Theory: Dynamic array in Java

33 minutes 0 / 5 problems solved

In Java, **dynamic arrays** are represented by the `ArrayList` class. To understand how dynamic arrays work under the hood, we will implement a simplified version of its class.

Our `DynamicArray` class must support the following operations:

- `void add(E value)` – add an element to the end of the list;
- `void add(int index, E element)` – add an element at the given index;
- `E get(int index)` – get the value at the given index;
- `E remove(int index)` – remove the element at the given index;
- `int size()` – return the size of the array.

Current topic:

Dynamic array in Java

Topic depends on:

✗ Dynamic array

✓ Iterating over arrays

✗ Throwing exceptions

✗ Generics and Object

§1. Implementation the main functionality

Let's get started. The class will be generic without any inheritance:

```
public class DynamicArray<E>
```

Private fields and constructors look like this:

```
private Object[] arr;
private int size;

private final int DEFAULT_CAPACITY = 10;
private final double SCALING_FACTOR = 1.5;

public DynamicArray() {
    this.arr = new Object[DEFAULT_CAPACITY];
    this.size = 0;

}



public DynamicArray(int initialCapacity) {


 this.arr = new Object[initialCapacity > 0 ? initialCapacity : DEFAULT_CAPACITY
];

    this.size = 0;

}
```

Table of contents:

↑ Dynamic array in Java

§1. Implementing the main functionality

§2. Conclusion

Feedback & Comments

Array `arr` contains the dynamic array itself. The variable `size` stores the number of elements in the array. Constant variables `DEFAULT_CAPACITY` and `SCALING_FACTOR` are used for the default capacity of the array and the capacity increase coefficient on reaching the capacity limit.

> It is not necessary for `SCALING_FACTOR` to be constant. You can initialize it in the class constructor.

In addition to these main functions, we need another private one. Its purpose is to check during the addition of an element whether there is enough space in the array for it. If not, the function will allocate more memory for the array.

```
private void tryIncrease() {
    if (arr.length == size)
        arr = Arrays.copyOf(arr, (int)(arr.length * SCALING_FACTOR));
}
```

Now it's time to move on to the implementation of main functions.

Addition:

```
public void add(E value) {
    tryIncrease();
    arr[size++] = value;
}

public void add(int index, E element) {
    if (index < 0 || index > size)
        throw new IndexOutOfBoundsException();
    tryIncrease();

    System.arraycopy(arr, index, arr, index + 1, size - index);

    arr[index] = element;

    size++;

}
```

First, we check if there is enough space for the new element, and then add it to its appropriate place (either to the end or to the place defined by index).

Removal:

```
public E remove(int index) {
    if (index < 0 || index >= size)
        throw new IndexOutOfBoundsException();
    E oldValue = (E) arr[index];
    int moveCount = size - index - 1;
    if (moveCount > 0)
        System.arraycopy(arr, index + 1, arr, index, moveCount);
    arr[--size] = null;
    return oldValue;

}
```

Since the function must return the deleted element, we save it before deleting and only then remove it while shifting all other elements to the right.

Get element at the index:

```
public E get(int index) {
    if (index < 0 || index >= size)
        throw new IndexOutOfBoundsException();
    return (E) arr[index];
}
```

This is a fairly simple function: just check if the given index is within the appropriate bounds, and then return the element at that place.

§2. Conclusion

We discussed the main functionality of the class. You can enrich it with extra functions like `isEmpty`, `contains`, `indexOf`, `clear`, etc. They are not that difficult compared to what we already saw, and you will find them useful for solving practice tasks.

Report a typo

14 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion