**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №5: Выбор и сохранение фракталов

по дисциплине «Кроссплатформенные технологии программирования»

В данной лабораторной работе генератор фракталов будет расширен двумя новыми функциями. Во-первых, вы добавите поддержку нескольких фракталов и реализуете возможность выбирать нужный фрактал из выпадающего списка. Во-вторых, вы добавите поддержку сохранения текущего изображения в файл.

Выполнил: студент

группы БСТ1803

Шевцов Никита

Вариант №24

Москва 2020

**Содержание**

[1 Задание на разработку программы 3](#_Toc53411799)

[2 Разработанные функции и классы 7](#_Toc53411800)

[2.1 Класс FractalExplorer 7](#_Toc53411801)

[2.1.1 Подкласс ActionListener button\_Click с методом actionPerformed 7](#_Toc53411802)

[2.2 Класс BurningShip 7](#_Toc53411803)

[2.2.1 Метод void getInitialRange 7](#_Toc53411804)

[2.2.2 Метод int numIterations 8](#_Toc53411805)

[2.3 Класс Tricorn 8](#_Toc53411806)

[2.3.1 Метод void getInitialRange 8](#_Toc53411807)

[2.3.2 Метод int numIterations 8](#_Toc53411808)

[2.4 Класс Complex 8](#_Toc53411809)

[2.4.1 Метод void zero 8](#_Toc53411810)

[2.4.2 Метод void set 8](#_Toc53411811)

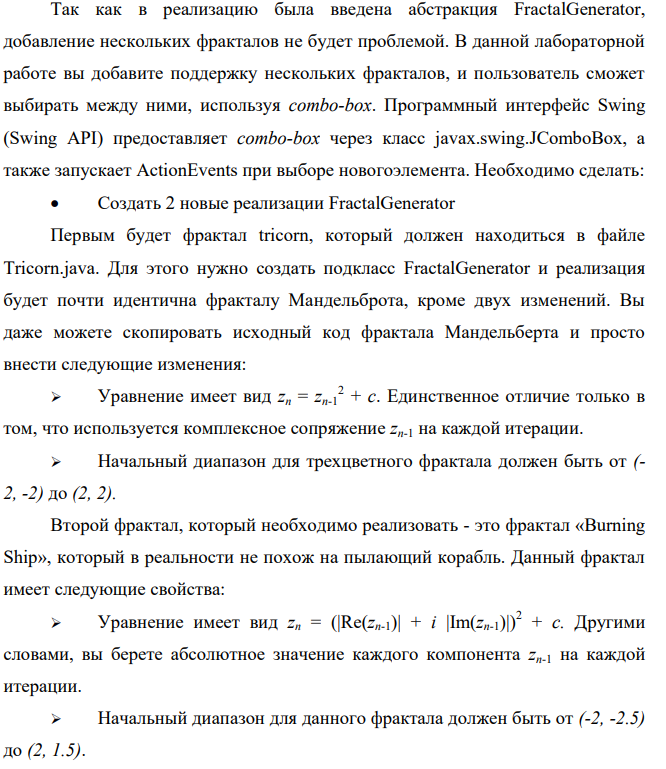
[2.4.3 Метод double get 9](#_Toc53411812)

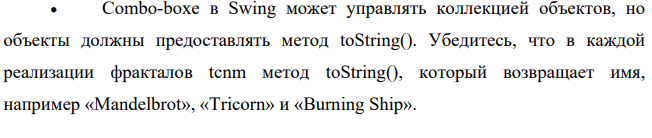
[2.4.4 Метод Complex conjugate 9](#_Toc53411813)

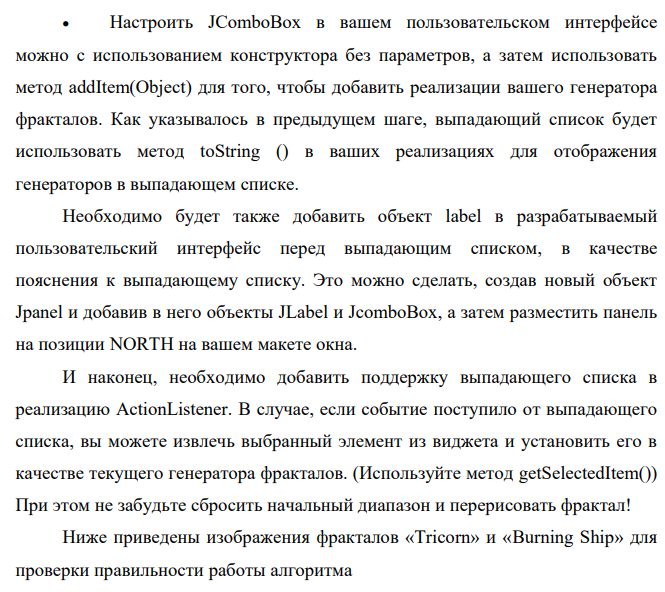
[3 Код программы 9](#_Toc53411814)

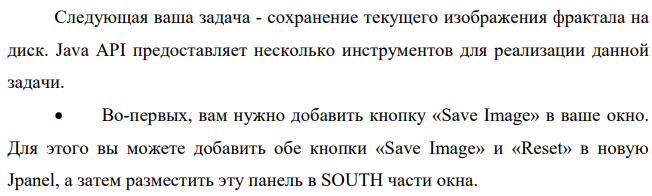
[4 Результат работы программы 15](#_Toc53411815)

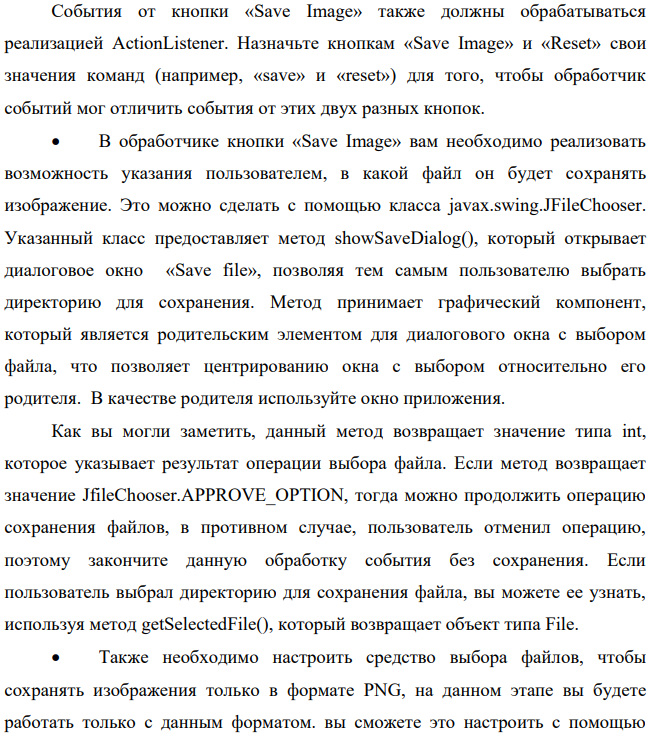
# Задание на разработку программы

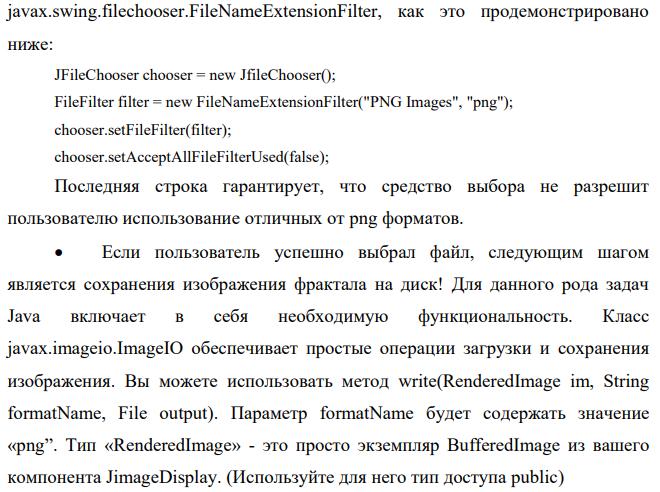


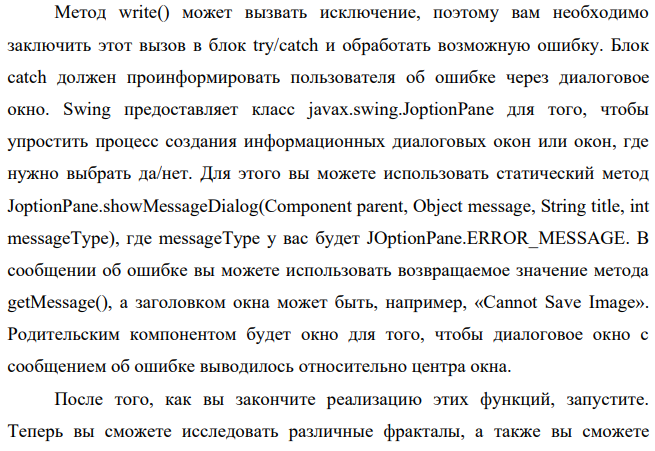


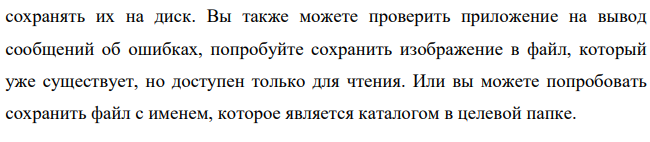












# Разработанные функции и классы

## Класс FractalExplorer

Класс отрисовывающий графический интерфейс Swing.

* + 1. Подкласс ActionListener button\_Click с методом actionPerformed

Добавлена обработка кнопок сохранения, выбора фрактала.

## Класс BurningShip

Реализует методы, необходимые для генерации фрактала “Горящий корабль”

* + 1. Метод void getInitialRange

Метод определяющий область комплексной плоскости для фрактала “Горящий корабль”

* + 1. Метод int numIterations

Метод, вычисляющий количество итераций для отрисовки точки

## Класс Tricorn

Реализует методы, необходимые для генерации фрактала “Треуголка”

* + 1. Метод void getInitialRange

Метод определяющий область комплексной плоскости для фрактала “Множество Мандельбара ”

* + 1. Метод int numIterations

Метод, вычисляющий количество итераций для отрисовки точки

## Класс Complex

* + 1. Метод void zero

Метод, обнуляющий комплексное число

* + 1. Метод void set

Метод, устанавливающий значения комплексному числу

* + 1. Метод double get

Метод, возвращающий выбранную часть комплексного числа

* + 1. Метод Complex conjugate

Метод, возвращающий комплексное сопряжение числа.

# Код программы

/\*\*

\* Класс комплексных чисел

\* **@author** niksh

\* **@version** 1.1

\*/

**public** **class** Complex **extends** Number{

**private** **static** **final** **long** ***serialVersionUID*** = 7746583188569394990L;

/\*\* Реальная составляющая числа \*/

**private** **double** real;

/\*\* Комплексная составляющая числа \*/

**private** **double** imag;

/\*\*

\* Конструктор - создает комплексное число с заданными значениями

\* **@param** real - реальная составляющая числа

\* **@param** imag - комплексная составляющая числа

\*/

**public** Complex(**double** real, **double** imag) {

**this**.real = real;

**this**.imag = imag;

}

/\*\*

\* Конструктор - создает комплексное число 0 + i0

\*/

**public** Complex() {

**this**(0,0);

}

/\*\*

\* Обнуление числа

\*/

**public** **void** zero() {

**this**.real = 0;

**this**.imag = 0;

}

/\*\*

\* Установка значений числа

\* **@param** real - действительная часть

\* **@param** imag - мнимая часть

\*/

**public** **void** set(**double** real, **double** imag) {

**this**.real = real;

**this**.imag = imag;

}

/\*\*

\* Возвращает одну из частей комплексного числа

\* **@param** sw - 'r' для действительной, 'i' для мнимой

\* **@return** В случае неправильного ключа возвращает 0.0

\*/

**public** **double** get(**char** sw) {

**switch** (sw) {

**case** ('r'):

**return** **this**.real;

**case** ('i'):

**return** **this**.imag;

}

**return** 0.0;

}

/\*\*

\* Операция сложения комплексных чисел

\* **@param** n - комплексное число для сложения

\* **@return** Комплексная сумма

\*/

**public** Complex add(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real + n.real;

res.imag = **this**.imag + n.imag;

**return** res;

}

/\*\*

\* Операция сложения комплексного и обычного числа

\* **@param** n - число для сложения

\* **@return** Комплексная сумма

\*/

**public** Complex add(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real += (**double**) n;

**return** res;

}

/\*\*

\* Операция вычитания комплексных чисел

\* **@param** n - комплексное число для вычитания

\* **@return** Комплексная разность

\*/

**public** Complex sub(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real - n.real;

res.imag = **this**.imag - n.imag;

**return** res;

}

/\*\*

\* Операция вычитания комплексного и обычного числа

\* **@param** n - число для вычитания

\* **@return** Комплексная разность

\*/

**public** Complex sub(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real -= (**double**) n;

**return** res;

}

/\*\*

\* Операция умножения комплексных чисел

\* **@param** n - комплексное число для умножения

\* **@return** Комплексное произведение

\*/

**public** Complex mul(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real\*n.real - **this**.imag\*n.imag;

res.imag = **this**.real\*n.imag + **this**.imag\*n.real;

**return** res;

}

/\*\*

\* Операция умножения комплексного и обычного числа

\* **@param** n - число для умножения

\* **@return** Комплексное произведение

\*/

**public** Complex mul(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real \*= (**double**)n;

res.imag \*= (**double**)n;

**return** res;

}

/\*\*

\* Операция деления комплексных чисел

\* **@param** n - комплексное число для деления

\* **@return** Комплексное частное

\*/

**public** Complex div(Complex n) {

Complex res = **new** Complex();

**double** divisor = n.real\*n.real + n.imag\*n.imag;

res.real = (**this**.real\*n.real + **this**.imag\*n.imag)/divisor;

res.imag = (**this**.imag\*n.real - **this**.real\*n.imag)/divisor;

**return** res;

}

/\*\*

\* Операция деления комплексного и обычного числа

\* **@param** n - число для деления

\* **@return** Комплексное частное

\*/

**public** Complex div(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real /= (**double**) n;

res.imag /= (**double**) n;

**return** res;

}

/\*\*

\* Квадрат модуля комплексного числа

\*/

**public** **double** absSqr() {

**return** **this**.real\***this**.real + **this**.imag\***this**.imag;

}

/\*\*

\* Комплексное сопряжение числа

\*/

**public** Complex conjugate() {

**return** **new** Complex(**this**.real,-**this**.imag);

}

/\*\* Модуль комплексного числа типа <code>double</code> \*/

@Override

**public** **double** doubleValue() {

**double** res = Math.*sqrt*(**this**.real\***this**.real + **this**.imag\***this**.imag);

**return** res;

}

/\*\* Модуль комплексного числа типа <code>float</code> \*/

@Override

**public** **float** floatValue() {

**float** res = (**float**) Math.*sqrt*(**this**.real\***this**.real + **this**.imag\***this**.imag);

**return** res;

}

/\*\* Модуль комплексного числа типа <code>int</code> \*/

@Override

**public** **int** intValue() {

**int** res = (**int**) Math.*sqrt*(**this**.real\***this**.real + **this**.imag\***this**.imag);

**return** res;

}

/\*\* Модуль комплексного числа типа <code>long</code> \*/

@Override

**public** **long** longValue() {

**long** res = (**long**) Math.*sqrt*(**this**.real\***this**.real + **this**.imag\***this**.imag);

**return** res;

}

}

**import** java.awt.geom.Rectangle2D.Double;

/\*\*

\* Класс, управляющий расчетом и начальным отображением фрактала "Горящий корабль"

\* **@author** niksh

\*

\*/

**public** **class** BurningShip **extends** FractalGenerator {

/\*\* Максимальное кол-во итераций при расчете \*/

**public** **static** **final** **int** ***MAX\_ITERATIONS*** = 2000;

/\*\* Размеры отображаемого участка (DIM\*DIM) \*/

**public** **static** **final** **double** ***REC\_DIMENSION*** = 4;

/\*\* x центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CORNER\_X*** = -2;

/\*\* y центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CORNER\_Y*** = -2.5;

/\*\*

\* Возвращает имя фрактала

\*/

**public** String toString() {

**return** "Горящий корабль";

}

/\*\*

\* Установка начального диапазона отображения фрактала

\*/

@Override

**public** **void** getInitialRange(Double range) {

range.height = ***REC\_DIMENSION***;

range.width = ***REC\_DIMENSION***;

range.x = ***REC\_CORNER\_X***;

range.y = ***REC\_CORNER\_Y***;

}

/\*\*

\* Итеративная формула расчета принадлежности точки к множеству "Горящий корабль"

\* **@param** x - x координата, приведенная к реальной части числа на комплексной плоскости

\* **@param** y - y координата, приведенная к комплексной части числа на комплексной плоскости

\* **@return** Кол-во итераций расчета до выхода за пределы множества, -1 если расчеты не вышли за пределы

\*/

@Override

**public** **int** numIterations(**double** x, **double** y) {

Complex coord = **new** Complex(x,y);

Complex zIter = **new** Complex();

Complex temp = **new** Complex();

**int** iter = 0;

zIter = zIter.add(coord);

**while** (zIter.absSqr() < 4 & iter < ***MAX\_ITERATIONS***) {

iter += 1;

temp.set(Math.*abs*(zIter.get('r')), Math.*abs*(zIter.get('i')));

zIter = temp.mul(temp).add(coord);

temp.zero();

}

**if** (iter == 2000) **return** -1;

**return** iter;

}

}

**import** java.awt.geom.Rectangle2D.Double;

/\*\*

\* Класс, управляющий расчетом и начальным отображением фрактала Мандельбара

\* **@author** niksh

\*

\*/

**public** **class** Tricorn **extends** FractalGenerator {

/\*\* Максимальное кол-во итераций при расчете \*/

**public** **static** **final** **int** ***MAX\_ITERATIONS*** = 2000;

/\*\* Размеры отображаемого участка (DIM\*DIM) \*/

**public** **static** **final** **double** ***REC\_DIMENSION*** = 4;

/\*\* x центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CORNER\_X*** = -2;

/\*\* y центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CORNER\_Y*** = -2;

/\*\*

\* Возвращает имя фрактала

\*/

**public** String toString() {

**return** "Множество Мандельбара";

}

/\*\*

\* Установка начального диапазона отображения фрактала

\*/

@Override

**public** **void** getInitialRange(Double range) {

range.height = ***REC\_DIMENSION***;

range.width = ***REC\_DIMENSION***;

range.x = ***REC\_CORNER\_X***;

range.y = ***REC\_CORNER\_Y***;

}

/\*\*

\* Итеративная формула расчета принадлежности точки к множеству Мандельбара

\* **@param** x - x координата, приведенная к реальной части числа на комплексной плоскости

\* **@param** y - y координата, приведенная к комплексной части числа на комплексной плоскости

\* **@return** Кол-во итераций расчета до выхода за пределы множества, -1 если расчеты не вышли за пределы

\*/

@Override

**public** **int** numIterations(**double** x, **double** y) {

Complex coord = **new** Complex(x,y);

Complex zIter = **new** Complex();

**int** iter = 0;

zIter = zIter.conjugate().mul(zIter.conjugate()).add(coord);

**while** (zIter.absSqr() < 4 & iter < ***MAX\_ITERATIONS***) {

iter += 1;

zIter = zIter.conjugate().mul(zIter.conjugate()).add(coord);

}

**if** (iter == 2000) **return** -1;

**return** iter;

}

}

# Результат работы программы

