ICS 3105
OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter 5.2

Class Diagrams

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Identify classes from a narrative.
  - Represent textual system description using class diagrams.

Introduction

- Object-oriented systems make use of abstraction in order to help make software less complex.
- An abstraction is something that relieves you from having to deal with details.
- Object-oriented systems combine procedural abstraction with data abstraction.

Introduction

- Data abstractions can help reduce some of a system's complexity.
- Records and structures were the first data abstractions to be introduced.
- The idea is to group together the pieces of data that describe some entity, so that programmers can manipulate that data as a unit.

Class Diagrams

- Class diagrams describe the data found in a software system.
- Many of the classes in these diagrams correspond to things in the real world e.g. example, in an airline reservation system there would be classes such as Flight, Passenger and Airport.

Class Diagrams

- The main symbols shown on class diagrams are:
  - Classes, which represent the types of data themselves.
  - Associations, which show how instances of classes reference instances of other classes.
  - Attributes, which are simple data found in instances.
  - Operations, which represent the functions performed by the instances.
  - Generalizations, which are used to arrange classes into inheritance hierarchies.

Classes

- A class is represented as a box with the name of the class inside.
- The class name should always be singular and start with a capital letter.
- When you draw a class in a class diagram, you are saying that the system will contain a class by that name, and that when the system runs, instances of that class will be created.

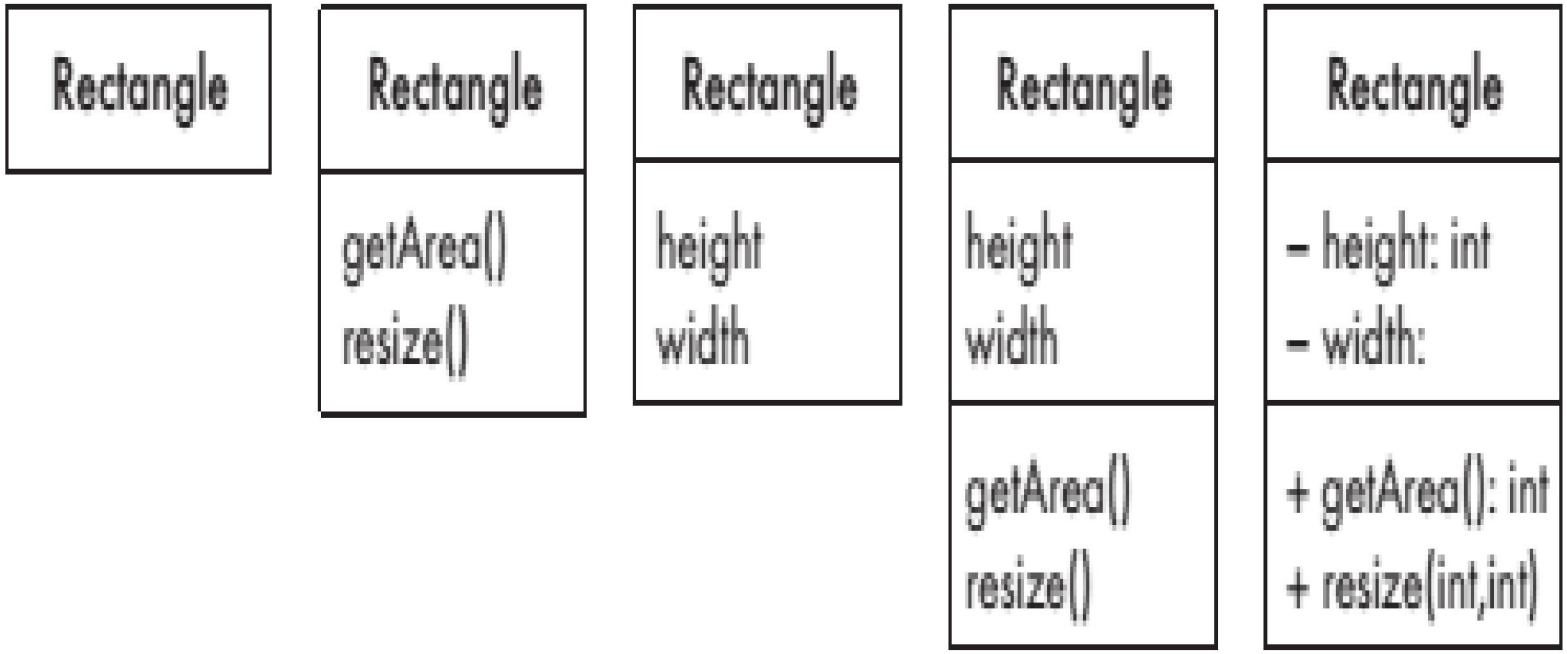
Classes

- Optionally, the class diagram may also show the attributes and operations contained in each class.
- This is done by dividing a class box into two or three smaller boxes:
  - Top box contains the class name.
  - Next box lists attributes.
  - Bottom box lists operations.

Classes

- If you do not want to specify attributes or operations, then you simply omit the box.
- How much detail you show depends on the phase of development and on what you wish to communicate.

Class drawn at several different levels of detail



Class drawn at several different levels of detail

- In the leftmost example, only the class name is shown, indicating merely that the class exists.
- Additional detail is shown in the other four representations of the same Rectangle class.

Class drawn at several different levels of detail

- The most detail, including the type of attributes, whether the feature is public (+) or private (–), and the signature of operations, is shown in the rightmost example.

Class drawn at several different levels of detail

- When shown in full detail, an operation's signature is specified using the following notation: *operationName(parameterName: parameterType,...): returnType*.

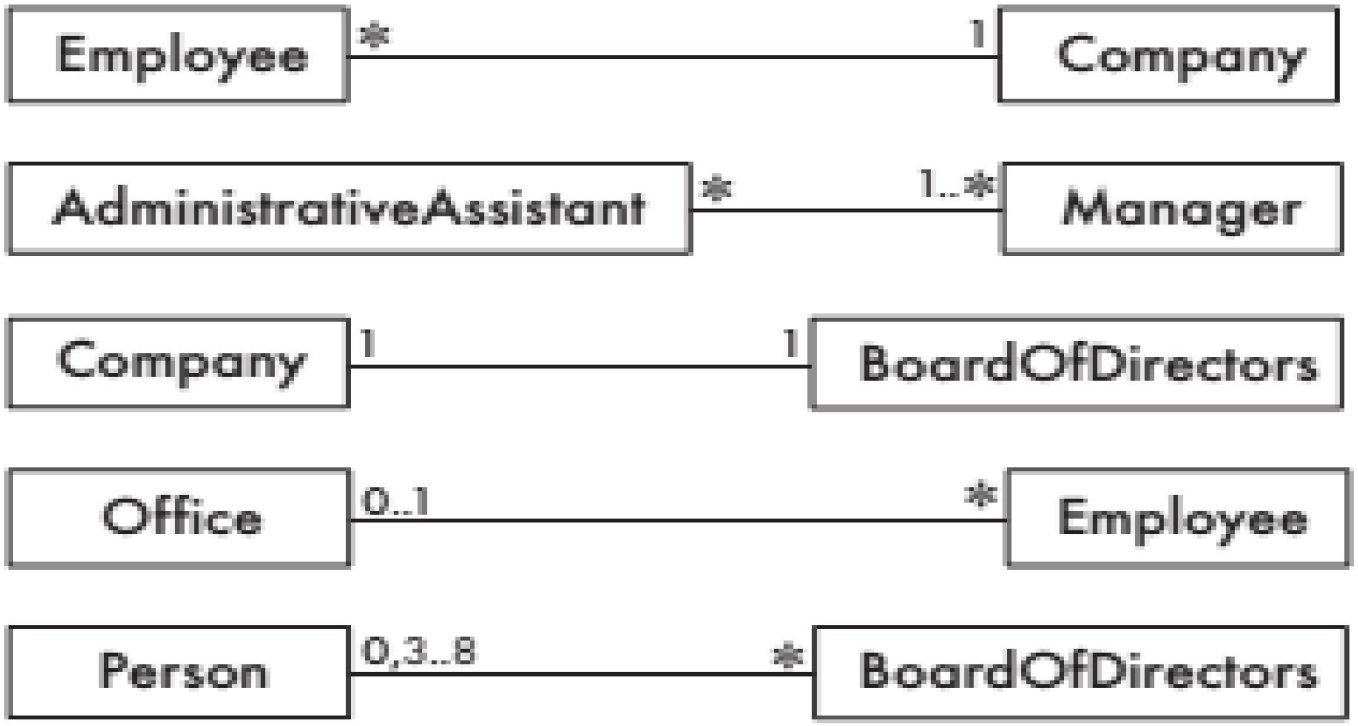
Associations and multiplicity

- An association is used to show how instances of two classes will reference each other.
- The association is drawn as a line between the classes.
- Symbols indicating multiplicity are shown at each end of the association.

Associations and multiplicity

- The multiplicity indicates how many instances of the class at this end of the association can be linked to an instance of the class at the other end of the association.

Examples of possible multiplicities



Associations and multiplicity

- A multiplicity of 1 indicates that there must be exactly one instance linked to each object at the other end of the association e.g. there can only be one Company associated with each Employee.

Associations and multiplicity

- A very common multiplicity is *, which is normally read as 'many', and means any integer greater than or equal to zero.
- For example, many employees can be associated with a company; one possibility being that a company has no employees.

Associations and multiplicity

- Although there is no theoretical upper bound, there is a practical upper bound that depends on the amount of memory and processing capacity available.

Associations and multiplicity

- If there can be either zero or one object linked to an object at the other end of the association, then the multiplicity is said to be 'optional', and the notation 0..1 is used.

Associations and multiplicity

- For example, there can be zero or one office per employee.
- In other words, it is optional that an employee is assigned to an office (some may work at home or in a job that does not require an office).

Associations and multiplicity

- You can also specify the multiplicity to be an interval, which is shown as two dots between the lower and upper bound.
- An interval is also sometimes called a range.

Associations and multiplicity

- The 0..1 notation is a special case of an interval.
- If an interval has no upper bound, then you use the asterisk; therefore 0..* and * mean the same thing, while 1..* means 'at least one'.

Associations and multiplicity

- The multiplicity can be a specific positive integer; and you can also specify several multiplicity values or ranges separated by commas.
- For example, imagine that a law in some jurisdiction states that a board of directors must have between three and eight members.

Associations and multiplicity

- Furthermore, if the board finds itself with insufficient members, then it is automatically dissolved and new elections must be held during the election process, the board has zero members.

Associations and multiplicity

- The multiplicity of the final example reflects this situation: there can be either zero, or between 3 and 8 persons on a BoardOfDirectors.

Associations and multiplicity

- Specific multiplicities involving intervals or exact numbers greater than two are not common, and should only be specified after careful thought.
- For example, you might be tempted to specify that a person should always have exactly two parents.

Associations and multiplicity

- However, if you do so, then you are requiring that the system always have a record of everybody's two parents.
- Adhering strictly to such a rule would be impossible because not everybody knows who their parents are, and also because the system would have to know the parents of the parents, ad infinitum.

Associations and multiplicity

- A more reasonable multiplicity for parents might be 0..2.
- If you do not specify the multiplicity of an association end, then it is said to be undefined.
- It is strongly recommend never leaving a multiplicity undefined, since much of the meaning of a class diagram comes from the multiplicities.

Labeling associations

- Each association can be labeled, to make explicit the nature of the association.
- There are two types of labels:
  - Association names and
  - Role names.

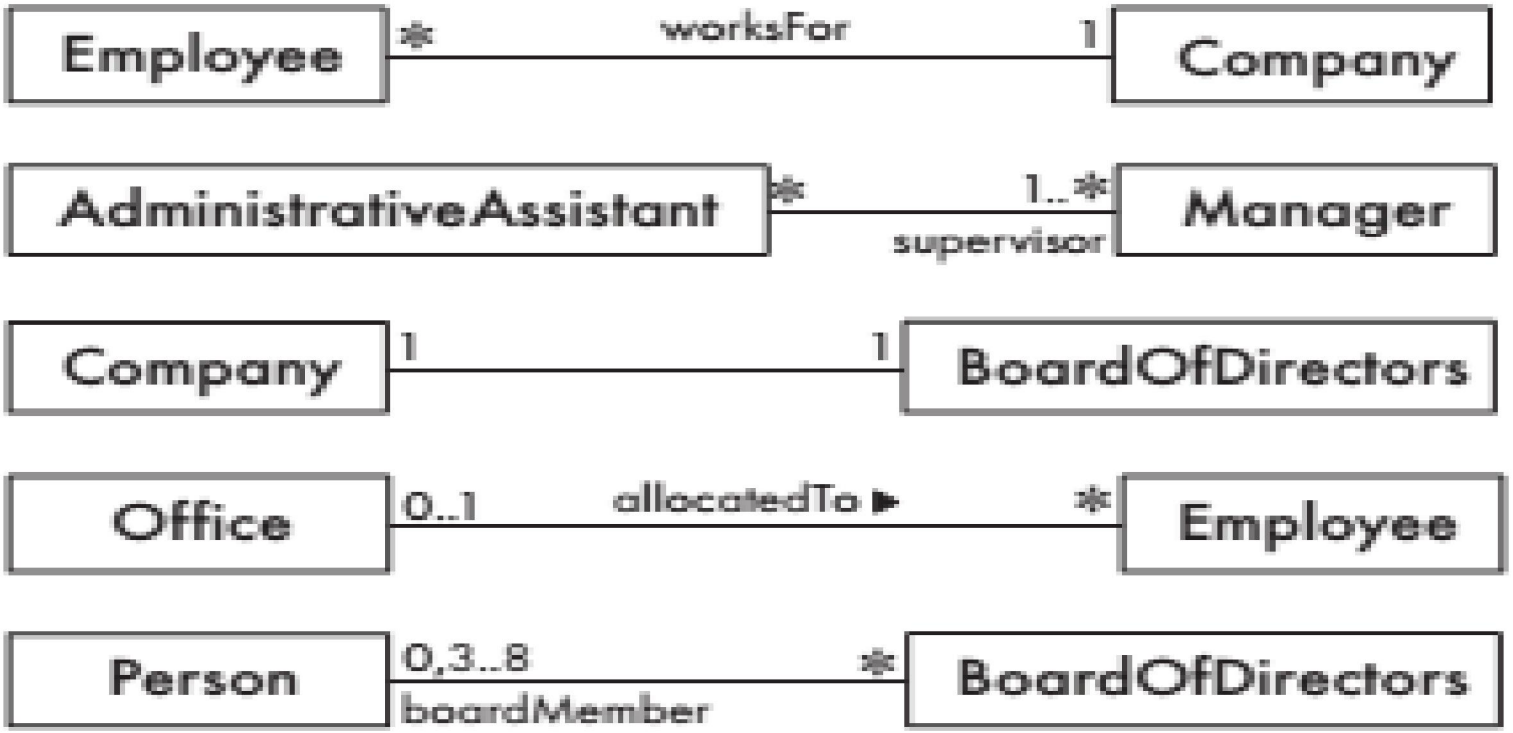
Labeling associations

- An association name should be a verb or verb phrase, and is placed next to the middle of the association.
- One class becomes the subject and the other class becomes the object of the verb.

Labeling associations

- The direction to read the association is normally obvious, but can be clarified by showing a little arrow (a filled triangle) next to the association name.

Labeling associations



Labeling associations

- Another way of labeling an association is to use a role name.
- Role names can be attached to either or both ends of an association.
- A role name acts, in the context of the association, as an alternative name for the class to which it is attached.

Labeling associations

- For example, in the association between Person and BoardOfDirectors, boardMember is a role name that describes the people who happen to be members of the board.
- You can read this association as, ‘a board of directors has either zero or 3 to 8 persons as board members’.

Labeling associations

- If you omit both the association name and role names, then consider that an association's name is simply *has*, by default.
- This is not very informative, but in some cases it is adequate, since the meaning of the association might be clear by simply looking at the two classes.

Labeling associations

- A good rule of thumb when performing analysis is: add sufficient names to make the association clear and unambiguous.
- It is normally not necessary to add both role names and an association name to the same association.

Analyzing and validating associations

- It is very common to make errors when creating associations i.e. it is particularly easy to get the multiplicity wrong.
- Therefore you should get into the habit of reading every association in both directions to verify that it makes sense.

Analyzing and validating associations

- Most importantly, you should always ask yourself whether a less restrictive multiplicity could also makes sense in some circumstances i.e. using ‘many’ or ‘optional’ instead of ‘one’ or some other specific number.

Three of the most common patterns of multiplicity

- One-to-many.
- Many-to-many.
- One-to-one.

One-to-one

- For each company, there is exactly one board of directors.
- Also, a board is the board of only one company.
- A company must always have a board, and a board must always be of some company.

One-to-one

- What would happen if the board members all resigned? We would still say that the board exists, but temporarily has zero members.
- One-to-one associations are less common. When you see such an association, you should ask yourself if in fact one or both ends should be changed to ‘optional’ or ‘many’.

One-to-one

- The implication of a one-to-one association is that whenever you create an instance of one of the classes, you must simultaneously create an instance of the other; and when you delete one you must delete the other.

One-to-many

- A company has many employees, but an employee can only work for one company.
- You might argue that this is incorrect, since somebody might moonlight, working for several companies.
- However, company policy might explicitly disallow moonlighting in companies managed by our system.

One-to-many

- This multiplicity pattern correctly indicates that a company can have zero employees, as in the case of a ‘shell’ company.
- Finally, since it is not possible to be an employee unless you work for a company, the multiplicity at the Company end is correctly shown to be exactly one, not optional.

Many-to-many

- An administrative assistant can work for many managers, and a manager can have many administrative assistants.
- A one-to-one relationship would be typical between any particular administrative assistant and manager, but in general there are assistants who work for a group of managers, and managers who are so senior that they have a group of assistants.

Many-to-many

- It is also the case that some managers might have zero assistants.
- Any time you see a many-to-many association, you should consider whether an association class is needed.

Many-to-many

- The most common multiplicity pattern is one-to-many.
- The next most common is many-to-many.
- Together, these two patterns account for the vast majority of associations.

Association classes

- In some circumstances, an attribute that concerns two associated classes cannot be placed in either of the classes.
- For example, imagine the association in which a student can register in any number of course sections, and a course section can have any number of students.
- In which class should the student's grade be put?

Association Classes

Analyzing and validating associations

- If you put the grade in the Student class, then a student could have only one grade, not one per course section.
- If you put the grade in the CourseSection class, then a course section could have only one grade, not one per student.
- The grade is therefore not a property of either class.

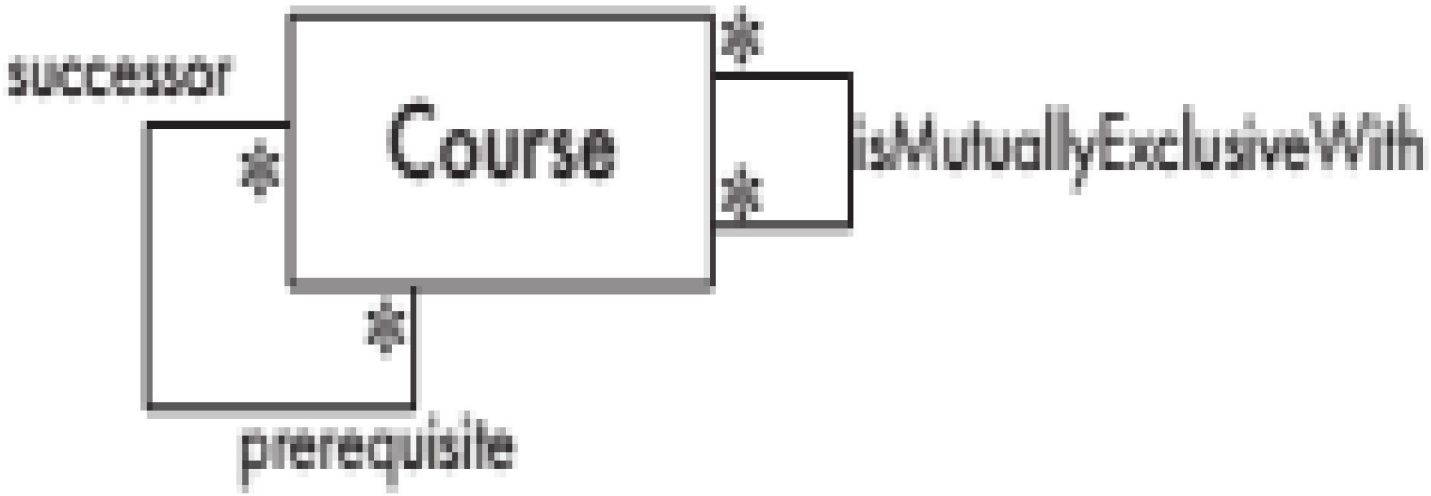
Reflexive associations

- It is possible for an association to connect a class to itself.
- E.g. A course can require other prerequisite courses to be taken first.

Reflexive associations

- If two courses cover nearly the same material, taking one of them may preclude a student from taking the other, and vice versa – such courses are said to be mutually exclusive.

Reflexive associations



Reflexive associations

- The first association is asymmetric, since the roles of the classes at each end are clearly different.
- The second, on the other hand, is symmetric.

Reflexive associations

- To make the meaning clear, you should label an asymmetric reflexive association using role names instead of an association name.

Links as instances of associations

- In the same way that we say an object is an instance of a class, we say that a link is an instance of an association.
- Each link connects two objects – an instance of each of the two classes involved in the association.

Directionality in associations

- Associations and links are by default bi-directional i.e. if a Driver object is linked to a Car object, then the Car is also implicitly linked to that Driver.
- If you know the car, you can find out its driver – or if you know the driver, you can find out the car.

Directionality in associations

- It is possible to limit the navigability of an association's links by adding an arrow at one end.
- Decisions about directionality should normally be deferred to later phases of development, when the detailed design is created.

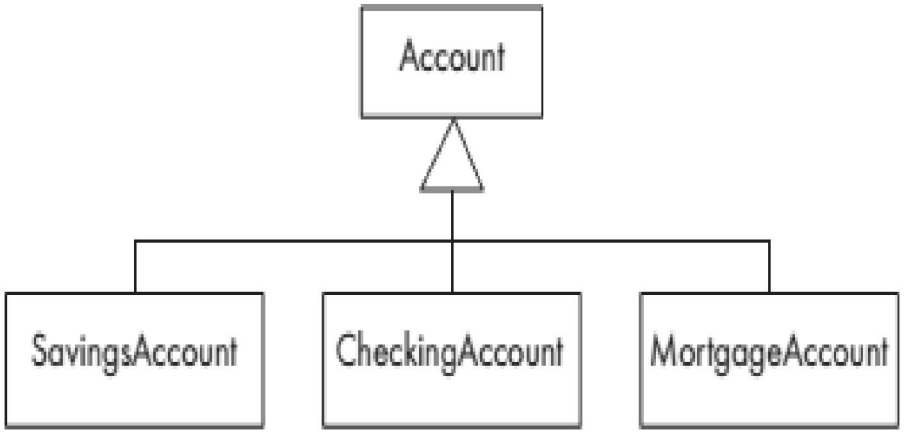
Directionality in associations

- Making associations unidirectional can improve efficiency and reduce complexity, but might also limit the flexibility of the system.

Generalization and Specialization

- The relationship between a subclass and an immediate super class is called a generalization.
- The subclass is called a specialization.
- A hierarchy with one or more generalizations is called an inheritance hierarchy, a generalization hierarchy or an *isa* hierarchy.

Generalization and Specialization



Basic inheritance hierarchy of bank accounts

Avoiding unnecessary generalizations

- A common mistake made by beginners is to overdo generalization.
- However, to justify the existence of each class, there must be some operation that will be done differently in that class.

Handling multiple generalization sets

- A generalization set is a labeled group of generalizations with a common super class; the label describes the criteria used to specialize the super class into two or more subclasses.

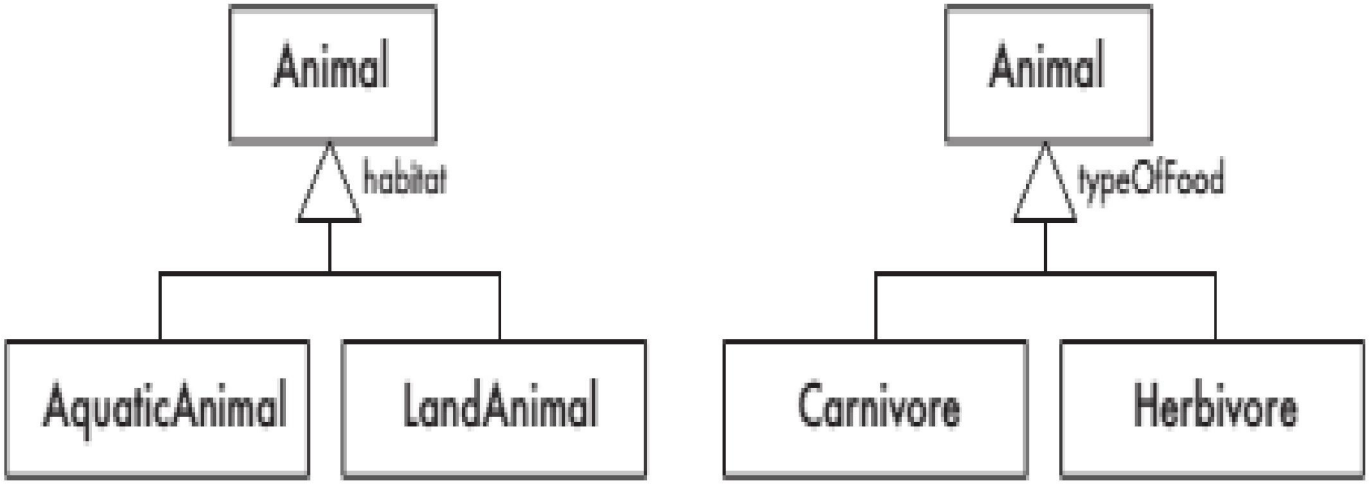
Handling multiple generalization sets

- It is clearest to unite all the generalizations in a set using a single open triangle.
- You place the label next to the open triangle.
- The label of a generalization set will typically be an attribute that has a different value in each subclass.

Handling multiple generalization sets

- Animals can be divided up by habitat into aquatic and land animals, or by type of food, into carnivores and herbivores.

Handling multiple generalization sets



Avoiding having objects change class

- Another issue that can arise when creating generalizations is avoiding the need for objects to change class.
- In general, an object should never need to change class.

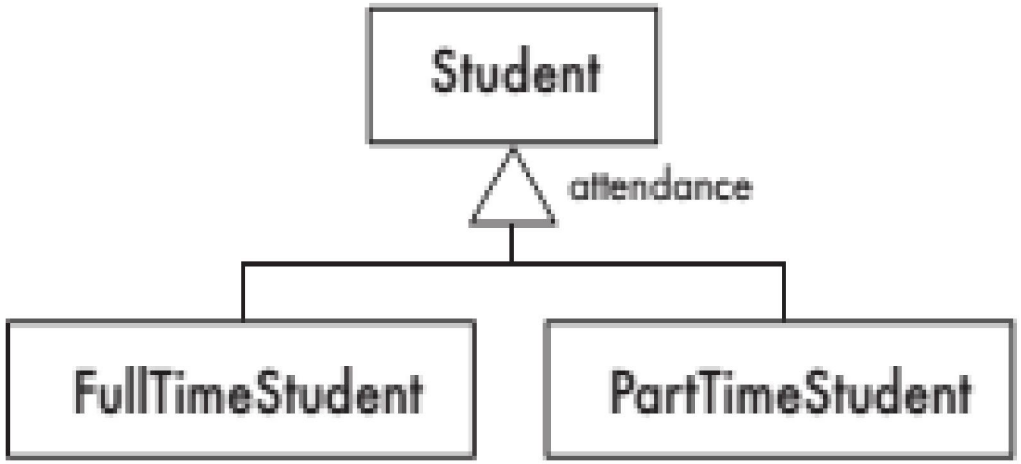
Avoiding having objects change class

- In most programming languages, changing class is simply not possible; therefore you have to completely destroy the original object and create a new instance of the second class.

Avoiding having objects change class

- This is complex and error prone because you have to copy all the instance variables and make sure that all links that connected to the old object now connect to the new one.

Avoiding having objects change class



Avoiding having objects change class

- It is clear that during his or her studies, the attendance status of a student can change from full-time to part-time and vice versa.
- You do not want to model this situation in your system by destroying a PartTimeStudent and creating a FullTimeStudent, or the opposite, each time the student's status changes.

Avoiding having objects change class

- For this reason, it is a poor model.
- A possible solution is simply to make attendance Status an attribute of Student and to omit the two subclasses completely.
- The problem with this is that we lose the advantage of polymorphism for any operations that would differ in PartTimeStudent and FullTimeStudent.

End of chapter 5.2