11_1 Recursion Basics

Recursion 첫번째 노트 강의하겠습니다.

Recursive `void` Methods

- A recursive method
  - A method that includes a call to itself
  - Problem solving technique of breaking down a task into subtasks
  - Used when a subtask is a smaller version of the original task

페이지 2

먼저 return value가 없는 recursive methods에 대해 알아봅니다.
Recursive method의 정의는 자기 자신에 대한 call을 포함하고 있는 method를 말합니다.
문제를 푸는데 있어
큰 task를 subtask들로 나누어 푸는 방식을 취하는 것인데
여기서 보통 subtask 문제의 크기는
original task보다 그 크기가 작아지게 됩니다.

Example: Vertical Numbers

```
public class VerticalNumbersRecursion {
    public static void main(String[] args) {
        System.out.println("writeVertical(3):");
        writeVertical(3);
        System.out.println("writeVertical(12):");
        writeVertical(12);
        System.out.println("writeVertical(123):");
        writeVertical(123);
    }
    public static void writeVertical(int n) {
        if (n < 10) System.out.println(n);
        else { // n is two or more digits long
            writeVertical(n / 10);
            System.out.println(n % 10);
        }
    }
}
```

```
writeVertical(3):
3
writeVertical(12):
1
2
writeVertical(123):
1
2
3
```

3

페이지 3

Vertical Numbers라는 example을 보도록 하겠습니다.
main에서 writeVertical이라는 method를
3, 12, 123의 서로 다른 parameter들과 함께
세번 call하였습니다.
어떤 task인지를 보기위해
method call과 output만을 보면
writeVertical(3) 이라는 call에 의해
3이 output됩니다.
input이 그대로 output으로 나오는 것은
3이 10보다 작은 한자리 수 이기 때문입니다.
그 다음으로 writeVertical(12)를 call하면
1 2가 차례로 세로로 print됩니다.
그 다음으로 writeVertical(123)을 call하면
1 2 3이 세로로 print됩니다.
writeVertical(int n) 을 recursive method로 구현하였습니다.
먼저 n이 한자리 수 일 경우에는 그대로 print out하게 됩니다.
한자리 보다 큰 경우에는

먼저 parameter를 n을 10으로 나눈 몫으로 만들어
writeVertical(n / 10) 을 recursive call하고
n을 10으로 나눈 나머지 n % 10 을 print 하도록 하였습니다.

Subtasks

- Subtasks
  - Subtask 1: Recursive case
    - Smaller version of the original task
    - Implemented with a recursive call
  - Subtask 2: Stopping case
    - The simple case

```
public static void writeVertical(int n) {
    if (n < 10) System.out.println(n);
    else {
        writeVertical(n / 10);
        System.out.println(n % 10);
    }
}
```

4

페이지 4

어떤 method를 recursion으로 이용하여 구현하려 할 때
두 가지의 subtask가 반드시 포함되어야 합니다.
먼저 첫번째 subtask는 recursive case로
original task의 smaller version을 만들어
recursive call로 구현하게 됩니다.
예를 들어 우리의 writeVertical method에서는
else 파트에서 n을 10으로 나눈 몫을 parameter로 하여
writeVertical을 call하는 것이 이 recursive subtask가 됩니다.
두번째 subtask는 stopping case로
simple case라고도 불리는데
task가 간단하여 더 이상 subtask로 나누지 않더라도
쉽게 풀릴 수 있는 case가 되겠습니다.
우리의 writeVertical method에서
n이 10보다 작은 경우, 즉, 한자리 수인 경우가
stopping case가 되는데
더 이상의 recursive call 없이
그대로 n을 print하면 됩니다.

이와 같이 recursive method를 사용한 solution에는
반드시 이 두가지 type의 subtask들이 존재해야 합니다.

Tracing a Recursive Call

```
writeVertical(123);
   writeVertical(12);          // writeVertical(123/10);
      writeVertical(1);        // writeVertical(12/10); stopping case
      System.out.println(1);   // print(1)
   Systemout.println(2);       // n % 10 = 12 % 10 = 2, print(2)
System.out.println(3);         // n % 10 = 123 % 10 = 3, print(3)
```

5

페이지 5

Recursive call을 좀 더 자세히 trace해 보도록 하겠습니다.
writeVertical(123) 이 call 되었을 때
10으로 나눈 몫인 12로 다시 writeVertical(12)를 call하고
다시 10으로 나눈 몫인 1로 writeVertical(1)을 call합니다.
이 때 stopping case에 걸리게 되고
1을 output한 후 writeVertical(1)을 끝내게 됩니다.
아까 writeVertical(1)을 실행한 바로 뒤에서
12를 10으로 나눈 나머지인 2를 print하게 되고
writeVertical(12)를 끝내고 return하게 됩니다.
그 뒤에 123을 10으로 나눈 나머지 3을 print하게 됩니다.
이렇게 순서대로 차근차근 trace를 하면
recursive method의 실행을 이해하는데 도움이 됩니다.

Infinite Recursion

- An alternative version of **`writeVertical`**
  - Note: No stopping (simple) case!

```
public static void newWriteVertical(int n)
{
  newWriteVertical(n/10);
  System.out.println(n%10);
}
```

```
newWriteVertical(123);
    newWriteVertical(12);  // newWriteVertical(123/10);
        newWriteVertical(1);  // newWriteVertical(12/10);
            newWriteVertical(0);  // newWritevertical(1/10);
                newWriteVertical(0);  // newWritevertical(0/10);
                    ...
```

6

페이지 6

이번에는 writeVertical을 수정하여 보았는데
stopping case를 처리하지 않는 경우로 해 보았습니다.
수정된 method는 newWriteVertical method로서
parameter로 주어진 n을 10으로 나눈 몫으로 recursive call을 하고
n을 10으로 나눈 나머지를 print하게 하였습니다.
따라서 이 newWriteVertical method는
stopping case를 가지지 않습니다.

newWriteVertical(123) 이 실행되는 순서를 보겠습니다.
123을 10으로 나눈 몫인 12를 parameter로 하여
newWriteVertical(12)를 call합니다.
그 안에서 다시 12를 10으로 나눈 몫인 1을 parameter로 하여
newWriteVertical(1)을 call합니다.
stopping case가 없기 때문에
다시 1을 10으로 나눈 몫인 0을 parameter로 하여
newWriteVertical(0) 을 call합니다.
0을 10으로 나눈 몫은 0이므로

그 안에서 또 다시 newWriteVertical(0)를 call하게 되고
결국 이 call을 무한히 계속하게 됩니다.
이렇게 무한한 recursion을 반복하게 되는 이유는
stopping case가 존재하지 않기 때문입니다.

Stacks for Recursion (1/2)

- To keep track of recursion
  - Most computer systems use a stack
  - Stack
    - Specialized kind of data structure
    - Analogous to a stack of paper
    - New paper is placed on top of the stack
    - A paper on the top removed first
    - LIFO: Last In First Out

7

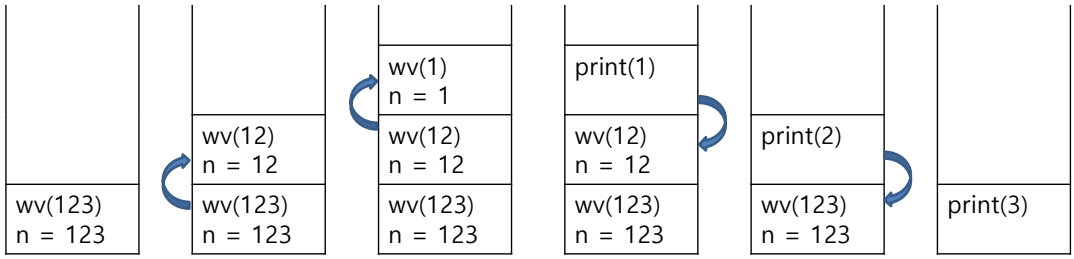
페이지 7

recursion을 구현하기 위해
거의 모든 컴퓨터 시스템은 stack을 사용하고 있습니다.
Stack은 특별한 data 구조로서
종이를 쌓아놓은 stack과 비슷합니다.
새로운 종이는 항상 stack의 맨 위에 놓이게 됩니다.
Stack에서 종이 한 장을 제거하려면 맨 위에 있는 것을 제거해야 합니다.
즉 가장 늦게 들어온 것이 가장 빨리 나가는
Last In First Out (LIFO) 구조 입니다.

Stacks for Recursion (2/2)

```
public static void wv(int n) {
    if (n < 10) {
        print(n);
    }
    else {
        wv(n / 10);
        print(n % 10);
    }
}
```

Stack of the activation records of called methods



8

페이지 8

Activation Record (AR)은 method call 될 때 하나씩 생기는데
method에 전달되는 parameter와 local variable의 값 등
method가 실행되는데 필요한 정보들을 담고 있습니다.
AR은 method에서 return 되는 순간 소멸됩니다.
writeVertical method의 recursive call에서
AR들은 stack 구조로 관리됩니다.
여기서는 writeVertical 을 줄여서 wv로 표현했습니다.
먼저 wv(123)이 call되면 empty stack에 AR이 하나 만들어 집니다.
이 때 AR 안의 parameter n의 값은 123 입니다.
recursive call로 wv(12)가 call 됩니다.
다시 recursive call로 wv(1) 이 call 됩니다.
n이 한자리 수 이므로 print(1) 이 실행되고 wv(1)이 끝나므로
stack top에 있던 AR이 소멸됩니다.
wv(12) 안에서 print(2)를 하고 wv(12)가 끝이납니다.
마지막으로 wv(123)에서 print(3)이 되고 AR이 소멸됩니다.

Recursion Versus Iteration

- Recursion is not absolutely necessary
  - Any task using recursion can also be done in a non-recursive manner
  - A non-recursive version of a method is called an **iterative** version
- A recursive version
  - simpler, but usually run slower than iterative version
  - spend more storage than iterative version

페이지 9

Recursion과 Iteration을 비교해 보겠습니다.
사실 recursion이 항상 필요한 것은 아닙니다.
recursion을 사용하여 해결할 수 있는 task는
non-recursive한 방식으로도 해결할 수 있다는 뜻입니다.
이러한 non-recursive version의 방법은 iteration을 사용하게 됩니다.
Recursive version은 보기에 간단하고 이해가 쉽습니다만
iterative version 보다 실행 속도는 느립니다.
바로 전 slide에서 보았듯이
method가 call될 때마다 activation record들이 하나씩 생겨야 하기 때문에
AR을 생성하고 소멸하는데 시간이 들게 됩니다.
또한 recursive version에서는
AR에 드는 storage 공간이 더 늘어나게 됩니다.
잘못해서 무한 루프에 빠지게 된 recursive version의 경우
시스템의 메모리를 모두 소모하여
에러를 발생시킬 수 있습니다.

Iterative version of `writeVertical`

```
public static void writeVertical(int n) {
    int nsTens = 1;
    int left = n;
    while (left > 9) {
        left = left / 10;
        nsTens = nsTens * 10;
    }

    // nsTens: power of 10 having the same number
    // of digits as n. ex) if n=2345, nsTen=1000.

    for (int pt = nsTens; pt > 0; pt = pt/10) {
        System.out.println(n/pt);
        n = n % pt;
    }
}
```

left	nsTens
2345	1
234	10
23	100
2	1000

pt	n	print
1000	2345	2
100	345	3
10	45	4
1	5	5

10

페이지 10

이 코드는 writeVertical method를 iterative version으로 바꾼 것입니다.
여기서는 writeVertical(2345)가 call되었다고 가정합니다.
먼저 nsTens의 값을 1, left를 2345로 초기화 합니다.
left가 9보다 큰 동안 while 문을 반복하는데
반복할 때마다 left는 10으로 나눈 몫으로 하고
nsTens에는 10을 곱해 나갑니다.
첫번째 loop에서 left는 234, nsTens는 10이 되었고
그 다음에 left는 23, nsTens는 100이 되고
한번 더 반복하면 left는 2, nsTens는 1000이 됩니다.
이제 left가 2로 9보다 작기 때문에 while loop를 빠져나오게 됩니다.
현재 nsTens의 값이 1000인데,
이것은 n과 같은 자리수를 가지는 10의 power 값입니다.
우리의 경우 n이 2345 이므로 nsTens는 1000이 된 것입니다.
이제 for loop에서 pt는 nsTens 값인 1000으로 초기화 하고
반복할 때마다 pt를 10으로 나눈 몫으로 바꿉니다.
for loop는 pt가 0보다 클 때까지만 반복을 수행합니다.
for문의 body에서 n을 pt로 나눈 몫을 print합니다.

첫번째 loop에서는 2345를 1000으로 나눈 몫인 2가 프린트 됩니다.
이제 n을 pt로 나눈 나머지로 바꾸면 345가 됩니다.
두번째 loop에서는 345를 100으로 나눈 몫인 3이 프린트 됩니다.
pt와 n은 각각 10과 45가 되었습니다.
다음 loop에서 45를 10으로 나눈 몫인 4가 프린트 됩니다.
pt와 n은 각각 1과 5가 되었습니다.
다음 loop에서 5를 1로 나눈 몫인 5가 프린트 됩니다.
pt와 n은 각각 0과 5가 됩니다.
pt가 0이므로 for loop를 빠져나오면서 프로그램이 끝나게 됩니다.