Topic 6
Nested for Loops

"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

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

Nested `for` loops

- A `for` loop can contain any kind of statement in its body, including another `for` loop.
  - The inner loop must have a different name for its loop counter variable so that it will not conflict with the outer loop.
- **nested loop**: Loops placed inside one another, creating a loop of loops.

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 2; j++) {
        System.out.println("six");
    }
}
```

Output:

```
six
six
six
six
six
six
```

More nested `for` loops

- All of the statements in the outer loop's body are executed 5 times.
  - The inner loop runs 10 times for each of those 5 times, for a total of 50 numbers printed.

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

Output:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
```

Nested `for` loop exercise

- What is the output of the following nested `for` loop?

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Output?
- Do you really need a nested for loop in this case?

Nested `for` loop exercise

- What is the output of the following nested `for` loop?

```
for (int i = 1; i <= 6; 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` loop?

```
for (int i = 1; i <= 6; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

- Output?

Nested `for` loop exercise

- Create a nested `for` loops produce the following output.

```
....1
...22
..333
.4444
55555
```

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?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= (5 - i); j++) {
        System.out.print(" ");
    }
    for (int k = 1; k <= i; k++) {
        System.out.print(i);
    }
    System.out.println();
}
```

- Answer:

```
    1
   22
  333
 4444
55555
```

Common nested loop bugs

- It is a common bug to accidentally type the wrong loop counter variable, which can lead to incorrect behavior.
  - What is the output of the following nested loops?

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; i <= 5; j++) {
        System.out.print(j);
    }
    System.out.println();
}
```

  - What is the output of the following nested loops?

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 5; i++) {
        System.out.print(j);
    }
    System.out.println();
}
```

How to comment: for loops

- Place a comment on complex loops explaining *what* they do from a conceptual standpoint, not the mechanics of the syntax.
  - Bad:

```
// This loop repeats 10 times, with i from 1 to 10.
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 5; j++) {  // loop goes 5 times
        System.out.print(j);  // print the j
    }
    System.out.println();
}
```

  - Better:

```
// Prints 12345 ten times on ten separate lines.
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.print(j);
    }
    System.out.println();  // end the line of output
}
```

Variable Scope

Variable scope

- **scope**: The portion of a program where a given variable exists.
  - A variable's scope is from its declaration to the end of the block (pair of `{ }` braces) in which it was declared.
    - If a variable is declared in a for loop (including the ***<initialization>*** or the loop's ***<statement(s)>***, it exists until the end of that for loop.
    - If a variable is declared in a method, it exists only in that method, until the end of the method.

```
public static void main(String[] args) {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.print(x);
    }
    // i no longer exists here
}  // x ceases to exist here
```

Scope and using variables

- It is a syntax error to try to use a variable outside of its scope.

```
public static void main(String[] args) {
    example();
    System.out.println(x);  // syntax error

    for (int i = 1; i <= 10; i++) {
        int y = 5;
        System.out.println(y);
    }
    System.out.println(y);  // syntax error
}

public static void example() {
    int x = 3;
    System.out.println(x);
}
```

Overlapping scope

- It is syntactically legal to declare variables with the same name, as long as their scopes do not overlap:

```
public static void main(String[] args) {
    int x = 2;

    for (int i = 1; i <= 5; i++) {
        int y = 5;
        System.out.println(y);
    }
    for (int i = 3; i <= 5; i++) {
        int y = 2;
        int x = 4;  // illegal
        System.out.println(y);
    }
}

public static void anotherMethod() {
    int i = 6;
    int y = 3;
    System.out.println(i + ", " + y);
}
```

Problem: redundant values

- Often in our programs we will have certain values (sometimes called *magic numbers*) that are used throughout the program:

```
public static void main(String[] args) {
    printTop();
    printBottom();
}

public static void printTop() {
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}

public static void printBottom() {
    for (int i = 3; i >= 1; i--) {
        for (int j = i; j >= 1; j--) {
            System.out.print(3);
        }
        System.out.println();
    }
}
```

Global constants

- **global (class) constant**: A special kind of variable that can be seen throughout the program.
  - The value of a constant can only be set once.
    It can not be changed while the program is running.

- Global constant syntax:

```
public static final <type> <name> = <value> ;
```

  - Constants' names are usually written 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;
```

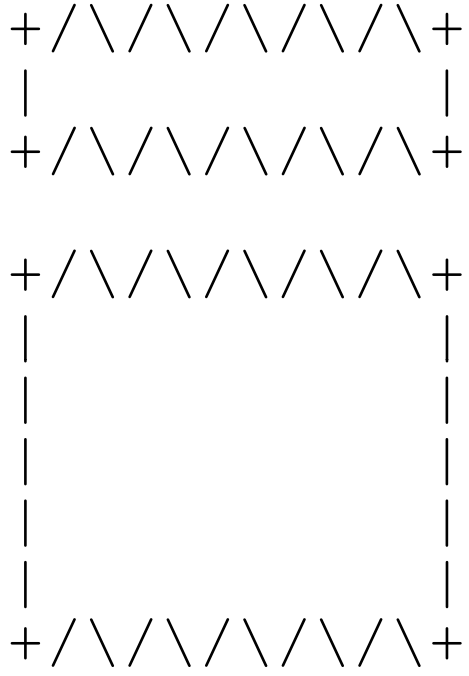
Global constant example

- Making the 3 a global constant removes the redundancy:
  - the global constant is declared outside methods, but inside the program,

```
public class PrintFigure{
   public static final int MAX_VALUE = 3;

   public static void main(String[] args) {
       printTop();
       printBottom();
   }

   public static void printTop() {
       for (int i = 1; i <= MAX_VALUE; i++) {
           for (int j = 1; j <= i; j++) {
               System.out.print(j);
           }
           System.out.println();
       }
   }

   public static void printBottom() {
       for (int i = MAX_VALUE; i >= 1; i--) {
           for (int j = i; j >= 1; j--) {
               System.out.print(MAX_VALUE);
           }
           System.out.println();
       }
   }
}
```

Another example: figure

- Consider the task of drawing the following figure:



- Each figure is strongly tied to the number 5 (or a multiple such as 10 ...)
- See code for drawing these figures.

Some repetitive code

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

```
public static void drawFigure1() {
    drawPlusLine();
    drawBarLine();
    drawPlusLine();
}

public static void drawPlusLine() {
    System.out.print("+");
    for (int i = 1; i <= 5; i++) {
        System.out.print("/\\");
    }
    System.out.println("+");
}

public static void drawBarLine() {
    System.out.print("|");
    for (int i = 1; i <= 10; i++) {
        System.out.print(" ");
    }
    System.out.println("|");
}
```

Output:

```
+/\/\/\/\/\+
|           |
+/\/\/\/\/\+
```

  - It would be cumbersome to resize the figure such as by changing all the 5s into 8s.

A failed attempt to fix it

- A normal variable cannot be used to fix the problem, because it is out of scope:

```
public static void main(String[] args) {
    int width = 5;
    drawFigure1();
}

public static void drawFigure1() {
    drawPlusLine();
    drawBarLine();
    drawPlusLine();
}

public static void drawPlusLine() {
    System.out.print("+");
    for (int i = 1; i <= width; i++) {    // ERROR
        System.out.print("/\\");
    }
    System.out.println("+");
}
```

Fixing our code with constant

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

```
public static final int FIGURE_WIDTH = 5;

public static void main(String[] args) {
    drawFigure1();
}

public static void drawFigure1() {
    drawPlusLine();

    System.out.print("|");
    for (int i = 1; i <= 2 * FIGURE_WIDTH; i++) {
        System.out.print(" ");
    }
    System.out.println("|");

    drawPlusLine();
}
```