Язык программирования Java

Лекция 4

Перевод курса CS11 Java Track

Файловые объекты

- Java работает с файлами с помощью класса java.io.File
  - Классу можно указывать относительный или абсолютный путь к файлу
- Абсолютный путь начинается в корневом каталоге файловой системы
  - “C:\Documents and Settings\Donnie Pinkston\Desktop\Foo.java”
    - Заметим, что символ “\” вставляется в строки Java с помощью esc последовательности
  - Или “/home/Donnie/Desktop/Foo.java”.
- Относительные пути начинаются в текущем каталоге
  - “.” можно использовать для указания текущего каталога
  - “..” для ссылки на родительский каталог текущего каталога

Файловые объекты (2)

- В классе java.io.File объявлены несколько констант
  - File.separator типа String содержит символ разделителя в пути файлов
    - В Windows это “\\” в Unix системах “/”
  - Тот же символ типа char задается константой File.separatorChar
- Есть также константа задающая разделитель между несколькими путями
  - File.pathSeparator это строка содержащая символ разделитель
    - В Windows это “;” в Unix “:”
  - Эти константы полезны в случае, если вы программно генерируете путь к классу или коллекцию путей к файлам или каталогам

Создание объектов файлов

- Класс File имеет несколько конструкторов:
  - File(String pathname)
    - Указывается абсолютный или относительный путь к файлу.
  - File(File parent, String child)
    - Parent в этом случае то каталог, в котором находится файл child
  - File(String parent, String child)
    - Тоже самое, но parent – имя каталога
- Эти конструкторы не проверяют, существует ли указанный файл

Информация о файловых объектах

- Для проверки файлов в классе имеется много полезных методов:
  - boolean exists()
    - Проверяет существует ли в файловой системе файл или каталог соответствующий объекту
  - boolean isFile()
    - Поверяет является ли объект “нормальным” файлом (а не каталогом)
  - boolean isDirectory()
    - Проверяет является ли объект каталогом.
  - boolean canRead()
    - Существует ли файл и можно ли прочитать его содержимое.
  - boolean canWrite()
    - Существует ли файл и можно ли в него записать данные.
  - long length()
    - Возвращает длину файла

Управление файлами

- Класс выполняет основные операции с файлами, такие как:
  - boolean delete()
    - Удаляет файл или каталог (если он пуст). Возвращает true если операция выполнена, иначе false
  - boolean renameTo(File dest)
    - Перемещает файл
    - Может не сработать, если файл уже существует или копируется в файловую систему другого типа

Навигация по файловой системе

- Класс File можно использовать для навигации по файловой системе:
  - File[] File.listRoots()
    - Статический метод возвращает массив объектов File в указанном каталоге
  - File[] listFiles
    - Метод возвращает массив объектов файлов в указанном каталоге
  - (поговорим о массивах в языке Java позже).
- Метод listFiles() может использовать фильтр для поиска файлов
  - Для исключения файлов из возвращаемого этим методом списка на основании заданных критериев, надо реализовать интерфейс FilenameFilter или FileFilter

Потоковый ввод/вывод

- В Java используется механизм потокового ввода/вывода
  - java.io.InputStream и java.io.OutputStream
    - Абстрактные классы включающие объявления всех операций с потоками
  - Поток, обычно, открывается с помощью какого либо особенного для этого вида потока механизма
    - Например, открывается файл и открывается доступ к его входному потоку
    - Или открывается сетевое соединение и открывается доступ к выходному потоку для передачи данных и к входному потоку для приема данных

Потоковый ввод/вывод (2)

- Методы класса InputStream:
  - read() читает один или несколько байтов
    - Блокирующий метод: не возвращает управление, до тех пор, пока данные не будут получены или пока не станет ясно, что чтение невозможно
  - available() возвращает количество байтов которое можно прочитать без блокировки
  - close() закрывает входной поток
    - Освобождает все ресурсы, связанные с потоком
- Методы класса OutputStream:
  - write() записывает в поток один или несколько байтов
  - flush() заставляет передать содержимое внутренних буферов записи Java операционной системе. (ОС тоже может буферизировать эти данные)
  - close() закрывает выходной поток

Потоковый ввод/вывод (3)

- InputStream и OutputStream это бинарные потоки
  - Данные передаются байтами
  - Как правило потоки такого типа не подходят для текстовых данных (особенно для данных зависящих от настроек языка)
- С символьными данными работают интерфейсы java.io.Reader и java.io.Writer
  - Они имеют тот же набор операций, но для значений типа char

Потоковый ввод/вывод (4)

- В Java поддерживается комбинирование потоков
  - Например: для чтения строк текстового файла его можно сначала открыть с помощью класса FileInputStream (потомка InputStream):

```
FileInputStream fis = new FileInputStream("foo.txt");
```

  - Затем “обернуть” входной поток в Reader для того чтобы считывать символьные данные:

```
InputStreamReader isr = new InputStreamReader(fis);
```

  - И добавить буферизацию, чтобы можно было считывать строки целиком:

```
BufferedReader br = new BufferedReader(isr);
```

- (Функции ввода/вывода Java немного раздражают …)

Потоковый ввод/вывод и исключения

- Объекты файлов информируют об ошибках через значения возвращаемые методами
  - boolean delete()
  - boolean renameTo(File dest)
- Большинство операций ввода/вывода для сообщения об ошибках используют *механизм исключений*
  - Обычно используется класс java.io.IOException или его потомки

Исключения

- В некоторых случаях код может обнаружить ошибку, но не в состоянии ее исправить
  - Например, FileInputStream способен обнаружить, что файл нельзя открыть, но что он может с этим поделать?
- Есть несколько способов сообщить об ошибке вызывающему коду:
  - Вернуть код ошибки
    - … если конечно все произошло не в конструкторе, который не имеет возвращаемого значения
  - Вызвать исключение для сообщения об ошибке
- Исключение прерывает текущие вычисления
  - Управление немедленно передается коду обработчика

Вызов исключений

- Вызвать исключение просто:

```
public double computeValue(double x) {
    if (x < 3.0) {
        throw new IllegalArgumentException(
                "x должен быть >= 3, получен " + x);
    }
    return 0.5 * Math.sqrt(x – 3.0);
}
```

- Создается новый объект исключения, и затем исключение вызывается
- Исключение содержит информацию о стеке
  - В месте, где оно было создано (а не где оно было вызвано …)
  - Лучше всего создавать и вызывать исключения в одном и том же месте

Вызов исключений (2)

- Когда исключение вызывается, управление немедленно передается обработчику этого исключения

```
public double computeValue(double x) {
    if (x < 3.0) {
        throw new IllegalArgumentException(
                "x должен быть >= 3, получен " + x);
    }
    return 0.5 * Math.sqrt(x – 3.0);
}
```

- В этом примере, после вызова исключения больше никакой код в функции не исполняется
- Для исключения можно задать текст сообщения об ошибке
  - В сообщении следует написать, что ожидали получить, и что случилось на самом деле.

Обработчики исключений

- Для обработки исключения код должен его перехватывать

```
void main(String[] args) {
    double x = getDouble();
    try {
        double result = computeValue(x);
        System.out.println("Результат:" + result);
    }
    catch (IllegalArgumentException e) {
        System.out.println("Неправильный результат: " +
                                e.getMessage());
    }
}
```

- Код внутри блока try может привести к вызову исключения
- Блок catch обрабатывает ошибки в случае их появления
  - Ошибки типа IllegalArgumentException

Обработчики исключений (2)

- Если исполнение функции computeValue приведет к вызову исключения, управление *немедленно* будет передано в блок catch с соответствующим типом исключения

```
void main(String[] args) {
    double x = getDouble();
    try {
        double result = computeValue(x);
        System.out.println("Результат:" + result);
    }
    catch (IllegalArgumentException e) {
        System.out.println("Неправильный результат: " +
                                e.getMessage());
    }
}
```

- Вместо результата на экране будет выведено сообщение об ошибке

Обработчики исключений (2)

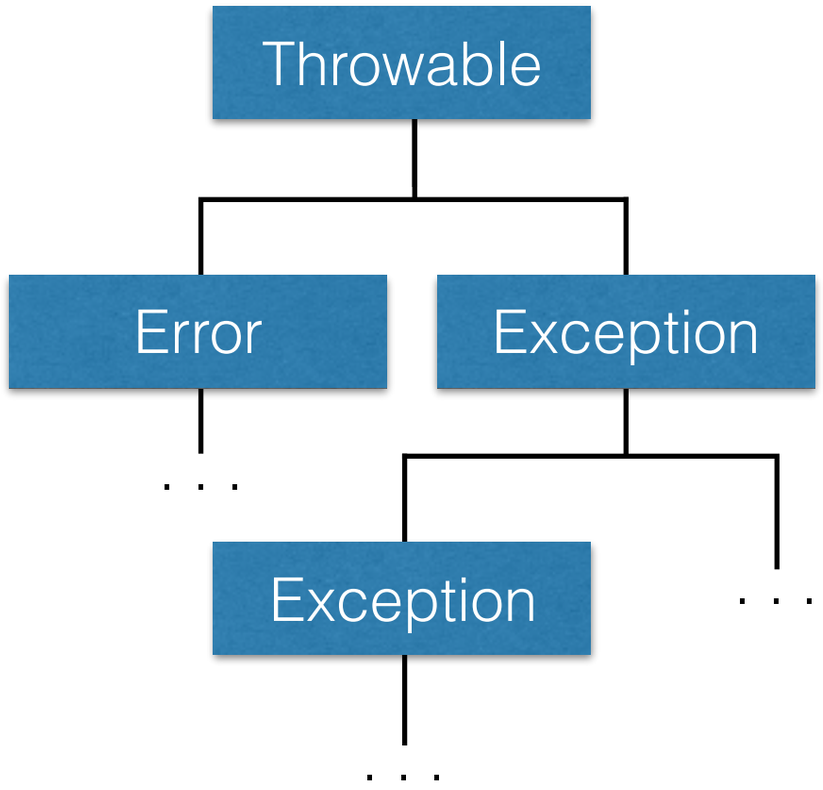
- Для обработки возможных исключений в коде, этот код следует поместить в блок try
  - Блок try не может обрабатывать исключения, возникающие за его пределами
- Тип исключения определяет блок catch, который должен обработать это исключение
  - Один или несколько блоков catch должны располагаться сразу за блоком try
  - Первый блок, тип исключения которого совпадает с типом вызванного исключения, обрабатывает это исключение.
  - После завершения исполнения блока catch исполнение программы возобновляется, начиная с места расположенного сразу за блоками try/ catch (всегда выполняется только один блок catch)

Исключения Java

- Java имеет ограничения касающиеся обработки исключений:
  - Классами исключений могут быть только экземпляры класса java.lang.Throwable или его потомков
  - В общем случае, методы должны объявлять, какие типы исключений они могут вызвать
    - Это еще один пример того как Java следит за корректностью кода
    - Программа должна обрабатывать исключения или явно указывать, какие типы исключений могут быть вызваны

Иерархия исключений Java

- Throwable
  - Базовый класс для всех классов исключений Java
- Error
  - Серьезные проблемы в виртуальной машине Java Приложения обычно не должны обрабатывать эти исключения
- Exception
  - Стандартные, рядовые ошибки, которые приложение обычно обрабатывает самостоятельно.
- RuntimeException
  - Приложение может обрабатывать или не обрабатывать эти ошибки, обычно указывающие на ошибки в программировании



Проверяемые исключения

- Проверяемым исключением может быть любой класс исключения
  - Наследованный от класса Exception или его дочерних классов, за исключением класса RuntimeException и его потомков
- Метод должен явно указывать исключения, которые он может вызвать:

```
import java.io.IOException;
public String getQuote() throws IOException {
    ...
    if (errorOccurred)
        throw new IOException("Ошибка!");
    return quote;
}
```

- Компилятор Java проверяет код на соответствие этой спецификации
- Исключения времени исполнения (RuntimeException) можно также указывать в объявлении методов, но это не обязательно

Проверяемые исключения (2)

- В методе можно указать базовый класс вызываемого исключения

```
public String getQuote() throws IOException {
    ...
}
```

  - Все эти ошибки наследованы от IOException:
    - UnknownHostException (неизвестно имя хоста)
    - EOFException (не ожидаемый конец файла)
    - SocketException (общая проблема сокетов)
- Метод в приведенном выше примере, может вызывать эти исключения, не меняя своего описания
  - Код метода может также перехватывать исключения родительских классов, то есть в нашем случае, IOException и выше

Какие исключения надо обрабатывать?

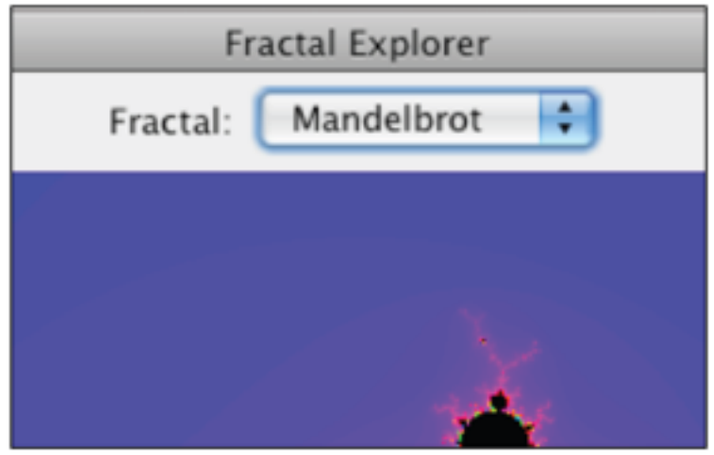
- В документации Java API перечислены классы исключений
  - И указаны случаи, когда вызываются эти исключения
- Сетевые библиотеки и библиотеки обслуживающие ввод/вывод могут вызывать множество исключений
- Библиотеки управления исполнением потоков также вызывают ряд исключений
- Всегда важно правильно обрабатывать исключения, для того чтобы ваше приложения было устойчиво к ошибкам

Задание

- На этой неделе добавим немного новых функций в вашу программу для изучения фракталов:
  - Возможность отображать несколько фракталов
    - Выбор фрактала сделаем выпадающим списком
  - Возможность сохранять изображение фрактала на диск
- Обе функции добавить не сложно
  - Для этого мы можем воспользоваться различными функциями Java API

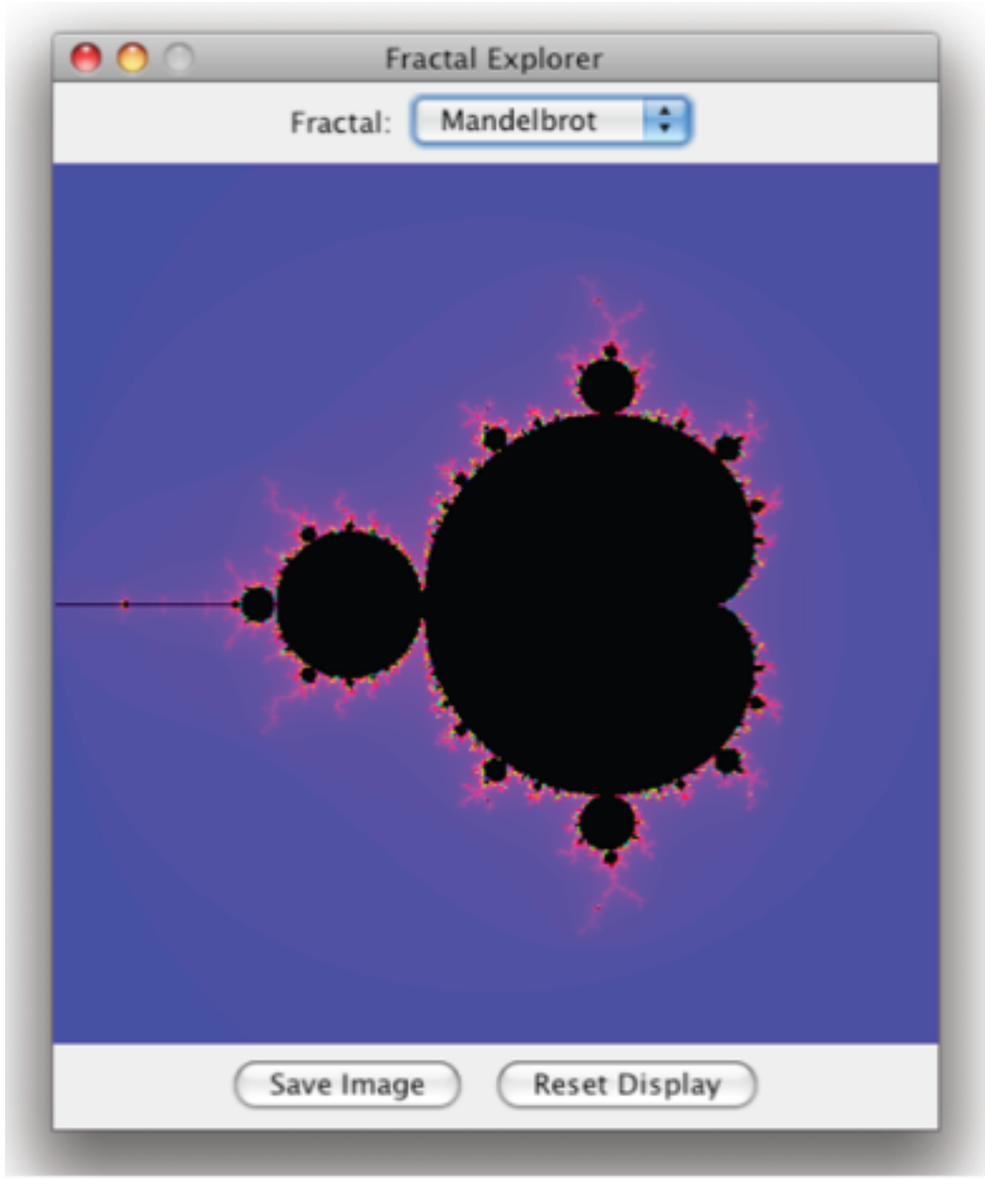
Несколько видов фракталов

- Большинство графических библиотек имеют в своем составе выпадающие списки
  - Позволяют выбирать из списка вариантов
- Swing имеет класс JComboBox
  - Его очень легко настроить и использовать
  - При выборе элемента списка он вызывает событие ActionEvent



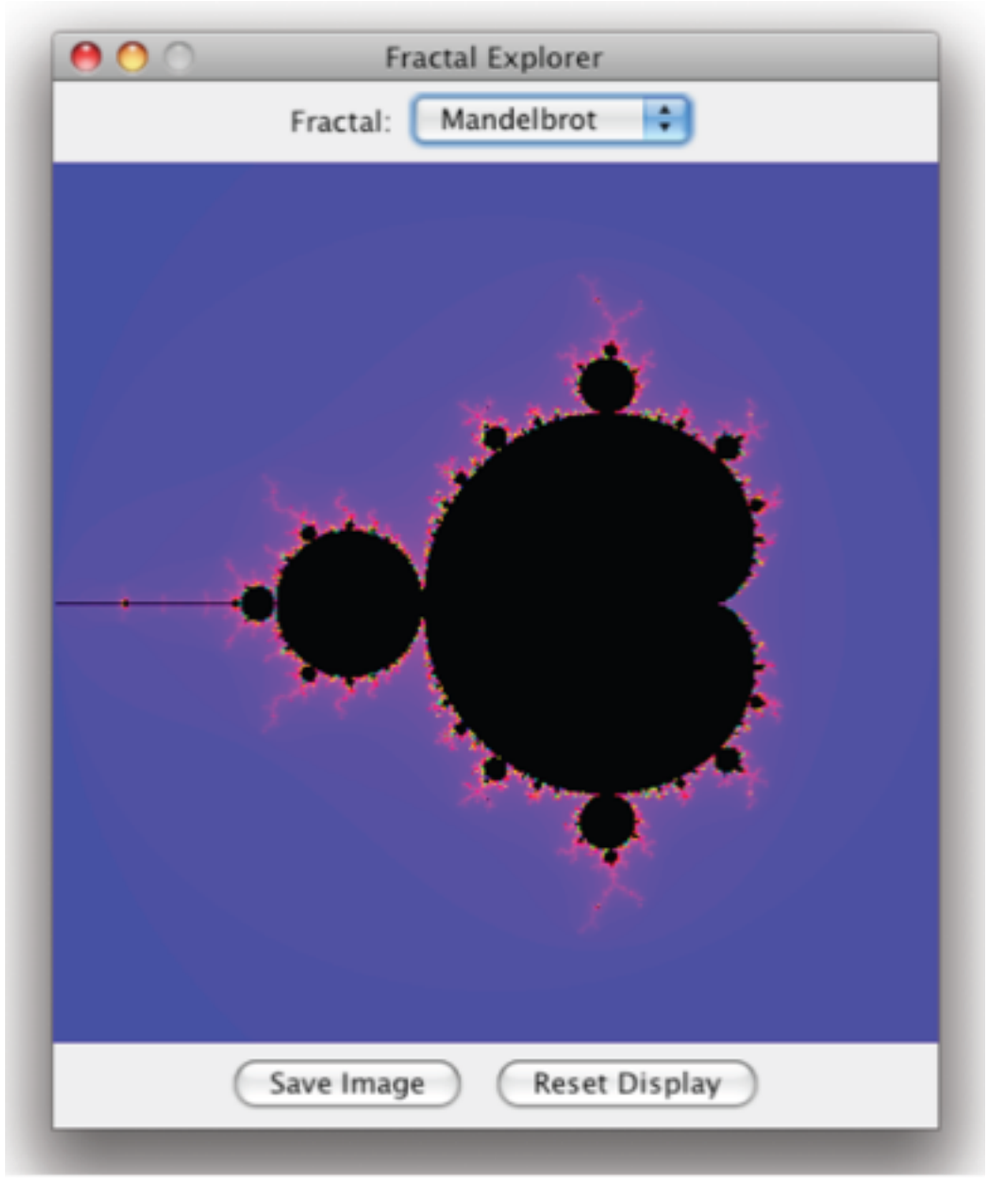
Сохранение изображений

- Добавим к интерфейсу кнопку для сохранения текущего изображения
- Swing имеет полезные классы:
  - JFileChooser, который позволяет выбрать файл для открытия или сохранения
  - JOptionPane для вывода диалоговых сообщений, если что то пойдет не так ☺



Сохранение изображений (2)

- Теперь появилось несколько источников событий
- Как правило, желательно ограничить общее количество объектов создаваемых программой
- Цель:
  - Создать один класс обработчик, который будет обслуживать события от всех источников



Команды действий

- Большинство компонентов, которые порождают события ActionEvent, имеют поле, в котором можно указать команду, обозначающую это действие
  - Используйте это поле для того чтобы указать назначение или действие совершаемое источником события

```
JButton saveButton = new JButton(“Сохранить изображение");
saveButton.setActionCommand("save");
```

- У других источников будут собственные значения команд
  - Значение команды передается в классе ActionEvent
    - Его можно получить, вызвав метод getActionCommand()
  - Теперь ActionListener может обрабатывать события от разных источников, и выполнять нужное действие в зависимости от значения команды

Пример обработки событий нескольких источников

```
void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();
    if (e.getSource() == fractalChooser) {
        ... // получить выбранный пользователем фрактал
        ... // и нарисовать его.
    }
    else if (cmd.equals("reset")) {
        ... // Сбросить изображение.
    }
    else if (cmd.equals("save")) {
        ... // Сохранить текущее изображение фрактала.
    }
}
```