Contents

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

2 Class Documentation ....................... 1
  2.1 DynArray<T> Class Template Reference . ........................................... 1
    2.1.1 Detailed Description ...................................................... 1
    2.1.2 Constructor & Destructor Documentation .............................. 3
    2.1.3 Member Function Documentation ........................................ 4

Index ............................................. 7

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 Template 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

Special Instructions

Copy your existing code into this lab and submit your changes/etc through this Lab only.
• Do not attempt to submit last week’s lab in place of this one.

• Submit your changes only by including them in this project/lab and submitting them from within this project/lab.

Engineering Change Order(s)

ECO 2 (supersedes ECO 1)

• The set(index, v) method is revised to restrict the index to an integer less than the current size() of the DynArray. Attempts to set any value at an index >= size() must throw an ArrayIndexOutOfBoundsException. Attempts to set a null in a DynArray that disallows nulls throws an IllegalArgumentException.

• All overrides, equals and toString should only operate on objects within the DynArray—i.e., objects whose indices are at least 0 and less than size.

• Sparse arrays, e.g., [null, null, ..., Object, null, ...] are still available, but the responsibility is shifted to the client to create and maintain these objects in their own implementation.

Invariant(s) for the DynArray class

Subsequent to the changes outlined in ECO 2, the class invariant for the DynArray class, as specified, is strengthened to read: No null ever appears in a DynArray that is created to disallow nulls.

Notes on required exceptions

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. Additional exceptions may be raised by attempting to use the set() method in such a manner as to either insert a value past this current object’s capacity or to insert a null where the object disallows nulls. Please see the documentation for the set() and get() methods for a more thorough treatment.

Notes on equals testing

This revision requires that the equals override not throw exceptions when comparing DynArrays that may contain nulls, such as the case where the client intends that the structures allow nulls. In addition, your implementation should override the toString() and the equals methods, but need not override the hashCode() method.

Previous Definitions remain in effect

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 >= 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.
2.1 DynArray\(<\ T\ >\) Class Template Reference

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.

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

| ensureCapacity | if provided, then the internal array is at least this size |
| allow nulls | true if null objects are allowed. |

2.1.2.4 `DynArray(\ DynArray\<\ T\ >\ 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 several exceptions depending upon several conditions:

- A `ArrayIndexOutOfBoundsException` exception is thrown if the index is greater than or equal to the current capacity;
- An `IllegalStateException` is thrown if the object located at the index is `null` but `nulls` are not allowed for this object. (This exception is retained from ECO 1.)

Parameters

| index | any integer ≥ 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 (size) 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 greater than or equal to 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.

- Should throw an ArrayIndexOutOfBoundsException exception when the index specified is beyond the capacity of the underlying storage.
- Should throw an IllegalArgumentException if the object parameter is null and null is not allowed by this object

Parameters

<table>
<thead>
<tr>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
</tr>
</tbody>
</table>

2.1.3.6 int size ( )

Returns the number of indexable objects stored in this vector. (This may not be the same as the capacity.)

Returns

an integer greater than or equal to 0

2.1.3.7 String toString ( )

Pretty prints the contents of this vector taking into account its current size.
**Index**

<table>
<thead>
<tr>
<th>Add</th>
<th>student_classes::DynArray</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynArray</td>
<td>student_classes::DynArray</td>
<td>3</td>
</tr>
<tr>
<td>DynArray&lt; T &gt;</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Equals</td>
<td>student_classes::DynArray</td>
<td>4</td>
</tr>
<tr>
<td>Get</td>
<td>student_classes::DynArray</td>
<td>4</td>
</tr>
<tr>
<td>Remove</td>
<td>student_classes::DynArray</td>
<td>4</td>
</tr>
<tr>
<td>Set</td>
<td>student_classes::DynArray</td>
<td>4</td>
</tr>
<tr>
<td>Size</td>
<td>student_classes::DynArray</td>
<td>5</td>
</tr>
<tr>
<td>student_classes::DynArray</td>
<td>add, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DynArray, 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>equals, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>get, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>remove, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>size, 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>toString, 5</td>
<td></td>
</tr>
<tr>
<td>ToString</td>
<td>student_classes::DynArray</td>
<td>5</td>
</tr>
</tbody>
</table>