

Statement Purpose:

Purpose of this Lab is to familiarize the students with the use and power of recursion in Java. They will learn what types of problems can be solved using recursion. They will also be taught what are the merits and demerits of using recursion. The students will learn how to solve the problems like finding factorial of any number, binary search, and merge sort using recursion.

Activity Outcomes:

After completing the activities given at the end of this lab material, the students will learn how to write simple Java programs using recursion. They will understand how to use recursion for solving various problems like

- Subtraction of all numbers from 1 up to given number
- Getting sum of all elements of an array where each is reformed and summed
- Finding sum of all odd numbers from 1 up to given number
- Displaying a triangle of numbers from 1 to a given number

Theory Review: (5 Minutes)

Recursion:

Recursion is a type of divide-and-conquer technique used to solve difficult programming problems by reducing a problem to simpler, but identical, "sub-problems." The sub-problems are then reduced to "sub-sub-problems." This multi-step reduction process continues until a *trivial case* (called base case) is reached. The solutions to each sub-problem, starting with the trivial case solution(s) at the lowest level and working upward to the final solution to the original problem, are then combined to produce solutions to the next higher level of problems.

When to use recursive algorithms:

Problems that can be solved using a recursive solution must have the following two characteristics:

1. The problem must be able to be stated in terms of simpler yet identical sub-problems.
2. There must be a trivial case (base case) that is easily solved.



Example (from theory):

Raising a number to an arbitrary power.

Solution by dividing problem into its sub-problem:

Suppose we want to find some base 'b' raised to some power 'e' i.e. b^e .

For example, if $b=2$ and $e=5$, then $b^e = 2^5 = 32$. In order to find 2^5 , we can use the following procedure:

$$2^5 = 2 * 2 * 2 * 2 * 2 \quad (2 \text{ is multiplied with itself } 5 \text{ times})$$

It can also be reduced to its sub-problem as

$$2^5 = 2 * 2^4$$

Now 2^4 can be further reduced to its sub-problem as

$$2^4 = 2 * 2^3$$

Now 2^3 can be further reduced to its sub-problem as

$$2^3 = 2 * 2^2$$

Now 2^2 can be further reduced to its sub-problem as

$$2^2 = 2 * 2^1$$

Now 2^1 can be further reduced to its sub-problem as

$$2^1 = 2 * 2^0$$

Now the sub-problem 2^0 cannot be further reduced. So, it becomes our base case i.e. when power becomes equal to 0, we should get 1.

Recursive Solution:

In order to find the recursive definition of this problem, we will give 'b' and 'e' as an input to our recursive method. Then after every step, we shall reduce the power by 1 in order to come closer to the base case. When we reach base case, we should return 1.

Recursive code for the method "Raising a number to an arbitrary power" is as follows:

```
public static int power(int b, int e) {  
    if (e == 0)  
        return 1;  
    else  
        return b*power(b, e-1);  
}
```



Note:

The more you write recursive code, the more you learn about recursion. For this, it is recommended that the students must visit the site: <http://codingbat.com/java/Recursion-1> and try to solve the problems given as an exercise.

Practice Activity with Lab Instructor: (15 Minutes)

Problem Statement (Triangle)

We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. **Compute recursively** (no loops or multiplication) **the total number of blocks in such a triangle** with the given number of rows.

For example:

triangle(0) → 0

triangle(1) → 1

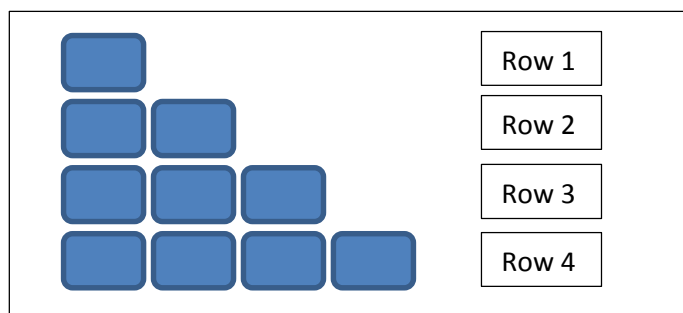
triangle(2) → 3

Defining Recursive Solution:

If we actually draw this triangle for rows = 4, we see that row 1 has one block, row 2 has 2 blocks, row 3 has 3 blocks and row 4 has 4 blocks and we have to add all these blocks.

So, our recursive method will have the following formula:

$$\text{triangle}(\text{rows}) = \text{rows} + \text{triangle}(\text{rows} - 1)$$



Solution:

1. Create a project with the name Recursion2Project.
2. Create a package with the name "Recursion2" within this project.
3. Create a main class in this package with the name "TriangleRecursion".
4. Add the following lines of code within "TriangleRecursion" main class.



```
Start Page  TriangleRecursion.java
Source History
1
2 package Recursion2;
3
4 import java.util.Scanner;
5 public class TriangleRecursion {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         System.out.print("Enter the number of rows: ");
9         int r = sc.nextInt();
10        System.out.println();
11        int SumBlocks = triangle(r);
12        System.out.println("Total blocks in triangle of rows "+r+" are "+SumBlocks);
13    }
14
15    public static int triangle(int rows) {
16        if (rows == 0)
17            return 0;
18        else
19            return rows + triangle(rows-1);
20    }
21 }
```

5. When you run this main class, you will get the following output.

Output 1:

```
Output - Recursion2Project (run)
run:
Enter the number of rows: 4

Total blocks in triangle of rows 4 are 10
BUILD SUCCESSFUL (total time: 6 seconds)
```



Output 2:

```
Output - Recursion2Project (run) %  
run:  
Enter the number of rows: 7  
Total blocks in triangle of rows 7 are 28  
BUILD SUCCESSFUL (total time: 2 seconds)
```

LAB EXERCISES: (60 Minutes)

1. Add a method named countA() in the program Recursion2.java (provided with Lab 5 notes) which counts the number of "A" in the given string.

Sample Run 1:

```
Output - CountAproject (run) %  
run:  
Please enter any string: BANANAS  
Number of A's in the string BANANAS is 3  
BUILD SUCCESSFUL (total time: 1 minute 58 seconds)
```

Sample Run 2:

```
Output - CountAproject (run) %  
run:  
Please enter any string: MANGO  
Number of A's in the string MANGO is 1  
BUILD SUCCESSFUL (total time: 30 seconds)
```

Sample Run 3:

```
Output - CountAproject (run) %  
run:  
Please enter any string: CURRY  
Number of A's in the string CURRY is 0  
BUILD SUCCESSFUL (total time: 34 seconds)
```



2. Add a method named `count7()` in the program `Recursion2.java` (provided with Lab 5 notes) which returns the count of the occurrences of 7 as a digit. For example 717 yields 2.

Note that `mod (%)` by 10 yields the rightmost digit (`126 % 10` is 6), while `divide (/)` by 10 removes the rightmost digit (`126 / 10` is 12).

`count7(717) → 2`

`count7(7) → 1`

`count7(123) → 0`

Sample Run 1:

```
Output - Count7Project (run) %
run:
Please enter any number: 717
Number of 7's in the number 717 is 2
BUILD SUCCESSFUL (total time: 4 seconds)
```

Sample Run 2:

```
Output - Count7Project (run) %
run:
Please enter any number: 1374972387
Number of 7's in the number 1374972387 is 3
BUILD SUCCESSFUL (total time: 13 seconds)
```

Sample Run 3:

```
Output - Count7Project (run) %
run:
Please enter any number: 123456
Number of 7's in the number 123456 is 0
BUILD SUCCESSFUL (total time: 3 seconds)
```



3. Add a method named `ChangeXY()` in the program `Recursion2.java` (provided with Lab 5 notes) which computes recursively (no loops) a new string where all the lowercase 'x' chars have been changed to 'y' chars.

`changeXY("codex") → "codey"`
`changeXY("xxhixx") → "yyhiyy"`
`changeXY("xhixhix") → "yhiyhiy"`

Sample Run 1:

```
Output - ChangeCharacters (run) ✖
run:
Please enter any string in lower case: codex

Old string was codex
New string after changing x's into y's is codey
BUILD SUCCESSFUL (total time: 10 seconds)
```

Sample Run 2:

```
Output - ChangeCharacters (run) ✖
run:
Please enter any string in lower case: xxhixx

Old string was xxhixx
New string after changing x's into y's is yyhiyy
BUILD SUCCESSFUL (total time: 2 seconds)
```

Sample Run 3:

```
Output - ChangeCharacters (run) ✖
run:
Please enter any string in lower case: yhiyhiy

Old string was yhiyhiy
New string after changing x's into y's is yhiyhiy
BUILD SUCCESSFUL (total time: 2 seconds)
```



4. Add a method named `pairStar()` in the program `Recursion2.java` (provided with Lab 5 notes) which computes recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "*".

```
pairStar("hello") → "hel*lo"  
pairStar("xxyy") → "x*xy*y"  
pairStar("aaaa") → "a*a*a*a"
```

Sample Run 1:

```
Output - PairStars (run) %  
run:  
Please enter any string: hello  
  
Old string was hello  
New string after inserting *s in adjacent identical characters is hel*lo  
BUILD SUCCESSFUL (total time: 13 seconds)
```

Sample Run 2:

```
Output - PairStars (run) %  
run:  
Please enter any string: xxyy  
  
Old string was xxyy  
New string after inserting *s in adjacent identical characters is x*xy*y  
BUILD SUCCESSFUL (total time: 57 seconds)
```

Sample Run 3:

```
Output - PairStars (run) %  
run:  
Please enter any string: aaaa  
  
Old string was aaaa  
New string after inserting *s in adjacent identical characters is a*a*a*a  
BUILD SUCCESSFUL (total time: 12 seconds)
```

