CS 312 – Final – Fall 2014

Your Name___________________________________

Your UTEID _________________________________

Problem Number	Topic	Points Possible	Points Off
1	expressions	12	
2	code tracing	15	
3	program logic	15	
4	return methods	15	
5	arrays 1	15	
6	critters	15	
7	arrays 2	15	
8	file processing	15	
9	2d arrays	15	
TOTAL POINTS OFF:			
EXTRA CREDIT (last page):			-
SCORE OUT OF 132:			

Instructions:

1. Please turn off your cell phones
2. You have 3 hours to complete the test.
3. You may not use a calculator or any other electronic device.
4. When code is required, write Java code.
5. Ensure you follow the restrictions of the question.
6. You are allowed to add helper methods. (break problem up)
7. Style is not evaluated when grading.
8. The proctors will not answer questions.
   If you believe there is an error or a question is ambiguous, state your assumptions and answer based on those assumptions.
9. When you finish, show the proctor your UTID, turn in the exam and all scratch paper.

1. Short Answer 1 - Expressions. 1 point each, 12 points total.

For each Java expression in the left hand column, indicate the resulting value in the right hand column.

You must show a value of the appropriate type. For example, 7.0 rather than 7 for a double, "7" instead of 7 for a String, and '7' for a char.

Answers that do not indicate the data type correctly are wrong.

A. `37 % 10 + 10 % 37` _______________

B. `3 * 5 + 30 / 6` _______________

C. `10 / 4 + 3` _______________

D. `2.0 + 3 / 2 - 6 / 1.5` _______________

E. `(int) (1.9 * 2)` _______________

F. `3 * 6 + 3 + "314"` _______________

G. `2 + 3 + "bit" + 2 + 14` _______________

H. `"Gates:Dell".substring(2, 6)` _______________

x is an int variable initialized previously

I. `x == 5 && x == -5` _______________

J. `"cache".charAt(3)` _______________

K. `!(15 % 3 == 0 && 15 % 5 ==0)` _______________

L. `(char) ('a' + 10 / 3)` _______________

2. Code Tracing - 1 point each, 15 points total. For each code snippet state the exact output to the screen. Assume all necessary imports have been made.
If the snippet contains a syntax error or other compile error, answer COMPILE ERROR.
If the snippet results in a runtime error or exception, answer RUNTIME ERROR.

```
A.
int xa = 3;
int ya = 3 + a(xa);
System.out.print(xa + " " + ya + " " + a(ya));

public static int a(int x) {
     int z = (x - 2) * 2;
     System.out.print(z + " ");
     return z;
}
```

```
B.
int xb = 2;
int yb = 5;
b(xb, yb);
System.out.print(xb + " " + yb);

public static void b(int xb, int yb) {
     xb += 3;
     yb--;
     System.out.print(xb + " " + yb);
}
```

```
C.
int[] cd = {4, -2, 2};
c(cd);
System.out.print(Arrays.toString(cd));

public static void c(int[] data) {
     data[0] += 2;
     data[data.length - 1] -= data[data.length - 2];
}
```

```
D.
System.out.print(recMystery(3));

public static int recMystery(int y) {
     if(y == 0)
          return 1;
     return recMystery(y - 1) + (y * 2);
}
```

```
E.
int[] de = {5, 3};
e(de);
System.out.print(Arrays.toString(de));

public static void e(int[] de) {
     de[1]++;
     de = new int[3];
     de[1] = de.length;
     System.out.print(" " + Arrays.toString(de));

}
```

```
F.
ArrayList<String> af = new ArrayList<String>();
af.add("A");
af.add("F");
af.add(0, "B");
af.add(2, "C");
af.add("A");
af.remove(1);
System.out.print(af.toString());
```

For the rest of the short answer questions consider these classes:

```
public class Room {
     public int size() { return 20; }
     public boolean hasScreen() { return false; }
     public String toString() {return "s: " + size(); }
}

public class Classroom extends Room {
     public int size() { return 100; }
     public boolean hasScreen() {return true; }
}

public class MeetingRoom extends Room {
     public int size() { return 10; }
     public String getName() { return "meet"; }
}
```

G. Consider the following statements. For each, answer if it is legal or if it causes a syntax error.

```
Object obje = new MeetingRoom();

Classroom ce = new Room();
```

`H.` Consider the following statements. For each, answer if it is legal or if it causes a syntax error.

```
Room rh = new MeetingRoom();

MeetingRoom mh = new Classroom();
```

For each code snippet state the exact output to the screen.

```
I.
Room ri1 = new Room();
Room ri2 = new Room();
System.out.print(ri1 == ri2);
```

```
J.
MeetingRoom mj1 = new MeetingRoom();
MeetingRoom mj2 = new MeetingRoom();
System.out.print(mj1.equals(mj2));
```

```
K.
MeetingRoom mk1 = new MeetingRoom();
System.out.print(mk1.hasScreen() + " " + mk1.getName());
```

```
L.
Room rl = new MeetingRoom();
System.out.print(rl.size() + " " + rl.getName());
```

```
M.
Room rm = new Classroom();
System.out.print(rm.size() + " " + rm);
```

```
N.
Classroom cn = new Classroom();
System.out.print(cn.size() + " " + cn.hasScreen());
```

```
O.
Room ro = new Room();
Classroom co = new Classroom();
MeetingRoom mo = new MeetingRoom();
int xo = ro.size() + co.size() + mo.size();
System.out.print(xo);
```

3. Program Logic - 15 points. Consider the following method. For each of the five points labeled by comments and each of the assertions in the table, write whether the assertion is ***always*** true, ***sometimes*** true, or ***never*** true at that point in the code.
Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N.

```
public static int assertionPracticeFinal(int num){
    int i = 2;
    int count = 0;
    int temp = 0;
    // POINT A
    while(i <= num) {
        if(num % i == 0) {
            // POINT B
            temp = (int) Math.sqrt(i);
            if(temp * temp == i) {
                count++;
                // POINT C
            }
        }
        i++;
        // POINT D
    }
    // Point E
    return count;
}
```

Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N.

	count == 0	temp * temp == i	i <= num
Point A			
Point B			
Point C			
POINT D			
POINT E			

4. Return methods - 15 Points. Write the method `limitChars`.
The method has two parameters: a `String str` and an `int limit`.

The method returns a new `String`. The new `String` is the same as the parameter `str` except a given character can appear no more than `limit` times in the result. The relative order of characters in the resulting `String` is the same as the original `String`.

Examples:
```
limitChars("", 3)-> returns ""
limitChars("AAAAA", 3)-> returns "AAA"
limitChars("AABBAABBC", 3)-> returns "AABBABC"
limitChars("ABBACCDCCACAB-B", 3)-> returns "ABBACCDCAB-"
limitChars("AAAAA", 1)-> returns "A"
limitChars("AABBAABBC", 1)-> returns "ABC"
limitChars("ABBACCDCCACAB-B", 1)-> returns "ABCD-"
limitChars("AAAAA", 5)-> returns "AAAAA"
limitChars("AABBAABBC", 5)-> returns "AABBAABBC"
limitChars("ABBACCDCCACAB-B", 5)-> returns "ABBACCDCCACAB-B"
```

You may use the `length` and `charAt` methods from the `String` class and `String` concatenation. Do not use any other Java classes or methods.

Assume `limit > 0` and `str != null`.

```
public static String limitChars(String str, int limit) {
```

5. Arrays 1 - 15 points. Write the method `minMaxResult`. The method has two parameters. Both parameters are arrays of `int`s. The method creates and returns an array of `int`s equal in length to the longer of the two arrays passed as parameters.

Elements in even numbered indices in the resulting array equal the minimum value from the corresponding element in the two parameters. Conversely, elements in odd numbered indices in the resulting array are equal to the maximum value from the corresponding element in the two parameters.

Examples:
```
minMaxResult({}, {2, 4, 6}) -> returns {2, 4, 6}
minMaxResult({-1, 5, 10}, {2, 4, 6}) -> returns {-1, 5, 6}
minMaxResult({-1, 10}, {2, 4, 6, -5}) -> returns {-1, 10, 6, -5}
minMaxResult({-1, -1, -5, 3}, {-1, -1, -5}) -> returns {-1, -1, -5, 3}
minMaxResult({}, {}) -> returns {}
minMaxResult({},{10}) -> returns {10}
```

You may not use any other Java class or methods except native arrays and methods from the Math class.

Assume neither parameter equals `null`.

Complete the following method:
```
public static int[] minMaxResult(int[] a1, int[] a2) {
```

Complete minMaxResult on the next page.

```
// a1 != null, a2 != null
public static int[] minMaxResult(int[] a1, int[] a2) {
```

6. Critters - 15 Points. Implement a complete `Squirrel` class from the Critters assignment.

constructor	`public Squirrel(int legLength)` - Assume `legLength > 0`
fight behavior	if Eastbound `SCRATCH`, if Westbound `POUNCE`
color	`Color.GRAY`
movement behavior	Squirrel's move back and forth. A squirrel will move `legLength` steps to the East, then `legLength` steps to the West, then `leglength` steps to the East, then `legLength` steps to the West, and so forth. If a Squirrel's **next** move would be to the East it is said to be an Eastbound Squirrel. If its **next** move would be to the West it is said to be a Westbound squirrel.
eating behavior	always `false` for Eastbound Squirrels, always `true` for Westbound Squirrels.
toString	`"}"` if Eastbound, `"{"` if Westbound

Recall the Critter class:

```
public abstract class Critter {

     public boolean eat() {  return false; }

     public Attack fight(String opponent) {
          return Attack.FORFEIT;
     }

     public Color getColor() { return Color.BLACK; }

     public Direction getMove() { return Direction.CENTER; }

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

     // constants for directions
     public static enum Direction {
          NORTH, SOUTH, EAST, WEST, CENTER
     };

     // constants for fighting
     public static enum Attack {
          ROAR, POUNCE, SCRATCH, FORFEIT
     };
```

Write the complete Squirrel class based on the above specification on the next page.

Include the class header and instance variables.

```
// Complete the Squirrel class below.
```

7. Arrays - 15 Points Complete a method `superStretch` that returns a stretched version of an array of `int`s.

The first element in the original array occurs one time in the resulting array.
The second element in the original array occurs two times in the resulting array.
The third element in the original array occurs three times in the resulting array, and so forth.
The relative order of the original items is unchanged.

Examples:

```
superStretch({}) -> returns {}
superStretch{5}) -> returns {5}
superStretch{5, 7}) -> returns {5, 7, 7}
superStretch{5, 7, 4}) -> returns {5, 7, 7, 4, 4, 4}
superStretch{5, 7, 4, 3}) -> returns {5, 7, 7, 4, 4, 4, 3, 3, 3, 3}
```

Recall 1 + 2 + 3 + … + (N - 2) + (N - 1) + N = (N + 1) * N / 2

You may assume the parameter does not equal `null`.

Do not use any other Java classes or methods except native arrays.

Complete the superStretch method on the next page.

```
// data != null
public static int[] superStretch(int[] data) {
```

8. File Processing - 15 Points. Write a method named `lineAverages` that accepts as a parameter a `Scanner` connected to a file. The method prints out the average of the `int` tokens on each line.

If a line does not contain any int tokens the method prints out `NONE` for that line.

For example if the `Scanner` were connect to the following file:

```
5 12 -12 5
cat 5 cat 12.5 10
cat dog

12.5 6 what?? 6.5 0 0  ??67
```

The method would produce the following output:

```
Line 1 average: 2.5
Line 2 average: 7.5
Line 3 average: NONE
Line 4 average: NONE
Line 5 average: 2.0
```

Tokens that cannot be converted to `int`s are ignored.

You may use methods from the Java `Scanner` and `String` classes.
You may not use any other Java classes or objects.

Complete the method on the next page.

```
// input != null, input already connected to File
public static void lineAverages(Scanner input) {
```

9. 2d Arrays - 15 points. Write a method `dropSpecialChecker` that drops a checker into a connect four board.

The `dropSpecialChecker` method accepts a 2d array of `char`s, an `int` indicting in which column to drop the checker, and a `boolean` indicating if the checker is red.

The method moves the check down to the lowest open spot in the column, just like regular connect 4.

The twist is that after the checker is dropped, the method swaps the color of every checker to the right of the dropped checker OR along the diagonal to the upper - right of the dropped checker.
The method randomly picks which direction it uses and each direction has a 50% chance of being chosen.

Consider the following two examples:

Initial Board 1

```
. . . . . . .
. . . . . . r
r . . . . . b
b . . . r . r
r . . . b r b
b r b r b r b
```

Initial Board 2

```
. . . . . . .
. . . . . . r
r . . . . . b
b . . . r . r
r . . . b r b
b r b r b r b
```

Final board after dropping a black checker in column 3 (0 based indexing) and swapping colors to the right.

```
. . . . . . .
. . . . . . r
r . . . . . b
b . . . r . r
r . . b r b r
b r b r b r b
```

Final board after dropping a black checker in column 3 (0 based indexing) and swapping colors along the diagonal to the upper-right.

```
. . . . . . .
. . . . . . b
r . . . . . b
b . . . b . r
r . . b b r b
b r b r b r b
```

Your method must place the checker in the proper row, determine which direction to swap colors (to the right or to the upper-right), and swap the color of every affected checker.

The board stores `'r'` for red checkers, `'b'` for black checkers, and `'.'` for open spaces.

You may assume the indicated column is in bounds.

You may assume the 2d array of chars is rectangular, same number of columns in every row.

You may assume there is at least one row open in the given column.

You may use the Math.random method.

You may not use any other Java classes or methods in your answer.

```
public static void dropSpecialChecker(char[][] board, int column,
                                      boolean redChecker) {
```

Extra Credit - 1 point each, 3 points total:

In class on the Wednesday before Thanksgiving we discussed image processing techniques such as those used in Photoshop and Instagram.

What were the three filters we implemented in class?

1.

2.

3.