Designing a Class, Adding Fields, Methods and Constructors

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Agenda

The elements you can add to a class

That participate to the behavior

And the state of this class

Fields

Methods

Constructors

And visibility modifiers

Adding Fields, Methods and Constructors

```java
class City {

    private String name;
    private int population;










}
```

```
private String name;
private String name;
```

Visibility modifier

```
    private
    protected
    public
```

no modifier

```
private String name;
```

$$\boxed{\texttt{String}}$$
$$\downarrow$$

Visibility modifier	**Type**
`private`	- **Primitive type**
`protected`	- **Class**
`public`	- **Abstract class**
no modifier	- **Interface**

```
private String name;
```

Visibility modifier

`private`
`protected`
`public`
no modifier

Type

- **Primitive type**
- **Class**
- **Abstract class**
- **Interface**

The name of the field

- **Cannot start with a number**
- **Cannot be an underscore**

```
private String name;
```

Visibility modifier

```
private
protected
public
```
no modifier

Type

- **Primitive type**
- **Class**
- **Abstract class**
- **Interface**

The name of the field

- **Cannot start with a number**
- **Cannot be an underscore**

```java
class City {

    private String name;
    private int population;

    public String getName() {
        return this.name;
    }

    public int getPopulation() {
        return this.population;
    }

    public int raiseBy(int percentage) {
        this.population = this.population*(100 + percentage)/100;
        return this.population;
    }
}
```

```
private String getName()
```

Visibility modifier

`private`
`protected`
`public`
no modifier

Returned Type

- **Primitive type**
- **Class**
- **Abstract class**
- **Interface**

The name of the method

- **With 0 or more parameters**

```
private void setName(String name)
```

Visibility modifier

`private`
`protected`
`public`
no modifier

Returned Type

- **Primitive type**
- **Class**
- **Abstract class**
- **Interface**

The name of the method

- **With 0 or more parameters**

The signature of a method is made of:

- its name
- its argument list

The returned type is not part of the signature

```java
class City {

    private String name;
    private int population;

    public City() {
        this.name = "";
        this.population = 0;
    }
}
```

```
private City()
```

Visibility modifier

```
private
protected
public
```

no modifier

No Returned Type

- **0 or more parameters**

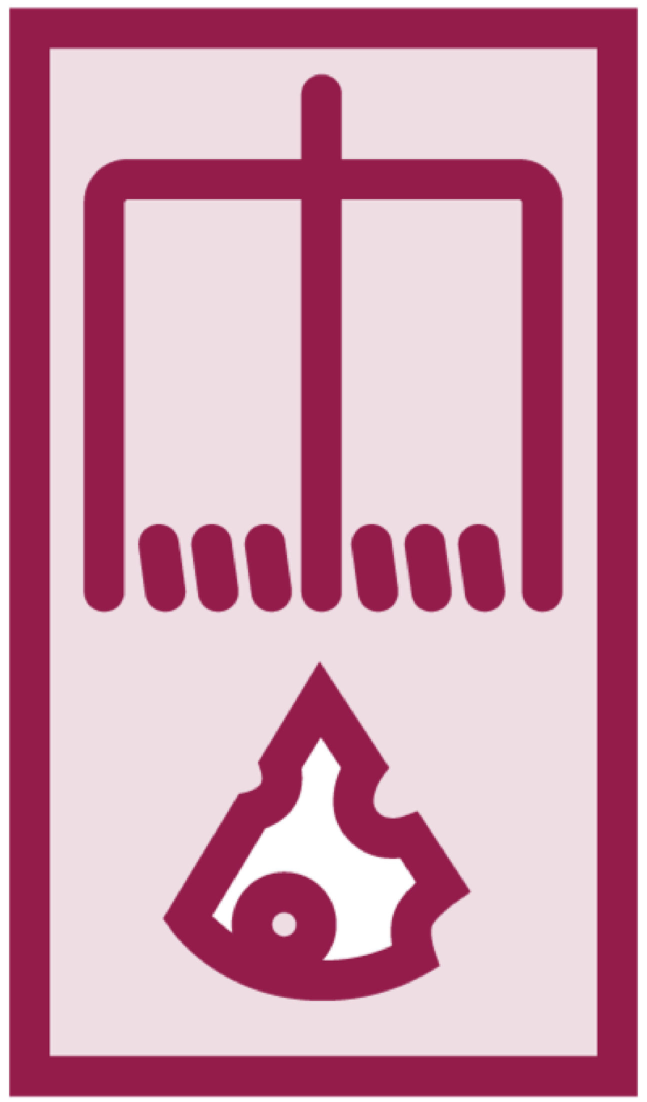
A Constructor has the same name as the class

Methods and Constructors can be overloaded

Overloading consists in creating another method or constructor

With the same name and returned type

But with a different set of parameters



The signature is:

- the name of the method
- the set of parameters

What is not part of the signature:

- the returned type
- the thrown exceptions
- the visibility modifier

Hiding and Exposing Class Members

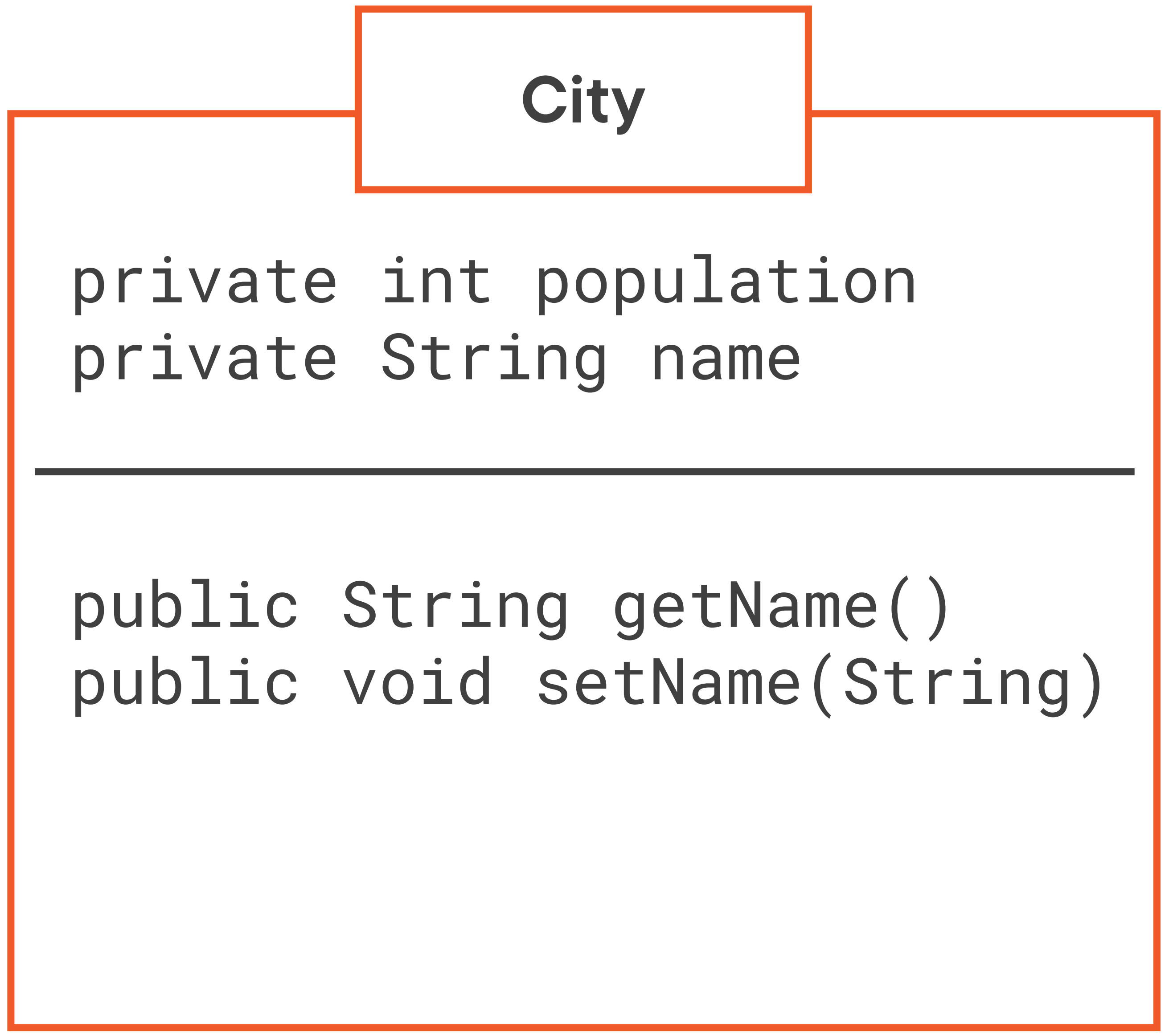
There are four ways to modify the visibility of a class member:

- private

- protected

- public

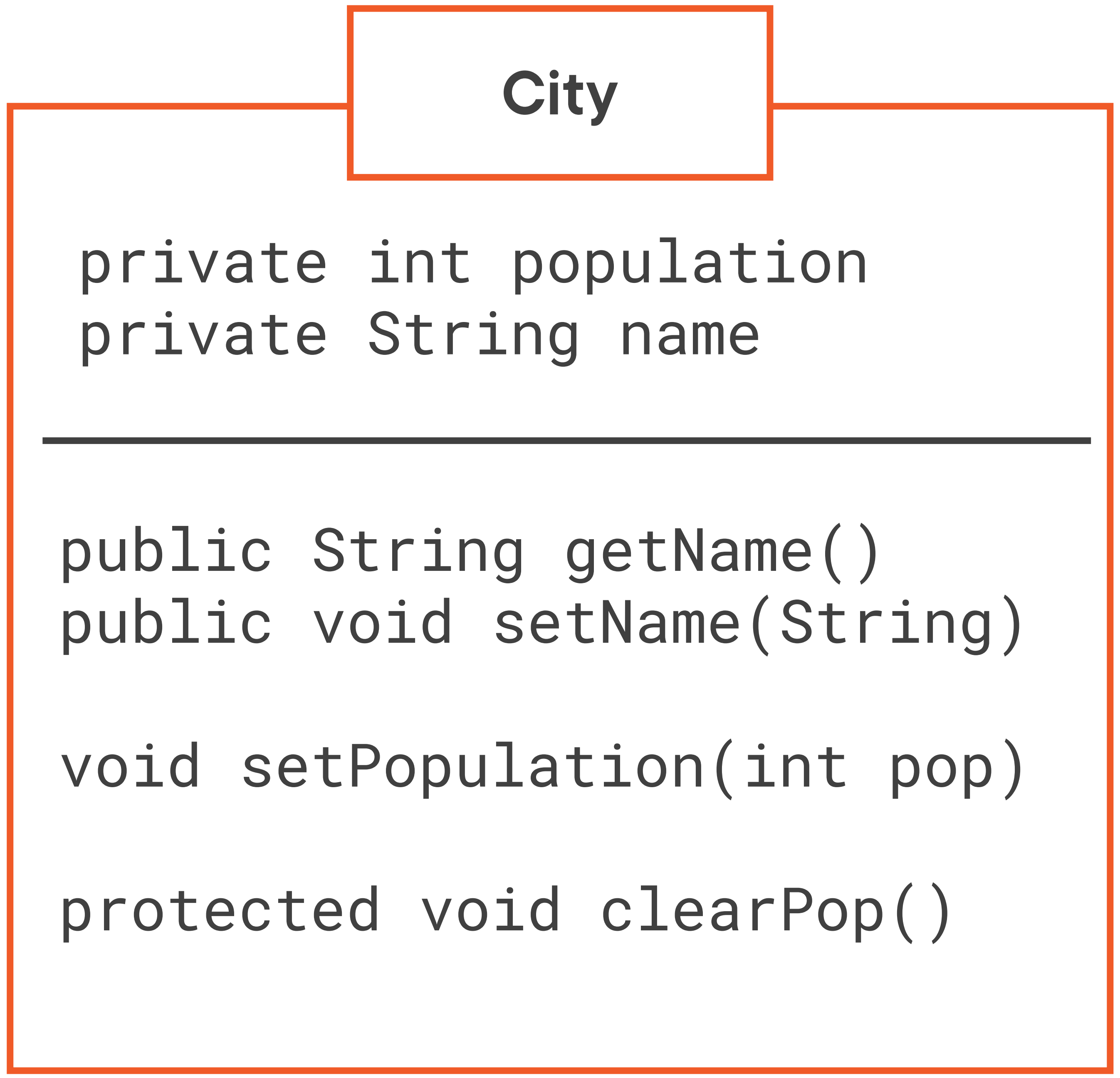
- no visibility modifier = package protected



```
private int population
private String name

public String getName()
public void setName(String)
```

private: hidden from outside the class

public: visible from everywhere



```
private int population
private String name

public String getName()
public void setName(String)

void setPopulation(int pop)

protected void clearPop()
```

private: hidden from outside the class

public: visible from everywhere

package protected:
    visible from the same package

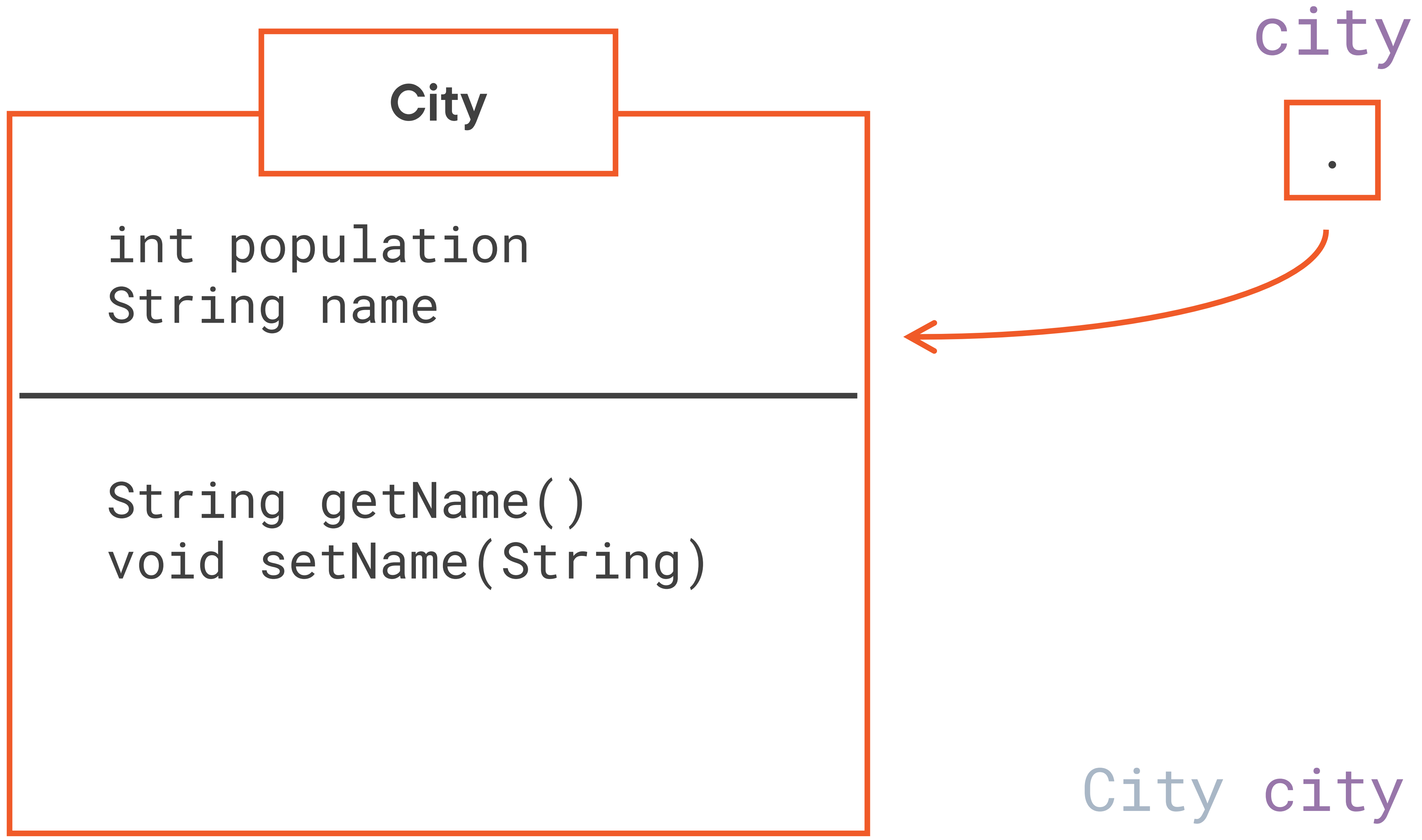
protected:
    visible from the extensions
    and visible from the same package

Laying out Objects in Memory

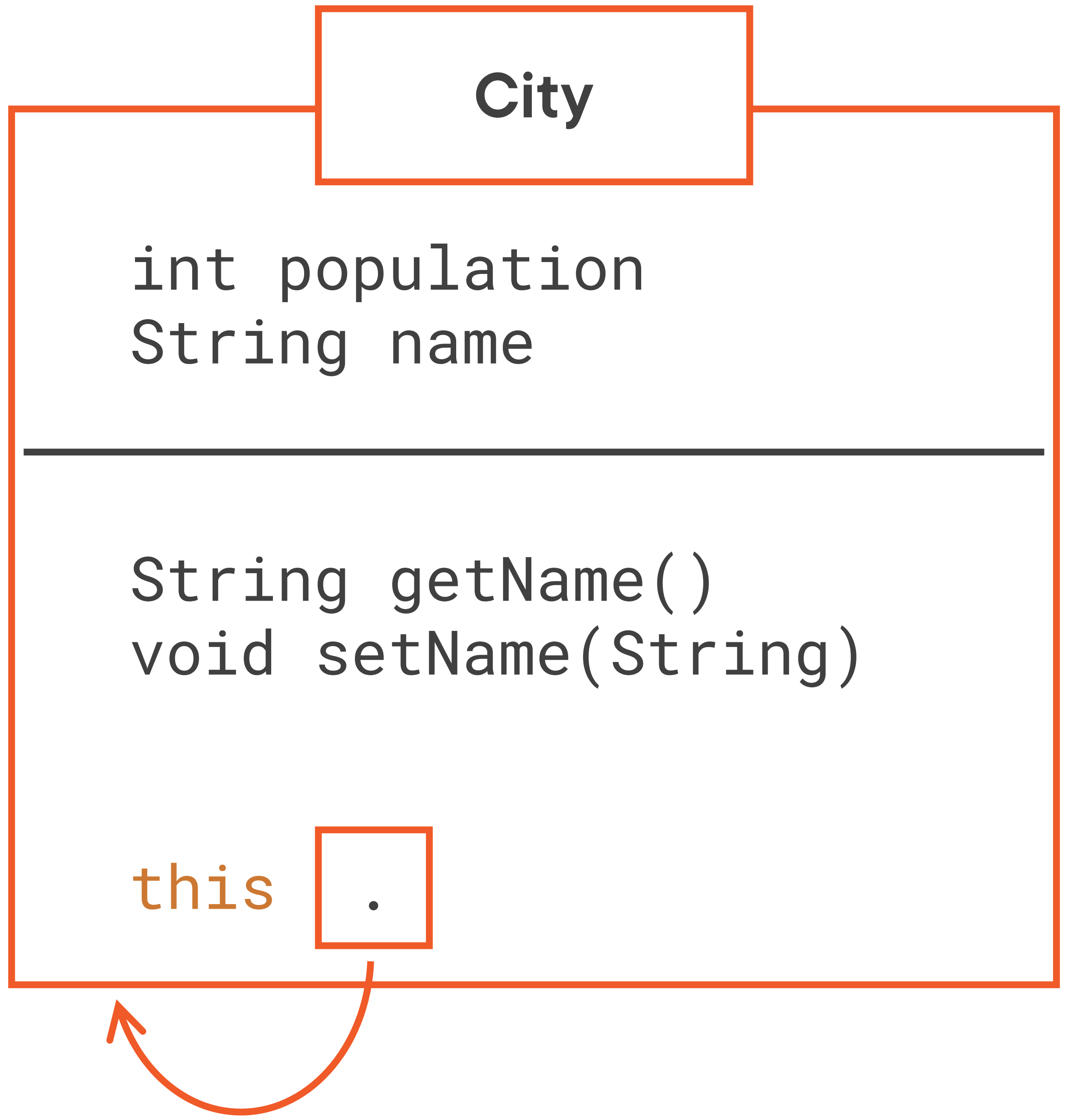
Understanding how these elements are organized in memory

To understand passing by value vs. passing by reference

Makes it easier to understand inheritance and overriding



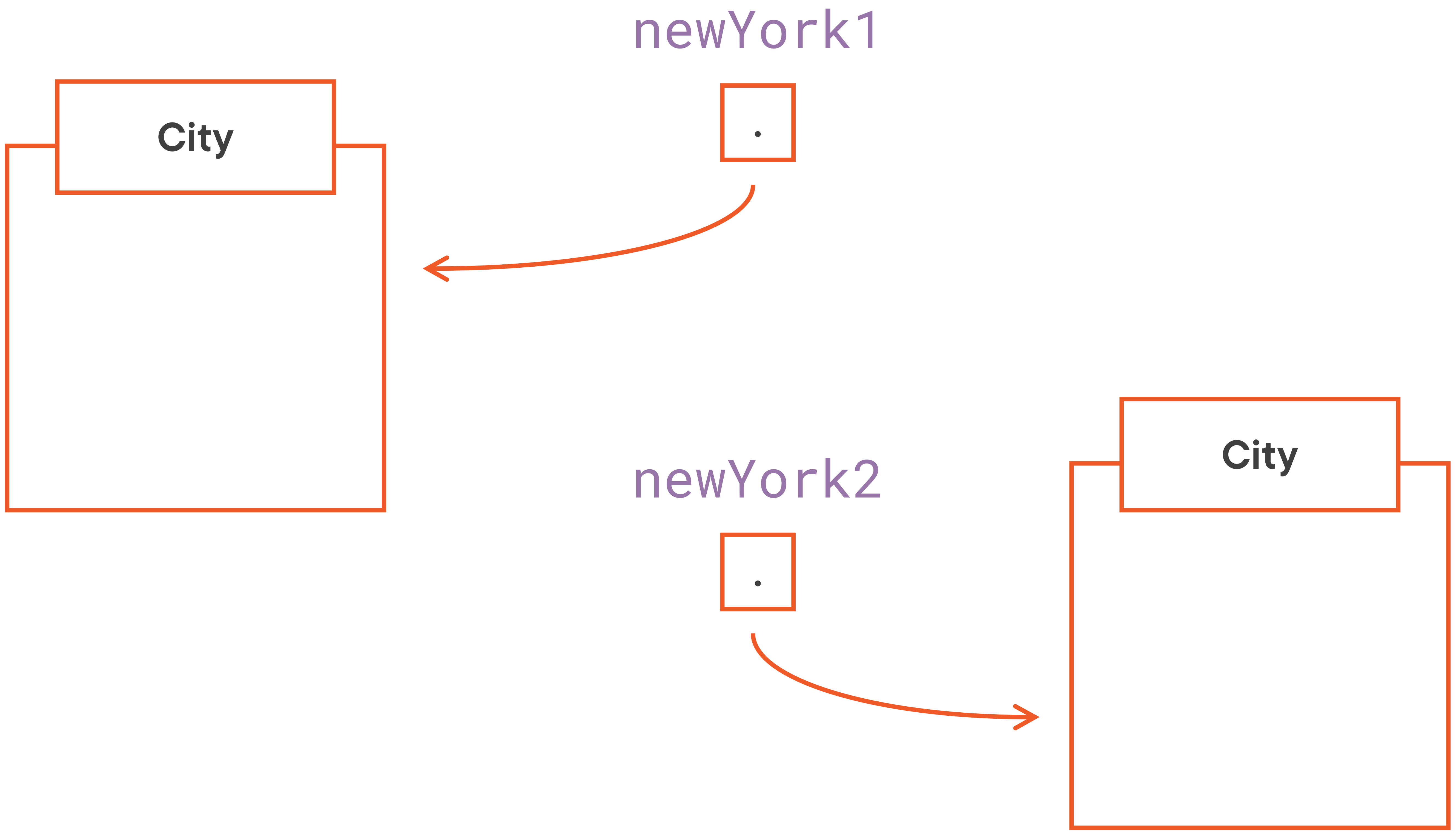
```
City city = new City(...);
```



```
void setName(String name) {
    this.name = name;
}
```

```java
City newYork1 = new City("New York");
City newYork2 = new City("New York");

System.out.println(newYork1 == newYork2);
```

What does this code print out?

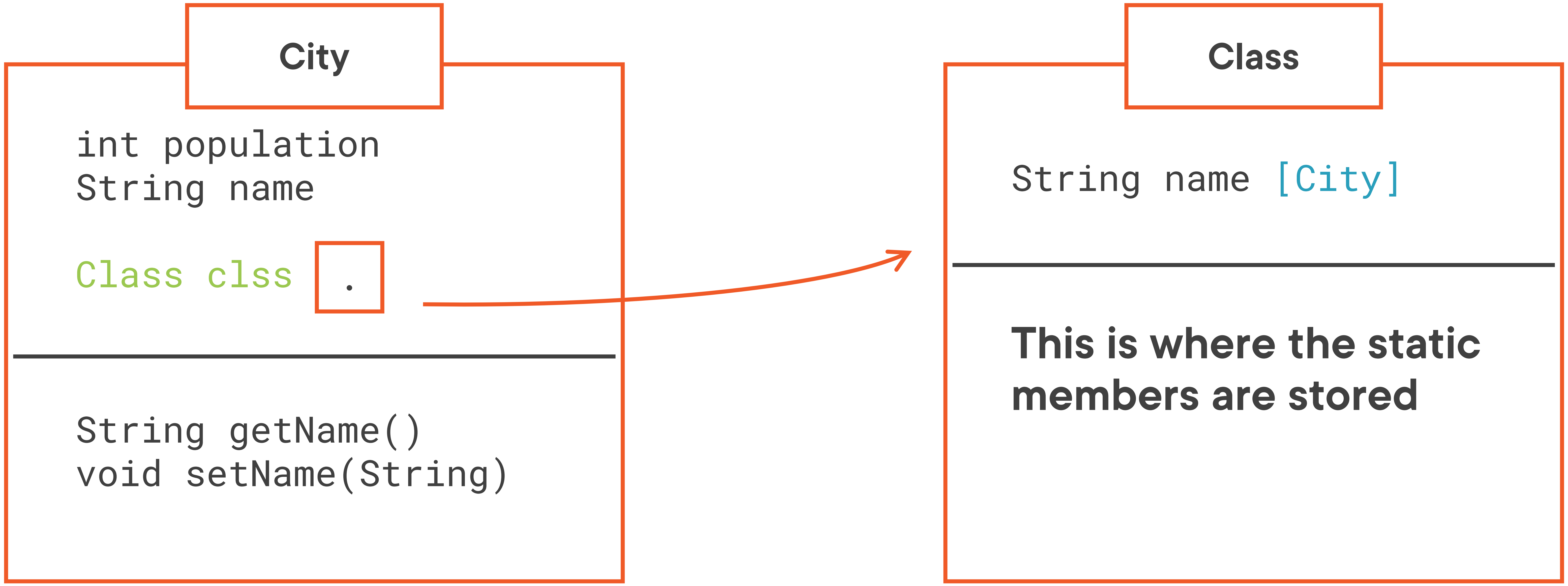

newYork1
City
newYork2
City

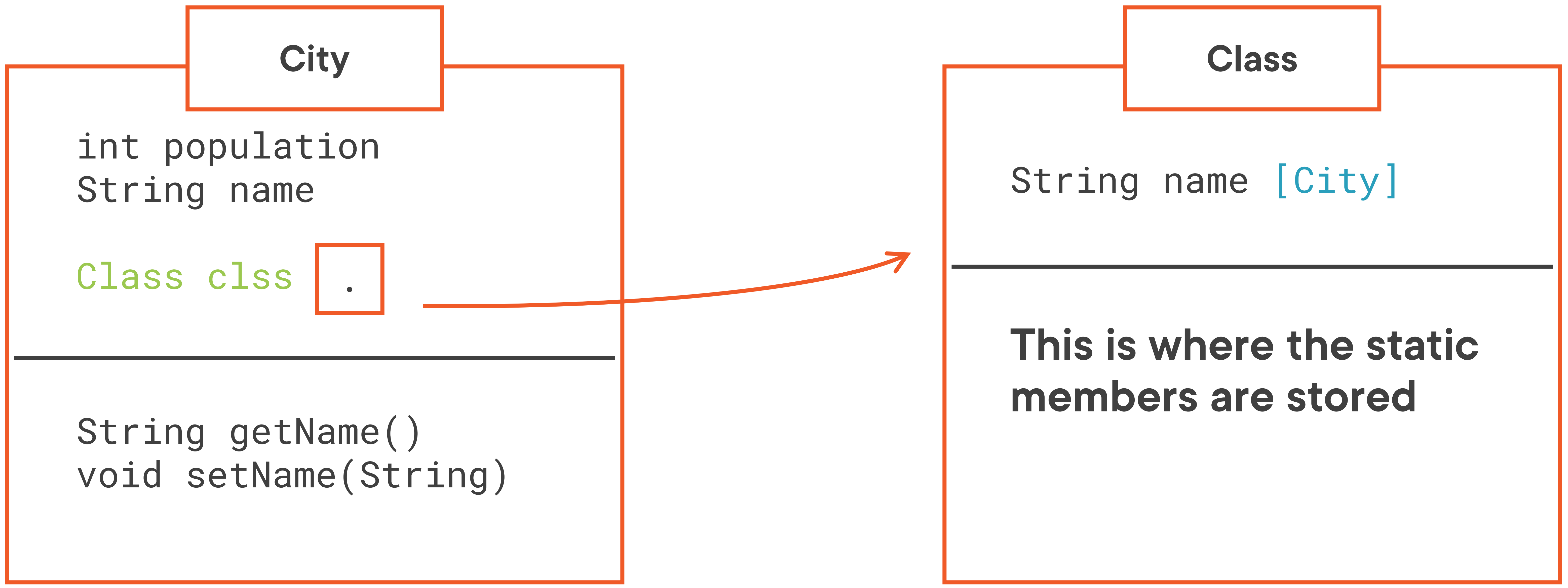
```java
City newYork1 = new City("New York");
City newYork2 = new City("New York");

System.out.println(newYork1 == newYork2);
```

What does this code print out?

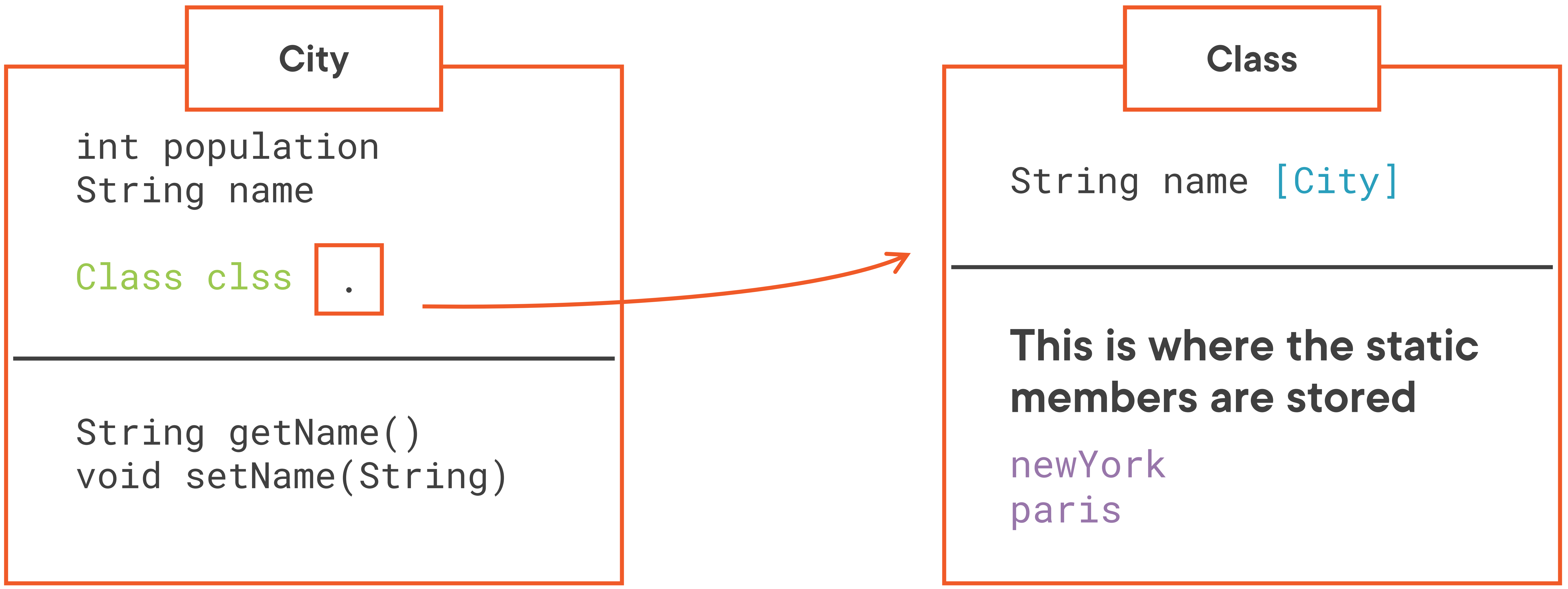
False! Because it compares references

Creating Static Fields and Methods


City
int population
String name
Class clss
.
String getName()
void setName(String)
Class
String name [City]
This is where the static members are stored



```
class City {
   public static City newYork = new City("New York");
   public static City paris   = new City("Paris");

}
```



```
class City {
   public static City newYork = new City("New York");
   public static City paris   = new City("Paris");

}
```

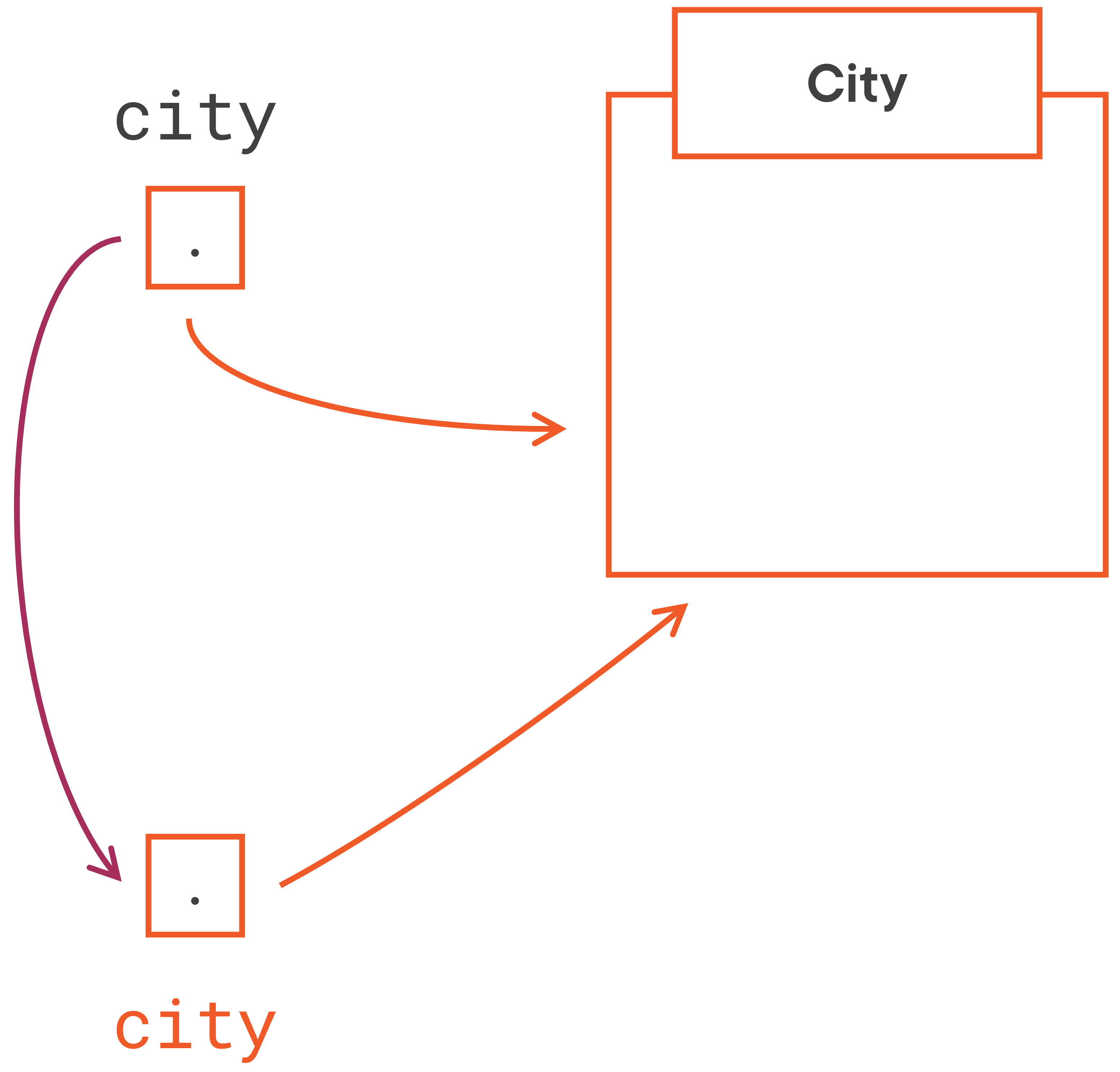
Passing Arguments by Value

Passing by reference? Passing by value?

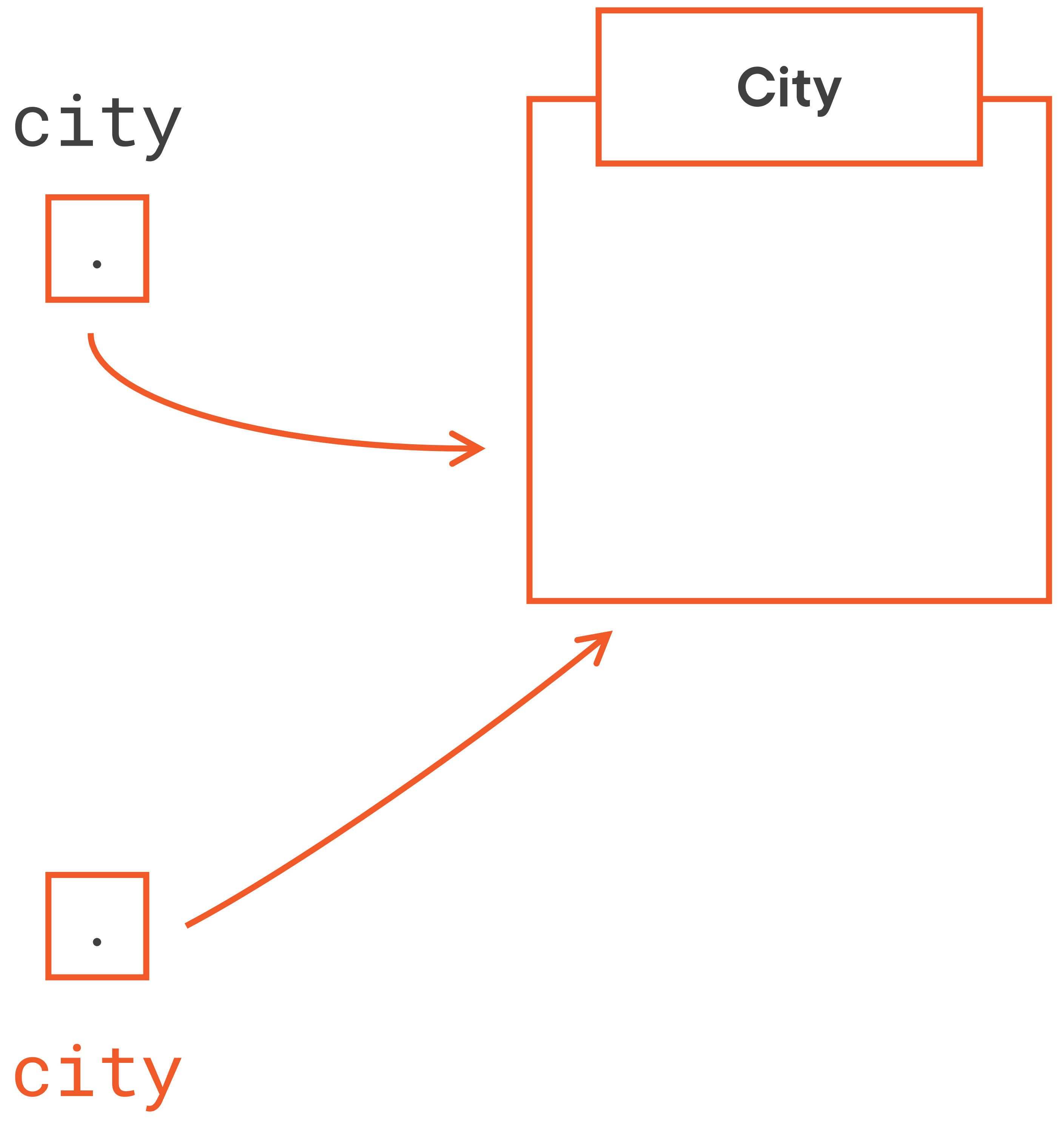
When you pass a parameter to a method

Java copies its value

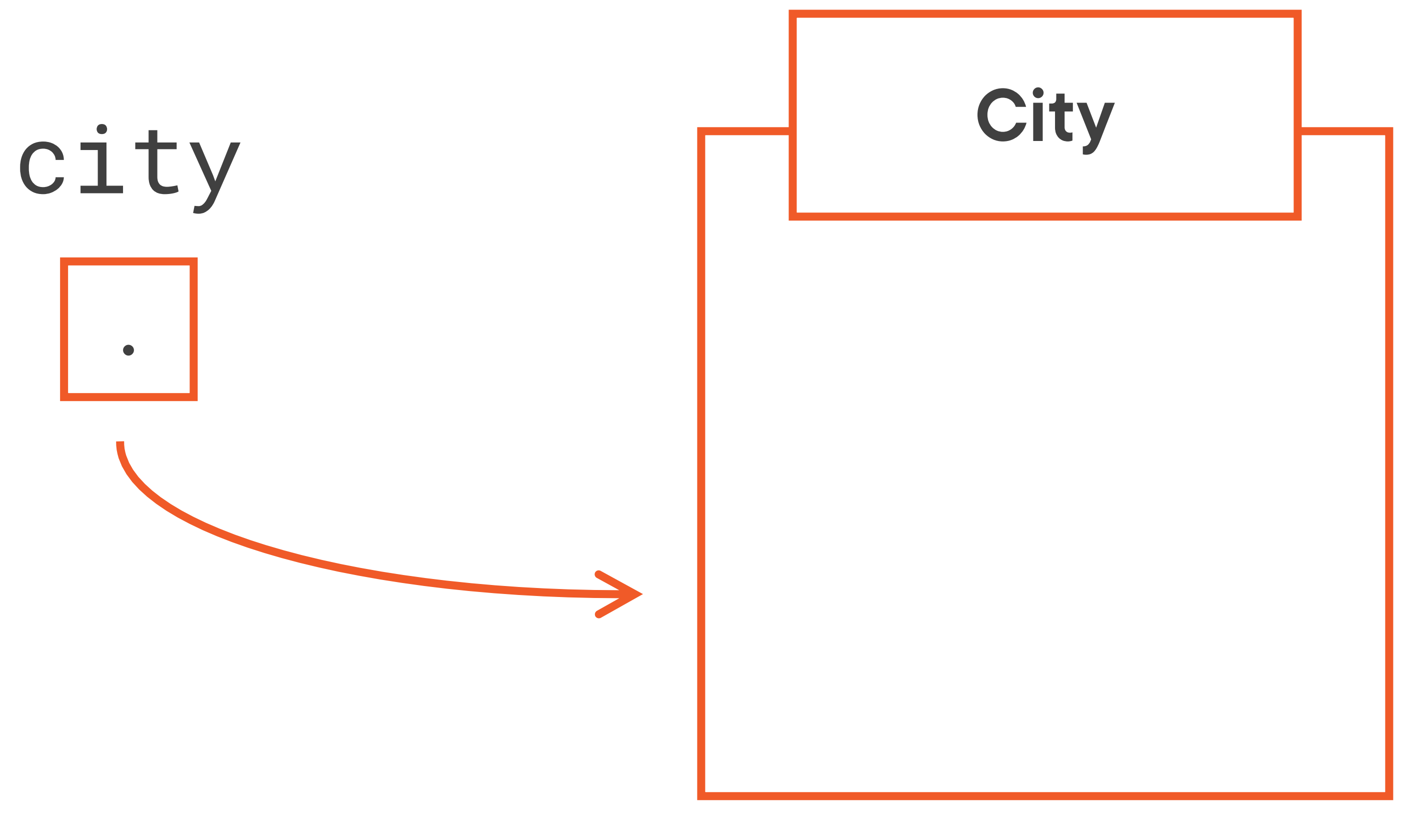
... and only its value!



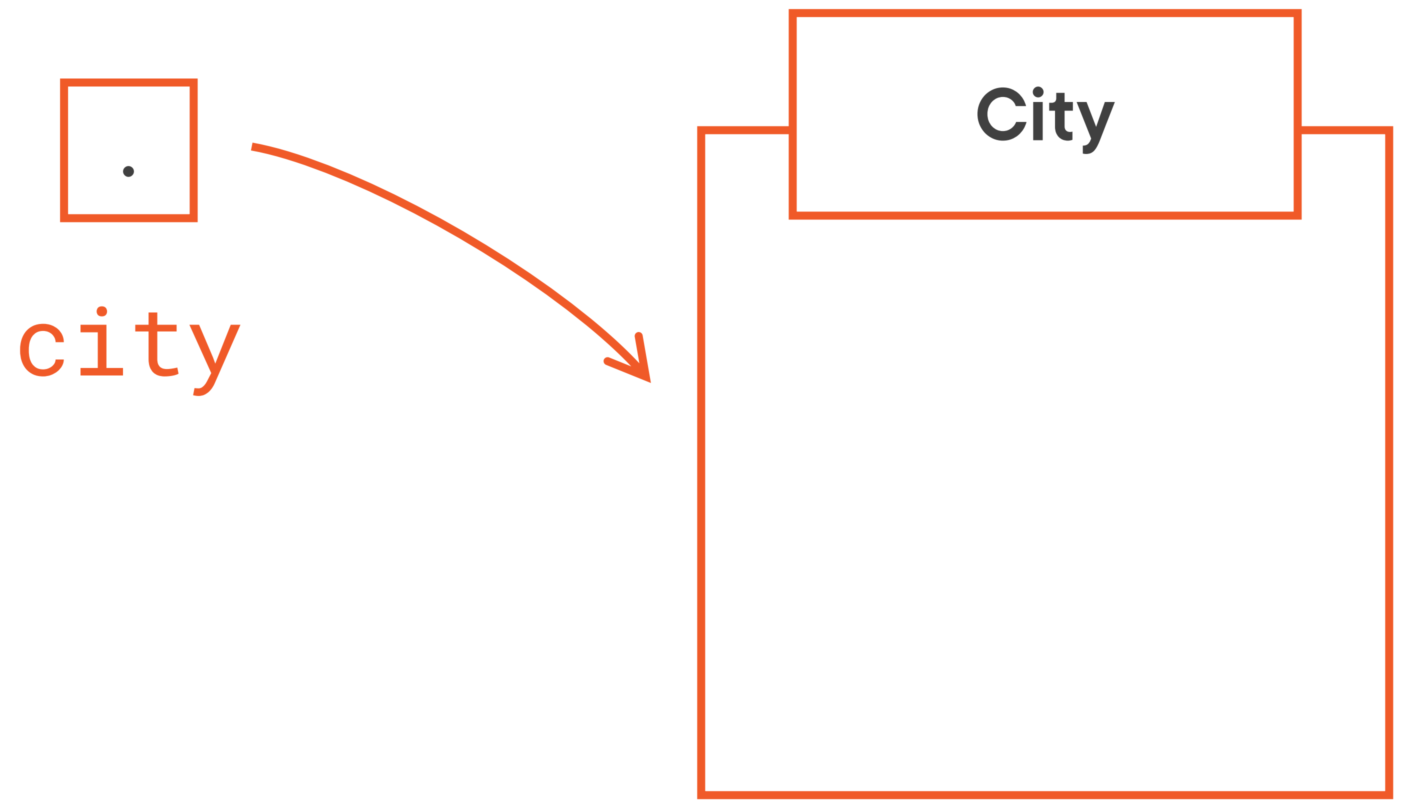
```
doSomethingWith(city);



void doSomethingWith(City city) {

    city.getName();
}
```



```
doSomethingWith(city);


void doSomethingWith(City city) {

    city.getName();
    city = new City("…");
}
```



```
doSomethingWith(city);
```



```
void doSomethingWith(City city) {

    city.getName();
    city = new City("…");
}
```

Passing by reference / passing by value is a subtle concept!

Understanding what is happening in memory is a good way to answer the certification questions correctly

Java passes by value, always

Demo

Live demo!

Let us create classes

And overloads

Module Wrap Up

What did you learn?

Classes, fields, methods, constructors

Visibility modifiers

Signature and Overloading

The this and the static keywords

Java passes by value

Up Next: Extending a Class with Another Class, Creating Abstract Classes