Building Java Programs

Chapter 12
recursive programming

reading: 12.2 - 12.4
public static int method(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return method(n - 2) + method(n - 1);
    }
}

What is the value returned by \text{method}(5)\,?
Recursion and cases

- Every recursive algorithm involves at least 2 cases:
  - base case: simple problem that can be solved directly.
  - recursive case: 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.
Tracing our algorithm

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

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

```java
class PoemTracer {
    public static void main(String[] args) {
        reverseLines(new Scanner("poem.txt"));
    }

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

- I have eaten the plums that were in the icebox
- the icebox that were in the plums I have eaten
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:
  \[3^{12} = 531441 = 9^6 = (3^2)^6\]
  \[531441 = (9^2)^3 = (((3^2)^2)^2)^3\]

- 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?
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

<table>
<thead>
<tr>
<th>place</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Write the method recursively and without using any loops.
Repeat Digits

- How did we break the number apart?

```java
public static int repeatDigits(int n) {
    if (n < 10) {
        return (10 * n) + n;
    } else {
        int a = repeatDigits(n / 10);
        int b = repeatDigits(n % 10);
        return (100 * a) + b;
    }
}
```
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)?
```java
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 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`
// 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 {
    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);
  }
}
Exercise solution 2

// 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));
    }
}
Exercise

• Write a method `print` 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.
Recursive Data

- A file is one of
  - A simple file
  - A directory containing files

- Directories can be nested to an arbitrary depth
## 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><code>File(String)</code></td>
<td>creates <code>File</code> object representing file with given name</td>
</tr>
<tr>
<td><code>canRead()</code></td>
<td>returns whether file is able to be read</td>
</tr>
<tr>
<td><code>delete()</code></td>
<td>removes file from disk</td>
</tr>
<tr>
<td><code>exists()</code></td>
<td>whether this file exists on disk</td>
</tr>
<tr>
<td><code>getName()</code></td>
<td>returns file's name</td>
</tr>
<tr>
<td><code>isDirectory()</code></td>
<td>returns whether this object represents a directory</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>returns number of bytes in file</td>
</tr>
<tr>
<td><code>listFiles()</code></td>
<td>returns a <code>File[]</code> representing files in this directory</td>
</tr>
<tr>
<td><code>renameTo(File)</code></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
// 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
        File[] subFiles = f.listFiles();
        for (int i = 0; i < subFiles.length; i++) {
            crawl(subFiles[i], indent + "    ");
        }
    }
}
Recursion Challenges

- Forgetting a base case
  - Infinite recursion resulting in `StackOverflowError`

- Working away from the base case
  - The recursive case must make progress towards the base case
  - Infinite recursion resulting in `StackOverflowError`

- Running out of memory
  - Even when making progress to the base case, some inputs may require too many recursive calls: `StackOverflowError`

- Recomputing the same subproblem over and over again
  - Refining the algorithm could save significant time