9.2 Superclasses and Subclasses

Often, an object of one class *is an* object of another class as well. For example, a `CarLoan` *is a* `Loan` as are `HomeImprovementLoan`s and `MortgageLoan`s. Thus, in Java, class `CarLoan` can be said to inherit from class `Loan`. In this context, class `Loan` is a superclass and class `CarLoan` is a subclass. A `CarLoan` *is a* specific type of `Loan`, but it's incorrect to claim that every `Loan` *is a* `CarLoan`—the `Loan` could be any type of loan. Figure 9.1 lists several simple examples of superclasses and subclasses—superclasses tend to be "more general" and subclasses "more specific."

Fig. 9.1

Inheritance examples.

Superclass	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`, `Sphere`, `Cube`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`

`BankAccount`	`CheckingAccount`, `SavingsAccount`

Because every subclass object *is an* object of its superclass, and one superclass can have many subclasses, the set of objects represented by a superclass is often larger than the set of objects represented by any of its subclasses. For example, the superclass `Vehicle` represents *all* vehicles, including cars, trucks, boats, bicycles and so on. By contrast, subclass `Car` represents a smaller, more specific subset of vehicles.

University Community Member Hierarchy

Inheritance relationships form treelike *hierarchical* structures. A superclass exists in a hierarchical relationship with its subclasses. Let's develop a sample class hierarchy (Fig. 9.2), also called an **inheritance hierarchy**. A university community has thousands of members, including employees, students and alumni. Employees are either faculty or staff members. Faculty members are either administrators (e.g., deans and department chairpersons) or teachers. The hierarchy could contain many other classes. For example, students can be graduate or undergraduate students. Undergraduate students can be freshmen, sophomores, juniors or seniors.

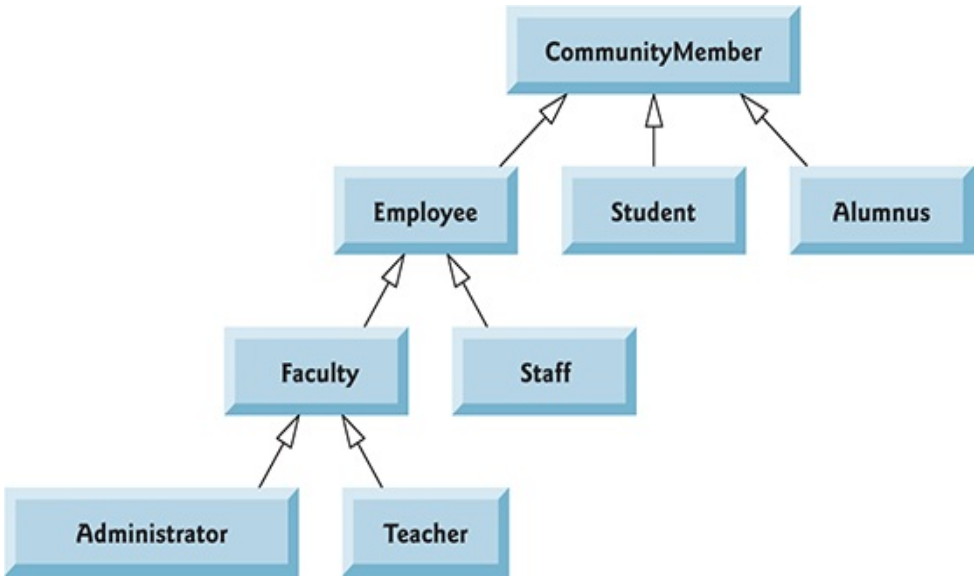

Fig. 9.2

Inheritance hierarchy UML class diagram for university `CommunityMember`s.

Each arrow in the hierarchy represents an *is-a* relationship. As we follow the arrows upward in this class hierarchy, we can state, for example, that "an `Employee` *is a* `CommunityMember`" and "a `Teacher` *is a* `Faculty` member." `CommunityMember` is the direct superclass of `Employee`, `Student` and `Alumnus` and is an indirect superclass of all the other classes in the diagram. Starting from the bottom, you can follow the arrows and apply the *is-a* relationship up to the topmost superclass. For example, an `Administrator` *is a* `Faculty` member, *is an* `Employee`, *is a* `CommunityMember` and, of course, *is an* `Object`.

Shape Hierarchy

Now consider the `Shape` inheritance hierarchy in Fig. 9.3. This hierarchy begins with superclass `Shape`, which is extended by subclasses `TwoDimensionalShape` and `ThreeDimensionalShape`—`Shape`s are either `TwoDimensionalShape`s or `ThreeDimensionalShape`s. The third level of this hierarchy contains *specific* types of `TwoDimensionalShape`s and `ThreeDimensionalShape`s. As in Fig. 9.2, we can follow the arrows from the bottom of the diagram to the topmost superclass in this class hierarchy to identify several *is-a* relationships. For example, a `Triangle` *is a* `TwoDimensionalShape` and *is a* `Shape`, while a `Sphere` *is a* `ThreeDimensionalShape` and *is a* `Shape`. This hierarchy could contain many other classes. For example, ellipses and trapezoids also are `TwoDimensionalShape`s.

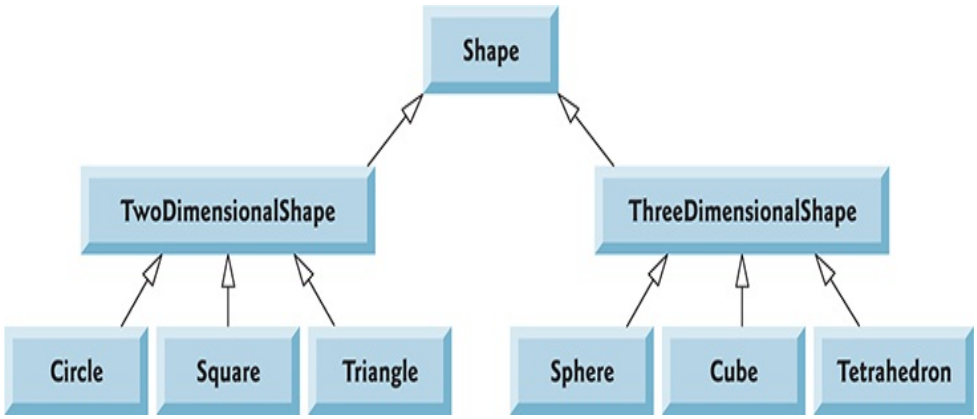

Fig. 9.3

Inheritance hierarchy UML class diagram for `Shape`s.

Not every class relationship is an inheritance relationship. In Chapter 8, we discussed the *has-a* relationship, in which classes have members that are references to objects of other classes. Such relationships create classes by *composition* of existing classes. For example, given the classes `Employee`, `BirthDate` and `TelephoneNumber`, it's improper to say that an `Employee` *is a* `BirthDate` or that an `Employee` *is a* `TelephoneNumber`. However, an `Employee` *has a* `BirthDate`, and an `Employee` *has a* `TelephoneNumber`.

It's possible to treat superclass objects and subclass objects similarly—their commonalities are expressed in the superclass's members. Objects of all classes that extend a common superclass can be treated as objects of that superclass—such objects have an *is-a* relationship with the superclass. Later in this chapter and in Chapter 10, we consider many examples that take advantage of the *is-a* relationship.

A subclass can customize methods that it inherits from its superclass. To do this, the subclass **overrides** (*redefines*) the superclass method with an appropriate implementation, as we'll see in the chapter's code examples.