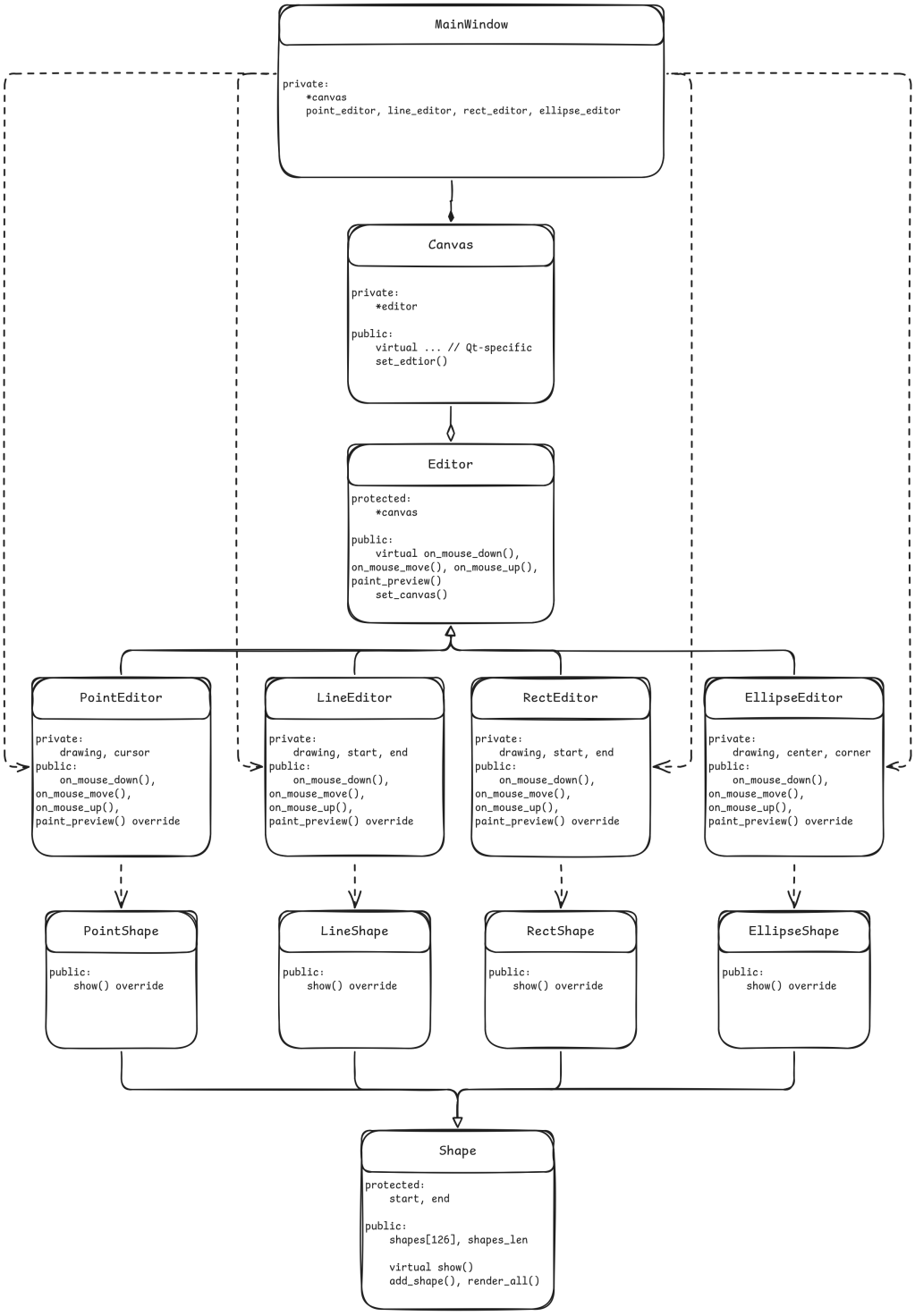

Рис. 1: Класова діаграма

Рис. 2: «Гумовий» слід

Рис. 3: Увід еліпса від центра до кута

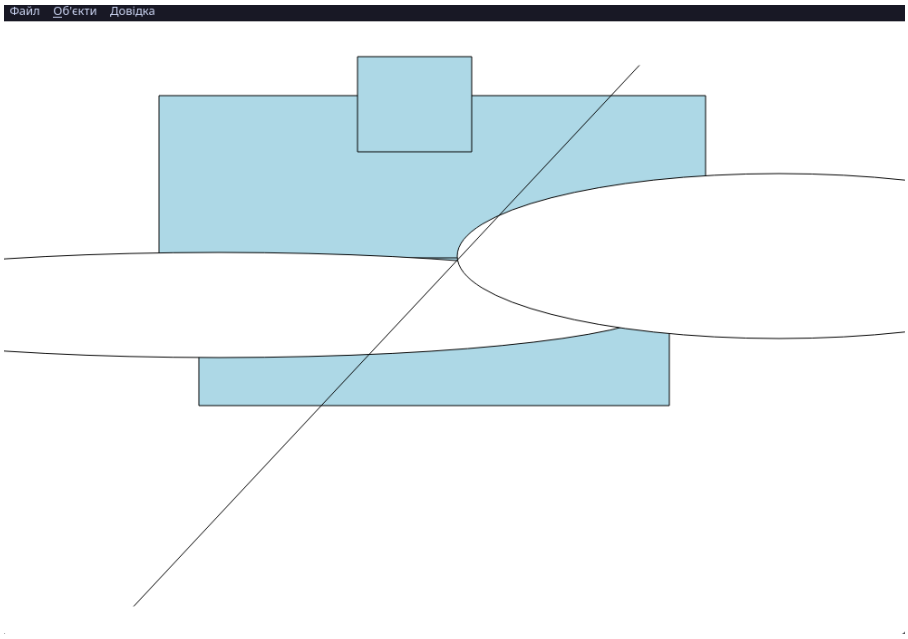

Рис. 4: Накладання різних фігур

Висновок

Реалізував застосунок для малювання геометричних примітивів. Створив ієрархію класів, де існує кілька конкретних реалізацій для різних фігур, і відповідний метод відображення викликається поліморфно. Зобразив діаграму класів.