

# Building Java Programs

Chapter 2

Lecture 2-3: Loop Figures and Constants

**reading: 2.4 - 2.5**

1

## Drawing complex figures

- Use nested `for` loops to produce the following output.
- Why draw ASCII art?
  - Real graphics require a lot of finesse
  - ASCII art has complex patterns
  - Can focus on the algorithms

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

2

# Development strategy

- Recommendations for managing complexity:

1. Design the program (think about steps or methods needed).
  - write an English description of steps required
  - use this description to decide the methods

2. Create a table for patterns of characters

- use tables to write your `for` loops

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

3

# 1. 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.
```

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

4

# Pseudo-code algorithm

## 1. Line

- # , 16 =, #

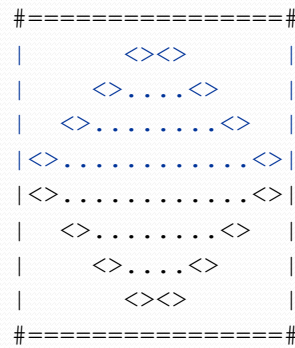
## 2. Top half

- |
- spaces (decreasing)
- <>
- dots (increasing)
- <>
- spaces (same as above)
- |

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

## 4. Line

- # , 16 =, #



5

# Methods from pseudocode

```
public class Mirror {
    public static void main(String[] args) {
        line();
        topHalf();
        bottomHalf();
        line();
    }

    public static void topHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }

    public static void bottomHalf() {
        for (int line = 1; line <= 4; line++) {
            // contents of each line
        }
    }

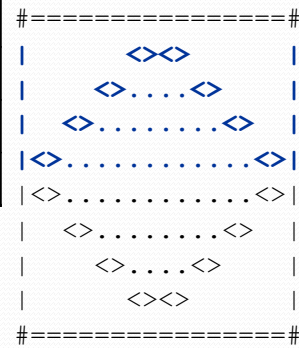
    public static void line() {
        // ...
    }
}
```

6

## 2. Tables

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

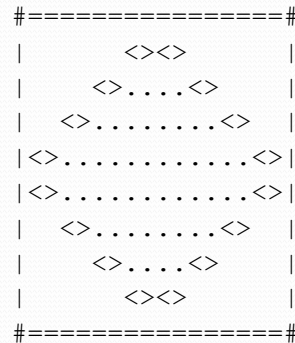
line	spaces	$-2 * \text{line} + 8$	dots	$4 * \text{line} - 4$
1	6	6	0	0
2	4	4	4	4
3	2	2	8	8
4	0	0	12	12



7

## 3. Writing the code

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



8

## Partial solution

```
// Prints the expanding pattern of <> for the top half of the figure.
public static void topHalf() {
    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("|");
    }
}
```

9

## Class constants and scope

**reading: 2.4**

10

## Scaling the mirror

- Let's modify our Mirror program so that it can scale.
  - The current mirror (left) is at size 4; the right is at size 3.
- We'd like to structure the code so we can scale the figure by changing the code in just one place.

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

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

11

## Limitations of variables

- Idea: Make a variable to represent the size.
  - Use the variable's value in the methods.
- Problem: A variable in one method can't be seen in others.

```
public static void main(String[] args) {
    int size = 4;
    topHalf();
    printBottom();
}

public static void topHalf() {
    for (int i = 1; i <= size; i++) { // ERROR: size not found
        ...
    }
}

public static void bottomHalf() {
    for (int i = size; i >= 1; i--) { // ERROR: size not found
        ...
    }
}
```

12

# 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

13

# Scope implications

- Variables without overlapping scope can have same name.

```
for (int i = 1; i <= 100; i++) {  
    System.out.print("/");  
}  
for (int i = 1; i <= 100; i++) { // OK  
    System.out.print("\\");  
}  
int i = 5; // OK: outside of loop's scope
```

- A variable can't be declared twice or used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {  
    int i = 2; // ERROR: overlapping scope  
    System.out.print("/");  
}  
i = 4; // ERROR: outside scope
```

14





## 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("|");
        }
    }
}
```

17

## Adding a 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("|");
        }
    }
}
```

18

# Complex figure w/ constant

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

A mirror of size 4:

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

A mirror of size 3:

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

19

# Using a constant

- Constant allows many methods to refer to same value:

```
public static final int SIZE = 4;  
  
public static void main(String[] args) {  
    topHalf();  
    printBottom();  
}  
  
public static void topHalf() {  
    for (int i = 1; i <= SIZE; i++) { // OK  
        ...  
    }  
}  
  
public static void bottomHalf() {  
    for (int i = SIZE; i >= 1; i--) { // OK  
        ...  
    }  
}
```

20

# Loop tables and constant

- Let's modify our loop table to use `SIZE`
  - This can change the amount added in the loop expression

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	$4*line - 4$

```

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

```

21

# Partial solution

```

public static final int SIZE = 4;
// Prints the expanding pattern of <> for the top half of the figure.
public static void topHalf() {
    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("|");
    }
}

```

22

## 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(".");
}
```

23

## Assignment 2: ASCII Art



24