DEPENDENCY INVERSION PRINCIPLE

DIP

- When building a complex system, programmers are often tempted to define “low-level” classes first and then build “higher-level” classes that use the low-level classes directly.
- But this approach is not flexible! What if we need to replace a low-level class? The logic in the high-level class will need to be replaced — an indication of high coupling.
- To avoid such problems, we introduce an **abstraction layer** between low-level classes and high-level classes.

DEPENDENCY INVERSION PRINCIPLE

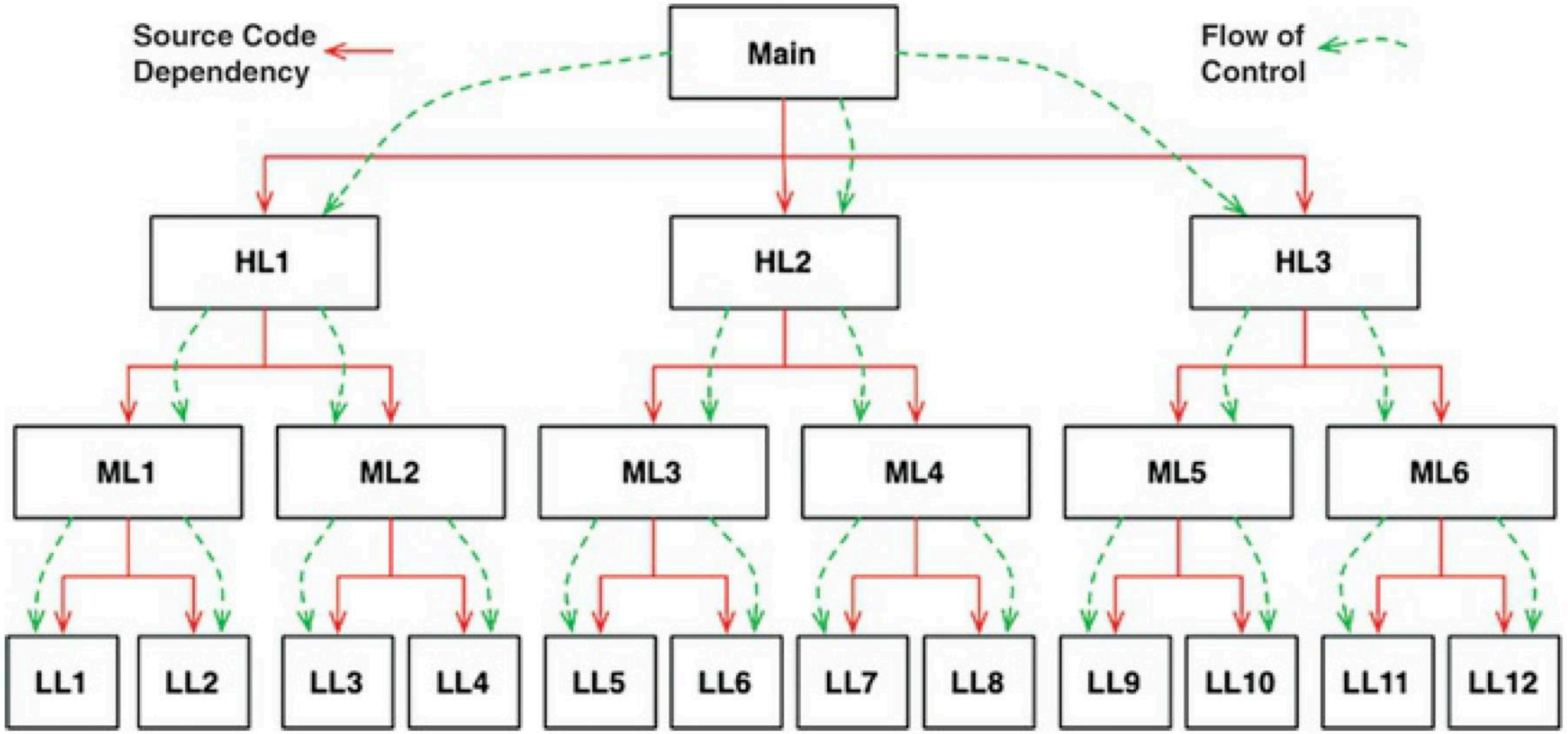


Figure 5.1 Source code dependencies versus flow of control

Clean Architecture, Robert C. Martin

DEPENDENCY INVERSION PRINCIPLE

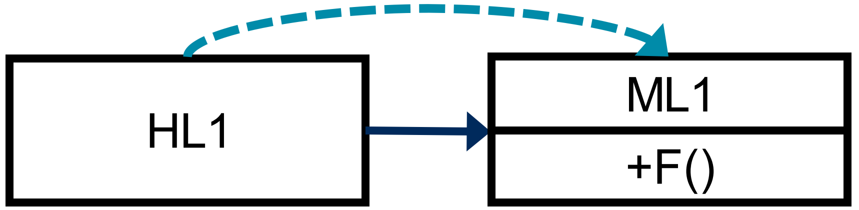
Goal:

- You want to decouple your system so that you can change individual pieces without having to change anything more than the individual piece.
- Two aspects to the dependency inversion principle:
  - High-level modules should not depend on low-level modules. Both should depend on abstractions.
  - Abstractions should not depend upon details. Details should depend upon abstractions.

DIP

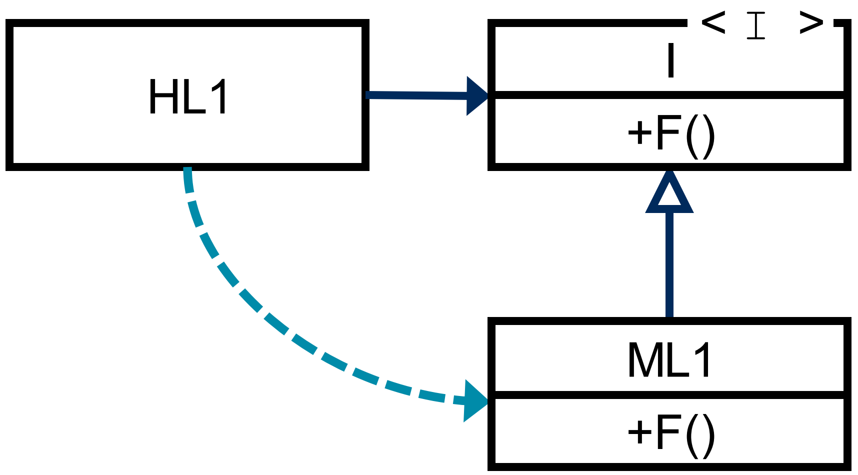
DEPENDENCY INVERSION PRINCIPLE

How do we invert the source code dependency?



We introduce an interface!

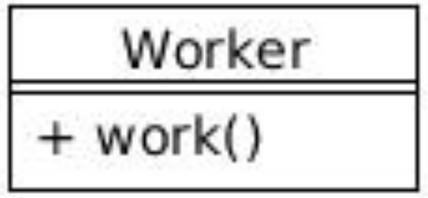
- The flow of control remains the same.
- HL1 depends on the interface and ML1 implements that interface.
- There is no longer a source code dependency between HL1 and ML1!

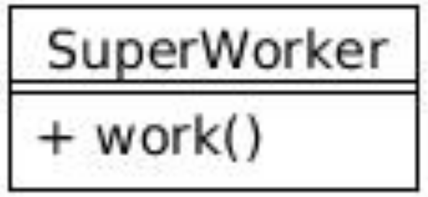


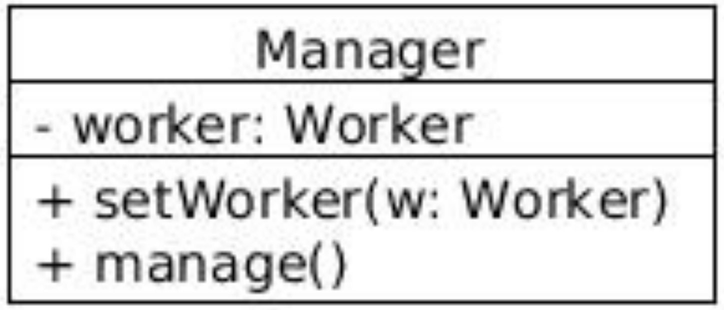
EXAMPLE FROM DEPENDENCY INVERSION PRINCIPLE ON OODESIGN

DIP

- A company is structured with managers and workers. The code representing the company's structure has Managers that manage Workers. Let's say that the company is restructuring and introducing new kinds of workers. They want the code updated to reflect this change.
- Your code currently has a Manager class and a Worker class. The Manager class has one or more methods that take Worker instances as parameters.
- Now there's a new kind of worker called SuperWorker whose behaviour and features are separate from regular Workers, but they both have some notion of "doing work".





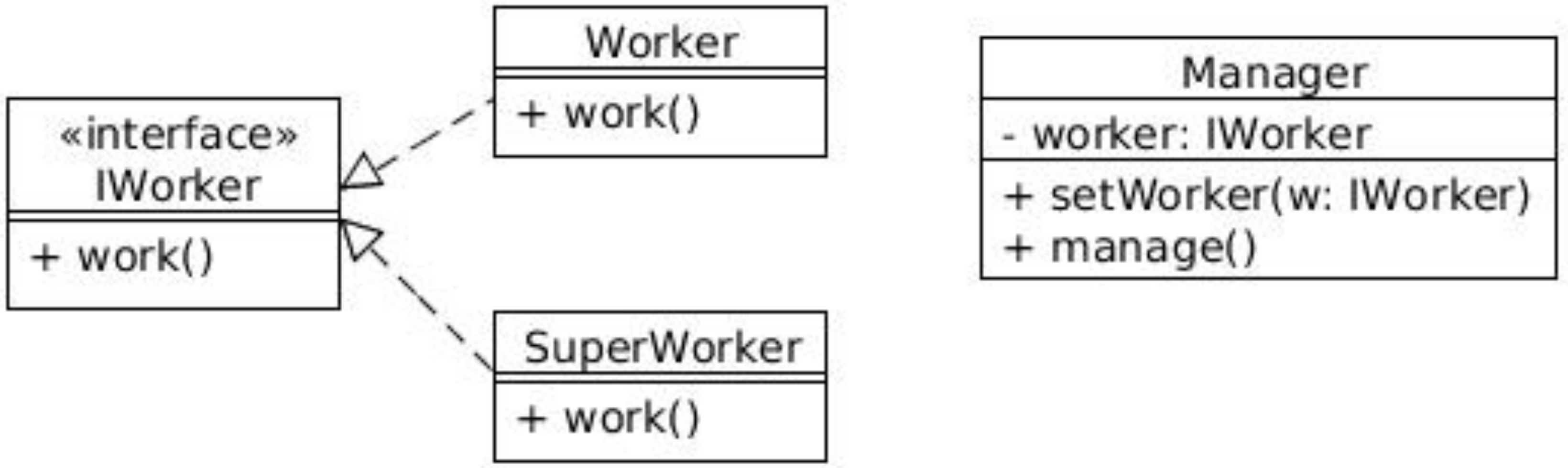


Computer Science
UNIVERSITY OF TORONTO

EXAMPLE FROM DEPENDENCY INVERSION PRINCIPLE ON OODESIGN

DIP

- To make Manager work with SuperWorker, we would need to rewrite the code in Manager (e.g. add another attribute to store a SuperWorker instance, add another setter, and update the body of manage())
- Solution: create an IWorker interface and have Manager depend on it instead of directly depending on the Worker and SuperWorker classes.
- In this design, Manager does not know anything about Worker, nor about SuperWorker. The code will work with any class implementing the IWorker interface and the code in Manager does not need to be rewritten.



Computer Science
UNIVERSITY OF TORO