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

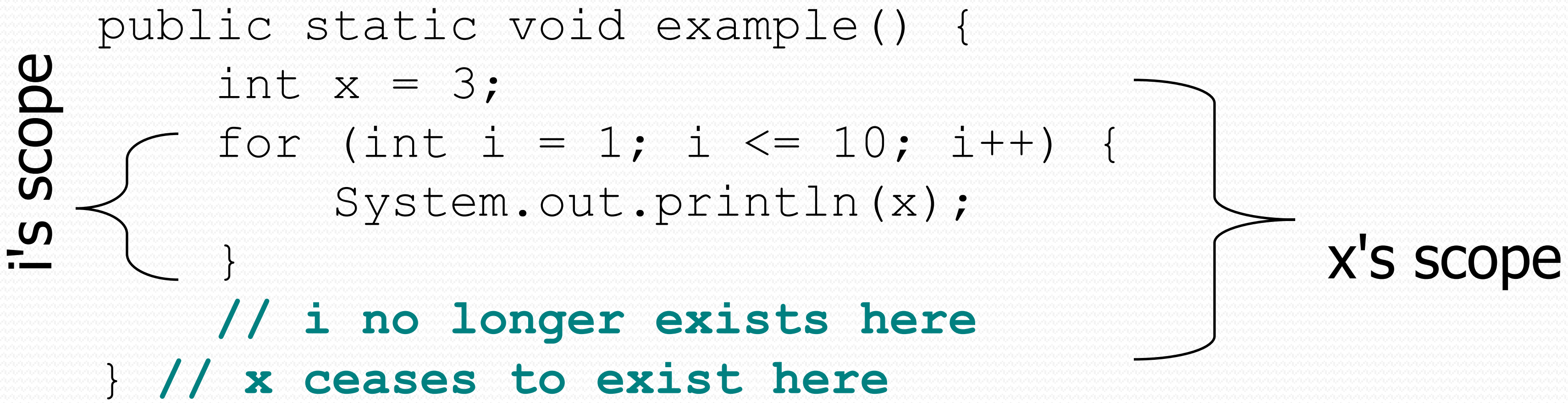
- Idea: Make a variable to represent the size.
    - Use the variable's value in the methods.

- Problem: A variable in one method can't be seen in others.

```
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
        ...
    }
}
```

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.



```
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
```

i's scope

x's scope

Scope implications

- Variables without overlapping scope can have same name.

```
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.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                     // ERROR: overlapping scope
    System.out.print("/");
}
i = 4;                             // ERROR: outside scope
```

Class constants

- **class constant**: A value visible to the whole class.
  - 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:

```
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

Repetitive figure code

```
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

```
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:

```
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 SIZE

SIZE	line	spaces	-2*line + (2*SIZE)	dots	4*line - 4
4	1,2,3,4	6,4,2,0	-2*line + **8**	0,4,8,12	4*line - 4
3	1,2,3	4,2,0	-2*line + **6**	0,4,8	4*line - 4

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

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

Partial solution

```
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

- It doesn't replace *every* occurrence of the original value.
  - "Different fours" for different reasons
  - A good reason to use variables to keep things straight

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

- Even if you're not interested in scaling, constants can make algorithms clearer
  - Avoids "magic numbers"

Building Java Programs

Chapter 3

Lecture 3-1: Parameters

reading: 3.1

self-check: #1-6

exercises: #1-3

videos: Ch. 3 #1, 4

Redundant recipes

- Recipe for baking **20** cookies:
  - Mix the following ingredients in a bowl:
    - **4** cups flour
    - **1** cup butter
    - **1** cup sugar
    - **2** eggs
    - **1** bag chocolate chips ...
  - Place on sheet and bake for about 10 minutes.

- Recipe for baking **40** cookies:
  - Mix the following ingredients in a bowl:
    - **8** cups flour
    - **2** cups butter
    - **2** cups sugar
    - **4** eggs
    - **2** bags chocolate chips ...
  - Place on sheet and bake for about 10 minutes.

Parameterized recipe

- Recipe for baking **20** cookies:
  - Mix the following ingredients in a bowl:
    - **4** cups flour
    - **1** cup sugar
    - **2** eggs
    - **...**
- Recipe for baking **N** cookies:
  - Mix the following ingredients in a bowl:
    - **N/5** cups flour
    - **N/20** cups butter
    - **N/20** cups sugar
    - **N/10** eggs
    - **N/20** bags chocolate chips ...
  - Place on sheet and bake for about 10 minutes.
- **parameter**: A variable that distinguishes similar tasks.

Redundant figures

- Consider the task of printing the following lines/boxes:

```
*************

*******

***********************************

**********
*        *
**********

*****
*   *
*   *
*****
```

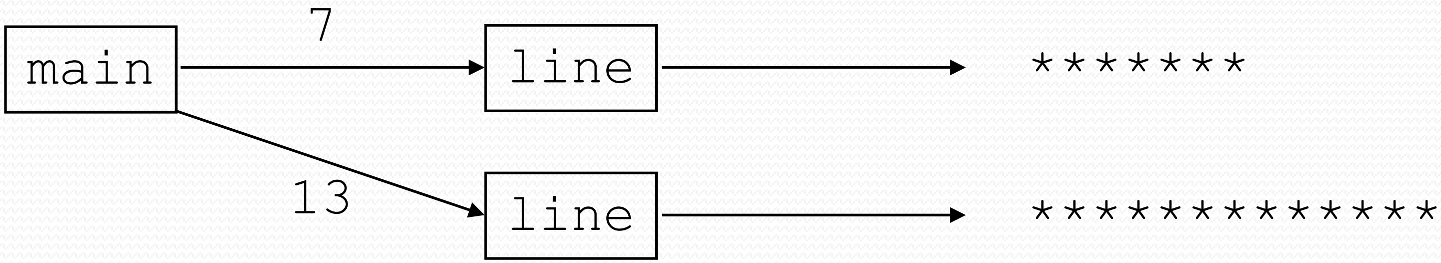
A redundant solution

```
public class Stars1 {
    public static void main(String[] args) {
        lineOf13();
        lineOf7();
        lineOf35();
        box10x3();
        box5x4();
    }

    public static void lineOf13() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf7() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf35() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

- This code is redundant.
- Would variables help?
  Would constants help?
- What is a better solution?
  - `line` - A method to draw a line of any number of stars.
  - `box` - A method to draw a box of any size.

Parameterization

- **parameter**: Something passed to a method by its caller
  - Instead of `lineOf7`, `lineOf13`, write `line` to draw any length.
    - When *declaring* the method, we will state that it requires a parameter for the number of stars.
    - When *calling* the method, we will specify how many stars to draw.



- A parameter is a variable with a slight twist:
  - Declared by a method; in scope for entire method
  - *Initialized by each call to the method*

Declaring a parameter

Stating that a method requires a parameter in order to run

```
public static void name ( type name ) {
    statement(s);
}
```

- Example:

  ```
  public static void sayPassword(int code) {
      System.out.println("The password is: " + code);
  }
  ```

  - When `sayPassword` is called, the caller must specify the integer code to print (i.e., initialize the parameter variable).

Passing parameters

Calling a method and specifying values for its parameters

name (**expression**);

This does the initialization; there is no = involved

- Example:

```
public static void main(String[] args) {
    sayPassword(42);
    sayPassword(12345);
}
```

Output:

```
The password is 42
The password is 12345
```

Parameters and loops

- A parameter can guide the number of repetitions of a loop.

```
public static void main(String[] args) {
    chant(3);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```

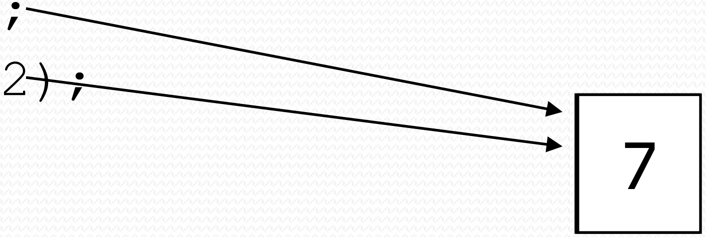
Output:

```
Just a salad...
Just a salad...
Just a salad...
```

How parameters are passed

- When the method is called:
  - The value is stored into the parameter variable.
  - The method's code executes using that value.

```
public static void main(String[] args) {
    int x = 5;
    chant(3);
    chant(x+2);
}
```



```
public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```

Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
chant();          // ERROR: parameter value required
```

- The value passed to a method must be of the correct type.

```
chant(3.7);       // ERROR: must be of type int
```

- Exercise: Change the `Stars` program to use a parameterized method for drawing lines of stars.

Stars solution

```
// Prints several lines of stars.
// Uses a parameterized method to remove redundancy.
public class Stars2 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

Back to our mirror

- Our mirror program had “offensive redundancy”
  - Repeated code in topHalf() and bottomHalf()
  - Longer program and multiple places to “fix the same bug”
- What we want is a method to print a mirror line
  - a |, some spaces, a <>, some dots, a <>, some spaces, a |
  - But how many spaces and dots depends on what line
  - The line number can be the parameter!
    - No other good way to do it, which is why we copied last lecture

```
public static void topHalf() {
    for (int i = 1; i <= SIZE; i++) {
        mirrorLine(i);
    }
}
public static void mirrorLine(int line) { … }
```

Multiple parameters

- A method can accept multiple parameters. (separate by ,)
  - When calling it, you must pass values for each parameter.

- Declaration:

```
public static void name (type name, ..., type name) {
    statement(s);
}
```

- Call:

```
methodName (value, value, ..., value);
```

Multiple parameters example

```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717

00000000
```

Stars with a box method

```
// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

Stars solution, cont'd.

```
    ...

    // Prints a box of stars of the given size.
    public static void box(int width, int height) {
        line(width);
        for (int line = 1; line <= height - 2; line++) {
            System.out.print("*");
            for (int space = 1; space <= width - 2; space++) {
                System.out.print(" ");
            }
            System.out.println("*");
        }
        line(width);
    }
}
```

Value semantics

- Modifying the parameter will not affect the caller’s variables, even those used to initialize the parameter.
  - Just like with other variables

```
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}

public static void main(String[] args) {
    int x = 23; // a “totally different x variable”
    strange(x);
    System.out.println("2. x = " + x);
    ...
}
```

Output:

```
1. x = 24
2. x = 23
```

A "Parameter Mystery" problem

```
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 5;
        int y = 9;
        int z = 2;

        mystery(z, y, x);

        mystery(y, x, z);
    }

    public static void mystery(int x, int z, int y) {
        System.out.println(z + " " + y + " " + x);
    }
}
```

Strings

- **string**: A sequence of text characters.

```
String name = "text";
String name = expression;
```

  - Examples:

```
String name = "Marla Singer";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        String teacher1 = "Dan";
        sayHello(teacher1);
        sayHello("Alan");
        sayHello(teacher1 + " and " + "Alan");
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

Output:

```
Welcome, Dan
Welcome, Alan
Welcome, Dan and Alan
```

A Better Stars solution

```
// Prints several lines and boxes made of stars.
// Fourth version with String parameters.

public class Stars4 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        repeat("*", count);
        System.out.println();
    }

    ...
```

Stars solution, cont'd.

```
    ...

    // Prints a box of stars of the given size.
    public static void box(int width, int height) {
        line(width);

        for (int line = 1; line <= height - 2; line++) {
            System.out.print("*");
            repeat(" ", width - 2);
            System.out.println("*");
        }

        line(width);
    }

    // Prints the given String the given number of times.
    public static void repeat(String s, int times) {
        for (int i = 1; i <= times; i++) {
            System.out.print(s);
        }
    }
}
```