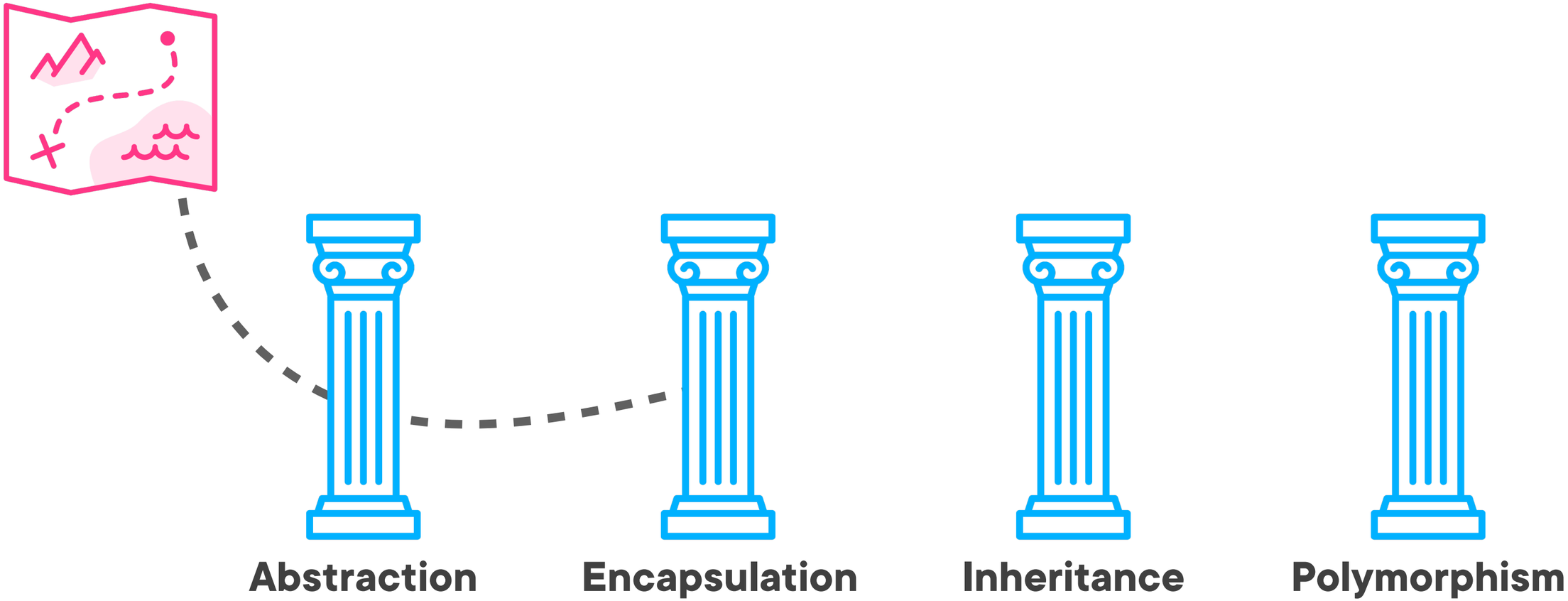
Inheriting from Another Class

Paolo Perrotta

Developer, Author

@nusco | www.paoloperrotta.com

The Four Pillars of OOP



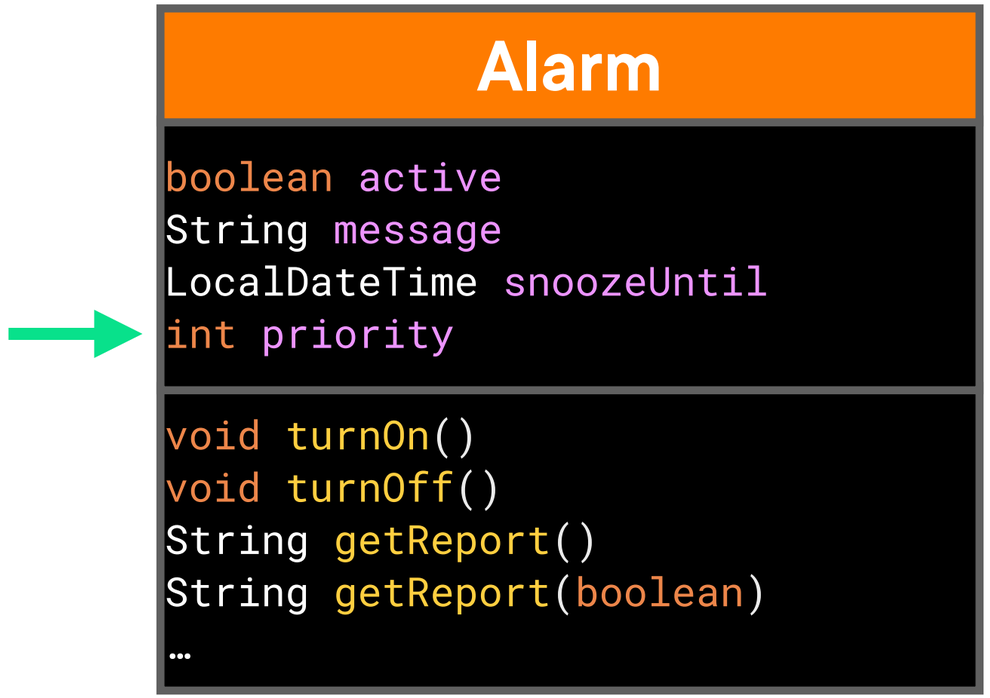
Inheritance and polymorphism work together.

A Request for New Features

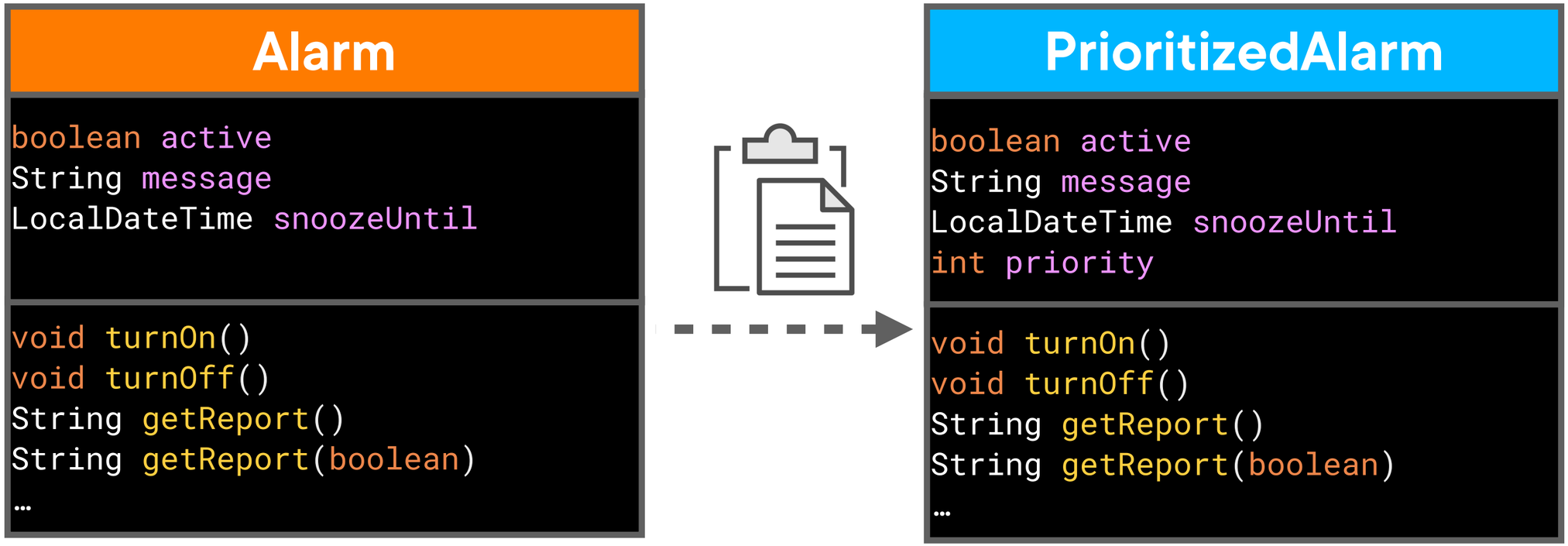
A special case of an alarm

- All the features of a regular alarm...
- ...plus a priority number

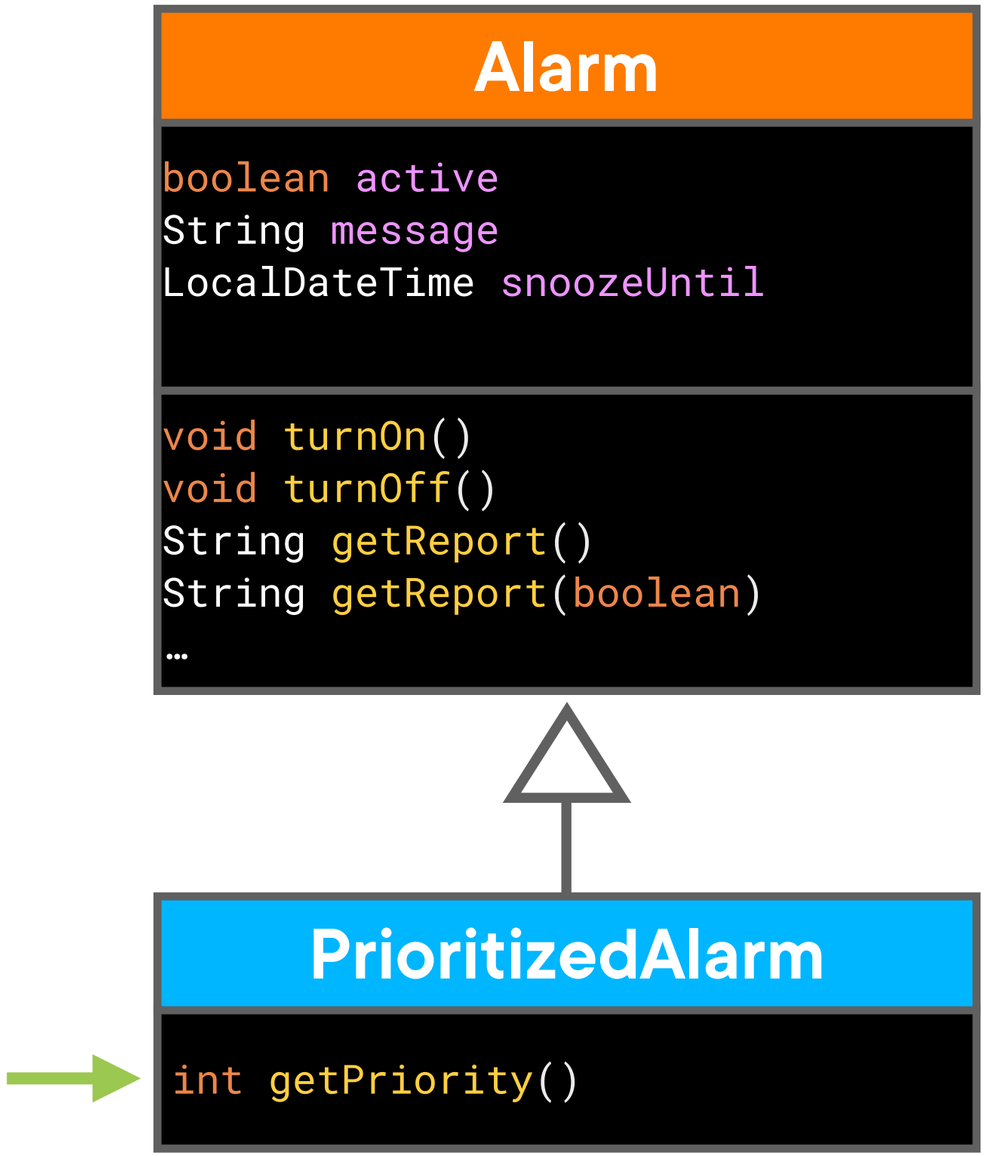
Adding a Priority



Adding a Priority



Adding a Priority



Another Request from the Customer

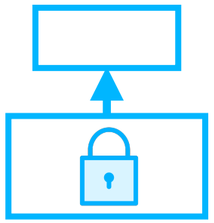
A “high visibility alarm”

- Its report ends with an exclamation mark
- If it’s not active, its report is still an empty string

Access Modifiers

private: visible in the class

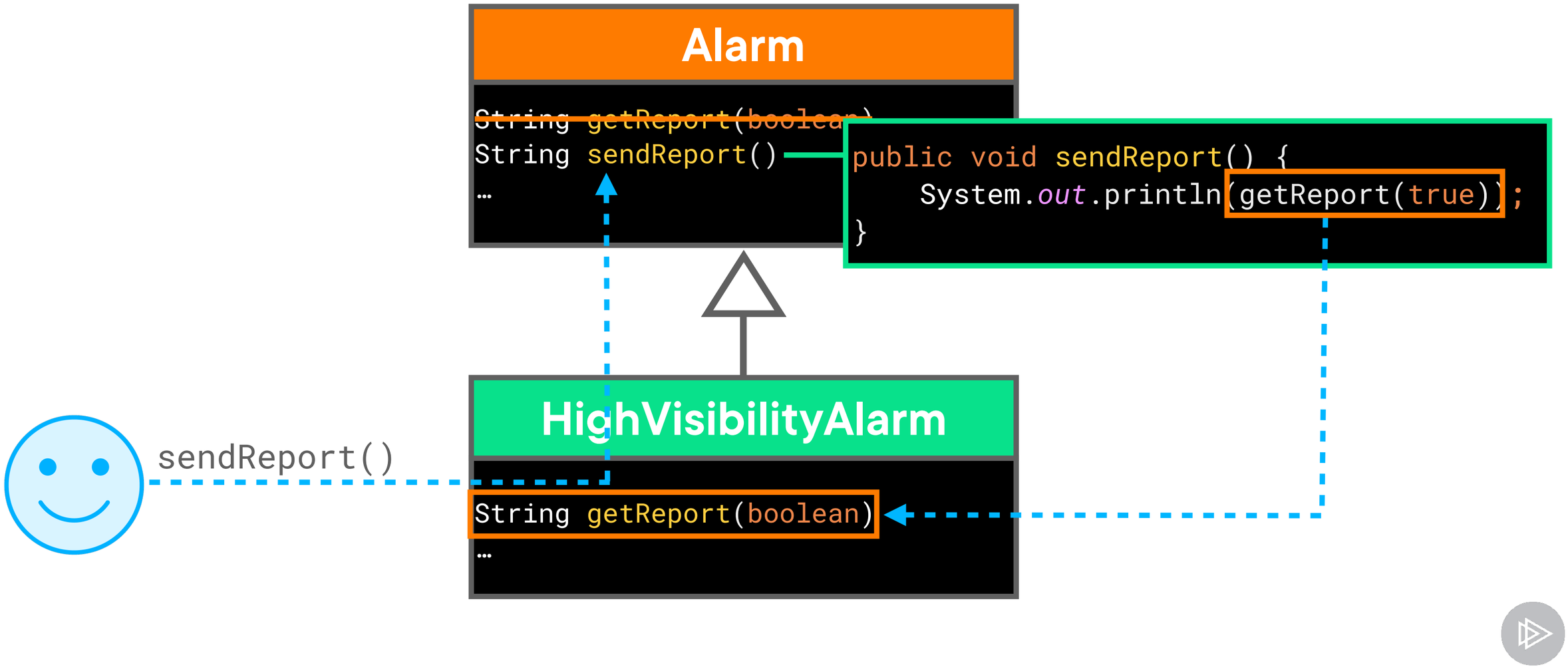
“package private”: visible in the package



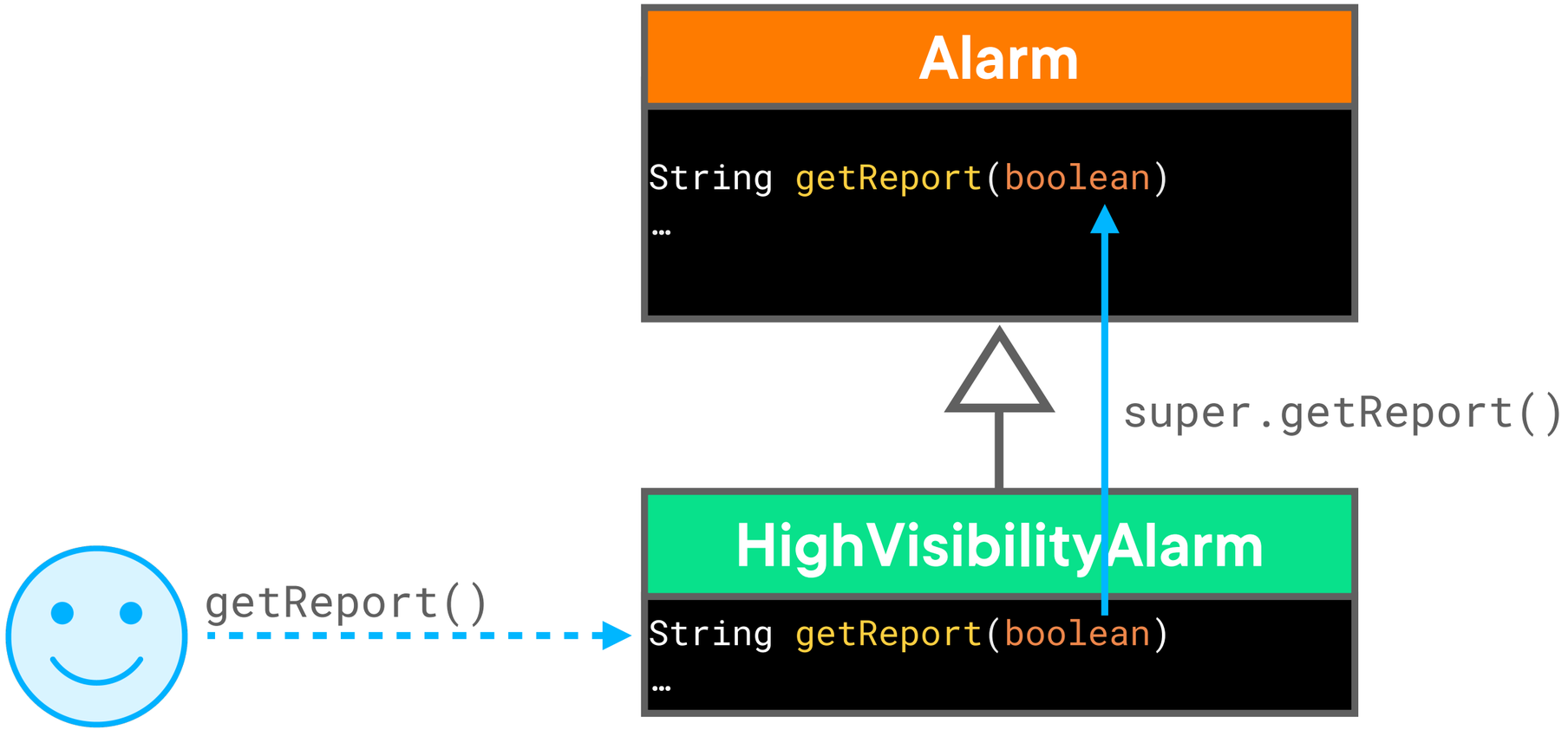
protected: visible to subclasses *and* to code in the package

public: visible anywhere

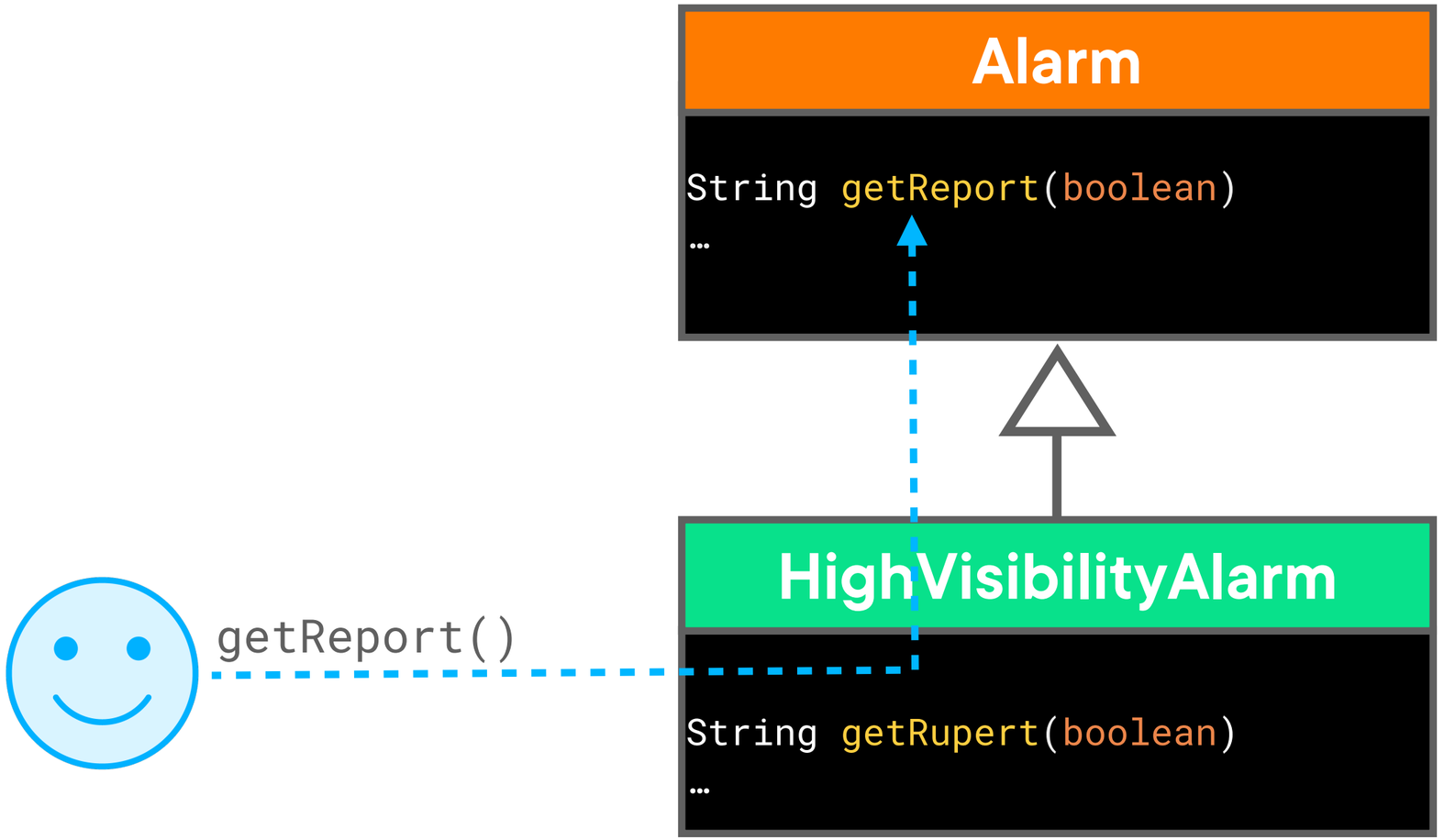
The Superclass Calling Into the Subclass



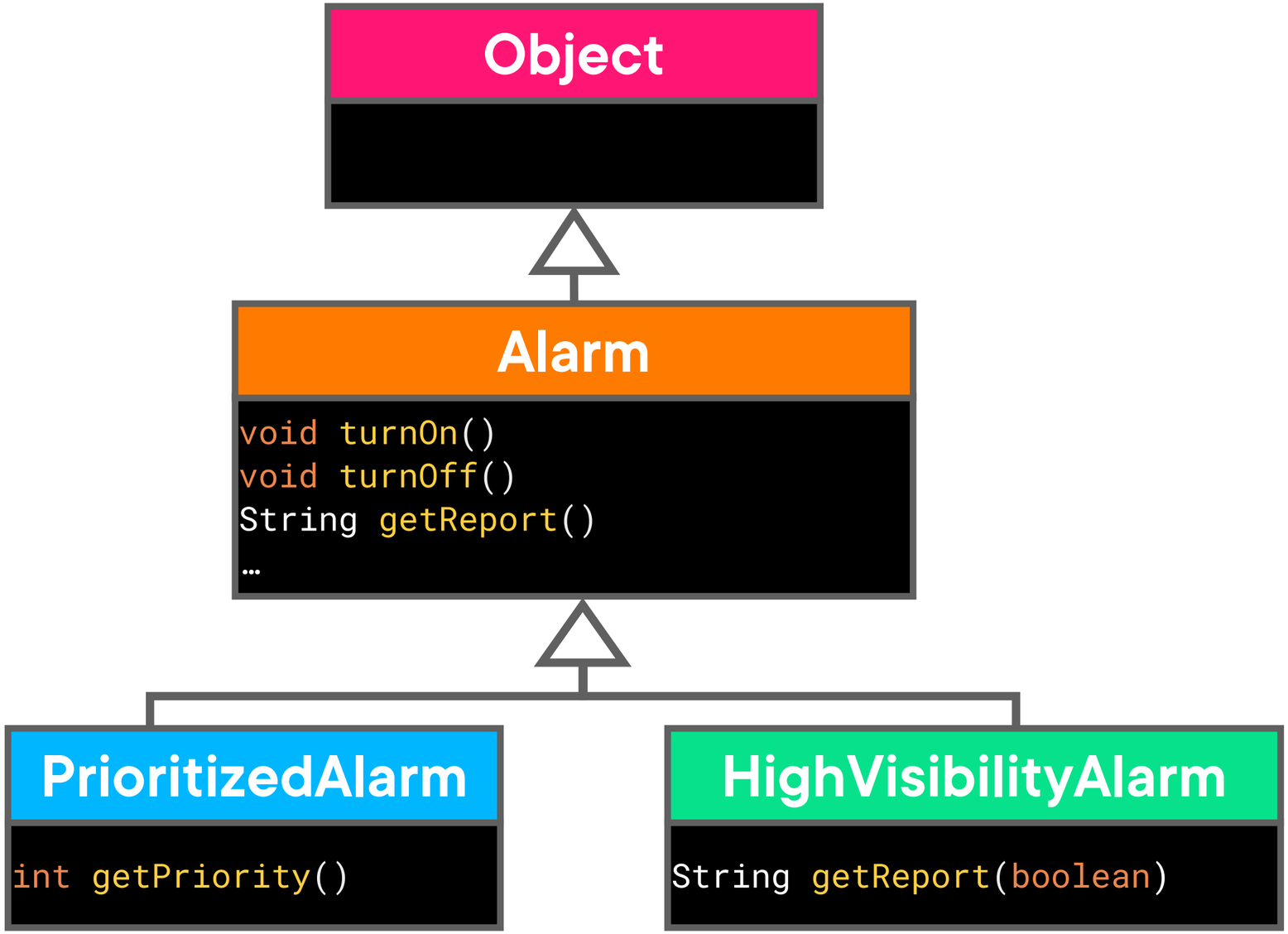
The Subclass Calling Into the Superclass



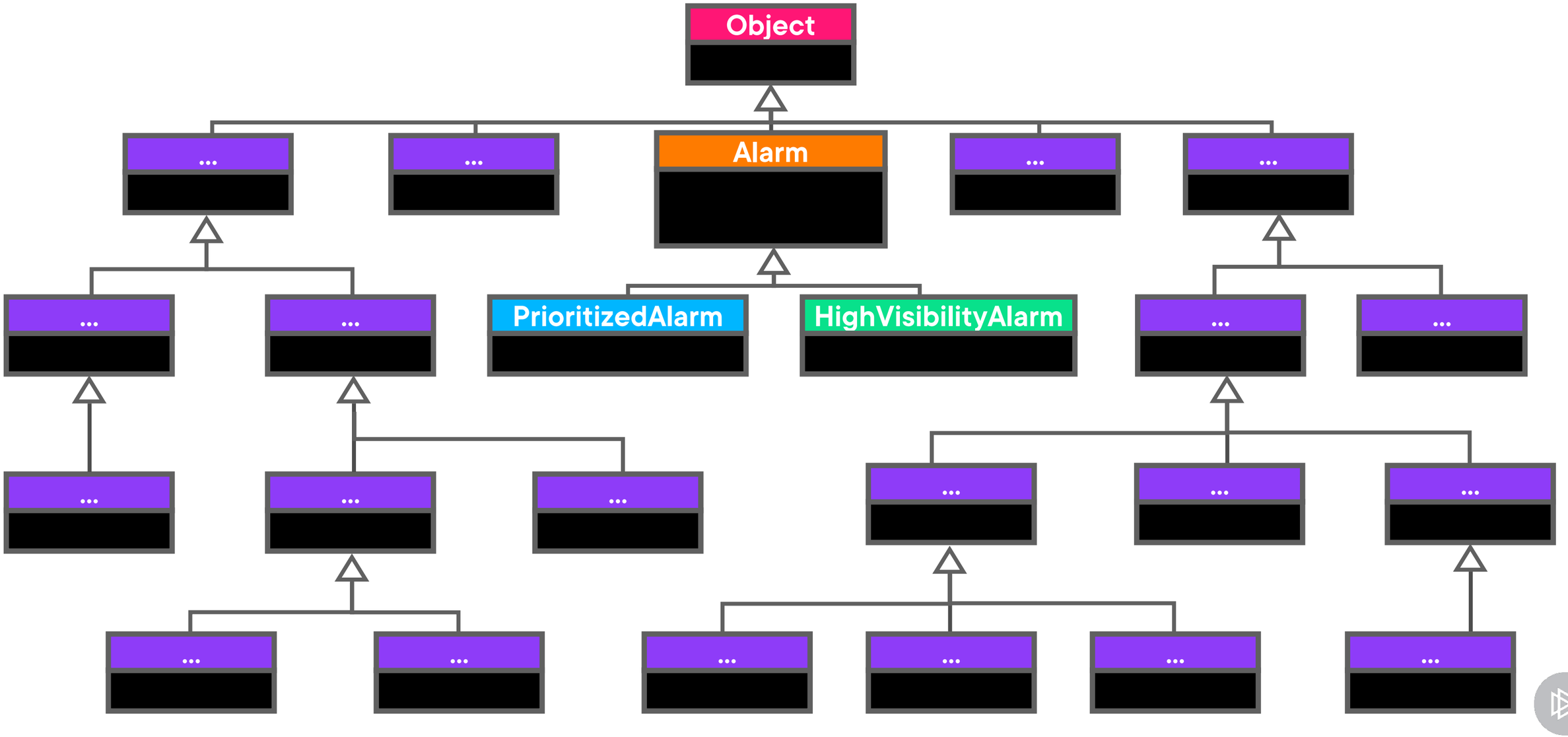
Calling the Wrong Method



A Hierarchy of Classes



The Singly Rooted Hierarchy



The *final* Keyword

On a variable/field

It means “cannot reassign”

On a method

It means “cannot override”

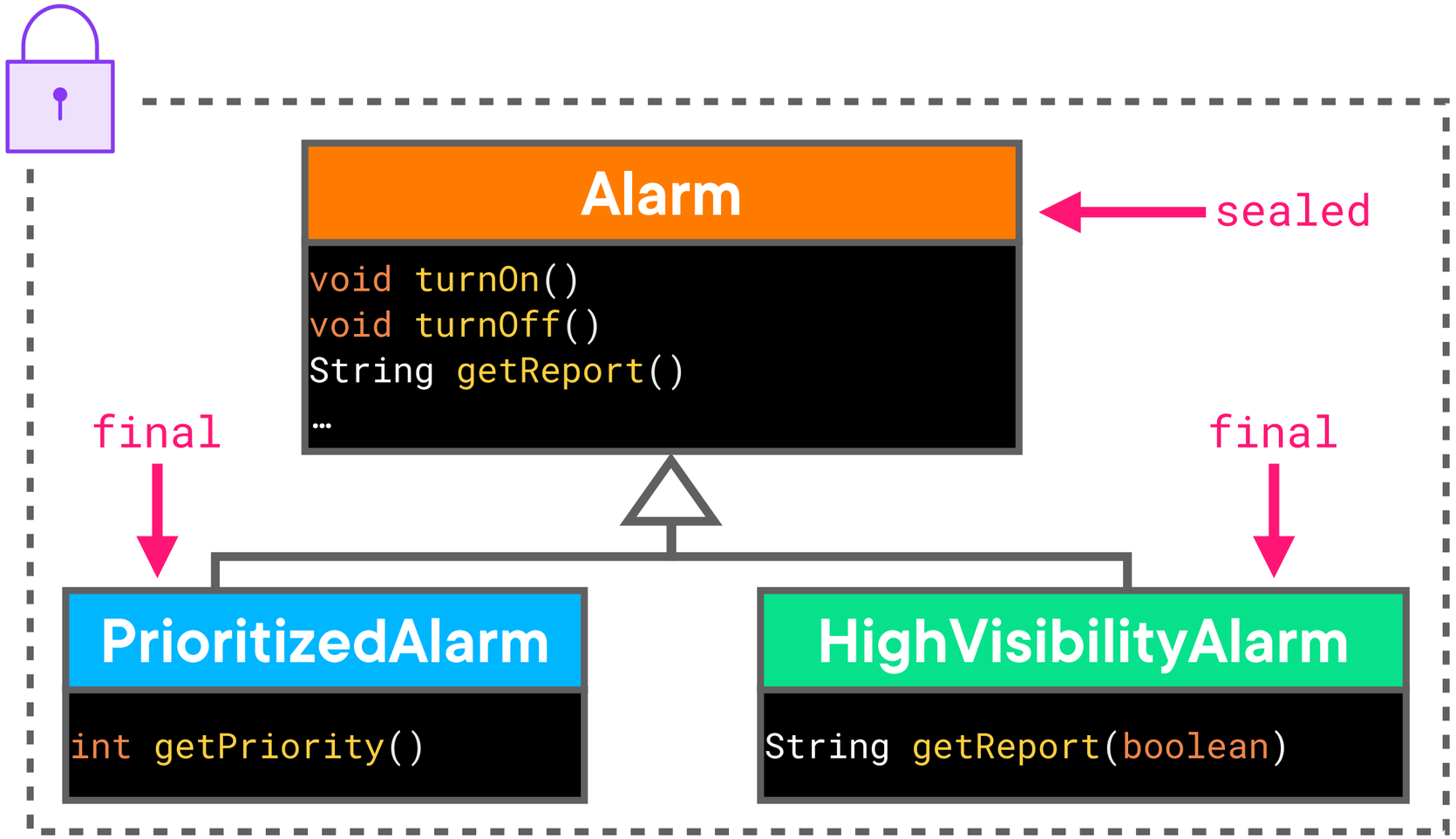
On a class

It means “cannot inherit”

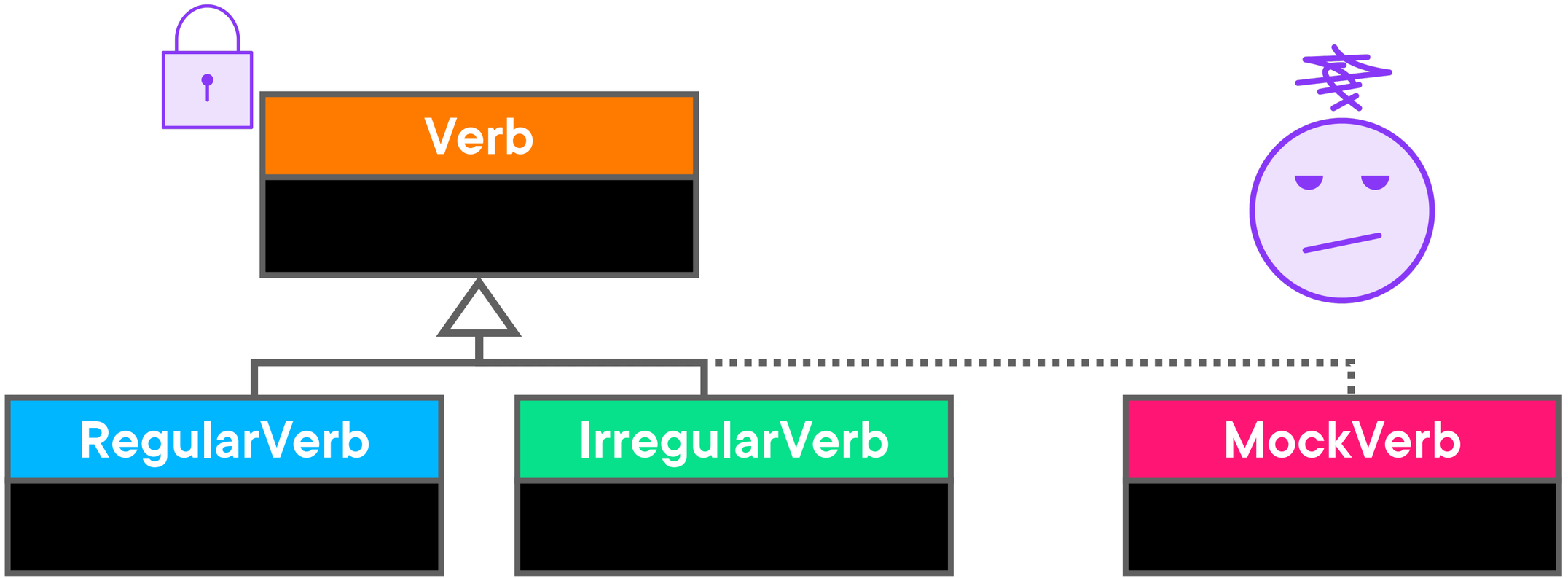
Methods called by a constructor should be either *private* or *final*.

Don’t overuse *final*.

The Alarm Hierarchy



An Example of Sealed Classes



Don’t overuse sealed classes either.

Summary

Defining a class that extends another class

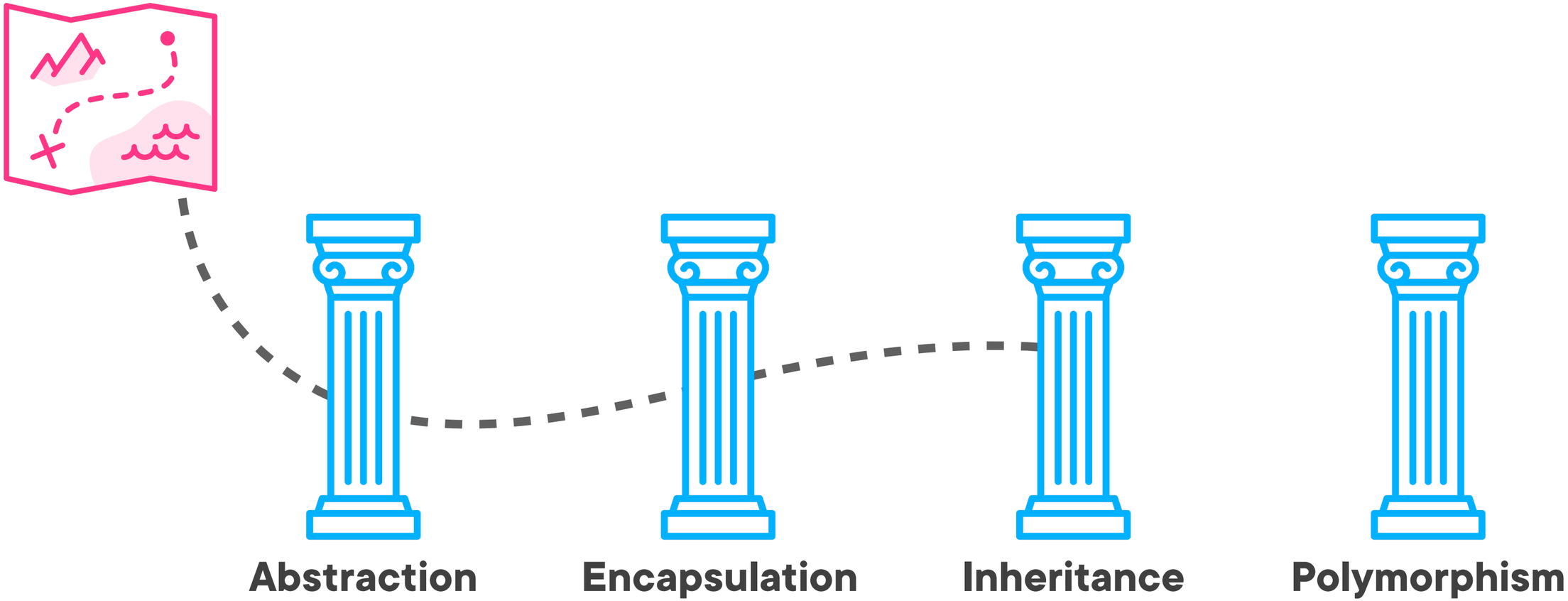
Overriding methods in the superclass

Calling into the superclass

Object and the singly rooted hierarchy

Controlling inheritance with *final* and *sealed*

The Four Pillars of OOP



Up Next:

Understanding Polymorphism