Chap. 14
Detailed Design

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill

Topics Covered

- What Do We Add in O-O Detailed Design?
- Class Specification
- Interfaces
- Criteria for Good Design
- Designing Associations
- Integrity Constraints
- Designing Operations

Detailed Design

- Object-oriented detailed design adds detail to the analysis model
  - types of attributes
  - operation signatures
  - assigning responsibilities as operations
  - additional classes to handle user interface
  - additional classes to handle data management
  - design of reusable components
  - assigning classes to packages

Class Specification : Attributes

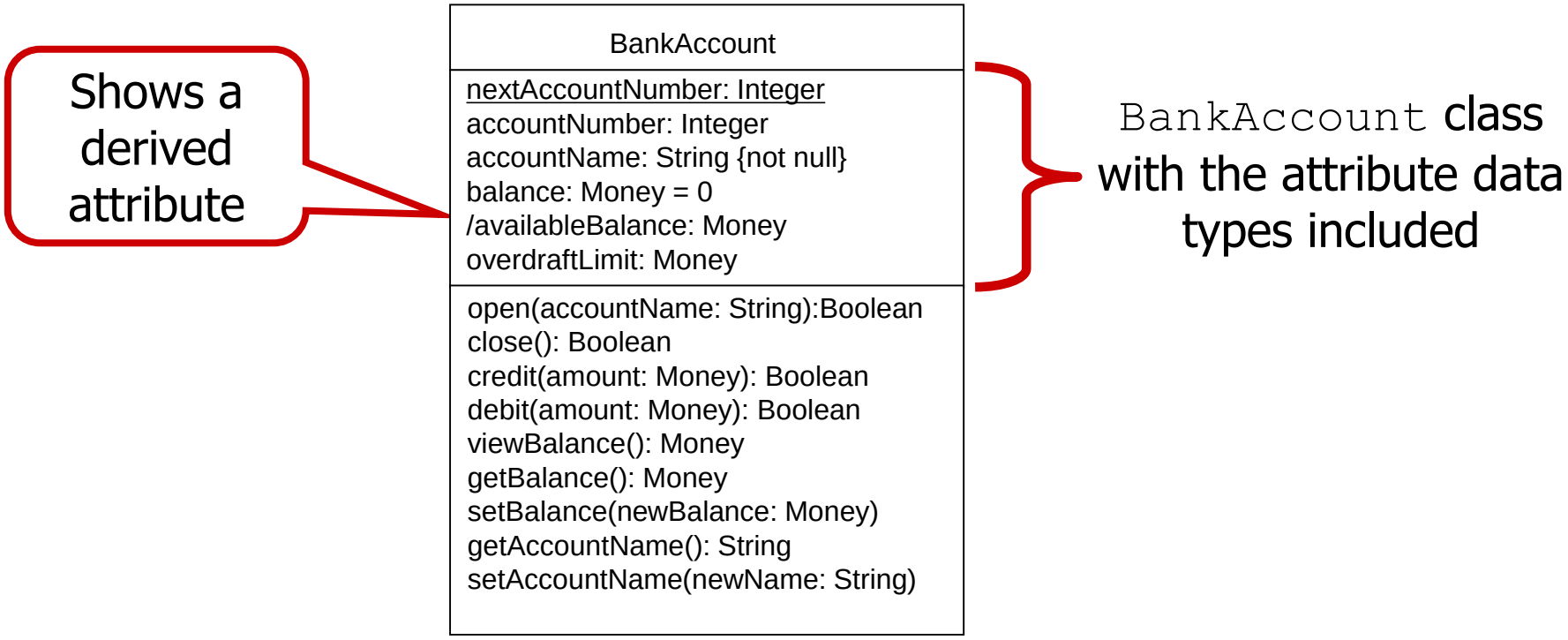
- An attribute's data type is declared in UML using the following syntax

```
name ':' type-expression '=' initial-value
        '{'property-string'}'
```

Where

- name is the attribute name
- type-expression is its data type
- initial-value is the value the attribute is set to when the object is first created
- property-string describes a property of the attribute, such as constant or fixed

Class Specification: Attributes



Class Specification: Attributes

- The attribute balance in a BankAccount class might be declared with an initial value of zero using the syntax

```
balance:Money = 0.00
```

- Attributes that may not be null are specified

```
accountName:String {not null}
```

- Arrays are specified

```
qualification[0..10]:String
```

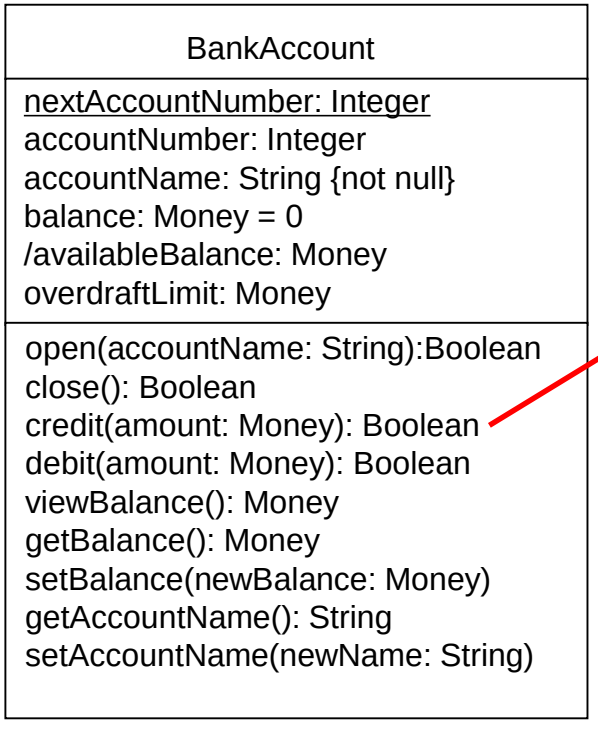
Class Specification: Operations

- The syntax used for an operation is

```
operation name '(' parameter-list ')' ':'
  return-type-expression
```

- An operation's *signature* is determined by
  - the operation's name
  - the number and type of its parameters
  - the type of the return value if any

Class Specification: Operations



- Message example

```
CreditOK = accObject.credit(500.0)
```

`BankAccount` class with operation signatures included.

Which Operations?

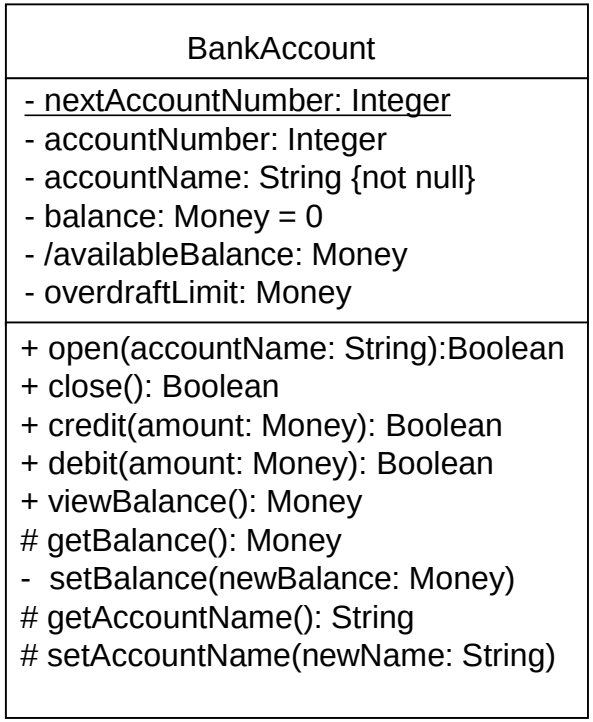
- Generally don't show primary operations
  - constructors, destructors, get and set operations
  - Only show constructors where they have special significance

Visibility

Visibility symbol	Visibility	Meaning
+	Public	The feature (an operation or an attribute) is directly accessible by an instance of any class.
-	Private	The feature may only be used by an instance the class that includes it.
#	Protected	The feature may be used either by the class that includes it or by a subclass or descendant of that class.
~	Package	The feature is directly accessible only by instances of a class in the same package.

Visibility

`BankAccount` class with visibility specified

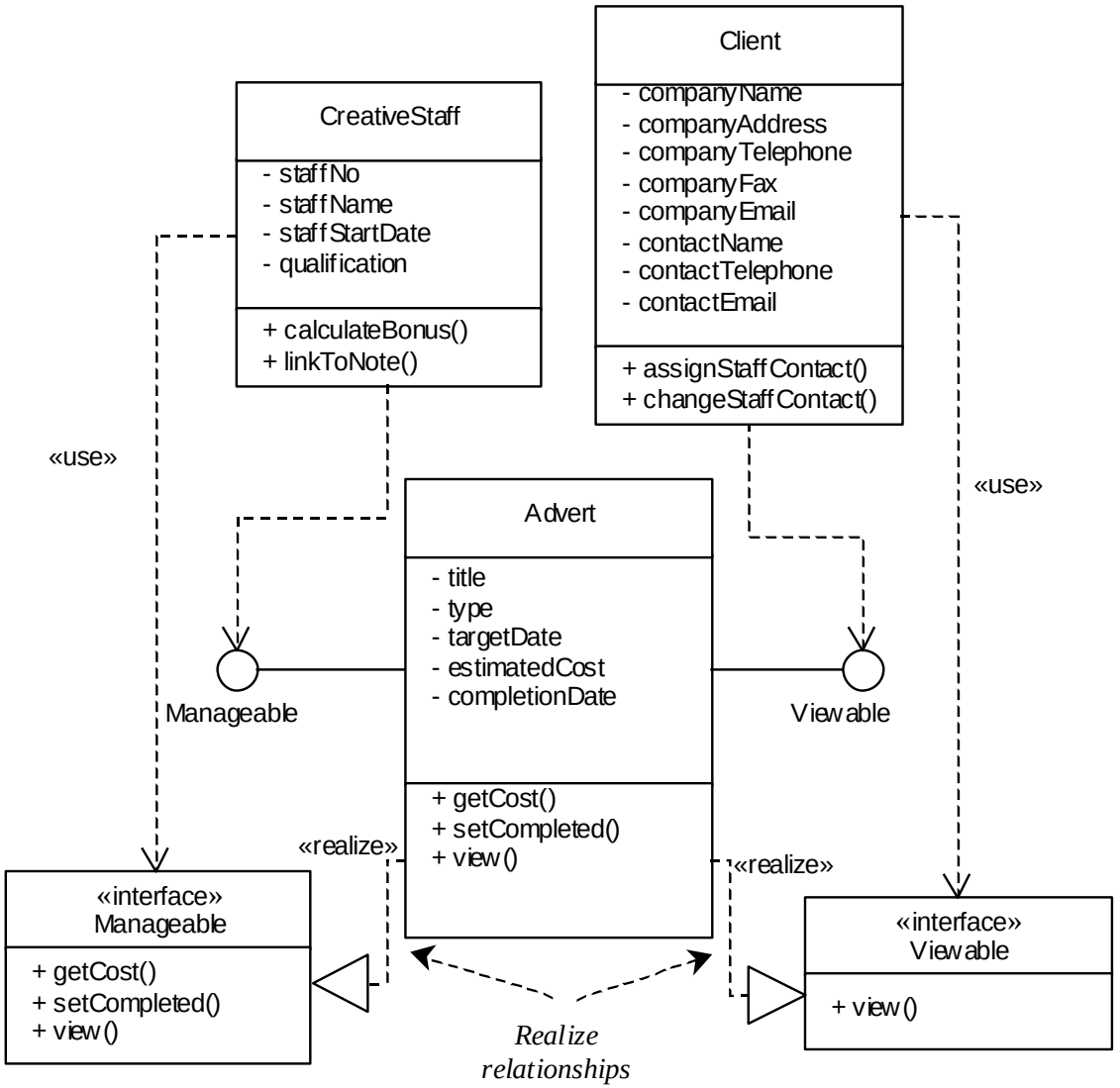


A class-scope attribute is attached to the class, not to any individual object

Interfaces

- UML supports two notations to show interfaces
  - The small circle icon showing no detail
  - A stereotyped class icon with a list of the operations supported
- The *realize* relationship, represented by the dashed line with a triangular arrowhead, indicates that the client class (e.g. Advert) supports at least the operations listed in the interface
  - Could indicate inheritance relationship

Interfaces for the `Advert` class



Criteria for Good Design

- Coupling
  - describes the degree of interconnectedness between design components
- Cohesion
  - a measure of the degree to which an element contributes to a single purpose

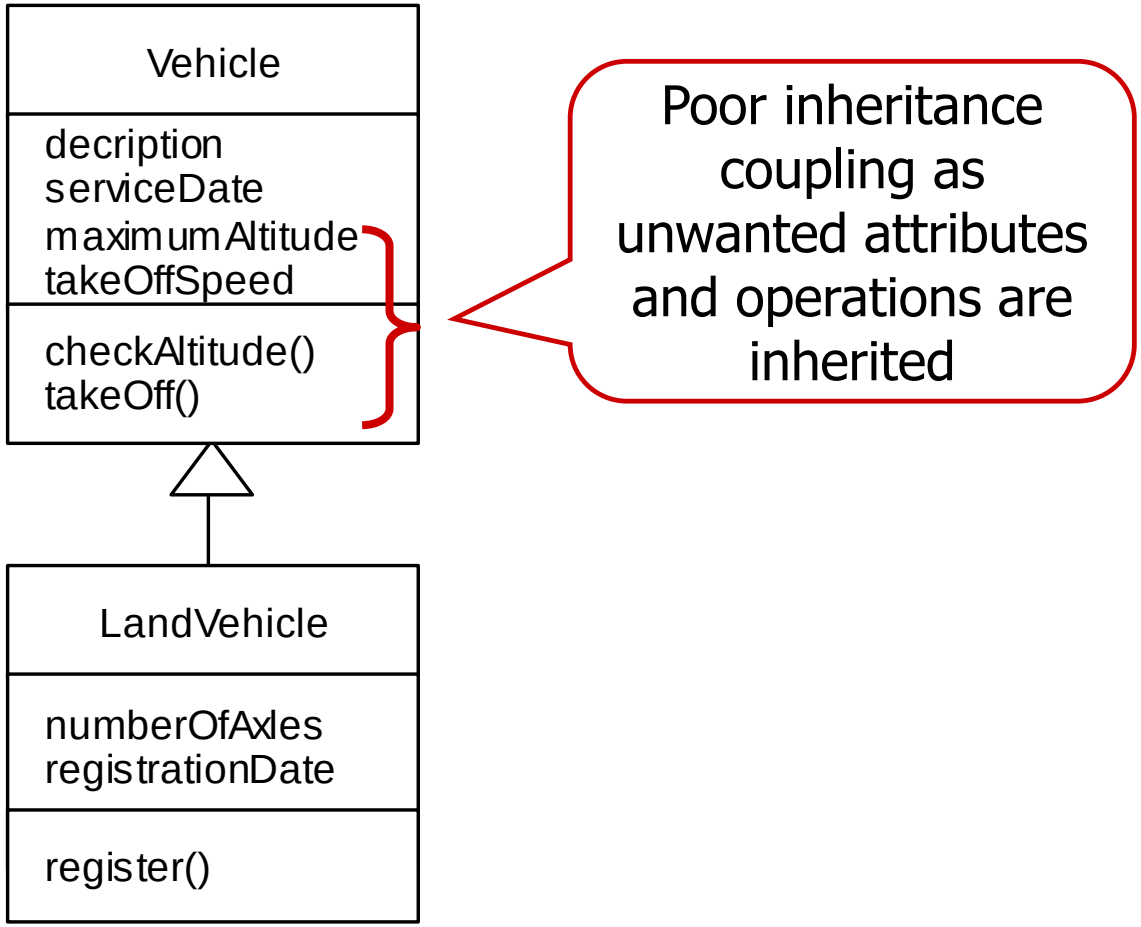
- The concepts of coupling and cohesion are not mutually exclusive but actually support each other

Interaction Coupling

- A measure of
  - the number of message types an object sends to other objects and
  - the number of parameters passed with these message types.
- Should be kept to a minimum
  - to reduce the possibility of changes and
  - to make reuse easier.

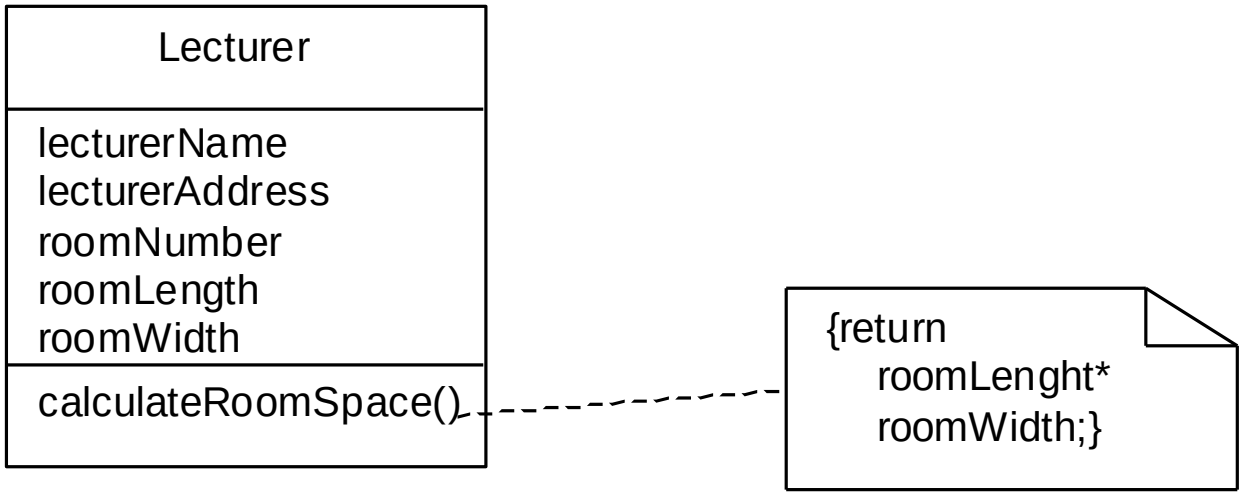
Inheritance Coupling

Inheritance Coupling describes the degree to which a subclass actually needs the features it inherits from its base class.



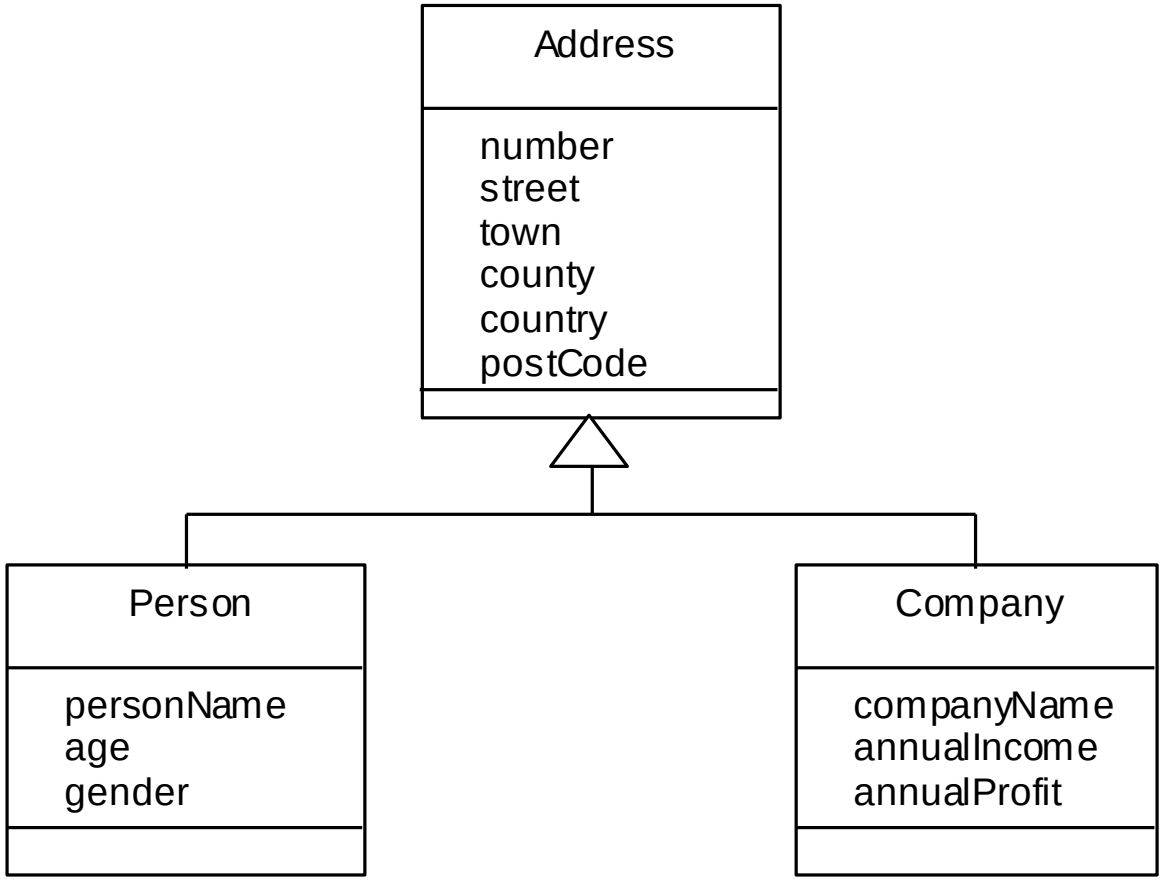
Operation/Class Cohesion

* **Operation cohesion measures the degree to which an operation focuses on a single functional requirement.**
* **Class cohesion reflects the degree to which a class is focused on a single requirement.**



Good operation cohesion but poor class cohesion

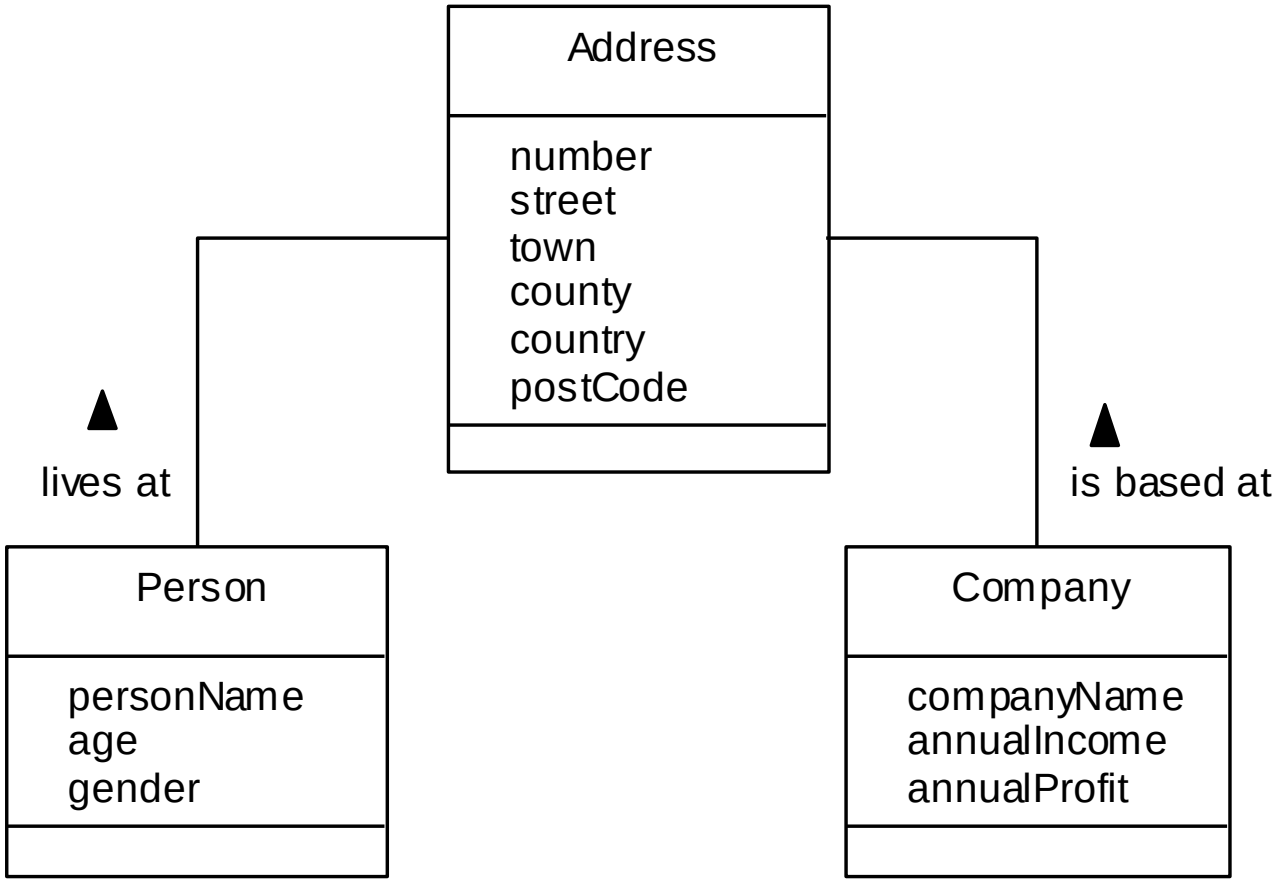
Poor Specialization Cohesion



Specialization Cohesion addresses the semantic cohesion of inheritance hierarchies

Improved Structure

Improved structure using `Address` class.

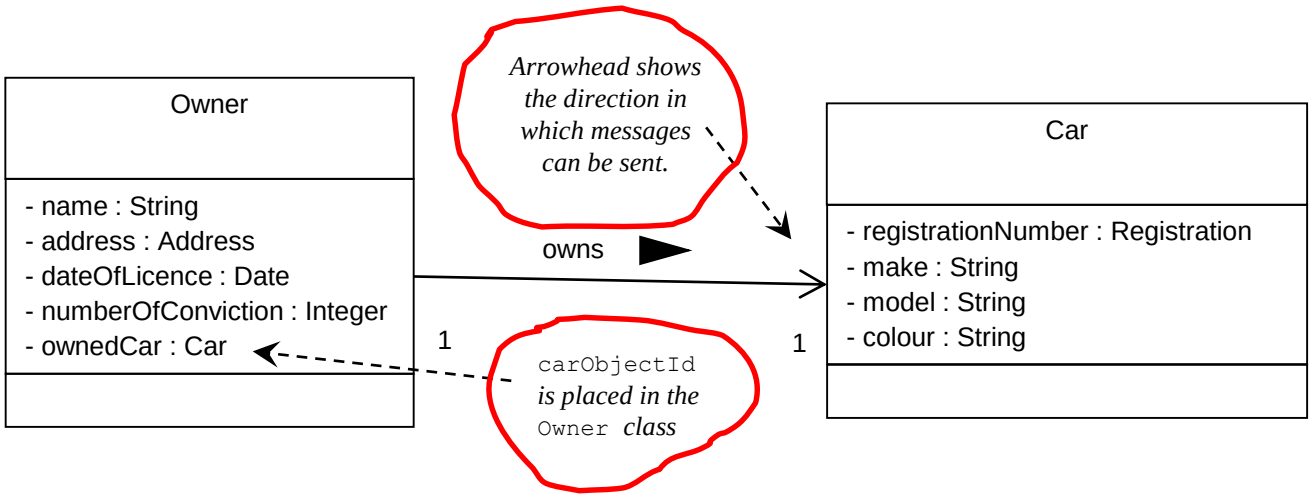


Designing Associations

- Association
  - Indicates possibility that links will exist between instances of the classes
- Link
  - Provide the connections necessary for message passing to occur

- Need to send messages to objects of a class
  - Place an attribute to hold the object identifier (object reference)

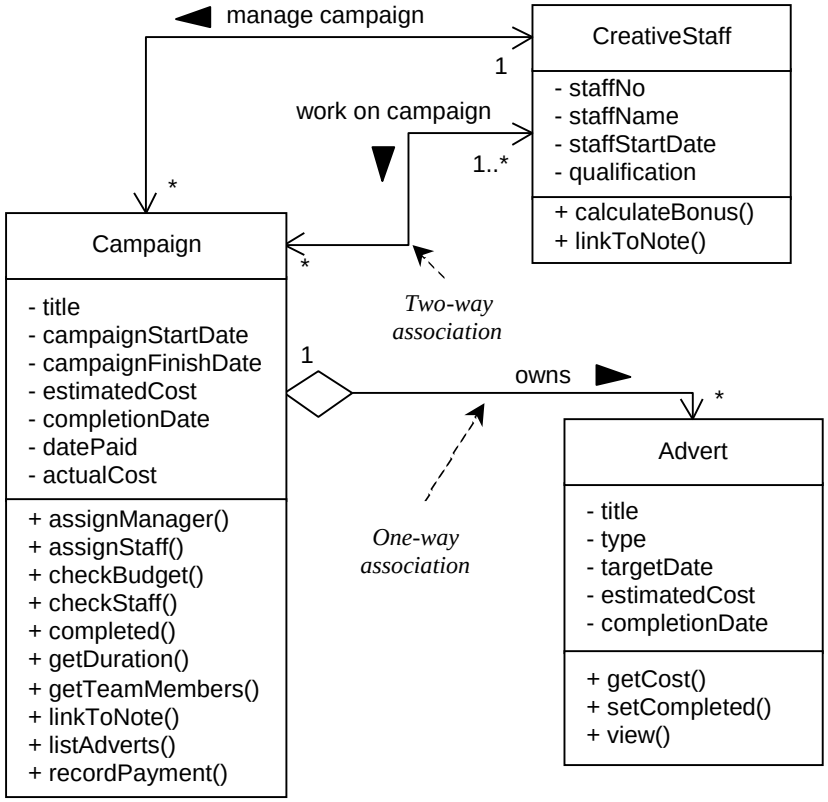
"One-way one-to-one" associations



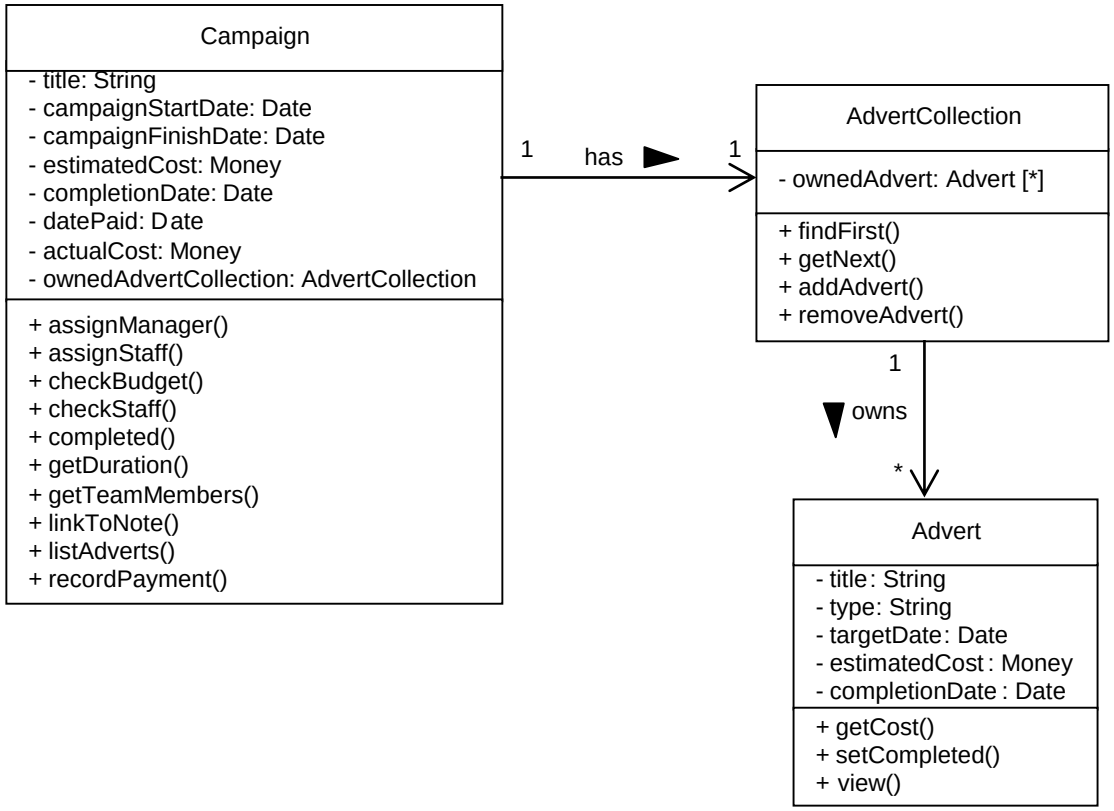
Designing Associations

- Two-way association
  - An association that has to support message passing in both directions
  - indicated with arrowheads at both ends
  - Minimizing the number of two-way associations keeps the coupling between objects as low as possible

Fragment of class diagram for the Agate case study



One-to-many associations using a collection class

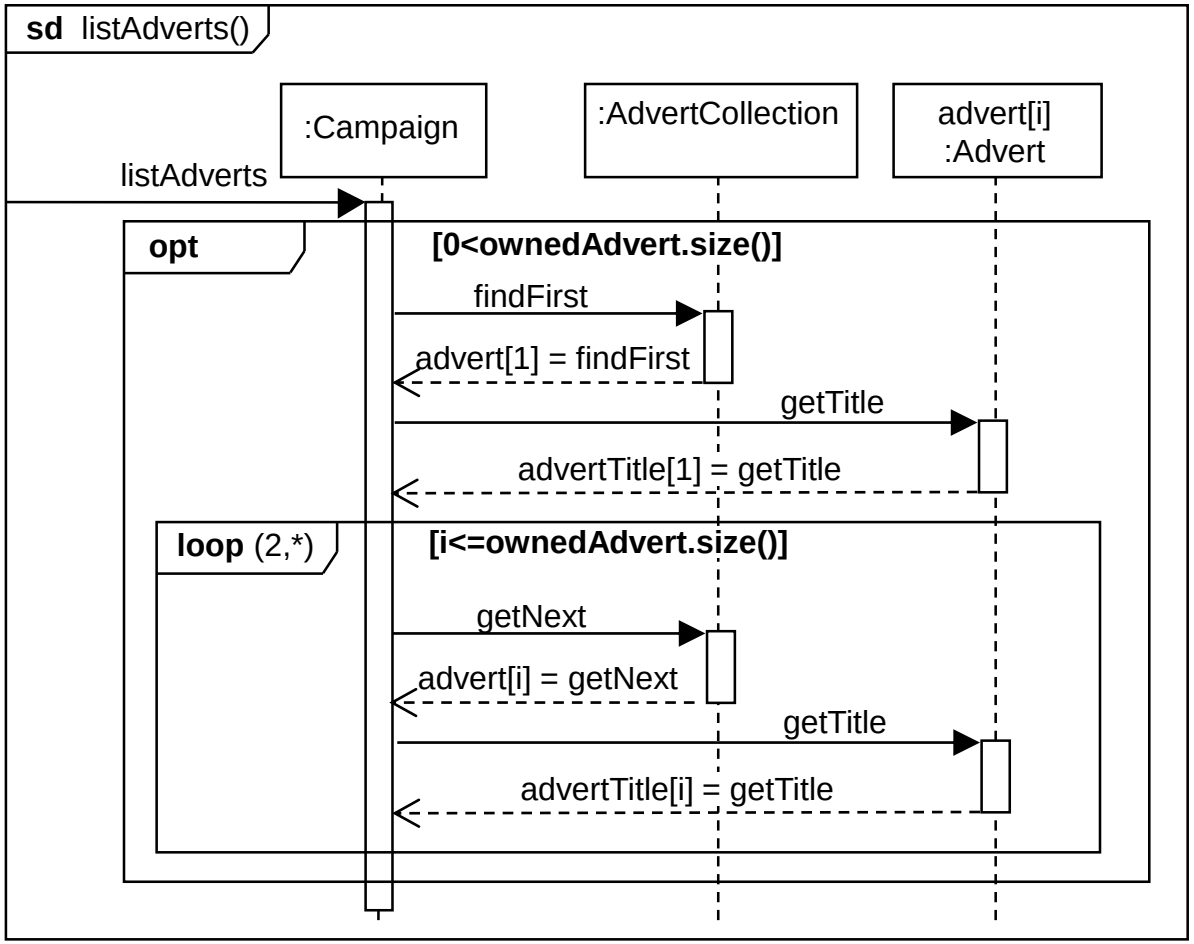


Collection Classes

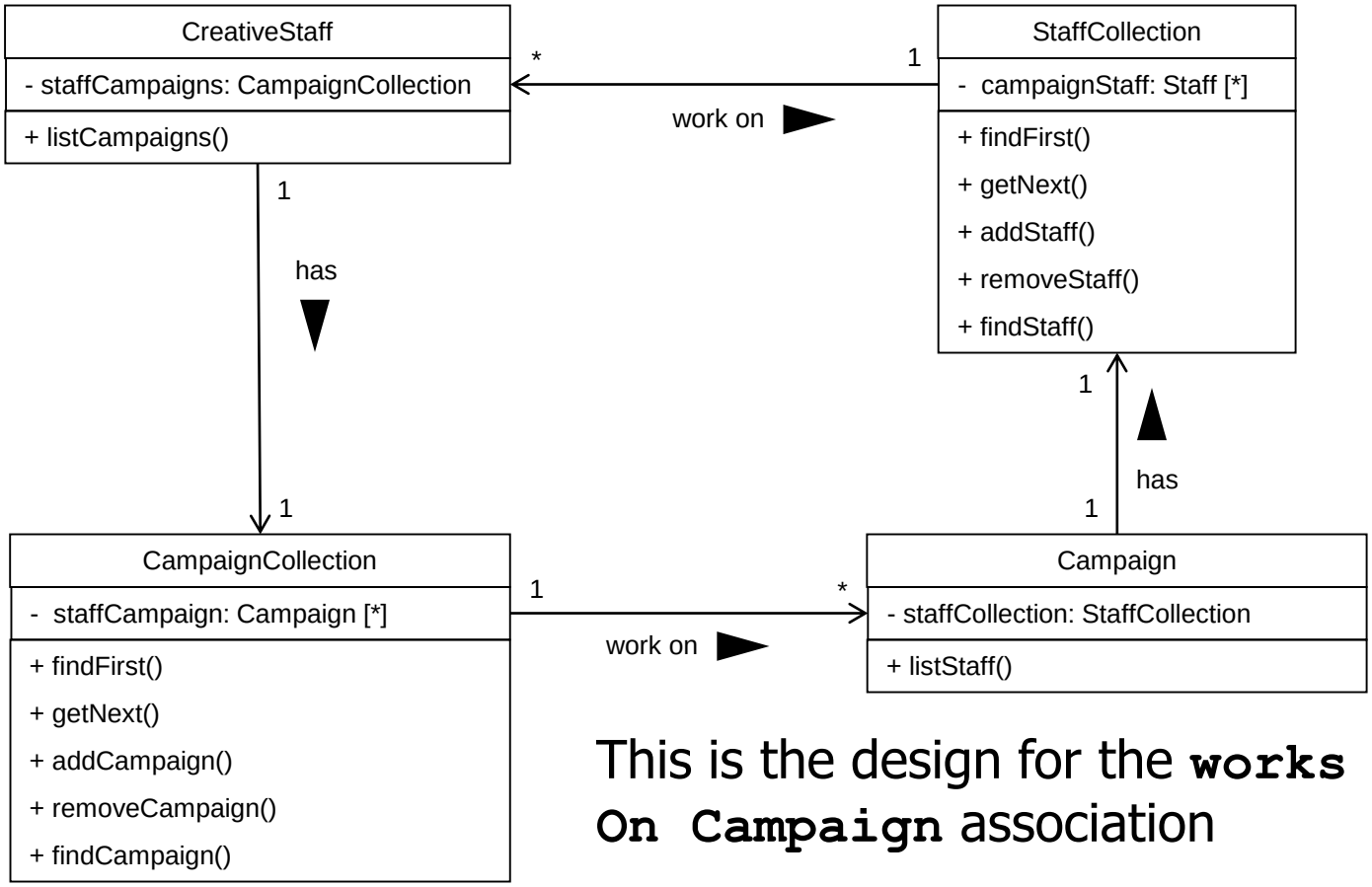
- Collection classes are used to separately hold object identifiers when message passing is required from one to many along an association
- OO languages provide support for these requirements. Frequently the collection class may be implemented as part of the sending class (e.g. `Campaign`) as some form of list

Sequence diagram for `listAdverts()`

This sequence diagram shows the interaction when using a collection class



Two-way many-to-many associations



This is the design for the **`works On Campaign`** association