### Events this week

#### Interview Skills Practice Session
- **Date:** Mon., March 15
- **Time:** 12 – 2 pm
- **Location:** Rm 202, ICICS/CS

#### Transport Canada Info Session
- **Date:** Tues., March 16
- **Time:** 4 – 6 pm
- **Location:** HENN 201

#### Financial Literacy 101
- **Date:** Wed., March 17
- **Time:** 12 – 1 pm
- **Location:** Angus 426

#### CS Distinguished Lecture Series Featuring Jeff Hawkins
- **Date:** Thurs., March 18
- **Time:** 3:30 – 4:50 pm
- **Location:** DMP 110

#### Townhall Meeting for CS Major/Honours Students
- **Date:** Thurs., March 18
- **Time:** 12:30 – 2 pm
- **Location:** DMP 310
- **Lunch will be served!**

### Events next week

#### ICICS/KPMG Seminar: What Industry Wants
- **Date:** Tues., Mar 23, 3:30 – 5 pm, Rm 2020, 2332 Main Mall (Kaiser Bldg.)

#### UBC Science Co-op 30th Anniversary Celebration!
- **Date:** Wed., Mar 24, 12 – 3 pm, Ladha Science Student Centre

#### Drop-In Resume and Cover Letter Editing
- **Date:** Wed., Mar 24, 12 – 2 pm, Rm 255, ICICS/CS

#### CSSS 2009-2010 Year-End Boat Cruise
- **Date:** Sat., Mar 27, 6:30 – 11 pm, Harbour Cruises Marina, 501 Denman St.

### Administrivia

- Assignment #3 is now looming closer
  - Due Thursday March 18, 10:00pm
- In case you hadn’t noticed
  - Classes end Thursday April 15th
- Final exam
  - Friday Apr 23, 7:00pm
  - BCS section: DMP 101
Recursive Methods

• Reading
  – 2nd Ed: Chapter 18
  – 3rd and 4th Eds: Chapter 13

• Other Resources
  • http://www.iol.ie/~jmchugh/csc302/
  • http://www2.hawaii.edu/~qzhang/ToyProject-TowerOfHanoi.htm

Learning Objectives
trace code that uses recursion to determine what the code does
draw a recursion tree corresponding to a recursive method call
draw a stack trace of code that uses single and multi-branch recursion
write recursive methods
replace a recursive implementation of a method with an iterative solution (may need to use a stack to model the run-time stack)
Recursive Methods

We have seen that a method can make a call to another method (e.g., a method calling a helper method).

Many programming languages, including Java, allow a method to make a call to itself – we call this *recursion*.

A method that makes a call to itself is known as a *recursive method*.

When a method calls itself, it is essentially repeating itself and so recursion is a form of looping.

Note that in some programming languages, recursion is the *only* way to loop through a block of code.

---

Recursive Methods

Some problems are more naturally solved using recursion than a looping construct such as a *for* loop.

Problems whose solution can be defined in terms of solutions to *smaller* sub-problems have natural recursive solutions.

There are also some data structures whose *structure* can be defined recursively (a binary tree or a list, for example). These structures can be processed recursively in a very natural way.

We'll start with some easy examples.
Example
Suppose we want to write a method to add up all the Integers in a List:

```java
static int sum(List<Integer> list) {
    int sumsofar = 0;
    for (int i: list) {
        sumsofar += i;
    }
    return sumsofar;
}
```
**Example**

We can also think of it this way:

```java
static int sum(List<Integer> list) {
    if (list.isEmpty())
        return 0;
    else
        return list.get(0) +
               sum(list.subList(1, list.size()));
}
```
Another Example

Suppose we want to write a method to draw the following down-triangle (because the pointy end is down) of size 4 on the screen:

```
****
***
** *
*  *
```

We can break this problem down as follows:

```
**** *** ** *
 *** **  *
**  ** *  *
```

Hence we define the problem in terms of smaller sub-problems.

Example cont'd

We can therefore write the following recursive definition of a triangle of a certain size:

\[
drawDownTriangle(size) = \begin{cases} 
\text{drawRow(size)} & \text{if size > 1} \\
\text{drawRow(1)} & \text{if size = 1}
\end{cases}
\]
Example cont'd

The following method will draw a row of stars:

```java
private void drawRow( int size ) {
    for( int count = 0; count < size; count++ )
        System.out.print( '*' );
    System.out.println();
}
```

This method will be used as a helper to draw our triangle.

Example cont'd

The following method uses the recursive definition given earlier to draw a down-triangle of a certain size:

```java
public void drawDownTriangle( int size ) {
    if( size <= 1 )
        drawRow( size );
    else {
        drawRow( size );
        drawDownTriangle( size - 1 );
    }
}
```
Example cont'd
- Trace the following method call: `drawDownTriangle(4);`

```plaintext
draw Row(4);
draw DT(3);
  draw Row(3);
    draw DT(2);
      draw Row(2);
      draw Row(1);
  draw Row(1);
```

Example
Suppose we want to write a method to draw the following up-triangle (up because the pointy end is up) of size 4 on the screen:

```
*  
** 
***
****
```

We can break this problem down as follows:

```
  *  
 ***
 ****
```

Hence we define the problem in terms of smaller sub-problems.
Example cont'd

We can therefore write the following *recursive* definition of a triangle of a certain size:

\[
drawUpTriangle(size) = \begin{cases} 
\text{drawUpTriangle}(size - 1) \text{ then drawRow}(size) & \text{if } size > 1 \\
\text{drawRow}(1) & \text{if } size = 1 
\end{cases}
\]

Example cont'd

The following method uses the recursive definition on the previous slide to draw a triangle of a certain size:

```java
public void drawUpTriangle( int size ) {
    if( size == 1 )
        drawRow( size );
    else {
        drawUpTriangle( size - 1 );
        drawRow( size );
    }
}
```
Example cont'd

- Trace the following method call: `drawUpTriangle( 4 );`

Example

Suppose we want to write a method to draw the following ramp of size 3 on the screen:

```
***
**
*
**
***
```

We can define this problem in terms of smaller ones as follows:

```
***
** *
* *
**
***
```
Example cont'd

We can therefore write the following *recursive* definition of a ramp of a certain size:

\[
\text{drawRamp(size)} = \begin{cases} 
\text{drawRow(size)} \ & \text{if } \text{size} > 1 \\ 
\text{drawRamp(size-1)} \ & \text{then drawRow(size)} \ & \text{if } \text{size} = 1 \\
\text{drawRow(size)} 
\end{cases}
\]

The following method uses the recursive definition on the previous slide to draw a triangle of a certain size:

```java
public void drawRamp( int size ) {
    if( size == 1 )
        drawRow( size );
    else {
        drawRow( size );
        drawRamp( size - 1 );
        drawRow( size );
    }
}
```