```
/////////////////////////////////////////
// Assertions for program correctness. //
//                                     //
// We use Binary Search as an example. //
//                                     //
// Martin Escardo, 3rd December 2019.  //
/////////////////////////////////////////

import java.util.Random;

class BinarySearch {

  // There are three versions of binary search. Choose one here:

  static int binarySearch (int [] a, int x) {

    return binarySearch3(a,x);

  }

  // Version without assertions and with -1 return code for errors:

  static int binarySearch1 (int [] a, int x) {
    int l = 0;
    int r = a.length-1;

    while (l <= r) {
      int m = (l+r)/2;

      if (a[m] < x)
        l = m+1;
      else
        if (a[m] > x)
          r = m-1;
        else
          return m;
    }

    return -1;
  }
```

```
// Variation of binary search.
//
// Instead of returning -1, return the position where the missing
// element would have to be inserted to keep the array sorted.

static int binarySearch2 (int [] a, int x) {
  int l = 0;
  int r = a.length-1;

  while (l <= r) {
    int m = (l+r)/2;

    if (a[m] < x)
      l = m+1;
    else
      if (a[m] > x)
        r = m-1;
      else
        return m;
  }

  return l;
}
```

```
//////////////
// Testing: //
//////////////

static String presenceMessage (int [] a, int x) {

  int i = binarySearch(a,x);

  if (0 <= i && i < a.length && a[i]==x)
     return ("The element " + x + " is present at position " + i);
  else
     if (i == -1)
       return ("The element " + x + " is absent");
     else
       return ("The element " + x + " has to be inserted at position " + i);
}

public static void main(String [] args) {

  // Simple minded test first:
  //          0    1    2    3    4    5    6     7    8    9   10
  int [] a = {2,   5,   6,   9,  12,  40,  50,  100, 230, 432, 564};

  System.out.println("Will first binary search with the array ");

  for (int i = 0; i < a.length; i++)
    System.out.print(a[i] + " (" + i +"), ");

  System.out.println("\n");

  System.out.println(presenceMessage(a,1));
  System.out.println(presenceMessage(a,2));
  System.out.println(presenceMessage(a,3));
  System.out.println(presenceMessage(a,12));
  System.out.println(presenceMessage(a,14));
  System.out.println(presenceMessage(a,40));
  System.out.println(presenceMessage(a,41));
  System.out.println(presenceMessage(a,564));
  System.out.println(presenceMessage(a,565));

  // More systematic testing.
  // Shoud be run with assertions enabled (see below).

  System.out.println("\nMore systematic testing follows:");

  int bigNumber = 10 * 1000 * 1000;
  System.out.println("Creating a sorted array with " + bigNumber + " elements...");

  int [] b = new int [bigNumber];

  Random r = new Random();
  int n = 0;

  // Should be chosen to be relatively small to avoid overflow:
  int step = 10;

  // In case there is overflow if bigNumber is too big:
  assert(bigNumber*step > 0 && bigNumber*step > bigNumber);

  for (int i = 0; i < b.length; i++) {
    n = n + r.nextInt(step);

    // In case there is overflow if bigNumber is too big.
    assert (n >= 0);

    b[i]=n;
  }

  int numberOfTrials = 10 * 1000 * 1000;

  System.out.println("Done.\n");
  System.out.println("Will now run binary search "
                     + numberOfTrials
                     + " times with the above array and random items.");

  int count = 0;

  for (int t = 0; t < numberOfTrials; t++) {
    int x = r.nextInt(bigNumber);
    int i = binarySearch(b,x);

    if (0 <= i && i < a.length && b[i]==x) {
      count++;
    }
  }

  System.out.println("Done.\n");
  System.out.println(count
                     + " random numbers were found in the array out of "
                     + numberOfTrials);
}
```

```
///////////////////////////////////////////////////////////////////////
// We now decorate the above binary search algorithm with assertions //
// for preconditions, postconditions and invariants.                 //
///////////////////////////////////////////////////////////////////////

// We need some code to check the assertions.

// Linear search to test whether an element is in the array:

static boolean isIn (int [] a, int x) {
  for (int y : a)
    if (y == x)
      return true;

  return false;
}

static boolean isSorted (int [] a) {
  for (int i = 0; i < a.length-1; i++)
    if (a[i] > a[i+1])
        return false;

  return true;
}

// Logical implication:

static boolean implies (boolean a, boolean b) {
  return (!a || b);
}

// Condition maintained by the body of the while loop:

static boolean BSInvariant (int [] a, int x , int l , int r) {
  // 0 <= l & r <= a.length-1;

  boolean b = 0 <= l & r <= a.length-1;

  // And if any position i of the array a has x in it,
  // then l <= i <= r.

  for (int i = 0; i < a.length; i++)
    b = b & implies(a[i] == x, l <= i & i <= r);

  return b;
}
```

```
// Condition satisfied by the returned value p:
static boolean BSPostCondition (int [] a, int x , int i) {
  return
       (i == -1 && !isIn(a,x))
    || (i != -1 && a[i] == x);
}
```

```
  // Version 1 of binary search decorated with assertions:

  static int binarySearch3 (int [] a, int x) {

    // Precondition:

    assert(isSorted(a));

    // If the precondition fails, we make no promises.

    int l = 0;
    int r = a.length-1;

    assert (BSInvariant(a,x,l,r));

    while (l <= r) {

      assert (BSInvariant(a,x,l,r));

      int m = (l+r)/2;

      if (a[m] < x)
        l = m+1;
      else
        if (a[m] > x)
          r = m-1;
        else {
          assert (BSPostCondition (a,x,m));
          return m;
        }

      assert (BSInvariant(a,x,l,r));
    }

    assert (l > r);
    assert (BSInvariant(a,x,l,r));

    // Now, l > r and the invariant together imply the post condition
    // with the position -1.

    assert (BSPostCondition (a,x,-1));
    return -1;
  }
}
```

```
// Run the above with enabled assertion checjing using the "-ea"
// javac option:
//
//   $ javac BinarySearch.java
//   $ java -ea BinarySearch.java

//////////////////////////////////////////////////////
// Exercise: decorate binarySearch2 with assertions //
// to reflect the different specification.          //
//////////////////////////////////////////////////////
```