12.03 Virtual Lecture Notes (Part 1)

Examine the ReverseUserInput class shown below and perform a quick mental desk check.

```
import java.util.Scanner;
public class ReverseUserInput
{
    Scanner in = new Scanner(System.in);

    //accept user input and then reprint it in reverse order
    public void reverseInput( )
    {
        System.out.print("Enter a word ('q' to quit): ");
        String aWord = in.next( );
        if(aWord.equals("q"))
           System.out.println( );
        else
            reverseInput( );
        System.out.println(aWord);
    }

    public static void main(String[ ] args)
    {
         ReverseUserInput reverseIt = new ReverseUserInput();
         System.out.print("Enter a list of words,”);
         System.out.print(“ press Enter after each word.  ”);
         System.out.println(“Type \’q\’ to finish.");
         reverseIt.reverseInput( );
    }
}
```

At first glance, the program looks pretty unremarkable. The **`main()`** method does three things:

1. A **`reverseIt`** object of type **`ReverseUserInput`** is created.
2. The user is told what to input and how to quit.
3. The **`reverseInput( )`** method is invoked.

Next examine the **`reverseInput()`** method:

1. Nothing is returned.
2. No parameters are required.
3. User input is requested with the **`next()`** method.
4. Input is assigned to a **`String`** variable.

5. There is an **`if-else`** statement and “q” is the terminating condition.
6. There are two print statements.

The **`reverseInput()`** method seems pretty straightforward except for one unusual feature: it calls itself! In fact, it appears that this method creates a ”loop” structure without using **`for`** or **`while`**. These are typical characteristics of a recursive method.

At this point, if you have not run the ReverseUserInput class, go ahead and do so. It is also a good idea to open the BlueJ Debugger before running any program involving recursion, so you can terminate execution if necessary. A sample run and output is shown below:

```
Enter a word ('q' to quit): Reverse
Enter a word ('q' to quit): this
Enter a word ('q' to quit): input
Enter a word ('q' to quit): q

q
input
this
Reverse
```

In this example, the input consists of three words typed from the keyboard on separate lines, followed by the letter q to quit.

Notice that **no output was generated** until after the letter q was entered.

The output shows that when the user terminates keyboard entry (by typing the letter q), the words that were typed are re-printed in reverse order.

Inquiring minds will have several questions at this point!

1. What is the flow of control through a recursive method?
2. What happens when a method calls itself?
3. How can all input values be retained using only one simple variable (i.e., **`aWord`**)?
4. How can all of the words be printed with only the last **`println()`** method?

Think about these questions carefully as you go through the first eIMACS lesson.