Data and Expressions

Variables

Primitives

Strings

Operators

"Just a darn minute! — Yesterday you said that X equals **two**!"

VARIABLES: DECLARE, INITIALIZE, ASSIGN

Variable Declaration

- A *variable* is a name for a location in memory
- Variables must be *declared*, meaning you state the:
  - variable type
  - variable name
- Examples:
  - `int total;`
  - `int count, sum;`
  - `String firstName;`

Variable Type

- All Java variables are either a primitive type or an Object.
  - Repeat after me: All Java variables are either primitives or Objects!

Variable Name

- Identifiers (including variable names) can consist of:
  - Letters
  - Digits (cannot start with a digit)
  - Underscore
  - Dollar sign
- By convention, Java variable names:
  - are descriptive
  - start with lower case
  - use camel case (e.g., studentFirstName)

Variable Declaration (cont.)

- Declaring a primitive variable sets aside the necessary space in memory, but does not assign a value.
- You cannot use a variable that has *only* been declared.

```
int n;
n++; // compiler error

String myString;
System.out.println(myString); // compiler error
```

Variable Initialization

- To use a variable, you must initialize it.
  - You can declare and initialize in separate statements or in the same statement.

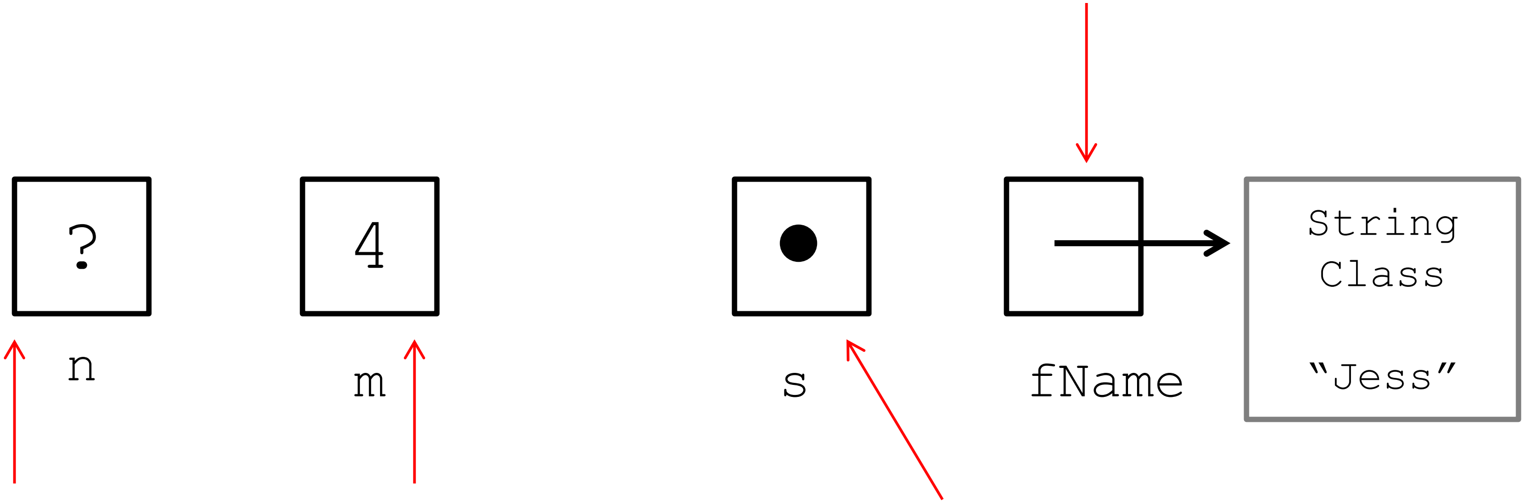
- Examples:
  - `int sum = 0;`

  -
    ```
    int n;
    n = 4;
    ```
  - `int base = 32, mid, max = 149;`

Declaration and Initialization: What Happens in Memory

```
int n;
int m = 4;
String s;
String fName = “Jess”;
```

a *pointer* or *reference* is stored that points to the place in memory that holds the object



enough space set aside to hold an int

the primitive value is stored directly in memory

default for objects is null

Memory: Primitives vs. Objects

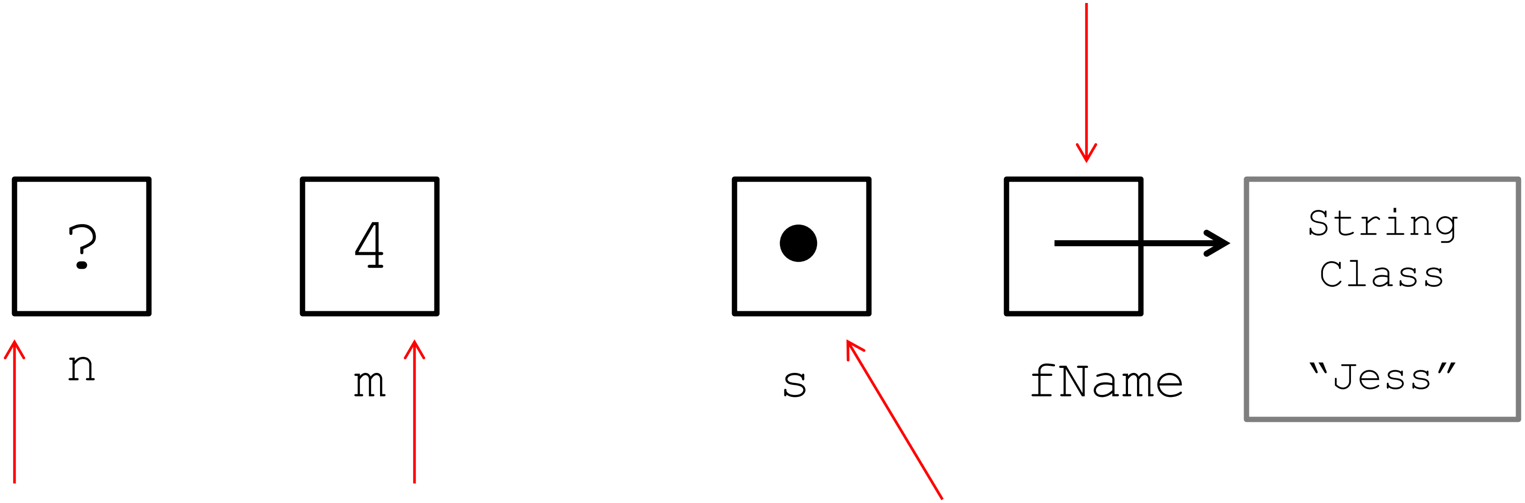
- Primitive variables hold a value directly in memory.
- Object (or reference) variables hold a pointer to a place in memory.
  - That place in memory stores all of the information needed for the object.
- **This is a critical distinction in Java!**

Assignment

- An *assignment* statement changes the value of a variable.
- The *assignment operator* is the equals sign =
- Everything on the right of the assignment operator is evaluated first.
  - Then the value is stored in the variable on the left.

Declaration and Initialization: What Happens in Memory

```
int n;
int m = 4;
String s;
String fName = “Jess”;
```



?

4

●

String Class

“Jess”

n

m

s

fName

a *pointer* or *reference* is stored that points to the place in memory that holds the object

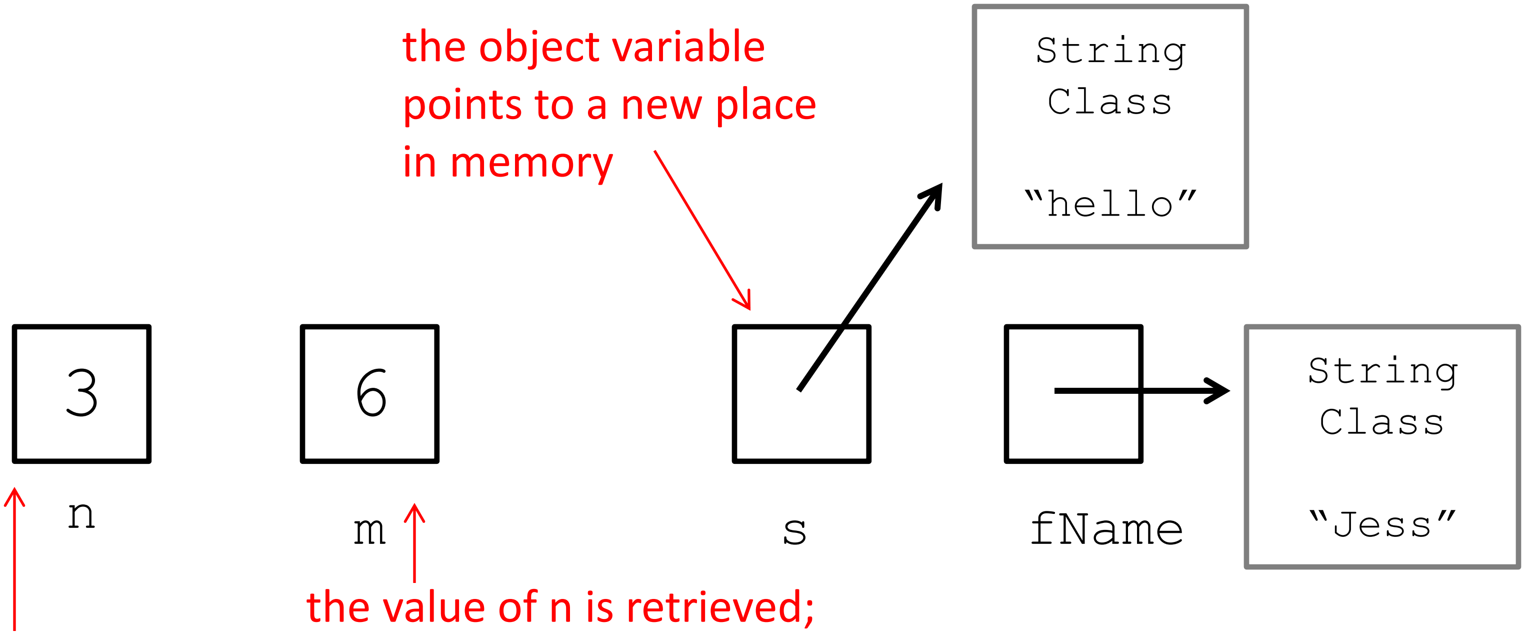
enough space set aside to hold an int

the primitive value is stored directly in memory

default for objects is null

Assignment: What Happens in Memory

```
n = 3;
m = n * 2;
s = "hello";
```



the object variable points to a new place in memory

a new value is assigned

the value of n is retrieved; a new value is calculated and assigned- the old value is overridden and forgotten

Key Point: Objects in Memory

- Object variables hold a pointer to a place in memory.
  - That place in memory stores all of the information needed for the object.
- What is stored directly in an object variable is a *memory location* (also called a *reference* or *pointer*).

Constants

- A *constant* is an identified that holds the same value throughout its existence.
  - The compiler will not let you change it.
- Use the `final` keyword to declare a constant.
  - By convention, constant names are in all caps and separated with an underscore.

```
final int MIN_HEIGHT = 60;
final double AVG_TEMP = 98.6;
```

Constants… Why?

- They give meaning to otherwise unclear literal values.
  - Example: `MAX_STUDENTS` has more meaning than 40.
- They facilitate program maintenance.
  - If a constant is used in multiple places, its value needs to only be updated in one place.
  - Example: if you can now hold 50 students instead of 40, you only have to edit the code that initializes `MAX_STUDENTS` instead of changing the number everywhere it appears.
- They formally establish that a value should not change, avoiding inadvertent errors by programmers.

PRIMITIVE DATA TYPES

Primitive Data Types

- There are eight primitive data types in Java. Primitives store simple pieces of data.
  - Integers (whole numbers)
    - `byte, short, int, long`
  - Floating point (decimal) numbers
    - `float, double`
  - Characters
    - `char`
  - Boolean values
    - `boolean`
- The difference in numeric types is the size and range of values.
  - `int` and `double` will be used for our basic programs.
- Larger-typed variables can store smaller-typed values, but not the other way around.
  - Example: a `long` variable can hold an `int` value, but an `int` value cannot hold a `long`.

Character Data

- A `char` variable stores a single character.
- Character literals are delimited by single quotes:
  - `‘a’   ‘X’  ‘7’  ‘$”  ‘,’  ‘\n’`
- In Java, each character has an associated numeric value.
- Examples:
  - `char topGrade = ‘A’;`
  - `char terminator = ‘;’, separator = ‘ ‘;`

Boolean Data

- A `boolean` value represents a true or false condition.
- The reserved words `true` and `false` are the only valid values for a boolean type.
  - `boolean done = false;`
- A `boolean` variable can be used to represent any two states
  - On/off
  - Pass/fail
  - Win/lose
  - Selected/unselected

The Bottom Line: Using Primitives

1. Declare a variable

   ```
   int n;
   boolean finished;
   double d;
   ```

2. Initialize (assign a value)

   ```
   n = 4;
   finished = true;
   d = -9.2;
   ```

- Note: Steps 1 and 2 can be combined into a single statement.

STRINGS

Strings

- *String*: A sequence of characters surrounded by double quotes (also called a *string literal*)
  - `“This is a string literal.”`
  - `“123 Main Street!”`
  - `“X”`
  - `“”`
- `String` is a class
- String variables are objects of the `String` class

String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another
  - Can also append a number to a String
- The result is a new `String`

```
String s1 = “Peanut butter” + “ and jelly”;
// s1 refers to a String
// “Peanut butter and jelly”

String s2 = “The answer is “ + 25
//s2 refers to a String “The answer is 25”
```

String Concatenation (cont.)

- A string cannot be broken across two lines in a program, so they must be concatenated.

```
System.out.println(“This is my really long
  string that stretches way out over this
  line and onto the next.” );
// This is invalid! What type of error?


System.out.println(“This is a really long”
  + “string that stretches way out over “
  + “two lines but since I used the “
  + “concatenation “operator it works!”
```

String Concatenation (cont.)

- The + operator is used for both string concatenation and for arithmetic addition.
  - If **both** operands are numeric, the + adds them.
  - If **either** or **both** of the operands are strings, the + performs string concatenation.
- The + operator is evaluated left to right, but parentheses can be used to force the order.

String Concatenation (cont.)

- What will print?

```
System.out.println(3+4);
System.out.println(“3” + “4”);
System.out.println(3 + 4 + “5”);
System.out.println(“3” + “4” + 5 + “6”);
System.out.println(“3” + (4 + 5))
System.out.println(3 + 4 + “5” + 6);
```

Escape Sequences

- What if we wanted to print the quote character?
  - `System.out.println(“I said “Hello“ to you.”)`
  - Invalid Syntax!
- An *escape sequence* is a series of characters that represent a special character.
- Escape sequences begin with a backslash (\)
  - `System.out.println(“I said \“Hello\“ to you.”)`

`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

EXPRESSIONS AND OPERATORS

Expressions

- An *expression* is a combination of operators and operands.
  - An expression produces a typed value when it is evaluated.
- What are the values and types of these expressions?

```
2 + 3
“hello” + 6
3 > 5
```

Operators

- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

– Addition	+
– Subtraction	-
– Multiplication	*
– Division	/
– Remainder (Modulus)	%

- Binary expressions take two operands.
- If *either* or *both* operands used by an arithmetic operator are floating point, the result will be floating point.
  - If *both* are integer, the result will be integer.

Division

- Division of two `int` variables results in an `int`.
  - The decimal portion is truncated.
  - This is called *integer division*.
- Division of an `int` with a `double` or `float` results in the full division.
- Examples:

```
14/5  =
14.0/5 =
8/12 =
8.0/12 =
(1+2+3)/8 =
(1.0+2+3)/8 =
```

Division

- Division of two `int` variables results in an `int`.
  - The decimal portion is truncated.
  - This is called *integer division*.
- Division of an `int` with a `double` or `float` results in the full division.
- Examples:

```
14/5  = 2
14.0/5 = 2.8
8/12 = 0
8.0/12 = 0.66666…
(1+2+3)/8 = 0
(1.0+2+3)/8 = 0.75
```

- Results are truncated- not rounded!

Remainder

- Also called *modulus*
- The remainder (or what is *left over*) after an integer division.
- x % y

    x / y = z plus r

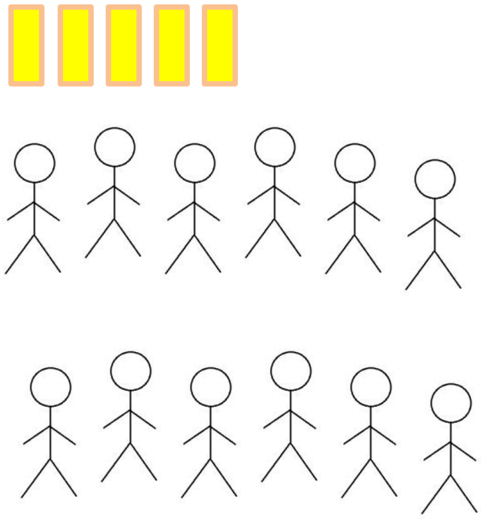
    x % y is the r
- 5 % 2

    5 / 2 is 2 plus 1 left over

    5 % 2 = 1
- Think about it like "gold bars mod people"

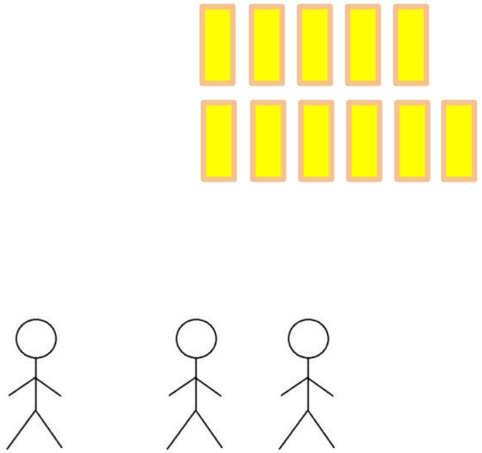
Remainder Example

- 5 % 13 = 5
- First think about integer division:
  - 5 / 13 (integer division) = 0
  - If you have 5 gold bars (bars cannot be broken up) to divide evenly between 13 people, how many bars will each person get? 0
- Then think about remainder:
  - How many gold bars are "left over" that couldn't be handed out? 5



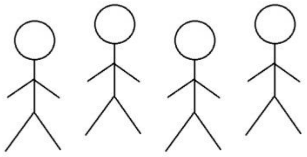
Remainder Example

- -11 % 3 = -2
- First think about integer division:
  - -11 / 3 (integer division) = -3
  - If you and your two unfortunate friends **owe** someone 11 gold bars (bars cannot be broken up), how many whole bars will each person have to pay? -3
- Then think about remainder:
  - How many gold bars are “left over” that are still owed? -2



Remainder Example

- 0 % 4 = 0
- First think about integer division:
  - 0 / 4 (integer division) = 0
  - If you have no gold bars to divide between 4 people, how many bars will each person get? 0
- Then think about remainder:
  - How many gold bars are “left over” that couldn’t be handed out? 0



Remainder Examples

- 13 % 3
- 5 % 13
- -8 % 2
- -1 % 5
- -11 % 3
- 0 % 4

Remainder Examples

- 13 % 3 = 1
- 5 % 13 = 5
- -8 % 2 = 0
- -1 % 5 = -1
- -11 % 3 = -2
- 0 % 4 = 0

Modulous… What is it good for?

- Counting by a number!
- Let’s say you had a counter:
  - Number of clicks: 0, 1, 2, 3, 4, 5, 6…
- And you want something to happen in cycles of three:
  - Green, Yellow, Red, Green, Yellow, Red, etc.

Modulous… What is it good for?

- Use modulous!

```
int remainder = clickCount % 3;
if(remainder ==0) {
  green!
} else if(remainder ==1) {
  yellow!
} else { // remainder == 2
  red!
}
```

- 0 % 3 = 0   1 % 3 = 1   2 % 3 = 2
- 3 % 3 = 0   4 % 3 = 1   5 % 3 = 2
- 6 % 3 = 0   etc.

Evaluating Expressions

- Operators are grouped by precedence to determine the order in which they are evaluated.
- To evaluate an expression:
  1. Scan from left to right looking for the highest precedence.
  2. When you find out, evaluate the expression and replace the operands/operator with the result.
  3. Continue scanning looking for that same precedence. If you find one, repeat Step 2.
  4. When you reach the end, repeat Step 1 with the next lowest precedence operators.

Operator Precedence

1. Parentheses
2. Multiplication, division, and remainder
    a) evaluated left to right
3. Addition, subtraction, and string concatenation
    a) evaluated left to right

- Parentheses can be used to:
  - make an expression clearer
  - override precedence rules
    - Parentheses always come first!
- Treat an expression inside parentheses as a “mini expression” and evaluate it using the same rules.

Precedence Examples

```
int num1 = 3, num2 = 5, num3 = 6, num4 = 4;
```

```
num2 + num3 / num2 * num4 + num1 % num3 - num2
```

```
num2 + num3 / (num2 + num3 * num1) + num3 - num2
```

Operator Precedence

1. Parentheses
2. Multiplication, division, and remainder
    a) evaluated left to right
3. Addition, subtraction, and string concatenation
    a) evaluated left to right
4. Assignment Operator
    - always calculate to the right of the equals sign and *then* assign the value to the variable

Precedence Examples

```
int num1 = 3, num2 = 5, num3 = 6, num4 = 4;
```

```
int num5 = num2 + num3 / num2 * num4 + num1 % num3 -
   num2
```

```
num2 = num2 + num3 / (num2 + num3 * num1) + num3 -
   num2
```

Unary Operators

- Unary operands take a single operand
- Increment ++

  `num++` is the same as

  `num = num + 1`

- Decrement --

  `num--` is the same as

  `num = num -1`

Operator Precedence

1. Parentheses
2. Postfix increment and decrement
3. Multiplication, division, and remainder
   a) evaluated left to right
4. Addition, subtraction, and string concatenation
   a) evaluated left to right
5. Assignment Operator
   – always calculate to the right of the equals sign and *then* assign the value to the variable

Assignment Operators

- =
- +=

```
num1 += num2
```

is the same as

```
num1  = num1 + num2
```

- -=

```
num1 -= num2
```

is the same as

```
num1 = num1 - num2
```

- *=
- /=
- %=

Assignment Operators (cont.)

- If the operands to the += operator are numbers, the assignment operator performs addition.

```
int n = 4;
String s = “hello”;
s += n;
n += 1;
System.out.println(s);
     // prints  hello4
System.out.println(n);
     // prints 5
```

- If the operands to the /= operator are integers, the assignment operator performs integer division.

```
int a = 4, b = 21;
b /= a;
System.out.println(b); // prints 5
```

Practice

- What is the type and value of num?

```
int num1 = 4, num2 = 3, num3 =
  2;
```

```
num1 +=
  num1 * num3 + num3 / num2 + num3 - num1 % num2 % num1;
```

Practice

- What is the type and value of dub1?

```
int num1 = 6,
num2 = 4,
num3 = 11;

double dub1 = 2,
dub2 = 2.5,
dub3 = 2;


dub1 *= num1 * dub2 + num3 / (num2 + (num1 -
  num1 / num2)) / dub3;
```

Text-Based Practice

- Write a text-based program that reads values from the time representing the number of hours, minutes, and seconds. Calculate and print out the total number of seconds.
- Write a text-based program that does the reverse (reads in total seconds and prints out hours, minutes, and seconds).