Программирование на языке Java. Часть 2.

Курс для самостоятельного изучения

Вторая часть?

- Предполагает, что вы уже:
  - Знакомы с классами, модификаторами доступа, наследованием, вложенными классами
  - Знакомы с исключениями и принципами обработки исключений
  - Знакомы со Swing API и событиями AWT
  - Понимаете принципы работы с классами коллекций Java
  - Знакомы с хорошим стилем программирования, и правилами назначения имен в Java
- Основное внимание уделяется:
  - Средствам автоматизации компиляции, модульного тестирования, генераторам документации и т.п.

Большой проект на языке

- Мы разработаем сетевой вариант игры в Боггл
- Боггл это игра в слова
  - Поле клеток 4x4 с буквами
    - “A” .. “Z” и “Qu”
  - Игроки составляют из них слова
    - Начинают с одной из клеток
    - Ход делается в любом

Большой проект на языке

- В конце каждого раунда игроки сравнивают свои списки слов
- Если у нескольких игроков есть одинаковые слова, они удаляются у всех
- Игроки получают очки за слова которые нашли только они.
- Очки за слова назначаются в зависимости от из длины
- 3–4 буквы: 1 очко
- 5 букв: 2 очков
- 6 букв: 3 очков
- 7 букв: 5 очков
- 8+ букв: 11 очков

На этой неделе: разминка

- Создаем класс для работы со списком слов
- Каждое слово должно появляться в списке только один раз
- Нужны эффективные операции добавления/удаления и проверки присутствия слова в списке
- Нужно реализовать несколько “операций над множеством”
  - Добавление списка слов к другому списку слов (объединение множеств)
  - Вычитание одного списка слов из другого (разность множеств)
- Нужно сделать загрузку списка слов из файла
  - Словаря “известных разрешенных слов”

Создание списка слов

- В Java есть средства позволяющие упростить нашу задачу
  - Операции со строками
  - Классы коллекций
  - Операции файлового ввода/вывода
- Используйте эти средства чтобы облегчить себе жизнь! ☺
  - Ваш код в этом задании должен получиться довольно простым.

Коллекции Java

- В версии Java 1.2 появился очень мощный набор классов для управления коллекциями объектов
- Включает:
  - Интерфейсы для разных типов коллекций
  - Реализации интерфейсов с разными характеристиками
  - Итераторы для перемещения по содержимому коллекций
- Очень полезны, но им далеко до мощности и гибкости C++ STL

Интерфейсы коллекций

- Базовая коллекция интерфейсов определена в java.util
  - Определен основной функционал каждого типа коллекций
- Коллекция – базовый “контейнер объектов”
- Список – линейная последовательность элементов, адресуемых по индексу
- Очередь – линейная последовательность элементов “для обработки”
  - Можно добавить элемент к очереди
  - Можно “извлечь следующий элемент” из очереди
  - Какой элемент считать “следующим” зависит от типа очереди
- Множество – коллекция с повторяющимися

Еще интерфейсы коллекций

- Некоторые другие интерфейсы коллекций:
  - SortedSet (расширение Set)
  - SortedMap (расширение Map)
  - Они гарантируют перечисление элементов в определенном порядке
- Элементы должны сравниваться
  - Нужно уметь определять что элемент “меньше чем” или “больше чем” другой элемент
  - Обеспечивает полное упорядочивание

Общие операции над коллекциями

- Коллекции обычно поддерживают следующие операции:
  - add(Object o) – добавляет элемент к коллекции
  - remove(Object o) – удаляет объект
  - clear() – удаляет все объекты коллекции
  - size() – возвращает количество объектов в коллекции
  - isEmpty() – возвращает true, если коллекция пуста
  - iterator() – перемещает по содержимому коллекции

Реализации коллекций

- Каждый интерфейс имеет несколько реализаций
  - Все имеют одинаковый набор базовых функций
  - Разные способы хранения данных
  - Разная производительность
  - Иногда другие расширения
- Детали в документации Java API!
  - В описании интерфейсов API Docs приводится список реализаций
  - Читайте в API Docs подробности о производительности и особенностях хранения

Реализации списков

- LinkedList – двунаправленный связный список
  - Каждый элемент имеет ссылку на предыдущих и следующий элемент
  - Время доступа к i-ому элементу равно O(N)
  - Постоянное время добавления/вставки
  - Элементы используют дополнительную память для хранения служебной информации (ссылки на предыдущий/следующий элементы и др.)
  - Лучше использовать, если список часто меняется
  - Имеют дополнительные функции для извлечения/удаления первого/последнего элементов
- ArrayList – хранит элементы в массиве
  - Постоянное время доступа к i-ому элементуq
  - Время добавления обычно постоянно
  - Время вставки равно O(N)
  - Лучше использовать для редко изменяющихся списков

Реализация множеств

- HashSet
  - Элементы группируются в “корзины” по значению хэш кода
  - Неименное время операций добавления/удаления
  - Неизменное время проверки “на принадлежность”
  - Элементы не хранятся в каком либо определенном порядке
  - Для элементов должны вычисляться хэш функции
- TreeSet
  - Элементы хранятся в отсортированном порядке
    - Внутренне хранятся в сбалансированной древовидной структуре
  - O(log(N))-время операций добавления/удаления

Реализация ассоциативных списков

- Очень похожи на множества
  - Это ассоциативные контейнеры
  - Ключи используются для доступа к значениям хранящимся в ассоциативном списке
  - Каждый ключ уникален (появляется не более одного раза)
    - (Коллекции Java не поддерживают мультимножества и ассоциативные списки с повторяющимися ключами)
- HashMap
  - Ключи хэшируются
  - Быстрый поиск, но случайный порядок расположения элементов
- TreeMap
  - Ключи сортируются
  - Поиск медленнее, но элементы хранятся в отсортированном порядке

Коллекции и объекты

- До версии Java 1.4, коллекции хранили только ссылки на тип Object

```
LinkedList points = new LinkedList();
points.add(new Point(3, 5));
Point p = (Point) points.get(0);
```

- Можно было добавить объект “не точку” в коллекцию точек!
  - Извлечение такой “не точки” из коллекции могло привести к вызову исключения ClassCastException
- И к тому же, каждый раз делать преобразование типов надоедает
  - Код по старому работающий с коллекциями был замусорен преобразованиями типов

Обобщенное программирование в Java 1.5

- В Java 1.5 введено обобщенное программирование
- Указывается тип объектов хранящихся в коллекции:

```
LinkedList<Point> points =
   new LinkedList<Point>();
points.add(new Point(3, 5));
Point p = points.get(0);
```

- Компилятор разрешает добавлять в коллекцию точек только объекты типа Point
  - Если попытаться добавить объект другого типа, получим ошибку во время компиляции

Коллекции и обобщенное программирование

- Для списков и множеств это просто:

```
HashSet<String> wordList = new HashSet<String>();
LinkedList<Point> waypoints = new LinkedList<Point>();
```

  - Тип элемента должен указываться при объявлении переменной и в выражении new
- Ассоциативные списки имеют более длинную запись:

```
TreeMap<String, WordDefinition> dictionary =
    new TreeMap<String, WordDefinition>();
```

  - Сначала указывается тип ключа, затем тип значения

Перечисление элементов коллекций

- Часто требуется перебрать значения элементов коллекции
- Для ArrayList сделать это легко:

```
ArrayList<String> quotes;
...
for (int i = 0; i < quotes.size(); i++)
System.out.println(quotes.get(i));
```

  - Но для других коллекций так сделать невозможно или не нужно!
- Для перемещения по содержимому коллекций используются итераторы
- Iterator это еще один простой интерфейс:
  - hasNext() – возвращает true если можно вызвать next()
  - next() – возвращает следующий элемент коллекции

Использование итераторов

- У коллекций есть метод iterator()
  - Возвращает итератор для перечисления элементов коллекции
- Пример:

```
HashSet<Player> players;
...
Iterator<Player> iter = players.iterator();
while (iter.hasNext()) {
Player p = iter.next();
... // Делаем что то с p
}
```

- Итератор может использовать обобщенное программирование
- Итераторы можно использовать для удаления

Расширенный синтаксис цикла for в Java 1.5

- Настройка и использование итераторов довольно скучное занятие
- В Java 1.5 для этой цели добавлен упрощенный синтаксис:

```
for (Player p : players) {
    ... // Делаем что то с p
}
```

  - Итератор не доступен в цикле
  - Хорошо подходит для простого перебора элементов коллекции
- Этот синтаксис можно использовать для перебора элементов массива:

```
float sum(float[] values) {
    float result = 0.0f;
    for (float val : values)
    result += val;
    return result;
}
```

Элементы коллекций

- Элементы коллекций иногда должны иметь определенный набор свойств
- От элементов List не требуется ничего особенного
- …если не используются методы contains(), remove(), и т.п.!
  - Тогда элементы должны иметь корректную реализацию метода equals()
- Требования к equals():
  - a.equals(a) возвращает true
  - a.equals(b) равно b.equals(a)
  - Если a.equals(b) равно true и b.equals(c) равно true, тогда a.equals(c) также равно true

Элементы множеств, ключи ассоциативных списков

- Элементы множеств и ассоциативных списков должны иметь особенные свойства
  - Множествам нужно производить эти операции над элементами а ассоциативным спискам над ключами
- метод equals() должен работать правильно
- Классам TreeSet, TreeMap нужна сортировка
  - Элемент или ключ должен содержать реализацию интерфейса java.lang.Comparable
  - Или подходящую реализацию java.util.Comparator
- Классам HashSet, HashMap нужен расчет значения хэша
  - Элементы или ключи должны иметь реализацию метода Object.hashCode()

Применение обобщенного программирования

- Вы написали такой код:

```
// Функция для печати содержимого списка
void printList(List<Object> lst) {
   for (Object o : lst)
   System.out.print(" " + o);
}
List<Point> points = new LinkedList<Point>();
... // Заполняем список точками.
printList(points);
```

- Будет ли этот код работать?

Применение обобщенного программирования (2)

- Если этот код работает, printList() может добавить к списку точек любой объект!

```
// Функция для печати содержимого списка
void printList(List<Object> lst) {
for (Object o : lst)
System.out.print(" " + o);
}
List<Point> points = new LinkedList<Point>();
... // Заполняем список точками.
printList(points);
```

- К счастью откомпилировать это в Java не получиться. ☺

Ввод/вывод в Java

- java.io package содержит классы для чтения и записи данных
  - Файловый ввод/вывод – чтение/запись файлов в файловой системе
  - Аппаратный ввод/вывод – сетевые сокеты, последовательные порты, другие внешние устройства
- Второй пакет добавлен в версии Java 1.4
  - java.nio, для расширения операций ввода/вывода
  - Примеры:
    - Отображение части файла в память для увеличения производительности чтения/записи

Базовые операции ввода/вывода в Java

- В java.io package есть два основных типа операций ввода/вывода
- Чтение и запись потоков байтов:
  - InputStream, OutputStream, и (много) дочерних классов
  - Подходят для чтения/записи данных “без структуры”
- Чтение и запись потоков символов:
  - Reader, Writer, и дочерние классы
  - Подходят для чтения/записи текста, особенно локализованного
- Потоки ввода/вывода и классы чтения/записи это абстрактные классы

Операции входного потока

- Входной поток и базовый класс чтения имеют набор основных операций

int read()

- Читает один байт

int read(byte[] b)

- Читает массив байтов

int available()

- Определяет какое количество байтов можно считать без блокировки

long skip(long n)

- Пропускает и удаляет, n

void mark(int rdlimit)

- Запоминает “текущую позицию” в потоке

void reset()

- Устанавливает позицию потока в последнюю отмеченную позицию

void close()

- Закрывает входной поток

Класс чтения почти идентичен,

но считывает символы вместо байтов

Не все потоки имеют эти

Операции выходного

- Выходной поток гораздо проще:

void write(int b)
  - Пишет один байт

void write(byte[] b)
  - Пишет массив байтов

void flush()
  - Сохраняет/передает все байты из буфера потока

void close()
  - Закрывает выходной поток

- Классы Write имеют сходный функционал
  - Эти классы работают с символами а не с байтами
  - И имеют несколько дополнительных методов для

Общий подход к использованию ввода/вывода в

1. Получаем входной или выходной поток источника или получателя данных

```
// filePath это путь и имя заданного файла
FileInputStream fis = new FileInputStream(filePath);
```

2. Если нужно добавить дополнительные функции, заворачиваем поток в другой поток

```
// Добавляем буферизацию потому что чтение по байтам менее
// эффективно
BufferedInputStream bis =
    new BufferedInputStream(fis);
```

3. Используем самый “внешний” поток для операций ввода/вывода.

```
// Читаем данные из входного файла.
byte[] buf = new byte[1024];
```

Некоторые полезные классы потоков

- java.io.FileInputStream и FileOutputStream для чтения и записи файлов данных
- java.net.Socket имеет методы getInputStream() и getOutputStream()
- Пакет java.util.zip включает библиотеки для сжатия
  - Можно открыть входной или выходной поток, напримр, к отдельной записи в.zip файле.
- java.io.ByteArrayInputStream и ByteArrayOutputStream
  - Поддерживают потоковые операции для

Потоки и классы чтения

- Большинство реализаций потоков ввода/вывода не имеют классов чтения/записи
- Два класса нужно конвертировать в классы чтения/записи:
  - java.io.InputStreamReader
    - В параметре конструктора передается объект InputStream
  - java.io.OutputStreamWriter
    - В параметре конструктора передается объект OutputStream
- Очень полезны, для чтения/записи текста в/из потоков ввода/вывода

Файловый ввод/вывод в Java

- Есть несколько способов задать файл или каталог
  - Строкой содержащей путь к файлу/каталогу
  - Объектом java.io.File
    - Есть много полезных свойств!
    - Можно преобразовать относительный путь в абсолютный и наоборот
    - Получить объекты File всех корневых каталогов файловой системы
    - Проверить, что файл существует, доступен ли он на чтение или запись и пр.
- В Java есть классы для того чтобы открыть потоки файлового ввода/вывода и открытия классов чтения из/записи в файлы
  - Они облегчают работу с двоичными и текстовыми файлами
  - Эти объекты понимают строковые пути или объекты File

Документация API

- Документирование кода очень важно
  - Указывайте требования и ожидаемое поведение кода
  - Записывайте проектные решения в коде
  - Любые важные подробности использования, ошибочные условия, и т.п.
- Лучше всего вставлять эту документацию прямо в код
  - Хорошая практика комментирования…
  - Легче обновлять если все находится в одном месте
- Автоматические средства документирования могут обрабатывать ваш исходный код и создавать полезную/аккуратную документацию

Javadoc!

- Sun включает инструмент javadoc в Java Developer Kit
- javadoc обрабатывает файлы исходного кода
  - Комментарии начинающиеся с /** называются комментариями javadoc
  - Должны стоять перед классами, полями, методами, и пр.
  - Комментарии внутри методов игнорируются.
- Пример:

```
/**
* Класс представляющий космический корабль игрока.
*/
public class PlayerShip {
```

Комментарии Javadoc

- Javadoc генерирует “краткие” комментарии и “подробные” комментарии
- Краткий комментарий эт первое предложение комментария javadoc
  - Используется в списках классов, методов, полей и пр.
- Подробные комментарии это комментарии полностью
  - Используются в документации класса, метода, поля, и пр.
- Учтите это при составлении первого предложения!
  - Короткое сообщение, содержащее главные детали.

Тэги javadoc!

- Комментарии javadoc смогут содержать тэги
- Ссылки на другие относящиеся к делу классы
- Привязывают замечания к элементам описания
- Формат тэга @tag, или {@inlinetag}
- Пример:

```
/**
* Класс представляющий космический корабль игрока.
*
* @author Donnie Pinkston
* @version 1.0
```

Применение тэгов javadoc

- Различные теги должны использоваться в разных местах
- С классами и интерфейсами можно использовать:
  - @author – автор класса/интерфейса
  - @version – текущая версия
- С конструкторами и методами можно использовать:
  - @param – описывает параметры
  - @return – описывает возвращаемое значение
  - @throws – какие исключения вызываются и в каких случаях
- Везде можно использовать:
  - @see – ссылка на другой класс, интерфейс, метод, и пр.
  - @since – версия в которой введена эта вещь

Ссылки на другие классы и

- Тэг @see позволяет вставить ссылку на другой класс и пр.
- Ссылка на другой класс:
  @see TargetZone
- Ссылка на поле или метод в другом классе:
  @see TargetZone#loc
  @see TargetZone#intersects(PlayerShip)
- Ссылка на поле или метод в этом классе:
  @see #dirAngle
  @see #turnLeft()
- В комментарий можно вставить тэг {@link ...}

Запуск Javadoc

- javadoc можно запустить из командной строки

  ```
  javadoc -d docs *.java
  ```

- Ключ -d указывает, куда поместить результаты
  - Можно указать относительный или абсолютный путь
  - Каталог создается автоматически
  - По умолчанию используется текущий каталог! Ой!
  - Точка входа документации API – файл

Задание на эту неделю

- Напишите базовый класс для хранения списков слов
  - Сделайте все операции необходимые для игры в Боггл
  - Добавьте возможность загрузки списков слов из фала
  - Напишите простой тестовый класс для проверки своего кода
- Закомментируйте свой код!
  - Используйте комментарии javadoc
  - Запустите javadoc для создания документации

На следующей неделе

- Добавление мета-даных к классам и методам с помощью аннотаций Java
- Создание пакетов автоматизированных тестов для ваши классов