CSE 142, Autumn 2010
Final Exam
Wednesday, December 15, 2010

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	Array Programming		15
6	Array Programming		15
7	Critters		15
8	Classes and Objects		10
TOTAL	**Total Points**		**100**

1. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = a.length - 1; i >= 1; i--) {
        if (a[i] > a[i - 1] + 10) {
            a[i - 1] = a[i - 1] + 5;
        }
    }
}
```

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, 99};` `arrayMystery(a1);`	_____
`int[] a2 = {6, 18, 4, 25};` `arrayMystery(a2);`	_____
`int[] a3 = {5, 10, 20, 35, 40};` `arrayMystery(a3);`	_____
`int[] a4 = {-20, 20, 26, 32, 50, 3};` `arrayMystery(a4);`	_____
`int[] a5 = {5, 10, 16, 21, 27, 40, 55};` `arrayMystery(a5);`	_____

2. Reference Semantics Mystery

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

```
public class Location {
    int lat;
    int lng;

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

    public String toString() {
        return lat + "," + lng;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int a = 2;
        int b = 7;
        Location home = new Location(47, 122);           // Seattle!

        mystery(a, b, home);
        System.out.println(a + " " + b + " " + home);

        home.lng += b;
        a = a * 2;

        mystery(a, b, home);
        System.out.println(a + " " + b + " " + home);
    }

    public static void mystery(int a, int b, Location loc) {
        loc.lat = loc.lng - 10;
        b--;

        System.out.println(a + " " + b + " " + loc);
    }
}
```

3. Inheritance Mystery

Assume that the following four classes have been defined:

```
public class HeMan extends Skeletor {
    public void attack() {
        super.attack();
        System.out.print("heman-A    ");
    }

    public String toString() {
        return "heman!";
    }
}

public class SheRa {
    public void attack() {
        System.out.print("shera-A    ");
    }

    public void train() {
        System.out.print("shera-T    ");
    }

    public String toString() {
        return "shera!";
    }
}
```

```
public class BattleCat extends HeMan {
    public void attack() {
        System.out.print("battlecat-A    ");
    }

    public void train() {
        attack();
        System.out.print("battlecat-T    ");
    }
}

public class Skeletor extends SheRa {
    public void train() {
        System.out.print("skeletor-T    ");
        attack();
    }
}
```

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

```
SheRa[] superheros = { new Skeletor(), new HeMan(), new SheRa(), new BattleCat() };

for (int i = 0; i < superheros.length; i++) {
    superheros[i].train();
    System.out.println();
    superheros[i].attack();
    System.out.println();
    System.out.println(superheros[i]);
    System.out.println();
}
```

4. File Processing

Write a static method named **`frequentFlier`** that accepts as its parameter a `Scanner` for an input file. The data in the `Scanner` represents all of the flights a person has taken. Your method should compute and return an integer representing the total frequent flyer miles the person has earned. The input consists of pairs of tokens where each pair begins with the type of ticket that the person bought ("coach", "firstclass", or "discount", case-sensitively) and is followed by the number of miles of the flight. The rules for awarding frequent flier miles are:

- 1 frequent flyer mile is earned for each mile traveled in coach.
- 2 frequent flyer miles are earned for each mile traveled in first class.
- Zero frequent flyer miles are earned on a discounted flight.

For example, if a `Scanner` named `input1` refers to an input file containing the following text:

```
coach 1500   firstclass 2000   discount 900   coach 3500
```

In this example, the person earns 1500 frequent flyer miles for the first flight, 4000 frequent flyer miles for the second flight, 0 frequent flyer miles for the third flight, and 3500 frequent flyer miles for the fourth flight. Therefore the call of `frequentFlier(input1)` should return `9000` (a total of 1500 + 2*2000 + 3500).

The input might span multiple lines and might have different spacing between tokens. You are to process all tokens on all lines together. For example, if a `Scanner` named `input2` refers to an input file containing the following text:

```
firstclass        5000  coach         1500         coach
100 firstclass

2000    discount 300
```

Then the call of `frequentFlier(input2)` should return `15600` (a total of 2*5000 + 1500 + 100 + 2*2000).

You may assume that the input file exists and follows the format above; that the file has an even number of tokens and contains at least 1 pair of tokens; that every other token is an integer; and that the other tokens are valid ticket types.

5. Array Programming

Write a static method named **`countCommon`** that accepts three arrays of strings as parameters and returns an integer count of how many indexes store exactly the same string in all three arrays. For example, if the arrays are:

```
// index         0     1      2      3      4       5         6         7
String[] a1 = {"hi", "Ed",  "how", "are", "you",  "folks", "DoIng", "today?"};
String[] a2 = {"hi", "Bob", "how", "are", "YOUR", "kids",  "doing", "today?"};
String[] a3 = {"hi", "you", "how", "are", "you",  "guys",  "DOING", "today?"};
```

Then the call of `countCommon(a1, a2, a3)` should return `4` because indexes 0, 2, 3, and 7 store exactly the same string in all three arrays. Indexes 1, 4, 5, and 6 do not. (Index 6 differs by capitalization.)

The arrays might not be the same length. An index must be within the bounds of all three arrays to be considered. For example, given the arrays below, the call of `countCommon(a4, a5, a6)` should return `3` because all three arrays store the same values at indexes 0, 2, and 5:

```
// index         0     1      2      3      4      5          6        7
String[] a4 = {"hi", "Ed",  "how", "ARE", "you", "Doing", "I'm",  "fine"};
String[] a5 = {"hi", "Bob", "how", "are", "YOU", "Doing"};
String[] a6 = {"hi", "you", "how", "is",  "you", "Doing", "this", "fine", "day?"};
```

For full credit, do not modify the elements of the arrays passed in. Do not make any assumptions about the length of the arrays or the length of the strings stored in them. You may assume that none of the arrays or elements are `null`.

6. Array Programming

Write a static method named **`delta`** that accepts an array of integers as a parameter and returns a new array formed by inserting between each pair of values the difference between those values. For example, given this array:

```
int[] numbers = {3, 8, 15};
```

The call of `delta(numbers)` should return the following array (new elements are bolded):

```
{3, 5, 8, 7, 15}
```

In this example, 5 is inserted between 3 and 8 because (8 - 3) is 5, and 7 is inserted between 8 and 15 because (15 - 8) is 7. The difference should always be computed as the second value minus the first, so you can get negative values. For example, given the following array:

```
int[] numbers2 = {3, 8, 2, 5, 1, 9};
```

The call of `delta(numbers2)` should return the array:

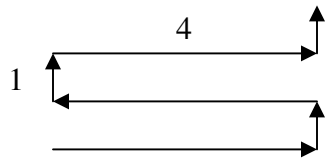
```
{3, 5, 8, -6, 2, 3, 5, -4, 1, 8, 9}
```

You may assume that the array passed is not `null`. If the array is empty, return an empty array.

7. Critters

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

A `Daisy` object generally moves in an upward zigzag pattern, going east four times, north once, west four times, north once, and repeating:



- E, E, E, E, N, W, W, W, W, N, E, E, E, E, N, W, W, W, W, N, ...

When a `Daisy` finds food, she eats it. The eating of food causes the `Daisy` to slow down for a while, so that each of the next 3 regular moves are preceded by a move to the center. For example:

- E, E, E, E, N, W, W *(eats)*, **C**, W, **C**, W, **C**, N, E, E, E, E, N *(eats)*, **C**, W, **C**, W, **C**, W, W, N, ...

If a `Daisy` eats food again before she has finished making her "slowed down" 3 moves, the slow-down counter goes back up to 3 again:

- E, E, E, E, N, W, W *(eats)*, **C**, W, **C**, W *(eats)*, **C**, N, **C**, E, **C**, E, E, E, N, W, W, W, W, N, ...

8. Classes and Objects

Suppose you have the class `Date` at right. (This is the same `Date` class from your homework. Only the headings are shown.)

Write an instance method **`isBefore`** to be placed inside the `Date` class. The method accepts another `Date` as a parameter and returns `true` if this date comes earlier in the year than the date passed in. If this date comes later in the year than the date passed, or if the two dates are the same, the method returns `false`.

For example, if these `Date` objects are declared in client code:

```
Date jan01 = new Date( 1,  1);
Date jul04 = new Date( 7,  4);
Date jul22 = new Date( 7, 22);
Date sep19 = new Date( 9, 19);
Date dec03 = new Date(12,  3);
```

The following calls should return `true`:

```
jul04.isBefore(jul22)
sep19.isBefore(dec03)
jan01.isBefore(sep19)
```

The following calls should return `false`:

```
dec03.isBefore(jul22)
jul22.isBefore(jul04)
sep19.isBefore(sep19)
```

Your method should not modify the state of either `Date` object, such as by modifying their `day` or `month` fields.

```
// this class ignores leap years
public class Date {
  private int month;
  private int day;

  // constructs the given month/day
  public Date(int m, int d)

  // returns the day
  public int getDay()

  // returns the month
  public int getMonth()

  // returns the number of days
  // in this date's month.
  public int daysInMonth()

  // compares dates (true if same)
  public boolean equals(Date d)

  // modifies this date's state
  // forward in time by 1 day,
  // wrapping month/year if needed
  public void nextDay()

  // set month/date to given values
  public void setDate(int m, int d)

  // your method would go here
}
```