```
/*A generic dynamic array implementation
 * @author William Fiset, william.alexandre.fiset@gmail.com */
package com.williamfiset.datastructures.dynamicarray;

@SuppressWarnings("unchecked")
public class DynamicArray<T> implements Iterable<T> {

  private T[] arr;
  private int len = 0; // length user thinks array is
  private int capacity = 0; // Actual array size

  public DynamicArray() {    this(16);  }

  public DynamicArray(int capacity) {
    if (capacity < 0) throw new IllegalArgumentException("Illegal Capacity: " +
capacity);
    this.capacity = capacity;
    arr = (T[]) new Object[capacity];
  }

  public int size() {    return len;  }

  public boolean isEmpty() {    return size() == 0;  }

  public T get(int index) {    return arr[index];  }

  public void set(int index, T elem) {    arr[index] = elem;  }

  public void clear() {
    for (int i = 0; i < len; i++) arr[i] = null;
    len = 0;
  }

  public void add(T elem) {

    // Time to resize!
    if (len + 1 >= capacity) {
      if (capacity == 0) capacity = 1;
```

```
    else capacity *= 2; // double the size
    T[] new_arr = (T[]) new Object[capacity];
    for (int i = 0; i < len; i++) new_arr[i] = arr[i];
    arr = new_arr; // arr has extra nulls padded
  }

  arr[len++] = elem;
}

// Removes an element at the specified index in this array.
public T removeAt(int rm_index) {
  if (rm_index >= len || rm_index < 0) throw new IndexOutOfBoundsException();
  T data = arr[rm_index];
  T[] new_arr = (T[]) new Object[len - 1];
  for (int i = 0, j = 0; i < len; i++, j++)
    if (i == rm_index) j--; // Skip over rm_index by fixing j temporarily
    else new_arr[j] = arr[i];
  arr = new_arr;
  capacity = --len;
  return data;
}

public boolean remove(Object obj) {
  int index = indexOf(obj);
  if (index == -1) return false;
  removeAt(index);
  return true;
}

public int indexOf(Object obj) {
  for (int i = 0; i < len; i++) {
    if (obj == null) {
      if (arr[i] == null) return i;
    } else {        if (obj.equals(arr[i])) return i;
    }
  }
  return -1;
}
```

```
  public boolean contains(Object obj) {    return indexOf(obj) != -1;
  }

  // Iterator is still fast but not as fast as iterative for loop
  @Override
  public java.util.Iterator<T> iterator() {
    return new java.util.Iterator<T>() {
      int index = 0;

      @Override
      public boolean hasNext() {
        return index < len;
      }

      @Override
      public T next() {       return arr[index++];
      }

      @Override
      public void remove() {       throw new UnsupportedOperationException();
      }
    };
  }

  @Override
  public String toString() {
    if (len == 0) return "[]";
    else {
      StringBuilder sb = new StringBuilder(len).append("[");
      for (int i = 0; i < len - 1; i++) sb.append(arr[i] + ", ");
      return sb.append(arr[len - 1] + "]").toString();
    }
  }
}
```