Inheritance

- A parent-child "is-a" relationship between classes
  - A child (derived class) extends a parent (base class)

Benefits:

- Code reuse
  - Children can automatically inherit code from parents
- Polymorphism
  - Ability to redefine existing behavior but preserve the interface
  - Children can override the behavior of the parent
  - Others can make calls on objects without knowing which part of the inheritance tree it is in
- Extensibility
  - Children can add behavior

Dynamic Dispatch (like Java)

- Usually, when a derived function is available for an object, we want the derived function to be invoked
  - This requires a run time decision of what code to invoke
- A member function invoked on an object should be the most-derived function accessible to the object’s visible type
  - Can determine what to invoke from the object itself
- Example:

```cpp
void PrintStock(Stock* s) { s->Print(); }
```

- Calls the appropriate Print() without knowing the actual type of *s, other than it is some sort of Stock

Poll Everywhere

- Whose Foo() is called?

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>We’re lost...</td>
</tr>
</tbody>
</table>

vtables and the vptr

- If a class contains any virtual methods, the compiler emits:
  
  - A (single) virtual function table (vtable) for the class
    - Contains a function pointer for each virtual method in the class
    - The pointers in the vtable point to the most-derived function for that class
  
  - A virtual table pointer (vptr) for each object instance
    - A pointer to a virtual table as a "hidden" member variable
    - When the object’s constructor is invoked, the vptr is initialized to point to the vtable for the object’s class
    - Thus, the vptr “remembers” what class the object is