Working with Inheritance

Principles of Object-Oriented Programming

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Understanding inheritance in Java to pass the Java SE 8 Programmer I Certification 1Z0-808

How to work with classes and interfaces

How to:

- design a class

- store data in a class

- implement behavior with a class

How to add more behavior or

Add more data

With inheritance

This is a Java course:

Basic knowledge of the language and The Collection framework

Create classes, interfaces

Create a simple application in an IDE

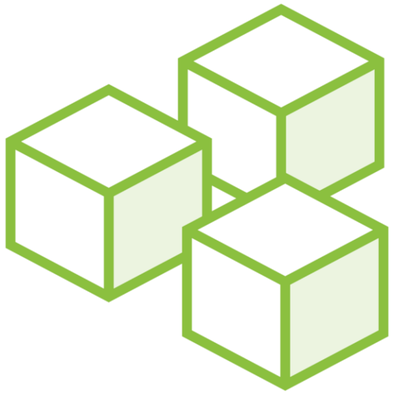
The IDE is IntelliJ

Agenda

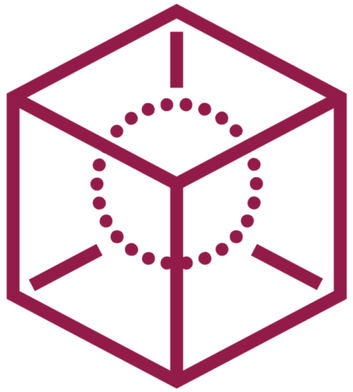
- Introducing the principles of Object-Oriented Programming
- How to design a class to encapsulate data
- How to extend a class with another class
- Introducing interfaces
- What is happening under the hood when you construct an object

What is Object-Oriented Programming?

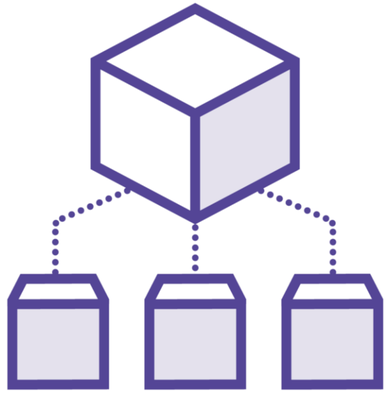
The Four Principles of OOP



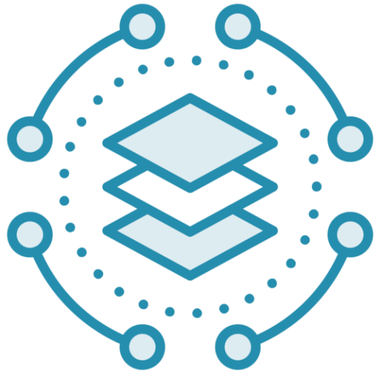
Abstraction



Encapsulation



Inheritance



Polymorphism

Abstracting Concrete Ideas

Abstraction

Abstraction is about finding the concepts you need for the application you are developping

Your application computes statistics on cities

A city is made of so many things...

What you need is only the name of the city

And its population

This is your abstraction

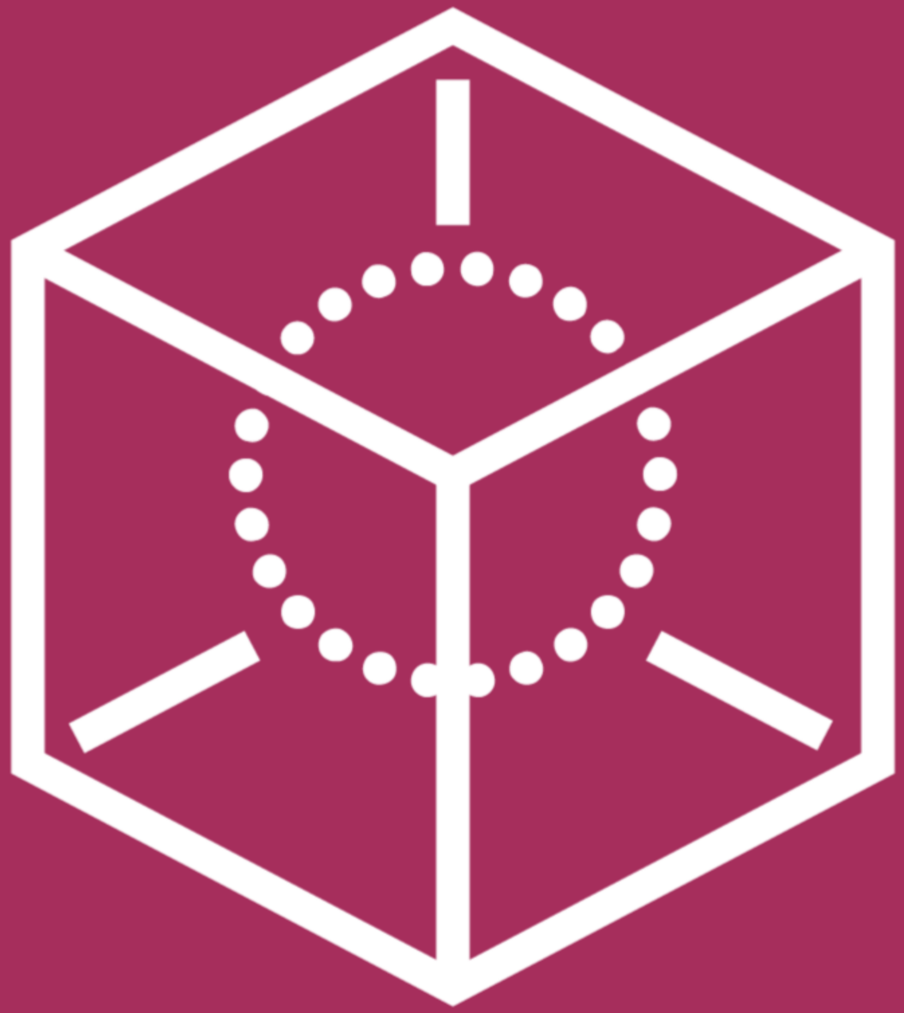
```
class City {

    String name;
    int population;
}
```

A City is modeled by a class

That defines a state composed of:

- the name of the city
- the population

Encapsulating Data



Encapsulation

Encapsulating the data stored in your classes consists in hiding it from the users of this class

Exposing the fields of your classes is a bad practice

Encapsulating consists in hiding the fields

And expose their value through accessors

Why is it a bad practice?

Suppose you need the population to be greater than zero

Exposing the population field prevents you from being able to validate it

```
class City {

    private String name;
    private int population;


}
```

A City is modeled by a class

That defines a state composed of:

- the name of the city
- the population

```
class City {

    private String name;
    private int population;

    public City(String name, int population) {
        this.name = name;
        if (population < 0) {
            throw new IllegalArgumentException(
                                "Population cannot be a negative number");
        this.population = population;
    }

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

```
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;
    }
}
```

Demo

Let us write some code!

Let us see this City class in action

Module Wrap Up

What did you learn?

1) Abstraction

- Designing classes to fit your needs
- Keeping them simple

2) Encapsulation

- Hiding the internal state of your class
- To control what the outside code can do with it

Up Next: Designing a Class, Adding Fields, Methods and Constructors