Objects from related classes usually share common behavior. For example, shovels, rakes, and clippers all perform gardening tasks. In this chapter, you will learn how the notion of inheritance expresses the relationship between specialized and general classes. By using inheritance, you will be able to share code between classes and provide services that can be used by multiple classes.

9.1 Inheritance Hierarchies

A subclass inherits data and behavior from a superclass.

In object-oriented design, **inheritance** is a relationship between a more general class (called the **superclass**) and a more specialized class (called the **subclass**). The subclass inherits data and behavior from the superclass. For example, consider the relationships between different kinds of vehicles depicted in Figure 1.

Every car *is a* vehicle. Cars share the common traits of all vehicles, such as the ability to transport people from one place to another. We say that the class `Car` inherits from the class `Vehicle`. In this relationship, the `Vehicle` class is the superclass and the `Car` class is the subclass. In Figure 2, the superclass and subclass are joined with an arrow that points to the superclass.

You can always use a subclass object in place of a superclass object.

Suppose we have an algorithm that manipulates a `Vehicle` object. Because a car is a special kind of vehicle, we can use a `Car` object in such an algorithm, and it will work correctly. The **substitution principle** states that you can always use a subclass object when a superclass object is expected. For example, consider a method that takes an argument of type `Vehicle`:

```
void processVehicle(Vehicle v)
```

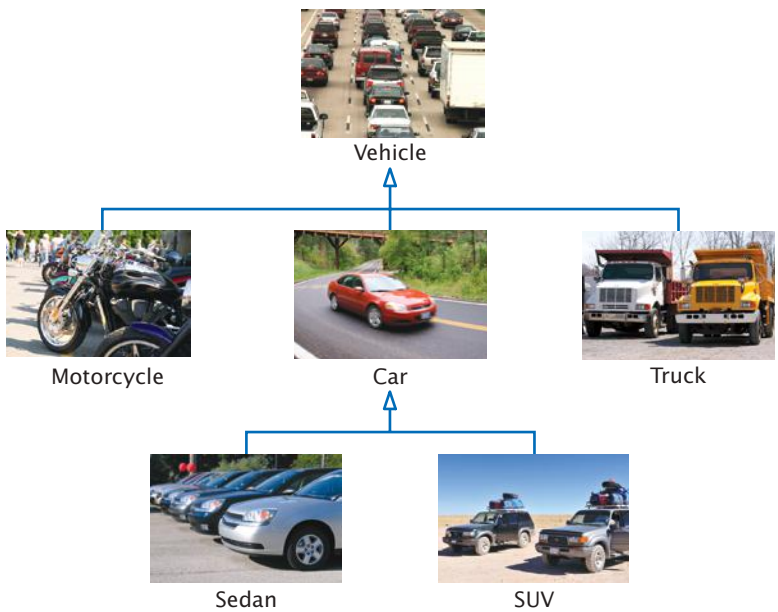
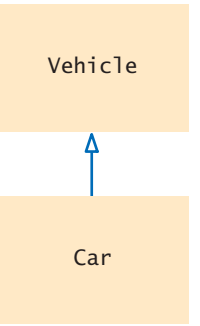


Figure 1 An Inheritance Hierarchy of Vehicle Classes

Figure 2
An Inheritance Diagram



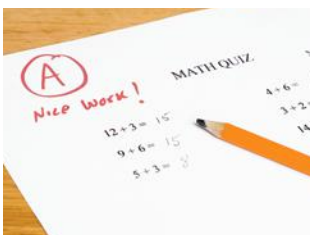
Because `Car` is a subclass of `Vehicle`, you can call that method with a `Car` object:

```
Car myCar = new Car(. . .);
processVehicle(myCar);
```

Why provide a method that processes `Vehicle` objects instead of `Car` objects? That method is more useful because it can handle *any* kind of vehicle (including `Truck` and `Motorcycle` objects). In general, when we group classes into an inheritance hierarchy, we can share common code among the classes.

In this chapter, we will consider a simple hierarchy of classes. Most likely, you have taken computer-graded quizzes. A quiz consists of questions, and there are different kinds of questions:

- Fill-in-the-blank
- Choice (single or multiple)
- Numeric (where an approximate answer is ok; e.g., 1.33 when the actual answer is 4/3)
- Free response



We will develop a simple but flexible quiz-taking program to illustrate inheritance.

Figure 3 shows an inheritance hierarchy for these question types.

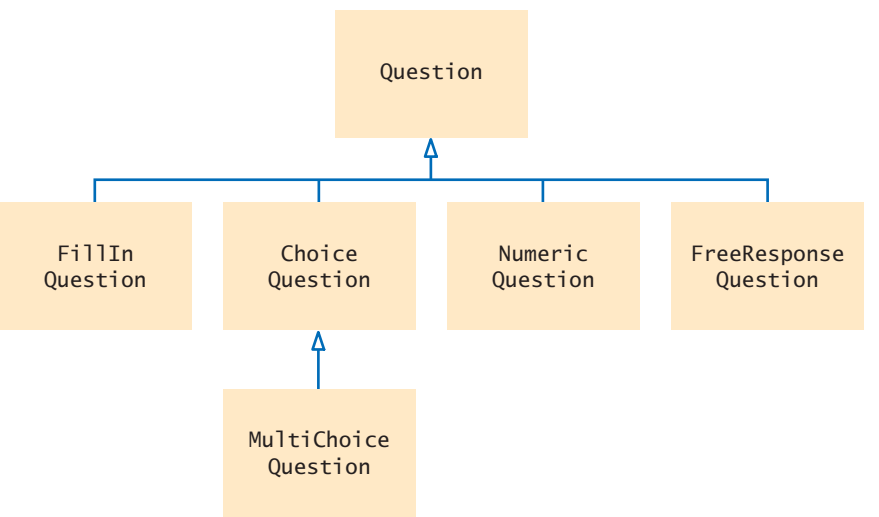


Figure 3
Inheritance Hierarchy of Question Types

At the root of this hierarchy is the `Question` type. A question can display its text, and it can check whether a given response is a correct answer.

section_1/Question.java

```
1  /**
2     A question with a text and an answer.
3  */
4  public class Question
5  {
6     private String text;
7     private String answer;
8
9     /**
10       Constructs a question with empty question and answer.
11    */
12    public Question()
13    {
14       text = "";
15       answer = "";
16    }
17
18    /**
19       Sets the question text.
20       @param questionText the text of this question
21    */
22    public void setText(String questionText)
23    {
24       text = questionText;
25    }
26
27    /**
28       Sets the answer for this question.
29       @param correctResponse the answer
30    */
31    public void setAnswer(String correctResponse)
32    {
33       answer = correctResponse;
34    }
35
36    /**
37       Checks a given response for correctness.
38       @param response the response to check
39       @return true if the response was correct, false otherwise
40    */
41    public boolean checkAnswer(String response)
42    {
43       return response.equals(answer);
44    }
45
46    /**
47       Displays this question.
48    */
49    public void display()
50    {
51       System.out.println(text);
52    }
53 }
```

This question class is very basic. It does not handle multiple-choice questions, numeric questions, and so on. In the following sections, you will see how to form subclasses of the `Question` class.

Here is a simple test program for the `Question` class:

section_1/QuestionDemo1.java

```
1  import java.util.Scanner;
2
3  /**
4     This program shows a simple quiz with one question.
5  */
6  public class QuestionDemo1
7  {
8     public static void main(String[] args)
9     {
10       Scanner in = new Scanner(System.in);
11
12       Question q = new Question();
13       q.setText("Who was the inventor of Java?");
14       q.setAnswer("James Gosling");
15
16       q.display();
17       System.out.print("Your answer: ");
18       String response = in.nextLine();
19       System.out.println(q.checkAnswer(response));
20    }
21 }
```

Program Run

```
Who was the inventor of Java?
Your answer: James Gosling
true
```

1. Consider classes `Manager` and `Employee`. Which should be the superclass and which should be the subclass?
2. What are the inheritance relationships between classes `BankAccount`, `CheckingAccount`, and `SavingsAccount`?
3. Figure 7.2 shows an inheritance diagram of exception classes in Java. List all superclasses of the class `RuntimeException`.
4. Consider the method `doSomething(Car c)`. List all vehicle classes from Figure 1 whose objects *cannot* be passed to this method.
5. Should a class `Quiz` inherit from the class `Question`? Why or why not?

Practice It Now you can try these exercises at the end of the chapter: R9.1, R9.7, R9.9.