**Міністерство освіти і науки України**

**Національний технічний університет України**

**“Київський політехнічний інститут”**

ІПСА

Кафедра Системного проектування

Лабораторна робота №1-2 з «Об’єктно-орієнтованого програмування»

на тему:

«Інкапсуляція, агрегування, перевантаження операторів, наслідування, поліморфізм»

Виконав:

Студент групи ДА-21

Круш І. В.

Варіант **№12**

Київ 2013

**Короткий опис**

Базу програми складують базовий клас Figure, похідні від нього Rectangle, Zigzag I Parallelogram, а також клас для роботи з ними FigureStorage, яке по суті є API. Для роботи з консоллю був написаний клас FiguresConsole.

Була також організована робота з групами, яка може містити в собі фігури різних типів. Ми маємо можливість робити щось одночасно з усіма фігурами групи, наприклад, змінювати колір границь.

**Консольні команди**

* **add figure rectangle** - додає фігуру прямокутник в сховище, заноситься також в пам'ять як поточна фігура
* **add figure parallelogram** - додає фігуру паралелограм в сховище, заноситься також в пам'ять як поточна фігура
* **add figure zigzag** - додає фігуру зигзаг в сховище, заноситься також в пам'ять як поточна фігура
* **add figure group** - додає фігуру паралелограм в поточну групу(якщо групи немаэ - вона створиться)
* **add group** - додає групу в сховище
* **get figure info** - виводить інформацію про поточну фігуру, якщо вона обрана
* **get figure count** - виводить кількість фігур в сховищі
* **get figure id** - знаходить фігуру по введеному ідентифікатору(ідентифікатор вводиться в діалозі)
* **get group id** - виводить ідентифікатор поточної групи
* **get group figures** - виводить ідентифікатори усіх фігур з групи
* **get group info** - виводить інформацію про всіх фігур з групи
* **do figure topLeft** - для зміни верньої лівої точки поточної фігури
* **do figure bottomRight** - для зміни нижньої правої точки поточної фігури
* **do figure border** - для зміни ліній поточної фігури
* **do figure background** - для зміни фону поточної фігури(неактуально для зигзагу)
* **do figure controlPoint** - для зміни контрольної точки паралелограма
* **do figure points resize** - для зміни кількості точок зигзагу
* **do figure points number** - для зміни потрібної точки зигзагу
* **do figure remove** - видаляє поточну фігуру
* **do group border** - змінює ліні всіх фігур групи
* **do group background** - змінює фон всіх фігур групи
* **do group remove** - видаляє поточну групу
* **exit** - вихід з програми

**Лістинг програми**

Figure.cpp

#ifndef Figure\_h\_

#define Figure\_h\_

#include "colorRGB.h"

#include "Border.h"

#include "Point.h"

/\*\*

\* class Figure

\* abstract class for all geometric figures

\*/

class Figure {

protected :

Point \_topLeft, \_bottomRight;

bool \_insurality;

Border \_border;

colorRGB \_backgroundRGB;

int \_id;

int \_type;

public :

static int lastId;

/\*\*

\* getTopLeft

\* @return double

\*/

Point getTopLeft() {

return this->\_topLeft;

}

/\*\*

\* setTopLeft

\* @param double coordinate - coordinate of this point

\* @return void

\*/

void setTopLeft(Point point) {

this->\_topLeft = point;

}

/\*\*

\* getBottomRight

\* @return double

\*/

Point getBottomRight() {

return this->\_bottomRight;

}

/\*\*

\* setBottomRight

\* @param double coordinate - coordinate of this point

\* @return void

\*/

void setBottomRight(Point point) {

this->\_bottomRight = point;

}

/\*\*

\* getPerimeter

\* @return double

\*/

virtual double getPerimeter() {}

/\*\*

\* getArea

\* @return double

\*/

virtual double getArea() {}

/\*\*

\* setBorder

\* @param width - width of border

\* @param type - type of border

\* @param RGB - array of RGB color

\*/

void setBorder(double width, string type, colorRGB RGB) {

this->\_border.width = width;

this->\_border.type = type;

this->\_border.RGB = RGB;

}

void setBorder(Border border) {

this->\_border = border;

}

Border getBorder() {

return this->\_border;

}

/\*\*

\* setBackground

\* @param RGB - color of background in rgb

\*/

void setBackground(colorRGB RGB) {

this->\_backgroundRGB = RGB;

}

colorRGB getBackground() {

return this->\_backgroundRGB;

}

virtual void horizontalReflect(){}

virtual void verticalReflect(){}

/\*\*

\* getId

\* @return int

\*/

int getId() {

return this->\_id;

}

bool inLimitingRectangle(Point point) {

return point.x >= this->\_topLeft.x && point.x <= this->\_bottomRight.x && point.y >= this->\_topLeft.y && point.y <= this->\_bottomRight.y;

}

int getType() {

return this->\_type;

}

/\*\*

\* Figure

\* default constructor

\*/

Figure() {

colorRGB borderRGB, backRGB;

borderRGB.r = 255;

borderRGB.g = 255;

borderRGB.b = 255;

backRGB.r = 255;

backRGB.g = 0;

backRGB.b = 0;

this->setBorder(1.0, "solid", borderRGB);

this->setBackground(backRGB);

this->\_id = ++Figure::lastId;

}

};

int Figure::lastId = 0;

#endif

**FiguresStorage.h**

#ifndef FiguresStorage\_h\_

#define FiguresStorage\_h\_

#include "colorRGB.h"

#include "Border.h"

#include "Point.h"

#include "Figure.h"

#include "Rectangle.h"

#include "Parallelogram.h"

#include "Zigzag.h"

#include "FiguresGroup.h"

/\*\*

\* class FiguresStorage

\*/

class FiguresStorage {

private :

vector<Figure\*> \_figures;

vector<FiguresGroup\*> \_groups;

int \_findNumber(int id) {

int requiredId = -1;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->getId() == id) {

requiredId = i;

break;

}

}

return requiredId;

}

int \_findGroupNumber(int id) {

int result = -1;

for(int i = 0; i < this->\_groups.size(); i++) {

if(id == this->\_groups[i]->getId()) {

result = i;

}

}

return result;

}

public :

/\*\*

\* addRectangle

\* @param Point topLeft

\* @param Point bottomRight

\* @param Point controlPoint

\* @param Border border

\* @param vector<int> background

\* @return void

\*/

Rectangle\* addRectangle(Point topLeft, Point bottomRight, Border border, colorRGB background) {

Rectangle \* newRect = new Rectangle(topLeft, bottomRight, border, background);

this->\_figures.push\_back(newRect);

return newRect;

}

Rectangle\* addRectangle(Point topLeft, Point bottomRight) {

Rectangle \* newRect = new Rectangle(topLeft, bottomRight);

this->\_figures.push\_back(newRect);

return newRect;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight, double controlPoint, Border border, colorRGB background) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight, controlPoint, border, background);

this->\_figures.push\_back(newPar);

return newPar;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight, double controlPoint) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight, controlPoint);

this->\_figures.push\_back(newPar);

return newPar;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight);

this->\_figures.push\_back(newPar);

return newPar;

}

Zigzag\* addZigzag(int stepsAmount, vector<Point> points, Border border) {

Zigzag \* newZig = new Zigzag(stepsAmount, points, border);

this->\_figures.push\_back(newZig);

return newZig;

}

/\*\*

\* getAmount

\* return int

\*/

int getAmount() {

return this->\_figures.size();

}

/\*\*

\* find

\* @param int id

\* @return Figure

\*/

Figure\* find(int id) {

Figure\* result = NULL;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->getId() == id) {

result = this->\_figures[i];

break;

}

}

return result;

}

/\*\*

\* find

\* @param Point

\* @return vector<Figure\*>

\*/

vector<Figure\*> find(Point point) {

vector<Figure\*> result;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->inLimitingRectangle(point)) {

result.push\_back(this->\_figures[i]);

}

}

return result;

}

void remove(int id) {

Figure \* forRemove = this->find(id);

int number = this->\_findNumber(id);

delete forRemove;

this->\_figures.erase(this->\_figures.begin() + number);

}

FiguresGroup \* addGroup() {

FiguresGroup\* newGroup = new FiguresGroup();

this->\_groups.push\_back(newGroup);

return newGroup;

}

void removeGroup(int id) {

int number = this->\_findGroupNumber(id);

delete this->\_groups[number];

this->\_groups.erase(this->\_groups.begin() + number);

}

FiguresGroup\* findGroup(int id) {

FiguresGroup\* result = NULL;

int number = this->\_findGroupNumber(id);

if(number > 0) {

result = this->\_groups[number];

}

return result;

}

};

#endif

**FiguresGroup.h**

#ifndef FiguresStorage\_h\_

#define FiguresStorage\_h\_

#include "colorRGB.h"

#include "Border.h"

#include "Point.h"

#include "Figure.h"

#include "Rectangle.h"

#include "Parallelogram.h"

#include "Zigzag.h"

#include "FiguresGroup.h"

/\*\*

\* class FiguresStorage

\*/

class FiguresStorage {

private :

vector<Figure\*> \_figures;

vector<FiguresGroup\*> \_groups;

int \_findNumber(int id) {

int requiredId = -1;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->getId() == id) {

requiredId = i;

break;

}

}

return requiredId;

}

int \_findGroupNumber(int id) {

int result = -1;

for(int i = 0; i < this->\_groups.size(); i++) {

if(id == this->\_groups[i]->getId()) {

result = i;

}

}

return result;

}

public :

/\*\*

\* addRectangle

\* @param Point topLeft

\* @param Point bottomRight

\* @param Point controlPoint

\* @param Border border

\* @param vector<int> background

\* @return void

\*/

Rectangle\* addRectangle(Point topLeft, Point bottomRight, Border border, colorRGB background) {

Rectangle \* newRect = new Rectangle(topLeft, bottomRight, border, background);

this->\_figures.push\_back(newRect);

return newRect;

}

Rectangle\* addRectangle(Point topLeft, Point bottomRight) {

Rectangle \* newRect = new Rectangle(topLeft, bottomRight);

this->\_figures.push\_back(newRect);

return newRect;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight, double controlPoint, Border border, colorRGB background) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight, controlPoint, border, background);

this->\_figures.push\_back(newPar);

return newPar;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight, double controlPoint) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight, controlPoint);

this->\_figures.push\_back(newPar);

return newPar;

}

Parallelogram\* addParallelogram(Point topLeft, Point bottomRight) {

Parallelogram \* newPar = new Parallelogram(topLeft, bottomRight);

this->\_figures.push\_back(newPar);

return newPar;

}

Zigzag\* addZigzag(int stepsAmount, vector<Point> points, Border border) {

Zigzag \* newZig = new Zigzag(stepsAmount, points, border);

this->\_figures.push\_back(newZig);

return newZig;

}

/\*\*

\* getAmount

\* return int

\*/

int getAmount() {

return this->\_figures.size();

}

/\*\*

\* find

\* @param int id

\* @return Figure

\*/

Figure\* find(int id) {

Figure\* result = NULL;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->getId() == id) {

result = this->\_figures[i];

break;

}

}

return result;

}

/\*\*

\* find

\* @param Point

\* @return vector<Figure\*>

\*/

vector<Figure\*> find(Point point) {

vector<Figure\*> result;

for(int i = 0; i < this->\_figures.size(); i++) {

if(this->\_figures[i]->inLimitingRectangle(point)) {

result.push\_back(this->\_figures[i]);

}

}

return result;

}

void remove(int id) {

Figure \* forRemove = this->find(id);

int number = this->\_findNumber(id);

delete forRemove;

this->\_figures.erase(this->\_figures.begin() + number);

}

FiguresGroup \* addGroup() {

FiguresGroup\* newGroup = new FiguresGroup();

this->\_groups.push\_back(newGroup);

return newGroup;

}

void removeGroup(int id) {

int number = this->\_findGroupNumber(id);

delete this->\_groups[number];

this->\_groups.erase(this->\_groups.begin() + number);

}

FiguresGroup\* findGroup(int id) {

FiguresGroup\* result = NULL;

int number = this->\_findGroupNumber(id);

if(number > 0) {

result = this->\_groups[number];

}

return result;

}

};

#endif

**UML-діаграма**