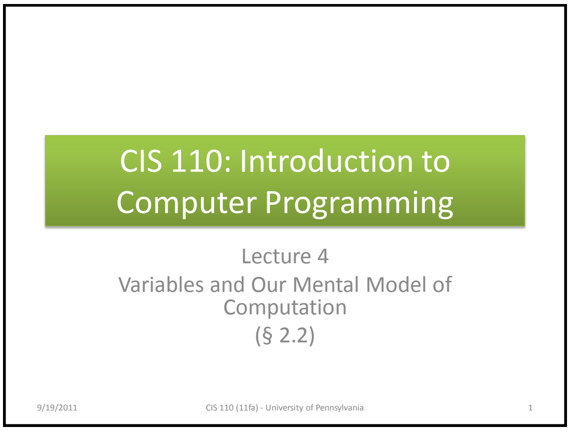
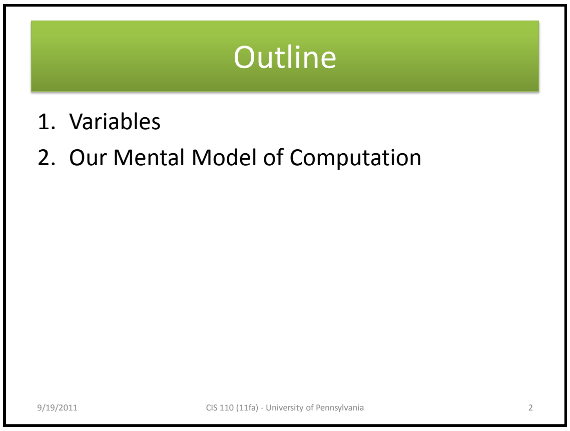


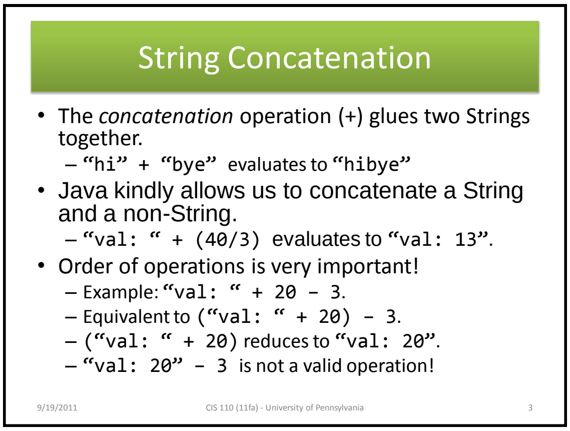
CIS 110: Introduction to Computer Programming

Lecture 4
Variables and Our Mental Model of Computation
(§ 2.2)



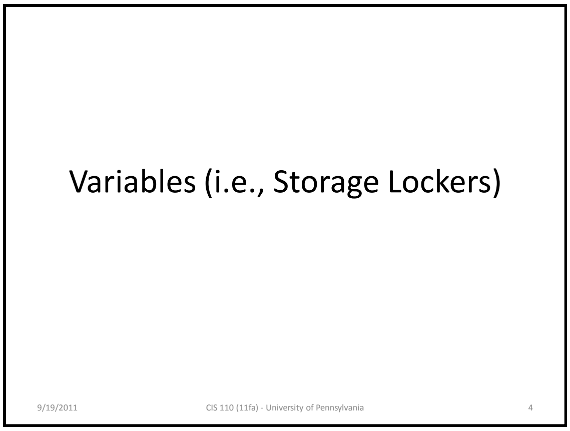
Outline

1. Variables
2. Our Mental Model of Computation

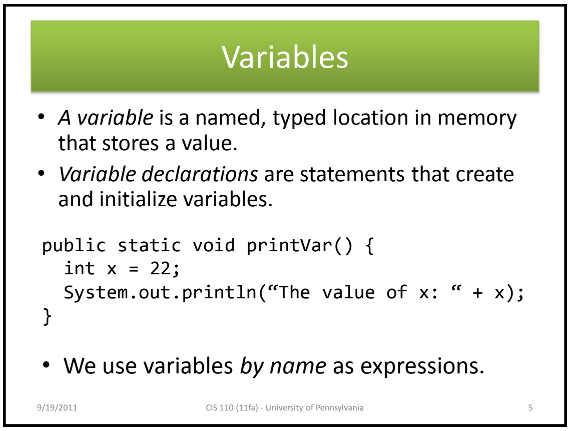


String Concatenation

- The *concatenation* operation (+) glues two Strings together.
  - “hi” + “bye” evaluates to “hibye”
- Java kindly allows us to concatenate a String and a non-String.
  - “val: “ + (40/3) evaluates to “val: 13”.
- Order of operations is very important!
  - Example: “val: “ + 20 – 3.
  - Equivalent to (“val: “ + 20) – 3.
  - (“val: “ + 20) reduces to “val: 20”.
  - “val: 20” – 3 is not a valid operation!



Variables (i.e., Storage Lockers)

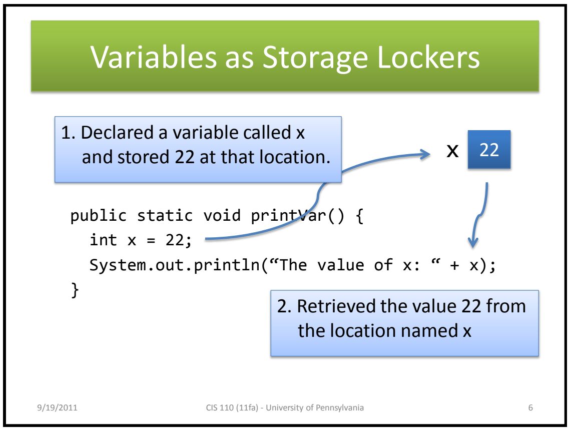


Variables

- *A variable* is a named, typed location in memory that stores a value.
- *Variable declarations* are statements that create and initialize variables.

```
public static void printVar() {
  int x = 22;
  System.out.println(“The value of x: “ + x);
}
```

- We use variables *by name* as expressions.

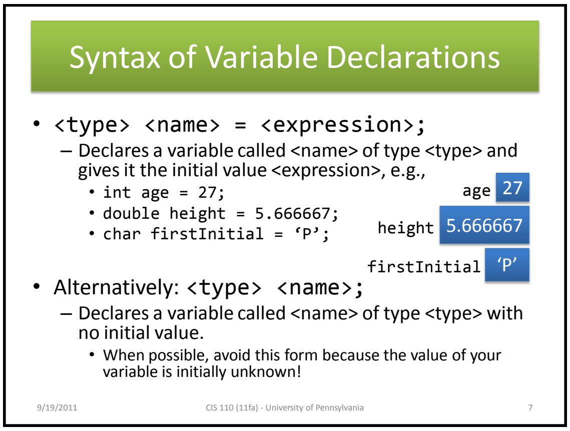


Variables as Storage Lockers

1. Declared a variable called x and stored 22 at that location.

```
public static void printVar() {
  int x = 22;
  System.out.println(“The value of x: “ + x);
}
```

2. Retrieved the value 22 from the location named x

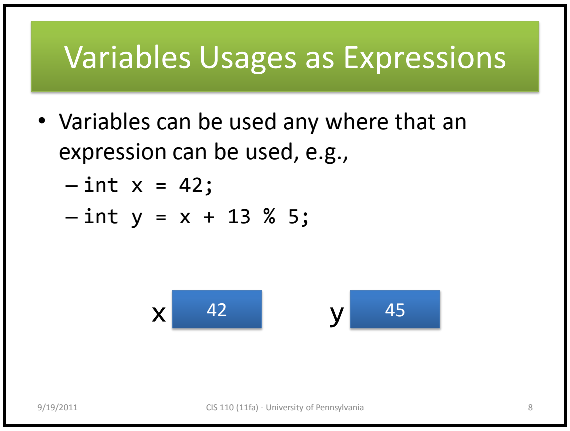


Syntax of Variable Declarations

- `<type> <name> = <expression>;`
  - Declares a variable called <name> of type <type> and gives it the initial value <expression>, e.g.,
    - `int age = 27;`
    - `double height = 5.666667;`
    - `char firstInitial = ‘P’;`

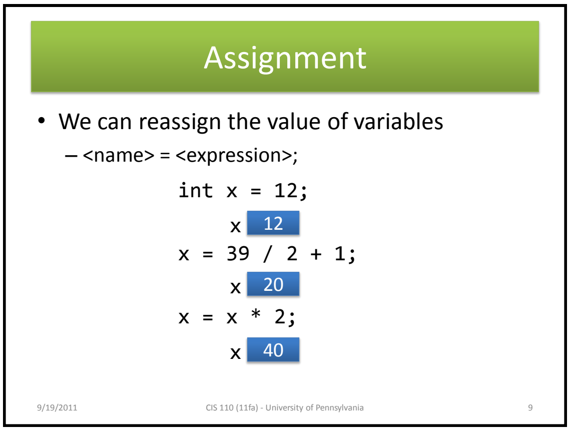
age 27
height 5.666667
firstInitial ‘P’

- Alternatively: `<type> <name>;`
  - Declares a variable called <name> of type <type> with no initial value.
    - When possible, avoid this form because the value of your variable is initially unknown!



Variables Usages as Expressions

- Variables can be used any where that an expression can be used, e.g.,
  - `int x = 42;`
  - `int y = x + 13 % 5;`



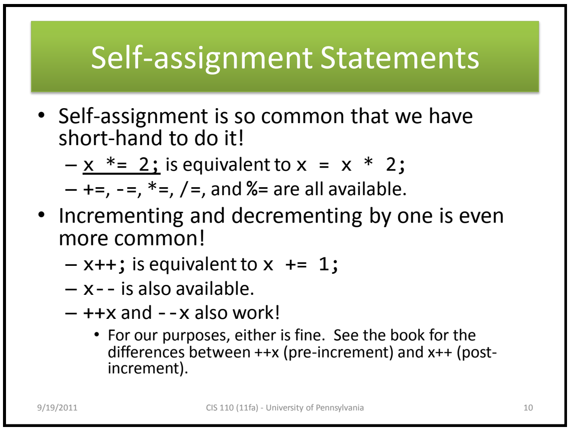
Assignment

- We can reassign the value of variables
  - <name> = <expression>;

```
int x = 12;
```

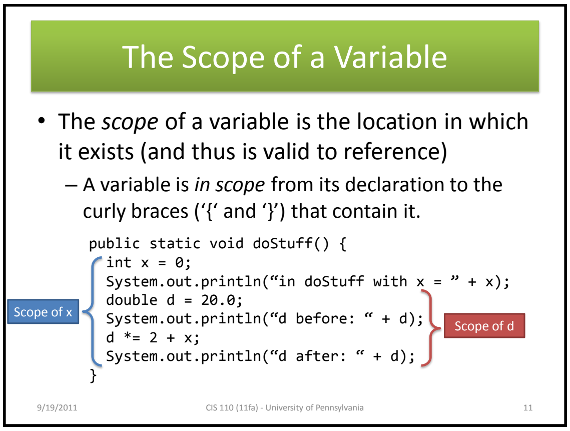
```
x = 39 / 2 + 1;
```

```
x = x * 2;
```



Self-assignment Statements

- Self-assignment is so common that we have short-hand to do it!
  - `x *= 2;` is equivalent to `x = x * 2;`
  - +=, -=, *=, /=, and %= are all available.
- Incrementing and decrementing by one is even more common!
  - `x++;` is equivalent to `x += 1;`
  - `x--` is also available.
  - `++x` and `--x` also work!
    - For our purposes, either is fine. See the book for the differences between ++x (pre-increment) and x++ (post-increment).

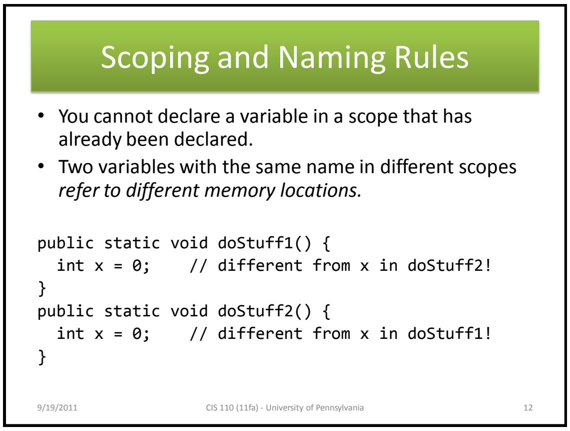


The Scope of a Variable

- The *scope* of a variable is the location in which it exists (and thus is valid to reference)
  - A variable is *in scope* from its declaration to the curly braces (‘{‘ and ‘}’) that contain it.

```
public static void doStuff() {
  int x = 0;
  System.out.println(“in doStuff with x = ” + x);
  double d = 20.0;
  System.out.println(“d before: “ + d);
  d *= 2 + x;
  System.out.println(“d after: “ + d);
}
```

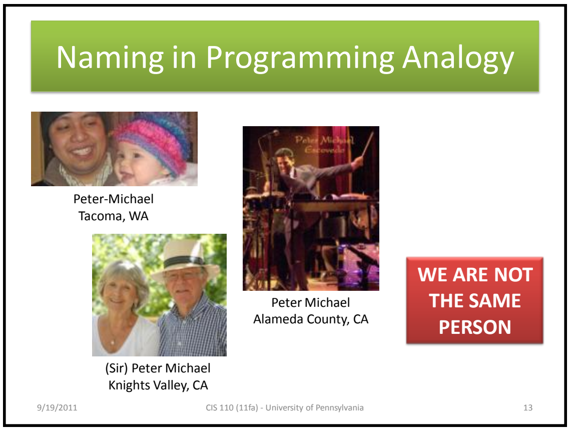
Scope of x
Scope of d



Scoping and Naming Rules

- You cannot declare a variable in a scope that has already been declared.
- Two variables with the same name in different scopes *refer to different memory locations.*

```
public static void doStuff1() {
  int x = 0;    // different from x in doStuff2!
}
public static void doStuff2() {
  int x = 0;    // different from x in doStuff1!
}
```



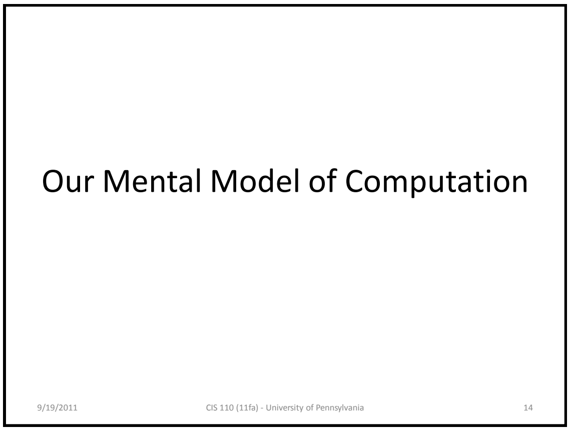
Naming in Programming Analogy

Peter-Michael
Tacoma, WA

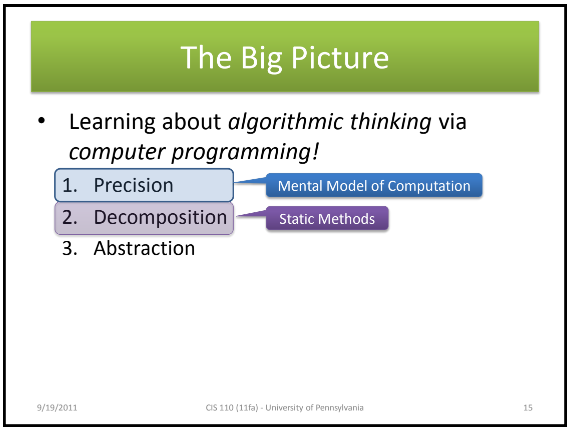
Peter Michael
Alameda County, CA

(Sir) Peter Michael
Knights Valley, CA

WE ARE NOT THE SAME PERSON



Our Mental Model of Computation

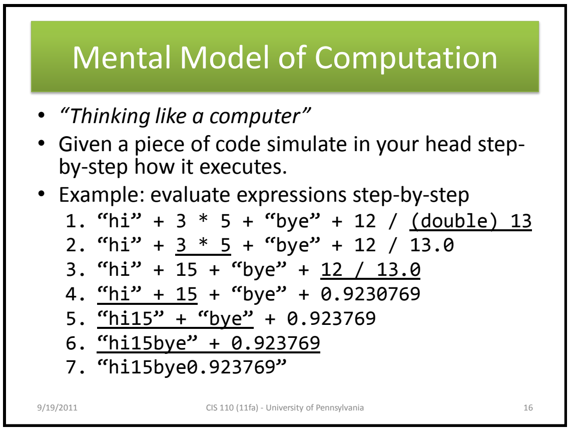


The Big Picture

- Learning about *algorithmic thinking* via *computer programming!*
  1. Precision
  2. Decomposition
  3. Abstraction

Mental Model of Computation

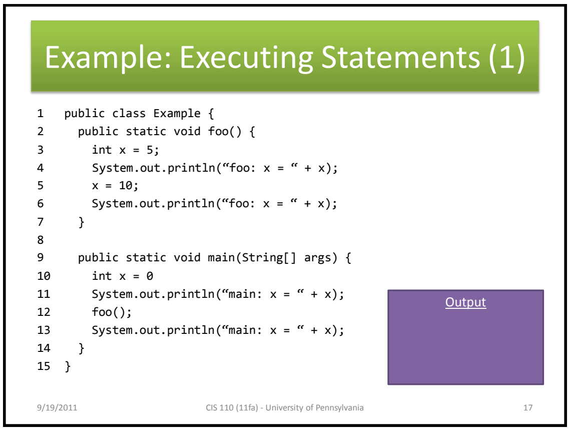
Static Methods



Mental Model of Computation

- *“Thinking like a computer”*
- Given a piece of code simulate in your head step-by-step how it executes.
- Example: evaluate expressions step-by-step

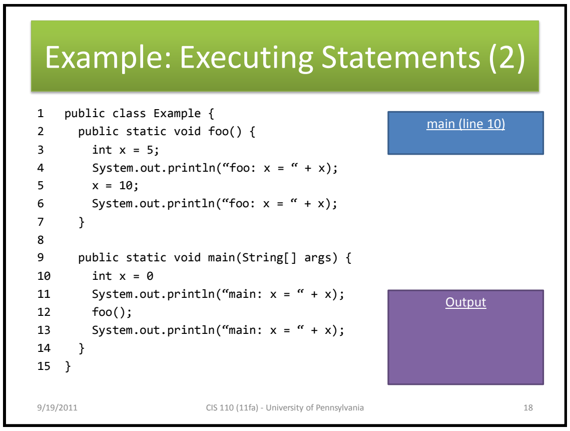
```
1. “hi” + 3 * 5 + “bye” + 12 / (double) 13
2. “hi” + 3 * 5 + “bye” + 12 / 13.0
3. “hi” + 15 + “bye” + 12 / 13.0
4. “hi” + 15 + “bye” + 0.9230769
5. “hi15” + “bye” + 0.923769
6. “hi15bye” + 0.923769
7. “hi15bye0.923769”
```



Example: Executing Statements (1)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

Output

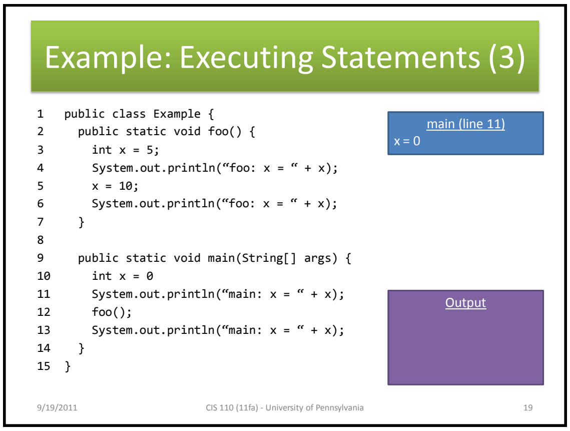


Example: Executing Statements (2)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 10)

Output

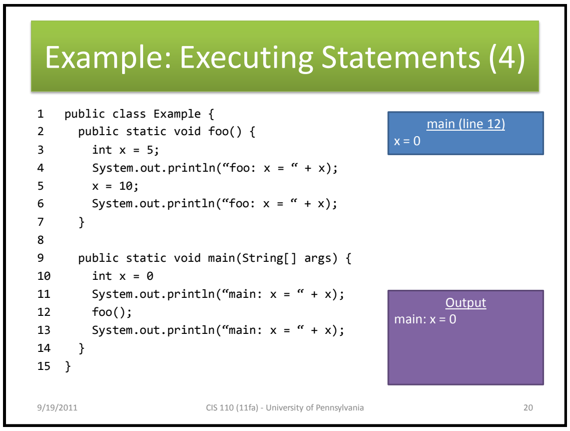


Example: Executing Statements (3)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 11)
x = 0

Output

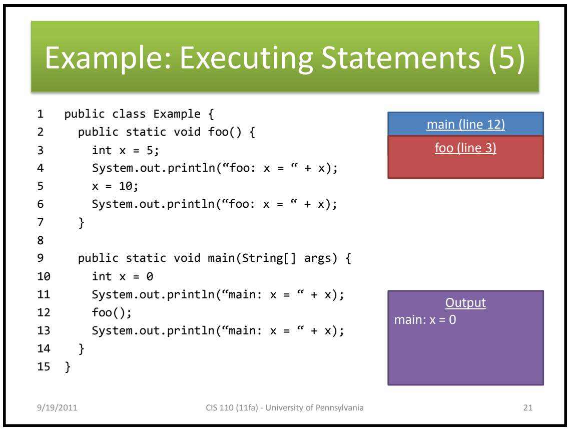


Example: Executing Statements (4)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)
x = 0

Output
main: x = 0



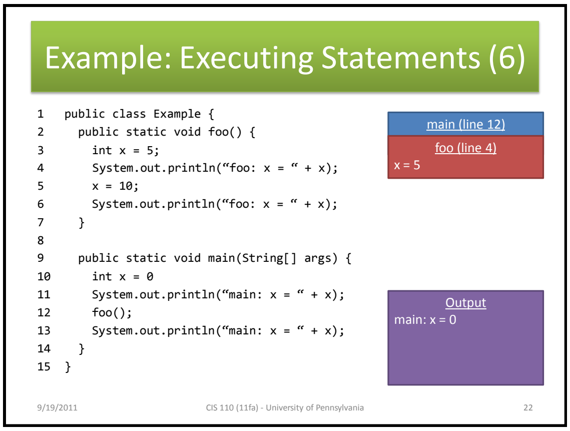
Example: Executing Statements (5)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)

foo (line 3)

Output
main: x = 0



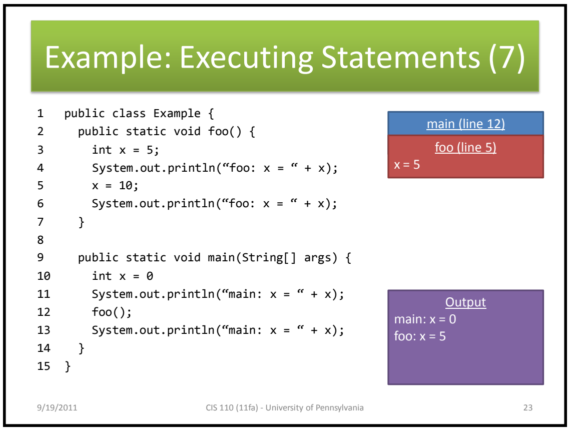
Example: Executing Statements (6)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)

foo (line 4)
x = 5

Output
main: x = 0



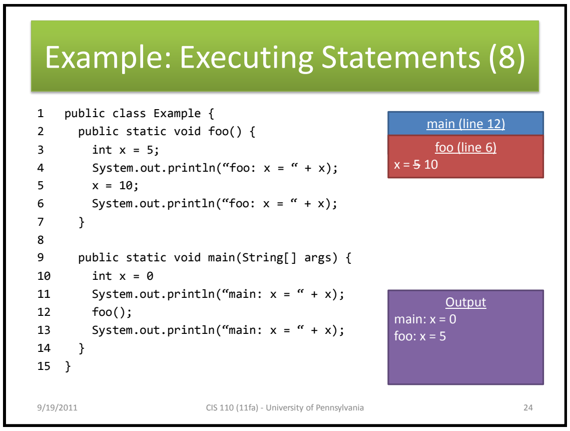
Example: Executing Statements (7)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)

foo (line 5)
x = 5

Output
main: x = 0
foo: x = 5



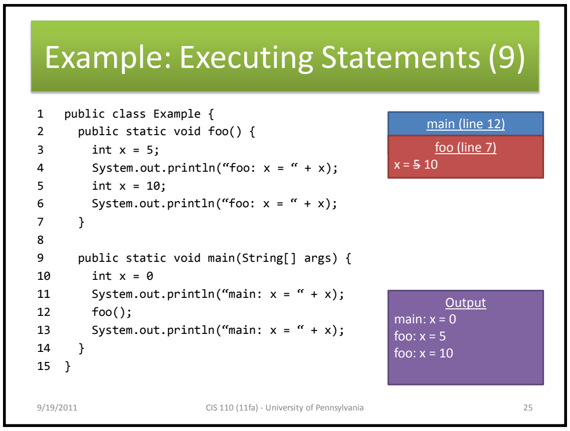
Example: Executing Statements (8)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)

foo (line 6)
x = ~~5~~ 10

Output
main: x = 0
foo: x = 5



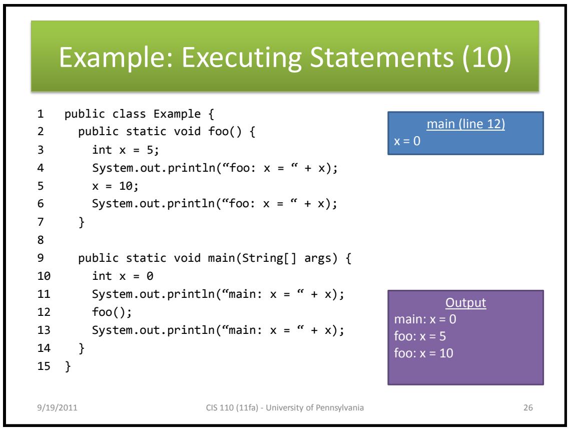
Example: Executing Statements (9)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       int x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)

foo (line 7)
x = ~~5~~ 10

Output
main: x = 0
foo: x = 5
foo: x = 10

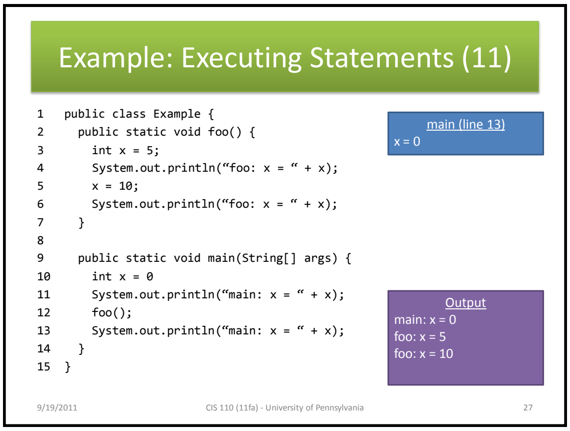


Example: Executing Statements (10)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 12)
x = 0

Output
main: x = 0
foo: x = 5
foo: x = 10

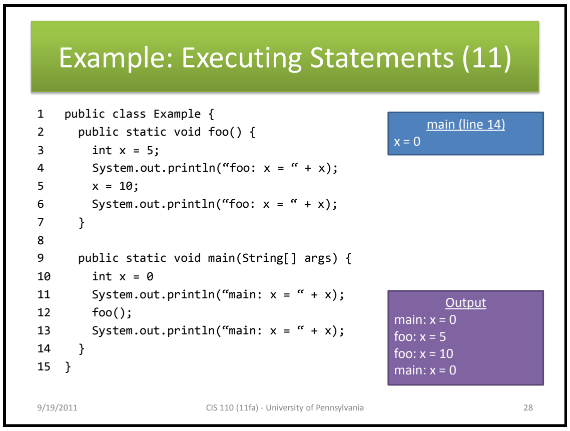


Example: Executing Statements (11)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 13)
x = 0

Output
main: x = 0
foo: x = 5
foo: x = 10

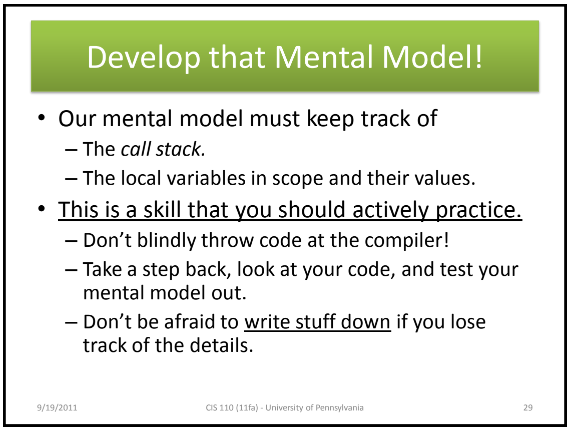


Example: Executing Statements (11)

```
1   public class Example {
2     public static void foo() {
3       int x = 5;
4       System.out.println(“foo: x = “ + x);
5       x = 10;
6       System.out.println(“foo: x = “ + x);
7     }
8
9     public static void main(String[] args) {
10      int x = 0
11      System.out.println(“main: x = “ + x);
12      foo();
13      System.out.println(“main: x = “ + x);
14    }
15  }
```

main (line 14)
x = 0

Output
main: x = 0
foo: x = 5
foo: x = 10
main: x = 0



Develop that Mental Model!

- Our mental model must keep track of
  - The *call stack.*
  - The local variables in scope and their values.
- This is a skill that you should actively practice.
  - Don’t blindly throw code at the compiler!
  - Take a step back, look at your code, and test your mental model out.
  - Don’t be afraid to write stuff down if you lose track of the details.