CSE 21
Intro to Computing II

Lecture 12 – Recursion

Announcement

- Lab #12 (not Lab #11) due before start of next lab
  - Type your answers in a text file and submit it as an attachment
- No lecture next Wednesday (Thanksgiving)
  - There will be new lab assignment (Lab #11, not Lab #12)
    - No lab sessions
    - Due in 2 weeks (same due date as Lab #13)
- Project #2
  - Due Monday (11/28) at 11:59pm
- Reading assignment
  - Chapter 9.1 to 9.5, 12.1 to 12.6 of textbook

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();
}
```

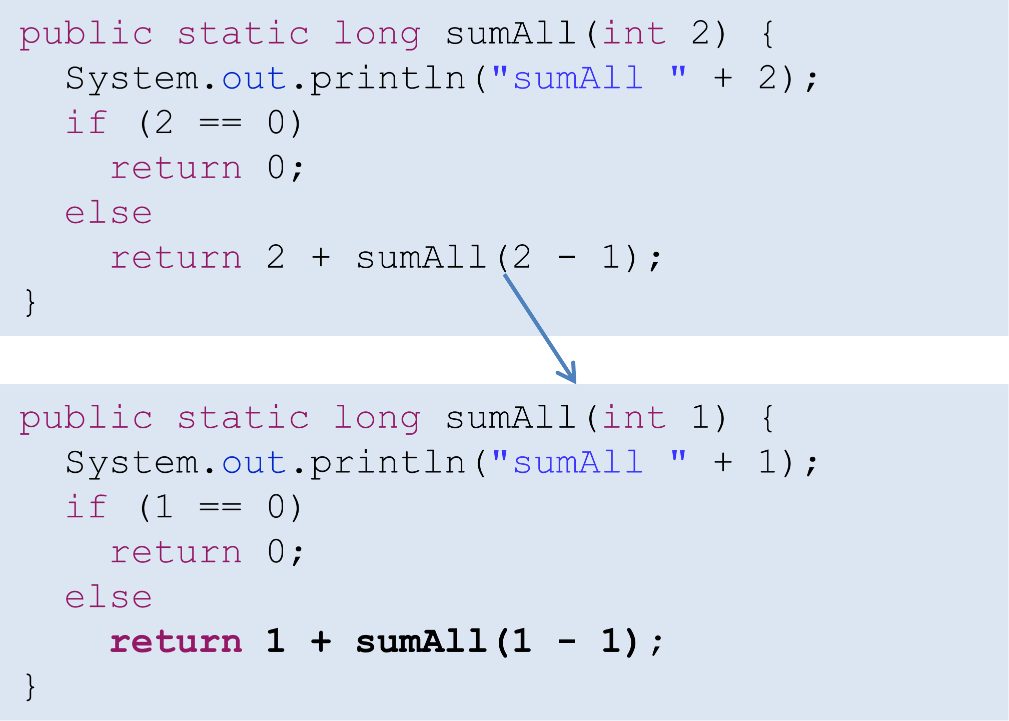
Call sumAll(2)

```
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)



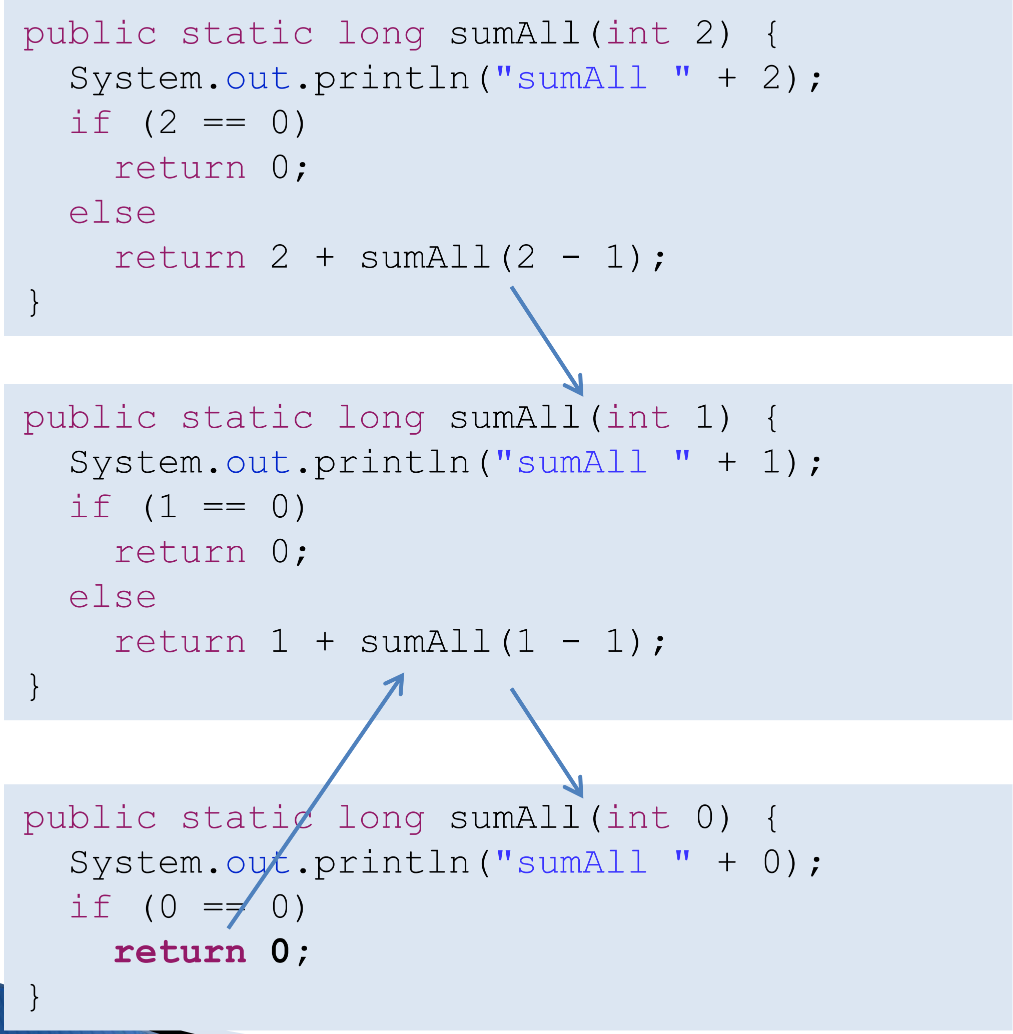
```java
public static long sumAll(int 2) {
  System.out.println("sumAll " + 2);
  if (2 == 0)
    return 0;
  else
    return 2 + sumAll(2 - 1);
}
```

```java
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);
}
```

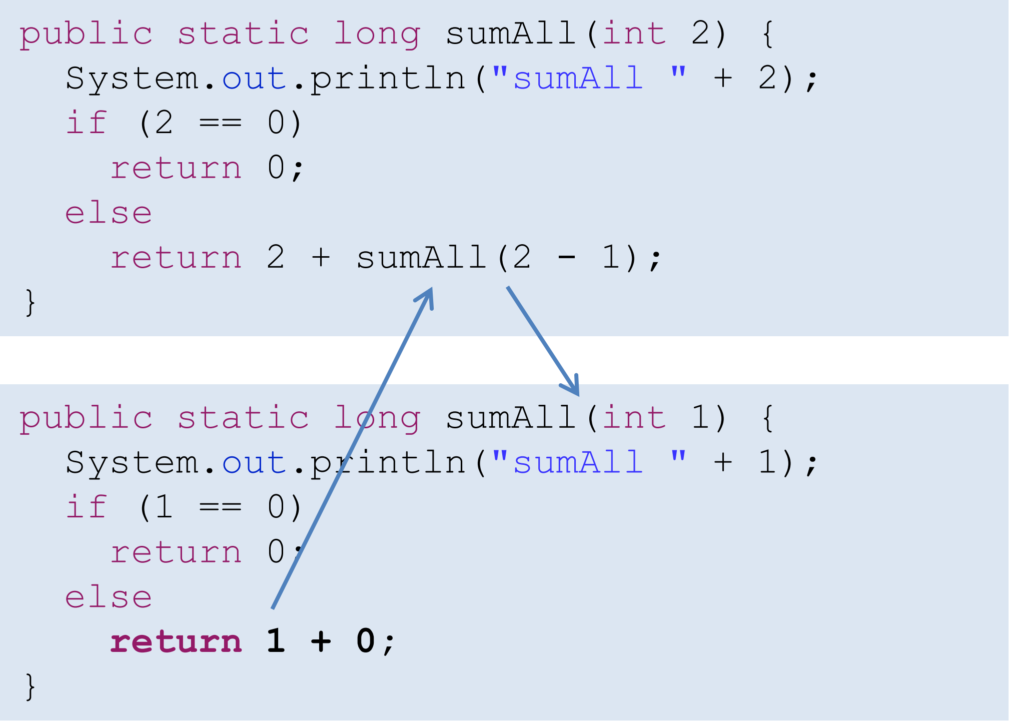
```
public static long sumAll(int 1) {
  System.out.println("sumAll " + 1);
  if (1 == 0)
    return 0;
  else
    return 1 + sumAll(1 - 1);
}
```

```
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;
}
```

OUTPUT:

```
sumAll 2
sumAll 1
sumAll 0

sumAll of 2 is 3
```

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

Factorial: definition

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

Recursive definition:

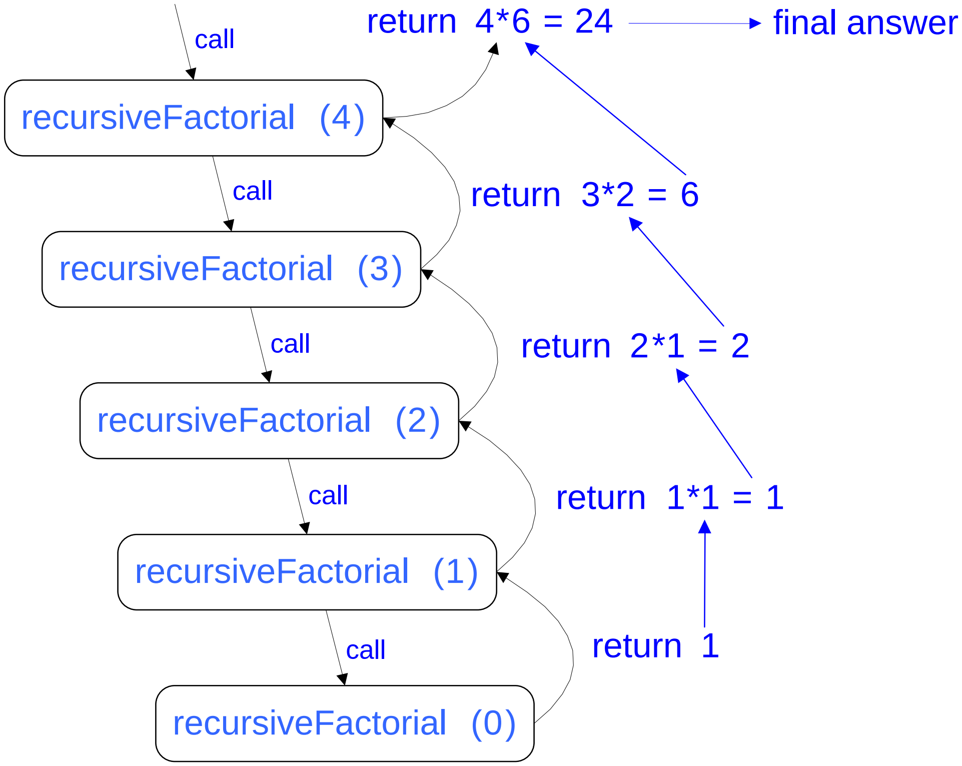
$$n! = \begin{cases} 1, & n = 0 \\ n \times (n-1)!, & n > 0 \end{cases}$$

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:

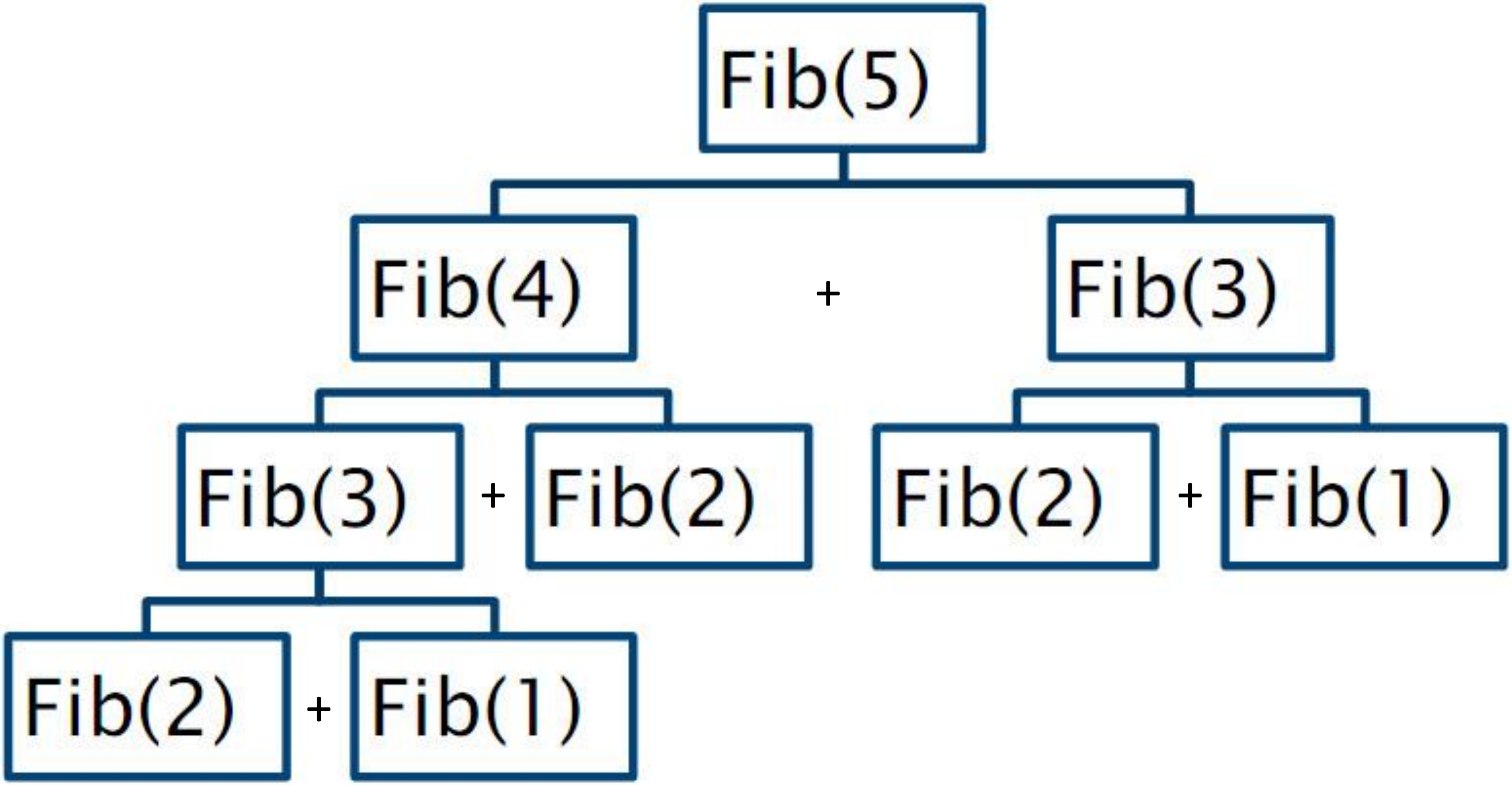
```
public static type recursive_function( … )
{
     …
     … recursive_function(…) …
     …
     … recursive_function(…) …
     …
}
```

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


Fib(5)
Fib(4)
+
Fib(3)
Fib(3)
+
Fib(2)
Fib(2)
+
Fib(1)
Fib(2)
+
Fib(1)