CODEGYM

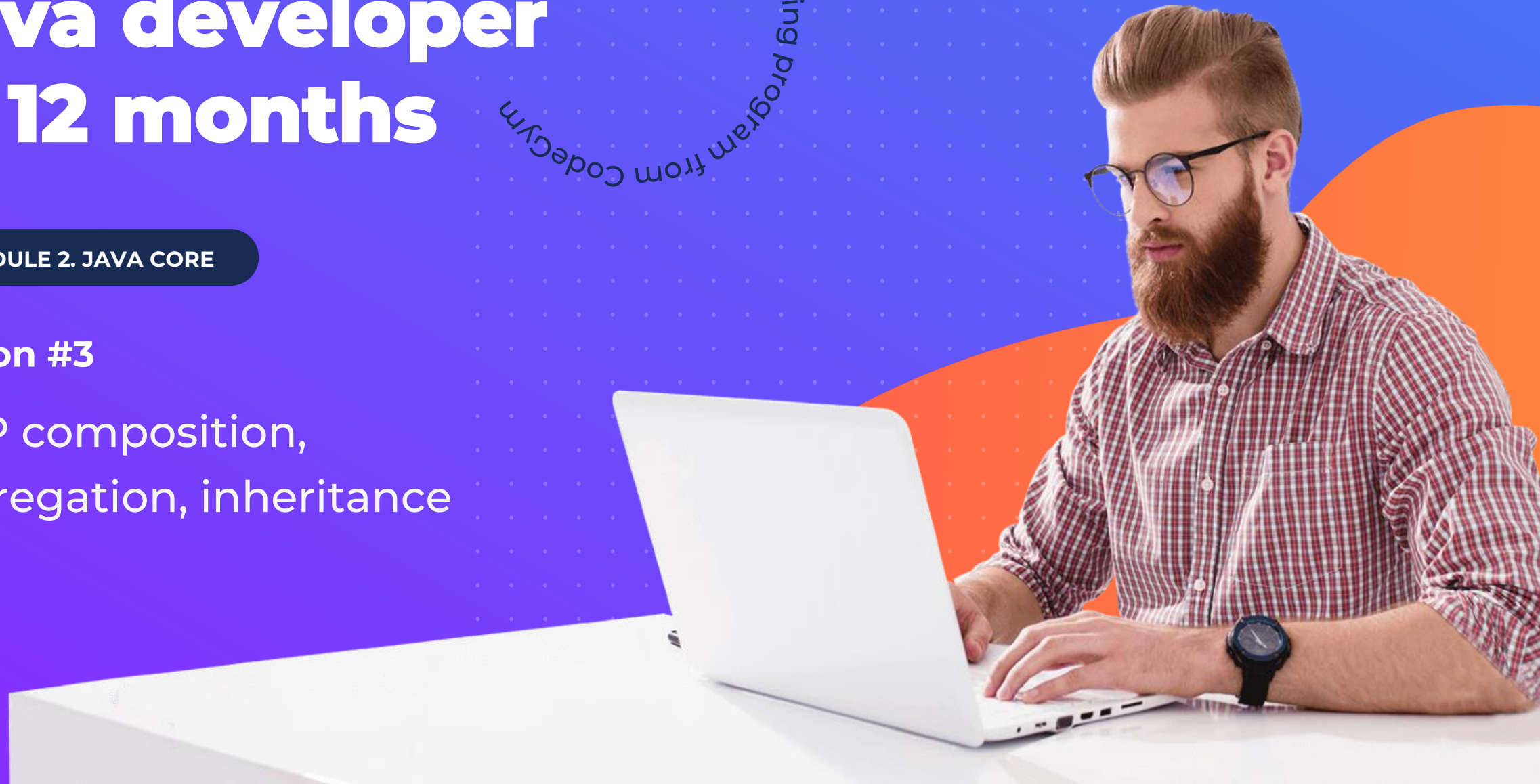
Java developer in 12 months

Mentor-supported training program from Codegym

MODULE 2. JAVA CORE

Lesson #3

OOP composition, aggregation, inheritance



CODEGYM

Lesson plan

- Inheritance
- Association: composition and aggregation

MODULE 2. JAVA CORE

CODEGYM

Inheritance

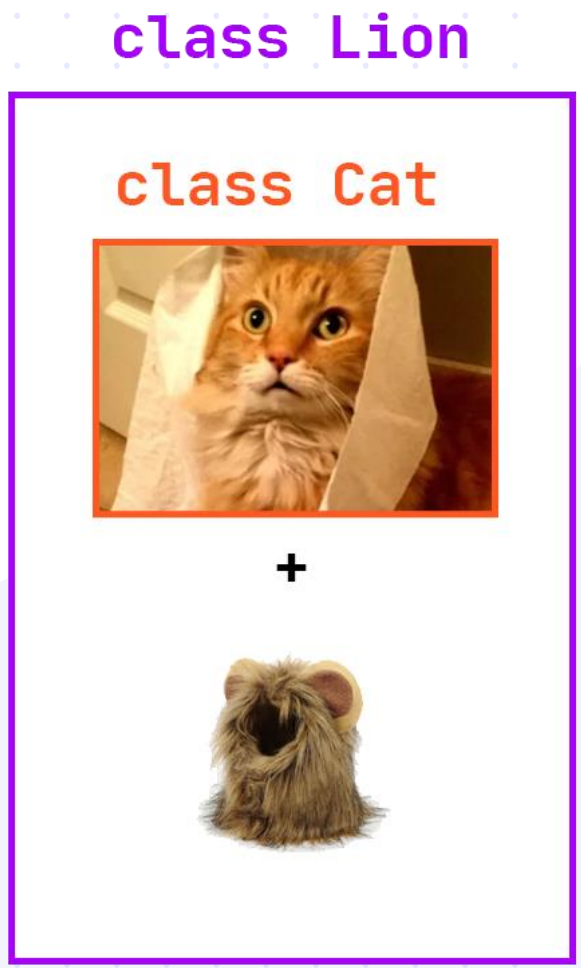
Suppose there is a class that implements 80% of the functionality we need with its methods. You can just copy its code into your class. But this solution has several disadvantages:

1. Such a class can already be compiled into bytecode, but you do not have access to its source code.
2. The source code of the class is there, but you work for a company that can be sued for a couple of billions for using even 6 lines of someone else's code. And then your employer will sue you.
3. Unnecessary duplication of a large amount of code. In addition, if the author of another class finds an error in it and fixes it, you will still have this error.

There is a more subtle solution, and without the need to get legal access to the original class code.

In Java, you can simply declare the class you want to be the parent of your class.

This will be equivalent to you adding the code of that class to the code of your own. All data and all methods of the parent class will appear in your class.



Inheritance can be used for other purposes as well. Let's say there are several classes that are very similar, have the same data and methods.

You can create a special base class, move this data (and the methods that work with it) into this base class, and make those several classes inherit the base class. In other words, indicate in each class that it has a parent class - a given base class.

You can only inherit from one class.
There is no multiple inheritance of classes in Java: a class can have only one parent class. But there is multiple inheritance of interfaces.

CODEGYM

Association: composition and aggregation

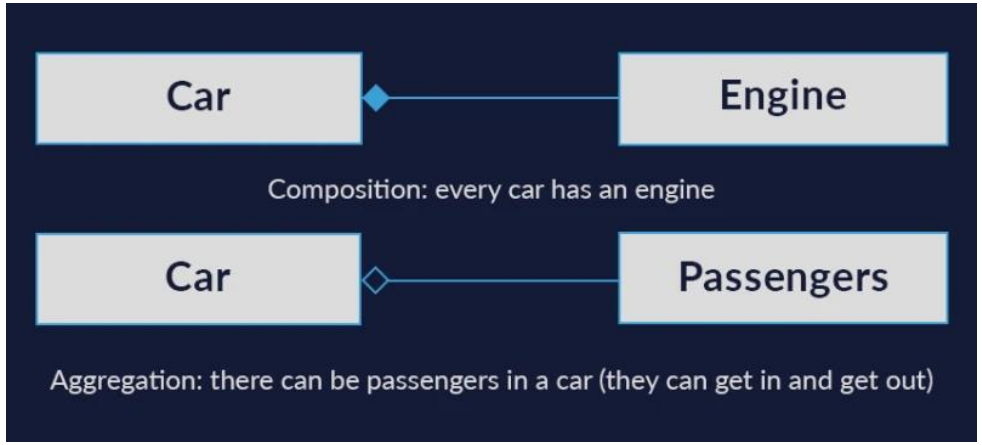
Classes and objects can be related to each other. Inheritance describes this relationship as “IS A“. Cat is a Pet, for example.

Another type of a relationship: not "is a", but "is a part of" ("HAS A"). The hand is not a person, but is part of a person.
“HAS A” relationships can be described in code using composition and aggregation mechanisms. The difference between them lies in the "strictness" of these connections.

There is a **Car car**. Each car has an **Engine engine,** and in addition each car has **Passenger [] passengers** inside.

One car can fit multiple passengers. If all passengers get out of the car, it will continue to function. The relationship between the Car class and the Passenger [] passengers array is less strict. **It's called aggregation**.

Composition is a stricter type of relationship.
With composition, an object is not only part of an object, but it cannot belong to another object of the same type. For example, a car engine. The engine is part of a car, but cannot be part of another car.



CODEGYM

Homework

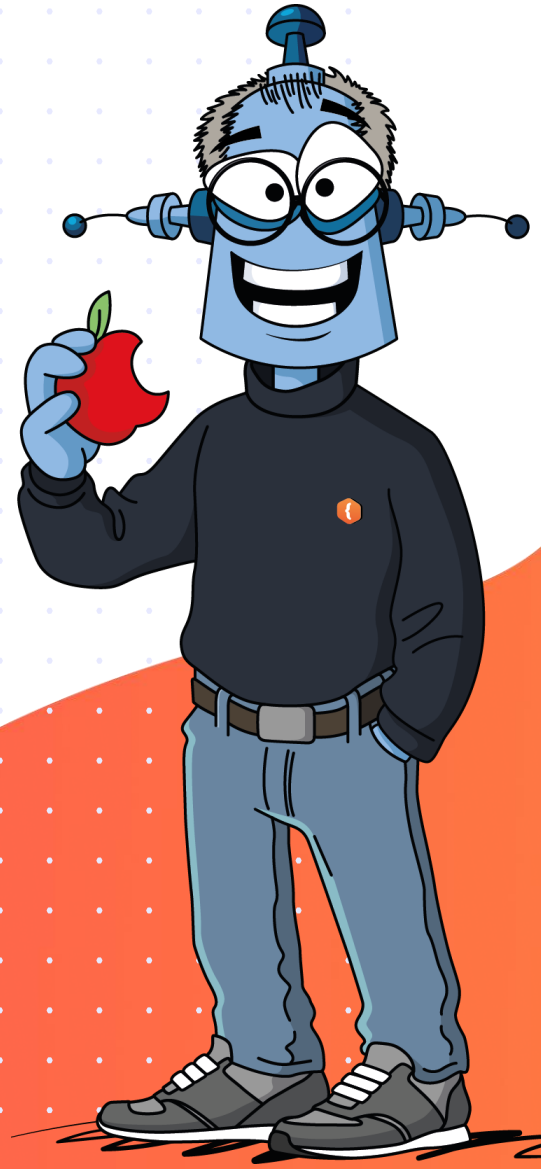
Module 2

Level 3. OOP composition, aggregation, inheritance

MODULE 2. JAVA CORE

CODEGYM

Questions and Answers



MODULE 2. JAVA CORE