CSC220 (CSI)
Computational Problem Solving

The College of New Jersey

Please turn off your cell phone!

Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact

Object-Oriented Programming

- The key to designing a solution:
  - breaking it down into manageable pieces
- For an object-oriented programming language such as Java, a fundamental entity is an object.
  - A student in a class
  - An employee in a company
  - …
- We will dissect our solutions into pieces called objects and classes

Objects

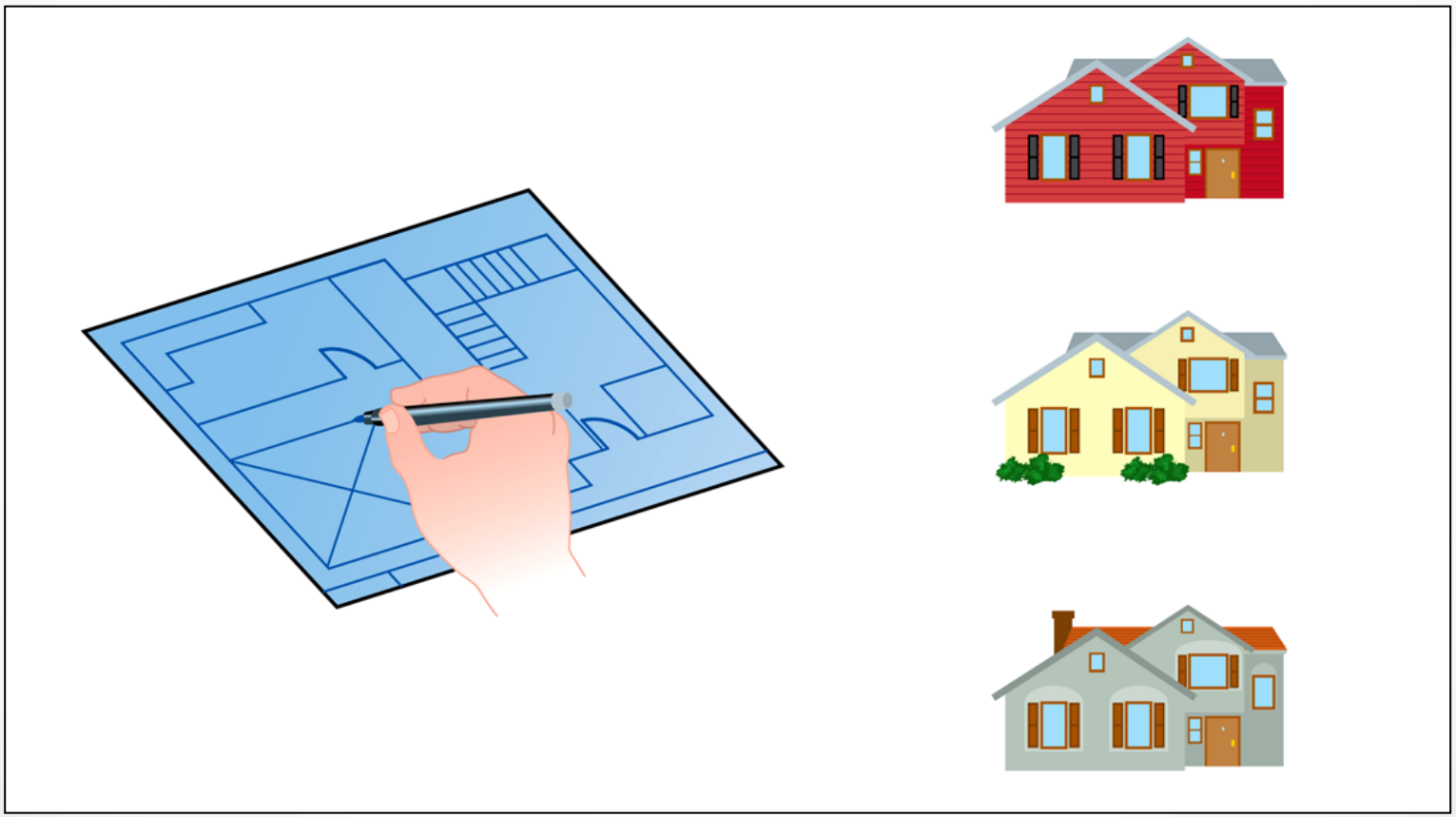
- An object has:
  - State – descriptive characteristics
  - Behaviors – what it can do (or what can be done to it)
- Example: a bank account
  - State
    - Account number
    - Current balance
    - …
  - Behavior
    - Make deposits
    - Make withdrawals
    - …

Classes

- An object is defined by a class
- A class is the blueprint of an object
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program

Class = Blueprint

- One blueprint to create several similar, but different, houses:



Objects and Classes

A class
(the concept)

Multiple objects
from the same class

An object
(the realization)

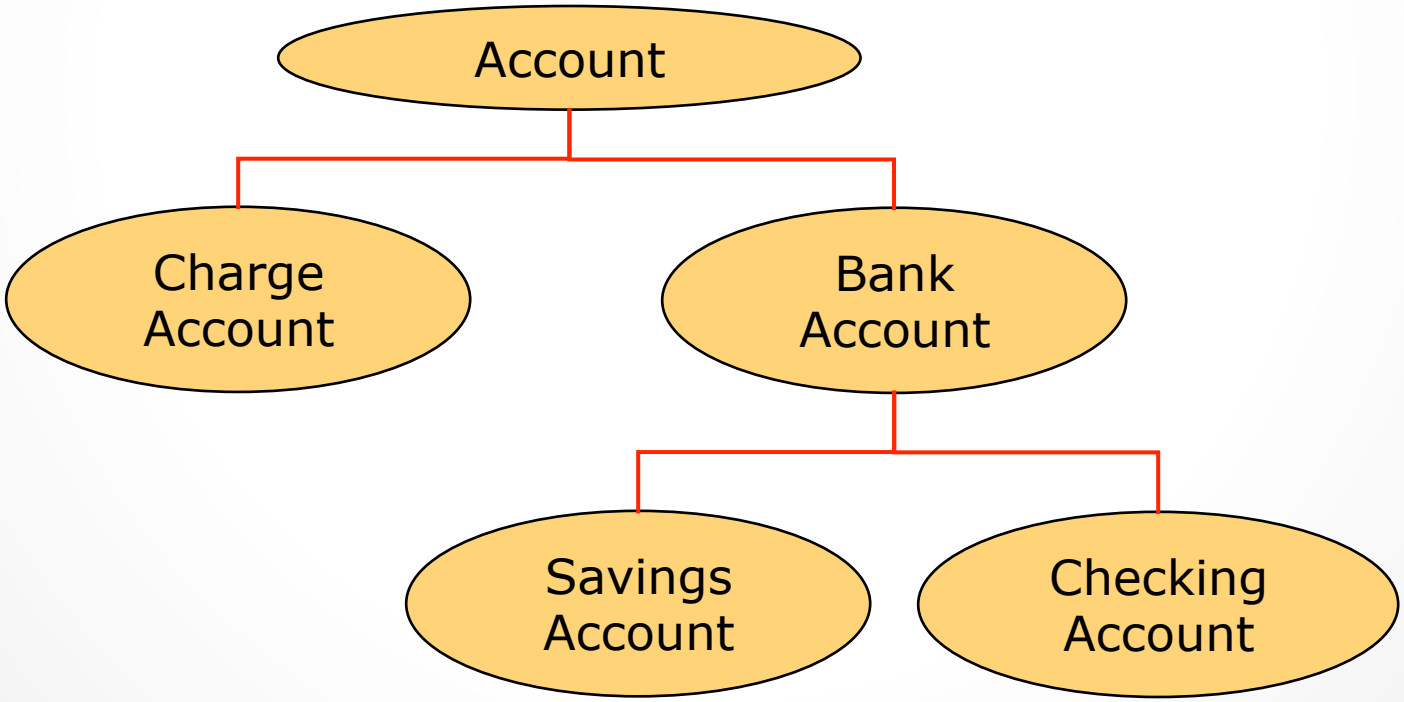
John's Bank Account
Balance: $5,257

Bill's Bank Account
Balance: $1,245,069

Mary's Bank Account
Balance: $16,833

Inheritance

- One class can be used to derive another via inheritance
- Classes can be organized into hierarchies



Fundamentals

Data and Expression

Character Strings

- A string literal is represented by putting double quotes around the text
- Examples:

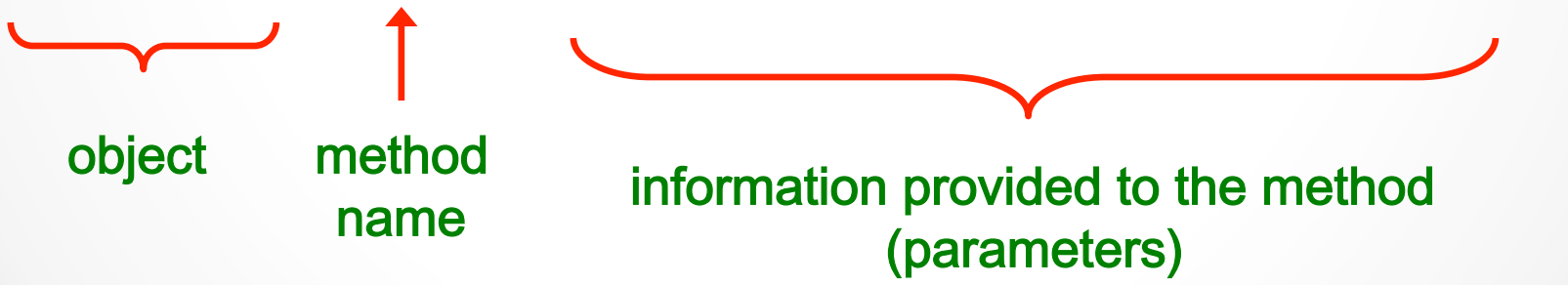
```
"This is a string literal."
"123 Main Street"
"X"
```

- Every character string is an object in Java, defined by the `String` class
- Every string literal represents a `String` object

The println Method

- In the `Lincoln` program from Chapter 1, we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```



Output

```
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.
```

```
//****
//  Co                                                          ****
//
//  Demonstrates the difference between print and println.
//********************************************************************

public class Countdown
{
   //-----------------------------------------------------------------
   //  Prints two lines of output representing a rocket countdown.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.print("Three... ");
      System.out.print("Two... ");
      System.out.print("One... ");
      System.out.print("Zero... ");
      System.out.println("Liftoff!");  // appears on first output line
      System.out.println("Houston, we have a problem.");
   }
}
```

String Concatenation

- The string concatenation operator (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program
- See `Facts.java`

Output

```
We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

```
   //  Prints various facts.
   //--------------------------------------------------------------
   public static void main(String[] args)
   {
      // Strings can be concatenated into one long string
      System.out.println("We present the following facts for your "
                         + "extracurricular edification:");
      System.out.println();
      // A string can contain numeric digits
      System.out.println("Letters in the Hawaiian alphabet: 12");
      // A numeric value can be concatenated to a string
      System.out.println("Dialing code for Antarctica: " + 672);
      System.out.println("Year in which Leonardo da Vinci invented "
                         + "the parachute: " + 1515);
      System.out.println("Speed of ketchup: " + 40 + " km per year");
   }
}
```

Output

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

```
//**********                                                    **********
//  Addition.
//
//  Demonstrates the difference between the addition and string
//  concatenation operators.
//********************************************************************

public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("24 and 45 concatenated: " + 24 + 45);

      System.out.println("24 and 45 added: " + (24 + 45));
   }
}
```

Quick Check

What output is produced by the following?

```
System.out.println("X: " + 25);
System.out.println("Y: " + (15 + 50));
System.out.println("Z: " + 300 + 50);
```

```
X: 25
Y: 65
Z: 30050
```

Escape Sequences

- What if we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println("I said "Hello" to you.");
```

- An escape sequence is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println("I said \"Hello\" to you.");
```

Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

- See `Roses.java`

Output

```
Roses are red,
   Violets are blue,
Sugar is sweet,
   But I have "commitment issues",
   So I'd rather just be friends
   At this point in our relationship.
```

```
//****
//  Ro
//
//  De
//****

public
{
   //-----------------------------------------------------------------
   //  Prints a poem (of sorts) on multiple lines.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```

Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home
tonight," he said mysteriously.

```
System.out.println("\"Thank you all for " +
   "coming to my home\ntonight,\" he said " +
   "mysteriously.");
```

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

The statement creates a `Scanner` object that reads from the keyboard.

The `new` operator creates the `Scanner` object.

Once created, the `Scanner` object can be used to invoke various input methods.

```
public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      String message;
      Scanner scan = new Scanner(System.in);

      System.out.println("Enter a line of text:");

      message = scan.nextLine();

      System.out.println("You entered: \"" + message + "\"");
   }
}
```

The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used.

The `nextLine` method reads all of the input until the end of the line is found.

```
import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      String message;
      Scanner scan = new Scanner(System.in);

      System.out.println("Enter a line of text:");

      message = scan.nextLine();

      System.out.println("You entered: \"" + message + "\"");
   }
}
```

Sample Run

```
Enter a line of text:
You want fries with that?
You entered: "You want fries with that?
```

```
//********
//  Ec
//
//  De
//  to
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      String message;
      Scanner scan = new Scanner(System.in);

      System.out.println("Enter a line of text:");

      message = scan.nextLine();

      System.out.println("You entered: \"" + message + "\"");
   }
}
```

```
//********************************************************************
//  GasMileage.java       Author: Lewis/Loftus
//
//  Demonstrates the use c
//********************
import java.util.Scanner;
public class GasMileage
{
   //-----------------
   //  Calculates fuel eff
   //  user.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int miles;
      double gallons, mpg;

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter the number of miles: ");
      miles = scan.nextInt();

      System.out.print("Enter the gallons of fuel used: ");
      gallons = scan.nextDouble();

      mpg = miles / gallons;

      System.out.println("Miles Per Gallon: " + mpg);
   }
}
```

Sample Run

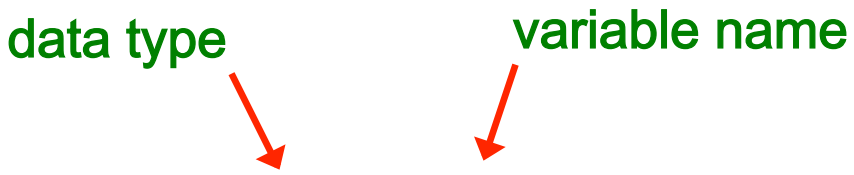
```
Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
```

Input Tokens

- Unless specified otherwise, white space is used to separate the elements (called tokens) of the input
- White space includes
  - space characters
  - tabs
  - new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types

Variables

- A variable is a name for a location in memory that holds a value
- A variable declaration specifies the variable's name and the type of information that it will hold




```
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- See `PianoKeys.java`

Output

```
A piano has 88 keys.
```

```
//********************************************************************
//  PianoKeys.java       Author: Lewis/Loftus
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable.
//********************************************************************

public class PianoKeys
{
   //-----------------------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int keys = 88;
      System.out.println("A piano has " + keys + " keys.");
   }
}
```

Assignment

- An assignment statement changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The value that was in `total` is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type
- See `Geometry.java`

Output

```
A heptagon has 7 sides.
A decagon has 10 sides.
A dodecagon has 12 sides.
```

```
//*************                                          ***************
//  Geometry.ja
//
//  Demonstrate                                        change the
//  value store
//*************                                          ***************

public class Geometry
{
   //-----------------------------------------------------------------
   //  Prints the number of sides of several geometric shapes.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int sides = 7;  // declaration with initialization
      System.out.println("A heptagon has " + sides + " sides.");

      sides = 10;  // assignment statement
      System.out.println("A decagon has " + sides + " sides.");

      sides = 12;
      System.out.println("A dodecagon has " + sides + " sides.");
   }
}
```

Constants

- A constant : an identifier that holds the same value during its entire existence
- It is constant, not variable (value can't be changed)
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Primitive Data

- There are eight primitive data types in Java
- Integers:
  - byte, short, int, long
- Floating point numbers:
  - float, double
- Characters:
  - char
- Boolean values:
  - boolean

Numeric Primitive Data

- The difference between the numeric primitive types is their size and the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
`double`	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	

- An **overflow** occurs when the value being assigned to an integer variable is greater than the maximum value the variable can store.

Characters

- A `char` variable stores a single character
- Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```

- Example declarations:

```
char topGrade = 'A';
char terminator = ';', separator = ' ';
```

- Difference between a primitive character variable and a String object?

Character Sets

- An ordered list of characters, with each character corresponding to a unique number
- A char variable in Java can store any character from the Unicode character set
  - uses sixteen bits per character, allowing for 65,536 unique characters
  - is an international character set, containing symbols and characters from many world languages

Characters

- The ASCII character set is older and smaller than Unicode, but is still quite popular

- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, ...

Boolean

- 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 also be used to represent any two states, such as a light bulb being on or off

Expressions

- An expression is a combination of one or more operators and operands
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands are floating point values, then the result is a floating point value

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3   equals   4
8 / 12   equals   0
```

- The remainder operator (%) returns the remainder after dividing the first operand by the second

```
14 % 3   equals   2
8 % 12   equals   8
```

Quick Check

What are the results of the following expressions?

```
12 / 2
12.0 / 2.0
10 / 4
10 / 4.0
4 / 10
4.0 / 10
12 % 3
10 % 3
3 % 10
```

Quick Check

What are the results of the following expressions?

```
   12 / 2  =  6
12.0 / 2.0 =  6.0
   10 / 4  =  2
 10 / 4.0  =  2.5
   4 / 10  =  0
 4.0 / 10  =  0.4
   12 % 3  =  0
   10 % 3  =  1
   3 % 10  =  3
```

Operator Precedence

- Operators can be combined into larger expressions

```
result = total + count / max - offset;
```

Precedence Level	Operator	Operation	Associates
1	+	unary plus	R to L
	-	unary minus	
2	*	multiplication	L to R
	/	division	
	%	remainder	
3	+	addition	L to R
	-	subtraction	
	+	string concatenation	
4	=	assignment	R to L

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

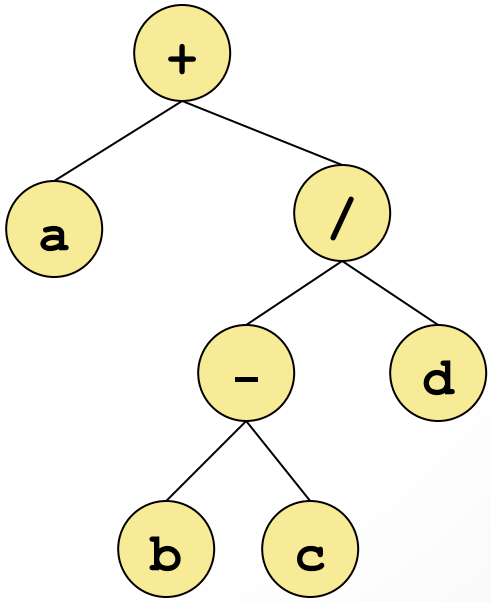
```
a / (b + c) - d % e
  2    1      4   3
```

```
a / (b * (c + (d - e)))
  4    3    2    1
```

Expression Trees

- Used to show the evaluation of a particular expression
- The operators lower in the tree have higher precedence for that expression

```
a + (b - c) / d
```



Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

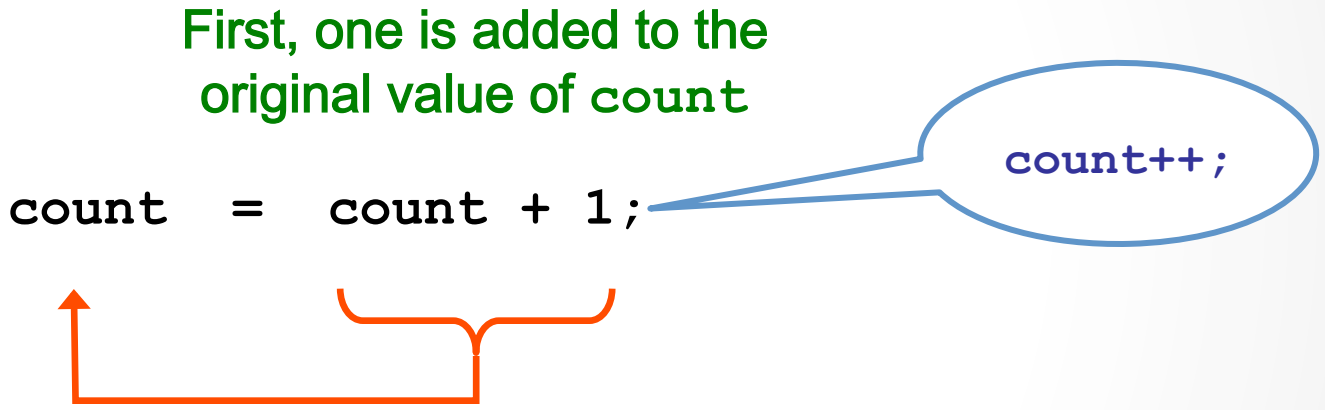
First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4       1   3     2
```

Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable



Increment and Decrement

- Postfix form:

```
count++     count--
```

- Prefix form:

```
++count     --count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Assignment Operators

- These two statements are equivalent:

```
num += count;

num = num + count;
```

- There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Example

```
result /= (total-MIN) % num;

result = result / ((total-MIN) % num);
```

- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)

Data Conversion

safest

can lose information

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

- In Java, data conversions can occur in three ways:
  - assignment conversion
  - promotion
  - casting

Assignment Conversion

- Assignment conversion occurs when a value of one type is assigned to a variable of another
- Example:

```
int dollars = 20;
double money = dollars;
```

- Only widening conversions can happen via assignment
- Note that the value or type of dollars did not change

Promotion

- Promotion happens automatically when operators in expressions convert their operands
- Example:

```
int count = 12;
double sum = 490.27;
result = sum / count;
```

- The value of count is converted to a floating point value to perform the division calculation

Casting

- Casting is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;
float result = (float) total / 6;
```

- Without the cast, the fractional part of the answer would be lost