Building Java Programs

Chapter 2: Primitive Data and Definite Loops

Chapter outline

- data concepts
  - primitive types, expressions, and precedence
  - variables: declaration, initialization, assignment
  - mixing types: casting, string concatenation
  - modify-and-reassign operators
  - System.out.print
- repetition
  - the for loop
  - nested loops
- managing complexity
  - variable scope
  - class constants
- drawing complex figures

Programs that examine data

- We have printed text with println and strings:
  System.out.println("Hello, world!");
- Now we will learn how to print and manipulate other kinds of data, such as numbers:
  System.out.println(42);
  System.out.println(1 + 2 * 3);
  System.out.println(12.5 / 3.0); // 4.1666666

Data types

- type: A category or set of data values.
  - Many languages have a notion of data types and ask the programmer to specify what type of data is being manipulated.
  - Examples: integer, real number, string.
- Internally, the computer stores all data as 0s and 1s.
  - Examples: 43 → 010101

Java's primitive types

- primitive types: Java's built-in simple data types for numbers, text characters, and logic.
  - Java has eight primitive types.
  - Types that are not primitive are called object types. (see later)
- Four primitive types we will use:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer [whole numbers]</td>
<td>42, -3, 0, 1.5, 1.1</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>3.14, -0.3, 0, 4e3</td>
</tr>
<tr>
<td>char</td>
<td>single text characters</td>
<td>'a', 'A', '0', '0'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>
Expressions

- expression: A data value, or a set of operations that compute a data value.
  - Examples: $1 + 4 \times 3$
  - The simplest expression is a literal value. Examples: 1
  - A complete expression can use operators and parentheses. The values to which an operator applies are called operands.

- Five arithmetic operators we will use:
  - + addition
  - - subtraction or negation
  - * multiplication
  - / division
  - % modulus, a.k.a. remainder

Integer division with /

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

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>54</td>
<td>31</td>
</tr>
</tbody>
</table>

- More examples:
  - 1453 / 27 is 53
  - 34 / 10 is 3
  - 84 / 15 is 5

- Dividing by 0 causes a runtime error in your program.

Integer remainder with %

- The % operator computes the remainder from a division of two integers.
  - Examples: 14 % 4 is 2
  - Examples: 218 % 5 is 3

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>

- What are the results of the following expressions?
  - 43 % 6
  - 2 % 2
  - 3 % 20
  - 11 % 2

Applications of % operator

- Obtains the last digit (units place) of a number:
  - Example: From 213427, obtain the 7.

- Obtain the last 4 digits of a Social Security Number:
  - Example: From 678214399, obtain 6439.

- Obtains a number’s second-to-last digit (tens place):
  - Example: From 1342, obtain the 4.

- Use the % operator to see whether a number is odd:
  - Can it help us determine whether a number is divisible by 37?

Evaluating expressions

- When your Java program executes and encounters a line with an expression, the expression is evaluated (its value is computed).
  - The expression $3 \times 4$ is evaluated to obtain 12.
  - System.out.println ($3 \times 4$); prints 12, not $3 \times 4$.
    (How could we print the text $3 \times 4$ on the screen?)

- When an expression contains more than one operator of the same kind, it is evaluated left-to-right.
  - Examples: $1 + 2 - 3$ is $(1 + 2) - 3$ which is 6
  - $1 - 2 - 3$ is $1 - (2 - 3)$ which is -4

Operator precedence

- precedence: Order in which operations are computed in an expression.
  - Multiplicative operators *, / have a higher level of precedence than additive operators +, -
  - Parentheses can be used to force a certain order of evaluation:
    - (1 + 3) * 4 is 16
  - Spacing does not affect order of evaluation:
    - 1 + 3 * 4 - 2 is 11
Precedence examples

- $1 + 2 * 3 / 5 - 4$
- $2 + 3 * 5 / 4$
- $2 + 15 / 4$
- $2 + 3 / 5$
- $1 - 4$

Precedence questions

- What values result from the following expressions?
  - $1 / 5$
  - $1 * 2 * 3 / 4$
  - $7 + 4 * 2$
  - $24 / 120 / 5$
  - $4 * 3 - 8 / 4$
  - $(5 - 7) * 4$
  - $4 + [18 + (17 - 12)]$

- Which parentheses above are unnecessary (which do not change the order of evaluation?)

Real numbers (double)

- Java can also manipulate real numbers (type double).
  - Examples: 6.022 -13.987 42.5 2.143=17

- The operators + - * / all work for real numbers.
  - The / produces an exact answer when used on real numbers.
  - Example: 15.0 / 2.0=7.5

- The same rules of precedence that apply to integers also apply to real numbers.
  - Evaluate [] before */ before +-.

Real number example

- $2.0 * 2.4 + 2.25 * 4.0 / 2.0$
  - $4.8 + 2.25 * 4.0 / 2.0$
  - $4.8 + 9.0 / 2.0$
  - $4.8 + 4.5$
  - $9.3$

Real number precision

- The computer internally represents real numbers in an imprecise way.
  - Example:
    - System.out.println(0.1 + 0.3);
    - The mathematically correct answer should be 0.3
    - Instead, the output is 0.3000000000000004

- Later we will learn some ways to produce a better output for examples like the above.

Mixing integers and reals

- When a Java operator is used on an integer and a real number, the result is a real number.
  - Examples: 42 * 3 is 126
  - $1 / 3.0$ is 0.3333333333333333

- The conversion occurs on a per-operator basis. It affects only its two operands.
  - $3 / 2$ produces a real number
  - $3 + 1$
  - $3 * 2$

- Notice how $3 / 2$ is still 1 above, not 1.5.
**Mixed types example**

- \(2.0 + 10 / 3 \times 2.5 - 6 / 4\)
- \(2.0 + \frac{3}{2.5 - 6 / 4}\)
- \(2.0 + \frac{7.5}{6 / 4}\)
- \(2.0 + \frac{7.5}{1}\)
- \(9.5 - 1\)
- \(8.5\)

**The computer's memory**

- Expressions are like using the computer as a calculator.
  - Calculators have "memory" keys to store and retrieve values.
    - In what situation(s) is this useful?
    - We'd like the ability to save and restore values in our Java programs, like the memory keys on the calculator.

**Variables**

- **variable**: A piece of your computer's memory that is given a name and type and can store a value.
  - Usage:
    - store an expression's result
    - store that result in a variable

**Declaring variables**

- **variable declaration statement**: A Java statement that creates a new variable of a given type.
  - A variable is declared by writing a statement that says its type and then its name.
  - Variables must be declared before they can be used.

**More on declaring variables**

- Declaring a variable sets aside a piece of memory in which you can store a value.

```java
int x;
int y;
```

- Part of the computer's memory:

```
x  y
```

(These memory slots have no values in them yet.)
Assignment statements

- **Assignment statement**: A Java statement that stores a value into a variable's memory location.
  - Variables must be declared before they can be assigned a value.

- **Assignment statement syntax**:
  ```java
  variable = value;
  ```
  - Example: `x = 3;`
  - Example: `myPA = 3.25;`

  `x` 3  myPA 3.25

More about assignment

- The `<value>` assigned can be a complex expression.
  - The expression is evaluated; the variable stores the result.
  - Example:
    ```java
    x = (2 + 1) / 3 + 5;
    ```

- A variable can be assigned a value more than once.
  - Example:
    ```java
    int y;
    y = 3;
    System.out.println(y); // 3
    ```

Assignment and algebra

- Though the assignment statement uses the = character, it is not an algebraic equation.
  - = means "store the value on the right in the variable on the left"
  - Some people read `x = 3` as "x becomes 3" or "x gets 3"
  - We would not say `3 = 1 + 2`; because 3 is not a variable.

- What happens when a variable is used on both sides of an assignment statement?
  ```java
  int x;
  x = 3;
  x = x + 2; // what happens?
  ```

  - The above wouldn’t make any sense in algebra...

Using variables' values

- Once a variable has been assigned a value, it can be used in an expression, just like a literal value.
  ```java
  int x;
  x = 3;
  System.out.println(x * 5 - 1);
  ```
  - The above has output equivalent to:
    ```java
    System.out.println(3 * 5 - 1);
    ```

Some errors

- A compiler error will result if you declare a variable twice, or declare two variables with the same name.
  - Example:
    ```java
    int x;
    int y; // ERROR: x already exists
    ```

- A variable that has not been assigned a value cannot be used in an expression or println statement.
  - Example:
    ```java
    int x;
    System.out.println(x); // ERROR: x has no value
    ```

Assignment and types

- A variable can only store a value of its own type.
  - Example: `int x; x = 2.5; // ERROR: x can only store int`

- An int value can be stored in a double variable.
  - The value is converted into the equivalent real number.
  - Example: `double myPA; myPA = 3.2;`
    ```java
    myPA 3.2
    ```
Assignment examples

What is the output of the following Java code?
```java
int number;
number = 2 + 3 * 4;
number = 16 % 6;
System.out.println(number); // Output: 4
```

What is the output of the following Java code?
```java
double average;
average = (11 + 6) / 2;
System.out.println(average); // Output: 8.5
```

Declaration/initialization

A variable can be declared and assigned an initial value in the same statement.

Example:
```java
int x = 11 k 3 % 12;
```

Multiple declaration error

The compiler will fail if you try to declare-and-initialize a variable twice.

Example:
```java
int x = 3;
System.out.println(x);
int x = 5; // ERROR: variable x already exists
System.out.println(x);
```

Multiple declarations per line

It is legal to declare multiple variables on one line:
```java
int <name>, ... <name> ;
```

Example:
```java
int a, b, c;
double x, y;
```

It is also legal to declare and initialize several at once:
```java
int <name> = <value> ; ... <name> = <value> ;
```

Example:
```java
int x = 2, b = 3, c = -4;
double grade = 3.5, delta = 0.1;
```

The variables must be of the same type.

Integer or real number?

- Categorize each of the following quantities by whether an int or double variable would best to store it:

<table>
<thead>
<tr>
<th>Integer [int]</th>
<th>Real number [double]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Temperature in degrees Celsius</td>
<td>7. Number of miles traveled</td>
</tr>
<tr>
<td>2. The population of Lemings</td>
<td>8. Number of dry days in the past month</td>
</tr>
<tr>
<td>3. Your grade-point average</td>
<td>9. Your locker number</td>
</tr>
<tr>
<td>4. A person’s age in years</td>
<td>10. Number of seconds left in a game</td>
</tr>
<tr>
<td>5. A person’s weight in pounds</td>
<td>11. The sum of a group of integers</td>
</tr>
<tr>
<td>6. A person’s height in meters</td>
<td>12. The average of a group of integers</td>
</tr>
</tbody>
</table>

Type casting

- **type cast**: A conversion from one type to another.
- **Common uses**:
  - To promote an int into a double to achieve exact division.
  - To truncate a double from a real number to an integer.
- **type cast syntax**:
  ```java
  <type> expression
  ```

  Example:
  ```java
  double result = (double) 13 / 5; // 2.6
  ```

  Example:
  ```java
  int result2 = (int) result; // 3
  ```
More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2; // 1`
  - `double y = 1 + (double) 1 / 2; // 1.5`
- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`

String concatenation

- String concatenation: Using the `+` operator between a String and another value to make a longer String.
  - Examples: (Recall: Precedence of `+` operator is below `/`) `"hello" + 42` is `"hello42"`
  - `1 + "abc" = 2` is `"1abc2"`
  - `1 + 2 + "abc" + 4` is `"1abc24"`

Printing String expressions

- String expressions with `+` are useful so that we can print complicated messages that involve computed values.
  - `double grade = (85.1 + 71.3 + 82.4) / 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.");`

Example variable exercise

- Write a Java program that stores the following data:
  - Section AA has 17 students.
  - Section AB has 8 students.
  - Section AC has 11 students.
  - Section AD has 23 students.
  - Section AE has 24 students.
  - The average number of students per section.

Modify-and-assign operators

- Java has several shortcut operators that allow you to quickly modify a variable's value:
  - Shorthand: `<variable> = <expression>`
  - Equivalent longer version: `<variable > = <variable > + <expression >;

Example:
- `x = 3;
  // x = x + 3;`
- `gpa = 3.5;
  // gpa = gpa - 0.5;`
- `number = 2;
  // number = number * 2;`

Increment and decrement

- The increment and decrement operators increase or decrease a variable's value by 1.
  - Shorthand: `<variable > = <variable > + 1;`
  - Equivalent longer version: `<variable > = <variable > + 1;`

Example:
- `int a = 2;
  // a = a + 1;`
- `double gpa = 2.5;
  // gpa = gpa - 1;`
  // gpa now stores 1.5`
System.out.print command

- Recall: System.out.println prints a line of output and then advances to a new line.

- Another command named System.out.print prints the given output without moving to the next line.
  * This allows you to print partial messages on the same line.

Example:
```java
System.out.println("Kind of like a cloud,");
System.out.print("I was up");
System.out.print("is the sky");
```
Output:
Kind of like a cloud, I was up
May up in the sky

Repetition with for loops

- So far, when we wanted to perform a task multiple times, we have written redundant code:
```java
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("I mean 3-N-A-T");
System.out.println("I mean 3-N-A-T");
```

- Java has a statement called a for loop statement that instructs the computer to perform a task many times.
```
for (int i = 1; i <= 5; i++) {
    System.out.println("I am so smart");
}
System.out.println("I mean 3-N-A-T");
System.out.println("I mean 3-N-A-T");
```

for loop syntax

- for loop: A Java statement that executes a group of statements repeatedly until a given test fails.

  * General syntax:
```java
for <initialization> ; <test> ; <update> {
    <statement>;
    <statement>;
    ...
    <statement>;
}
```

- Example:
```
for (int i = 1; i <= 10; i++) {
    System.out.println("Hi name is Robert Reardon");
}
```

for loop over range of ints

- We'll write for loops over integers in a given range.
  * The loop declares a loop counter variable that is used in the test, update, and body of the loop.
```
for (int <range> = i; <condition> = <value>; <range>++)
```

- Example:
```
for (int i = 1; i <= 6; i++) {
    System.out.println("squared is "+i*i); 
}
```
* Possible interpretation: For each int from 1 through 6, ...
* Output:
```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
```

for loop flow diagram

- Behavior of the for loop:
  * Start out by performing the <initialization> once.
  * Repeatedly execute the <statement[0]> followed by the <update> as long as the <condition> is still true statement.
Loop walkthrough
Let's walk through the following for loop:

```java
for (int i = 1; i <= 3; i++)
    System.out.println(i + " squared is " + (i * i));
```

Output:
1 squared is 1
2 squared is 4
3 squared is 9

Another example for loop
The body of a for loop can contain multiple lines.

Example:
```java
for (int i = 1; i <= 3; i++)
    System.out.println("i = ");
System.out.println("i = ");
System.out.println("i = ");
```

Output:
i = 
i = 
i = 

Some for loop variations
The initial and final values for the loop counter variable can be
arbitrary numbers or expressions:

Example:
```java
for (int i = -3; i <= 2; i++)
    System.out.println(i);
```

Output:
-3 -2 -1 0 1 2

Example:
```java
for (int i = 1; i <= 3 * 4; i <= 100; i++)
    System.out.println(i + " squared is " + (i * i));
```

Downward-counting for loop
The update can also be a -- or other operator, to make
the loop count down instead of up.

This also requires changing the test to say >= instead of <=.
```java
for (int i = 10; i >= 1; i--)
    System.out.println("i = ");
System.out.println("blastoff!");
```

Output:
i = 
... 
i = 1
blastoff!

Single-line for loop
When a for loop only has one statement in its body, the
{ } braces may be omitted:
```java
for (int i = 1; i <= 3; i++)
    System.out.println("i = ");
```

However, this can lead to mistakes where a line appears
to be inside a loop, but is not:
```java
for (int i = 1; i <= 3; i++)
    System.out.println("This is printed 3 times");
System.out.println("This is the... or is it?");
```

Output:
This is printed 3 times
This is the... or is it?

for loop questions
Write a loop that produces the following output:
On day #1 of Christmas, my true love sent to me
On day #2 of Christmas, my true love sent to me
On day #3 of Christmas, my true love sent to me
On day #4 of Christmas, my true love sent to me
...
On day #12 of Christmas, my true love sent to me...
```

Write a loop that produces the following output:
```java
for (int i = 1; i <= 3; i++)
    System.out.println("Who do we appreciate");
```
Mapping loops to numbers

• Suppose that we have the following loop:
  ```java
  for (int count = 1; count <= 5; count++) {
      ...
  }
  ```
• What statement could we write in the body of the loop that would make the loop print the following output?
  ```
  3 6 9 12 15
  ```
• Answer:
  ```java
  for (int count = 1; count <= 5; count++) {
      System.out.print(3 * count + " ");
  }
  ```

Loop number tables

• What statement could we write in the body of the loop that would make the loop print the following output?
  ```
  2 7 12 17 22
  ```
• To find the pattern, it can help to make a table of the count and the number to print.
  • Each time count goes up by 1, the number should go up by 5.
  • But count * 5 is too great by 3, so we must subtract 3.
  ```
<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
</tr>
</tbody>
</table>
  ```

Degenerate loops

• Some loops execute 0 times, because of the nature of their test and update.
  ```java
  // a degenerate loop
  for (int i = 0; i < 5; i++) {
      System.out.println("How many times do I print? ");
  }
  ```
• Some loops execute endlessly (or far too many times), because the loop test never fails.
  ```java
  // a degenerate loop
  for (int i = 0; i < 5; i++) {
      System.out.println("Runaway Java program!!! ");
  }
  ```

Mapping loops to numbers 2

• Now consider another loop of the same style:
  ```java
  for (int count = 1; count <= 5; count++) {
      ...
  }
  ```
• What statement could we write in the body of the loop that would make the loop print the following output?
  ```
  4 7 10 13 16
  ```
• Answer:
  ```java
  for (int count = 1; count <= 5; count++) {
      System.out.print(3 * count + 1 + " ");
  }
  ```

Loop table question

• What statement could we write in the body of the loop that would make the loop print the following output?
  ```
  1 15 31 47 63
  ```
• Let’s create the loop table together.
  • Each time count goes up by 1, the number should ...
  • But this multiple is off by a margin of ...
  ```
<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>63</td>
</tr>
</tbody>
</table>
  ```

Nested loops

• **nested loop:** Loops placed inside one another.
  • The inner loop’s counter variable must have a different name.
  ```java
  for (int i = 1; i <= 3; i++) {
      for (int j = 1; j <= 5; j++) {
          System.out.println("Hi! ");
      }
  }
  ```
• Output:
  ```
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  Hi!
  ```
More nested loops

- In this example, all of the statements in the outer loop’s body are executed 5 times.
- The inner loop prints 10 numbers each of those 5 times, for a total of 50 numbers printed.

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

Output:
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14

Nested for loop exercise

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

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

- Output:
```
  ********
  ********
  ********
  ********
  ********
```

Nested for loop exercise

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

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

- Output:
```
  ********
  ********
  ********
  ********
  ********
```

Nested for loop exercise

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

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

- Output:
```
  ********
  ********
  ********
  ********
  ********
```

Nested for loop exercise

- What nested for loops produce the following output?

```java
1, 1
2, 1
3, 1
1, 2
2, 2
3, 2
```

- Answer:
Nested for loop exercise
• First we write the outer loop, which always goes from 1 to the number of lines desired:
  ```java
  for (int line = 1; line <= n; line++) {
    ...
  }
  ```
• We notice that each line has the following pattern:
  * some number of dots (8 dots on the last line)
  * a number
  ```
  1
  2
  3
  4
  5
  ```
• Answer:
  ```
  1
  4
  1
  ```

Nested for loop exercise
• Next we make a table to represent any necessary patterns on that line:
  ```
<table>
<thead>
<tr>
<th>line # of dots</th>
<th>value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
  ```
• Answer:
  ```
  for (int line = 1; line <= n; line++) {
    ...
    System.out.print(".");
    System.out.println(line);
  }
  ```

Nested for loop exercise
• A for loop can have more than one loop nested in it:
• What is the output of the following nested for loops?
  ```java
  for (int i = 1; i <= 5; i++) {
    System.out.print(" ");
    for (int j = 1; j <= i; j++) {
      System.out.print(i);
    }
  }
  ```
• Answer:
  ```
  1
  1 
  2 2
  3 3 3
  4 4 4 4
  ```

How to comment: for loops
• Place a comment on each loop explaining what they do conceptually, not the mechanics of the syntax.
• Bad:
  ```java
  for (int i = 1; i <= 10; i++) {
    System.out.printf("%d", i);
  }
  ```
• Better:
  ```java
  // Print 10 numbers to the console
  for (int i = 1; i <= 10; i++) {
    System.out.printf("%d", i);
  }
  ```
• Common nested loop bugs
• It is easy to accidentally type the wrong loop variable.
  * What is the output of the following nested loops?
    ```java
    for (int i = 1; i <= 5; i++) {
      System.out.println(i);
    }
    ```
  * What is the output of the following nested loops?
    ```java
    for (int i = 1; i <= 5; i++) {
      System.out.println(i);
    }
    ```
• How to comment: for loops
**Drawing complex figures**

*When the task is as complicated as this one, it may help to write down steps on paper before we write our code:*

1. A pseudo-code description of the algorithm (written in English).
2. A table of each line's contents, to help see the pattern in the input.

---

**A pseudo-code algorithm**

1. Draw top line with `*`, `16` `-`, then `*`.
2. Draw the top half with the following on each line:
   - spaces (decreasing in number as we go downward)
   - `*` decreasing in number as we go downward
   - spaces (same number as above)
   - `*` decreasing in number as we go downward
3. Draw the bottom half, which is the same as the top half but upside-down.
4. Draw bottom line with `*`, `16` `-`, then `*`.

*Our pseudo-code suggests that we should write a table to learn the pattern in the top and bottom halves of the figure.*

---

**Pseudo-code**

*pseudo-code: A written English description of an algorithm to solve a programming problem.*

*Example: Suppose we are trying to draw a box of stars on the screen which is 12 characters wide and 7 tall.*

*A possible pseudo-code for this algorithm:*

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

---

**Tables to examine output**

*A table of the contents of the lines in the "top half" of the figure:*

<table>
<thead>
<tr>
<th>Line</th>
<th>Number</th>
<th>Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3 - 0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4 + 4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4 + 4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
Implementing the figure
• Let’s implement the code for this figure together.
• Some questions we should ask ourselves:
  * How many loops do we need on each line of the top half of the output?
  * Which loops are nested inside which other loops?
  * How should we use static methods to represent the structure and redundancy of the output?

Partial solution
// Print the expanding patterns of O for the top half of the figure:
public static void DrawFigO() {
    for (int i = 0; i < r; i++) {
        System.out.print("\n");
        System.out.print("* ");
        for (int j = 0; j < i; j++) {
            System.out.print(" ");
        }
        System.out.print("*");
        for (int j = 0; j < r - i; j++) {
            System.out.print(" ");
        }
        System.out.print("*");
        for (int j = 0; j < r - i; j++) {
            System.out.print(" ");
        }
        System.out.print("*");
        System.out.println();
    }
    System.out.println("**************");
}

Variable scope
• scope: The part of a program where a variable exists.
  * A variable’s scope is from its declaration to the end of the { } blocks in which it was declared.
  * If a variable is declared in a for loop, it exists only in that loop.
  * If a variable is declared in a method, it exists in that method.

```java
public static void example1() {
    int x = 5;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    System.out.println("x’s scope");
    // x no longer exists here
    // x ceases to exist here
}
```

Scope and using variables
• It is illegal to try to use a variable outside of its scope.

```java
public static void main(String[] args) {
    example0();
    System.out.println("* ");
    for (int i = 1; i <= 10; i++) {
        System.out.println("*");
    }
    System.out.println("*");
    System.out.println("*");
    // illegal
    public static void example1() {
        int x = 3;
        System.out.println(x);
    }
    System.out.println("*");
    System.out.println("*");
    System.out.println("*");
}
```

Overlapping scope
• It is legal to declare variables with the same name, as long as their scopes do not overlap.

```java
public static void main(String[] args) {
    int x = 5;
    for (int i = 1; i <= 5; i++) {
        System.out.println(y);
    }
    System.out.println("*");
    System.out.println("*");
    System.out.println("*");
    public static void main(String[] args) {
        int x = 10;
        System.out.println(x);
    }
    System.out.println("*");
    System.out.println("*");
    System.out.println("*");
}
```
**Problem: redundant values**

- **magic number**: A value used throughout the program.
  - Magic numbers are bad; what if we have to change them?
  - A normal variable cannot be used to fix the magic number problem, because its scope is not large enough.

```java
public static void main(String[] args) {
  int max = 5;
  int magic = 10;

  for (int i = 0; i < max; i++) {
    System.out.println(i * magic);
  }
}
```

**Class constants**

- **class constant**: A variable that can be seen throughout the program.
  - The value of a constant can only be set when it is declared. It cannot be changed while the program is running.

```java
public static final int NUM_TAX = 10;
```

**Class constant example**

- Making the 3 a class constant removes the redundancy:

```java
public class Example {
  public static final int MAX_TAX = 10;
  public static void main(String[] args) {
    System.out.println(MAX_TAX);
    for (int i = 0; i < MAX_TAX; i++) {
      System.out.println(i);
    }
  }
}
```

**Congratulations and figures**

- Consider the task of drawing the following figures:

```java
// Each figure is strongly tied to the number 5
// (or a multiple of 5, such as 10, 20, 30, ...)

// Use a class constant so that these figures will be reusable.
```

**Repetitive figure code**

- Note the repetition of numbers based on 5 in the code:

```java
public static void printStarPattern(int n) {
  for (int i = 0; i < n; i++) {
    System.out.print("*");
  }
  System.out.println();
}
```

**Fixing our code with constant**

- A class constant will fix the "magic number" problem:

```java
public static final int FIGURE_SIZE = 5;
```

```java
public static void printStarPattern() {
  System.out.println("*");
}
```

```java
public static void printStarPattern(int n) {
  for (int i = 0; i < FIGURE_SIZE; i++) {
    System.out.println("*");
  }
}
```

```java
public static void printStarPattern(int n, int repeat) {
  for (int i = 0; i < FIGURE_SIZE * repeat; i++) {
    System.out.println("*");
  }
}
```

```java
// Fig1.java
```