CSE 142, Summer 2010
Final Exam Part B
Friday, August 20, 2010

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

Rules:

- You have 60 minutes to complete Part B of this exam (problems 6 – 8).
  You may receive a deduction if you keep working after the instructor calls for papers.
- This test is open-book/notes.
- You may not use any computing devices of any kind including calculators.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- You do not need to write any `import` statements in your code.
- Please do not abbreviate code, such as writing ditto marks ("") or dot-dot-dot marks (...). The only abbreviations allowed are `S.o.p`, `S.o.pln`, and `S.o.pf` for `System.out.print`, `println`, and `printf`.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your Student ID to a TA or instructor for your submitted exam to be accepted.

Good luck!

Problem	Description	Earned	Max
1	Array Mystery	?	10
2	Reference Mystery	?	10
3	Inheritance Mystery	?	10
4	File Processing	?	13
5	Array Programming	?	12
6	File Processing		15
7	Array Programming		15
8	Classes and Objects		15
X	Extra Credit		+1
TOTAL	**Total Points**		**100**

6. File Processing

Write a static method named **`playlist`** that accepts as its parameter a `Scanner` for an input file representing a sequence of songs. Your method will reformat the song names and lengths found in the file to be more readable and output them to the console. Additionally, your method will return whether or not all the songs in the file would fit on a standard CD. A standard CD can hold up to, and including, 80 minutes of audio data.

Each line of the playlist file consists of a song name consisting of one or more words, a semi-colon (i.e. ;), and the song's length as two integers, minutes and seconds. Your method should read each line and output the song information in a more readable format to the console. Separate each token of the song name with a single space, capitalize the first letter of each word, and lowercase the rest. If the total time needed to play all the songs is 80 minutes or less, return `true`. Otherwise, return `false`.

The table below shows two example files. The first file has five songs, which fit on a CD because the total playtime is under 80 minutes. The second file has six songs, but all six songs cannot fit on a CD because it takes more than 80 minutes to play them. Regardless of the total playtime, the reformatted playlist is always printed to the console.

Input File	Console Output	Value Returned

`Alejandro ;  8 43`
`  hOme  ; 5 3`
`WHEN yoU Love SomEBODY ;  2 5`
`  rudE to rile ; 3  30`
`CALIFORNIa  gurls ; 3  56`
	`Alejandro 8:43`
	`Home 5:3`
	`When You Love Somebody 2:5`
	`Rude To Rile 3:30`
	`California Gurls 3:56`
		`true`

`tubULaR  bElls ; 25 01`
`  reVolution 9 ;  8 22`
`lasT  caLL ; 12 41`
`ShARKS  and SAILORS ; 8 13`
`osAKA part  I  ; 38 58`
`SHINE ON YOU  crazy diamond ; 26 01`
	`Tubular Bells 25:1`
	`Revolution 9 8:22`
	`Last Call 12:41`
	`Sharks And Sailors 8:13`
	`Osaka Part I 38:58`
	`Shine On You Crazy Diamond 26:1`
		`false`

Assume valid input. Assume the file contains data for at least one song, every line contains one song's data, and every song is in the format described above. Each song's minutes and seconds are in the range of 0 – 59.

(blank page intended for problem #6)

7. Array Programming

In grade school, when a child is caught skipping another child in line, that child gets punished by having to go to the back of the line. Write a static method named **`caughtSkipping`** that accepts an array of Strings *line*, a String *skipper*, and a second String *skipped*. If *skipper* is found in the *line* array before (at a lower numbered index than) *skipped*, your method should perform the following actions:

- The *skipper* is temporarily pulled out of the *line*.
- All elements that were after *skipper* are shifted over one place (left-shifted to one index lower).
- The *skipper* is moved to the end of the *line*.

Otherwise, if *skipper* is not found before *skipped*, no change to the *line* should be made.

For example, given the *line* array declared as follows:

```
String[] line = {"Iva", "Ben", "Brett", "Ally", "Kim", "Jesse"};
```

If a call were made `caughtSkipping(line, "Jesse", "Kim")` to see if Jesse skipped Kim, no change would be made to the order of the elements in `line`. However, if a second call were made `caughtSkipping(line, "Ben", "Ally")` to see if Ben skipped Ally, after the call the order of the elements in *line* would be:

```
{"Iva", "Brett", "Ally", "Kim", "Jesse", "Ben"}
```

Do not make any assumptions about the length of the array or the range of values it might contain. For example, the array might contain both, just one, or neither *skipper* and *skipped*. You may assume that the *line* array, the String *skipper*, and the String *skipped* are not `null`.

You may not use any temporary arrays to help you solve this problem. (But you may declare as many simple variables as you like, such as `int`s.) You also may not use any other data structures or complex types such as `String`s, or other data structures that were not taught in CSE 142 such as the `ArrayList` class from Chapter 10.

(blank page intended for problem #7)

8. Classes and Objects

Suppose that you are provided with a pre-written class `ClockTime` as described at right. Assume that the fields, constructor, and methods shown are implemented. You may refer to them or use them in solving this problem.

This version of `ClockTime` can represent times in either standard or military time format. Military time is an alternative way to describe times where hours go from 0 – 24 instead of 1 – 12 (the AM and PM suffixes are not used). The following table shows how to convert between standard and military time:

Standard Time	Military Time
12:00AM (midnight)	24:00 hours
12:01AM to 12:59AM	00:01 hours to 00:59 hours
1:00AM to 11:59AM	01:00 hours to 11:59 hours
12:00PM (noon) to 12:59PM	12:00 hours to 12:59 hours
1:00PM to 11:59PM	13:00 hours to 23:59 hours

Write an instance method named **`toStandardTime`** that will be placed inside the `ClockTime` class to become a part of each `ClockTime` object's behavior. The `toStandardTime` method converts the `ClockTime` object into standard time if it is in military time. You can tell which format a particular `ClockTime` object is using by examining its `label` field. For example, if the following object is declared in client code:

```
ClockTime t1 = new ClockTime(15, 27, "hours");
```

```
// A ClockTime object represents
// an hour:minute time in
// in either standard time such
// as 10:45 AM or 6:27 PM or
// military time such as
// 10:45 hours or 18:27 hours.

public class ClockTime {
    private int hour;
    private int minute;
    private String label;


    // Constructs a new time for
    // the given hour/minute.
    // If time is in standard
    // time, the label is "AM"
    // or "PM".  If the time is
    // in military time, the
    // label is "hours".
    public ClockTime(int h,
            int m, String label)

    // returns the field values
    public int getHour()
    public int getMinute()
    public String getLabel()

    // returns String for time;
    // example: "06:27 PM"
    // example: "18:27 hours"
    public String toString()

    // your method would go here

}
```

The following call to your method would cause `03:27 PM` to be printed to the console:

```
t1.toStandardTime();
System.out.println(t1);     // 03:27 PM
```

Here are some other objects. Their results when used with your method and then printed to the console are shown at right in comments:

```
ClockTime t2 = new ClockTime(24, 00, "hours");   // 12:00 AM
ClockTime t3 = new ClockTime( 0, 30, "hours");   // 12:30 AM
ClockTime t4 = new ClockTime(10, 35, "hours");   // 10:35 AM
ClockTime t5 = new ClockTime(12, 15, "hours");   // 12:15 PM
ClockTime t6 = new ClockTime(13, 00, "hours");   // 01:00 PM
ClockTime t7 = new ClockTime(19, 11, "hours");   // 07:11 PM
ClockTime t9 = new ClockTime(23, 59, "hours");   // 11:59 PM
ClockTime ta = new ClockTime( 9, 30, "AM");      // 09:30 AM
ClockTime tb = new ClockTime( 5, 01, "PM");      // 05:01 PM
```

Assume that the state of the `ClockTime` object is valid and that the `label` field stores `"AM"`, `"PM"`, or `"hours"`.

(blank page intended for problem #8)

X. Extra Credit (+1 point)

What animal do you think your TA most resembles? Draw a picture of what your TA if he/she as an animal.

(Any picture that appears to reflect a nontrivial effort will receive the bonus point.)