Building Java Programs Chapter 12

Recursion

A small set of the total slides just to demonstrate – this is a more advanced topic (CSE 143)

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" - A different way of thinking of problems

- Can solve some kinds of problems better than iteration

- Leads 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 the rest of our assignments in CSE 143

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 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 <u>without using any loops</u>.
  - 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
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):

```
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);
    }
}
```

"Recursion Zen"

- 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)

Recursive tracing

- Consider the following recursive method:

```
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?
  ```
  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;
```