Lecture 9: Recursive Programming

07/13/22
Announcements

• A1 resubmission due Wednesday, July 13\textsuperscript{th} @ 11:59pm
  • Tonight!
• A3 due Thursday, July 14\textsuperscript{th} @ 11:59pm
Recursion and cases

Every recursive algorithm involves at least 2 cases:

• **base case**: the simplest case
• **recursive case**: does a tiny bit of work, then breaks down the problem into a smaller version of itself

Some recursive algorithms have more than one base or recursive case, but all have at least one of each.
Roadmap for the week

- Monday
  - Introduce idea of recursion
  - Goal: Understand idea of recursion and read recursive code

- Tuesday
  - Practice reading recursive code

- Wednesday
  - More complex recursive examples
  - Goal: Identify recursive structure in problem and write recursive code

- Thursday
  - Practice writing recursive code
// post: returns an integer where every digit of n is replaced by two of that digit.
// Example: doubleUp(348) returns 334488
// Example: doubleUp(-348) returns -334488

public static int doubleUp(int n) {
    if (n < 0) {
        return -doubleUp(-n);
    } else if (n < 10) {
        return n * 11;
    } else {
        return 100 * doubleUp(n / 10) + doubleUp(n % 10);
    }
}
Below is a trace of the call `doubleUp(-348):

doubleUp(-348)
 is < 0, so execute first branch
compute doubleUp(-n), which is doubleUp(348)
  |   not < 0, not < 10, so execute third branch
  |   compute doubleUp(34)
  |     |   not < 0, not < 10, so execute third branch
  |     |   compute doubleUp(3)
  |     |     |   not < 0, but is < 10, so execute second branch
  |     |     |   return n * 11 (33)
  |     |     compute doubleUp(4)
  |     |     |   not < 0, but is < 10, so execute second branch
  |     |     |   return n * 11 (44)
  |     |     return first * 100 + second (33 * 100 + 44 = 3344)
  |     compute doubleUp(8)
  |     |   not < 0, but is < 10, so execute second branch
  |     |   return n * 11 (88)
  |     return first * 100 + second (3344 * 100 + 88 = 334488)
return the negation of that result (-334488)
// post: returns a string where every character of str
// is replaced by two of that character
// Example: doubleUp("cat") returns "ccaatt"
// Example: doubleUp(""") returns ""

public static String doubleUp(String str) {
    if (str.length() <= 1) {
        return str + str;
    } else {
        char c = str.charAt(0);
        return "" + c + c + doubleUp(str.substring(1));
    }
}
Recursive Data - File

- A file is one of:
  - A simple file (image, text file, etc.)
  - A directory containing files

- Directories can be nested to an arbitrary depth
**print method**

- 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.

animals
  - cat_jump.png

dogs
  - cool_dogs.txt
  - happy_dog.jpeg

pandas
  - red_pandas
    - cat
      - cat_and_panda.jpeg
      - iloveredpandas.jpeg
      - rp_cubs.png
      - waving_panda.png
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>