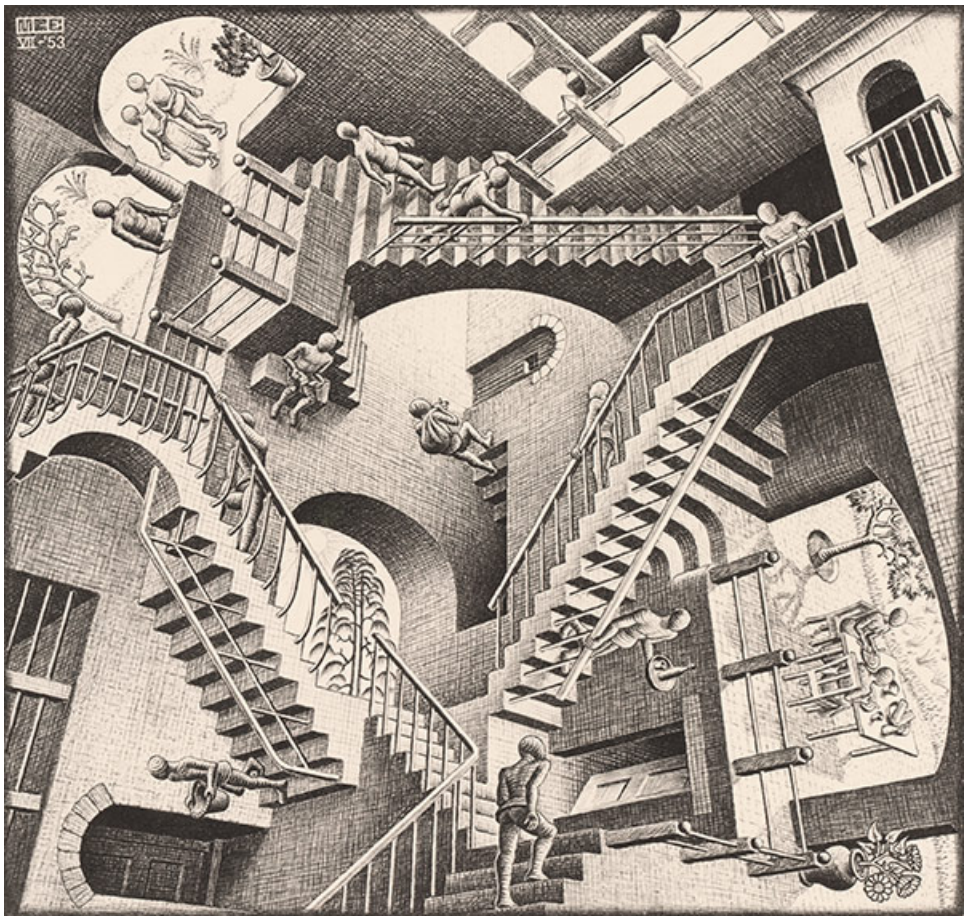
Recursive Programming in Java or the Infinite Mirror

Mary Theis



So what is recursive programming and why would you use recursion in Java?

First, let's look at what the word recursive means. In general, the word recursive means doing or saying the same thing several times in order to produce a particular result or effect. Recursive programming is where you write a function that references part of itself in the function to solve a problem. Here is an example of a simple Java recursive function:

```
void RecursForeverFun(int counter)
{
   RecursForeverFun(--counter);
   System.out.println("> "+counter+" ");
   return;
}
```

Think of it as an infinity mirror. With that in mind, you can get into trouble using recursive functions as they can spiral into an infinite loop and error out if you do not have a conditional check that ends and exits the function, known as the base case or base condition. That's why you would rewrite the above code like this:

```
void RecursFiniFun(int counter)
{
if(counter > 0)
        RecursFiniFun(--counter);
        System.out.println("> "+counter+" ");
}
```

Why use recursive programming?

You could accomplish the above result with an iterative function. However, a recursive function is more succinct and easier to read. Recursive functions are useful for complex sorting, as with the classic Tower of Hanoi problem, or for binary searches or tree traversals. They can also save memory.

Types of recursion

There are many different types of recursions. You would use each one or a combination of them, depending on the problem you are trying to solve. Here are a few:

- Head recursion
- Tail recursion
- Direct recursion
- Indirect recursion
- Multi-recursion

The head recursion is where the task is done as the first step in a function. The sample code above is an example of a head excursion. However, recursive code at the start of a function tends to use more memory and CPU time.

This is where tail recursions come to the rescue. The tail recursion is when the task is completed as it exits the function. This method has the advantage of doing the same task but using less memory.

```
void TailTale(int counter)
{
if(counter == 6)
   return;
else
   {
    System.out.println("> "+counter+" ");
    TailTale(++counter);
   }
}
```

Direct recursion is where the function is directly called from itself as in the example above. Where indirect recursion is more coy about it by having a series of function calls that lead back to the original function (see below).

```
void IndirectFun1(int counter)
{
    IndirectFun2(counter);
    System.out.println("> "+counter+" ");
}
void IndirectFun2 (int counter2)
{
    IndirectFun3(counter2);
}
void IndirectFun3 (int counter3)
{
    if (counter3 > 0)
        IndirectFun1(--counter3);
}
```

Multi-recursion is where a function that calls itself multiple times.

```
void MultiFun(int counter)
{
if(counter > 1)
        newnumber = MultiFun(counter - 1) + MultiFun(counter - 2);
        System.out.println("> "+newnumber+" ");
}
```

How to best write a recursive program

Recursive programs can be elegant but tricky to write. The best way to write a recursive program is to keep these objectives in mind:

- Break the problem into the smallest problem to be solved
- Get the smallest task in the code to work before solving the larger problem
- Have the smaller task as part of the solution to the larger problem
- Determine your exit condition(s)
- Use comments to describe what each task is doing
- Don't double-dip variables (using the same variable for different tasks)

Remember that the debugger or display statements are your friends when coding and testing recursive code. Have fun experimenting with different recursive functions. Soon you will be an expert at using the infinite mirror of code.

Other Resources:

Wikipedia Recursion (Computer Science)

Derek Banas Java Recursions video