Теория и технология программирования

Программирование на языке Java

Лекция 6. Библиотека коллекций (продолжение)

Глухих Михаил Игоревич, к.т.н., доц.
mailto: glukhikh@mail.ru

Пример использования Set – множество точек на плоскости

```
public class Point implements Cloneable {
    private double x;
    private double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public Point moveTo(double x, double y) {
        this.x = x;
        this.y = y;
        return this;
    }
    public double getX() {   return x;  }
    public double getY() {   return y;  }
    // ...
}
```

Пример использования Set – методы класса точка, множество

```
public class Point implements Cloneable {
    @Override
    public boolean equals(Object obj) {
        if (this==obj) return true;
        if (obj instanceof Point) {
            final Point p = (Point)obj;
            return x==p.x && y==p.y;
        } else return false;
    }
    @Override
    public Point clone() { ... }
    @Override
    public int hashCode() { ... }
}
public class PointSet extends HashSet<Point> {}
```

Пример использования Set – класс "множество точек" и тест

```
public class PointSetTest {
    private static void assertContains(
        final PointSet set, final Point p) {
        assertTrue(set.contains(p));
    }
    @Test
    public void testContains() {
        final PointSet set = new PointSet();
        final Point p = new Point(1.0, 2.0);
        set.add(p);
        assertContains(set, p);
        assertContains(set, p.clone());
        p.moveTo(2.0, 1.0);
        assertContains(set, p);
    }
}
```

Демонстрация

- См. пример

Демонстрация

- Вопрос – почему тест работает настолько странным образом?

Интерфейс Map<K,V>

- Ассоциативный массив, хранящий в себе пары ключ-значение (Key-Value)
- Ключи и значения неупорядочены, но каждое значение жестко привязано к своему ключу
- Совпадение ключей не допускается
- Интерфейс-помощник **Entry** – одна пара ключ-значение

Возможности интерфейса Map<K,V>

```
public interface Map<K,V> {
   int size();
   boolean isEmpty();
   boolean containsKey(Object key);
   boolean containsValue(Object value);
   V get(Object key);
   V put(K key, V value);
   V remove(Object key);
   void putAll(Map<? extends K, ? extends V> m);
   void clear();
   Set<K> keySet();
   Collection<V> valueSet();
   Set<Entry<K,V>> entrySet();
}
```

Возможности интерфейса Entry<K,V>

```
public interface Entry<K,V> {
    K getKey();
    V getValue();
    V setValue();
}
```

Реализации интерфейса Map<K,V>

- Имеется частичная реализация – AbstractMap (скелетальная, на базе **одной** функции entrySet)
- Реализация на основе хэш-таблицы – HashMap
  - используется хэш-поиск для обращения к элементам
  - при удачно написанной хэш-функции время выполнения put, remove, get, containsKey не зависят от размера массива
  - HashSet<E> – на самом деле HashMap<E, Object>

Реализации интерфейса Map<K,V>

- Реализация с перечислением-ключом: EnumMap
- В основе используется массив, индексом которого является номер элемента перечисления

```
Map<Planet, Set<Properties>> planetProperties =
    new EnumMap<Planet, Set<Properties>>(Planet.class);
```

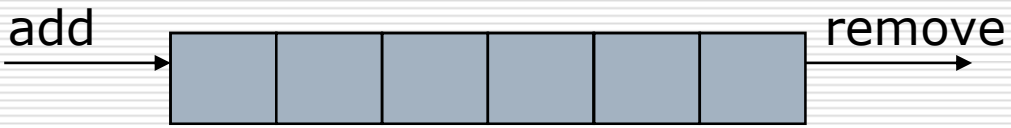
- Зачем аргумент Planet.class??

Реализации интерфейса Map<K,V>

- Реализация на основе бинарного дерева – TreeMap<K, V>
  - реализует интерфейс SortedMap
  - порядок либо на основе Comparable<K>, либо на основе Comparator<V>
  - используется бинарный поиск для обращения к элементам
  - логарифмическое время поиска
  - несколько проще добраться до соседних элементов, чем в HashSet

Интерфейс Queue<E>

- Коллекция, с которой можно работать, как с очередью (FIFO)
- Добавлены методы
  - add(e)/offer(e) – добавление элемента в хвост очереди
  - remove()/poll() – удаление элемента из головы очереди
  - element()/peek() – просмотр элемента из головы очереди
- Контракты прочих методов не меняются

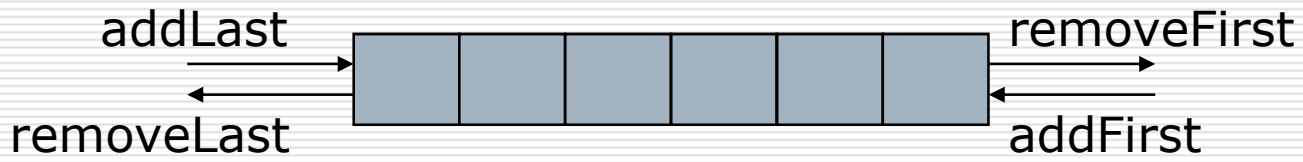


Реализация интерфейса Queue

- Частичная реализация – AbstractQueue
- Одна из полных реализаций – LinkedList (обратите внимание, что данный класс реализует как интерфейс List, так и интерфейс Queue)

Интерфейс Deque<E>

- Коллекция, с которой можно работать, как с двухсторонней очередью (можно добавлять/удалять элементы как из головы, так и из хвоста)
- Расширяет Collection, добавлены методы
  - addFirst(e)/offerFirst(e)/addLast(e)/offerLast(e) – добавление элемента в начало/конец очереди
  - removeFirst()/pollFirst()/removeLast()/pollLast() – удаление элемента из начала/конца очереди
  - getFirst()/peekFirst()/getLast()/peekLast() – просмотр элемента из начала/конца очереди
  - push(e)/pop() – работа с очередью как со стеком
- Контракты прочих методов не меняются



Реализация интерфейса Deque

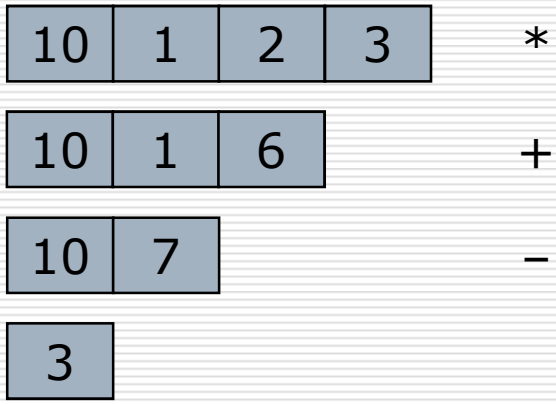
- ArrayDeque – реализация на основе массива
- LinkedList – реализация на основе списка (да, и этот интерфейс LinkedList тоже реализует)

Пример на Map и Deque – вычислитель выражений

- Выражения представлены в обратной польской записи (знак операции следует за операндами) – например
  - 10 1 2 3 * + –
- Необходимо реализовать класс для вычисления подобных выражений и протестировать его

Вычислитель выражений – основные классы

- Собственно вычислитель
  - разбивает строку на отдельные элементы и передает их в арифметический стек для расчета
- Арифметический стек



Реализация класса "Арифметический стек"

- Стек, хранящий вещественные числа и умеющий делать операции вида "взять два числа с вершины, сложить, положить обратно"
- Можно унаследоваться от ArrayDeque или LinkedList

Реализация класса "Арифметический стек"

- Однако, у этих классов слишком много лишних возможностей
- Используем прием "композиция" – сделаем Deque закрытым членом класса и открытые функции push, pop, top

Базовый класс "стек"

```
public class Stack<E> {
    private final Deque<E> stack = new ArrayDeque<E>();

    public void push(E elem) {
        stack.push(elem);
    }

    public E pop() throws NoSuchElementException {
        return stack.pop();
    }

    public E top() throws NoSuchElementException {
        return stack.getFirst();
    }
}
```

Производный класс "арифметический стек"

```
public class ArithmeticStack extends Stack<Double> {
    public void add() throws NoSuchElementException {
        push(pop() + pop());
    }
    public void sub() throws NoSuchElementException {
        double x = pop();
        double y = pop();
        push(y - x);
    }
    public void mul() throws NoSuchElementException { ... }
    public void div() throws NoSuchElementException {
        double x = pop();
        double y = pop();
        push(y / x);
    }
}
```

Функциональный класс "вычислитель польской записи"

```
public class Polish {
    static public double calc(String expr)
            throws IllegalArgumentException {
        final ArithmeticStack stack = new ArithmeticStack();
        final String[] args = expr.split(" ");
        for (String arg: args) {
            try {
                double x = Double.parseDouble(arg);
                stack.push(x);
            } catch (NumberFormatException e) {
                // Что делать дальше? ...
            }
        }
        return stack.top();
    }
}
```

Разбор операций

- Мы могли бы написать код типа:

```
if ("+".equals(arg)) { ... }
else if ("-".equals(arg)) { ... }
else if ...
```

- Хорошо, но довольно громоздко

Разбор операций

- Было бы лучше сделать отдельный тип "операция" и использовать его
- Тогда можно сделать ассоциативный массив String – Operation и с его помощью преобразовывать строку в операцию

Арифметический стек и операции

```
public class ArithmeticStack extends Stack<Double> {
    public static enum Operation {
        ADD, SUB, MUL, DIV;
    }
    public void execute(Operation op)
            throws NoSuchElementException {
        double x = pop();
        double y = pop();
        switch (op) {
            case ADD: push(x + y); break;
            case SUB: push(y - x); break;
            case MUL: push(x * y); break;
            case DIV: push(y / x); break;
        }
    }
}
```

Преобразование строка - операция

```
public class Polish {
    static Map<String, Operation> operationMap =
            new HashMap<String, Operation>();
    static {
        operationMap.put("+", Operation.ADD);
        operationMap.put("-", Operation.SUB);
        operationMap.put("*", Operation.MUL);
        operationMap.put("/", Operation.DIV);
    }
    // ...
}
```

Преобразование строка - операция

```
public class Polish {
    static public double calc(String expr)
            throws IllegalArgumentException {
        // ...
        try }
        } catch (NumberFormatException e) {
            Operation op = operationMap.get(arg);
            if (op==null) throw new
               IllegalArgumentException();
            try {
                stack.execute(op);
            } catch (NoSuchElementException ex) {
                throw new IllegalArgumentException();
            }
        }
    }
}
```

Тестирование

```
public class PolishTest {
    @Test
    public void test2() {
        assertEquals(10.0, Polish.calc("6 -4 -"), 1e-10);
    }
    @Test
    public void test3() {
        assertEquals(3.0, Polish.calc("10 1 2 3 * + -"),
                     1e-10);
    }
    @Test(expected=IllegalArgumentException.class)
    public void test4() {
        Polish.calc("1 -");
    }
}
```

Демонстрация

- См.

Вопрос по проекту

- Функция Polish.calc предполагает, что элементы обратной польской записи отделены друг от друга одним пробелом
- Как ее усовершенствовать – так, чтобы разделителями могли быть пробелы и табуляции в произвольном количестве?

Коллекции из старых версий JDK

- Vector – примерно повторяет ArrayList
- Hashtable – примерно повторяет HashMap
- Данные коллекции поддерживаются, но использование их не рекомендуется

Другие классы общего назначения из JDK

- Math – содержит ряд математических функций, константы e и pi
- Calendar, Date, TimeZone – работа с датами, временем, поясами
- Random – генератор случайных чисел
- System – методы взаимодействия с системой
- Runtime – методы взаимодействия с JVM

Работа с датами, основные методы

- Calendar c = Calendar.getInstance() – получение объекта-календаря
- c.getTime() – получить время (Date)
- c.getTimeInMillis() – получить время в миллисекундах от 01.01.1970
- c.getTimeZone() – получить временной пояс
- c.setTime(), c.setTimeInMillis(), c.setTimeZone() – установка времени/пояса

Работа со случайными числами, основные методы

- Random r = new Random() – получить генератор
- r.setSeed(Calendar.getInstance().
  getTimeInMillis()) –
  установить стартовое число
- r.nextInt(n) – получить случайное число 0...n-1
- r.nextBoolean() – случайное логическое значение
- r.nextDouble() – случайное вещественное число в интервале [0, 1), равномерное распределение
- r.nextGaussian() – случайное вещественное число, нормальное распределение, матожидание 0.0, среднеквадратичное отклонение 1.0

Взаимодействие с JVM, основные методы

- Runtime r = Runtime.getRuntime() – получение экземпляра Runtime
- r.gc() – принудительно запустить Garbage Collector (сборщик мусора)
- r.totalMemory(), r.freeMemory() – сколько памяти всего использует JVM и сколько ее сейчас свободно
- r.halt(-1), r.exit(-1) – прервать выполнение JVM (второй способ более мягкий)
- r.loadLibrary(libraryName) – загрузить указанную динамическую библиотеку
- r.exec(commandString) – выполнить указанную команду операционной системы