

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

2006 Feb 17 Sat

1. Review

Questions from any previous lecture.

Questions about any programming assignment.

2. Recursion

In computer science, recursion is when a method calls itself as part of its execution. It is essentially a form of mathematical induction in programming language form.

In mathematical induction, we have 2 ideas:

- The base case
- The inductive hypothesis

In computer science recursion, we have the same ideas, but switched around a bit: instead of starting from a number, oftentimes we go backwards, from some number down to 0.

- The driver (the n-th case)
- The recursive method (the inductive hypothesis, in reverse)
- The terminating condition (the base case)

Think of the call stack.

Another way to think of recursion (which will come in handy later on) is that we have some condition we want to reach, and in order to reach that condition we will “break off” a tiny chunk of the problem each time. The important thing to understand is that each time we break the problem down, each “half” of the problem is nothing more than a smaller version of the original problem.

3. Examples of Recursion

- Counting

Down

```
// This is the driver
public static void countDownToN(int n)
{
    recursiveCountDownToN(n);
}

// This is the recursive method
public static void recursiveCountDownToN(int i)
{
    // this is the base case
    if (i == 0)
    {
        System.out.println("We're at zero, so we're done.");
    }
    else
    {
        System.out.println("currentValue = " + i);
        recursiveCountDownToN(i - 1);
    }
}
```

Up

```
// This is the driver
public static void countUpToN(int n)
{
    recursiveCountUpToN(n);
}

// This is the recursive method
public static void recursiveCountUpToN(int i)
{
    // this is the base case
    if (i == 0)
    {
        System.out.println(
            "We're at zero, so now we start printing."
        );
        return;
    }
    else
    {
        recursiveCountUpToN(i - 1);
        System.out.println("currentValue = " + i);
    }
}
```

- Sum of numbers

```
public static int sumToN(int n)
{
    System.out.println(
        "The sum from 0 to " + n + " is " + recursiveSumToN(n)
    );
}

public static int recursiveSumToN(int n)
{
    if (n == 0)
    {
        return 0;
    }
    else
    {
        return n + recursiveSumToN(n - 1);
    }
}
```

- Factorial of numbers

```
public static int factorial(int n)
{
    System.out.println(
        "The factorial of " + n + " is " + resursiveFactorial(n)
    );
}

public static int resursiveFactorial(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * recursiveFactorial(n - 1);
    }
}
```

3. Assignments

Recursion

- Counting
 - Description
 - See if you can modify recursiveCountUpToN so that it prints out “We counted up to n!” after the last number. It's probably easier if you start from scratch, and recurse “up” to the “base case” instead of “down” to zero as it is currently written.

- Implement a math function
 - Description
 - Consider the following mathematical function:

$$f(x) = f(x - 1) + f(x - 2)$$

This says that the value of the function at x is equal to the value of the function at $x - 1$ added to the value of the function at $x - 2$. By definition of the function, $f(0) = 1$ and $f(1) = 1$. For example, $f(2) = f(1) + f(0) = 1 + 1 = 2$, and $f(3) = f(2) + f(1) = 2 + 1 = 3$, and $f(4) = f(3) + f(2) = 3 + 2 = 5$.

- Task
 - Write a program which implements this function. Allow the user to specify a value for x , and then print the value of the function at x .
 - When you are testing your program, don't pick values that are too large; (depending on how you implemented it) it might take forever to execute! Stick to less than 20.
- Note
 - This function calculates a well-known arithmetic series called the Fibonacci sequence. The Fibonacci sequence is interesting because there are different ways to implement it, some very slow, and some very fast. It is the canonical example of recursion. It is also a popular interview question for programming positions.