Building Java Programs Chapter 2

Primitive Data and Definite Loops

Data types

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types

  - Examples: integer, real number, string

- Internally, computers store everything as 1s and 0s

  ```
  104    01101000
  "hi"    011101000110101
  ```

Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

Name	Description		Examples
int	integers	(up to $2^{31} - 1$)	42, -3, 0, 926394
double	real numbers	(up to 10^{308})	3.1, -0.25, 9.4e3
char	single text characters		'a', 'X', '?', '\n'
boolean	logical values		true, false

- Why does Java distinguish integers vs. real numbers?

Expressions

- **expression**: A value or operation that computes a value.

  - Examples:
    ```
    1 + 4 * 5
    (7 + 2) * 6 / 3
    42
    ```

  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.

Arithmetic operators

- **operator**: Combines multiple values or expressions.

`+`	addition
`–`	subtraction (or negation)
`*`	multiplication
`/`	division
`%`	modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.
  - `1 + 1` evaluates to `2`
  - `System.out.println(3 * 4);` prints `12`
    - How would we print the text `3 * 4` ?

Integer division with /

- When we divide integers, the quotient is also an integer.
  - `14 / 4` **is** `3`, **not** `3.5`

```
         3                    4                        52
   4 )  14            10 )   45                 27 )  1425
        12                   40                       135
         2                    5                        75
                                                       54
                                                       21
```

- More examples:
  - `32 / 5`   **is** `6`
  - `84 / 10`   **is** `8`
  - `156 / 100` **is** `1`

  - Dividing by 0 causes an error when your program runs.

Integer remainder with %

- The `%` operator computes the remainder from integer division.
  - `14 % 4` **is** `2`
  - `218 % 5` **is** `3`

```
         3
   4 )  14
        12
         2
```

```
          43
    5 )  218
         20
          18
          15
           3
```

What is the result?
```
45 % 6
2 % 2
8 % 20
11 % 0
```

- Applications of `%` operator:
  - Obtain last digit of a number: `230857 % 10` **is** `7`
  - Obtain last 4 digits: `658236489 % 10000` **is** `6489`
  - See whether a number is odd: `7 % 2` **is** `1`, `42 % 2` **is** `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `*` `/` `%` have a higher level of precedence than `+` `-`
    `1 + ` **`3 * 4`** `            ` is `13`

    `6 + ` **`8 / 2`** `* 3`
    `6 + ` `    ` **`4`** `    ` **`* 3`**
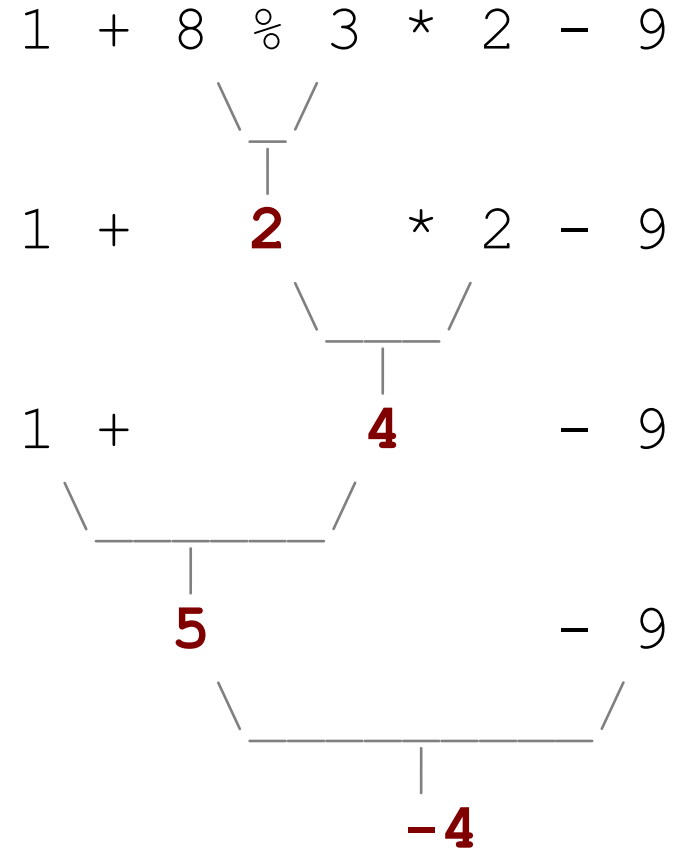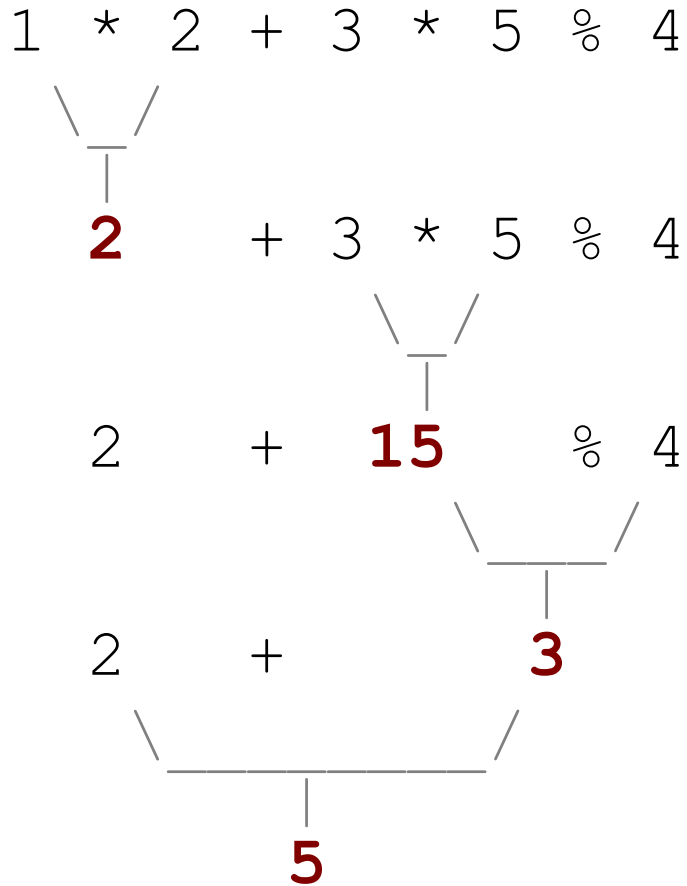   `6 + ` `    12` `            ` is `18`

  - Parentheses can force a certain order of evaluation:
    `(1 + 3) * 4` `            ` is `16`

  - Spacing does not affect order of evaluation
    `1+3 * 4-2` `            ` is `11`

```
1  *  2  +  3  *  5  %  4
   \  /
    |
    2      +  3  *  5  %  4
               \   /
                |
    2      +   15      %  4
                        \    /
                         |
    2      +            3
     \                 /
      |
          5
```

```
1  +  8  %  3  *  2  -  9
      \  /
       |
1  +     2       *  2  -  9
          \     /
           |
1  +            4       -  9
 \                     /
  |
      5                  -  9
       \                /
        |
           -4
```

Precedence questions

- What values result from the following expressions?

  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`

Real numbers (type `double`)

- Examples: `6.022, -42.0, 2.143e17`

  - Placing `.0` or `.` after an integer makes it a `double`.

- The operators `+ - * / % ()` all still work with `double`.

  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`

  - Precedence is the same: `()` before `* / %` before `+ -`

Real number example

```
2.0 * 2.4 + 2.25 * 4.0 / 2.0
     \____/
        |
     4.8      + 2.25 * 4.0 / 2.0
                    \____/
                       |
     4.8     +     9.0    / 2.0
                      _____/
                          |
     4.8     +           4.5
          _____/
                      |
                     9.3
```
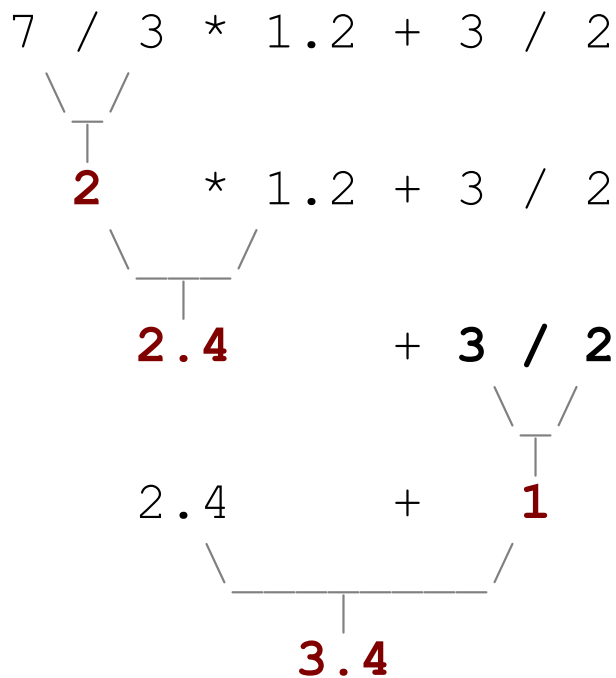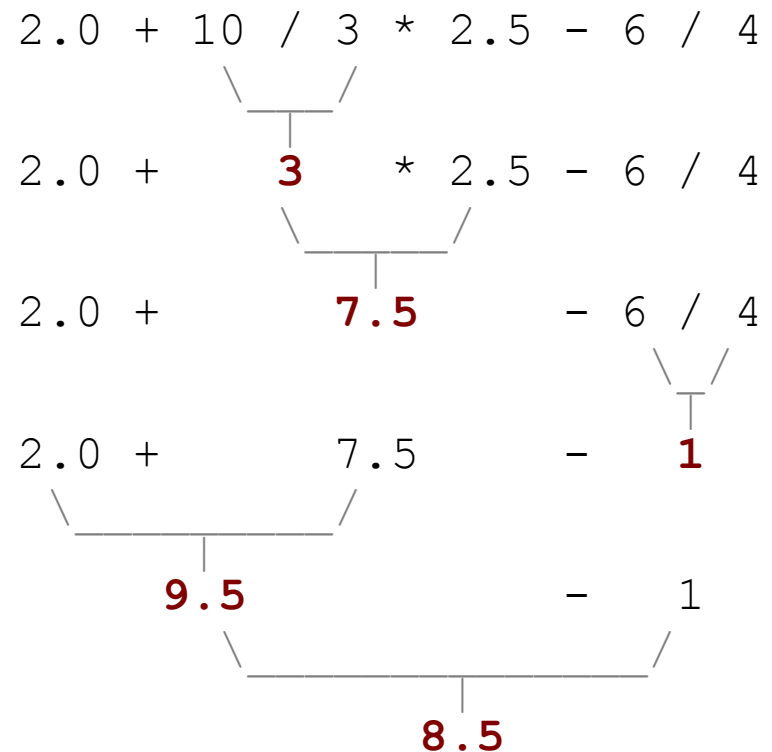
Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`

- The conversion is per-operator, affecting only its operands.

```
7 / 3 * 1.2 + 3 / 2
  \ | /
    |
    2    * 1.2 + 3 / 2
      \      |
           2.4        + 3 / 2
                        \ | /
                          |
         2.4       +      1
           \              |
               \ _____ /
                    |
                   3.4
```

```
2.0 + 10 / 3 * 2.5 - 6 / 4
        \ | /
          |
2.0 +     3    * 2.5 - 6 / 4
           \         |
2.0 +         7.5       - 6 / 4
                        \ | /
                          |
2.0 +         7.5       -    1
   \                    |
     \ _____ /
              |
             9.5              -    1
               \                   |
                 \ _____ /
                        |
                       8.5
```

  - `3 / 2` is `1` above, not `1.5`.

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42    is "hello42"
1 + "abc" + 2   is "1abc2"
"abc" + 1 + 2   is "abc12"
1 + 2 + "abc"   is "3abc"
"abc" + 9 * 3   is "abc27"
"1" + 1         is "11"
4 - 1 + "abc"   is "3abc"
```

- Use + to print a string and an expression's value together.

    - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`

  - Output: `Grade: 83.5`

Variables

Receipt example

What's bad about the following code?

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  type name;

  - The name is an *identifier*.

x	
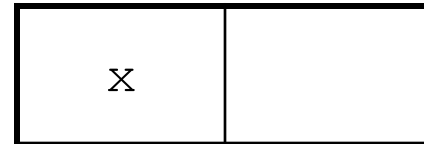
  - `int x;`

myGPA	

  - `double myGPA;`

Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**;

  - ```
    int x;
    ```
    x = 3;

x	3

  - ```
    double myGPA;
    ```
    myGPA = 1.0 + 2.25;

myGPA	3.25

Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
System.out.println("x is " + x);      // x is 3

System.out.println(5 * x - 1);        // 5 * 3 - 1
```

- You can assign a value more than once:

```
int x;
x = 3;
System.out.println(x + " here");      // 3 here

x = 4 + 7;
System.out.println("now x is " + x);  // now x is 11
```

x	**11**

Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

  type name = **value**;

  – `double myGPA = 3.95;`

myGPA	3.95

  – `int x = (11 % 3) + 12;`

x	14

Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.

    =   means, *"store the value at right in variable at left"*

    - The right side expression is evaluated first,
      and then its result is stored in the variable at left.

- What happens here?

```
int x = 3;
x = x + 2;     // ???
```

x	5

Assignment and types

- A variable can only store a value of its own type.

  - `int x = 2.5;`   **// ERROR: incompatible types**

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.

  - `double myGPA = 4;`

myGPA	4.0

  - `double avg = `**`11 / 2;`**

avg	**5.0**

  - Why does `avg` store `5.0` and not `5.5` ?

Compiler errors

- A variable can't be used until it is assigned a value.

  - ```
    int x;
    System.out.println(x);    // ERROR: x has no value
    ```

- You may not declare the same variable twice.

  - ```
    int x;
    int x;                     // ERROR: x already exists
    ```

  - ```
    int x = 3;
    int x = 5;                 // ERROR: x already exists
    ```

    - How can this code be fixed?

Printing a variable's value

- Use + to print a string and a variable's value on one line.

  - ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 11 + 17 + 4 + 19 + 14;
    System.out.println("There are " + students +
                       " students in the course.");
    ```

  - Output:

    ```
    Your grade was 83.2
    There are 65 students in the course.
    ```

Receipt question

Improve the receipt program using variables.

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);

        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);

        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);

        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                          (38 + 40 + 30) * .15 +
                          (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```

The `for` loop

Repetition with `for` loops

- So far, repeating a statement is redundant:
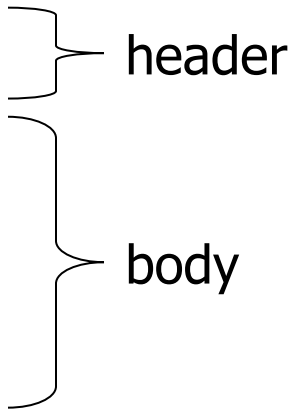
  ```
  System.out.println("Homer says:");
  System.out.println("I am so smart");
  System.out.println("I am so smart");
  System.out.println("I am so smart");
  System.out.println("I am so smart");
  System.out.println("S-M-R-T... I mean S-M-A-R-T");
  ```

- 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) {
    statement;
    statement;
    ...
    statement;
}
```

header

body

– 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

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

1 **2** **3**

5

Output:

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


36
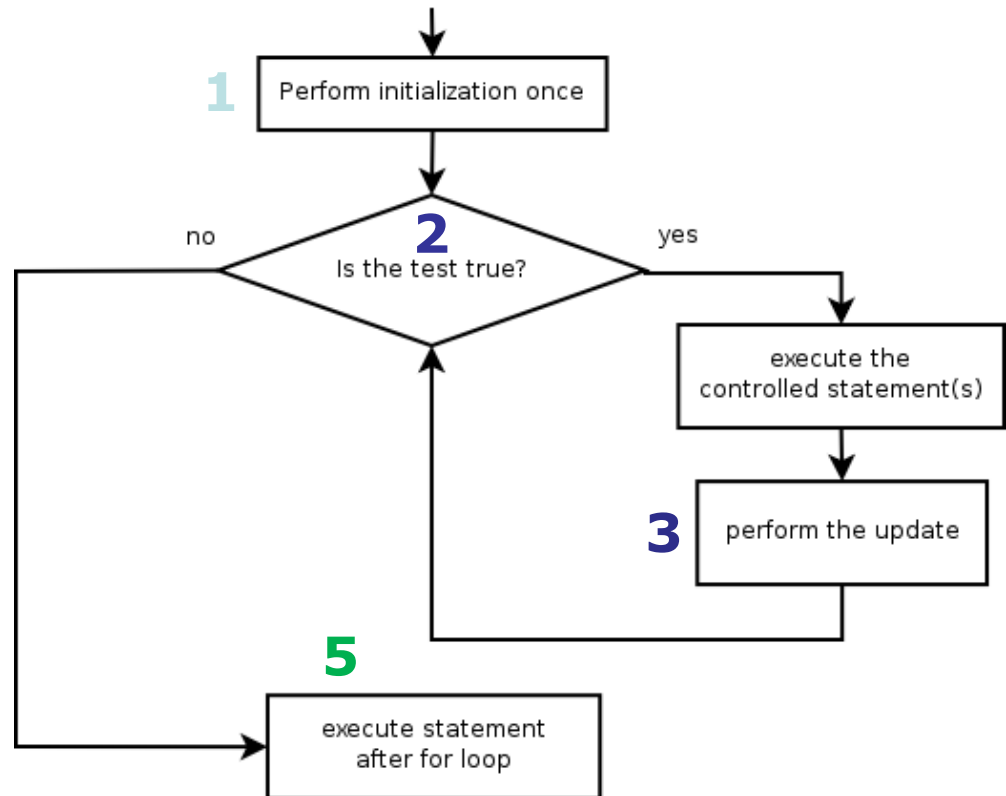
Multi-line loop body

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

– Output:
```
+----+
\    /
/    \
\    /
/    \
\    /
/    \
+----+
```

Expressions for counter

```java
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.
```

Nested `for` loops

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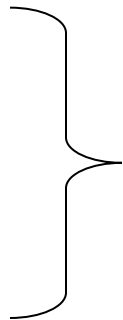
```
....1
...2
..3
.4
5
```

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

- 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.

```java
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 =, #

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

```
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?

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

```java
// 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
        ...
    }
}
```

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.

```java
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.

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

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

69

Repetitive figure code

```java
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
        ...
    }
}
```

- 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

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

```java
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(".");
}
```