CS 312 – Final –2013

Your Name SOLUTION SOLUTION SOLUTION

Your UTEID _____

Problem Number	Topic	Points Possible	Points Off
1	short answer 1	10	
2	ASN	15	
3	short answer 2	10	
4	arrays 1	12	
5	strings	14	
6	critters	19	
7	arrays 2	17	
8	methods	13	
9	2d arrays	20	
		TOTAL POINTS OFF:	
		SCORE OUT OF 130:	

Instructions:

1. Please turn off your cell phones
2. There are 9 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator or any other electronic device.
5. Please make your answers legible.
6. When code is required, write Java code.
7. Style is not evaluated when grading.
8. The proctors will not answer questions. If you feel a question is ambiguous, state your assumptions and answer based on those assumptions.

The exam is worth 130 points. Grades will be scaled to 400 for gradebook.

1. Short Answer 1 - Expressions. 1 point each, 10 points total.

For each Java expression in the left hand column, indicate its 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 and "7" instead of 7 for a String. Answers that do not indicate the data type correctly are wrong.

A. `3 * 4 + 6 / 3` **14**

B. `5 * 3 + 12612 / 1000` **27**

C. `5.0 + 3 / 2` **6.0**

D. `23 % 8` **7**

E. `8 % 23` **8**

F. `3 + "CA" + "DG"` **"3CADG"** -1 no quotes

G. `1 + 2 + "CS" + 1 + 2` **"3cs12"**

H. `12 * 12 > 100 && 0 != 1` **true**

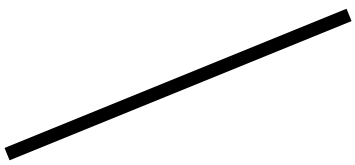
x is an int variable initialized previously

I. `x != 3 || x != 5` **true**

2. Program Logic - 15 points. Consider the following method. For each of the 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. If there is a forward slash in the cell, do not place an answer in the cell.

```
// assume data != null
public static void assertionPractice2(int[] data){
     final int LIMIT = data.length - 1;
     int i = 0;
     int j = i + 1;
     int m = 0;
     if(data.length >= 2) {
          // POINT A
          for(i = 0; i < LIMIT; i++) {
               m = i;
               for(j = i + 1; j < data.length; j++)
                    if(data[j] < data[i]) {
                         m = j;
                         // POINT B
                    }
               int t = data[i];
               data[i] = data[m];
               data[m] = t;
               // POINT C
          } // end of out for loop
     } // end of if(data.length >= 2)
     // POINT D
}
```

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

	j == i	data[i] is minimum value between index i and the end of the array	m == i	data.length >= 2
Point A	**N**	**S**	**A**	**A**
Point B	**N**	**N**	**N**	**A**
Point C	**N**	**A**	**S**	**A**
POINT D	**N**		**S**	**S**

3. Short Answer 2 - 1 point each, 10 points total. For each code snippet state the exact output to the screen.
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.

```
String[] names = new String[10];
System.out.print(names[5].length());
```

Runtime Error or Exception

B.

```
int[] data = {2, 2, 1, 7, 1, 2};
data[2] = data[0] + data[4] + data[2];
System.out.print(data[2] + " " + data.length);
```

4 6

C.

```
int[] data2 = new int[10];
int index = data2.length / 2;
System.out.print(data2[index * 4]);
```

Runtime Error or Exception

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

```
public class UTPerson {
     public String parking() { return "X"; }
     public String toString() {return "per " + parking(); }
}

public class Staff extends UTPerson {
     public String toString() { return "stf " + parking(); }
}

public class Faculty extends UTPerson {
     public String parking() { return "F"; }
     public String toString() { return "fac " + parking(); }
}

public class Student extends UTPerson {
     public String study() {return "study"; }
}

public class EveningStaff extends Staff {
     public String parking() { return "N"; }
}
```

D.

```
Student st1 = new Student();
System.out.print(st1.toString());
```
per X

E.

```
UTPerson ut2 = new Faculty();
System.out.print(ut2.toString());
```
fac F

F.

```
Staff st3 = new EveningStaff();
System.out.print(st3.toString());
```
stf N

G.

```
EveningStaff es4 = new UTPerson();
System.out.print(es4.toString());
```
Compile Error or Syntax Error

H.

```
Student st5 = new Student();
System.out.print(st5.study() + " " + st5.toString());
```

study per X

I.

```
Staff st6 = new Staff();
System.out.print(st6.toString());
```
stf X

J.

```
Object obj7 = new UTPerson();
System.out.print(obj7.toString());
```
per X

4. **Arrays 1 - 12 points.** Write a method that given an array of `int`s and a candidate value, returns `true` if the candidate value is the *majority element* in the array. A candidate value is the majority element in an array if **more than 50%** of the elements in the array equal the candidate value.
The method does not alter the array in any way. Examples:

```
array: [1], candidate 1, result -> true
array: [1], candidate 2, result -> false
array: [2, 1], candidate 1, result -> false (1 occurs 50% of the time,
                                        not more that 50% of the time)
array: [2, 1, 2, 3, 1], candidate 1, result -> false
array: [-1, 1, 1, 3, 1], candidate 1, result -> true
array: [-1, 1, 1, 3, 1], candidate 10, result -> false
```

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

```
// data.length >= 1, the array data is unchanged by this method
public static boolean isMajorityElement(int[] data, int candidate) {

     int count = 0;

     for(int i = 0; i < data.length; i++)
          if(data[i] == candidate;
               count++;

     return 1.0 * count / data.length > 0.5
}
```

11 points total:

counter variable: 1 point

loop correct: 3 points off by one error - 1,

if check with correct boolean condition: 3 points

increment count correctly: 1 point

calculate percent correctly: 3 points, -1 int div, -1 >= 0.5

return boolean: 1 point

5. Strings - 14 points. Write a method that prints all the *trigrams* in a `String` ignoring all non-letter characters in the `String`. Print out the letter-only trigrams in the order they appear in the `String`, one per line, to standard output.

A *trigram* consists of three consecutive characters in a `String`. For example the `String "compute"` contains the following trigrams: `"com", "omp", "mpu", "put", "ute".`

In this question you **must ignore any non-letter characters in the String**.
For example, the `String "t**he c!!-A  {}t"`
has the following letter-only trigrams: `"the", "hec", "ecA", "cAt".`
Print out all letter-only trigrams in the order they appear even if this means printing out the same trigram multiple times.
If the `String` does not contain any letter-only trigrams, then there is no output.

You may use the `String` class, the static `Character.isLetter(char ch)` method that returns `true` if `ch` is a letter, `false` otherwise, and `System.out.println()`. Do not use any other classes or methods in your answer.

```
public static void printLetterTrigrams(String src) {
      String lettersOnly = "";
      for(int i = 0; i < src.length(); i++) {
            char ch = src.charAt(i);
            if(Character.isLetter(ch))
                  lettersOnly += ch;

      final int NUM_TRIGRAMS = lettersOnly.length() - 2;

      for(int i = 0; i < NUM_TRIGRAMS; i++)
            System.out.println(lettersOnly.substring(i, i + 3);
}
```

```
14 points total:
letters only or temp String + init correctly: 2 points
loop with correct bounds: 3 points, off by one error - 1
check if current char is letter correctly: 2 points
add letters to temp String: 2 points
calculate number of trigrams or second loop bounds correct: 1 point
second loop correct: 1 point
print correct trigrams, substring: 3 points (-1 or -2 for minor / major errors)


// alt:

String tri = ""; // 2 points
for(int i = 0; i < src.length; i++) { // 3 points
      char ch = src.charAt(i);
      if(Character.isLetter(ch)) {     // 2 points
            tri += ch; // 2 points
            if(tri.length() == 3) {     // 2 points
                  System.out.println(tri);    // 2 points
                  tri = "";  // 1 point
            }
      }
}
```

6. Critters - 19 Points. Write a complete `Prowler` class that implements the `Critter` interface from assignment 10.

- The constructor has two parameters: an `int` that specifies the `Prowler`'s number of steps per leg and its `Color`.
- `Prowler`s always fight with `PAPER`.
- The `toString` method for `Prowler`s always returns "P".
- The `getColor` method returns the `Color` sent to the `Prowler`'s constructor.
- Prowlers generally move in an up and down pattern. They start by moving `NORTH` the number of steps specified in the constructor. For example, if the number of steps per leg is 5, the `Prowler` will respond with `NORTH` for the first 5 calls to `getMove`. A `Prowler` then reverses its direction and moves `SOUTH` the same number of steps before reversing direction to `NORTH`, and so forth.

  There is special case for movement. If the `CritterInfo` object indicates there is another `Critter` of any type `EAST` or `WEST` of the `Prowler`, then the `Prowler` moves `EAST` or `WEST` towards the other `Critter`. If there is a critter to the `EAST` **and** `WEST` the `Prowler` moves `EAST`.

  If a `Prowler` moves `EAST` or `WEST` instead of `NORTH` or `SOUTH` its step count is reset. For example if a `Prowler`'s steps per leg is 5 and it has taken 2 steps to the `SOUTH` and then it moves `EAST`, the number of steps taken to the `SOUTH` is reset to zero and the `Prowler` will move to the `SOUTH` 5 times unless interrupted by another `EAST` or `WEST` move. The overall steps per leg never changes.

```
public interface Critter {
    // methods to implement
    public int fight(String opponent);
    public Color getColor();
    public int getMove(CritterInfo info);
    public String toString();

    // Definitions for the ints NORTH, SOUTH, EAST, WEST, CENTER,
    // ROCK, PAPER, and SCISSORS
}

public interface CritterInfo  {
    /* Takes a direction as a parameter (one of the constants NORTH,
       SOUTH, EAST, WEST or CENTER from the Critter interface). Method
       returns the display String for the critter that is one unit away
       in that direction (or "." if the square is empty).
       If multiple critters occupy this space, returns a randomly chosen
       one of them.*/
    public String getNeighbor(int direction);

     // other methods not shown
}
```

Complete your `Prowler` class on the next page. Assume the `Color` class has been imported correctly.

```
// complete the Prowler class here:
```

```
// more room for the Prowler class on the next page if needed
```

```
// more room for Prowler class if needed
```

7. Arrays 2 - 17 Points. Write a method that given an array of `ints` and a min and max value, creates and returns a **new array** that contains only the values from the original array that are greater than or equal to min AND less than or equal to max. The relative order of the elements in the returned array is the same as the original array. The original array is not altered. **You may not use any other Java classes or methods in your answer except arrays.** Examples:

```
[-5, -5, -5, 100, 100, 0], min = 1, max = 10, returned array -> []
[1, 5, 1, 2, 5, 5, 3, 1], min = 1, max = 4, returned array -> [1, 1, 2, 3, 1]
[1, 6, 10, 12, -10], min = -20, max = 30, returned array -> [1, 6, 10, 12, -10]

// Assume data != null and min <= max. The array data is unchanged by this method
public static int[] getValuesInRange(int[] data, int min, int max) {

      int count = 0;

      for(int i = 0; i < data.length; i++)
            if(min <= data[i] && data[i] <= max)
                  count++;

      int[] result = new int[count];

      int indexResult = 0;

      for(int i = 0; i < data.length; i++) {
            if(min <= data[i] && data[i] >= max) {
                  result[indexResult] = data[i];
                  indexResult++;
            }
      }

      return result;
}
```

variable to count in range: 1 point

one loop to go through all elements int data: 4 points (-1 for off by one)

check if current element in range: 3 points

increment count if in range: 1 point

create resulting array of correct size: 1 point

index to track location in resulting array: 2 points

second loop: 1 point

add in range element at correct spot in result: 2 points

increment index in result: 1 point

return result: 1 point

IF SOLUTION IS INCORRECT WITH SINGLE LOOP: -8

8. Methods - 13 points.

A. 4 points. Write a method named `inbounds`. The method has three parameters: a 2d array of chars and two `int`s that specify a row and column for a possible cell. The method returns `true` if the row and column specify a cell that is inbounds (exists) in the given 2d array of `char`s or `false` if the row and column specify a cell that is out of bounds. (Doesn't exist in the 2d array of `char`s).

Complete the `inbounds` method below.

```
public static boolean inbounds(char[][] mat, int r, int c) {
     return 0 <= t && r < mat.length && 0 <= col && col < mat[r].length;
}
```

5 points total:

1 point header and parameters correct
1 point check row correctly
1 point check col correctly
1 point return correct answer

B. 9 points. Write a method named `countNeighbors`. The method has 4 parameters: a 2d array of `char`s, two `int`s that specify the row and column of a cell, and a target `char`.

Assume the row and column are inbounds.

The method returns the number of adjacent cells to the specified cell (given by the row and column) that equal the target `char`. A cell has at most four adjacent cells: the cells above, below, left, and right. Diagonal cells are not considered adjacent cells for this question.

The specified cell will be inbounds, but it may be an edge or corner cell that has only 2 or 3 adjacent cells. Call the `inbounds` method from part A to check if a potential adjacent cell actually exists to avoid array index out of bounds exceptions. **Do not rewrite the functionality of `inbounds` here in part B.**

```
public static int countNeighbors(char[][] mat, int r, int c, char tgt) {

     int count = 0;
     if(inbounds(mat, r - 1, c) && mat[r - 1][c] == tgt)
          count++;

     if(inbounds(mat, r + 1, c) && mat[r + 1][c] == tgt)
          count++;

     if(inbounds(mat, r, c - 1) && mat[r][c - 1] == tgt)
          count++;

     if(inbounds(mat, r, c + 1) && mat[r][c + 1] == tgt)
          count++;

     return count;
}
```

9 points total

header correct: 1 point

temporary variable for count: 1 point

call inbounds correctly: 1 points

check one adjacent cells if inbounds correctly: 2 points

check other adjacent cells, 1 point each, 3 points total

return correct count: 1 point

9. 2D arrays and simulation - 20 points. Fire!! Write a method that simulates the spread of a forest fire.

- The forest is represented by a 2d array of `char`s.
- Each element in the 2d array is either `'X'`, `'F'`, `'T'`, or `'O'`.
- `'X'` represents cells that were on fire but have burned out.
- `'F'` represents cells that are on fire.
- `'T'` represents cells with trees that have not caught fire.
- `'O'` represents open cells with no plants. These cells never catch fire.

Write a method that takes the current state of the forest and fire, represented by a 2d array of chars, and determines the next step of the forest fire in the simulation.

- Cells that are `'X'` or `'O'` do not change from one step of the simulation to another.
- Cells that are on fire (`'F'`) have a 75% chance of burning out and becoming `'X'` in the next step of the simulation.
- Cells with trees (`'T'`) have a 0% chance of catching fire if zero adjacent cells are on fire.
- Cells with trees (`'T'`) have a 50% chance of catching fire if one adjacent cell is on fire.
- Cells with trees (`'T'`) have a 100% chance of catching fire if two or more adjacent cells are on fire.

A cell has at most 4 adjacent cells: up, down, left, right.

Cells in the corners only have 2 adjacent cells.

Cells on the edges (but not the corners) only have 3 adjacent cells.

Your method must simulate that all the changes occur at once.

Call the `countNeighbors` method from question 8.B as appropriate.
Do not rewrite the functionality of `countNeighbors` in this question.

The only other Java class and method you may use are arrays and the `Math.random()` method.

Complete the following method on the next page:

```
/*   forest is not null.
     forest is a rectangular matrix, all rows have the same number of columns.
     All elements of forest are 'X', 'F', 'T', or 'O'.
     The method returns a new 2d array of chars that represents the forest in
          the next step of the simulation.
*/
public static char[][] getNextStep(char[][] forest) {
```

COMPLETE THIS METHOD ON THE NEXT PAGE.

```
public static char[][] getNextStep(char[][] forest) {
```