Primitive data, expressions, and variables

Readings: 2.1 – 2.2

How the computer sees the world

- Internally, the computer stores everything in terms of 1's and 0's
  - Example:

```
h    → 0110100
"hi" → 01101000110101
104  → 0110100
```

- How can the computer tell the difference between an h and `104`?

Data types

- **type**: A category of data values.
  - Example: integer, real number, string
- Data types are divided into two classes:
  - **primitive types**: Java's built-in *simple* data types for numbers, text characters, and logic.
  - **object types**: Coming soon!

Primitive types

- Java has eight primitive types. We will cover two for now.

Name	Description	Examples
`int`	integers	`42`, `-3`, `0`, `926394`
`double`	real numbers	`3.4`, `-2.53`, `91.4e3`

- Numbers with a decimal point are treated as real numbers.
- Question: Isn't every integer a real number? Why bother?

Integer or real number?

- Which category is more appropriate?

integer (`int`)	real number (`double`)

1. Temperature in degrees Celsius
2. The population of lemmings
3. Your grade point average
4. A person's age in years
5. A person's weight in pounds
6. A person's height in meters
7. Number of miles traveled
8. Number of dry days in the past month
9. Your locker number
10. Number of seconds left in a game
11. The sum of a group of integers
12. The average of a group of integers

- credit: Kate Deibel, http://www.cs.washington.edu/homes/deibel/CATs/

Manipulating data via expressions

- **expression**: A data value or a set of operations that produces a value.
  - Examples:

```
1 + 4 * 3
3
"CSE142"
(1 + 2) % 3 * 4
```

The operators

- Arithmetic operators we will use:

`+`	addition
`-`	subtraction or negation
`*`	multiplication
`/`	division
`%`	modulus, a.k.a. remainder

Evaluating expressions

- When Java executes a program and encounters an expression, the expression is *evaluated* (i.e., computed).
  - Example: `3 * 4` *evaluates* to 12
- `System.out.println(3 * 4)` *prints* 12 (after evaluating `3 * 4`)
  - How could we print the text `3 * 4` on the console?

Evaluating expressions: Integer division

- When dividing integers, the result is also an integer.
  - Example: `14 / 4` evaluates to 3, not `3.5` (truncate the number)

```
     3               52
4 ) 14         27 ) 1425
    12              135
     2               75
                     54
                     21
```

  - Examples:
    - `1425 / 27` is `52`
    - `35 / 5` is `7`
    - `84 / 10` is `8`
    - `156 / 100` is `1`
    - `24 / 0` is illegal

Evaluating expressions: The modulus (%)

- The modulus computes the remainder from a division of integers.
  - Example: `14 % 4` is 2
    `1425 % 27` is 21

```
     3               52
4 ) 14         27 ) 1425
    12              135
     2               75
                     54
                     21
```

- What are the results of the following expressions?

```
45 % 6
2 % 2
8 % 20
11 % 0
```


Applying the modulus

- What expression obtains…
  - the last digit (unit's place) of a number?
    - Example: From `230857`, obtain the `7`.
  - the last 4 digits of a Social Security Number?
    - Example: From `658236489`, obtain `6489`.
  - the second-to-last digit (ten's place) of a number?
    - Example: From `7342`, obtain the `4`.

Applying the modulus

- How can we use the `%` operator to determine whether a number is odd?
- How about if a number is divisible by, say, 27?

Precedence: Remember PEMDAS?

- **precedence**: Order in which operations are computed in an expression.
  - Operators on the same level are evaluated from left to right.
    Example: `1 - 2 + 3` is `2` (not `-4`)
  - Spacing does not affect order of evaluation.
    Example: `1+3 * 4-2` is `11`

Parentheses	`()`
Multiplication, Division, Mod	`*  /  %`
Addition, Subtraction	`+  -`

Precedence examples

```
1 * 2 + 3 * 5 / 4
 \_/
  |
  2   + 3 * 5 / 4
         \_/
          |
  2   +  15   / 4
          \___/
            |
  2   +     3
   \________/
        |
        5
```

```
1 + 2 / 3 * 5 - 4
     \_/
      |
1 +   0   * 5 - 4
       \___/
          |
1 +       0   - 4
 \________/
     |
     1        - 4
      \_________/
           |
          -3
```


Precedence exercise

- Evaluate the following expressions:
  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`
- Which parentheses are unnecessary?

Real numbers (`double`)

- The operators also work with real numbers.
  - The division operator produces an exact answer.
- Examples:
  `15.0 / 2.0` is `7.5`
  `15.3 + 2.5` is `17.8`
  `1.23 + 15.0 * 2.0` is `31.23`
- The same precedence rules apply.

Real numbers example

```
2.0 * 2.4 + 2.25 * 4.0 / 2.0
  \___/
    |
   4.8    + 2.25 * 4.0 / 2.0
              \___/
                |
   4.8    +    9.0    / 2.0
                 \_____/
                    |
   4.8    +        4.5
     \____________/
            |
           9.3
```


Precision in real numbers

- The computer internally represents real numbers in an imprecise way.
- Example:
  `System.out.println(0.1 + 0.2);`
  - The output is `0.30000000000000004`!

Mixing integers and real numbers

- When an operator is used on an integer and a real number, the result is a real number.
  - Examples:
    `4.2 * 3` is `12.6`
    `1 / 2.0` is `0.5`
- The conversion occurs on a *per-operator* basis. It affects only its two operands.

```
7 / 3 * 1.2 + 3 / 2
 \_/
  |
  2   * 1.2 + 3 / 2
   \___/
     |
    2.4     + 3 / 2
               \_/
                |
    2.4     +   1
      \_________/
           |
          3.4
```

- Notice how `3 / 2` is still `1` above, not `1.5`.

Mixed types example

```
2.0 + 10 / 3 * 2.5 - 6 / 4
       \___/
         |
2.0 +    3   * 2.5 - 6 / 4
          \_____/
             |
2.0 +       7.5    - 6 / 4
                      \_/
                       |
2.0 +       7.5    -   1
   \_________/
        |
       9.5         -   1
         \_____________/
                |
               8.5
```


Type casting

- **type cast**: A conversion from one type to another.
  Common uses:
  - To promote an `int` into a `double` to achieve exact division.
  - To truncate a `double` from a real number to an integer.
- General syntax:
  (***<type>***) ***<expression>***
- Examples:

```
(double)19 / 5      // 3.8
(int)3.8            // 3
```


Type casting

- Type casting has high precedence and only casts the item immediately next to it.

```
(double)1 + 1 / 2   // 1.0
(double)1 / 2;      // 0.5
```

- You can use parentheses to force evaluation order.

```
(double)(7 + 3 + 4) / 3
```

- A conversion to `double` can be achieved in other ways.

```
1.0 * (7 + 3 + 4) / 3
```


Concatenation: Operating on strings

- **string concatenation**: Using the + operator between a string and another value to make a longer string.
- Examples:
  `"hello" + 42` is `"hello42"`
  `1 + "abc" + 2` is `"1abc2"`
  `"abc" + 1 + 2` is `"abc12"`
  `1 + 2 + "abc"` is `"3abc"`
  `"abc" + 9 * 3` is `"abc27"` (what happened here?)
  `"1" + 1` is `"11"`
  `4 - 1 + "abc"` is `"3abc"`

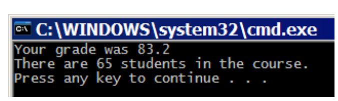
  `"abc" + 4 - 1` causes a compiler error. Why?

String expressions

- Let's print more complicated messages with computed values.

```
System.out.println("Your grade was " + ((95.1 + 71.9 +
82.6) / 3.0));

System.out.println("There are " + (11 + 17 + 4 + 19 +
14) + " students in the course.");
```



What was the answer again?

- Using the data from the last slide, what if we wanted to print the following?

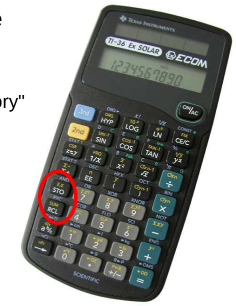
```
Your grade was 83.2
Summary:
Course grade: 83.2
```

- Answer?

```
System.out.println("Your grade was " + ((95.1 + 71.9 +
82.6) / 3.0));
System.out.println("Summary:");
System.out.println("Course grade: " + ((95.1 + 71.9 +
82.6) / 3.0));
```


What was the answer again?

- Evaluating expressions are somewhat like using the computer as a calculator.
    - A good calculator has "memory" keys to store and retrieve a computed value.



Variables

- **variable**: A piece of your computer's memory that is given a name and type and can store a value.
    - Usage:
        - compute an expression's result
        - store that result into a variable
        - use that variable later in the program
- Variables are a bit like preset stations on a car stereo:

Declaring variables

- To use a variable, first it must be *declared*.
- Variable declaration syntax:
  <type> <name>;
- Convention: Variable identifiers follow the same rules as method names.
- Examples:

```
int x;
double myGPA;
int varName;
```


Declaring variables

- Declaring a variable sets aside a piece of memory in which you can store a value.

```
int x;
int y;
```

- Inside the computer:

  x: | ? | y: | ? |

  (The memory still has no value yet.)

Setting variables

- **assignment statement**: A Java statement that stores a value into a variable.
    - Variables must be declared before they can be assigned a value.
- Assignment statement syntax:
  <variable> = ***<expression>***;
- Examples:

```
x = 2 * 4;
myGPA = 3.25;
```

  x: | 8 | myGPA: | 3.25 |

Setting variables

- A variable can be assigned a value more than once.
- Example:

```
int x;
x = 3;
System.out.println(x);   // 3

x = 4 + 7;
System.out.println(x);   // 11
```


Using variables

- Once a variable has been assigned a value, it can be used in any expression.

```
int x;
x = 2 * 4;
System.out.println(x * 5 - 1);
```

  - The above has output equivalent to:

```
System.out.println(8 * 5 - 1);
```

- What happens when a variable is used on both sides of an assignment statement?

```
int x;
x = 3;
x = x + 2;   // what happens?
```


Errors in coding

- **ERROR:** Declaring two variables with the same name
  - Example:

```
int x;
int x;         // ERROR: x already exists
```

- **ERROR:** Reading a variable's value before it has been assigned
  - Example:

```
int x;
System.out.println(x);   // ERROR: x has no value
```


Assignment vs. algebra

- The assignment statement is not an algebraic equation!
- ***<variable>*** = ***<expression>*;** means:
  - "store the value of ***<expression>*** into ***<variable>***"
- Some people read `x = 3 * 4;` as
  - "x gets the value of `3 * 4`"
- **ERROR**: `3 = 1 + 2;` is an illegal statement, because 3 is not a variable.

Assignment and types

- A variable can only store a value of its own type.
  - Example:

```
int x;
x = 2.5;   // ERROR: x can only store int
```

- An `int` value can be stored in a `double` variable. Why?
  - The value is converted into the equivalent real number.
  - Example:

```
double myGPA;            myGPA: [ 2.0 ]
myGPA = 2;
```


Assignment exercise

- What is the output of the following Java code?

```
int x;
x = 3;
int y;
y = x;
x = 5;
System.out.println(x);
System.out.println(y);
```

Assignment exercise

- What is the output of the following Java code?

```
int number;
number = 2 + 3 * 4;
System.out.println(number - 1);
number = 16 % 6;
System.out.println(2 * number);
```

- What is the output of the following Java code?

```
double average;
average = (11 + 8) / 2;
System.out.println(average);
average = (5 + average * 2) / 2;
System.out.println(average);
```

37

Shortcut: Declaring and initializing

- A variable can be declared and assigned an initial value in the same statement.
- Declaration/initialization statement syntax:

  <type> <name> `=` ***<expression>***`;`

  - Examples:

```
double myGPA = 3.95;
int x = (11 % 3) + 12;
```

38

Shortcut: Declaring many variables at once

- It is legal to declare multiple variables on one line:

  <type> <name>`,` ***<name>***`, ...,` ***<name>***`;`

  - Examples:

```
int a, b, c;
double x, y;
```

- It is also legal to declare/initialize several at once:

  <type> <name> `=` ***<expression>*** `, ...,` ***<name>*** `=` ***<expression>***`;`

  - Examples:

```
int a = 2, b = 3, c = -4;
double grade = 3.5, delta = 0.1;
```

- NB: The variables must be of the same type.

39

Shortcut: Modify and assign

- Java has several shortcut operators that allow you to quickly modify a variable's value.

Shorthand	Equivalent longer version
<variable> `+=` ***<exp>***`;`	***<variable>*** `=` ***<variable>*** `+` **(*<exp>*)**`;`
<variable> `-=` ***<exp>***`;`	***<variable>*** `=` ***<variable>*** `-` **(*<exp>*)**`;`
<variable> `*=` ***<exp>***`;`	***<variable>*** `=` ***<variable>*** `*` **(*<exp>*)**`;`
<variable> `/=` ***<exp>***`;`	***<variable>*** `=` ***<variable>*** `/` **(*<exp>*)**`;`
<variable> `%=` ***<exp>***`;`	***<variable>*** `=` ***<variable>*** `%` **(*<exp>*)**`;`

- Examples:
  - `x += 3 - 4;`   `// x = x + (3 - 4);`
  - `gpa -= 0.5;`   `// gpa = gpa – (0.5);`
  - `number *= 2;`   `// number = number * (2);`

40

Shortcut: Increment and decrement

- *Incrementing* and *decrementing* 1 is used often enough that they have a special shortcut operator!

Shorthand	***Equivalent longer version***
<variable>`++;`	***<variable>*** `=` ***<variable>*** `+ 1;`
<variable>`--;`	***<variable>*** `=` ***<variable>*** `- 1;`

- Examples:

```
int x = 2;
x++;                  // x = x + 1;
                      // x now stores 3

double gpa = 2.5;
gpa++;                // gpa = gpa + 1;
                      // gpa now stores 3.5
```

41

Putting it all together: Exercise

- Write a program that stores the following data:
  - Section AA has 17 students.
  - Section AB has 8 students.
  - Section AC has 11 students.
  - Section AD has 23 students.
  - Section AE has 24 students.
  - Section AF has 7 students.
  - The average number of students per section.
- Have your program print the following:

```
There are 24 students in Section AE.
There are an average of 15 students per section.
```

42

The `for` loop and scope

Readings: 2.3 – 2.4

Repetition

- How can we eliminate this redundancy?

```
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
System.out.println("I will not throw the principal's toupee down the toilet");
```


Looping via the `for` loop

- **`for` loop**: A block of Java code that executes a group of statements repeatedly until a given test fails.
- General syntax:

```
for (<initialization>; <test>; <update>) {
    <statement>;
    <statement>;
    ...
    <statement>;
}
```

- Example:

```
for (int i = 1; i <= 30; i++) {
    System.out.println("I will not throw...");
}
```


The `for` loop is **NOT** a method

- The `for` loop is a ***control structure***—a syntactic structure that *controls* the execution of other statements.
- Example:
  - "Shampoo hair. Rinse. **Repeat**."

`for` loop over range of `int`s

- We'll write `for` loops over integers in a given range.
  - The ***<initialization>*** declares a *loop counter* variable that is used in the test, update, and body of the loop.

```
for (int <name> = 1; <name> <= <value>; <name>++) {
```

- Example:

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

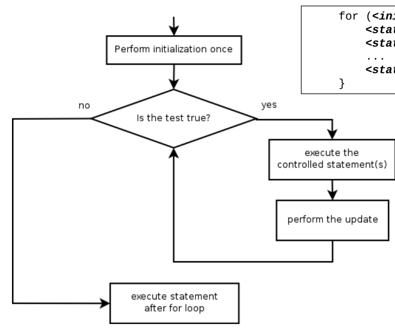
"For each `int` *i* from 1 through 4, ..."

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
```


`for` loop flow diagram

```
for (<init>; <test>; <update>) {
    <statement>;
    <statement>;
    ...
    <statement>;
}
```



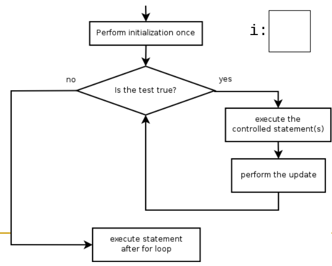
Loop walkthrough

- Code:

```
for (int i = 1; i <= 3; i++) {
    System.out.println(i + " squared is " + (i * i));
}
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
```



Loop example

- Code:

```
System.out.println("+----+");
for (int i = 1; i <= 3; i++) {
    System.out.println("\\    /");
    System.out.println("/    \\");
}
System.out.println("+----+");
```

Output:

```
+----+
\    /
/    \
\    /
/    \
\    /
/    \
+----+
```


Varying the `for` loop

- The initial and final values for the loop counter variable can be arbitrary expressions:
- Example:

```
for (int i = -3; i <= 2; i++) {
    System.out.println(i);
}
```

Output:

```
-3
-2
-1
0
1
2
```

- Example:

```
for (int i = 1 + 3 * 4; i <= 5248 % 100; i++) {
    System.out.println(i + " squared is " + (i * i));
}
```


Varying the `for` loop

- The update can be a `--` (or any other operator).
  - *Caution*: This requires changing the test from <= to >= .

```
System.out.print("T-minus");
for (int i = 3; i >= 1; i--) {
    System.out.println(i);
}
System.out.println("Blastoff!");
```

Output:

```
T-minus
3
2
1
Blastoff!
```


Errors in coding

- When controlling a single statement, the `{}` braces are optional.

```
for (int i = 1; i <= 6; i++)
    System.out.println(i + " squared is " + (i * i));
```

- This can lead to errors if a line is not properly indented.

```
for (int i = 1; i <= 3; i++)
    System.out.println("This is printed 3 times");
    System.out.println("So is this... or is it?");
```

Output:

```
This is printed 3 times
This is printed 3 times
This is printed 3 times
So is this... or is it?
```

- **Moral**: Always use curly braces and always use proper indentation.

Errors in coding

- **ERROR**: Loops that never execute.

```
for (int i = 10; i < 5; i++) {
    System.out.println("How many times do I print?");
}
```

- **ERROR**: Loop tests that never fail.
  - A loop that never terminates is called an *infinite loop*.

```
for (int i = 10; i >= 1; i++) {
    System.out.println("Runaway Java program!!!");
}
```

`for` loop exercises

- Write a loop that produces the following output.

```
On day #1 of Christmas, my true love sent to me
On day #2 of Christmas, my true love sent to me
On day #3 of Christmas, my true love sent to me
On day #4 of Christmas, my true love sent to me
On day #5 of Christmas, my true love sent to me
...
On day #12 of Christmas, my true love sent to me
```

55

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 { } braces in which it was declared.
  - Special case: If a variable is declared in the ***<initialization>*** part of a `for` loop, its scope is the `for` loop.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {      i's scope
        System.out.println(x);
    }                                              x's scope
    // i no longer exists here
} // x ceases to exist here
```

- Why not just have the scope of a variable be the whole program?

56

Errors in coding

- **ERROR:** Using a variable outside of its scope.

```
public static void main(String[] args) {
    int z = 0;
    example();
    System.out.println(x);  // illegal: x is out of scope

    for (int i = 1; i <= 10; i++) {
        int y = 5;
        System.out.println(y);
    }
    System.out.println(y);  // illegal: y is out of scope
}

public static void example() {
    int x = 3;
    System.out.println(x);
    System.out.println(z);  // illegal: z is out of scope
}
```

57

Errors in coding

- **ERROR**: Declaring variables with the same name with overlapping scope.

```
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 x = 2;
    int y = 3;
    System.out.println(i + ", " + x + ", " + y);
}
```

58

Mapping loops to numbers

- Suppose that we have the following loop:

```
for (int count = 1; count <= 5; count++) {
    ...
}
```

- What statement could we write in the body of the loop that would make the loop print the following output?

```
3 6 9 12 15
```

59

But first… How to print on the same line

- `System.out.print` prints the given output **without** moving to the next line.

```
System.out.print("T-minus ");
for (int i = 3; i >= 1; i--) {
    System.out.print(i + " ");
}
System.out.println("Blastoff!");
```

<u>Output</u>:

```
T-minus 3 2 1 Blastoff!
```

60

Mapping loops to numbers

- Suppose that we have the following loop:

```
for (int count = 1; count <= 5; count++) {
    ...
}
```

- What statement could we write in the body of the loop that would make the loop print the following output?

`3 6 9 12 15`

- Answer:

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + " ");
}
```

61

Mapping loops to numbers

- Now consider another loop of the same style:

```
for (int count = 1; count <= 5; count++) {
    ...
}
```

- What statement could we write in the body of the loop that would make the loop print the following output?

`4 7 10 13 16`

- Answer:

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
```

62

Loop number tables

- What statement could we write in the body of the loop that would make the loop print the following output?

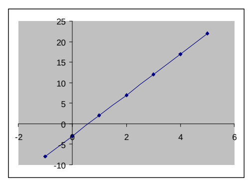
`2 7 12 17 22`

- To find the pattern, it can help to make a table.
  - Each time `count` goes up by `1`, the number should go up by `5`.
  - But `count * 5` is too big by `3`, so we must subtract `3`.

`count`	number to print	`count * 5`	`count * 5 - 3`
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

63

Another perspective: Slope-intercept



`count (x)`	number to print (y)
1	2
2	7
3	12
4	17
5	22

64

Another perspective: Slope-intercept

- *Caution*: This is algebra, not assignment!
- Recall: slope-intercept form (`y = mx + b`)
- Slope is defined as "rise over run" (i.e. rise / run). Since the "run" is always `1` (we increment along `x` by `1`), we just need to look at the "rise". The rise is the difference between the `y` values. Thus, the slope (`m`) is the difference between `y` values; in this case, it is `+5`.
- To compute the y-intercept (`b`), plug in the value of `y` at `x = 1` and solve for `b`. In this case, `y = 2`.

```
y = m * x + b
2 = 5 * 1 + b
```

Then `b = -3`

- So the equation is

```
y = m * x + b
y = 5 * x - 3
y = 5 * count - 3
```

`count (x)`	number to print (y)
1	2
2	7
3	12
4	17
5	22

65

Another perspective: Slope-intercept

- Algebraically, if we always take the value of `y` at `x = 1`, then we can solve for `b` as follows:

  $y = m * x + b$

  $y_1 = m * 1 + b$

  $y_1 = m + b$

  $b = y_1 - m$

- In other words, to get the y-intercept, just subtract the slope from the first `y` value (`b = 2 - 5 = -3`)
  - This gets us the equation

```
y = m * x + b
y = 5 * x - 3
y = 5 * count - 3
```

  (which is exactly the equation from the previous slides)

66

Loop table exercise

- What statement could we write in the body of the loop that would make the loop print the following output?
  `17 13 9 5 1`
- Let's create the loop table together.
  - Each time count goes up 1, the number should ...
  - But this multiple is off by a margin of ...

count	number to print	count * -4	count * -4 + 21
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

Nested `for` loops

- **nested loop**: Loops placed inside one another.
  - *Caution*: Make sure the inner loop's counter variable has a different name!

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

Output:

```
i = 1
  j = 1
  j = 2
i = 2
  j = 1
  j = 2
i = 3
  j = 1
  j = 2
```


Nested loops example

- Code:

```
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 loops example

- Code:

```
for (int i = 1; i <= 6; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Output:

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


Nested loops example

- Code:

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

Output:

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


Nested loops example

- Code:

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

Output:

```
1
22
333
4444
55555
666666
```

Nested loops example

- Code:
```
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();
}
```

  Output:
```
    1
   22
  333
 4444
55555
```


Nested loops

- What nested `for` loops produce the following output?

  inner loop (repeated characters on each line)
```
....1
...2
..3
.4
5
```
  outer loop (loops 5 times because there are 5 lines)

- Key idea:
  - outer "vertical" loop for each of the lines
  - inner "horizontal" loop(s) for the patterns within each line

Nested loops

- First, write the outer loop from `1` to the number of lines desired.
```
for (int line = 1; line <= 5; line++) {
    ...
}
```
- Notice that each line has the following pattern:
  - some number of dots (0 dots on the last line)
  - a number
```
....1
...2
..3
.4
5
```


Nested loops

- Make a table:
```
....1
...2
..3
.4
5
```

line	# of dots	line * -1 + 5	value displayed
1	4	4	1
2	3	3	2
3	2	2	3
4	1	1	4
5	0	0	5

- Answer:
```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (line * -1 + 5); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```


Nested Loops

- Modify the previous code to produce this output:
```
....1
...2.
..3..
.4...
5....
```

line	# of dots	value displayed	# of dots
1	4	1	0
2	3	2	1
3	2	3	2
4	1	4	3
5	0	5	4

- Answer:
```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (line * -1 + 5); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}
```


Errors in coding

- **ERROR**: Using the wrong loop counter variable.
  - What is the output of the following piece of code?
```
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 piece of code?
```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 5; i++) {
        System.out.print(j);
    }
    System.out.println();
}
```

Managing complexity

Readings: 2.4 – 2.5

Drawing complex figures

- Write a program that produces the following figure as its output:

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

Where do we even start??

Drawing complex figures: Strategy

- Write down some steps on paper before coding:
  1. A *pseudo-code* description of the algorithm (in English)
  2. A table of each line's contents, to help see the pattern in the input

Pseudo-code

- **pseudo-code**: A written English description of an algorithm
- Example: Suppose we are trying to draw a box of stars which is 12 characters wide and 7 tall.

print 12 stars.
for each of 5 lines,
  print a star.
  print 10 spaces.
  print a star.
print 12 stars.

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


Drawing complex figures: Pseudo-code

- A possible pseudo-code for our complex figure task:
  1. *Draw top line with # , 16 =, then #*
  2. *Draw the top half with the following on each line:*
     - |
     - *some spaces (possibly 0)*
     - <>
     - *some dots (possibly 0)*
     - <>
     - *more spaces (possibly 0)*
     - |
  3. *Draw the bottom half, which is the same as the top half but upside-down*
  4. *Draw bottom line with # , 16 =, then #*

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


Drawing complex figures: Tables

- A table of the lines in the "top half" of the figure:

line	spaces	line * -2 + 8	dots	line * 4 - 4
1	6	6	0	0
2	4	4	4	4
3	2	2	8	8
4	0	0	12	12

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

Drawing complex figures: Questions

- How many loops do we need on each line of the top half of the output?
- Which loops are nested inside which other loops?
- How should we use static methods to represent the structure and redundancy of the output?

Partial solution

```
// Prints the expanding pattern of <> for the top half of the figure.
public static void drawTopHalf() {
    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("|");
    }
}
```

- Question: Is there a pattern to the numbers?

Magic numbers

- Sometimes we have values (called *magic numbers*) that are used throughout the program.
  - A normal variable cannot be used to fix the magic number problem. Why not?

```
public static void main(String[] args) {
    int max = 3;
    printTop();
    printBottom();
}

public static void printTop() {
    for (int i = 1; i <= max; i++) {        // ERROR: max not found (out of scope)
        for (int j = 1; j <= i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}

public static void printBottom() {
    for (int i = max; i >= 1; i--) {        // ERROR: max not found (out of scope)
        for (int j = i; j >= 1; j--) {
            System.out.print(max);          // ERROR: max not found (out of scope)
        }
        System.out.println();
    }
}
```


Solution: Class constants

- **class constant**: A variable that can be seen throughout the program.
  - The value of a constant can only be set when it is declared.
  - It cannot be changed while the program is running, hence the name: *constant*.

Class constant: Syntax

- Syntax:
  `public static final` ***<type> <name>*** `=` ***<value>***`;`
- Class constants have to be declared outside the methods.
- Convention: Constant identifiers are written in uppercase with words separated by underscores.
- Examples:

```
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 3.5;
public static final int SSN = 658234569;
```


Class constant example

- Class constants eliminates redundancy.

```
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();
    }
}
```

Constants and figures

- Consider the task of drawing the following figures.

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

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

- How can a class constant help?

91

Boo! Redundancy! Boo!

- 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.

92

Class constants to the rescue!

- A class constant will fix the "magic number" problem.

```
public static final int FIGURE_WIDTH = 5;

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

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

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

93

Drawing complex figures: Resizing

- Modify the previous program to use a constant so that it can show figures of different sizes.
  - The figure originally shown has a size of 4.

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

A figure of size 3:

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

94

Drawing complex figures: Resizing

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

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

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

95

Partial solution

```
public static final int SIZE = 4;

// Prints the expanding pattern of <> for the top half of the figure.
public static void drawTopHalf() {
    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("|");
    }
}
```

96

Class constant trickiness

- Adding a constant often changes the amount that is added to a loop expression, but the multiplier (slope) is usually unchanged.

```
public static final int SIZE = 4;

for (int space = 1; space <= (line * -2 + (2 * SIZE)); space++) {
    System.out.print(" ");
}
```

- *Caution*: A constant does **NOT** always replace every occurrence of the original value.

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


Complex figure exercise

- Write a program that produces the following figure as its output.
  - Use nested `for` loops and static methods where appropriate.

```
====+====
#   |   #
#   |   #
#   |   #
====+====
#   |   #
#   |   #
#   |   #
====+====
```

- Add a constant so that the figure can be resized.

Assignment 2: Space Needle