Topic 6
loops, figures, constants

"Complexity has and will maintain a strong fascination for many people. It is true that we live in a complex world and strive to solve inherently complex problems, which often do require complex mechanisms. However, this should not diminish our desire for elegant solutions, which convince by their clarity and effectiveness. Simple, elegant solutions are more effective, but they are harder to find than complex ones, and they require more time, which we too often believe to be unaffordable."

-Niklaus Wirth

clicker question

• What is the base 10 equivalent of the base 2 number 1011001
  A. 27
  B. 89
  C. 93
  D. 127
  E. 1011001

clicker question

• What does 5! equal?
  A. 5  
  B. 10
  C. 32
  D. 120
  E. a lot

clicker question

• Which of the following is closest to the value that overflows the Java int data type when calculating N!?
  A. 1
  B. 15
  C. 60
  D. 100
  E. 1000
Drawing complex figures

- Use nested for loops to produce the following output.
- Why draw ASCII art?
  - Real graphics require more finesse
  - ASCII art has complex patterns
  - Can focus on the algorithms

1. Syntax
def draw_box(width, height):
    for line in range(height):
        if line == 0 or line == height - 1:
            print('*' * width)
        else:
            print('* ' * (width - 2) + '*

2. 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.
```

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 for patterns of characters
     - use tables to write your for loops

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 <= 3; line++) {
            // contents of each line
        }
    }

    public static void bottomHalf() {
        for (int line = 1; line <= 3; 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>-2 * line + 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?

```
// Prints the expanding pattern of <> for the top half of the figure.
public static void topHalf() {
    for (int line = 1; line <= 3; line++) {
        System.out.print("\n");
        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("\n");
    }
}
```

Partial solution
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.

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 = 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.

```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("\"\n");
} // OK: outside of loop's scope
```

- A variable cannot 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("/");
} // 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, hence the name: *constant*

- **Syntax**: public static final `<type> <name> = <exp>;`
  - name in ALL_UPPER_CASE by convention

  - Examples:
    - public static final int DAYS_IN_WEEK = 7;
    - public static final double INTEREST_RATE = 0.5;
    - 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

```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("/\"\n");
        }
        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("|");
        }
    }
}
```

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:

```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 table to use \texttt{SIZE}
- This can change the amount added in the loop expression

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>-2<em>line + (2</em>SIZE)</th>
<th>dots</th>
<th>4*line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>-2*line + 8</td>
<td>0,4,8,12</td>
<td>4*line - 4</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>-2*line + 6</td>
<td>0,4,8</td>
<td>4*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 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 \textit{every} occurrence of the original value.
    ```java
    for (int dot = 1; dot <= (line * 4 - 4); dot++) {
        System.out.print(".");
    }
    ```

Assignment 2: ASCII Art

![Image of a building](image-url)