11 Recursion

11-1 Recursion Basics

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

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
```

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);
    }
}
```

Tracing a Recursive Call

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

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

Pitfall: 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);
}
```

```
writeVertical(123);
   writeVertical(12);  // writeVertical(123/10);
      writeVertical(1);  // writeVertical(12/10);
         writeVertical(0);  // writevertical(1/10);
            writeVertical(0);  // writevertical(0/10);
               ...
```

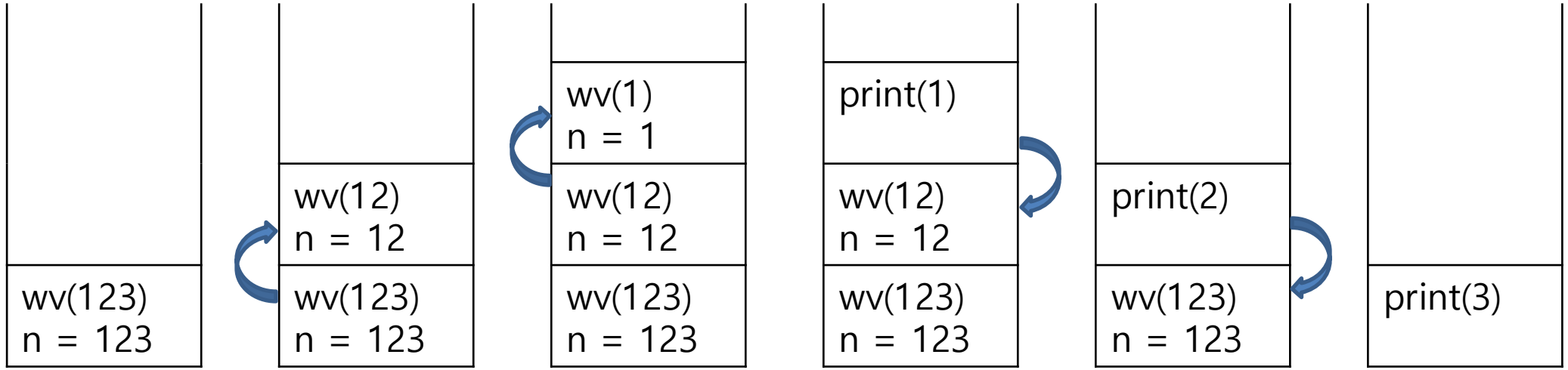
Stacks for Recursion (1/2)

- To keep track of recursion
  - Most computer systems use a stack
  - Stack
    - Specialized kind of memory 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

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



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

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