Unit 10: Recursion

**Topic 1 Lab 4: Recursion Applications**

| **Name:** |  | | | **HANDWRITE ALL STACK TABLES!** |
| --- | --- | --- | --- | --- |

**Work on these with your partner using the U10T1 Lab 4 group Replit**

| ArrayLists & Recursion | |
| --- | --- |
| Determine what will be printed if the code in the main method is executed:  -------------------------------------------------  public class Main {  public static void main(String[] args) {  ArrayList<Integer> numList = new ArrayList<Integer>();  numList.add(5);  numList.add(7);  numList.add(8);  numList.add(5);  numList.add(6);  numList.add(5);  numList.add(4);  numList.add(1);  ArrayListMystery myObj = new ArrayListMystery(5);  int value = myObj.mystery(numList);  System.out.println(value);  System.out.println(numList);  }  }  -------------------------------------------------  public class ArrayListMystery {  private int num;  public ArrayListMystery(int num) {  this.num = num;  }  public int mystery(ArrayList<Integer> list) {  if (list.size() == 0) {  return 0;  }  int current = list.remove(0);  if (current == num) {  return 1 + mystery(list);  } else {  return mystery(list);  }  }  }  **1. Create a handwritten call stack table and use it to obtain the two printed values**   | What **two** values do you expect to be printed, based on your table? (note there are *two* print statements) |  | | --- | --- |   **2. Confirm your answer! After** using a handwritten table to determine the output, **copy/paste the code above** into Replit and execute it.   | What is the actual printed output? |  | | --- | --- | | Was your answer correct? If not, why not? |  |  [Check output](#_1zqfnqqlxqqx)and in case you want to see it, [here is a sample table](#_vjfxinfsj09x)  | **Multiple Choice:**  Which of the following best describes the behavior of the mystery method?  **(A)** The method removes only the first occurrence of num in list.  **(B)**  The method removes all occurrences of num in list and leaves list unchanged.  **(C)** The method determines the number of times num occurs in list without changing the contents of list.  **(D)** The method determines the number of times num occurs in list and leaves list empty.  **(E)** The method determines the number of elements in list that are not equal to num and leaves list empty. | | | --- | --- | | **Your answer:** |  |  [Check answer](#_ify1q98mw6up) **OPTIONAL:** The College Board **10.1: Daily Video 2** (in AP Classroom) discusses this example and multiple choice question at time **6:48 - 8:38** | |

| **Fibonacci!** | |
| --- | --- |
| Recall that the *Fibonacci sequence* starts with 0, 1, 1 and is formed by summing the previous two numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89… You will use this in the problem below!  Analyze the following class and determine the output when the main method is executed. As you see in the fibonacci method below, it’s possible to have *multiple recursive calls*.  public class Main {  public static void main(String[] args) {  System.out.println(fibonacci(11));  }    private static int fibonacci(int x) {  if (x == 1) {  return 0;  }  if (x == 2) {  return 1;  }  **// double recursive method call**  return fibonacci(x - 1) + fibonacci(x - 2);  }  }   | **Before doing anything,** which of these options best describes what this code will do? A or B?   1. it will print the 11th number in the Fibonacci sequence 2. it will print the first 11 numbers of the Fibonacci sequence | [Check](#_3g8ytc5dl58f) | | --- | --- |   **1. Create a handwritten call stack table and use it to obtain the printed value.**   | What value do you expect to be printed, based on your table? |  | | --- | --- |   **2. Confirm your answer! After** using a handwritten table to determine the output, **copy/paste the code above** into Replit and execute it.   | What is the actual printed output? |  | | --- | --- | | Was your answer correct? If not, why not? |  |  [Check answer](#_829y11ioxxyb)and in case you want to see it, [here is a sample table](#_3wcruptdojrs) Now, **modify the code** in some way to *print out the first 11 numbers of the Fibonacci sequence* on the same line, like this:   [hint!](#_b525w0o2fpv6) Insert the entire class:   |  | | --- |  [sample solution](#_nlq06axtih7n) | |

**Lab continues on the next page**

**Collaborative Recursive Challenge**

Code this up together!

| Tricky! Recursive Minimum | |
| --- | --- |
| Write a **recursive** method that finds the *minimum* value in an ArrayList of numbers. The user will first be prompted for a list of numbers and enter them one at a time. One way to think of finding a minimum recursively is to think: “the minimum number is either the *last* element in the ArrayList, or the minimum value in the rest of the ArrayList”. For example, if you have the ArrayList [3, 2, 1, 15, 23, 7, 9], the minimum value in this ArrayList is either 9 (the last number) or the minimum value in the *rest* of the ArrayList (excluding 9):  [3, 2, 1, 15, 23, 7]  **Copy/paste the starting code below into Main.java**, then write the findMinimum method using ***recursion*** to find and return the minimum (yes, you could do this easily without recursion, but the challenge here is to figure out a way to do it recursively! Give it a shot 😎).  import java.util.ArrayList;  import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  ArrayList<Integer> numbers = new ArrayList<Integer>();  int inputNum = 0;  while (inputNum != -1) {  System.out.print("Please enter numbers. Enter -1 to quit: ");  inputNum = input.nextInt();  if (inputNum != -1) {  numbers.add(inputNum);  }  }    int minimum = **findMinimum(numbers);**  System.out.println("Minimum: " + minimum);  }    public static int findMinimum(ArrayList<Integer> numbers) {  **// WRITE THIS METHOD using recursion to find the minimum value in numbers**  }  } [**Hint!**](#_k2ahyyrcdzp3) **TEST CASE 1**    **TEST CASE 2**    **TEST CASE 3**    **Copy/paste your recursive findMinimum method below:**   |  | | --- |  A sample solution will be posted later! | |

**Done!**

Submit in Google Classroom:



### Answer ([back](#_sbevbu4a4vbh))

From two lines of code:

| What is the actualprinted output?  System.out.println(value); →  System.out.println(numList); → | **3**  **[]** |
| --- | --- |

###### 

### Answer ([back](#_ilalskbwxbk7))

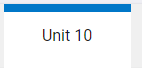
**Correct answer: D**

| **7. Multiple Choice:**  Which of the following best describes the behavior of the mystery method?  **(A)** The method removes only the first occurrence of num in list.  **(B)**  The method removes all occurrences of num in list and leaves list unchanged.  **(C)** The method determines the number of times num occurs in list without changing the contents of list.  **(D) The method determines the number of times num occurs in list and leaves list empty.**  **(E)** The method determines the number of elements in list that are not equal to num and leaves list empty. | |
| --- | --- |
| **Correct answer:** | **D** |

###### 

**FOR DETAILED EXPLANATION OF THIS QUESTION:**

Watch College Board’s **10.1 Daily Video 2** on AP Classroom at time **6:48 - 8:38**





### Answer ([back](#_fn6p65e5xnj1))

| What is the actualprinted output? | **55**  This is the **11**th number in the Fibonnaci sequence:  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, **55**, 89, 144… |
| --- | --- |

###### 

### Hint ([back](#_9fnoukhiwuj))

The trick is to *remove* the *last* element in the ArrayList each time through to make the ArrayList a little shorter, similar to the first problem in this lab.

### Check ([back](#_urmcdup3g0o3))

| **Before doing anything,** which of these options best describes what this code will do? A or B?   1. **it will print the 11th number in the Fibonacci sequence** 2. it will print the first 11 numbers of the Fibonacci sequence | **A**  The fibonacci method returns an int, and there is only one print statement, which is in the main method -- there are no print statements occurring in the fibonacci method, so only one int ultimately gets returned and printed |
| --- | --- |

public class Fibonacci {

public static void main(String[] args) {

**System.out.println**(fibonacci(11)); **// a single print statement**

}

private static **int** fibonacci(int x) { **// returns an int & no print**

**// statements in this method**

if (x == 1) {

return 0;

}

if (x == 2) {

return 1;

}

// double recursive method call

return fibonacci(x - 1) + fibonacci(x - 2);

}

}

### Sample solution ([back](#_z1hzl494piv1))

Add a for loop in the main method that goes from 1 to 11. Each time through the loop, the call to fibonacci(i) prints out one value -- the ith number in the Fibonnaci sequence -- and so looping from 1 to 11 ensure that it prints out the 1st, 2nd, 3rd, 4th, 5th… 10th, then finally the 11th number in the sequence.

You don't need to change anything in the fibonacci method:



### Table ([back](#_ysgkhj1ez2yt))

**note: "f" as shorthand for "fibonacci" method**

**CALL STACK BEFORE SUBSTITUTION (shown going *downwards*):**

| f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → f(6) + f(5)  f(6) → f(5) + f(4)  f(5) → f(4) + f(3)  f(4) → f(3) + f(2)  f(3) → f(2) + f(1)  f(2) → 1 **// one base case**  f(1) → 0 **// the *other* base case** |
| --- |

**SUBSTITUTING**

| **first substitution: the 0 and the 1**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → f(6) + f(5)  f(6) → f(5) + f(4)  f(5) → f(4) + f(3)  f(4) → f(3) + f(2)  f(3) → **1** + **0** = **1**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 1 and the 1**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → f(6) + f(5)  f(6) → f(5) + f(4)  f(5) → f(4) + f(3)  f(4) → **1** + **1** = **2**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 1 and the 2**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → f(6) + f(5)  f(6) → f(5) + f(4)  f(5) → **2** + **1** = **3**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 2 and the 3**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → f(6) + f(5)  f(6) → **3** + **2** = **5**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 3 and the 5**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → f(7) + f(6)  f(7) → **5** + **3** = **8**  ~~f(6) →~~ **~~3~~** ~~+~~ **~~2~~** ~~=~~ **~~5~~**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 5 and the 8**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → f(8) + f(7)  f(8) → **8** + **5** = **13**  ~~f(7) →~~ **~~5~~** ~~+~~ **~~3~~** ~~=~~ **~~8~~**  ~~f(6) →~~ **~~3~~** ~~+~~ **~~2~~** ~~=~~ **~~5~~**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 8 and the 13**  f(11) → f(10) + f(9)  f(10) → f(9) + f(8)  f(9) → **13** + **8** = **21**  ~~f(8) →~~ **~~8~~** ~~+~~ **~~5~~** ~~=~~ **~~13~~**  ~~f(7) →~~ **~~5~~** ~~+~~ **~~3~~** ~~=~~ **~~8~~**  ~~f(6) →~~ **~~3~~** ~~+~~ **~~2~~** ~~=~~ **~~5~~**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **next substitution: the 13 and the 21**  f(11) → f(10) + f(9)  f(10) → **21** + **13** = **34**  ~~f(9) →~~ **~~13~~** ~~+~~ **~~8~~** ~~=~~ **~~21~~**  ~~f(8) →~~ **~~8~~** ~~+~~ **~~5~~** ~~=~~ **~~13~~**  ~~f(7) →~~ **~~5~~** ~~+~~ **~~3~~** ~~=~~ **~~8~~**  ~~f(6) →~~ **~~3~~** ~~+~~ **~~2~~** ~~=~~ **~~5~~**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **final substitution: the 21 and the 34**  f(11) → **34** + **21** = **55**  ~~f(10) →~~ **~~21~~** ~~+~~ **~~13~~** ~~=~~ **~~34~~**  ~~f(9) →~~ **~~13~~** ~~+~~ **~~8~~** ~~=~~ **~~21~~**  ~~f(8) →~~ **~~8~~** ~~+~~ **~~5~~** ~~=~~ **~~13~~**  ~~f(7) →~~ **~~5~~** ~~+~~ **~~3~~** ~~=~~ **~~8~~**  ~~f(6) →~~ **~~3~~** ~~+~~ **~~2~~** ~~=~~ **~~5~~**  ~~f(5) →~~ **~~2~~** ~~+~~ **~~1~~** ~~=~~ **~~3~~**  ~~f(4) →~~ **~~1~~** ~~+~~ **~~1~~** ~~=~~ **~~2~~**  ~~f(3) →~~ **~~1~~** ~~+~~ **~~0~~** ~~=~~ **~~1~~**  ~~f(2) →~~ **~~1~~**  ~~f(1) →~~ **~~0~~**  **the returned value is 55** |
| --- |

### 

### ([back](#_ysgkhj1ez2yt))

### Table ([back](#_9vkug5blxqlj))

**note: "m" as shorthand for "mystery" method**

**CALL STACK BEFORE SUBSTITUTION (shown going *downwards*):**

| m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + m([6, 5, 4, 1], 5)  m([6, 5, 4, 1], 5) → m([5, 4, 1], 5)  m([5, 4, 1], 5) → 1 + m([4, 1], 5)  m([4, 1], 5) → m([1], 5)  m([1], 5) → m([], 5)  m([], 5) → 0 **// base case, size of arraylist is 0**  This method modifies the arraylist object referenced by the parameter (by removing elements), which is the *same* arraylist object referenced by numList in the main method. Therefore, numList (declared in the main method and passed as the parameter) is empty. |
| --- |

**SUBSTITUTING**

| **first substitution: the 0**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + m([6, 5, 4, 1], 5)  m([6, 5, 4, 1], 5) → m([5, 4, 1], 5)  m([5, 4, 1], 5) → 1 + m([4, 1], 5)  m([4, 1], 5) → m([1], 5)  m([1], 5) → **0** = **0**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 0**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + m([6, 5, 4, 1], 5)  m([6, 5, 4, 1], 5) → m([5, 4, 1], 5)  m([5, 4, 1], 5) → 1 + m([4, 1], 5)  m([4, 1], 5) → **0** = **0**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 0**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + m([6, 5, 4, 1], 5)  m([6, 5, 4, 1], 5) → m([5, 4, 1], 5)  m([5, 4, 1], 5) → 1 + **0** = **1**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 1**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + m([6, 5, 4, 1], 5)  m([6, 5, 4, 1], 5) → **1** = **1**  ~~m([5, 4, 1], 5) → 1 +~~ **~~0~~** ~~=~~ **~~1~~**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 1**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → m([5, 6, 5, 4, 1], 5)  m([5, 6, 5, 4, 1], 5) → 1 + **1** = **2**  ~~m([6, 5, 4, 1], 5) →~~ **~~1~~** ~~=~~ **~~1~~**  ~~m([5, 4, 1], 5) → 1 +~~ **~~0~~** ~~=~~ **~~1~~**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 2**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → m([8, 5, 6, 5, 4, 1], 5)  m([8, 5, 6, 5, 4, 1], 5) → **2** = **2**  ~~m([5, 6, 5, 4, 1], 5) → 1 +~~ **~~1~~** ~~=~~ **~~2~~**  ~~m([6, 5, 4, 1], 5) →~~ **~~1~~** ~~=~~ **~~1~~**  ~~m([5, 4, 1], 5) → 1 +~~ **~~0~~** ~~=~~ **~~1~~**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **next substitution: the 2**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + m([7, 8, 5, 6, 5, 4, 1], 5)  m([7, 8, 5, 6, 5, 4, 1], 5) → **2** = **2**  ~~m([8, 5, 6, 5, 4, 1], 5) →~~ **~~2~~** ~~=~~ **~~2~~**  ~~m([5, 6, 5, 4, 1], 5) → 1 +~~ **~~1~~** ~~=~~ **~~2~~**  ~~m([6, 5, 4, 1], 5) →~~ **~~1~~** ~~=~~ **~~1~~**  ~~m([5, 4, 1], 5) → 1 +~~ **~~0~~** ~~=~~ **~~1~~**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **final substitution: the 2**  m([5, 7, 8, 5, 6, 5, 4, 1], 5) → 1 + **2** = **3**  ~~m([7, 8, 5, 6, 5, 4, 1], 5) →~~ **~~2~~** ~~=~~ **~~2~~**  ~~m([8, 5, 6, 5, 4, 1], 5) →~~ **~~2~~** ~~=~~ **~~2~~**  ~~m([5, 6, 5, 4, 1], 5) → 1 +~~ **~~1~~** ~~=~~ **~~2~~**  ~~m([6, 5, 4, 1], 5) →~~ **~~1~~** ~~=~~ **~~1~~**  ~~m([5, 4, 1], 5) → 1 +~~ **~~0~~** ~~=~~ **~~1~~**  ~~m([4, 1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([1], 5) →~~ **~~0~~** ~~=~~ **~~0~~**  ~~m([], 5) →~~ **~~0~~**  **the returned value is 3** |
| --- |

### ([back](#_9vkug5blxqlj))

### Hint ([back](#_7u9it0k19g1f))

The most direct way to do this involves modifying the code in the main method to use a loop, and leaving the fibonacci method unchanged!