Objectives:
This lab will reinforce recursion, methods, and return values.

Preparation:
If you are working on a lab computer, issue the following commands:

```
mkdir ~/cs170/lab6
cd ~/cs170/lab6
gedit Lab6.java &
```

Background:
We will write 2 recursive algorithms in this lab. But first, some theory on how to think about recursive algorithms.

- As discussed in class, a recursive method will invoke itself, but the invocation always uses a parameter value that is smaller than the original value.
- Example: factorial

```
public static int factorial( int n ) {
    if ( n == 0 ) {
        return 1;     // A solution is available for a base case
    } else {
        int sol = factorial ( n - 1 );
        return sol * n;
    }
}
```

The original parameter is \( n \), and \( \text{factorial}(n) \) invokes itself with a smaller parameter value \( (n-1) \).

- The key in understanding recursive methods is recognizing that you are:
  - Not: "solving a smaller problem" to solve the original problem
  - But: "using the solution of a smaller problem" to solve the original problem.
  - Think of the invocation \( \text{factorial}(n-1) \) as something that has been completed and has returned the solution of the smaller problem that you can use.
- Example: compute 20!

  - We know that 20! = 20 \times 19!
  - Suppose we know that 19! = 121645100408832000. Then we can use the solution of this smaller problem (19!) to solve the original problem of 20!:
    \[
    20! = 20 \times 121645100408832000
    = 2432902008176640000
    \]

- The base case(s): In every recursive method, there is at least one or more base cases that must have been solved (i.e., you know the solution to without breaking the problem down further.) These readily solved problems are called base cases.
  - General rule: If you use the solution of the smaller problem \( \text{Problem}(n-k) \) to solve the problem \( \text{Problem}(n) \), you will need \( k \) base cases.
- Structure of a recursive method
  - The easiest recursive method uses one smaller problem to solve the original problem.
The structure of a recursive method to solve a problem of size \( n \) using the solution of a problem of size \( n-1 \) is:

```java
public static ReturnType solveProblem( int n ) {
    ReturnType sol1, mySol;

    if ( n is a base case )    {
        return solution;     // A solution is available for a base case
    }

    else {
        sol1 = solveProblem( n - 1 );  //solution to smaller problem
        mySol = solve problem of size n using solution sol1;
        return mySol;
    }
}
```

**Task 1: A simple recursive method with integer input**

- **Problem description**: Write a recursive method \( \text{power}(\text{int } a, \text{int } n) \) that computes \( a^n \)

- **How to approach the problem**:
  - Q: What is the problem solved by the method?
    - A: \( \text{power}(a, n) \) computes \( a^n \) (example: \( \text{power}(2,10) \) computes \( 2^{10} \))
  - Q: What is a smaller problem of the same nature?
    - A: \( \text{power}(a, n-1) \) computes \( a^{n-1} \). (example: \( \text{power}(2,9) \) computes \( 2^9 \))
  - Q: How can you use use the solution of \( \text{power}(a, n-1) \) to solve the original problem?
    - A: If we are solving \( 2^{10} \) and we know the solution of \( 2^9 \) is 512, we can solve \( 2^{10} \) as: \( 2 \times 512 \).
    - In general: If we are solving \( a^n \) and we know the solution \( a^{n-1} = \text{sol} \), we can solve \( a^n \) as \( a \times \text{sol} \).
  - Q: What is a base case that we have a readily available solution for?
    - A: \( \text{power}(a, 0) \) because anything raised to the 0 power is equal to 1 (so, \( \text{power}(2,0) \) is 1).

- Now use the structured approach above to write the recursive method \( \text{power}(\text{int } a, \text{int } n) \). Do not use any loops in the method. Your solution must be recursive to receive credit.

**Other kinds of smaller problems:**

- The most common smaller problem that a recursive method solves within itself uses the parameter \( n-1 \)
- But, that is not always the case. Problems involving strings that are solved by recursive method usually uses a shorter length string. The most common smaller problem that a "string typed" recursive method solves within itself uses a string of shorter length \( n-1 \).

**Task 2: Another recursion exercise**

- **Problem description**: Given a non-negative number \( n \), write a recursive method \( \text{sumDigits}(\text{int } n) \) that computes sum of the digits in the number \( n \).

- **How to approach the problem**:
  - It is easier to describe the method by listing out the digits of a number, so I will use the notation:
    \( n = xyz \)
to represent the fact that the number $n$ consists of the digits $xyz$.

- Q: What is the problem solved by the method?
  - A: $\text{sumDigits}(n=xyz)$ computes $x + y + z$ (example: $\text{sumDigits}(345)$ returns $3 + 4 + 5$ or 12)

- Q: What is a smaller problem of the same nature?
  - A: $\text{sumDigits } (n/10 = xy)$ computes $x+y$ (example: $\text{sumDigits}(34)$ computes 7). Note that $34 == 345/10$.

- Q: How can you use the solution of $\text{sumDigits}(n/10)$ to solve the original problem?
  - A: If we are solving $\text{sumDigits}(345)$ and we know the solution for $\text{sumDigit}(34) = 7$, we can solve $\text{sumDigits}(345)$ by adding 7 plus the last digit of 345 (5). In other words: $7 + 345\%5$.
    More generally: If we are solving $\text{sumDigits}(n = xyz)$ and we know the solution of $\text{sumDigit}(n/10 = xy = sol1$, we can solve $\text{sumDigit}(n = xyz)$ by:
    $sol1 + z$ or
    $sol1 + (n \% 10)$

- Q: What is a base case that we have a readily available solution for?
  - A: $\text{sumDigits}(0)$ is equal to 0

- Use the above information to write the recursive method $\text{sumDigits}(\text{int } n)$. You must use recursion (not loops!) in order to receive credit.

**Turnin:**

- Comment your code and make sure your style is correct. Make sure you file and class are named `Lab6.java`

- Get credit for your work by either:
  - Submitting your work to the Lab6 assignment on Blackboard. Make sure you submit! Do not just save a draft.
  - Put the following lines in your main method and then demo to your TA:
    ```java
    System.out.println("power(2,10) is "+power(2,10));
    System.out.println("sumDigits(40912) is "+sumDigits(40912));
    ```

- If you do not finish during your lab time, your submission is due to BB by 5pm on Saturday. No late submissions are accepted for labs.