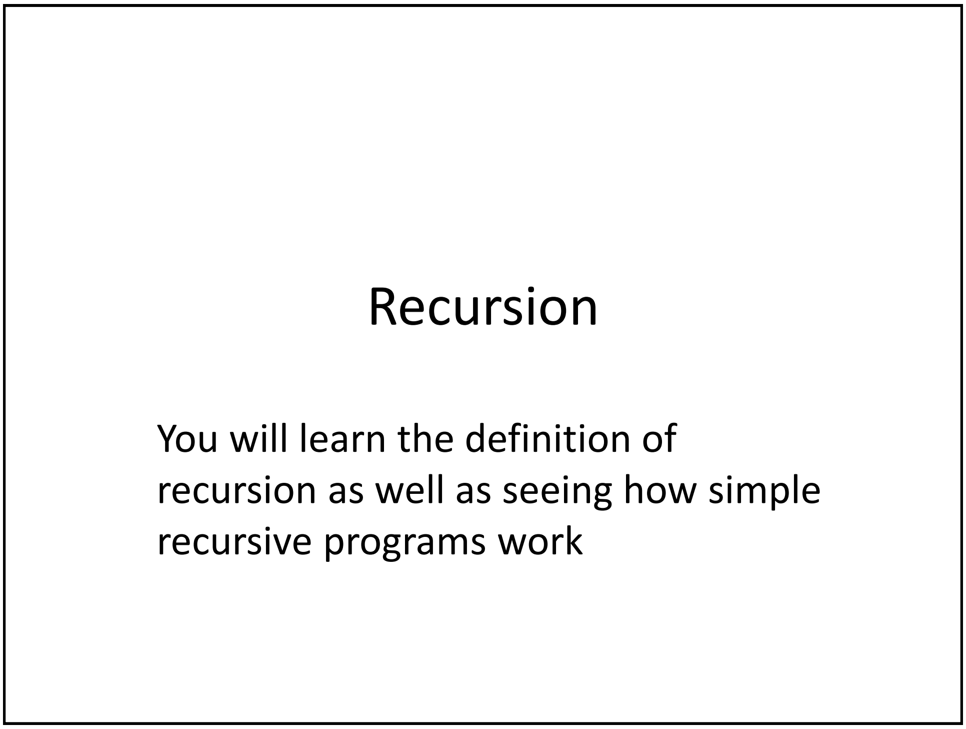
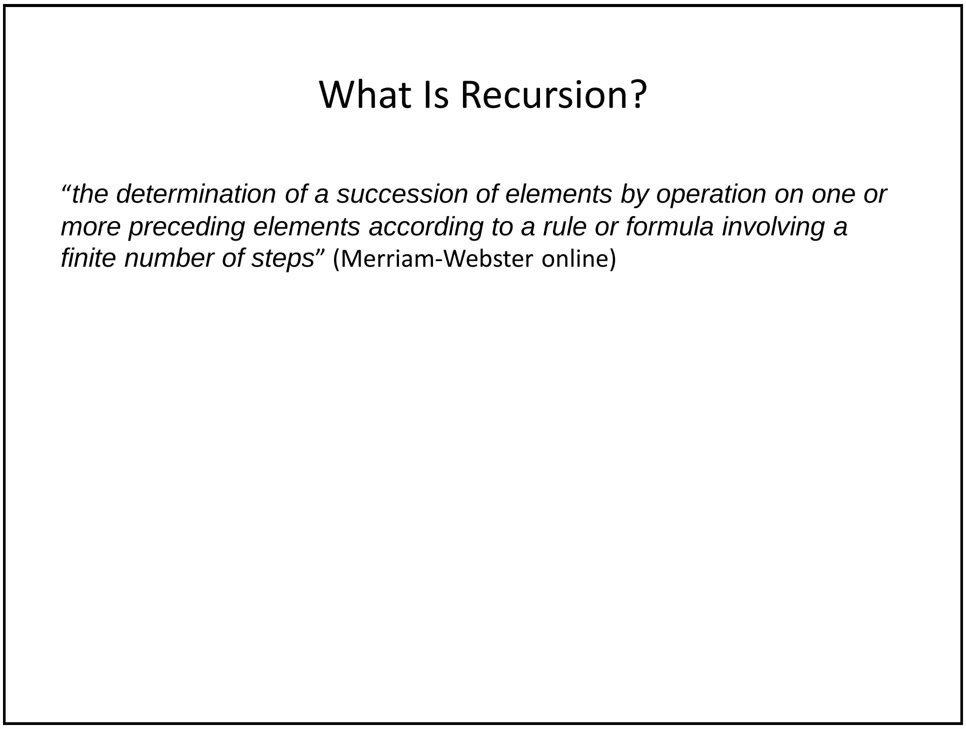


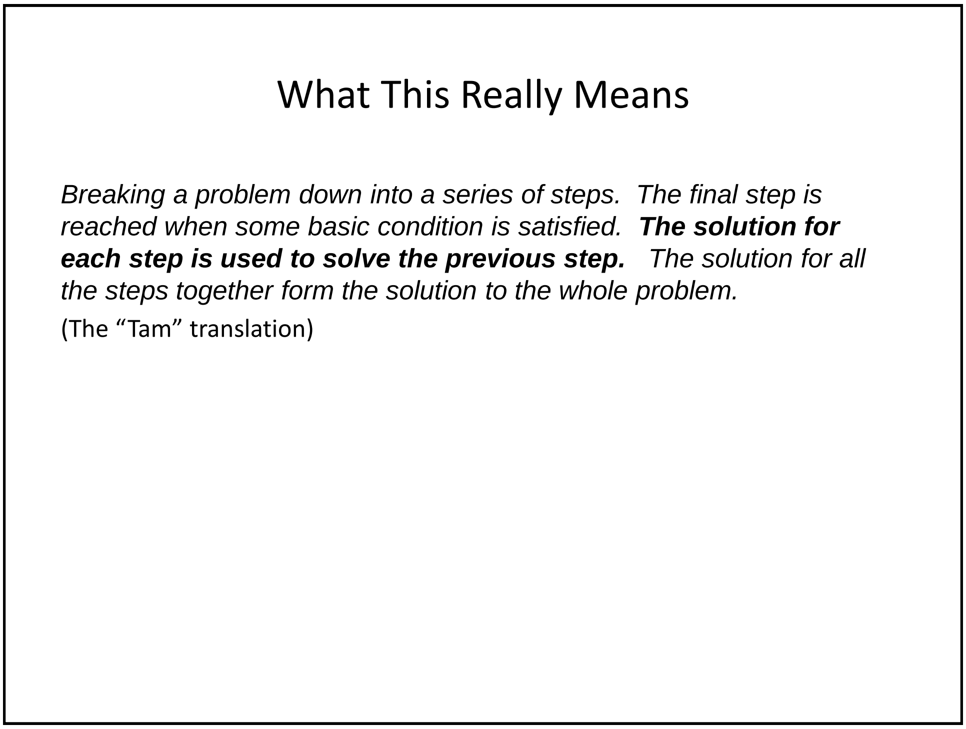
Recursion

You will learn the definition of recursion as well as seeing how simple recursive programs work



What Is Recursion?

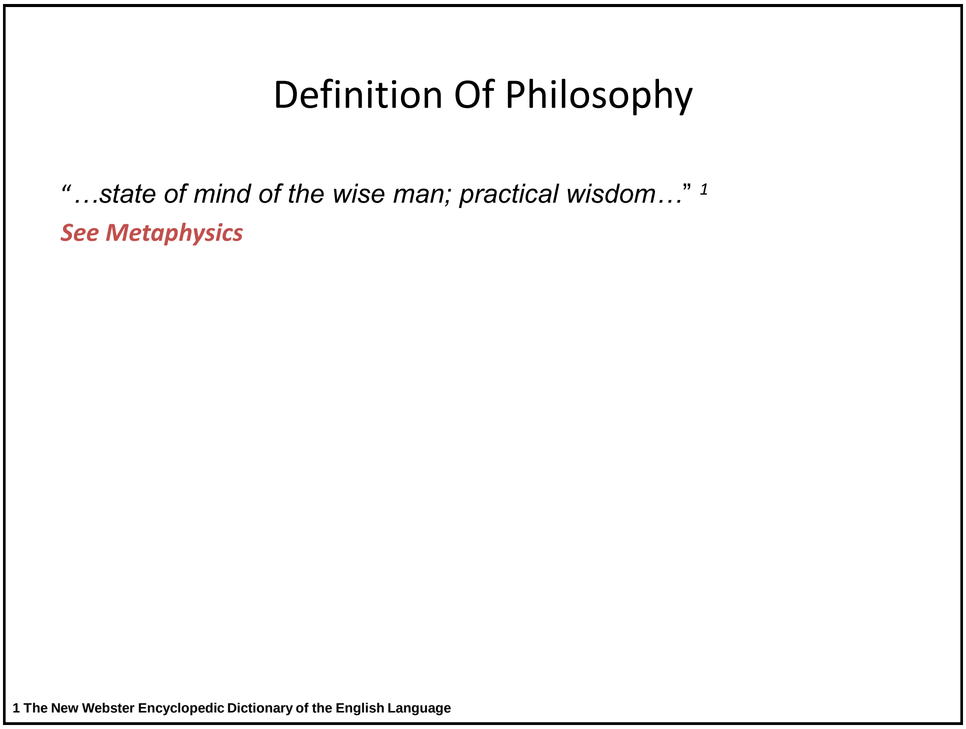
"the determination of a succession of elements by operation on one or more preceding elements according to a rule or formula involving a finite number of steps" (Merriam-Webster online)



What This Really Means

Breaking a problem down into a series of steps. The final step is reached when some basic condition is satisfied. ***The solution for each step is used to solve the previous step.*** *The solution for all the steps together form the solution to the whole problem.*

(The “Tam” translation)



Definition Of Philosophy

“…state of mind of the wise man; practical wisdom…” [1]

See Metaphysics

[1] The New Webster Encyclopedic Dictionary of the English Language

Metaphysics

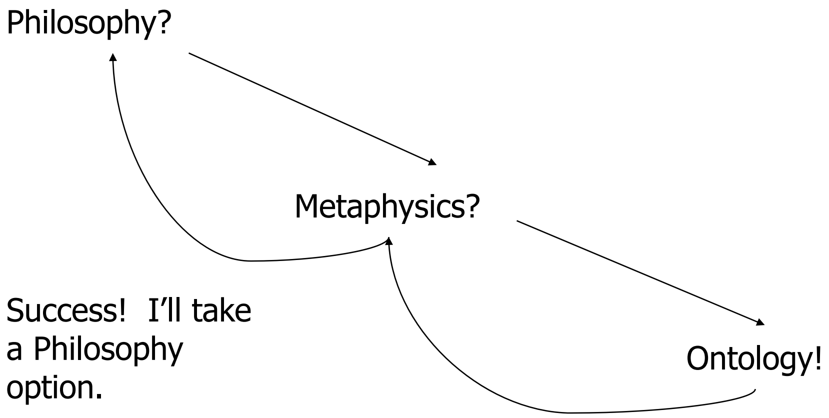
"…know the ultimate grounds of being or what it is that really exists, embracing both psychology and ***ontology.****"* [2]

[2] The New Webster Encyclopedic Dictionary of the English Language

Result Of Lookup , Possibility One: Success

- I know what Ontology means!

Result Of Lookup, Possibility One



Philosophy?

Metaphysics?

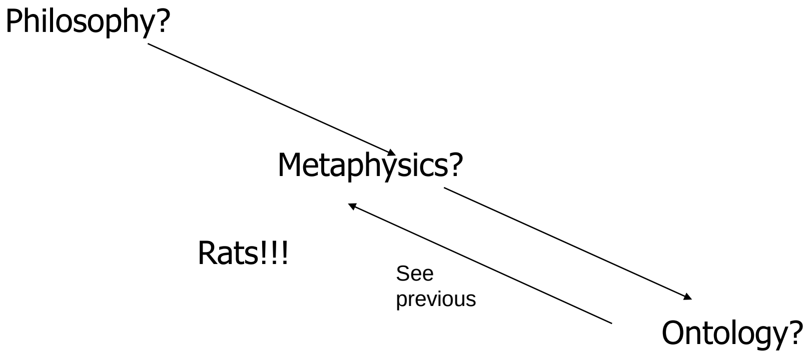
Ontology!

Success! I'll take a Philosophy option.

Result Of Lookup, Possibility Two: Failure

- Lookup 'loops' back.

Result Of Lookup, Possibility Two



Philosophy?

Metaphysics?

Rats!!!

See previous

Ontology?

Ontology

"…equivalent to metaphysics."[3]

[3] The New Webster Encyclopedic Dictionary of the English Language

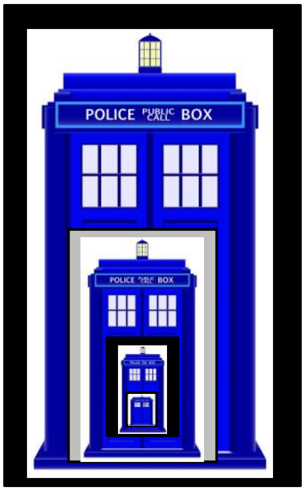
Wav file from Tam

Result Of Lookup, Possibility Three: Failure

- You’ve looked up everything and still don’t know the definition!

Related Material: Recursion

- *“A programming technique whereby a function or method calls itself either directly or indirectly.”*

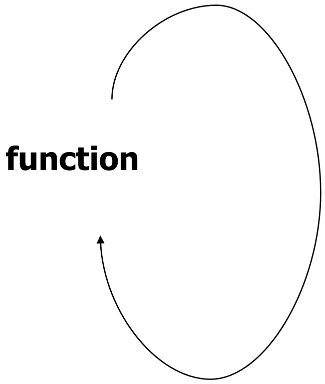


‘Tardis’ images: colourbox.com

Looking Up A Word

```
if (you completely understand a definition) then
     return to previous definition (using the definition that's
     understood)
else
    lookup (unknown word(s))
```

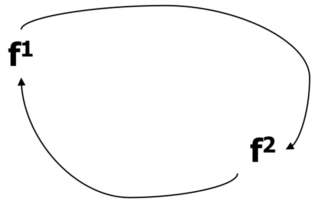
Direct Call



function

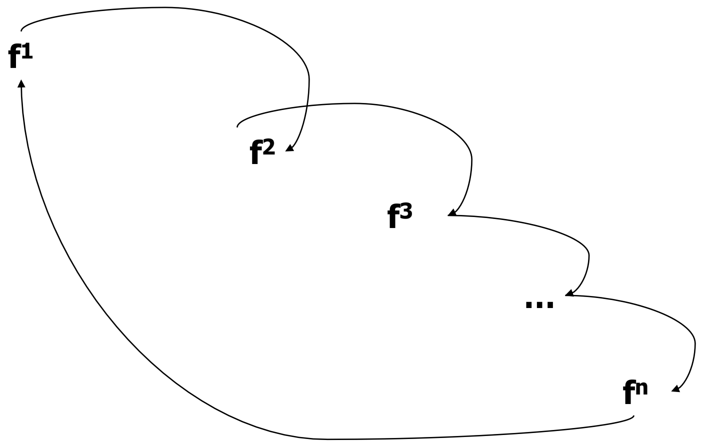
```
void fun()
   ...
   fun();
```

Indirect Call



James Tam

Indirect Call



James Tam

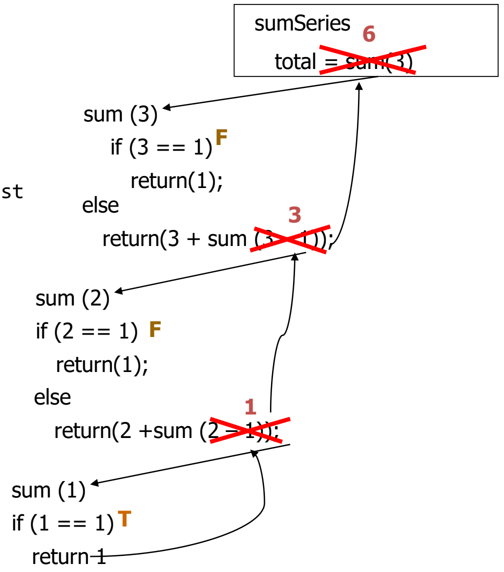
Requirements For *Sensible* Recursion

1) Base case

2) Progress is made (towards the base case)

Example Program: `SumSeries.java`

```
static int sum(int no) {
   if (no == 1)
      return(1);
   else:
      return(no + sum(no-1));
}

main(String args []) {
    ...
    System.out.print("Enter the last
      number: ");
    last = in.nextInt();
    total = sum(last);
    System.out.println("The sum of
      the series from " +
      "1 to " + last + " is " +
      total);
}
```



sumSeries

6

total = ~~sum(3)~~

sum (3)

if (3 == 1) F

return(1);

else

3

return(3 + sum ~~(3 – 1)~~);

sum (2)

if (2 == 1) F

return(1);

else

1

return(2 +sum ~~(2 – 1)~~);

sum (1)

if (1 == 1) T

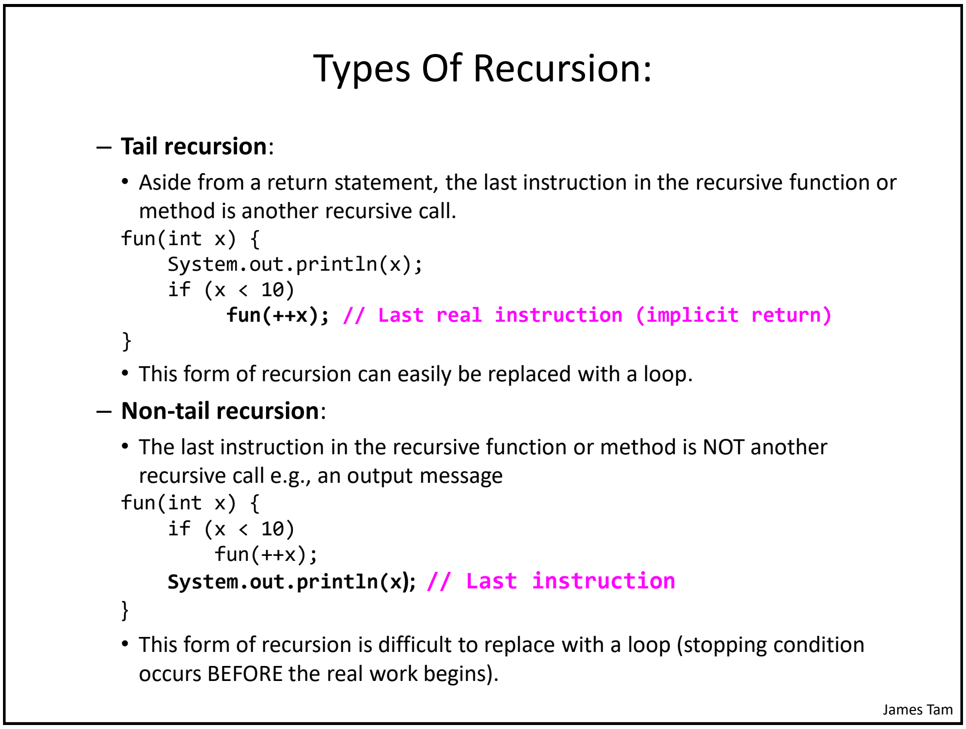
return 1

When To Use Recursion

- When a problem can be divided into steps.
- The result of one step can be used in a previous step.
- There is a scenario when you can stop sub-dividing the problem into steps (step = recursive call) and return to a previous step.
  - Algorithm goes back to previous step with a partial solution to the problem (back tracking)
- All of the results together solve the problem.

When To Consider Alternatives To Recursion

- When a loop will solve the problem just as well
- Types of recursion (for both types a `return` statement is excepted)
  - **Tail recursion**
    - The last statement in the function is another recursive call to that function This form of recursion can easily be replaced with a loop.
  - **Non-tail recursion**
    - The last statement in the recursive function is not a recursive call.
    - This form of recursion is very difficult (read: impossible) to replace with a loop.



Types Of Recursion:

- **Tail recursion**:
  - Aside from a return statement, the last instruction in the recursive function or method is another recursive call.

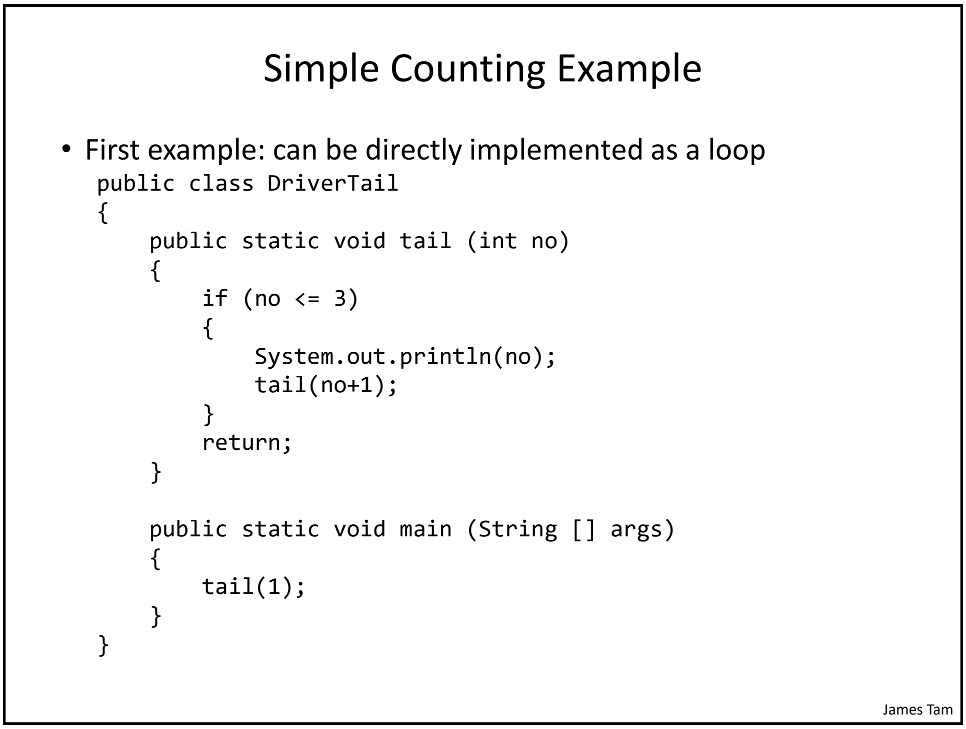
```
fun(int x) {
    System.out.println(x);
    if (x < 10)
         fun(++x); // Last real instruction (implicit return)
}
```

  - This form of recursion can easily be replaced with a loop.
- **Non-tail recursion**:
  - The last instruction in the recursive function or method is NOT another recursive call e.g., an output message

```
fun(int x) {
    if (x < 10)
        fun(++x);
    System.out.println(x); // Last instruction
}
```

  - This form of recursion is difficult to replace with a loop (stopping condition occurs BEFORE the real work begins).

James Tam



Simple Counting Example

- First example: can be directly implemented as a loop

```
public class DriverTail
{
    public static void tail (int no)
    {
        if (no <= 3)
        {
            System.out.println(no);
            tail(no+1);
        }
        return;
    }

    public static void main (String [] args)
    {
        tail(1);
    }
}
```

James Tam

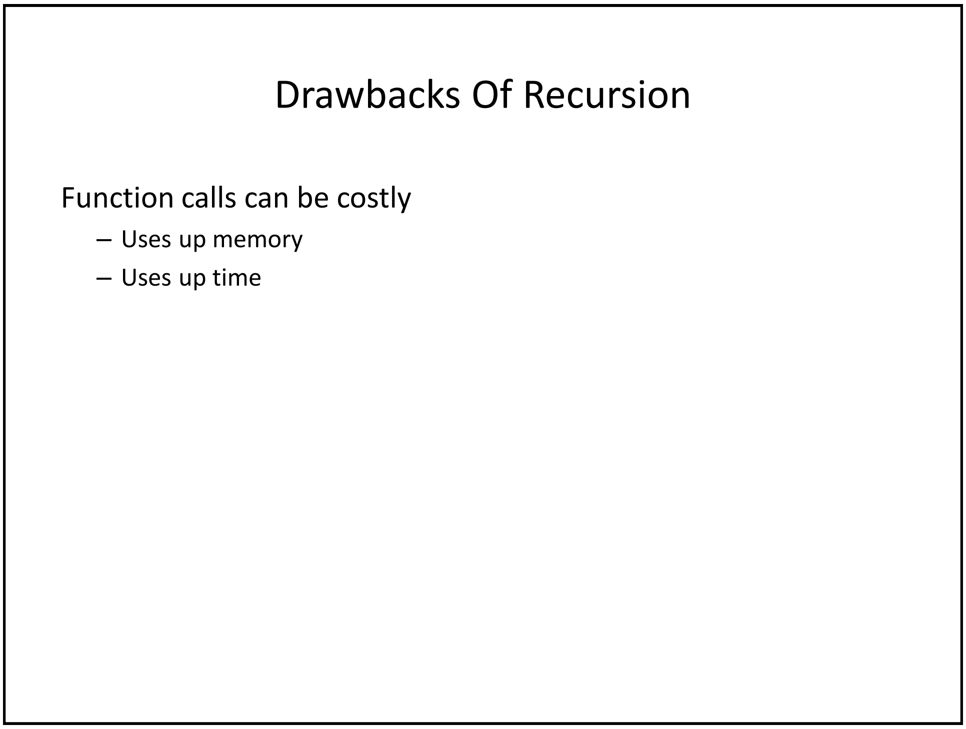
‘Reversed’ Counting Example

```
public class DriverNonTail
{
    public static void nonTail(int no)
    {
        if (no < 3)
            nonTail(no+1);
        System.out.println(no);
        return;
    }

    public static void main (String [] args)
    {
        nonTail(1);
    }
}
```

Error Handling Example Using Recursion (2)

– Iterative/looping solution (day must be between 1 – 31)

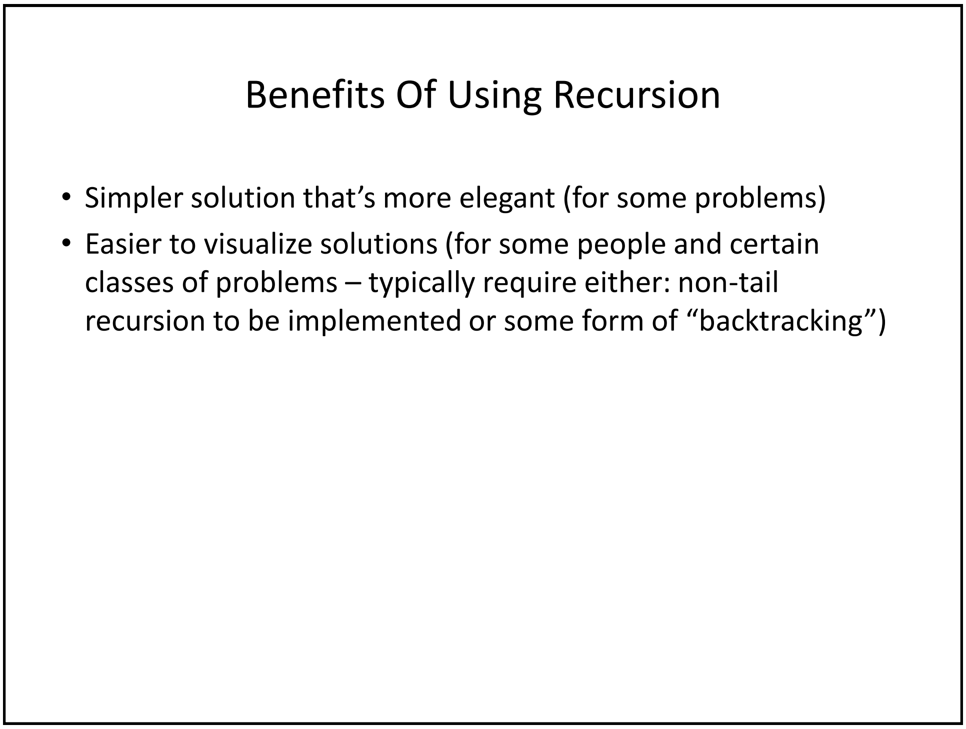
```
public int promptDay() {
    int day = -1;
    Scanner in = new Scanner(System.in);
    System.out.print("Enter day of birth (1-31): ");
    day = in.nextInt();
    if ((day < 1) or (day > 31)) {
        day = promptDay()
    }
    return(day);

...
day = promptDay()
```



Drawbacks Of Recursion

Function calls can be costly

- Uses up memory
- Uses up time



Benefits Of Using Recursion

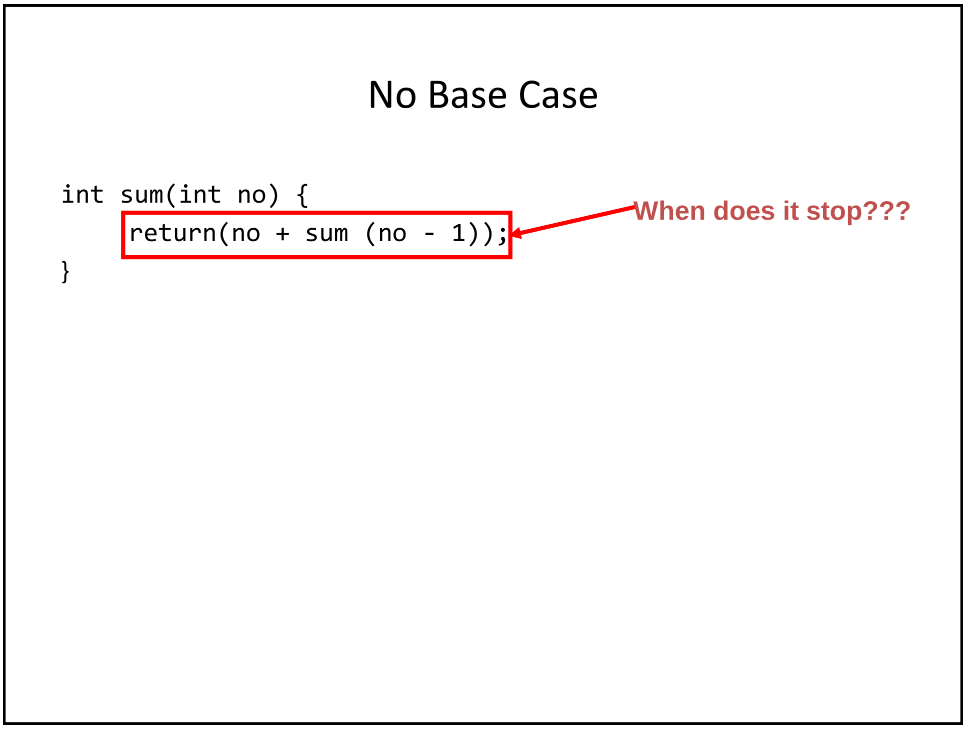
- Simpler solution that’s more elegant (for some problems)
- Easier to visualize solutions (for some people and certain classes of problems – typically require either: non-tail recursion to be implemented or some form of “backtracking”)

Common Pitfalls When Using Recursion

- These three pitfalls can result in a runtime error
  - No base case
  - No progress towards the base case
  - Using up too many resources (e.g., variable declarations) for each function call

No Base Case

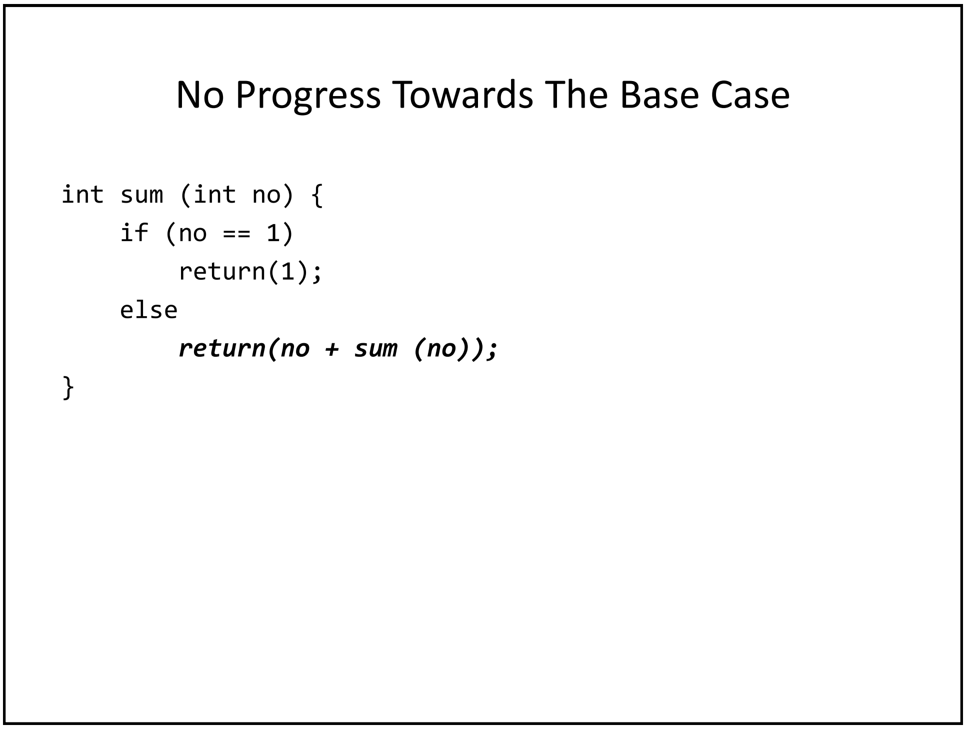
```
int sum(int no) {
     return(no + sum (no - 1));
}
```



No Base Case

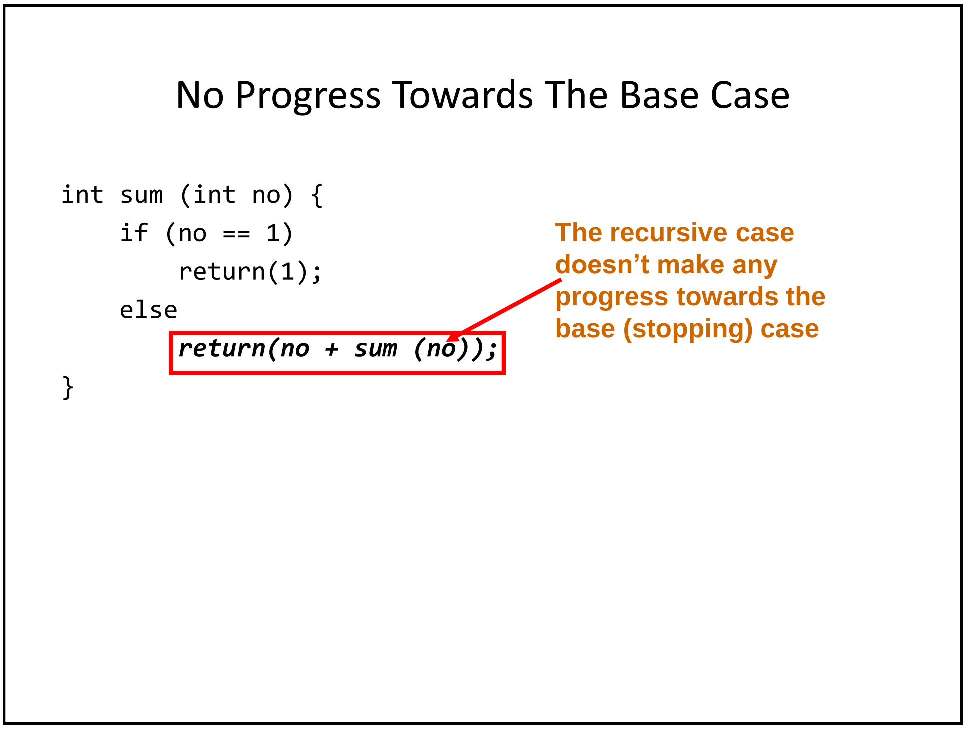
```
int sum(int no) {
     return(no + sum (no - 1));
}
```

When does it stop???



No Progress Towards The Base Case

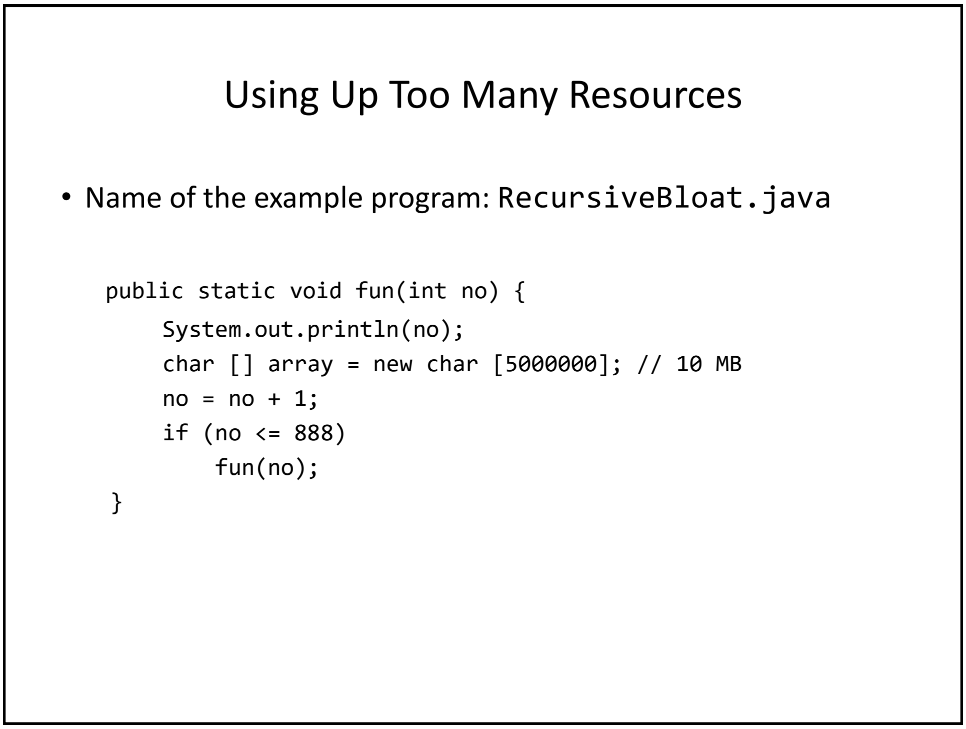
```
int sum (int no) {
     if (no == 1)
          return(1);
     else
          return(no + sum (no));
}
```



No Progress Towards The Base Case

```
int sum (int no) {
    if (no == 1)
        return(1);
    else
        return(no + sum (no));
}
```

The recursive case doesn’t make any progress towards the base (stopping) case



Using Up Too Many Resources

- Name of the example program: `RecursiveBloat.java`

```
public static void fun(int no) {
    System.out.println(no);
    char [] array = new char [5000000]; // 10 MB
    no = no + 1;
    if (no <= 888)
        fun(no);
}
```

Undergraduate Student Definition Of Recursion

Word: **re·cur·sion**

Pronunciation: ri-'k&r-zh&n

Definition: See recursion

Wav file courtesy of "James Tam"

Recursion: Job Interview Question

- http://www.businessinsider.com/apple-interview-questions-2011-5#write-a-function-that-calculates-a-numbers-factorial-using-recursion-9

You Should Now Know

- What is a recursive computer program
- How to write and trace simple recursive programs
- What are the requirements for recursion/What are the common pitfalls of recursion

After This Section You Should Now Know

- What is a linked list and how it differs from an array implementation
- How to implement basic list operations using a linked list
  - Creation of new empty list
  - Destruction of the entire list
  - Display of list elements (iterative and recursive)
  - Searching the list
  - Inserting new elements
  - Removing existing elements
- How to write a recursive equivalent of an iterative solution
- What is the benefit of a recursive vs. iterative implementation
  - What is backtracking
- How to trace a recursive program
  - Programs that are the equivalent of an iterative solutions
  - Programs that employ backtracking

James Tam