КОЛЕКЦИИ

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

24 септември 2008

КОЛЕКЦИИ

Забележка: Тази лекция е адаптация на лекция от курса:

•6.092 Java Preparation for 6.170, Януари 2006
- Lucy Mendel
- Corey McCaffrey
- Rob Toscano
- Justin Mazolla Paluska
- Scott Osler
- Ray He

Интернет адрес:
http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-092January--IAP--2006/CourseHome/index.htm

СЪДЪРЖАНИЕ

КОЛЕКЦИИ

- Служат за съхранение на данни (като масивите) и работа с тях
- Съвкупност от класове и интерфейси служещи за:
  - Добавяне на обекти
  - Съхранение на обекти
  - Сортиране на обекти
  - Обхождане на обекти
  - Извличане на обекти
- Съществува общ начин за достъп до различни реализации на колекции

ПРИМЕР
За използване на колекции

- Класовете и интерфейсите са дефинирани в групата пакети java.util.*
- За улеснение могат да бъдат импортирани (в началото на файла, в който ще се ползват)

```
package  lab2;

import  java.util.*;

public  class CollectionUser {
   List<String> list = new ArrayList<String>();

   // ... останалата част от класа

}
```

Основни методи в колекция от обекти от тип *Foo*

- **boolean** add(Foo o);
- **boolean** contains(Object o);
- **boolean** remove(Foo o);
- **int** size();

```
List<Name> iapjava = new ArrayList<Name>();

iapjava.add(new Name("Laura", "Dern"));
iapjava.add(new Name("Toby", "Keeler"));
System.out.println(iapjava.size());            // => 2

iapjava.remove(new Name("Toby", "Keeler"));
System.out.println(iapjava.size());            // => 1

List<Name> iapruby = new ArrayList<Name>();
iapruby.add(new Name("Scott", "Ostler"));
iapjava.addAll(iapruby);
System.out.println(iapjava.size());            // => 2
```

ТИПИЗИРАНИ КОЛЕКЦИИ

- Предоставя се възможност да се окаже явно типът на обектите, които ще бъдат съхранявани в колекцията
- По този начин се гарантира, че в колекцията може да има единствено обекти от даден тип (**String** в примера по-долу)
- Не е задължително да се указва типът, но е препоръчително и много удобно

```
List<String> strings;
```

ПРИМЕР
Ползване на типизирани колекции

```
// без типизация:
List untyped = new ArrayList();

Object obj = untyped.get(0);
String sillyString = (String) obj;

// с типизация:
List<String> typed = new ArrayList<String>();

String smartString = typed.get(0);
```

ОБХОЖДАНЕ НА КОЛЕКЦИИ

```
// Обхождане на Collection<Foo> coll;

// посредством Iterator:
Iterator<Foo> it = coll.iterator();
while (it.hasNext()) {
    Foo obj = it.next();
    // действия с обекта obj
}

// посредством For each конструкция:
for (Foo obj : coll) {
    // действия с обекта obj
}
```

ИЗВАЖДАНЕ НА ОБЕКТИ ОТ КОЛЕКЦИИ

- ```
  // Елементи не могат да бъдат изваждани от колекция
  // докато тя се обхожда:
for (Foo obj : coll) {
   coll.remove(obj); // throws
ConcurrentModificationException
}
  ```

- ```
  // Елементите могат да бъдат изваждани от колекция,
  // посредством итератора, по време на обхождане:
  Iterator<Foo> it = coll.iterator();
  while (it.hasNext()) {
     Foo obj = it.next();
     it.remove(); // ВМЕСТО coll.remove(obj);
  }
  ```

- ```
  // Методът remove() не е задължителен и не всеки итератор
  // го поддържа (throws UnsupportedOperationException)
  ```
  ```

ОСНОВНИ ТИПОВЕ КОЛЕКЦИИ

- Списък, List
  - ArrayList
- Множество, Set
  - HashSet
  - TreeSet
- Асоциативен контейнер, Map
  - HashMap

СПИСЪК, LIST

- Контейнер за данни (съхранява данни)
- Подредено множество от елементи, подобно на масив
- За разлика от масивите големината им е променлива
- В общия случай редът на елементите съвпада с реда на вмъкването им

```
List<String> strings = new ArrayList<String>();
strings.add("one");
strings.add("two");
strings.add("three");

// strings = [ “one”, “two”, “three”]
```

ДРУГИ ОПЕРАЦИИ

```
List<String> strings = new ArrayList<String>();

// Вмъкване след последния елемент
strings.add("one");
strings.add("three");

// Вмъкване на определена позиция
strings.add(1, "two");

// strings = [ “one”, “two”, “three”]

// Достигане до елемент на определена позиция:
System.out.println(strings.get(0));          // => “one”
System.out.println(strings.indexOf("one")); // => 0
```

МНОЖЕСТВО, SET

- Колекция от данни (като списъка, масивите и т.н.)
- Реализира математическата абстракция множество (set)
- Редът на вмъкване на елементите не влияе на редът им в множеството
- Не позволява вмъкването на два еднакви елемента (**add()** връща **false**):

```
Set<Name> names = new HashSet<Name>();
names.add(new Name("Jack", "Nance"));
names.add(new Name("Jack", "Nance"));

System.out.println(names.size()); // => 1
```

ИЗИСКВАНИЯ КЪМ ЕЛЕМЕНТИТЕ НА МНОЖЕСТВОТО

- След като даден елемент се **вмъкне** в множеството той не би следвало да бъде ползван по никакъв начин, който би го **променило** (неговия хеш код и данните за equal())
- **Промяната** на елемент от множеството би могло да доведе до **неочаквани** резултати:

```
Set<Name> names = new HashSet<Name>();
Name jack = new Name("Jack", "Nance");
names.add(jack);
System.out.println(names.size());          // => 1
System.out.println(names.contains(jack));  // => true;

jack.last = "Vance";

System.out.println(names.contains(jack));  // => false
System.out.println(names.size());          // => 1
```

РЕШЕНИЕ НА ПРОБЛЕМА

- **Няма! Т.е. не трябва да се прави**
- Добре е да се ползват обекти, които не могат да се променят
- Трябва да се внимава
- При промяна на обект първо той трябва да се премахне от множеството (за да се избегнат неочаквани резултати)

АСОЦИАТИВЕН КОНТЕЙНЕР ***MAP***

- Служи за съхранение на връзка между ключ и стойност
- На всеки ключ съответства стойност
- Удобно, когато трябва да се осъществи връзка между два обекта и по единия от тях да може бързо да се намери другия
- Ключовете трябва да са уникални
- Стойностите не е задължително да са уникални
- Примери:
  - Връзка между човек и телефонния му номер
  - Речник – връзка между дума и значение на думата

РАЗЛИКИ С ДРУГИТЕ КОНТЕЙНЕРИ

- ```
  // Не поддържа операциите:
  boolean add(Foo obj);
  boolean contains(Object obj);
  ```

- ```
  // За сметка на това поддържа еквивалентните:
  boolean put(Foo key, Bar value);
  boolean containsKey(Foo key);
  boolean containsValue(Bar value);
  ```

ПРИМЕР
Използване

```
      Map<String, String> dns = new HashMap<String,
String>();

•//Вмъкване на ключ "scotty.mit.edu" със стойност
"18.227.0.87"
      dns.put("scotty.mit.edu", "18.227.0.87");

      System.out.println(dns.get("scotty.mit.edu"));
      // => "18.227.0.87"
      System.out.println(dns.containsKey("scotty.mit.edu"));
      // => true
      System.out.println(dns.containsValue("18.227.0.87"));
      // => true

•// Изключваме елемент по ключ "scotty.mit.edu"
      dns.remove("scotty.mit.edu");
      System.out.println(dns.containsValue("18.227.0.87"));
      // => false
```

ДРУГИ МЕТОДИ

- **KeySet()**
  - Връща множество (**set**, т.е. не може да има повторения) от всички ключове
- **Values()**
  - Връща колекция (може да има повторения) от всички стойности
- **EntrySet()**
  - Връща множество (**set**, т.е. не може да има повторения) от двойките ключ-стойност на асоциативния контейнер
  - Всяка двойка е обект от тип **Map.Entry**
    - Той поддържа методи за достъп до ключа и стойността **getKey(), getValue(), setValue()**

ПРОБЛЕМИ ПРИ ПРОМЯНА НА КЛЮЧ

- Обектът, който е ключ в асоциативен контейнер, не трябва да се променя
- Проблемът е аналогичен на този, разглеждан за множествата
- Ако ключът бъде променен, съответстващата му стойност би станала недостъпна

ПРИМЕР
Промяна на ключ

```
   Name isabella = new Name("Isabella", "Rosellini");
   Map<Name, String> directory = new HashMap<Name, String>();
   directory.put(isabella, "123-456-7890"); // добавяме
   System.out.println(directory.get(isabella));

   isabella.first = "Dennis"; // променяме ключа
   System.out.println(directory.get(isabella));

   // добавяме още един обект със стойност като оригиналния
   directory.put(new Name("Isabella", "Rosellini"),
"555-555-1234");
   // връщаме стойността на оригиналния
   isabella.first = "Isabella";

   System.out.println(directory.get(isabella));
```

ПРИМЕР
Промяна на ключ

```
   Name isabella = new Name("Isabella", "Rosellini");
   Map<Name, String> directory = new HashMap<Name, String>();
   directory.put(isabella, "123-456-7890"); // добавяме
   System.out.println(directory.get(isabella));

   isabella.first = "Dennis"; // променяме ключа
   System.out.println(directory.get(isabella));

   // добавяме още един обект със стойност като оригиналния
   directory.put(new Name("Isabella", "Rosellini"),
"555-555-1234");
   // връщаме стойността на оригиналния
   isabella.first = "Isabella";

   System.out.println(directory.get(isabella));

   // => 123-456-7890, null, 555-555-1234
```

ПРИМЕР
Промяна на ключ

```
   Name isabella = new Name("Isabella", "Rosellini");
   Map<Name, String> directory = new HashMap<Name, String>();
   directory.put(isabella, "123-456-7890"); // добавяме
   System.out.println(directory.get(isabella));

   isabella.first = "Dennis"; // променяме ключа
   System.out.println(directory.get(isabella));

   // добавяме още един обект със стойност като оригиналния
   directory.put(new Name("Isabella", "Rosellini"),
"555-555-1234");
   // връщаме стойността на оригиналния
   isabella.first = "Isabella";

   System.out.println(directory.get(isabella));
   // Не ни интересува какво ще се изведе при изпълнението
   // на тези редове, защото никога не трябва да правим така!
   // Поведението е недефинирано!
```

РЕШЕНИЕ НА ПРОБЛЕМА ЧРЕЗ КОПИРАНЕ НА КЛЮЧА

```
Name dennis = new Name("Dennis", "Hopper");

// копиране на ключа:
Name copy = new Name(dennis.first, dennis.last);
map.put(copy, "555-555-1234"); // използва се копието

// промяната на dennis не би попречила

// но въпреки това остава опасност от промяна на ключа:
for (Name name : map.keySet()) {
   name.first = "u r wrecked"; // ГРЕШКА!
}
```

РЕШЕНИЕ НА ПРОБЛЕМА КЛЮЧОВЕ, КОИТО НЕ СЕ ИЗМЕНЯТ

```
public  class Name {
   public final String first; // т.е. не може да се променя
   public final String last;

   public Name(String first, String last) {
      this.first = first;
      this.last = last;
   }

   public boolean equals(Object o) {
      return (o instanceof Name &&
            ((Name) o).first.equals(this.first) &&
            ((Name) o).last.equals(this.last));
   }
}
```

РЕШЕНИЕ НА ПРОБЛЕМА КОНСТАНТЕН ДЕЛЕГАТ

```
        Map<String, String> dir = new HashMap<String,
String>();
        Name naomi = new Name("Naomi", "Watts");

        String key = naomi.first + "," + naomi.last;
        dir.put(key, "888-444-1212");

        // String-ът не може да бъде променян.
        // Това защитава асоциативния контейнер
```

РЕШЕНИЕ НА ПРОБЛЕМА „ЗАМРАЗЯВАНЕ“ НА КЛЮЧА

```
public  class Name {
    private String first;
    private String last;
    private boolean frozen = false;

    // ...
    public void setFirst(String s) {
        if (!frozen) // проверка дали е „замразен“
            first = s;
    }
    // аналогично за setLast()
    // ...

    // „замръзяваме“ обекта
    public void freeze() {
        frozen = true;
    }
}
```

ЧЕСТО СРЕЩАНИ ГРЕШКИ

- Изтриване на елемент от списък докато го обхождаме
- Промяна на елемент в множество
- Промяна на ключ в асоциативен контейнер