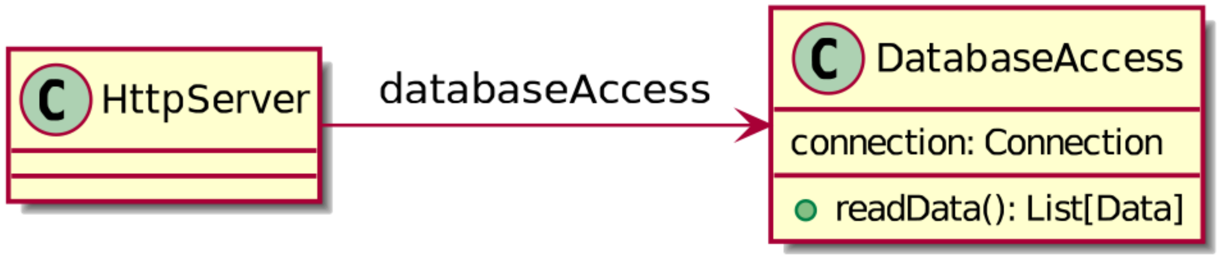
EPFL

Mocking

Effective Programming in Scala

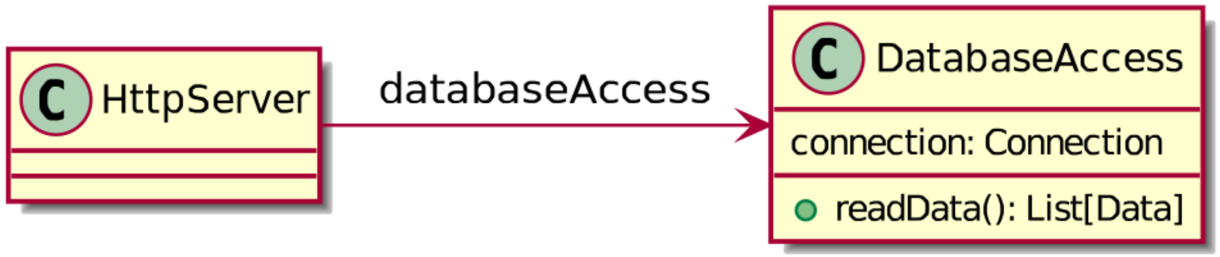
Mocking

In a system made of components that depend on other components, how do we test one component without having to also set up all the components it depends on?



Mocking

In a system made of components that depend on other components, how do we test one component without having to also set up all the components it depends on?

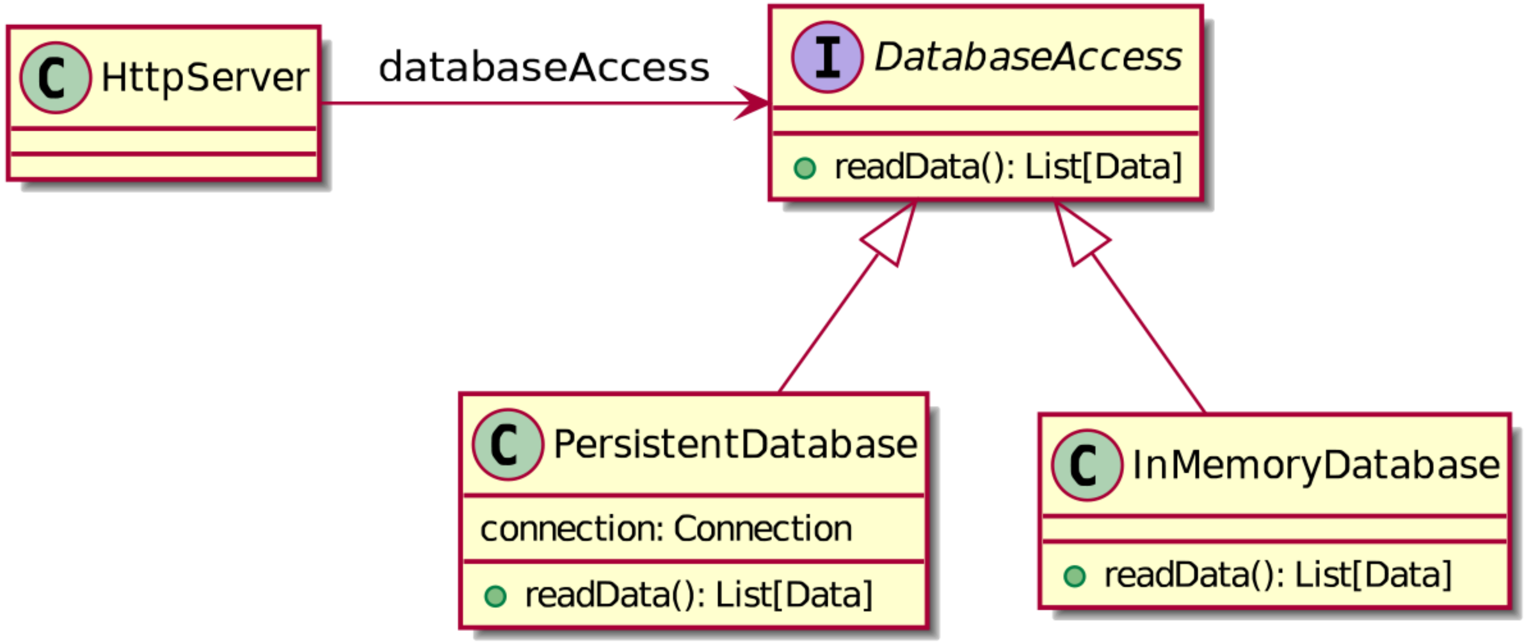


- Mock the components that are depended on.

What Is Mocking?

Mocking consists in providing a fake implementation of a component that can be substituted to the component for the purpose of a test.

We have already seen in a previous lesson how to design components so that they are easier to mock:



Mocking in Scala (1)

Main sources:

```
class HttpServer(databaseAccess: DatabaseAccess):
  ...

trait DatabaseAccess:
  def readData(): List[Data]

class PersistentDatabase(connection: Connection) extends DatabaseAccess:
  ...
```

Mocking in Scala (2)

Test sources:

```
class HttpServerSuite extends munit.FunSuite:
  val databaseAccess = InMemoryDatabase()
  val httpServer     = HttpServer(databaseAccess)
  test(”something”) {
    ...
  }

class InMemoryDatabase extends DatabaseAccess:
  ...
```

Advanced Mocking

In some cases, this solution is not applicable: the operations to implement are too numerous, or too complicated.

In this situation, you can use a mocking library such as ScalaMock. Such libraries make use of advanced techniques available on the JVM to create fake component implementations. Mastering these techniques is out of the scope of this course.

Summary

Mocking components makes it easier to set up tests. The tests may run faster, and the test outcome can not be influenced by a bug in a dependency of the tested component.

However, this technique does not test the whole system as it will be deployed to production.