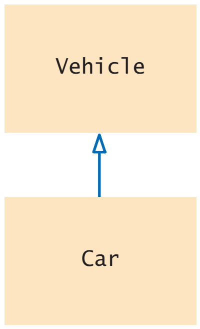
Figure 2
An Inheritance Diagram



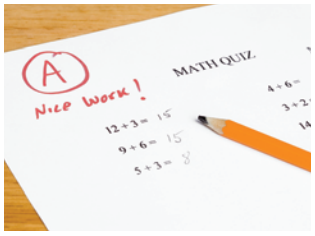
Because `Car` is a subclass of `Vehicle`, you can call that method with a `Car` object:

```
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.

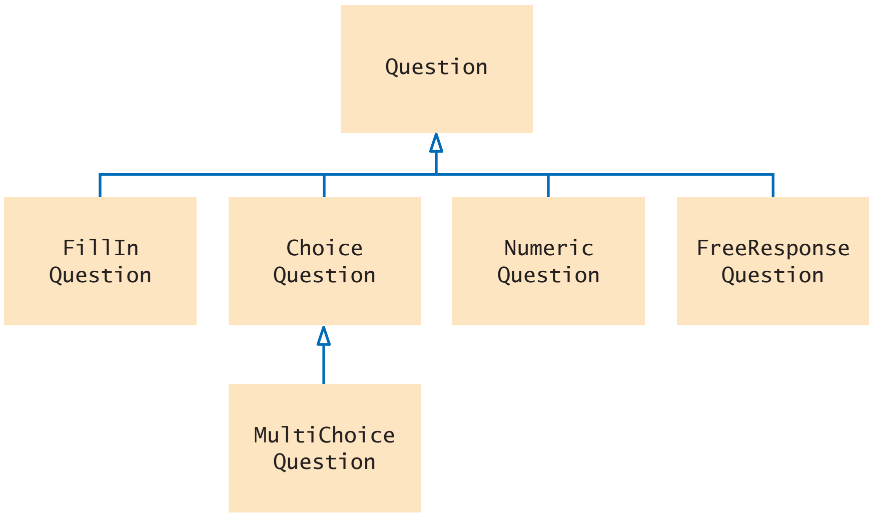


Figure 3
Inheritance Hierarchy of Question Types

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.

sec01/Question.java

```java
/**
   A question with a text and an answer.
*/
public class Question
{
   private String text;
   private String answer;

   /**
      Constructs a question with empty question and answer.
   */
   public Question()
   {
      text = "";
      answer = "";
   }

   /**
      Sets the question text.
      @param questionText the text of this question
   */
   public void setText(String questionText)
   {
      text = questionText;
   }

   /**
      Sets the answer for this question.
      @param correctResponse the answer
   */
   public void setAnswer(String correctResponse)
   {
      answer = correctResponse;
   }

   /**
      Checks a given response for correctness.
      @param response the response to check
      @return true if the response was correct, false otherwise
   */
   public boolean checkAnswer(String response)
   {
      return response.equals(answer);
   }

   /**
      Displays this question.
   */
   public void display()
   {
      System.out.println(text);
   }
}
```

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:

sec01/QuestionDemo1.java

```java
import java.util.Scanner;

/**
   This program shows a simple quiz with one question.
*/
public class QuestionDemo1
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      Question q = new Question();
      q.setText("Who was the inventor of Java?");
      q.setAnswer("James Gosling");

      q.display();
      System.out.print("Your answer: ");
      String response = in.nextLine();
      System.out.println(q.checkAnswer(response));
   }
}
```

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`, `Checking-Account`, 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.

Programming Tip 9.1

Use a Single Class for Variation in Values, Inheritance for Variation in Behavior

The purpose of inheritance is to model objects with different *behavior*. When students first learn about inheritance, they have a tendency to overuse it, by creating multiple classes even though the variation could be expressed with a simple instance variable.

Consider a program that tracks the fuel efficiency of a fleet of cars by logging the distance traveled and the refueling amounts. Some cars in the fleet are hybrids. Should you create a subclass `HybridCar`? Not in this application. Hybrids don't behave any differently than other cars when it comes to driving and refueling. They just have a better fuel efficiency. A single `Car` class with an instance variable

```
double milesPerGallon;
```

is entirely sufficient.

However, if you write a program that shows how to repair different kinds of vehicles, then it makes sense to have a separate class `HybridCar`. When it comes to repairs, hybrid cars behave differently from other cars.

9.2 Implementing Subclasses

In this section, you will see how to form a subclass and how a subclass automatically inherits functionality from its superclass.

Suppose you want to write a program that handles questions such as the following:

```
In which country was the inventor of Java born?
1. Australia
2. Canada
3. Denmark
4. United States
```

You could write a `ChoiceQuestion` class from scratch, with methods to set up the question, display it, and check the answer. But you don't have to. Instead, use inheritance and implement `ChoiceQuestion` as a subclass of the `Question` class (see Figure 4).

A subclass inherits all methods that it does not override.

In Java, you form a subclass by specifying what makes the subclass different from its superclass.

Subclass objects automatically have the instance variables that are declared in the superclass. You only declare instance variables that are not part of the superclass objects.

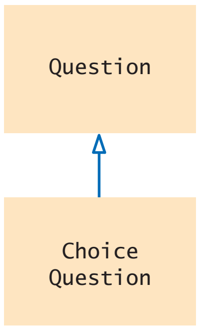


Figure 4 The `ChoiceQuestion` Class is a Subclass of the `Question` Class

Like the manufacturer of a stretch limo, who starts with a regular car and modifies it, a programmer makes a subclass by modifying another class.

A subclass can override a superclass method by providing a new implementation.

The subclass inherits all public methods from the superclass. You declare any methods that are *new* to the subclass, and *change* the implementation of inherited methods if the inherited behavior is not appropriate. When you supply a new implementation for an inherited method, you **override** the method.

A `ChoiceQuestion` object differs from a `Question` object in three ways:

- Its objects store the various choices for the answer.
- There is a method for adding answer choices.
- The `display` method of the `ChoiceQuestion` class shows these choices so that the respondent can choose one of them.

When the `ChoiceQuestion` class inherits from the `Question` class, it needs to spell out these three differences:

```
public class ChoiceQuestion extends Question
{
   // This instance variable is added to the subclass
   private ArrayList<String> choices;

   // This method is added to the subclass
   public void addChoice(String choice, boolean correct) { . . . }

   // This method overrides a method from the superclass
   public void display() { . . . }
}
```

The `extends` reserved word indicates that a class inherits from a superclass.

The reserved word `extends` denotes inheritance.

Figure 5 shows the layout of a `ChoiceQuestion` object. It has the `text` and `answer` instance variables that are declared in the `Question` superclass, and it adds an additional instance variable, `choices`.

The `addChoice` method is specific to the `ChoiceQuestion` class. You can only apply it to `ChoiceQuestion` objects, not general `Question` objects.

In contrast, the `display` method is a method that already exists in the superclass. The subclass overrides this method, so that the choices can be properly displayed.

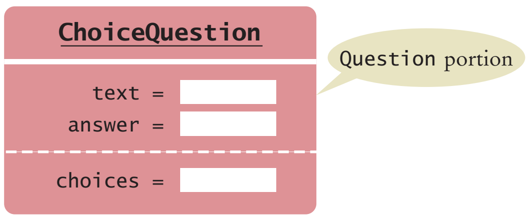
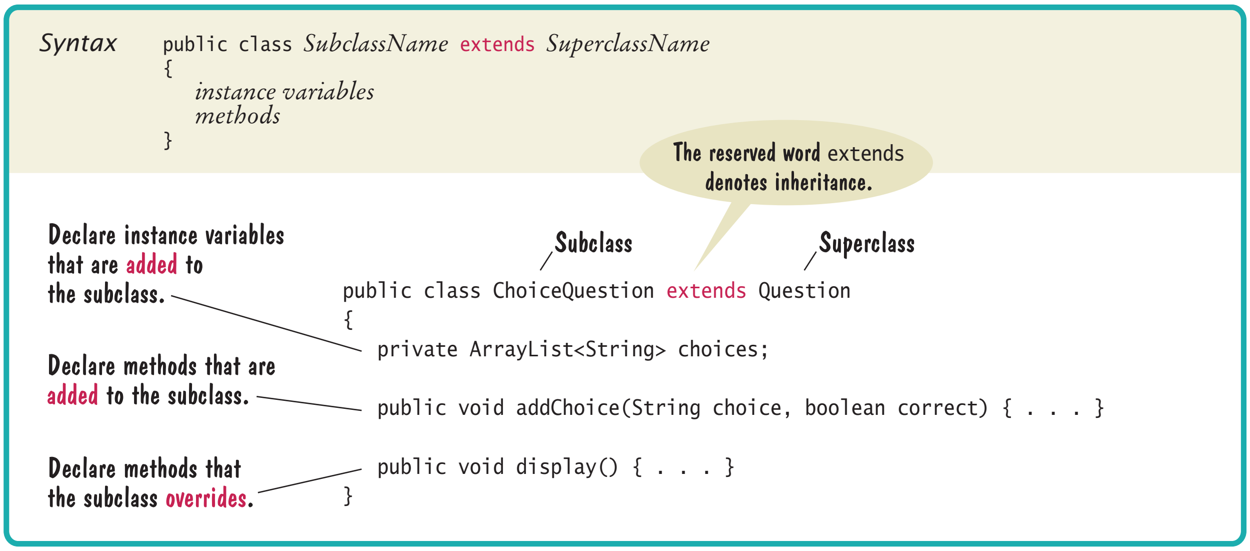


Figure 5 Data Layout of Subclass Object

Syntax 9.1 Subclass Declaration



Syntax

```
public class SubclassName extends SuperclassName
{
   instance variables
   methods
}
```

```
public class ChoiceQuestion extends Question
{
   private ArrayList<String> choices;

   public void addChoice(String choice, boolean correct) { . . . }

   public void display() { . . . }
}
```

The reserved word extends denotes inheritance.
Subclass
Superclass
Declare instance variables that are added to the subclass.
Declare methods that are added to the subclass.
Declare methods that the subclass overrides.

All other methods of the Question class are automatically inherited by the ChoiceQuestion class.

You can call the inherited methods on a subclass object:

```
choiceQuestion.setAnswer("2");
```

However, the private instance variables of the superclass are inaccessible. Because these variables are private data of the superclass, only the superclass has access to them. The subclass has no more access rights than any other class.

In particular, the ChoiceQuestion methods cannot directly access the instance variable answer. These methods must use the public interface of the Question class to access its private data, just like every other method.

To illustrate this point, let's implement the addChoice method. The method has two arguments: the choice to be added (which is appended to the list of choices), and a Boolean value to indicate whether this choice is correct. For example,

```
question.addChoice("Canada", true);
```

The first argument is added to the choices variable. If the second argument is true, then the answer instance variable becomes the number of the current choice. For example, if choices.size() is 2, then answer is set to the string "2".

```
public void addChoice(String choice, boolean correct)
{
   choices.add(choice);
   if (correct)
   {
      // Convert choices.size() to string
      String choiceString = "" + choices.size();
      setAnswer(choiceString);
   }
}
```

You can't just access the answer variable in the superclass. Fortunately, the Question class has a setAnswer method. You can call that method. On which object? The

FULL CODE EXAMPLE

Go to wiley.com/go/bjlo2code to download a program that shows a simple Car class extending a Vehicle class.

question that you are currently modifying—that is, the implicit parameter of the ChoiceQuestion.addChoice method. As you saw in Chapter 8, if you invoke a method on the implicit parameter, you don't have to specify the implicit parameter and can write just the method name:

```
setAnswer(choiceString);
```

If you prefer, you can make it clear that the method is executed on the implicit parameter:

```
this.setAnswer(choiceString);
```

SELF CHECK

6. Suppose q is an object of the class Question and cq an object of the class ChoiceQuestion. Which of the following calls are legal?

a. `q.setAnswer(response)`

b. `cq.setAnswer(response)`

c. `q.addChoice(choice, true)`

d. `cq.addChoice(choice, true)`

7. Suppose the class Employee is declared as follows:

```
public class Employee
{
   private String name;
   private double baseSalary;

   public void setName(String newName) { . . . }
   public void setBaseSalary(double newSalary) { . . . }
   public String getName() { . . . }
   public double getSalary() { . . . }
}
```

Declare a class Manager that inherits from the class Employee and adds an instance variable bonus for storing a salary bonus. Omit constructors and methods.

8. Which instance variables does the Manager class from Self Check 7 have?

9. In the Manager class, provide the method header (but not the implementation) for a method that overrides the getSalary method from the class Employee.

10. Which methods does the Manager class from Self Check 9 inherit?

Practice It Now you can try these exercises at the end of the chapter: R9.3, E9.6, E9.13.

Common Error 9.1

Replicating Instance Variables from the Superclass

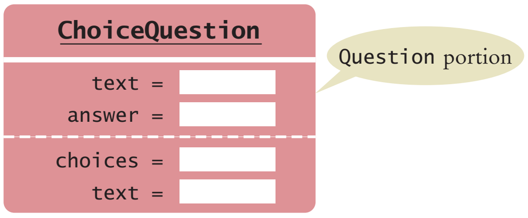
A subclass has no access to the private instance variables of the superclass.

```
public ChoiceQuestion(String questionText)
{
   text = questionText; // Error—tries to access private superclass variable
}
```

When faced with a compiler error, beginners commonly "solve" this issue by adding *another* instance variable with the same name to the subclass:

```
public class ChoiceQuestion extends Question
{
      private ArrayList<String> choices;
      private String text; // Don't!
      . . .
   }
```

Sure, now the constructor compiles, but it doesn't set the correct text! Such a `ChoiceQuestion` object has two instance variables, both named `text`. The constructor sets one of them, and the `display` method displays the other.



Common Error 9.2

Confusing Super- and Subclasses

If you compare an object of type `ChoiceQuestion` with an object of type `Question`, you find that

- The reserved word `extends` suggests that the `ChoiceQuestion` object is an extended version of a `Question`.
- The `ChoiceQuestion` object is larger; it has an added instance variable, `choices`.
- The `ChoiceQuestion` object is more capable; it has an `addChoice` method.

It seems a superior object in every way. So why is `ChoiceQuestion` called the *subclass* and `Question` the *superclass*?

The *super/sub* terminology comes from set theory. Look at the set of all questions. Not all of them are `ChoiceQuestion` objects; some of them are other kinds of questions. Therefore, the set of `ChoiceQuestion` objects is a *subset* of the set of all `Question` objects, and the set of `Question` objects is a *superset* of the set of `ChoiceQuestion` objects. The more specialized objects in the subset have a richer state and more capabilities.

9.3 Overriding Methods

An overriding method can extend or replace the functionality of the superclass method.

The subclass inherits the methods from the superclass. If you are not satisfied with the behavior of an inherited method, you *override* it by specifying a new implementation in the subclass.

Consider the `display` method of the `ChoiceQuestion` class. It overrides the superclass `display` method in order to show the choices for the answer. This method *extends* the functionality of the superclass version. This means that the subclass method carries out the action of the superclass method (in our case, displaying the question text), and it also does some additional work (in our case, displaying the choices). In other cases, a subclass method *replaces* the functionality of a superclass method, implementing an entirely different behavior.

Let us turn to the implementation of the `display` method of the `ChoiceQuestion` class. The method needs to

- Display the question text.
- Display the answer choices.

The second part is easy because the answer choices are an instance variable of the subclass.

```
public class ChoiceQuestion
{
   . . .
   public void display()
   {
      // Display the question text
      . . .
      // Display the answer choices
      for (int i = 0; i < choices.size(); i++)
      {
         int choiceNumber = i + 1;
         System.out.println(choiceNumber + ": " + choices.get(i));
      }
   }
}
```

But how do you get the question text? You can't access the `text` variable of the superclass directly because it is private.

Use the reserved word `super` to call a superclass method.

Instead, you can call the `display` method of the superclass, by using the reserved word `super`:

```
public void display()
{
   // Display the question text
   super.display(); // OK
   // Display the answer choices
   . . .
}
```

If you omit the reserved word `super`, then the method will not work as intended.

```
public void display()
{
   // Display the question text
   display(); // Error—invokes this.display()
   . . .
}
```

Because the implicit parameter `this` is of type `ChoiceQuestion`, and there is a method named `display` in the `ChoiceQuestion` class, that method will be called—but that is just the method you are currently writing! The method would call itself over and over.

Here is the complete program that lets you take a quiz consisting of two `ChoiceQuestion` objects. We construct both objects and pass them to a method `presentQuestion`. That method displays the question to the user and checks whether the user response is correct.

sec03/QuestionDemo2.java

```
import java.util.Scanner;

/**
   This program shows a simple quiz with two choice questions.
*/
public class QuestionDemo2
{
   public static void main(String[] args)
   {
      ChoiceQuestion first = new ChoiceQuestion();
      first.setText("What was the original name of the Java language?");
      first.addChoice("*7", false);
      first.addChoice("Duke", false);
      first.addChoice("Oak", true);
      first.addChoice("Gosling", false);

      ChoiceQuestion second = new ChoiceQuestion();
      second.setText("In which country was the inventor of Java born?");
      second.addChoice("Australia", false);
      second.addChoice("Canada", true);
      second.addChoice("Denmark", false);
      second.addChoice("United States", false);

      presentQuestion(first);
      presentQuestion(second);
   }

   /**
      Presents a question to the user and checks the response.
      @param q the question
   */
   public static void presentQuestion(ChoiceQuestion q)
   {
      q.display();
      System.out.print("Your answer: ");
      Scanner in = new Scanner(System.in);
      String response = in.nextLine();
      System.out.println(q.checkAnswer(response));
   }
}
```

sec03/ChoiceQuestion.java

```
import java.util.ArrayList;

/**
   A question with multiple choices.
*/
public class ChoiceQuestion extends Question
{
   private ArrayList<String> choices;

   /**
      Constructs a choice question with no choices.
   */
   public ChoiceQuestion()
   {
      choices = new ArrayList<String>();
   }

   /**
      Adds an answer choice to this question.
      @param choice the choice to add
      @param correct true if this is the correct choice, false otherwise
   */
   public void addChoice(String choice, boolean correct)
   {
      choices.add(choice);
      if (correct)
      {
         // Convert choices.size() to string
         String choiceString = "" + choices.size();
         setAnswer(choiceString);
      }
   }

   public void display()
   {
      // Display the question text
      super.display();
      // Display the answer choices
      for (int i = 0; i < choices.size(); i++)
      {
         int choiceNumber = i + 1;
         System.out.println(choiceNumber + ": " + choices.get(i));
      }
   }
}
```

Program Run

```
What was the original name of the Java language?
1: *7
2: Duke
3: Oak
4: Gosling
Your answer: *7
false
In which country was the inventor of Java born?
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

SELF CHECK

11. What is wrong with the following implementation of the `display` method?

```
public class ChoiceQuestion
{
   . . .
   public void display()
   {
      System.out.println(text);
      for (int i = 0; i < choices.size(); i++)
      {
         int choiceNumber = i + 1;
         System.out.println(choiceNumber + ": " + choices.get(i));
      }
   }
}
```

12. What is wrong with the following implementation of the `display` method?

```
public class ChoiceQuestion
{
```