Министерство образования Республики Башкортостан

Федеральное государственное бюджетное образовательное учреждение высшего образования "Уфимский государственный авиационный технический университет"

**Пояснительная записка к**

**расчётно-графической работе**

по дисциплине

Объектно-ориентированное программирование

ВЫПОЛНИЛ: ПРОВЕРИЛ:

студент группы ПРО-213сз преподаватель

***Григорьев В.Э.*** ***Фридлянд А.М.***

Уфа, 2018

Оглавление

[1. Постановка задачи 3](#_Toc508925681)

[2. Инструкция пользователя 5](#_Toc508925682)

[3. Описание классов. 7](#_Toc508925683)

[4. Диаграмма классов. 10](#_Toc508925684)

[5. Исходные коды 11](#_Toc508925685)

# Постановка задачи

Цель: разработка программного продукта с применением объектно-ориентированного программирования.

Задача: разработать графический редактор с изменяемыми векторными объектами с возможностью сохранения и загрузки результата.

Виды фигур:

* звезда;
* выпуклый многоугольник;
* круг;

Функциональность:

* добавление фигур на холст в рамках границы;
* удаление фигур;
* редактирование фигур после добавления:
  + изменение цвета фигуры;
  + изменение цвета границ фигуры;
  + редактирование размера;
  + редактирование угла наклона;
  + перемещение;
  + изменение количества углов;
* выборка фигуры:
  + выборка фигуры из списка;
  + выборка фигуры по нажатию на элемент;
  + визуальное выделение фигуры при выборе посредством изменения;
* сохранение результата (все данные о фигур) в файл;
* загрузка фигур из ранее сохраненного файла;

Системные требования:

* возможность запуска приложения в операционной среде windows через исполняемый .exe-файл или через web-браузер;

Требование к архитектуре:

* классы фигур должны быть потомками одного объекта;
* работа с методами и атрибутами фигур должна быть реализована при помощи полиморфизма.

# Инструкция пользователя

Рабочая область программы представляет собой окно с инструментами рисования и свойствами объектов Рисунок 1.

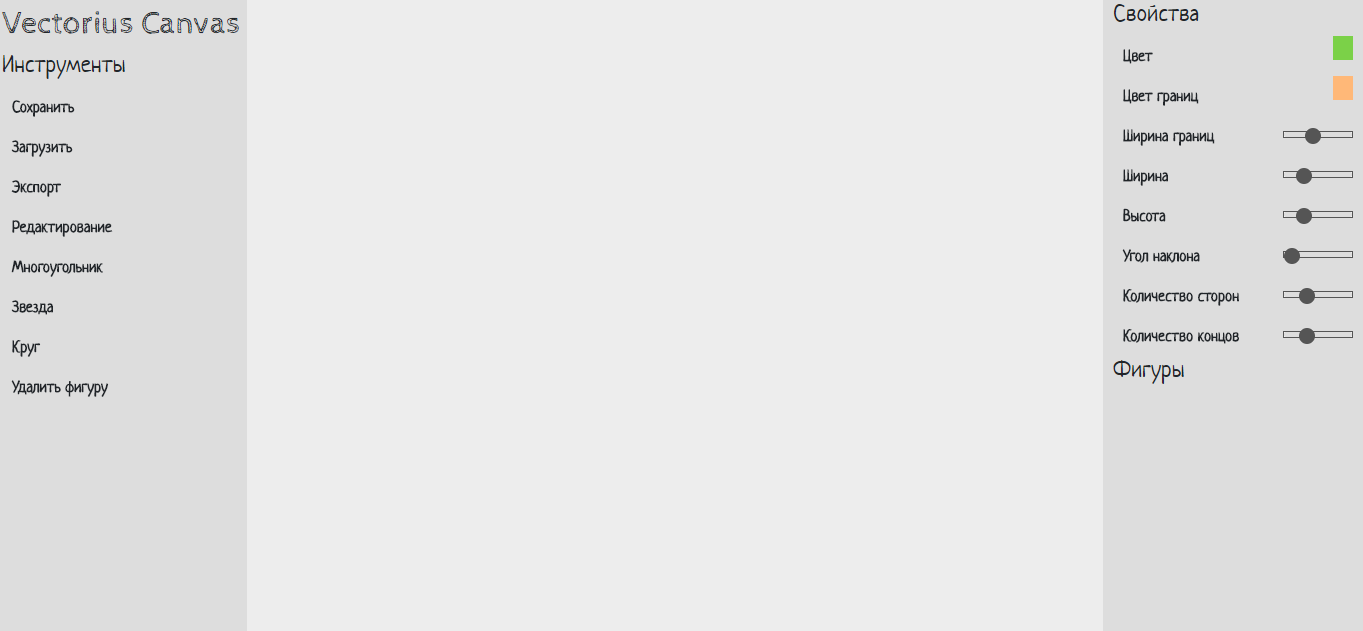


Рисунок 1

На рисунке 2 представлены указатели на основные рабочие зоны

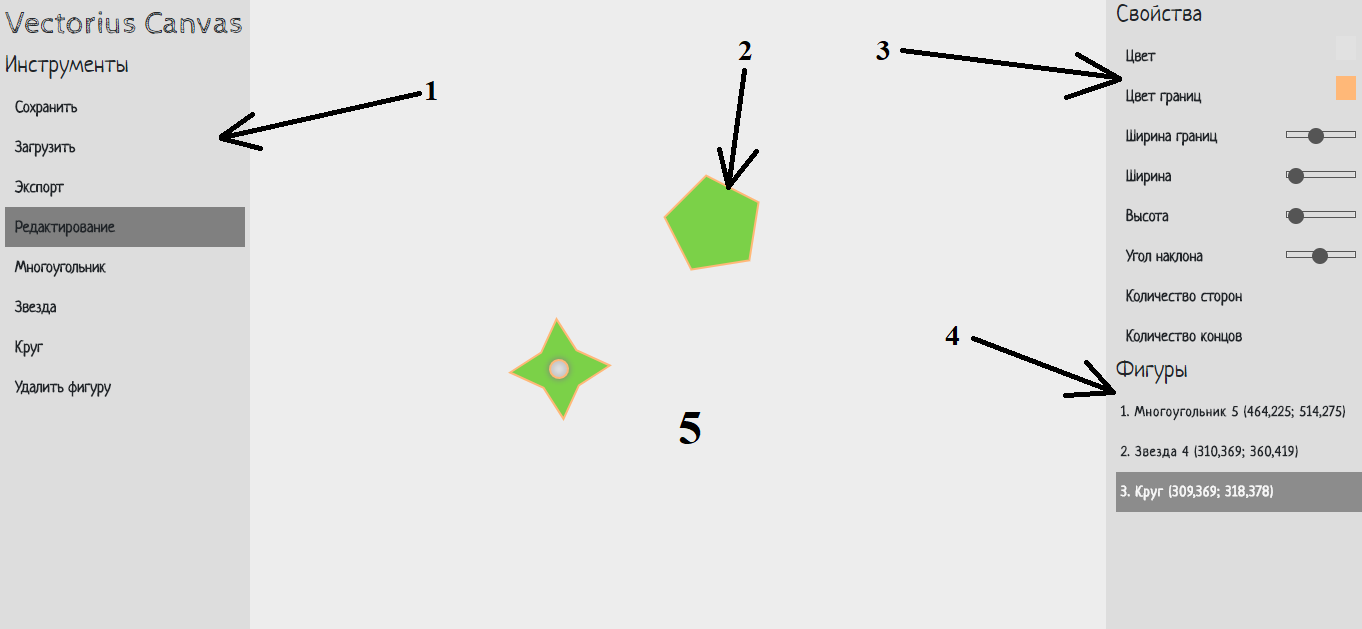


Рисунок 2

1) В этой зоне расположены основные инструменты программного продукта:

* сохранить: сохранение данного изображения;
* загрузить: загрузка необходимого изображения из дерева файлов;
* редактирование: эквивалентно курсору в графическом редакторе типа paint позволяет выбрать необходимую фигуру или передвинуть её по нажатию лкм;
* многоугольник: команда создает в рабочей зоне фигуру многоугольник с возможностью его дальнейшего редактирование;
* звезда: команда создает в рабочей зоне фигуру звезда с возможностью его дальнейшего редактирование;
* круг: команда создает в рабочей зоне фигуру круг с возможностью его дальнейшего редактирование;
* удалить фигуру: команда удаляет выбранную фигуру из рабочей зоны;

2) Созданная в рабочей зоне фигура.

3) Зона редактирования выбранной фигуры. В ней представлены функции изменения цвета фигура, её границ, изменения ширины границ, и изменения её физических параметров: ширины, высоты количества сторон и концов, и угла наклона, кроме круга.

4) История слоев. В ней находятся указатели на имеющиеся в данный момент фигуры. Так же в этой зоне можно выбрать нужную на данный момент фигуру.

5)Рабочая область программы в которой рисуются выбранные фигуры.

# Описание классов.

Связующее звено между DOM-элементами (Document Object Model) и классом Cicle. Реагирует на события пользовательского интерфейса.

**class** Interface{

ctx;

canvas;

currentFigure;

\_currentOperation;

\_options;

constructor(ctx, pointPathCanvasCtx, domElements);

getCurrentFigure();

getFigure(select);

setCurrentFigure(select);

setCurrentOperation(select);

selectFigure();

refreshFiguresList();

}

Класс, содержащий список фигур, методы работы со списком, отрисовку фигур на холсте и определение фигуры по координатам.

**class** Canvas{

\_items;

\_ctx;

constructor(ctx, pointPathCanvasCtx);

addPolygon(x, y, options);

addStar(x, y, options);

addCircle(x, y, options);

addItem(figure);

deleteItem(index);

draw();

getFigureIndexByCoord(x, y);

clear();

}

Aбстрактный класс, описывает общую структуру наследуемых классов

**class** Dot{

\_ctx;

\_x;

\_y;

\_color;

constructor(ctx, x, y, color);

draw(ctx);

getTitle();

toJson();

getFigureName();

getPosition();

move(x, y);

changeColor(color);

}

Абстрактный класс, содержит общие методы и атрибуты для объемных фигур. Наследуется от Dot.

**class** Figure **extends** Dot{

\_width;

\_height;

\_angle;

\_borderWidth;

\_borderColor;

\_shadow;

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow);

draw(ctx);

toJson();

getFigureName();

getPosition();

rotate(angle);

resize(width, height);

move(x,y);

changeBorderColor(borderColor);

changeBorderWidth(borderWidth);

setshadow(shadow);

}

Класс, описывающий отрисовку, сохранение в файл и отображение в списке круга. Наследуется от Figure.

**class** Circle **extends** Figure{

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow);

getFigureName();

draw(ctx);

toJson();

}

Класс, описывающий отрисовку, сохранение в файл и отображение в списке многоугольника. Наследуется от Figure.

**class** Polygon **extends** Figure{

\_sideCount;

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, sideCount, shadow);

getFigureName();

draw(ctx);

toJson();

changeSideCount(sideCount);

}

Класс, описывающий отрисовку, сохранение в файл и отображение в списке звезды. Наследуется от Figure.

**class** Star **extends** Figure{

\_spikeCount;

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, spikeCount, shadow);

getFigureName();

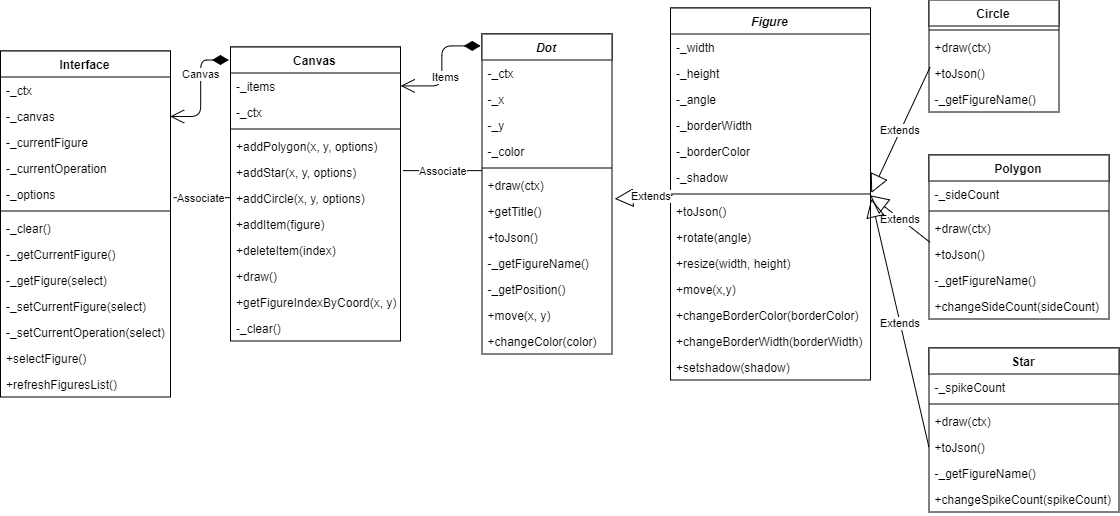
draw(ctx);

toJson();

changeSpikeCount(spikeCount);

}

# Диаграмма классов.



# Исходные коды

main.js

$( document ).ready(**function**(){

$('select[name="colorpicker\_color"]').simplecolorpicker({ picker: **true** });

$('select[name="colorpicker\_color"]').simplecolorpicker('selectColor', '#7bd148');

$('select[name="colorpicker\_bordercolor"]').simplecolorpicker({ picker: **true** });

$('select[name="colorpicker\_bordercolor"]').simplecolorpicker('selectColor', '#ffb878');

**let** canvas = document.getElementById('canvas');

**let** ctx = canvas.getContext('2d');

canvas.width = canvas.parentElement.clientWidth - 10;

canvas.height = canvas.parentElement.clientHeight - 1;

**let** pointPathCanvas= document.getElementById('point\_path\_canvas\_div');

**let** pointPathCanvasCtx = canvas.getContext('2d');

pointPathCanvasCtx.width = canvas.width;

pointPathCanvasCtx.height = canvas.height;

interfac = **new** Interface(ctx, pointPathCanvasCtx, {

'canvas': $('#canvas'),

'color': $('select[name="colorpicker\_color"]'),

'borderColor': $('select[name="colorpicker\_bordercolor"]'),

'borderWidth':$('#borderWidth'),

'width': $('#width'),

'height': $('#height'),

'angle': $('#angle'),

'sideCount': $('#sideCount'),

'spikeCount': $('#spikeCount'),

'polygon': $('#polygon'),

'star': $('#star'),

'circle': $('#circle'),

'edit': $('#edit'),

'deleteFigure': $('#delete\_figure'),

'figuresList': $('#figures\_list'),

'export': $('#export'),

'save': $('#save'),

'load': $('#load'),

});

});

Interface.js

**const** EDIT = 10;

**const** DELETE\_FIGURE = 11;

**const** ADD\_STAR = 21;

**const** ADD\_POLYGON = 22;

**const** ADD\_CIRCLE = 23;

**const** MOVE = 24;

**class** Interface{

constructor(ctx, pointPathCanvasCtx, domElements){

**if** (!ctx) **throw** **new** Error(`The ctx must implement.`);

**this**.\_ctx = ctx;

**if** (!pointPathCanvasCtx) **throw** **new** Error(`The pointPathCanvasCtx must implement.`);

**this**.\_canvas = **new** Canvas(**this**.ctx, pointPathCanvasCtx);

**this**.\_currentFigure = **null**;

**this**.\_currentOperation = **null**;

**this**.\_options = **new** Options();

**if** (!domElements) **throw** **new** Error(`The domElements must implement.`);

**if** (!domElements.canvas) **throw** **new** Error(`The domElements.canvas must implement.`);

**if** (!domElements.color) **throw** **new** Error(`The domElements.color must implement.`);

**if** (!domElements.borderColor) **throw** **new** Error(`The domElements.borderColor must implement.`);

**if** (!domElements.borderWidth) **throw** **new** Error(`The domElements.borderWidth must implement.`);

**if** (!domElements.width) **throw** **new** Error(`The domElements.width must implement.`);

**if** (!domElements.height) **throw** **new** Error(`The domElements.height must implement.`);

**if** (!domElements.angle) **throw** **new** Error(`The domElements.angle must implement.`);

**if** (!domElements.sideCount) **throw** **new** Error(`The domElements.sideCount must implement.`);

**if** (!domElements.spikeCount) **throw** **new** Error(`The domElements.spikeCount must implement.`);

**if** (!domElements.polygon) **throw** **new** Error(`The domElements.polygon must implement.`);

**if** (!domElements.star) **throw** **new** Error(`The domElements.star must implement.`);

**if** (!domElements.circle) **throw** **new** Error(`The domElements.circle must implement.`);

**if** (!domElements.edit) **throw** **new** Error(`The domElements.edit must implement.`);

**if** (!domElements.deleteFigure) **throw** **new** Error(`The domElements.delete\_figure must implement.`);

**if** (!domElements.save) **throw** **new** Error(`The domElements.save must implement.`);

**if** (!domElements.load) **throw** **new** Error(`The domElements.load must implement.`);

**if** (!domElements.**export**) **throw** **new** Error(`The domElements.**export** must implement.`);

**this**.\_domElements = domElements;

**this**.\_options.color = domElements.color.val();

**this**.\_options.borderColor = domElements.borderColor.val();

**this**.\_options.borderWidth = Number.parseInt(domElements.borderWidth.val());

**this**.\_options.width = Number.parseInt(domElements.width.val());

**this**.\_options.height = Number.parseInt(domElements.height.val());

**this**.\_options.angle = Number.parseInt(domElements.angle.val());

**this**.\_options.sideCount = Number.parseInt(domElements.sideCount.val());

**this**.\_options.spikeCount = Number.parseInt(domElements.spikeCount.val());

**let** that = **this**;

domElements.color.on('change', **function**(){

that.\_options.color = $(**this**).val();

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().changeColor(that.\_options.color);

that.\_canvas.draw();

}

});

domElements.borderColor.on('change', **function**(){

that.\_options.borderColor = $(**this**).val();

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().changeBorderColor(that.\_options.borderColor);

that.\_canvas.draw();

}

});

domElements.borderWidth.on('change', **function**(){

that.\_options.borderWidth = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().changeBorderWidth(that.\_options.borderWidth);

that.\_canvas.draw();

}

});

domElements.width.on('input', **function**(){

that.\_options.width = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().resize(that.\_options.width, that.\_options.height);

that.\_canvas.draw();

}

});

domElements.height.on('input', **function**(){

that.\_options.height = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().resize(that.\_options.width, that.\_options.height);

that.\_canvas.draw();

}

});

domElements.angle.on('input', **function**(){

that.\_options.angle = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && that.\_currentFigure != **null**){

that.\_getCurrentFigure().rotate(that.\_options.angle);

that.\_canvas.draw();

}

});

domElements.sideCount.on('input', **function**(){

that.\_options.sideCount = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && (that.\_getCurrentFigure() **instanceof** Polygon)){

that.\_getCurrentFigure().changeSideCount(that.\_options.sideCount);

that.\_canvas.draw();

}

});

domElements.spikeCount.on('input', **function**(){

that.\_options.spikeCount = Number.parseInt($(**this**).val());

**if** (that.\_currentOperation == EDIT && (that.\_getCurrentFigure() **instanceof** Star)){

that.\_getCurrentFigure().changeSpikeCount(that.\_options.spikeCount);

that.\_canvas.draw();

}

});

domElements.canvas.on('click', **function**(event){

**let** xPos = Math.ceil(event.pageX - $(**this**).offset().left);

**let** yPos = Math.ceil(event.pageY - $(**this**).offset().top);

**switch** (that.\_currentOperation) {

**case** ADD\_POLYGON:

that.\_canvas.addPolygon(xPos, yPos, that.\_options);

**break**;

**case** ADD\_STAR:

that.\_canvas.addStar(xPos, yPos, that.\_options);

**break**;

**case** ADD\_CIRCLE:

that.\_canvas.addCircle(xPos, yPos, that.\_options);

**break**;

**case** EDIT:

that.\_setCurrentFigure(that.\_canvas.getFigureIndexByCoord(xPos, yPos));

**break**;

**case** MOVE:

that.\_currentOperation = EDIT;

}

that.\_canvas.draw();

that.refreshFiguresList();

})

domElements.canvas.on('mousedown', **function**(event){

**switch** (that.\_currentOperation) {

**case** EDIT:

**let** xPos = Math.ceil(event.pageX - $(**this**).offset().left);

**let** yPos = Math.ceil(event.pageY - $(**this**).offset().top);

**let** mfigure = that.\_canvas.getFigureIndexByCoord(xPos, yPos);

**if** (that.\_currentFigure != **null** && that.\_currentFigure == mfigure) {

that.\_currentOperation = MOVE;

}

}

})

domElements.canvas.on('mousemove', **function**(event){

**switch** (that.\_currentOperation) {

**case** MOVE:

**let** xPos = Math.ceil(event.pageX - $(**this**).offset().left);

**let** yPos = Math.ceil(event.pageY - $(**this**).offset().top);

that.\_getCurrentFigure().move(xPos, yPos);

that.\_canvas.draw();

}

})

domElements.figuresList.on('change', '.radiobox-div', **function**(event){

that.\_setCurrentFigure(Number.parseInt(**this**.getAttribute('attr-index')));

that.\_setCurrentOperation(EDIT);

that.selectFigure();

that.\_canvas.draw();

})

domElements.edit.on('click', **function**(){

that.\_setCurrentOperation(EDIT);

**if** (that.\_canvas.items.length > 0)

that.\_setCurrentFigure(that.\_canvas.items.length-1);

**else**

that.\_setCurrentFigure(**null**);

that.selectFigure();

that.\_canvas.draw();

});

domElements.deleteFigure.on('click', **function**(){

**if** (that.\_currentFigure != **null**){

that.\_canvas.deleteItem(that.\_currentFigure);

**if** (that.\_canvas.items.length == 0){

that.\_setCurrentFigure(**null**);

}**else** **if** (that.\_currentFigure == that.\_canvas.items.length){

that.\_setCurrentFigure(that.\_canvas.items.length-1);

}

} **else** {

that.\_setCurrentFigure(that.\_canvas.items.length-1);

}

that.\_canvas.draw();

that.refreshFiguresList();

that.\_setCurrentOperation(EDIT);

});

domElements.polygon.on('click', **function**(){

that.\_setCurrentOperation(ADD\_POLYGON);

that.\_setCurrentFigure(**null**);

that.\_domElements.spikeCount.hide();

that.\_domElements.sideCount.show();

});

domElements.star.on('click', **function**(){

that.\_setCurrentOperation(ADD\_STAR);

that.\_setCurrentFigure(**null**);

that.\_domElements.spikeCount.show();

that.\_domElements.sideCount.hide();

});

domElements.circle.on('click', **function**(){

that.\_setCurrentOperation(ADD\_CIRCLE);

that.\_setCurrentFigure(**null**);

that.\_domElements.spikeCount.hide();

that.\_domElements.sideCount.hide();

});

domElements.save.on('click', **function**(){

**let** jsonArr = [];

**for** (**var** i = 0; i < that.\_canvas.items.length; i++) {

jsonArr.push(that.\_canvas.items[i].toJson());

}

**var** blob = **new** Blob([JSON.stringify(jsonArr)], {type: "text/plain;charset=utf-8"})

saveAs(blob, "vecotoriass.txt");

});

domElements.**export**.on('click', **function**(){

that.\_ctx.canvas.toBlob(**function**(blob) {

saveAs(blob, "pretty image.png");

})

});

domElements.load.on('change', **function**(e){

**let** file = e.target.files[0];

**if** (!file) {

**return**;

}

**var** reader = **new** FileReader();

reader.onload = **function**(e) {

**let** contents = e.target.result;

**let** figures = JSON.parse(contents);

**for**(**let** i=0;i<figures.length;i++){

**switch** (figures[i].type) {

**case** "Polygon":

that.\_canvas.addItem(**new** Polygon(that.ctx, figures[i].x, figures[i].y, figures[i].width, figures[i].width, figures[i].angle, figures[i].borderWidth, figures[i].color, figures[i].borderColor, figures[i].sideCount, **false**))

**break**;

**case** "Star":

that.\_canvas.addItem(**new** Star(that.ctx, figures[i].x, figures[i].y, figures[i].width, figures[i].width, figures[i].angle, figures[i].borderWidth, figures[i].color, figures[i].borderColor, figures[i].spikeCount, **false**))

**break**;

**case** "Circle":

that.\_canvas.addItem(**new** Circle(that.ctx, figures[i].x, figures[i].y, figures[i].width, figures[i].width, figures[i].angle, figures[i].borderWidth, figures[i].color, figures[i].borderColor, **false**))

**break**;

}

}

that.\_canvas.draw();

that.refreshFiguresList();

};

reader.readAsText(file);

})

}

\_getCurrentFigure(){

**if** (**this**.\_currentFigure != **null**)

**return** **this**.\_canvas.items[**this**.\_currentFigure];

**else**

**return** **null**;

}

\_getFigure(select){

**if** (select != **null**)

**return** **this**.\_canvas.items[select];

**else**

**return** **null**;

}

\_setCurrentFigure(select){

**if** (**this**.\_currentFigure != select) {

**if** (select != **null**) {

**this**.\_getFigure(select).setshadow(**true**);

}

**if** (**this**.\_getCurrentFigure() != **null**) {

**this**.\_getCurrentFigure().setshadow(**false**);

}

**this**.\_currentFigure = select;

}

}

\_setCurrentOperation(select){

**if** (**this**.\_currentOperation != select) {

**this**.\_domElements.polygon.removeClass("select\_toolbar");

**this**.\_domElements.star.removeClass("select\_toolbar");

**this**.\_domElements.circle.removeClass("select\_toolbar");

**this**.\_domElements.edit.removeClass("select\_toolbar");

**switch** (select) {

**case** ADD\_POLYGON:

**this**.\_domElements.polygon.addClass("select\_toolbar");

**break**;

**case** ADD\_STAR:

**this**.\_domElements.star.addClass("select\_toolbar");

**break**;

**case** ADD\_CIRCLE:

**this**.\_domElements.circle.addClass("select\_toolbar");

**break**;

**case** EDIT:

**this**.\_domElements.edit.addClass("select\_toolbar");

**break**;

}

**this**.\_currentOperation = select;

}

}

selectFigure(){

**if** (**this**.\_currentFigure != **null**){

$('.radiobox-div[attr-index=' + **this**.\_currentFigure + '] .radiobox input').prop("checked", **true**);

**let** figure = **this**.\_getCurrentFigure();

**this**.\_domElements.width.val(figure.width);

**this**.\_domElements.height.val(figure.height);

**this**.\_domElements.angle.val(figure.angle);

**this**.\_domElements.borderWidth.val(figure.borderWidth);

**this**.\_domElements.color.simplecolorpicker('selectColor',figure.color);

**this**.\_domElements.borderColor.simplecolorpicker('selectColor',figure.borderColor);

**if** (figure **instanceof** Star){

**this**.\_domElements.spikeCount.val(figure.spikeCount);

**this**.\_domElements.spikeCount.show();

}**else**

**this**.\_domElements.spikeCount.hide();

**if** (figure **instanceof** Polygon){

**this**.\_domElements.sideCount.val(figure.sideCount);

**this**.\_domElements.sideCount.show();

}**else**

**this**.\_domElements.sideCount.hide();

}

}

refreshFiguresList(){

**let** figuresList = **this**.\_domElements.figuresList;

figuresList.html("");

**for** (**var** i = 0; i < **this**.\_canvas.items.length; i++) {

name = (i+1) + '. ' + **this**.\_canvas.items[i].getTitle();

**let** listItem = "<div class='radiobox-div' attr-index="+i+"><label class='radiobox'><input type='radio' name='selectedIdEneble'><span class='item'>"+name+"</span></label></div>"

figuresList.append(listItem);

}

**this**.selectFigure();

}

get ctx(){ **return** **this**.\_ctx; }

set ctx(value) { **throw** **new** Error(`The ctx property cannot be written. ${value} was passed.`); }

}

Canvas.js

**class** Canvas{

constructor(ctx, pointPathCanvasCtx){

**if** (!ctx) **throw** **new** Error(`The ctx must implement.`);

**this**.\_ctx = ctx;

**this**.\_pointPathCanvasCtx = pointPathCanvasCtx;

**this**.\_items = [];

}

addPolygon(x, y, options){

**let** polygon = **new** Polygon(**this**.ctx, x, y, options.width, options.width, options.angle, options.borderWidth, options.color, options.borderColor, options.sideCount, **false**);

**this**.addItem(polygon);

}

addStar(x, y, options){

**let** star = **new** Star(**this**.ctx, x, y, options.width, options.width, options.angle, options.borderWidth, options.color, options.borderColor, options.spikeCount, **false**);

**this**.addItem(star);

}

addCircle(x, y, options){

**let** circle = **new** Circle(**this**.ctx, x, y, options.width, options.width, options.angle, options.borderWidth, options.color, options.borderColor, **false**);

**this**.addItem(circle);

}

addItem(figure){

**if** (figure **instanceof** Figure){

**this**.\_items.push(figure);

}**else**{

**throw** **new** Error(`Parametr must be instance of Figure child`);

}

}

deleteItem(index){

**this**.items.splice(index, 1);

}

draw(){

**this**.\_clear();

**this**.items.forEach(**function**(figure) {

figure.draw();

});

}

getFigureIndexByCoord(x, y){

**for** (**var** i = **this**.items.length-1; i > -1 ; i--) {

**this**.items[i].draw(**this**.\_pointPathCanvasCtx);

**if** (**this**.\_pointPathCanvasCtx.isPointInPath(x, y))

**return** i

}

**return** **null**

}

\_clear(){

**this**.ctx.clearRect(0, 0, **this**.ctx.canvas.width, **this**.ctx.canvas.height)

}

get ctx(){ **return** **this**.\_ctx; }

set ctx(value) { **throw** **new** Error(`The ctx property cannot be written. ${value} was passed.`); }

get items(){ **return** **this**.\_items; }

set items(value) { **throw** **new** Error(`The items property cannot be written. ${value} was passed.`); }

}

Dot.js

**class** Dot{

constructor(ctx, x, y, color){

**this**.\_ctx = ctx;

**if** (!Number.isInteger(x)) **throw** **new** Error(`The x must be integer.`);

**if** (!Number.isInteger(y)) **throw** **new** Error(`The y must be integer.`);

**if** (!color) **throw** **new** Error(`The color must implement.`);

**this**.\_x = x;

**this**.\_y = y;

**this**.\_color = color;

}

draw(ctx){

**if** (!ctx) ctx = **this**.ctx;

ctx.fillRect(**this**.x,**this**.y,1,1);

}

getTitle(){

**return** **this**.\_getFigureName() + " "+ **this**.\_getPosition()

}

toJson(){

**return** {

'type': 'Doc',

'x': **this**.x,

'y': **this**.y,

'color': **this**.color

}

}

\_getFigureName(){

**return** "Точка"

}

\_getPosition(){

**return** "("+**this**.x+","+**this**.y+")"

}

move(x, y){

**if** (!Number.isInteger(x)) **throw** **new** Error(`The x must be integer.`);

**if** (!Number.isInteger(y)) **throw** **new** Error(`The y must be integer.`);

**this**.\_x = x;

**this**.\_y = y;

}

changeColor(color){

**if** (!color) **throw** **new** Error(`The color must implement.`);

**this**.\_color = color;

}

get x(){ **return** **this**.\_x; }

set x(value) { **throw** **new** Error(`The x property cannot be written. ${value} was passed.`); }

get y(){ **return** **this**.\_y; }

set y(value) { **throw** **new** Error(`The y property cannot be written. ${value} was passed.`); }

get color(){ **return** **this**.\_color; }

set color(value) { **throw** **new** Error(`The color property cannot be written. ${value} was passed.`); }

get ctx(){ **return** **this**.\_ctx; }

set ctx(value) { **throw** **new** Error(`The ctx property cannot be written. ${value} was passed.`); }

}

Figure.js

**class** Figure **extends** Dot{

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow){

**super**(ctx, x, y, color);

**if** (!Number.isInteger(width)) **throw** **new** Error(`The width must be integer.`);

**if** (!Number.isInteger(height)) **throw** **new** Error(`The height must be integer.`);

**if** (!Number.isInteger(angle)) **throw** **new** Error(`The angle must be integer.`);

**if** (!Number.isInteger(borderWidth)) **throw** **new** Error(`The angle must be integer.`);

**if** (!borderColor) **throw** **new** Error(`The borderColor must implement.`);

**this**.\_width = width;

**this**.\_height = height;

**this**.\_angle = angle+45;

**this**.\_borderColor = borderColor;

**this**.\_borderWidth = borderWidth;

**this**.\_shadow = shadow;

}

draw(ctx){

**if** (!ctx) ctx = **this**.ctx;

**throw** **new** Error('You have to implement the method draw!');

}

toJson(){

**let** json = **super**.toJson();

Object.assign(json, {

'type': 'Figure',

'width': **this**.width,

'height': **this**.height,

'angle': **this**.angle,

'borderColor': **this**.borderColor,

'borderWidth': **this**.borderWidth

});

**return** json;

}

\_getFigureName(){

**return** "Фигура"

}

\_getPosition(){

**return** "("+**this**.x+","+**this**.y+"; "+(**this**.x+**this**.width)+","+(**this**.y+**this**.height)+")"

}

rotate(angle){

**if** (!Number.isInteger(angle)) **throw** **new** Error(`The angle must be integer.`);

**this**.\_angle = angle+45;

}

resize(width, height){

**if** (!Number.isInteger(width)) **throw** **new** Error(`The width must be integer.`);

**if** (!Number.isInteger(height)) **throw** **new** Error(`The height must be integer.`);

**this**.\_width = width;

**this**.\_height = height;

}

move(x,y){

**if** (!Number.isInteger(x)) **throw** **new** Error(`The x must be integer.`);

**if** (!Number.isInteger(y)) **throw** **new** Error(`The y must be integer.`);

**this**.\_x = x;

**this**.\_y = y;

}

changeBorderColor(borderColor){

**this**.\_borderColor = borderColor;

}

changeBorderWidth(borderWidth){

**if** (!Number.isInteger(borderWidth)) **throw** **new** Error(`The angle must be integer.`);

**this**.\_borderWidth = borderWidth;

}

setshadow(shadow){

**this**.\_shadow = shadow;

}

get width(){ **return** **this**.\_width; }

set width(value) { **throw** **new** Error(`The width property cannot be written. ${value} was passed.`); }

get height(){ **return** **this**.\_height; }

set height(value) { **throw** **new** Error(`The height property cannot be written. ${value} was passed.`); }

get angle(){ **return** **this**.\_angle; }

set angle(value) { **throw** **new** Error(`The angle property cannot be written. ${value} was passed.`); }

get borderColor(){ **return** **this**.\_borderColor; }

set borderColor(value) { **throw** **new** Error(`The borderColor property cannot be written. ${value} was passed.`); }

get borderWidth(){ **return** **this**.\_borderWidth; }

set borderWidth(value) { **throw** **new** Error(`The borderWidth property cannot be written. ${value} was passed.`); }

get shadow(){ **return** **this**.\_shadow; }

set shadow(value) { **throw** **new** Error(`The shadow property cannot be written. ${value} was passed.`); }

}

Circle.js

**class** Circle **extends** Figure{

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow){

**super**(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow);

}

\_getFigureName(){

**return** "Круг"

}

draw(ctx){

**if** (!ctx) ctx = **this**.ctx;

ctx.translate(**this**.x, **this**.y);

ctx.rotate((Math.PI / 180) \* **this**.angle);

**if** (**this**.shadow) {

ctx.shadowColor = 'gray';

ctx.shadowBlur = 10;

}

ctx.beginPath();

ctx.ellipse(0, 0, **this**.width, **this**.height, 0, 0, 2 \* Math.PI);

ctx.closePath();

**if** (**this**.color != **undefined**) {

ctx.fillStyle = **this**.color;

ctx.fill();

}

**if** (**this**.borderColor != **undefined** && **this**.borderWidth){

ctx.lineWidth = **this**.borderWidth;

ctx.strokeStyle = **this**.borderColor;

ctx.stroke();

}

ctx.rotate(-(Math.PI / 180) \* **this**.angle);

ctx.translate(-**this**.x, -**this**.y);

**if** (**this**.shadow) {

ctx.shadowBlur = 0;

}

}

toJson(){

**let** json = **super**.toJson();

Object.assign(json, {

'type': 'Circle'

});

**return** json;

}

}

Polygon.js

**class** Polygon **extends** Figure{

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, sideCount, shadow){

**super**(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow);

**if** (!Number.isInteger(sideCount)) **throw** **new** Error(`The sideCount must be integer.`);

**this**.\_sideCount = sideCount;

}

\_getFigureName(){

**return** "Многоугольник " + **this**.sideCount;

}

draw(ctx){

**if** (!ctx) ctx = **this**.ctx;

ctx.translate(**this**.x, **this**.y);

ctx.rotate((Math.PI / 180) \* **this**.angle);

**if** (**this**.shadow) {

ctx.shadowColor = 'gray';

ctx.shadowBlur = 10;

}

ctx.beginPath();

ctx.moveTo(**this**.width, 0);

**let** a = ((Math.PI \* 2)/**this**.sideCount);

**for** (**let** i = 1; i < **this**.sideCount; i++) {

ctx.lineTo(**this**.width \* Math.cos(a \* i), **this**.height \* Math.sin(a \* i));

}

ctx.closePath();

**if** (**this**.color != **undefined**) {

ctx.fillStyle = **this**.color;

ctx.fill();

}

**if** (**this**.borderColor != **undefined** && **this**.borderWidth){

ctx.lineWidth = **this**.borderWidth;

ctx.strokeStyle = **this**.borderColor;

ctx.stroke();

}

ctx.rotate(-(Math.PI / 180) \* **this**.angle);

ctx.translate(-**this**.x, -**this**.y);

**if** (**this**.shadow) {

ctx.shadowBlur = 0;

}

}

toJson(){

**let** json = **super**.toJson();

Object.assign(json, {

'type': 'Polygon',

'sideCount': **this**.sideCount

});

**return** json;

}

changeSideCount(sideCount){

**if** (!Number.isInteger(sideCount)) **throw** **new** Error(`The sideCount must be integer.`);

**this**.\_sideCount = sideCount;

}

get sideCount(){ **return** **this**.\_sideCount; }

set sideCount(value) { **throw** **new** Error(`The sideCount property cannot be written. ${value} was passed.`); }

}

Star.js

**class** Star **extends** Figure{

constructor(ctx, x, y, width, height, angle, borderWidth, color, borderColor, spikeCount, shadow){

**super**(ctx, x, y, width, height, angle, borderWidth, color, borderColor, shadow);

**this**.\_spikeCount = spikeCount;

}

\_getFigureName(){

**return** "Звезда " + **this**.spikeCount;

}

draw(ctx){

**if** (!ctx) ctx = **this**.ctx;

**let** rot = Math.PI / 2 \* 3;

**let** x = **this**.x;

**let** y = **this**.y;

**let** step = Math.PI / **this**.spikeCount;

**let** outerRadius = **this**.\_width;

**let** innerRadius = Math.ceil(**this**.\_width \* 0.5);

ctx.translate(**this**.x, **this**.y);

ctx.rotate((Math.PI / 180) \* **this**.angle);

**if** (**this**.shadow) {

ctx.shadowColor = 'gray';

ctx.shadowBlur = 10;

}

ctx.beginPath();

ctx.moveTo(0, 0 - outerRadius)

**for** (**let** i = 0; i < **this**.spikeCount; i++) {

x = Math.cos(rot) \* outerRadius;

y = Math.sin(rot) \* outerRadius;

ctx.lineTo(x, y)

rot += step

x = Math.cos(rot) \* innerRadius;

y = Math.sin(rot) \* innerRadius;

ctx.lineTo(x, y)

rot += step

}

ctx.lineTo(0, 0 - outerRadius)

ctx.closePath();

**if** (**this**.color != **undefined**) {

ctx.fillStyle = **this**.color;

ctx.fill();

}

**if** (**this**.borderColor != **undefined** && **this**.borderWidth){

ctx.lineWidth = **this**.borderWidth;

ctx.strokeStyle = **this**.borderColor;

ctx.stroke();

}

ctx.rotate(-(Math.PI / 180) \* **this**.angle);

ctx.translate(-**this**.x, -**this**.y);

**if** (**this**.shadow) {

ctx.shadowBlur = 0;

}

}

toJson(){

**let** json = **super**.toJson();

Object.assign(json, {

'type': 'Star',

'spikeCount': **this**.spikeCount

});

**return** json;

}

changeSpikeCount(spikeCount){

**if** (!Number.isInteger(spikeCount)) **throw** **new** Error(`The sideCount must be integer.`);

**this**.\_spikeCount = spikeCount;

}

get spikeCount(){ **return** **this**.\_spikeCount; }

set spikeCount(value) { **throw** **new** Error(`The spikeCount property cannot be written. ${value} was passed.`); }

}