

Building Java Programs

Chapter 12
introduction to recursion

reading: 12.1

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

Getting down stairs



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

- Most code will look like:

```
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.

Recursion vs Iteration

```
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);  
    }  
}
```

Recursion vs Iteration

```
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);  
    }  
}
```


Recursion vs Iteration

```
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  
    }  
}
```


Exercise

- Write a recursive method `reverseLines` that accepts a `FileScanner` 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?

Tracing our algorithm

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

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

```
public static void reverseLines(Scanner input) {  
    if (input.hasNextLine()) {  
        String line = input.nextLine(); // "I have eaten"  
public static void reverseLines(Scanner input) {  
    if (input.hasNextLine()) {  
        String line = input.nextLine(); // "the plums"  
public static void reverseLines(Scanner input) {  
    if (input.hasNextLine()) {  
        String line = input.nextLine(); // "that were in"  
public static void reverseLines(Scanner input) {  
    if (input.hasNextLine()) {  
        String line = input.nextLine(); // "the icebox"  
public static void reverseLines(Scanner input) {  
    if (input.hasNextLine()) { // false  
        ...  
    }  
}
```

I have eaten
the plums
that were in
the icebox

the icebox
that were in
the plums
I have eaten