University of Wisconsin
LA CROSSE
Computer Science

Week 03:
Memory Management

CS 220: Software Design II — D. Mathias

Computer Memory

memory is the physical components of a computer that store data

All components of a program need to be stored in memory

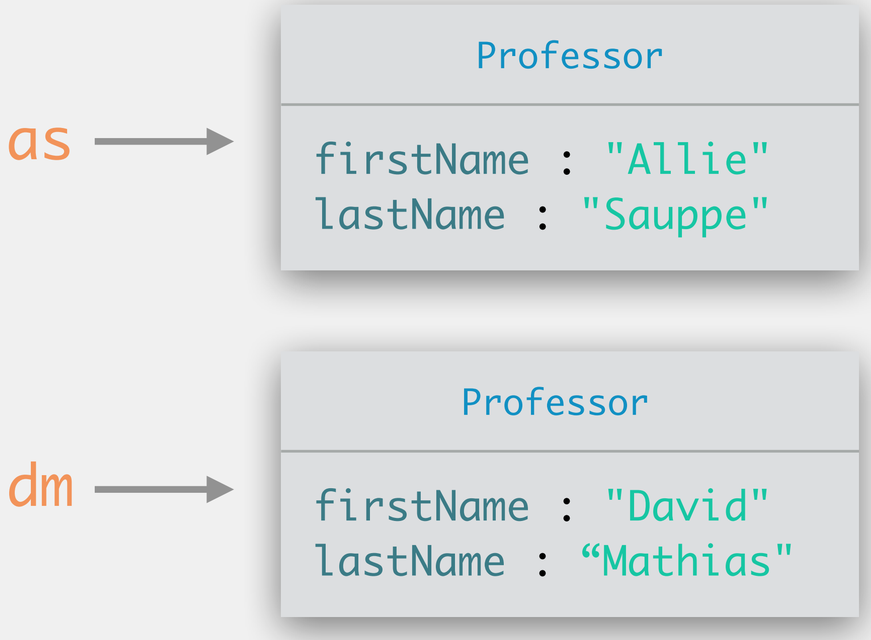
- the program itself
- values of variables
- location of the next program instruction to execute

memory management determines where data related to the execution of a program should be stored

Example: Memory Management

```
Professor as = new Professor("Sauppe", "Allie");

Professor dm = new Professor("Mathias", “David");

overwrite(as, dm);
System.out.println(dm);
System.out.println(as);
```

```
public static void overwrite(Professor p1, Professor p2) {
    p1.lastName = p2.lastName;
}
```

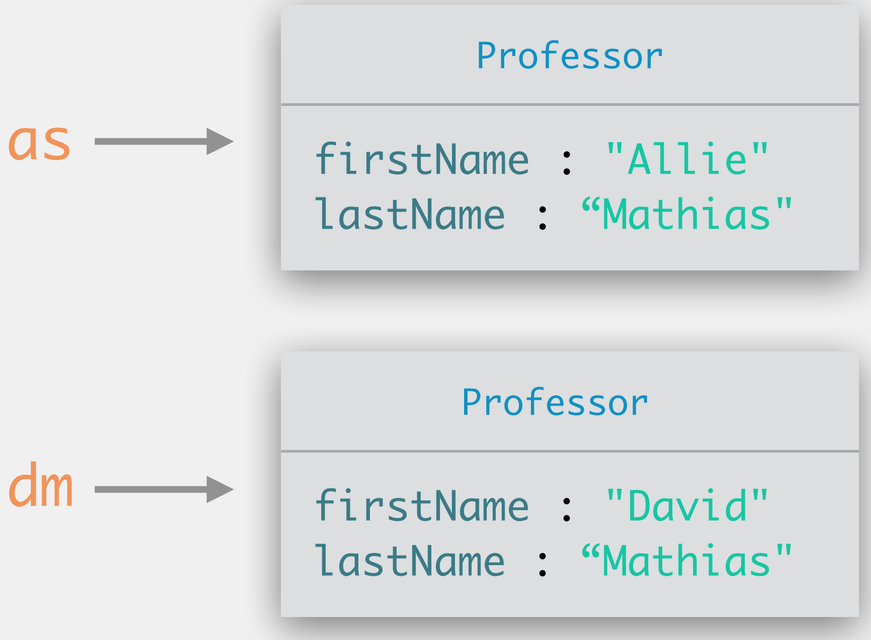


Example: Memory Management

```
Professor as = new Professor("Sauppe", "Allie");

Professor dm = new Professor("Mathias", “David");

overwrite(as, dm);
System.out.println(dm);
System.out.println(as);
```

```
public static void overwrite(Professor p1, Professor p2) {
    p1.lastName = p2.lastName;
}
```

```
David Mathias
Allie Mathias
```

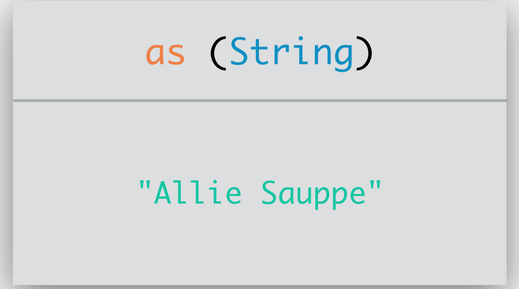


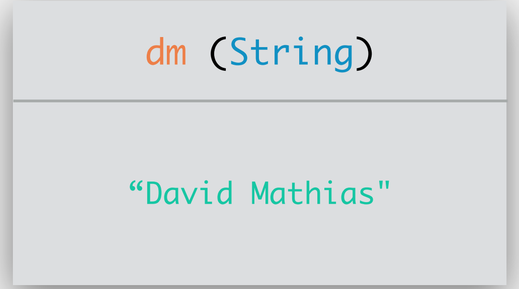
Example: Memory Management

```
String as = "Allie Sauppe";

String dm = “David Mathias";

overwrite(as, dm);
System.out.println(dm);
System.out.println(as);
```

```
public static void overwrite(String s1, String s2) {
    s1 = s2;
}
```

```
David Mathias
Allie Sauppe
```

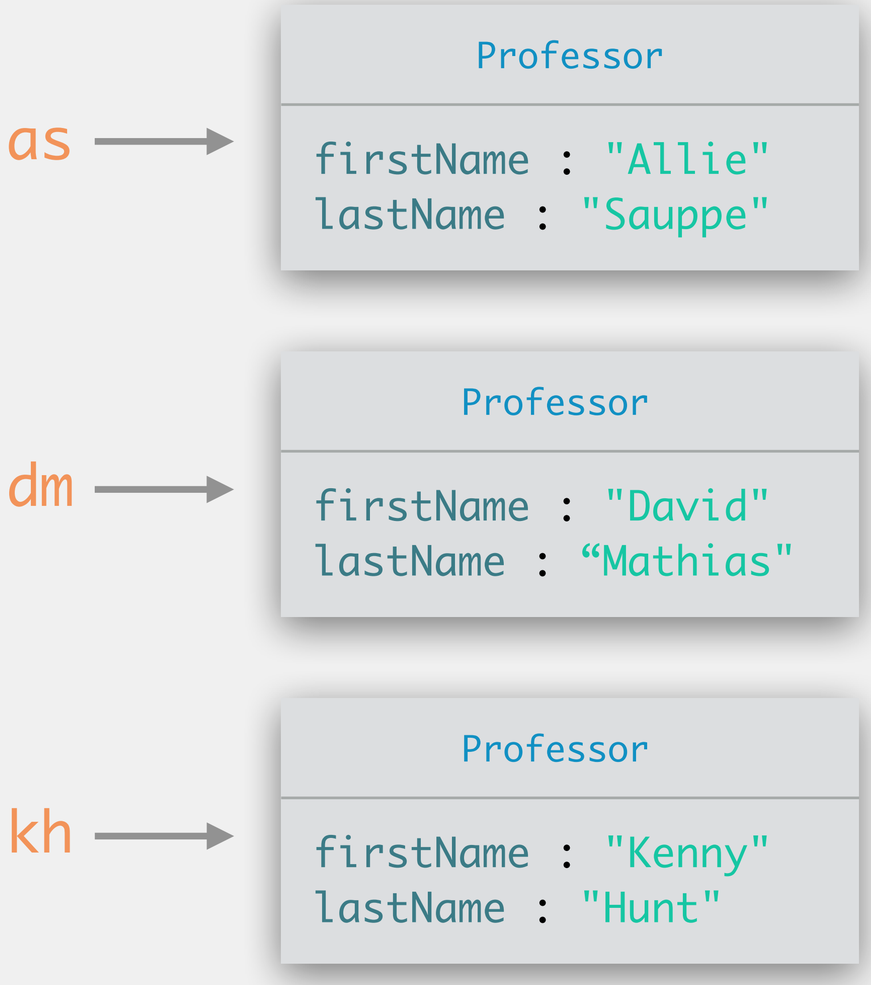




Example: Memory Management

```
Professor as = new Professor("Sauppe", "Allie");
Professor kh = new Professor("Hunt", "Kenny");
Professor dm = new Professor("Mathias", “David");
overwrite(kh, dm);
overwrite(as, dm);
System.out.println(dm);
System.out.println(as);
```

```
public static void overwrite(Professor p1, Professor p2) {
    p1.lastName = p2.lastName;
}
```

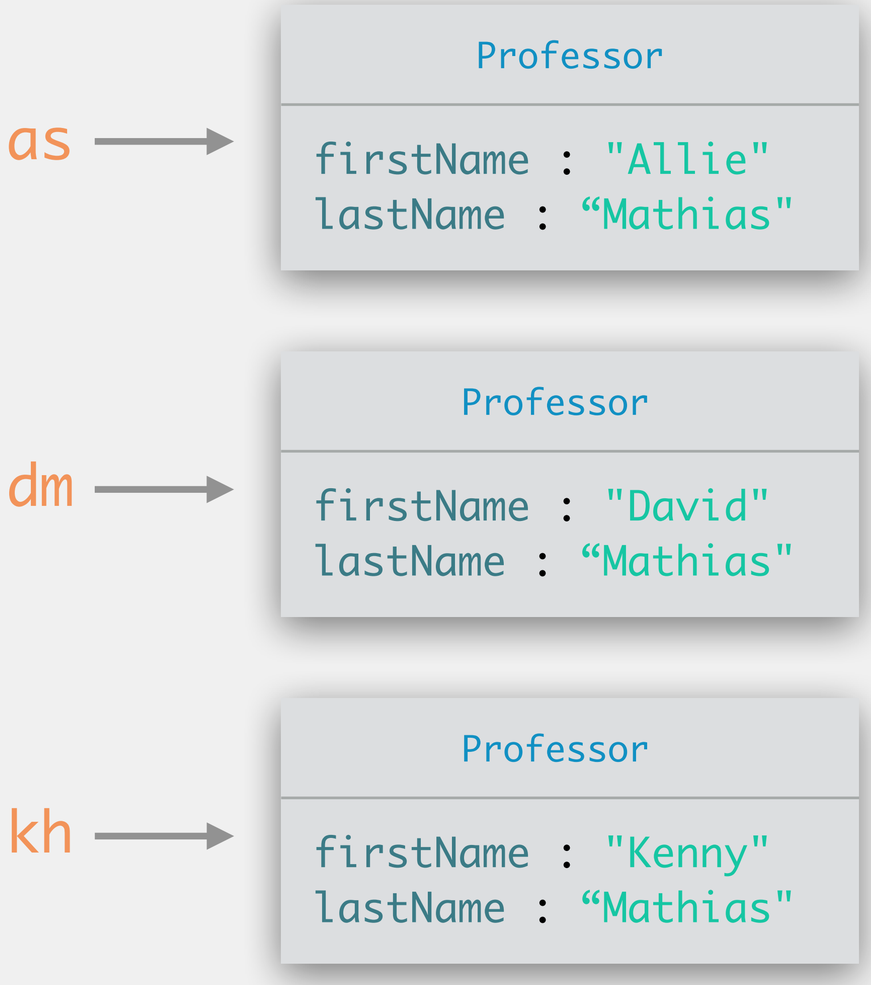


Example: Memory Management

```
Professor as = new Professor("Sauppe", "Allie");
Professor kh = new Professor("Hunt", "Kenny");
Professor dm = new Professor("Mathias", “David");
overwrite(kh, dm);
overwrite(as, dm);
System.out.println(dm);
System.out.println(as);
```

```
public static void overwrite(Professor p1, Professor p2) {
    p1.lastName = p2.lastName;
}
```

```
David Mathias
Allie Mathias
```



Memory Management Components

stack: tracks method calls, where the program execution should continue when the current call ends, and values of local primitive variables
(aka *static memory*)

heap: stores global variables, and objects
(aka *dynamic memory*)

Primitive Data Types in Java

Integer Numeric Types (can only be whole numbers)

`byte`	1 byte	-128	through	127
`short`	2 bytes	-32678	through	32677
`int`	4 bytes	-2147483648	through	2147483647
`long`	8 bytes	-9223372036854775808	through	9223372036854775807

Decimal Numeric Types (can be whole or decimal numbers)

`float`	4 bytes	7 decimal digits of accuracy
`double`	8 bytes	15 decimal digits of accuracy

Character Type

`char`	2 bytes	any keyboard character

Logical Type

`boolean`	1 byte	`true` or `false`

The Heap

Assigns memory locations to objects/ global variables

- need to know the size (in bytes) of the object
- heap size bounded by computer's limitations

Slower to access than the stack

```
                Professor  22 bytes
firstName : "Allie" 10 bytes
lastName : "Sauppe" 12 bytes
```

N.B.: a single char is 2 bytes, so we can count the number of characters and multiply

The Stack

```
<frame5>
  <var1>
  <var2>
```

```
<frame4>
  <var1>
  <var2>
```

```
<frame3>
  <var1>
  <var2>
```

```
<frame2>
  <var1>
  <var2>
```

```
<frame1>
  <var1>
  <var2>
```

Tracks what methods have been “suspended”, plus the current method

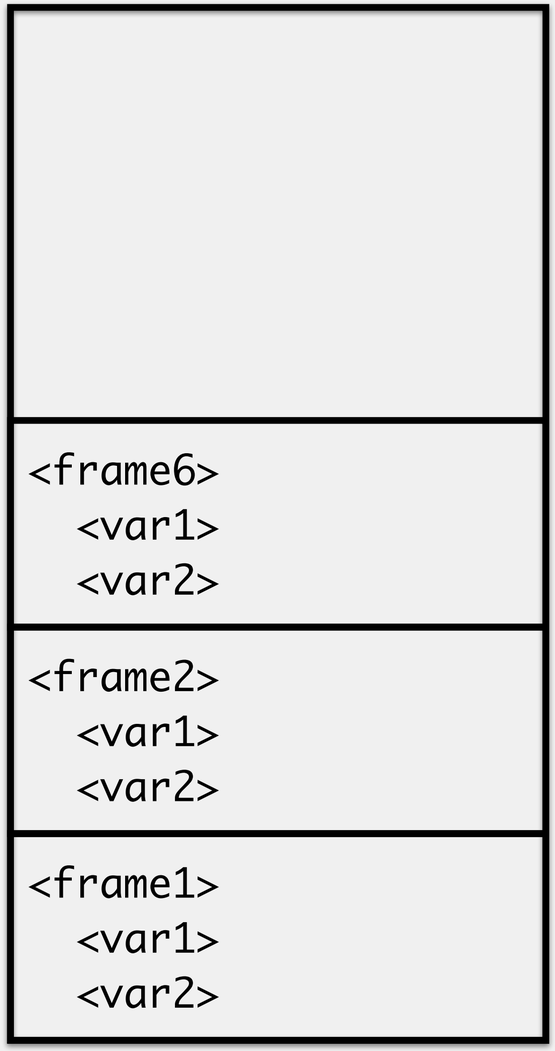
- where in the method suspension occurred
- what the value of any primitive variables were

Each method is stored in a *frame*

Frames are stored *last in, first out* (*LIFO*)

- addition of a frame is always done onto the top
- removal is always done from the top (hence, LIFO)

The Stack



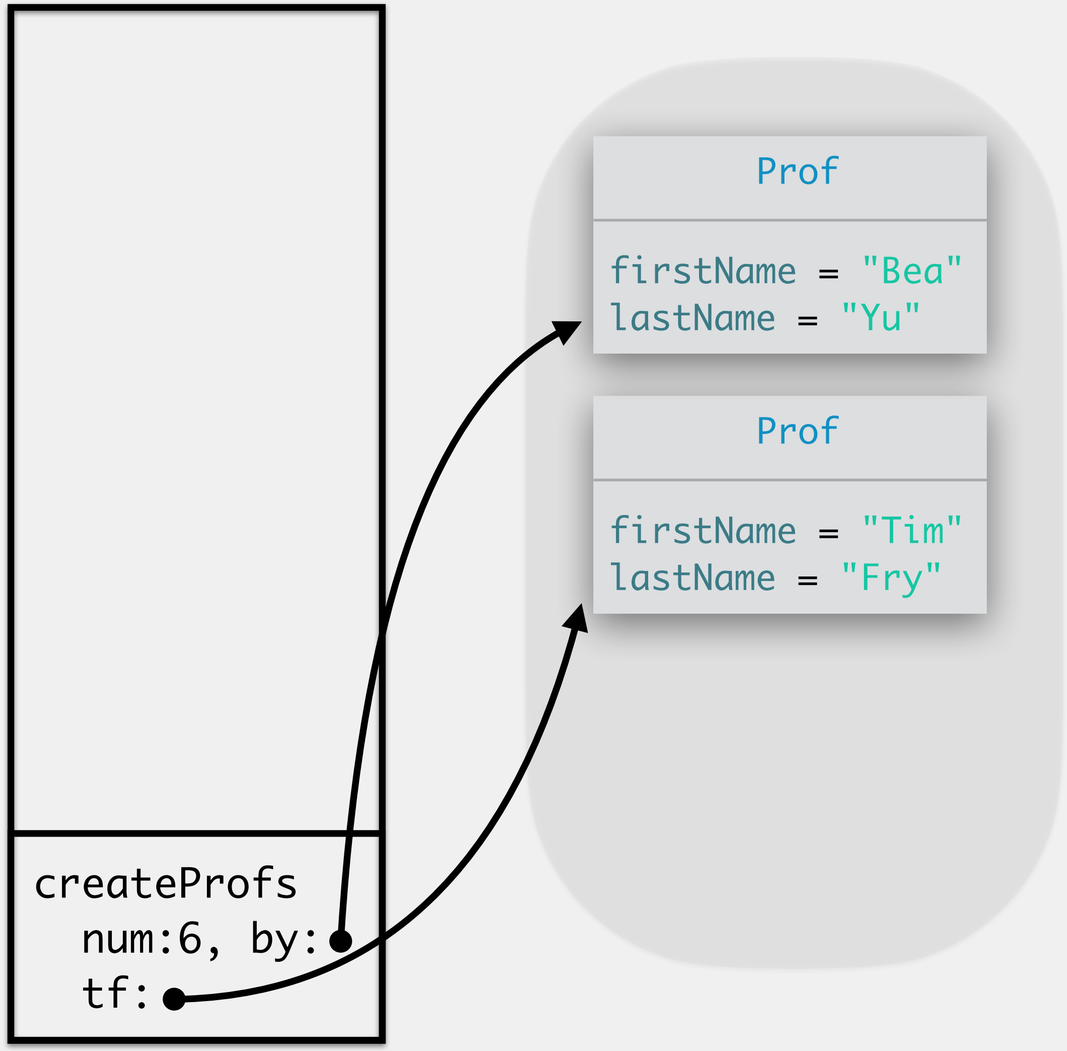
Bounded in size by the compiler

- can be adjusted
- filling the stack produces a `StackOverflowError`

Faster for data access than the heap

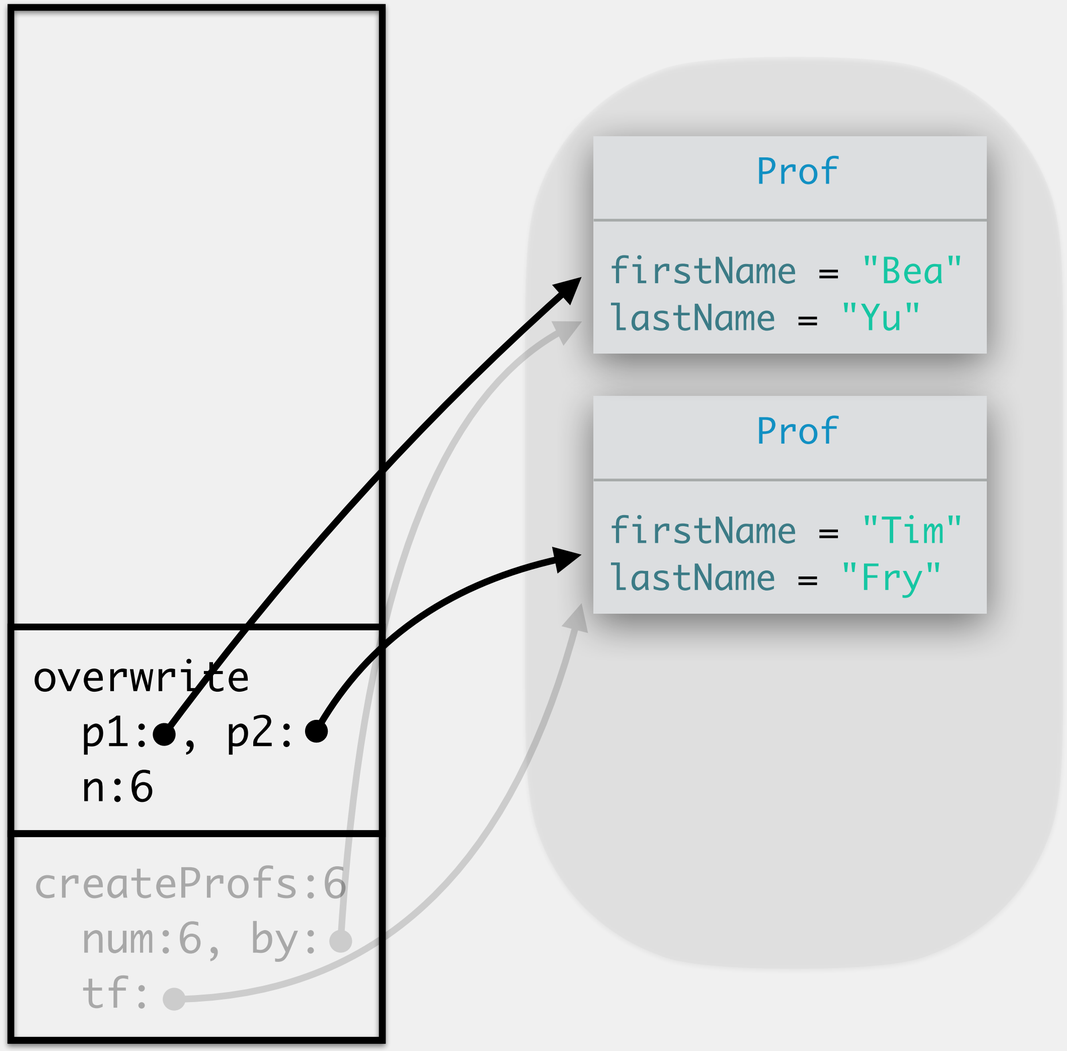
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



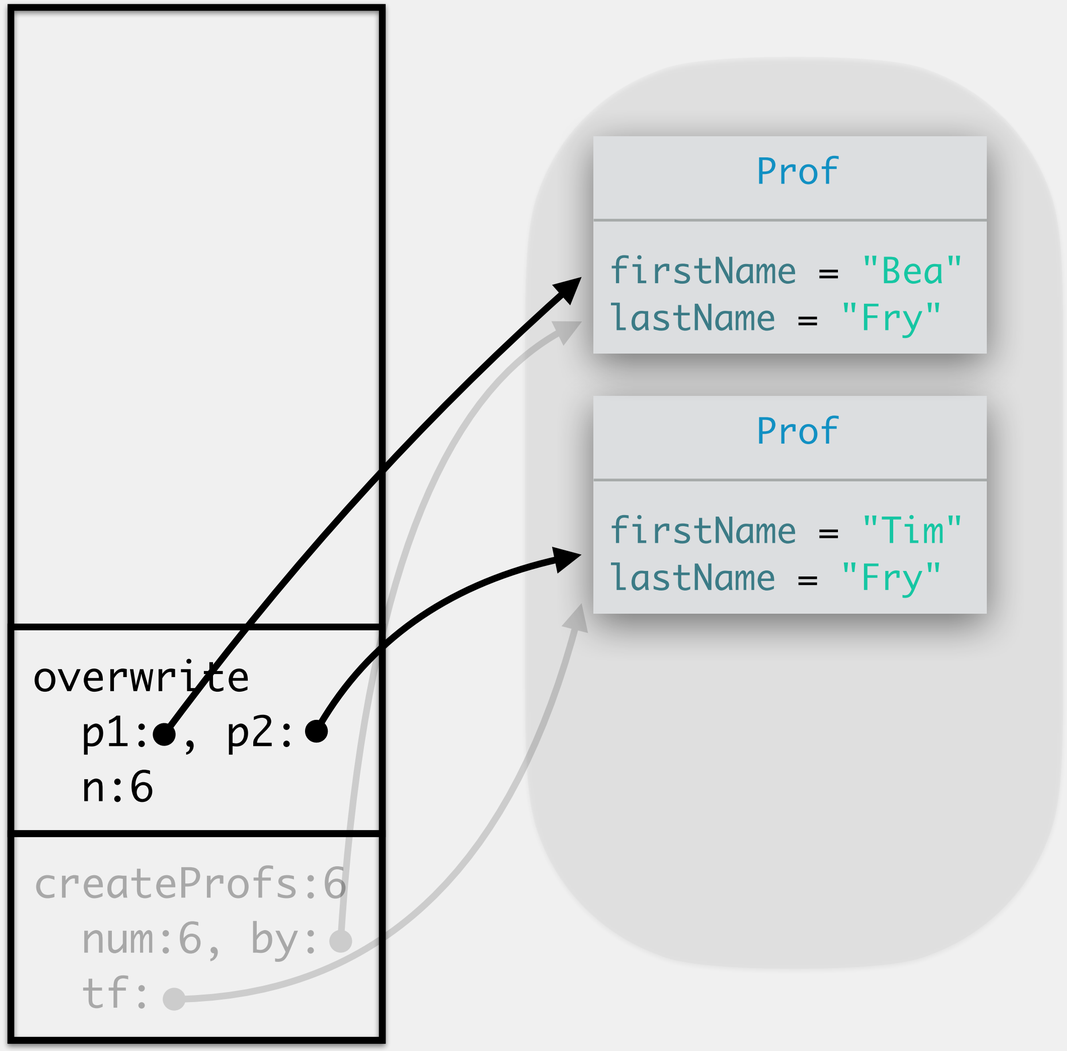
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12 >    p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



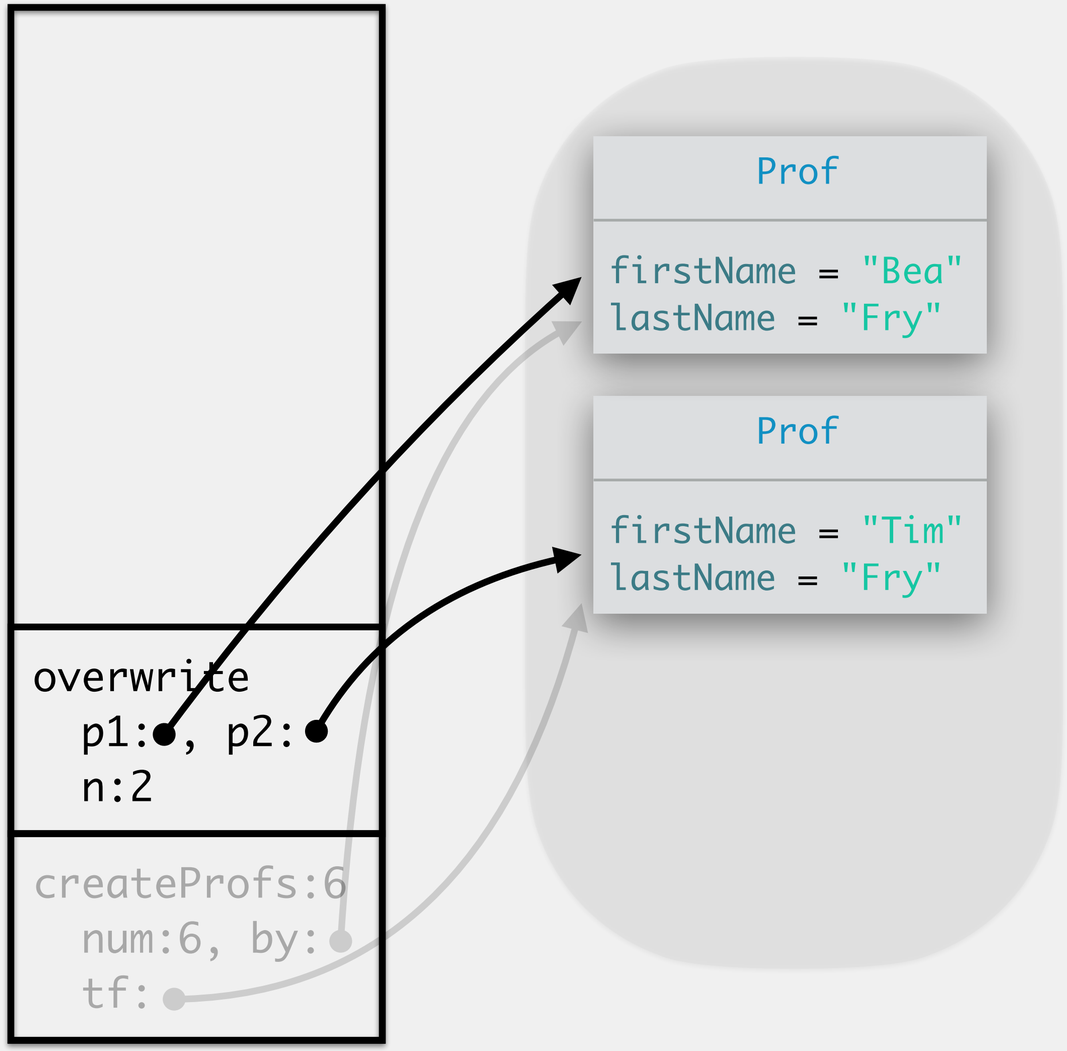
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13 >    n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



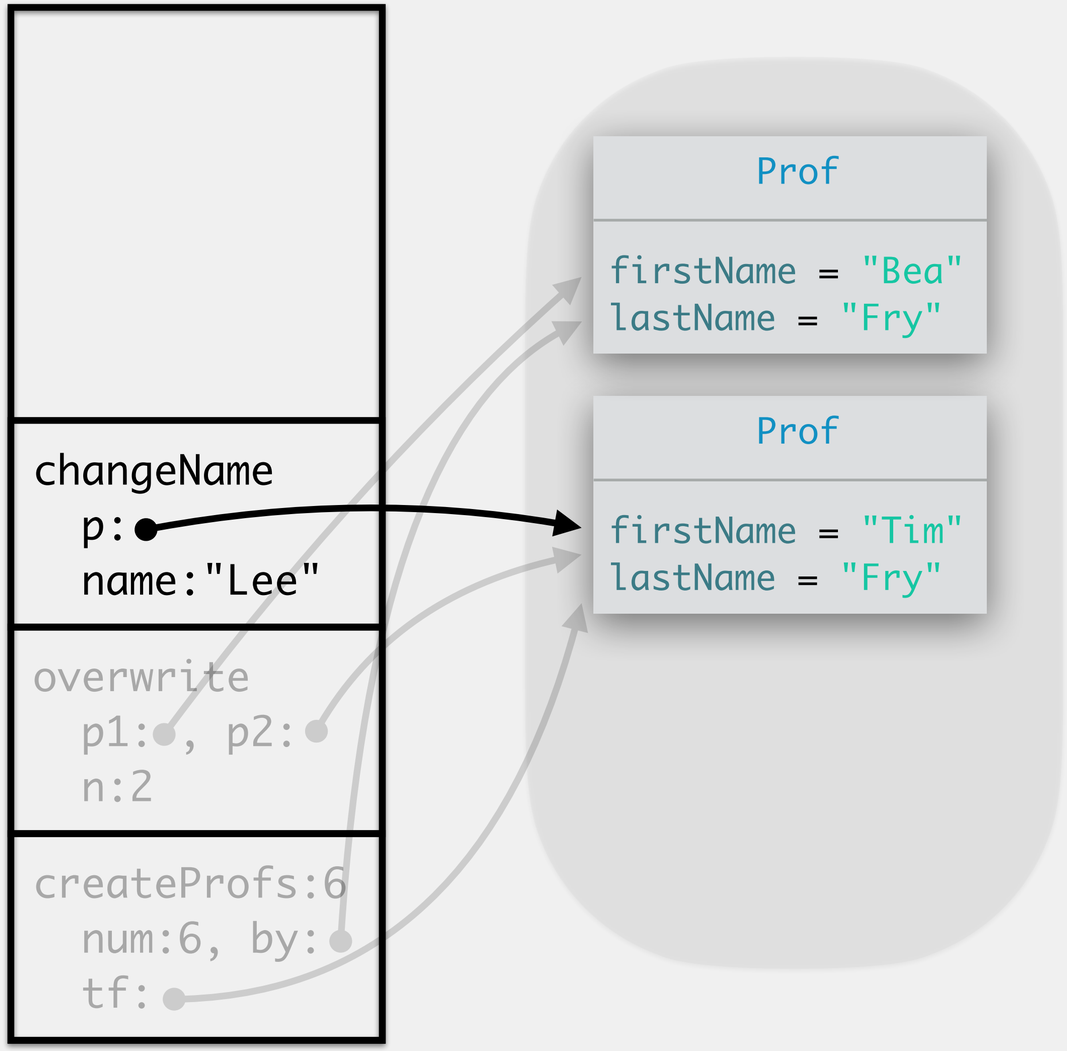
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14 >    changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



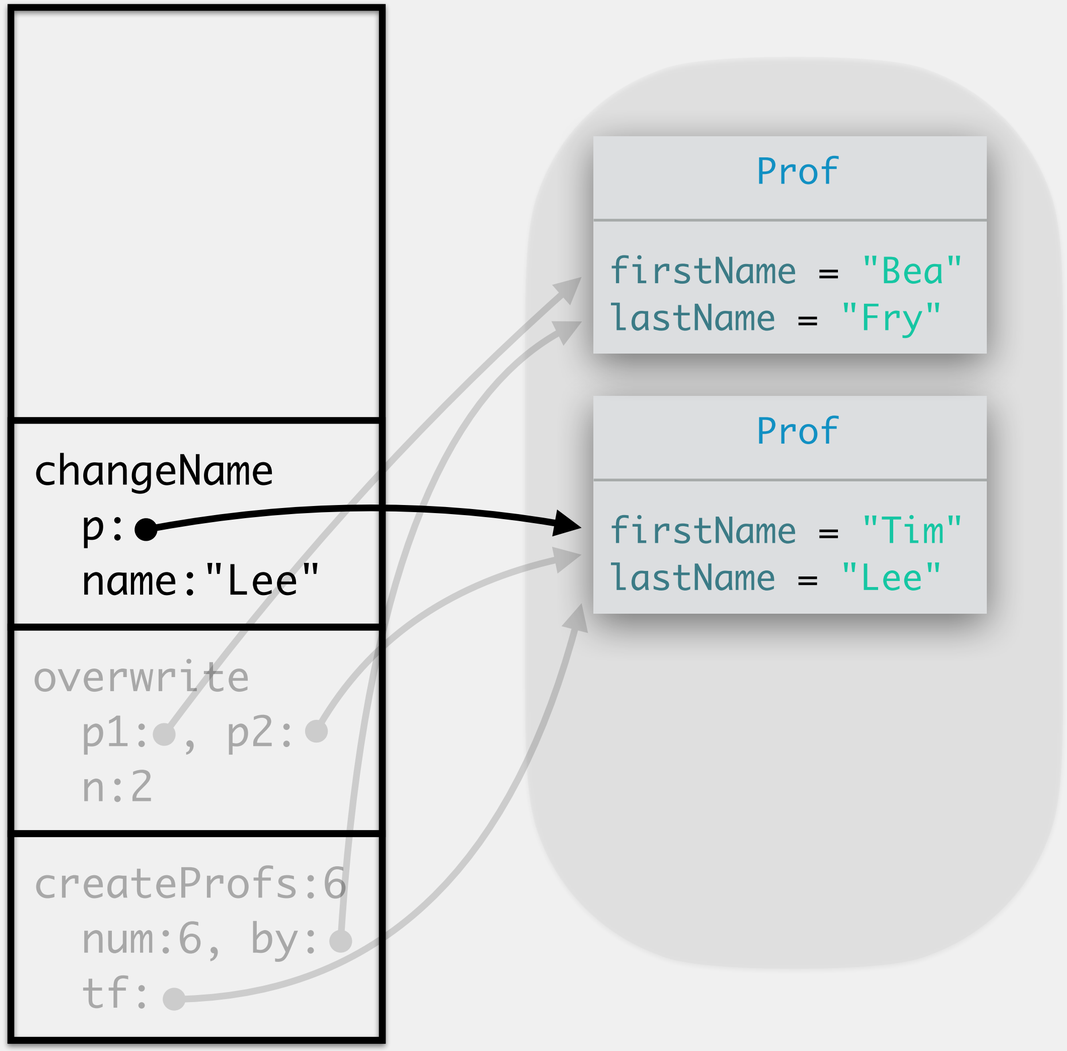
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14 >    changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18 >    p.lastName = name;
19  }
```



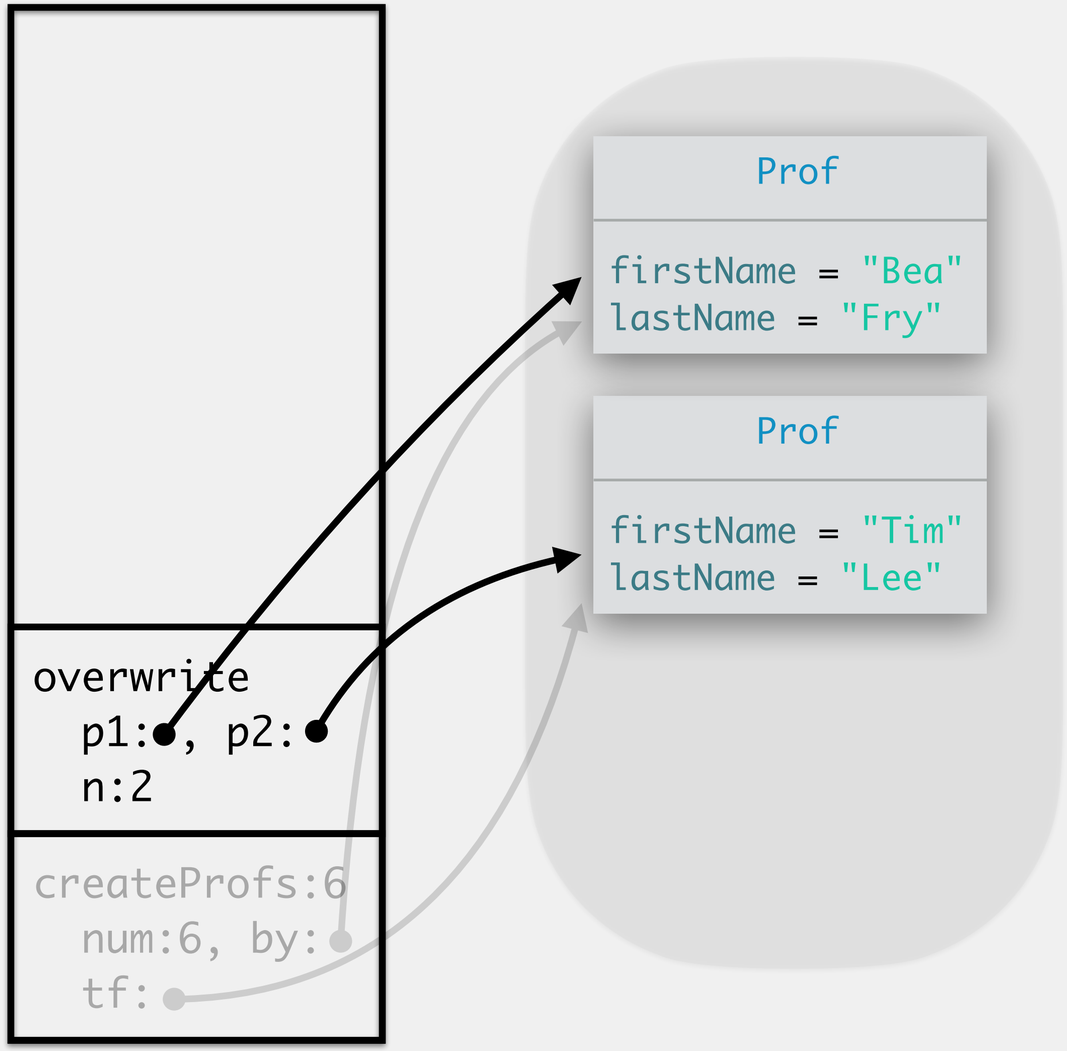
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14 >    changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19 >}
```



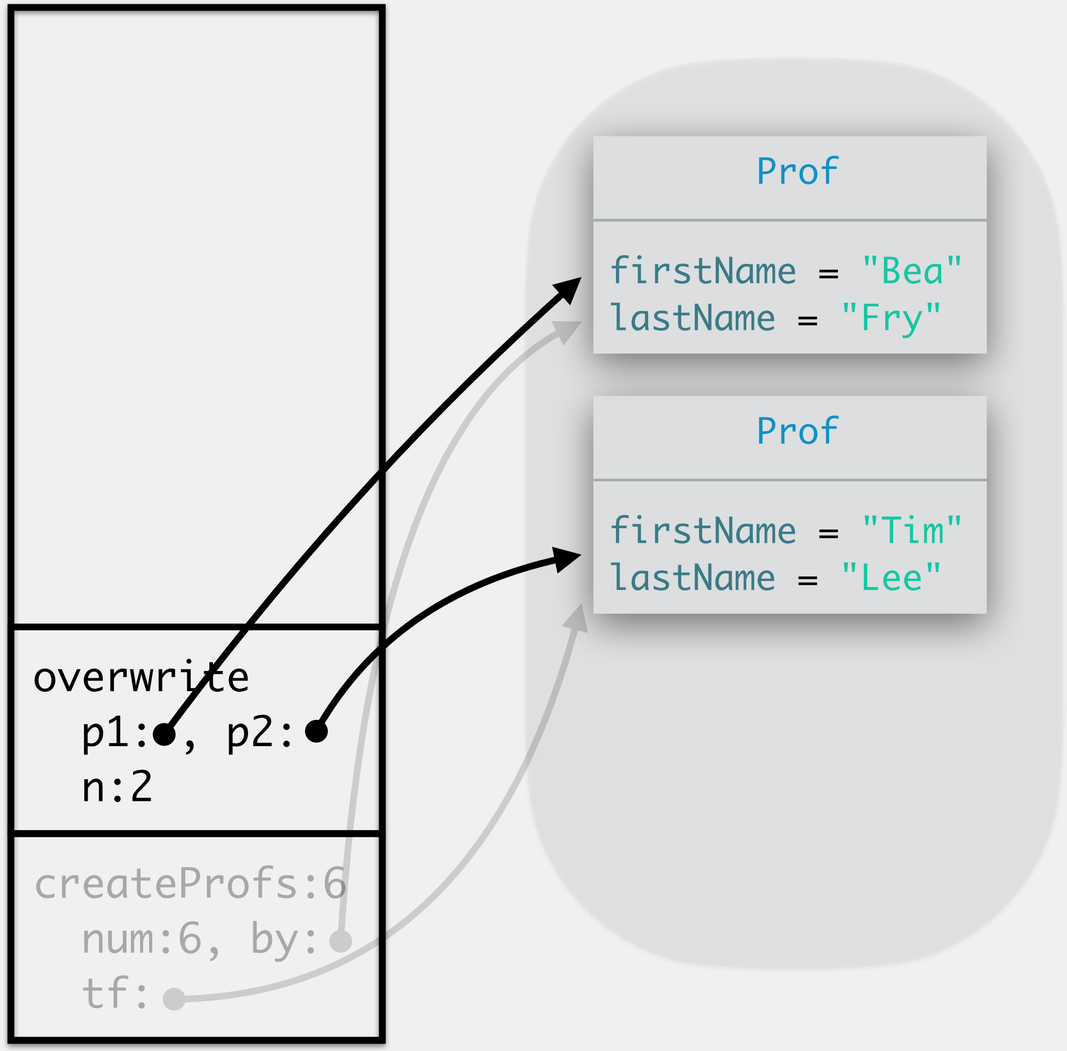
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14 >    changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



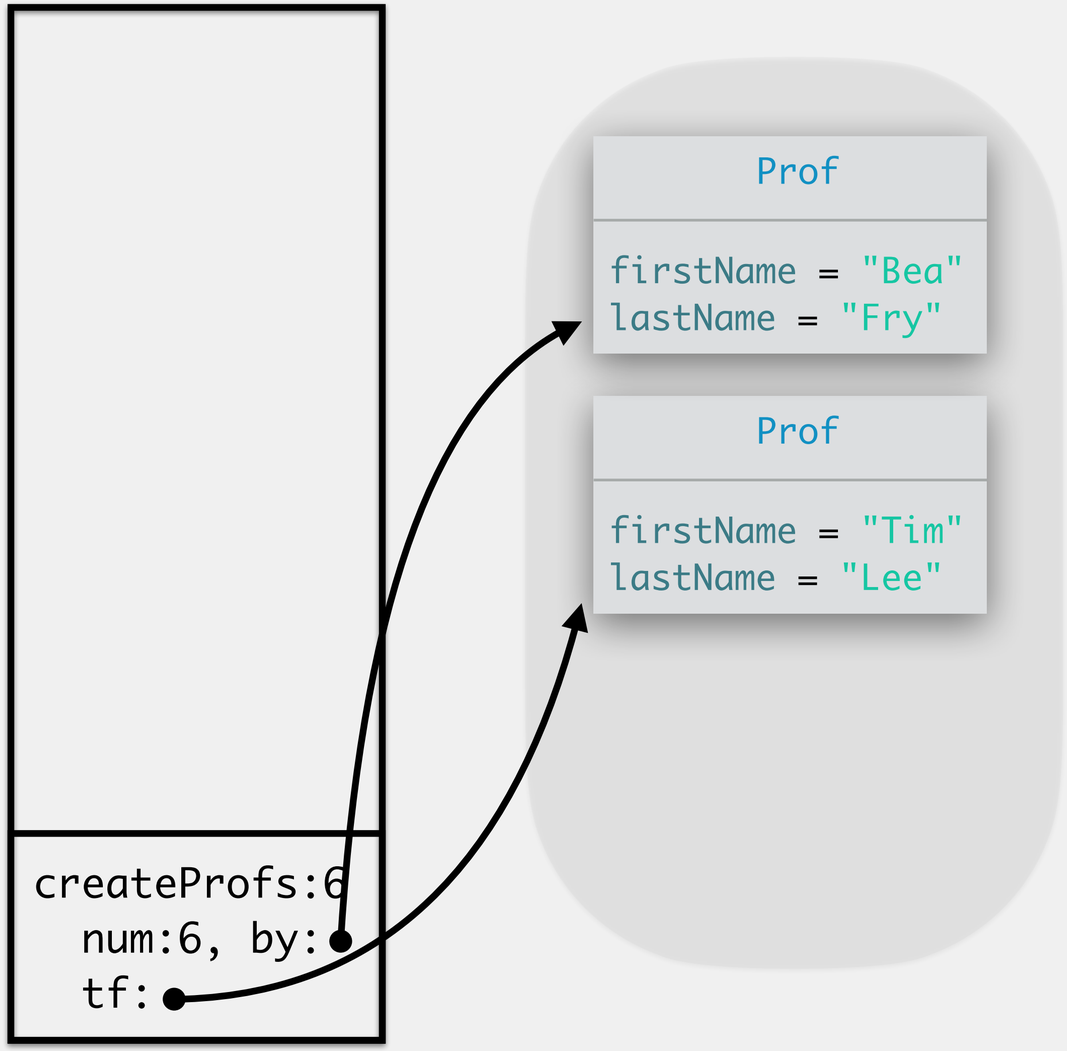
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15 >}
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



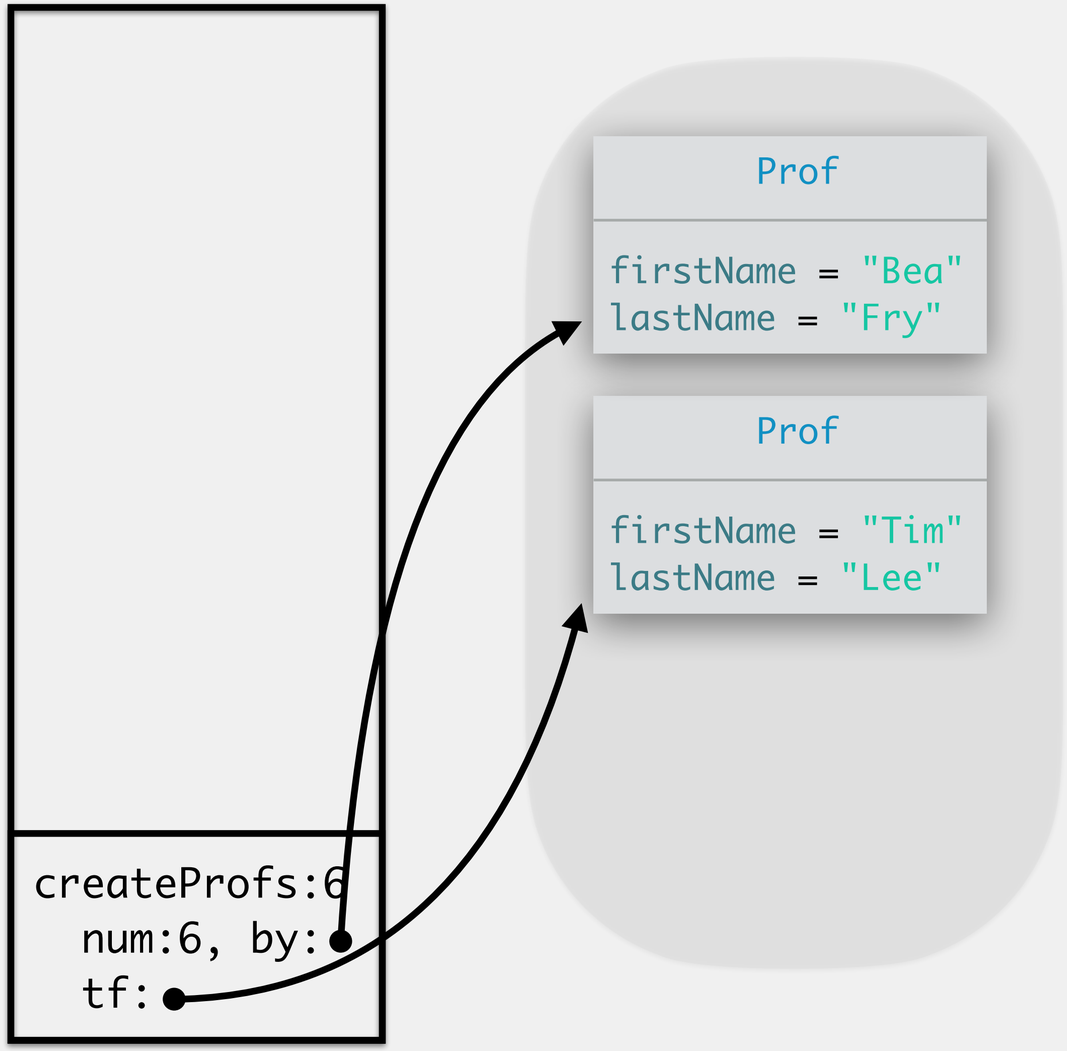
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



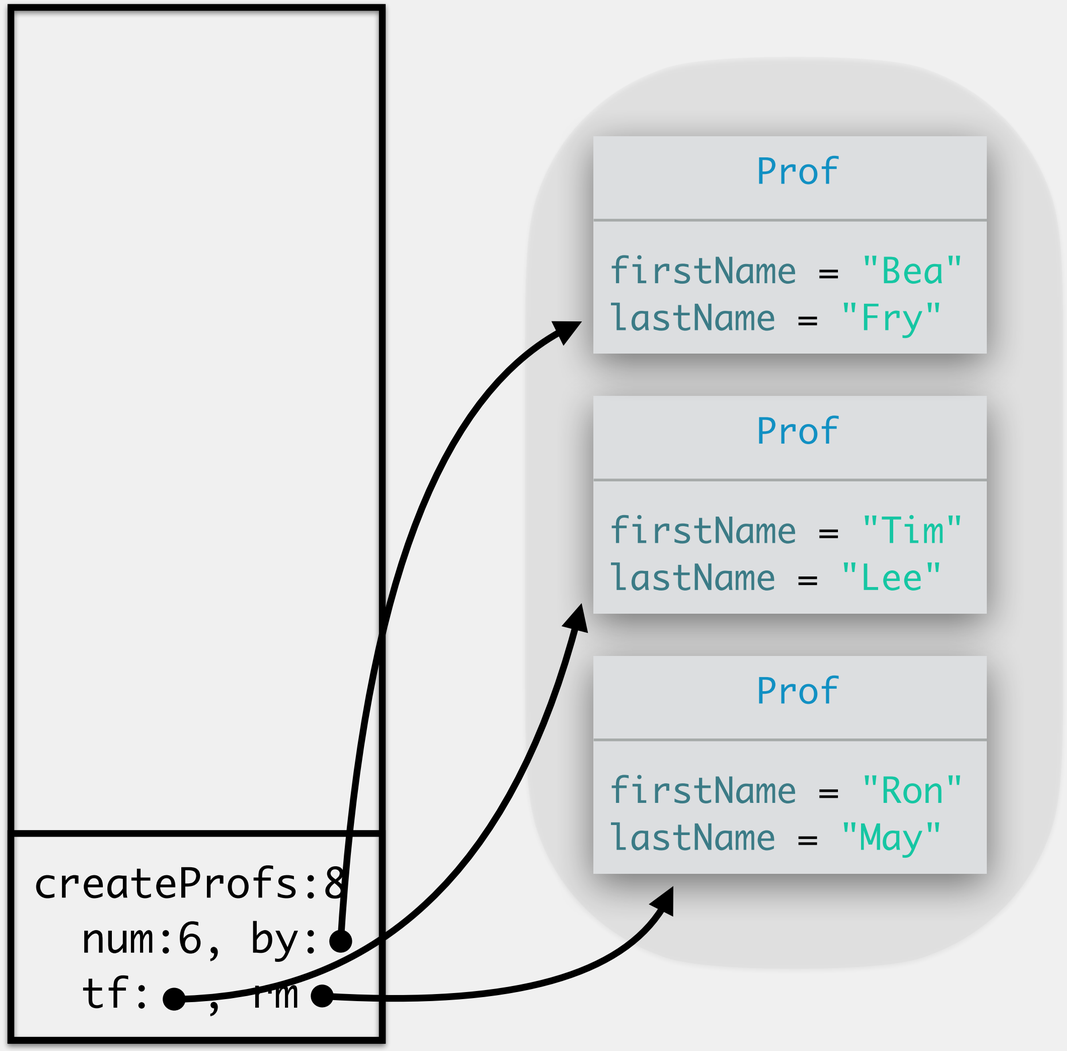
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7   >   Prof rm = new Prof("May", "Ron");
8       overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



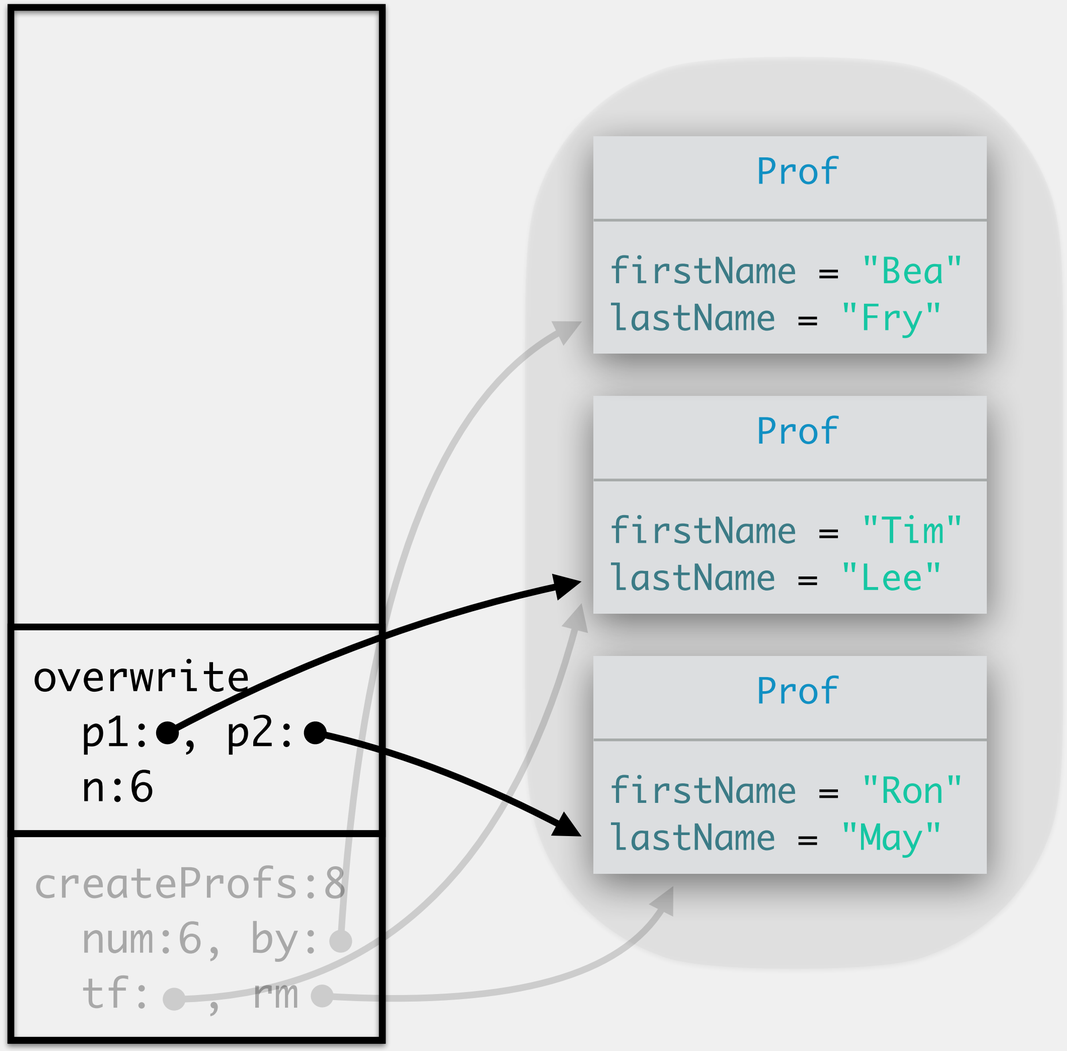
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



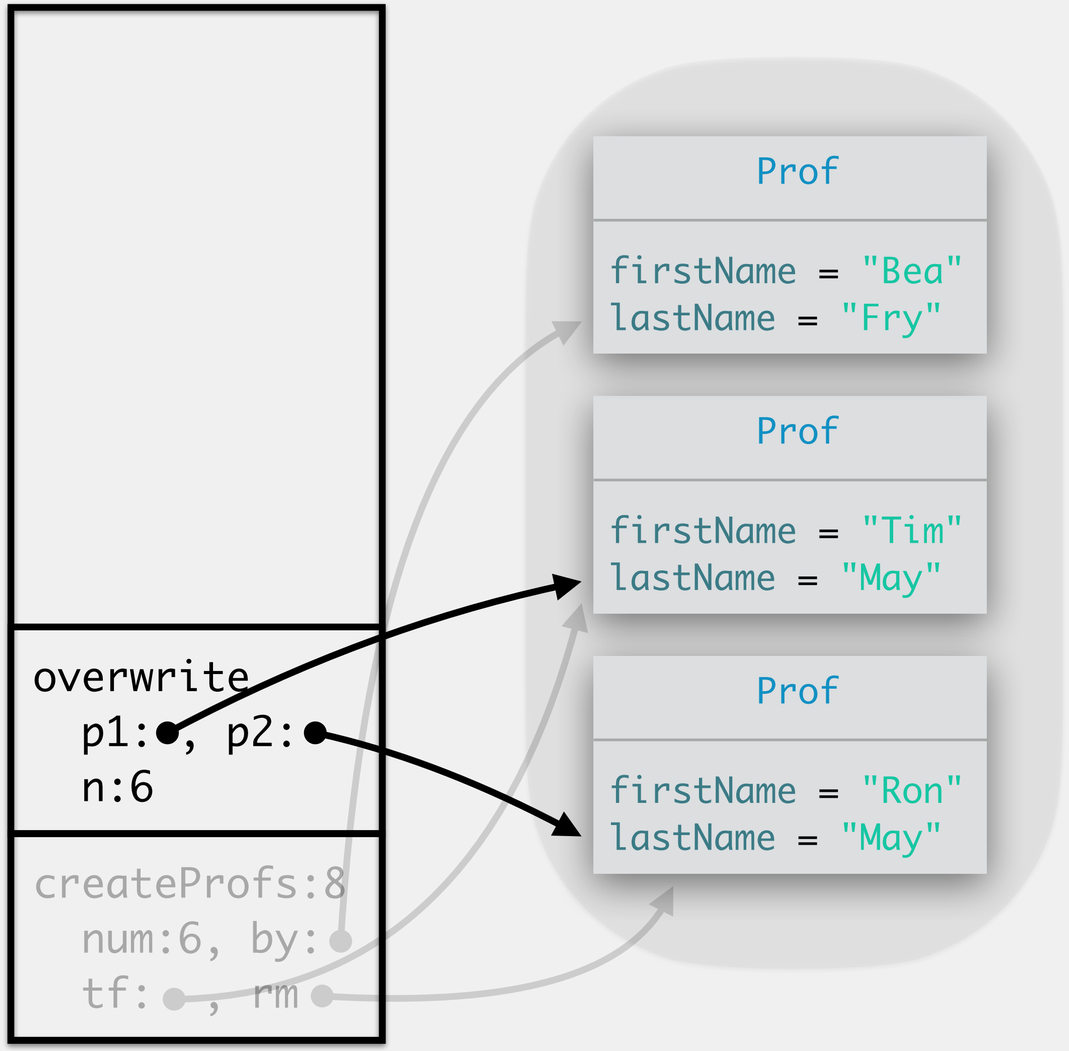
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12  >   p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



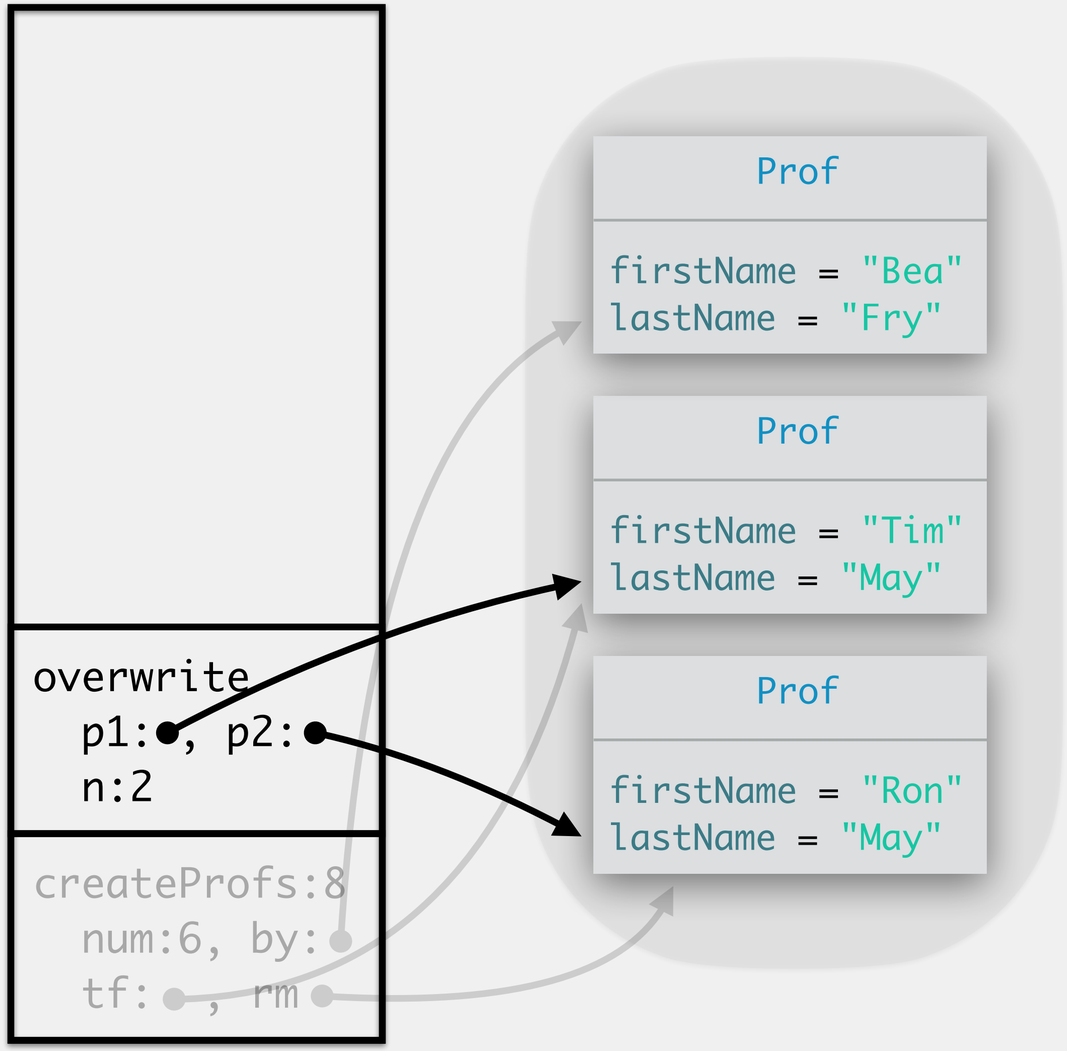
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13  >   n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



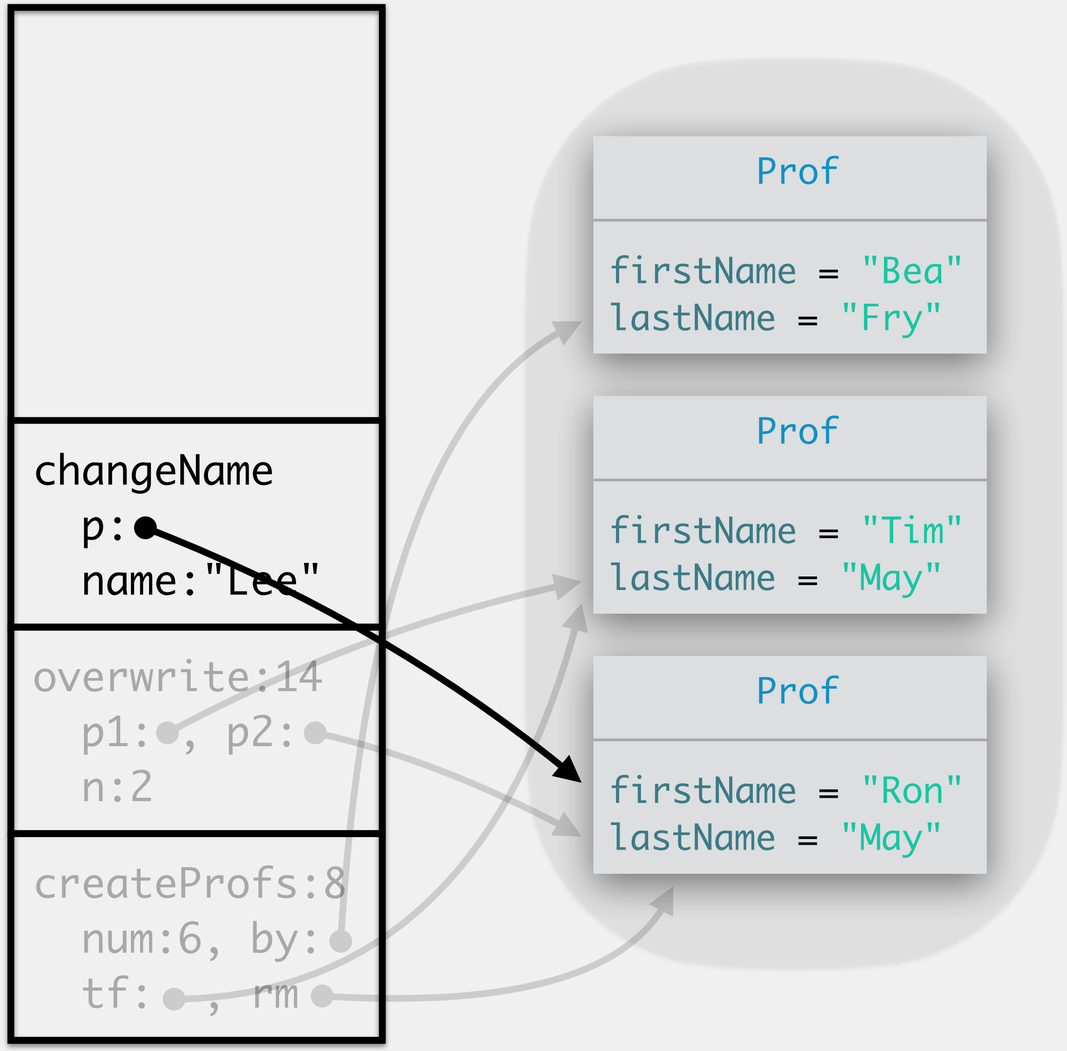
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14  >   changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



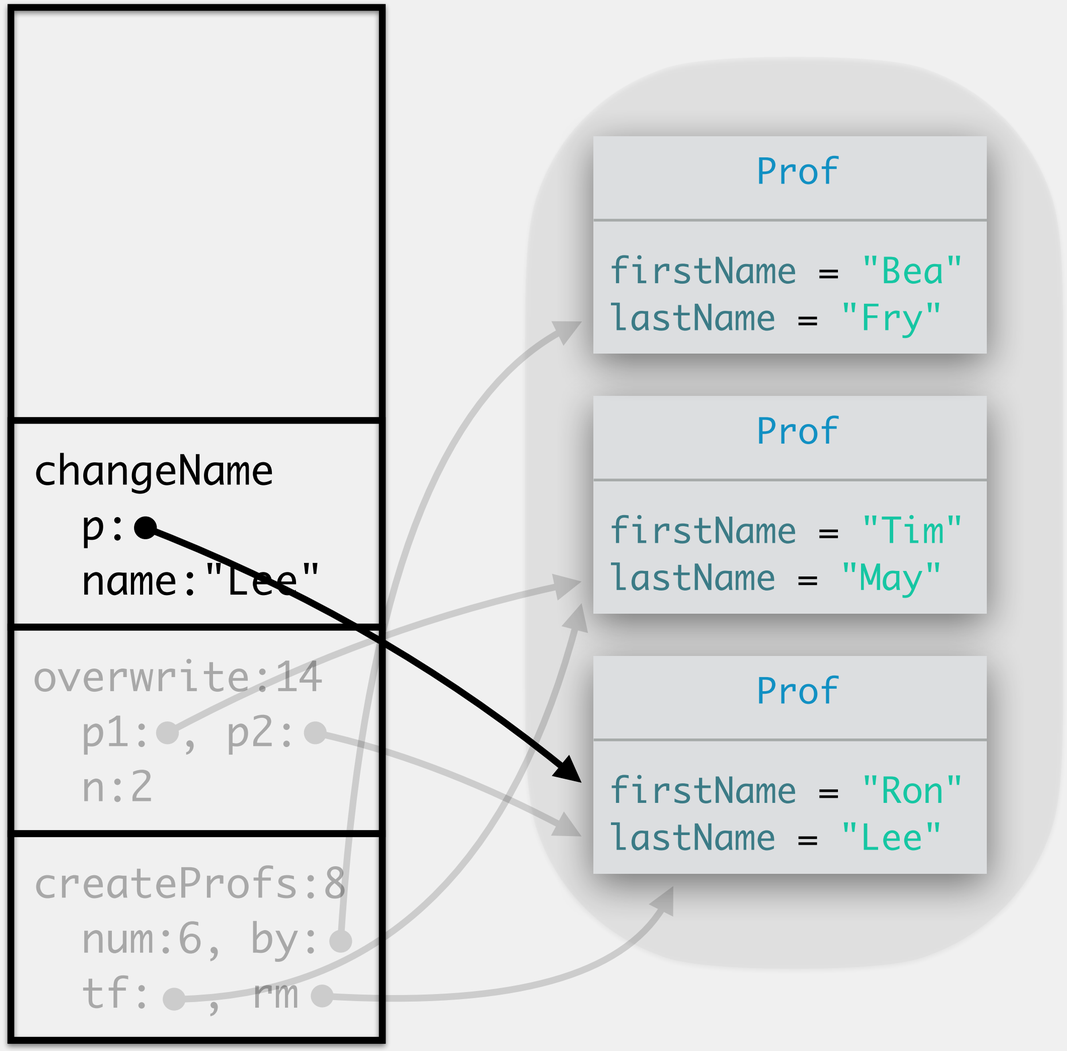
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14  >   changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18  >   p.lastName = name;
19  }
```



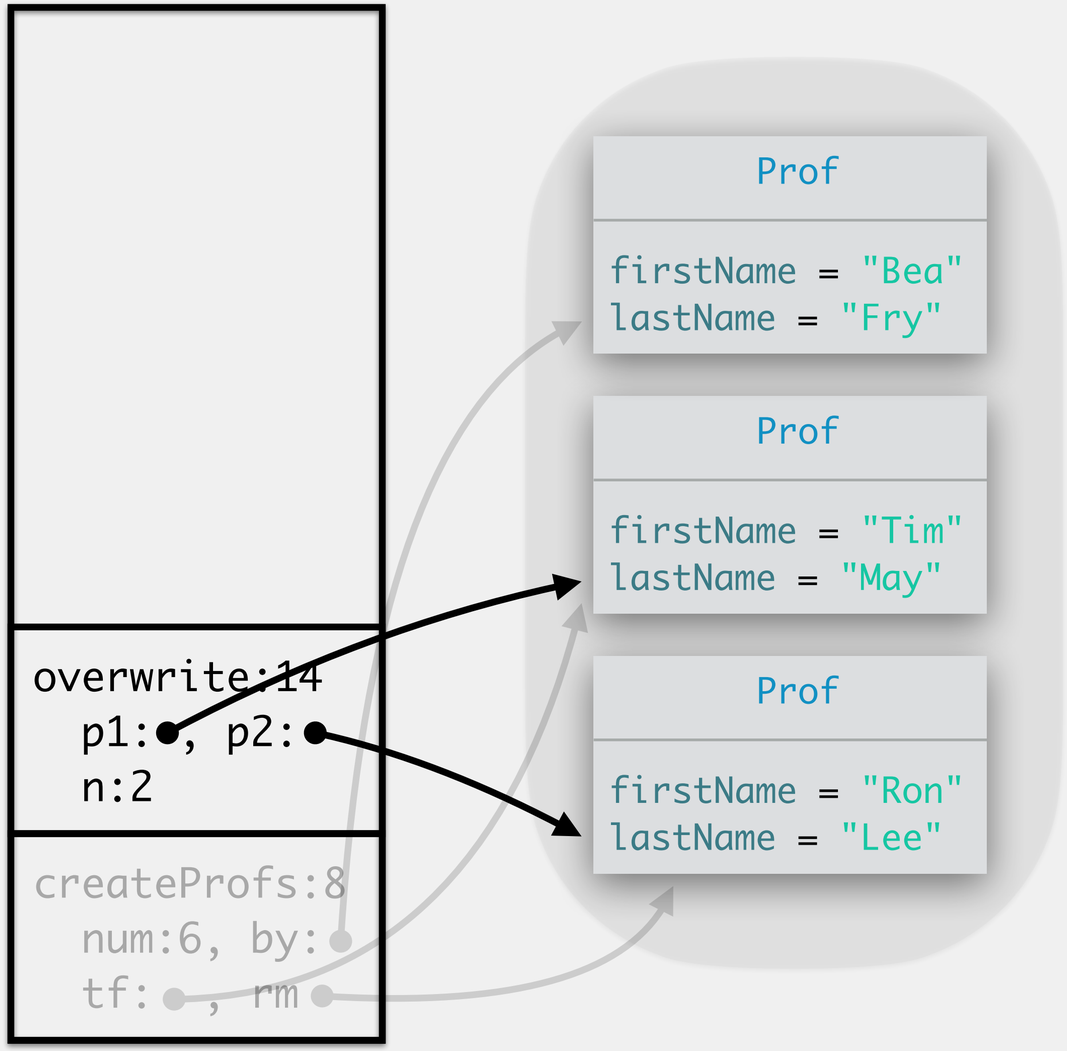
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14  >   changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  >}
```



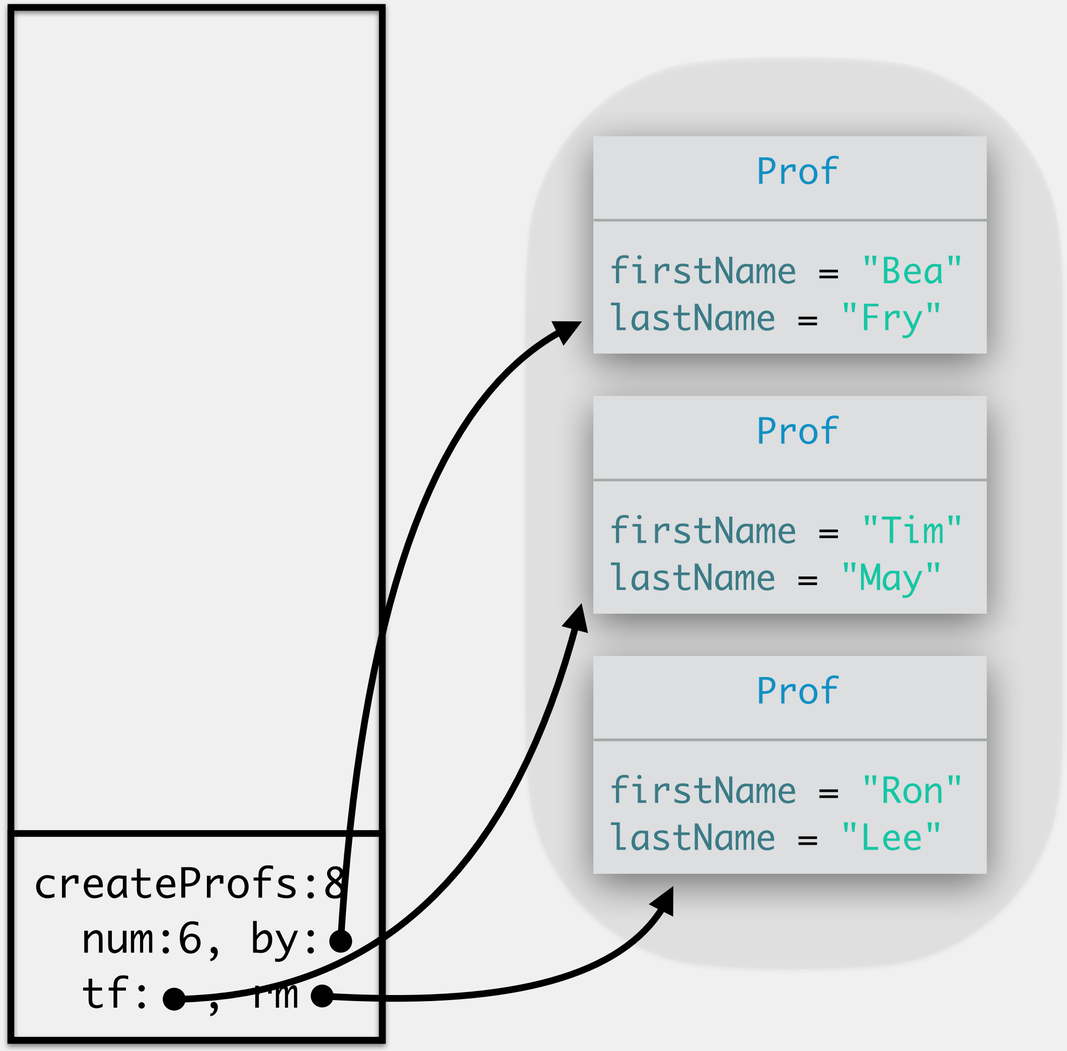
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14  >   changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



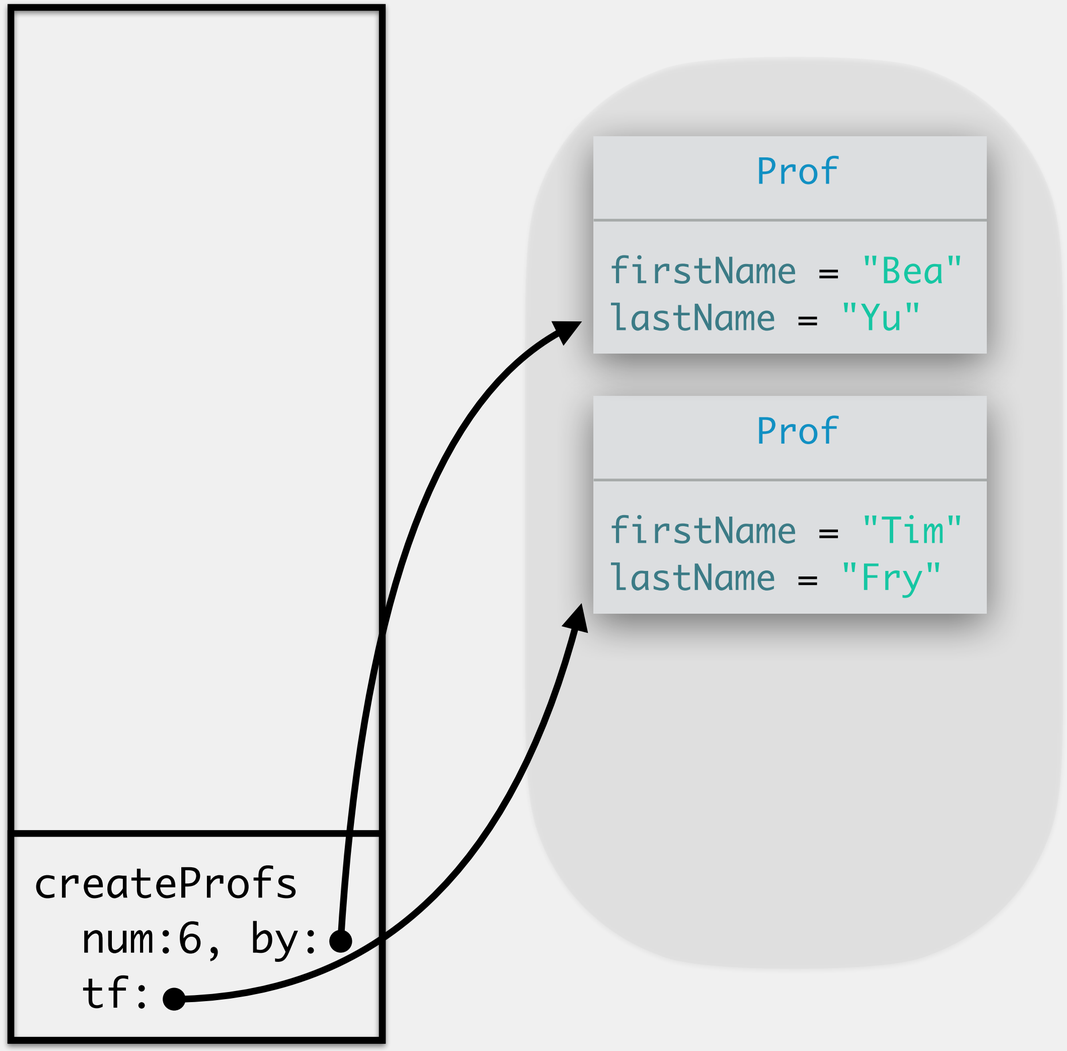
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6       overwrite(by, tf, num);
7       Prof rm = new Prof("May", "Ron");
8   >   overwrite(tf, rm, num);
9   }
10
11  public static void overwrite(Prof p1, Prof p2, int n) {
12      p1.lastName = p2.lastName;
13      n = 2;
14      changeName(p2, "Lee");
15  }
16
17  public static void changeName(Prof p, String name) {
18      p.lastName = name;
19  }
```



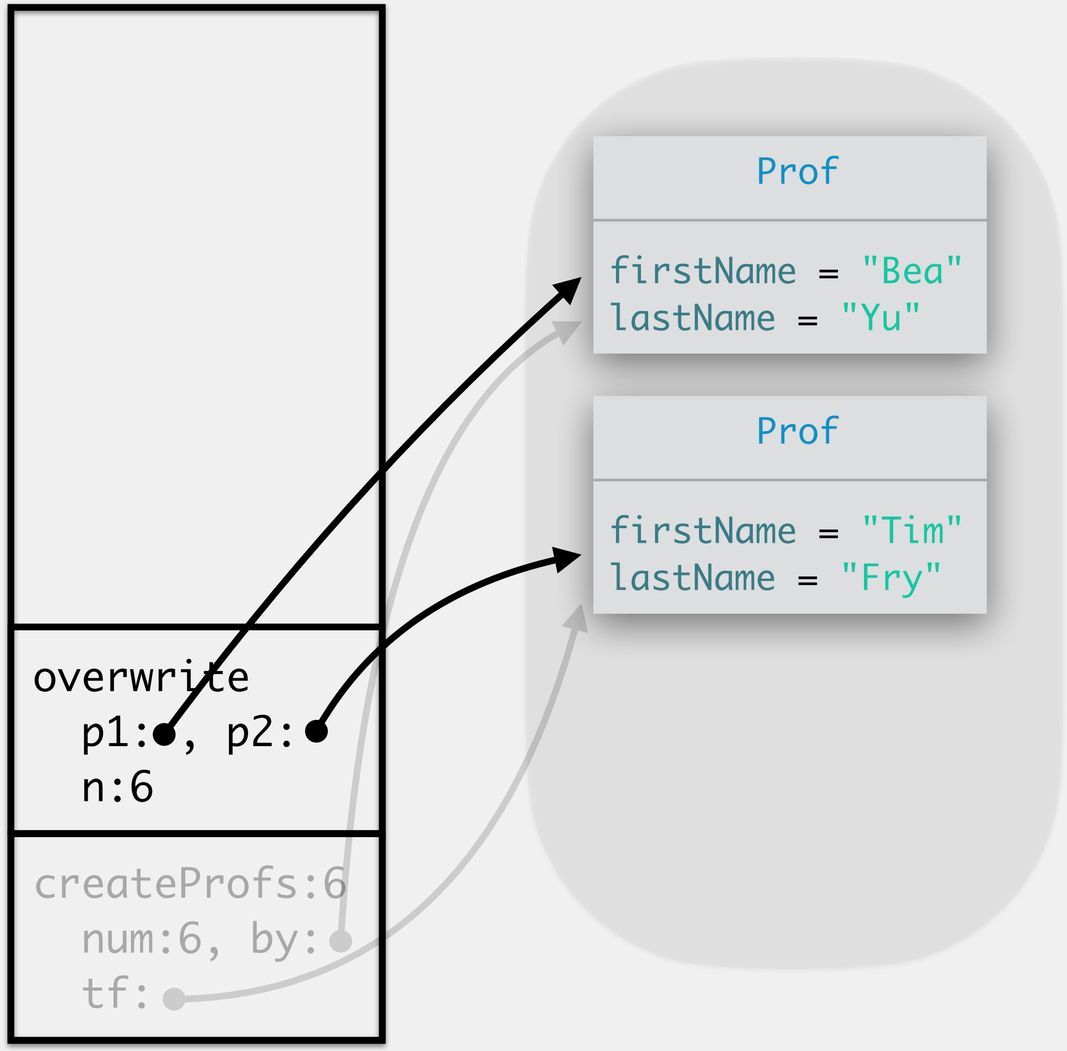
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12      changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



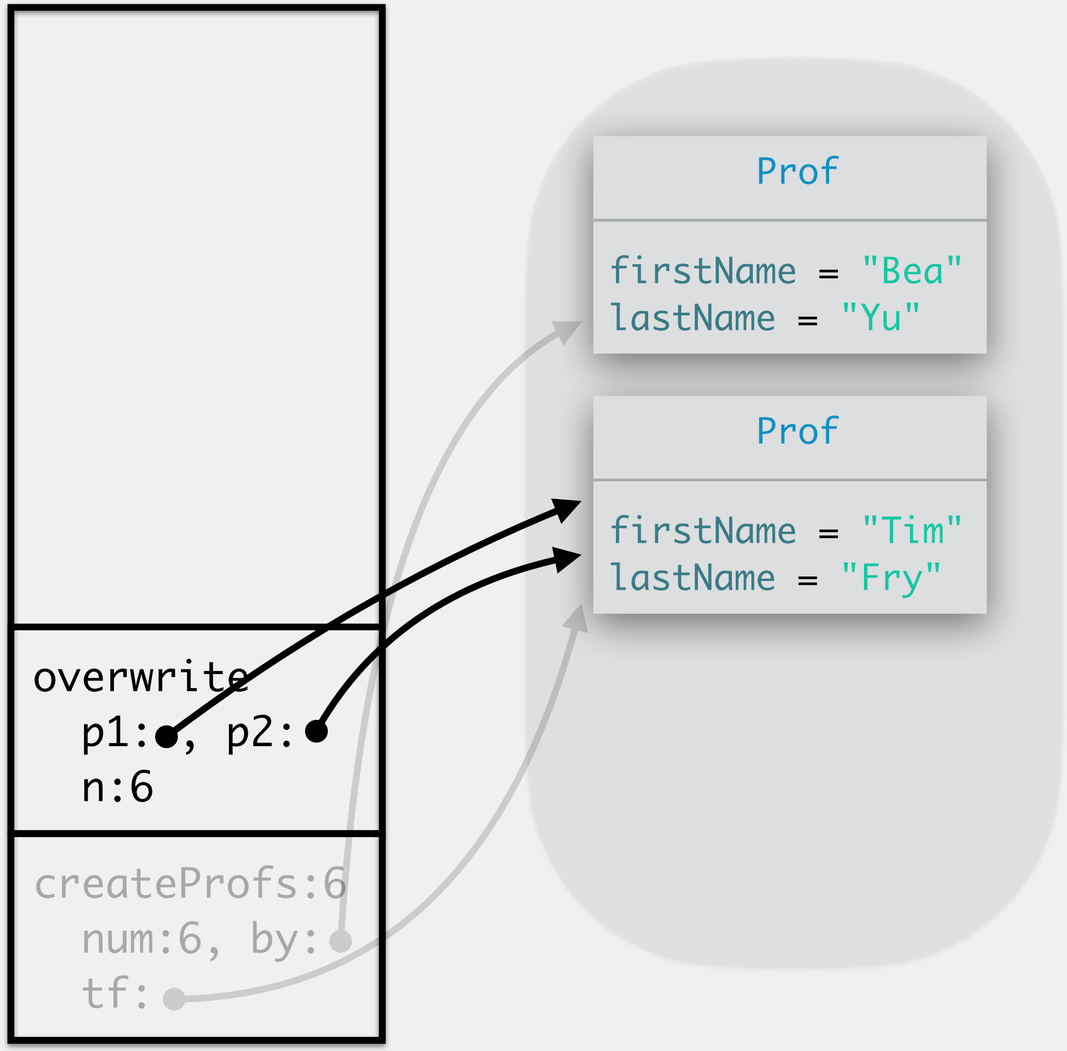
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10 >    p1 = p2;
11      n = 2;
12      changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



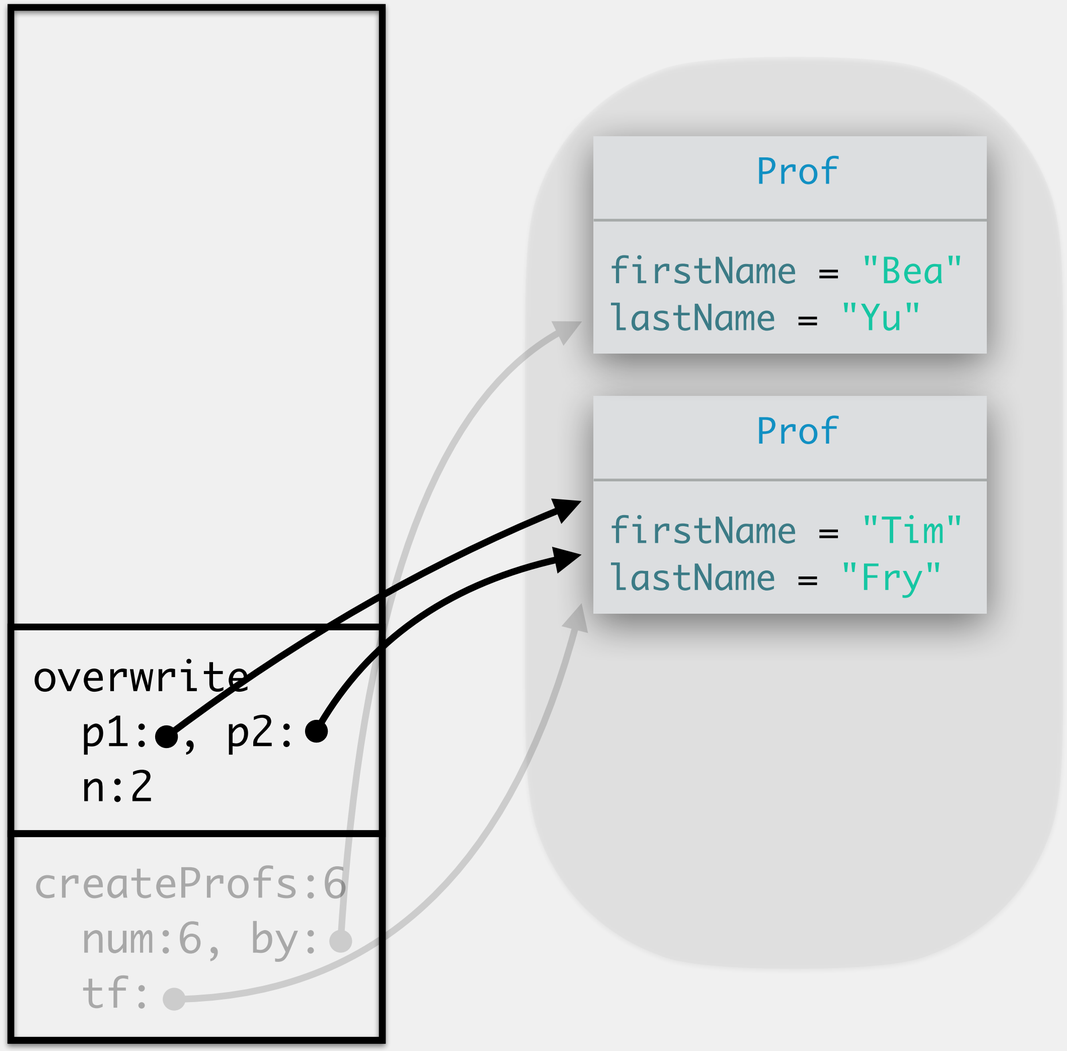
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11 >    n = 2;
12      changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



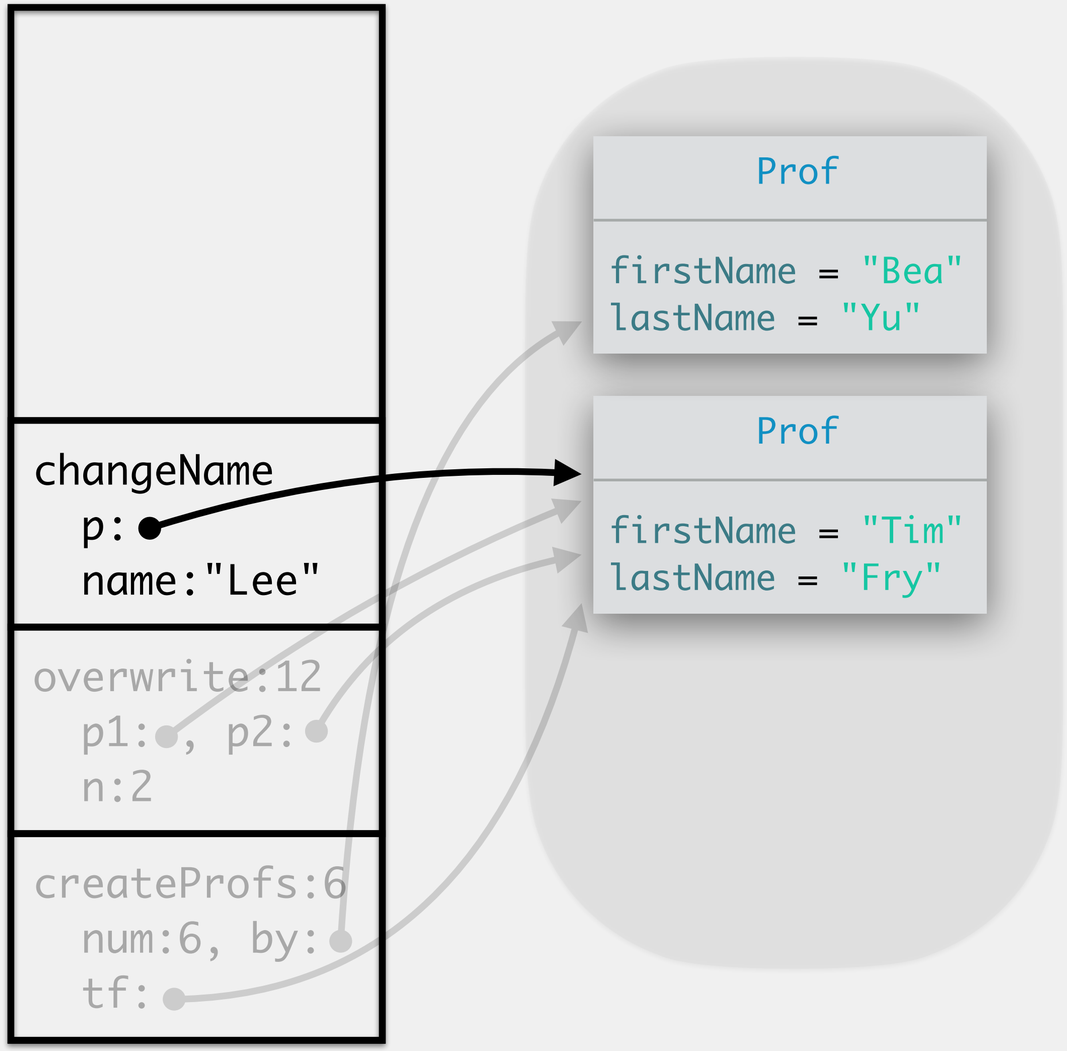
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12 >    changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



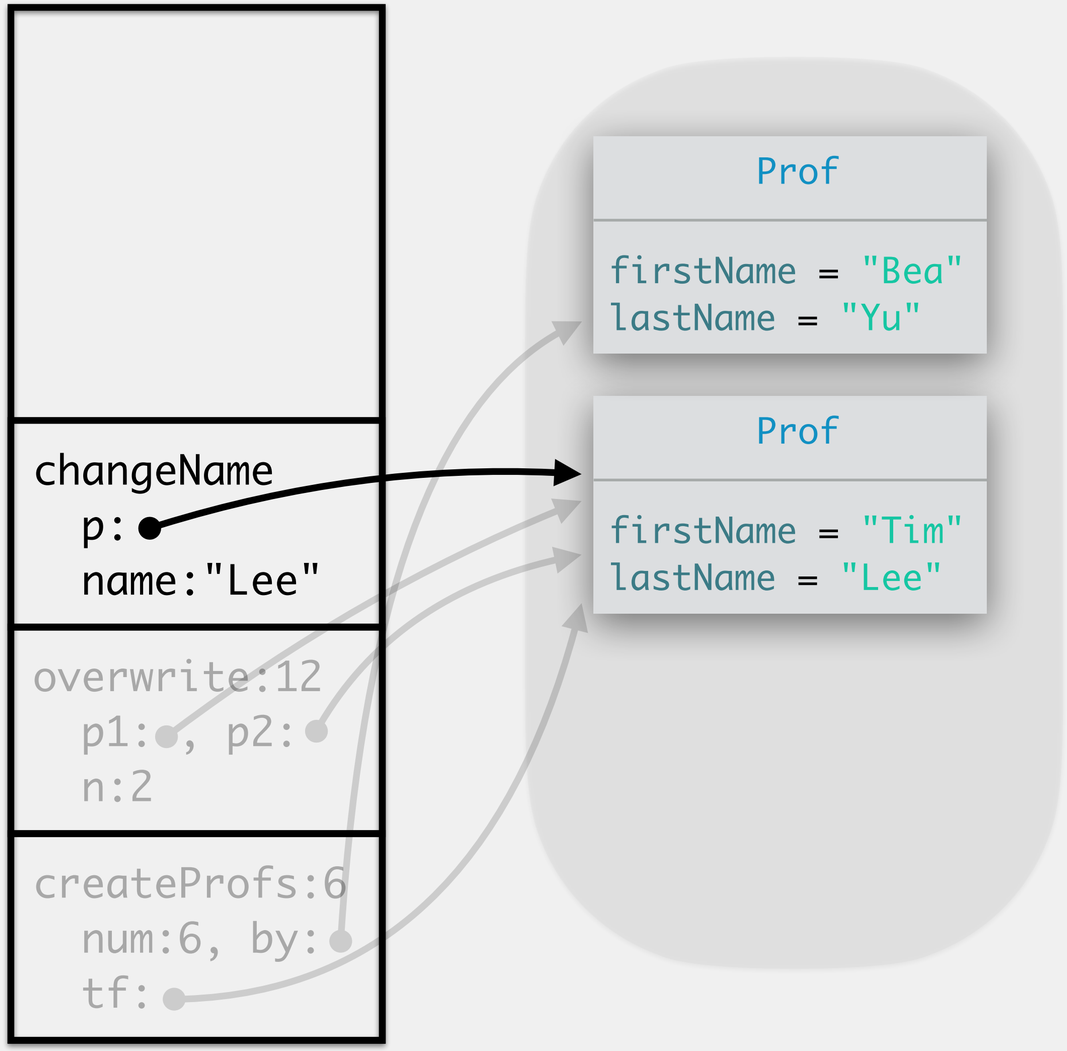
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12 >    changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16 >    p.lastName = name;
17  }
```



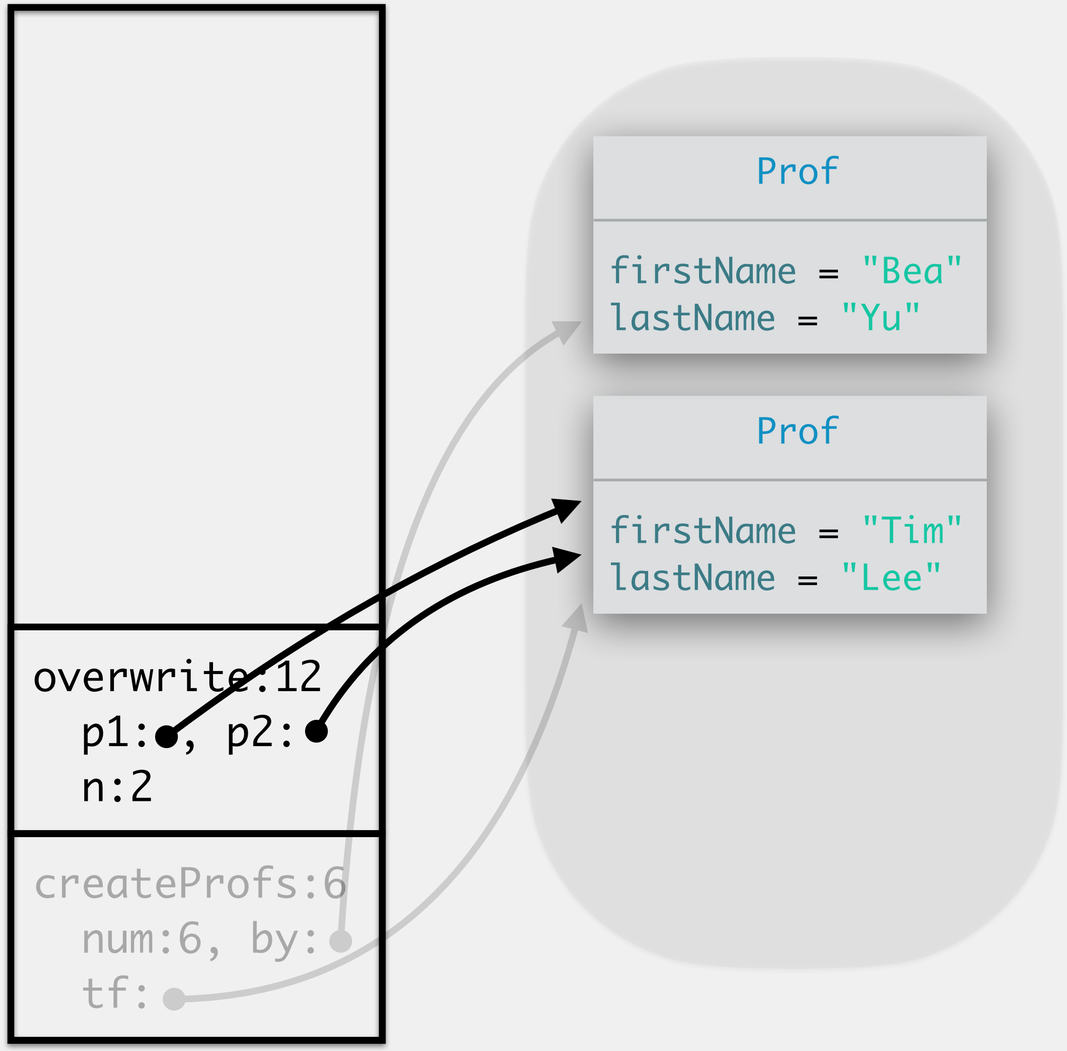
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12 >    changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17 >}
```



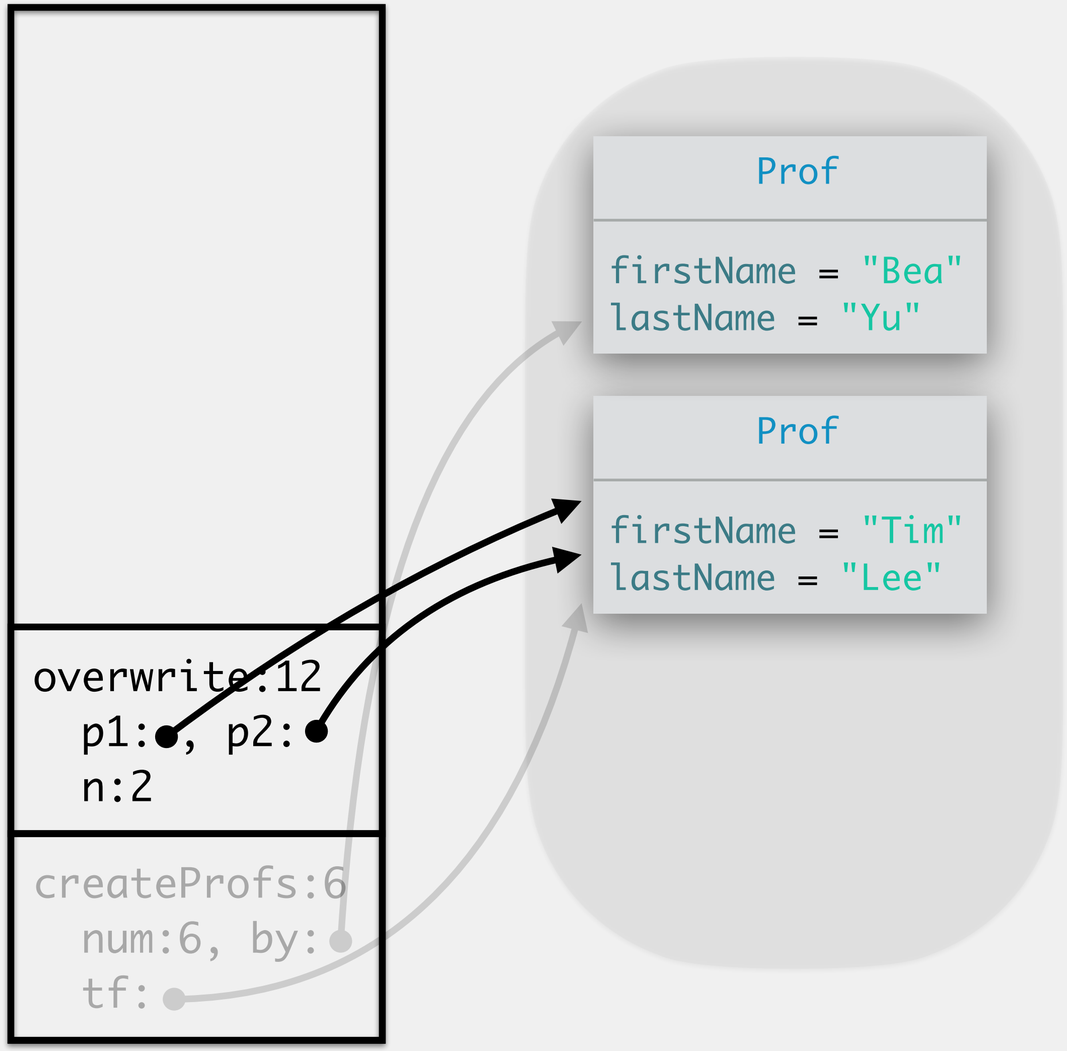
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12 >    changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



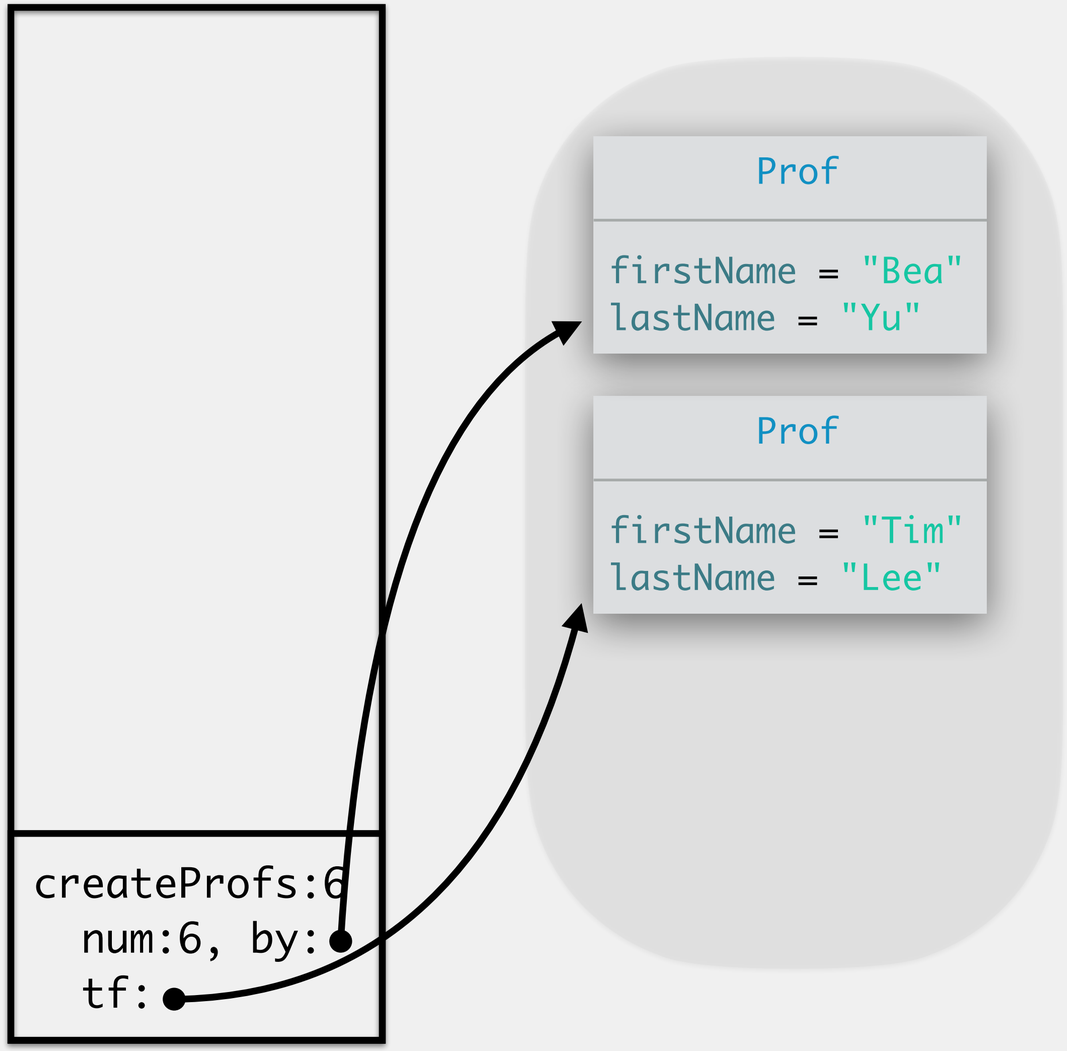
Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12      changeName(p1, "Lee");
13  >}
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



Using the Stack & Heap

```
1   public static void createProfs() {
2       Prof by = new Prof("Yu", "Bea");
3       Prof tf = new Prof("Fry", "Tim");
4       int num = 6;
5
6 >     overwrite(by, tf, num);
7   }
8
9   public static void overwrite(Prof p1, Prof p2, int n) {
10      p1 = p2;
11      n = 2;
12      changeName(p1, "Lee");
13  }
14
15  public static void changeName(Prof p, String name) {
16      p.lastName = name;
17  }
```



The Stack and Heap in Java

Heap employs *automatic memory management*

- space is appropriately allocated every time the new keyword is used
- memory is freed by the *garbage collector* when an object no longer has a reference to it
- novel feature introduced in Java!

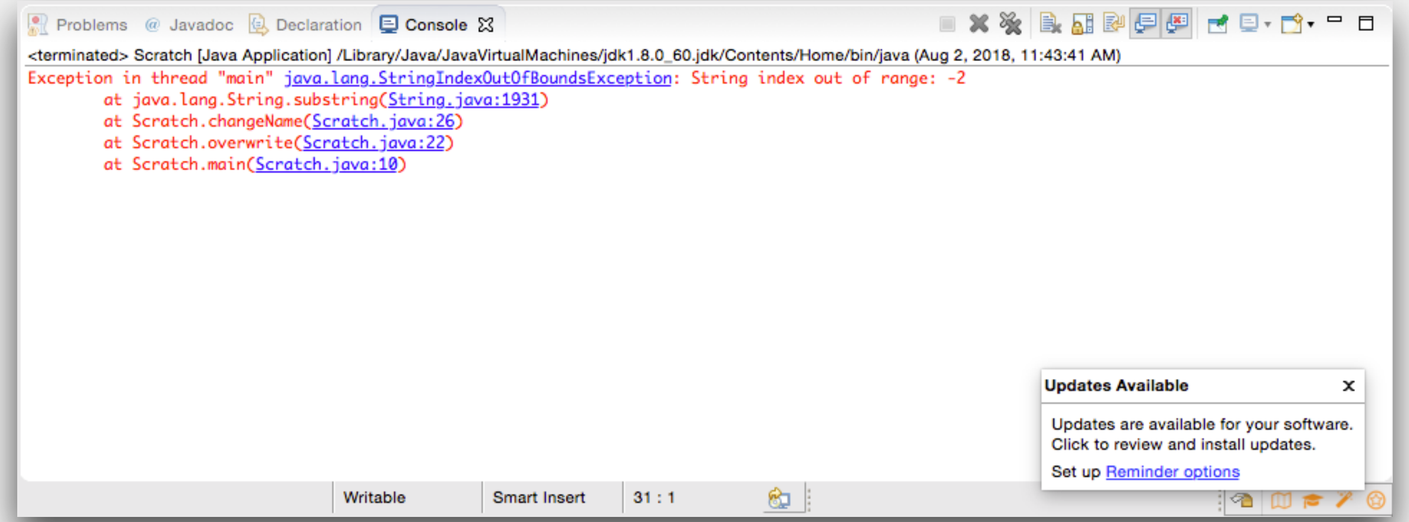
Stack is also used to aid in debugging

Stack Trace in Exceptions

stack trace pops off the entire stack and related info

usually when an exception occurs

Allows programmers to trace where things may have gone wrong



The Debugger

Allows the programmer to freeze and manually advance program execution

Common terms:

- *breakpoints* allows the programmer to define where to pause execution on one or more lines of code
  - can set breakpoint by double clicking just to the left of a line number
- *stepping* allows the programmer to advance execution in some way
- *resuming* allows the program to continue execution until the next breakpoint

resume

stepping

begin debuggin g

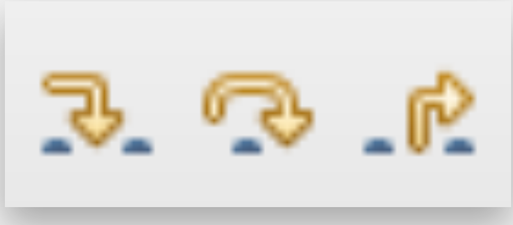
return to your normal Eclipse perspective

The Debugger

Allows the programmer to freeze and manually advance program execution

Common terms:

- *breakpoints* allows the programmer to define where to pause execution on one or more lines of code
- *stepping* allows the programmer to advance execution in some way
- *resuming* allows the program to continue execution until the next breakpoint



step into (a method) | step over (a line of code) | step out of (a method)

Stack Trace When Debugging

Debugger provides stack trace to aid in debugging

Can examine values of variables on each frame

- expand objects to see their attributes