© Lisa Thornberg/iStockphoto.

Chapter Ten: Inheritance

Slides by Evan Gallagher

Big C++ by Cay Horstmann

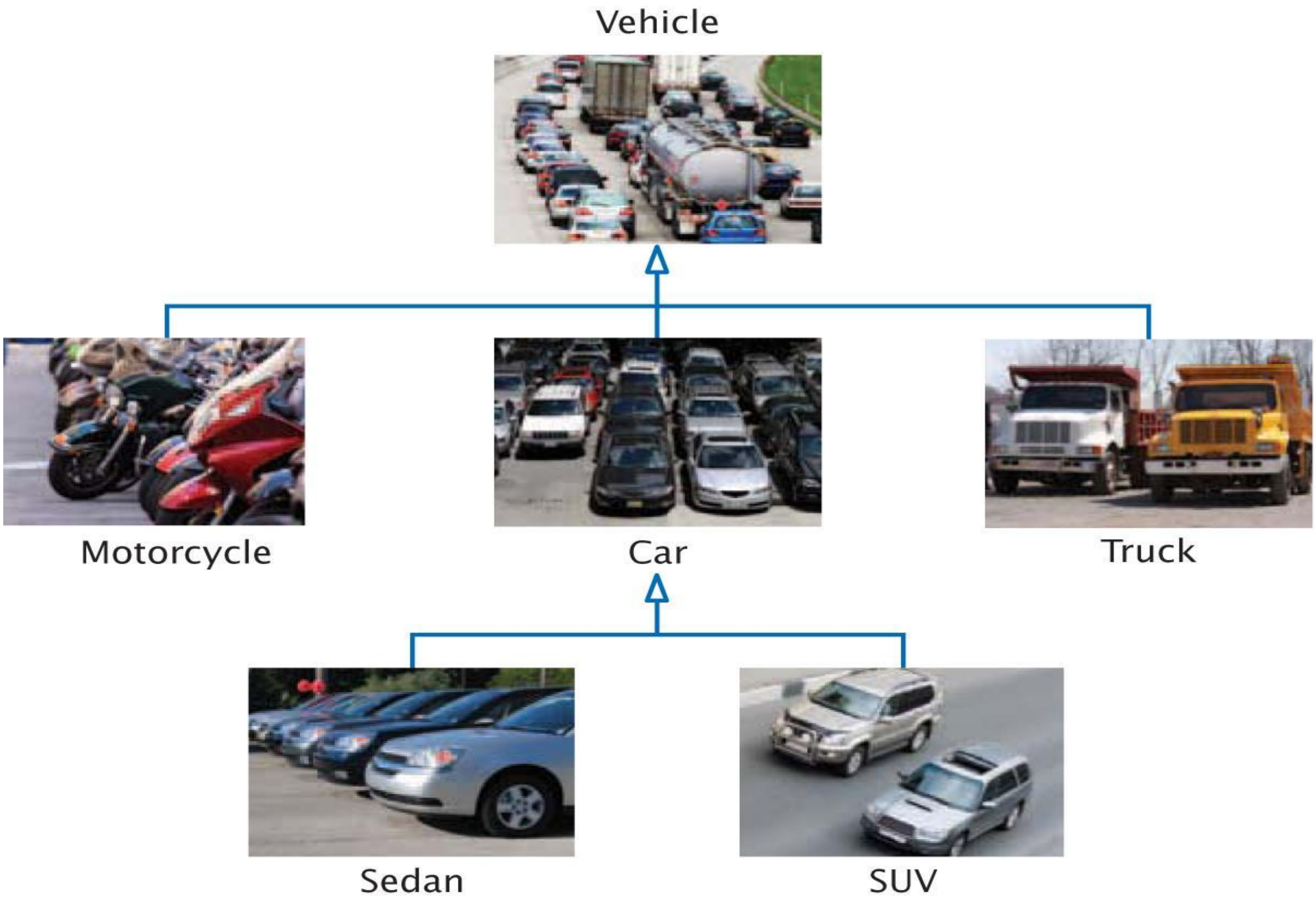
Chapter Goals

- To understand the concepts of inheritance and polymorphism
- To learn how to inherit and override member functions
- To be able to implement constructors for derived classes
- To be able to design and use virtual functions

Topic 1

1. Inheritance hierarchies
2. Implementing derived classes
3. Overriding member functions
4. Virtual functions and polymorphism

Inheritance Hierarchies



Inheritance

In object-oriented design,
inheritance is a relationship between
a more general class (called the **base class**)
and a more specialized class (called the **derived class**).

The derived class *inherits* data and
behavior from the base class.

Every car **is a** vehicle.

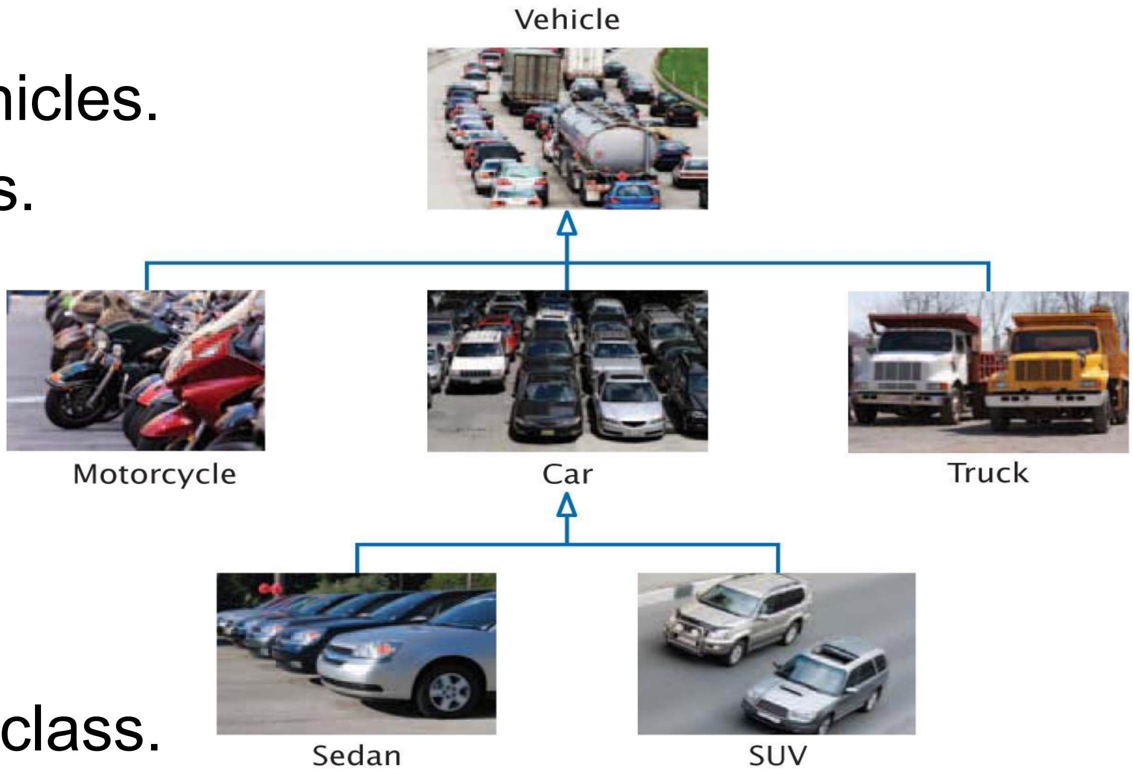
IS-A
denotes ***inheritance*.**

Inheritance: The IS-A Relationship

All Cars are Vehicles.

All Motorcycles are Vehicles.

All Sedans are Vehicles.



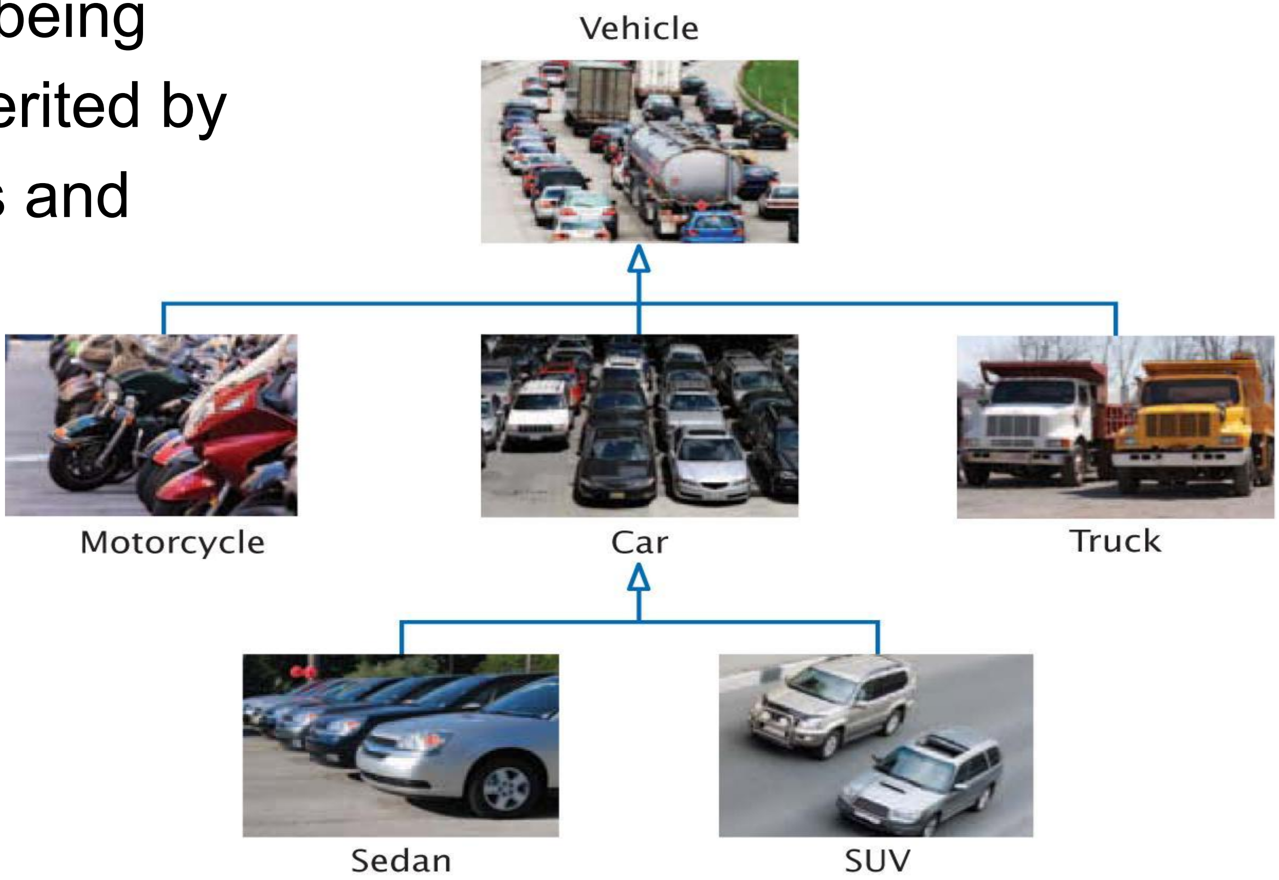
`Vehicles` is the *base* class.

`Car` is a *derived* class.

`Truck` *derives* from **`Vehicle`**

Everything a Vehicle Has is Inherited by Cars and Trucks

Everything about being a **`Vehicle`** is inherited by **`Car`**s and **`Truck`**s and **`SUV`**s.



Those things specific to **`Car`**s are *only* inherited by **`Sedan`**s and **`SUV`**s.

The Substitution Principle

The *substitution principle* states that you can always use a derived-class object when a base-class object is expected.

Suppose we have an algorithm or function that manipulates a **`Vehicle`** object.

Since a car IS-A vehicle, we can supply a **`Car`** object to such an algorithm or function, and it will work correctly.

The Substitution Principle: streams

```
void process_input(istream& in);
```

You can call this function with
an **`ifstream`** object or with an **`istream`** object.

Why?

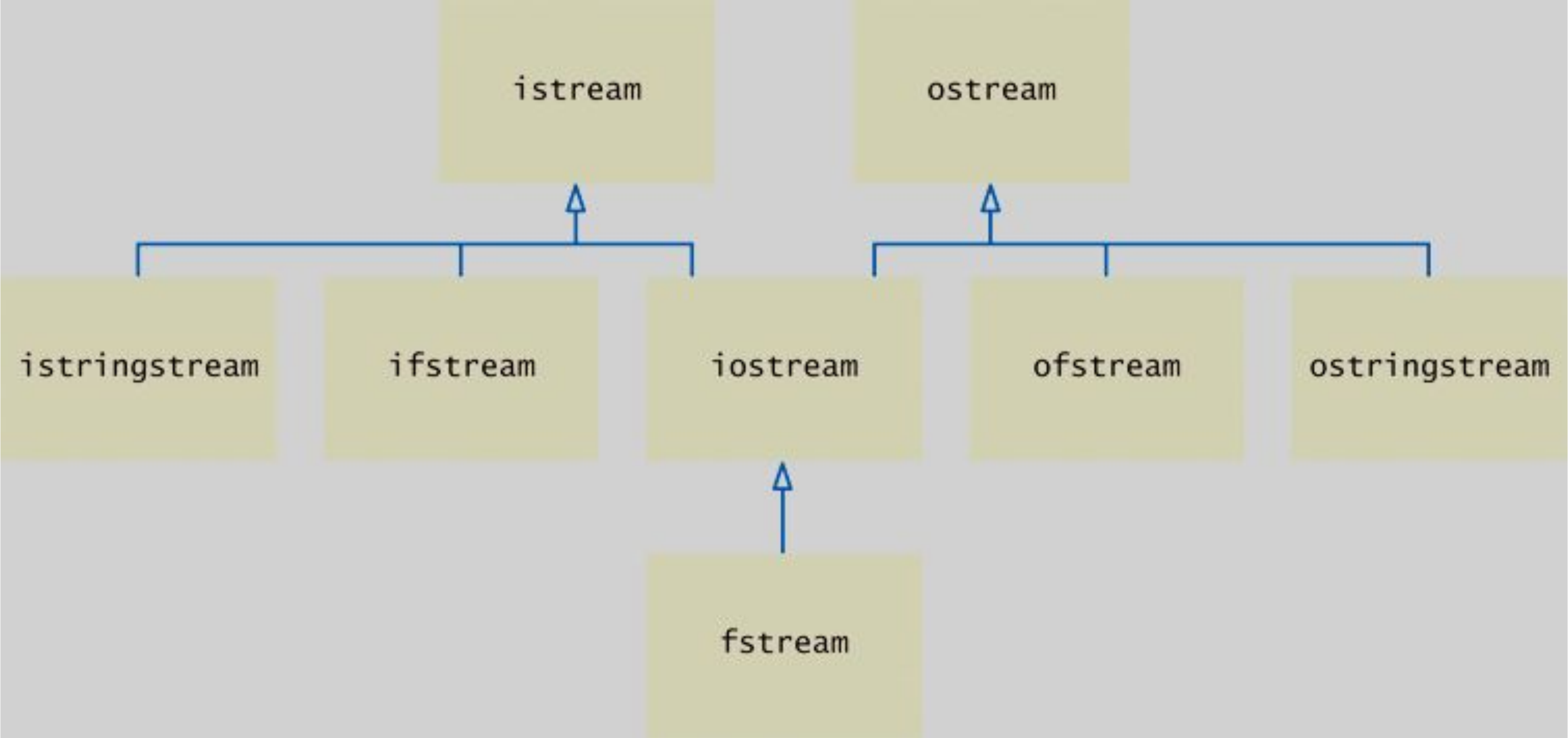
Because **`istream`** is more *general* than **`ifstream`**.

```
void process_input(ifstream& in);
```

This works by inheritance:

Big C++ by Cay Horstmann

The C++ Stream Class Hierarchy



`istream` is the base class of **`ifstream`**.

`ifstream, istringstream, and oistream all` inherit data and functions from **`istream`**.

Big C++ by Cay Horstmann

Class Hierarchy Example for a Quiz Question

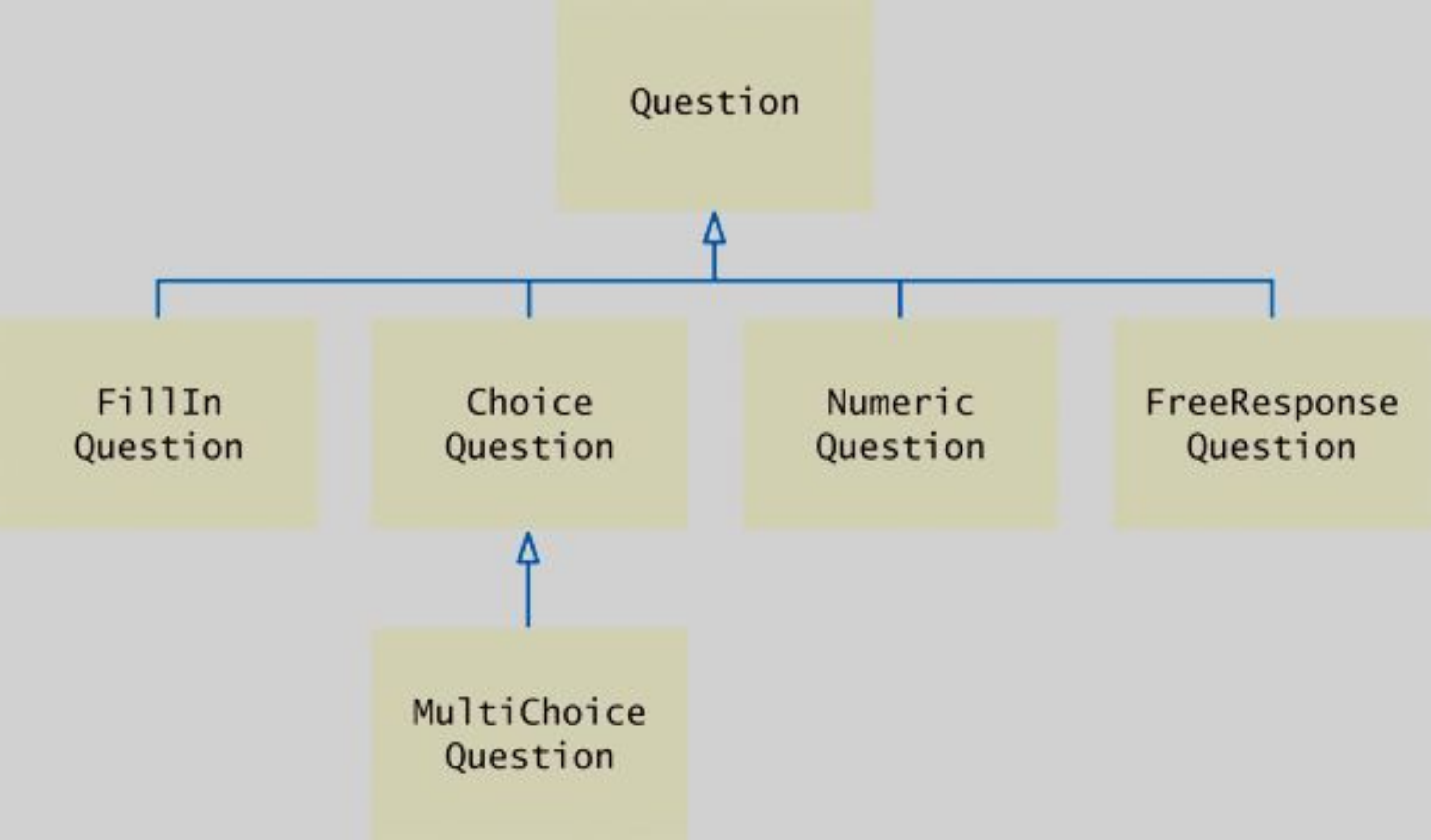
Quizzes consist of different kinds of questions:

- Fill-in-the-blank
- Choice (single or multiple)
- Numeric
  (we'll allow approximate answers to be OK)
- Free response

(We like multiple guess questions.)

Question Hierarchy

Here is the UML diagram that resulted from our analysis:



The Base Class: `Question`

We want a object of Question type to work like this:

1. First, the programmer sets the question text and the correct answer in the Question object.

2. When a user takes the test, the programmer asks the **`Question`** to display the text of the question

3. The program gets the use's response and passes it to the **`Question`** object for evaluation, to display true or false.

Big C++ by Cay Horstmann

The Base Class Code: `Question`

```
class Question
{
public:
   Question();
   void set_text(string question_text);
   void set_answer(string correct_response);
   bool check_answer(string response) const;
   void display() const;
private:
   string text;
   string answer;
};
```

Question Class & Test Program (1)

Here's a complete program
to test our Question class.

```
// sec01/demo.cpp
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

class Question
{
public:
   /**
      Constructs a question with empty text and answer.
   */
   Question();
```

Question Class & Test Program (2)

```
    /**
       @param question_text the text of this question
    */
    void set_text(string question_text);

    /**
       @param correct_response the answer to this question
    */
    void set_answer(string correct_response);

    /**
       @param response the response to check return
       @true if the response was correct,false otherwise
    */
    bool check_answer(string response) const;

    /**
       Displays this question.
    */
    void display() const;

 private:
    string text;
    string answer;
};
```

Question Class & Test Program (3)

```
Question::Question()
{ //no need to initialize here, as strings default to empty
}:

void Question::set_text(string question_text)
{
   text = question_text;
}

void Question::set_answer(string correct_response)
{
   answer = correct_response;
}

bool Question::check_answer(string response) const
{
   return response == answer;
}

void Question::display() const
{
   cout << text << endl;
}
```

Question Class & Test Program (4)

```
int main()
{
   string response;

   // Show Boolean values as true, false
   cout << boolalpha; // Notice this manipulator

   Question q1;
   q1.set_text("Who was the inventor of C++?");
   q1.set_answer("Bjarne Stroustrup");

   q1.display();
   cout << "Your answer: ";
   getline(cin, response);
   cout << q1.check_answer(response) << endl;

   return 0;
}
```

Practice It: Inheritance

Suppose you have designed an inheritance hierarchy that includes the following relationships:

- `Guitar` is derived from `Instrument`
- `AcousticGuitar` is derived from `Guitar`
- `ElectricGuitar` is derived from `Guitar`

Given the declarations below, which of the objects CANNOT be passed to the function **`tune(Guitar& g)`**?

❑ **`AcousticGuitar ag;`**

❑ **`ElectricGuitar eg;`**

❑ **`Guitar my_guitar;`**

❑ **`Instrument my_instrument;`**

Big C++ by Cay Horstmann