CSE 142, Summer 2010
Final Exam Part A
Thursday, August 19, 2010

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

Rules:

- You have 60 minutes to complete Part A of this exam (problems 1 – 5). You will complete Part B of this exam (problems 6 – 8) tomorrow in lecture.
- 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		15
5	Array Programming		15
6	File Processing		15
7	Array Programming		15
8	Classes and Objects		10
X	Extra Credit		+1
TOTAL	**Total Points**		**100**

1. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = 1; i < a.length; i++) {
        a[i] = a[a.length - 1 - i] - a[i - 1];
    }
}
```

Indicate in the right-hand column what values would be stored in the array after the method `arrayMystery` executes if the array in the left-hand column is passed as its parameter.

Original Contents of Array	Final Contents of Array
`int[] a1 = {42, 42};` `arrayMystery(a1);`	_____
`int[] a2 = {6, 2, 4};` `arrayMystery(a2);`	_____
`int[] a3 = {7, 7, 3, 8, 2};` `arrayMystery(a3);`	_____
`int[] a4 = {4, 2, 3, 1, 2, 5};` `arrayMystery(a4);`	_____
`int[] a5 = {6, 0, -1, 3, -2, 0, 4};` `arrayMystery(a5);`	_____

2. Reference Semantics Mystery

The following program produces 4 lines of output. Write the output below, as it would appear on the console.

```
import java.util.*;   // for Arrays class

public class Island {
    int lat;
    int lng;

    public Island(int initialLat, int initialLng) {
        lat = initialLat;
        lng = initialLng;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int n = 4;
        int[] a = {8, 15};
        Island isle = new Island(16, 23);

        mystery(n, a, isle);
        System.out.println(n + " " + Arrays.toString(a) + " " + isle.lat + " " + isle.lng);

        n++;
        a[0] += 2;
        isle.lat = 42;
        mystery(n, a, isle);
        System.out.println(n + " " + Arrays.toString(a) + " " + isle.lat + " " + isle.lng);
    }

    public static void mystery(int n, int[] a, Island isle) {
        n = n - 2;
        a[1]++;
        isle.lng += 100;
        System.out.println(n + " " + Arrays.toString(a) + " " + isle.lat + " " + isle.lng);
    }
}
```

3. Inheritance Mystery

Assume that the following four classes have been defined:

```
public class Sawyer extends Jack {
     public void follow() {
         System.out.print("sawyer-F   ");
         super.follow();
     }
 }

public class Kate {
     public void lead() {
         System.out.print("kate-L   ");
     }

     public void follow() {
         System.out.print("kate-F   ");
     }

     public String toString() {
         return "kate";
     }
}
```

```
public class Jack extends Locke {
     public void lead() {
         super.lead();
         System.out.print("jack-L   ");
     }

     public String toString() {
         return "jack";
     }
}

public class Locke extends Kate {
     public void follow() {
         lead();
         System.out.print("locke-F   ");
     }
}
```

Given the classes above, what output is produced by the following code?

```
Kate[] characters = {new Locke(), new Jack(), new Sawyer(), new Kate()};
for (int i = 0; i < characters.length; i++) {
    System.out.println(characters[i]);
    characters[i].lead();
    System.out.println();
    characters[i].follow();
    System.out.println();
    System.out.println();
}
```

4. File Processing

Write a static method named **`triathlon`** that accepts as its parameter a `Scanner` for an input file whose data represents triathlon race results for athletes. Your method should add up the swimming, biking, and running times for each athlete and report the total time for each athlete and their time difference from the winner's. The input consists of a series of tokens. Each athlete's data is represented by four tokens in the following order: athlete's first name, swimming time, biking time, and running time (all times are given in minutes). The data for an athlete may or may not span multiple lines, but you are guaranteed the athletes will come in the order they finished the race (the winner's data is first, the second place triathlete's data comes next, etc.).

Here is an example input file. Notice the spacing and that the order the athletes appear in the file are in the same order that they finished the race:

```
Meghan 12 40 23 Bryan 16
42 20 Lori 14 41 29 Jessica 18

     37 30 Toni 19  43
29 Tamara 17 42 34
```

For this input, you method should produce the output below. For example, Meghan swam for 12 minutes, biked for 40 minutes, and ran for 23 minutes so it took for Meghan 75 minutes to finish the race. Notice that for the athletes that did not win the race, in addition to reporting their total race time, the difference in their time from the winner's is also reported. For example, Bryan swam for 16 minutes, biked for 42 minutes, and ran for 20 minutes so it took Bryan 78 minutes to finish the race, which is 3 minutes longer than Meghan's winning time of 75 minutes.

```
Meghan: 75 min
Bryan: 78 min (+3 min)
Lori: 84 min (+9 min)
Jessica: 85 min (+10 min)
Toni: 91 min (+16 min)
Tamara: 93 min (+18 min)
```

You may assume that the file contains data for at least one athlete. You may also assume that the input is valid; that the input has four tokens per athlete, the first token for an athlete is a String and the following three are integers.

5. Array Programming

Write a static method named **`sandwich`** that accepts a string *bread*, an array of strings *fillings*, and an integer *fillFactor*. Your method should return a new array *sandwich* which has *bread* as its first and last element and has *fillFactor* repetitions of the *fillings* array occupying the rest of its elements. For example, if *fillings* stores `{"chicken", "lettuce"}` and you call **`sandwich`** with `sandwich("wheat", fillings, 2)`, your method should return `{"wheat", "chicken", "lettuce", "chicken", "lettuce", "wheat"}` because you add the elements of *fillings* as the inner elements of the *sandwich* array twice.

The table below shows some additional calls to your method and the expected values returned:

Arrays	Call and Value Returned
`String[] fillings1 = {"tuna", "mayo", "salt", "carrot"};`	`sandwich("bun", fillings1, 1)` returns `{"bun", "tuna", "mayo", "salt", "carrot", "bun"}`
`String[] fillings2 = {"beef"};`	`sandwich("pita", fillings2, 4)` returns `{"pita", "beef", "beef", "beef", "beef", "pita"}`
`String[] fillings3 = {"ham", "cheese"};`	`sandwich("rye", fillings3, 3)` returns `{"rye", "ham", "cheese", "ham", "cheese", "ham", "cheese", "rye"}`
`String[] fillings4 = {"banana", "nutella"};`	`sandwich("white", fillings4, 0)` returns `{"white", "white"}`
`String[] fillings5 = {};`	`sandwich("", fillings5, 5)` returns `{"", ""}`

For full credit, do not modify the elements of *fillings*. You may assume that *fillings* is not `null` and *fillFactor* is non-negative.

Hint: Nested loops can be used to solve this, but they are not necessary.