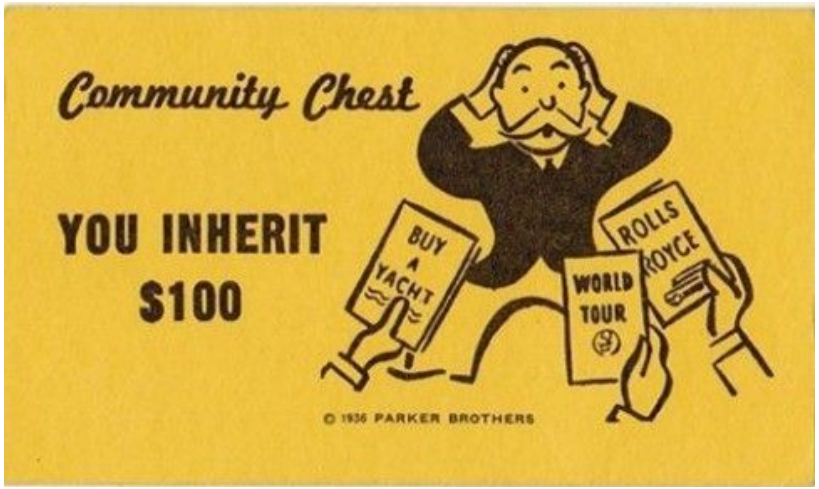
Lecture 35-36: Inheritance

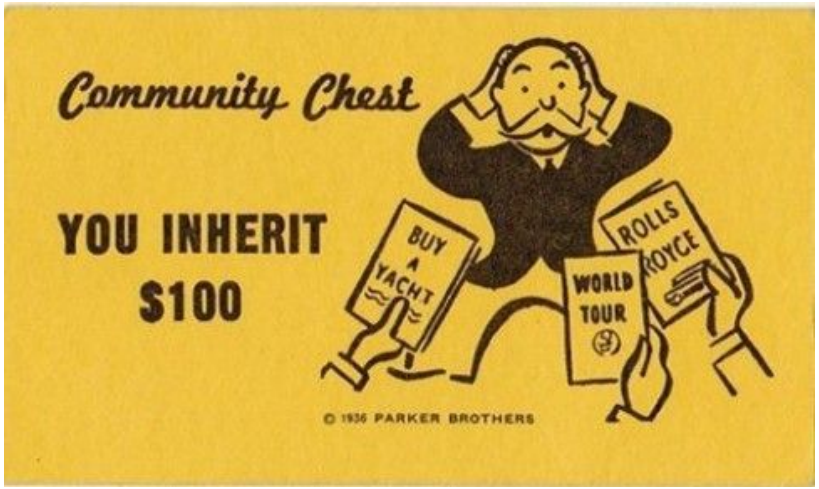


Announcements and reminders

- **Project 3!** A good goal:
  - **initialization/set-up** + **display** the whole “world”/game state -- by ~Monday
  - After that, you ***“just”*** have to move stuff around / calculate things / use mutators!

- Project 2 interview grading **by Monday April 15**

- Homework 9
  Due Wednesday April 17 by 11 PM



Last time on *Intro Computing…*

We looked at **what to do when we're handed a big project**

- Identify what are the key **structures**
- … and how those structures **relate** to one another
- Identify what are the key **functions**
- … and how these functions are related to our structures

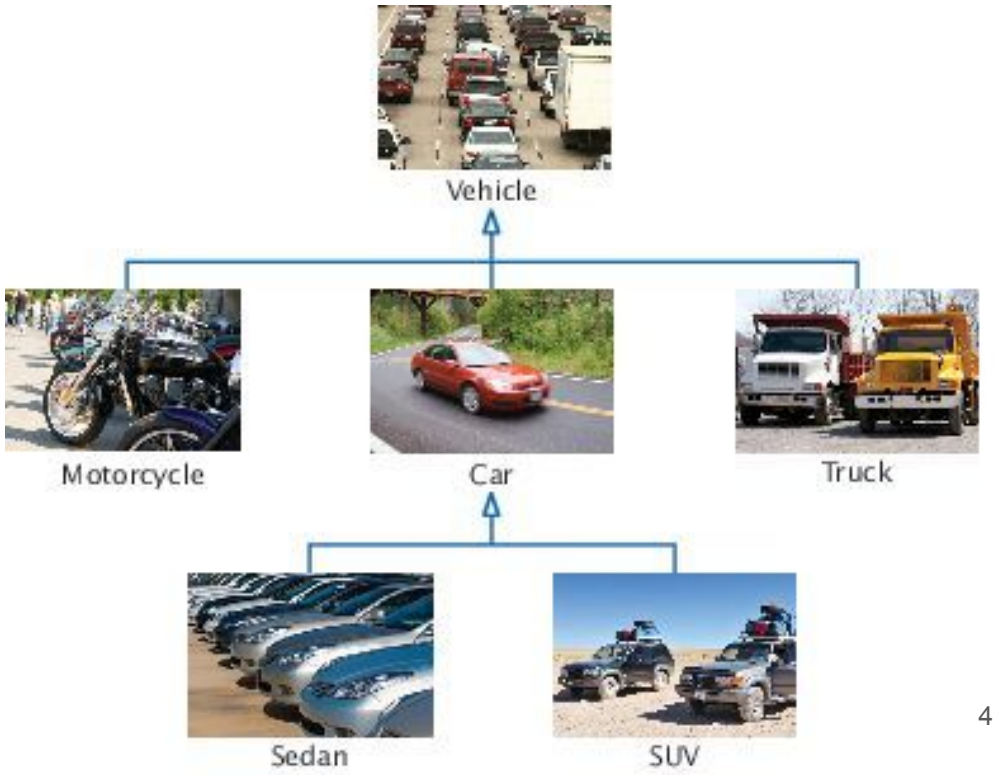
Inheritance -- what is it?

In the real-world, **inheritance** is the process by which one object (or ***class of objects***) is assigned traits/characteristics based on a larger class that it belongs to.

Can think of it as an "**is a**" relationship.

Examples:

- Every car **is a** vehicle.
- Every sedan **is a** car.



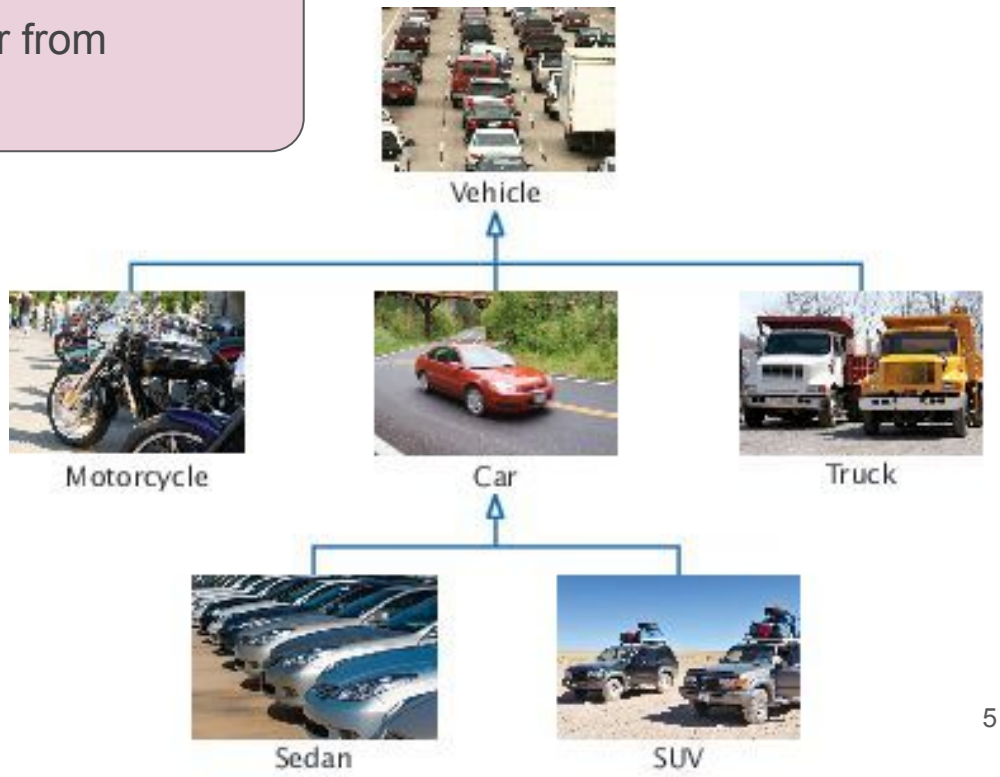
Inheritance -- what is it?

We call the more general class the **base class** and the more specialized class the **derived class**.

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

Examples:

- Every car **is a** vehicle.
- Every sedan **is a** car.

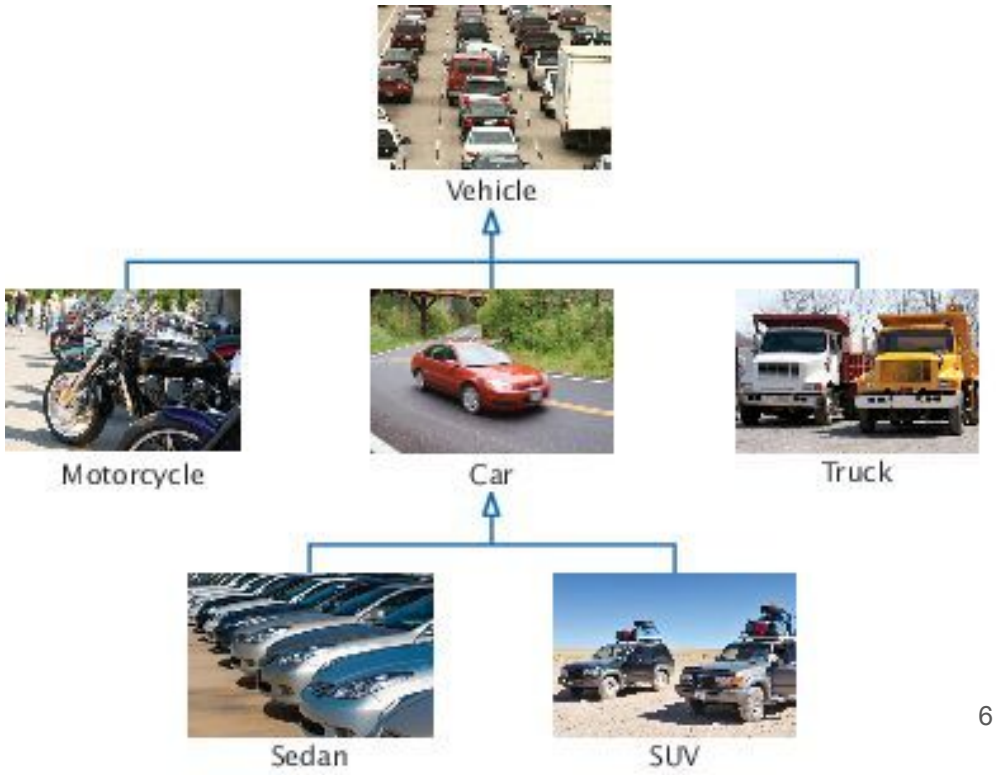


Question: In each of those examples, which is the base class and which is the derived class?

Inheritance -- what is it?

Classes become **more specific** as you go deeper into the **inheritance hierarchy**.

- Vehicle is most general
- Car **has all the properties of a vehicle**, plus some specific to cars
- Sedan **has all the properties of a car**, plus some specific to sedans
    - Sedans and SUVs **inherit** properties specific to cars



Substitution principle

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

Example: `double gasMileage(Vehicle v) {...}` // fcn to return the gas mileage of a vehicle

→ We could send a `Car` object into this function and it'd work fine, because a `Car` ***is a*** `Vehicle`.

Quiz question hierarchy

Quizzes consist of different kinds of questions:

- Fill-in-the-blank
- Choice (single or multiple)
- Numeric
- Free response

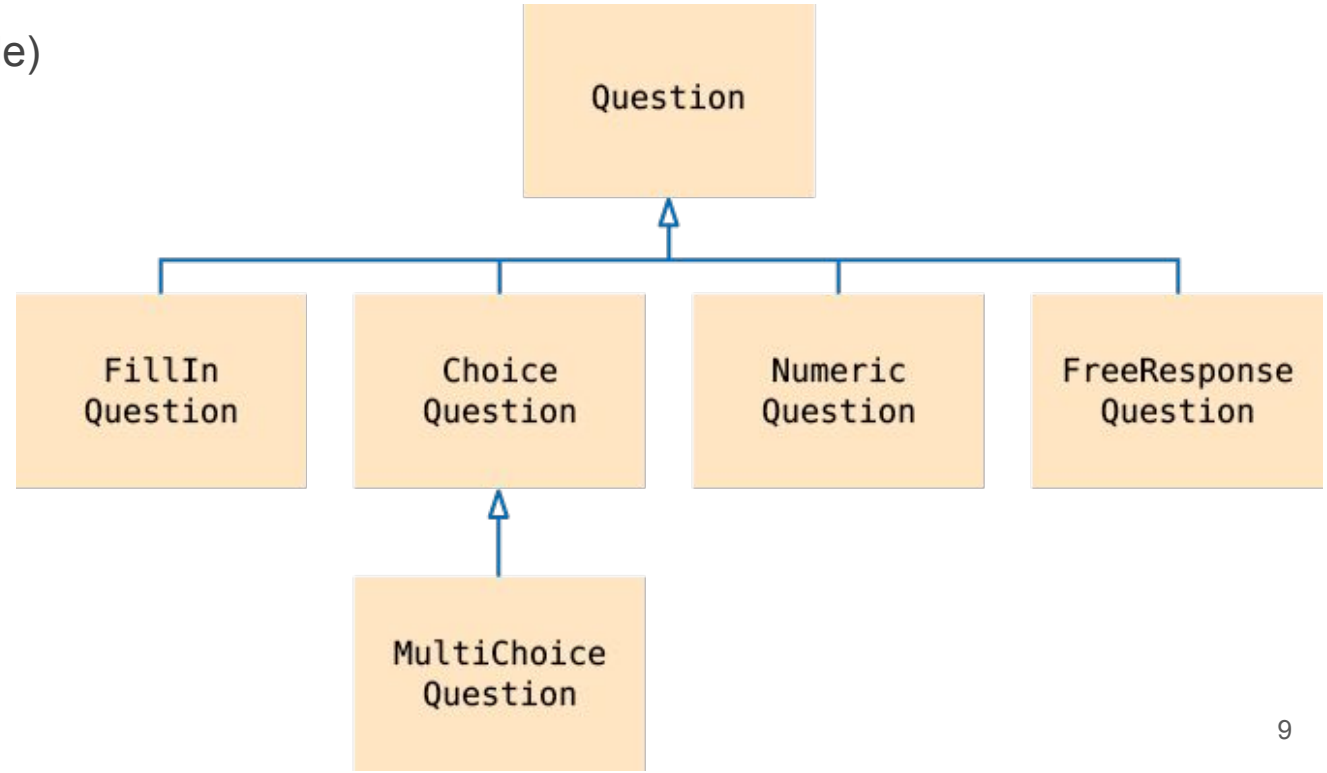
Can we sketch out this
inheritance hierarchy?

Quiz question hierarchy

Quizzes consist of different kinds of questions:

- Fill-in-the-blank
- Choice (single or multiple)
- Numeric
- Free response

Can we sketch out this inheritance hierarchy?



The base class: `Question`

We want an object of `Question` type to work like this:

1) First, the programmer sets the question text and the correct answer (stored in the `Question` object)

2) When a user takes the quiz, the programmer asks the `Question` to display the text of the question

3) The program gets the user's response and passes it to the `Question` object for evaluation, to display True or False

Member functions/data members we need:

The base class: Question

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

1) First, the programmer sets the question text and the correct answer (stored in the Question object)

2) When a user takes the quiz, the programmer asks the Question to display the text of the question

3) The program gets the user's response and passes it to the Question object for evaluation, to display True or False

Member functions/data members we need:

Member functions: `set_text`, `set_answer`, `check_answer`, `display`, constructor(s)

Data members: `text`, `answer`

Coding it up: *question.cpp*

Implementing derived classes

Now that `Question` class was the bare bones, most basic version of a quiz question. We want different types of question!

- Each special type of question ***is a*** `Question`
  (e.g., a multiple choice question ***is a*** `Question`)
- So we start with the base class (`Question`) …
- … then write code for what makes each different type of question a ***special version*** of the more general `Question` type
  - Through inheritance, each of the derived classes have the **data members** and **member functions** that we set up in the base `Question` class
  - ***AND*** we can define new stuff that makes the derived classes special!

Derived classes: ChoiceQuestion

```
class ChoiceQuestion : public Question
{
public:
   // new and/or changed member functions go here
private:
   // additional data members go here
};
```

Notes:

- The : denotes **inheritance**
- The **public** reserved word is needed for technical reasons
    - *We want to inherit **publicly**, otherwise we wouldn't be able to use our `Question` member functions except within `ChoiceQuestion`*

Derived classes: ChoiceQuestion

Let's analyze what we need to do to make our **specialized derived class.**

1) Still need to set question text (done!)
2) But now need to set **several multiple choice answer options**…
3) … which means we need to **display the question a little bit differently**.

→ we will do what is called **overriding** a member function
(overriding = rewrite a specialized version for our derived class)

Member functions/data members we need to add/modify:

Derived classes: ChoiceQuestion

Let's analyze what we need to do to make our **specialized derived class.**

1) Still need to set question text (done!)

2) But now need to set **several multiple choice answer options**…

3) … which means we need to **display the question a little bit differently**.

→ we will do what is called **overriding** a member function
(overriding = rewrite a specialized version for our derived class)

Member functions/data members we need to add/modify:

Member functions: `add_choice`, `display`, constructor(s)

Data members: `choices`

Design question:

`add_choice()` will add a choice to our answer options

What kind of data type should `choices` be: **vector** or **array**?

Derived classes: ChoiceQuestion

```
class ChoiceQuestion : public Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool correct);
   void display() const;
private:
   vector<string> choices;
};
```

Notes:

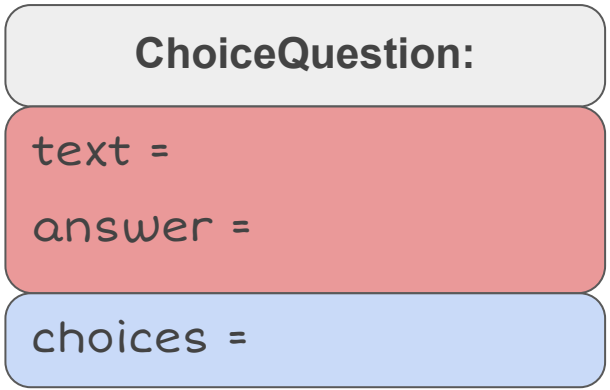
- We first specify the class we are inheriting from (**Question**) …
- … then only need to specify the differences (**new** or **modified**)
- You could slap this derived class definition right below the base class definition

Derived classes: ChoiceQuestion

ChoiceQuestion is **one type**, but made up of two parts:

- One part is inherited from Question (text, answer)
- Another part is new (choices)

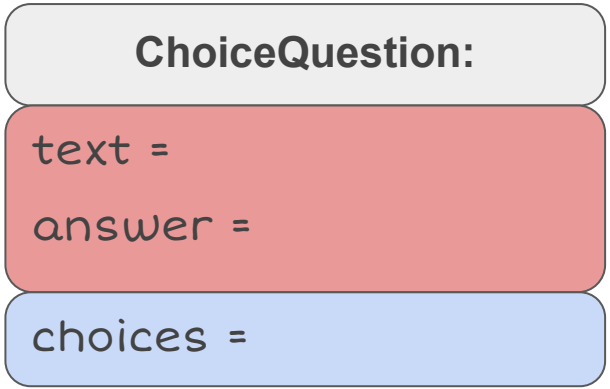
But data members from the base class are **still private**:



Derived classes: ChoiceQuestion

ChoiceQuestion is **one type**, but made up of two parts:

- One part is inherited from Question (text, answer)
- Another part is new (choices)



But data members from the base class are **still private**:

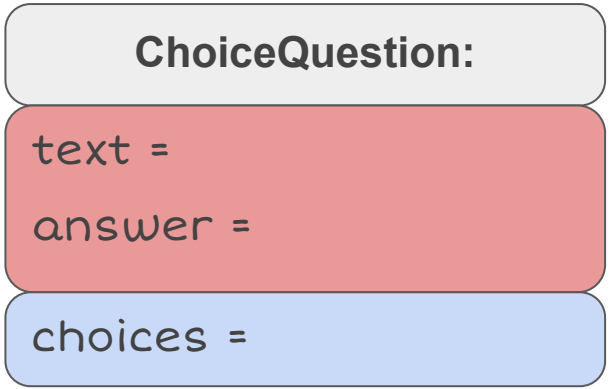
```
ChoiceQuestion q1;
q1.set_answer("2");    // calls public member fcn of base class - okay!


q1.answer = "2";     // ERROR!
```

^-- here, the answer data member is **private** to the Question class,
so even the derived type cannot access it

Moral of the story:

- When writing the ChoiceQuestion member functions, we cannot directly access the **private** data members from the Question class
- Just like any other function, we must **use our getters** from the base class



But data members from the base class are **still private**:

```
ChoiceQuestion q1;
q1.set_answer("2");    // calls public member fcn of base class - okay!


q1.answer = "2";     // ERROR!
```

^-- here, the answer data member is **private** to the Question class, so even the derived type cannot access it

New member function: add_choice

```
void ChoiceQuestion::add_choice(string choice, bool correct) {
   choices.push_back(choice);   // add the new choice
   if (correct) {               // change answer to this one's number
      // convert choices.size() to string
      string num_str = to_string(choices.size());
      // set num_str as the answer, using public member function:
      set_answer(num_str);
}
```

Notes:

New member function: add_choice

```
void ChoiceQuestion::add_choice(string choice, bool correct) {
   choices.push_back(choice);   // add the new choice
   if (correct) {               // change answer to this one's number
      // convert choices.size() to string
      string num_str = to_string(choices.size());
      // set num_str as the answer, using public member function:
      set_answer(num_str);
}
```

Notes:

- We can directly call the member fcn **set_answer()** because it's **public**, and we're already in the **Question** class (by virtue of being in the **ChoiceQuestion** class!)
- That means we have the (Choice)Question object passed as an **implicit parameter**

Derived classes: overriding member functions

Recall that our design requires that the **display** member function be rewritten in the **ChoiceQuestion** class.

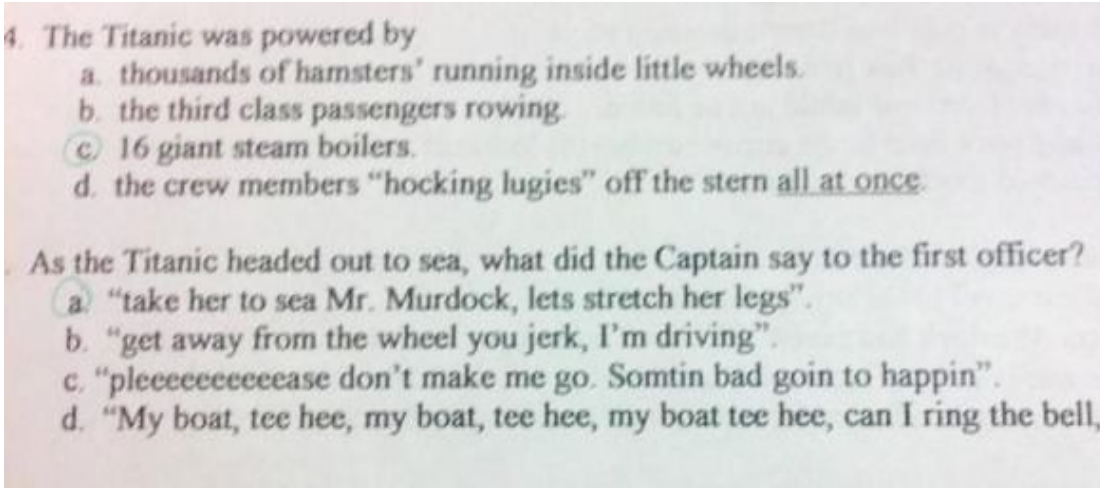
→ This is called **overriding** a member function

Question display member fcn:

1) cout the question text

ChoiceQuestion display member fcn:

1) cout the question text
2) **then cout the answer choices**



Derived classes: overriding member functions

The second part is easy -- just a loop to print out the derived-class **choices** data member:

```
void ChoiceQuestion::display() const {
    // Display the question text
    ...
    // Display the answer choices
    for (int i = 0; i < choices.size(); i++) {
        cout << i+1 << ": " << choices[i] << endl;
    }
}
```

Derived classes: overriding member functions

The first part ***seems*** easy -- just call the **display** member fcn in the **Question** class:

```
void ChoiceQuestion::display() const {
    // Display the question text
    display();    // ERROR: calls the ChoiceQuestion version of display
    // Display the answer choices
    for (int i = 0; i < choices.size(); i++) {
        cout << i+1 << ": " << choices[i] << endl;
    }
}
```

But the will call the ***ChoiceQuestion*** version of the **display** function!

Derived classes: overriding member functions

The first part ***seems*** easy -- just call the **display** member fcn in the **Question** class:

```
void ChoiceQuestion::display() const {
    // Display the question text
    Question::display();    // Yay! Calls the Question version of display
    // Display the answer choices
    for (int i = 0; i < choices.size(); i++) {
        cout << i+1 << ": " << choices[i] << endl;
    }
}
```

But the will call the ***ChoiceQuestion*** version of the **display** function!

→ We remedy this by forcing it to call the **Question** version by prefixing with **Question::**

Derived classes: overriding member functions

The first part ***seems*** easy -- just call the **display** member fcn in the **Question** class:

```
void ChoiceQuestion::display() const {
    // Display the question text
    Question::display();    // Yay! Calls the Question version of display

    // Display the answer choices
    for (int i = 0; i < choices.size(); i++) {
        cout << i+1 << ": " << choices[i] << endl;
    }
}
```

Note that we do not necessarily **need** to use the base class's function. We could have just rewritten the code to display the question text.

But in that case, we'd need a getter for the question **text** data member, which is **private**. So we would need a new getter for **text** in the **Question** class

What just happened?!

We just saw… **class inheritance!**

- ***Hierarchies*** showing an ***is a*** relationship
  - Ex: A car **is a** vehicle
  - Ex: An apartment **is a** household
- **Base class** vs **derived class**
  - Base = most general
  - Derived = more specific
- **Derived classes:**
  - New member fcns, data members
  - **Overriding** old member fcns