Оглавление

Лабораторная работа №1: Основы синтаксиса Java

В данной лабораторной работе вы изучите основы синтаксиса Java с помощью нескольких простых задач программирования. Далее вы узнаете, как использовать компилятор Java и виртуальную машину Java для запуска программы. От вас потребуется решить следующие задачи:

Простые числа

Создайте программу, которая находит и выводит все простые числа меньше 100.

1. Создайте файл с именем Primes.java, в этом файле опишите следующий класс:

```
public class Primes {
    public static void main(String[] args) {
    }
}
```

Воспользовавшись данным классом, соберите и запустите программу. Так как в данной программе нет конкретной реализации, результата выполнения ее вы не увидите.

2. Внутри созданного класса, после метода main(), опишите функцию IsPrime (Int n), которая определяет, является ли аргумент простым числом или нет. Можно предположить, что входное значение n всегда будет больше 2. Полное описание функции будет выглядеть так:

```
public static boolean isPrime(int n)
{
}
```

Данный метод вы можете реализовать по вашему усмотрению, однако простой подход заключается в использовании цикла for. Данный цикл должен перебирать числа, начиная с 2 до (но не включая) n, проверяя существует ли какое-либо значение, делящееся на n без остатка. Для этого можно

использовать оператора остатка “%”. Например, 17%7 равняется 3, и 16%4 равно 0. Если какая-либо переменная полностью делится на аргумент, сработает оператор return false. Если же значение не делится на аргумент без остатка, то это простое число, и оператор покажет return true. (Оператор return в Java используется для возврата данных из функции, таким способом закрывается метод.)

3. После того, как этот участок будет реализован, приступайте к заполнению основного метода main() другим циклом, который перебирает числа в диапазоне от 2 до 100 включительно. Необходимо вывести на печать те значения, которые ваш помощник IsPrime () посчитал простыми.

4. После завершения вашей программы скомпилируйте и протестируйте её. Убедитесь, что результаты правильные. В интернете вы сможете найти списки простых чисел.

Кроме того, как видно из примера, не следует забывать об использовании комментариев: перед классом с его назначением и перед методом с его целью. Когда вы пишете программы, крайне важно писать подобные комментарии.

Палиндромы

Вторая программа, которую вам необходимо будет написать, показывает, является ли строка палиндромом.

1. Для этой программы, создайте класс с именем Palindrome в файле под названием Palindrome.java. На этот раз вы можете воспользоваться следующим кодом:

```
public class Palindrome {
   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++) {
         String s = args[i];
} }
}
```

Скомпилируйте и запустите эту программу в таком виде, результат работы не будет отображен.

2. Ваша первая задача состоит в том, чтобы создать метод, позволяющий полностью изменить символы в строке. Сигнатура (последовательность) метода должна быть следующей:

public static String reverseString(Strings)

Вы можете реализовать этот метод путем создания локальной переменной, которая начинается со строки "", а затем добавлять символы из входной строки в выходные данные, в обратном порядке. Используйте метод length(), который возвращает длину строки, и метод charAt(int index), который возвращает символ по указанному индексу. Индексы начинаются с 0 и увеличиваются на 1. Например:

```
String s = "pizzeria";
System.out.println(s.length());   //Выводим 8
System.out.println(s.charAt(5));  //Выводим r
```

Вы можете использовать оператор конкатенации (соединения) строк + или оператор +=, на ваше усмотрение.

3. После того, как вы применили метод reverseString (), создайте еще один метод public static boolean isPalindrome(String s). Этот метод должен перевернуть слово s, а затем сравнить с первоначальными данными. Используйте метод Equals (Object) для проверки значения равенства. Например:

```
String s1 = "hello";
String s2 = "Hello";
String s3 = "hello";
s1.equals(s2);  // Истина
s1.equals(s3);  // Ложь
```

Не используйте == для проверки равенства строк. Этим занимается другой тест в Java, который будет рассмотрен далее.

4. Скомпилируйте и протестируйте программу! На этот раз входными данными будут аргументы командной строки, например:

java Palindrome madam racecar apple kayak song noon

Ваша программа должна вывести ответ, является ли каждое слово палиндром.

5. Убедитесь в наличии комментариев, где указаны назначения вашей программы и используемых методов.

Лабораторная работа №2: Основы объектно-ориентированного программирования

Java позволяет использовать объекты. В данной лабораторной работе необходимо использовать классы по одному на файл, чтобы описать, как эти объекты работают. Вот код для простого класса, который представляет двумерную точку:

```
/ **
* двумерный класс точки.
**/
public class Point2d {
/ ** координата X **/
private double xCoord;
/ **  координата Y **/
private double yCoord;
/ ** Конструктор инициализации **/
public Point2d ( double x,  double y) {
xCoord = x;
yCoord = y;
}
/ ** Конструктор по умолчанию. **/
public Point2d () {
//Вызовите конструктор с двумя параметрами и определите источник.
this(0, 0);
}
/ ** Возвращение координаты X **/
public double getX () {
return xCoord;
}
/ ** Возвращение координаты Y **/
```

```
public double getY () {
return yCoord;
}
/ ** Установка значения координаты X. **/
public void setX ( double val) {
xCoord = val;
}
/ **  Установка значения координаты Y. **/
public void  setY ( double val) {
yCoord = val;
}
}
```

Сохраните данный код в файле с именем Point2d.java, согласно требованиям Java к именам классов и именам файлов.

Экземпляр класса можно также создать, вызвав любой из реализованных конструкторов, например:

```
Point2d myPoint = new Point2d ();//создает точку (0,0)
Point2d myOtherPoint = new Point2d (5,3);//создает точку (5,3)
Point2d aThirdPoint = new Point2d ();
```

Примечание: myPoint != aThirdPoint, несмотря на то, что их значения равны. Объясняется это тем, что оператор равенства == (и его инверсия, оператор неравенства !=) сравнивает ссылки на объекты. Другими словами, == оператор вернет true, если две ссылки указывают на один и тот же объект. В данном случае myPoint и aThirdPoint ссылаются на разные объекты класса Point2d, поэтому операция сравнения myPoint == aThirdPoint вернет false, несмотря на то, что их значения те же!

Для того, чтобы проверить равны ли сами значения, а не ссылки, необходимо создать метод в классе Point2d, который будет сравнивать значения соответствующих полей объектов класса Point2d.

Рекомендации при программировании

Стиль программирования является неотъемлемой частью при создании программного обеспечения. При создании приложений большая часть времени уходит на отладку программы. Читаемый код и использование комментариев в нем позволяют сэкономить время при отладке программы.

CS11 - хорошая возможность изучить и использовать хороший стиль кодирования. Прежде чем приступить к заданию, изучите инструкции CS11 Java Style Guidelines. На сайте clusterCS есть программа проверки стиля, которая проанализирует ваш код и проинформирует о проблемах.

Ваши задачи:

1. Создайте новый класс Point3d для представления точек в трехмерном Евклидовом пространстве. Необходимо реализовать:

- создание нового объекта Point3d с тремя значениями с плавающей точкой (double);
- создание нового объекта Point3d со значениями (0.0, 0.0, 0.0) по умолчанию,
- возможность получения и изменения всех трех значений по отдельности;
- метод для сравнения значений двух объектов Point3d.

Нельзя предоставлять непосредственный доступ к внутренним элементам объекта класса Point3d .

2. Добавьте новый метод distanceTo, который в качестве параметра принимает другой объект Point3d, вычисляет расстояние между двумя точками с точность двух знаков после запятой и возвращает полученное значение.

3. Создайте другой класс под названием Lab1, который будет содержать статический метод main. Помните, что метод main должен быть общедоступным (public) с возвращаемым значением void, а в качестве аргумента должен принимать строку (String). Этот класс должен иметь следующую функциональность:

- Ввод координат трех точек, находящихся в трехмерном пространстве. Создание трех объектов типа Point3d на основании полученных данных. (Предполагается, что пользователь вводит корректные данные.)
- Создайте второй статический метод computeArea, который принимает три объекта типа Point3d и вычисляет площадь треугольника, образованного этими точками. (Вы можете использовать формулу Герона.) Верните получившееся значение площади в формате типа double.
- На основе полученных данных и с использованием реализованного алгоритма посчитайте площадь и выведите полученное значение пользователю.

Перед вызовом метода computeArea проверьте на равенство значений всех трех объектов Point3d. Если одна из точек равна другой, то выведите соответствующее сообщение пользователю и не вычисляйте площадь.

4. Скомпилируйте оба исходных файла вместе:

javac Point3d.java Lab1.java

и затем запустите программу Lab1, тестируя ее с несколькими образцами треугольников.

Лабораторная работа №3: Алгоритм A* («A star»)

Если вы когда-нибудь играли в какую-либо игру на компьютере на основе карт, то вы, вероятно, сталкивались с органами компьютерного управления, которые умеют самостоятельно рассчитывать путь из пункта А в пункт Б. На самом деле это обычная распространенная проблема как в играх, так и в других видах программного обеспечения - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Один очень широко используемый алгоритм для такого рода проблемы называют A* (произносится "A-star"). Он является наиболее эффективным алгоритмом для поиска пути в компьютерной программе. Концепция алгоритма довольно проста, начиная с исходного местоположения, алгоритм постепенно строит путь от исходной точки до места назначения, используя наикратчайший путь, чтобы сделать следующий шаг. Это гарантирует, что полный путь будет также оптимальным.

Вам не придется реализовывать алгоритм A*; это было уже сделано за вас. На самом деле существует даже пользовательский интерфейс для того, чтобы экспериментировать с этим алгоритмом:

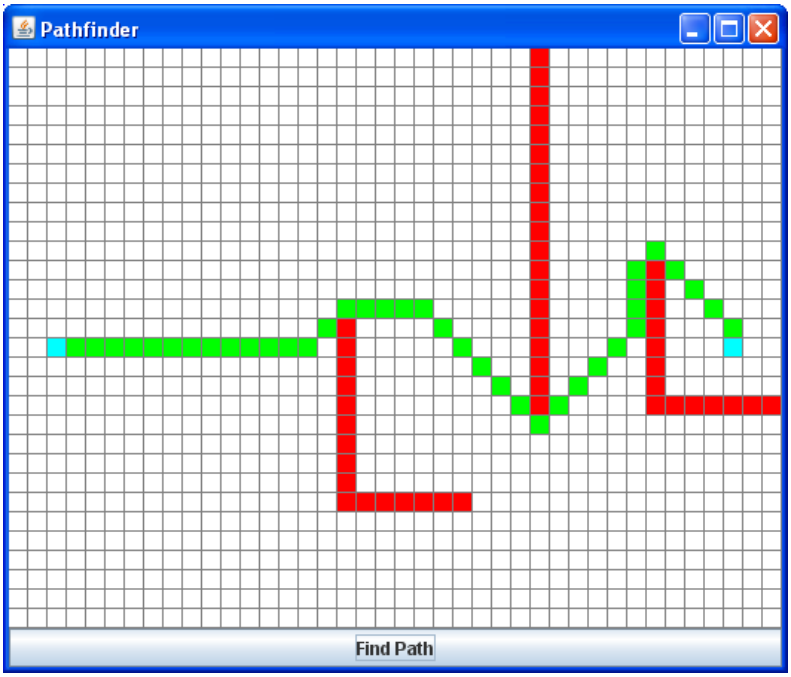

Рисунок 5.1. Пользовательский интерфейс для работы с алгоритмом A*.

Вы можете щелкнуть по различным квадратам, чтобы сделать из них в барьеры (красные) или проходимые клетки (белые). Синие клетки обозначают начало и конец пути. Нажав на кнопку "Find Path", программа вычислит путь, используя алгоритм A*, и затем отобразит его зеленым цветом. В случае если путь не будет найден, то программа просто не отобразит путь.

Алгоритм A* содержит много информации для отслеживания, и классы Java идеально подходят для такого рода задач. Существует два основных вида информации для организации управления алгоритмом A*:

- **Локации** – это наборы координат конкретных клеток на двумерной карте. Алгоритм A* должен иметь возможность ссылаться на определенные места на карте.
- **Вершины** - это отдельные шаги на пути, которые генерирует алгоритм A*. Например, продемонстрированные выше зеленые ячейки - это последовательность вершин на карте.

Каждая путевая точка владеет следующей информацией:

- Расположение ячейки для вершины.
- Ссылка на предыдущую вершину маршрута. Конечный путь - это последовательность вершин от пункта назначения до исходной точки.
- Фактическая стоимость пути от начального местоположения до текущей вершины по определенному пути.
- Эвристическая оценка (приблизительная оценка) остаточной стоимости пути от текущей вершины до конечной цели.

Так как алгоритм A* строит свой путь, он должен иметь два основных набора вершин:

- Первый набор хранит "открытые вершины" или вершины, которые все еще должны учитываться алгоритмом A*.
- Второй набор хранит "закрытые вершины" или вершины, которые уже были учтены алгоритмом A* и их не нужно будет больше рассматривать.

Каждая итерация алгоритма A* довольно проста: найти вершину с наименьшей стоимостью пути из набора открытых вершин, сделать шаг в каждом направлении от этой вершины для создания новых открытых вершин, а затем переместить вершины из открытого набора в закрытый. Это повторяется до тех пор, пока не будет достигнута конечная вершина! Если во время этого процесса заканчиваются открытые вершины, то пути от начальной вершины до конечной вершины нет.

Данная обработка в первую очередь зависит от расположения вершин, поэтому очень полезно сохранять путевые точки как отображение местоположений до соответствующих вершин. Таким образом, вы будете использовать хранилище java.util.HashMap для каждого из этих наборов с объектами Location в качестве ключей, и объектами Waypoint в качестве значений.

Прежде чем начать

Прежде чем начать, вам необходимо скачать исходные файлы для данной лабораторной работы:

- Map2D.java - представляет карту, по которой перемещается алгоритм A*, включая в себя информацию о проходимости ячеек
- Location.java - этот класс представляет координаты конкретной ячейки на карте
- Waypoint.java - представляет отдельные вершины в сгенерированном пути
- AStarPathfinder.java - этот класс реализует алгоритм поиска пути A* в виде статического метода.
- AStarState.java - этот класс хранит набор открытых и закрытых вершин, и предоставляет основные операции, необходимые для функционирования алгоритма поиска A*.
- AStarApp.java - простое Swing-приложение, которое обеспечивает редактируемый вид 2D-карты, и запускает поиск пути по запросу
- JMapCell.java - это Swing -компонент, который используется для отображения состояния ячеек на карте

Обратите внимание, что приложение будет успешно компилироваться в том виде, какое оно есть, но функция поиска пути не будет работать, пока вы не завершите задание. Единственные классы, которые вы должны изменить это Location и AStarState. Все остальное - это код платформы, которая позволяет редактировать карту и показывать путь, который генерирует алгоритм. (Не рекомендуется редактировать исходный код других файлов)

Локации

Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java. Поскольку вы будете использовать контейнеры для хеширования для выполнения данного задания, то для этого необходимо:

- Обеспечить реализацию метода equals ().
- Обеспечить реализацию метода hashcode().

Добавьте реализацию каждого из этих методов в класс Location, следуя шаблонам в классе. После этого вы можете использовать класс Location в качестве ключевого типа в контейнерах хеширования, таких как HashSet и HashMap.

Состояния A*

После того, как класс Location готов к использованию, вы можете завершить реализацию класса AStarState. Это класс, который поддерживает наборы открытых и закрытых вершин, поэтому он действительно обеспечивает основную функциональность для реализации алгоритма A*.

Как упоминалось ранее, состояние A* состоит из двух наборов вершин, один из открытых вершин и другой из закрытых. Чтобы упростить алгоритм, вершины будут храниться в хэш-карте, где местоположение вершин является ключом, а сами вершины являются значениями. Таким образом, у вас будет такой тип:

HashMap<Location, Waypoint>

(Очевидный вывод из всего этого заключается в том, что с каждым местоположением на карте может быть связана только одна вершина.)

Добавьте два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". Кроме того, не забудьте инициализировать каждое из этих полей для ссылки на новую пустую коллекцию.

После создания и инициализации полей, вы должны реализовать следующие методы в классе AStarState:

1) public int numOpenWaypoints()

Этот метод возвращает количество точек в наборе открытых вершин.

2) public Waypoint getMinOpenWaypoint()

Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью. Если в "открытом" наборе нет вершин, функция возвращает NULL.

Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

3) public boolean addOpenWaypoint(Waypoint newWP)

Это самый сложный метод в классе состояний A*. Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

- Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину.
- Если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей. (Убедитесь, что используете не общую стоимость.) Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую

Как вы могли заметить, что в таком случае вам потребуется извлечь существующую вершину из «открытого набора», если таковая имеется. Данный шаг довольно прост - замените предыдущую точку на новую, используя метод HashMap.put(), который заменит старое значение на новое. Пусть данный метод вернет значение true, если новая вершина была успешно добавлена в набор, и false в противном случае.

4) public boolean isLocationClosed(Location loc)

Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном

случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

5) public void closeWaypoint(Location loc)

Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины.

Процесс должен быть простым:

- Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».
- Добавьте вершину, которую вы удалили, в набор закрытых вершин. Ключом должно являться местоположение точки.

Компиляция и тестирование

Как только вы реализуете вышеуказанную функциональность, запустите программу поиска пути, чтобы проверит правильность ее выполнения. Если вы реализовали все правильно, то у вас не должно возникнуть проблем при создании препятствий и последующим поиском путей вокруг них.

Скомпилируйте и запустите программу также, как и всегда:

javac *.java

java AStarApp

Лабораторная работа № 4: Рисование фракталов

В следующих нескольких лабораторных работ вы создадите небольшое JAVA-приложение, которое сможет рисовать фракталы. Если вы никогда не играли с фракталами раньше, вы будете удивлены тем, как просто можно создать красивые изображения. Это будет сделано с помощью фреймворка Swing и Java API, который позволяет создавать графические пользовательские интерфейсы.

Начальная версия приложения будет довольно проста, но в следующих лабораторных работах будут добавлены некоторые полезные функции такие как, как сохранение сгенерированных изображений, возможность переключения между различными видами фракталов. И графический интерфейс (GUI), и механизм для поддержки различных фракталов будут зависеть от иерархий классов.

Вот простой пример графического интерфейса в его начальном состоянии:

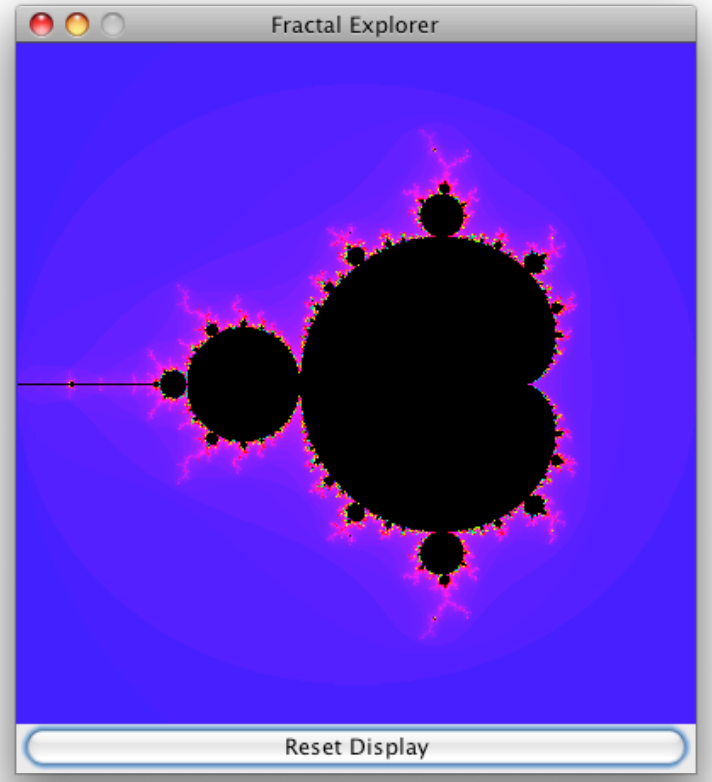

Рисунок 5.2. Пример графического интерфейса

И, вот некоторые интересные области фрактала: слоны и морские коньки!

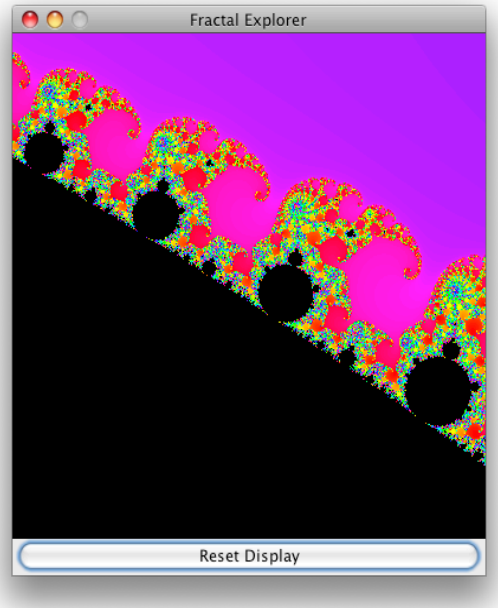

Рисунок 5.3. Фрактал «Слоны»

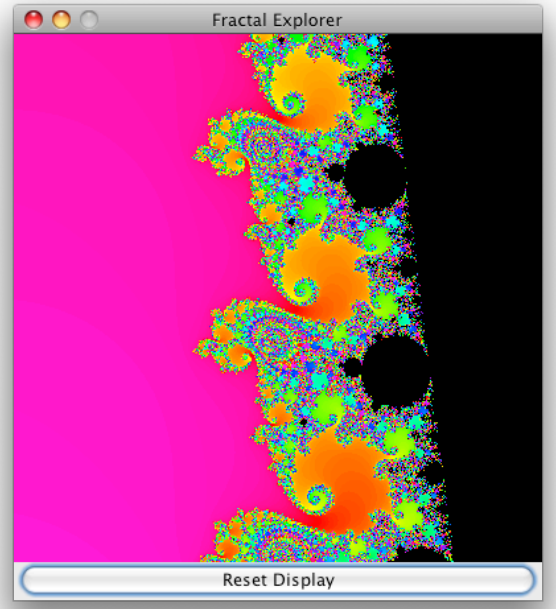

Рисунок 5.4. Фрактал «Морские коньки»

Создание пользовательского интерфейса

Прежде чем рисовать фракталы, необходимо создать графический виджет, который будет их отображать. Swing не предоставляет такой компонент, но его очень просто создать самостоятельно. Обратите внимание на то, что в этой лабораторной работе мы будем использовать широкий спектр классов Java AWT и Swing, детали которых здесь будут не раскрыты. Для более подробной информации вы можете воспользоваться онлайн-документами по API Java. Просто перейдите к пакету данного класса Java, выберите сам класс, а затем прочтите подробную информацию о том, как использовать класс.

- Создайте класс JImageDisplay, производный от javax.swing.JComponent. Класса должен иметь одно поле с типом доступа private, экземпляп java.awt.image.BufferedImage. Класс BufferedImage управляет изображением, содержимое которого можно записать.

- Конструктор JImageDisplay должен принимать целочисленные значения ширины и высоты, и инициализировать объект BufferedImage новым изображением с этой шириной и высотой, и типом изображения TYPE_INT_RGB. Тип определяет, как цвета каждого пикселя будут

представлены в изображении; значение TYPE_INT_RGB обозначает, что красные, зеленые и синие компоненты имеют по 8 битов, представленные в формате int в указанном порядке.

Конструктор также должен вызвать метод setPreferredSize() родительского класса метод с указанной шириной и высотой. (Вы должны будете передать эти значения в объект java.awt.Dimension) Таким образом, когда ваш компонент будет включен в пользовательский интерфейс, он отобразит на экране все изображение.

- Пользовательские компоненты Swing должны предоставлять свой собственный код для отрисовки, переопределяя защищенный метод JComponent paintComponent (Graphics g). Так как наш компонент просто выводит на экран данные изображения, реализация будет очень проста! Во-первых, нужно всегда вызывать метод суперкласса paintComponent (g) так, чтобы объекты отображались правильно. После вызова версии суперкласса, вы можете нарисовать изображение в компоненте, используя следующую операцию:

g.drawImage (image, 0, 0, image.getWidth(), image.getHeight(), null);

(Мы передаем значение null для ImageObserver, поскольку данная функциональность не требуется.)

- Вы также должны создать два метода с доступом public для записи данных в изображение: метод clearImage (), который устанавливает все пиксели изображения в черный цвет (значение RGB 0), и метод drawPixel (int x, int y, int rgbColor), который устанавливает пиксель в определенный цвет. Оба метода будут необходимы для использования в методе setRGB () класса BufferedImage.

Не забывайте про комментарии!

Вычисления фрактала Мандельброта

Следующая ваша задача: написать код для вычисления фрактала Мандельброта.

Для создания фракталов используйте следующий исходный файл FractalGenerator.java, от которого будут унаследованы все ваши фрактальные

генераторы. Как вы могли заметить данный файл предоставляет также некоторые полезные операции для перевода из экранных координат в систему координат вычисляемого фрактала.

Виды фракталов, с которыми нужно будет работать, вычисляются в комплексном виде и включают в себя простые математические функции, которые выполняются многократно, пока не выполнится определенное условие. Функция для фрактала Мандельброта имеет вид: $z_n = z_{n-1}^2 + c$, где все значения — это комплексные числа, $z_0 = 0$, и c - определенная точка фрактала, которую мы отображаем на экране. Вычисления повторяются до тех пор, пока $|z| > 2$ (в данной ситуации точка находится не во множестве Мандельброта), или пока число итераций не достигнет максимального значения, например, 2000 (в этом случае делается предположение, что точка находится в наборе).

Процесс построения фрактала Мандельброта прост: необходимо перебрать все пиксели изображения, рассчитать количество итераций для соответствующей координаты, и затем установить пиксель в цвет, основанный на количестве рассчитанных итераций. Все это будет сделано позже, на данном этапе необходимо реализовать приведенные выше вычисления.

- Создайте подкласс FractalGenerator с именем Mandelbrot. в нем вам необходимо будет обеспечить только два метода: getInitialRange() и numIterations().
- getInitialRange (Rectangle2D.Double) - метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости для конкретного фрактала. Обратите внимание на то, что методу в качестве аргумента передается прямоугольный объект, и метод должен изменить поля прямоугольника для отображения правильного начального диапазона для фрактала. (Пример можно увидеть в методе FractalGenerator.recenterAndZoomRange().) В классе Mandelbrot этот метод должен установить начальный диапазон в *(-2 - 1.5i) - (1 + 1.5i)*. Т.е. значения x и y будут равны -2 и -1.5 соответственно, а ширина и высота будут равны 3.

- Метод numIterations(double, double) реализует итеративную функцию для фрактала Мандельброта. Константу с максимальным количеством итераций можно определить следующим образом:

public static final int MAX_ITERATIONS = 2000;

Затем вы сможете ссылаться на эту переменную в вашей реализации.

Обратите внимание на то, что у Java нет подходящего типа данных для комплексных чисел, поэтому необходимо будет реализовать итеративную функцию, используя отдельные переменные для действительной и мнимой частей. (Вы можете реализовать отдельный класс для комплексных чисел.) Ваш алгоритм должен обладать быстродействием, например, не стоит сравнивать $|z|$ с 2; сравните $|z|^2$ с 2^2 для того, чтобы избежать сложных и медленных вычислений квадратного корня. Также не стоит использовать метод Math.pow () для вычисления небольших степеней, лучше перемножьте значение, иначе ваш быстродействие вашего кода сильно упадет.

В случае, если алгоритм дошел до значения MAX_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.

Ваши задачи

Создайте класс FractalExplorer, который позволит вам исследовать различные области фрактала, путем его создания, отображения через графический интерфейс Swing и обработки событий, вызванных взаимодействием приложения с пользователем.

Как видно их приведенных выше изображений пользовательского интерфейса, FractalExplorer очень прост, он состоит из JFrame, который в свою очередь содержит объект JImageDisplay, который отображает фрактал,и объект JButton для сброса изображения, необходимый для отображения целого фрактала. Данный макет можно создать, установив для фрейма BorderLayout, затем поместив отображение в центр макета и кнопку сброса в "южной" части макета.

- Класс FractalExplorer должен отслеживать несколько важных полей для состояния программы:

1) Целое число «размер экрана», которое является шириной и высотой отображения в пикселях. (Отображение фрактала будет квадратным.)

2) Ссылка JImageDisplay, для обновления отображения в разных методах в процессе вычисления фрактала.

3) Объект FractalGenerator. Будет использоваться ссылка на базовый класс для отображения других видов фракталов в будущем.

4) Объект Rectangle2D.Double, указывающий диапазона комплексной плоскости, которая выводится на экран.

Все вышеприведенные поля будут иметь тип доступа private.

- У класса должен быть конструктор, который принимает значение размера отображения в качестве аргумента, затем сохраняет это значение в соответствующем поле, а также инициализирует объекты диапазона и фрактального генератора. Данный конструктор не должен устанавливать какие-либо компоненты Swing; они будут установлены в следующем методе.

- Создайте метод createAndShowGUI (), который инициализирует графический интерфейс Swing: JFrame, содержащий объект JimageDisplay, и кнопку для сброса отображения. Используйте java.awt.BorderLayout для содержимого окна; добавьте объект отображения изображения в позицию BorderLayout.CENTER и кнопку в позицию BorderLayout.SOUTH.

Вам необходимо дать окну подходящий заголовок и обеспечить операцию закрытия окна по умолчанию (см. метод JFrame.setDefaultCloseOperation ()).

После того, как компоненты пользовательского интерфейса инициализированы и размещены, добавьте следующую последовательность операций:

```
frame.pack ();
frame.setVisible (true);
```

frame.setResizable (false);

Данные операции правильно разметят содержимое окна, сделают его видимым (окна первоначально не отображаются при их создании для того, чтобы можно было сконфигурировать их прежде, чем выводить на экран), и затем запретят изменение размеров окна.

- Реализуйте вспомогательный метод с типом доступа private для вывода на экран фрактала, можете дать ему имя drawFractal (). Этот метод должен циклически проходить через каждый пиксель в отображении (т.е. значения x и y будут меняться от 0 до размера отображения), и сделайте следующее:

  - Вычислите количество итераций для соответствующих координат в области отображения фрактала. Вы можете определить координаты с плавающей точкой для определенного набора координат пикселей, используя вспомогательный метод FractalGenerator.getCoord (); например, чтобы получить координату x, соответствующую координате пикселя X, сделайте следующее:

```
//x - пиксельная координата; xCoord - координата в пространстве фрактала
double xCoord = FractalGenerator.getCoord (range.x, range.x + range.width, displaySize, x);
```

  - Если число итераций равно -1 (т.е. точка не выходит за границы, установите пиксель в черный цвет (для rgb значение 0). Иначе выберите значение цвета, основанное на количестве итераций. Можно также для этого использовать цветовое пространство HSV: поскольку значение цвета варьируется от 0 до 1, получается плавная последовательность цветов от красного к желтому, зеленому, синему, фиолетовому и затем обратно к красному! Для этого вы можете использовать следующий фрагмент:

```
float hue = 0.7f + (float) numIters / 200f;
int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);
```

Если вы придумали другой способ отображения пикселей в зависимости от количества итераций, попробуйте реализовать его!

- Отображение необходимо обновлять в соответствии с цветом для каждого пикселя.
- После того, как вы закончили отрисовывать все пиксели, вам необходимо обновить JimageDisplay в соответствии с текущим изображением. Для этого вызовите функцию repaint() для компонента. В случае, если вы не воспользуетесь данным методом, изображение на экране не будет обновляться!

- Создайте внутренний класс для обработки событий java.awt.event.ActionListener от кнопки сброса. Обработчик должен сбросить диапазон к начальному, определенному генератором, а затем перерисовать фрактал.

После того, как вы создали этот класс, обновите метод createAndShowGUI ().

- Создайте другой внутренний класс для обработки событий java.awt.event.MouseListener с дисплея. Вам необходимо обработать события от мыши, поэтому вы должны унаследовать этот внутренний класс от класса MouseAdapterAWT. При получении события о щелчке мышью, класс должен отобразить пиксельные кооринаты щелчка в область фрактала, а затем вызвать метод генератора recenterAndZoomRange() с координатами, по которым щелкнули, и масштабом 0.5. Таким образом, нажимая на какое-либо место на фрактальном отображении, вы увеличиваете его!

Не забывайте перерисовывать фрактал после того, как вы меняете область фрактала.

Далее обновите метод createAndShowGUI (), чтобы зарегистрировать экземпляр этого обработчика в компоненте фрактального отображения.

- В заключении, вам необходимо создать статический метод main() для FractalExplorer так, чтобы можно было его запустить. В main необходимо будет сделать:

- Инициализировать новый экземпляр класса FractalExplorer с размером отображения 800.
- Вызовите метод createAndShowGUI () класса FractalExplorer.
- Вызовите метод drawFractal() класса FractalExplorer для отображения начального представления.

После выполнения приведенных выше действий, вы сможете детально рассмотреть фрактал Мандельброта. Если вы увеличите масштаб, то вы можете столкнуться с двумя проблемами:

- Во-первых, вы сможете заметить, что в конечном итоге уровень детализации заканчивается; это вызвано тем, что в таком случае необходимо более 2000 итераций для поиска точки во множестве Мандельброта! Можно увеличить максимальное количество итераций, но это приведет к замедлению работы алгоритма.
- Во-вторых, при сильном увеличении масштаба, вы столкнетесь с пиксельным выводом отображения! Это вызвано тем, что вы работаете в пределе того, что могут предоставить значения с плавающей запятой с двойной точностью.

При рисовании фрактала экран ненадолго зависает. Следующая лабораторная работа будет направлена на решение данной проблемы.

Лабораторная работа №5. Выбор и сохранение фракталов

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. Ниже приведен скриншот, где продемонстрировано, как будет выглядеть новая программа

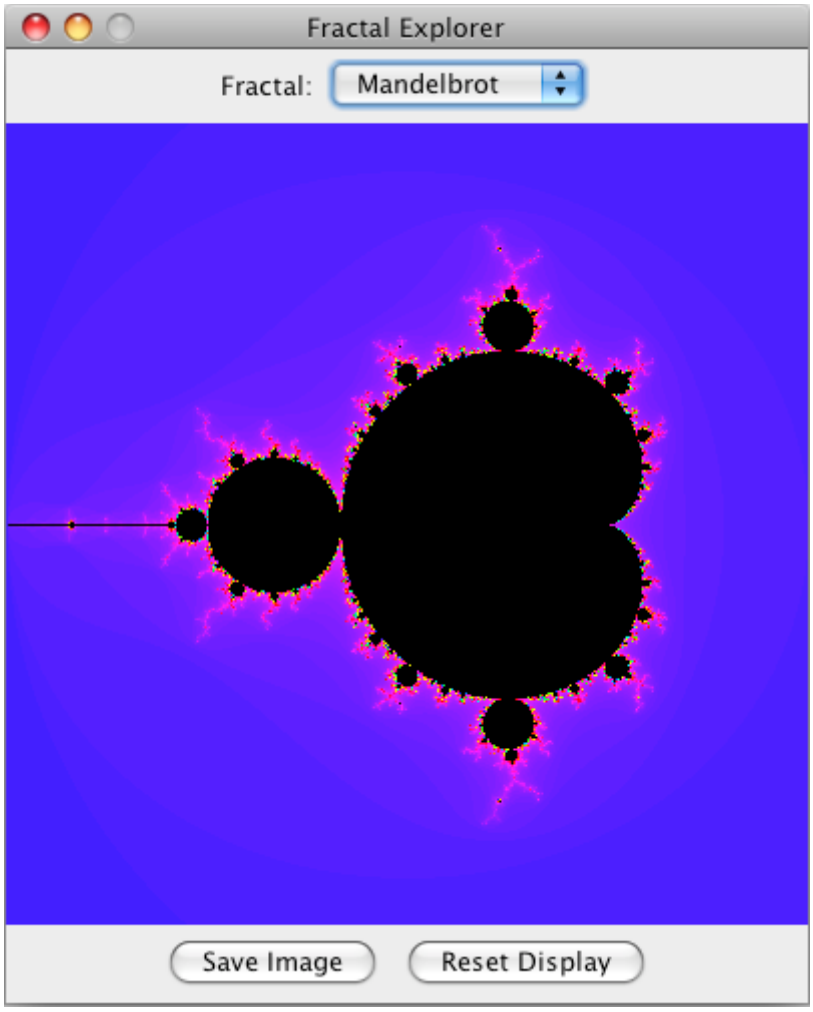

Рисунок 5.5. Графический интерфейс нового приложения

Верхняя панель генератора фракталов включает в себя 2 виджета, позволяющих пользователю выбирать фрактал, а нижняя панель включает в себя кнопку "Save Image", которая сохраняет текущее изображения фрактала.

Так как теперь будет несколько источников событий (action-event sources), вы сможете попрактиковаться в обработке всех источников с использованием одного метода ActionListener в вашем классе.

Поддержка нескольких фракталов

Так как в реализацию была введена абстракция FractalGenerator, добавление нескольких фракталов не будет проблемой. В данной лабораторной работе вы добавите поддержку нескольких фракталов, и пользователь сможет выбирать между ними, используя *combo-box*. Программный интерфейс Swing (Swing API) предоставляет *combo-box* через класс javax.swing.JComboBox, а также запускает ActionEvents при выборе новогоэлемента. Необходимо сделать:

- Создать 2 новые реализации FractalGenerator

Первым будет фрактал tricorn, который должен находиться в файле Tricorn.java. Для этого нужно создать подкласс FractalGenerator и реализация будет почти идентична фракталу Мандельброта, кроме двух изменений. Вы даже можете скопировать исходный код фрактала Мандельберта и просто внести следующие изменения:

- Уравнение имеет вид $z_n = z_{n\text{-}1}^2 + c$. Единственное отличие только в том, что используется комплексное сопряжение $z_{n\text{-}1}$ на каждой итерации.
- Начальный диапазон для трехцветного фрактала должен быть от *(-2, -2)* до *(2, 2)*.

Второй фрактал, который необходимо реализовать - это фрактал «Burning Ship», который в реальности не похож на пылающий корабль. Данный фрактал имеет следующие свойства:

- Уравнение имеет вид $z_n = (|\mathrm{Re}(z_{n\text{-}1})| + i\ |\mathrm{Im}(z_{n\text{-}1})|)^2 + c$. Другими словами, вы берете абсолютное значение каждого компонента $z_{n\text{-}1}$ на каждой итерации.
- Начальный диапазон для данного фрактала должен быть от *(-2, -2.5)* до *(2, 1.5)*.

- Combo-boxe в Swing может управлять коллекцией объектов, но объекты должны предоставлять метод toString(). Убедитесь, что в каждой реализации фракталов tcnm метод toString(), который возвращает имя, например «Mandelbrot», «Tricorn» и «Burning Ship».

- Настроить JComboBox в вашем пользовательском интерфейсе можно с использованием конструктора без параметров, а затем использовать метод addItem(Object) для того, чтобы добавить реализации вашего генератора фракталов. Как указывалось в предыдущем шаге, выпадающий список будет использовать метод toString () в ваших реализациях для отображения генераторов в выпадающем списке.

Необходимо будет также добавить объект label в разрабатываемый пользовательский интерфейс перед выпадающим списком, в качестве пояснения к выпадающему списку. Это можно сделать, создав новый объект Jpanel и добавив в него объекты JLabel и JcomboBox, а затем разместить панель на позиции NORTH на вашем макете окна.

И наконец, необходимо добавить поддержку выпадающего списка в реализацию ActionListener. В случае, если событие поступило от выпадающего списка, вы можете извлечь выбранный элемент из виджета и установить его в качестве текущего генератора фракталов. (Используйте метод getSelectedItem()) При этом не забудьте сбросить начальный диапазон и перерисовать фрактал!

Ниже приведены изображения фракталов «Tricorn» и «Burning Ship» для проверки правильности работы алгоритма

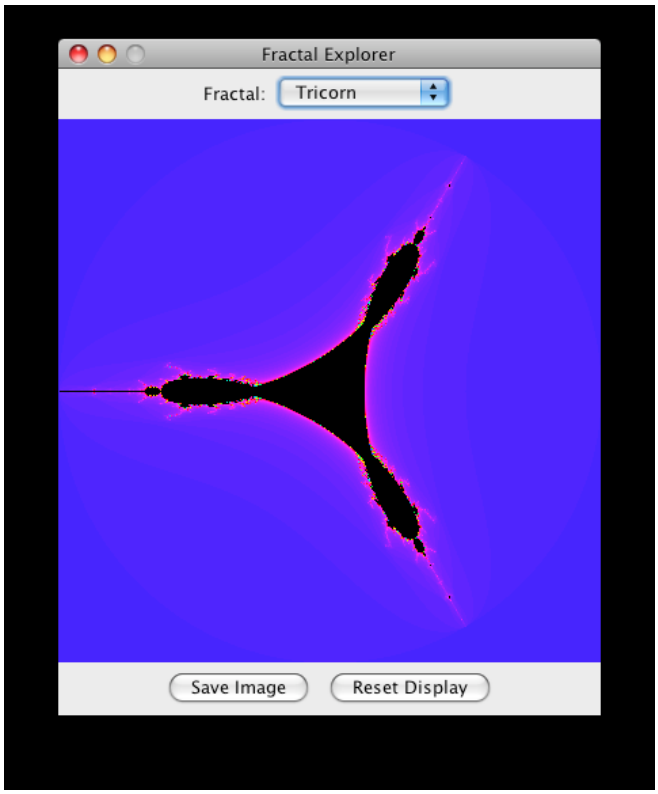

Рисунок 5.6. Фрактал «Tricorn»

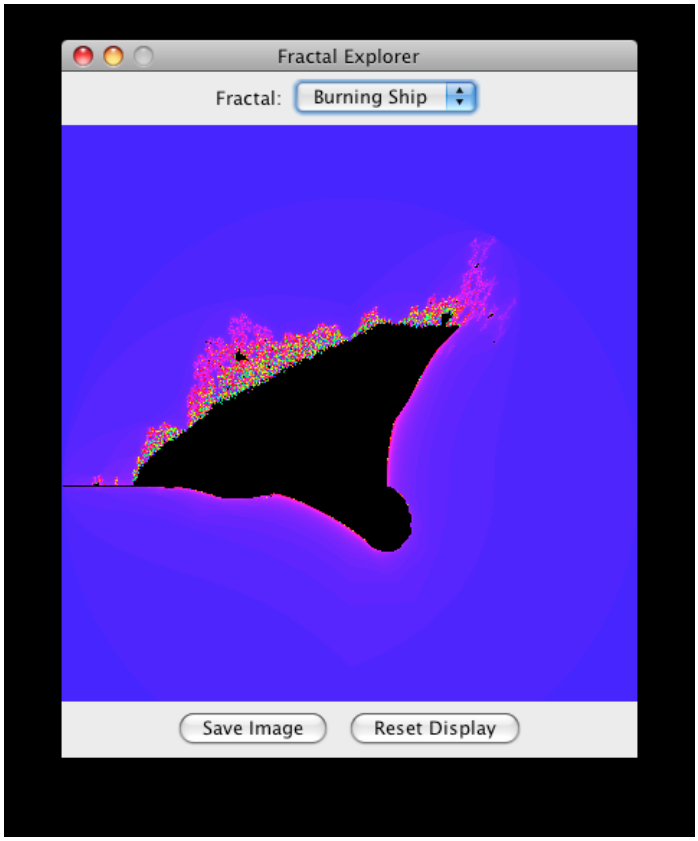

Рисунок 5.7. Фрактал «Burning Ship»

Сохранения изображения Фрактала

Следующая ваша задача - сохранение текущего изображения фрактала на диск. Java API предоставляет несколько инструментов для реализации данной задачи.

- Во-первых, вам нужно добавить кнопку «Save Image» в ваше окно. Для этого вы можете добавить обе кнопки «Save Image» и «Reset» в новую Jpanel, а затем разместить эту панель в SOUTH части окна.

События от кнопки «Save Image» также должны обрабатываться реализацией ActionListener. Назначьте кнопкам «Save Image» и «Reset» свои значения команд (например, «save» и «reset») для того, чтобы обработчик событий мог отличить события от этих двух разных кнопок.

- В обработчике кнопки «Save Image» вам необходимо реализовать возможность указания пользователем, в какой файл он будет сохранять изображение. Это можно сделать с помощью класса javax.swing.JFileChooser. Указанный класс предоставляет метод showSaveDialog(), который открывает диалоговое окно «Save file», позволяя тем самым пользователю выбрать директорию для сохранения. Метод принимает графический компонент, который является родительским элементом для диалогового окна с выбором файла, что позволяет центрированию окна с выбором относительно его родителя. В качестве родителя используйте окно приложения.

Как вы могли заметить, данный метод возвращает значение типа int, которое указывает результат операции выбора файла. Если метод возвращает значение JfileChooser.APPROVE_OPTION, тогда можно продолжить операцию сохранения файлов, в противном случае, пользователь отменил операцию, поэтому закончите данную обработку события без сохранения. Если пользователь выбрал директорию для сохранения файла, вы можете ее узнать, используя метод getSelectedFile(), который возвращает объект типа File.

- Также необходимо настроить средство выбора файлов, чтобы сохранять изображения только в формате PNG, на данном этапе вы будете работать только с данным форматом. вы сможете это настроить с помощью

javax.swing.filechooser.FileNameExtensionFilter, как это продемонстрировано ниже:

```
JFileChooser chooser = new JfileChooser();
FileFilter filter = new FileNameExtensionFilter("PNG Images", "png");
chooser.setFileFilter(filter);
chooser.setAcceptAllFileFilterUsed(false);
```

Последняя строка гарантирует, что средство выбора не разрешит пользователю использование отличных от png форматов.

- Если пользователь успешно выбрал файл, следующим шагом является сохранения изображения фрактала на диск! Для данного рода задач Java включает в себя необходимую функциональность. Класс javax.imageio.ImageIO обеспечивает простые операции загрузки и сохранения изображения. Вы можете использовать метод write(RenderedImage im, String formatName, File output). Параметр formatName будет содержать значение «png”. Тип «RenderedImage» - это просто экземпляр BufferedImage из вашего компонента JimageDisplay. (Используйте для него тип доступа public)

Метод write() может вызвать исключение, поэтому вам необходимо заключить этот вызов в блок try/catch и обработать возможную ошибку. Блок catch должен проинформировать пользователя об ошибке через диалоговое окно. Swing предоставляет класс javax.swing.JoptionPane для того, чтобы упростить процесс создания информационных диалоговых окон или окон, где нужно выбрать да/нет. Для этого вы можете использовать статический метод JoptionPane.showMessageDialog(Component parent, Object message, String title, int messageType), где messageType у вас будет JOptionPane.ERROR_MESSAGE. В сообщении об ошибке вы можете использовать возвращаемое значение метода getMessage(), а заголовком окна может быть, например, «Cannot Save Image». Родительским компонентом будет окно для того, чтобы диалоговое окно с сообщением об ошибке выводилось относительно центра окна.

После того, как вы закончите реализацию этих функций, запустите. Теперь вы сможете исследовать различные фракталы, а также вы сможете

Type your text

сохранять их на диск. Вы также можете проверить приложение на вывод сообщений об ошибках, попробуйте сохранить изображение в файл, который уже существует, но доступен только для чтения. Или вы можете попробовать сохранить файл с именем, которое является каталогом в целевой папке.