ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

Московский технический университет связи и информатики

Кафедра математической кибернетики и информационных технологий

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

по дисциплине

Технологии программирования

(для направлений 09.03.01, 27.03.04)

Москва 2017

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

по дисциплине

Технологии программирования

(для студентов направлений подготовки 09.03.01, 27.03.04)

Составители: М.Г. Городничев, к.т.н., доцент

М.М. Волков, ст. преподаватель

Издание утверждено советом факультета ИТ.
Протокол № 11 от 23.05.2017 г.

Рецензент: А.Г. Таташев, д.ф.-м.н., профессор

Задание № 0: Java-Разминка

Вы начнете изучать основы синтаксиса Java с помощью нескольких простых задач программирования. Далее вы сможете узнать, как использовать компилятор Java и виртуальную машину Java для запуска программы. От вас потребуется решить следующие задачи:

Простые числа

Простые числа не раз занимали видное место в заданиях по программированию, вот и сейчас без них не обойтись! Для первой задачи вы будете создавать программу, которая находит и выводит все простые числа меньше 100. Вы можете использовать этот пример для практики написания Java-функций и циклов.

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

```
 * TODO:  Комментарий, описывающий класс

public class Primes {

    * TODO:  Комментарий, описывающий метод

   public static void main(String[] args) {

      // TODO:  Реализация программы

   }

}
```

Воспользовавшись данным классом, вы должны собрать и запустить программу, но, конечно, на практике пока ничего не произойдет, потому что вы еще не применили код.

2. Внутри этого класса, после метода Main (), опишите функцию IsPrime (Int n), которая сообщает, является 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)

{

    // TODO:  Реализация

}
```

Вы можете применять этот метод по вашему усмотрению, однако простой подход заключается в написании цикла for. Данный цикл перебирает числа, начиная с 2 до (но не включая) n, проверяя существует ли какое-либо значение, делящееся на n без остатка. Это можно протестировать с помощью 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 отработает, 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. Когда ваша программа будет завершена, скомпилируйте и протестируйте её. Убедитесь, что результаты правильны. В интернете вы сможете найти списки простых чисел, поэтому будет легко проверить результаты.

Кроме того, как видно из примера, следует не забыть написать комментарий о назначении класса, а также необходим комментарий перед каждым методом с описанием его цели. Когда вы пишете программы, крайне важно писать подобные комментарии.

Палиндромы

Вторая программа, которую вы будете писать, показывает, является ли строка палиндромом. Существует несколько различных способов реализации такой программы, но мы обратимся к способу, описанному в первой лекции.

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

```
  /**
 * TODO:  Описание класса
 */
public class Palindrome {
   /**
```

```
        * TODO:  Описание метода

        */

       public static void main(String[] args) {

          for (int i = 0; i < args.length; i++) {

             String s = args[i];

} }

}
```

Опять же, вы должны будете скомпилировать и запустить эту программу как она есть, но она пока не покажет никакого интересного результата.

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(String s)

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 сообщает 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
```

Для этого вы можете использовать оператор конкатенации (соединения) строк + , или же можете использовать оператор +=, если предпочитаете.

3. После того как вы применили метод reverseString (), можете создать еще один метод public static boolean isPalindrome(String s). Все, что этому методу нужно сделать, это создать обратную версию s, а затем сравнить с первоначальными входными данными. С помощью String (и со всеми объектами Java), 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 мы будем обсуждать на следующей лекции.

4. После того как вы закончили писать свой код, скомпилируйте и протестируйте программу! На этот раз, вы будете давать вашей программе входные данные в качестве аргументов командной строки, например:

```
java Palindrome madam racecar apple kayak song noon
```

Это должно привести к тому, что ваша программа выведет ответ, является ли каждое слово палиндромом.

5. Как и прежде, убедитесь, что цель вашей программы и каждый метод в программе задокументированы.

Все готово!

Задача № 1: Java - Сразу к делу

Java позволяет нам программировать с объектами. Мы используем классы, по одному для каждого файла, чтобы описать, как эти объекты работают - план, *если хотите*. Вот код для простого класса, который представляет двумерную точку:

```
/ **
 * двумерный класс точки.
 **/
общедоступный класс Point2d {

   / ** X координат точки **/
   частный двойной xCoord;

   / ** Y координата точки **/
   частный двойной yCoord;

   / ** Конструктор, чтобы инициализировать точку к (x, y) значение.
**/
   общедоступный Point2d (удваивают x, дважды y) {
      xCoord = x;
      yCoord = y;
   }
/ ** Конструктор без параметров: значения по умолчанию к точке в
источнике. **/
   общедоступный Point2d () {
      //Вызовите конструктор с двумя параметрами и определите
источник.
      это (0, 0);
   }

   / ** Верните X координат точки. **/
   общественность удваивает getX () {
      верните xCoord;
   }

   / ** Возвратите координату Y точки. **/
   общественность удваивает getY () {
      верните yCoord;
   }

   / ** Набор X координат точки. **/
   общественность освобождает setX (удвойте val) {
      xCoord = val;
   }
```

```
    / ** Набор координата Y точки. **/
    общественность освобождает setY (удвойте val) {
        yCoord = val;
    }
}
```

Этот код должен быть сохранен в файле по имени Point2d.java, согласно требованиям Java к именам классов и именам файлов. Копия этого файла предназначена для вашего использования. Вы можете сохранить копию к вашему рабочему указателю и эффективно использовать его для этой лаборатории.

Вспомните, что мы можем создать экземпляр нашего класса где-либо еще в нашем коде, вызвав любой конструктор, который мы определили, так:

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

Будьте осторожны: myPoint! = aThirdPoint, даже при том, что их значения - те же. Это вызвано тем, что оператор равенства == (и его инверсия, оператор неравенства ! =) это две ссылки объекта на предмет равенства. Другими словами, == верните true, если эти две ссылки указывают на тот же объект. В этом коде myPoint и aThirdPoint каждый обращается к различному экземпляру класса Point2d, таким образом, myPoint == aThirdPoint возвращает false, даже при том, что их значения - те же!

Чтобы проверить на равенство значения и не ссылочное равенство, мы определяем метод класса Point2d, вызванный равенством, который берет другой объект в качестве параметра и выполняет надлежащие тесты для равенства. Помните, что сравниваемый объект должен быть корректным типом, и у его членских полей должны быть те же значения.

Прежде чем вы начнете «кодить»

Стиль программирования очень важен в любом проекте программного обеспечения, над которым вы работаете. Хотите верьте, хотите нет, подавляющее большинство времени жизни успешной программы тратится в фазах отладки и обслуживания. В этих фазах жизненного цикла продукта хорошо задокументированный и читаемый код становится огромным активом, сохраняя серьезное количество времени.

К сожалению, значение хорошего стиля кодирования часто изучается только болезненным опытом... Но, CS11 - хорошая возможность изучить и попрактиковать хороший стиль кодирования. Прежде чем вы начнете это задание, рассмотрите инструкции по стилю CS11 Java. На кластере CS есть даже полезная программа проверки стиля, которая сообщит о любых проблемах с вашей программой.

Ваша задача

1. Создайте новый класс Point3d чтобы представить, что эти точки представлены в трехмерном Евклидовом пространстве. Должно быть возможно:

• создание нового Point3d, описанного любыми тремя тосками, с плавающей запятой (тип double) значениями,

• создание нового Point3d в (0.0, 0.0, 0.0) по умолчанию,

• получение доступа и видоизменение всех трех значений индивидуально,

• сравнение двух Point3ds для равенства значения с использованием соответствующего эквивалентного метода.

Не следует непосредственно получать доступ к внутренним элементам данных любого объекта Point3d.

2. Кроме того, добавьте новый метод distanceTo, который берет другой Point3d в качестве параметра и вычисляет двойную точность приближения плавающий точкой расстояния по прямой между двумя точками и возвращает это значение.

3. Создайте второй класс под названием Lab1, который существует, прежде всего, чтобы содержать статический основной метод. Помните, что главным должна быть общедоступность, которая имеет пустой тип возврата и принимает набор последовательностей как параметр. В этом классе добавьте некоторую функциональность:

• ввод трех упорядоченных последовательностей утраивается от пользователя, каждый представляя координаты одной точки с тремя пространствами. Генерируйте три объекта Point3d от этих данных. (На данный момент вы можете принять, что пользователь не вводит недопустимые данные.)

Если вы не знаете, как получить ввод пользователя, вы можете использовать функцию в этом файле. Поместите его как статический метод в ваш класс Lab1. Обратите внимание на то, что этот метод использует классы в комплекс java.io, который не видим вашему коду по умолчанию. Чтобы сделать его видимым, добавьте его к очень главному из вашего файла:

импорт java.io.*;

Это делает все классы в комплексе java.io видимыми вашему коду Lab1. (Вы не должны делать этого с классами комплекса java.lang , так как те сделаны доступными для ваших классов по умолчанию.)

• Запишите второй статический метод computeArea, который берет три Point3d и вычисляет область в треугольнике, ограниченном ими. (Можно использовать формулу Heron's.) Верните эту область как двойную.

• Используйте данные и коды, собранные и записанные вами, чтобы определить область и распечатать это для пользователя.

Прежде чем вы вызовете computeArea, проверьте равенство значения между всеми тремя Point3d. Если какая-либо пара точек "равна", сообщите об этом пользователю и не вычисляйте область.

4. Соберите оба свои исходные файлы вместе:

javac Point3d.java Lab1.java

и затем выполните вашу программу Lab1, тестируя с несколькими демонстрационными треугольниками.

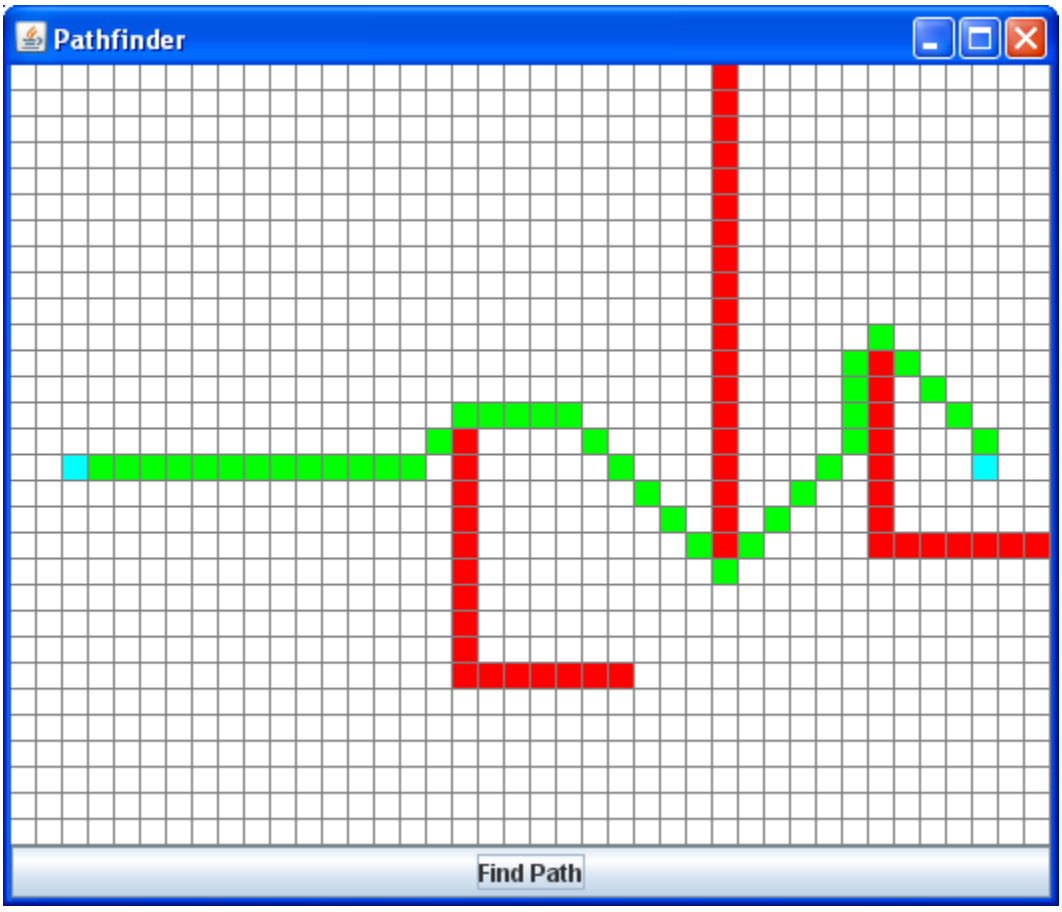
5. Когда вы закончите с лабораторией 1, вы можете представить свои файлы на csman веб-сайте.

Задача № 3: Java-Я-звезда!

Если вы когда-нибудь играли в какую-либо карточную игру на компьютере, вы, вероятно, столкнулись с подразделениями с компьютерного управления, которые знают, как добраться из пункта А в точку Б сами по себе. На самом деле это обычная проблема как в играх, так и в других видах программного обеспечения - как сгенерировать путь от начального местоположения до нужного пункта назначения, который успешно преодолевает препятствия.

Один очень широко используемый алгоритм для этого вида проблемы называют A* (произносится "Я-звезда"), и это - очень эффективный алгоритм для новаторства в компьютерной программе. Алгоритм концептуально довольно прост. Запускаясь с начального расположения, алгоритм постепенно создает путь от источника до места назначения, всегда используя "до сих пор лучший путь ", чтобы сделать следующий шаг. Это гарантирует, что завершающий путь также будет оптимальным. (Если вы хотите узнать больше о A* новаторский алгоритм, вы можете открыть статьи Wikipedia на A* и следовать по ссылкам, которые дает статья.)

К счастью, вам не придется выполнять A* алгоритм; это было уже сделано за вас. На самом деле, есть юзабилити пользовательский интерфейс для вас, чтобы экспериментировать с этим алгоритмом:



Вы можете щелкнуть по различным квадратам, чтобы превратить их в барьеры (красные) или проходимые клетки (белые). Синие клетки обозначают запуск и конец пути. Нажатие на кнопку "Find Path" вычислит путь, используя A*, и затем выведет на экран его в зеленом, или, если не будет никакого пути от запуска, чтобы завершиться, программа просто не покажет пути.

A* алгоритм имеет много информации, чтобы отслеживать путь, и набор классов Java (идеально подходит для такого рода задач). Есть два основных вида информации, которыми A* должно осуществлять управление:

Локации – это просто наборы координат конкретных клеток на двухмерной карте. A* алгоритм должен иметь возможность ссылаться на определенные места на карте. Путевые точки - это отдельные шаги на пути, которые A* алгоритм генерирует. Например, вышеперечисленные шаги зеленого пути - это просто последовательность точек через карту. Каждая точка содержит несколько элементов информации, связанны с ней:

- расположение ячейке как точка пути;
- ссылка на предыдущие точки маршрута в пути. Конечный путь - это просто последовательность точек из пункта назначения обратно к исходной точке;
- фактические расходы движения от начального местоположения до текущего местоположения этой точки, следующие конкретному пути, которые заканчиваются точкой;
- эвристическая оценка (предположение, другими словами) оставшейся стоимости передвижения от этого места до конечного пункта назначения.

Так как A* алгоритм строит свой путь, он должен содержать два основных набора точек:

- первый набор "открытые точки" или точки, которые еще должны быть рассмотрены алгоритмом A*;
- второй набор "закрытые точки" или точки, которые уже были рассмотрены алгоритмом A* и их не надо будет снова рассматривать.

Каждая итерация A* алгоритма довольно проста: найти наименее дорогостоящий пункт из набора открытых точек, сделать шаг в любом направлении от этой точки для создания новой открытой точки, а затем переместить путевую точку из открытого набора в закрытый набор. Это повторяется до тех пор, пока текущая точка не достигнет пункта назначения! Если во время этого процесса алгоритм открытых точек заканчивается, то нет пути от начальной точки до пункта назначения. Такая обработка в первую очередь зависит от расположения точек, поэтому он очень полезен для хранения путевых точек как отображение от места до соответствующих точек. Таким образом, вы будете использовать

хранилище java.util. hashmap для каждого из этих наборов с расположением объектов в качестве кодов, и путевых точек объектов в качестве значений.

Прежде чем начать

Прежде чем начать, вы должны скачать исходные файлы для данного учебно-методического пособия:

- Map2D.java - представляет собой карту, по которой A* алгоритм двигается, в том числе проходимы ли клетки.
- Location.java - этот тип представляет собой координаты конкретной ячейки на карте.
- Waypoint.java - представляет отдельные точки в созданный путь.
- AStarPathfinder.java - этот тип реализует A* алгоритм поиска пути как статический метод.
- AStarState.java - этот тип хранит набор открытых точек и закрытых точек и обеспечивает базовые 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 клеток на карте.

Обратите внимание, что приложение будет успешно компилироваться как оно и есть, но путь к установлению функций не будет работать, пока вы не выполните задание. Единственные типы, которые вы должны изменить это расположение и AStarState. Все остальное-это код платформы, которая позволяет редактировать карту и показывать путь, по которому алгоритм генерирует. (Если вы редактируете любой из других исходных файлов, чтобы выполнить практикум, то остановитесь и попросите о помощи!)

Размещение

Первое, что должно быть сделано – тип должен быть подготовлен для использования с наборами типов Java. Поскольку вы будете использовать кеширование хранилищ для выполнения данного задания, то это предполагает:

- обеспечение реализации метода equals ();
- обеспечение реализации метода hashcode().

Добавьте реализацию каждого из этих методов в тип Location, как описано описано в общих чертах в типе. Как только это будет завершено, вы можете

использовать тип Location как ключевой код хеширования в хранилище, то есть hashset и hashmap.

A* форма

После того как тип Location готов к использованию в качестве кода, вы можете закончить реализацию типа AStarState. Это тип, который хранит наборы открытых и закрытых точек, так что это действительно обеспечивает базовую функциональность для A* реализации.

Как упоминалось ранее, A* форма состоит из двух наборов путевых точек: одна из открытых точек, и другие из закрытых точек. Чтобы облегчить алгоритм, путевые точки будут храниться в хэш-карте, в месте, в котором находятся коды, путевые точки и сами значения. Таким образом, у вас будет тип такой:

HashMap<Location, Waypoint>

(Очевидный вывод из всего этого заключается в том, что каждая локация карты может иметь только один путь, связанный с ней. Это именно то, что мы хотим.)

Добавить две (нестатические) области класса AStarState с этим типом: одну для "открытых точек" и другую для "закрытых точек." Кроме того, убедитесь, что нужно инициализировать каждую из этих областей, чтобы отнести в новую пустую коллекцию.

Если у вас есть созданные и инициализированые области, вы должны реализовать следующие методы типа AStarState:

1) public int numOpenWaypoints()

Этот метод просто возвращает количество точек в набор открытых точек. (Да, это будет остроумное замечание...)

2) public Waypoint getMinOpenWaypoint()

Эта функция должна проверить все точки в наборе открытых точек и вернуть ссылку на точку с наименьшей общей стоимостью. Если нет точки в "открытых" наборах, вернитесь NULL.

Не удаляйте путевую точку из набора, когда вы вернули ее, просто верните ссылку на точку с наименьшей общей стоимостью.

3) public boolean addOpenWaypoint(Waypoint newWP)

Это самый сложный способ в A* форме. A делает его сложнее, чем остальные то, что он должен только добавить указанную точку при существующей путевой точке. Вот то, что этот метод должен делать:

- Если в настоящее время нет точек для этого места в "открытых точках" набора, то просто добавить новую точку.
- Если точка уже в этом месте в "открытой точке" набора, то потом добавить новый пункт, только если "старая цена" за новую точку меньше "старой цены" за текущую точку. (Убедитесь, что используете прежнюю стоимость, а не общую стоимость.) Другими словами, если новая точка представляет собой более короткий путь к этому месту, чем текущий маршрут, заменить текущую точку на новую.

Вы можете заметить, что вам придется вернуть существующую маршрутную точку с «открытого» набора, если он есть, и действовать в соответствии с правилом открытого набора. К счастью, это очень просто - заменить предыдущую точку на новую; просто используйте метод hashmap.put (), как обычно, и он заменит старый код, а значения сопоставит с новым кодом.

Наконец, верните действительный метод, если новый пункт был добавлен в набор открытых точек, или ошибочный, если новые точки не добавляются.

4) public boolean isLocationClosed(Location loc)

Эта функция должна вернуть действительный метод, если указанное расположение появляется в наборе закрытых точек или в случае неверного метода. Так как закрытые методы сохранены в хэш-таблице с расположениями значений кода, это довольно просто реализовать.

5) public void closeWaypoint(Location loc)

Эта функция берет точку и перемещает её от набора «открытых точек» к набору «закрытых точек». Так как точки закодированы их расположением, метод берет расположение точек.
Процесс должен быть простым:

- удалите соответствие точки указанному расположению из набора «открытых точек»;
- добавьте точку, которую вы просто удалили из набора закрытых точек. Конечно, ключ должен быть расположением точки.

Компиляция и тестирование

Как только вы получаете вышеупомянутую реализованную функциональность, даете новаторской программе выполнить работу, чтобы у видеть правильное выполнение. Если вы реализовали все правильно, у вас не должно быть проблем при создании помех, а затем и нахождении путей вокруг них.

Вы можете скомпилировать и выполнить программу тем же путем, что и всегда:

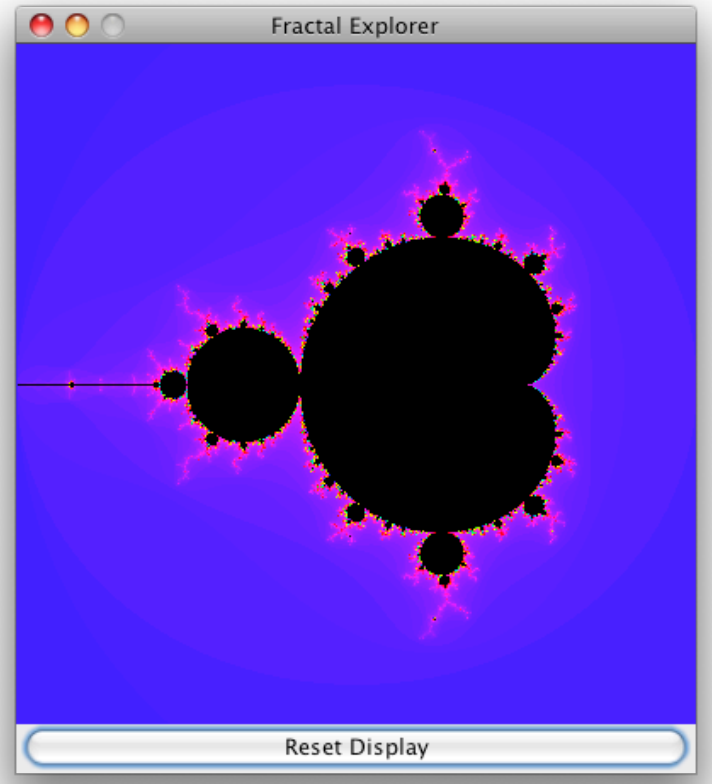
```
javac*.java
    java AStarApp
```

Задача № 4: Фрактал Эксплорер

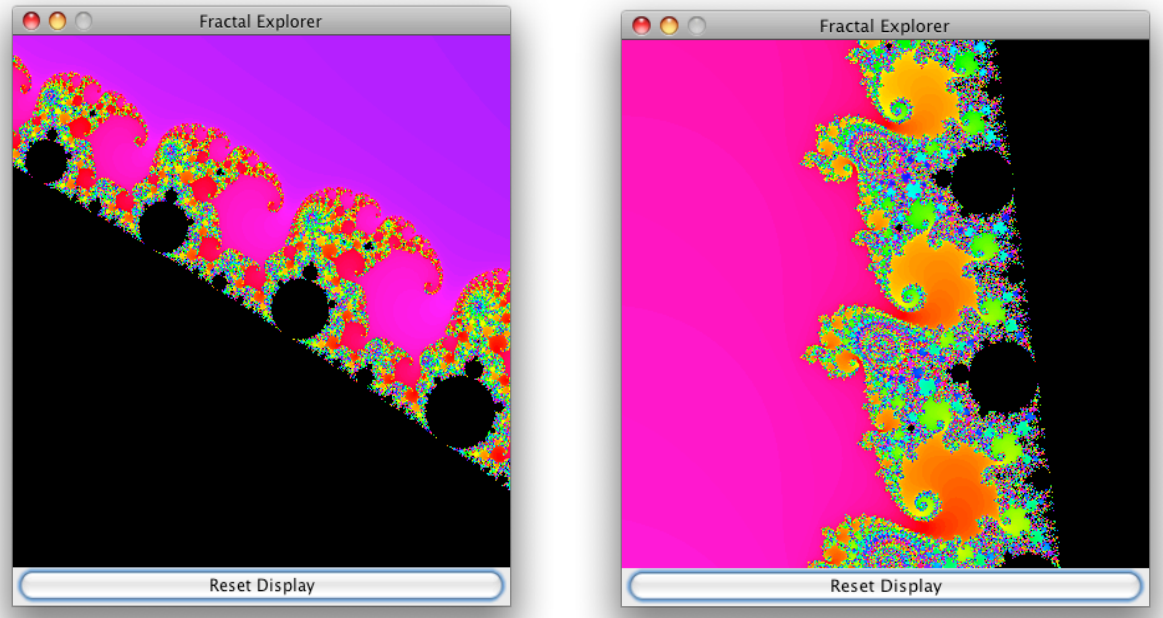
Для следующих нескольких задач вы соедините небольшое JAVA-приложение, которое может нарисовать некоторые удивительные фракталы. Если вы никогда не играли с фракталами прежде, вы будете поражены тем, как просто можно создать некоторые захватывающие дух красивые изображения. Мы сделаем это все с платформами Swing, API Java, которые позволяют вам создавать графические интерфейсы пользователя.

Мы будем создавать это приложение по многократным задачам, таким образом, наша начальная версия будет довольно проста, но само приложение мы создадим в следующих задачах, чтобы добавить другие функции, такие, как способность сохранять образы, которые мы генерируем, и способность переключаться между различными видами фракталов. И GUI, и механизм для поддержки различных фракталов, будут зависеть от иерархий классов.

Вот простой пример GUI в его начальном состоянии:



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



Создание пользовательского интерфейса

Прежде чем мы сможем нарисовать любые фракталы, мы должны будем создать графический виджет, который позволит нам отображать их. Swing не обеспечивает такой компонент, но очень просто создать его самостоятельно. Обратите внимание на то, что мы будем использовать широкий спектр Java AWT и классы Swing в этой задаче, и нет никакого простого способа, которым мы можем объяснить их детали . Однако нет никакой потребности в этом, потому что онлайновые документы API Java очень всесторонни и просты в использовании. Просто переместитесь к пакету данного класса Java, выберите сам класс и затем прочтите подробную информацию о том, как использовать класс.

Создайте класс JImageDisplay, который возьмём из javax.swing. JComponent. У класса должны быть одно частное поле, пример java.awt.image. BufferedImage. Класс BufferedImage управляет изображением, в содержание которого мы можем записать информацию.

Конструктор JImageDisplay должен взять целочисленную ширину и высоту и инициализировать ее члена BufferedImage, чтобы быть новым изображением той ширины и высоты, и типом изображения TYPE_INT_RGB. Тип просто определяет, как цвета каждого пикселя представлены в изображении; это определенное значение означает, что красные, зеленые, и синие компоненты - каждые 8 бит, и они появляются в интервале в том порядке.

Ваш конструктор должен сделать еще одну вещь: это должно вызвать setPreferredSize родительского класса () - метод с указанной шириной и высотой. (Вы должны будете передать эти значения в java.awt. Объект размерности вы создаете, в частности, для этого вызова.) Таким образом, когда ваш компонент будет включен в пользовательский интерфейс, он на самом деле выведет на экран все изображение.

Пользовательские компоненты Swing должны обеспечить свой собственный код для прорисовки, переопределив защищенный paintComponent (графика g) - метод JComponent. Так как наш компонент просто выведет на экран сами данные изображения, наша реализация будет очень проста! Во-первых, реализацию суперкласса paintComponent (g) нужно всегда вызывать так, чтобы любые границы или другие функции были оттянуты правильно. Как только вы вызвали версию суперкласса, вы можете вовлечь изображение в компонент, используя работу:

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

(Мы передаем нуль для ImageObserver, так как нам не нужна эта функциональность.)

Вы также должны обеспечить два открытых метода для записи данных в изображение: clearImage () - метод, который устанавливает все пиксели в данных изображения к черному цвету (значение RGB 0), и drawPixel (интервал x, интервал y, интервал rgbColor) - метод, который устанавливает определенный цвет для пикселя. Оба эти метода должны будут использовать один из setRGB () - метод на классе BufferedImage.

Конечно, не забывайте писать четкую, полную и краткую документацию для своего класса и методы, объясняя, что все делает.

Вычисления фрактала Mandelbrot

Затем вы запишете код, чтобы вычислить очень известный фрактал Мальдерброта. Чтобы поддерживать многократные фракталы в будущем, вам предоставляют исходный файл FractalGenerator.java, из которого произойдут все Ваши фрактальные генераторы. вы также заметите, что некоторые очень полезные операции обеспечены, чтобы перевести из экранных координат в систему координат вычисляемого фрактала.

Виды фракталов, с которыми мы будем работать, вычислены в комплексной плоскости и включают очень простые математические функции, которые неоднократно выполняются с помощью итераций, пока некоторое условие не удовлетворено. Для фрактала Мальдерброта функция - цинк = цинк 12 + c, где все значения - комплексные числа, z0 = 0, и c - определенная точка во фрактале, который мы выводим на экран. Это вычисление выполнено с помощью итераций до любого |z |> 2 (в этом случае точка не находится во множестве Мандельброта), или пока количество итераций не поражает максимальное значение, например, 2000 (в этом случае, мы предполагаем, что точка находится в наборе).

Процесс графического изображения множества Мандельброта очень прост: мы просто выполняем итерации по каждому пикселю в нашем изображении, вычисляем количество итераций для соответствующей координаты, и затем устанавливаем пиксель в цвет на основе количества итераций, которые мы вычислили. Но, мы доберемся до этого через секунду - на данный момент, вы просто должны реализовать вышеупомянутое вычисление.

- Создайте подкласс FractalGenerator по имени Mandelbrot. Вы будете видеть, что есть только два метода, которые вы должны обеспечить в подклассе, getInitialRange () и numIterations ().
- getInitialRange (Rectangle2D.Double) метод позволяет фрактальному генератору определять, какая часть комплексной плоскости является самой "интересной" для определенного фрактала. Обратите внимание на то, что прямоугольный объект передан как параметр методу, и метод должен изменить поля прямоугольника, чтобы отразить надлежащий начальный диапазон для фрактала. (Вы видите пример этого в методе FractalGenerator.recenterAndZoomRange ().) Реализация этого метода должна установить начальный диапазон в (-2 - 1.5i) - (1 + 1.5i). Т.е. x и значения y будут -2 и -1.5 соответственно, и ширина и высота будут равняться 3.
- numIterations - метод реализует итеративную функцию для фрактала Мальдерброта. Вы можете определить константу для "максимальных итераций",

```
public static final int MAX_ITERATIONS = 2000;
```

Тогда вы можете обратиться к этому значению в вашей реализации.

Обратите внимание на то, что у Java нет типа данных для комплексных чисел, таким образом, вы должны будете реализовать итеративную функцию, используя отдельные двойные компоненты для действительных и мнимых частей. (Я предполагаю, что вы могли реализовать свой собственный класс комплексного числа, но это не будет стоить того). Вы должны попытаться сделать свою реализацию быстро; например, не сравнивайте |z | с 2; сравните |z| 2 с 22, чтобы избежать сложных и медленных вычислений квадратного корня. И не используйте Math.pow (), чтобы вычислить маленькие целочисленные числа; умножьте их непосредственно, иначе ваш код будет очень медленным.

Наконец, при переборе, ваша функция, если вы попали MAX_ITERATIONS, потом просто возвращает значение -1, чтобы указать, что точка не выходит за границы.

Соединим всё это

Наконец мы готовы начать отображать фракталы! Теперь вы создадите класс FractalExplorer, который позволяет вам исследовать различные части фрактала, создавая и показывая GUI Swing и обрабатывая события, вызванные различным взаимодействием с пользователем.

Как вы видите от вышеупомянутых изображений пользовательского интерфейса, Фрактальный Проводник очень прост, состоит из JFrame, содержащего JImageDisplay, который выводит на экран фрактал, и единственный JButton для сброса дисплея, чтобы показать весь фрактал. Вы можете достигнуть этого простого расположения, установив фрейм, имеющий BorderLayout, затем

поместив дисплей в центр расположения и кнопку сброса в "южной" части расположения.

- Ваш класс FractalExplorer должен будет отслеживать несколько важных полей для состояния программы:
    1) целое число "выводит на экран размер", который является просто шириной и высотой дисплея в пикселях. (Наш фрактальный дисплей будет квадратным);
    2) ссылка JImageDisplay, так, чтобы мы могли обновить наш дисплей из различных методов, поскольку мы вычисляем фрактал;
    3) объект FractalGenerator. Мы будем использовать ссылку базового класса так, чтобы мы могли показать другие виды фракталов в будущем;
    4) определение объекта Rectangle2D.Double диапазона комплексной плоскости, которую мы в настоящее время выводим на экран.

Конечно, все эти поля будут частными.

- У класса должен быть конструктор, который берет размер дисплея в качестве параметра, затем хранит это значение в соответствующем поле и также инициализирует объекты фрактального генератора и диапазон. Обратите внимание на то, что конструктор не должен устанавливать компоненты Swing; они будут установлены в следующем методе.
- Обеспечьте createAndShowGUI () метод, который инициализирует GUI Swing: JFrame, содержащий JImageDisplay, возвращает и кнопку для сброса дисплея. Вы должны установить фрейм, использующий java.awt. BorderLayout для его содержания; добавьте объект дисплея изображения в BorderLayout. Центральная позиция и кнопку в BorderLayout.

Вы должны также дать фрейму подходящий заголовок для своего приложения и установить операцию закрытия фрейма по умолчанию "выходить" (см. JFrame.setDefaultCloseOperation ()).

Наконец, после того как компоненты UI инициализированы и размечены, включают эту последовательность операций:

frame.pack ();

frame.setVisible (истина);

frame.setResizable (ложь).

Это правильно разметит содержание фрейма, заставит его быть видимым (окна не видимы, когда они создаются: сконфигурировать их прежде, чем вывести на экран), и затем отказаться от изменения размеров окна.

- Вы должны реализовать частный метод помощника, чтобы вывести на экран фрактал, например, вызванный drawFractal (). Этот метод должен циклично выполниться через каждый пиксель в дисплее (т.е. x будет колебаться от 0 до размера дисплея), и сделайте следующее:

Вычислите количество итераций для соответствующих координат в области дисплея фрактала. Вы можете определить координаты с плавающей точкой для определенного набора пиксельных координат, используя FractalGenerator.getCoord () метод помощника; например, чтобы получить x-координату, соответствующую пиксельной - X координате, вы сделали бы это:

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

Если количество итераций - 1 (т.е. точка не выходит, se цвет пикселя к черному цвету (rgb оценивают 0)). Иначе вы должны выбрать значение на основе количества итераций. Мы можем на самом деле использовать цветовое пространство HSV для этого: поскольку оттенок колеблется от 0 до 1, мы получаем гладкую последовательность цветов от красного до желтого, зеленого, синего, фиолетового, и затем назад к красному! Вы можете реализовать работу

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

Конечно, если вы придумываете некоторый другой интересный способ окрасить пиксели на основе количества итераций, не стесняйтесь использовать его!

Дисплей должен быть обновлен с цветом для каждого пикселя, таким образом, вы будете использовать свой drawPixel (), работающий с запоминанием предыдущего результата.

Когда вы закончили тянуть все пиксели, вы должны вынудить JImageDisplay быть перекрашенным, чтобы соответствовать текущему содержанию его изображения. Сделайте это, вызвав перекрашивание на компоненте. Если вы забудете сделать это, ваш дисплей никогда не будет обновляться!

Создайте внутренний класс, чтобы обработать java.awt.event. События ActionListener от кнопки сброса. Обработчик просто должен сбросить диапазон к начальному диапазону, определенному генератором, и затем потянуть фрактал.

Как только вы завершили этот класс, обновите свой createAndShowGUI () метод, чтобы зарегистрировать экземпляр этого обработчика на кнопке сброса.

Создайте другой внутренний класс, чтобы обработать java.awt.event. События MouseListener от дисплея. Действительно, вы только должны обработать событие щелчка мышью, таким образом, вы должны получить этот внутренний класс из класса MouseAdapter AWT. Когда этот обработчик получает событие щелчка мышью, он должен отобразить пиксельные координаты щелчка

в область фрактала, который выводится на экран, и затем вызовать recenterAndZoomRange генератора - метод с координатами, по которым щелкнули, и с масштабом 0.5. Таким образом, только, нажимая на расположение во фрактальном дисплее увеличит масштаб этого расположения!

Конечно, не забывайте перерисовывать фрактал после того, как вы измените область выводимого на экран фрактала.

После того как этот класс сделан, обновите свой createAndShowGUI () метод, чтобы зарегистрировать экземпляр этого обработчика на компоненте фрактального дисплея.

Наконец, вы должны создать статический основной метод для фрактального проводника так, чтобы это могло быть запущено. Основной метод будет очень прост в данный момент:

Инициализируйте новый экземпляр FractalExplorer с размером дисплея 800.

Вызовите createAndShowGUI () на объекте проводника.

Вызовите drawFractal() на проводнике, чтобы видеть начальное представление!

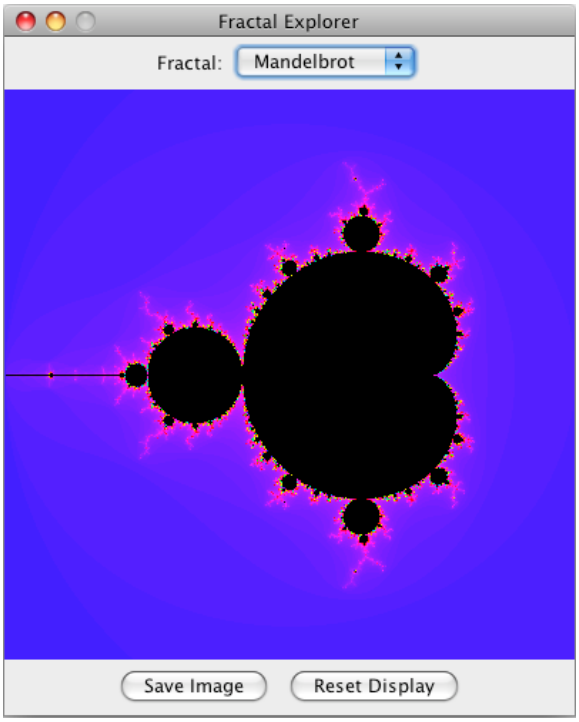
Как только вы завершили все эти шаги, вам необходимо «путешествовать» вокруг рассмотрения фрактала Мальдерброта. Если вы увеличите масштаб достаточно, то столкнетесь с двумя интересными проблемами:

- во-первых, вы в конечном счете найдете, что уровень детализации заканчивается; это вызвано тем, что нам были бы нужны больше чем 2000 итераций, чтобы узнать, находится ли точка во множестве Мандельброта или нет. Конечно, мы могли бы испытать желание увеличить максимальные итерации, но тогда черные области фрактала действительно замедлят нас;
- во-вторых, если вы увеличите масштаб действительно сильно, вы столкнетесь с пикселированным выводом дисплея! Это вызвано тем, что вы сталкиваетесь с пределом того, какую двойную точность значения с плавающей точкой могут представлять.

Вы, также заметите, что это довольно раздражающе, когда весь дисплей зависает, в то время как фрактал оттягивается. Это то, что мы будем исследовать в следующих задачах, а также использование в своих интересах многоядерных процессоров, чтобы потянуть наши фракталы намного быстрее. Но на данный момент, как только у вас есть свой фрактальный завершенный проводник (и хорошо закомментированный).

Задание №5. Выбор и сохранение фракталов

В этой задаче вы расширите генератор фракталов двумя новыми функциями. Во-первых, вы научитесь работать с множеством фракталов и сможете выбирать нужный вам фрактал с помощью выпадающего списка. Во - вторых, вы научитесь сохранять текущее изображения фрактала в файл. Ниже приведен скриншот, как будет выглядеть ваша новая программа.



Верх генератора фракталов включает два виджета, которые позволяют пользователю выбирать фрактал, а внизу есть функция "Сохранить", чтобы сохранить текущее изображения фрактала.

Так как теперь будет несколько источников событий (action-event sources), вы можете научиться обращаться с ними с первоначальной реализацией одного ActionListener в классе.

Поддержка множественных фракталов

Очень легко добавить несколько фракталов в ваш генератор фракталов, именно потому что мы ввели абстракцию FractalGenerator в нашу изначальную реализацию. Научимся добавлять поддержку нескольких фракталов, и пользователь сможет выбирать между ними, используя *combo-box* . Эта самая распространенная метафора пользовательского интерфейса (UI) (т.е. стиль взаимодействия с которым пользователи уже знакомы из множества других пользовательских интерфейсов), поэтому это должно быть легко для понимания. Программный интерфейс Swing (Swing API) предоставляет *combo-bo) через* javax.swing.JComboBox класс, а еще лучше удаляет ActionEvents когда выбран новый предмет. Вот, что вам нужно сделать.

Создайте две новые имплементации генератора фракталов. Первый фрактал - tricorn, который должен быть в файле Tricorn.java. Как и раньше, вам следует создать подкласс FractalGenerator, и имплементация будет

почти идентична фракталу Мандельброта, кроме двух изменений. На самом деле, вы даже можете скопировать код источника Мандельберта и просто изменить следующие строки:

Равенство is $z_n = z_{n\text{-}1}^2 + c$. Единственное отличие только в том, что мы берем сложное сопряжение $z_{n\text{-}1}$. Каждая итерация должна начинаться с изначальной областьи определения фрактала tricorn и должна быть от (-2, -2) до (2,2).

Второй фрактал, который мы будем имплементировать, это «горящий корабль», который так назван, потому что очень похож на горящий корабль. Вот детали:

Равенство $z_n = (|\mathrm{Re}(z_{n\text{-}1})| + i\ |\mathrm{Im}(z_{n\text{-}1})|)^2 + c$. Другими словами, вы берете абсолютное значение каждого компонента $z_{n\text{-}1}$. Каждую итерацию начинаем с изначальной области определения данного фрактала, т.е. от (-2, -2.5) до (2,1.5).

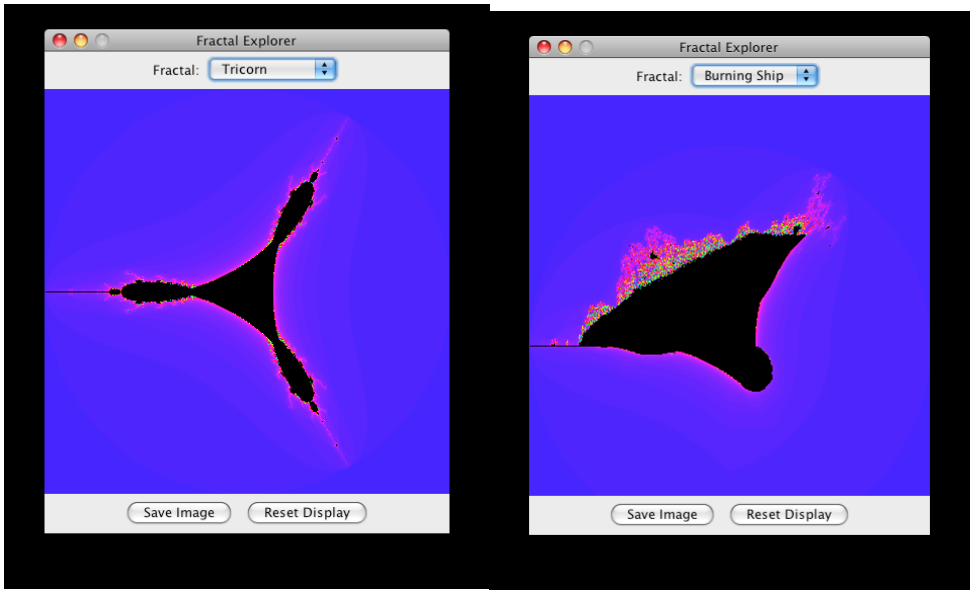
Swing combo-boxes может управлять коллекцией объектов, но объекты должны предоставлять имплементацию to String(). Убедитесь, что вы сделали имплементацию toString() для каждой имплементации фрактала, которая дает имя, например Мандельброт, Трикорн и «Горящий корабль».

Очень легко настроить JComboBOx в вашем пользовательском интерфейсе. Вы можете использовать безаргументный конструктор и затем использовать метод addItem(Object), чтобы добавить каждую имплементацию вашего генератора фракталов. Как говорилось в предыдущем шаге, выпадающий список будет использовать toString () метод вашей имплементации, чтобы отобразить генераторы в выпадающем списке.

Вам также следует добавить label в ваш пользовательский интерфейс перед выпадающим списком, что объяснит, для чего нужен выпадающий список. Вы можете это сделать, создав новый объект JPanel, добавив JLabel объект и JComboBOx объект в него, а затем поставить панель в ваш фрейм в позицию NORTH от слоя фрейма.

И наконец, вам нужно добавить поддержку вашего выпадающего списка имплементации ActionListener. Вы можете проверить, является ли выпадающий список источником action-event и если да, вы можете извлечь выбранный объект из виджета и сделать его текущим генератором фракталов (вы можете использовать метод getSelectedItem()). Не забудьте сбросить изначальную область определения и перерисовать фрактал!

Чтобы убедиться в правильности выполненной работы, вот изображения Трикорна и «горящего корабля»



Сохранения изображения фрактала

Следующее ваше задание состоит в сохранении текущего изображения фрактала на диск. Возможно звучит пугающе, но Java API (программный интерфейс) предоставляет несколько инструментов чтобы сделать это задание простым.

Во-первых вам нужно добавить кнопку «Сохранить изображение» на ваш дисплей. Вы можете поставить обе кнопки «Сохранить» и «Сбросить» на вашу новую JPanel, а затем поставить эту панель в SOUTH часть фрейма, почти так же, как вы добавляли label и выпадающий список. (Не используйте одинаковые объекты панели (panel objects) в обоих местах, иначе получите довольно странные результаты!)

Кнопка Сохранить также должна обрабатываться имплементацией ActionListener (action-events handled by your ActionListener implementation. Вам нужно дать вашим кнопкам «Сохранить» и «Сбросить» свои команды (например, сохранить и сбросить), чтобы обработчик (action-handler) событий мог отличить эти две кнопки.

В вашем обработчике кнопки Сохранить вам нужно разрешить пользователю указывать, какой файл он хочет сохранить в изображение. Вы можете сделать это с классом javax.swing.JFileChooser, который упрощает задачу. Этот класс предоставляет метод showSaveDialog(), который дает всплывающее диалоговое окно «Сохранить файл», позволяя пользователю выбрать директорию для сохранения. Этот метод принимает графический компонент, который является родительским элементом выбора. Это позволяет центрировать выбор по отношению к его родительскому элементу. Используйте фрейм приложения как родительский. Вы можете заметить, что этот метод возвращает int значение, указывая исход операции выбора файла. Если метод возвращает JFileChooser.APPROVE_OPTION, тогда вы можете продолжить операцию сохранения файлов, в противном случае, пользователь отменил

запрос, поэтому просто вернитесь. Если пользователь выбрал место для сохранения файла, это место доступно через getSelectedFile() метод, как объект файла.

Вам также нужно настроить выборщик файлов, чтобы он сохранял только png изображения, это пока единственный формат, с которым вы будете работать. вы можете это сделать с javax.swing.filechooser.FileNameExtensionFilter с таким фрагментом кода

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

Последняя строка гарантирует, что пользователь не разрешает использование не png файлов.

Если пользователь успешно выбрал файл, следующим шагом является сохранение изображения фрактала на диск. Это может быть сложным, но Java уже включает в себя эту функцию. Класс javax.imageio.ImageIO позволяет реализовывать простые операции загрузки и хранения изображения. Вы можете использовать версию write(RenderedImage im, String formatName, File output. Формат будет png. Рендерингованное изображение - это просто под программа буферного изображения из вашего компонента JImage Display (Вам также нужно предоставить публичный доступ этому члену/пользователю)

Конечно, вы заметите что метод write() может выдать исключение, поэтому вам нужно закрыть wrap запрос в блоке try/catch и разобраться с возможной ошибкой. Ваш блок catch должен проинформировать пользователя диалоговым окном. Опять же swing предоставляет класс javax.swing.JOptionPane чтобы упростить процесс создания информационных диалоговых окон или создания ввода да/нет. В таком случае, вы можете использовать статичный метод JOptionPane.showMessageDialog (Component parent, Object message, String title, int messageType) с сообщением JOptionPane.ERROR_MESSAGE. Ваше сообщение об ошибке может быть тем, что исключение возвращает из метода getMessage (), а заглавие может быть наподобие такого «Невозможно сохранить изображение». Родительский компонент должен быть фреймом, чтобы ошибка появлялась в центре фрейма.

Как только вы закончили с этим, опробуйте ваш feature. Теперь вы можете исследовать различные фракталы и видеть их во всей красе, а также вы сможете сохранять их на диск. Вы также можете опробовать ваш написанный код для ошибок, попробовав сохранить картинку в файл, который уже существует, но в котором стоит атрибут только чтение. Вы можете попробовать сохранить файл, чье имя является директорией папки для сохранения.

Задача № 6. Многоядерный исследователь/генератор фракталов

В данной задаче мы закончим с генератором фракталов с еще одной функцией – возможность нарисовать фрактал с многопоточными фоновыми процессами. Это приведет к следующим улучшениям: во-первых, пользовательский интерфейс не зависнет во время рисования нового фрактала, а во-вторых, если ваш компьютер многоядерный, ваш рисунок будет нарисован намного быстрее. Вносить изменения будем, используя встроенную поддержку Swing для фоновых потоков.

До сих пор, осознали ли вы это или нет, все наши фрактальные вычисления и чертежи/рисование были выполнены в Swing's Event-Dispatch Thread. Это поток, который обрабатывает все события Swing, такие, как нажатие кнопок, перерисовка и т. д. Вот почему ваш пользовательский интерфейс зависает во время вычисления фрактала. Поскольку вычисление выполняется в ядре обработке событий (event dispatch thread), никакие события не могут быть обработаны до завершения вычисления.

На этой неделе мы изменим программу так, чтобы она использовала один или несколько фоновых ядер для вычисления фрактала. В частности, ядро обработки dispatch thread не будет использоваться для вычисления фрактала. Теперь, если вычисление будет выполняться несколькими ядрами, нам нужно будет разбить его на несколько независимых частей. При рисовании фракталов это очень легко сделать - мы можем просто дать каждому ядру одну строку вычисления фрактала. Более сложная часть заключается в том, что необходимо следить за важным ограничением Swing, т.е. мы только взаимодействуем с компонентами Swing в ядре обработке событий (event dispatch thread), но, к счастью, Swing снова предоставляет инструменты, чтобы сделать это задание простым.

На самом деле это очень распространенная ситуация в программировании пользовательского интерфейса: интерфейс должен запускать длительную операцию, но операция должна выполняться в фоновом режиме, чтобы пользовательский интерфейс оставался рабочим. Web-браузеры, вероятно, являются наиболее ярким примером этого. В то время как страница загружается и визуализируется, пользователь должен иметь возможность отменить операцию или щелкнуть ссылку, или выполнить любые другие операции. Чтобы облегчить такое взаимодействие, Swing предоставляет класс javax.swing.SwingWorker, который упрощает выполнение задачи в фоновом потоке. SwingWorker - абстрактный класс; Swing ожидает, что вы его расширите и предоставите функциональность для выполнения фоновой задачи. Наиболее важными методами для реализации являются:

DoInBackground () - этот метод фактически выполняет фоновую операцию. Swing вызывает этот метод в фоновом потоке, а не в ядре обработки событий (event dispatch thread);

Done () - этот метод вызывается, когда фоновая задача выполнена. Он вызывается в ядре обработки событий (event dispatch thread), поэтому этому методу разрешено взаимодействовать с пользовательским интерфейсом.

Класс SwingWorker имеет запутанную спецификацию, на самом деле это SwingWorker <T, V>. Тип T - это тип значения, возвращаемого функцией doInBackground (), когда вся задача завершена. Тип V используется, когда фоновая задача возвращает промежуточные значения в процессе работы. Эти промежуточные значения будут отображаться методами publish () и process (). Не всегда необходимо использовать один или оба этих типа; В этих случаях мы можем просто указать Object для неиспользуемого типа (типов).

Рисование в фоновом режиме

Вы будете работать преимущественно с классом FractalExplorer. Некоторые из кодов будут новыми, но некоторые из них будут фактически переведены на код, который вы уже написали.

- Вы должны создать подкласс SwingWorker под названием FractalWorker, который является внутренним классом FractalExplorer. Это самый простой способ написать этот код, так как ему потребуется доступ к нескольким внутренним членам FractalExplorer. Помните, что класс SwingWorker является общим/родовым generic, поэтому вам нужно будет указать параметры - вы можете просто указать Object для них обоих, потому что на самом деле мы не будем их использовать. Поэтому в итоге вы получите следующую строку кода:

  private class FractalWorker extends SwingWorker<Object, Object>.

- Класс FractalWorker будет отвечать за вычисление значений цвета для одного строки/ряда row фрактала, поэтому ему понадобятся два поля: целочисленная y-координата строки/ряда row, которая будет вычислена, и массив значений int для хранения вычисленных значений RGB Для каждого пикселя в этой строке. Конструктор должен взять y-координату в качестве аргумента и сохранить ее. Конструктору больше ничего не нужно будет делать. (В частности, вам не следует выделять массив ints, поскольку он не понадобится, пока строка/ряд не будет фактически вычислена).
- Помните, что метод doInBackground () вызывается в фоновом потоке и отвечает за выполнение долгосрочной задачи. Поэтому в вашей имплементации/реализации вам нужно будет взять часть кода из вашей ранней функции «draw fractal» и поместить ее в этот метод. Конечно, вместо рисования на изображение-экран, петле loop нужно будет сохранить каждое значение RGB в соответствующий элемент целочисленного массива. На самом деле вы не сможете изменять изображение-дисплей из этого ядра, потому что вы нарушите ограничения потоковой обработки Swing!
- Вместо этого выделите массив целых чисел в начале этого метода (он должен быть достаточно большим, чтобы хранить целую строку значений цвета), а затем сохраните цвет каждого пикселя в этот массив. Должно быть очень легко адаптировать код, который вы писали ранее,

единственные различия в том, что вам нужно будет вычислить фрактал для указанной строки и что вы еще не обновляете экран/изображение.
Ваш метод doInBackground () должен возвратить что-то типа Object, так как это и есть объявление declaration SwingWorker <T, V>. Просто верните null!

- Метод done () вызывается, когда фоновая задача завершена, и этот метод вызывается из ядра обработки событий Swing. Это означает, что вы можете модифицировать компоненты Swing в соответствии с содержанием вашего ядра heart. Поэтому в этом методе вы можете просто перебрать массив строк данных, рисуя в пикселях, которые были вычислены в doInBackground (). Проще не бывает.

Как и раньше, когда строка кода закончит рисование, вам нужно будет сообщить Swing, перерисовать часть дисплея/изображения, который был изменен. Поскольку вы изменили только одну строчку, было бы слишком сложно говорить о Swing, чтобы перерисовать весь дисплей! Поэтому вы можете использовать версию JComponent.repaint (), которая позволяет вам указать область для перерисовки. Обратите внимание, что метод немного странный - на переднем плане у него есть неиспользованный long параметр, и вы можете просто указать 0 для этого аргумента. Для остальных просто укажите строку, которая была нарисована - начиная с (0, y) и с размером (displaySize, 1).

После того как вы закончили свой класс фоновой задачи, следующим шагом будет его привязка к процессу рисования фракталов. Вы уже переместили часть своего кода из функции «draw fractal» в рабочий класс, поэтому теперь вы можете изменить свою функцию «draw fractal». Для этого в каждой строке на дисплее создайте отдельный рабочий объект, а затем вызовите execute () для объекта. Это запустит фоновый поток и запустит задачу в фоновом режиме!

Это все изменения, которые вам нужно сделать! Помните, что рабочий класс отвечает за генерацию данных строки и затем рисует строку, поэтому ваша функция «рисовать фрактал» должна быть очень простой.

После завершения и отладки этой функции у вас должен появиться гораздо более быстрый и отзывчивый пользовательский интерфейс. Если у вас несколько ядер, вы обязательно увидите существенное улучшение.

Вы заметите одну проблему с вашим пользовательским интерфейсом - если вы нажмете на экран или на кнопку во время перерисовки, программа обработает его, хотя клик должен быть проигнорирован до завершения операции. К счастью, это довольно просто исправить.

Игнорирование событий во время перерисовки

Самый простой способ решить проблему игнорирования событий во время перерисовки - отслеживать количество оставшихся строк, которые должны быть завершены, и игнорировать или отключать взаимодействия пользователя до тех пор, пока не будут нарисованы все строки. Это нужно делать очень осторожно, иначе у нас будут очень неприятные ошибки. Вот что мы можем сделать - добавить поле «оставшиеся строки» в наш класс Fractal Explorer

и использовать его, чтобы узнать, когда будет завершена перерисовка. Мы будем читать и записывать это значение только из ядра обработки событий, чтобы мы никогда не вводили какие-либо параллельные обращения. Если мы будем взаимодействовать только с ресурсом из одного потока, у нас не будет никаких ошибок параллелизма. Вот что вам нужно сделать.

- Создайте функцию void enableUI (boolean val), которая будет включать или отключать кнопки и выпадающий список вашего интерфейса на основе указанного значения. Для включения или отключения этих компонентов можно использовать метод Swing setEnabled (boolean). Убедитесь, что ваш метод обновляет включенное состояние кнопки сохранения, кнопки сброса и выпадающего списка.

  Ваша функция «draw fractal» должна уметь делать еще две вещи. Во-первых, прежде всего, она должна вызывать enableUI (false), чтобы отключить все элементы пользовательского интерфейса во время рисования. Во-вторых, он должен установить значение «rows rows» в общее число строк, которые должны быть нарисованы. Сделайте это, прежде чем приступать к выполнению каких-либо рабочих задач, иначе это не будет обновлено должным образом.
- В вашем рабочем методе done () уменьшите значение «rows rows» на 1 в качестве последнего шага этой операции. Затем, если после уменьшения оставшихся строк не осталось, вызовите enableUI (true).
- Наконец, измените имплементацию вашего приемника-мышки (mouse-listener), чтобы немедленно возвращаться, если значение «rows rows» не равно нулю. Другими словами, вы будете реагировать на щелчки мышью, только если больше нет строк, которые должны быть нарисованы. Обратите внимание, что нам не нужно делать аналогичные изменения в обработчике события действия, потому что мы отключаем все эти компоненты с помощью метода enableUI ().)

После того как вы выполните эти шаги, у вас должна получиться красивая программа отрисовки фракталов, которая может рисовать фракталы с несколькими потоками и это не позволит пользователям ничего делать, пока процесс рендеринга происходит в фоновом режиме.

Все сделано!

Задача № 7. Серфинг в Интернете

В этой задаче, вы будите писать код рудиментарного поискового робота. Ваш робот будет автоматически загружать веб-страницы из Интернета, искать новые ссылки на этих страницах и повторять. Сканер будет примерно таким простым, каким только можно себе представить: он будет просто искать новые URL-адреса (местоположения веб-страниц) на каждой странице, собирать их и распечатывать в конце. Более сложные веб-сканеры используются для того, чтобы делать такие вещи, как индексирование содержимого Интернета или очистка адресов электронной почты от спама; Если вы когда-либо использовали поисковую систему, вы запрашивали данные, генерируемые поисковым роботом.

Термины

- **URL:** унифицированный указатель ресурса. Это адрес веб-страницы. В нашем случае он состоит из строки, за которым следует местоположение веб-сервера, а затем путь к веб-странице на сервере. Если последнее имя в пути не заканчивается на ".html", то фактически веб-страница предоставляется сервером. Это может быть "index.html", "index.shtml", "index.php", "default.htm", или что-либо другое, которое веб-сервер считает документом «по умолчанию» для конкретного каталога.
  Примечание: существуют и другие допустимые URL-адреса, начиная с (например) "mailto://" или "ftp://". Мы не будем беспокоиться об этом для этого задания.
- **HTTP:** Hyper Text Transfer Protocol (Протокол передачи гипертекста). Это стандартный текстовый протокол, используемый для передачи данных веб-страницы через Интернет. Последней спецификацией HTTP является версия 1.1, которую мы и будем использовать.
  Обратите внимание, что HTTP-запрос ДОЛЖЕН оканчиваться пустой строкой, иначе запрос будет проигнорирован. Также обратите внимание на заглавные буквы. Если вы попытаетесь отправить «Get» или «host», то веб-сервер будет вами недоволен.
  Некоторые URL-адреса не указывают документ или ресурс для извлечения. В этих случаях вы должны указать «/» в качестве ресурса для извлечения. Другими словами, запрашиваемый ресурс всегда будет начинаться с символа «/».
- **Socket(Сокет):** Сокет (разъем) - это ресурс, предоставляемый операционной системой, который позволяет вам обмениваться данными с другими компьютерами по сети. Вы можете использовать сокет для установки соединения с веб-сервером, но вы должны использовать сокет TCP и «говорить» HTTP-протокол для того, чтобы сервер мог ответить.
- **Port(Порт):** несколько разных программ на одном сервере могут прослушивать соединения, прослушивая разные порты. Каждый порт обозначается номером в диапазоне 1..65535. 1 - 1024 зарезервированы для операционной системы. У большинства видов серверов есть порт по умолчанию. Для HTTP-соединений мы обычно используем порт 80.

Программа для записи

Вот описание программы, которую вы должны написать.

1. Программа должна принимать в командной строке два параметра:
   - строку, представляющую URL, с которого можно начать просмотр;
   - положительное целое число, представляющее максимальную глубину поиска (см. ниже).

Если правильные аргументы не указаны, программа должна немедленно остановить и распечатать сообщение об использовании, например:

usage: java Crawler <URL> <depth>

2. Программа должна хранить URL в виде строки вместе со своей глубиной (которая равна 0 для начала). Вы должны создать специальный класс для представления пар [URL, depth].

3. Программа должна подключиться к данному сайту в URL-адресе на порт 80 с помощью сокета (см. ниже) и запросить указанную веб-страницу.

4. Программа должна анализировать возвращаемый текст, если он есть, построчно для любых подстрок, имеющих формат:

<a href="[любой URL начинающийся с http://]">

Найденные URL-адреса должны быть сохранены вместе с новым значением глубины в пары LinkedList (URL, depth) (подробнее о LinkedLists см. ниже). Новое значение глубины должно быть больше, чем значение глубины URL-адреса, соответствующего анализируемой странице.

5. Далее программа должна закрыть соединение сокета с хостом.

6. Затем программа должна повторять шаги с третьего по шестой для каждого нового URL-адреса, если глубина, соответствующая URL-адресу, меньше максимальной. Обратите внимание, что при получении и поиске определенного URL глубина поиска увеличивается на 1. Если глубина URL-адреса достигает максимальной глубины (или больше), не извлекайте и не просматривайте эту веб-страницу.

7. Наконец, программа должна распечатать все URL, посещенные вместе с их глубиной поиска.

Допущения

Это довольно сложно разобрать, а тем более подключить, ко всем правильно и неправильно сформированным гиперссылкам в Интернете. Предположим, что каждая ссылка/ссылки правильно сформированы, с полностью квалифицированным именем хоста, ресурсным путем и всеми вышеперечисленными параметрами. Кроме того, на удивление, есть несколько

больших сайтов, которые имеют много URL-адресов этой формы; вы можете попробовать http://slashdot.org/ или http://www.nytimes.com. (Заключительный слэш на slashdot.org важен)

Обратите внимание, что один из наиболее распространенных видов URL-адресов, который неприемлем, - это тот, который начинается с чего-то, кроме «http: //». Общие примеры включают «mailto: //» и «ftp: //». Если вы найдете их, вы должны их игнорировать.

Предположим, когда ваш BufferedReader возвращает значение null, сервер завершил отправку веб-страницы. На самом деле это может быть неверным для очень медленных веб-серверов, но для наших целей это должно быть вполне приемлемым.

Полезные классы и методы

Классы и методы должны помочь вам начать работу. Обратите внимание, что большинство этих методов выбрасывают различные виды исключений, с которыми вам придется работать. Опять же, посмотрите Java API, чтобы узнать, что это такое.

Сокет(разъем)

Чтобы использовать сокеты, вы должны включить эту строку в свою программу:

*Import java.net. *;*

Конструктор

Socket (String host, int port) создает новый сокет из строки, представляющей хост и номер порта, и устанавливает соединение.

Методы

- *Void setSoTimeout (int timeout)* задает тайм-аут сокета в миллисекундах. Вы должны вызвать это после создания сокета, чтобы он знал, сколько времени ждать передачи данных с другой стороны. В противном случае он будет ждать вечно, что, вероятно, не является хорошей идеей, если мы хотим использовать эффективный сканер.
- *InputStream getInputStream ()* возвращает InputStream, связанный с Socket. Это позволяет Socket получать данные с другой стороны соединения.
- *OutputStream getOutputStream ()* возвращает *OutputStream*, связанный с Socket. Это позволяет Socket отправлять данные на другую сторону соединения.
- *Void close ()* закрывает *Socket*.

Потоки

Чтобы использовать потоки, вы должны включить эту строку в свою программу:

*Import java.io. *;*

Чтобы эффективно использовать сокеты, вы захотите преобразовать InputStream и OutputStream, связанные с Socket, в нечто более полезное. Объекты InputStream и OutputStream являются очень примитивными объектами. Они могут читать только байты или массивы байтов (даже не символы). Поскольку вы хотите читать и писать символы, вы должны иметь объекты, которые преобразуют байты и символы и печатают целые строки. К сожалению, Java API делает это несколько различными способами для ввода и вывода.

Входные потоки

Для входных потоков вы можете использовать классы InputStreamReader следующим образом:

InputStreamReader in = new InputStreamReader (my_socket.getInputStream ());

Теперь in является InputStreamReader, который может читать символы из Socket. Тем не менее, это все еще не очень дружелюбно, потому что вам по-прежнему приходится работать с отдельными символами или массивами символов. Было бы неплохо читать целые строки. Для этого вы можете использовать класс BufferedReader. Вы можете создать BufferedReader, указав экземпляр InputStreamReader, а затем вызвать метод readLine в BufferedReader. Это будет прочитано в целой строке с другого конца сокета.

Потоки вывода

Потоки вывода немного проще. Вы можете создать экземпляр PrintWriter непосредственно из объекта OutputStream сокета, а затем вызвать его метод println для отправки строки текста на другой конец сокета. вы должны использовать этот конструктор:

PrintWriter (OutputStream out, boolean autoFlush)

с установкой autoFlush на true. Это очистит буфер вывода после каждого println.

Строковые методы

Вы найдете эти методы String полезными. См. документацию по API.

- Boolean equals (Object anObject)
- String substring (int beginIndex)
- String substring (int beginIndex, int endIndex)
- Boolean startsWith (String prefix)

ПРИМЕЧАНИЕ. Не используйте == для сравнения строк для равенства! Он будет возвращать true, только если две строки будут одним и тем же объектом. Если вы хотите сравнить содержимое двух строк, используйте метод equals.

Списки

Списки очень похожи на массивы объектов, за исключением того, что они могут легко расширяться или сокращаться при необходимости, и у них нет

скобки-обозначения для поиска отдельных элементов. Чтобы использовать Списки, вы должны включить эту строку в свою программу:

*Import java.util. *;*

Вы должны хранить пары (URL, depth) в LinkedList, которые является конкретной реализацией List. Создайте его следующим образом:

LinkedList <URLDepthPair> myList = новый LinkedList <URLDepthPair> ();

А затем увидеть API для множества полезных методов в списках и различных реализаций списков. (В частности, вы заметите, что разные реализации List предоставляют разные функции. Именно поэтому рекомендуется LinkedList, некоторые из его функций особенно хорошо подходят для этого назначения.)

Специальный синтаксис для создания LinkedList выше, использует новую поддержку Java 1.5 generics. Этот специальный синтаксис означает, что вам не нужно бросать объекты, которые вы храните или извлекаете из списка.

Исключения

Когда вы найдете что-то, похожее на URL-адрес, то не начинайте с «http: //», вы должны выдать MalformedURLException, который является частью Java API.

Советы по проектированию

Ниже приведены некоторые рекомендации по разработке вашего поискового робота.

Пары URL-глубины

Как упоминалось выше, вы должны создать специальный класс URLDepthPair, каждый экземпляр которого включает в себя поле String, представляющее URL, и int, представляющее глубину поиска. Вы также должны иметь метод toString, который распечатает содержимое пары. Это значительно упрощает вывод результатов вашего веб-сканирования.

Отдельные URL-адреса необходимо разбить на части. Этот синтаксический разбор URL и манипуляция должны быть частью созданного вами класса пар URL-глубины. Хороший объектно-ориентированный дизайн диктует, что если какой-то конкретный класс собирается хранить определенный тип данных, тогда любые манипуляции с этими данными также должны быть реализованы в этом классе. Итак, если вы пишете какие-либо функции для разрыва URL-адреса или для проверки, является ли URL-адрес допустимым, поместите его в этот класс!

Сканеры

Как уже упоминалось выше, вы должны спроектировать класс Crawler, который будет реализовывать основные функциональные возможности приложения. Этот класс должен иметь метод getSites, который будет возвращать

список всех пар URL-глубины, которые были посещены. вы можете вызвать это в вашем основном методе, как только обход завершен; Получить список, затем выполнить итерацию и распечатать все URL-адреса.

Самый простой способ отслеживания посещенных сайтов состоит в том, чтобы иметь два списка, по одному для всех сайтов, рассмотренных до сих пор, и один, который включает только сайты, которые еще не обработаны. Вам следует проделать итерацию по всем сайтам, которые не были обработаны, удалить каждый сайт перед загрузкой его содержимого, и каждый раз, когда вы находите новый URL, вы должны поместить его в необработанный список. Когда необработанный список пуст, вы все сделали - вы нашли все сайты.

Хотя вы можете подумать про себя: «Открытие сокета по URL-адресу - операция, связанная с URL-адресом, и, следовательно, должна быть реализована в классе пар URL-глубины», что было бы слишком специализированным для целей парного класса. Это действительно просто место для хранения URL-адресов и значений глубины, а также нескольких дополнительных утилит. Искатель - это класс, который перемещает веб-страницы и ищет URL-адреса, поэтому класс искателя должен содержать код, который фактически открывает и закрывает сокеты.

Вам нужно будет создать новый экземпляр Socket для каждого URL, с которого вы загружаете текст. Обязательно закройте сокет, когда вы закончите сканирование этой веб-страницы, чтобы операционная система не исчерпала сетевые ресурсы! (Существует так много сокетов, что компьютер может оставаться открытым сразу). Кроме того, не используйте рекурсию для поиска более глубоко вложенных веб-страниц; Реализовать эту функцию в виде цикла. Это также сделает ваш поисковый робот более эффективным с точки зрения использования ресурсов.

Константы!

В вашей программе, несомненно, будут строки типа *«http: //» и «a href = \»*, и вы, вероятно, будете испытывать желание просто повторять эти строки везде, где вам это нужно. Кроме того, вам понадобятся эти длины для разных строк операций, так что у вас также возникнет соблазн жестко «закодить» длины этих строк в ваш код. Не делайте этого! Это делает код очень неподдерживаемым! Если вы делаете опечатку или позже меняете поиск, вам придется обновлять много разных строк кода.

Вместо этого создайте строковые константы в своих классах. Например, у вас может быть следующее:

Public static final string URL_PREFIX = "http: //";

Теперь, если вам нужна эта строка, вместо того, чтобы напрямую ее кодировать, используйте константу URL_PREFIX. Если вам нужна его длина, вам повезло - URL_PREFIX является объектом String, поэтому вы можете вызвать URL_PREFIX.length () и вернуть его длину.

Вы также должны думать о том, где вы ставите эти константы. Вам нужно, чтобы каждая константа отображалась один раз во всем вашем проекте, и вы должны поместить константу там, где она имеет смысл. Например, поскольку префикс URL необходим, чтобы определить, действителен ли URL-адрес, вы должны поместить эту константу в свой класс пары URL-глубины. Если у вас есть еще одна константа для HTML-ссылок, поместите ее в свой класс искателя. Если поисковик когда-либо нуждается в префиксе URL, он может просто ссылаться на константу пары URL-depth, вместо того, чтобы дублировать эту константу.

Дополнительное задание

- Добавьте код, чтобы добавлять сайты в неподготовленный список, если они не были замечены ранее.
- Расширьте возможности поиска гиперссылок поискового робота, используя поиск регулярных выражений на собранных данных. Вам также понадобится больше логики, чтобы решить, какую машину подключить к следующей. Ваш искатель должен иметь возможность перемещаться по ссылкам на различных популярных сайтах. Использование регулярных выражений требует больше знаний, но на самом деле гораздо проще, чем вручную искать подстроки в строках.
- Создайте пул из пяти (или более!) Искателей, каждый в своем потоке, каждый из которых может получить URL для просмотра, и каждый из них вернет список ссылок по окончании. Посылайте новые URL-адреса в этот пул, как только они становятся доступными.
- Расширьте свой многопоточный поисковый робот, чтобы выполнить поиск на глубину до 1 000 000. Сохраняйте результаты каждого обхода в базе данных через JDBC и обратите внимание, сколько раз каждая конкретная уникальная страница ссылается на другие. Включите интеллектуальный алгоритм для «поиска смысла» на каждой странице путем взвешивания слов и фраз на основе повторяемости, близости к началу абзацев и разделов, размера шрифта или стиля заголовка и мета-ключевых слов, по крайней мере.

Задача № 8: лучший Web Crawler

Прошлый поисковый робот не был особенно эффективен. Вы расширите свой поисковый робот, чтобы использовать поточную обработку Java, так, чтобы многократные веб-страницы могли быть проверены параллельно. Это произведет значительное повышение производительности, потому что время, которое каждый поток поискового робота тратит на ожидание сетевых операций, чтобы завершить их, может быть перекрыт с другими операциями по обработке в других потоках.

Расширение Web Crawler

Вы расширите и измените свою предыдущую программу.

- Реализуйте класс под названием URLPool, который сохранит список всех URL, которые будут искать, вместе с относительным "уровнем" каждого из тех URL (также известный как "поисковая глубина"). Первый URL, который вы ищете, будет на поисковой глубине 0, URL, найденные на той странице, будут поисковой глубиной 1 и т.д. вы должны сохранить URL и их поисковую глубину вместе, как экземпляры класса под названием URLDepthPair, как вы делали ранее. LinkedList рекомендуют для хранения элементов, так как это поможет выполнить требуемые операции очень эффективно.
- Должен быть путь к пользователю класса URLPool, чтобы выбрать пару глубины URL от пула и удалить его из списка одновременно. Должен также быть способ добавить пару глубины URL к пулу. Обе из этих операций должны быть ориентированы на многопотоковое исполнение, так как многократные потоки будут взаимодействовать с URLPool одновременно.
- В отличие от примера FIFO, обсужденного в классе, у пула URL не должно быть максимального предела размера. Этому пулу действительно просто нужны списки незаконченных URL, спискм отсканированных URL.
- Чтобы выполнить веб-проверку в многократных потоках, вы должны создать класс CrawlerTask, который реализует интерфейс Runnable. У каждого экземпляра CrawlerTask должна быть ссылка на один экземпляр класса URLPool, описанного выше. (Обратите внимание на то, что вся доля экземпляров CrawlerTask это единственный пул!) Задание поискового робота должно быть таким:

  1) получите пару глубины URL от пула, ожидая, если вы не сразу доступны;

  2) получите веб-страницу, упомянутую URL;

  3) ищите страницу больше URL. Для каждого URL, найденного страницей, задача поискового робота должна добавить новую пару глубины URL к пулу URL;

  4) вернитесь к шагу 1.

Это должно продолжаться, пока больше нет (URL, глубина) пар в пуле, чтобы проверить.

Так как Ваш поисковый робот будет порождать некоторое количество потоков поискового робота, обновить Вашу программу, чтобы принять третий параметр командной строки, определяя количество потоков поискового робота. Ваша основная функция будет управлять чем-то вроде этого:

1) обработайте параметры командной строки. Сообщение пользователю о любых входных ошибках;
2) создайте экземпляр пула URL и поместите определенный пользователями URL в пул с глубиной 0;
3) создайте количество задач поискового робота (и потоки, чтобы выполнить их) требуемых пользователю. Каждой задаче поискового робота нужно дать ссылку на пул URL, который вы инсталлировали;
4) ожидайте веб-проверки, чтобы завершиться;
5) распечатайте получающийся список URL, которые были найдены.

Удостоверьтесь, что синхронизировали свой объект пула URL в любой и всех критических точках, так как это должно теперь быть ориентировано на многопотоковое исполнение.

Поисковые роботы, постоянно опрашивают пул URL относительно другого URL. Вместо этого делайте, чтобы ожидание было эффективно, когда никакой URL не будет доступен. Вы должны реализовать, это при наличии пула URL "добирается, URL" метод ожидает () внутренне, если никакой URL не в настоящее время доступен. Соответственно, пул URL, "добавляя URL" метод, должен уведомить (), когда новый URL добавлен к пулу.

Обратите внимание на то, что потоки поискового робота не будут самостоятельно выступать, любой из них синхронизируется / ожидает / уведомляет операции. Это по той же причине, что пул URL скрывает детали того, как URL сохранены и получены: инкапсуляция! Точно так же, как вы хотите, чтобы пользователи пула URL не волновались о деталях реализации.

Совет проекта

Вот некоторые советы для успешного выполнения задачи №7:

Вы можете использовать некоторые части кода с прошлого выполнения, но с небольшим изменением. Класс URLDepthPair не должен быть изменен вообще. Также, большая часть веб-кода может быть снова использована. Основные отличия - то, что код URL-download/page-crawl находится теперь в классе и будет получать и добавлять ссылки на экземпляр URLPool.

Вы должны синхронизировать доступ к внутренним полям URLPOOL, поскольку к нему будут обращаться из нескольких потоков. Как обсуждалось в классе, самый простой подход заключается в использовании synchronized методов, а не в попытке синхронизировать части кода внутри методов класса.

Вам не нужно синхронизировать конструктор URLPool. Подумайте о том, какие методы должны быть синхронизированы.

Опишите методы вашего URLPOOL, для того чтобы использовать wait() и notify() так, чтобы потоки поисковика могли ожидать новые URL, чтобы стать доступными.

Дополнительное задание

- Обновите свою пару глубины URL, чтобы использовать java.net. Класс URL и обновление Ваш поисковый робот, чтобы следовать за относительными URL, а также абсолютными URL.
- Выход из вашей программы System.exit () не самый лучший. После проверки всей программы найдите более быстрый способ выхода.
- Реализуйте список URL-адресов, которые не будут просмотрены, и избегайте возврата к старым ссылкам. Используйте один из классов коллекций java, чтобы сделать это проще. Какой-то набор, который поддерживает постоянное время поиска и вставки, будет наиболее подходящим.
- Добавьте другой дополнительный параметр командной строки, чтобы определить, сколько времени поток поискового робота должен ожидать сервера, чтобы возвратить требуемую веб-страницу. Используйте это время в качестве старта для запуска получения страницы, и затем поддерживайте дополнительный параметр maxPatience, указывающий точку, в которой задача поискового робота прервет проверку страницы и перейдет к следующей.