Лекция 13b

Колекция от структури данни (Част II)

Основни теми

- Дефиниране на съвкупност от структури данни.
- Използване на ***class Arrays*** за работа с масиви.
- Използване на базисна система (***framework***) от библиотечни реализации на структури от данни.
- Използване на алгоритмите (***search***, ***sort*** и ***fill***) , реализирани в базисната система (***framework***) от библиотечни реализации на структури от данни.

Основни теми

- Използване на ***interface***- ите в базисната система (***framework***) от библиотечни реализации на структури от данни за тяхната полиморфична обработка.
- Използване на итератори за обхождане на съвъкупност от данни.
- Използване на съхранени hash таблиции работа с обекти от ***class Properties***.

13b.7 *class* **Stack** в *package java.util*

13b.8 *class PriorityQueue* и interface Queue

13b.9 Видове приложения на интерфейс *Set*

13b.10 Видове приложения на интерфейс *Map*

13b.11 Properties

Задачи

Литература:

Java How to Program, 10 Edition, глава 16

13b.6 Collections class и алгоритми

List

Collection

java.util.Collections	
+sort(list: List): void	Sorts the specified list.
+sort(list: List, c: Comparator): void	Sorts the specified list with the comparator.
+binarySearch(list: List, key: Object): int	Searches the key in the sorted list using binary search.
+binarySearch(list: List, key: Object, c: Comparator): int	Searches the key in the sorted list using binary search with the comparator.
+reverse(list: List): void	Reverses the specified list.
+reverseOrder(): Comparator	Returns a comparator with the reverse ordering.
+shuffle(list: List): void	Shuffles the specified list randomly.
+shuffle(list: List): void	Shuffles the specified list with a random object.
+copy(des: List, src: List): void	Copies from the source list to the destination list.
+nCopies(n: int, o: Object): List	Returns a list consisting of *n* copies of the object.
+fill(list: List, o: Object): void	Fills the list with the object.
+max(c: Collection): Object	Returns the max object in the collection.
+max(c: Collection, c: Comparator): Object	Returns the max object using the comparator.
+min(c: Collection): Object	Returns the min object in the collection.
+min(c: Collection, c: Comparator): Object	Returns the min object using the comparator.
+disjoint(c1: Collection, c2: Collection): boolean	Returns true if c1 and c2 have no elements in common.
+frequency(c: Collection, o: Object): int	Returns the number of occurrences of the specified element in the collection.

13b.6 Collections алгоритми

`Collections` библиотеката включва високо производителни (ефективни) `static` алгоритми за работа с елементи на колекция

- ✓ **`List` алгоритми**
  - `sort`
  - `binarySearch`
  - `reverse`
  - `shuffle`
  - `fill`
  - `copy`

13b.6 Collections алгоритми

- ✓ `Collection` **алгоритми**
  - `min`
  - `max`
  - `addAll`
  - `frequency`
  - `disjoint`

Алгоритъм	Описание
`sort`	Сортира елементите на даден `List`.
`binarySearch`	Намира обект в даден `List`.
`reverse`	Нарежда в обратен ред елементите на `List`.
`shuffle`	Нарежда по случаен начин елементите на `List`.
`fill`	Инициализира всеки елемент на `List` да реферира един и същ зададен обект данни.
`Copy`	Копира референциите към обекти данни от даден `List` в друг `List`.
`min`	Връща най- малкия елемент в `Collection`.
`max`	Връща най- големия елемент в `Collection`.
`addAll`	Добавя всички елементи от зададен масив в колекция.
`frequency`	Пресмята колко елемента от зададена колекция са равни на зададен обект.
`disjoint`	Определя дали две зададени колекции имат общи елементи данни.

Fig. 13b.7 | Collections алторитми.

Software Engineering факти 13b.4

Алгоритмите, реализирани в базисната библиотека от колекции, е полиморфична. Това означава, че всеки алгоритъм може да оперира върху обекти, които имплементират определени интерфейси, без да има значение конкретната реализация на тези интерфейси.

13b.6.1 sort Алгоритъм

Sort

- **Служи за сортиране на `List` елементи**
  - **Редът за подреждане се определя от типа на елементите на `List`**
  - **По правило, `List` елементите трябва да са `Comparable` елементи, за да се определи подреждането на елементите при сортиране чрез метода `compareTo` на `interface Comparable<T>`**
  - **Алтернативно, `sort` допуска за втори аргумент `interface Comparator` обект за задаване на подреждането на елементите при сортиране**

13b.6.1 sort Алгоритъм

Sort примери

- Сортиране във възходящ ред (Fig. 13b.8)
  - Приложение на метода sort на Collections
- Сортиране в низходящ ред (Fig. 13b.9)
  - Приложение на static метода reverseOrder на Collections
- Сортиране с interface Comparator(Fig. 13b.10)
  - Създаване на потребителски клас, който е Comparator

```
1  // Fig. 19.8: Sort1.java
2  // Using algorithm sort.
3  import java.util.List;
4  import java.util.Arrays;
5  import java.util.Collections;
6
7  public class Sort1
8  {
9     private static final String suits[] =
10       { "Hearts", "Diamonds", "Clubs", "Spades" };
11
12    // display array elements
13    public void printElements()
14    {
15       List< String > list = Arrays.asList( suits ); // create List
16
```

Създава **List** от **масив** от **String** - ове

```
17       // output list
18       System.out.printf( "Unsorted array elements:\n%s\n", list );
19
20       Collections.sort( list ); // sort ArrayList
21
22       // output list
23       System.out.printf( "Sorted array elements:\n%s\n", list );
24    } // end method printElements
25
26    public static void main( String args[] )
27    {
28       Sort1 sort1 = new Sort1();
29       sort1.printElements();
30    } // end main
31 } // end class Sort1
```

Неявно извикване **`toString`** метода на **`list`** за извеждане на данните на **`list`**

Използва **`sort`** за сортиране във възходящ ред на **`list`**

```
Unsorted array elements:
[Hearts, Diamonds, Clubs, Spades]
Sorted array elements:
[Clubs, Diamonds, Hearts, Spades]
```

```
1  // Fig. 19.9: Sort2.java
2  // Using a Comparator object with algorithm sort.
3  import java.util.List;
4  import java.util.Arrays;
5  import java.util.Collections;
6
7  public class Sort2
8  {
9     private static final String suits[] =
10       { "Hearts", "Diamonds", "Clubs", "Spades" };
11
12    // output List elements
13    public void printElements()
14    {
15       List< String > list = Arrays.asList( suits ); // create List
16
```

```
17       // output List elements
18       System.out.printf( "Unsorted array elements:\n%s\n", list );
19
20       // sort in descending order using a comparator
21       Collections.sort( list, Collections.reverseOrder() );
22
23       // output List elements
24       System.out.printf( "Sorted list elements:\n%s\n", list );
25    } // end method printElements
26
27    public static void main( String args[] )
28    {
29       Sort2 sort2 = new Sort2();
30       sort2.printElements();
31    } // end main
32 } // end class Sort2
```

```
Unsorted array elements:
[Hearts, Diamonds, Clubs, Spades]
Sorted list elements:
[Spades, Hearts, Diamonds, Clubs]
```

Методът **reverseOrder** на **class Collections** **връща** обект от тип **Comparator** за **сортиране в низходящ (обратен) ред**

Методът **sort** на **class Collections** **алтернативно** може да има втори аргумент обект от тип **Comparator** за **задаване на наредбата** на сортиране на **List**

```java
// Fig. 19.10: TimeComparator.java
// Custom Comparator class that compares two Time2 objects.
import java.util.Comparator;

public class TimeComparator implements Comparator< Time2 >
{
   public int compare( Time2 time1, Time2 time2 )
   {
      int hourCompare = time1.getHour() - time2.getHou

      // test the hour first
      if ( hourCompare != 0 )
         return hourCompare;

      int minuteCompare =
         time1.getMinute() - time2.getMinute(); // compare minute

      // then test the minute
      if ( minuteCompare != 0 )
         return minuteCompare;

      int secondCompare =
         time1.getSecond() - time2.getSecond(); // compare second

      return secondCompare; // return result of comparing seconds
   } // end method compare
} // end class TimeComparator
```

Потребителски клас **`TimeComparator`** имплементира **interface `Comparator`** и сравнява **`Time2`** обект

Имплементира метода **`compare`** за определяне на наредбата на **`Time2`** обекти

13b.6.1 sort Алгоритъм

Comparator пример

- interface Comparator е обобщаващ клас с параметър за тип (в дадения случай типът е Time2).
- Методът сравнява (редове 7 – 26) Time2 обекти .
- Ред 9 сравнява *часовете, минутите и секундите* на Time2 обекти дадени като аргументи.
- Методът връща **нула**, когато часовете, минутите и секундите **съвпадат**
- Методът връща **положително** число, когато първият аргумент **е по- голям** от втория
- Методът връща **отрицателно** число, когато първият аргумент **е по- малък** от втория

```
1  // Fig. 19.11: Sort3.java
2  // Sort a list using the custom Comparator class TimeComparator.
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.util.Collections;
6
7  public class Sort3
8  {
9     public void printElements()
10    {
11       List< Time2 > list = new ArrayList< Time2 >(); // create List
12
13       list.add( new Time2(  6, 24, 34 ) );
14       list.add( new Time2( 18, 14, 58 ) );
15       list.add( new Time2(  6, 05, 34 ) );
16       list.add( new Time2( 12, 14, 58 ) );
17       list.add( new Time2(  6, 24, 22 ) );
18
```

Няма нужда от **`Arrays.asList()`**

```
19       // output List elements
20       System.out.printf( "Unsorted array elements:\n%s\n", list );
21
22       // sort in order using a comparator
23       Collections.sort( list, new TimeComparator() );
24
25       // output List elements
26       System.out.printf( "Sorted list elements:\n%s\n", list );
27    } // end method printElements
28
29    public static void main( String args[] )
30    {
31       Sort3 sort3 = new Sort3();
32       sort3.printElements();
33    } // end main
34 } // end class Sort3
```

```
Unsorted array elements:
[6:24:34 AM, 6:14:58 PM, 6:05:34 AM, 12:14:58 PM, 6:24:22 AM]
Sorted list elements:
[6:05:34 AM, 6:24:22 AM, 6:24:34 AM, 12:14:58 PM, 6:14:58 PM]
```

Наредбата за сортиране се определя от **`TimeComparator`**

13b.6.2 Алгоритъм `shuffle`

`shuffle`

- **Подрежда по случаен начин елементите на `List`**

Пример- *разбъркване на карти*

Разглеждаме `class Card` (редове 8- 41) представящ карта от тесте от карти. Всяка карта има боя и сила. Редове 10- 12 декларират `enum` типове `Face` и `Suit` представящи съответно `силата` и `боята` на картата.

13b.6.2 Алгоритъм `shuffle`

Пример- ***разбъркване на карти***

Методът `toString` (редове 37- 40) връща `String` представящ силата и боята на `Card` разделени от низа “... `of` ...". При превръщане на `enum` константа в низ, името на променливата се взима за нейно текстово представяне. По тази причина нарушаваме стила за именуване на константи изцяло с главни букви. Това позволява картите да се изписват по- разбираемо (например , `"Ace of Spades"`)

```
1  // Fig. 19.12: DeckOfCards.java
2  // Using algorithm shuffle.
3  import java.util.List;
4  import java.util.Arrays;
5  import java.util.Collections;
6
7  // class to represent a Card in a deck of cards
8  class Card
9  {
10    public static enum Face { Ace, Deuce, Three, Four, Five, Six,
11       Seven, Eight, Nine, Ten, Jack, Queen, King  };
12    public static enum Suit { Clubs, Diamonds, Hearts, Spades };
13
14    private final Face face; // face of card
15    private final Suit suit; // suit of card
16
17    // two-argument constructor
18    public Card( Face cardFace, Suit cardSuit )
19    {
20        face = cardFace; // initialize face of card
21        suit = cardSuit; // initialize suit of card
22    } // end two-argument Card constructor
23
24    // return face of the card
25    public Face getFace()
26    {
27       return face;
28    } // end method getFace
29
```

```
30    // return suit of Card
31    public Suit getSuit()
32    {
33       return suit;
34    } // end method getSuit
35
36    // return String representation of Card
37    public String toString()
38    {
39       return String.format( "%s of %s", face, suit );
40    } // end method toString
41 } // end class Card
42
43 // class DeckOfCards declaration
44 public class DeckOfCards
45 {
46    private List< Card > list; // declare List that will store Cards
47
48    // set up deck of Cards and shuffle
49    public DeckOfCards()
50    {
51       Card[] deck = new Card[ 52 ];
52       int count = 0; // number of cards
53
```

```
54       // populate deck with Card objects
55       for ( Card.Suit suit : Card.Suit.values() )
56       {
57          for ( Card.Face face : Card.Face.values() )
58          {
59             deck[ count ] = new Card( face, suit );
60             count++;
61          } // end for
62       } // end for
63
64       list = Arrays.asList( deck ); // get List
65       Collections.shuffle( list );  // shuffle deck
66    } // end DeckOfCards constructor
67
68    // output deck
69    public void printCards()
70    {
71       // display 52 cards in two columns
72       for ( int i = 0; i < list.size(); i++ )
73          System.out.printf( "%-20s%s", list.get( i ),
74             ( ( i + 1 ) % 2 == 0 ) ? "\n" : "\t" );
75    } // end method printCards
76
77    public static void main( String args[] )
78    {
79       DeckOfCards cards = new DeckOfCards();
80       cards.printCards();
81    } // end main
82 } // end class DeckOfCards
```

Използваме **enum** тип **Suit** извън class **Card** като **enum** реферираме типа **Suit** чрез името на **Card** разделени с точка (**.**)

Използваме **enum** тип **Face** извън class **Card** като **enum** реферираме типа **Face** чрез името на **Card** разделени с точка (**.**)

Извиква **static** метода **asList** от **class Arrays** за получаване на **List** **изглед** на масива deck

Използваме метод **shuffle** на **class Collections** за разбъркване на **List** по случаен начин, чрез **Collections.shuffle()**

```
King of Diamonds    Jack of Spades
Four of Diamonds    Six of Clubs
King of Hearts      Nine of Diamonds
Three of Spades     Four of Spades
Four of Hearts      Seven of Spades
Five of Diamonds    Eight of Hearts
Queen of Diamonds   Five of Hearts
Seven of Diamonds   Seven of Hearts
Nine of Hearts      Three of Clubs
Ten of Spades       Deuce of Hearts
Three of Hearts     Ace of Spades
Six of Hearts       Eight of Diamonds
Six of Diamonds     Deuce of Clubs
Ace of Clubs        Ten of Diamonds
Eight of Clubs      Queen of Hearts
Jack of Clubs       Ten of Clubs
Seven of Clubs      Queen of Spades
Five of Clubs       Six of Spades
Nine of Spades      Nine of Clubs
King of Spades      Ace of Diamonds
Ten of Hearts       Ace of Hearts
Queen of Clubs      Deuce of Spades
Three of Diamonds   King of Clubs
Four of Clubs       Jack of Diamonds
Eight of Spades     Five of Spades
Jack of Hearts      Deuce of Diamonds
```

13b.6.3 Алгоритми reverse, fill, copy, max и min

Алгоритми за:

reverse

- **Обръщане на <u>поредността</u> на List елементи(не сортира)**

fill

- **Запълване на List елементи с данни (*при инициализиране*)**

copy

- **Създаване на копие на List**

max

- **Връща максималния елемент в List**

min

- **Връща минималния елемент в List**

```java
// Fig. 19.13: Algorithms1.java
// Using algorithms reverse, fill, copy, min and max.
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Algorithms1
{
   private Character[] letters = { 'P', 'C', 'M' };
   private Character[] lettersCopy;
   private List< Character > list;
   private List< Character > copyList;

   // create a List and manipulate it with methods from Collections
   public Algorithms1()
   {
      list = Arrays.asList( letters ); // get List
      lettersCopy = new Character[ 3 ];
      copyList = Arrays.asList( lettersCopy ); // list view of lettersCopy

      System.out.println( "Initial list: " );
      output( list );

      Collections.reverse( list ); // reverse order
      System.out.println( "\nAfter calling reverse: " );
      output( list );
```

Получаваме `List` от `Characters`

Използва метод **`reverse`** от **class `Collections`** за **обръщане на поредността** на елементите на `List`

```
28       Collections.copy( copyList, list ); // copy List
29       System.out.println( "\nAfter copying: " );
30       output( copyList );
31
32       Collections.fill( list, 'R' ); // fill list with Rs
33       System.out.println( "\nAfter calling fill: " );
34       output( list );
35    } // end Algorithms1 constructor
36
37    // output List information
38    private void output( List< Character > listRef )
39    {
40       System.out.print( "The list is: " );
41
42       for ( Character element : listRef )
43          System.out.printf( "%s ", element );
44
45       System.out.printf( "\nMax: %s", Collections.max( listRef ) );
46       System.out.printf( "  Min: %s\n", Collections.min( listRef ) );
47    } // end method output
48
```

Използва метод **copy** от **class** **Collections** за получаване на **копие** на **List**

Използва метод **fill** от class **Collections** за **запълване** на **List** със символа 'R'

Намира **максималната** стойност на **List**

Намира **минималната** стойност на **List**

```
49    public static void main( String args[] )
50    {
51       new Algorithms1();
52    } // end main
53 } // end class Algorithms1
```

```
Initial list:
The list is: P C M
Max: P  Min: C

After calling reverse:
The list is: M C P
Max: P  Min: C

After copying:
The list is: M C P
Max: P  Min: C

After calling fill:
The list is: R R R
Max: R  Min: R
```

13b.6.4 Алгоритъм за binarySearch

binarySearch

- Търси съвпадение на даден обект с елемент на `List`
  - **Връща индекса на съвпадащия** елемент от `List` при наличие на съвпадение
  - **Връща отрицателна стойност** ако няма съвпадение (*също, както при* `Arrays.binarySearch()`)
    - Пресмята се индекс място за вмъкване
    - Взима се отрицателната стойност на индекса
    - Изважда се 1-ца от индекса и така получената стойност се връща от `Collections.binarySearch()` при липса на съвпадение

```
1  // Fig. 19.14: BinarySearchTest.java
2  // Using algorithm binarySearch.
3  import java.util.List;
4  import java.util.Arrays;
5  import java.util.Collections;
6  import java.util.ArrayList;
7
8  public class BinarySearchTest
9  {
10    private static final String colors[] = { "red", "white",
11       "blue", "black", "yellow", "purple", "tan", "pink" };
12    private List< String > list; // ArrayList reference
13
14    // create, sort and output list
15    public BinarySearchTest()
16    {
17       list = new ArrayList< String >( Arrays.asList( colors ) );
18       Collections.sort( list ); // sort the ArrayList
19       System.out.printf( "Sorted ArrayList: %s\n", list );
20    } // end BinarySearchTest constructor
21
```

Сортира `List` във възходящ ред

```
22    // search list for various values
23    private void search()
24    {
25       printSearchResults( colors[ 3 ] ); // first item
26       printSearchResults( colors[ 0 ] ); // middle item
27       printSearchResults( colors[ 7 ] ); // last item
28       printSearchResults( "aqua" ); // below lowest
29       printSearchResults( "gray" ); // does not exist
30       printSearchResults( "teal" ); // does not exist
31    } // end method search
32
33    // perform searches and display search result
34    private void printSearchResults( String key )
35    {
36       int result = 0;
37
38       System.out.printf( "\nSearching for: %s\n", key );
39       result = Collections.binarySearch( list, key );
40
41       if ( result >= 0 )
42          System.out.printf( "Found at index %d\n", result );
43       else
44          System.out.printf( "Not Found (%d)\n",result );
45    } // end method printSearchResults
46
```

Използва метод **`binarySearch`** от **class `Collections`** за търсене на **съвпадение** на даден `key` в `list`

```
47    public static void main( String args[] )
48    {
49       BinarySearchTest binarySearchTest = new BinarySearchTest();
50       binarySearchTest.search();
51    } // end main
52 } // end class BinarySearchTest
```

```
Sorted ArrayList: [black, blue, pink, purple, red, tan, white, yellow]

Searching for: black
Found at index 0

Searching for: red
Found at index 4

Searching for: pink
Found at index 2

Searching for: aqua
Not Found (-1)

Searching for: gray
Not Found (-3)

Searching for: teal
Not Found (-7)
```

13b.6.5 Алгоритми addAll, frequency и disjoint

addAll

- **Вмъква всичките елементи на масив в дадена колекция**

frequency

- **Пресмята колко пъти един елемент се среща в дадена колекция**

disjoint

- **Определя дали две колекции имат едни и същи обекти**

```
1  // Fig. 19.15: Algorithms2.java
2  // Using algorithms addAll, frequency and disjoint.
3  import java.util.List;
4  import java.util.Vector;
5  import java.util.Arrays;
6  import java.util.Collections;
7
8  public class Algorithms2
9  {
10    private String[] colors = { "red", "white", "yellow", "blue" };
11    private List< String > list;
12    private Vector< String > vector = new Vector< String >();
13
14    // create List and Vector
15    // and manipulate them with methods from Collections
16    public Algorithms2()
17    {
18       // initialize list and vector
19       list = Arrays.asList( colors );
20       vector.add( "black" );
21       vector.add( "red" );
22       vector.add( "green" );
23
24       System.out.println( "Before addAll, vector contains: " );
25
```

```
26       // display elements in vector
27       for ( String s : vector )
28          System.out.printf( "%s ", s );
29
30       // add elements in colors to list
31       Collections.addAll( vector, colors );
32
33       System.out.println( "\n\nAfter addAll, vector contains: " );
34
35       // display elements in vector
36       for ( String s : vector )
37          System.out.printf( "%s ", s );
38
39       // get frequency of "red"
40       int frequency = Collections.frequency( vector, "red" );
41       System.out.printf(
42          "\n\nFrequency of red in vector: %d\n", frequency );
43
```

Извиква метод **addAll** за **добавяне на всички** елементи от масива **colors** към колекцията **vector**

Пресмята брой съвпадения на **String** “red” в **Collection vector** чрез метод **frequency**

```
44        // check whether list and vector have elements in common
45        boolean disjoint = Collections.disjoint( list, vector );
46
47        System.out.printf( "\nlist and vector %s el
48           ( disjoint ? "do not have" : "have"  ) );
49     } // end Algorithms2 constructor
50
51     public static void main( String args[] )
52     {
53        new Algorithms2();
54     } // end main
55  } // end class Algorithms2
```

Извиква метод **disjoint** за определяне дали **колекциите** **list** и **vector** имат **общи елементи**

```
Before addAll, vector contains:
black red green

After addAll, vector contains:
black red green red white yellow blue

Frequency of red in vector: 2

list and vector have elements in common
```

13b.7 class Stack от пакета java.util

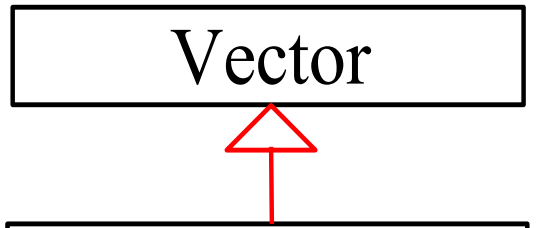
Stack

- **Имплементира stack структура от данни**
- **Производен клас на class Vector**
- **Служи за съхраняване на референции към обекти като LIFO структура**
- **Основни методи**

  push

  pop

13b.7 class Stack от пакета java.util



+empty(): boolean	Returns true if this stack is empty
+peek(): Object	Returns the top element in this stack
+pop(): Object	Returns and removes the top element in this stack
+push(o: Object) : Object	Adds a new element to the top of this stack
+search(o: Object) : int	Returns the position of the specified element in this stack

```java
// Fig. 19.16: StackTest.java
// Program to test java.util.Stack.
import java.util.Stack;
import java.util.EmptyStackException;

public class StackTest
{
   public StackTest()
   {
      Stack< Number > stack = new Stack< Number >();

      // create numbers to store in the stack
      Long longNumber = 12L;
      Integer intNumber = 34567;
      Float floatNumber = 1.0F;
      Double doubleNumber = 1234.5678;

      // use push method
      stack.push( longNumber ); // push a long
      printStack( stack );
      stack.push( intNumber ); // push an int
      printStack( stack );
      stack.push( floatNumber ); // push a float
      printStack( stack );
      stack.push( doubleNumber ); // push a double
      printStack( stack );

```

Създава празен **Stack** от **тип Number**

Stack методът **push** добавя елемент върху **Stack**

```
28      // remove items from stack
29      try
30      {
31         Number removedObject = null;
32
33         // pop elements from stack
34         while ( true )
35         {
36            removedObject = stack.pop(); // use pop method
37            System.out.printf( "%s popped\n", removedObject );
38            printStack( stack );
39         } // end while
40      } // end try
41      catch ( EmptyStackException emptyStackException )
42      {
43         emptyStackException.printStackTrace();
44      } // end catch
45   } // end StackTest constructor
46
47   private void printStack( Stack< Numbe
48   {
49      if ( stack.isEmpty() )
50         System.out.print( "stack is empty\n\n" ); // the stack is empty
51      else  // stack is not empty
52      {
53         System.out.print( "stack contains: " );
54
```

Stack методът **pop** **премахва** елемент от върха на **Stack**

Stack методът **pop** **хвърля** **EmpyStackException**

при празен Stack

Stack методът r**isEmpty** връща **true** ако **Stack** е празен

```
55          // iterate through the elements
56          for ( Number number : stack )
57             System.out.printf( "%s ", number );
58
59          System.out.print( "(top) \n\n" ); // indicates top of the stack
60       } // end else
61    } // end method printStack
62
63    public static void main( String args[] )
64    {
65       new StackTest();
66    } // end main
67 } // end class StackTest
```

```
stack contains: 12 (top)

stack contains: 12 34567 (top)

stack contains: 12 34567 1.0 (top)

stack contains: 12 34567 1.0 1234.5678 (top)

1234.5678 popped
stack contains: 12 34567 1.0 (top)

1.0 popped
stack contains: 12 34567 (top)

34567 popped
stack contains: 12 (top)

12 popped
stack is empty

java.util.EmptyStackException
        at java.util.Stack.peek(Unknown Source)
        at java.util.Stack.pop(Unknown Source)
        at StackTest.<init>(StackTest.java:36)
        at StackTest.main(StackTest.java:65)
```

Обичайна грешка при програмиране 13b.1

Понеже `Stack` е производен на `Vector`, всеки `public Vector` метод може да се извика от `Stack` обект, даже и в случай, когато той не е съгласуван с опеделението за `Stack`. Например, `Vector` метода `add` може да вмъква елемент навсякъде в `Stack` и това може да "развали" the `Stack`. Когато се работи със `Stack`, трябва да се използват само `push` и `pop` методите за добавяне и премахване на `Stack` елементи.

13b.8 class PriorityQueue и interface Queue

interface Queue

- Нов интерфейс на колекция въведен с J2SE 5.0
- Производен на interface Collection
- Доставя допълнителни методи за вмъкване, изтриване и четене (разглеждане) на елементи на опашка (queue)

13b.8 class PriorityQueue и interface Queue

class PriorityQueue

- Имплементира **interface Queue**
- Подрежда елементите **по естествения** им ред
  - **Типично**, редът се задава с метода **compareTo** на **Comparable**
  - **Алтернативно**, може да се използва и **Comparator** чрез конструктора на **class PriorityQueue**
- Основни методи, **служат за** :
  - **offer – добавяне на елемент**
  - **poll – изтриване на елемент в началото на опашката (елемента с най- висок приоритет)**
  - **peek – чете елемента в началото на опашката**
  - **size – дава текущия брой елементи**

```java
// Fig. 19.17: PriorityQueueTest.java
// Standard library class PriorityQueue test program.
import java.util.PriorityQueue;

public class PriorityQueueTest
{
   public static void main( String args[] )
   {
      // queue of capacity 11
      PriorityQueue< Double > queue = new PriorityQueue< Double >();

      // insert elements to queue
      queue.offer( 3.2 );
      queue.offer( 9.8 );
      queue.offer( 5.4 );

      System.out.print( "Polling from queue: " );

      // display elements in queue
      while ( queue.size() > 0 )
      {
         System.out.printf( "%.1f ", queue.peek() ); // view top element
         queue.poll(); // remove top element
      } // end while
   } // end main
} // end class PriorityQueueTest
```

```
Polling from queue: 3.2 5.4 9.8
```

Създава `PriorityQueue` с елементи `Double` с начален капацитет от **11 елемента и подрежда** елементите **по естествения им ред (възходящ)**

Използва метод **`offer`** за **добавяне** на елементи към опашка с приритет

Използва метод **`size`** за определяне **дали опашката е празна**

Използва метод **`peek`** за **четене** на елемента **в *началото на опашката***

Използва метод **`poll`** за **изтриване** на елемента **в *началото на опашката, връща null ако опашката е празна***

13b.9 Множества

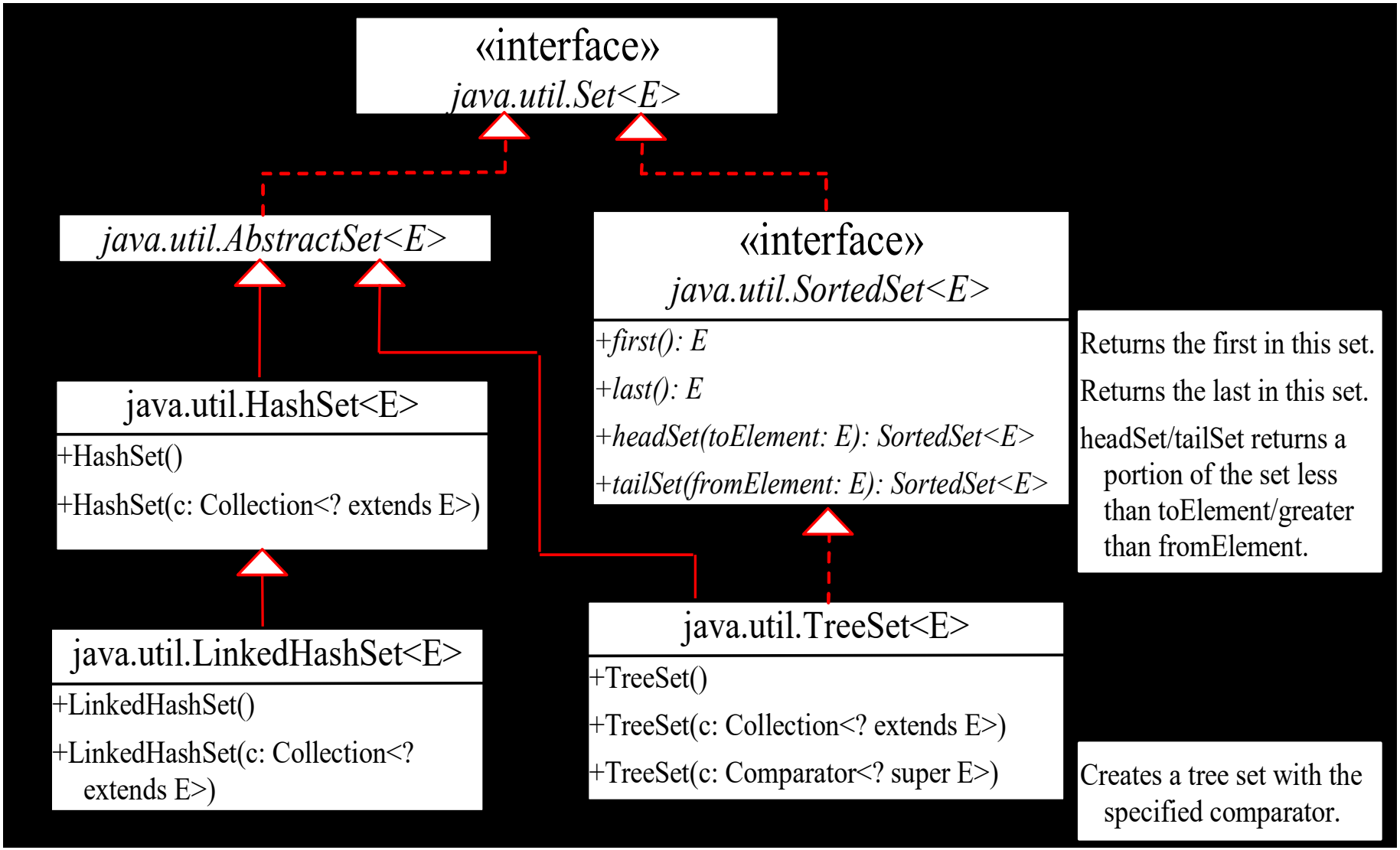
interface Set- определение

- Това е Collection , която съдържа **не повтарящи** се елементи

Приложения- структури данни

- class HashSet, съхранява елементи в hash таблица
- class TreeSet, съхранява елементи в дърво

13b.9 Множества



13b.9 Множества- hash таблица

Hashing in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties. A true Hashing function must follow this rule:

***Hash function should return the same hash code each and every time, when function is applied on same or equal objects. In other words, two equal objects must produce same hash code consistently*.**

13b.9 Множества- hash таблица

Note: All objects in java inherit a default implementation of `hashCode()` function defined in `Object` class. This function produce hash code by typically converting the internal address of the object into an integer, thus producing different hash codes for all different objects

A map by definition is : "***An object that maps keys to values***"

13b.9 Множества- hash таблица

Съхраняването и извличането на данни от масив е ефективно, ако данните са логически свързани с индекс от масива и когато ключовете за достъп до тези данни са неповтарящи се и плътно запълнени със стойности.

Например, ако имаме 100 служителя с десет цифрени ЕГН и искате да извличате данни по ЕГН като ключ, то ще трябва масив с индекси (000,000,0000- 9,999,999,999) и тогава използването на масив не е оправдано.

13b.9 Множества- hash таблица

В редица приложение има същия проблем:

- **ключовете са от неподходящ тип (не са положителни)**
  или
- **са от подходящ тип, но не са разпределени плътно в един голям интервал.**

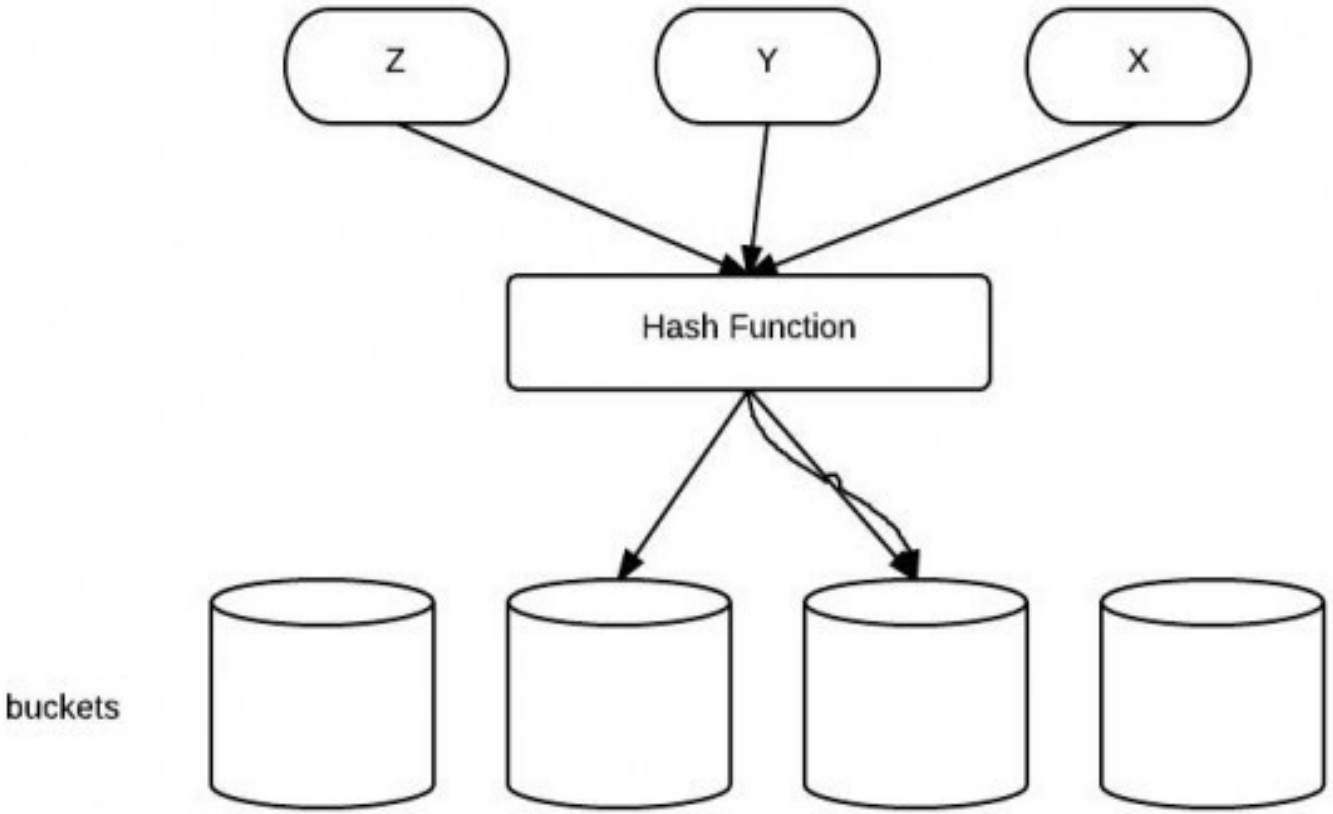
Нужна е схема за преобразуване на ЕГН, ид. номера на части и пр. в неповтарящи се индекси на масив. Тогава когато има нужда да пишем в масив по тази схема намираме за всеки ключ съответния му индекс в масива и така съответната данна може да бъде записана в масива.

13b.9 Множества- hash таблица

Схемата, по която се осъществява преобразуване на неповтарящ се `ключ` в `индекс` на масив и обратно е в основата на техника за изобразяване на стойности, наречена *хеширане* (`hashing`).

Математическата функция, по която се извършва това преобразуване се нарича *хеш функция.*

13b.9 Множества- hash таблица



13b.9 Множества- hash таблица

Важно: Няма гаранции, че hashCode() връща същия резултат при различни изпълнения на програмата. Справка в JavaDoc:

„Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.“

13b.9 Множества- hash таблица

Извод: hashCode() не е препоръчително да се използва в приложения за разпределено смятане. Отдалеченият обект може да са е различен hashCode от обект на локалната машина, въпреки че двата обекта са равни. Сорс кода не трябва да зависи от конкретни стойности на hashCode.

Извод: За създаване на уникални идентификатори се препоръчва използване на UUID (universally unique identifier)

13b.9 Множества- hash таблица

```
import java.util.UUID;

public class GenerateUUID
{

    public static final void main(String... aArgs)
    {
        //generate random UUIDs
        UUID idOne = UUID.randomUUID();
        UUID idTwo = UUID.randomUUID();
        log("UUID One: " + idOne);
        log("UUID Two: " + idTwo);
    }

    private static void log(Object aObject)
    {
        System.out.println(String.valueOf(aObject));
    }
}
```

```
run:
UUID One: b5b74ef7-dd5b-4844-b3dd-30343edcb0d5
UUID Two: b1dd6121-4d48-44ef-b4a3-ba2155363452
BUILD SUCCESSFUL (total time: 1 second)
```

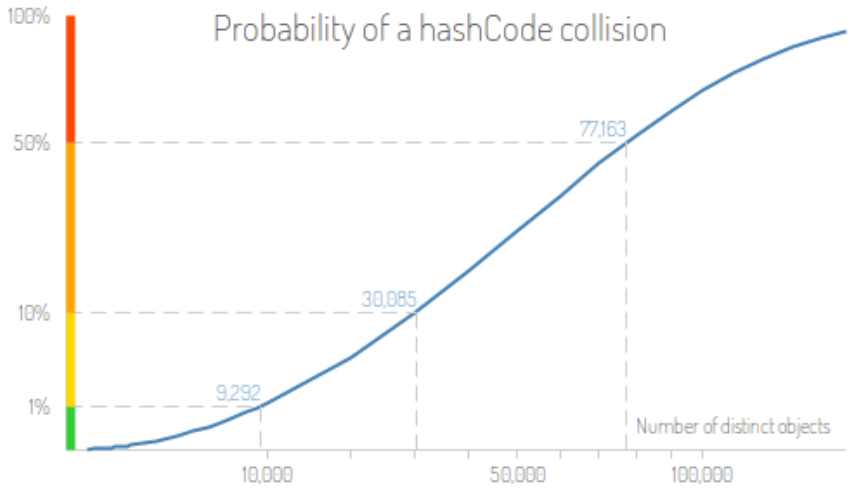

13b.9 Множества- hash таблица

Възможно е дублиране (колизия) на хеш стойностите. Един възможен подход е всеки елемент на масива, в който записваме стойностите да се разглежда като “`кутия`” , в която се съхраняват *ключовете и съответните им стойности* с дублирана *хеш* стойност на ключа. Тези двойки *ключ- стойност* и могат да се съхраняват в тази “`кутия`” под формата на свързан списък .

Така са реализирани класовете `Hashtable` и `HashMap`

13b.9 Множества- hash таблица

Вероятността от колизия на `hashCode` при стандартната й реализация в class `Object` е 50% при наличие на повече от 77, 163 различни обекта- `hashCode()` връща `int` стойност, която е изображение в 2^{32} на адреса в паметта на обекта



13b.9 Множества- hash таблица

Пример: различни стрингове- еднакъв hashCode

([Birthday paradox])

```
public class TestHash
{
    public static void main(String[] args)
    {       String aa = "Aa";
            String bb = "BB";
            System.out.println(bb.hashCode());
            System.out.println(aa.hashCode());
    }
}
run:
2112
2112
BUILD SUCCESSFUL (total time: 2 seconds)
```


13b.9 Множества- hash таблица

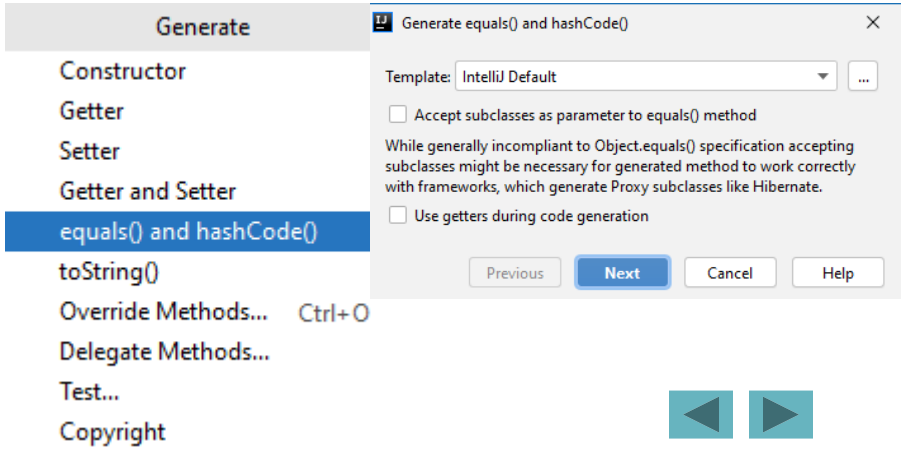
Задължителна връзка (contract) между equals() и hashCode()

1. Ако два обекта са равни, то те трябва да имат еднакъв hash code.

2. Когато два обекта имат еднакъв hashcode, те може и да не са равни.

Извод: *Memodume* equals() и hashCode() трябва да се предефинират заедно.

Виж IntelliJ: *right Click->Generate> equals() and hashCode()*



```
1  // Fig. 19.18: SetTest.java
2  // Using a HashSet to remove duplicates.
3  import java.util.List;
4  import java.util.Arrays;
5  import java.util.HashSet;
6  import java.util.Set;
7  import java.util.Collection;
8
9  public class SetTest
10 {
11    private static final String colors[] = { "red", "white", "blue",
12       "green", "gray", "orange", "tan", "white", "cyan",
13       "peach", "gray", "orange" };
14
15    // create and output ArrayList
16    public SetTest()
17    {
18       List< String > list = Arrays.asList( colors );
19       System.out.printf( "ArrayList: %s\n", list );
20       printNonDuplicates( list );
21    } // end SetTest constructor
22
```

Създава **`List`** който съдържа **`String`** обекти по даден масив от низове

```
23    // create set from array to eliminate duplicates
24    private void printNonDuplicates( Collection< String > collection )
25    {
26       // create a HashSet
27       Set< String > set = new HashSet< String >( collection );
28
29       System.out.println( "\nNonduplicates are: " );
30
31       for ( String s : set )
32          System.out.printf( "%s ", s );
33
34       System.out.println();
35    } // end method printNonDuplicates
36
37    public static void main( String args[] )
38    {
39       new SetTest();
40    } // end main
41 } // end class SetTest
```

Метод **printNonDuplicates** има аргумент **Collection** от тип **String**

Създава **HashSet** от **Collection** аргумента

```
ArrayList: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray,
orange]

Nonduplicates are:
red cyan white tan gray green orange blue peach
```

13b.9 Множества

`interface` SortedSet- определение

- **Сортирано множество от елементи или в естествен ред (във възходящ ред) или в ред определен от даден обект `Comparator`.**

Приложение:

`class TreeSet` ***имплементира*** `SortedSet`.

Пример:

- **Преобразуваме масив от низове в `TreeSet`.**
- **Низовете се сортират в процеса на добавяне в `TreeSet`.**
- **Илюстрира работа с подмножество от елементи**

```
1  // Fig. 19.19: SortedSetTest.java
2  // Using TreeSet and SortedSet.
3  import java.util.Arrays;
4  import java.util.SortedSet;
5  import java.util.TreeSet;
6
7  public class SortedSetTest
8  {
9     private static final String names[] = { "yellow", "green",
10       "black", "tan", "grey", "white", "orange", "red", "green" };
11
12    // create a sorted set with TreeSet, then manipulate it
13    public SortedSetTest()
14    {
15       // create TreeSet
16       SortedSet< String > tree =
17          new TreeSet< String >( Arrays.asList( names ) );
18
19       System.out.println( "sorted set: " );
20       printSet( tree ); // output contents of tree
21
```

Създава **TreeSet** от **масива** **names**

```
22       // get headSet based on "orange"
23       System.out.print( "\nheadSet (\"orange\"):   " );
24       printSet( tree.headSet( "orange" ) );
25
26       // get tailSet based upon "orange"
27       System.out.print( "tailSet (\"orange\"):   " );
28       printSet( tree.tailSet( "orange" ) );
29
30       // get first and last elements
31       System.out.printf( "first: %s\n", tree.first() );
32       System.out.printf( "last : %s\n", tree.last() );
33    } // end SortedSetTest constructor
34
35    // output set
36    private void printSet( SortedSet< String > set )
37    {
38       for ( String s : set )
39          System.out.printf( "%s ",  s );
40
```

Използва `TreeSet` метода `headSet` за получаване на `TreeSet` **подмножество** от елементи, които са по-малки от "`orange`"

Използва `TreeSet` метода `tailSet` за получаване на `TreeSet` **подмножество** от елементи, които са по-големи от "`orange`"

Методите `first` и `last` дават най- малкия и най-голямия `TreeSet` елемент , съответно.

Извежда **на печат подмножество** от елементи на `TreeSet`

```
41       System.out.println();
42    } // end method printSet
43
44    public static void main( String args[] )
45    {
46       new SortedSetTest();
47    } // end main
48 } // end class SortedSetTest
```

```
sorted set:
black green grey orange red tan white yellow

headSet ("orange"):  black green grey
tailSet ("orange"):  orange red tan white yellow
first: black
last : yellow
```

13b.9 Множества

Редове 16- 17 изпозват конструктор на `TreeSet` от `String` елементи, който въвежда елементите на масив names от низове и преобразува `TreeSet` до `SortedSet`.

Както `SortedSet` , така и `TreeSet` са *пораждащи класове(*generic) .

Ред 24 извиква TreeSet метода `headSet` за получаване на *подмножество* от елементи.

Ако се правят промени върху подмножеството, то те се отразяват и на основното дърво `TreeSet`.

13b.10 Съответствия

`interface Map`

- Свързва ключ към всяка стойност
- Не може да има дублирани ключове
  - Едно- към- Едно изображение
- `Map` се различава от `Set` по това, че `Map` съдържа *ключове и стойности* , докато `Set` съдържа само стойности.

Приложения в класове :

`Hashtable` - съхранява елементи в hash таблица

`HashMap` - съхранява елементи в hash таблица

`TreeMap` - съхранява елементи в дърво.

13b.10 Съответствия

`Hashtable` is synchronized, whereas `HashMap` is not. This makes `HashMap` better for non-threaded applications, as `unsynchronized` Objects typically perform better than `synchronized` ones.

`Hashtable` does not allow `null` keys or values. `HashMap` allows one `null` key and any number of `null` values.

13b.10 Множества- hash таблица

Key Notes (http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/)

1. Data structure to store `Entry` objects is an array named `table` of type `Entry[]`. Class Entry<K ,V> implements Map.Entry<K ,V> and has key and value mapping stored as attributes. Key has been marked as final and two more fields are there: next of type Entry and hash of type int.
2. A particular index location in array `table` is referred as bucket, because it can hold the first element of a `LinkedList` of `Entry` objects.
3. Key object's `hashCode`() is required to calculate the index location of an `Entry` object.
4. Key object's `equals`() method is used to maintain uniqueness of Keys in map.
5. Value object's `hashCode`() and `equals`() method are not used in `HashMap's get`() and `put`() methods.
6. Hash code for `null` keys is always `zero`, and such `Entry` object is always stored in `zero` index in `Entry[]`.

13b.10 Съответствия

`interface SortedMap`

- Производен на `Map`
- Съхранява ключовете в сортиран вид

Приложение в клас:

`class TreeMap`

имплементира

`SortedMap`

13b.10 Съответствия

Map	
+clear(): void	Removes all mappings from this map
+containsKey(key: Object): boolean	Returns true if this map contains a mapping for the specified key.
+containsValue(value: Object): boolean	Returns true if this map maps one or more keys to the specified value.
+entrySet(): Set	Returns a set consisting of the entries in this map
+get(key: Object): Object	Returns the value for the specified key in this map
+isEmpty(): boolean	Returns true if this map contains no mappings
+keySet(): Set	Returns a set consisting of the keys in this map
+put(key: Object, value: Object): Object	Puts a mapping in this map
+putAll(m: Map): void	Adds all mappings from m to this map
+remove(key: Object): Object	Removes the mapping for the specified key
+size(): int	Returns the number of mappings in this map
+values(): Collection	Returns a collection consisting of values in this map

```
1  // Fig. 19.20: WordTypeCount.java
2  // Program counts the number of occurrences of each word in a string
3  import java.util.StringTokenizer;
4  import java.util.Map;
5  import java.util.HashMap;
6  import java.util.Set;
7  import java.util.TreeSet;
8  import java.util.Scanner;
9
10 public class WordTypeCount
11 {
12    private Map< String, Integer > map;
13    private Scanner scanner;
14
15    public WordTypeCount()
16    {
17       map = new HashMap< String, Integer >(); // create HashMap
18       scanner = new Scanner( System.in ); // create scanner
19       createMap(); // create map based on user input
20       displayMap(); // display map content
21    } // end WordTypeCount constructor
22
```

Създава празен **HashMap** с **капацитет** по подразбиране 16 и подразбиращ се **фактор на запълване** 0.75. Ключовете са от тип **String** а стойностите са от тип **Integer**

```
23  // create map from user input
24  private void createMap()
25  {
26     System.out.println( "Enter a string:" ); // prompt for user input
27     String input = scanner.nextLine();
28
29     // create StringTokenizer for input
30     StringTokenizer tokenizer = new StringTokenizer( input );
31
32     // processing input text
33     while ( tokenizer.hasMoreTokens() ) // while more input
34     {
35        String word = tokenizer.nextToken().toLowerCase(); // get word
36
37        // if the map contains the word
38        if ( map.containsKey( word ) ) // is word in map
39        {
40           int count = map.get( word ); // get current count
41           map.put( word, count + 1 );  // increment count
42        } // end if
43        else
44           map.put( word, 1 ); // add new word with a count of 1 to map
45     } // end while
46  } // end method createMap
47
```

StringTokenizer разделя входния низ на думи (ключ)- броят на повторение на дума е стойността

Map метода containsKey определя **дали ключът** даден като аргумент **е в хеш** таблицата

StringTokenizer метода hasMoreTokens определя дали има още думи

StringTokenizer метода nextToken служи за намиране на следващата дума

Методът get намира съответния ключ по зададената като аргумент стойност

Нарастваме с 1 стойността и с метод put записваме новата стойност на ключа (word)

Създаваме нов елемент на map, където word е ключ , а Integer обект на числото 1 е стойноста

```
48  // display map content
49  private void displayMap()
50  {
51     Set< String > keys = map.keySet(); // get keys
52
53     // sort keys
54     TreeSet< String > sortedKeys = new TreeSet< String >( keys );
55
56     System.out.println( "Map contains:\nKey\t\tValue" );
57
58     // generate output for each key in map
59     for ( String key : sortedKeys )
60        System.out.printf( "%-10s%10s\n", key, map.get( key ) );
61
62     System.out.printf(
63        "\nsize:%d\nisEmpty:%b\n", map.size(), map.isEmpty() );
64  } // end method displayMap
65
```

Използваме **HashMap** метода **keySet** за извличане на множеството от ключове

Четем ключ(**key**) и съответната му стойност в **map**

Извикваме метода **size** на **Map** за намиране на двойките (ключ-стойност) в **map**

Извикваме метода **isEmpty** на **Map** за да определим дали **map** е празно

```
66    public static void main( String args[] )
67    {
68       new WordTypeCount();
69    } // end main
70 } // end class WordTypeCount
```

```
Enter a string:
To be or not to be: that is the question Whether 'tis nobler to suffer
Map contains:
Key                  Value
'tis                    1
be                      1
be:                     1
is                      1
nobler                  1
not                     1
or                      1
question                1
suffer                  1
that                    1
the                     1
to                      3
whether                 1

size:13
isEmpty:false
```

13b.10 Съответствия

Няма утвърден начин за обхождане на съответствие (`Map`). Когато има нужда да се обходят елементите на `Map` обект които удовлетворяват определено условие първо е нужно да се създаде "изглед" или "представяне" (`view`) на `Map` обекта.

Съществуват три представяния

- **Представяне като `key-value` двойки, използва се метод `entrySet()` , връща `Set<Map.entry>` като отделните елементи са достъпни чрез методите getKey() и getValue()**
- **Представяне като всички `keys` , използва се метод `keySet()` , връща Set представяне на ключовете на `Map` обекта**
- **Представяне като всички `values` , използва се метод `values()` връща Collections представяне на стойностите на `Map` обекта (премахване на елемент от това представяне води до премахване на двойката `key-value` и в `Map` обекта)**

13b.10 Съответствия

Няма утвърден начин за обхождане на съответствие (`Map`). Когато има нужда да се обходят елементите на `Map` обект които удовлетворяват определено условие първо е нужно да се създаде "`изглед`" или "`представяне`" (`view`) на `Map` обекта.

Съществуват три представяния

- **Представяне като `key-value` двойки, използва се метод `entrySet()` , връща `Set<Map.Entry>` като отделните елементи са достъпни чрез методите getKey() и getValue()**
- **Представяне като всички `keys` , използва се метод `keySet()` , връща Set представяне на ключовете на `Map` обекта**
- **Представяне като всички `values` , използва се метод `values()` връща Collections представяне на стойностите на `Map` обекта (премахване на елемент от това представяне води до премахване на двойката `key-value` и в `Map` обекта)**

```
1. import java.util.* ;
2. public class H
3. {
4.   static HashMap first = new HashMap();
5.   static
6.   {
7.      first.put("20030120" , new Integer (56));
8.      first.put("20030118" , new Integer (19));
9.      first.put("20030125" , new Integer (25));
10.     first.put("20030122" , new Integer (32));
11.     first.put("20030117" , new Integer (67));
12.     first.put("20030123" , new Integer (34));
13.     first.put("20030124" , new Integer (42));
14.     first.put("20030121" , new Integer (19));
15.     first.put("20030119" , new Integer (98));
16.   }
```

Сортиране на Map по стойностите

```
17.     public static void main( String[] args )
18.     {
19.         ArrayList as = new ArrayList( first.entrySet() );
20.
21.         Collections.sort( as , new Comparator() {
22.             public int compare( Object o1 , Object o2 )
23.             {
24.                 Map.Entry e1 = (Map.Entry)o1 ;
25.                 Map.Entry e2 = (Map.Entry)o2 ;
26.                 Integer first = (Integer)e1.getValue();
27.                 Integer second = (Integer)e2.getValue();
28.                 return first.compareTo( second );
29.             }
30.         });
31.
32.         Iterator i = as.iterator();
33.         while ( i.hasNext() )
34.         {
35.             System.out.println( (Map.Entry)i.next() );
36.         }
37.     }
38. }
```

Създаваме **Set<Map.entry>** представяне и **Comparator** за сортиране по възходящ ред на стойностите

```
1. public class EntryValueComparator implements Comparator{
2.       public int compare(Object o1, Object o2) {
3.             return compare((Map.Entry)o1, (Map.Entry)o2);
4.       }
5.       public int compare(Map.Entry e1, Map.Entry e2) {
6.             int cf = ((Comparable)e1.getValue()).compareTo(e2.getValue());
7.             if (cf == 0) {
8.                   cf = ((Comparable)e1.getKey()).compareTo(e2.getKey());
9.             }
10.             return cf;
11.       }
12. }
```

В най- общ вид **Comparator** може да се напише по следния начин

```
private static Map<String, Integer> first = new Hashtable<>();
private static Random rand = new Random();
public static void main(String[] args) {
    // write your code here
    for (int i = 0; i < 10; i++) {
        int rNum = rand.nextInt(90) + 10;
        first.put(String.format("%s%02d%s", "2021", i, "" + rNum), rNum);
    }
    var list = new ArrayList<>(first.entrySet());
    Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
                @Override
                public int compare(Map.Entry<String, Integer> e1,
                                   Map.Entry<String, Integer> e2) {
                    int greaterByValue = e1.getValue().compareTo(e2.getValue());
                    return greaterByValue != 0 ? greaterByValue :
                            e1.getKey().compareTo(e2.getKey());
                }
            }
    );
    for (var element : list) {
        System.out.printf("[%s,%d]%n", element.getKey(), element.getValue());
    }
}
```

Outline

Day.java

```
1  public enum Day {
2      SUNDAY(1),
3      MONDAY(2),
4      TUESDAY(3),
5      WEDNESDAY(4),
6      THURSDAY(5),
7      FRIDAY(6),
8      SATURDAY(7);
9      private final int value;
10     private Day(int value) {
11         this.value = value;
12     }
13     public int getValue() {
14         return this.value;
15     }
16     // overrides the default definition of toString() for Enumeration
17     @Override
18     public String toString() {
19         switch(this) {
20             case FRIDAY:
21                 return "Friday: "    + value;
22             case MONDAY:
23                 return "Monday: "    + value;
24             case SATURDAY:
25                 return "Saturday: " + value;
26             case SUNDAY:
27                 return "Sunday: "    + value;
28             case THURSDAY:
29                 return "Thursday: " + value;
30             case TUESDAY:
31                 return "Tuesday: "   + value;
32             case WEDNESDAY:
33                 return "Wednesday: "+ value;
34             default:
35                 return null;
36         }
37     }
38 }
```

In order to retrieve the value of each constant of the `enum`, you can define a `public` method inside the `enum`

An enum can override the `toString()` method, just like any other Java class

Outline

car.java

You can define **abstract** methods inside an **enum** in Java. Each constant of the enum implements each **abstract** method independently.

```
 1 public enum Car {
 2     AUDI {
 3         @Override
 4         public int getPrice() {
 5             return 25000;
 6         }
 7     },
 8     MERCEDES {
 9         @Override
10         public int getPrice() {
11             return 30000;
12         }
13     },
14     BMW {
15         @Override
16         public int getPrice() {
17             return 20000;
18         }
19     };
20
21     public abstract int getPrice();
22 }
```

Outline

EnumMapExample.java

An **EnumMap** is a specialized **Map** implementation. All of the keys in an **EnumMap** **must come from a single** **enum** type that is specified, explicitly or implicitly, when the map is created. **EnumMaps** are **represented internally as arrays**. Also, **EnumMaps** are maintained in the **natural order of their keys**

```
1  public static void main(String[] args) {
2      // Create an EnumMap that contains all constants of the Car enum.
3      EnumMap<Car, Integer> cars = new EnumMap<Car, Integer>(Car.class);
4
5      // Put some values in the EnumMap.
6      cars.put(Car.BMW, Car.BMW.getPrice());
7      cars.put(Car.AUDI, Car.AUDI.getPrice());
8      cars.put(Car.MERCEDES, Car.MERCEDES.getPrice());
9
10     // Print the values of an EnumMap.
11     for(Car c: cars.keySet())
12         System.out.println(c.name());
13
14     System.out.println(cars.size());
15
16     // Remove a Day object.
17     cars.remove(Car.BMW);
18     System.out.println("After removing Car.BMW, size: " + cars.size());
19
20     // Insert a Day object.
21     cars.put(Car.valueOf("BMW"), Car.BMW.getPrice());
22     System.out.println("Size is now: " + cars.size());
23 }
```

```
run:
AUDI
MERCEDES
BMW
3
After removing Car.BMW, size: 2
Size is now: 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Outline

EnumSetExample.java

```
1  public static void main(String[] args) {
2      // Create an EnumSet that contains all days of the week.
3      EnumSet<Day> week = EnumSet.allOf(Day.class);
4
5      // Print the values of an EnumSet.
6      for(Day d: week)
7          System.out.println(d.name());
8
9      System.out.println(week.size());
10
11     // Remove a Day object.
12     week.remove(Day.FRIDAY);
13     System.out.println("After removing Day.FRIDAY, size: " + week.size());
14
15     // Insert a Day object.
16     week.add(Day.valueOf("FRIDAY"));
17     System.out.println("Size is now: " + week.size());
18 }
```

```
run:
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
7
After removing Day.FRIDAY, size: 6
Size is now: 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

An **EnumSet** is a specialized **Set** implementation. All of the elements in an **EnumSet** **must come from a single** **enum** type that is specified, explicitly or implicitly, when the set is created. **EnumSet** are **represented internally as bit vectors.** Also, an **iterator traverses the elements in their natural order**.

13b.11 Properties Class

- Properties
  - Persistent Hashtable
    - Can be written to output stream
    - Can be read from input stream
  - Provides methods setProperty and getProperty
    - Store/obtain key-value pairs of Strings
- Preferences API
  - Replace Properties
  - More robust mechanism

Outline

PropertiesTest.java

(1 of 5)

Line 16

Lines 19-20

```
1  // Fig. 19.21: PropertiesTest.java
2  // Demonstrates class Properties of the java.util package.
3  import java.io.FileOutputStream;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.util.Properties;
7  import java.util.Set;
8
9  public class PropertiesTest
10 {
11    private Properties table;
12
13    // set up GUI to test Properties table
14    public PropertiesTest()
15    {
16       table = new Properties(); // create Properties table
17
18       // set properties
19       table.setProperty( "color", "blue" );
20       table.setProperty( "width", "200" );
21
22       System.out.println( "After setting properties" );
23       listProperties(); // display property values
24
25       // replace property value
26       table.setProperty( "color", "red" );
27
```

Create empty `Properties`

`Properties` method `setProperty` stores value for the specified key

Outline

```
28      System.out.println( "After replacing properties" );
29      listProperties(); // display property values
30
31      saveProperties(); // save properties
32
33      table.clear(); // empty table
34
35      System.out.println( "After clearing properties" );
36      listProperties(); // display property values
37
38      loadProperties(); // load properties
39
40      // get value of property color
41      Object value = table.getProperty( "color" );
42
43      // check if value is in table
44      if ( value != null )
45         System.out.printf( "Property color's value is %s\n", value );
46      else
47         System.out.println( "Property color is not in table" );
48   } // end PropertiesTest constructor
49
```

Use `Properties` method `clear` to empty the hash table

Use `Properties` method `getProperty` to locate the value associated with the specified key

PropertiesTest.java

(2 of 5)

Line 33

```
50    // save properties to a file
51    public void saveProperties()
52    {
53       // save contents of table
54       try
55       {
56          FileOutputStream output = new FileOutputStream( "props.dat" );
57          table.store( output, "Sample Properties" ); // save properties
58          output.close();
59          System.out.println( "After saving
60          listProperties();
61       } // end try
62       catch ( IOException ioException )
63       {
64          ioException.printStackTrace();
65       } // end catch
66    } // end method saveProperties
67
```

`Properties` method `store` saves `Properties` contents to `FileOutputStream`

Outline

PropertiesTest.java

(3 of 5)

Line 57

Outline

PropertiesTest.java

(of 5)

Line 75

Line 89

Line 95

```
68    // load properties from a file
69    public void loadProperties()
70    {
71       // load contents of table
72       try
73       {
74          FileInputStream input = new FileInputStream( "props.dat" );
75          table.load( input ); // load properties
76          input.close();
77          System.out.println( "After loading proper
78          listProperties(); // display property val
79       } // end try
80       catch ( IOException ioException )
81       {
82          ioException.printStackTrace();
83       } // end catch
84    } // end method loadProperties
85
86    // output property values
87    public void listProperties()
88    {
89       Set< Object > keys = table.keySet(); // get
90
91       // output name/value pairs
92       for ( Object key : keys )
93       {
94          System.out.printf(
95             "%s\t%s\n", key, table.getProperty( ( String ) key ) );
96       } // end for
97
```

`Properties` method `load` restores `Properties` contents from `FileInputStream`

Use `Properties` method `keySet` to obtain a `Set` of the property names

Obtain the value of a property by passing a key to method `getProperty`

```
98       System.out.println();
99    } // end method listProperties
100
101   public static void main( String args[] )
102   {
103      new PropertiesTest();
104   } // end main
105} // end class PropertiesTest
```

```
After setting properties
color   blue
width   200

After replacing properties
color   red
width   200

After saving properties
color   red
width   200

After clearing properties

After loading properties
color   red
width   200

Property color's value is red
```

Outline

PropertiesTest.java

(5 of 5)

Program output

Задачи

Задача 1.

Обяснете приликите и разликите между `Set` и `Map` в `Collections` библиотеката на `Java`

Може ли да отпечатаме всички елементи на колекция без да се използва `Iterator`? Ако да, обяснете как.

`class Stack` в `Collections` библиотеката на `Java` е производен на `class Vector`. Обяснете какви проблеми произтичат от този модел за реализиране на `class Stack`.

Задачи

Задача 2.

Обяснете накратко какво правят следните методи на `class HashMap`:

a) `put`

b) `get`

c) `isEmpty`

d) `containsKey`

e) `keySet`

Задачи

Задача 3.

Определете кое от следните твърдения е вярно и кое е грешно.

- **Елементите на `Collection` трябва да са сортирани във възходящ ред преди да се изпълни `binarySearch`.**
- **Методът `first` връща първият елемент на `TreeSet`.**
- **Всеки `List` създаден с метода `asList` на клас `Arrays` е с динамично изменяема дължина.**
- **class `Arrays` има `static` метод `sort` за сортиране на масиви**

Задачи

Задача 3.

Определете кое от следните твърдения е вярно и кое е грешно.

- **Елементите на `Collection` трябва да са сортирани във възходящ ред преди да се изпълни `binarySearch`.**
- **Методът `first` връща първият елемент на `TreeSet`.**
- **Всеки `List` създаден с метода `asList` на клас `Arrays` е с динамично изменяема дължина.**
- **class `Arrays` има `static` метод `sort` за сортиране на масиви**

Задачи

Задача 4.

Променете програмата на `Fig. 13b.20` да преброява броя на срещане на всяка `буква`, вместо на всяка дума. Например, низът `"HELLO THERE"` се състои от 2 букви `H`, три `E`, две `L`, една `O`, една `T` и едно `R`. Изведете крайните резултати

Задача 5.

При извеждане на крайните резултати в `Fig. 13b.17` (`PriorityQueueTest`) се вижда, че `PriorityQueue` сортира `Double` елементите във възходящ ред. Пренапишете `Fig. 13b.17` така че да сортира `Double` елементите в низходящ ред.

Задачи

Задача 6.

Напишете програма, която използва `StringTokenizer` за разбиване на думи на изречение, въведено от потребителя и поставя всяка дума в `TreeSet`. Изведете на стандартен изход елементите на `TreeSet`, при което те ще се изведат във възходящ ред