Building Java Programs

Chapter 12
recursive programming

reading: 12.2 - 12.4

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.

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

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.

`File` objects

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

Constructor/method	Description
`File(`**String**`)`	creates `File` object representing file with given name
`canRead()`	returns whether file is able to be read
`delete()`	removes file from disk
`exists()`	whether this file exists on disk
`getName()`	returns file's name
`isDirectory()`	returns whether this object represents a directory
`length()`	returns number of bytes in file
`listFiles()`	returns a `File[]` representing files in this directory
`renameTo(`**File**`)`	changes name of file

Public/private pairs

- We cannot vary the indentation without an extra parameter:

```
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 print(File f) {
    print(f, "");   // call private recursive helper
}

// Recursive helper to implement crawl/indent
  behavior.
private static void print(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++) {
            print(subFiles[i], indent + "    ");
        }
    }
}
```

Recursive Data

- A file is one of
  - A simple file
  - A directory containing files
- Directories can be nested to an arbitrary depth
- Iterative code to crawl a directory structure requires data structures
  - In recursive solution, we use the call stack