CSE 21
Intro to Computing II

Lecture 11
Recursion
Multi-dimensional Arrays (1)

Today

- Recursion, Multi-dimensional Arrays (1)
- Lab
  - Lab 12 due this week (4/15 – 4/21)
  - Lab 13 assigned this week
    - 2D Arrays and File Output
    - Due in one week (**No grace period**)
    - **Required** to show work to a TA (or me) for full credit
- Reading Assignment
  - Sections 5.9 and 12.1 – 12.6 (including participation activities)
    - Work on the **Participation Activities** in each section to receive participation grade at the end of semester (based on at least 80% completion)
    - **Each Section must have 80% completion to receive grade. <80% is a ZERO**
    - **Work on Challenge Activities to receive extra credit (Up to 5% of overall grade)**
  - Participation and Challenge activities evaluated at the end of semester
  - **ALL Activities due on May 6th at 11:59pm**

Parse Strings (review)

- We can iterate over all "ascii codes" in a string as an array:
  - Using the **charAt(int i)** method

will print 32, the ascii code of the space character

```
String s1 = "my string";
System.out.println ((int)s1.charAt(2));
for(int i = 0; i < s1.length() ; i++)
        System.out.print(s1.charAt(i));    // print 1 character a time
System.out.println(s1);                    // print the whole string
```

- We check if a character is numeric, lower/upper case, etc, by checking its ascii code

```
if(s1.charAt(i) == '1') …
if(s1.charAt(i) <= 'Z' && s1.charAt(i) >= 'A')
```

String methods (review)

- Substring
  - The **String** method **substring** creates a new **String** object containing a portion of another **String**.
  - The forms of this method are:

```
s.substring(int start);               // From [start] to end of string s
s.substring(int start, int end);      // [start] to [end]
```

  - This method returns another **String** object containing the characters from ***start*** to ***end-1*** (or the end of the **string**).
- Concatenation
  - The **String** method ***concat*** creates a new **String** object containing the contents of two other strings.
  - The form of this method is:

```
s1.concat(String s2);    // Combine s1 and s2
```

  - This method returns a **String** object containing the contents of **s1** followed by the contents of **s2**.

Some Additional String Methods (review)

Method	Description
`int` **`compareTo`**`(String s)`	Compares the string object to another string lexicographically. Returns: 0 if string is equal to `s` <0 if string less than `s` >0 if string greater than `s`
`boolean` **`equals`**`(Object o)`	Returns true if `o` is a `String`, and `o` contains exactly the same characters as the string.
`boolean` **`equalsIgnoreCase`**`(String s)`	Returns true if `s` contains exactly the same characters as the string, disregarding case.
`int` **`IndexOf`**`(String s)`	Returns the index of the first location of substring `s` in the string.
`int` **`IndexOf`**`(String s, int start)`	Returns the index of the first location of substring `s` at or after position `start` in the string.
`String` **`toLowerCase`**`()`	Converts the string to lower case.
`String` **`toUpperCase`**`()`	Converts the string to upper case.
`String` **`trim`**`()`	Removes white space from either end of the string.

Parsing Strings (review)

```
String s1 = "+This,is+an,example";
Scanner line = new Scanner (s1);
line.useDelimiter("[,+]");
while (line.hasNext()) {
     System.out.println(line.next());
}
```

- Delimiting characters are comma and plus: ',' and '+'
- OUTPUT:

```
This
is
an
example
```

Reading Files (review)

```
import java.io.*;
String filename = "nums.txt";
Scanner input = new Scanner (new FileReader(filename)); // or FileInputStream
                                                        // instead of FileReader
input.useDelimiter("[\t\r]");   // use tab and carriage return
while (input.hasNext()) {
      System.out.println(input.next());
}
input.close();
```

- Import **io** object library
- Define a file name
- Define a scanner to open a file and read its content
- Close scanner when reading is done
- Exceptions must be handled when reading files:
  - FileNotFoundException (fine does not exist)
  - NoSuchElementException (cannot perform input.next())

Reading line by line (review)

```
System.out.print("Enter the file name: ");
Scanner kdb = new Scanner(System.in);
String filename = kdb.next();

try { // TRY it out
        Scanner input = new Scanner (new FileReader(filename));
        while (input.hasNextLine()) {
                Scanner line = new Scanner(input.nextLine());
                line.useDelimiter("[\t\r]"); // Tab delimited file
                while (line.hasNext())
                        System.out.print(line.next()); // Read each token
                System.out.println();  // Done reading one line
                line.close();
        }
        input.close();
} catch (FileNotFoundException e){ // Catch Error
        System.out.println(e);
} catch (NoSuchElementException e) { // Catch Error
        System.out.println(e);
}
```

2 scanner objects!
1 for reading the whole file, 1 for reading each line.

Different Scanner Methods (review)

```
while (input.hasNextLine()) {
        Scanner line = new Scanner(input.nextLine());
        line.useDelimiter("[\t\r]");

        short s = line.nextShort();

        int i = line.nextInt();

        double d = line.nextDouble();

        float f = line.nextFloat();

        String str = line.next();
        char c = line.next().charAt(0);

        String rest = line.nextLine();
}
```

Example File Out (review)

```
String filename = "Result.txt";

try {
        FileWriter output = new FileWriter(filename);
        String outstr = "";
        for (int i = 0; i < arr.length; i++) {
                outstr = (arr[i] + "\t");
                output.write(outputstr);
        }
        output.close();
} catch (Exception e) {
        System.out.println(e);
}
```

Recursion Problem: Sum All

- Summation of numbers 1 to max
- Steps
  - subTotal = 0;
  - subTotal += 1;
  - subTotal += 2;
  - …
  - subTotal += max;
- Loop
  - Begin –> 1
  - End –> max
  - Increment –> increase by 1
  - Body –> add current number to running total

Sum All – Sub Problem

- Summation of numbers 1 to max
- Steps
  - subTotal = 0;
  - subTotal += 1;
  - subTotal += 2;
  - …
  - subTotal += max;
- Re-written
  - sumAll(0) → 0
  - sumAll(1) → sumAll(0) + 1
  - sumAll(2) → sumAll(1) + 2
  - ….
  - sumAll(n) → sumAll(n-1) + n

Final answer contains solution of the problem

Two Versions

- Iterative (loop)

```
subTotal = 0;
for (int i = 1; i <= max ; i++) {
        subTotal += i;
}
```

- Recursive

```
public static int sumAll(int n) {
        if (n == 0)
                return 0;
        else
                return n + sumAll(n - 1);
```

Call the method again with a new argument

Declaration and Invocation

```
public static long sumAll(int n) { // Declaration
    System.out.println("sumAll " + n);
    if (n == 0)
        return 0;
    else
        return n + sumAll(n - 1);
}

public static void main(String[] args) {
    System.out.println("sumAll output for 5 is " + sumAll(5)); // Invoke
    System.out.println("sumAll output for 10 is " + sumAll(10));
    System.out.println("sumAll output for 20 is " + sumAll(20));
    System.out.println("sumAll output for 15 is " + sumAll(15));
    System.out.println();
}
```

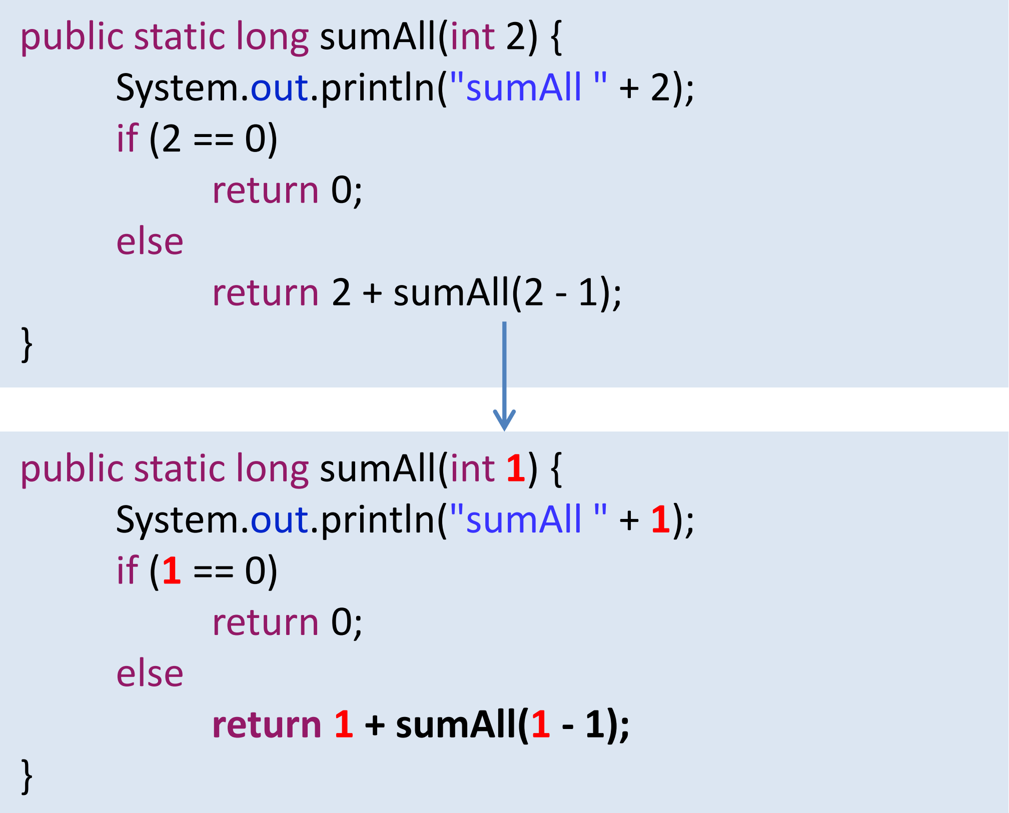
Call sumAll(2)

```
public static long sumAll(int 2) {
        System.out.println("sumAll " + 2);
        if (2 == 0)
                return 0;
        else
                return 2 + sumAll(2 - 1);
}
```

OUTPUT:

```
sumAll 2
```

Call sumAll(2)



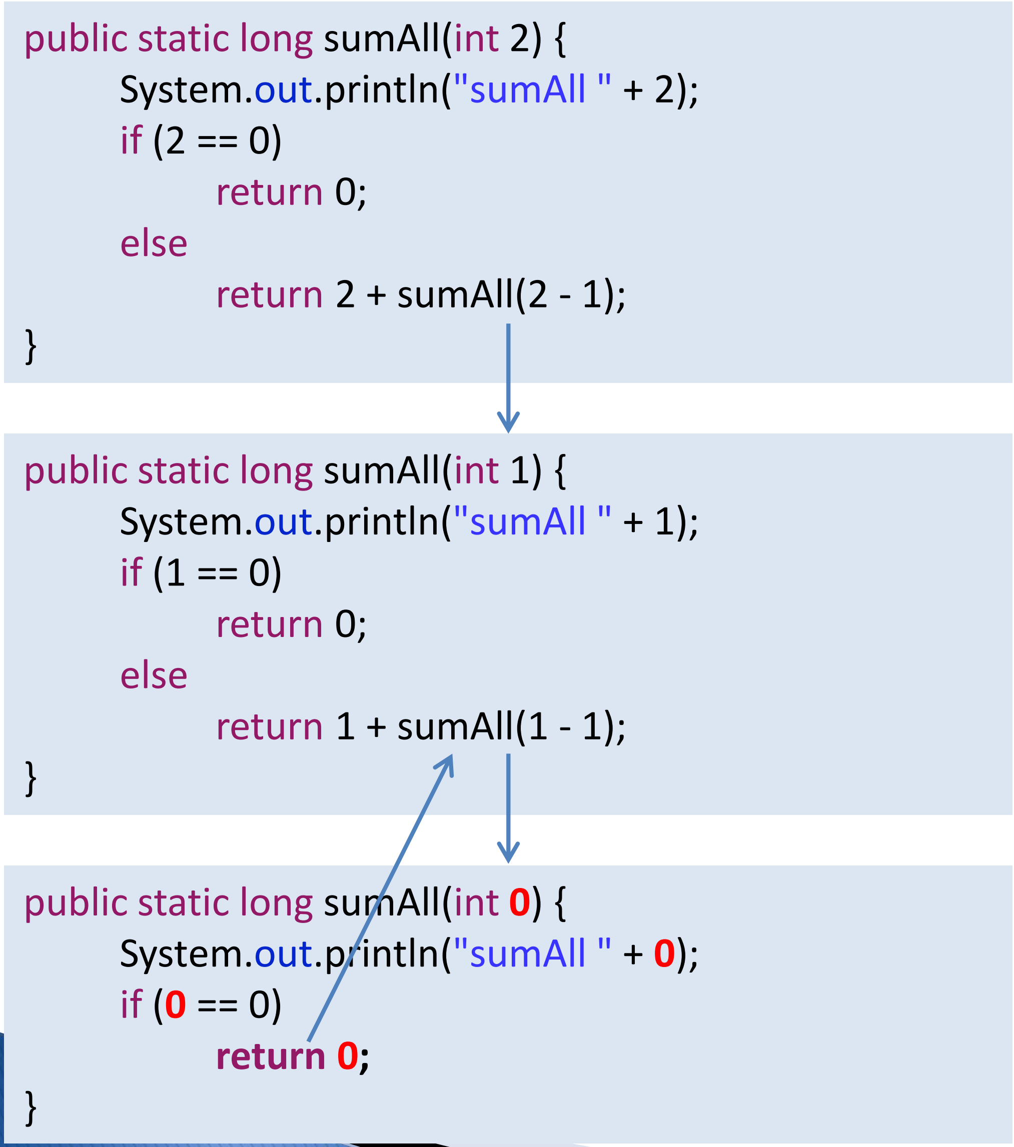
```
public static long sumAll(int 2) {
        System.out.println("sumAll " + 2);
        if (2 == 0)
                return 0;
        else
                return 2 + sumAll(2 - 1);
}
```

```
public static long sumAll(int 1) {
        System.out.println("sumAll " + 1);
        if (1 == 0)
                return 0;
        else
                return 1 + sumAll(1 - 1);
}
```

OUTPUT:

```
sumAll 2
sumAll 1
```

Call sumAll(2)



```
public static long sumAll(int 2) {
        System.out.println("sumAll " + 2);
        if (2 == 0)
                return 0;
        else
                return 2 + sumAll(2 - 1);
}
```

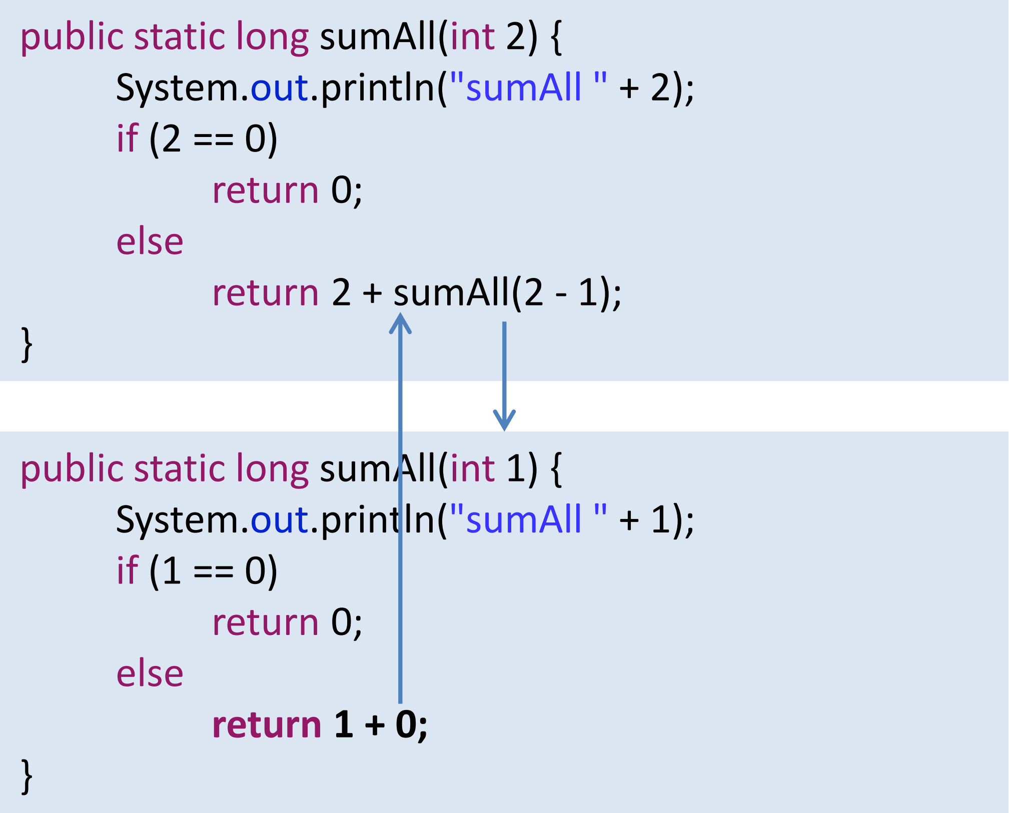
```
public static long sumAll(int 1) {
        System.out.println("sumAll " + 1);
        if (1 == 0)
                return 0;
        else
                return 1 + sumAll(1 - 1);
}
```

```
public static long sumAll(int 0) {
        System.out.println("sumAll " + 0);
        if (0 == 0)
                return 0;
}
```

OUTPUT:

```
sumAll 2
sumAll 1
sumAll 0
```

Call sumAll(2)



```
public static long sumAll(int 2) {
        System.out.println("sumAll " + 2);
        if (2 == 0)
                return 0;
        else
                return 2 + sumAll(2 - 1);
}
```

```
public static long sumAll(int 1) {
        System.out.println("sumAll " + 1);
        if (1 == 0)
                return 0;
        else
                return 1 + 0;
}
```

OUTPUT:

```
sumAll 2
sumAll 1
sumAll 0
```

Call sumAll(2)

```
public static long sumAll(int 2) {
        System.out.println("sumAll " + 2);
        if (2 == 0)
                return 0;
        else
                return 2 + 1;
}
```

```
public static void main(String[] args) {
        System.out.println("sumAll of 2 is " + sumAll(2));
}
```

OUTPUT:

```
sumAll 2
sumAll 1
sumAll 0

sumAll of 2 is 3
```

Recursion Problem: Factorial

$$n! = \begin{cases} 1, & n = 0 \\ n \times \underbrace{(n-1) \times (n-2) \ldots \times 2 \times 1,}_{?\quad (n-1)!} & n > 0 \end{cases}$$

Recursive definition:

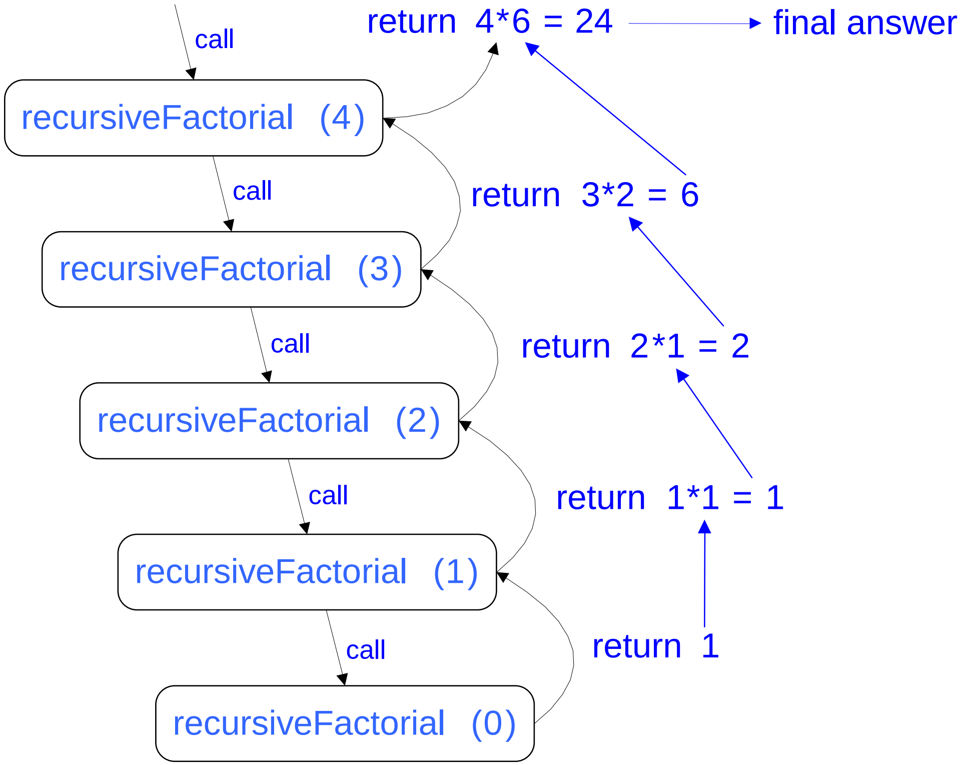
$$n! = \begin{cases} 1, & n = 0 \\ n \times (n-1)!, & n > 0 \end{cases}$$

Recursive factorial steps

```
factorial(4)
              = 4 * factorial(3)
              = 4 * (3 * factorial(2))
              = 4 * (3 * (2 * factorial(1)))
              = 4 * (3 * (2 * (1 * factorial(0)))
              = 4 * (3 * (2 * (1 * 1)))
              = 4 * (3 * (2 * 1))
              = 4 * (3 * 2)
              = 4 * 6
              = 24
```

Recursive trace

- Box for each recursive call.
- Arrow from each caller to callee.
- Arrow from each callee to caller showing return value.



Linear versus Binary Recursion

- Linear recursion: function calls itself once
- Binary recursion: function calls itself twice

Linear Recursion:

```
public static type recursive_function( … )
{
        …
        … recursive_function(…) …
        …
}
```

Binary Recursion:

```
public static type recursive_function( … )
{
        …
        … recursive_function(…) …
        …
        … recursive_function(…) …
        …
}
```

Fibonacci numbers

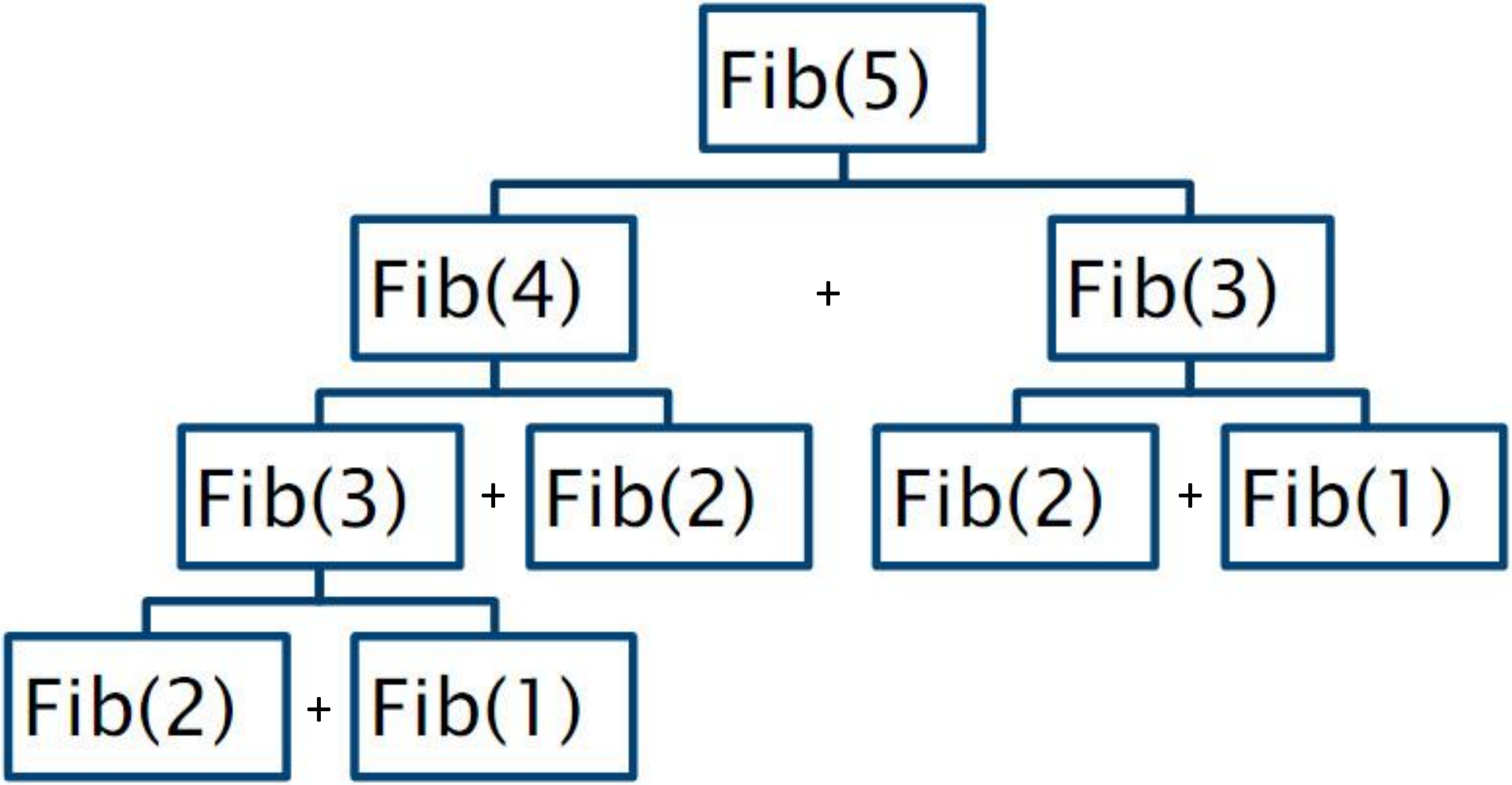
$F_0 = 0$

$F_1 = 1$

$F_k = F_{k-1} + F_{k-2}$

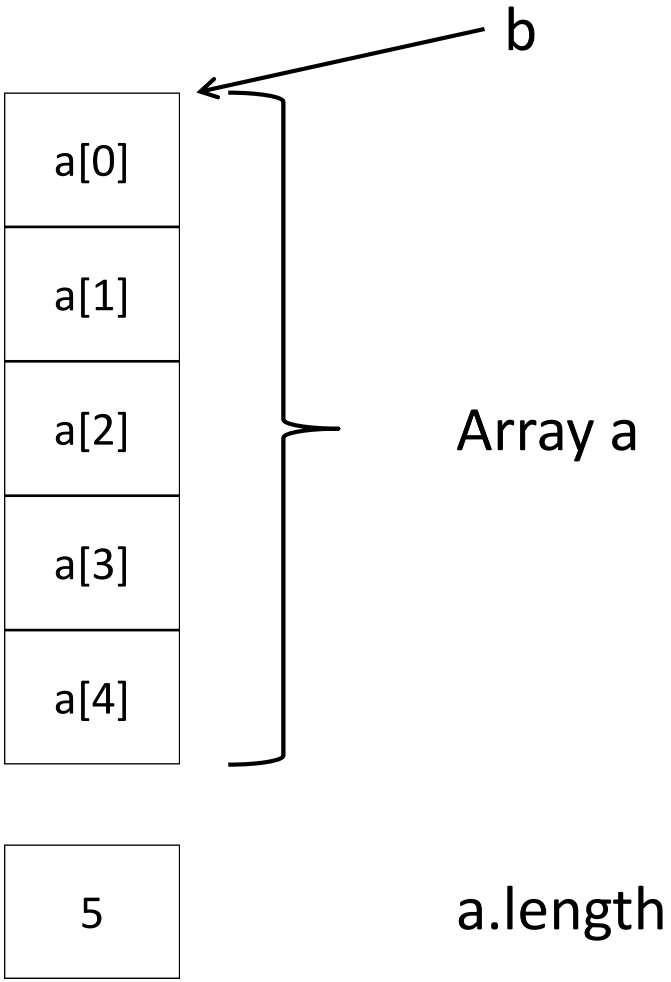
0, 1, 1, 2, 3, 5, 8, 13, …..

Binary (Tree) Recursion



Review: 1D Arrays

- An array is a special object containing:
  - A group of contiguous memory locations that all have the same type
  - A special (hidden) variable containing the number of elements in the array
- int[] a = new int[5];
- int[] b = a;



b

a[0]

a[1]

a[2]

a[3]

a[4]

Array a

5

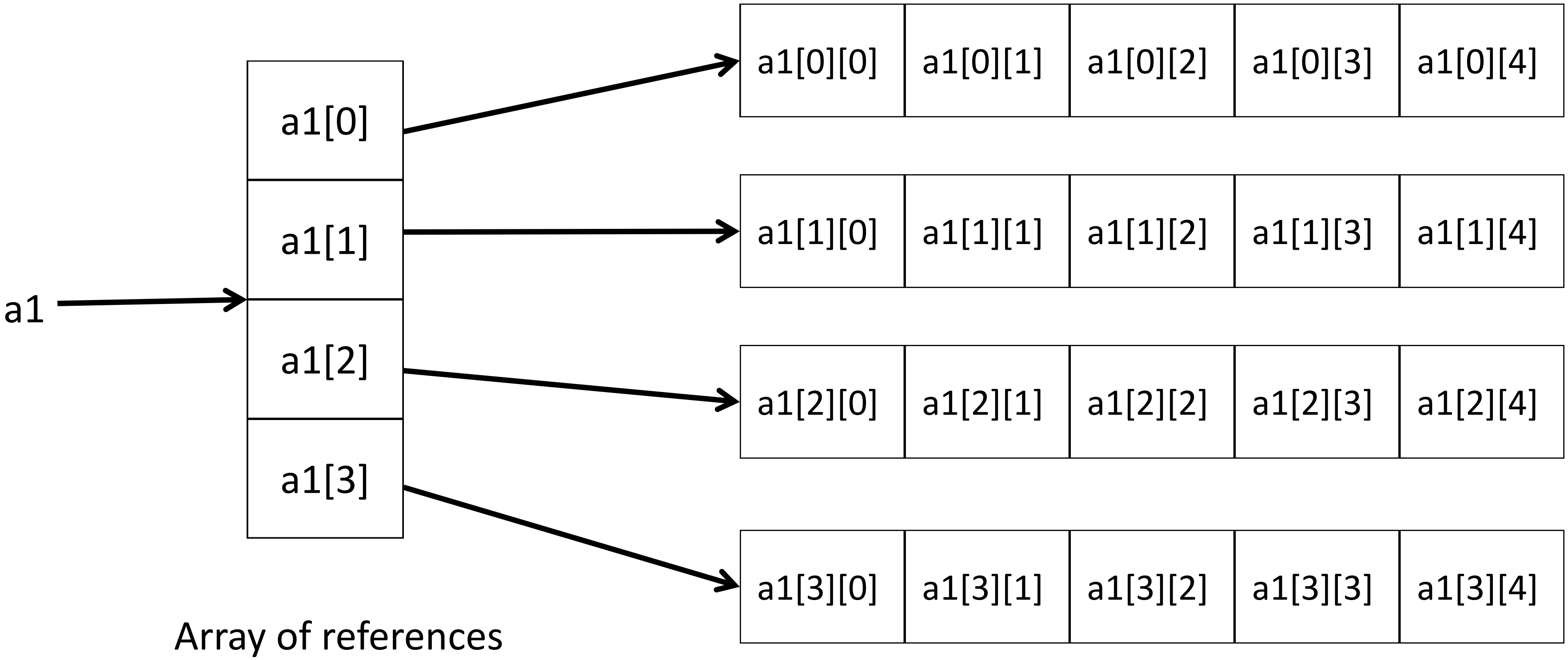
a.length

2D Arrays

- A two-dimensional (2D) array is an array of references (or pointers) to other arrays:
  - All arrays must be of the same type
- This creates a 2D data structure
- As matrices (in math), individual elements in the array are addressed with two subscripts, specifying the Row and Column (in that order) of the particular data item. (called row major)
- Memorize RC
  - (Radio Control, Royal Crown, Rice Cracker, RC Cola)

2D Arrays diagram

```
int[][] a1 = new int[4][5];
```



a1

a1[0]
a1[1]
a1[2]
a1[3]

a1[0][0] a1[0][1] a1[0][2] a1[0][3] a1[0][4]
a1[1][0] a1[1][1] a1[1][2] a1[1][3] a1[1][4]
a1[2][0] a1[2][1] a1[2][2] a1[2][3] a1[2][4]
a1[3][0] a1[3][1] a1[3][2] a1[3][3] a1[3][4]

Array of references
to other arrays

Arrays containing data

Declaring 2D Arrays

- We first declare an array of *references to other arrays*, then declare the arrays.
- Example:

```
double[][] a; //the actual array
a = new double[3][5];
```

- These steps can be combined on a single line, just like in 1D:

```
double[][] a = new double[3][5];
```

- 2D arrays may be initialized with nested array initializers

```
int[][] b = { {1,2,3}, {4,5,6} };
```

1st row   2nd row

Using 2D Arrays

- 2D arrays are used to represent data that is a function of two variables (or indices)
- A 2D array element is addressed using the array name followed by a integer subscript in brackets: **a[3][5]**
- Sizes of the arrays
  - a.length is the number of rows
  - a[0].length is the number of elements (cols) in row 0
  - a[1].length is the number of elements (cols) in row 1