“A fine balance must be maintained between computation time and the ensuing complexity of the coding.”

Rear Admiral Grace Hopper, Ph.D. and Howard Aiken, Ph.D.

“A Manual of Operation for the Automatic Sequence Controlled Calculator”

Based on slides by Marty Stepp and Stuart Reges from http://www.buildingjavaprograms.com/
Drawing complex figures

- Use nested for loops to produce the following output.
1. Design 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. What are the patterns?

- Top half:
  - Line 4: 0 spaces, <>, 12 dots, <>
2. What are the patterns?

- Top half:
  - Line 4: 0 spaces, <> , 12 dots, <>
  - Line 3: 2 spaces, <> , 8 dots, <>

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

```
Line 1: |  <><>  |
Line 2: | <>....<>|
Line 3: | <>........<>|
Line 4: | <>............<>|
|<>............<>|
|<>............<>|
|<>............<>|
|<>....<>|
|<>><>
#=================================
```
2. What are the patterns?

- **Top half:**
  - Line 4: 0 spaces, $<>$, 12 dots, $<>$
  - Line 3: 2 spaces, $<>$, 8 dots, $<>$
  - Line 2: 4 spaces, $<>$, 4 dots, $<>$

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

```
Line 1: |    <><>   |
Line 2: |   <>.....<> |
Line 3: |  <>........<> |
Line 4: |<>............<>|
        |<>............<>|
        |<>............<>|
        |<>....<>|
        |<>><>|
```

```text
#=================================
```
2. What are the patterns?

- Top half:
  - Line 4: 0 spaces, <>, 12 dots, <>
  - Line 3: 2 spaces, <>, 8 dots, <>
  - Line 2: 4 spaces, <>, 4 dots, <>
  - Line 1: 6 spaces, <>, 0 dots, <>
2. What are the patterns?

- Top half:
  - Line 4: 0 spaces, <>, 12 dots, <>
  - Line 3: 2 spaces, <>, 8 dots, <>
  - Line 2: 4 spaces, <>, 4 dots, <>
  - Line 1: 6 spaces, <>, 0 dots, <>

So,
line = 1 → dots = 0
line = 2 → dots = 4
line = 3 → dots = 8
line = 4 → dots = 12
2. What are the patterns?

Every time we add 1 to line, we add 4 to dots.

<table>
<thead>
<tr>
<th>line</th>
<th>dots</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>
2. What are the patterns?

Every time we add 1 to line, we add 4 to dots.

So, dots = 4*line + something.

- line = 1 → dots = 0
- line = 2 → dots = 4
- line = 3 → dots = 8
- line = 4 → dots = 12
2. What are the patterns?

Every time we add 1 to line, we add 4 to dots.

So, dots = 4*line + something.

What is something?

Plug in 1 for line, 0 for dots.

0 = 4 * 1 + something
So, something = -4.
2. What are the patterns?

Every time we add 1 to line, we add 4 to dots.

So, dots = 4*line + something.

What is something?

Plug in 1 for line, 0 for dots.

0 = 4 * 1 + something
So, something = -4.

So, dots = 4 * line - 4.
2. What are the patterns?

- Top half:
  - Line 4: 0 spaces, <>, 12 dots, <>
  - Line 3: 2 spaces, <>, 8 dots, <>
  - Line 2: 4 spaces, <>, 4 dots, <>
  - Line 1: 6 spaces, <>, 0 dots, <>

So,

line = 1 → spaces = 6
line = 2 → spaces = 4
line = 3 → spaces = 2
line = 4 → spaces = 0

#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
2. What are the patterns?

Every time we add 1 to line, we subtract 2 from spaces.

So, spaces = \(-2\)\(\cdot\) line + something.

What is something?

Plug in 1 for line, 6 for spaces.

\[ 6 = -2 \cdot 1 + \text{something} \]

So, something = 8.

So, spaces = \(-2\)\(\cdot\) line + 8.
// 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

reading: 2.4
Scaling the mirror

- Modify the 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.
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
```
Scope implications

- Variables whose scope does NOT overlap 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

Examples:

```java
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 0.5;
public static final int SSN = 658234569;
```

Not okay:

```java
public static final int DAYS_IN_WEEK;
DAYS_IN_WEEK = 7;

public static final int DAYS_IN_WEEK = 7;
DAYS_IN_WEEK = 9;
```
Modify the Mirror code to be resizable using a constant.

<table>
<thead>
<tr>
<th>A mirror of size 4:</th>
<th>A mirror of size 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>#================#</td>
<td>#================#</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;&lt;&gt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;....&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;........&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;............&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;............&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;........&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;&lt;&gt;&gt;</td>
</tr>
<tr>
<td>#================#</td>
<td>#================#</td>
</tr>
</tbody>
</table>
What’s the pattern now?

Top half: (3 lines, not 4)

line = 1 → 4 spaces, <>, 0 dots, <>

line = 2 → 2 spaces, <>, 4 dots, <>

line = 3 → 0 spaces, <>, 8 dots, <>

spaces = -2*line + 6

dots = 4*line - 4

A mirror of size 3:

#============#
|    <><>    |
|  <>....<>  |
|<>........<>|
|<>........<>|
|  <>....<>  |
|    <><>    |
#============#
Complex figure w/ constant

So, when mirror SIZE is 3:
spaces = -2*line + 6
dots = 4*line – 4

When mirror SIZE is 4:
spaces = -2*line + 8
dots = 4*line – 4

A mirror of size 3:
#==============#
|   <><>    |
| <>....<>  |
|<>........<>|
|<>........<>|
| <>....<>  |
|    <><>    |
#==============#
So, when mirror SIZE is 3:
spaces = \(-2\times \text{line} + 6\)
dots = \(4\times \text{line} - 4\)

When mirror SIZE is 4:
spaces = \(-2\times \text{line} + 8\)
dots = \(4\times \text{line} - 4\)

Do same approach for:
SIZE = 3 → 6
SIZE = 4 → 8

So, number is 2 * SIZE.

A mirror of size 3:
#=============#
|    <><>    |
|  <>....<>  |
|<>........<>|
|<>........<>|
|  <>....<>  |
|    <><>    |
#=============#

A mirror of size 4:
#=============#
|    <><>    |
|  <>....<>  |
|<>........<>|
|<>........<>|
|  <>....<>  |
|    <><>    |
|<> . . . . <>|
|<> . . . . . <>|
|<> . . . . . <>|
|<> . . . . . <>|
|  <>....<>  |
|    <><>    |
#=============#
Complex figure w/ constant

So, number is $2 \times \text{SIZE}$.

So for any mirror with size $\text{SIZE}$:
spaces $= -2\times\text{line} + 2 \times \text{SIZE}$
dots $= 4\times\text{line} - 4$
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("|");
    }
}

Another Example

- Create a program to produce the following ASCII art rocket
- SIZE = 3 for the rocket to the right
SIZE = 4 Rocket
SIZE = 5 Rocket
Assignment 2: ASCII Art