Contents

1 Class Index 1
  1.1 Class List ......................................................... 1

2 Class Documentation 1
  2.1 DynArray< T > Class Reference ............................... 1
  2.1.1 Detailed Description ........................................... 1
  2.1.2 Constructor & Destructor Documentation .................... 2
  2.1.3 Member Function Documentation .............................. 3

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

 DynArray< T > 1

2 Class Documentation

2.1 DynArray< T > Class Reference

Inherits RandomAccess, and Iterable< T >.

Public Member Functions

- DynArray (boolean allowNulls)
- DynArray ()
- DynArray (int ensureCapacity, boolean allow_nulls)
- DynArray (DynArray< T > other)
- void add (T ele)
- T remove (int atIndex)
- T get (int index)
- void set (int index, T object)
- int size ()
- String toString ()
- boolean equals (Object other)

2.1.1 Detailed Description

DynArrays are dynamically re-sizable arrays that may contain any kind of first-class Objects. DynArray objects differ from linked-lists in that they are optimized for array-style access, i.e., accessing elements by indices (ints \( \geq 0 \)). As such, DynArray objects must declare that they implement the RandomAccess marker interface.

Some additional considerations: At least four public constructors are required for this implementation:
1. **DynArray()** (the default constructor) which creates a dynamic array whose internal array is a default size and that allows clients to store null values.

2. **DynArray( boolean nullOk )** a minimal constructor that allows the client to specify whether or not null objects are permitted through the use of the nullOk flag.

3. **DynArray( int ensureCapacity, boolean nullOk )** This constructor creates a DynArray object that is at least large enough to ensureCapacity; note, the nullOk parameter is used to delegate calls to **DynArray( boolean nullOk )**, described above.

4. **DynArray( DynArray other )** This is a standard copy-constructor that creates a shallow copy of the underlying storage; it must also preserve all relevant properties.

Note that attempts to store null values in DynArray objects that do not allow such values must result in a NullPointerException being raised. Note also that calling any of the methods that require indexing may result in unchecked ArrayIndexOutOfBoundsException exceptions being thrown. (This does not mean that we cannot create an array whose values are null no matter what the value of nulls_allowed. In fact, this cannot be avoided because of how Java creates arrays of Objects. Rather, a null cannot be added to a DynArray that prohibits nulls by passing the appropriate values to these constructors.)

In addition, your implementation should override the **toString()** and the **equals** methods, but need not override the **hashCode()** method.

Prohibited Constructions/Classes/Utilities, etc

Obviously, you should not use any of Java's collection classes to implement this class. In other words, you cannot use any collection class from the java.util.* library, except for the **Iterable** interface that you will implement.

Author

UMD CS Department.

Parameters

| <T> | any subclass of Object |

2.1.2 Constructor & Destructor Documentation

2.1.2.1 **DynArray( boolean allowNulls )**

Creates a DynArray object that may allow or disallow its elements to be null values, depending upon the value provided for the **allowNulls** parameter. Note, the internal array created by this constructor is a small power of two that is provided by the implementor.

Parameters

| allowNulls | set to true to allow null objects. |

2.1.2.2 **DynArray( )**

Default ctor: creates a DynArray object that permits null values; this object's internal array is a small power of two which is determined by the implementation.

2.1.2.3 **DynArray( int ensureCapacity, boolean allow_nulls )**

Full service constructor: creates a DynArray that permits null objects and whose array is sized by the max( ensureCapacity, quanta ).
Parameters

<table>
<thead>
<tr>
<th>ensureCapacity</th>
<th>if provided, then the internal array is at least this size</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_nulls</td>
<td>true if null objects are allowed.</td>
</tr>
</tbody>
</table>

2.1.2.4 DynArray (DynArray &lt; T &gt; other)

Copy constructor for Dynamic Array class. Note: this need only ensure shallow-copy semantics, but it must preserve all of the properties of the Dynamic Array being copied.

Parameters

| other |

2.1.3 Member Function Documentation

2.1.3.1 void add (T ele)

Adds the ele to the end of the vector. Note, this action may require that the internal array be grown. Should this happen, the new internal array has a length determined by the current capacity plus some quanta, which is a small power of two that is a private fixed property of the implementation. Also note that ele may not be null, unless allow_nulls was set to true through a constructor.

Parameters

| ele | any subclass of Object |

2.1.3.2 boolean equals (Object other)

Two Dynamic Arrays are equal iff they have the same objects in the same locations.

2.1.3.3 T get (int index)

Returns the object located at index. Note, this method may throw an ArrayIndexOutOfBoundsException exception.

Parameters

| index | any integer &ge; 0, but within bounds. |

Returns

the object located at the index

2.1.3.4 T remove (int atIndex)

Removes and returns the object found at atIndex. Note: as a result of calling this method, the effective index of this object's internal array is adjusted.

Note: attempts to remove from an empty vector, or attempts to remove from an invalid location (i.e., a bad index) results in an ArrayIndexOutOfBoundsException exception being thrown.

Parameters
atIndex | any integer $\geq 0$, but within bounds.

Returns

the object located at Index (which has been removed)

2.1.3.5 void set ( int index, T object )

Replace the object found at index with object.

- Note: may throw an ArrayIndexOutOfBoundsException exception.
- May throw an IllegalArgumentException if the result of executing this method would leave the underlying structure in an inconsistent state, e.g., creating a situation where null references appeared in an instance of the DynArray class that prohibits null object references.

Parameters

| object | |
| index | any integer $\geq 0$, but within bounds. |

2.1.6 int size ( )

Returns the number of indexable objects available in this vector.

Returns

an integer greater than or equal to 0

2.1.7 String toString ( )

Delegates to Arrays.toString( Object ... ) method.