Программирование на языке Java. Часть 2.

Курс для самостоятельного изучения

Темы на сегодня

- Утверждения
- Аннотации Java 1.5
- Classpath
- Модульное тестирование!
- Подсказки для второго задания ☺

Утверждения!

- Утверждения это очень полезное свойство языка
- Дает два важных преимущества
- Может проверять предположения которые делает ваш код
  - Добавляет в код директивы проверяющие ваши допущения
  - Утверждения проверяются во время исполнения
  - Если утверждения нарушается программа завершается с ошибкой
- Утверждения также документируют ваши допущения
  - Как и в случае javadoc, код сам определяет собственные допущения

Утверждения в Java

- В версии Java 1.4 добавлено ключевое слово assert

  assert result >= 0;

  – Результатом вычисления условия должно быть логическое значение
  – Скобки вокруг условия не требуются
- Если условие равно false во время исполнения выдается исключение java.lang.AssertionError
  – AssertionError расположено в ветке Error иерархии исключений Java
    - Так как это исключение, оно включает трассировку стека в место где оно произошло
  – Из документации Java API по java.lang.Error:
    - Error это дочерний класс Throwable, который указывает на серьезную проблему, которую приложение не должно перехватывать

Утверждения в Java (2)

- Простой синтаксис assert:
  assert cond;
  - cond должно вычисляться в значение типа boolean
- Можно указать дополнительные детали когда происходит сбой:
  assert cond : expr;
- expr должно возвращать какой либо результат
- т.е. не может быть функцией void
- expr вычисляется только если условие cond равно false
- Детали ошибки должны указывать что пошло неправильно
- Облегчает отладку программного обеспечения!

Отключение утверждений

- Иногда проверка исключений потребляет много ресурсов
  - Пример: класс который сортирует свое содержимое

```
public class Sorter {
    public boolean inOrder() {
            ... // перечисление элементов для проверки
    }
    public void sort() {
            ... // Делаем здесь что то волшебное
            assert inOrder() : «Сортировка не сработала!";
    }
}
```

- В Java утверждения во время исполнения можно включать/ отключать
  - Применение утверждений в классе фиксируется во время его загрузки

Отключение утверждений

- Аргументы виртуальной машины Java для утверждений:
  - -enableassertions (или -ea)
    - Включает утверждения во всех классах кроме системных
  - -disableassertions (или -da)
    - Выключает утверждения во всех классах кроме системных
- Примеры:
- -ea package.ClassName или -da package.ClassName
    - Включает/выключает утверждения в указанном классе
  - -ea package... или -da package...
    - Включает/выключает все утверждения во всех классах пакета
- Для включения/выключения утверждений в системных классах:
  - -enablesystemassertions or -esa

Правила применения утверждений

- Не используйте утверждения Java для проверки аргументов публичных API!
- Стандартное решение в Java это использование исключений для информирования о неправильных аргументах
- Небольшой набор примеров из Java API:
  - NullPointerException
    - Для обязательного аргумента ссылочного типа указано значение null
  - IndexOutOfBoundsException
    - Значение аргумента индекса за пределами допустимого диапазона
  - NumberFormatException
    - Стоковое представление числа имеет неверный формат
  - IllegalArgumentException

Правила применения утверждений (2)

- Не помещайте необходимый для работы код внутрь тестов утверждений!

```
assert set.remove(obj) : "obj не найден: " + obj;
```

  - Проблема?
    - Когда утверждения отключены этот код не выполняется!
- Есть множество правил и рекомендаций по использованию утверждений в Java Many more guidelines for assertions in Java
  - Дополнительную информацию можно найти в разделе “Programming with Assertions”

    http://java.sun.com/javase/6/docs/technotes/guides/language/assert.html

Соглашения о назначении

- Обычно классы используются в Java так:
  - Классы создаются для какой либо программной оболочки
  - Часто имя класса должно удовлетворять некоторым принятым соглашениям
    - Оболочка может обращаться к методам или полям класса
    - Внешние средства проектирования могут разбирать код класса и находить в нем методы и поля
- Пример: каркасы веб-приложений J2EE
  - Enterprise JavaBeans (EJB) инкапсулирует логику веб-приложения
  - EJB должен содержать реализацию определенных интерфейсов, и методов, должны использоваться определенные соглашения при назначении имен
  - Если эти правила нарушаются, J2EE сервер перестает правильно работать.

Аннотации Java

- В версии 1.5 предложено простое решение:
  - К классам их полям и методам добавляются аннотации (т.е. метаданные)
- Аннотации можно извлечь с помощью внешних инструментов
  - Вместо поиска методов с указанными именами или сигнатурой, можно извлечь все методы с заданной аннотацией
- Аннотации используются компилятором Java и ВМ
  - Примеры:
    - “этот метод устарел”
    - “этот метод реализует метод интерфейса”

Аннотации Java (2)

- Аннотации как классы
- Они имеют определенный тип
- Они могут содержать поля для хранения деталей аннотации
- Спецификация аннотации включает:
- Где она может появиться (т.е. только в классе, или только в методе)
- Политика хранения: где и когда они доступны
- Аннотации “исходного кода” – только во время компиляции
- Аннотации “класса” –включаются в откомпилированный class файл, но JVM может убрать их во время загрузки

Простой пример

- Пишем класс двумерной точки

```
public class Point2d {
   private double xCoord, yCoord;
   public boolean equals(Point2d obj) {
            ... // Реализация метода equals
   }
}
```

- Проблемы?
  - Неправильное объявление equals()!
  - Аргумент должен иметь тип Object
- Компилятор не сообщает об этой проблеме!
  - Списки могут неправильно работать с этим кодом.

Теперь с аннотациями

- Java имеет несколько полезных аннотаций
  - @Override – Метод переопределяет метод родительского класса
- Меняем код:

```
public class Point2d {
    private double xCoord, yCoord;
    @Override
    public boolean equals(Point2d obj) {
                ... // Реализация equals
    }
}
```

- Так как мы неправильно определили equals() , эта функция не переопределяет Object.equals()
  - Компилятор сообщает об ошибке, и теперь можно ее исправить.

Другие подробности про аннотации

- Вы можете создавать собственные аннотации!
  - Создавать собственные средства обработки классов
  - Различные инструменты и оболочки имеют собственные аннотации, которые можно использовать в вашем коде
- Документация по аннотациям Java

  http://java.sun.com/javase/6/docs/technotes/guides/language/annotations.html

Classpath

- Если Java программа ссылается на класс его определение надо откуда то взять

```
import javax.vecmath.Vector3f;
...
Vector3f v = new Vector3f(1.0f, 0.0f, 0.0f);
```

  - Когда код компилируется, javac должен найти определение класса javax.vecmath.Vector3f
  - Когда код запускается, JVM тоже должна найти это определение
- classpath сообщает Java где искать определения классов
  - По умолчанию classpath указывает на текущий каталог “.”
  - (Системные классы Java ищутся другим способом…)

Как задать Classpath

- Classpath нужно указывать, когда вы используете внешние библиотеки
  - javax.vecmath.Vector3f находится в библиотеке Java3D
  - Не в стандартной библиотеке Java API, и не в нашем локальном каталоге!
- Есть два способа:
  - Используйте ключ -classpath (или -cp) в командной строке javac и java
  - Задайте переменную среды окружения CLASSPATH
- Значение этой переменной содержит пути
  - Разделители фалов и каталогов зависят от используемой ОС!
  - Windows: -cp C:\path\one;C:\path\two
  - Linux/Mac: -cp /path/one:/path/two

Как задать Classpath (2)

- classpath может содержать:
  - Путь к каталогу, если каталог содержит.class файлы
  - Путь к JAR файлу
    - JAR файлы это файлы архивы классов Java; JAR = Java ARchive
    - Еще о JAR файлах поговорим позднее
    - (Если интересно, см. документацию и утилиту jar)
- Classpath не может просто указывать на

Пример classpath

- Если наш класс Vector3f определен в vecmath.jar
  - Если vecmath.jar в том же каталоге:
    - javac -cp vecmath.jar MyClass.java
  - Если vecmath.jar находится где то еще:
    - javac -cp /path/to/vecmath.jar MyClass.java
  - Запуск кода выглядит примерно так же:
    - java -cp /path/to/vecmath.jar MyClass
- Задание classpath удаляет текущий каталог из пути
  - В некоторых случаях надо делать так:

Проверка списка слов

- На прошлой неделе вы сделали класс для списка слов
  - Написали для него простой тест
  - Осталось много непроверенного функционала!
- Нужно создать набор тестов для проверки класса
  - Каждый тест проверяет одно из свойств класса
  - Если тест не выполняется, нужно определить и устранить проблему
- Модульное тестирование:
  - Проверяет наименьшие доступные для проверки модули программы

Цели модульного

- В идеальном случае, набор тестов должен исполнять весь ваш код
  - Каждый путь кода внутри программы
  - Тесты, которые проверяют нормальное поведение
  - Тесты, которые проверяют обработку ошибок тоже!
    - Называются “негативные тесты”
    - Проверьте что в случае ошибок вызываются нужные исключения
    - Проверьте что программа не завершается неправильным состоянием
    - Проверьте, что программа освобождает все выделенные ей ресурсы
- Средство расчета покрытия кода вычисляет сколько кода исполняется набором тестов
  - Есть разные меры покрытия кода
  - Часто требуется 100% покрытие кода

Цели модульного тестирования (2)

- Модульное тестирование пытается изолировать каждый класс и в идеале каждый метод
  - Облегчает поиск и устранение ошибок
- Классы часто содержат ссылки на другие классы…
  - Часто бывает трудно протестировать один класс отдельно от других
- Модульное тестирование мотивирует отделение интерфейса от реализации
  - Классы взаимодействуют друг с другом через хорошо определенные интерфейсы
  - Набор тестов обеспечивает реализации-заглушки для тестируемого класса

Ограничения модульного тестирования

- Модульное тестирование это простой способ улучшить качество программного обеспечения
  - Нет никаких оправданий для того чтобы не делать модульное тестирование для вашего программного обеспечения
- Проверяет только отдельные модули…
  - Могут остаться проблемы большого масштаба, несовместимости и пр..
- Интеграционное тестирование:
  - Отдельные компоненты и модули комбинируются и проверяются в группе
  - Обычно начинается после завершения модульного тестирования
- Системное тестирование:
  - Все программное обеспечение тестируется и проверяется

Регрессионное

- Еще одна важная методология тестирования о которой следует знать, это регрессионное тестирование
- Сценарий:
  - Вы работаете над программным проектом у которого есть набор тестов
  - Делаете изменения в проекте …
  - Внезапно появляются несколько ошибок в тестах которые раньше выполнялись!
- Это называется регрессией
  - Вы портите свойство, которое работало раньше (чаще всего)
  - Вы добавили код со скрытой ошибкой (реже)
- Очень важно предотвратить регрессию!
  - Особенно важно когда ошибки исправляются на работающем программном обеспечении
  - Клиенту нужен релиз без ошибок, который делает их жизнь лучше, а не хуже.

Регрессионное

- Два основных подхода к определению и предотвращению регрессии!
- Первый подход:
  - Когда добавите новое свойство или исправите ошибку, запустите полный набор тестов программного обеспечения
  - Если ваш набор тестов полный, он быстро обнаружит любую регрессию вызванную сделанными изменениями
- Второй подход:
  - Когда найдена новая ошибка, напишите отдельный тест для ее проверки
- Хорошие компании разработчики программного обеспечения применяют оба подхода в своих

Библиотеки модульного тестирования в Java

- Легче всего управлять тестированием можно с помощью библиотеки тестирования
  - Каждый модульный тест создается как отельный метод
  - Тесты можно группировать в разные категории
    - например “дымовые тесты (так называются тесты на явные ошибки),” “регрессионные тесты,” “длинные тесты”
  - Запуск групп тестов из общей точки входа
  - Просмотр результатов в удобном и понятном виде
- В Java есть две хорошо известные библиотеки
  - JUnit (http://www.junit.org)
    - Более старая и хорошо отлаженная, но с некоторыми важными ограничениями
  - TestNG (http://testng.org)
    - Новая альтернатива созданная для того, чтобы устранить недостатки JUnit

JUnit vs. TestNG

- JUnit в первую очередь сосредоточена на модульном тестировании
  - Прекрасно выполняет простое модульное тестирование
  - Но не так хорошо подходит для интеграционного тестирования или других продвинутых методов тестирования
- TestNG разработана для проведения различных видов тестирования
  - Поддерживается модульное и интеграционное тестирование
  - Можно задать зависимости между тестами
    - Интеграционное тестирование может состоять из нескольких шагов

Тестирование и аннотации

- Старых подход в JUnit 3.x:
  - В тестовом классе делаются тестовые методы
    - Имя метода должно начинаться с “test”
    - Сигнатура метода: нет аргументов, нет возвращаемого значения
    - Метод должен иметь модификатор public, и не может быть статическим.
- Подход в JUnit 4 и TestNG:
  - Тестовые методы имеют аннотацию @Test
  - Других требований для тестовых методов нет

Простой пример TestNG

- Простой тестовый класс для проверки списка слов:

```
import org.testng.annotations.*;
public class TestWordList {
   /** Проверка конструктора по умолчанию WordList. */
   @Test
   public void testDefaultCtor() {
                WordList wl = new WordList();
                assert wl.size() == 0;
                // Проверяем что внутреннее множество
    // инициализировано.
                assert !wl.contains("random");
   }
}
```

- Добавляем другие методы отмеченные аннотацией @Test.

Компиляция тестов

- Компилятор Java должен знать о JAR-файле TestNG
  - Он содержит, в частности, аннотации TestNG
- Пример командная строка *nix:

```
javac -cp .:testng-5.8-java15.jar
  TestWordList.java
```

- …предполагаем6 что все файлы включая TestNG JAR, в текущем

Запуск тестов

- TestNG имеет конфигурационный XML файл
- testng.xml
- Подробности есть на веб сайте TestNG
- Для начала, просто задайте имя тестового класса в командной строке

```
java -cp .:testng-5.8-java15.jar org.testng.TestNG \
   -testclasses TestWordList
```

- Ели классов несколько, отделяйте имена пробелами

Группировка тестов

- Для каждого теста можно задать одну или несколько групп

```
/** Проверка конструктора по умолчанию WordList. */
@Test(groups = {"basic"})
public void testDefaultCtor() {
    WordList wl = new WordList();
    assert wl.size() == 0;
    // Проверяем, что внутренне множество проинициализировано.
    assert !wl.contains("random");
}
```

- groups это массив строк
- Можно задать несколько групп:

```
@Test(groups = {"basic", "fileio"})
```

- Для запуска тестов из одной или нескольких групп:

```
java ... org.testng.TestNG ... -groups basic fileio
```

Негативные тесты

- Тесты должны также проверять обработчики ошибок
  - Методы Java сообщают об ошибках вызывая исключения
- Создадим тест для проверки того, что конструктор WordList вызывает исключение если указать недопустимое имя файл

```
/** Проверка поведения при отсутствии файла. */
@Test(groups={"fileio"},
expectedExceptions={IOException.class})
public void testMissingFile() {
    File f = new File("missing.txt");
    assert !f.exists();
    WordList wl = new WordList(f);
}
```

  - Тест считается невыполненным, если не вызвано исключение или вызвано исключение другого типа

Задание на эту неделю

- Создать класс BoggleBoard для хранения состояния доски
  - Добавить поддержку досок NxN (не только 4x4)
  - Заполнить доску строками содержащими A..Z, или Вопрос: как генерировать случайные буквы?
- В Java для генерации случайных чисел есть класс java.util.Random
  - У него много разных методов!
  - public int nextInt(int n)

Генерация случайных букв

- Можно генерировать случайные числа в диапазоне [0, 26)
  - Как преобразовать их в буквы алфавита?
- Несколько идей:
  - Заполните ArrayList<Character> значениями всех 26-ти символов
    - Используйте случайные числа как индексы для перестановки элементов коллекции
  - Вычислите значение напрямую:

    char ch = (char) (65 + rand.nextInt(26));
    - Что такое 65?!

    char ch = (char) ('A' + rand.nextInt(26));
    - Всегда используйте символы вместо числового кода!

Генерация случайных букв

- Зачем вставлять преобразование в символ за пределами выражения?

```
char ch = (char) ('A' + rand.nextInt(26));
```

  - Что не так с:

```
char ch = 'A' + (char) rand.nextInt(26);
```

- В Java, результат + должен быть:
  - double, float, long или int
  - В нашем случае: char + char = int

Правила преобразования типов арифметических операций в

- Из спецификации языка Java, раздел 5.6.2:
  - Если один из операторов имеет тип double, другой преобразуется в double.
  - Иначе, если один из операторов имеет тип float, другой преобразуется в float.
  - Иначе, если один оператор имеет тип long, другой преобразуется в long.
  - Иначе, ода оператора преобразуются в int.
- Эти правила в частности применяются для арифметических операторов Java
  - Помните о них, когда пишите выражения включающие разные типы данных…

Задание на эту неделю (2)

- Кроме создания класса доски Боггл, нужно также сделать набор тестов для вашего кода
  - Для класса WordList, создайте класс TestWordList
  - Для BoggleBoard, создайте TestBoggleBoard
- Используйте аннотации TestNG для запуска своих тестовn
- Ваш набор тестов должен быть полным!