

# BiL481 Homework - Part 1 - Due March 14, 23:59 *Prepared By Dr. Cagdas Evren Gerede, TOBB Ekonomi ve Teknoloji Üniversitesi*

Email your scanned handwritten pages to TA

1: Implement the *findMostFrequentItem* method of the App class so that it passes all the tests shown in AppTest class. Try to understand the comment above the method declaration. Your implementation should match the described documentation. In other words, it should pass the tests AND it should follow the specification.

```
public class App {
    // Returns the item with the biggest count. Each item has a corresponding count.
    // The number of counts is as many as the number of items. ith item's count is
    // the ith item of the counts.
    // Examine the unit tests to understand all the requirements.
    public static String findMostFrequentItem(String[] items, int... counts) {

        // ADD YOUR IMPLEMENTATION HERE.
    }
}
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.Test;

class AppTest {
    @Test void findsMostFrequentItemSuccessfully() {
        String[] items = new String[] {"A", "B", "C"};
        assertEquals("B", App.findMostFrequentItem(items, 10, 20, 15));
        assertEquals("A", App.findMostFrequentItem(items, 20, 10, 15));
        assertEquals("C", App.findMostFrequentItem(items, 15, 10, 20));
    }

    @Test void throwsException_WhenNoItemsProvided() {
        String[] items = new String[0];
        assertThrows(
            IllegalArgumentException.class,
            () -> { App.findMostFrequentItem(items); });
    }

    @Test void throwsException_WhenMoreItemsThanCounts() {
        final String[] items = new String[] {"A", "B", "C"};
        assertThrows(
            IllegalArgumentException.class,
            () -> { App.findMostFrequentItem(items, 10, 20); });
    }

    @Test void throwsException_WhenMoreCountsThanItems() {
        final String[] items = new String[] {"A", "B"};
        assertThrows(
            IllegalArgumentException.class,
            () -> { App.findMostFrequentItem(items, 10, 20, 15); });
    }
}
```

2. Implement AppTest class below so that the test coverage for the sum method is %100 in jacoco. Implement a separate unit test method for each test case. Start with the AppTest class shown below and implement various testxxx methods. Make sure the name of the method clearly describes the test case.

```
public static int sum(int lowBoundary, int highBoundary, int...numbers) {
    if (numbers.length == 0 ||
        lowBoundary > highBoundary ||
        lowBoundary < 0) {
        throw new IllegalArgumentException();
    }

    int sum = 0;
    for (int value : numbers) {
        if (lowBoundary <= value && value <= highBoundary) {
            sum += value;
        }
    }
    return sum;
}
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.Test;

class AppTest {

    // Add your test methods here

}
```

**3:** Below you can see the Monitor class. The MonitorTest class tests the Monitor class by mocking DataService and EmailService. Complete the implementation in the MonitorTest class correctly so that all the tests pass and the tests are testing the test case the test method name describes.

```
import java.util.List;

public class Monitor {
    DataService dataService;
    EmailService emailService;
    static final String ALERT_EMAIL = "alert@acme.com";

    public Monitor(DataService dataService, EmailService emailService) {
        this.dataService = dataService;
        this.emailService = emailService;
    }

    public void alertIfAboveThreshold(String metricName, int threshold) {
        List<Integer> dataPoints =
            dataService.lastWindowOfData(metricName);
        for (int point : dataPoints) {
            if (point > threshold) {
                String alertMessage = String.format(
                    "Metric %s is above the threshold: %d vs. %d",
                    metricName,
                    point,
                    threshold);

                emailService.sendEmail(ALERT_EMAIL, alertMessage);
                return;
            }
        }

        if (dataPoints.size() == 0) {
            String alertMessage = String.format(
                "No data for metric %s", metricName);
            emailService.sendEmail(ALERT_EMAIL, alertMessage);
        }
    }
}

public class DataService {
    public List<Integer> lastWindowOfData(String metricName) { ... }
}

public class EmailService {
    public void sendEmail(String to, Object body) { ... }
}
```

```
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.ArgumentMatchers.eq;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
```

```
import java.util.Arrays;
import java.util.List;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class MonitorTest {
    Monitor monitor;
    DataService dataService;
    EmailService emailService;

    @BeforeEach void setUp() {
        .... // IMPLEMENT
    }

    @Test void alertsWhenAboveThreshold() {
        .... // IMPLEMENT
    }

    @Test void doesNotAlertsWhenBelowThreshold() {
        .... // IMPLEMENT
    }

    @Test void alertsWhenNoDataPoints() {
        .... // IMPLEMENT
    }
}
```

**4:** Give an example of a block code and its tests such that the test coverage is 100% and the tests pass successfully but the code still has a bug. Demonstrate the bug with an example input and for this problematic input the method should not throw an exception (throwing an exception in a sense may mean the bug is caught. I am looking for a silent bug that does not show itself clearly with an exception. Therefore, the caller may not immediately be aware of the issue after the method returns successfully)

**5:** Write the file contents in a) the working directory, b) the index, c) the HEAD at the end of the following commands:

```
git init
echo Isaac Asimov > f.txt
git add .
echo Jules Verne > g.txt
git commit -m "update"
echo Arthur C. Clarke > h.txt
```

(Use the following table to provide your answers. For the other questions, also provide your answers in a similar table format)

	Working Directory	Index	HEAD
f.txt			
g.txt			
h.txt			

**6:** Write the files in a) the working directory, b) the index, c) the HEAD at the end of the following commands:

```
git init
echo Isaac Asimov > f.txt
git add .
echo Jules Verne > g.txt
git commit -m "update"
echo Arthur C. Clarke >> f.txt
git add f.txt
git commit -m "update 2"
echo Jules Verne >> f.txt
git reset --hard HEAD~1
```


**7:** Write the outputs of each of the three `/s` commands below (Note: The linux command `ls` lists the files in the working directory)

```
git init
echo Andy Weir > martian.txt
git add .
git commit -m "update"
echo Arthur C. Clarke > space_odyssey.txt
git checkout -b bugfix
echo George Orwell > animal_farm.txt
git add .
git commit -m "update 2"
ls
git checkout master
ls
git merge bugfix
ls
```

**8:** Draw the commit graph after all the commands shown below are applied. Every node in the graph should be annotated with the commit message. Draw the parent link(s) of all commits. Show where HEAD and branch pointers pointing to at the end. Each node in the graph should show the contents of all files.

```
git init
echo "world" > f.txt
git add .
git commit -m "first"
git checkout -b bugfix
echo "jupiter" > g.txt
git add .
git commit -m "second"
echo "venus" > g.txt
git add .
git commit -m "third"
git checkout master
echo "saturn" > f.txt
git add .
git commit -m "fourth"
git merge bugfix -m "fifth"
echo "neptune" > f.txt
git add .
git commit -m "sixth"
git checkout bugfix
echo "pluto" > g.txt
git add .
git commit -m "seventh"
```

**21:** Starting from an empty directory, write the list of **git** and **echo** commands that create the shown commit graph (Note: There are 2 branches: *master* and *bugfix*. There are 7 commits.

Each commit's message is labeled under the box such as "1 Ocak". The arrow  shows the parent commit. Each big box shows a commit and each small box inside a big box shows a file with the filename on top of the box such as "f.txt". The content of a file is shown inside the box such as "armut").

