10.10 Summary

This chapter has presented a first view of inheritance. All classes in Java are arranged in an inheritance hierarchy. Each class may have an explicitly declared superclass, or it inherits implicitly from the class `Object`.

Subclasses usually represent specializations of superclasses. Because of this, the inheritance relationship is also referred to as an is-a relationship (a car *is-a* vehicle).

Subclasses inherit all fields and methods of a superclass. Objects of subclasses have all fields and methods declared in their own classes, as well as those from all superclasses. Inheritance relationships can be used to avoid code duplication, to reuse existing code, and to make an application more maintainable and extendable.

Subclasses also form subtypes, which leads to polymorphic variables. Subtype objects may be substituted for supertype objects, and variables are allowed to hold objects that are instances of subtypes of their declared type.

Inheritance allows the design of class structures that are easier to maintain and more flexible. This chapter contains only an introduction to the use of inheritance for the purpose of improving program structures. More uses of inheritance and their benefits will be discussed in the following chapters.

Terms introduced in this chapter:

inheritance, superclass (parent), subclass (child), *is-a*, inheritance hierarchy, abstract class, subtype substitution, polymorphic variable, type loss, cast

Concept summary

- **inheritance** Inheritance allows us to define one class as an extension of another.
- **superclass** A superclass is a class that is extended by another class.
- **subclass** A subclass is a class that extends (inherits from) another class. It inherits all fields and methods from its superclass.
- **inheritance hierarchy** Classes that are linked through inheritance relationships form an inheritance hierarchy.
- **superclass constructor** The constructor of a subclass must always invoke the constructor of its superclass as its first statement. If the source code does not include such a call, Java will attempt to insert a call automatically.
- **reuse** Inheritance allows us to reuse previously written classes in a new context.
- **subtype** As an analog to the class hierarchy, types form a type hierarchy. The type defined by a subclass definition is a subtype of the type of its superclass.
- **variables and subtypes** Variables may hold objects of their declared type or of any subtype of their declared type.

- **substitution** Subtype objects may be used wherever objects of a supertype are expected. This is known as substitution.
- **object** All classes with no explicit superclass have `Object` as their superclass.

Exercise 10.16 Go back to the *lab-classes* project from Chapter 1. Add instructors to the project (every lab class can have many students and a single instructor). Use inheritance to avoid code duplication between students and instructors (both have a name, contact details, etc.).

Exercise 10.17 Draw an inheritance hierarchy representing parts of a computer system (processor, memory, disk drive, DVD drive, printer, scanner, keyboard, mouse, etc.).

Exercise 10.18 Look at the code below. You have four classes (`O`, `X`, `T`, and `M`) and a variable of each of these.

```
O o;
X x;
T t;
M m;
```

The following assignments are all legal (assume that they all compile):

```
m = t;
m = x;
o = t;
```

The following assignments are all illegal (they cause compiler errors):

```
o = m;
o = x;
x = o;
```

What can you say about the relationships of these classes? Draw a class diagram.

Exercise 10.19 Draw an inheritance hierarchy of `AbstractList` and all its (direct and indirect) subclasses as they are defined in the Java standard library.

This page intentionally left blank