Building Java Programs
Chapter 2

Definite Loops

Definite Loops

Reading

- Building Java Programs, Ch. 2.3 - 3.4

Learning Outcomes

- For loops in Java
- Increment / decrement operators
- Variable scope
- Constants

Repetition with `for` loops

- So far, repeating a statement is redundant:

```
System.out.println("Homer says:");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```

- Java's **`for` loop** statement performs a task many times.

```
System.out.println("Homer says:");
for (int i = 1; i <= 4; i++) {   // repeat 4 times
    System.out.println("I am so smart");
}
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```

`for` loop syntax

```
for (initialization; test; update) {          } header
    statement;                                 }
    statement;                                 }
    ...                                        } body
    statement;                                 }
}
```

– Perform **initialization** once.
– Repeat the following:
  - Check if the **test** is true. If not, stop.
  - Execute the **statement**s.
  - Perform the **update**.

Initialization

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Tells Java what variable to use in the loop
  - Performed once as the loop begins
  - The variable is called a loop counter
    - can use any name, not just `i`
    - can start at any value, not just `1`

Test

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Tests the loop counter variable against a limit
  - Uses comparison operators:

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable`++;`	**variable** `=` **variable** `+ 1;`
variable`--;`	**variable** `=` **variable** `- 1;`

```
int x = 2;
x++;                    // x = x + 1;
                           // x now stores 3

double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                           // gpa now stores 1.5
```

Modify-and-assign

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += **value**;	**variable** = **variable** + **value**;
variable -= **value**;	**variable** = **variable** - **value**;
variable *= **value**;	**variable** = **variable** * **value**;
variable /= **value**;	**variable** = **variable** / **value**;
variable %= **value**;	**variable** = **variable** % **value**;

```
x += 3;                 // x = x + 3;
gpa -= 0.5;             // gpa = gpa - 0.5;
number *= 2;            // number = number * 2;
```

Repetition over a range

```
System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for (int i = 1; i <= 6; i++) {
        System.out.println(i + " squared = " + (i * i));
}
```

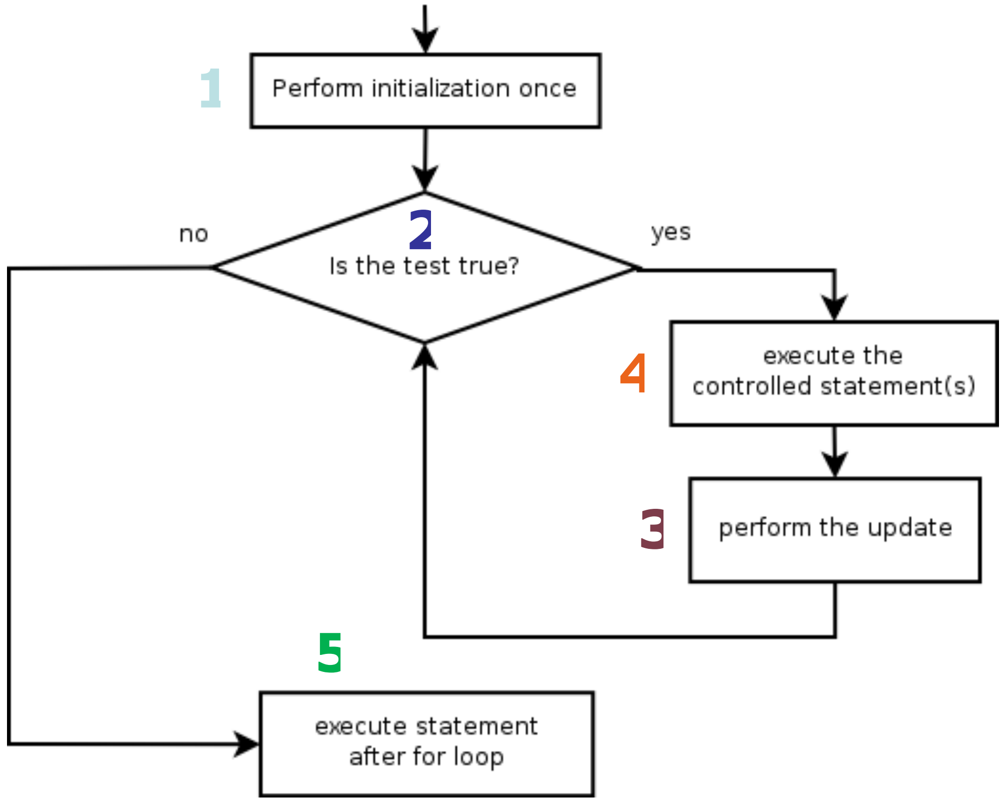
- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
                1       2     3
for (int i = 1; i <= 4; i++) {
   4System.out.println(i + " squared = " + (i * i));
}
System.out.println("Whoo!");
5
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```


1
Perform initialization once
2
Is the test true?
no
yes
4
execute the controlled statement(s)
3
perform the update
5
execute statement after for loop

Multi-line loop body

```
System.out.println("+-----+");
for (int i = 1; i <= 3; i++) {
    System.out.println("\\     /");
    System.out.println("/     \\");
}
System.out.println("+-----+");
```

– Output:

```
+-----+
\     /
/     \
\     /
/     \
\     /
/     \
+-----+
```

Expressions for counter

```
int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.println(i * 1.8 + 32);
}
```

– Output:

```
  26.6
28.4
30.2
32.0
33.8
35.6
```

System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line

```
int highestTemp = 5;
for (int i = -3; i <= highestTemp / 2; i++) {
    System.out.print((i * 1.8 + 32) + "  ");
}
```

- Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

- Concatenate `"  "` to separate the numbers

Counting down

- The **update** can use `--` to make the loop count down.
  - The **test** must say > instead of <

```
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
     System.out.print(i + ", ");
}
System.out.println("blastoff!");
System.out.println("The end.");
```

  - Output:

```
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
blastoff!
  The end.
```

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.

```
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
```

Nested loops

- **nested loop**: A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();    // to end the line
}
```

- Output:

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

- The outer loop repeats 5 times; the inner one 10 times.
  - "sets and reps" exercise analogy

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Output:

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

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

- Output:

```
1
22
333
4444
55555
```

Common errors

- Both of the following sets of code produce infinite loops:

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}

for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Complex lines

- What nested `for` loops produce the following output?

inner loop (repeated characters on each line)

```
....1
...2
..3
.4
5
```

outer loop (loops 5 times because there are 5 lines)

- We must build multiple complex lines of output using:
  - an outer "vertical" loop for each of the lines
  - inner "horizontal" loop(s) for the patterns within each line

Outer and inner loop

- First write the outer loop, from 1 to the number of lines.

```
for (int line = 1; line <= 5; line++) {
    ...
}
```

- Now look at the line contents. Each line has a pattern:
  - some dots (0 dots on the last line), then a number

```
....1
...2
..3
.4
5
```

  - Observation: the number of dots is related to the line number.

Mapping loops to numbers

```
for (int count = 1; count <= 5; count++) {
    System.out.print( ... );
}
```

– What statement in the body would cause the loop to print:

```
   4 7 10 13 16
```

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
```

Loop tables

- What statement in the body would cause the loop to print:
  `2 7 12 17 22`

- To see patterns, make a table of `count` and the numbers.
  - Each time count goes up by 1, the number should go up by 5.
  - But `count * 5` is too great by 3, so we subtract 3.

`count`	number to print	`5 * count`	`5 * count - 3`
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Loop tables question

- What statement in the body would cause the loop to print:
  `17 13 9 5 1`

- Let's create the loop table together.
  - Each time `count` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

`count`	number to print	`-4 * count`	`-4 * count + 21`
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

Nested `for` loop exercise

- Make a table to represent any patterns on each line.

```
....1
...2
..3
.4
5
```

line	# of dots	`-1 * line`	`-1 * line + 5`
1	4	-1	4
2	3	-2	3
3	2	-3	2
4	1	-4	1
5	0	-5	0

- To print a character multiple times, use a `for` loop.

```
for (int j = 1; j <= 4; j++) {
    System.out.print(".");          // 4 dots
}
```

Nested `for` loop solution

- Answer:

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

- Output:

```
....1
...2
..3
.4
5
```

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
        System.out.print(line);
    }
    System.out.println();
}
```

- Answer:

```
....1
...22
..333
.4444
55555
```

Nested `for` loop exercise

- Modify the previous code to produce this output:

```
....1
...2.
..3..
.4...
5....
```

- Answer:

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}
```

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

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

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 of patterns of characters
     - use table to write your `for` loops

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

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.
```

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

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 =, #

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

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() {
        // ...
    }
}
```

2. Tables

- A table for the 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

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

3. Writing the code

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

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

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("|");
    }
}
```

Class constants and scope

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.

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

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

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
        ...
    }
}
```

Class constants

- **class constant**: A fixed value visible to the whole program.
  - value can be set only at declaration; cannot be reassigned

- 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;
    ```

Constants and figures

- Consider the task of drawing the following scalable figure:

```
+/\/\/\/\/\/\/\/\/\/\+
|                    |
|                    |
|                    |
|                    |
|                    |
+/\/\/\/\/\/\/\/\/\/\+
```

Multiples of 5 occur many times

```
+/\/\/\/\+
|        |
|        |
+/\/\/\/\+
```

The same figure at size 2

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

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

Complex figure w/ constant

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

A mirror of size 4:

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

A mirror of size 3:

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

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
        ...
    }
}
```

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

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

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

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("|");
    }
}
```

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(".");
}
```