**Lesson 3 – Recursion I**

***Lesson Objectives:***

* Define recursion.
* Know the parts of a recursive algorithm.
* Trace through the execution of a simple recursive method.
* Write a simple recursive method.

***Readings:***

* Read Chapter 3, Section 1.

***Recursive methods:***

A method that calls itself is said to be recursive. A method **f1** is also recursive if it calls a method **f2**, which under some circumstances calls **f1**, creating a cycle in the sequence of calls.

Problems that lend themselves to a recursive solution have the following characteristics:

* One or more simple cases of the problem have a straightforward, non-recursive solution.
* The other cases can be redefined in terms of problems that are closer to the simple cases.
* By applying the redefinition process every time the recursive method is called, eventually the problem is reduced entirely to simple cases, which are relatively easy to solve.

Recursive algorithms generally consist of an if statement in the form:

*if this is a simple case*

*solve it*

*else*

*redefine the problem using recursion*

// calculate the factorial, n!

**public** **static** **int** fact(**int** x)

{

**if** (x == 0)

**return** 1;

**else**

**return** x \* *fact*(x - 1);

}

Exercise: Write a method that recursively sums numbers from 0 to n.

// Write a method that recursively sums numbers from 0 to n.

**public** **static** **int** sum\_seq(**int** x)

{

**if** (x == 0)

**return** 0;

**else**

**return** x + *sum\_seq*(x - 1);

}

***Multiple recursive calls:***

A recursive method sometimes makes multiple recursive calls: within the same

// calculate the nth Fibonacci number

public static int fib(int n) {

if ((n == 0) || (n == 1))

return 1;

else

return fib(n - 1) + fib(n - 2);

}

***Head recursion vs. tail recursion:***

* We can also organize recursion by head recursion and tail recursion:
  + In head recursion, the recursive call when it happens, comes before other processing in the function

*if this is a simple case*

*solve it*

*else*

*recursive call*

*process data*

* + In tail recursion, it’s the opposite; the processing occurs before the recursive call.

*if this is a simple case*

*solve it*

*else*

*process data*

*recursive call*

* + Choosing between the two styles may seem arbitrary, but the choice can make a significant difference.
* For example:

public static void head(int n) {

if (n == 0)

System.out.println(n);

else {

head(n - 1);

System.out.println(n);

}

}

public static void tail(int n){

if (n == 0)

System.out.println(n);

else {

System.out.println(n);

tail(n - 1);  
 }

}

public static void main(String[] args) {

head(5);

System.out.println();

tail(5);

}

* ***Recursively processing an attribute array:***
  + We can use private helper methods to provide a cleaner signature for client code.
    - We do this when we want to use recursion, but there is a difference between the parameters needed for the recursion and the parameters needed for the client code.
  + Though we can recursively process an array that is passed as an argument, we can also recursively process an array that is a class attribute:

// An array attribute of the class

private int[] intArray;

// Constructor creating the array attribute

public ConstructorName() {

intArray = { 23, 34, 45, 56, 67, 78 };

}

// calculate the sum of the elements of an array using head recursion

private static int sumByHead(int index) {

if (index == intArray.length - 1)

return intArray[index];

else

return intArray[index] + sumByHead(index + 1);

}

public static int sumByHead() {

return sumByHead(0);

}

// calculate the sum of the elements of an array using tail recursion

private static int sumByTail(int index, int s) {

if (index == intArray.length - 1)

return s + intArray[index];

else

return sumByTail(index + 1, s + intArray[index]);

}

public static int sumByTail() {

return sumByTail(0, 0);

}

* + In this example, we are assuming that all locations in the array contain relevant integers. Also, the array to be processed is passed as an argument to the method.
* ***In-class Exercise:***
  + Write a method that uses head recursion to find the maximum value in an array.
  + Write a method that uses tail recursion to find the maximum value in an array.

private int findMaxHR(int index) {

if (index == intArray.length - 1)

return intArray[index];

else {

int maxFromRest = findMaxHR(index + 1);

if (maxFromRest > intArray[index])

return maxFromRest;

else

return intArray[index];

}

}

public static double findMaxHR() {

return findMaxHR(values, 0);

}

private static double findMaxTR(int index, int maxSoFar) {

if (index == intArray.length)

return maxSoFar;

else {

if (intArray[index] > maxSoFar)

return findMaxTR(index + 1, values[index]);

else

return findMaxTR(values, index + 1, maxSoFar);

}

}

public static double findMaxTR(double[] values) {

return findMaxTR(values, 0, 0);

}