CSE 143

Lecture 10: recursive programming

reading: 12.2 - 12.3

Hey baby, this pickup line is recursive. The "hey" is short for "hey baby, this pickup line is recursive. The ""hey"" is short for "hey baby, this pickup line is recursive. The..."

Benoit Mandelbrot: Master of seduction.
TO UNDERSTAND RECURSION, YOU MUST FIRST:

UNDERSTAND RECURSION.
Exercise

• Write a recursive method `pow` accepts an integer base and exponent and returns the base raised to that exponent.
  • Example: `pow(3, 4)` returns 81

• Solve the problem recursively and without using loops.
An optimization

- Notice the following mathematical property:
  \[3^{12} = 531441 = 9^6 = (3^2)^6 = (9^2)^3 = ((3^2)^2)^3\]

- When does this "trick" work?
- How can we incorporate this optimization into our pow method?
- What is the benefit of this trick if the method already works?
There are only 10 types of people in the world: Those who understand binary and those who don't.
Exercise

- Write a recursive method `printBinary` that accepts an integer and prints that number's representation in binary (base 2).
  
  - Example: `printBinary(7)` prints 111
  - Example: `printBinary(12)` prints 1100
  - Example: `printBinary(42)` prints 101010

- Write the method recursively and without using any loops.

```
place  10  1
value  4   2
```

```
32 16  8  4  2  1  
1   0  1  0  1  0
```
Case analysis

- Recursion is about solving a small piece of a large problem.
  - What is 69743 in binary?
    - Do we know *anything* about its representation in binary?
  - Case analysis:
    - What is/are easy numbers to print in binary?
    - Can we express a larger number in terms of a smaller number(s)?
printBinary solution

// Prints the given integer's binary representation.
// Precondition: n >= 0
public static void printBinary(int n) {
    if (n < 2) {
        // base case; same as base 10
        System.out.println(n);
    } else {
        // recursive case; break number apart
        printBinary(n / 2);
        printBinary(n % 2);
    }
}

• Can we eliminate the precondition and deal with negatives?
Exercise

• Write a method `crawl` accepts a `File` parameter and prints information about that file.
  • If the `File` object represents a normal file, just print its name.
  • If the `File` object represents a directory, print its name and information about every file/directory inside it, indented.

```
cse143
  handouts
    syllabus.doc
    lecture_schedule.xls
  homework
    1-tiles
      TileMain.java
      TileManager.java
    index.html
  style.css
```

• **recursive data**: A directory can contain other directories.
File objects

- A File object (from the java.io package) represents a file or directory on the disk.

<table>
<thead>
<tr>
<th>Constructor/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File(String)</td>
<td>creates File object representing file with given name</td>
</tr>
<tr>
<td>canRead()</td>
<td>returns whether file is able to be read</td>
</tr>
<tr>
<td>delete()</td>
<td>removes file from disk</td>
</tr>
<tr>
<td>exists()</td>
<td>whether this file exists on disk</td>
</tr>
<tr>
<td>getName()</td>
<td>returns file's name</td>
</tr>
<tr>
<td>isDirectory()</td>
<td>returns whether this object represents a directory</td>
</tr>
<tr>
<td>length()</td>
<td>returns number of bytes in file</td>
</tr>
<tr>
<td>listFiles()</td>
<td>returns a File[] representing files in this directory</td>
</tr>
<tr>
<td>renameTo(File)</td>
<td>changes name of file</td>
</tr>
</tbody>
</table>
Public/private pairs

• We cannot vary the indentation without an extra parameter:

```java
public static void crawl(File f, String indent) {
```

• Often the parameters we need for our recursion do not match those the client will want to pass.

In these cases, we instead write a pair of methods:
1) a **public**, non-recursive one with parameters the client wants
2) a **private**, recursive one with the parameters we really need
Exercise solution 2

// Prints information about this file, // and (if it is a directory) any files inside it.
public static void crawl(File f) {
    crawl(f, ""); // call private recursive helper
}

// Recursive helper to implement crawl/indent behavior.
private static void crawl(File f, String indent) {
    System.out.println(indent + f.getName());
    if (f.isDirectory()) {
        // recursive case; print contained files/dirs
        for (File subFile : f.listFiles()) {
            crawl(subFile, indent + "    ");
        }
    }
}
Exercise

• Write a recursive method `isPalindrome` accepts a String and returns `true` if it reads the same forwards as backwards.

  • `isPalindrome("madam")` → `true`
  • `isPalindrome("racecar")` → `true`
  • `isPalindrome("step on no pets")` → `true`
  • `isPalindrome("able was I ere I saw elba")` → `true`
  • `isPalindrome("Java")` → `false`
  • `isPalindrome("rotater")` → `false`
  • `isPalindrome("byebye")` → `false`
  • `isPalindrome("notion")` → `false`
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;  // base case
    } else {
        char first = s.charAt(0);
        char last  = s.charAt(s.length() - 1);
        if (first != last) {
            return false;
        }  // recursive case
        String middle = s.substring(1, s.length() - 1);
        return isPalindrome(middle);
    }
}
// Returns true if the given string reads the same
// forwards as backwards.
// Trivially true for empty or 1-letter strings.
public static boolean isPalindrome(String s) {
    if (s.length() < 2) {
        return true;    // base case
    } else {
        return s.charAt(0) == s.charAt(s.length() - 1) && isPalindrome(s.substring(1, s.length() - 1));
    }
}