Question B

The goal of this question is to write a software library that accepts 2 version string as input and returns whether one is greater than, equal, or less than the other. As an example: “1.2” is greater than “1.1”. Please provide all test cases you could think of.

ANSWER

Java software library that accepts two version strings as input and determines whether one is greater than, equal to, or less than the other:

```
public class VersionComparer {
	public static int compareVersions(String version1, String version2) {
		// Split version strings into arrays of string
		String[] parts1 = version1.split("\\.");
		String[] parts2 = version2.split("\\.");

		// Compare version components
		for (int i = 0; i < Math.max(parts1.length, parts2.length); i++) {
			int v1 = (i < parts1.length) ? Integer.parseInt(parts1[i]) : 0;
			int v2 = (i < parts2.length) ? Integer.parseInt(parts2[i]) : 0;
			if (v1 < v2) // version1 is less than version2
				return -1;
			if (v1 > v2) // version1 is greater than version2
				return 1;
		}
		return 0; // version1 is equal to version2
	}
	public static void main(String[] args) {
		String version1 = "1.2";
		String version2 = "1.1";
		int result = compareVersions(version1, version2);
		if (result < 0) {
			System.out.println('"' + version1 + '"' + " is less than " + '"' + version2 + '"' + ".");
		} else if (result > 0) {
			System.out.println('"' + version1 + '"' + " is greater than " + '"' + version2 + '"' + ".");
		} else {
			System.out.println('"' + version1 + '"' + " is equal to " + '"' + version2 + '"' + ".");
		}
	}
}
```

This program defines a VersionComparer class with a compareVersions method, which takes two version strings and compares them component by component. The result is -1 if the first version is less than the second, 1 if it's greater, and 0 if they are equal.

Here are some test cases for the library:

compareVersions("1.2", "1.1") should return "1.2 is greater than 1.1."

compareVersions ("1.1", "1.2") should return "1.1 is less than 1.2."

compareVersions ("1.2", "1.2") should return "1.2 is equal to 1.2."

compareVersions ("1.2.3", "1.2.3.4") should return "1.2.3 is less than 1.2.3.4."

compareVersions ("2.0", "1.9.9.9") should return "2.0 is greater than 1.9.9.9."

compareVersions ("1.0.0.0", "1.0") should return "1.0.0.0 is equal to 1.0."

compareVersions ("1.0", "1.0.0.0") should return "1.0 is equal to 1.0.0.0."

These test cases cover scenarios with varying numbers of components and different values, ensuring that the compareVersions method works correctly.

Using TDD approach, I have defined more test cases in below Junit test case file (also uploaded in github).

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
class VersionComparerTest {
	@Test
	void testCompareVersions() {
		assertEquals(1, VersionComparer.compareVersions("1.2", "1.1"));
		assertEquals(-1, VersionComparer.compareVersions("1.1", "1.2"));
		assertEquals(0, VersionComparer.compareVersions("1.2", "1.2"));
		assertEquals(1, VersionComparer.compareVersions("2.0.1", "2.0"));
		assertEquals(-1, VersionComparer.compareVersions("2.0", "2.0.1"));
		assertEquals(1, VersionComparer.compareVersions("1.2.3.4", "1.2.3.3"));
		assertEquals(-1, VersionComparer.compareVersions("1.2.3.3", "1.2.3.4"));
		assertEquals(0, VersionComparer.compareVersions("1.2.3.4", "1.2.3.4"));
		assertEquals(0, VersionComparer.compareVersions("1.0.0.0", "1.0"));
		assertEquals(0, VersionComparer.compareVersions("1.0", "1.0.0.0"));
		assertEquals(-1, VersionComparer.compareVersions("1.9.9.9", "2.0"));
	}
	@Test
	void testCompareVersionsWithDifferentLengths() {
		assertEquals(1, VersionComparer.compareVersions("1.2.3", "1.2"));
		assertEquals(-1, VersionComparer.compareVersions("1.2", "1.2.3"));
	}
	@Test
	void testCompareVersionsWithSingleDigit() {
		assertEquals(0, VersionComparer.compareVersions("1", "1"));
		assertEquals(1, VersionComparer.compareVersions("2", "1"));
		assertEquals(-1, VersionComparer.compareVersions("1", "2"));
	}
}
```