Тема 9. Колекции (Контейнери на Java) (продължение)

Съдържание

- Класове в колекциите след 5.0;
  - Общи имплементации;
  - Основни особености – ефективност, правила за избор.
- Параметризирани класове (Generics) – създаване, използване в колекциите;

Имплементации на колекциите

Същност:

Имплементации на колекциите са даннови обекти, които се използват за съхраняване на колекциите, които имплементират интерфейсите, описани в предната лекция

Имплементации на колекциите

Видове:

- Общи имплементации-най-често използваните;
- Специални – проектирани са за използване в специални ситуации и за по-специални изисквания по бързодействие/памет или поведение;
- Имплементации за използване в конкурентен режим, най-често за многонишков режим;
- Имплементации в заграждащи класове;
- Мини имплементации за специални цели;

Имплементации на колекциите

Абстрактни видове - Абстрактни имплементации-скелети, позволяващи създаването на потребителски дефинирани имплементации:

http://docs.oracle.com/javase/8/docs/technotes/guides/collections/index.html
http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html
https://docs.oracle.com/javase/9/docs/api/java/util/doc-files/coll-reference.html
https://docs.oracle.com/javase/10/docs/api/java/util/doc-files/coll-reference.html

Общи имплементации на Set

Видове:

В 5.0. Set се имплементира с:

- HashSet;
- TreeSet;
- LinkedHashSet

Типове колекции. SortedSet

Дефиниция :

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Общи имплементации на Set

Видове:

В 5.0. Set се имплементира с:

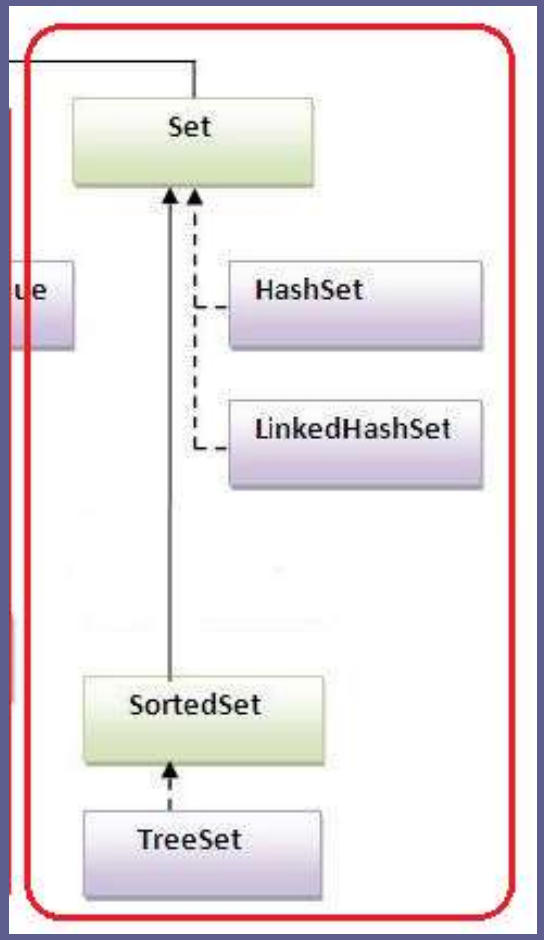
- HashSet;
- TreeSet;
- LinkedHashSet

Характеристики:

- HashSet е по-бърз от TreeSet;

константно и логаритмично време за повечето операции

- Не предоставя подредба.



Общи имплементации на Set

Област на приложение:

TreeSet:

- Ако трябва да се изпълняват операции на интерфейса SortedSet;
- Когато се изисква подредба по стойностите. Основа на подредбата:
  - comparable: e1.compareTo(e2);
  - comparator.compare(e1, e2).

HashSet: при липса на горните изисквания.

Общи имплементации на Set

LinkedHashSet (LinkedHashMap)
Характеристики:

- LinkedHashSet и LinkedHashMap комбинират хеш алгоритъм със свързан списък, който поддържа реда при вмъкване на елементите.

Колекциите (LinkedHashSet и LinkedHashMap) са подредени, но не сортирани колекции.

.

Общи имплементации на Set
Оценка на ефективността

- LinkedHashSet е междинен вариант HashSet и TreeSet:
  - Организация на хеширана таблица и свързан списък:
  - Скоростта на достъпа е почти като на HashSet;
  - Осигурява по-подреден спрямо HashSet вариант на организация, без да се губи ефективността при работата (TreeSet).

Общи Общи имплементации на Set Оценка на ефективността

- При HashSet достъпът е линеен по сума от броя на елементите и капацитета. Следствия:
  - Високият начален капацитет влошава двата параметъра;
  - Малкият – довежда до принудително копиране при превишаване на капацитета.
  - Подразбираща стойност на капацитета - 16. Винаги се закръгля по степените на две.

Общи имплементации на Set
Оценка на ефективността

HashSet - създаване.

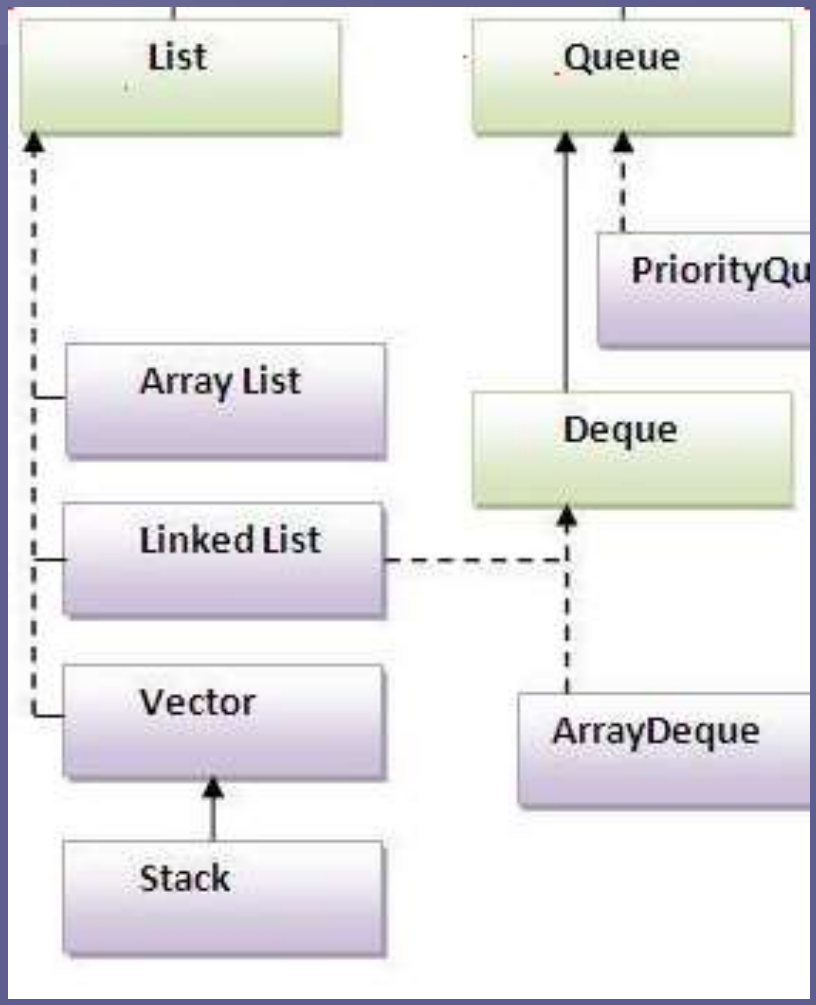
- Има експлицитен конструктор с параметър капацитета (Винаги се закръгля по степените на две):

Пример за начален капацитет 64:

```
Set<String> s = new HashSet<String>(64);
```

Общи имплементации на List

Имплементации на List:



Типове колекции върху List

Дефиниция:

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);     //optional
    boolean add(E element);          //optional
    void add(int index, E element); //optional
    E remove(int index);             //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```

Общи имплементации на List

Две общи имплементации на List-ArrayList и LinkedList

1. **ArrayList -** Най-често се използва:
   - постоянно време за достъп по известен индекс;
   - не позволява да се алокира единичен елемент за всеки нов обект на списъка;
   - използва System.arraycopy за пренасяне (в частност извеждане) на множество елементи;

Общи имплементации на List

2. LinkedList :

- Ако често се налага да се добавят елементи в началото на List;
- Достъп чрез итератор;
- Изтриване на елементи чрез итератор
  - Тези операции изискват постоянно време при LinkedList и линейно при ArrayList;

Общи имплементации на List

- Влошено бързодействие на достъпа при LinkedList;
- Високото бързодействие при добавяне в началото и средата на LinkedList.
- По-ниско бързодействие при достъпа по (индекс):
  - Линейно при LinkedList – O(n);
  - Константно при ArrayList – O(1).

Общи имплементации на List

Практически съвети:

- Не бива да се предоверява на теоретичните параметри при сравняване на LinkedList и ArrayList;
- Най-добре е да се измери производителността при двата варианта и да се подбере по-добрия;
- В повечето случаи това е ArrayList;

ArrayList има начално инициализиране на капацитета *initial capacity*, който определя броя елементи, преди да се разширява. LinkedList няма такъв параметър.

Общи имплементации на List

LinkedList има 7 допълнителни операции:

- addFirst;
- getFirst;
- removeFirst;
- addLast;
- getLast;
- removeLast.

LinkedList имплементира интерфейса Queue.

Общи имплементации върху Map

Дефиниция:

```
public interface Map<K,V> {

    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();

    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Общи имплементации на Map

Има 3 вида общи имплементации:

- HashMap;
- TreeMap;
- LinkedHashMap.

Общи имплементации на Map

Дефиниция на SortedMap:

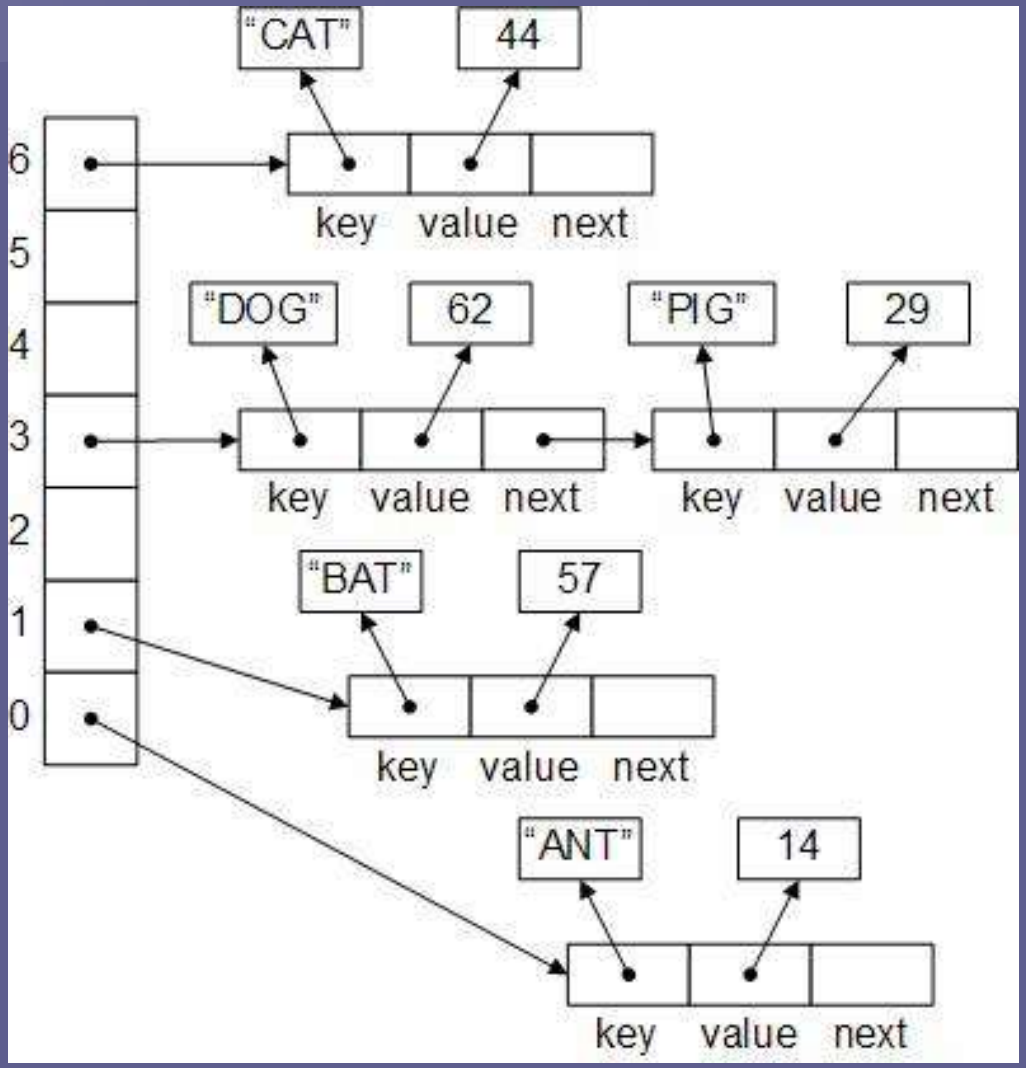
```
public interface SortedMap<K, V> extends Map<K, V>{

    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();

    Comparator<? super K> comparator();
}
```

Общи имплементации на Map

- HashMap-

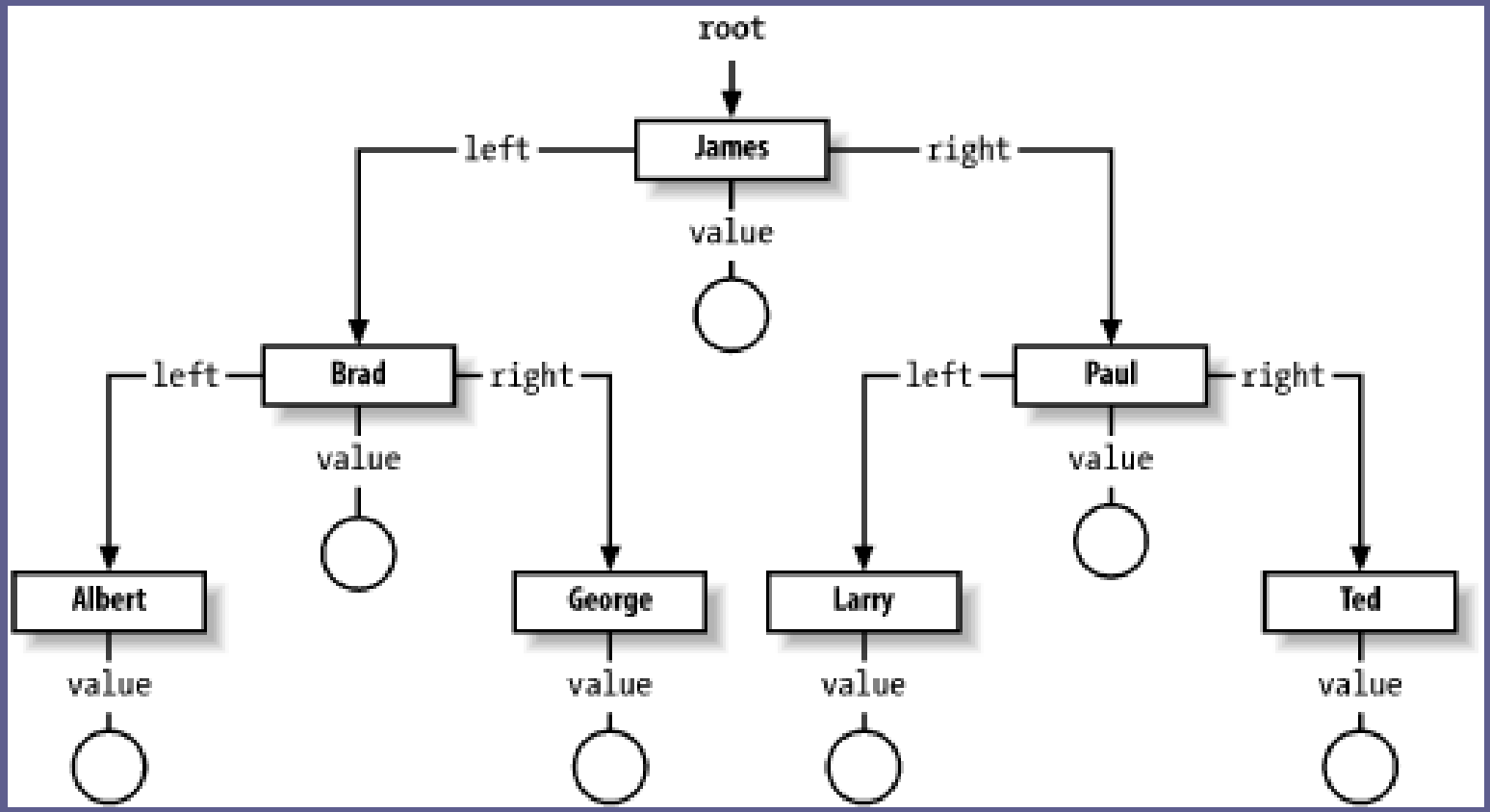
Пример за организация на двойки от типове:

<String,Integer>;



Общи имплементации на Map

- TreeMap, пример с ключ тип String;



Общи имплементации на Map

Интерфейс SortedMap – имплементация:

Клас TreeMap. Кога се използва:

- Ако са необходими операциите на интерфейса SortedMap:
  - subMap( );
  - headMap( );
  - tailMap( )
- Колекции, подредени по стойността на определен ключ;
- Достъп чрез итерация на сортирани двойки по ключ;

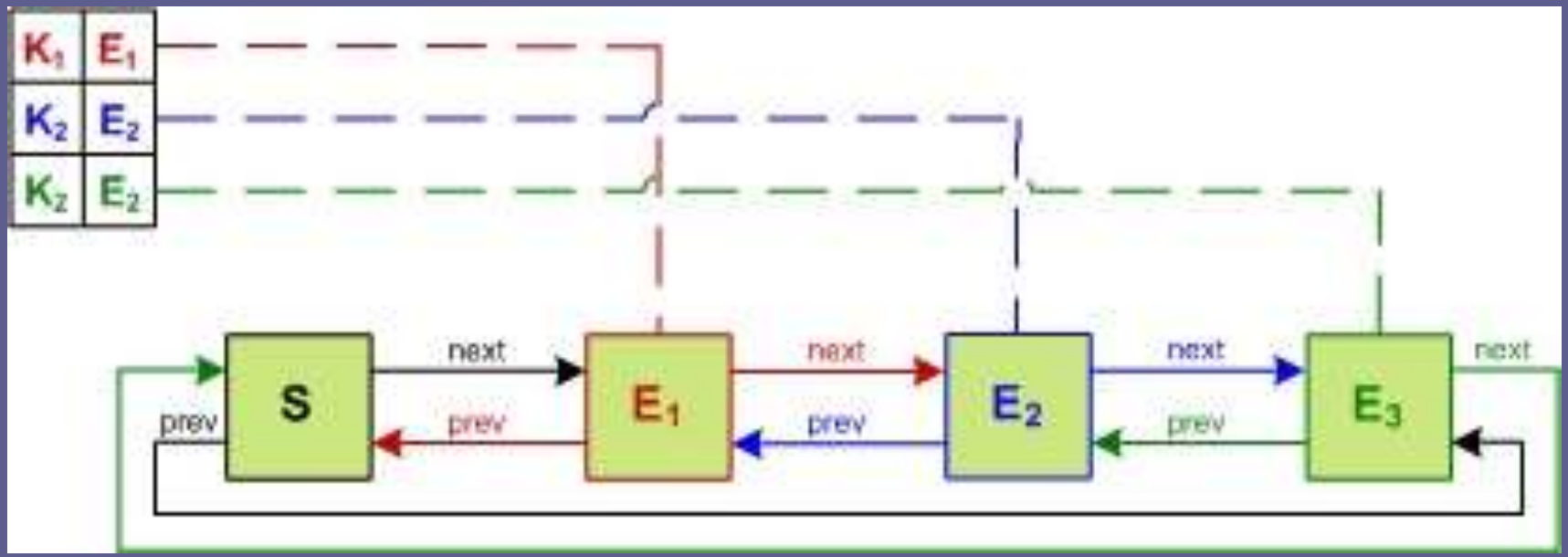
Общи имплементации на Map

Използване на имплементациите HashMap (несинхронизирана имплементация):

- Няма изисквания за паралелни действия - синхронизиране на достъпа;
  - HashTable е синхронизиран вариант
- При изисквания за максимална скорост;
- Нестроги изисквания за реда на итерация на елементите;

Общи имплементации на Map

Организация на **LinkedHashMap**



Общи имплементации на Map

Използване на имплементациите LinkedHashMap:

При междинни случаи:

- За осигуряване на близко до бързодействието на HashMap;
- Чести операции вмъкване/достъп:
  - LinkedHashMap предоставя допълнителни спрямо LinkedHashSet възможности.
- При създаване на LinkedHashMap, може да се подреди на базата на ключов достъп вместо на вмъкване.

Общи имплементации на Queue

LinkedList имплементира интерфейса основан на организация на опашката FIFO при операциите add, poll и др.

PriorityQueue е клас, който имплементира приоритетна опашка, основана на данновата структура heap. Видове подредби:

- подрежда елементите в съответствие с реда на създаване – подразбираща се подредба;
- Подредба по експлицитно зададен сравнител от класа Comparator;

Общи имплементации на Queue PriorityQueue

Операциите за достъп-poll, remove, peek, и element — достъпват елемента **глава (head)** на Queue. Еламентът *head* е последният елемент в съответствие със специфицирания ред.

- Ако има дублиране на елементи, за глава се избира един от тях-редът е случайно избран.

Общи имплементации на Queue PriorityQueue

- offer(…) – добавя елемент накрая на опашката - O(log(n))
- poll() – премахва+ връща елемента от началото на опашката - - O(log(n))
- peek() – връща елементът от началото на опашката без да го премахва - const
- clear() – премахва всички елементи от опашката - const
- contains(…) – проверява дали елемента се съдържа в опашката - - O(n)

Общи имплементации на Queue PriorityQueue

Особености:

- PriorityQueue и нейният итератор имплементират всички методи на интерфейсите Collection и Iterator;
- Итераторът не гарантира преминаването през елементите на PriorityQueue в техния естествен ред;
- Ако се налага подредба на елементите, може да се използва сортиращ алгоритъм:
  - например: Arrays.sort(pq.toArray())

Общи имплементации на Queue PriorityQueue

Пример за клас сравнител:

```
class StringLengthComparator implements
   Comparator<String>{
 public int compare(String x, String y) {
     if (x.length() < y.length()) { return 1;}
     if (x.length() > y.length()) { return -1;}
     return 0;
 }
}
```

Общи имплементации на Queue PriorityQueue

Пример:

```
public class Test{
    public static void main(String[] args) {
        Comparator<String> comparator = new StringLengthComparator();
//        PriorityQueue<String> queue = new PriorityQueue<String>(10);
        PriorityQueue<String> queue =
            new PriorityQueue<String>(10, comparator);
        queue.add("short");
        queue.add("superlong");
        queue.add("medium");
        queue.remove();
        while (queue.size() != 0)  {
            System.out.println(queue.remove());
        }
    }
}
```

Параметризирани класове (Generics)

Дефиниция:

- Терминът generics означава шаблонен или параметризиран тип. Въведен е от Java 2 V5.0
- T е типизиран параметър, който се замества с реален тип;
- T е името на типа на параметъра;
- Това наименование на параметъра се използва като променлива за получаване на стойност от фактическия тип, която ще се използва в класа като тип, когато се създава обект от параметризирания клас.

Параметризирани класове (Generics)

- Пример за създаване на собствен шаблонен клас:

```
class GenericClass<T> {
  T ob; // като поле
  GenericClass(T o) { // като формален параметър
    ob = o;
  }
  T getob() { // като връщана стойност
    return ob;
  }
  void showType() { // тестово извеждане на типа
    System.out.println("Type of T is " +
    ob.getClass().getName());
  }
}
```

Параметризирани класове (Generics)

```
public class MainClass {
  public static void main(String args[]) {
    // Създаване на референция на GenericClass за Integer.
    GenericClass<Integer> iOb = new GenericClass<Integer>(88);
    iOb.showType();
    // не се изисква преобразуване към прост тип.
    int ival = iOb.getob();
    System.out.println("value: " + ival);
    //  Създаване на референция на GenericClass за  Strings.
    GenericClass<String> strOb = new GenericClass<String>("Gen Test");
    strOb.showType();
    String str = strOb.getob();
    System.out.println("value: " + str);
  }
}
```

Параметризирани класове (Generics)

Резултат:

Type of T is java.lang.Integer value: 88
Type of T is java.lang.String value: Gen Test

Параметризирани класове. Особености

За разлика от езика C++ в Java не могат да се използват като параметри на шаблона всички типове данни.

- **Използват се само с обекти – не допускат прости типове данни**!

Пример:

```
Gen<int> strOb = new Gen<int>(53);
// Предизвиква грешка, защото се
// използва прост тип
```

Параметризирани класове. Особености

- **Използване на повече от един параметър на типа. Пример:**

```
class TwoGen<T, V> {
      T ob1;
      V ob2;
      TwoGen(T o1, V o2) {
        ob1 = o1;
        ob2 = o2;
      }
     // метод за извеждане на имената на типовете
      void showTypes() {
        System.out.println("Type of T is " + ob1.getClass().getName());
        System.out.println("Type of V is " + ob2.getClass().getName());
      }

      T getob1() {
        return ob1;
      }

      V getob2() {
        return ob2;
      }
}
```

Параметризирани класове. Особености

- Използване на повече от един параметър на типа. Пример за тестов клас:

```
public class MainClass {
  public static void main(String args[]) {
    TwoGen<Integer, String> tgObj = new TwoGen<Integer,
    String>(88, "Generics");
    tgObj.showTypes();

    int v = tgObj.getob1();
    System.out.println("value: " + v);

    String str = tgObj.getob2();
    System.out.println("value: " + str);
  }
}
```

Параметризирани класове. Особености

- Резултат :

Type of T is java.lang.Integer

Type of V is java.lang.String

value: 88

value: Generics

Контейнерни параметризирани типове. Особености

Съществуващите в пакета java.util.* параметризирани типове са колекциите на езика.

- Могат да приемат параметри;
- Параметризирани типове са синоними на общите (generic) типове на езика Java;
- В скобите могат да се поставят само референтни типове;

Контейнерни параметризирани типове. Особености

Формат:

List<E> myList;

Където E е променливата на типа, т.е. променливата, която се замества с конкретния тип. Особености:

- Типът, който използва типовата променлива E позволява да се постави E когато се декларира или се създава инстанция на параметризирания тип;
- Ако E е клас, може да се предава наследник на класа E;
- Ако E е интерфейс, може да се постави вместо него клас, който имплементира интерфейса E;
- За съвместимост при дефиниране на шаблонните класове за име на типовата променлива се използва единична главна буква;

Контейнерни параметризирани типове. Особености

Пример

```
import java.util.ArrayList;
import java.util.List;
public class MainClass {
    public static void main(String[] args) {
   // основен вариант
   List stringList1 = new ArrayList();

      stringList1.add("Java 5");
      stringList1.add("without generics");
// изисква типово преобразуване
      String s1 = (String) stringList1.get(0);
      System.out.println(s1.toUpperCase());//JAVA 5
```

Контейнерни параметризирани типове. Особености

Пример

```
    // параметризиран вариант
       List<String> stringList2 = new ArrayList<String>();
       stringList2.add("Java 5");
       stringList2.add("with generics");
     // не изисква типово преобразуване
       String s2 = stringList2.get(0);
       System.out.println(s2.toUpperCase()); //JAVA 5
      }
}
```

Вложени контейнерни параметризирани типове

- Параметризираните типове са типове на езика Java и могат да се влагат като типови променливи. Например, може да се декларира списък за съхраняване на списък от стрингове:

```
List<List<String>> myListOfListsOfStrings;
```

За да се получи достъп до първия стринг на първия списък от обекта myListOfListsOfStrings се използва:

```
String s = myListOfListsOfStrings.get(0).get(0);
```

Вложени контейнерни параметризирани типове

Пример за използване:

```
import java.util.ArrayList;
import java.util.List;
public class MainClass {
 public static void main(String[] args) {
   List<String> listOfStrings = new ArrayList<String>();
   listOfStrings.add("Hello again");
   List<List<String>> listOfLists = new ArrayList<List<String>
    >();
   listOfLists.add(listOfStrings);
   String s = listOfLists.get(0).get(0);
   System.out.println(s); // изход "Hello again"
 }
}
```

Контейнерни параметризирани типове с повече параметри

Пример е интерфейса Map и класа HashMap:

```
import java.util.HashMap;
import java.util.Map;

public class MainClass {
    public static void main (String[] args) {
        Map<String, String> map = new HashMap<String,
    String>();
        map.put ("key1", "value1");
        map.put ("key2", "value2");
        String value1 = map.get("key1");
    }
}
```

Непроверени типове. Сигурност на кода

За да се осигури преминаването към параметризирани типове и съвместимостта с предходните версии Java позволява да се пропуснат параметрите на типа. Това създава непроверена референция към типа:

Пример:

```
class Gen<T> {
  T ob;

  Gen(T o) {
    ob = o;
  }

  T getob() {
    return ob;
  }
```

(продължение)

Непроверени типове. Сигурност на кода

```
public static void main(String args[]) {
 Gen<Integer> iObject = new Gen<Integer>(88);
 Gen<String> strObject = new Gen<String>("Generics Test");
 String str = strObject.getob();  // шаблонен
 Gen raw = new Gen(new Double(98.6)); // непроверено по тип

 // Необходимо е преобразуване към (Double)поради неизвестния тип
 double d = (Double) raw.getob();
 System.out.println("value: " + d);

 strObject = raw; // потенциална грешка (предупреждение)

// припокриване на типа
raw = iObject; // потенциална грешка
d = (Double) raw.getob();
 }
}
```

Непроверени типове. Сигурност на кода

Резултат от примерната програма:

```
value: 98.6
Exception in thread "main" java.lang.ClassCastException:
    java.lang.Double cannot be cast to java.lang.String
    at Gen.main(Gen.java:23)
```

Примерно използване на имплементациите и техните интерфейси

Примери за създаване на обекти от класовете:

```
package tu_varna;
import java.io.*;
import java.io.RandomAccessFile;
import java.util.*;
public class CollectionSamples {

//   public Collection oColl = new ArrayList(); // Създаване на колекция от ArrayList
//   public Collection oColl = new Vector(); // Създаване на колекция от Vector
//   public Collection oColl = new Stack();  // Създаване на колекция от Stack
//  public Collection oColl = new LinkedList(); //Създаване на колекция от LinkedList

//   public List oColl = new ArrayList();  // Създаване на списък от класа      ArrayList
//   public List oColl = new Vector(); // Създаване на списък от класа        Vector
//   public List oColl = new Stack(); // Създаване на списък от класа         Stack
//   public List oColl = new LinkedList(); //Създаване на списък от класа LinkedList

//   public Set oColl = new HashSet(); //Създаване на множество от клас HashSet
```

Примерно използване на имплементациите и техните интерфейси

```
public Set oColl = new TreeSet();  // Създаване на множество от клас  TreeSet
public CollectionSamples(String fileName ) {
 try { // използване на Scanner с файлов поток
   Scanner sc = new Scanner(new File(fileName));
   while( sc.hasNextLine() )  {
     Object oPersonExpl= new Student(sc.next(), sc.next(), sc.next(), sc.next());
     oColl.add(oPersonExpl);
  }
  sc.close();
} catch (FileNotFoundException e){
  System.out.println("File Not Found ...");
 }
}
```

Примерно използване на имплементациите и техните интерфейси

```
// извеждане  на конзолен изход с итератор
public void printColl(Collection c) {
    for(Iterator It = c.iterator(); It.hasNext();)
       System.out.println(It.next().toString());
  }

// стрингова интерпретация
 public String toString() {
     return oColl.toString();
 }
// главна функция
 public static void main(String[] args) {
    CollectionSamples obj = new CollectionSamples("Input.txt");
    obj.printColl(obj.oColl);
   }
}
```

Въпроси?