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. Write an English description of steps required (*pseudo-code*)
     - use pseudo-code to decide methods

  2. Create a table of patterns of characters
     - use table to write loops in each method

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

1. Pseudo-code

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

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

```plaintext
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

```java
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

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>line * -2 + 8</th>
<th>dots</th>
<th>4 * line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

#================#

|      <><>      |
|    <>....<>    |
|  <>........<>  |
| <>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |

#================#

3. Writing the code

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

```java
public static void topHalf() {
    for (int line = 1; line <= 4; line++) {
        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

**reading:** 2.4

self-check: 28

exercises: 11

videos: Ch. 2 #5
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.

```java
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 = max; i >= 1; i--) {
        // ERROR: size not found
    }
}
```
Variable scope

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

```java
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
}
```

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("/");
}
int i = 4;                      // ERROR: outside scope
```
Class constants

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

**Syntax:**
```
public static final type name = value;
```
- name is usually in ALL_UPPER_CASE

**Examples:**
```java
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 |
|                           |
+---------------------------+
Repertitive 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

```java
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("|");
        }
    }
}
```
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:
```java
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 to use SIZE
- This can change the $b$ in $y = mx + b$

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>$-2\text{line} + (2\text{SIZE})$</th>
<th>dots</th>
<th>$4\text{line} - 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>$-2\text{line} + 8$</td>
<td>0,4,8,12</td>
<td>$4\text{line} - 4$</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>$-2\text{line} + 6$</td>
<td>0,4,8</td>
<td>$4\text{line} - 4$</td>
</tr>
</tbody>
</table>

Partial solution

```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.

```java
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.

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