CSE 143
Lecture 14

More Recursive Programming; Grammars

reading: 12.2 - 12.3, 12.5; 13.3

slides created by Marty Stepp
http://www.cs.washington.edu/143/

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-sortedintlist
            ArrayIntList.java
            SortedIntList.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 the 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 + "    ");
        }
    }
}
```

Recursive binary search (13.3)

- Write a method `binarySearch` that accepts a sorted array of integers and a target integer value and returns the index of an occurrence of that target value in the array.
  - If the target value is not found, return a negative number.
  - Write the method recursively.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

```
int index  = binarySearch(data, 42);  // 10
int index2 = binarySearch(data, 66);  // -1 or -14
```

Exercise solution

```
// Returns the index of an occurrence of the given value in
// the given array, or a negative number if not found.
// Precondition: a is sorted
public static int binarySearch(int[] a, int target) {
    return binarySearch(a, target, 0, a.length - 1);
}

// Recursive helper to implement search behavior.
private static int binarySearch(int[] a, int target,
                                int min, int max) {
    if (min > max) {
        return -1;   // not found
    } else {
        int mid = (min + max) / 2;
        if (a[mid] == target) {
            return mid;   // found it!
        } else if (a[mid] < target) {
            // middle element too small; search right half
            return binarySearch(a, target, mid + 1, max);
        } else {  // a[mid] < target
            // middle element too large; search left half
            return binarySearch(a, target, mid + 1, max);
        }
    }
}
```

Languages and grammars

- (formal) **language**: A set of words or symbols.

- **grammar**: A description of a language that describes which sequences of symbols are allowed in that language.
  - describes language *syntax* (rules) but not *semantics* (meaning)
  - can be used to generate strings from a language, or to determine whether a given string belongs to a given language

Backus-Naur (BNF)

- **Backus-Naur Form (BNF)**: A syntax for describing language grammars in terms of transformation *rules*, of the form:

  <**symbol**> : := <**expression**> | <**expression**> *...* | <**expression**>

  - **terminal**: A fundamental symbol of the language.
  - **non-terminal**: A high-level symbol describing language syntax, which can be transformed into other non-terminal or terminal symbol(s) based on the rules of the grammar.

  - developed by two Turing-award-winning computer scientists in 1960 to describe their new ALGOL programming language

An example BNF grammar

```
<s> ::= <n> <v>
<n> ::= Marty | Victoria | Stuart | Jessica
<v> ::= cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
Marty slept
Jessica belched
Stuart cried
```

BNF grammar version 2

```
<s> ::= <np> <v>
<np> ::= <pn> | <dp> <n>
<pn> ::= Marty | Victoria | Stuart | Jessica
<dp> ::= a | the
<n> ::= ball | hamster | carrot | computer
<v> ::= cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
the carrot cried
Jessica belched
a computer slept
```

BNF grammar version 3

```
<s> ::= <np> <v>
<np> ::= <pn> | <dp> <adj> <n>
<pn> ::= Marty | Victoria | Stuart | Jessica
<dp> ::= a | the
<adj> ::= silly | invisible | loud | romantic
<n> ::= ball | hamster | carrot | computer
<v> ::= cried | slept | belched
```

- Some sentences that could be generated from this grammar:

```
the invisible carrot cried
Jessica belched
a computer slept
a romantic ball belched
```

Grammars and recursion

```
<s> ::= <np> <v>
<np> ::= <pn> | <dp> <adjp> <n>
<pn> ::= Marty | Victoria | Stuart | Jessica
<dp> ::= a | the
<adjp> ::= <adj> <adjp> | <adj>
<adj> ::= silly | invisible | loud | romantic
<n> ::= ball | hamster | carrot | computer
<v> ::= cried | slept | belched
```

- Grammar rules can be defined *recursively*, so that the expansion of a symbol can contain that same symbol.
  - There must also be expressions that expand the symbol into something non-recursive, so that the recursion eventually ends.

Grammar, final version

```
<s>::=<np> <vp>
<np>::=<dp> <adjp> <n>|<pn>
<dp>::=the|a
<adjp>::=<adj>|<adj> <adjp>
<adj>::=big|fat|green|wonderful|faulty|subliminal
<n>::=dog|cat|man|university|father|mother|child
<pn>::=John|Jane|Sally|Spot|Fred|Elmo
<vp>::=<tv> <np>|<iv>
<tv>::=hit|honored|kissed|helped
<iv>::=died|collapsed|laughed|wept
```

- Could this grammar generate the following sentences?

  `Fred honored the green wonderful child`

  `big Jane wept the fat man fat`

- Generate a random sentence using this grammar.

Sentence generation