Extending a Class with Another Class, Creating Abstract Classes

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Agenda

Extending class using inheritance

What is inheritance?

What is overriding and polymorphism?

Creating abstract classes

Preventing a class from being extended

Extending a Class

```
public class City {

}

public class Capital extends City {

}
```

The extension expresses a “is a” relationship

It is better to think of a “behaves as” relationship

City is the superclass of Capital

Capital is an extension of City

```java
Capital capital = new Capital();
doSomething(capital);

public void doSomething(City city) {

}
```

If a method takes a class as a parameter

Then you can call it with any object instance of any extending class

By extending a class, you can:

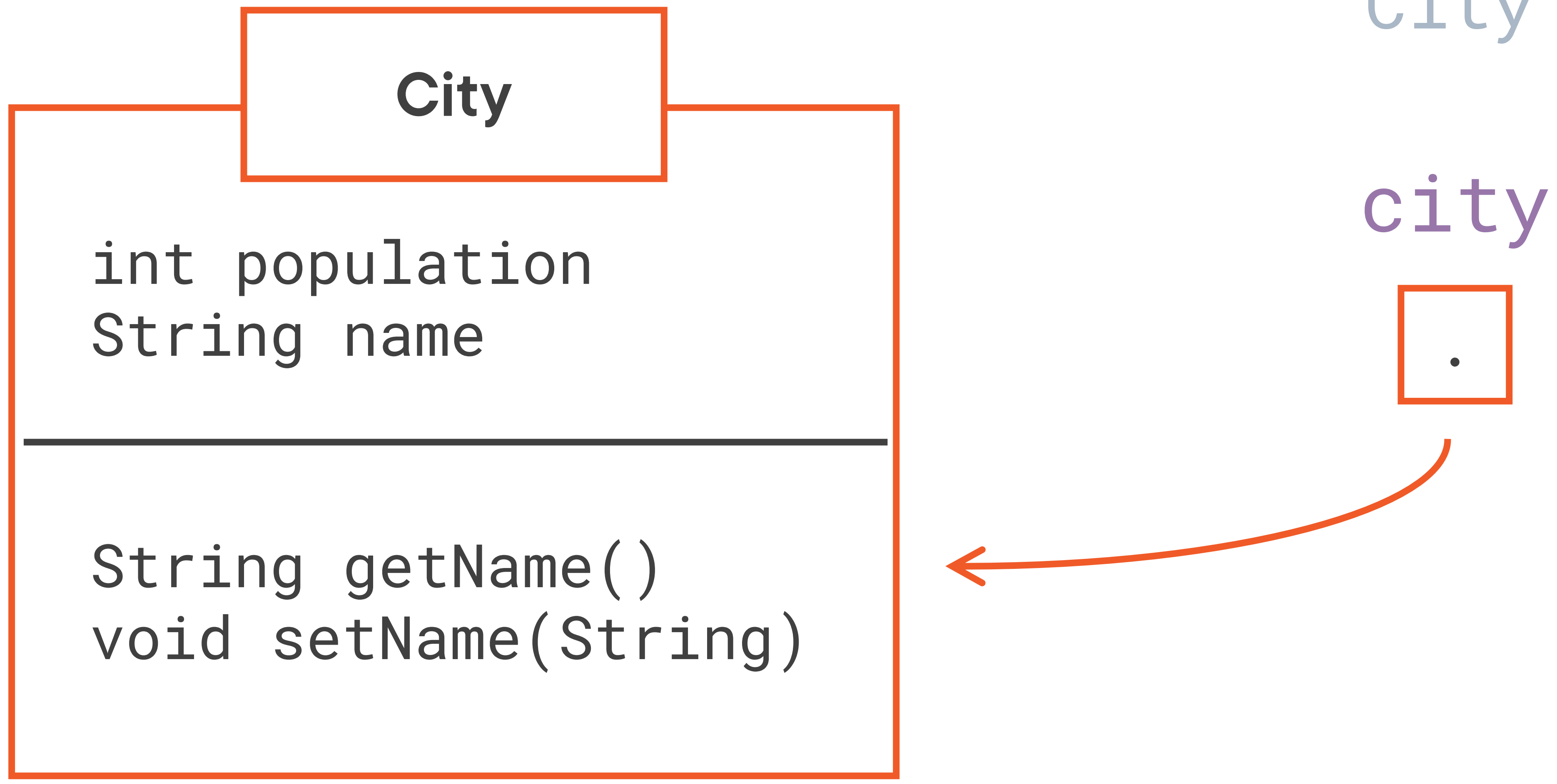
- add fields

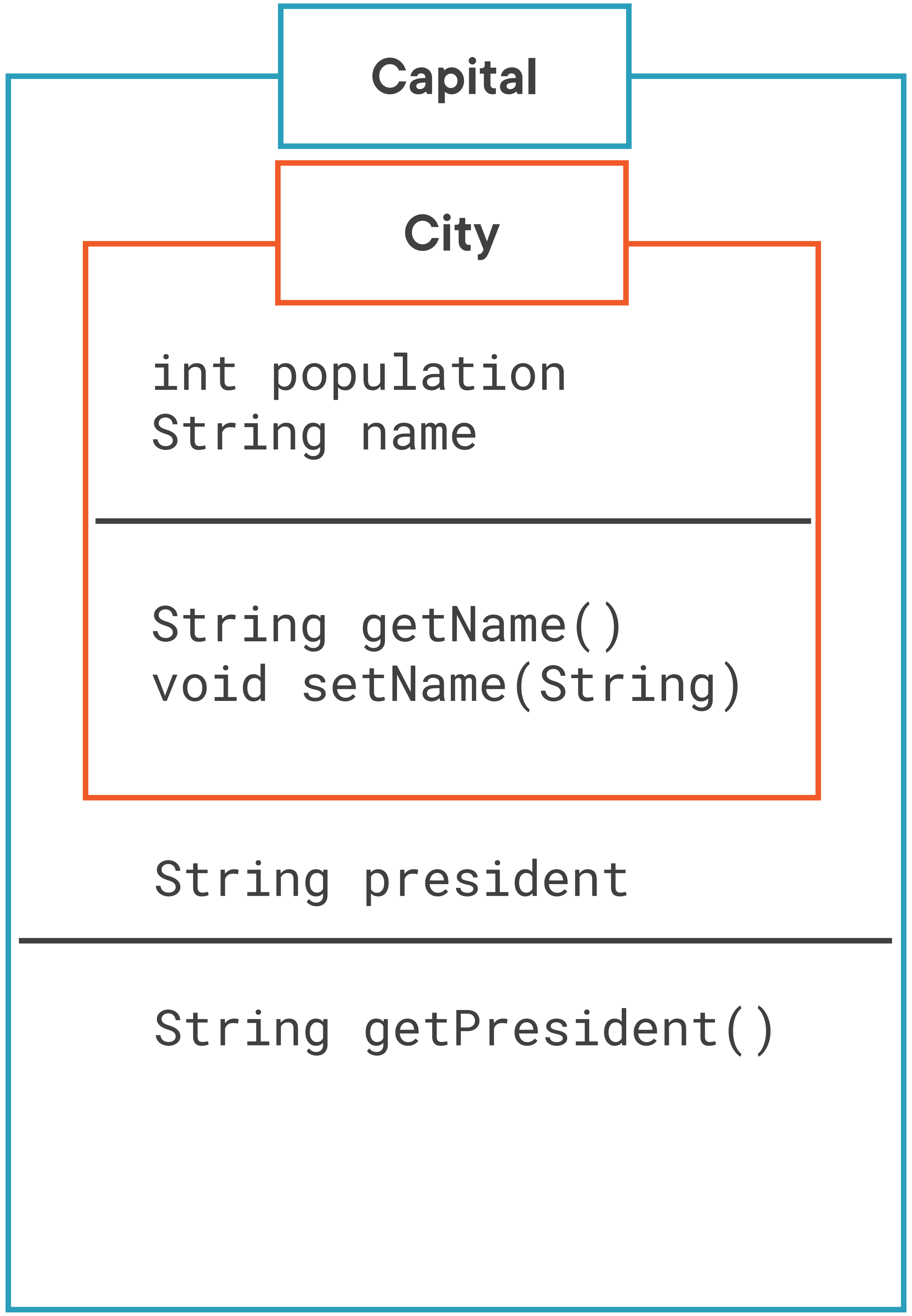
- add methods

- add constructors

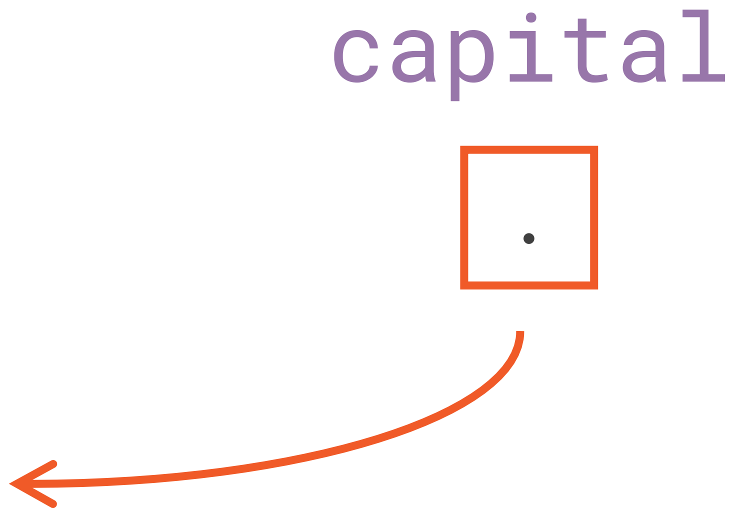
To an existing class

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





```
Capital capital = new Capital(...);
```



All the methods of the City class

are available through

a reference to a Capital object

Inheritance makes the elements of a class available through a reference to an extending class

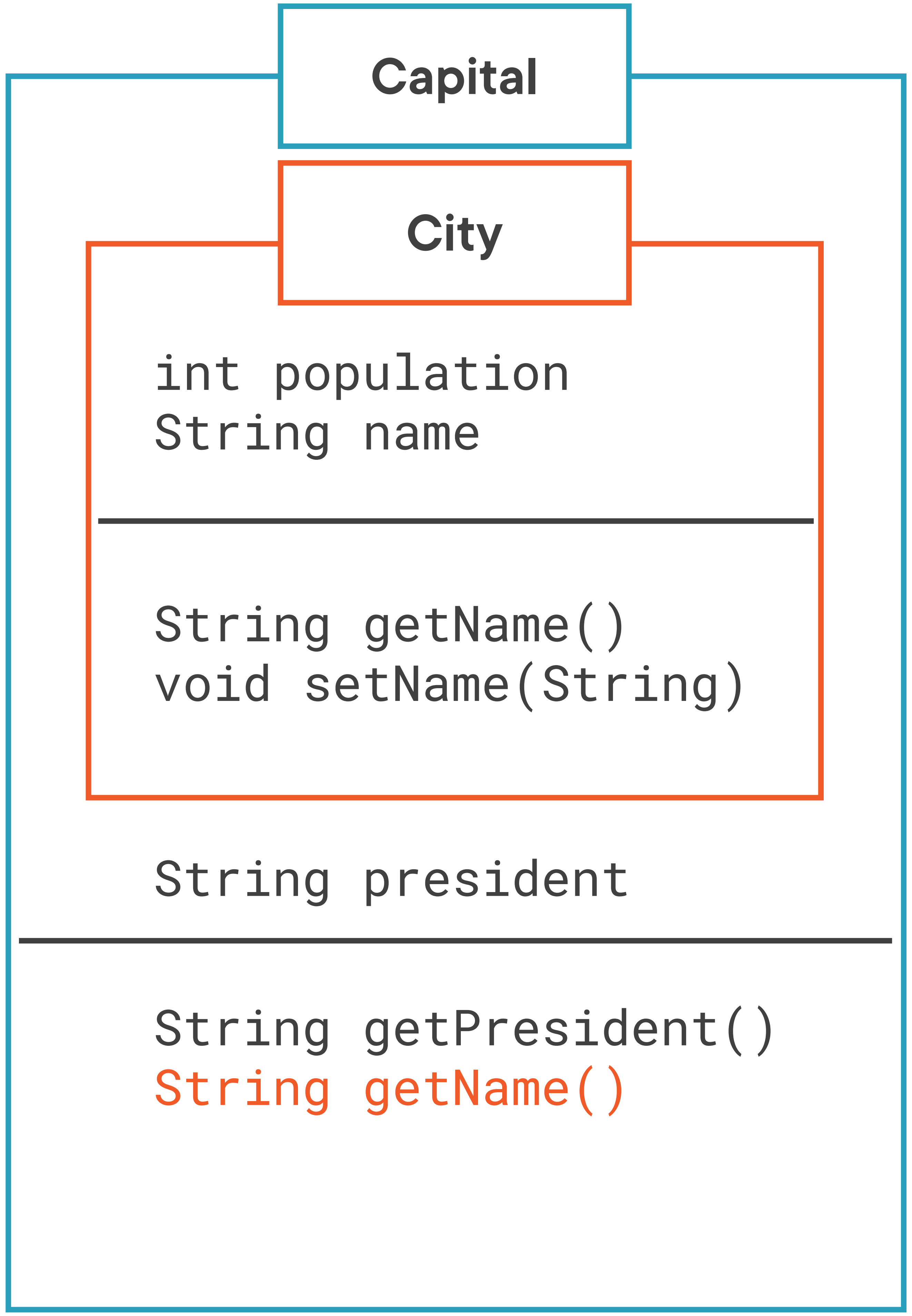
All the visibility rules still apply

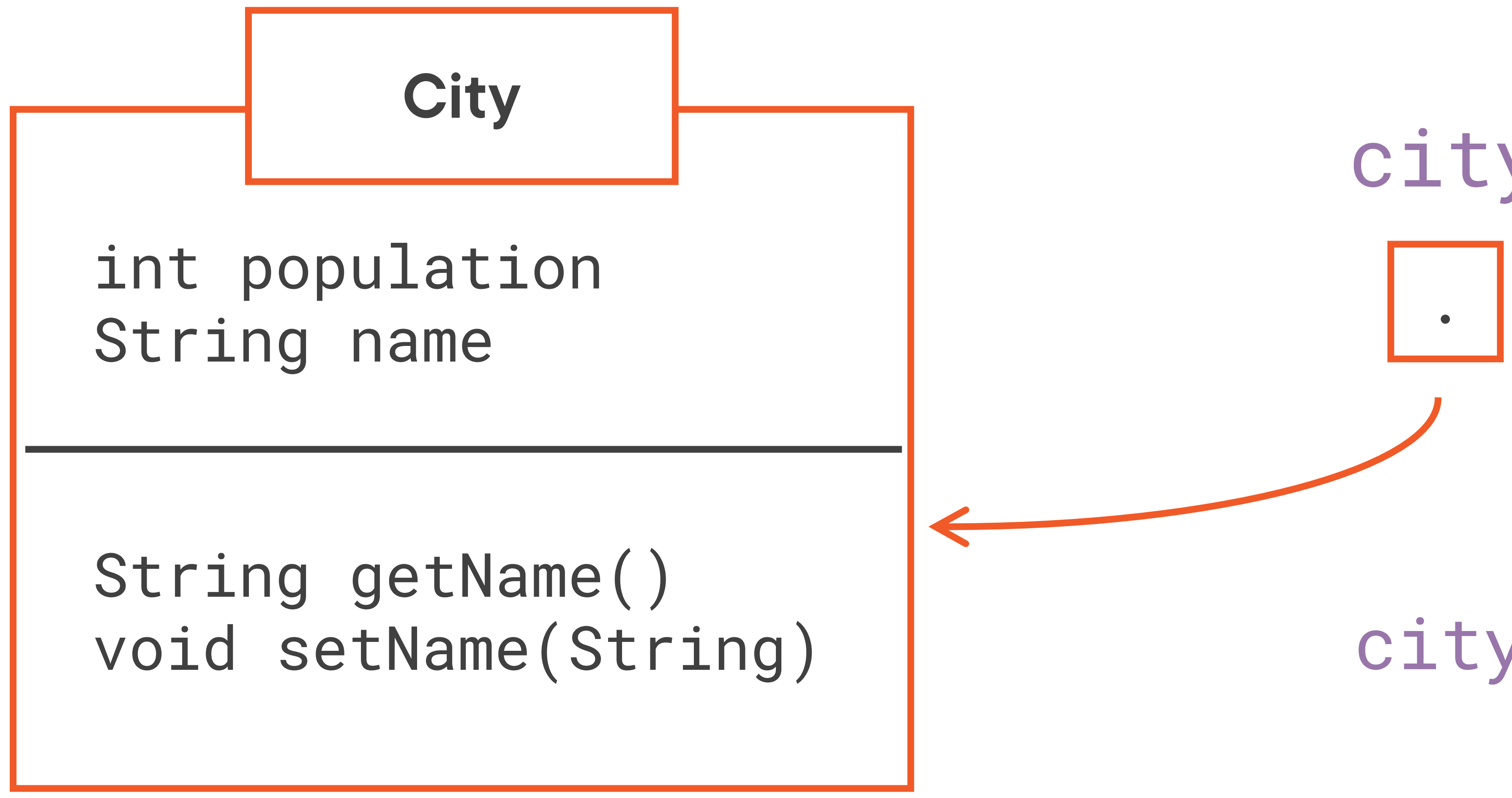
Capital is an external class to City

The private members of City are not accessible to Capital

What is happening if you define a method

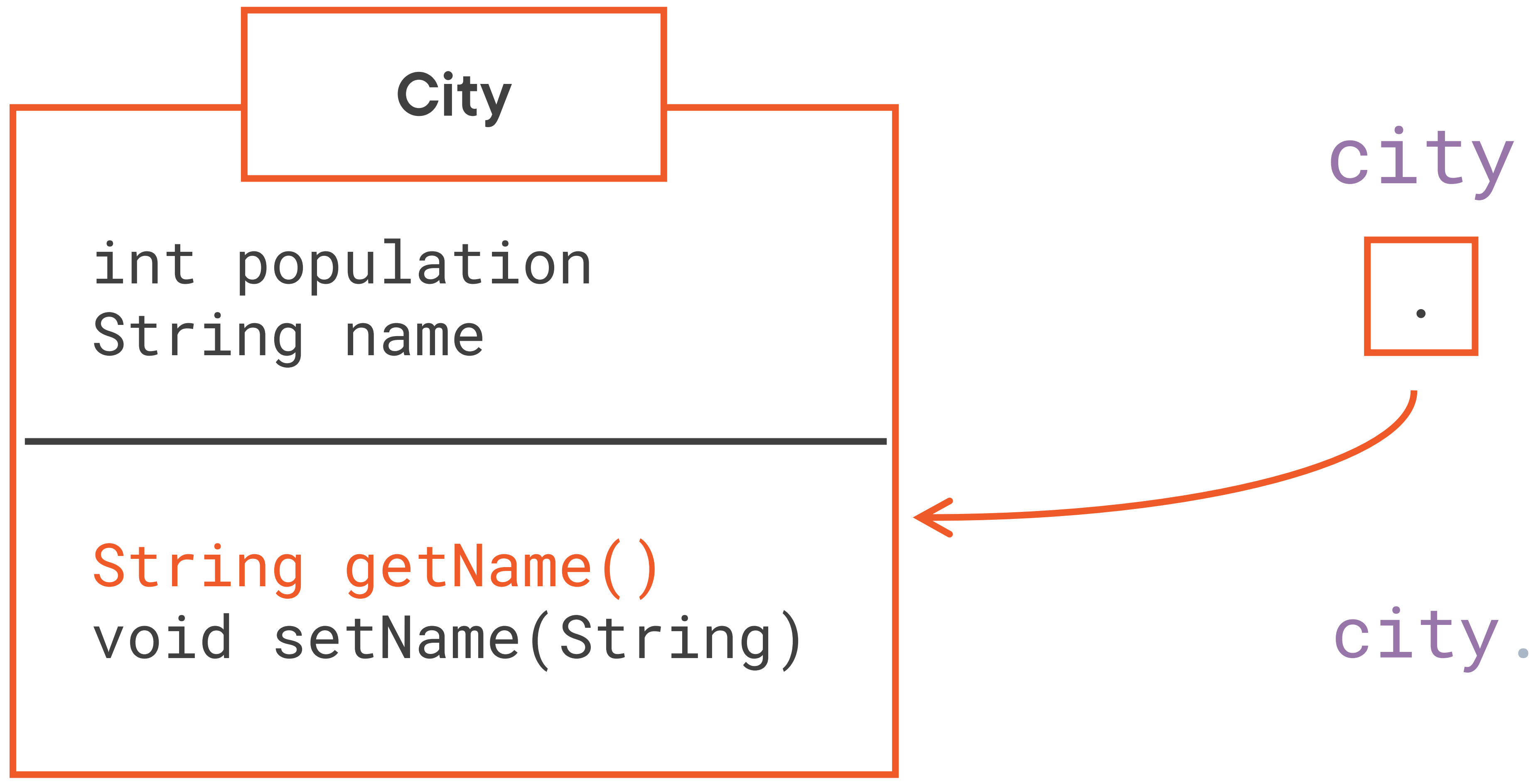
That already exists in a superclass?


Capital
City
int population
String name
String getName()
void setName(String)
String president
String getPresident()
String getName()



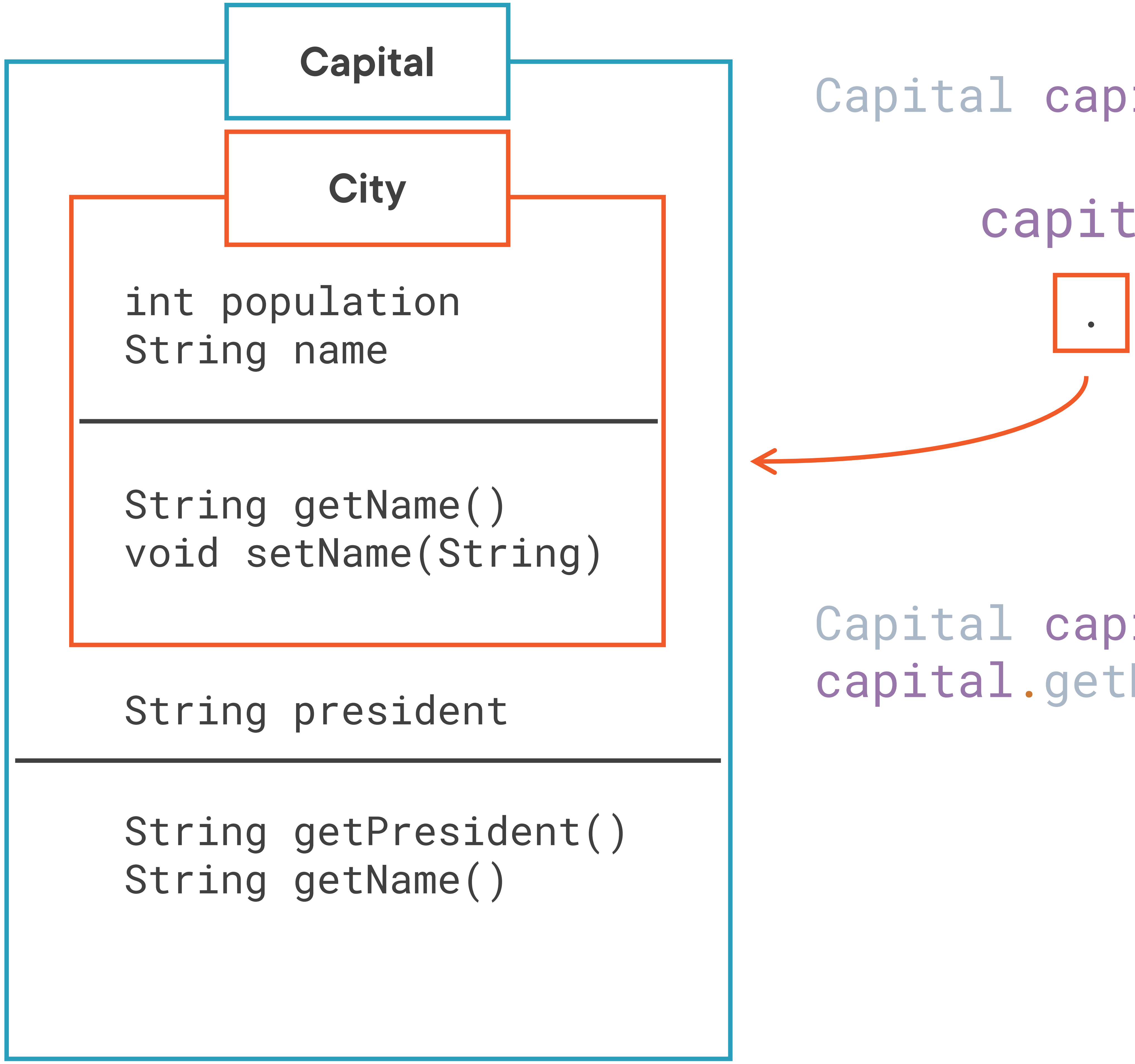
```
City city = new City(...);
```

```
city.getName();
```



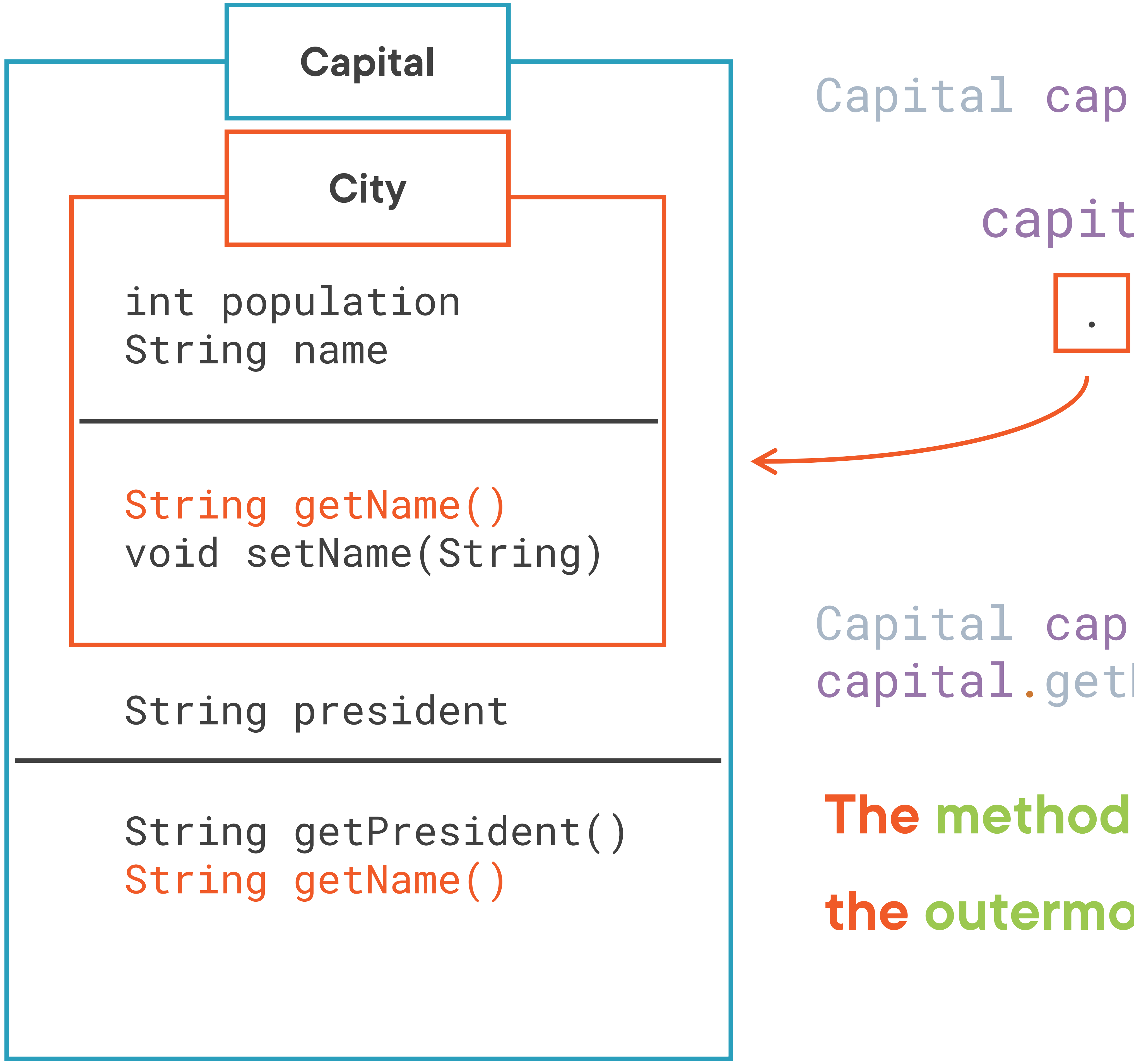
```
City city = new City(...);
```

```
city.getName();
```



```
Capital capital = new Capital(...);
```

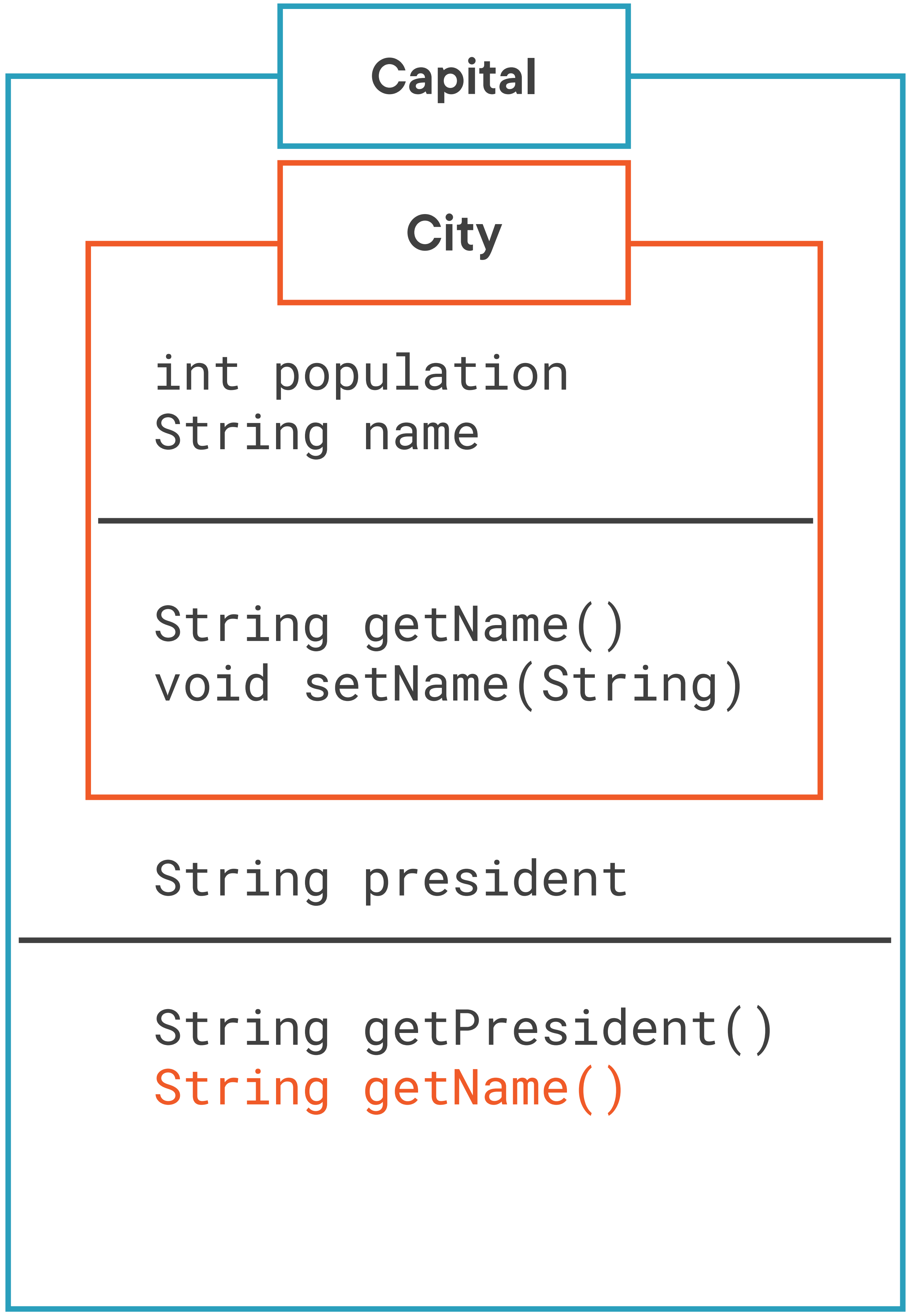
```
Capital capital = new Capital(...);
capital.getName();
```



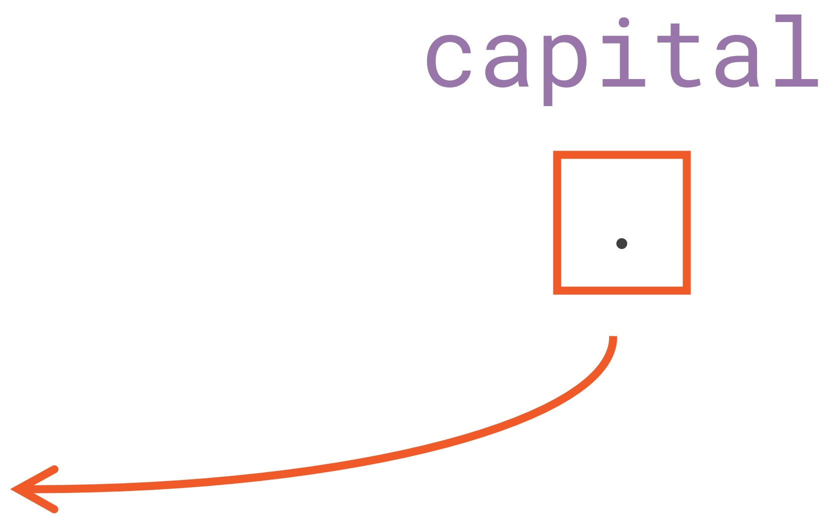
```
Capital capital = new Capital(...);
```

```
Capital capital = new Capital(...);
capital.getName();
```

The method that is called is the outermost method

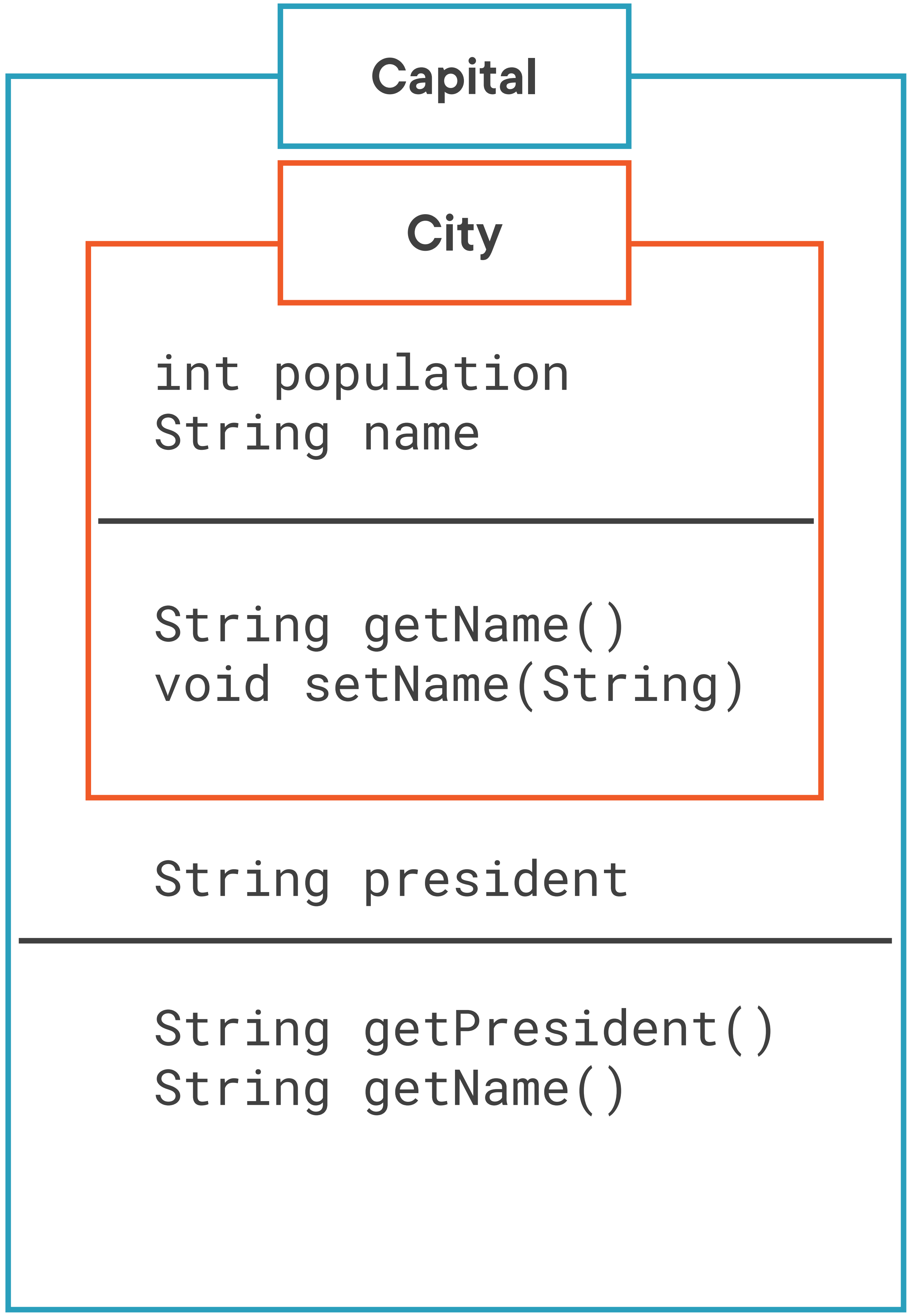


```
Capital capital = new Capital(...);
```



```
Capital capital = new Capital(...);
capital.getName();
```

The method that is called is the outermost method



```java
Capital capital = new Capital(...);
```

```java
Capital capital = new Capital(...);
doSomething(capital);
```

```java
public void doSomething(City city) {
        city.getName();
}
```

Polymorphism:
calls the outermost method
found at runtime

Extending the Object Class

All classes extend the Object class

So, the methods of Object are available on any object

The most important are:

- toString()
- equals() and hashCode()

One is deprecated since Java SE 9

- finalize()

Demo

Seeing polymorphism in action

Creating Abstract Classes

An abstract class is a class

Where some methods have a signature

But no implementation

They must be declared abstract

How can you instantiate abstract classes?

You cannot!

To instantiate an abstract class

You need to extend it with a concrete class

```java
public abstract class AbstractCollection {

    public abstract int size();

    public boolean isEmpty() {
        return size() == 0;
    }
}
```

It is legal to call an abstract method in your code

At runtime, a concrete class will extend this abstract class

And will provide an implementation for the size() method

You cannot instantiate an abstract class

To instantiate it, you need to extend it

With a concrete class

Such a class must provide an implementation for all the abstract methods

It can be convenient to define methods that accept abstract classes

They will in fact accept any extension

Creating Final Classes

The final keyword can be added:

- on a class definition
- on a method definition
- on a field definition

When placed on a class or a method

The final keyword prevents that class or method to be overridden

There are many final classes in the JDK:

- String

- all the wrapper classes: Integer, Double, …

When placed on a field

The final keyword makes that field immutable

Module Wrap Up

What did you learn?

Class inheritance and method overriding

How can a subclass inherit the methods of a superclass

How can a subclass override a method

What is polymorphism

Concrete class vs. abstract class

The final keyword

Up Next: Modeling Object Behavior with Interfaces