Building Java Programs

Chapter 12
introduction to recursion

reading: 12.1
## Road Map

### CS Concepts
- Client/Implementer
- Efficiency
- **Recursion**
- Regular Expressions
- Grammars
- Sorting
- Backtracking
- Hashing
- Huffman Compression

### Java Language
- Exceptions
- Interfaces
- References
- Comparable
- Generics
- Inheritance/Polymorphism
- Abstract Classes

### Data Structures
- Lists
- Stacks
- Queues
- Sets
- Maps
- Priority Queues

### Java Collections
- Arrays
- ArrayList
- LinkedList
- Stack
- TreeSet / TreeMap
- HashSet / HashMap
- PriorityQueue
Exercise

• (To a student in the front row)
How many students total are directly behind you in your "column" of the classroom?

• You have poor vision, so you can see only the people right next to you. So you can't just look back and count.

• But you are allowed to ask questions of the person next to you.

• How can we solve this problem? (recursively )
The idea

- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
  - Each person can solve a small part of the problem.
    - What is a small version of the problem that would be easy to answer?
    - What information from a neighbor might help me?
Recursive algorithm

- Number of people behind me:
  - If there is someone behind me, ask him/her how many people are behind him/her.
    - When they respond with a value $N$, then I will answer $N + 1$.
  - If there is nobody behind me, I will answer 0.
Recursion

- **recursion**: The definition of an operation in terms of itself.
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.

- **recursive programming**: Writing methods that call themselves to solve problems recursively.
  - An equally powerful substitute for *iteration* (loops)
  - Particularly well-suited to solving certain types of problems
Why learn recursion?

- "Cultural experience" – think differently about problems
- Solves some problems more naturally than iteration
- Can lead to elegant, simplistic, short code (when used well)
- Many programming languages ("functional" languages such as Scheme, ML, and Haskell) use recursion exclusively (no loops)
- A key component of many of our assignments in CSE 143
Getting down stairs

• Need to know two things:
  • Getting down one stair
  • Recognizing the bottom

• Most code will look like:

```java
if (simplest case) {
    compute and return solution
} else {
    divide into similar subproblem(s)
    solve each subproblem recursively
    assemble the overall solution
}
```
Recursion and cases

Every recursive algorithm involves at least 2 cases:

- **base case**: A simple occurrence that can be answered directly.

- **recursive case**: A more complex occurrence of the problem that cannot be directly answered, but can instead be described in terms of smaller occurrences of the same problem.

- Some recursive algorithms have more than one base or recursive case, but all have at least one of each.

- A crucial part of recursive programming is identifying these cases.
Linked Lists are Self-Similar

• a linked list is:
  • null
  • a node whose next field references a list

• recursive data structure: a data structure partially composed of smaller or simpler instances of the same data structure
Another recursive task

- How can we remove exactly half of the M&M's in a large bowl, without dumping them all out or being able to count them?
  - What if multiple people help out with solving the problem? Can each person do a small part of the work?
  - What is a number of M&M's that it is easy to double, even if you can't count?
    - (What is a "base case"?)
Recursion in Java

- Consider the following method to print a line of * characters:

```java
// Prints a line containing the given number of stars.
// Precondition: n >= 0
public static void printStars(int n) {
    for (int i = 0; i < n; i++) {
        System.out.print("*");
    }
    System.out.println(); // end the line of output
}
```

- Write a recursive version of this method (that calls itself).
- Solve the problem without using any loops.
- Hint: Your solution should print just one star at a time.
A basic case

- What are the cases to consider?
  - What is a very easy number of stars to print without a loop?

```java
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else {
        ...
    }
}
```
Handling more cases

- Handling additional cases, with no loops (in a bad way):

```java
class PrintStars {
    public static void printStars(int n) {
        if (n == 1) {
            // base case; just print one star
            System.out.println("*");
        } else if (n == 2) {
            System.out.print("*");
            System.out.println("*");
        } else if (n == 3) {
            System.out.print("*");
            System.out.print("*");
            System.out.println("*");
        } else if (n == 4) {
            System.out.print("*");
            System.out.print("*");
            System.out.print("*");
            System.out.println("*");
        } else ...
    }
}
```
Handling more cases 2

- Taking advantage of the repeated pattern (somewhat better):

```java
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else if (n == 2) {
        System.out.print("*");
        printStars(1);  // prints "*"
    } else if (n == 3) {
        System.out.print("*");
        printStars(2);  // prints "**"
    } else if (n == 4) {
        System.out.print("*");
        printStars(3);  // prints "***"
    } else ...
}
```
Using recursion properly

- Condensing the recursive cases into a single case:

```java
public static void printStars(int n) {
    if (n == 1) {
        // base case; just print one star
        System.out.println("*");
    } else {
        // recursive case; print one more star
        System.out.print("*");
        printStars(n - 1);
    }
}
```
The real, even simpler, base case is an \( n \) of 0, not 1:

```java
public static void printStars(int n) {
    if (n == 0) {
        // base case; just end the line of output
        System.out.println();
    } else {
        // recursive case; print one more star
        System.out.print("*");
        printStars(n - 1);
    }
}
```

Recursion Zen: The art of properly identifying the best set of cases for a recursive algorithm and expressing them elegantly. (A CSE 143 informal term)
Recursion vs Iteration

```java
public static void writeStars(int n) {
    while (n > 0) {
        System.out.print("*");
        n--;
    }
    System.out.println();
}

public static void writeStars(int n) {
    if (n == 0) {
        System.out.println();
    } else {
        System.out.print("*");
        writeStars(n - 1);
    }
}
```
Recursion vs Iteration

public static void writeStars(int n) {
    while (n > 0) {
        System.out.print("*");
        n--;
    }
    System.out.println(); // base case. assert: n == 0
}

public static void writeStars(int n) {
    if (n == 0) {
        System.out.println(); // base case
    } else {
        System.out.print("*");
        writeStars(n - 1);
    }
}
public static void writeStars(int n) {
    while (n > 0) { // "recursive" case
        System.out.print("*"); // small piece of problem
        n--;
    }
    System.out.println();
}

public static void writeStars(int n) {
    if (n == 0) {
        System.out.println();
    } else { // "recursive" case. assert: n > 0
        System.out.print("*"); // small piece of problem
        writeStars(n - 1);
    }
}
public static void writeStars(int n) {
    while (n > 0) {  // "recursive" case
        System.out.print("*");
        n--; // make the problem smaller
    }
    System.out.println();
}

public static void writeStars(int n) {
    if (n == 0) {
        System.out.println();
    } else {  // "recursive" case. assert: n > 0
        System.out.print("*");
        writeStars(n - 1); // make the problem smaller
    }
}
Recursive tracing

Consider the following recursive method:

```java
public static int mystery(int n) {
    if (n < 10) {
        return n;
    } else {
        int a = n / 10;
        int b = n % 10;
        return mystery(a + b);
    }
}
```

What is the result of the following call?

```java
mystery(648)
```
A recursive trace

mystery(648):
- int a = 648 / 10; // 64
- int b = 648 % 10; // 8
- return mystery(a + b); // mystery(72)

mystery(72):
- int a = 72 / 10; // 7
- int b = 72 % 10; // 2
- return mystery(a + b); // mystery(9)

mystery(9):
- return 9;
Recursive tracing 2

Consider the following recursive method:

```java
public static int mystery(int n) {
    if (n < 10) {
        return (10 * n) + n;
    } else {
        int a = mystery(n / 10);
        int b = mystery(n % 10);
        return (100 * a) + b;
    }
}
```

What is the result of the following call?
`mystery(348)`
A recursive trace 2

mystery(348)
- int a = mystery(34);
  - int a = mystery(3);
    - return (10 * 3) + 3; // 33
  - int b = mystery(4);
    - return (10 * 4) + 4; // 44
    - return (100 * 33) + 44; // 3344
- int b = mystery(8);
  - return (10 * 8) + 8; // 88
  - return (100 * 3344) + 88; // 334488

What is this method really doing?
Exercise

• Note: We did `reverseDeck` in lecture but they are the **exact same problem**
• Write a recursive method `reverseLines` that accepts a file `Scanner` and prints the lines of the file in reverse order.

  - Example input file: 
    
    I have eaten  
    the plums  
    that were in  
    the icebox  

  - Expected console output: 
    
    the icebox  
    that were in  
    the plums  
    I have eaten  

  - What are the cases to consider? 
    • How can we solve a small part of the problem at a time? 
    • What is a file that is very easy to reverse?
Reversal pseudocode

- Reversing the lines of a file:
  - Read a line L from the file.
  - Print the rest of the lines in reverse order.
  - Print the line L.

- If only we had a way to reverse the rest of the lines of the file....
public static void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        // recursive case
        String line = input.nextLine();
        reverseLines(input);
        System.out.println(line);
    }
}

- Where is the base case?
Tracing our algorithm

- **call stack:** The method invocations currently running

```java
reverseLines(new Scanner("poem.txt"));
```

```java
public static void reverseLines(Scanner input) {
  if (input.hasNextLine()) {
    String line = input.nextLine(); // "I have eaten"
  }
}
```

```java
public static void reverseLines(Scanner input) {
  if (input.hasNextLine()) {
    String line = input.nextLine(); // "the plums"
  }
}
```

```java
public static void reverseLines(Scanner input) {
  if (input.hasNextLine()) {
    String line = input.nextLine(); // "that were in"
  }
}
```

```java
public static void reverseLines(Scanner input) {
  if (input.hasNextLine()) {
    String line = input.nextLine(); // "the icebox"
  }
}
```

```java
public static void reverseLines(Scanner input) {
  if (input.hasNextLine()) {
    // false
    ...
  }
}
```

I have eaten the plums that were in the icebox