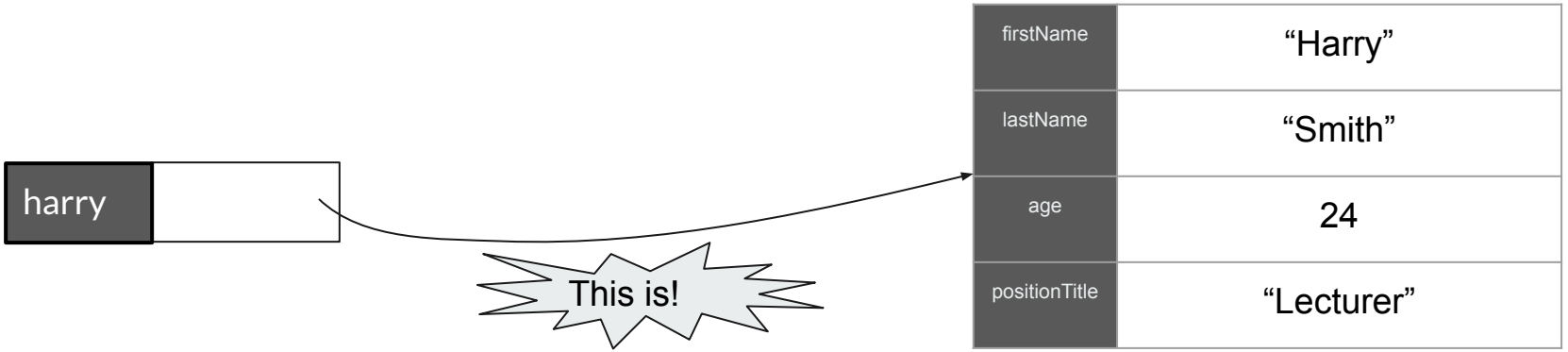
References in Java

CIS 110 (3/24/2021)

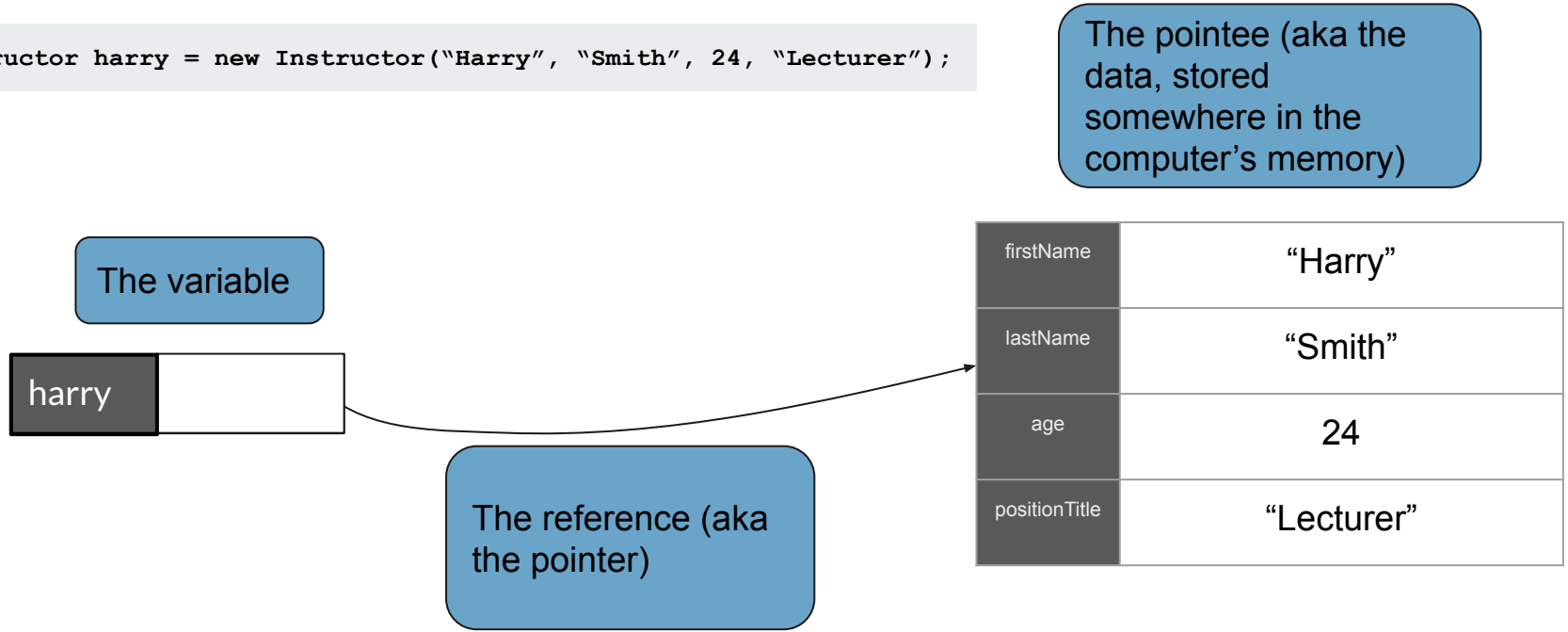
For non-primitive types, variable assignment works by passing a reference.

What's a reference?



What's a reference?

```
Instructor harry = new Instructor("Harry", "Smith", 24, "Lecturer");
```



Why does the pointee look like this?

```java
public class Instructor {
  private String firstName;
  private String lastName;
  private int age;
  private String positionTitle;

  ...
}
```

The pointee is a “box” containing the data that any object of its Class must have based on the fields (name, age, etc.)

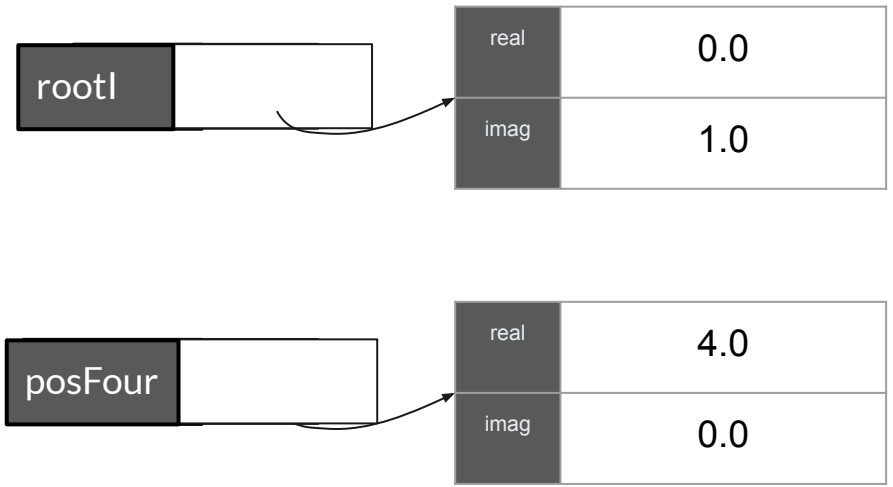
The pointee (aka the data, stored somewhere in the computer’s memory)

firstName	“Harry”
lastName	“Smith”
age	24
positionTitle	“Lecturer”

Worked Example of Variable Assignment

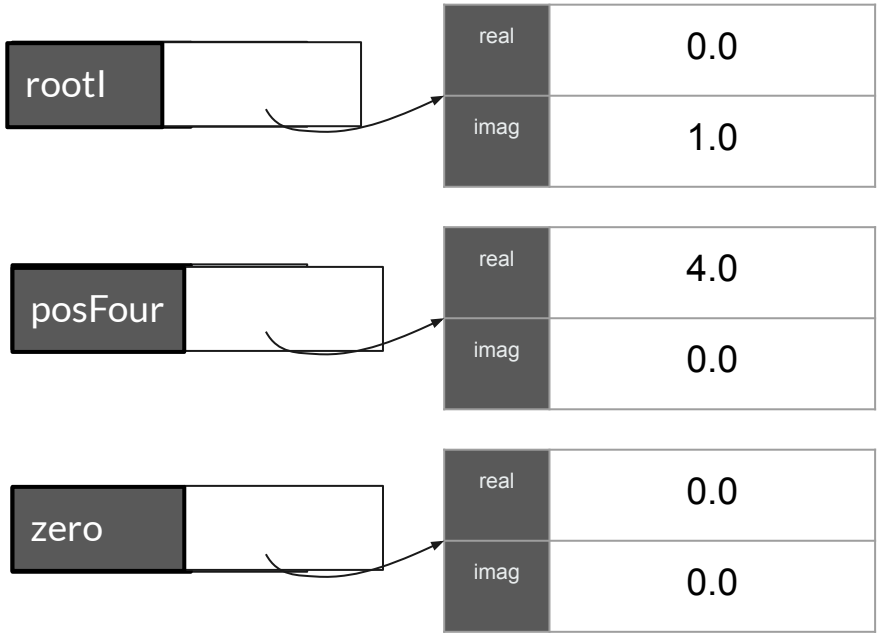
```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);
```

```java
public class Complex {
  private double real;
  private double imag;

  public Complex(double real, double imag) {
    this.real = real;
    this.imag = imag;
  }

  public void copyValues(Complex other) {
    this.real = other.real;
    this.imag = other.imag;
  }
  ...
}
```



Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
```

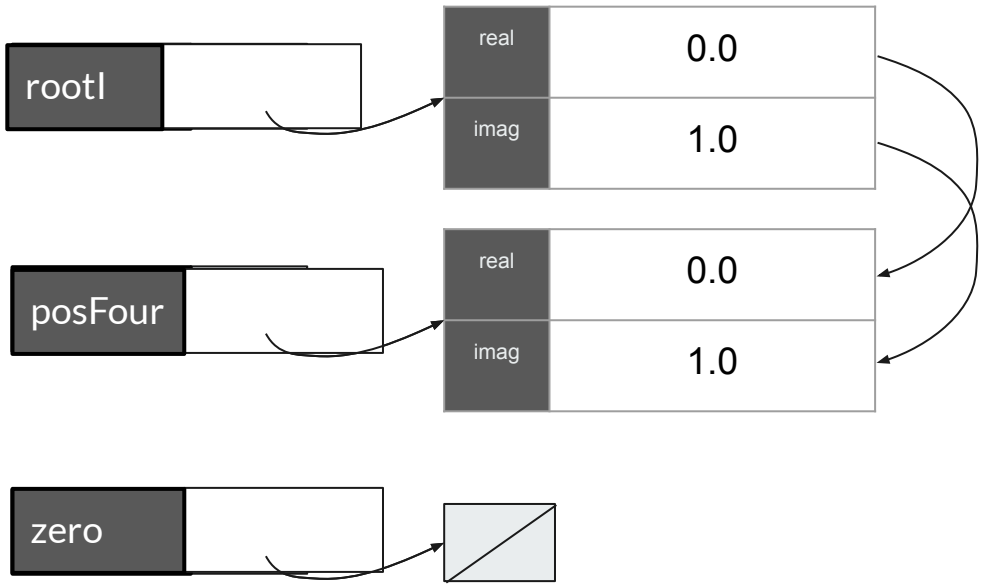


Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;
```

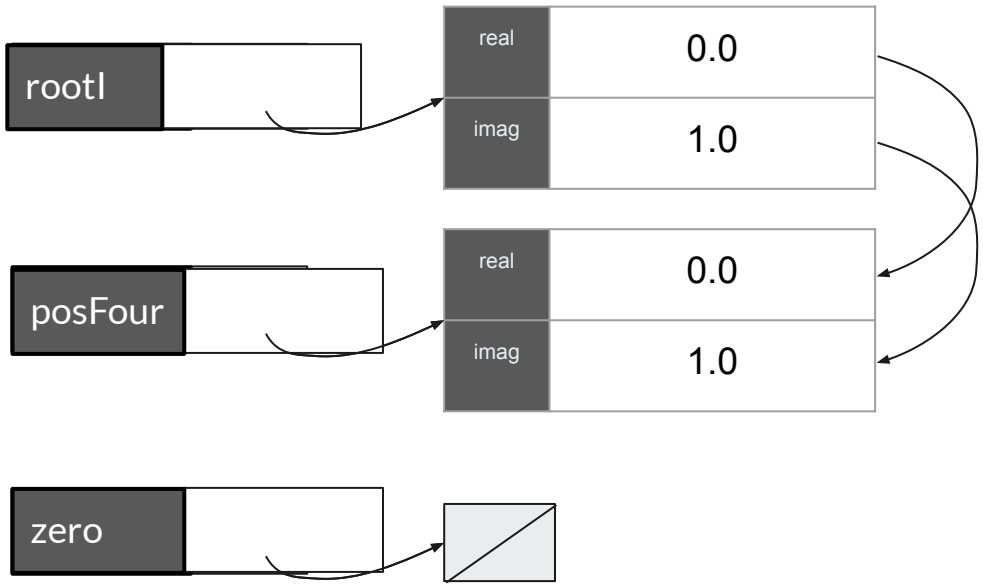
Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);
```



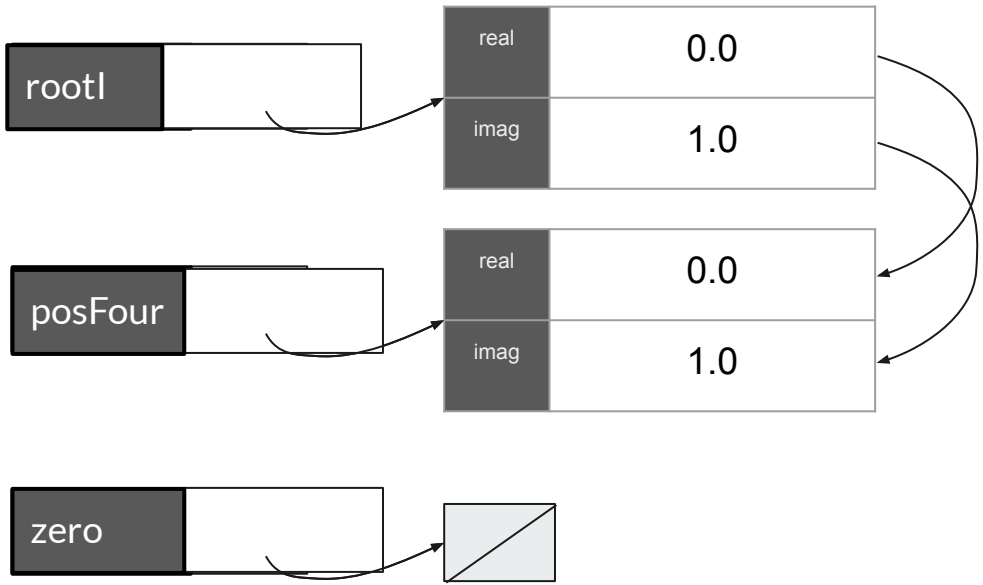
Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);
```



Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);

posFour == rootI; //How does this evaluate?
```



How does posFour == rootI evaluate here?

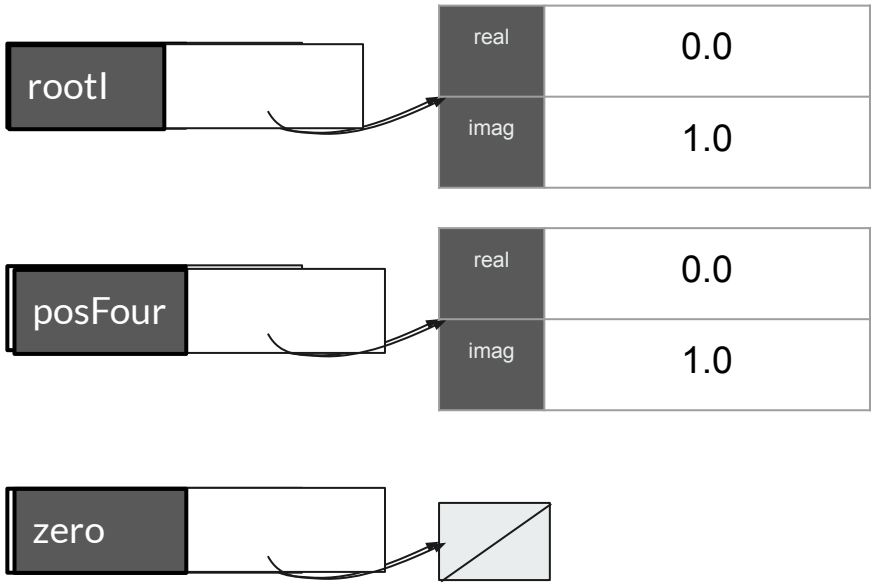
true

false

Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);

posFour == rootI; // false

posFour.equals(rootI); // true or false?
```

```
public boolean equals(Complex other) {
  return this.real == other.real &&
         this.imag == other.imag;
}
```



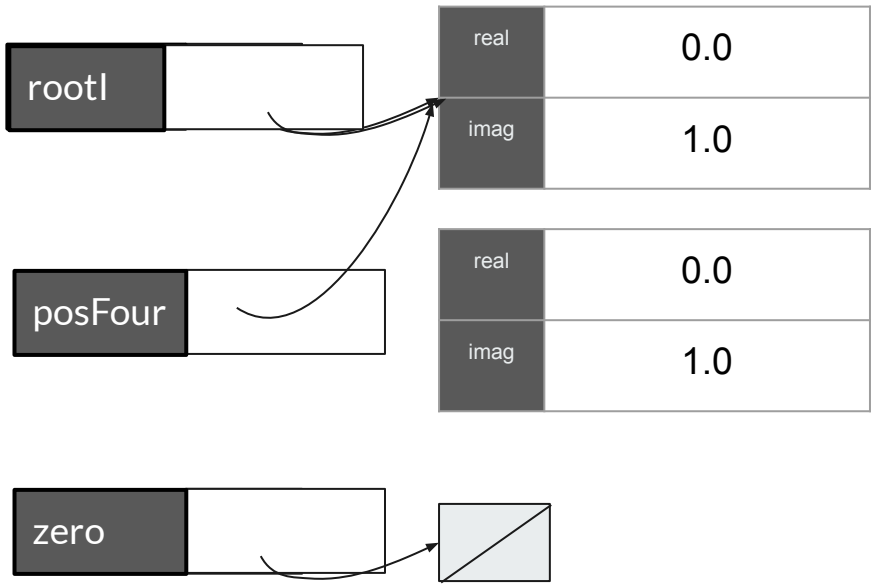
How does posFour.equals(rootI) evaluate?

true

false

Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);

posFour == rootI; // false

posFour.equals(rootI); // true

posFour = rootI;

// how do these evaluate?
posFour == rootI;
posFour.equals(rootI);
```

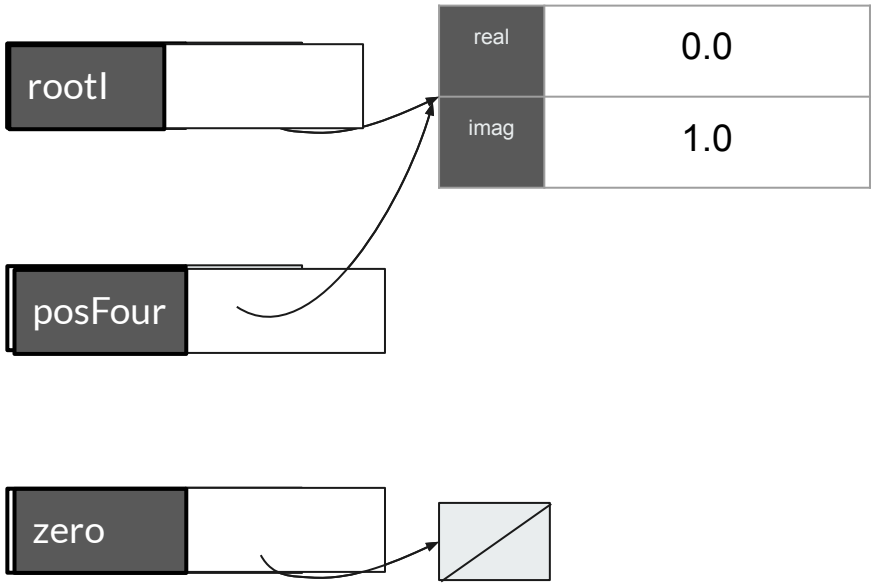


How do these evaluate: posFour == rootI; posFour.equals(rootI);

- false, false
- false, true
- true, false
- true, true

Worked Example of Variable Assignment

```
Complex rootI = new Complex(0.0, 1.0);
Complex posFour = new Complex(4.0, 0.0);

Complex zero = new Complex(0.0, 0.0);
zero = null;

posFour.copyValues(rootI);

posFour == rootI; // false

posFour.equals(rootI); // true

posFour = rootI;

// how do these evaluate?
posFour == rootI; // true
posFour.equals(rootI); // true
```

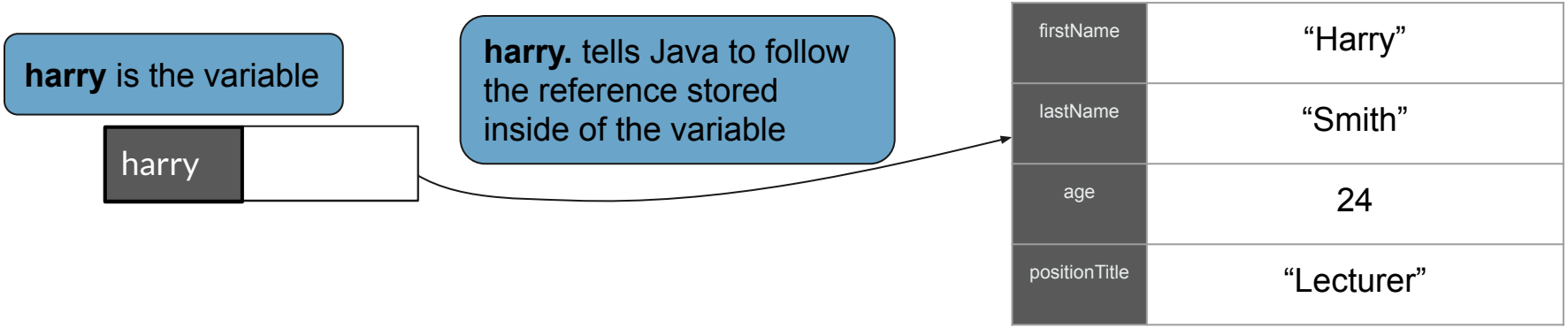


Dereferencing

Dereferencing uses the . operator

```
Instructor harry = new Instructor("Harry", "Smith", 24, "Lecturer");

System.out.println(harry.lastName + ", " + harry.firstName)
```

For harry.lastName...



Dereferencing uses the . operator

```
Instructor harry = new Instructor("Harry", "Smith", 24, "Lecturer");

System.out.println(harry.lastName + ", " + harry.firstName)
```

harry.lastName is the value of the field "lastName" of the object that we've arrived at following the reference.

For harry.lastName...

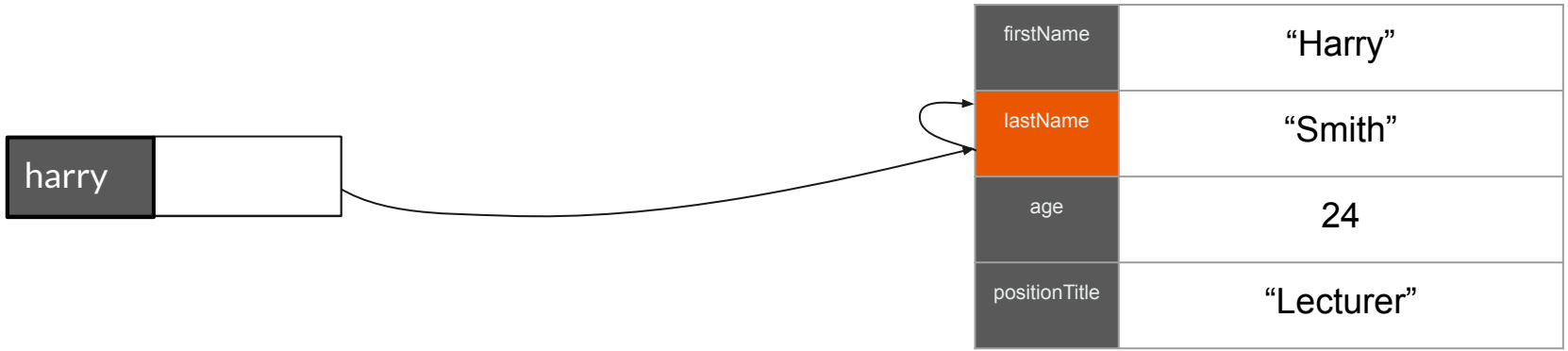
harry is the variable

harry. tells Java to follow the reference stored inside of the variable

firstName	"Harry"
lastName	"Smith"
age	24
positionTitle	"Lecturer"

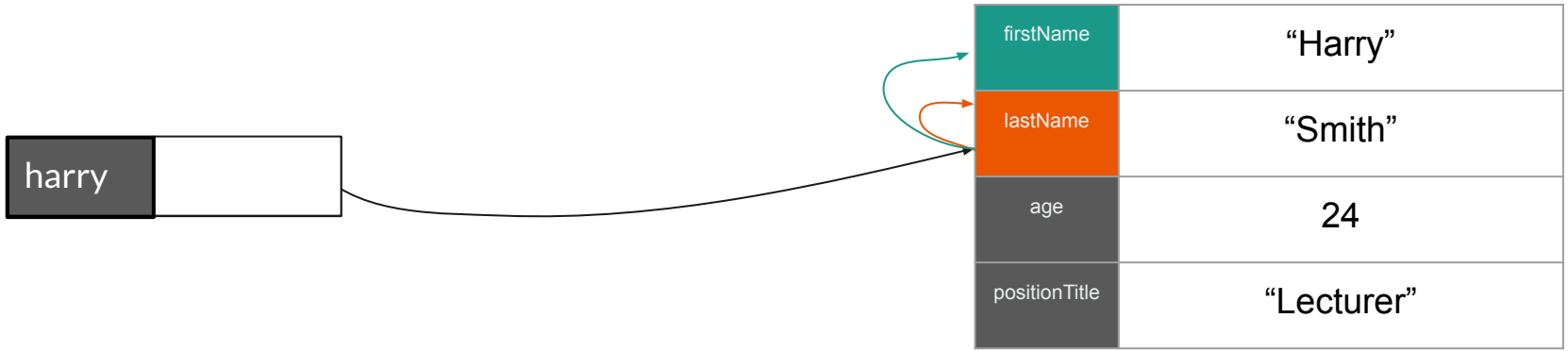
Dereferencing uses the . operator

```
Instructor harry = new Instructor("Harry", "Smith", 24, "Lecturer");

System.out.println(harry.lastName + ", " + harry.firstName)
```



Dereferencing uses the . operator

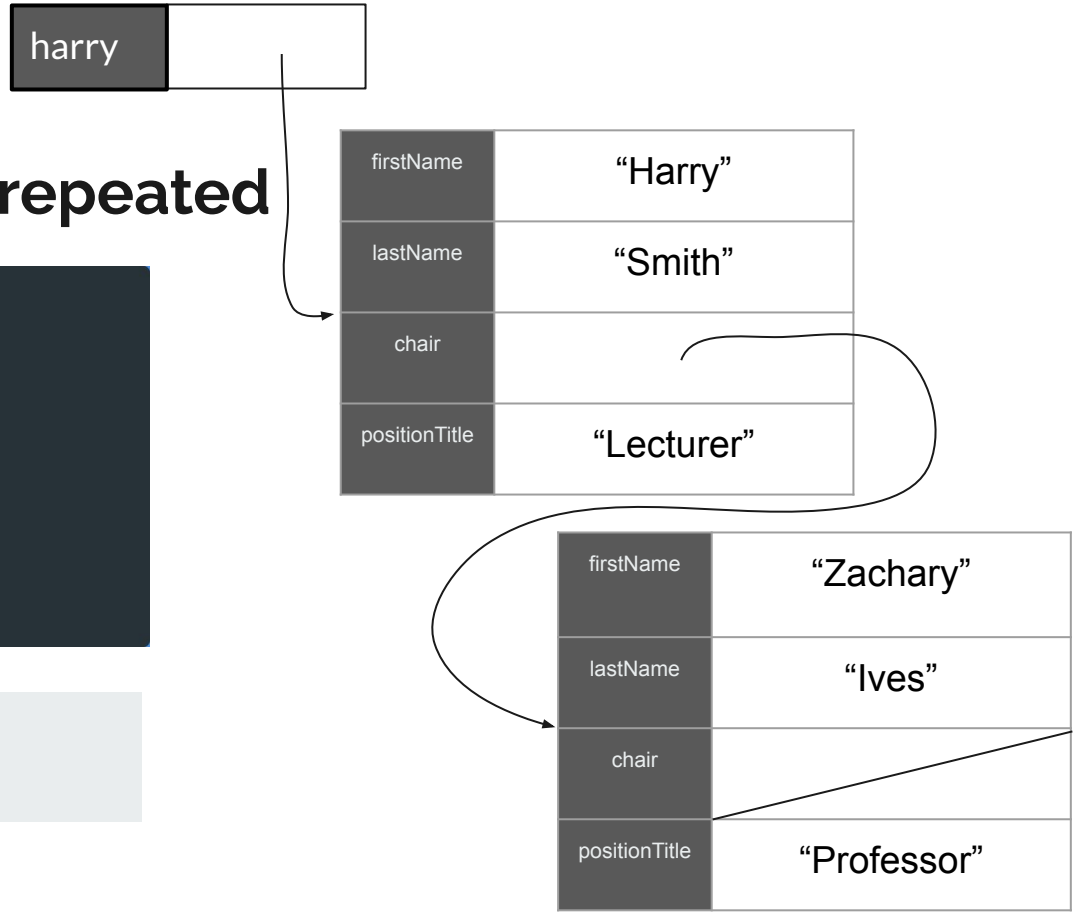
```
Instructor harry = new Instructor("Harry", "Smith", 24, "Lecturer");

System.out.println(harry.lastName + ", " + harry.firstName)
```



Dereferencing can be repeated

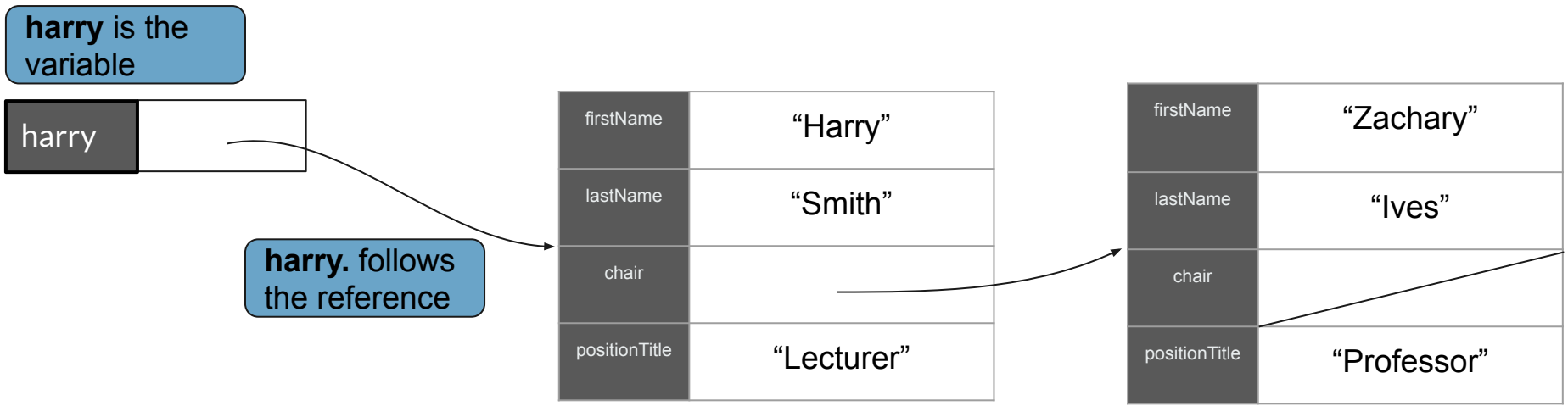
```java
public class Instructor {
  private String firstName;
  private String lastName;
  private Instructor chair;
  private String positionTitle;

  ...
}
```

Replace the age field with the Instructor's department chair...



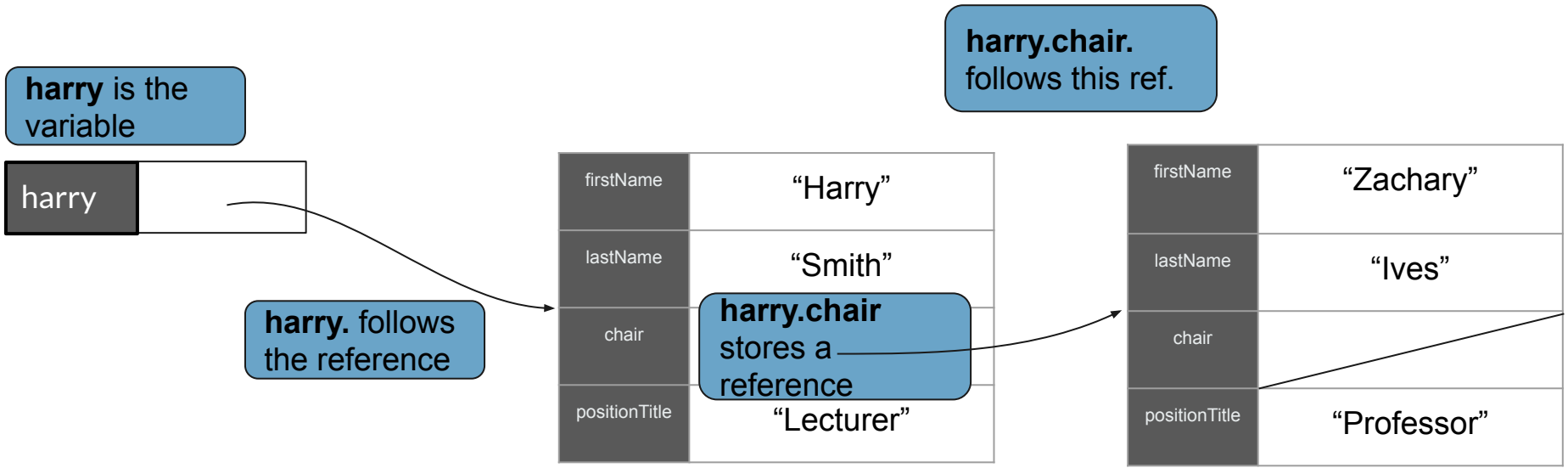
Dereferencing can be repeated

```
harry.chair.lastName
```



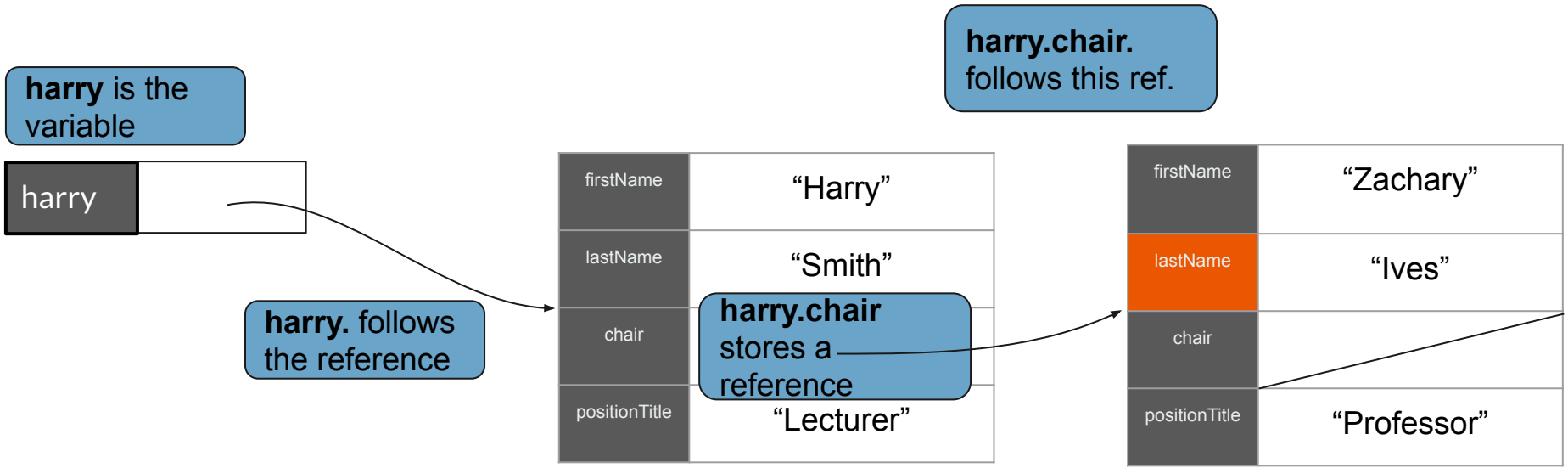
Dereferencing can be repeated

```
harry.chair.lastName
```



Dereferencing can be repeated

```
harry.chair.lastName
```



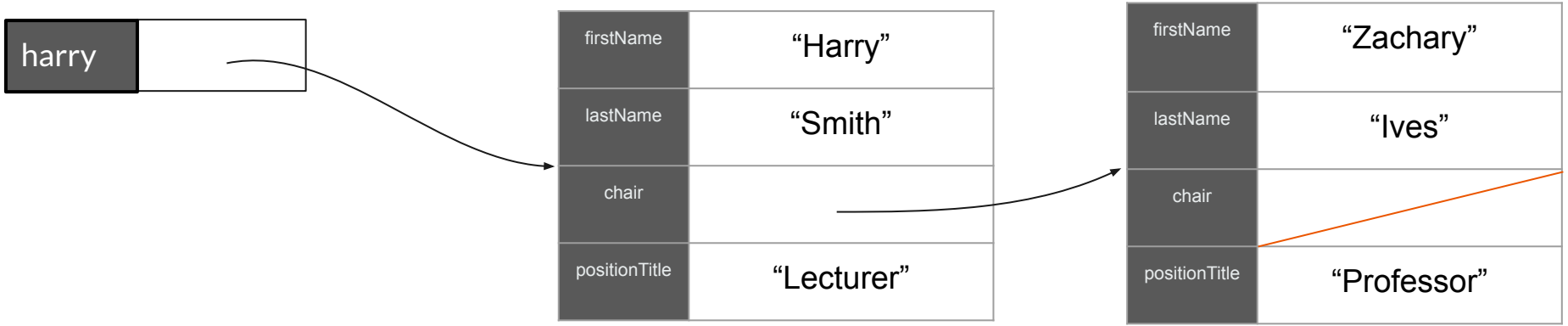
What is the value of harry.chair.chair? (This is tricky!)

What is the value of harry.chair.chair.firstName? (Even trickier.)

Dereferencing a null pointer gives a NullPointerException

```
harry.chair.chair.lastName
```

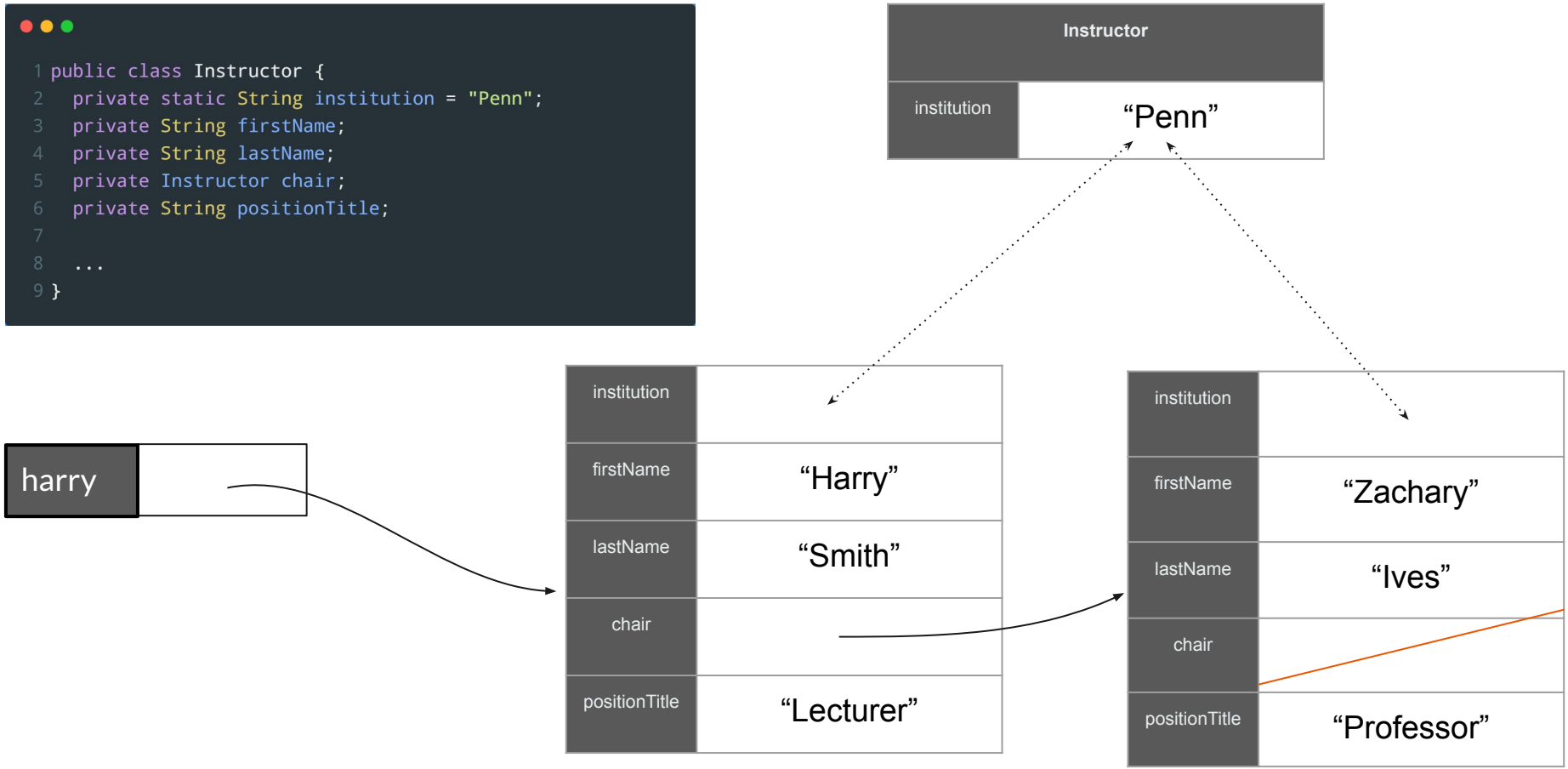
harry.chair.chair is null, so **harry.chair.chair.lastName** asks Java to follow a reference that doesn't exist!



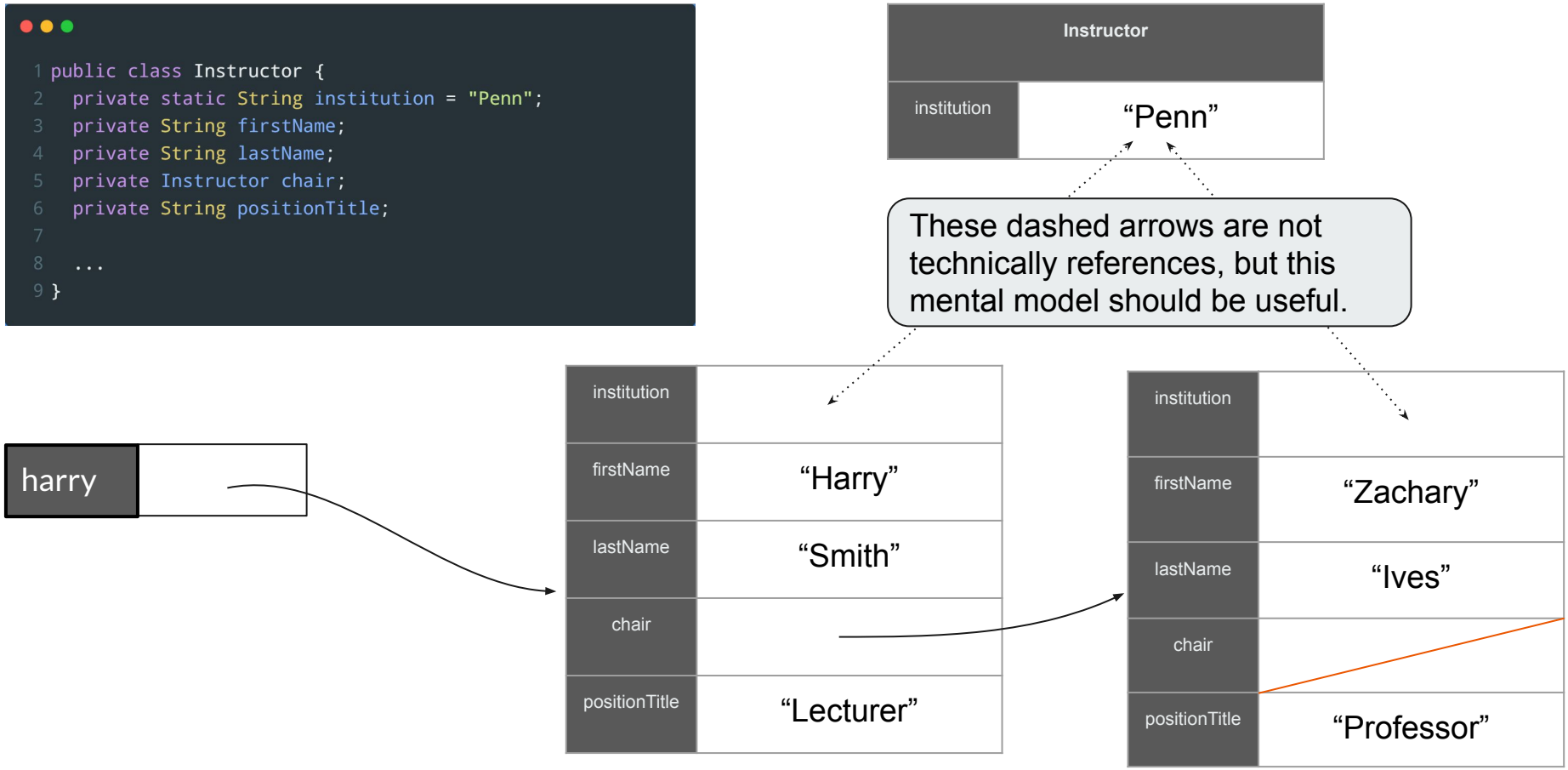
And finally... STATIC!

Static fields and methods belong to the entire Class. All objects of that Class share its same Static values.

```java
public class Instructor {
  private static String institution = "Penn";
  private String firstName;
  private String lastName;
  private Instructor chair;
  private String positionTitle;

  ...
}
```



```java
public class Instructor {
  private static String institution = "Penn";
  private String firstName;
  private String lastName;
  private Instructor chair;
  private String positionTitle;

  ...
}
```



Only static fields can be referenced inside of static methods.