of the superclass `DieView` to actually draw the pips. Notice especially how the call to the method from the superclass is made. The normal approach `self.setValue(value)` would refer to the `setValue` method of the `ColorDieView` class, since `self` is an instance of `ColorDieView`. In order to call the original `setValue` method from the superclass, it is necessary to put the class name where the object would normally go.

```
DieView.setValue(self, value)
```

The actual object to which the method is applied is then sent as the first parameter.

12.5 Chapter Summary

This chapter has not introduced very much in the way of new technical content. Rather it has illustrated the process of object-oriented design through the racquetball simulation and dice poker case studies. The key ideas of OOD are summarized here:

- Object-oriented design (OOD) is the process of developing a set of classes to solve a problem. It is similar to top-down design in that the goal is to develop a set of black boxes and associated interfaces. Where top-down design looks for functions, OOD looks for objects.
- There are many different ways to do OOD. The best way to learn is by doing it. Some intuitive guidelines can help:
    1. Look for object candidates.
    2. Identify instance variables.
    3. Think about interfaces.
    4. Refine nontrivial methods.
    5. Design iteratively.
    6. Try out alternatives.
    7. Keep it simple.
- In developing programs with sophisticated user interfaces, it's useful to separate the program into model and view components. One advantage of this approach is that it allows the program to sport multiple looks (e.g., text and GUI interfaces).

- There are three fundamental principles that make software object oriented:

  Encapsulation Separating the implementation details of an object from how the object is used. This allows for modular design of complex programs.

  Polymorphism Different classes may implement methods with the same signature. This makes programs more flexible, allowing a single line of code to call different methods in different situations.

  Inheritance A new class can be derived from an existing class. This supports sharing of methods among classes and code reuse.

12.6 Exercises

Review Questions

True/False

1. Object-oriented design is the process of finding and defining a useful set of functions for solving a problem.
2. Candidate objects can be found by looking at the verbs in a problem description.
3. Typically, the design process involves considerable trial and error.
4. GUIs are often built with a model-view architecture.
5. Hiding the details of an object in a class definition is called instantiation.
6. Polymorphism literally means "many changes."
7. A superclass inherits behaviors from its subclasses.
8. GUIs are generally easier to write than text-based interfaces.

Multiple Choice

1. Which of the following was not a class in the racquetball simulation?
   a) `Player` b) `SimStats` c) `RBallGame` d) `Score`

2. What is the data type of `server` in an `RBallGame`?
   a) int b) `Player` c) bool d) `SimStats`

3. The `isOver` method is defined in which class?
   a) `SimStats` b) `RBallGame` c) `Player` d) `PokerApp`

4. Which of the following is not one of the fundamental characteristics of object-oriented design/programming?
   a) inheritance b) polymorphism
   c) generality d) encapsulation

5. Separating the user interface from the "guts" of an application is called a(n) ___ approach.
   a) abstract b) object-oriented
   c) model-theoretic d) model-view

Discussion

1. In your own words, describe the process of OOD.

2. In your own words, define *encapsulation*, *polymorphism*, and *inheritance*.

Programming Exercises

1. Modify the Dice Poker program from this chapter to include any or all of the following features:

   a) Splash Screen. When the program first fires up, have it print a short introductory message about the program and buttons for "Let's Play" and "Exit." The main interface shouldn't appear unless the user selects "Let's Play."

   b) Add a "Help" button that pops up another window displaying the rules of the game (the payoffs table is the most important part).

   c) Add a high score feature. The program should keep track of the 10 best scores. When a user quits with a good enough score, he/she is invited to type in a name for the list. The list should be printed in the splash screen when the program first runs. The high-scores list will have to be stored in a file so that it persists between program invocations.

2. Using the ideas from this chapter, implement a simulation of another racquet game. See the programming exercises from Chapter 9 for some ideas.

3. Write a program to keep track of conference attendees. For each attendee, your program should keep track of name, company, state, and email address. Your program should allow users to do things such as add a new attendee, display information on an attendee, delete an attendee, list the names and email addresses of all attendees, and list the names and email addresses of all attendees from a given state. The attendee list should be stored in a file and loaded when the program starts.

4. Write a program that simulates an automatic teller machine (ATM). Since you probably don't have access to a card reader, have the initial screen ask for user ID and a PIN. The user ID will be used to look up the information for the user's accounts (including the PIN to see whether it matches what the user types). Each user will have access to a checking account and a savings account. The user should able to check balances, withdraw cash, and transfer money between accounts. Design your interface to be similar to what you see on your local ATM. The user account information should be stored in a file when the program terminates. This file is read in again when the program restarts.

5. Find the rules to an interesting dice game and write an interactive program to play it. Some examples are craps, yacht, greed, and skunk.

6. Write a program that deals four bridge hands, counts how many points they have, and gives opening bids. You will probably need to consult a beginner's guide to bridge to help you out.

7. Find a simple card game that you like and implement an interactive program to play that game. Some possibilities are war, blackjack, various solitaire games, and crazy eights.

8. Write an interactive program for a board game. Some examples are Othello (reversi), Connect Four, Battleship, Sorry!, and Parcheesi.

9. (Advanced) Look up a classic video game such as Asteroids, Frogger, Breakout, Tetris, etc. and create your own version using the animation techniques from Chapter 11.