CSIT121

Object Oriented Design and Programming

Lesson 8

Recursion

What is Recursion?

- A function keeps calling itself until an end point. Then it begins to call back until the start.

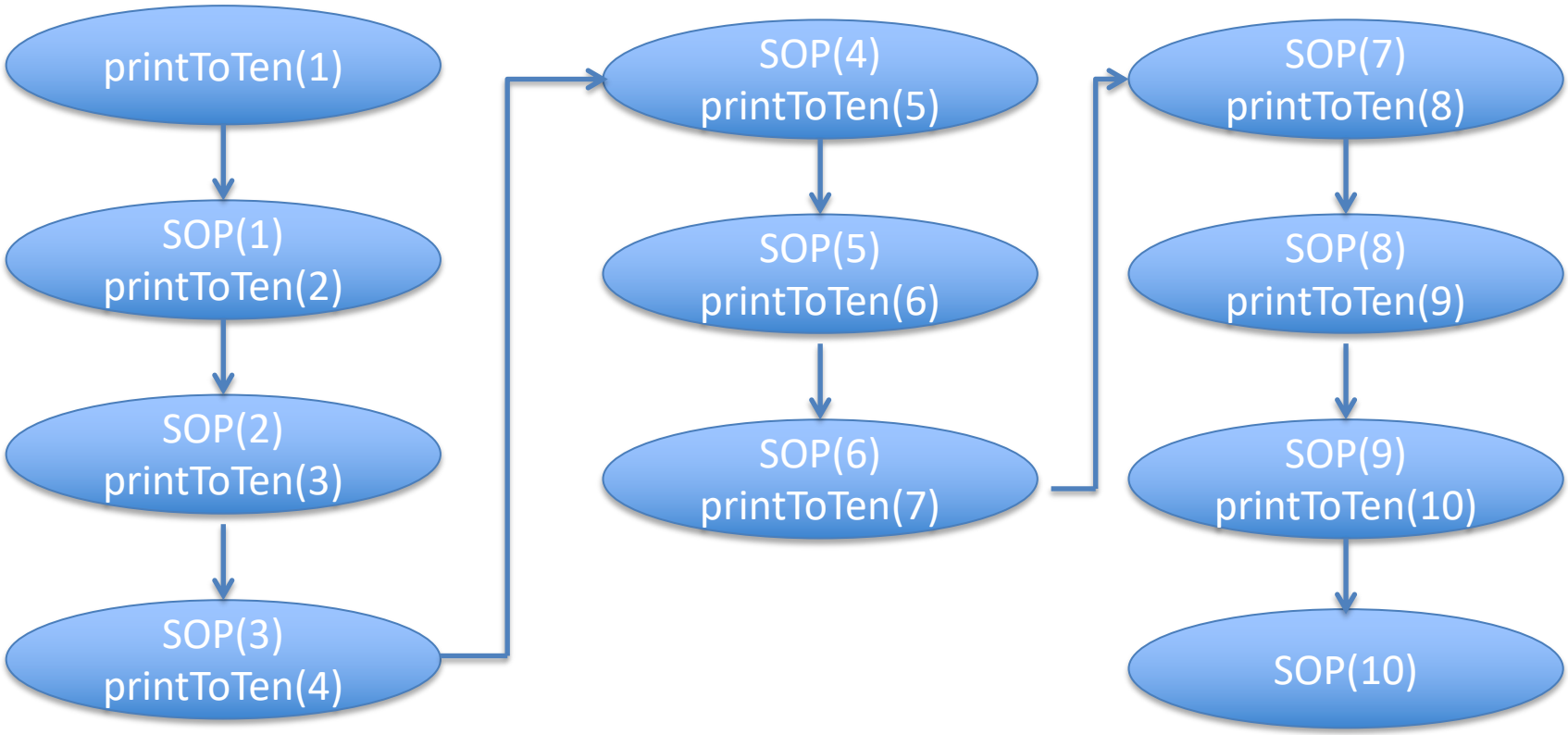
- All recursive function must have an end point. If not the function will have no end and hung the whole PC. Drain the computer memory until it hung.

Print to 10

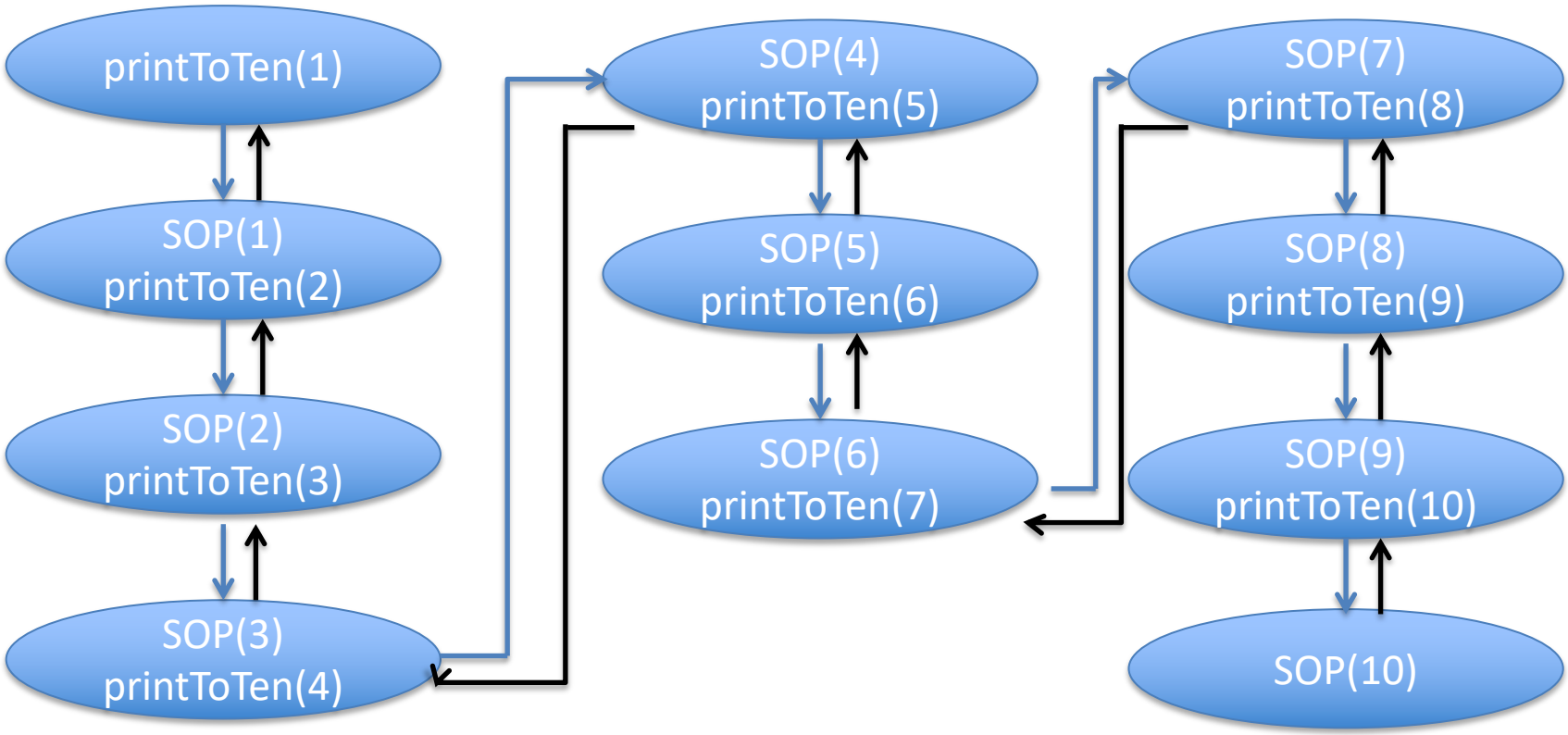
```
//Non recursive – for loop
public static void main(String [] inputs)
{
       printToTen(1);

}

public static void printToTen(int n)
{
     for(int i=n;i<=10;i++)
      {
       System.out.println(i);
      }
}
```

```
//Recursive
public static void main(String [] inputs)
{
            printToTen(1);

}

public static void printToTen(int n)
{
      if(n!=11) //end point
      {

            System.out.println(n);
            printToTen(n+1);
       }
}
```

Recursive analysis (forward call)



Recursive analysis (back call)



A memory killer apps – Recursion without end in mind

```
public static void printToTen(int n)
{
    if(n==n) //end point  that is always true
    {
        System.out.println(n);
        printToTen(n+1);
    }
}
```

Factorial – Math Revision

- The **factorial function** (symbol: **!**)
- **multiply all whole numbers** from our chosen number down to 1.

- Examples:
  - **4!** = 4 × 3 × 2 × 1 = 24
  - **7!** = 7 × 6 × 5 × 4 × 3 × 2 × 1 = 5040
  - **1!** = 1

- Note:
  - 0! = 1

Factorial – Non-Recursive

```
public static void main(String [] inputs)
{
	int answer = fact(12);
	System.out.println(answer);
}

static int factorial(int n)
{
	int fact = 1;
	for(int i=2;i<n;i++)
		fact = fact * i;

	return fact;
}
```

Recursive Factorial (forward)

- 5! = 1 x 2 x 3 x 4 x 5
- 5! = **4!** x 5 = (5-1)! x 5

Recursive Factorial (forward)

- 5! = 1 x 2 x 3 x 4 x 5
- 5! = **4!** x 5 = (5-1)! x 5
- 4! = 1 x 2 x 3 x 4
- 4! = **3!** x 4 = (4-1)! x 4

Recursive Factorial (forward)

- 5! = 1 x 2 x 3 x 4 x 5
- 5! = **4!** x 5 = (5-1)! x 5
- 4! = 1 x 2 x 3 x 4
- 4! = **3!** x 4 = (4-1)! x 4
- 3! = 1 x 2 x 3
- 3! = **2!** x 3 = (3-1)! x 3

Recursive Factorial (forward)

- 5! = 1 x 2 x 3 x 4 x 5
- 5! = **4!** x 5 = (5-1)! x 5
- 4! = 1 x 2 x 3 x 4
- 4! = **3!** x 4 = (4-1)! x 4
- 3! = 1 x 2 x 3
- 3! = **2!** x 3 = (3-1)! x 3
- 2! = 1 x 2
- 2! = **1!** x 2 = (2-1)! x 2

Recursive Factorial (forward)

- 5! = 1 x 2 x 3 x 4 x 5
- 5! = **4!** x 5 = (5-1)! x 5
- 4! = 1 x 2 x 3 x 4
- 4! = **3!** x 4 = (4-1)! x 4
- 3! = 1 x 2 x 3
- 3! = **2!** x 3 = (3-1)! x 3
- 2! = 1 x 2
- 2! = **1!** x 2 = (2-1)! x 2
- 1! = 1 (end point)

Recursive Factorial (backward)

- 5! = **4!** x 5
- 4! = **3!** x 4
- 3! = **2!** x 3
- 2! = **1!** x 2 = 1 x 2 = 2
- 1! = 1 (end point)

Recursive Factorial (backward)

- 5! = **4!** x 5
- 4! = **3!** x 4
- 3! = **2!** x 3 = 2 x 3 = 6
- 2! = **1!** x 2 = 1 x 2 = 2
- 1! = 1 (end point)

Recursive Factorial (backward)

- 5! = **4!** x 5
- 4! = **3!** x 4 = 6 x 4 = 24
- 3! = **2!** x 3 = 2 x 3 = 6
- 2! = **1!** x 2 = 1 x 2 = 2
- 1! = 1 (end point)

Recursive Factorial (backward)

- 5! = **4!** x 5 = 24 x 5 = 120
- 4! = **3!** x 4 = 6 x 4 = 24
- 3! = **2!** x 3 = 2 x 3 = 6
- 2! = **1!** x 2 = 1 x 2 = 2
- 1! = 1 (end point)

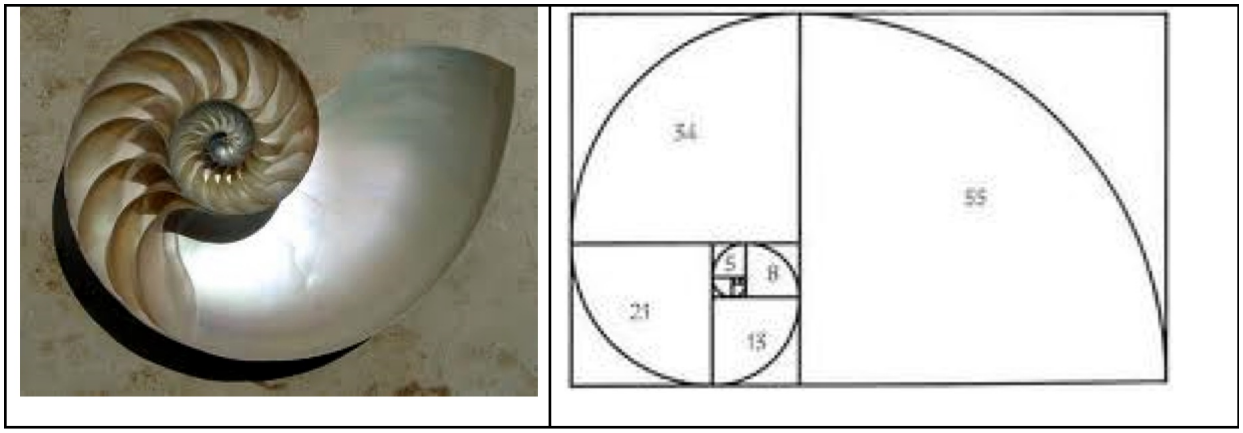
Factorial – Recursive

```
public static void main(String [] inputs)
{
    int answer = fact(12);
    System.out.println(answer);
}

public static int fact(int n)
{
    if (n==0) return 1;  //end point
    else return n*fact(n-1);
}
```

Fibonacci Numbers

- The first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

n	0	1	2	3	4	5	6	7	……
fibo	0	1	1	2	3	5	8	13	…….



Fibonacci Numbers – Non-Recursive

```
public static void main(String [] inputs){
        int answer = fiboNumber(7);
        System.out.println(answer);
}

public static int fiboNumber(int n){
        int fibo = 0;
        if (n <= 1)
        {  fibo = n; }
        else
        {
            int a = 0, b = 1;
            for(int i = 2;i<=n;i++)
            {
                fibo=a+b;
                a=b;
                b=fibo;
             }
        }
        return fibo;
}
```

Fibonacci Numbers – Recursive

```
public static void main(String [] inputs)
{
        int answer = fiboNumber(7);
        System.out.println(answer);
}

public static int fiboNumber(int n)
{
      if (n <= 1)
      {
        return n;
      }
      else
      {
         return fiboNumber(n-1) + fiboNumber(n-2);
      }
}
```

Fibonacci Numbers
Recursive Trace (forward)

- fibo(5) = fibo(4) + fibo(3)
- fibo(4) = fibo(3) + fibo(2)
- fibo(3) = fibo(2) + fibo(1)
- fibo(2) = fibo(1) + fibo(0)
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Fibonacci Numbers
Recursive Trace (backward)

- fibo(5) = fibo(4) + fibo(3)
- fibo(4) = fibo(3) + fibo(2)
- fibo(3) = fibo(2) + fibo(1)
- fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Fibonacci Numbers
Recursive Trace (backward)

- fibo(5) = fibo(4) + fibo(3)
- fibo(4) = fibo(3) + fibo(2)
- fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2
- fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Fibonacci Numbers
Recursive Trace (backward)

- fibo(5) = fibo(4) + fibo(3)
- fibo(4) = fibo(3) + fibo(2)
- fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2
- fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Fibonacci Numbers
Recursive Trace (backward)

- fibo(5) = fibo(4) + fibo(3)
- fibo(4) = fibo(3) + fibo(2) = 2 + 1 = 3
- fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2
- fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Fibonacci Numbers
Recursive Trace (backward)

- fibo(5) = fibo(4) + fibo(3) = 3 + 2 = 5
- fibo(4) = fibo(3) + fibo(2) = 2 + 1 = 3
- fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2
- fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1
- fibo(1) = 1 (end point)
- fibo(0) = 0 (end point)

Advantages and disadvantages of using recursive function

- Advantage:
  - Shorter and elegant codes.

- Disadvantage:
  - Might be difficult to understand and debug. (Danger of endless recursive)
  - More runtime memory required.