On November 6, 1893, Pyotr Ilyich Tchaikovsky, a well-known Russian composer, died during the cholera epidemic. He died a few days after the premiere of his Sixth Symphony "Pathetique," a piece full of suffering and pain. In the US, Tchaikovsky is most known for his music to the Nutcracker ballet - usually performed at Christmas time, and for his 1812 Overture, often performed on July 4.

1a. Many of his musical pieces -- e.g., all his symphonies -- consist of several parts (called "movements"). Write a method named average that, given the total duration \( d \) of a piece and the number of parts \( p \) in this piece, returns the average duration of a movement.

1b. The Sixth Symphony lasts 46 minutes and consists of 4 movements. Call (invoke) your method average in the main method to compute the average duration of each movement. You do not need to write the entire main method, just the part that assigns values to the corresponding variables duration and parts and calls your method.

1c. Trace, step by step, how the computer will perform the needed computations, and check that the result is indeed correct.

```java
public static double average(int d, int p) {
    double result = ((double)d / p);
    return result;
}
```

b.) In the main method:

```java
double a = average(46, 4);
```

c.) Trace:

```
Main
average

11.5 a

result = 46.0 / 4 = 11.5
```
2a. Define a class `Symphony` whose objects are different symphonies written by Tchaikovsky. The description of each symphony should contain its title, its duration, and the number of parts (movements). Your class should contain a constructor method, get- and set-methods, and a method for computing the average duration of a movement.

2b. Use your class in the `main` method to define a new object `pathetique` of type `Symphony` describing Tchaikovsky's Sixth Symphony that lasts for 46 minutes and consists of 4 movements. Compute and print the average duration of a movement. El Paso Symphony performed it in 48 minutes. Replace 46 with the new value 48, and compute and print the new average duration of a movement.

2c. Trace your program step-by-step.

```java
public class Symphony {
    private String title;
    private int duration;
    private int movements;
    public Symphony(String ti, int dur, int move) {
        title = ti;
        duration = dur;
        movements = move;
    }
    public void setTitle(String ti) { title = ti; }
    public void setDuration(int dur) { duration = dur; }
    public void setMovements(int move) { movements = move; }
    public String getTitle() { return title; }
    public int getDuration() { return duration; }
    public int getMovements() { return movements; }
    public double findAverage() {
        double result = (double) duration / movements;
        return result;
    }
}

b.) In the main method:
    Symphony pathetique = new Symphony("Tchaikovsky's Sixth Symphony", 46, 4);
    System.out.println(pathetique.findAverage());
    pathetique.setDuration(48);
    System.out.println(pathetique.findAverage());
```

Part (c) written on back of page →
c. Trace:

- **Symphony**:
  - T.S.S title
  - 40 dur
  - 4 move

- **Set Title**:
  - Tchaikovsky's Sixth Symphony

- **Set Duration**:
  - 46.0

- **Set Movements**:
  - 4

- **Get Title**:
  - Returns title

- **Get Duration**:
  - Returns duration

- **Get Movements**:
  - Returns movements

- **Find Average**:
  - \( \text{result} = \frac{46.0}{4} = 11.5 \)
  - \( \text{result} = \frac{48.0}{4} = 12.0 \)

- **Prints**:
  - 11.5
  - 12.0
3a. Write a piece of code that, given an array `duration` of durations of different symphonies and an array `parts` of numbers of movements in different symphonies, creates a new array `av` consisting of average durations of a movement in each symphony. Assume the arrays `duration` and `parts` have been declared, initialized, and that they have the same length.

3b. To check the correctness of the code you wrote in Part 3a, write a piece of code that defines a new array `duration` with values 46 and 60 and a new array `parts` with elements 4 and 5.

3c. Trace step-by-step how the piece of code you wrote in Part 3a will compute the corresponding average durations.

```java
a) public static double[] problem3(int[] duration, int[] parts) {
    double[] av = new double[parts.length];
    for (int i = 0; i < duration.length; i++) {
        av[i] = (double) duration[i] / parts[i];
    }
    return av;
}

b) In main method:
    int[] duration = {46, 60};
    int[] parts = {4, 5};
    double[] av = problem3(duration, parts);

c) Trace:

    duration 46 60
    parts 4 5
    x

    duration
    parts
    i
    0 x 2

    av[0] = 46.0/4 = 11.5
    av[1] = 60.0/5 = 12.0
```

file://G:\cs1401.13\test3v.html
4a. Which of Tchaikovsky's symphonies is the longest? Write a method that, given an array \( d \) of durations of different symphonies and the array \( t \) of their titles, returns the title of the longest symphony. Assume the arrays have been declared, initialized, and have the same non-zero length.

4b. To check the correctness of your method, write a piece of code that defines new arrays \( \text{duration} \) consisting of 3 elements 46, 60, and 40, and \( \text{titles} \) with elements P, C, and X.

4c. Trace step-by-step how the piece of code you wrote in Part 4a will find the title of the longest symphony.

\[
\text{public static String problem4 (int[] d, String[] t)}
\]

\[
\text{int longest_duration;}
\]

\[
\text{longest_duration = d[0];}
\]

\[
\text{String longest_symphony;}
\]

\[
\text{longest_symphony = t[0];}
\]

\[
\text{for (int i = 1; i < d.length; i++)}
\]

\[
\text{if (d[i] > longest_duration)}
\]

\[
\text{d[i] = longest_duration;}
\]

\[
\text{t[i] = longest_symphony;}
\]

\[
\text{return longest_symphony;}
\]

b.) In main method:

\[
\text{int[] duration = \{46, 60, 40\};}
\]

\[
\text{String[] titles = \{"P", "C", "X"\};}
\]

\[
\text{String x = problem4 (duration, titles);}
\]

c.) Trace:
5a. Describe what is white-box and black-box testing.

5b. Describe the main rules for testing.

a.) **White-box testing** is a strategy for testing a program in which the tester is fully aware of the internal workings of the program's code.

**Black-box testing** is an alternate strategy for testing a program in which the tester has no knowledge of the program's code.

b.) For **white-box testing**, the rule is to trace the program step-by-step.

For **black-box testing**, the tester simply evaluates the accuracy of outputs for given inputs.

- Test all branches
- Test on boundary values
- Test on simple & typical values
- Test on random values