Lecture 10: Intro to Recursion

- This week we're taking a break from data structures --> a new CONTROL STRUCTURE
  - We've been writing lots of code with LOOPS
    - This technique is called ITERATION
  - Now we're going to look at an alternative technique called RECURSION
    - Easy to explain, but hard to understand
    - You'll have an "AH HA!" moment
  - Equally powerful as loops
    - Why study if we can't get anything extra out of it?
    - Recursion vs. iteration is a fundamental debate in computer science (almost religious - a la Christianity vs Buddhism)
    - You need to be aware of this debate
    - One is not right or wrong - they are attempting to solve the same problem differently
    - Some programming languages ONLY have recursion - no loops
  - Also, some problems are naturally easier to express recursively vs. iteratively
- A non-programming example
  - "What row are you in?"
    - The iterative solution is to have the student count
    - Start at the front of the room, and do "counter++", moving eyes back by one row each time, while there are still rows to count
  - But now imagine instead of people, you are robots and you have very poor vision
    - You can't count all the way to the front of the room
    - You can only see/communicate with the robots next to you
    - What can you ask the robot in front of you that would help?
      - NOT "Do you know what row I'm in?"
      - "What row are YOU in?"
    - Then the robot in front can ask the person in front of it, etc.
    - Until when?
      - The first row - the robot can see that no one is in front of it
    - Each robot then responds with the row # plus 1
  - This is RECURSION
    - Break a problem into smaller pieces that are somehow all the same
    - Defining an operation in terms of itself
- Also, consider walking down a staircase (draw picture)
  - You only have to know how to do two things:
    - (1) how to take a step downwards
    - (2) how to recognize that you're at the bottom
  - Then you can put the steps together
  - We distinguish between these two cases:
    - The BASE CASE: when to stop
    - The RECURSIVE CASE: how to take a step to a smaller problem (taking a step down decreases the total number of steps left)

    - If we didn't have the BASE CASE...
        - You'd "trip and fall" - something bad
    - If we didn't have the RECURSIVE CASE...
        - You'd never get down the stairs at all
    - Some problems may have more than one base case (ways to stop) or more than one recursive case (ways to step down), but all problems will have at least 1 of each
- Another example
    - I have a bowl of M&Ms
    - I want to double the number of M&Ms in the bowl
    - But I can't count them
- Programming example: writeStars(int n)
    - Produces a line of output with *n* stars (for some *n* >= 0)
    - (show the iterative solution)
        - Initialization before the loop (*int i = 0*)
        - A test at each iteration of the loop (*i* < *n*)
        - Some code run at each iteration (*print* and *i*++)
        - Concluding code (*println*)
    - We want to write this recursively, so we think of CASES
        - Recursion always involves case analysis
        - Usually I try to think of the base case first - always ask "What is my base case?"
        - But for you, the recursive case may be easier to do first -
    - Base case? What is the EASIEST line of stars to write?
        - One star?
        - Actually, zero stars is even easier!
            - We just print a blank line
        - This forms our base case:

```
if (n == 0) {
    System.out.println();
} else {
    ...
}
```

    - If-else statements are common in recursion - either we have reached the end (stoping condition - base case), or we have some work left to do
    - Recursive case? (this can be an "ah ha!" solution - so don't blurt it out!)
        - What you'd want to do is use a for-loop to print all the stars...
        - But we can't do that. Instead we have to do a piece of the task and then delegate
        - What small piece of the task can we do? Get a little closer to being done?
            - Print a single star
        - Now what is left to do?
            - Print n - 1 stars
        - How can we do that?

- This is what we call the LEAP OF FAITH
  - "If only we had a method that could write n-1 stars, we could call it"
  - But we do have that method - the one we're writing!

```
public void writeStars2(int n) {
    if (n == 0)
        System.out.println();
    else {
        System.out.print("*");
        writeStars2(n - 1);
    }
}
```

- This seems wrong to us b/c we're used to solving the entire problem ourselves
- How can we call the same method again and it will work?
- Don't think of calling the same method again - it's another robot with the same code
- Also, *n* is smaller - what would happen if we called writeStars2(n) again?
- Explain how this relates to the robots-counting-row-number example

writeStars2(2)
writeStars2(1)
writeStars2(0)

- Another example: print lines from a file in reverse order
  - How would we do this with a loop?
    - Remember, we need to print the last line first! So we need to "remember" the first lines
    - So we'd need a structure - like an ArrayList - to store the lines
  - But with recursion, we don't need an extra structure
  - Write the header - take a Scanner as a parameter
  - What is the easiest file to reverse?
    - A 1-line file? Actually, a 0-line file
  - What do we want to do in this "base case"? How do reverse 0 lines?
    - You do nothing!

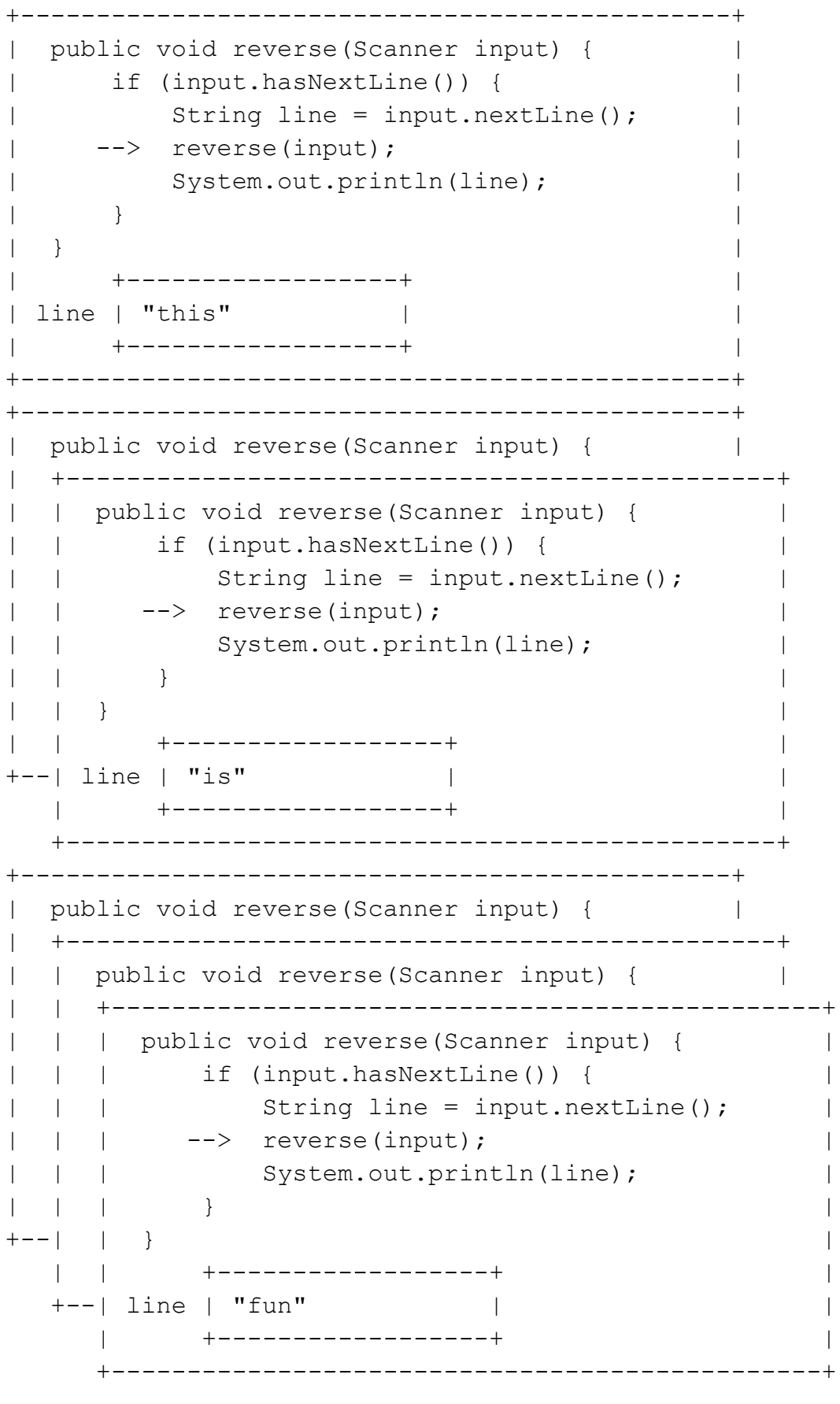
```
if (input.hasNextLine()) {
}
```

    - We could leave a blank if-statement, but it's usually better practice in this case to reverse the condition and have a blank else - and then just leave off the else altogether! (else is "implied")

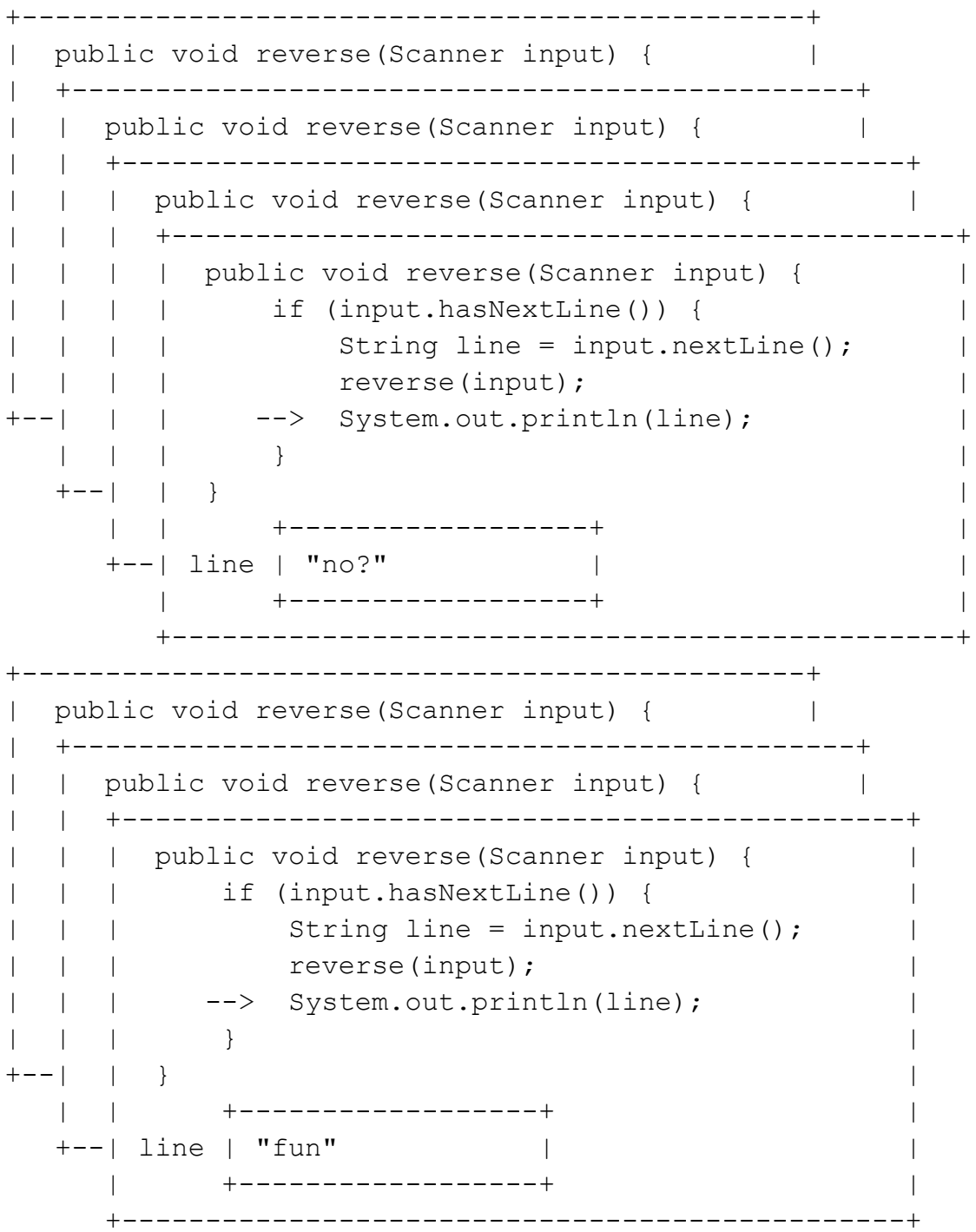
```
if (input.hasNextLine()) {
    ...
}
```

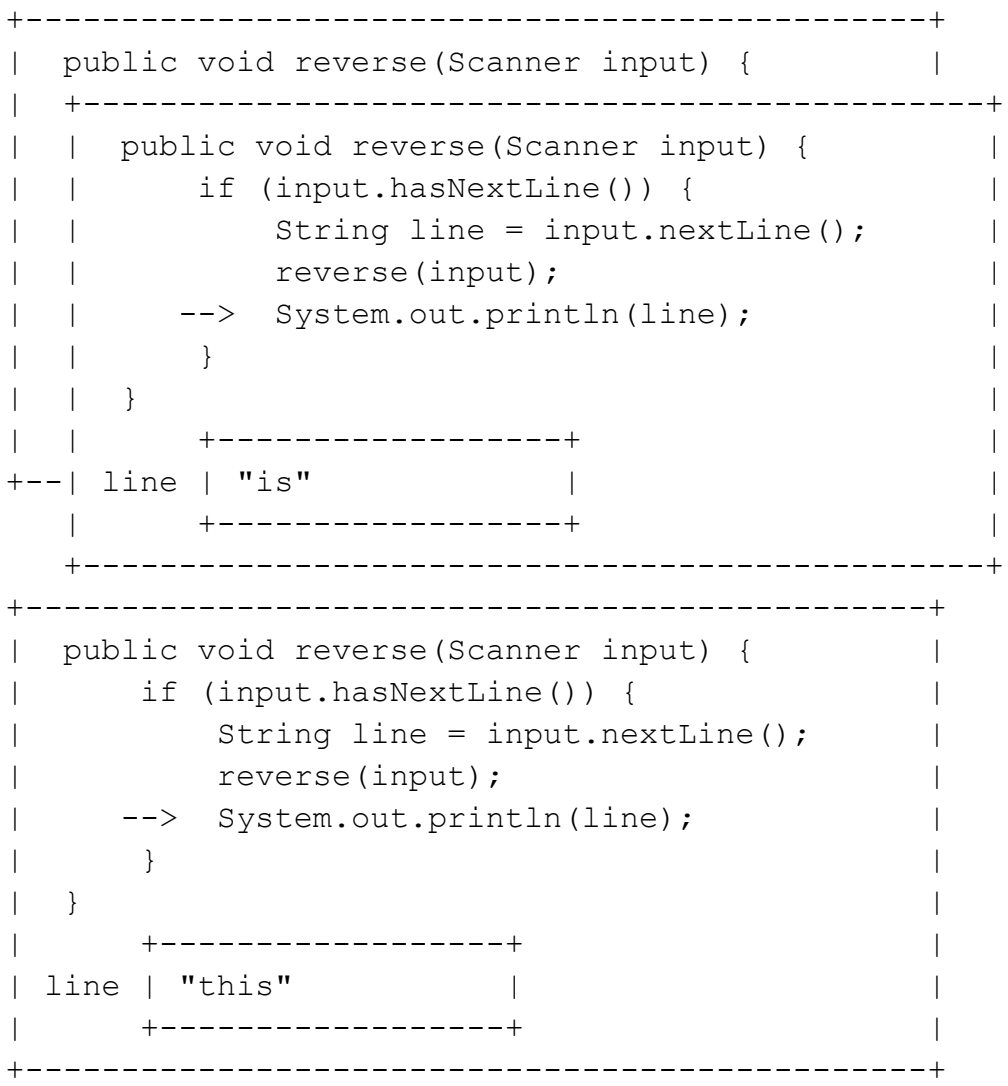
    - OK, now the recursive case
        - We need to do a little bit of work - how do we do that?
            - Read a line of the file
            - “this”
        - What work is left to do after that?
            - “is/fun/no?”
        - What happens if we make a recursive call?
            - LEAP OF FAITH that the recursive call will reverse “is/fun/no?” into “no?/fun/is”
        - What is the result that we want?
            - “no?/fun/is/this”
        - So if we make a recursive call, what work is left to do?
            - Just print the final line
    - So we can complete the code!
    - Wow, can this be it?
- Simulate a call on “reverse” using paper (below)
    - Visual representation of what is going on
    - We call the stack of papers the “call stack”
- Another example: stutter an integer
    - Ex. given 348, turn it into 334488
    - Replace each digit with 2 of that digit
    - Base case: what is the easiest number to stutter?
        - 0 - incorrect! We can’t really return “00” because that’s not a valid integer
        - It turns out that we’ll just return 0 in that case
        - So what’s an alternative easy number to stutter?
        - Any number < 10 (one digit)
        - (note: you can’t have 0 digits!)
    - How do we stutter a digit (for the base case)?
        - well, 1 --> 11, 2 --> 22, 3 --> 33,...
        - n0 + n = nn (10 of n + n), or 11 n
        - so we can do 11*n
    - So how do we deal with larger numbers - how can we split them up?
        - We need to split into smaller chunks
        - Since our base case is 1 digit, we need to process 1 digit
        - We can do this with /10 and %10
        - Then we have 34 and 8. Let’s trust that our method works; we could call it to turn these into 3344 and 88.
        - How do we combine 3344 and 88 into a single number?
        - Can’t simple add --> would give 3432
        - Instead, we want 334 THOUSAND - we need to shift over the 3344!
        - Multiply by 100
    - What about negative numbers?
        - Special case for these

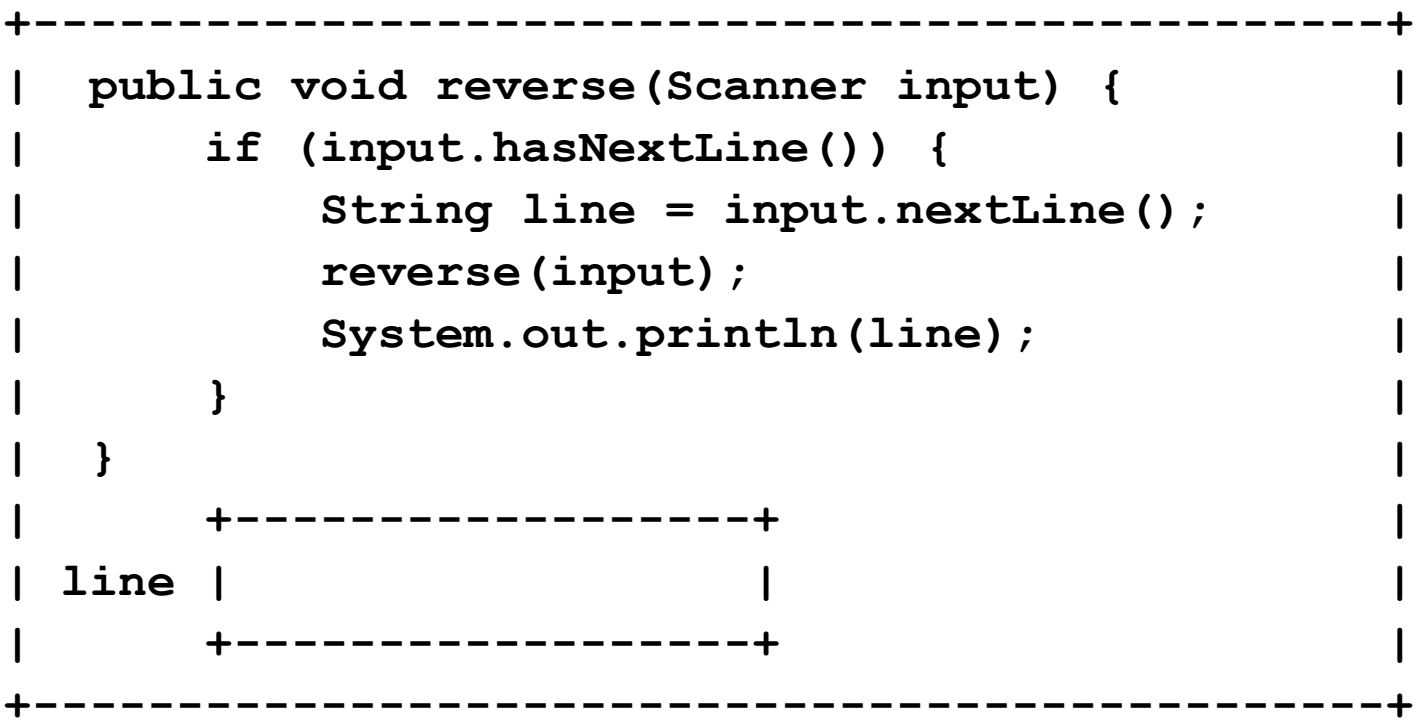
(if time, writeBinary) - same trick but /2 and %2 - can you tell anything about the binary from just the number? You can tell what it ends with

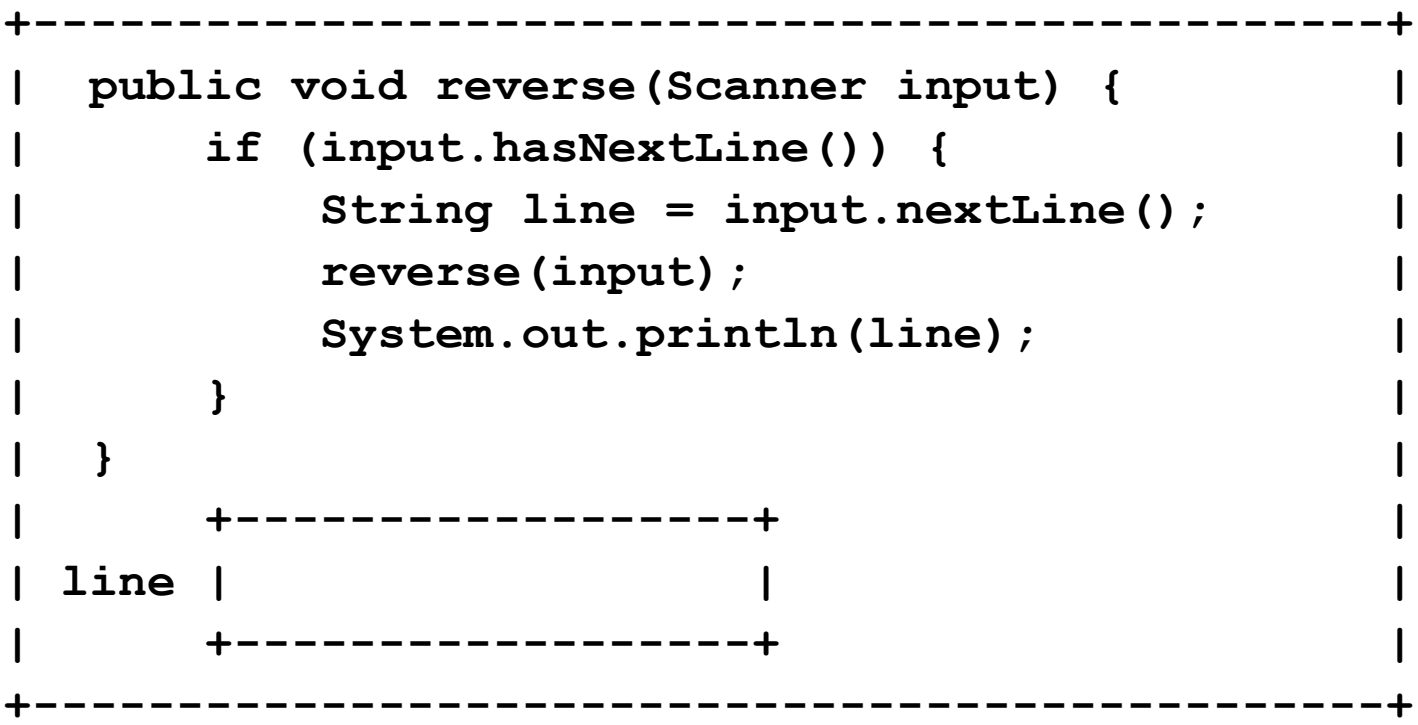

public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
--> reverse(input);
System.out.println(line);
}
}
line | "this"
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
--> reverse(input);
System.out.println(line);
}
}
line | "is"
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
--> reverse(input);
System.out.println(line);
}
}
line | "fun"

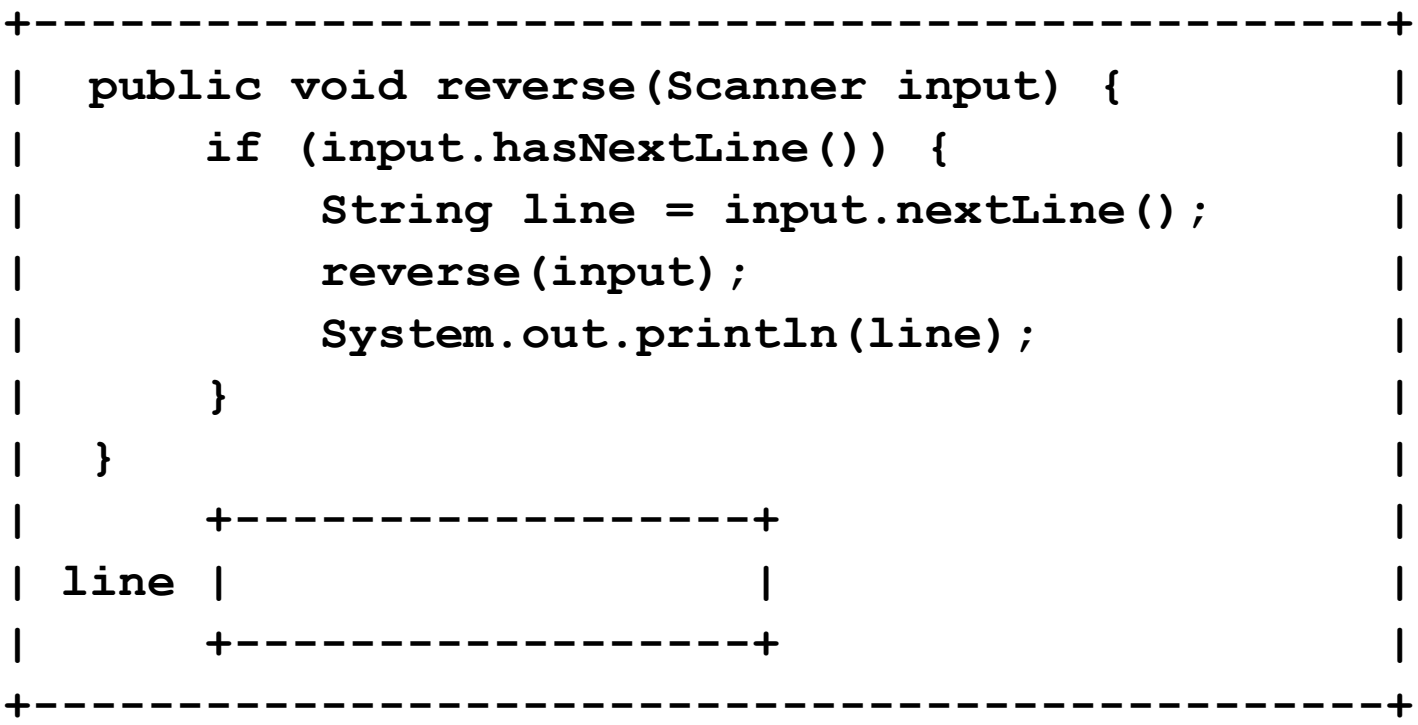

public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
--> reverse(input);
System.out.println(line);
}
}
line | "no?"
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line

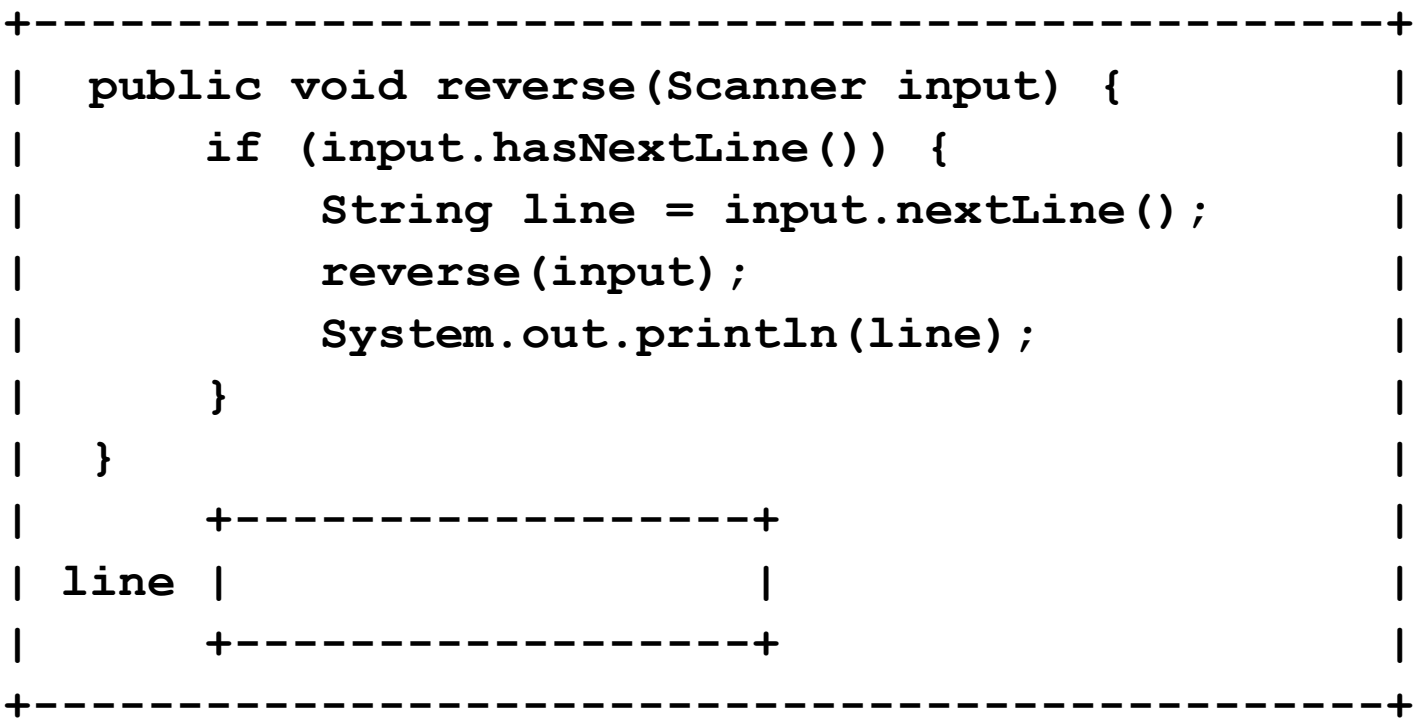

public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
--> System.out.println(line);
}
}
line | "no?"
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
--> System.out.println(line);
}
}
line | "fun"

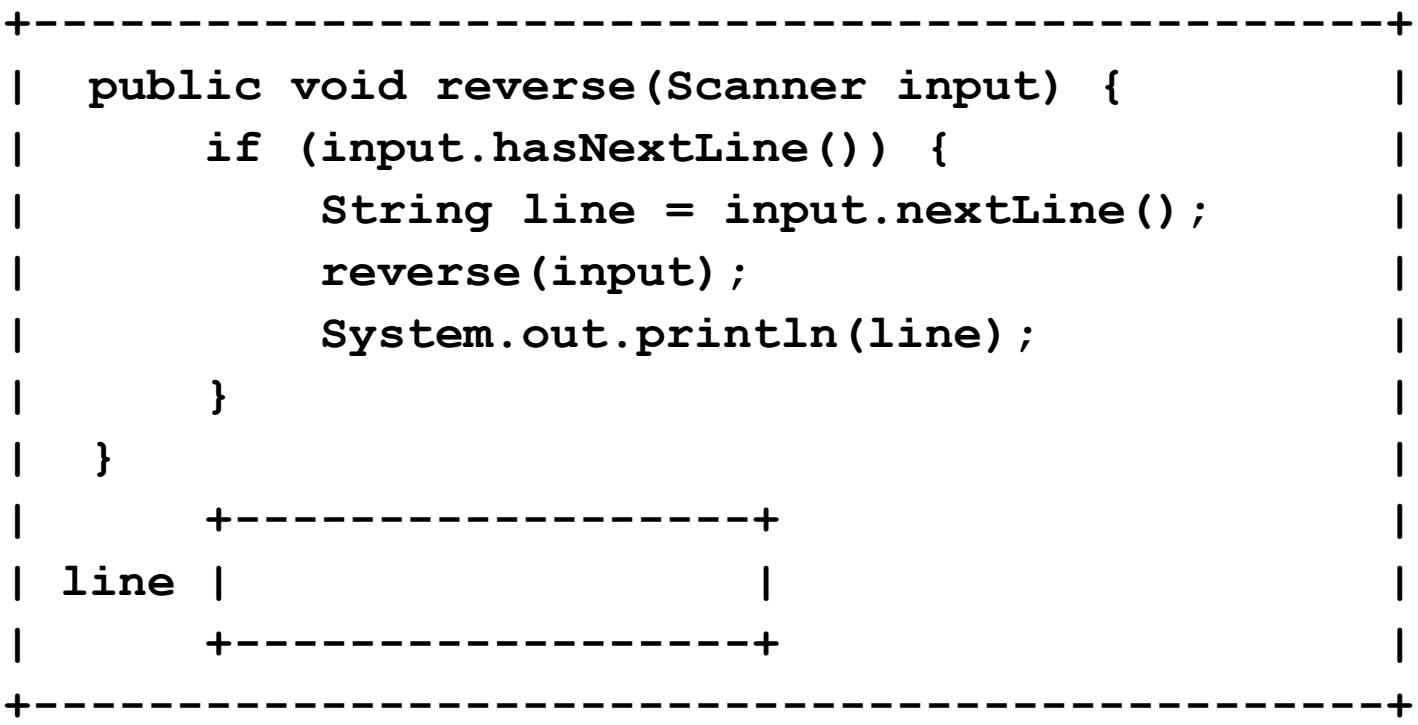

public void reverse(Scanner input) {
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
--> System.out.println(line);
}
}
line | "is"
public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
--> System.out.println(line);
}
}
line | "this"


public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line


public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line


public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line


public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line


public void reverse(Scanner input) {
if (input.hasNextLine()) {
String line = input.nextLine();
reverse(input);
System.out.println(line);
}
}
line