This exercise is taken from Stanford's CS106 midterm exam. Use the unit tests underneath the exercise to check your algorithm. Do not change the tests.

Problem One: Oncogenes **(18 Points)**

//

Recall from the first midterm exam that a DNA sequence can be represented as a string of the letters A, C, T, and G. Let's define a ***gene*** to be a string of letters that appears somewhere within a DNA sequence. For example, **CAT** and **AGCCA** are genes in **TAGCATCAGCCAG**, but neither **TCAT** nor **GAT** are genes in this sequence.

Certain genes (called ***oncogenes***) are known to increase the risk of cancer. For the purposes of this problem, we'll consider an oncogene to be a gene that appears in a higher fraction of cancer cells than of normal healthy cells. For example, if a gene appears in 5% of cancer cells but only 1% of healthy cells, we would consider that gene to be an oncogene. On the other hand, if a gene appears in 10% of cancer cells and 10% of normal cells, we would not consider it an oncogene.

Suppose that you have a gene that you suspect may be an oncogene. To determine whether it is, you gather DNA from healthy cells and cancer cells, then store the DNA in two `ArrayList<String>`s, one holding DNA from healthy cells and one holding DNA from cancer cells.

Write a method

```
private boolean isOncogene(ArrayList<String> healthySequences,
                           ArrayList<String> cancerSequences,
                           String candidate)
```

that accepts as input two `ArrayList<String>`s representing DNA samples from healthy and cancer cells, along with a `String` that you suspect is an oncogene. Your method should then return whether that gene is an oncogene (that is, whether it appears a higher fraction of the time in the cancer DNA sequences than in the healthy DNA sequences).

As an example, if you wanted to check whether **TGC** was an oncogene given the following samples:

Healthy Cell DNA	Cancer Cell DNA
TGCATCC	ATTTGCAGG
AAATTTGGGCCC	TGCAAATTA
ATGCGCTA	AAAGGGCCCTTT
GGGTACGGAG	TGCGATACGTAGGACCA
TTAATTGGG	ACTCATTAGTGC
	AAACGCTAGACACACAAG

Your method would return **`true`**, because the sequence **TGC** appears in only 40% of healthy DNA sequences (2 of 5) but in 66.7% of cancer cell DNA sequences (4 of 6).

However, given the gene **GGG**, your method would return **`false`**, because **GGG** appears 60% of the time in healthy cells (3 of 5) and only 16.7% of the time in cancer cells (1 of 6).

(continued on next page)

You should assume the following:

- All of the strings given as input are made up purely of the characters A, T, C, and G.
- The **`healthySequences`** and **`cancerSequences`** lists are guaranteed to be nonempty, though they do not have to be the same length.
- You should not modify any of the input lists.
- When determining whether a gene is present within a DNA sequence, you do not need to count the number of times that gene appears within the sequence. All that matters is whether or not that gene is present at all.

```
private boolean isOncogene(ArrayList<String> healthySequences,
                           ArrayList<String> cancerSequences,
                           String candidate) {
```

```
//Copyright Wintriss Technical Schools 2016

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import java.util.ArrayList;
import java.util.Arrays;

import org.junit.Ignore;
import org.junit.Test;

public class OncogeneTest
{
	private OncogeneDetector instance = new OncogeneDetector();

	ArrayList<String> healthySequences = new ArrayList<String>(
		Arrays.asList(
			new String[] {"TGCATCC", "AAATTTGGGCCC", "ATGCGCTA", "GGGTACGGAG", "TTAATTGGG"}
		)
	);

	ArrayList<String> cancerSequences = new ArrayList<String>(
		Arrays.asList(
			new String[] {"ATTTGCAGG", "TGCAAATTA", "AAAGGGCCCTTT", "TGCGATACGTAGGACCA",
"ACTCATTAGTGC", "AAACGCTAGACACACAAG", "GGGGGGGGGGGG"}
		)
	);

	ArrayList<String> sequencesWithInvalidElement = new ArrayList<String>(
		Arrays.asList(
			new String[] {"ATTTGGG", "TGCGATTTA", "TGCGGHACCA", "ACTCATTAGTGC"}
		)
	);

	@Test
	public void testOncogeneDetectorWithTGC()
	{
		assertTrue(instance.isOncogene(healthySequences, cancerSequences, "TGC"));
	}

	@Test
	public void testOncogeneDetectorWithGGG()
	{
		assertFalse(instance.isOncogene(healthySequences, cancerSequences, "GGG"));
	}

	@Test
	public void testOncogeneDetectorWithEmptyCandiate()
	{
		assertFalse(instance.isOncogene(healthySequences, cancerSequences, ""));
	}

	// The following are extra credit tests, comment out the @Ignore to run them.
	@Ignore
	@Test(expected = NullPointerException.class)
```

```
public void level0ExtraCreditTestOncogeneDetectorWithNullHealthySequence()
{
	instance.isOncogene(null, cancerSequences, "TGC");
}

@Ignore
@Test(expected = NullPointerException.class)
public void level0ExtraCreditTestOncogeneDetectorWithNullCancerSequence()
{
	instance.isOncogene(healthySequences, null, "TGC");
}

@Ignore
@Test
public void level1ExtraCreditTestOncogeneDetectorWithEmptyHealthySequence()
{
	try {
		instance.isOncogene(new ArrayList<>(), cancerSequences, "TGC");
		fail("IllegalArgumentException not thrown as expected");
	} catch (IllegalArgumentException e) {
		assertEquals("healthySequences must contain at least one element", e.getMessage());
	}
}

@Ignore
@Test
public void level1ExtraCreditTestOncogeneDetectorWithEmptyCancerSequence()
{
	try {
		instance.isOncogene(healthySequences, new ArrayList<>(), "TGC");
		fail("IllegalArgumentException not thrown as expected");
	} catch (IllegalArgumentException e) {
		assertEquals("cancerSequences must contain at least one element", e.getMessage());
	}
}

@Ignore
@Test
public void level2ExtraCreditTestOncogeneDetectorWithInvalidCandidate()
{
	try {
		instance.isOncogene(healthySequences, cancerSequences, "ABC");
		fail("IllegalArgumentException not thrown as expected");
	} catch (IllegalArgumentException e) {
		assertEquals("candidate must contain only the letters A, C, G or T", e.getMessage());
	}
}


@Ignore
@Test(expected = IllegalArgumentException.class)
public void level3ExtraCreditTestOncogeneDetectorWithInvalidHealthySequence()
{
	instance.isOncogene(sequencesWithInvalidElement, cancerSequences, "TGC");
}

@Ignore
```

```
	@Test(expected = IllegalArgumentException.class)
	public void level3ExtraCreditTestOncogeneDetectorWithInvalidCancerSequence()
	{
		instance.isOncogene(healthySequences, sequencesWithInvalidElement, "TGC");
	}
}
```