CSE 143
Lecture 10

Recursion

slides created by Alyssa Harding
http://www.cs.washington.edu/143/

Example: `writeBinary`

- Before we look at binary, a visual review of decimal numbers:

- We get 348 by adding powers of 10

  348 =

  300 + 40 + 8 =

  $3 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$

- That's why the decimal system is **base 10**

Example: `writeBinary`

- Binary is exactly the same, but **base 2**

Decimal	Binary	Sum
0	0	0×2^0
1	1	1×2^0
2	10	$1 \times 2^1 + 0 \times 2^0$
3	11	$1 \times 2^1 + 1 \times 2^0$
4	100	$1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
5	101	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
6	110	$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
7	111	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
8	1000	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

Example: `writeBinary`

- So, what is 8574 in binary?
- ….anyone?
- How about the last digit?
- It'll be 0, just like all the other even numbers in the table

Example: `writeBinary`

- This gives us an idea of how to break our problem down

- Recall that we can split a decimal number
  to get the leading digits in base-10 and the final digit:

857	4
n/10	n%10

- To get the leading digits in base-2 and the final digit,
  we can do the same:

4287	0
n/2	n%2

Example: `writeBinary`

- So, what's the base case?

```
public static void writeBinary(int n) {
    if ( n < 2 ) {
        System.out.print(n);
    } else {
        ...
    }
}
```

We know what to do with
a single binary digit, 0 or 1

Example: `writeBinary`

- What's the recursive case?

```
public static void writeBinary(int n) {
    if ( n < 2 ) {
        System.out.print(n);
    } else {
        ...
        System.out.print(n%2);
    }
}
```

If we have a bigger number, we can still print out its last binary digit

Example: `writeBinary`

- What's the recursive case?

```
public static void writeBinary(int n) {
    if ( n < 2 ) {
        System.out.print(n);
    } else {
        writeBinary(n/2);
        System.out.print(n%2);
    }
}
```

And we can recurse to print out the remaining part of the number

Example: `writeBinary`

- What's the recursive case?

```
public static void writeBinary(int n) {
    if ( n < 2 ) {
        System.out.print(n);
    } else {
        writeBinary(n/2);
        System.out.print(n%2);
    }
}
```

And we can recurse to print out the remaining part of the number

Example: `pow`

- Now we want to recursively take **`x`** to the **`y`** power
- We'll only deal with **`int`**s, so **`x`** and **`y`** must be **`int`**s
- This also means that **`y`** must be positive so that the return value is an **`int`**

Example: pow

- What's the base case?

```
// returns x to the y power
// pre: y >= 0
public static int pow(int x, int y) {
   if ( y == 0 ) {
      return 1;
   } else {
      ...
   }
}
```

If y is 0, we automatically know that the power is 1!

Example: pow

- What's the recursive case?

```
// returns x to the y power
// pre: y >= 0
public static int pow(int x, int y) {
  if ( y == 0 ) {
    return 1;
  } else {
    return x * pow(x,y-1);
  }
}
```

Otherwise, we can break it down into two smaller problems:

$x^y = x^1 \times x^{y-1}$

Example: pow

- Taking care of negative inputs...

```
// returns x to the y power
// if y < 0, throws IllegalArgumentException
public static int pow(int x, int y) {
  if ( y < 0 ) {
    throw new IllegalArgumentException();
  } else if ( y == 0 ) {
    return 1;
  } else {
    return x * pow(x,y-1);
  }
}
```

Example: `pow`

- But is this the best way?
- If I asked you to compute 2^{32}, you wouldn't hit multiply on your calculator 32 times (hopefully!)
- We can also note that $x^y = (x^2)^{(y/2)}$
  - Example: $3^4 = 9^2 = 81$
- This only works for even values of `y` because of integer division

Example: pow

- Putting in our new idea...

```
public static int pow2(int x, int y) {
  if ( y < 0 ) {
    throw new IllegalArgumentException();
  } else if ( y == 0 ) {
    return 1;
  } else if ( y % 2 == 0 ) {
    return pow(x*x, y/2);
  } else {
    return x * pow2(x,y-1);
  }
}
```

Example: pow

- Why is this better?
- Consider calculating 2 to the billionth power (10^9)
- Our original algorithm would be like hitting "* 2" on your calculator a billion times
  - This is an O(n) algorithm
- The second algorithm decreases y by half each time, a much bigger gain. This is like binary search, when we cut our search space in half each time. This would take about 30 steps
  - This is an O(log(n)) algorithm

Example: `crawler`

- Now we want to make a crawler method that will print out files and directories

- Java provides us with a **`File`** object
  - http://java.sun.com/javase/6/docs/api/java/io/File.html

- Given the file or directory we want, we can print its name

```
public static void print(File f) {
    System.out.println(f.getName());
}
```

We also want to print the contents of a directory, though

Example: `crawler`

- We can get an array of the files in the directory and print their names:

```
public static void print(File f) {
    System.out.println(f.getName());
    for (File subF : f.listFiles()) {
      System.out.println("  "+subF.getName());
    }
}
```

But this only works if `f` is a directory, otherwise it throws a NullPointerException

Example: `crawler`

- So we check to make sure **`f`** is a directory first:

```
public static void print(File f) {
   System.out.println(f.getName());
   if ( f.isDirectory() ) {
      for (File subF : f.listFiles()) {
        System.out.println(" “+subF.getName());
      }
   }
}
```

But this still only works for one level of directories

Example: `crawler`

- So we check to make sure **`f`** is a directory first:

```
public static void print(File f) {
  System.out.println(f.getName());
    if ( f.isDirectory() ) {
      for (File subF : f.listFiles()) {
        System.out.println(" "+subF.getName());
         if ( subF.isDirectory() ) {
           for (File subSubF : subF.listFiles() ) {
                 ...
           }
         }
      }
   }
}
```

Redundant! And it still only works for one more level of directories

Example: `crawler`

- If only we had a method
  that would print the files in a directory…but we do!

```
public static void print(File f) {
   System.out.println(f.getName());
   if ( f.isDirectory() ) {
     for (File subF : f.listFiles()) {
       print(subF);
     }
   }
}
```

We probably want to add indentation, as well

Example: `crawler`

- So we add a **`String`** parameter to show how much each file name should be indented

```
public static void print(File f, String indent) {
   System.out.println(indent + f.getName());
   if ( f.isDirectory() ) {
     for (File subF : f.listFiles()) {
       print(subF, indent + "   ");
     }
   }
}
```

We don't want our client to need to pass a **`String`**, though

Example: `crawler`

- Instead, we make a public method for the client and hide the extra parameter by making our other method private:

```
public static void print(File f) {
    print(f, "");
  }
private static void print(File f, String indent) {
  System.out.println(indent + f.getName());
  if ( f.isDirectory() ) {
    for (File subF : f.listFiles()) {
      print(subF, indent + "   ");
    }
  }
}
```

Using a public/private pair is common in recursion when we want to work with extra parameters

Example: `Sierpinski`

- Chapter 12 in the book discusses the Sierpinski triangle, a fractal (recursive image):