10.15 Wrap-Up

This chapter introduced polymorphism—the ability to process objects that share the same superclass in a class hierarchy as if they were all objects of the superclass. We discussed how polymorphism makes systems extensible and maintainable, then demonstrated how to use overridden methods to effect polymorphic behavior. We introduced abstract classes, which allow you to provide an appropriate superclass from which other classes can inherit. You learned that an abstract class can declare abstract methods that each subclass must implement to become a concrete class and that a program can use variables of an abstract class to invoke the subclasses' implementations of abstract methods polymorphically. You also learned how to determine an object's type at execution time. We explained the notions of `final` methods and classes. The chapter discussed declaring and implementing an interface as a way for possibly disparate classes to implement common functionality, enabling objects of those classes to be processed polymorphically.

We introduced the interface enhancements in Java SE 8—`default` methods and `static` methods—and in Java SE 9—`private` methods. Next, we discussed the purpose of `private` constructors. Finally, we discussed programming to an interface vs. programming to an implementation, and how the `Employee` hierarchy can be reimplemented using a `CompensationModel` interface.

You should now be familiar with classes, objects, encapsulation, inheritance, interfaces and polymorphism—the most essential aspects of object-oriented programming.

In the next chapter, you'll learn about exceptions, useful for handling errors during a program's execution. Exception handling provides for more robust programs.