**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Ефимов Александр

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата: 23.12.2019

Оценка:

Москва, 2019

**Постановка задачи**

Спроектировать простейший графический векторный редактор.

Требование к функционалу редактора:

* создание нового документа
* импорт документа из файла
* экспорт документа в файл
* создание графического примитива (согласно варианту задания)
* удаление графического примитива
* отображение документа на экране (печать перечня графических объектов и их характеристик)
* реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

* Создание графических примитивов необходимо вынести в отдельный класс – Factory.
* Сделать упор на использовании полиморфизма при работе с фигурами;
* Взаимодействие с пользователем (ввод команд) реализовать в функции main;

Вариант 1: *Треугольник, прямоугольник, квадрат*.

1. **Описание программы**

Шаблон квадрата написан в *Shape.h*, вместе шаблонными функциями *Area*, *Centre* и *Print*, которые ищут его площадь, центр фигуры и печатают его соответственно.

В *main.cpp* содержится меню, позволяющее работать с вектором, содержащим в себе общие указатели на абстрактный класс *Square*, тип точек которого **int**. Меню самостоятельно выполняет команды, которые не влияют на вектор (т.е. печать содержимого, печать содержимое).

Действия, такие как добавить в вектор, удалить из него, передаются в объект класса *CommandManager*, вместе с аргументами. Передаются они путем создания одного из унаследованных подклассов класса *Command*, который, кроме того, как выполняет действие, записывает данные, необходимые для его выполнения. Это позволяет отменить, а в будущем даже восстановить данные.

Класс *Factory* отделяет создание объектов класса Shape, вызывая их конструкторы.

1. **Набор тестов**

* **Test\_01.txt**

Использование функций сохранения и загрузки

* **Test\_02.txt**

Использование вставки и удаления, undo и redo, печати фигур

1. **Результаты выполнения тестов**

Все тесты возвращают ожидаемые результаты

**test\_01**

|  |
| --- |
| 1. Add shape  2. Print all elements  3. Delete element  4. Undo  5. Redo  6. Print this menu  7. Reload document  8. Load document  9. Save document  0. Exit  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 1  Input point coordinates  A: 4 9  B: -2 4  C: -1 -2  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 2  Input point coordinates clockwise or counter clockwise  A: -1 -3  B: 4 2  C: -8 14  D: -13 9  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 3  Input point coordinates clockwise or counter clockwise  A: -6 10  B: -5 6  C: -9 5  D: -10 9  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  3. Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  -----------------------  Menu choice: 9  Save path: savefile.sav  -----------------------  Menu choice: 7  Reload completed.  -----------------------  Menu choice: 2  No shapes inserted  -----------------------  Menu choice: 8  Any unsaved changes will be lost. Continue? (y/n): y  Load path: savefile.sav  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  3. Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  -----------------------  Menu choice: |

Успешное сохранение и загрузка документа

**test\_02**

|  |
| --- |
| 1. Add shape  2. Print all elements  3. Delete element  4. Undo  5. Redo  6. Print this menu  7. Reload document  8. Load document  9. Save document  0. Exit  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 1  Input point coordinates  A: 4 9  B: -2 4  C: -1 -2  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 2  Input point coordinates clockwise or counter clockwise  A: -1 -3  B: 4 2  C: -8 14  D: -13 9  -----------------------  Menu choice: 1  1. Triangle  2. Rectangle  3. Square  Any other number returns to menu  Pick the type of shape: 3  Input point coordinates clockwise or counter clockwise  A: -6 10  B: -5 6  C: -9 5  D: -10 9  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  3. Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  -----------------------  Menu choice: 3  Number of the shape: 3  -----------------------  Menu choice: 3  Number of the shape: 1  -----------------------  Menu choice: 2  1. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  -----------------------  Menu choice: 4  Commander: Undid Delete  -----------------------  Menu choice: 4  Commander: Undid Delete  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  3. Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  -----------------------  Menu choice: 4  Commander: Undid Add  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  -----------------------  Menu choice: 5  Commander: Redid Add  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  3. Square: (-6 , 10); (-5 , 6); (-9 , 5); (-10 , 9); Centre = (-7.5 , 7.5); Area = 17  -----------------------  Menu choice: 5  Commander: Redid Delete  -----------------------  Menu choice: 2  1. Triangle: (4 , 9); (-2 , 4); (-1 , -2); Centre = (0.333333 , 3.66667); Area = 20.5  2. Rectangle: (-1 , -3); (4 , 2); (-8 , 14); (-13 , 9); Centre = (-4.5 , 5.5); Area = 120  -----------------------  Menu choice: |

Функции *undo* и *redo* успешно отменяют и выполняют заново удаление и вставку

1. **Листинг программы**

**Shape.h**

|  |
| --- |
| #ifndef SHAPE\_H  #define SHAPE\_H  #include <iostream>  #include <utility>  #include <cmath>  #include <vector>  typedef std::pair<double, double> DoublePoint;  //---------------------------------------------------------------------  // Necessary for friend declaration  //---------------------------------------------------------------------  template <typename T>  class Shape;  template <typename T>  std::ostream& operator << (std::ostream& os, const Shape<T>& shape);  //---------------------------------------------------------------------  template <typename T>  class Shape {  public:  friend std::ostream& operator << <T> (std::ostream& os, const Shape& shape);  typedef std::pair<T, T> Point;    DoublePoint getCentre() const  { return centre; }  double getArea() const  { return area; }  bool getAbstract() const  { return abstract; }  int getType() const  { return type; }  std::vector<Point> getPoints() const  { return p; }  protected:  DoublePoint centre;  double area = 0;  bool abstract = false;  int type;  std::vector<Point> p;  private:  virtual void Centre() = 0;  virtual void Area() = 0;  virtual std::ostream& print(std::ostream&) const = 0;  };  template <typename T>  class Triangle : public Shape<T> {  public:  using typename Shape<T>::Point;  using Shape<T>::getCentre;  using Shape<T>::getArea;  using Shape<T>::centre;  using Shape<T>::area;  using Shape<T>::abstract;  using Shape<T>::type;  using Shape<T>::p;  Triangle(Point a, Point b, Point c);  Triangle(std::istream& is);  void Centre() override;  void Area() override;  std::ostream& print(std::ostream&) const override;  };  template <typename T>  class Rectangle : public Shape<T> {  public:  using typename Shape<T>::Point;  using Shape<T>::getCentre;  using Shape<T>::getArea;  using Shape<T>::centre;  using Shape<T>::area;  using Shape<T>::abstract;  using Shape<T>::type;  using Shape<T>::p;  Rectangle(Point a, Point b, Point c, Point d);  Rectangle(std::istream& is);  void Centre() override;  void Area() override;  std::ostream& print(std::ostream&) const override;  };  template <typename T>  class Square : public Shape<T> {  public:  using typename Shape<T>::Point;  using Shape<T>::getCentre;  using Shape<T>::getArea;  using Shape<T>::centre;  using Shape<T>::area;  using Shape<T>::abstract;  using Shape<T>::type;  using Shape<T>::p;  Square(Point a, Point b, Point c, Point d);  Square(std::istream& is);  void Centre() override;  void Area() override;  std::ostream& print(std::ostream&) const override;  };  //-----------------------------------------------------------------------------------------------------------  // Geometrical correctness  //-----------------------------------------------------------------------------------------------------------  bool DoubleEqual(double lhs, double rhs)  {  const double EPS = 0.00001;  if (lhs > rhs)  return (lhs - rhs) < EPS ? true : false;  else  return (rhs - lhs) < EPS ? true : false;  }  bool IsRight(std::pair<double, double> a, std::pair<double, double> b, std::pair<double, double> c)  {  std::pair<double, double> vec1 = { b.first - a.first, b.second - a.second };  std::pair<double, double> vec2 = { c.first - a.first, c.second - a.second };  double result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (DoubleEqual(result, 0)) return true;  return false;  }  bool IsRight(std::pair<int, int> a, std::pair<int, int> b, std::pair<int, int> c)  {  std::pair<int, int> vec1 = { b.first - a.first, b.second - a.second };  std::pair<int, int> vec2 = { c.first - a.first, c.second - a.second };  int result = vec1.first \* vec2.first + vec1.second \* vec2.second;  if (result == 0) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  // Check if shape is a rectangle  //-----------------------------------------------------------------------------------------------------------  template <class T>  bool IsTriangle(std::vector< std::pair<T, T> > &p)  {  return !DoubleEqual  ( ((p[1].first - p[0].first) \* (p[2].second - p[0].second) - (p[1].second - p[0].second) \* (p[2].first - p[0].first)), 0 );  }  //-----------------------------------------------------------------------------------------------------------  // Check if shape is a rectangle  //-----------------------------------------------------------------------------------------------------------  template <class T>  bool IsRectangle(const std::vector< std::pair<T, T> > &p)  {  std::pair<T, T> null(0, 0);  if (p[0] == null && p[1] == null && p[2] == null && p[3] == null) return false;  if (  IsRight(p[2], p[1], p[3]) &&  IsRight(p[3], p[2], p[0]) &&  IsRight(p[1], p[0], p[2]) &&  IsRight(p[0], p[3], p[1])  ) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  // Check if shape is a square  //-----------------------------------------------------------------------------------------------------------  template <class T>  double distance(const std::pair<T, T> a, std::pair<T, T> b)  {  return (b.first - a.first) \* (b.first - a.first) + (b.second - a.second) \* (b.second - a.second);  }  template <class T>  bool IsSquare(const std::vector< std::pair<T, T> > &p)  {  if (  IsRectangle(p) &&  DoubleEqual(distance(p[0], p[1]), distance(p[1], p[2])) &&  DoubleEqual(distance(p[1], p[2]), distance(p[2], p[3])) &&  DoubleEqual(distance(p[2], p[3]), distance(p[3], p[0])) &&  DoubleEqual(distance(p[3], p[0]), distance(p[0], p[1]))  ) return true;  return false;  }  //-----------------------------------------------------------------------------------------------------------  // Polymorphism safe out operator overload  //-----------------------------------------------------------------------------------------------------------  template <class T>  std::ostream& operator << (std::ostream& os, const Shape<T>& shape)  {  shape.print(os);  return os;  }  template <class T>  std::ostream& operator << (std::ostream& os, const std::pair<T,T>& p)  {  os << '(' << p.first << " , " << p.second << ')';  return os;  }  //-----------------------------------------------------------------------------------------------------------  // Triangle functions  //-----------------------------------------------------------------------------------------------------------  template <class T>  Triangle<T>::Triangle(Point a, Point b, Point c)  {  p.push\_back(a);  p.push\_back(b);  p.push\_back(c);  if (!IsTriangle(p)) { abstract = true; }  Centre();  Area();  type = 0;  }  template <class T>  Triangle<T>::Triangle(std::istream& is)  {  Point points[3];  std::cout << "Input point coordinates\nA: ";  is >> points[0].first >> points[0].second;  std::cout << "B: ";  is >> points[1].first >> points[1].second;  std::cout << "C: ";  is >> points[2].first >> points[2].second;  p.push\_back(points[0]);  p.push\_back(points[1]);  p.push\_back(points[2]);  if (!IsTriangle(p)) { abstract = true; }  Centre();  Area();  }  template <class T>  void Triangle<T>::Centre()  {  for (int i = 0; i < 3; ++i)  {  centre.first += p[i].first;  centre.second += p[i].second;  }  centre.first /= 3;  centre.second /= 3;  }  template <class T>  void Triangle<T>::Area()  {  area = (double) (  p[0].first \* (p[1].second - p[2].second)  + p[1].first \* (p[2].second - p[0].second)  + p[2].first \* (p[0].second - p[1].second)  ) / 2;  if (area < 0) area = -area;  }  template <class T>  std::ostream& Triangle<T>::print(std::ostream& os) const  {  if (abstract)  {  os << "Abstract: ";  }  else  {  os << "Triangle: ";  }  os << p[0] << "; " << p[1] << "; " << p[2]  << "; Centre = " << getCentre()  << "; Area = " << getArea();  return os;  }  //-----------------------------------------------------------------------------------------------------------  // Rectangle functions  //-----------------------------------------------------------------------------------------------------------  template <class T>  Rectangle<T>::Rectangle(Point a, Point b, Point c, Point d)  {  p.push\_back(a);  p.push\_back(b);  p.push\_back(c);  p.push\_back(d);  if (!IsRectangle(p)) { abstract = true; }  Centre();  Area();  type = 1;  }  template <class T>  Rectangle<T>::Rectangle(std::istream& is)  {  Point points[4];  std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";  is >> points[0].first >> points[0].second;  std::cout << "B: ";  is >> points[1].first >> points[1].second;  std::cout << "C: ";  is >> points[2].first >> points[2].second;  std::cout << "D: ";  is >> points[3].first >> points[3].second;  p.push\_back(points[0]);  p.push\_back(points[1]);  p.push\_back(points[2]);  p.push\_back(points[3]);  if (!IsRectangle(p)) { abstract = true; }    Centre();  Area();  type = 1;  }  template <class T>  void Rectangle<T>::Centre()  {  for (int i = 0; i < 4; ++i)  {  centre.first += p[i].first;  centre.second += p[i].second;  }  centre.first /= 4;  centre.second /= 4;  }  template <class T>  void Rectangle<T>::Area()  {  area = sqrt( distance(p[0], p[1]) \* distance(p[1], p[2]) );  }  template <class T>  std::ostream& Rectangle<T>::print(std::ostream& os) const  {  if (abstract)  {  os << "Abstract: ";  }  else  {  os << "Rectangle: ";  }    os << p[0] << "; " << p[1] << "; " << p[2] << "; " << p[3]  << "; Centre = " << getCentre()  << "; Area = " << getArea();  return os;  }  //-----------------------------------------------------------------------------------------------------------  // Square functions  //-----------------------------------------------------------------------------------------------------------  template <class T>  Square<T>::Square(Point a, Point b, Point c, Point d)  {  p.push\_back(a);  p.push\_back(b);  p.push\_back(c);  p.push\_back(d);  if (!IsSquare(p)) { abstract = true; }  Centre();  Area();  type = 2;  }  template <class T>  Square<T>::Square(std::istream& is)  {  Point points[4];  std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";  is >> points[0].first >> points[0].second;  std::cout << "B: ";  is >> points[1].first >> points[1].second;  std::cout << "C: ";  is >> points[2].first >> points[2].second;  std::cout << "D: ";  is >> points[3].first >> points[3].second;  p.push\_back(points[0]);  p.push\_back(points[1]);  p.push\_back(points[2]);  p.push\_back(points[3]);  if (!IsSquare(p)) { abstract = true; }  Centre();  Area();  type = 2;  }  template <class T>  void Square<T>::Centre()  {  for (int i = 0; i < 4; ++i)  {  centre.first += p[i].first;  centre.second += p[i].second;  }  centre.first /= 4;  centre.second /= 4;  }  template <class T>  void Square<T>::Area()  {  area = sqrt( distance(p[0], p[1]) \* distance(p[1], p[2]) );  }  template <class T>  std::ostream& Square<T>::print(std::ostream& os) const  {  if (abstract)  {  os << "Abstract: ";  }  else  {  os << "Square: ";  }  os << p[0] << "; " << p[1] << "; " << p[2] << "; " << p[3]  << "; Centre = " << getCentre()  << "; Area = " << getArea();  return os;  }  //-----------------------------------------------------------------------------------------------------------  // Compare points  //-----------------------------------------------------------------------------------------------------------  bool operator == (std::pair<int, int> lhs, std::pair<int, int> rhs)  {  return lhs.first == rhs.first && lhs.second == rhs.second;  }  bool operator == (std::pair<double, double> lhs, std::pair<double, double> rhs)  {  return DoubleEqual(lhs.first, rhs.first) && DoubleEqual(lhs.second, rhs.second);  }  #endif |

**Factory.h**

|  |
| --- |
| #ifndef FACTORY\_H  #define FACTORY\_H  #include <iostream>  #include "Shape.h"  template <typename T>  class Factory {  private:  enum Type { triangle, rectangle, square };  public:  static std::shared\_ptr<Shape<T>> create(int num, std::istream &is)  {  switch (num)  {  case triangle:  return std::make\_shared< Triangle<T> >(is);  break;    case rectangle:  return std::make\_shared< Rectangle<T> >(is);  break;    case square:  return std::make\_shared< Square<T> >(is);  break;  default:  throw std::logic\_error("No such number");  break;  }  }  static std::shared\_ptr<Shape<T>> create(int num, std::pair<T, T> (&p)[4])  {  switch (num)  {  case triangle:  return std::make\_shared< Triangle<T> >(p[0], p[1], p[2]);  break;    case rectangle:  return std::make\_shared< Rectangle<T> >(p[0], p[1], p[2], p[3]);  break;    case square:  return std::make\_shared< Square<T> >(p[0], p[1], p[2], p[3]);  break;  default:  throw std::logic\_error("No such number");  break;  }  }  };  #endif |

**Command.h**

|  |
| --- |
| #ifndef COMMAND\_H  #define COMMAND\_H  #include <fstream>  #include <vector>  #include <deque>  #include <memory>  #include <string>  #include "Shape.h"  #include "Factory.h"  class commander\_error : public std::logic\_error {  public:  commander\_error(std::string const& msg) : std::logic\_error(msg)  {}  };  template <class T>  class Command;  template <class T>  class CommandManager{  public:  CommandManager(std::vector< std::shared\_ptr<Shape<T>> >& vec) : shapes(vec) { }  void add(std::shared\_ptr< Command<T> > comm)  {  comm->execute();  listOfComm.erase(listOfComm.begin() + currentCommand, listOfComm.end());  listOfComm.push\_back(comm);  ++currentCommand;  }  void undo()  {  if (currentCommand == 0) throw commander\_error("No operations to undo");  --currentCommand;  listOfComm[currentCommand]->undo();  std::cout << "Commander: Undid " << listOfComm[currentCommand]->getCommName() << "\n";  }  void redo()  {  if (currentCommand == listOfComm.size()) throw commander\_error("No operations to redo");  listOfComm[currentCommand]->execute();  std::cout << "Commander: Redid " << listOfComm[currentCommand]->getCommName() << "\n";  ++currentCommand;  }  void reload()  {  listOfComm.clear();  shapes.clear();  currentCommand = 0;  }  void save()  {  std::string filePath;  std::ofstream outfile;  std::cout << "Save path: ";  std::cin >> filePath;    outfile.open(filePath, std::ios::out);  if (outfile.fail())  {  throw std::runtime\_error("File open operation failed");  }  std::vector< std::pair<T,T> > points;  outfile << shapes.size() << '\t';  for(std::shared\_ptr<Shape<T>> e : shapes)  {  points = e->getPoints();  outfile << e->getType() << '\t';  switch(e->getType())  {  case 0:  for (int i = 0; i < 3; ++i)  {  outfile << points[i].first << '\t' << points[i].second << '\t';  }  break;    case 1:  case 2:  for (int i = 0; i < 4; ++i)  {  outfile << points[i].first << '\t' << points[i].second << '\t';  }  break;  }  }  outfile.close();  }    void load()  {  std::string filePath;  std::ifstream infile;  char ch;  std::cout << "Any unsaved changes will be lost. Continue? (y/n): ";  do {  std::cin >> ch;  if (ch == 'y') { ; }  else if (ch == 'n') { return; }  else  {  std::cout << "(y/n): ";  }  } while (ch != 'y');  std::cout << "Load path: ";  std::cin >> filePath;    infile.open(filePath, std::ios::in);  if (infile.fail())  {  throw std::runtime\_error("File open operation failed");  }  reload();  int size, type;  std::pair<T, T> p[4];  infile >> size;  for (int i = 0; i < size; ++i)  {  infile >> type;  switch (type)  {  case 0:  for (int j = 0; j < 3; ++j)  {  infile >> p[j].first >> p[j].second;  }  break;  case 1:  case 2:  for (int j = 0; j < 4; ++j)  {  infile >> p[j].first >> p[j].second;  }  break;  }  shapes.push\_back(Factory<T>::create(type, p));  }  }  private:  int currentCommand = 0;  std::vector< std::shared\_ptr<Shape<T>> >& shapes;  std::deque< std::shared\_ptr<Command<T>> > listOfComm;  };  template <class T>  class Command{  public:  virtual void execute() = 0;  virtual void undo() = 0;  std::string getCommName() const  { return commName; }  protected:  typedef std::pair<T, T> Point;  std::string commName = "Undefined";  };  template <class T>  class Add : public Command<T> {  public:  using typename Command<T>::Point;  using Command<T>::commName;  Add(std::vector< std::shared\_ptr<Shape<T>> >& vec, int type, Point (&points)[4]) :  shapes(vec), shapeType(type), p(points) { commName = "Add"; }  void execute() override  {  shapes.push\_back(Factory<T>::create(shapeType, p));  }  void undo() override  {  shapes.pop\_back();  }  private:  int shapeType;  Point p[4];  std::vector< std::shared\_ptr<Shape<T>> >& shapes;  };  template <class T>  class Delete : public Command<T> {  public:  using typename Command<T>::Point;  using Command<T>::commName;  Delete(std::vector< std::shared\_ptr<Shape<T>> >& vec, int num) : shapes(vec), pos(num)  {  commName = "Delete";  std::vector< std::pair<T, T> > points;  points = vec[num]->getPoints();  shapeType = vec[num]->getType();  switch(shapeType)  {  case 0:  for (int i = 0; i < 3; ++i)  {  p[i] = points[i];  }  break;  case 1:  case 2:  for (int i = 0; i < 4; ++i)  {  p[i] = points[i];  }  break;  }  }  void execute()  {  shapes.erase(shapes.begin() + pos);  }  void undo()  {  shapes.insert(shapes.begin() + pos, Factory<T>::create(shapeType, p));  }  private:  int pos;  int shapeType;  Point p[4];  std::vector< std::shared\_ptr<Shape<T>> >& shapes;  };  #endif |

**main.cpp**

|  |
| --- |
| #include <iostream>  #include <string>  #include <exception>  #include <vector>  #include <memory>  #include "Shape.h"  #include "Command.h"  using namespace std;  using WorkingType = int;  void getMenu()  {  cout  << "\n1. Add shape\n"  << "2. Print all elements\n"  << "3. Delete element\n"  << "4. Undo\n"  << "5. Redo\n"  << "6. Print this menu\n"  << "7. Reload document\n"  << "8. Load document\n"  << "9. Save document\n"  << "0. Exit" << endl;  }  int GetNum(void)  {  int k;  bool got = false;  while (!got)  {  cin >> k;  if (!cin.good())  {  cout << "Bad input ignored: " << flush;  cin.clear();  cin.ignore(256, '\n');  }  else got = true;  }  return k;  }  int main()  {  std::vector< std::shared\_ptr<Shape<WorkingType>> > shapes;  CommandManager<WorkingType> Commander(shapes);  std::pair<WorkingType, WorkingType> points[4];  int figNum = 1;  int k = 1;  getMenu();  do  {  cout << "-----------------------\nMenu choice: ";  try  {  k = GetNum();  if (!cin) throw runtime\_error("Failed to get input");  switch (k)  {  case 1:  cout  << "1. Triangle\n"  << "2. Rectangle\n"  << "3. Square\n"  << "Any other number returns to menu\n"  << "Pick the type of shape: ";  k = GetNum();  if (k >= 1 && k <= 3)  {  --k;  switch (k)  {  case 0:  std::cout << "Input point coordinates\nA: ";  cin >> points[0].first >> points[0].second;  std::cout << "B: ";  cin >> points[1].first >> points[1].second;  std::cout << "C: ";  cin >> points[2].first >> points[2].second;  break;    case 1:  // Intentionally empty  case 2:  std::cout << "Input point coordinates clockwise or counter clockwise\nA: ";  std::cin >> points[0].first >> points[0].second;  std::cout << "B: ";  std::cin >> points[1].first >> points[1].second;  std::cout << "C: ";  std::cin >> points[2].first >> points[2].second;  std::cout << "D: ";  std::cin >> points[3].first >> points[3].second;  break;    default:  break;  }  Commander.add(make\_shared< Add<WorkingType> >(shapes, k, points));  }  k = 1;  break;  case 2:  if (shapes.empty())  {  cout << "No shapes inserted" << endl;  }  else  {  for (std::shared\_ptr< Shape<WorkingType> > e : shapes)  {  std::cout << figNum++ << ". " << \*e << "\n";  }  figNum = 1;  }  break;  case 3:  if (shapes.empty()) throw runtime\_error("No shapes to delete");  cout << "Number of the shape: ";  k = GetNum();  if (k < 1 || k > shapes.size()) throw runtime\_error("Incorrect index");  Commander.add(make\_shared< Delete<WorkingType> >(shapes, --k));  k = 5;  break;  case 4:  Commander.undo();  break;  case 5:  Commander.redo();  break;  case 6:  getMenu();  break;  case 7:  Commander.reload();  std::cout << "Reload completed." << std::endl;  break;  case 8:  Commander.load();  break;  case 9:  Commander.save();  break;  case 0:  break;  default:  cout << "No such number" << endl;  }  }  catch (runtime\_error& err)  {  cerr << err.what() << endl;  }  catch (commander\_error& err)  {  cerr << "Commander: " << err.what() << endl;  }  catch (logic\_error& err)  {  cerr << err.what() << endl;  }  catch (...)  {  cerr << "Exception thrown, but undefined." << endl;  }  } while (k);  return 0;  } |

1. **Выводы**

Использование шаблонов, которые описывают методы имплементации, такие, как шаблоны **Command** и **Factory**, значительно упрощают разработку, помогая отделять более часто используемые виды кода на независимые друг от друга части.

1. **Список литературы**

* “The Gang of Four”, Design Patterns, Addison-Wesley, 1994