 Answers to review questions from Chapter 6

1. Define each of the following terms: *object, structure, class, instance variable, method*.

**An *object* is a data value that combines representation and behavior.**

**A *structure* is a data value consisting of heterogeneous components.**

**A *class* is a template for creating objects that share a common structure.**

**An *instance variable* is a variable within a class and therefore belongs to each object of that class.**

**A *method* is a function defined as part of a class and therefore available to each object.**

2. In a C++ class definition, what do the keywords **public** and **private** mean?

**The keyword public indicates that the method or variable is available to all clients of that class. The keyword private limits access to the class itself.**

3. True or false: In C++, the only difference between the keyword **struct** and the keyword **class** is that **struct** makes fields public by default.

**True.**

4. What operator does C++ use to select an instance variable from an object?

**The dot (.) operator.**

5. What is the syntax for a C++ constructor?

**A constructor looks just like a method definition except that it lacks a return type. The name of the constructor is always the name of the class.**

6. How many arguments are passed to the *default constructor?*

**None.**

7. What are *getters* and *setters?*

**A *getter* is a method that returns the value of an instance variable, which is typically declared as private. A *setter* is a method that changes the value of an instance variable.**

8. What does it mean for a class to be *immutable?*

**An immutable class is one in which the internal state cannot be changed after creating an object.**

9. When you separate the interface and implementation of a class, how does the implementation let the compiler know to which class a particular method definition belongs?

**Methods that are part of a class are tagged with the class name followed by a double colon (::).**

10. What strategy is used in the **.h** files in this chapter to prevent clients from seeing the contents of the private section?

**The comments before the private section note that the details are interesting only to the implementation. In the figures presented in the book, the private section is replaced by a box.**

11. In C++, what method name would you use to overload the **%** operator?

**operator%**

12. How does C++ differentiate between the prefix and suffix versions of the **++** and **--** operators?

**The method used for the suffix forms takes a dummy argument of type int.**

13. Why does the overloaded implementation of the **<<** operator require the use of return by reference?

**It is illegal to copy streams in C++. Returning the value by reference avoids making a copy.**

14. True or false: Return by reference is used as frequently in C++ programs as call by reference.

**False.**

15. Describe the differences between the method‑based and free‑function‑based approaches to overloading the operators for a class. What are the advantages and disadvantages of each style?

**Operators defined as methods use the current object as their first argument and are defined within the class. Operators defined as free‑functions specify both arguments. The free‑function approach treats the arguments symmetrically, which is more intuitive with respect to the operator form. Free functions, however, have no access to the private instance variables of the class and must therefore be declared as friends.**

16. What does it mean for one class to declare a method or another class as a friend?

**Declaring a method or class as a friend gives it access to the private variables and methods of the befriending class.**

17. What reason does this chapter offer for overloading the **++** operator for the **Direction** type?

**Overloading the ++ operator makes it possible to use the standard for loop idiom to iterate through the four direction values: North, East, South, and West.**

18. What are the five steps suggested in this chapter as guidelines for designing a class?

**1. Think generally about how clients are likely to use the class.**

**2. Determine what information belongs in the private state of each object.**

**3. Define a set of constructors to create new objects.**

**4. Enumerate the operations that will become the public methods of the class.**

**5. Code and test the implementation.**

19. What is a *rational number?*

**A rational number is the quotient of two integers.**

20. What restrictions does the **Rational** constructor place on the values of the **num** and **den** variables?

**The denominator must be positive, and the rational number must be reduced to lowest terms.**

21. The code for the **Rational** constructor on page <ref>290 includes an explicit check to see whether **x** is zero. Would the **Rational** class still work the same way if this check were eliminated?

**Yes.**

22. In the **rational.h** file in Figure 6‑7, why is it necessary to designate the operator methods for **+**, **-**, **\***, and **/** as friends but not the operator method for the **<<** insertion operator?

**The << operator requires no access to the private variable of the class.**

23. What is a *token?*

**A *token* is a string of characters from an input source that makes sense as a unit.**

24. What is the standard pattern for reading all tokens from a string?

TokenScanner scanner(str);

while (scanner.hasMoreTokens()) {

string token = scanner.nextToken();

}

25. How do you initialize a **TokenScanner** object so that it ignores spaces, tabs, and other whitespace characters in the input?

**Call scanner.ignoreWhitespace().**

26. In your own words, explain the technique of embedding a program in a class.

**In this technique, all code is part of a class with the exception of the main method, which has no role except to create an instance of the class and then call a method within it that initiates the computation.**