CSE 11
Accelerated Intro to Programming
Lecture 3

Greg Miranda, Spring 2021

This lecture is being recorded

Announcements

- Quiz 3 due Monday @ 8am
- Survey 1 due tonight @ 11:59pm → required as engagement
- PA1 due Wednesday @ 11:59pm

PA0.5 → resubmission → 100%

↳ late adds

Program Steps

```
class Example {
  int x = 3 + 2;
  int y = this.x * 4;
}
```

⇒

```
class Example {
  int x = 5;
  int y = this.x *4;
}
```

→ 5 is a value

⇓

```
class Example {
    int x = 5;
    int y = 5 * 4;
}
```

⇓

```
class Example {
    int x = 5
    int y = 20;
}
```

```
new Example:(
    this.x = 5
    this.y = 20
)
```

Expressions

$$\overline{op1 + op2}^{\text{binary operator}}$$

$-$

$*$

$/$

- int x = 3 + 2;
  - 3 + 2
    - Arithmetic expression
    - Binary operator expression
- int y = this.x * 4;
  - this.x
    - Field access expression → style
  - this.x * 4
    - Arithmetic expression where the left-hand operand is a field access expression

this.x

Methods

- New class – MethodExample
- In programming, we often want to describe a computation once
  - Then reuse it on different numbers, or different values
  - Write once, use it over and over again
- Example:
  - Take two numbers and add up their squares → $w^2 \rightarrow w * w$
    - int sos1 = 3 * 3 + 5 * 5;
    - int sos2 = 4 * 4 + 7 * 7;

- Define a method to do the same thing

```
int sumSquares(int n, int m) {
  return n * n + m * m;
}
```

- Vocabulary:
  - Method definition
  - Parameters
  - Method body
    - return keyword

- Running it…
  - Method definition doesn't change what prints out or any of the fields
  - Run command – only prints out the values of the fields
- Can use sumSquares() to do the calculation
  - int ans1 = this.sumSquares(3, 5);
  - int ans2 = this.sumSquares(4, 7);
- Vocabulary:
  - Called the method
  - Arguments

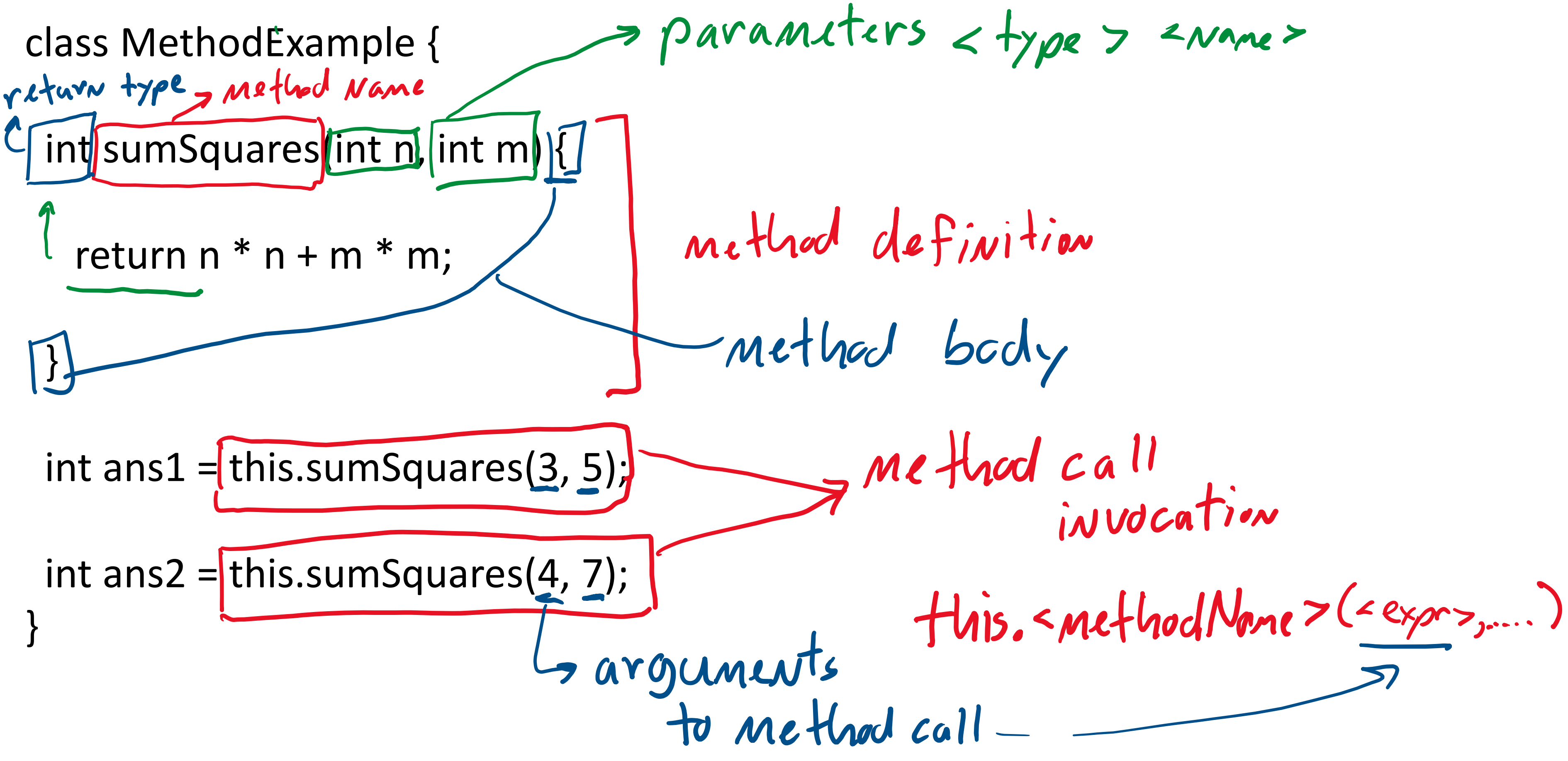
- Methods: one of the building blocks for building programs
  - Not just useful for arithmetic
  - Useful for many more things
- Why do we care about methods?
  - Methods give us a centralized place to write a calculation ✓
    - Change in one place, every place that uses the method will see that update
  - As program gets large:
    - Might have 100s of places where we want to use a formula or calculation
      - Update them all by changing one place
  - Methods are self documenting – with meaningful names

↑

descriptive (style)

↳ camelCase

↑ ↑



```
class MethodExample {
  int sumSquares(int n, int m) {

    return n * n + m * m;

  }

  int ans1 = this.sumSquares(3, 5);

  int ans2 = this.sumSquares(4, 7);
}
```

arguments → copied into the parameters

```
int / string / ...
↓
<type>  <methodName> ([<parameter>, ...]) {
    [<body>]

    return <expr>;
}
```

optional

<type> → void → return nothing → return;

↑

Process

```
class MethodExample {
  int sumSquares(int n, int m) {
    return n * n + m * m;
  }          34
  int ans1 = this.sumSquares(3, 5);
  int ans2 = this.sumSquares(4, 7);
}            65
```

method call

sumSquares(n=3, m=5) {

return 3*3 + 5*5;

}

9 + 25

$\boxed{34}$

method call

sumSquares(n=4, m=7) {

return 4*4 + 7*7;

16 + 49

$\boxed{65}$

new MethodExample!!(

this.ans1 = 34

this.ans2 = 65)

```
                                     3          5
String sumSquares2 ( int n , int m ) {

        return "" + (n*n + m * m)
}                    3 3   5   5
                      9  + 25
                         34
                         ‾‾
    "34"
```

order of arguments matters

power (5, 2) → 5^2 → 25

power (2, 5) → 2^5 → 32