# CIS 143 Lab 10: Additional work in recursion, inheritance, and debugging (100 points)

Please leave the lab questions/instructions/rubrics/etc. in place. Just paste your screenshots and code below the rubric.

Purpose/knowledge/skills: This lab gives you a chance to practice and expand skills you already have for recursion and debugging.

## Task 1: Perform a recursive trace of a program (10 points)

Given the function reverseString, complete the recursive trace tables below.

**public** **static** String reverseString(String strVar) {

// Base case

**if** (strVar.length() <= 1)

**return** strVar;

// Recursive case

**else** {

**int** n = strVar.length()-1;

**return** strVar.charAt(n) + *reverseString*(strVar.substring(0,n));

}

}

|  |
| --- |
| Recursive calls  **Example** |
| reverseString(“hello”) |
| “o” + reverseString(“hell”) |
| “o” + “l” + reverseString(“hel”) |
| “o” + “l” + “l” + reverseString(“he”) |
| “o” + “l” + “l” + “e” + reverseString(“h”) |
| “o” + “l” + “l” + “e” + “h” |
| “olleh” |
| End |

|  |
| --- |
| Recursive calls |
| reverseString(“pastramo”) |
| “o” + reverseString(“pastram”) |
| “o” + “m” + reverseString(“pastra”) |
| “o” + “m” + “a” + reverseString(“pastr”) |
| “o” + “m” + “a” + “r” + reverseString(“past”) |
| “o” + “m” + “a” + “r” + “t” + reverseString(“pas”) |
| “o” + “m” + “a” + “r” + “t” + “s” + reverseString(“pa”) |
| “o” + “m” + “a” + “r” + “t” + “s” + “a” + reverseString(“p”) |
| “o” + “m” + “a” + “r” + “t” + “s” + “a” + “p” |
| “omartsap” |
| End |

|  |
| --- |
| Recursive calls |
| reverseString(“folder”) |
| “r” + reverseString(“folde”) |
| “r” + “e” + reverseString(“fold”) |
| “r” + “e” + “d” + reverseString(“fol”) |
| “r” + “e” + “d” + “l” + reverseString(“fo”) |
| “r” + “e” + “d” + “l” + “o” + reverseString(“f”) |
| “r” + “e” + “d” + “l” + “o” + “f” |
| “redlof” |
| End |

Rubric:  
Two recursive traces: 10 points total, 5 points each, about 0.3 points per line

## Task 2: Perform a recursive trace of a program (15 points)

Given the function to compute factorials, complete the recursive trace tables below.

The tables may have extra rows; feel free to delete extra rows as needed.

// Sample code from https://www.programiz.com/java-programming/recursion

static int factorial( int n ) {

if (n != 0)

return n \* factorial(n-1); // recursive call

else

return 1; // base case

}

|  |
| --- |
| Recursive calls |
| factorial(3) |
| 3 \* factorial (2) |
| 3 \* 2 \* factorial (1) |
| 3 \* 2 \* 1 |
| 6 |
| End |

|  |
| --- |
| Recursive calls |
| factorial(5) |
| 5 \* factorial(4) |
| 5 \* 4 \* factorial(3) |
| 5 \* 4 \* 3 \* factorial(2) |
| 5 \* 4 \* 3 \* 2 \* factorial(1) |
| 5 \* 4 \* 3 \* 2 \* 1 |
| 120 |
| End |

|  |
| --- |
| Recursive calls |
| factorial(7) |
| 7 \* factorial(6) |
| 7 \* 6 \* factorial(5) |
| 7 \* 6 \* 5 \* factorial(4) |
| 7 \* 6 \* 5 \* 4 \* factorial(3) |
| 7 \* 6 \* 5 \* 4 \* 3 \* factorial(2) |
| 7 \* 6 \* 5 \* 4 \* 3 \* 2 \* factorial(1) |
| 7 \* 6 \* 5 \* 4 \* 3 \* 2 \* 1 |
| 5040 |
| End |

Rubric:  
Recursive traces: 15 points total, 5 points each, points per line vary by trace

## Task 3: Perform a recursive trace of a program (20 points)

Given the function for recursive selection sort:

**public** **static** **void** main(String args[]) {

**int**[] testA = { 10, 5, 3, 7, 4, 2, 1 };

*recSS*(testA, 0, testA.length);

**for** (**int** i : testA)

System.***out***.println(i);

}

// Thanks to Aho and Ullman Foundations of Computer Science C Edition

**public** **static** **void** recSS(**int** A[], **int** i, **int** n) {

**int** j, small, temp;

**if** (i < n - 1) {

/\* basis is when i = n-1, in which case \*/

/\* the function returns without changing A \*/

/\* induction follows \*/

small = i;

**for** (j = i + 1; j < n; j++)

**if** (A[j] < A[small])

small = j;

temp = A[small];

A[small] = A[i];

A[i] = temp;

*recSS*(A, i + 1, n);

}

}

Please show the recursive trace for recSS given the function calls below.  
 It may be helpful to type in this program and confirm that it runs as expected.

|  |
| --- |
| Recursive calls |
| recSS([12, 5, 3, 7, 4, 2, 1], 0, 7) |
| recSS([1, 5, 3, 7, 4, 2, 12], 1, 7) |
| recSS([1, 2, 3, 7, 4, 5, 12], 2, 7) |
| recSS([1, 2, 3, 7, 4, 5, 12], 3, 7) |
| recSS([1, 2, 3, 4, 7, 5, 12], 4, 7) |
| recSS([1, 2, 3, 4, 5, 7, 12], 5, 7) |
| recSS([1, 2, 3, 4, 5, 7, 12], 6, 7) |
| End |

Rubric:

* First two function calls: provided, 0 points
* Five function calls: 20 points total, 4 points each

## Task 4: Create classes that inherit from a base class in Java (15 points)

Given the following starter code:

// Student Name Today's Date

**public** **class** test {

**public** **static** **void** main(String[] args) {

cat fluffy = **new** cat();

fluffy.makeNoise();

dog rover = **new** dog();

rover.makeNoise();

}

**public** **static** **class** animal {

**public** **void** makeNoise() {

System.***out***.println("Generic animal noise.");

}

}

// Define additional classes here

}

Create class dog and class cat that inherit from class animal. Override the makeNoise() method by declaring a new **public** **void** makeNoise() for dog and cat.

When a dog makes noise, it should say “woof”.

When a cat makes noise, it should say “meow”.

“Generic animal noise” should not appear in the program output.

When this works, paste a screenshot of a successful program run and your code below.

Rubric:

* Student name and today’s date is a comment in the first line of the programs: -10 points if fails
* Screenshot and program code: -10 points if fails
* cat, dog class definition: 4 points each
* Method override for makeNoise: 4 points total, 2 points each dog and cat
* Method definition for makeNoise: 4 points total, 2 points each dog and cat
* “Generic animal noise” does not appear in output: 3 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

|  |
| --- |
|  |
| // Kai Gillespie 20240317  **public** **class** Task\_4 {  **public** **static** **void** main(String[] args) {  Cat fluffy = **new** Cat();  fluffy.makeNoise();  Dog rover = **new** Dog();  rover.makeNoise();  }  **public** **static** **class** Animal {  **public** **void** makeNoise() {  System.***out***.println("Generic animal noise.");  }  }  // Define additional classes here  **public** **static** **class** Cat **extends** Animal {  @Override  **public** **void** makeNoise() {  System.***out***.println("meow");  }  }  **public** **static** **class** Dog **extends** Animal {  @Override  **public** **void** makeNoise() {  System.***out***.println("woof");  }  }  } |

## Task 5: Create classes that inherit from a base class in Java (20 points)

Given class GradedActivity:

**public** **class** GradedActivity {

**private** **double** score; // Numeric score

**public** **void** setScore(**double** s) {

score = s;

}

**public** **double** getScore() {

**return** score;

}

**public** **char** getGrade() {

**char** letterGrade;

**if** (score >= 90)

letterGrade = 'A';

**else** **if** (score >= 80)

letterGrade = 'B';

**else** **if** (score >= 70)

letterGrade = 'C';

**else** **if** (score >= 60)

letterGrade = 'D';

**else**

letterGrade = 'F';

**return** letterGrade;

}

}

10.5.1. Design a FinalExam class that extends the GradedActivity class. The FinalExam class should determine the grade a student receives for an exam.

The student’s exam score can be up to 100 and is determined in the following manner:

MultipleChoice: 50 points

FillInTheBlank: 20 points

Coding: 30 points

Some common student issues:

* The finalExam class must be passed these three values: multipleChoice, fillInTheBlank, Coding
* Calculating the total score must happen in finalExam, not the main program
* You are not allowed to modify the GradedActivity class,except to make it static if you want to include it in the same .java file as the main program

10.5.2. Demonstrate the class in a simple program that assigns scores for each of the 3 sections and displays the resulting letter grade.

One possible main program, although this can be done several other ways:

FinalExam f = **new** FinalExam();

f.setScore(45, 10, 25);

System.***out***.println("The student earned a " + f.getGrade());

With the output



Rubric:

* Student name and today’s date is a comment in the first line of the programs: -10 points if fails
* Screenshot and program code: -10 points if fails
* FinalExam class is passed three values (MultipleChoice, FillInTheBlank, Coding): -20 points if fails
* GradedActivity included as-is: 7 points
* FinalExam inheritance and method override(s): 7 points (No override required?)
* FinalExam data structures as/if needed: 3 points
* Main program to demonstrate FinalExam: 3 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your **.java file that defines class FinalExam**, here.

|  |
| --- |
| **public** **class** GradedActivity {  **private** **double** score; // Numeric score  **public** **void** setScore(**double** s) {  score = s;  }  **public** **double** getScore() {  **return** score;  }  **public** **char** getGrade() {  **char** letterGrade;  **if** (score >= 90)  letterGrade = 'A';  **else** **if** (score >= 80)  letterGrade = 'B';  **else** **if** (score >= 70)  letterGrade = 'C';  **else** **if** (score >= 60)  letterGrade = 'D';  **else**  letterGrade = 'F';  **return** letterGrade;  }  } |
| // Kai Gillespie 20240317  **public** **class** FinalExam **extends** GradedActivity {    // Constructor taking three scores and computing the total score  **public** FinalExam(**double** multipleChoice, **double** fillInTheBlank, **double** coding) {  // Calculate total score and set it using the setScore method  **double** totalScore = multipleChoice + fillInTheBlank + coding;  setScore(totalScore);  }  } |
| // Kai Gillespie 20240317  **public** **class** Task\_5 {  **public** **static** **void** main(String[] args) {  // Example usage:  **double** multipleChoiceScore = 40; // out of 50  **double** fillInTheBlankScore = 18; // out of 20  **double** codingScore = 25; // out of 30  FinalExam studentExam = **new** FinalExam(multipleChoiceScore, fillInTheBlankScore, codingScore);  // The constructor calculates the total score  **char** studentGrade = studentExam.getGrade(); // Uses the method from GradedActivity class  System.***out***.println("The student's letter grade is: " + studentGrade);  }  } |
|  |

## Task 6: Debug a Java program with multiple errors (20 points)

This program does not work as expected.

// Student name Today's Date

**public** **static** **void** original(String[] args) {

// **TODO** Auto-generated method stub

String[] names = { "Juanita", "Patel" };

String[] words = { "Good luck", " Fare well" };

**int** totalLength = 0;

**for** (**int** i = 0; i < names.length(); i++)

totalLength += names[i].length;

System.***out***.println(totalLength);

System.***out***.println(words.toUpperCase());

}

Fix this program so it generates the expected output:



Another sample output, for different starting values:

String[] names = { "Jim", "Hanna", "Vo" };

String[] words = { "Eat ", "Well ", "Be ", "Well" };



Notes:

* You only need to include one test case in your program and output
* The program must work for any reasonable initialization of names[] and words[], not just the examples

Rubric:

* Student name and today’s date is a comment in the first line of the programs: -10 points if fails
* Program works for names[] and words[] other than the sample cases: 5 points
* Loop through names[] array: 3 points
* Total length of names[] strings: 2 points
* Output words[] array as upper case: 5 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

|  |
| --- |
|  |
| // Kai Gillespie 20240317  **public** **class** Task\_6 {  **public** **static** **void** main(String[] args) {  // First example  String[] names1 = { "Juanita", "Patel" };  String[] words1 = { "Good luck", "Fare well" };  *processStrings*(names1, words1);  System.***out***.println(""); //Spacer for readability.  // Second example  String[] names2 = { "Jim", "Hanna", "Vo" };  String[] words2 = { "Eat ", "Well ", "Be ", "Well" };  *processStrings*(names2, words2);  }  // Calculate the total length of the names in the names array  **private** **static** **int** calculateTotalLength(String[] names) {  **int** totalLength = 0;  **for** (String name : names) {  totalLength += name.length();  }  **return** totalLength;  }  // Join the words array elements with a space, then convert to uppercase  **private** **static** String combineWordsToUpper(String[] words) {  **return** String.*join*(" ", words).toUpperCase().trim();  }  // Print outside the loop instead of inside the loop.  **private** **static** **void** processStrings(String[] names, String[] words) {  System.***out***.println(*calculateTotalLength*(names)); // Print the total length of the names in the names array  System.***out***.println(*combineWordsToUpper*(words)); // Print the combined words in the words array in uppercase  }  } |