CS 312 – Final – Fall 2015

Your Name: SOLUTION SOLUTION SOLUTION SOLUTION

Your UTEID _____

Problem Number	Topic	Points Possible	Points Off
1	expressions	10	
2	program logic	15	
3	code tracing	15	
4	Strings	15	
5	arrays 1	15	
6	critters	15	
7	arrays 2	15	
8	file processing	15	
9	2d arrays	15	
TOTAL POINTS OFF:			
SCORE OUT OF 130:			

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. 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.
8. When you finish, show the proctor your UTID, turn in the exam and all scratch paper.

1. Short Answer 1 - Expressions. 1 point each, 10 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.	`17 % 8 * 2 + 8 % 17`	10
B.	`2 * 5 + 17 / 5`	13
C.	`2.5 * 3 + 10 / 8`	8.5
D.	`(int) (3.117 * 5)`	15
E.	`(double) (13 / 5 + 10 / 20)`	2.0
F.	`"CS" + 4 * 5 + 6`	"CS206"
G.	`"121" + "35" + "M"`	"12135M"
H.	`"visual_basic".substring(3, 8)`	"ual_b"
I.	`!(5 * 2 < 10 || 3 - 2 == 2 - 3)`	true
J.	`"secondary".toUpperCase().charAt(5)`	'D'

2. 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 x = 0;
    int y = 0;
    // POINT A

    while (num > 0) {
        // POINT B

        y = num % 10;

        if(y % 3 == 0) {
            x++;
        } else {
            x = 0;
            // POINT C
        }

        num = num / 10;
        // POINT D
    }
    // Point E
    return x;
}
```

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

	num != 0	y % 3 == 0	x > 0
Point A	**S**	**A**	**N**
Point B	**A**	**S**	**S**
Point C	**A**	**N**	**N**
POINT D	**S**	**S**	**S**
POINT E	**S**	**S**	**S**

3. Code Tracing - 1 point each, 15 points total. For each code snippet state the exact output to the screen. **Placed your answers to the right of the snippet.**
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.
If the snippet results in an infinite loop, answer INFINITE LOOP.

A.

```
int xa = 5;
int ya = 3;
int za = a(xa, ya);
System.out.print(xa + " " + ya + " " + za);

public static int a(int xa, int ya) {
     xa = xa * 2;
     ya--;
     return xa * ya;
}
```

5 3 20

B.

```
int xb = 4;
int yb = 5;
xb = b(yb, xb);
System.out.print(xb + " " + yb);

public static int b(int x, int y) {
     x /= 2;
     y *= 2;
     return x - y;
}
```

-6 5

C.

```
int[] cd = new int[4];
c(cd);
System.out.print(Arrays.toString(cd));

public static void c(int[] data) {
     data[0] = 5;
     data[data.length - 1] = data[0] + data[1] + data.length;
     data = new int[3];
     data[0] = 3;
     data[2] = 5;
}
```

[5, 0, 0, 9]

D.

```
int[] dd = {5, 3, 10, 4};
d(dd);
System.out.print(Arrays.toString(dd));

public static void d(int[] data) {
     data[2] += data[0];
     data[0] = data[1] * 2;
     data[3]--;
}
```

[6, 3, 15, 3]

E.

```
String[] ed = new String[5];
System.out.print(e(ed));

public static int e(String[] data) {
     int result = 0;
     for(int i = 0; i < data.length; i++) {
          result += data[i].length();
     }
     return result;
}
```

Runtime error or NullPointerException or Exception

F.

```
String[] fd = {"Cat", "", "cat", "dog", ""};
System.out.print(f(fd));

public static int f(String[] data) {
     int result = 0;
     for(String s: data) {
          result++;
          result += s.length();
     }
     return result;
}
```

14

G.

```
int xg = 3;
int yg = 5;
System.out.print(g(xg, yg) + " " + xg + " " + g(xg, xg));

public static int g(int x, int y) {
     x++;
     y--;
     System.out.print(x + y + " ");
     return x * y;

}
```

8 6 16 3 8

H.

```
ArrayList<String> list = new ArrayList<String>();
list.add("A");
list.add("C");
list.add("AA");
list.add(1, "B");
list.add(3, "D");
list.remove(2);
System.out.print(list);
```

[A, B, D, AA]

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

```
public class Book {
     private int pages;
     public Book(int p) { pages = p; }
     public int getPages() { return pages; }
     public int days() { return pages / 10; }
     public void addChapter() { pages += 10; }
     public String toString() { return "pages: " + pages; }
}

public class EBook extends Book {
     public EBook(int pages) { super(pages - 10); }
     public int days() { return getPages() / 20; };
}
```

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

```
Book bi = new Book();

Object obj = new Book(100);
```

syntax error
legal or valid or okay

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

```
EBook ej = new Book(100);

String sj = new EBook(110);
```

syntax error
syntax error

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

K.

```
Book b1 = new Book(100);
Book b2 = new Book(110);
b1.addChapter();
System.out.print(b1 == b2);
```

false

L.

```
Book b3 = new Book(100);
process(b3);
System.out.print(b3.getPages());

public static void process(Book b) {
     b.addChapter();
     b.addChapter();
     System.out.print(b.getPages() + " ");
     b.addChapter();
}
```

120 130

M.

```
EBook em = new EBook(110);
em.addChapter();
em.addChapter();
System.out.print(em.days());
```

6

N.

```
Book bn = new EBook(110);
System.out.print(bn.days());
```

5

O.

```
EBook eo = new EBook(60);
System.out.print(eo.toString());
```

pages: 50

4. Return methods - 15 Points. Write the method `clickerPoints`. The method accepts two parameters, both Strings.

The first String represents a student's responses to clicker questions. All characters in this `String` will be capital letters A, B, C, D, or E OR a dash, `-`. The capital letters indicated the student's answer for the question. A dash indicates the student did not provide an answer to the question.

The second `String` represents the correct answer to the clicker questions. All characters in this `String` will be capital letters A, B, C, D, or E or an asterisk, `*`. The capital letters indicate the correct answer for a question. An asterisk indicates **any** attempt at the question is considered correct.

The `String`s are parallel, meaning the correct answer for question 0 is at position 0 in the answers `String` and the student's response to question 0 is also at position 0, but in the response `String`.

The method returns the number of points the student earns. For this question students get 5 points for each correct answer and 2 points for each attempted answer that was not correct.

Additionally there are two ways to get bonus points. If a student has **attempted** every question they get 5 extra points. If a student has answered 80% or more of **all** questions asked correctly they get 10 extra points.

Examples. Incorrect attempts are bolded and underlined

```
Correct Answers = "DE**DECAAA"
Student Answers = "----------" -> 0 points, no attempts

Correct Answers = "DE**DECAAA"
Student Answers = "BAAAAAABBB" -> 31 points, 8 incorrect, 2 correct
                                  5 points for attempted every question

Correct Answers = "DE**DECAAA"
Student Answers = "DECCCCAAA" -> 59 points, 2 incorrect, 8 correct
                                 5 points for attempted every question
                                 10 points for 80% or more correct

Correct Answers = "DE**DECAAA"
Student Answers = "DE-CCCAAA" -> 39 points, 2 incorrect, 7 correct
```

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

You may assume the following about the parameters:

- neither parameter stores `null`
- the length of the parameters are equal
- the student response `String` only contains capital letters A, B, C, D, E and dashes
- the answer `String` only contains capital letters A, B, C, D, E and asterisks

```
// answers != null, responses != null,
// answers.length() == responses.length()
public static int clickerPoints (String answers, String
responses) {
    int correct = 0;
    int wrong = 0;
    for (int i = 0; i < answers.length(); i++) {
        char answer = answers.charAt(i);
        char response = responses.charAt(i);
        if (response != '-') {
            if (answer == response || answer == '*') {
                correct++;
            } else {
                wrong++;
            }
        }
    }
    int points = correct * 5 + wrong * 2;
    if (correct + wrong == answers.length()) {
        points += 5;
    }
    if (1.0 * correct / answers.length() >= 0.8) {
        points += 10;
    }
    return points;
}
```

Points:

loop through String correctly: 3

access characters from Strings correctly: 1

logic for correct answers right: 3

track attempts correctly (wrong or total): 2

correct logic for all attempted: 2

correct logic for attempted 80% or more: 2

calcualtion of points correct: 1

return correct answer: 1

5. Arrays 1 - 15 points. Write the method `insert`. The method has two parameters. The first parameter is an array of `int`s that is already sorted into ascending order. The array has at least one element.
The second parameter is a single `int`.

The method inserts the single `int` into the array so that the elements are still in sorted ascending order. The old last element in the array is overwritten, but is returned by the method

Examples:
`insert({-5, 6, 10}, 12)` returns 10, array becomes `{-5, 6, 12}`
`insert({-5, 6, 10}, -5)` returns 10, array becomes `{-5, -5, 6}`
`insert({-5, 6, 10}, -7)` returns 10, array becomes `{-7, -5, 6}`
`insert({5, 5, 5, 6, 6}, 6)` returns 6, array becomes `{5, 5, 5, 6, 6}`
`insert({5, 5, 5, 6, 6}, 5)` returns 6, array becomes `{5, 5, 5, 5, 6}`
`insert({5, 5, 5, 6, 6}, 0)` returns 6, array becomes `{0, 5, 5, 5, 6}`
`insert({5, 5, 5, 6, 6}, 15)` returns 6, array becomes `{5, 5, 5, 6, 15}`

You may not use any other built in Java class or methods except native arrays.

Assume the array of `int`s the parameter `data` refers to is not `null`, contains at least one element, and that the elements in the array are already sorted in ascending order.

Complete the following method:

```
public static int insert(int[] data, int newValue) {
```

Complete the method on the next page.

```
// data != null, data.length > 0, and
// elements in data are sorted in ascending order

public static int insert(int[] data, int newValue) {
    int index = data.length - 1;
    int result = data[index];
    data[index] = newValue;
    while(index > 0 && data[index] < data[index - 1]) {
        int temp = data[index - 1];
        data[index - 1] = data[index];
        data[index] = temp;
        index--;
    }
    return result;
}
```

Loop until done: 2

Logic for inserting based on comparison correct: 4

Logic for smallest / largest value correct: 4 (NO AIOBE possible)

Swap element correctly: 4

return old largest element: 1

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

constructor	`public Turtle(Direction dir)`
fight behavior	`POUNCE` if the opponent looks like a Stone Critter, "S". If the opponent does not look like a Stone Critter then `ROAR`, **if** the `eat` method would return `true` **if** called, `SCRATCH` otherwise.
color	Always returns `Color.GREEN`.
movement behavior	As you can imagine turtles are slow. Turtles alternate between moving the direction sent to the constructor and not moving by responding with the direction `CENTER`. The first time `getMove` is called turtles return the direction sent to the constructor. Turtles pick a random number between 5 and 10 inclusive when created. This is their wait period. The next "wait period" times the `getMove` method is called a turtle returns `CENTER`. After the wait period the turtle again returns the direction sent to the constructor, then responds "wait period" times again with `CENTER`, and so forth.
eating behavior	Returns `true` if the Turtle has moved in its given direction since last eating, `false` otherwise. Returns `false` for a Turtle that has never moved.
toString	Always returns "T".

```
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 Turtle class based on the above specification on the next page.

Include the class header and instance variables. Assume any necessary imports are already made.

You may use the Critter, Random, and Color classes, and the Attack andDirection enumerated types, but no other built in Java classes or methods.

```
public class Turtle extends Critter {

    private int waitPeriod;
    private int turnsWaited;
    private Direction direction_I_Move;
    private boolean iAmHungry;

    private static Random randNumGen = new Random();

    public Turtle(Direction d) {
        direction_I_Move = d;
        final int MIN_TURNS = 5;
        final int RANGE = 6;
        waitPeriod = randNumGen.nextInt(RANGE) + MIN_TURNS;
        turnsWaited = waitPeriod;
        // default value for iAmHungry is correct.
    }

    public boolean eat() {
        boolean result = iAmHungry;
        iAmHungry = false;
        return result;
    }

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

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

    public Attack fight(String opponent) {
        Attack result = Attack.SCRATCH;
        if (opponent.equals("S")) {
            result = Attack.POUNCE;
        } else if (iAmHungry) {
            result = Attack.ROAR;
        }
```

```
        return result;
    }

    public Direction getMove() {
        Direction result = Direction.CENTER;
        if (turnsWaited == waitPeriod) {
            result = direction_I_Move;
            turnsWaited = 0;
            iAmHungry = true;
        } else {
            turnsWaited++;
        }
        return result;
    }
}
```

Points:

header: 1

Constructor: 3

getMove: 3

fight: 3

eat: 3

toString: 1

getColor: 1

7. Arrays - 15 Points Complete a method `longestRunLength`. The method has two parameters, an array of `String` object variables and a `char`.

The method finds and returns the length of the longest consecutive run of `String`s in the array that contain the given character.

Examples. The longest runs of Strings that contain at least one instance of the target character are underlined for clarity.

`longestRunLength({"AA", "BB", "A", "BB"}, 'a') -> returns 0`
None of the `String`s in the array contain the character 'a', so the longest run length of `String`s that contain an 'a' is 0.

`longestRunLength({"AA", "BB", "A", "BB"}, 'A') -> returns 1`
Two of the `String`s in the array contain at least one `'A'`, but they are followed by `String`s that **do not** contain an `'A'` so the longest run length is 1.

`longestRunLength({"ABA", "BAC", "ABCA", "BB"}, 'A') -> returns 3`
The longest run of consecutive `String`s that contain at least one `'A'` is three.

`longestRunLength({}, 'A') -> returns 0`
Array of length 0.

`longestRunLength({"CB", "ABA", "BAC", null, "ABCA", "AaA", "BB"}, 'A') -> returns 2`
Two runs both equal to length 2. The array may contain `String` object variables that store `null` indicating they do not refer to any `String`.

`longestRunLength({"CBC", "BAC", "ABCA", "BAAB"}, 'A') -> returns 3`
The longest run of consecutive `String`s that contain at least one `'A'` is three.

`longestRunLength({null, null, null, null}, 'n') -> returns 0`
All `null`s.

`longestRunLength({"ABA", "BAC", "ABCA", "AA"}, 'A') -> returns 4`
Every element contains at least one `'A'`.

`longestRunLength({"CB", "AB", "AA", "ab", "BA", "A", "A", "CC"}, 'A') -> returns 3`

You may use native arrays and the following `String` methods:

```
int length(),
int indexOf(char ch) // returns first index of ch in this String or
                     // -1 if ch not present in this String.
```

```
// strs != null
// strs is not altered as a result of this method call
public static int longestRunLength(String[] strs, char target) {
    int max = 0;
    int currentRun = 0;
    for (int i = 0; i < strs.length; i++) {
        String current = strs[i];
        // does current String contain the target character?
        if (current != null && current.indexOf(target) != -1) {
            currentRun++;
            // has this run exceeded the best so far?
            // A little gacky. Just need to check when run over
            if (currentRun  > max) {
                max = currentRun;
            }
        } else {
            // no longer in a run
            currentRun = 0;
        }
    }
    return max;
}

public static int longestRunLengthSlow(String[] strs,
                                                    char target) {
    // Slow version and gacky
    int max = 0;
    for (int i = 0; i < strs.length; i++) {
       String current = strs[i];
        if (current != null && current.indexOf(target) != -1) {
            int run = 1;
            int index = i + 1;
            // index is in bounds, and next String not null, and
            // next String contains the target char.
            while(index < strs.length && strs[index] != null &&
                    strs[index].indexOf(target) != -1) {
                index++;
                run++;
            }
            if (run > max) {
                max = run;
            }
        }
    }
    return max;
```

Points:

track best run: 1

loop through array: 2

check String not null: 2

check String contains char correctly: 2

track current run correctly: 5

compare current run to best and update if necessary: 2

return correct answer: 1

8. File Processing - 15 Points. Write a method, `billTotal`, that calculates the total cost for a shopping trip at a local store. The items purchased on the trip appear in a file.
Each item consists of four elements with the following form:

<Item Name> <Discount Category> <Item Price> <Number of Items>

All tokens will be separated by white space.

<Item Name> will be a single token with the name of the item.

<Discount Category> will be a single token. If the token is GREEN the item's actual price is reduced by 20% off of the <Item Price>. If the token is RED the item's actual price is reduced by 50% off of the <Item Price>. Any other value means the item's price is not reduced.

<Item Price> is the price of the item. This will be a double with 2 digits after the decimal.

<Number of Items> is the number of items of this type being purchased. This shall be an integer greater than 0.

An example file is shown below

```
Ornaments RED 0.50 10
Tree GREEN 20.00 1
Paper        RED       2.50      2      Juice NONE
0.50
10

water_bottles

red

1.25

9
```

Note there is no guarantee on the formatting other than each item will have the four elements described above in the order shown. It is possible for an item to be split between lines, for blank lines to appear, and for the whitespace between tokens to vary.

In the example shown the method would return 37.25.
Ornaments: 10 * 0.50 * 0.50 = 2.50
Tree: 1 * 20.00 * 0.8 = 16.00
Paper: 2 * 2.50 * 0.5 = 2.50
Juice: 10 * 0.50 = 5.00
water_bottles: 9 * 1.25 = 11.25 (Case sensitive, red != RED so no discount)

The Scanner sent to the method is already connected to the data file.

You may us methods from the `Scanner` class, the `String` class (any methods you want) ,but no other built in Java methods or classes.

```
// input != null, input is already connected to File that only
// contains properly formatted items
public static double billTotal(Scanner input) {
    double result = 0.0;
    while (input.hasNext()) {
        // don't need the name;
        input.next();
        String discountCode = input.next();
        double price = input.nextDouble();
        int quantity = input.nextInt();
        double total = price * quantity;
        if (discountCode.equals("RED")) {
            total *= 0.5; // 50% off
        } else if (discountCode.equals("GREEN")) {
            total *= 0.8; // 20% off
        }
        result += total;
    }
    return result;
}
```

Points:

track total: 1
loop while still tokens: 3
skip item name: 1
read discount code: 1
read price: 1
read quantity: 1
determine discount correctly: 5 (== is -3)
add current item's total to running total: 1
return correct answer: 1

9. 2d Arrays - 15 points. Write a method `averageOfNeighborhood` that determines the average of the elements in a 2d array of `int`s centered at a given location and including a neighborhood centered around the given location.

Consider the following 2d array of ints

1	0	10	4	0	7	20
11	5	5	1	0	3	7
-1	12	4	12	0	5	0
5	4	5	13	**12**	0	0
10	2	-2	1	5	5	0
3	1	3	5	7	3	5
3	3	5	5	9	11	-5

The box shows the neighborhood if the center cell is at row 3, column 4 and the neighborhood size is 2 rows and 1 column. The center cell is bolded and underlined.
The neighborhood is outlined with the dashed box.

With a value of 2 rows the neighborhood includes the 2 rows above and below the row with the center cell, along with the row that contains the center cell. With a value of 1 column the neighborhood includes the columns to the left and right of the column with the center cell, along with the column that contains the center cell.

In the example above the method would return 4.8.
(1 + 0 + 3 + 12 + 0 + 5 + 13 + 12 + 0 + 1 + 5 + 5 + 5 + 7 + 3) / 15 = 72 / 15 = 4.8

Consider another example with the center cell at row 1 and column 1.
The neighborhood size is 1 row and 2 columns.

1	0	10	4	0	7	20
11	**5**	5	1	0	3	7
-2	12	4	12	0	5	0
5	4	5	13	12	0	0
10	2	-2	1	5	5	0
3	1	3	5	7	3	5
3	3	5	5	9	11	-5

In this example the neighborhood is cut short because some cells are out of bounds. The method would return 5.25. (1 + 0 + 10 + 4+ 11 + 5 + 5 + 1 + -2 + 12 + 4 + 12) / 12 = 63 / 12 = 5.25.

- You may assume the 2d array of `int`s is rectangular, same number of columns in every row.
- You may assume the coordinates of the center cell are in bounds.
- You may not use any other built in Java classes or methods in your answer.

```
// mat != null, mat is a rectangular matrix, mat.length > 0,
// mat[0].length > 0. cenRow and cenCol are in bounds.
// rows >= 0, cols >= 0
public static double averageOfNeighborhood(int[][] mat,
            int cenRow, int cenCol, int rows, int cols) {

    double total = 0.0;
    int count = 0;
    int startRow = cenRow - rows;
    if (startRow < 0)
        startRow = 0;
    int startCol = cenCol - cols;
    if (startCol < 0)
        startCol = 0;
    int endRow = cenRow + rows;
    if (endRow >= mat.length)
        endRow = mat.length - 1;
    int endCol = cenCol + cols;
    if (endCol >= mat[0].length)
        endCol = mat[0].length;
    for (int r = startRow; r <= endRow; r++) {
       for (int c = startCol; c <= endCol; c++){
            total += mat[r][c];
            count++;
        }
    }
    return total / count;
}
```

Points:

track total correctly: 2

track number of cells correctly: 2

loop through all cells correctly: 6

avoid AIOBE correctly: 4 (does NOT have to be separate method)

calculate and return correct average: 1

```
// mat != null, mat is a rectangular matrix, mat.length > 0,
```

```
// mat[0].length > 0. cenRow and cenCol are in bounds.
// rows >= 0, cols >= 0
public static double averageOfNeighborhood(int[][] mat,
            int cenRow, int cenCol, int rows, int cols) {

   // ternary operator shortens code, not any more efficient
    double total = 0.0;
    int startRow = (cenRow - rows < 0) ? 0 : cenRow - rows;
    int startCol = (cenCol - cols < 0) ? 0 : cenCol - cols;
    int endRow = (cenRow + rows >= mat.length) ? mat.length
             : cenRow + rows;
    int endCol = (cenCol + cols >= mat[0].length)
             ? mat[0].length - 1 : cenCol + cols;

    for (int r = startRow; r <= endRow; r++) {
       for (int c = startCol; c <= endCol; c++){
             total += mat[r][c];
        }
    }
    int numCells = (endRow - startRow + 1)
                         * (endCol - startCol + 1);

    return total / numCells; // total is a double
}
```