Class Design Principles

What is Cohesion?

- In terms of object oriented systems, the cohesion is the measure of how strongly related and focused the responsibilities of an object are.

Single Responsibility Principle

- In order to keep our objects focused, understandable, manageable and with low coupling we should assign each one of our object with a single responsibility only.
- Doing so, the cohesion will remain high and our software system will be more understandable and manageable.

The Open Closed Design Principle

- “Software entities (classes, modules, functions etc.) should be open for extension, but closed for modification.” (Martin Fowler, 1999)

  Open for Extension

  It should be possible to to extend the behavior of the software entity. Over time, requirements change and new needs emerge. It should be possible to extend the behavior of the software entity in accordance with our changing needs.

  Closed for Modification

  Being capable of extending the behavior of a software entity shouldn't be involved with changes of the source code or the binary code of the module itself.

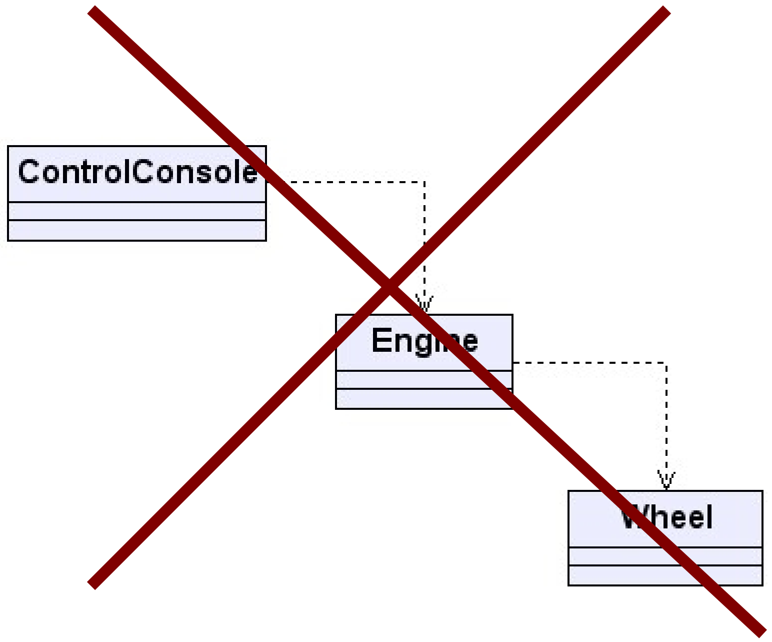
The Liskov Substitution Principle

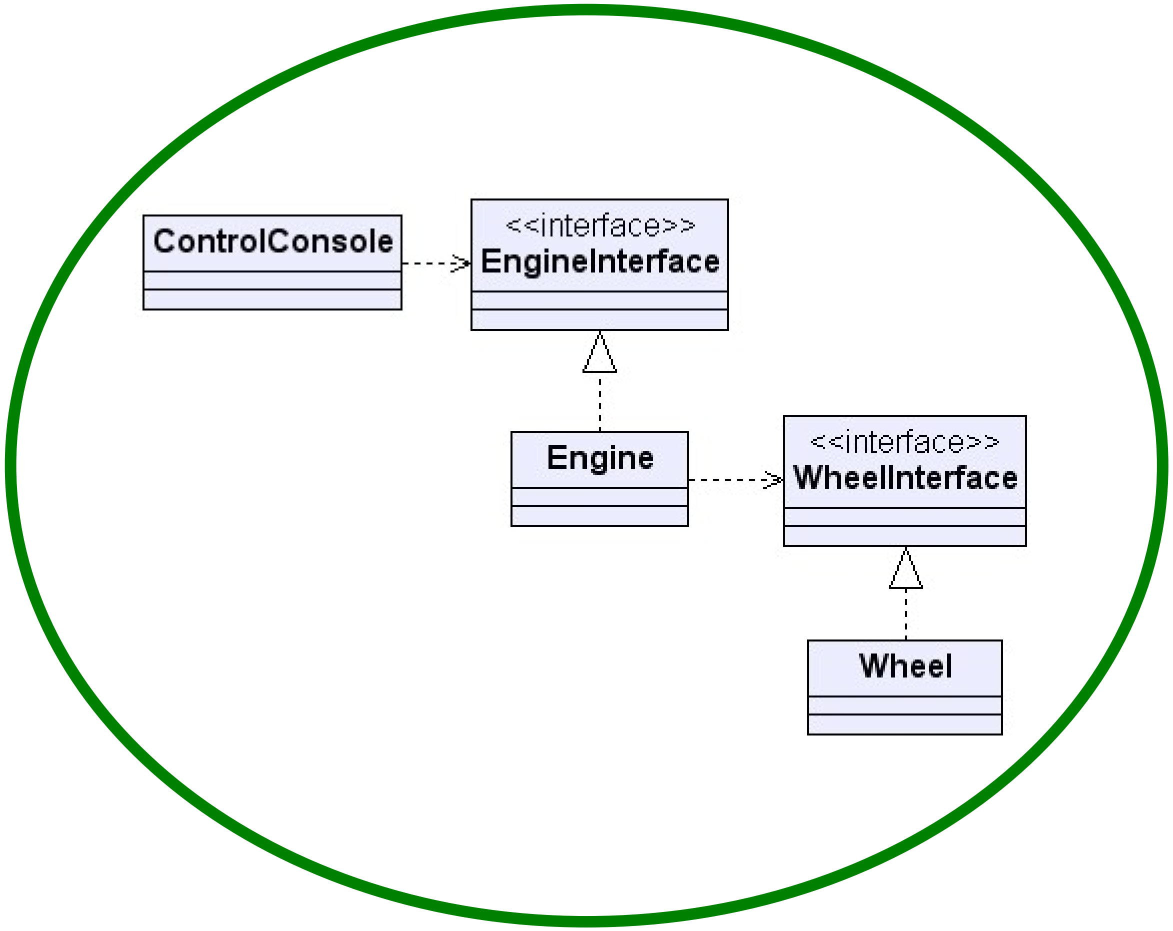
- “What is wanted is something like the following substitution property: If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2, then S is a subtype of T. (Barbara Liskov, 1988)

The Dependency Inversion Principle

- “High-level modules should not depend on low-level modules. Both should depend on abstractions.” (Robert Martin, 2002)
- “Abstractions should not depend on details. Details should depend on abstractions.” (Robert Martin, 2002)

The Dependency Inversion Principle

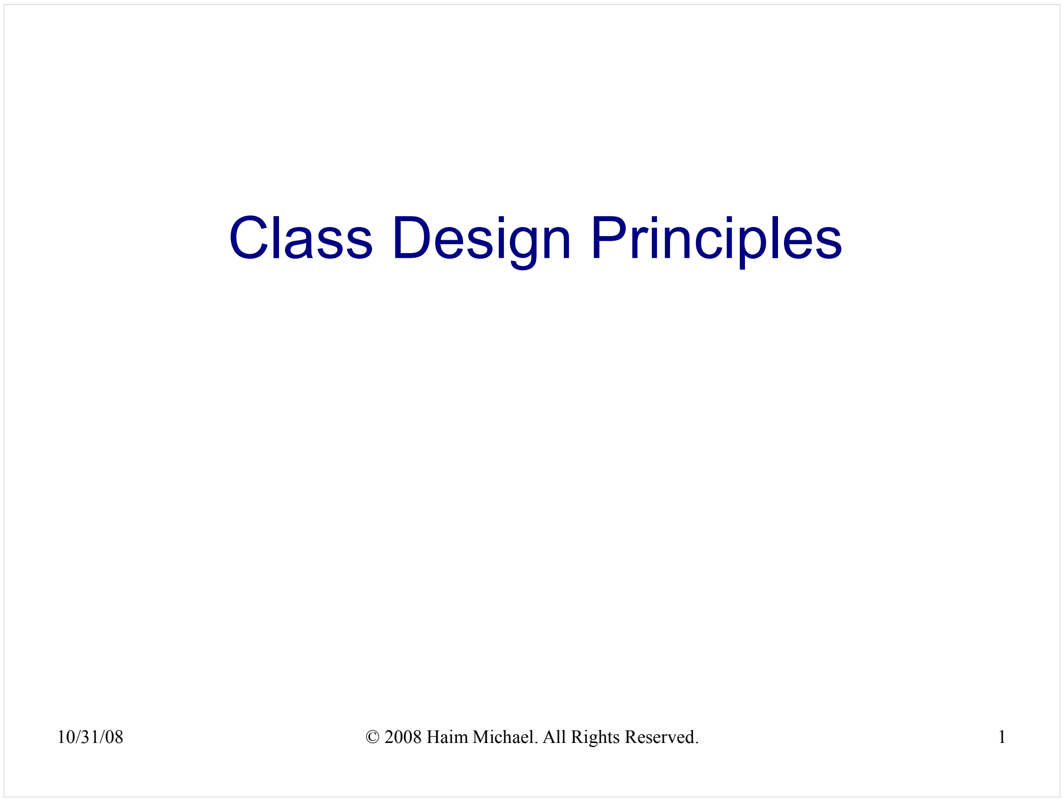




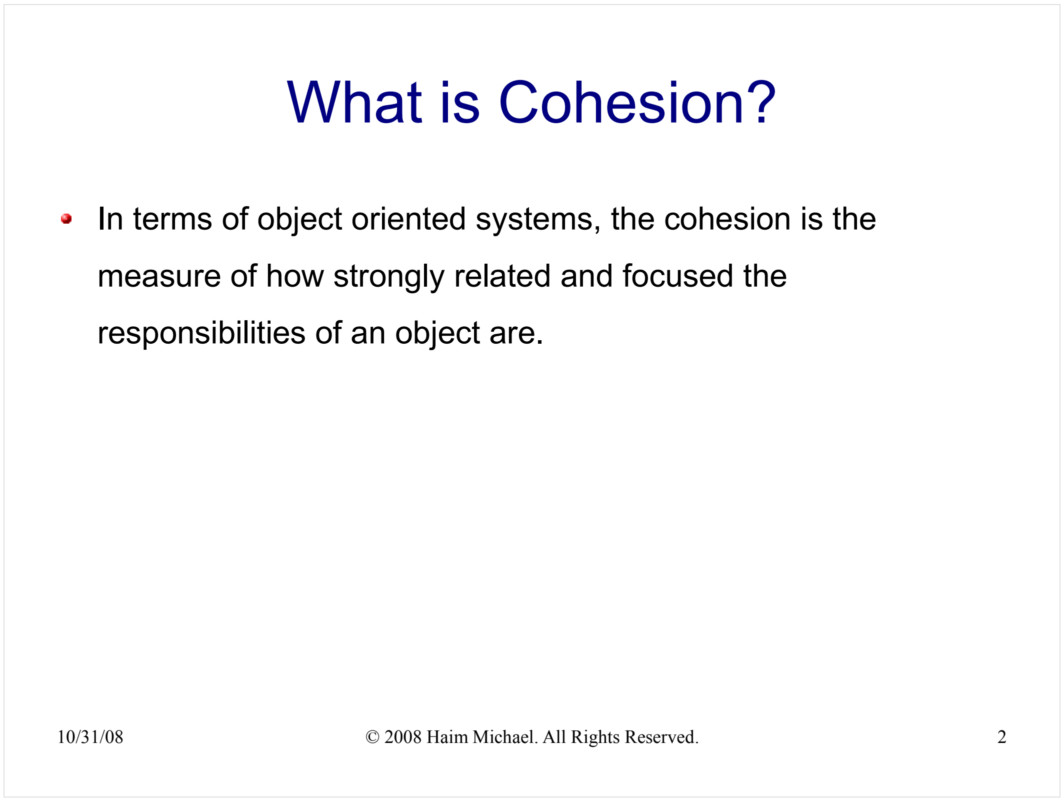
The Interface Segregation Principle

- “Clients should not be forced to depend on methods that they do not use.” (Robert Martin, 2002)

  Based on this principle, we should prefer having many specific interfaces than one general purpose one. When having one class that depends on another, that dependency should be on the smallest possible interface. Having a fat interface is a very bad practice. A fat interface is not cohesive.

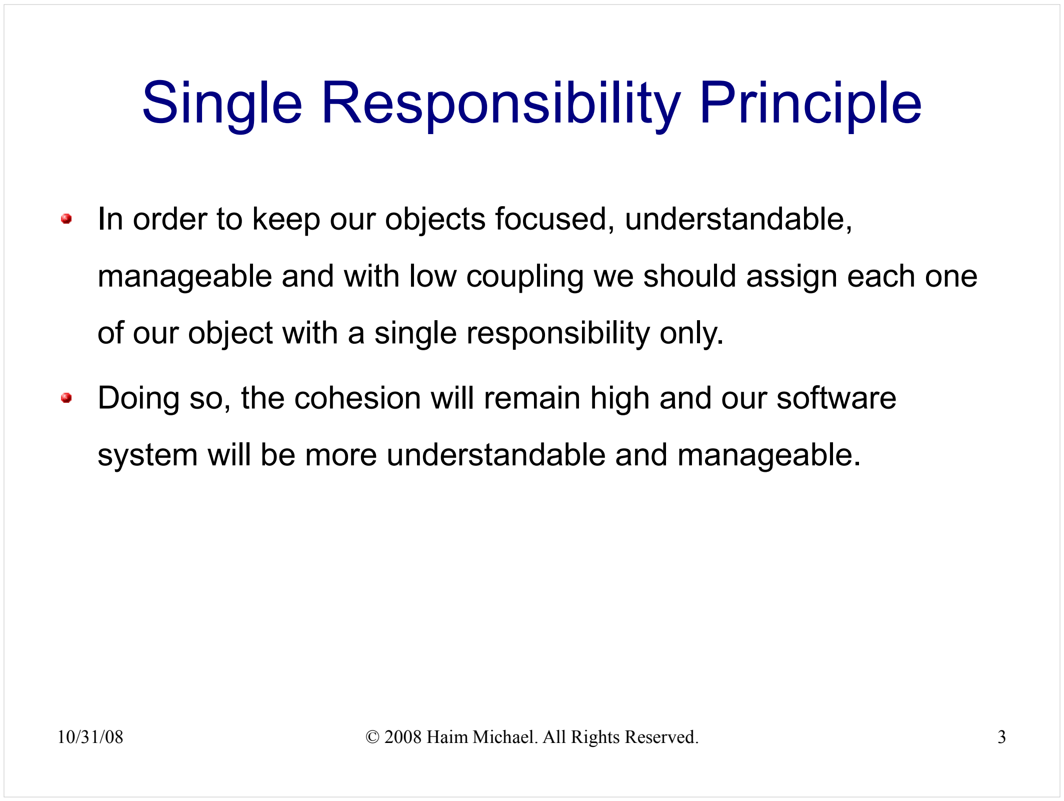


Class Design Principles



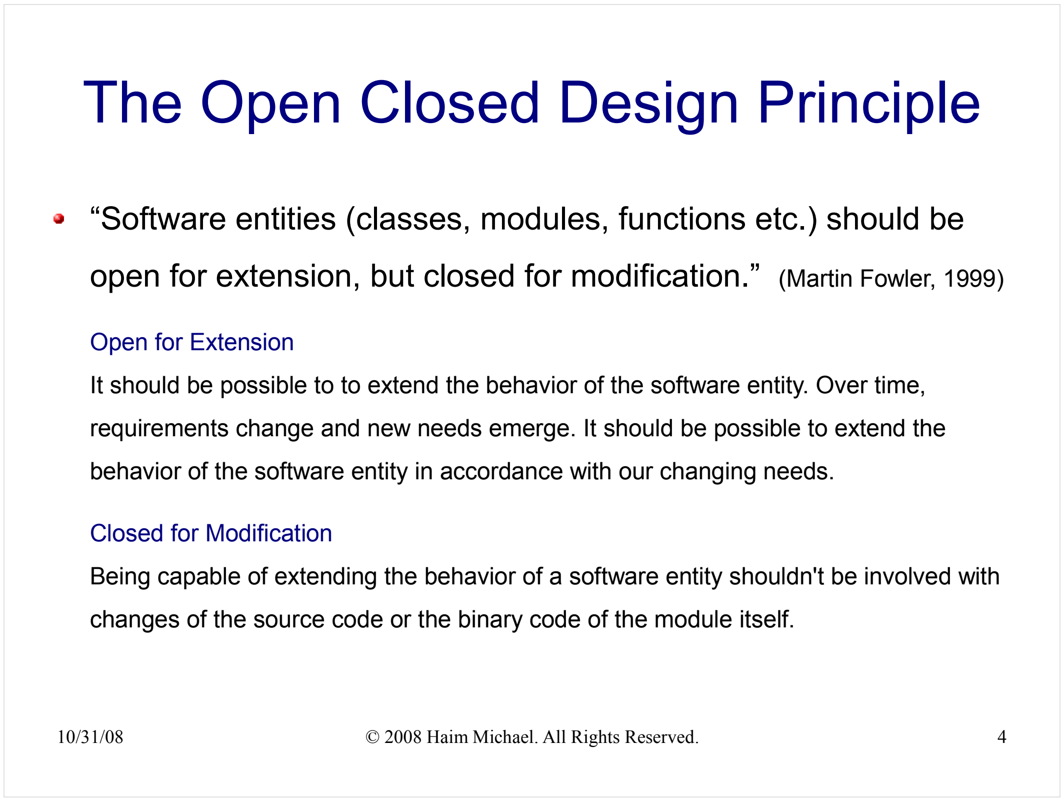
What is Cohesion?

- In terms of object oriented systems, the cohesion is the measure of how strongly related and focused the responsibilities of an object are.



Single Responsibility Principle

- In order to keep our objects focused, understandable, manageable and with low coupling we should assign each one of our object with a single responsibility only.
- Doing so, the cohesion will remain high and our software system will be more understandable and manageable.



The Open Closed Design Principle

- “Software entities (classes, modules, functions etc.) should be open for extension, but closed for modification.” (Martin Fowler, 1999)

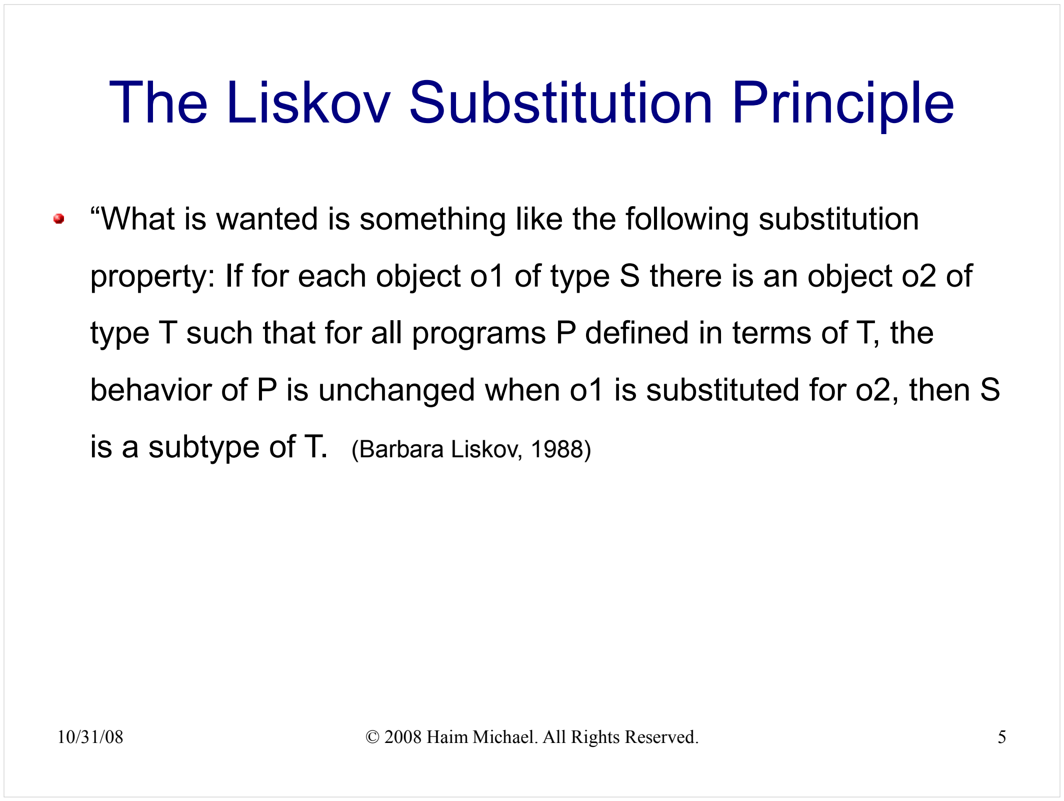
  Open for Extension

  It should be possible to to extend the behavior of the software entity. Over time, requirements change and new needs emerge. It should be possible to extend the behavior of the software entity in accordance with our changing needs.

  Closed for Modification

  Being capable of extending the behavior of a software entity shouldn't be involved with changes of the source code or the binary code of the module itself.

10/31/08 © 2008 Haim Michael. All Rights Reserved. 4



The Liskov Substitution Principle

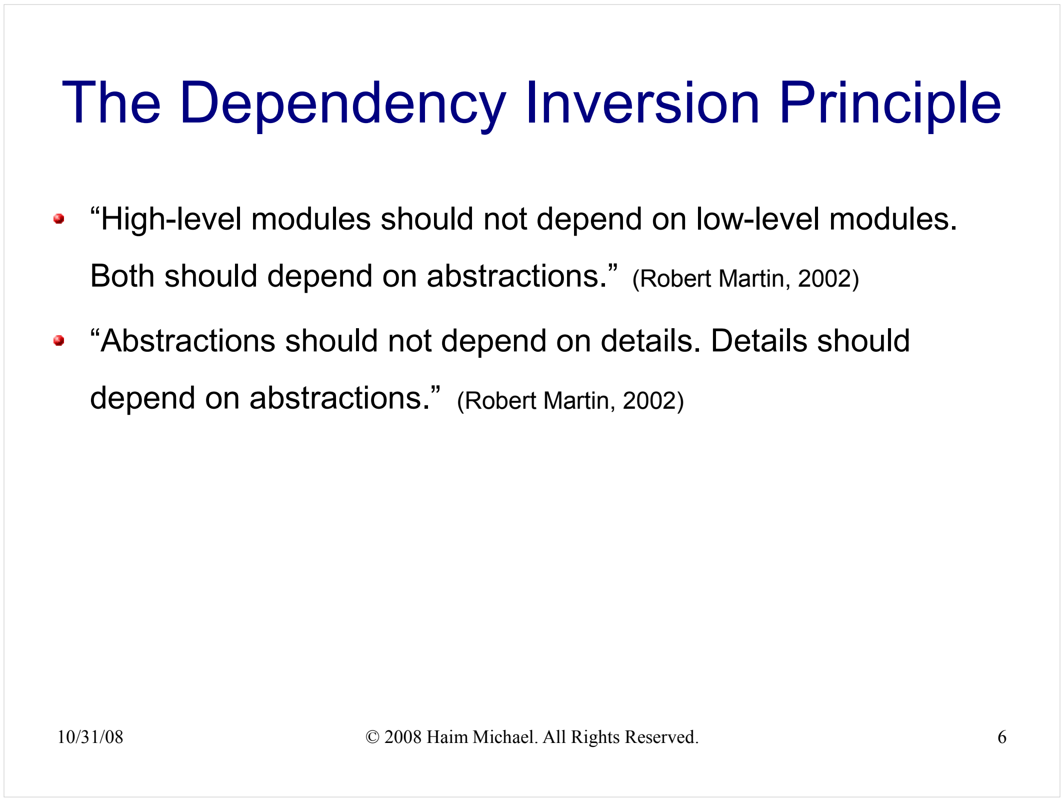
- "What is wanted is something like the following substitution property: If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2, then S is a subtype of T. (Barbara Liskov, 1988)

10/31/08 © 2008 Haim Michael. All Rights Reserved. 5

The Liskov Substitution principle is important for two reasons:

1. If Liskov Substitution principle doesn't exist then the class hierarchies would be a mess. Whenever a subclass instance is passed as a parameter strange behavior would have happened.

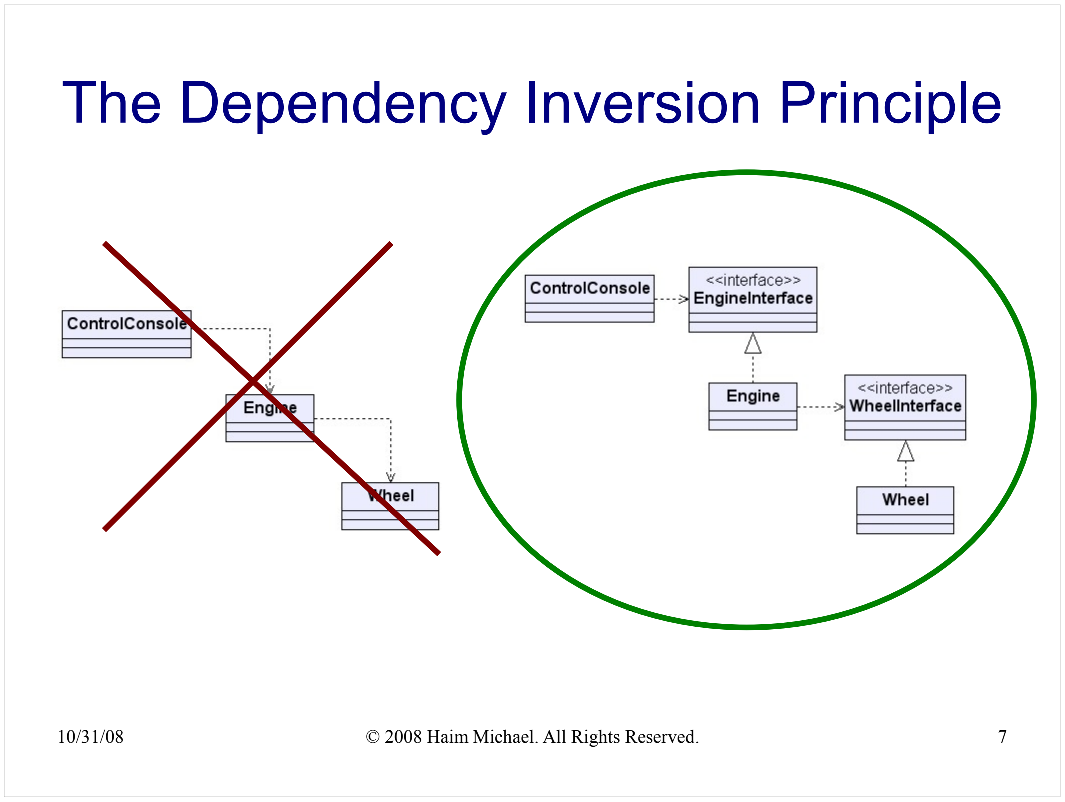
2. If Liskov Substitution principle doesn't exist then Unit tests would have failed for the subclasses.



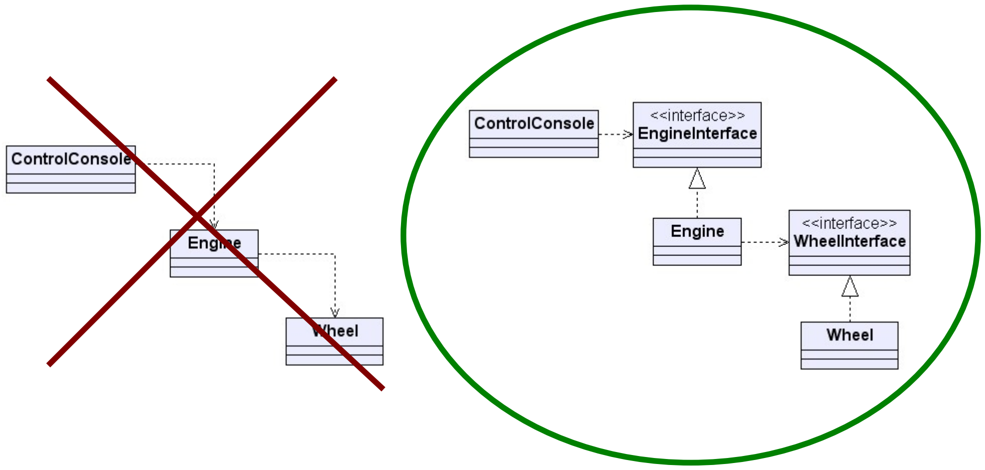
The Dependency Inversion Principle

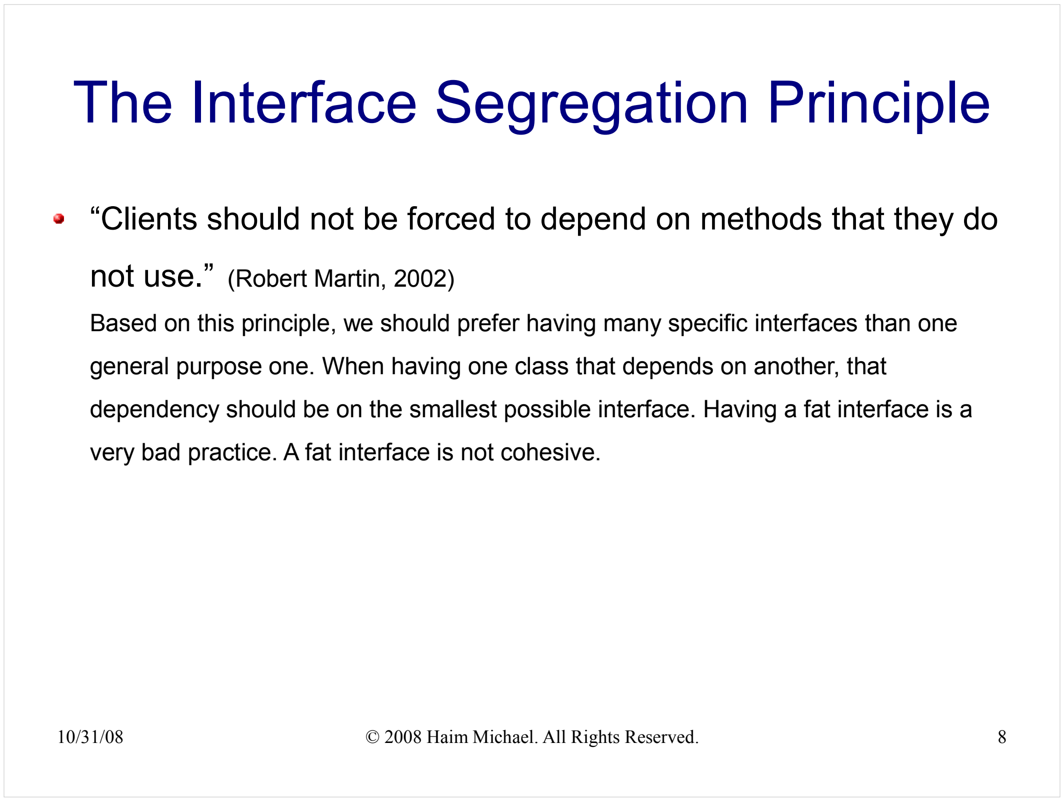
- “High-level modules should not depend on low-level modules. Both should depend on abstractions.” (Robert Martin, 2002)
- “Abstractions should not depend on details. Details should depend on abstractions.” (Robert Martin, 2002)

10/31/08 © 2008 Haim Michael. All Rights Reserved. 6



The Dependency Inversion Principle





The Interface Segregation Principle

- “Clients should not be forced to depend on methods that they do not use.” (Robert Martin, 2002)

  Based on this principle, we should prefer having many specific interfaces than one general purpose one. When having one class that depends on another, that dependency should be on the smallest possible interface. Having a fat interface is a very bad practice. A fat interface is not cohesive.

10/31/08 © 2008 Haim Michael. All Rights Reserved. 8