CSE 142, Autumn 2009
Final Exam
Wednesday, December 16, 2009

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

Rules:

- You have 110 minutes to complete this exam.
  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	File Processing		10
6	Array Programming		15
7	Array Programming		10
8	Critters		10
9	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 - 1; i++) {
        a[i] = a[i - 1] - a[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, 5, 0, -3};` `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 BasicPoint {
    int x;
    int y;

    public BasicPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int n = 10;
        int[] a = {20};  // an array with just one element
        BasicPoint p = new BasicPoint(30, 40);

        mystery(n, a, p);
        System.out.println(n + " " + Arrays.toString(a) + " " + p.x + "," + p.y);

        a[0]++;
        p.x++;
        mystery(n, a, p);
        System.out.println(n + " " + Arrays.toString(a) + " " + p.x + "," + p.y);
    }

    public static int mystery(int n, int[] a, BasicPoint p) {
        n++;
        a[0]++;
        p.y++;
        System.out.println(n + " " + Arrays.toString(a) + " " + p.x + "," + p.y);
        return n;
    }
}
```

3. Inheritance Mystery

Assume that the following four classes have been defined:

```
public class Biggie extends JayZ {
    public void a() {
        System.out.print("Biggie a   ");
        super.a();
    }

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

public class JayZ extends Tupac {
    public void a() {
        System.out.print("JayZ a   ");
        b();
    }
}
```

```
public class FiftyCent extends Biggie {
    public void b() {
        System.out.print("FiftyCent b   ");
    }
}
public class Tupac {
    public void a() {
        System.out.print("Tupac a   ");
    }

    public void b() {
        System.out.print("Tupac b   ");
    }

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

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

```
Tupac[] elements = {new Biggie(), new Tupac(), new JayZ(), new FiftyCent()};
for (int i = 0; i < elements.length; i++) {
    elements[i].a();
    System.out.println();
    elements[i].b();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

4. File Processing

Write a static method named **`countCoins`** that accepts as its parameter a `Scanner` for an input file whose data represents a person's money grouped into stacks of coins. Your method should add up the cash values of all the coins and print the total money at the end. The input consists of a series of pairs of tokens, where each pair begins with an integer and is followed by the type of coin, which will be either "pennies" (1 cent each), "nickels" (5 cents each), "dimes" (10 cents each), or "quarters" (25 cents each), case-insensitively. A given coin might appear more than once on the same line.

For example, if the input file contains the following text:

```
3 pennies 2 quarters 1 pennies 3 nickels 4 dimes
```

3 pennies are worth 3 cents, and 2 quarters are worth 50 cents, and 1 penny is worth 1 cent, and 3 nickels are worth 15 cents, and 4 dimes are worth 40 cents. The total of these is 1 dollar and 9 cents, therefore your method would produce the following output if passed this input data. Notice that it says 09 for 9 cents.

```
Total money: $1.09
```

Here is a second example. Suppose the input file contains the following text. Notice the capitalization and spacing:

```
12   QUARTERS      1   Pennies      33
PeNnIeS

  10    niCKELs
```

Then your method would produce the following output:

```
Total money: $3.84
```

You may assume that the file contains at least 1 pair of tokens. You may also assume that the input is valid; that the input has an even number of tokens, that every other token is an integer, and that the others are valid coin types.

5. File Processing

Write a static method named **`matchIndex`** that accepts as its parameter a `Scanner` for an input file. Your method should compare each neighboring pair of lines (the first and second lines, then the third and fourth lines, and so on) looking for places where the character at a given 0-based index from the two lines is the same. For example, in the strings "hello" and "belt", the characters at indexes 1 ('e') and 2 ('l') match. Your code should be case-sensitive; for example, "J" does not match "j".

For each pair of lines, your method should print output showing the character indexes that match, separated by spaces in the format shown below. If no characters match, print "none" instead as shown below.

For example, suppose the input file contains the following text. (Line numbers and character indexes are shown around the input and matching characters are shown in bold, but these markings do not appear in the actual file.)

```
    01234567890123456789012345678901234567890123456789
 1  The quick brown fox
 2  Those achy down socks
 3  Wheels on the school bus go round
 4  The wipers go swish swish swish
 5  His name is Robert Paulson
 6  So long 'n thanks for all the fish
 7  Humpty Dumpty sat on a wall
 8  And then he also had a great fall
 9  booyakasha
10  Bruno Ali G Borat
```

When passed the above file, your method would produce the following output:

```
lines 1 and 2: 0 1 7 12 13 14 15 17
lines 3 and 4: 1 2 13 14 23
lines 5 and 6: none
lines 7 and 8: 4 14 20 21 22
lines 9 and 10: none
```

Notice that lines are not generally the same length. You may assume that the file contains an even number of lines.

6. Array Programming

Write a static method named **`longer`** that accepts two arrays of strings *a1* and *a2* as parameters and returns a new array *a3* such that each element of *a3* at each index *i* stores whichever string has greater length (more characters) between the elements at that same index *i* in arrays *a1* and *a2*. If there is a tie, take the element from *a1*.

For example, if *a1* and *a2* store the following elements:

```
String[] a1 = {"star", "pie", "jelly bean", "car"};
String[] a2 = {"cookie", "fig", "banana", "soda"};
```

Then your method should return the new array `{"cookie", "pie", "jelly bean", "soda"}`.

If the arrays *a1* and *a2* are not the same length, the result returned by your method should have as many elements as the larger of the two arrays. If a given index *i* is in bounds of *a1* but not *a2* (or vice versa), there are not two elements to compare, so your result array's element at index *i* should store the value `"oops"`. For example, if *a1* and a2 store the following elements:

```
String[] a1 = {"Splinter", "Leo", "April", "Don", "Raph"};
String[] a2 = {"Krang", "Shredder", "Bebop"};
```

Then your method should return the new array `{"Splinter", "Shredder", "April", "oops", "oops"}`.

For full credit, do not modify the elements of *a1* or *a2*. Do not make any assumptions about the length of *a1* or *a2* or the length of the strings. You may assume that neither array is `null` and no element of either array is `null`.

7. Array Programming

Write a static method named **`evenBeforeOdd`** that accepts an array of integers as a parameter and rearranges its elements so that all even values appear before all odds. For example, if the following array is passed to your method:

```
int[] numbers = {5, 2, 4, 9, 3, 6, 2, 1, 11, 1, 10, 4, 7, 3};
```

Then after the method has been called, one acceptable ordering of the elements would be:

```
{4, 2, 4, 10, 2, 6, 3, 1, 11, 1, 9, 5, 7, 3}
```

The exact order of the elements does not matter, so long as all even values appear before all odd values. For example, the following would also be an acceptable ordering:

```
{2, 2, 4, 4, 6, 10, 1, 1, 3, 3, 5, 7, 9, 11}
```

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 no even elements or no odd elements. You may assume that the array is 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. You will lose points if you use `Arrays.sort` in your solution.

Hint: Look for elements that are at inappropriate places in the array and move them to better locations.

8. Critters

Write a critter class **`Tigger`** along with its movement and eating behavior. All unspecified aspects of `Tigger` use the default behavior. Write the complete class with any fields, constructors, etc. necessary to implement the behavior.

Bouncing is what Tiggers do best! The `Tigger`'s movement is to bounce up and down to increasingly large heights. A `Tigger` object is passed an integer when it is constructed that represents his initial bounce height. (You may assume that the bounce height is at least 1.) Whatever bounce height is passed, he will move that many steps NORTH, then that many steps SOUTH, then repeat for a bounce height 1 larger. For example, a `new Tigger(4)` will move NORTH **4** times, then SOUTH 4 times, then NORTH **5** times, then SOUTH 5 times, then NORTH **6** times, then SOUTH 6 times, and so on.

When a `Tigger` finds food, he eats it and completely starts over his bouncing behavior. That is, he starts over on a bounce whose height is equal to the initial bounce height with which the `Tigger` was constructed. For example, the following would be a sequence of moves for a `new Tigger(2)`. Notice how he starts over every time he eats:

- N,N, S,S, N,N,N, S,S,S, N,N,N,N, S,S,S,S, N *(eats food)*, **N,N, S,S**, N,N,N, S,S,S, N,N,N,N, ...

9. Classes and Objects

Suppose that you are provided with a pre-written class `ClockTime` as described at right. (This was shown on one of our practice exams, except with the `advance` method from that exam added.) Assume that the fields, constructor, and methods shown are implemented. You may refer to them or use them in solving this problem.

```
// A ClockTime object represents
// an hour:minute time during
// the day or night, such as
// 10:45 AM or 6:27 PM.

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


    // Constructs a new time for
    // the given hour/minute
    public ClockTime(int h,
            int m, String ap)

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

    // returns String for time;
    // example: "6:27 PM"
    public String toString()

    // advances this ClockTime
    // by the given # of minutes
    public void advance(int m)


    // your method would go here


}
```

Write an instance method named **`isWorkTime`** that will be placed inside the `ClockTime` class to become a part of each `ClockTime` object's behavior. The `isWorkTime` method returns `true` if the `ClockTime` object represents a time during the normal "work day" from 9:00 AM to 5:00 PM, inclusive. Any times outside that range would cause the method to return a result of `false`.

For example, if the following object is declared in client code:

```
ClockTime t1 = new ClockTime(3, 27, "PM");
```

The following call to your method would return `true`:

```
if (t1.isWorkTime()) {        // true
```

Here are some other objects. Their results when used with your method are shown at right in comments:

```
ClockTime t2 = new ClockTime(12, 45, "AM"); //false
ClockTime t3 = new ClockTime( 6, 02, "AM"); //false
ClockTime t4 = new ClockTime( 8, 59, "AM"); //false
ClockTime t5 = new ClockTime( 9, 00, "AM"); //true
ClockTime t6 = new ClockTime(11, 38, "AM"); //true
ClockTime t7 = new ClockTime(12, 53, "PM"); //true
ClockTime t8 = new ClockTime( 3, 15, "PM"); //true
ClockTime t9 = new ClockTime( 4, 59, "PM"); //true
ClockTime ta = new ClockTime( 5, 00, "PM"); //true
ClockTime tb = new ClockTime( 5, 01, "PM"); //false
ClockTime tc = new ClockTime( 8, 30, "PM"); //false
ClockTime td = new ClockTime(11, 59, "PM"); //false
```

Your method should not modify the state of the `ClockTime` object. Assume that the state of the `ClockTime` object is valid at the start of the call and that the `amPm` field stores either `"AM"` or `"PM"`.

X. Extra Credit (+1 point)

Draw a picture of what your TA would look like if he/she chose to fully embrace hip-hop culture. Make sure to write your TA's name (and/or hip-hop nickname) on your picture so we know who it is!

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