Dependencies in software

- A dependency of A on B mean that a piece of software A is adapted to and requires a piece of software B
  - For example: a structure simulation A depends on a linear algebra library B
- When there is such a dependency:
  - A requires B to compile and/or run
  - Many other independent simulations can reuse the same linear algebra library
  - A is specifically tailored to B and replacing B by another linear algebra library is a complex

    => **Tight coupling**

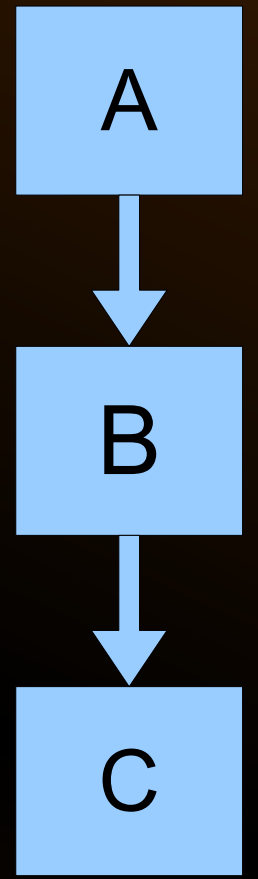
Coupling in object-oriented code

- In object-oriented code, the implementation of a class A is coupled to a class B if
  - B is a concrete class
  - A manipulates directly the concrete type B
- In that case, one can not replace the class B by another without modifying A

  => Tight coupling

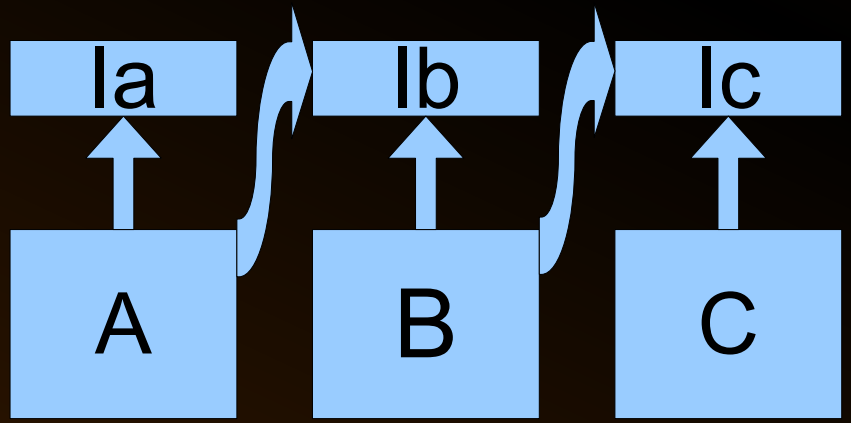
The issue with tight coupling

- Layered code typically exhibits tight coupling
  - Code A uses library B that uses library C, etc.
- The lower the libraries in the stack, the more complex they become to
  - Replace
  - Modify
  - Adapt



The Dependency Inversion Principle

- The Dependency Inversion Principle (DIP) states:
  - High-level modules should not depend on low-level modules. Both should depend on abstractions.
  - Abstractions should not depend on details. Details should depend on abstractions.

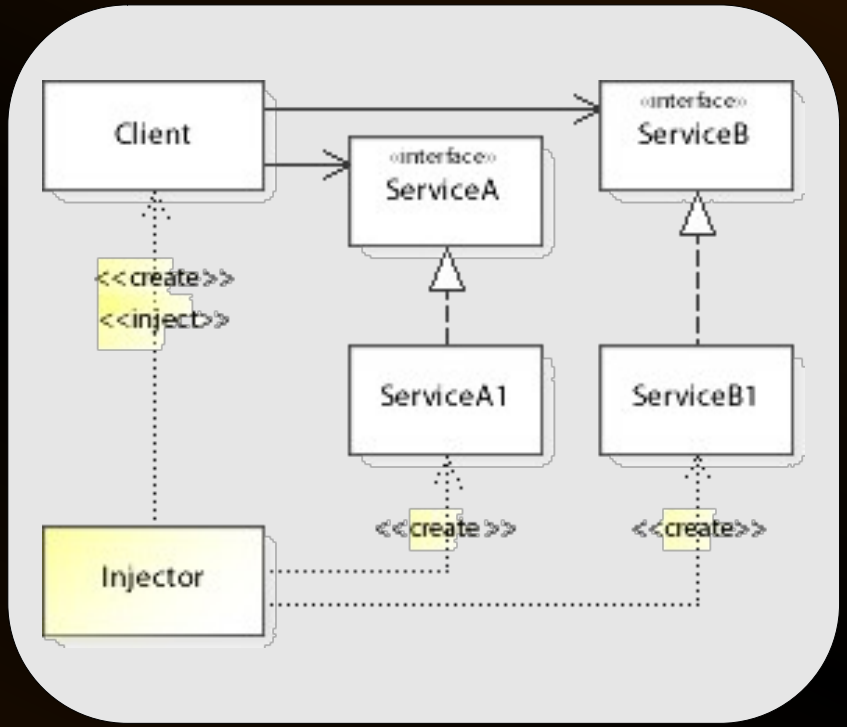


Dependency Inversion Principle : HOW ?

- The Dependency Inversion Principle is…
  - A design **principle**: a goal
  - Not an implementation
  - Not a design pattern
  - Not a solution or a way to achieve this goal
- So… How to achieve this goal

The dependency injection pattern

- **Warning**: *Inversion* vs. *Injection* !!!
- Provides a way to implement the DIP
- The Client
  - Uses interfaces only
  - Does not know the classes implementing the interfaces
- The Injector
  - Instantiates the concrete classes
  - “Injects” them into the client

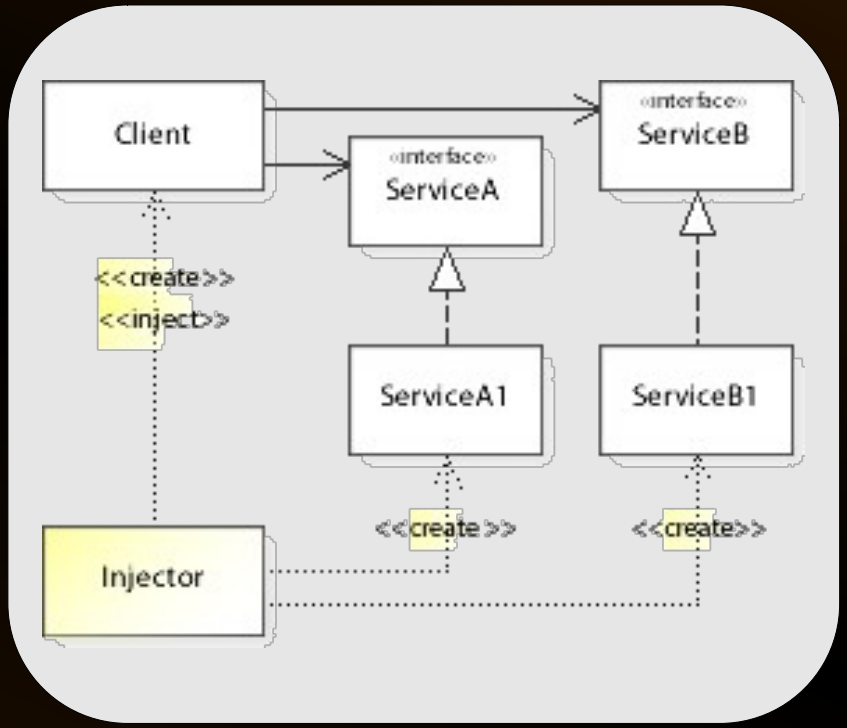


The Inversion of Control Principle

- Another principle closely related
- There is Inversion of Control (IoC) when
  - The purpose-written code does not call reused code
  - Instead, the reused code calls the purpose-written code

The Inversion of Control Principle

- Another principle closely related
- There is Inversion of Control (IoC) when
  - The purpose-written code does not call reused code
  - Instead, the reused code calls the purpose-written code
- The dependency injection pattern implement IoC



To summarize

- Tight code coupling limits code evolutivity
- The DIP states that code should not be layered
  - Dependencies should be on abstractions only
- The dependency injection pattern offers a way to support the DIP
- The dependency injection pattern achieve this by IoC