Computer Engineering Fundamentals
Assignment 0.

Engr 9859, 2013

Due Friday Sept 13 at 23:00.

For each question you will be marked on programming style as well as correctness. To see my opinion about what constitutes good programming style see http://www.engr.mun.ca/~theo/Courses/ds/pub/style.pdf. In short:

- All .java files must be professionally commented; in particular, each file should contain a comment header that gives your name, student number and each subroutine should have a comment at the start of it. I encourage you to use the "javadoc" conventions for comments.
- Code and comments must be consistently indented; if tab characters are used, tab stops should be set every 4 characters.
- Names must be chosen carefully and spelled correctly. (Use names starting with lower case letters for variables and methods; use names starting with upper case letters for classes and interfaces.)
- Use subroutines to avoid redundant coding.
- Keep control structures and data structures simple.

All classes must be tested by you prior to being submitted. You are welcome to share test code with each other.

The assignment is to be done alone. Each file should contain the following declaration in comments near the top. "This file was prepared by [your name here]. It was completed by me alone.". If you obtained help in doing the assignment, do not include this declaration, but rather an explanation of the nature of any help that you received in doing the assignment.

Theme: The theme for this assignment is arrays and methods. We will also introduce classes and interfaces in question 1. I will try to cover interfaces a bit on Monday. But in case I don't, you can read up on them in the text book. The question 1 also has a bit of object composition and is an example of the Command pattern.

Submission. Information on how to submit will be forthcoming.

Files. I'll post a number of files on the web site to help you with this assignment.

Q0. Write a public class named Utilities in a package named huffman. Recall that characters of

(a) Utilities should contain a method with the following signature.

static void frequencyCount(**char**[] in, **int**[] frequency)

As a precondition you may assume that

- frequency refers to an array of length less or equal to 2^{16}.
- No character in array in corresponds to an integer greater or equal to frequency.length.[1]

Your code should check these preconditions using the Assert.check method in class Assert, which I will distribute.

The method should change nothing but the items in frequency.

As a postcondition. The final value of frequency[i], for each $i \in \{0, ..\text{frequency.length}\}$,[2] is equal to the number of times character i is found in in That is, the postcondition is

$$\forall i \in \{0, ..\ell\} \cdot \text{frequency}'[i] = \sum_{j \in \{0,..\text{in.length}\}} \delta(\text{in}[j] = char(i))$$

where $\ell = \text{frequency.length}$, $\text{frequency}'$ means the final value of frequency, $\delta(\text{true}) = 1$, $\delta(\text{false}) = 0$, and $char(i)$ means the character with unicode (UFT-16) code equal to i, for example, $char(65) = $'A'.

Test your method. You will also submit a JUnit 4 tests class called UtilitiesTests in package huffman.

(b) Utilities should contain a method with the following signature.

static void sort(**int**[] frequency, **char**[] symbol)

As a precondition you may assume that

- both arrays have the same length

Your code should check this precondition using Assert.check .

The idea is that initially frequency[i] represent the frequency of character symbol[i]. After the method has executed, this should still be true, but the values in frequency should be be sorted.

The method should sort the frequencies array so that the frequencies are ordered from largest to smallest. At the same time it should rearrange the symbol

[1] Recall that characters corresponds to integers from 0 up to but not including 2^{16}.

[2] Notation note: The notation $\{i, ..k\}$ means

$$\{j \in \mathbb{Z} \mid i \leq j < k\}$$

where $\mathbb{Z}$ is the set of all integers and $i, k \in \mathbb{Z}$, i.e., the set of all integers greater or equal to i and less than k.

array so that the correspondence between frequencies and symbols remains the same. That is, the following set

$$\{(\mathsf{symbol}[i], \mathsf{frequency}[i]) \mid i \in \{0, ..\ell\}\}$$

does not change. For example suppose initially item 9 of the frequency array is the largest and symbol[9] is x. After sorting, frequency[0] will be the largest item in the frequency array and symbol[0] will be x.This is easy to arrange. If your sorting algorithm works by swapping items of the frequency array, you simply add commands to swap the corresponding items of the symbol array at the same time.

Test your method. You will also submit a JUnit 4 tests class called UtilitiesTests in package huffman.

Q1. For this question we will represent sound clips as arrays. Each clip is represented by an array of 32 bit integers.

Each integer will be a positive number between 0 and $2^{31} - 1$, inclusive. Each integer represents air pressure (on a linear scale) at a particular time. The sampling times are equally spaced and may be between 8,000 and 44,000 Hz.

You will design classes that all implement the same interface. The interface is called SoundTransform and is in the sound package.

The interface is

```
package sound;

public interface SoundTansformIntf {
    public int[] doTransform( int[] clip ) ;
}
```

As you can see from its signature, each doTransform method takes an array as a parameter and returns an array as a result. **The contents of the parameter array must not be changed**. Each invocation of the doTransform method must create the array that is returned.

In addition to implementing the interface, each of your classes should also override the toString method from class Object. (See Horstmann and Cornell for more on toString().)

(a) Change volume For this part create a transform that will change the pressures by a given amount

$$p_{\text{out}} = \lfloor c \times p_{\text{in}} \rfloor$$

When $c \times p_{\text{in}}$ exceeds Integer.MAXINT $(2^{31} - 1)$, the volume should be capped at that maximal value.

The coefficient c will be passed in as a parameter to the classes constructor. So you should create a class with the following signature

```
package sound;

public class ChangeVolumeTransform implements SoundTansformIntf {
```

```
    public ChangeVolumeTransform( double c ) { ... }

    public int[] doTransform(int[] clip) { ... }

}
```

Test your method. You will also submit a JUnit 4 tests class called TransformTests is package sound.

(b) Speed-up. For this part, create a transform that will reduce the length of the clip by cutting out all but the first m samples out of each group of n samples, where $0 \leq m \leq n$ and $n > 0$. For example with $m = 4$ and $n = 5$

0	1	2	3	4	5	6	7

will be transformed to an output of

0	1	2	3	5	6	7

With $m = 2$ and $n = 5$

0	1	2	3	4	5	6	7

will be transformed to an output of

0	1	5	6

Using $m = 1$, $n = 2$ you can speed up sound clips, raising the pitch by an octave. Using $m = 300$, $n = 400$ you will speed up the clip without affecting the pitch. (Depending on sampling rate. This trick is sometimes used by TV stations to speed up movies, so that they have more time for commercials. It is also used for voice-mail.) Hint: If the length of the input clip is k, the length of the output clip should be

$$m \times (k \operatorname{div} n) + \min(m, k \bmod n)$$

where

$$k \operatorname{div} n = \lfloor k/n \rfloor$$
$$k \bmod n = n - n \lfloor k/n \rfloor$$

```
package sound;

public class SpeedUpTransform implements SoundTansformIntf {

    public SpeedUpTransform( int m, int n ) { ... }
```

```
    public int[] doTransform(int[] clip) { ... }

}
```

Test your method. You will also submit a JUnit 4 tests class called TransformTests in package sound.

(c) Compose. The CompositionTransform will apply two transforms in sequence: first f and then g.

```
package sound;

public class ComposeTransform implements SoundTansformIntf {

    public ComposeTransform(SoundTansformIntf f, SoundTansformIntf g){...}

    public int[] doTransform(int[] clip) { ... }

}
```

Test your method. You will also submit a JUnit 4 tests class called TransformTests in package sound.

(d) Your own transformation. Create your own transformation. Call it MyTransform .Document it carefully.

Test your method. You will also submit a JUnit 4 tests class called TransformTests in package sound.