Building Java Programs

Chapter 2: Primitive Data and Definite Loops

Lecture outline

- managing complexity
  - variable scope
  - class constants
- drawing complex figures with `for` loops

Drawing complex figures

reading: 2.4 - 2.5

self-checks: 27

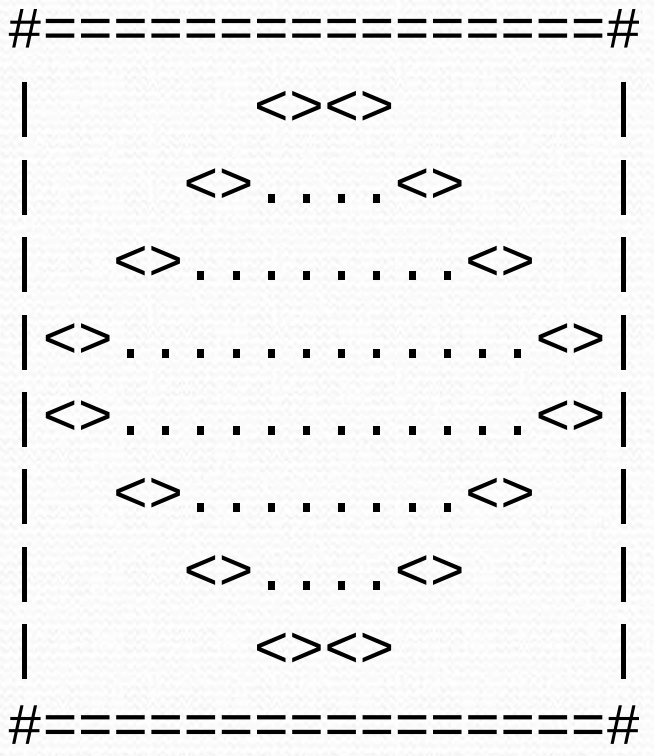
exercises: 16-17

projects: 1 - 4

Drawing complex figures

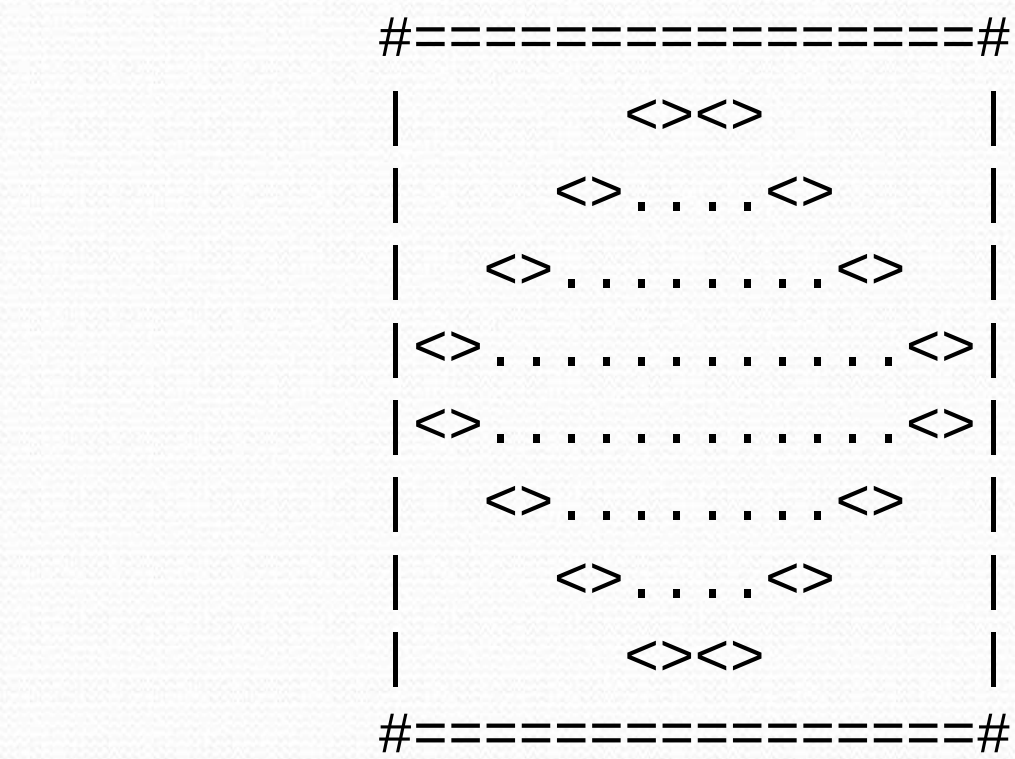
- Use nested `for` loops to produce the following output

- ASCII art?! We're in the new millennium, now…
  - graphics require a lot of finesse
  - a lot of the details are boring
  - this captures the algorithmic parts



Drawing complex figures

- Recommendations for approaching complexity:
  - 1. Write an English description of the steps required (*pseudo-code*)
  - 2. Create a table to see the patterns for different characters



Pseudo-code

- **pseudo-code**: An English description of an algorithm.

- Example: Drawing a 12 wide by 7 tall box of stars

```
print 12 stars.
for (each of 5 lines) {
    print a star.
    print 10 spaces.
    print a star.
}
print 12 stars.
```

```
************
*          *
*          *
*          *
*          *
*          *
************
```

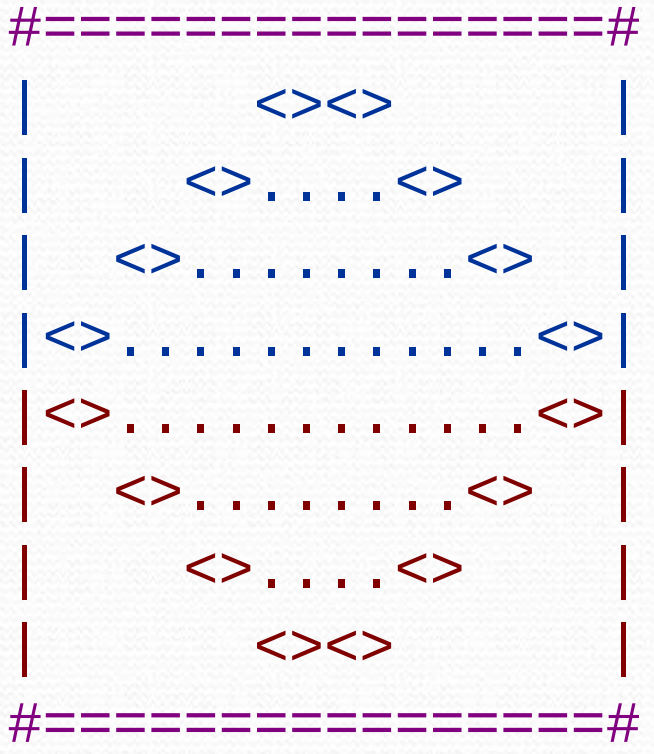
A pseudo-code algorithm

1. *Line with # , 16 =, then #*

2. *Top half. Each line:*
   - |
   - *spaces (increasing)*
   - <>
   - *dots (decreasing)*
   - <>
   - *spaces (same as above)*
   - |

3. *Bottom half (top half upside-down)*

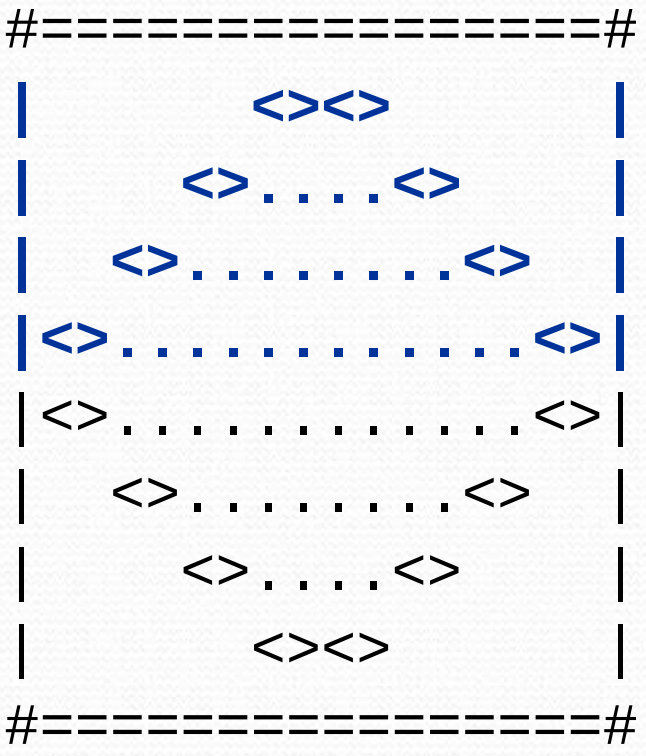
4. *Line with # , 16 =, then #*



Tables to examine output

- A table for "top half":
  - Compute spaces and dots expressions from line number

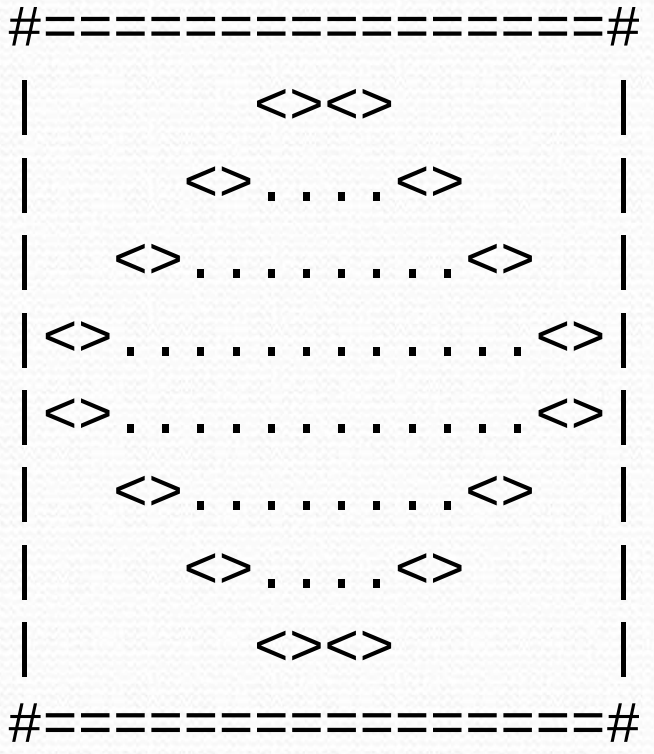
line	spaces	line * -2 + 8	dots	4 * line - 4
1	6	6	0	0
2	4	4	4	4
3	2	2	8	8
4	0	0	12	12



Implementing the figure

- Useful questions about the top half:
  - Number of (nested) loops per line?
  - What methods? (think structure and redundancy)

- Useful to write comments first



Partial solution

```
// Prints the expanding pattern of <> for the top half of the figure.
public static void drawTopHalf() {
    for (int line = 1; line <= 4; line++) {
        System.out.print("|");

        for (int space = 1; space <= (line * -2 + 8); space++) {
            System.out.print(" ");
        }

        System.out.print("<>");

        for (int dot = 1; dot <= (line * 4 - 4); dot++) {
            System.out.print(".");
        }

        System.out.print("<>");

        for (int space = 1; space <= (line * -2 + 8); space++) {
            System.out.print(" ");
        }

        System.out.println("|");
    }
}
```

Scope and class constants

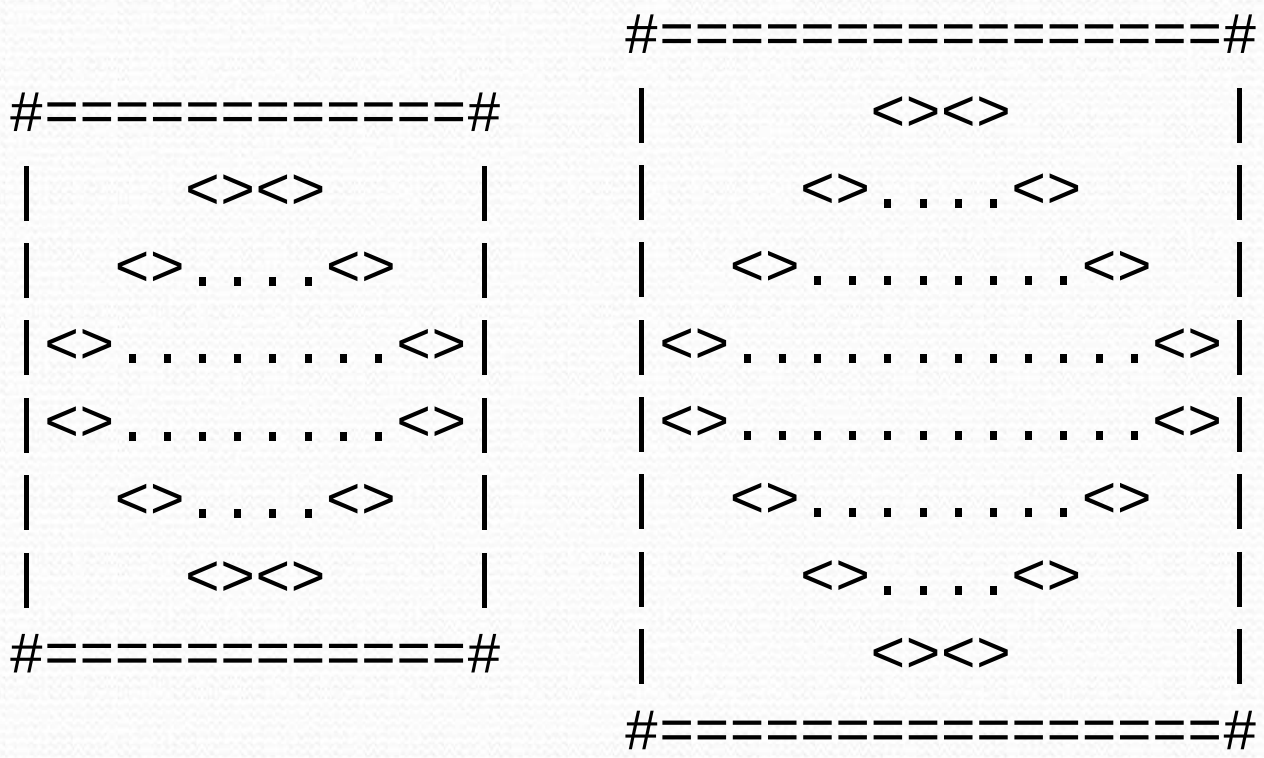
reading: 2.4

self-check: 28

exercises: 11

Scaling the mirror

- Drawing symmetrical figures seems useful enough, but they'll often need to scale
  - Imagine zooming in on a mirror in a CG movie



Variable scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the { } braces
    - A variable declared in a for loop exists only in that loop.
    - A variable declared in a method exists only in that method.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    // i no longer exists here
} // x ceases to exist here
```

i's scope

x's scope

Scope implications

- Variables without overlapping scope can have same name
- A variable can't be used outside of its scope

```
public static void printHellaStars() {
    for (int line = 1; line <= 4; line++) {
        for (int stars = 1; stars <= 100 * line; stars++) {
            System.out.print("*");
        }
        System.out.println();
    }
    int line = 23; // fine: outside of outer for loop scope
}

public static void uselessMethod() {
  int uselessVar = 4;
  int uselessVar = 0;   // ERROR: overlapping scope
  line = 0;             // ERROR: outside scope
}
```

Problem: redundant values

- A normal variable's scope is not large enough to fix this:

```
public static void main(String[] args) {
    int max = 3;
    printTop();
    printBottom();
}

public static void printTop() {
    for (int i = 1; i <= max; i++) {        // ERROR: max not found
        for (int j = 1; j <= i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}

public static void printBottom() {
    for (int i = max; i >= 1; i--) {        // ERROR: max not found
        for (int j = i; j >= 1; j--) {
            System.out.print(max);          // ERROR: max not found
        }
        System.out.println();
    }
}
```

Class constants

- **class constant**: A value visible to the whole program.
  - value can only be set at declaration
  - value can't be changed while the program is running

- Syntax:

```
public static final <type> <name> = <value>;
```

  - Name is usually in ALL_UPPER_CASE.
  - Examples:

```
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 3.5;
public static final int SSN = 658234569;
```

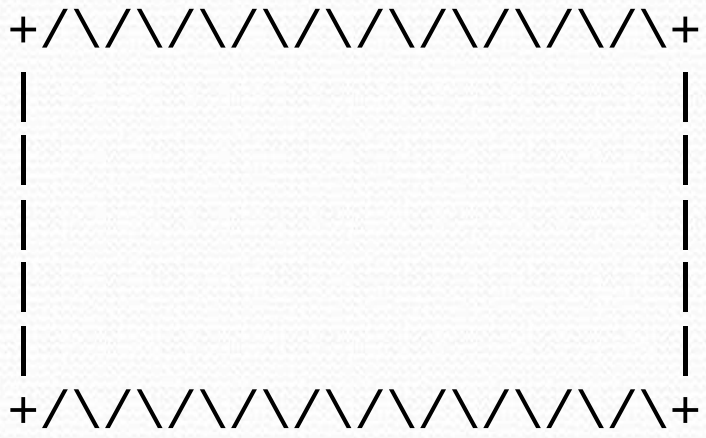
Class constant example

- Making the 3 a class constant removes the redundancy:

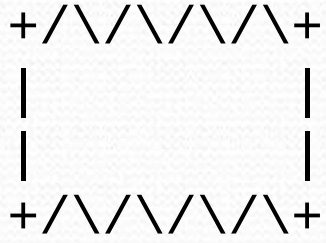
```
public static final int MAX_VALUE = 3;

public static void main(String[] args) {
    printTop();
    printBottom();
}

public static void printTop() {
    for (int i = 1; i <= MAX_VALUE; i++) {
        for (int j = 1; j <= i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}

public static void printBottom() {
    for (int i = MAX_VALUE; i >= 1; i--) {
        for (int j = i; j >= 1; j--) {
            System.out.print(MAX_VALUE);
        }
        System.out.println();
    }
}
```

Constants and figures

- Consider the task of drawing the following scalable figure:



Hey, multiples of 5 keep coming up…



Repetitive figure code

```
public class Sign {


    public static void main(String[] args) {
        drawLine();
        drawBody();
        drawLine();
    }

    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= 10; i++) {
            System.out.print("/\\");
        }
        System.out.println("+");
    }

    public static void drawBody() {
        for (int line = 1; line <= 5; line++) {
            System.out.print("|");
            for (int spaces = 1; spaces <= 20; spaces++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```

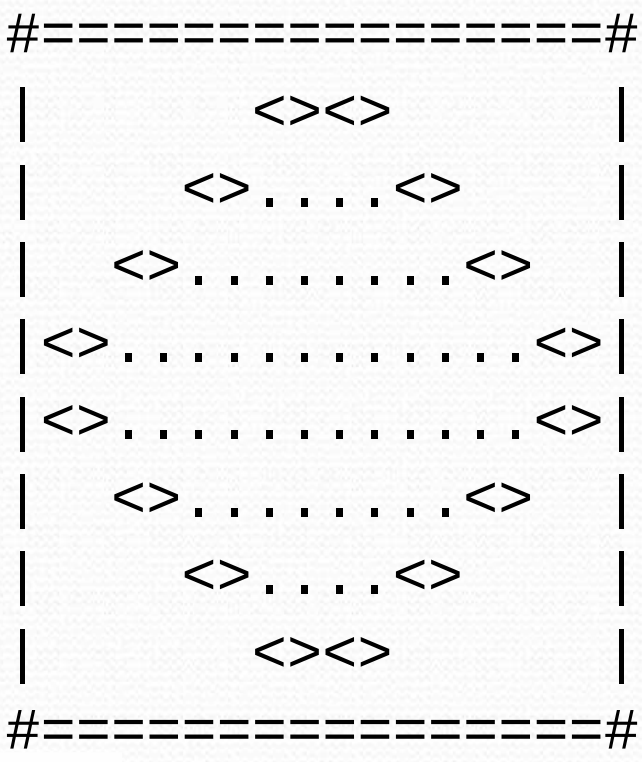
Fixing our code: constant

```
public class Sign {
    public static final int HEIGHT = 5;

    public static void main(String[] args) {
        drawLine();
        drawBody();
        drawLine();
    }

    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= HEIGHT * 2; i++) {
            System.out.print("/\\");
        }
        System.out.println("+");
    }

    public static void drawBody() {
        for (int line = 1; line <= HEIGHT; line++) {
            System.out.print("|");
            for (int spaces = 1; spaces <= HEIGHT * 4; spaces++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```

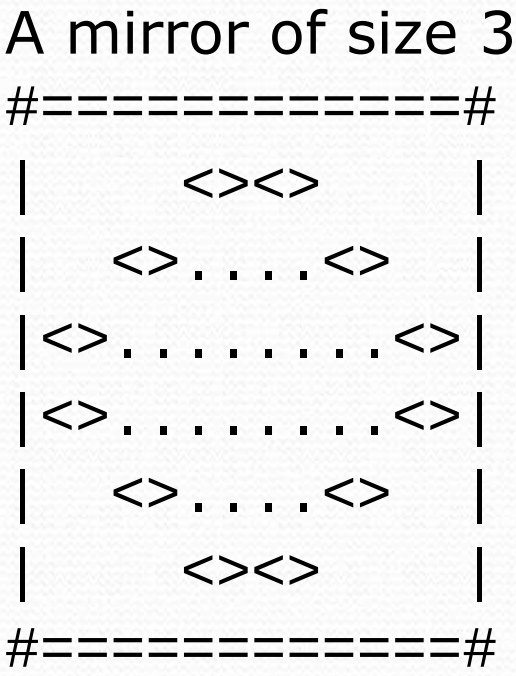
Complex figure w/ constant

- Modify the Mirror code to be resizable using a constant.

A mirror of size 4:



A mirror of size 3:



Loop tables and constant

- Let's modify our loop table to use SIZE
  - This can change the *b* in $y = mx + b$

SIZE	line	spaces	-2*line + (2*SIZE)	dots	4*line - 4
4	1,2,3,4	6,4,2,0	-2*line + **8**	0,4,8,12	4*line - 4
3	1,2,3	4,2,0	-2*line + **6**	0,4,8	4*line - 4

```
#================#
|       <><>     |
|     <>....<>   |
|   <>........<> |
| <>............<>|
| <>............<>|
|   <>........<> |
|     <>....<>   |
|       <><>     |
#================#
```

```
#============#
|     <><>   |
|   <>....<> |
| <>........<>|
| <>........<>|
|   <>....<> |
|     <><>   |
#============#
```

Partial solution

```
public static final int SIZE = 4;

// Prints the expanding pattern of <> for the top half of the figure.
public static void drawTopHalf() {
    for (int line = 1; line <= SIZE; line++) {
        System.out.print("|");
        for (int space = 1; space <= (line * -2 + (2 * SIZE)); space++) {
            System.out.print(" ");
        }
        System.out.print("<>");
        for (int dot = 1; dot <= (line * 4 - 4); dot++) {
            System.out.print(".");
        }
        System.out.print("<>");
        for (int space = 1; space <= (line * -2 + (2 * SIZE)); space++) {
            System.out.print(" ");
        }
        System.out.println("|");
    }
}
```

Observations about constant

- The constant can change the "intercept" in an expression.
  - Usually the "slope" is unchanged.

```
public static final int SIZE = 4;

for (int space = 1; space <= (line * -2 + (2 * SIZE)); space++) {
  System.out.print(" ");
}
```

- It doesn't replace *every* occurrence of the original value.

```
for (int dot = 1; dot <= (line * 4 - 4); dot++) {
    System.out.print(".");
}
```