Object-oriented: polymorphism under the hood

Carol Zander

At compile time a symbol table is built to hold all identifiers with their attributes (e.g., an array size for each dimension and type of its contents, a function return type and how many and types of all parameters). When some identifier is used, it is looked up in the symbol table, checked for correctness, and code is generated (e.g., retrieving an address). This is called **early binding**; information is known at compile time. After the compiler generates code, the symbol table is no longer needed and is discarded.

In general, binding binds *something* to *something*. A variable is bound to its type at compile time. An int const is bound to its value at compile time. A variable is bound to its address at run time. A function parameter is bound to its value at run time.

Late Binding (bound at run time)

Late binding is needed when you don't know what to do at compile time. For example, print() is virtual, `getOwner()` is not virtual:

```
Fruit* ptr;
Apple* pApple = new Apple(...);
Orange* pOrange = new Orange(...);
if (blah blah blah)
   ptr = pApple;
else
   ptr = pOrange;
ptr->print();             // isn't known which print to use at compile time
blah blah = ptr->getOwner();         // getOwner() is known at compile time
```

To accomplish late binding, the compiler creates a table, called a VTable, for each class that contains virtual functions. The addresses of the virtual functions are placed in the VTable. A pointer, called a VPtr, is placed in each class with virtual functions to point to the VTable for that class. The VTable is kept around at run-time.

When you make a virtual function call through a base-class pointer (a polymorphic call), the compiler generates code to fetch the VPtr and look up the function address in the VTable to call the right function at execution time.

For example, take the Fruit class and derived classes: Apple, Orange, and Grape. And suppose besides the pure virtual function print, there is also a virtual function `operator==` (as used in an example demonstrating casting) and a virtual function operator<. Each class' VPtr points to its VTable:

```
Apple VPtr ------------->   @Apple::print
                            @Apple::operator==
                            @Apple::operator<

Orange VPtr ------------>   @Orange::print
                            @Orange::operator==
                            @Orange::operator<

Grape VPtr ------------->   @Grape::print
                            @Grape::operator==
                            @Grape::operator<

Fruit VPtr ------------->   @Fruit::print
                            @Fruit::operator==
                            @Fruit::operator<
```

In the above code, ptr is set to either an Apple or Orange pointer. When the print() function is invoked, the correct VPtr is used and a look up in the appropriate VTable occurs. Thus the correct print() is used.