CODEGYM

Mentor-supported training program from Codegym

Java developer in 12 months

MODULE 2. JAVA CORE

Lesson #2

OOP: overloading, overriding, abstract classes

CODEGYM

Lesson plan

- Abstract classes
- Why do classes get called “abstract”
- Why there is no multiple inheritance in Java
- Implementing abstract methods
- Overloading methods
- Overriding methods

CODEGYM

Abstract classes

1. An abstract class can contain a method declaration without its implementation. Such a method is called **abstract**.

2. An abstract method is marked with the special **abstract** keyword.
If a class has at least one abstract method, the class is also marked with the **abstract** keyword.

3. You can not create an object of an abstract class. Such code will simply not compile.

4. If you inherited your class from an abstract class, then you need to override all inherited abstract methods - write an implementation for them. Otherwise, such a class will also have to be declared abstract. If a class has at least one unimplemented method declared directly in it or inherited from a parent class, then the class is considered abstract.

CODEGYM

Why do classes get called “abstract”

Abstraction — one of the principles of OOP, according to which, when designing classes and creating objects, it is necessary to highlight only the main properties of an entity and discard secondary ones.

Abstract class — the most abstract, v-e-e-e-ry approximate “template” for a group of future classes. This group can not be used in finished form — it’s too “raw”. But it describes a certain general state and behavior that future classes, the heirs of the abstract class, will have.

- A class is abstract if at least one of its methods is abstract.
- An instance of an abstract class can not be created.
- If a method is marked as **abstract**, each derived class must either implement it or be declared abstract. Otherwise the compiler will throw an exception.

CODEGYM

Why there is no multiple inheritance in Java

Let's say we have two classes and they both have the same method. If Java had multiple inheritance, child classes would not be able to decide which behavior to choose.

CODEGYM

Implementing abstract classes

1. Implicit implementation of an abstract method
2. Widening of visibility.
3. Narrowing the result's type.

CODEGYM

Method overloading

Method's name and parameter types must be unique. This union is also called **method signatures**.

Parameter names do not play a role - they are lost during compilation. After compilation, only the method name and parameter types are known about the method.

In the process of determining which method to call, parameter **types can only widen, not narrow.**

CODEGYM

Method overloading

In OOP, method overloading is one of the features of a programming language that allows a subclass or child class to provide a specific implementation of a method already implemented in one of the superclasses or parent classes.

Method overriding has a **number of limitations**:

1. The overridden method must have the same arguments as the parent method.
2. The overridden method must have the same return type as the parent method.
3. The access modifier of the overridden method cannot differ from the "original" one either.

Method overriding in Java is one of the tools to implement the idea of **polymorphism**. Therefore, the main advantage of using it will be the same flexibility that we talked about earlier.

We can build a simple and logical class system, each of which will have specific behavior (dogs bark, cats meow), but with a single interface - a **voice()** method for everyone instead of a bunch of **voiceDog()**, **voiceCat()** etc. methods.

CODEGYM

Homework

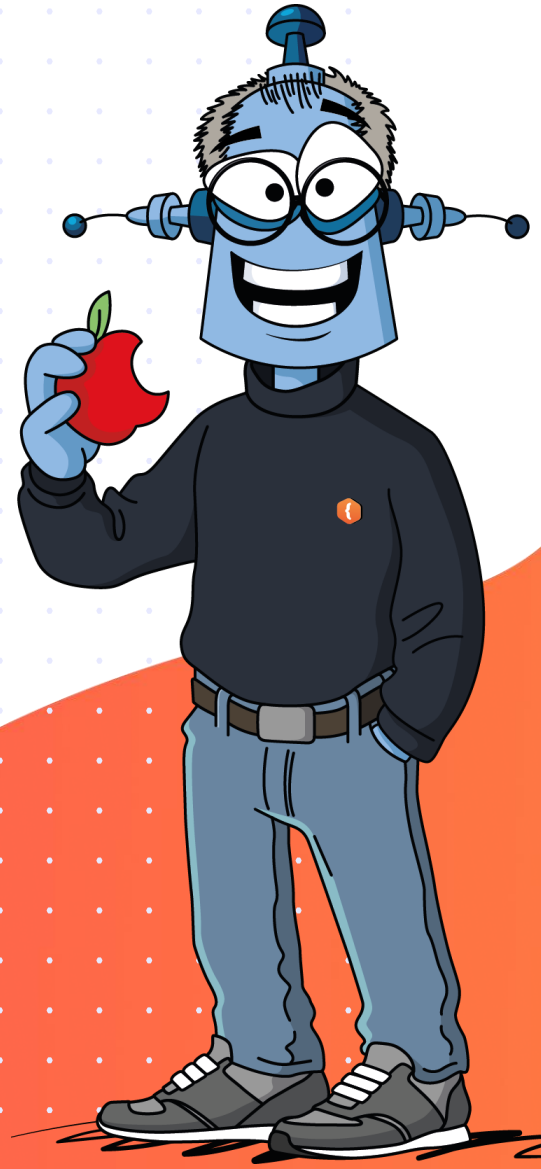
Module 2

Level 2. OOP overloading, overriding, abstract classes

MODULE 2. JAVA CORE

CODEGYM

Questions and Answers



MODULE 2. JAVA CORE