# CS4125
# SYSTEMS ANALYSIS
## SPRING SEMESTER 2010-2011

J.J. Collins
Dept of CSIS
University of Limerick

Lecture R – W12L1

Detailed Design

# 1. Further Design Guidelines



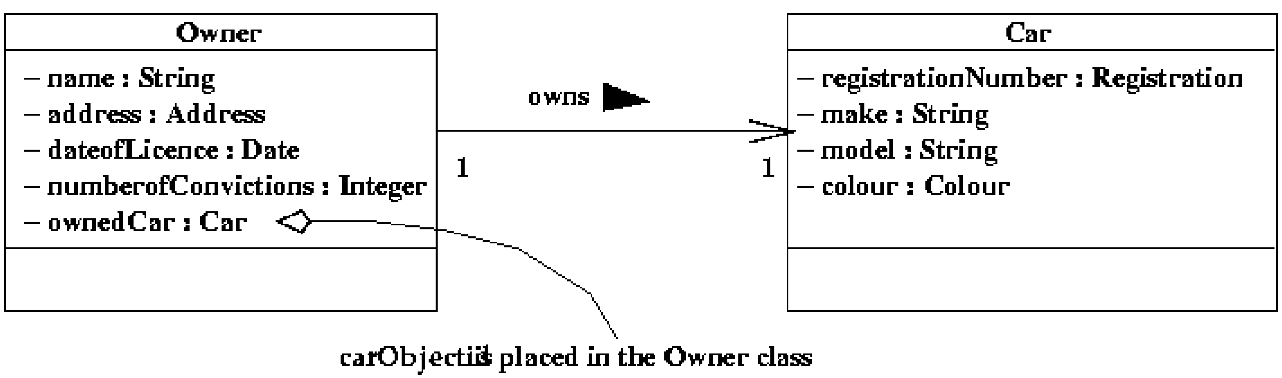
Coad and Yourdon (1994) and Yourdon (1994) suggest:

- Design clarity - use standard design protocols that have been specified.
- Don't over design - systems that are over designed may be difficult to extend if the modifications are not sympathetic to the existing structure.
- Control inheritance hierarchies - Rumbaugh et al. (1991) suggest that four is the maximum.
- Keep messages and operations simple. Limit number of parameters passed to three. Method specification should be no more than one page.
- Design volatility: a good design will be stable and commensurate with changes in the requirements. Enforcing encapsulation is a key factor here.
- Evaluate by scenario - role play it against use cases using CRC cards.
- Design by delegation - a complex object should be decomposed into component objects using composition or aggregation.
- Keep classes separate.

# 2. Designing Associations



- An association between two classes indicates the possibility that links will exist between objects of the classes.
- Links provide the connections necessary for message passing to occur.
- Objects of the class Owner need to send messages to objects of the class Car, but not vice versa.
- Before an association can be designed, must determine navigability or directions in which messages are sent.
- Two questions:
  - Do objects of class A send messages to objects of class B.
  - Does an A object have to provide some other object with B object identifiers.
- If the answer is yes, then object A needs object B's identifier.
- However, if object A gets object B's identifier in an incoming message, no need for A to remember B's identifier.
- Minimising two-way associations keeps the coupling between object as low as possible.

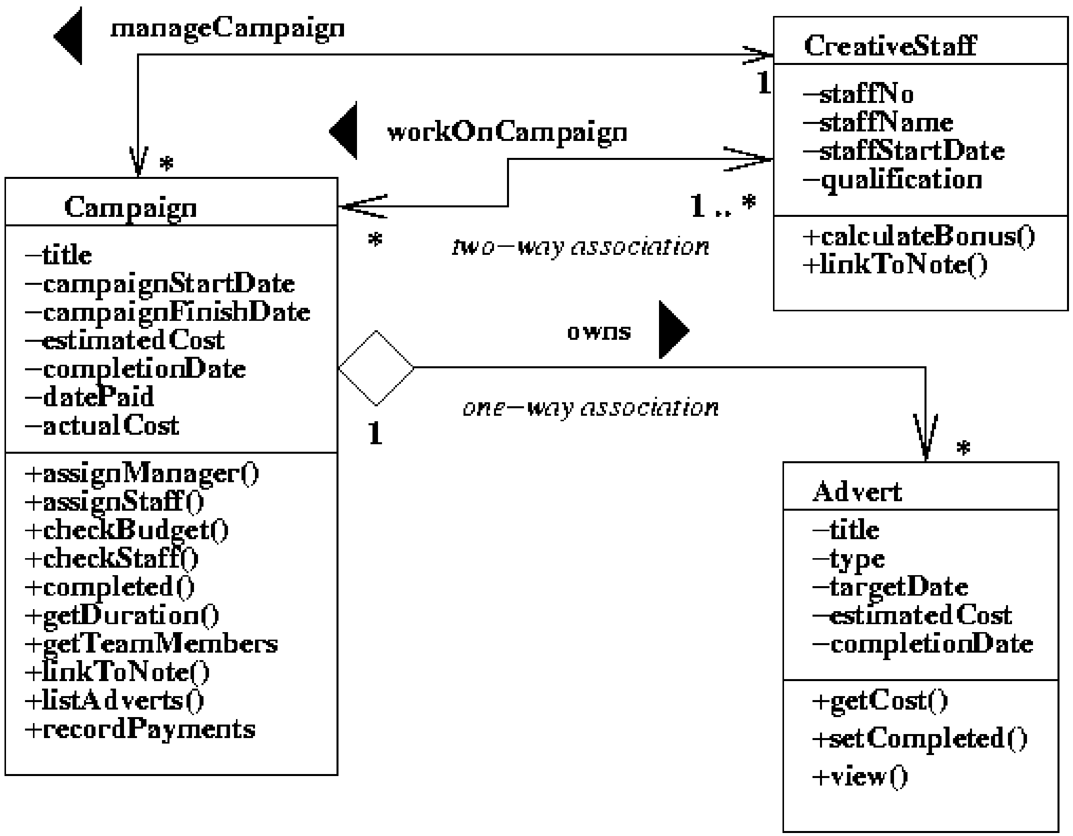



**One way one to one association**

# 2. Designing Associations



- In figure, objects of the class Campaign need to send messages to objects of the class Advert but not vice versa.

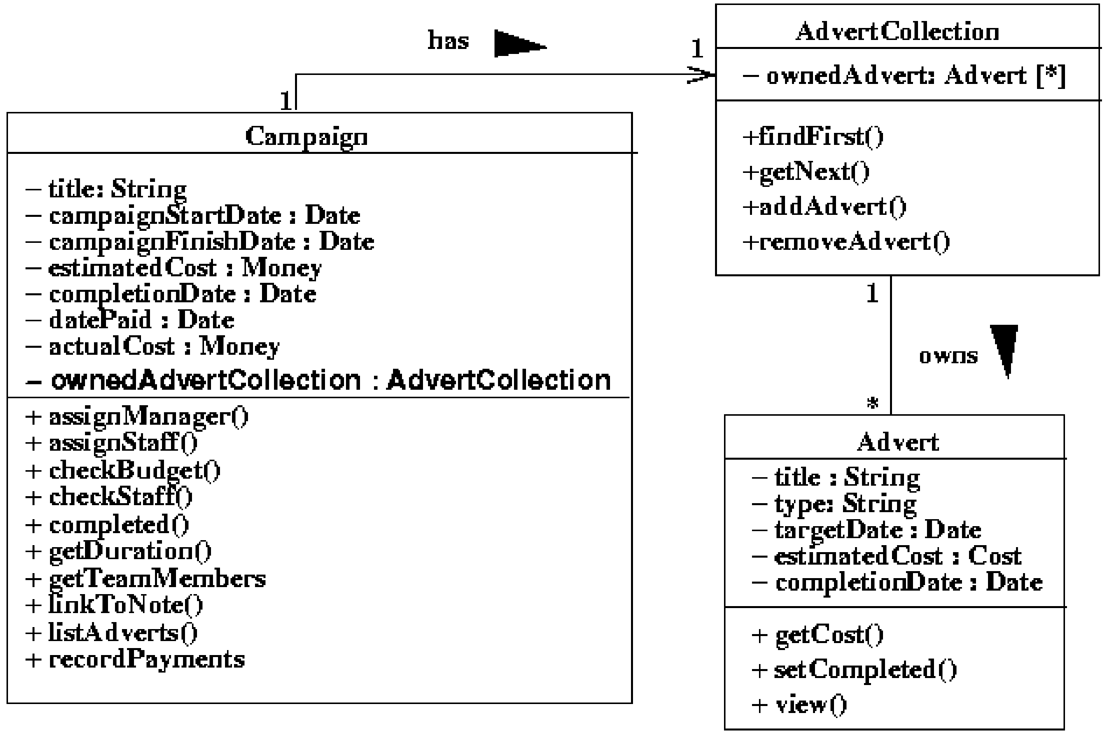



Fragment of class diagram for Agate case study

# 2. Designing Associations



- If the association between the classes was one to one, association could be implemented by placing an attribute to hold the object identifier for Advert in Campaign.
- But the multiplicity is one to many.
- Could use a 1D array or a collection class.

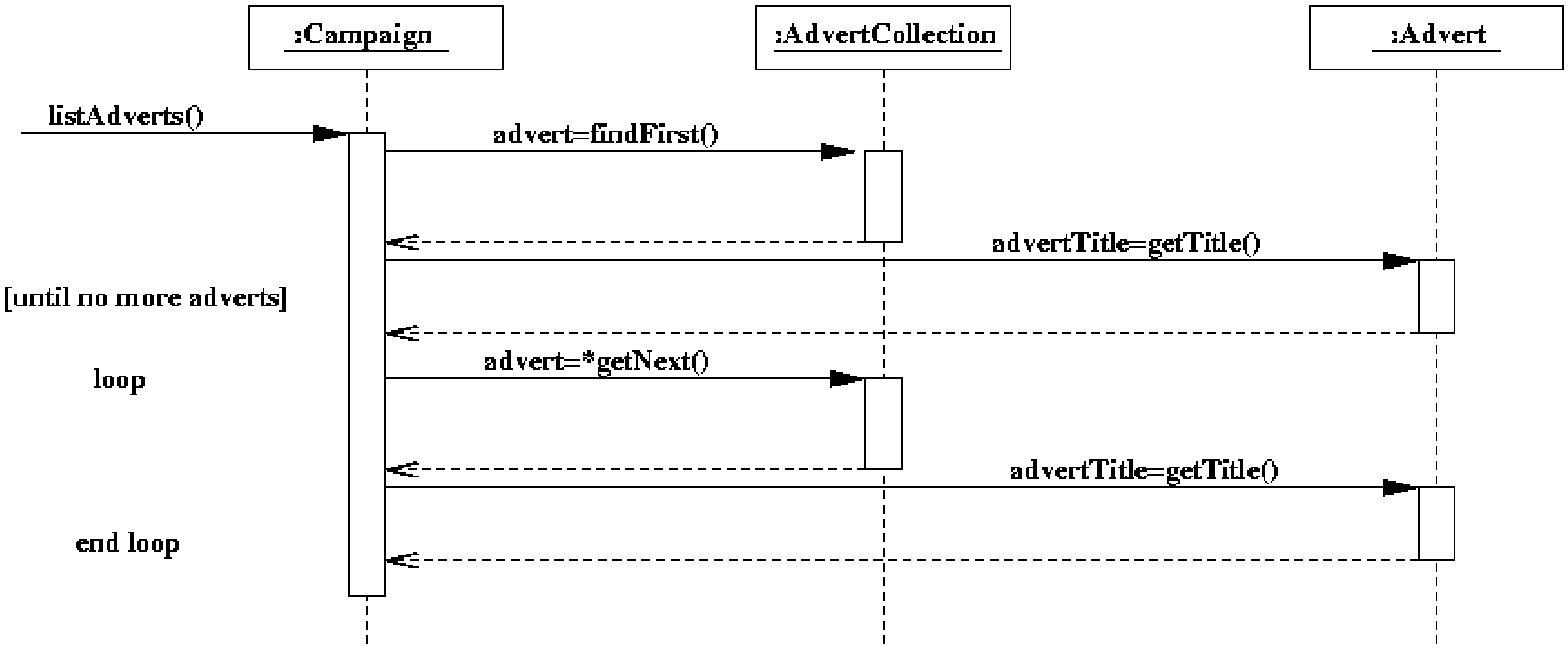



One to many associations using a collection class

# 2. Designing Associations



- Note that the Advert collection class has operations specifically concerned with the management of the collection. Also facilitates reuse.
- Shown is the sequence diagram for the interaction that would enable a Campaign object to prepare a list of its adverts with their titles.

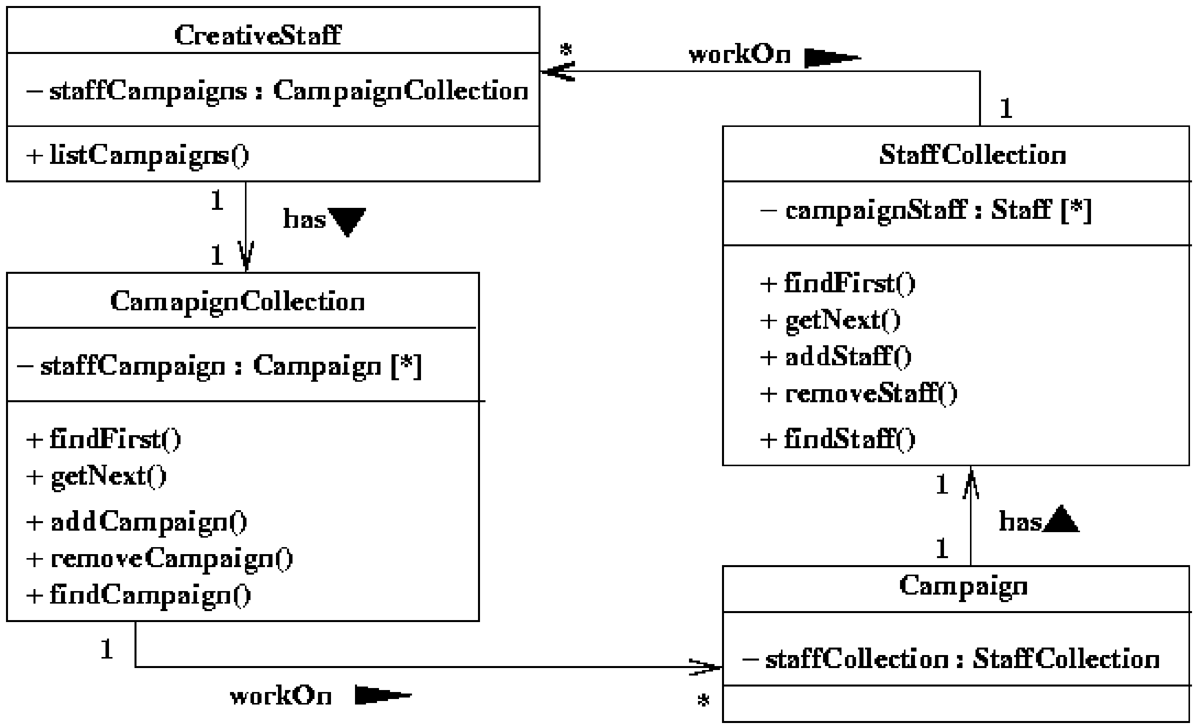



Sequence diagram for listAdverts()

# 2. Designing Associations



- If there is a requirement to find out if an employee works on a campaign with a particular title, a message may be sent from CreativeStaff object to each Campaign object the employee works on to get its title until either a match is found or the end of the collection has been reached.




Two–way many–to–many association

# 3. Integrity Constraints



- Referential integrity ensures that an object identifier in an object is actually referring to an object that exits - through use of constructors and destructors.
- Dependency constraints ensures that attribute dependencies are maintained correctly - use of synchronisation messages.
- Domain Integrity ensures that attributes only hold permissible values.

# 4. Normalisation



- Functional dependency: for two attributes A and B, A is functionally dependent on B if for every value of B, there is precisely one value of A associated with it at any given time.

$$B \rightarrow A$$

- Normalisation prevents redundancy – duplication of attributes in system model. Duplication leads to inconsistency because of synchronisation problems.
- Normalisation used when specifying tables for relational databases.
- Most OO approaches do not view normalisation as essential.
- However, if OO paradigm applied with suitable quality constraints, artefacts will be produced that are largely redundancy free.

# 3. Coupling and Cohesion

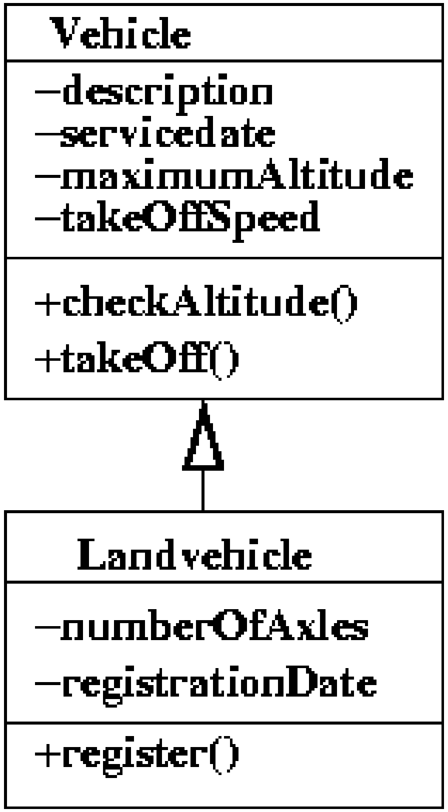


- Yourdon and Constantine (1979) identified two concerns – coupling and cohesion, that could be used in decomposition of system into modules.
- Examples of poor cohesion:
  1. Coincidental cohesion
  2. Logical cohesion
  3. Temporal cohesion
  4. Sequential cohesion

# 3. Coupling and Cohesion



- Low coupling and high cohesion are among the criteria for good design.
- Cohesion is a measure of the degree to which an element contributes to a single purpose.
- Coad and Yourdon (1991) describe the following:
  - Interaction coupling: the number of messages an object sends to other objects, and the number of parameters in a message. Should be minimised.
  - Inheritance coupling describes the degree to which a subclass actually needs the features it inherits from its superclass.

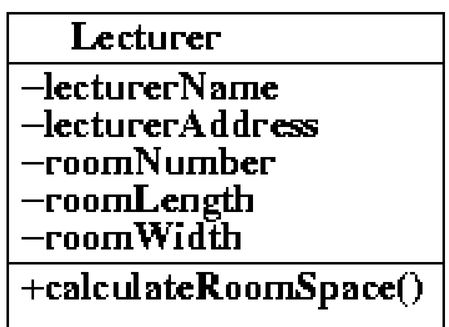



Inheritance Coupling

# 3. Coupling and Cohesion



- Operation cohesion: measures the degree to which an opertaion focuses on a single functional requirement. Good design produces highly cohesive operations. For example, operation calculateRoomSpace() is highly cohesive.
- Class cohesion reflects the degree to which a class is focued on a single requirement. The class Lecture exhibits low level of cohesion.

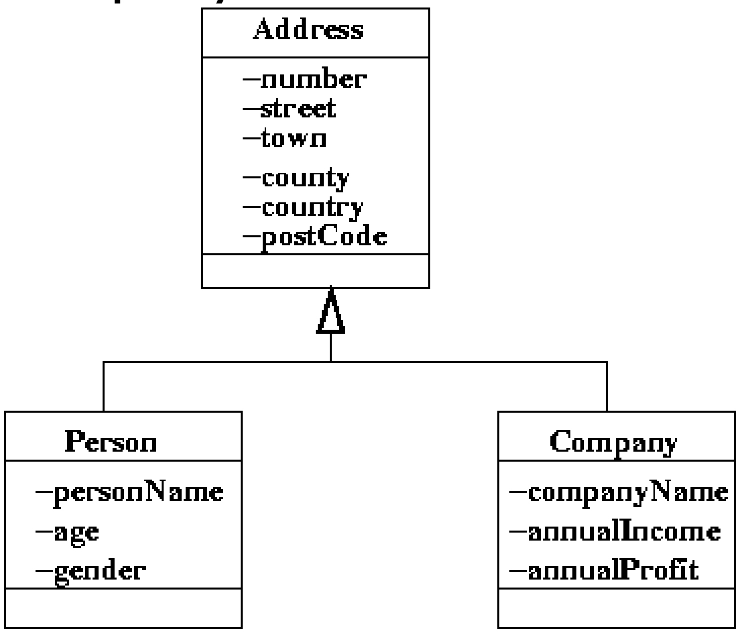



**Good operation cohesion, but poor class cohesion**

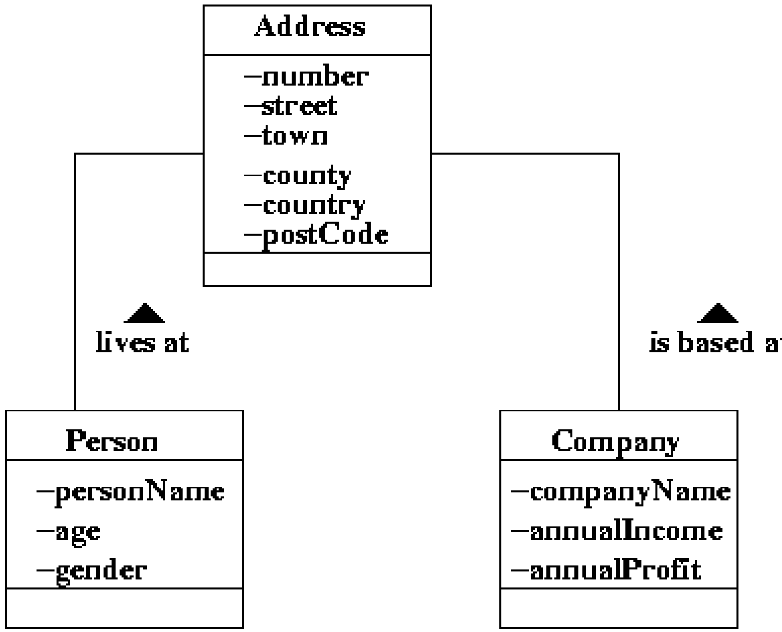
# 3. Coupling and Cohesion



- Specialisation cohesion addresses the semantic cohesion of inheritance hierarchies.
- All the features of the superclass Address are used by derived classes., the hierarchy has high inheritance coupling, but has low specialisation cohesion because it is NOT true that a person or a company is a kind of address. Should have used associations.




Poor specialisation cohesion

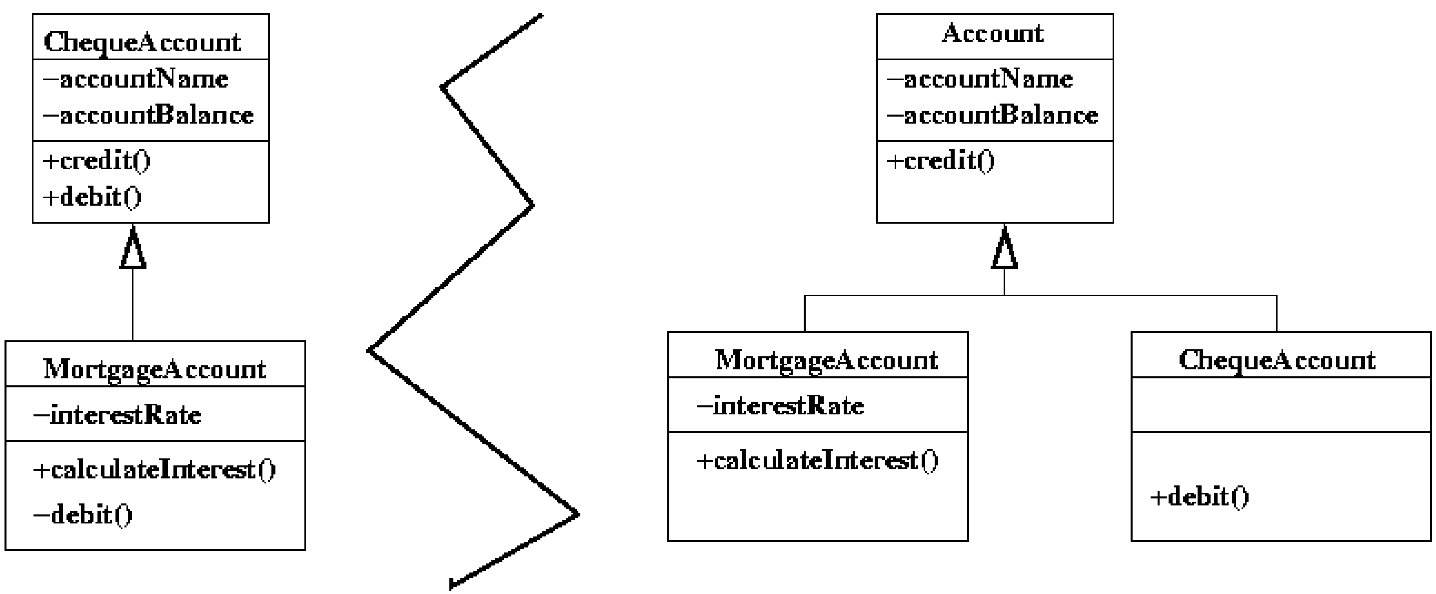



Improved structure using associations

# 4. Liskov Substitution Principle



- Applicable to inheritance hierarchies.
- It sates that in object interactions, it should be possible to treat a derived class as if it were a base class.
- If the principle is not applied, then it may be possible to violate the integrity of the derived class.
- Applying the LSP normally results in a design with maximal inheritance coupling.




**Application of the Liskov Substitution Principle**

# 5. Reading



- Chapter 14 in Bennett et al. (Fourth Edition).