```
package arraypractice;

import java.util.Arrays;

public class ArrayPractice {
  /* sets every item in A[] to initialValue */
  public static void initialize(int A[], int initialValue) {
    for (int i = 0; i < A.length; i++) {
      A[i] = initialValue;
    }
    return;
  }

  /*
   * returns the average of the items in A
   * Be careful: A[] is an array of int and the method returns
   * double. What do we do to handle this?
   */
  public static double average(int A[]) {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
      sum += A[i];
    }
    return sum / A.length;
  }

  /* returns the number of times that x appears in A[] */
  public static int numOccurrences(int A[], int x) {
    int count = 0;
    for (int i = 0; i < A.length; i++) {
      if (A[i] == x) {
        count++;
      }
    }
    return count;
  }

  /*
   * returns the index of the first occurrence of
   * x in A[] or -1 if x doesn't exist in A[]
   */
  public static int find(int A[], int x) {
    for (int i = 0; i < A.length; i++) {
      if (A[i] == x) {
        return i;
      }
    }
    return -1;
  }

  /*
   * Returns the index of the first occurrence of
   * item within the first n elements of A[] or -1
   * if item is not among the first n elements of A[]
   */
  public static int findN(int A[], int item, int n) {
    for (int i = 0; i < n; i++) {
      if (A[i] == item) {
        return i;
      }
    }
    return -1;
  }
```

```
  /*
   * returns the index of the last occurrence of
   * x in A[] or -1 if x doesn't exist in A[]
   */
  public static int findLast(int A[], int x) {
    for (int i = A.length - 1; i > 0; i--) {
      if (A[i] == x) {
        return i;
      }
    }
    return -1;
  }

  /* returns the largest item found in A */
  public static int largest(int A[]) {
    int large_num = -1;
    for (int i = 0; i < A.length; i++) {
      if (A[i] >= large_num) {
        large_num = A[i];
      }
    }
    return large_num;
  }

  /* returns the index of the largest item found in A */
  public static int indexOfLargest(int A[]) {
    int large_num_index = -1;
    int large_num = -1;
    for (int i = 0; i < A.length; i++) {
      if (A[i] >= large_num) {
        large_num = A[i];
        large_num_index = i;
      }
    }
    return large_num_index;
  }

  /*
   * returns the index of the largest odd number
   * in A[] or -1 if A[] contains no odd numbers
   */
  public static int indexOfLargestOdd(int A[]) {
    int large_num_index = -1;
    int large_num = -1;
    for (int i = 0; i < A.length; i++) {
      if (A[i] >= large_num && A[i] % 2 == 1) {
        large_num = A[i];
        large_num_index = i;
      }
    }
    return large_num_index;
  }

  /* inserts n into A[] at A[index] shifting all */
  /* the previous items one place to the right. For example */
  /* if A is */
  /* |---+---+---+---+---+---+---+---+---+---| */
  /* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | */
  /* |---+---+---+---+---+---+---+---+---+---| */
  /* | 5 | 7 | 6 | 9 | 4 | 3 | 0 | 0 | 0 | 0 | */
  /* |---+---+---+---+---+---+---+---+---+---| */

  /* and we call insert(A, 15, 1), A then becomes */

  /* |---+----+---+---+---+---+---+---+---+---| */
```

```
  /* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | */
  /* |---+---+---+---+---+---+---+---+---+---| */
  /* | 5 | 15 | 7 | 6 | 9 | 4 | 3 | 0 | 0 | 0 | */
  /* |---+---+---+---+---+---+---+---+---+---| */
  /* the element in A[] that's in the right-most */
  /* position is removed. */
  /*                                              */
  /* if index < 0 or index >= A.length-1, the method */
  /* does nothing */

  public static int[] insert(int A[], int n, int index) {
    for (int i = 0; i < A.length - 1; i++) {
      if (i == index) {
        for (int j = A.length - 1; j >= index; j--) {
          A[j] = A[j - 1];
        }
        A[index] = n;
      }
    }
    return A;
  }

  /*
   * returns a new array consisting of all of the
   * elements of A[]
   */
  public static int[] copy(int A[]) {
    int[] B = new int[A.length];
    for (int i = 0; i < B.length; i++) {
      B[i] = A[i];
    }
    return B;
  }

  /*
   * Returns a new array consisting of all of the
   * first n elements of A[]. If n>A.length, returns a
   * new array of size n, with the first A.length elements
   * exactly the same as A, and the remaining n-A.length elements
   * set to 0. If n<=0, returns null.
   */
  public static int[] copyN(int A[], int n) {
    if (n > 0) {
      int[] B = new int[n];
      for (int i = 0; i < A.length && i < B.length; i++) {
        System.out.println(A[i] + " " + i);
        B[i] = A[i];
      }
      if (B.length > A.length) {
        for (int i = A.length; i < B.length; i++) {
          B[i] = 0;
        }
      }
      return B;
    }

    return null;
  }

  /*
   * returns a new array consisting of all of the
   * elements of A[] followed by all of the
   * elements of B[]. For example, if
   * A[] is: {10,20,30} and
   * B[] is: {5, 9, 38}, the method returns the
```

```
     * array : {10,20,30,5,9,38}
     */
    public static int[] copyAll(int A[], int B[]) {
      int[] C = new int[A.length + B.length];
      for (int i = 0; i < A.length; i++) {
        C[i] = A[i];
      }
      for (int i = 0; i < B.length; i++) {
        C[A.length + i] = B[i];
      }
      return C;
    }

    /*
     * reverses the order of the elements in A[].
     * For example, if A[] is:
     * {10,20,30,40,50}, after the method, A[] would
     * be {50,40,30,20,10}
     */
    public static void reverse(int A[]) {
      int temp = 0;
      for (int i = 0; i < A.length / 2; i++) {
        temp = A[i];
        A[i] = A[A.length - 1 - i];
        A[A.length - 1 - i] = temp;
      }
      return;
    }

    /*
     * Extra credit:
     *
     * Returns a new array consisting of all of the
     * elements of A, but with no duplicates. For example,
     * if A[] is {10,20,5,32,5,10,9,32,8}, the method returns
     * the array {10,20,5,32,9,8}
     */
    public static int[] uniques(int A[]) {
      return null;
    }

    public static void main(String[] args) {
      return;
    }
  }
```