6.5 Notes on Declaring and Using Methods

Calling Methods

There are three ways to call a method:

1. Using a method name by itself to call another method of the *same* class—such as `maximum(number1, number2, number3)` in line 18 of Fig. 6.3.
2. Using a variable that contains a reference to an object, followed by a dot (`.`) and the method name to call a non-`static` method of the referenced object—such as the method call in line 14 of Fig. 3.2, `myAccount.getName()`, which calls a method of class `Account` from the `main` method of `AccountTest`. Non-`static` methods are typically called **instance methods**.
3. Using the class name and a dot (`.`) to call a `static` method of a class—such as `Math.sqrt(900.0)` in Section 6.3.

Returning from Methods

There are three ways to return control to the statement that calls a method:

- When the method-ending right brace is reached in a method with return type `void`.

- When the following statement executes in a method with return type `void`

  ```
  return;
  ```

- When a method returns a result with a statement of the following form in which the *expression* is evaluated and its result (and control) are returned to the caller:

```
return expression;
```

Common Programming Error 6.4

Declaring a method outside the body of a class declaration or inside the body of another method is a syntax error.

Common Programming Error 6.5

Declaring a local variable in a method with the same name as one of the method’s parameters is a compilation error.

Common Programming Error 6.6

Forgetting to return a value from a method that should return a value is a compilation error. If a return type other than `void` *is specified, the method* must *contain a* `return` *statement that returns a value consistent with the method's return type. Returning a value from a method whose return type has been declared* `void` *is a compilation error.*

`static` Members Can Access Only the Class's Other `static` Members Directly

A `static` method can call directly (that is, using the method name by itself) *only* other `static` methods of the same class and can manipulate directly *only* `static` variables in the same class. To access a class's instance variables and instance methods (that is, its non-`static` members), a `static` method must use a reference to an object of that class. Instance methods can access all fields (`static` variables and instance variables) and methods of the class.

Many objects of a class, each with its own copies of the instance variables, may exist at the same time. Suppose a `static` method were to invoke a non-`static` method directly. How would the method know which object's instance variables to manipulate? What would happen if no objects of the class existed at the time the non-`static` method was invoked?