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.

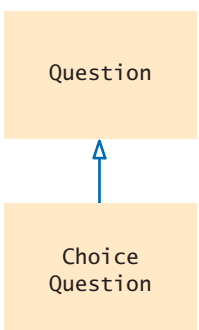


Figure 4
The `ChoiceQuestion` Class is a Subclass of the `Question` Class

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.

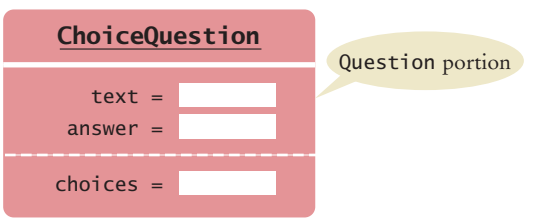
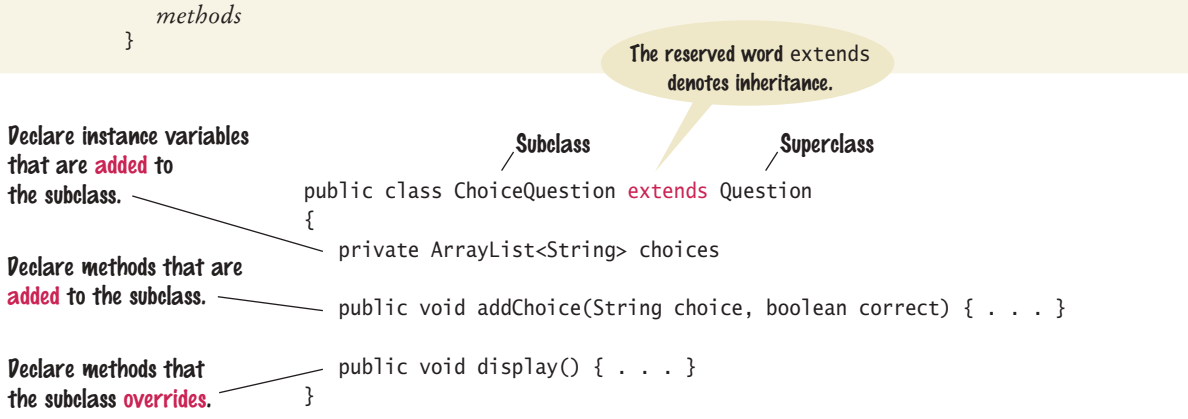


Figure 5 Data Layout of Subclass Object

Syntax 9.1 Subclass Declaration

Syntax

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



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

ONLINE EXAMPLE

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);
```

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, P9.6, P9.10.

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
{
```