Programski jezik Java

Interno gradivo za predmet

Algoritmi in programski jeziki (4. letnik)

ArrayList

(neprečiščeno besedilo)

bostjan.vouk@tsc.si

ArrayList

- Java class ArrayList(java.util.ArrayList) je hiter in za uporabo preprost razred, ki predstavlja enodimenzijsko tabelo
- ArrayList ni sinhroniziran - uporaba v več kot eni niti lahko povzroči težave.

bostjan.vouk@tsc.si

ArrayList

- Razred ArrayList vsebuje metode, ki omogočajo osnovne operacije nad tabelami:
- **add(Object o)**
  - puts reference to object into ArrayList
- **get(int index)**
  - retrieves object reference from ArrayList index position
- **size()**
  - returns ArrayList size
- **remove(int index)**
  - removes the element at the specified position in this list. Shifts any subsequent elements to the left and returns the element that was removed from the list.
- **indexOf(Object o)**
  - finds the index in this list of the first occurrence of the specified element
- **clear()**
  - removes all of the elements

bostjan.vouk@tsc.si

Primer

- Napiši program, ki uporabnika vpraša po imenu in mu predlaga mesto počitnikovanja.
  - ArrayList "myarr" je napolnjena z letovišči(metoda add).
  - Ko uporabnik vnese svoje ime izračunamo ostanek pri deljenju z dolžino uporabnikovega imena in dolžino myarr.
  - Rezultat deljenja je vrednost med 0 in myarr.size()-1
    - Rezultat ustreza indeksu tabele

Primer

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class Ex01 {
   public static void main(String[] args) throws IOException {
      BufferedReader userInput = new BufferedReader
         (new InputStreamReader(System.in));
```


Primer

```
ArrayList<String> myArr = new ArrayList<String>();
    myArr.add("Italian Riviera");
    myArr.add("Jersey Shore");
    myArr.add("Puerto Rico");
    myArr.add("Los Cabos Corridor");
    myArr.add("Lubmin");
    myArr.add("Coney Island");
    myArr.add("Karlovy Vary");
    myArr.add("Bourbon-l'Archambault");
    myArr.add("Walt Disney World Resort");
    myArr.add("Barbados");

    System.out.println("Stupid Vacation Resort Adviser");
    System.out.println("Enter your name:");
    String name = userInput.readLine();
```

Primer

```
int nameLength = name.length();
    if (nameLength == 0)
    {
        System.out.println("empty name entered");
        return;
    }

    int vacationIndex = nameLength % myArr.size();

    System.out.println("\nYour name is "+name+", its length is " +
                nameLength + " characters,\n" +
                "that's why we suggest you to go to "
                + myArr.get(vacationIndex));
  }
}
```

bostjan.vouk@tsc.si

Pozor

- ArrayList stores only object references. That's why, it's impossible to use primitive data types **like double** or **int**.
  - *Use wrapper class (like Integer or Double) instead.*
- For multi-theaded (synchronized) array class use Vector (java.lang.Vector)

bostjan.vouk@tsc.si

ArrayList<E>

- java.util.ArrayList<E> allows for expandable arrays, and is basically the same as the older the Collections Vector class. An ArrayList has these characteristics:
- An ArrayList automatically expands as data is added.
- Access to any element of an ArrayList is O(1). Insertions and deletions are O(n).
- An ArrayList has methods for inserting, deleting, and searching.
- An ArrayList can be traversed using a foreach loop, iterators, or indexes.

bostjan.vouk@tsc.si

Arrays or ArrayList?

- Programmers are frequently faced with the choice of using a simple array or an ArrayList.
- If the data has a known number of elements or small fixed size upper bound, or where efficiency in using primitive types is important, arrays are often the best choice.
- However, many data storage problems are not that simple, and ArrayList (or one of the other Collections classes) might be the right choice.

bostjan.vouk@tsc.si

Automatic expansion.

- Use ArrayList when there will be a large variation in the amount of data that you would put into an array.
- Arrays should be used only when there is a constant amount of data.
- For example, storing information about the days of the week should use an array because the number of days in a week is constant.
- Use an array list for your email contact list because there is no upper bound on the number of contacts.

bostjan.vouk@tsc.si

Objects only.

- A possible disadvantage of ArrayList is that it holds only object types and not primitive types (eg, int).
- To use a primitive type in an ArrayList, put it inside an object or use of the wrapper classes (eg, Integer, Double, Character, ...).
- The wrapper classes are immutable, so if you use, eg, Integer, you will not be able to change the integer value. In this case it may be more useful to define your own mutable class.

bostjan.vouk@tsc.si

Implementation.

- ArrayLists are implemented with an underlying array, and when that array is full and an additional element is added, a new, larger, array is allocated and the elements are copied from the old to the new.
- Because it takes time to create a bigger array and copy the elements from the old array to the new array, it is a slightly faster to create an ArrayList with a size that it will commonly be when full.
- Of course, if you knew the final size, you could simply use an array.
- However, for non-critical sections of code programmers typically don't specify an initial size.

bostjan.vouk@tsc.si

Common ArrayList methods and constructors

- Here are some of the most useful ArrayList methods. Assume these declarations.
- Note that E is the notation for the type of an element in a collection.
- Sun recommends using single upper case letters for generic types.
- int i;
- ArrayList<E> a;
- E e;
- Iterator<E> iter;
- ListIterator<E> liter;
- E[] earray;
- Object[] oarray;

bostjan.vouk@tsc.si

Result	Method	Description
Constructors		
a =	new ArrayList<E>()	Creates ArrayList with initial default capacity 10.
a =	new ArrayList<E>(cap)	Creates ArrayList with initial int capacity *cap*.
a =	new ArrayList<E>(coll<E>)	Creates ArrayList from the *Collection coll*.
Adding elements		
	a.add(e)	adds e to end of ArrayList a
	a.add(i, e)	Inserts e at index i, shifting elements up as necessary.
Replacing an element		
	a.set(i,e)	Sets the element at index i to e.
Getting the elements		
e =	a.get(i)	Returns the object at index i.
oarray =	a.toArray()	Returns values in array of objects.
earray =	a.toArray(E[])	The array parameter should be of the E class. Returns values in that array (or a larger array is allocated if necessary).
Iterators		
iter =	a.iterator()	Returns an Iterator for forward traversal.
liter =	a.listIterator(i)	Returns a ListIterator for forward / backward / modifying traversal, starting at index i. Start from end with a.listIterator(a.size())
liter =	a.listIterator()	Returns a ListIterator for forward / backward / modifying traversal.
Searching		
b =	a.contains(e)	Returns true if ArrayList a contains e
i =	a.indexOf(e)	Returns index of first occurrence of e, or -1 if not there.
i =	a.lastIndexOf(e)	Returns index of last occurrence of e, or -1 if not there.
Removing elements		
	a.clear()	removes all elements from ArrayList a
	a.remove(i)	Removes the element at position i.
	a.removeRange(i, j)	Removes the elements from positions i thru j.
Other		
i =	a.size()	Returns the number of elements in ArrayList a.

Adding elements to the end of an ArrayList, getting them by index

- ArrayList<E> a = new ArrayList<E>(); // Default size.
- E s; // Declare s to be an object type E.
- . . .
- a.add(s); // Adds s to the end of the ArrayList a
- . . .
- s = a.get(i); // Assigns ith element from a to s.

To get successive elements from an ArrayList - Four ways

- Use either a for loop with an integer index to get all the elements from an ArrayList, or go over all elements in a ArrayList using an Iterator (forward) or ListIterator (forward / backward).
  1. foreach loop.
  2. for loop with index.
  3. Iterator<E>
  4. ListIterator<E>

foreach loop.

- This is fast and works for all kinds of lists, but is not entirely flexible (only sequential forward with no deletions, additions, or multiple references).
- This should be your first choice in programming. Works efficiently with both ArrayList and LinkedList.
- ArrayList<String> a = new ArrayList<String>();

```
. . .
for (String s : a) {
   System.out.println(s);
}
```

for loop with index

- This is fast, but should not be used with a LinkedList. It does allow orders other than sequentially forward by one.
- for (int i = 0; i < a.size(); i++) {
  System.out.println(a.get(i));
  }

Iterator<E>

- Allows simple forward traversal. Can be used for the largest number of other kinds of data structures.
- This example uses an Iterator to print all elements (Strings) in an ArrayList. It uses hasNext(), which returns true if there are more elements, and next(), which returns the next element.
- Works with both ArrayList and LinkedList.
- for (Iterator<String> iter = a.iterator(); iter.hasNext();) {
  System.out.println(iter.next());
  }

ListIterator<E>

- Allows traversal of the ArrayList, but it is more general than a simple Iterator, allowing inserts and deletes (although both are very slow for an ArrayList).
- It also allows bidirectional traversal. Works efficiently with both ArrayList and LinkedList.

Sorting

- If the data in your ArrayList has a natural sorting order (ie, implements Comparable, as do String, Integer, ...), you can simply call the static Collections.sort() method.
- This is a stable, guaranteed **n log n** sort.
- ***Collections.sort(yourArrayList);***

bostjan.vouk@tsc.si

Sorting

- If you want to choose a different sort criterion or your data doesn't implement xyz, you will have to define a Comparator and pass that to the sort() method.
- ***Collections.sort(yourArrayList, yourComparator);***

bostjan.vouk@tsc.si