Programming Challenge 32

01) What is late binding/dynamic binding, and what mechanism does C++ have that allows for late binding/dynamic binding?

According to our textbook late binding/dynamic binding is "the technique of waiting until runtime to determine the implementation of a procedure" (649, 3rd edition). When a class member function is designated as being "virtual" the compiler regards it differently and allows for a decision to be made during program execution. Late binding is the compiler mechanism that facilitates this behavior.

The book also mentions about virtual functions: "A virtual function is indicated by including the modifier virtual in the member function declaration (which is given in the definition of the class). If a function is virtual and a new definition of the function is given in a derived class, then for any object of the derived class, that object will always use the definition of the virtual function that was given in the derived class, even if the virtual function is used indirectly by being invoked in the definition of an inherited function. This method of deciding which definition of a virtual function to use is known as late binding" (655, 3rd edition). This is then followed by a brief description of polymorphism, which relates all three terms to one another: "Polymorphism refers to the ability to associate many meanings to one function name by means of the late-binding mechanism. Thus, polymorphism, late binding, and virtual functions are really the same topic" (655, 3rd edition).

02) What do we call a class that contains one or more pure virtual functions?

A class with one or more pure virtual functions is called an "abstract class." According to our textbook, an abstract class "can only be used as a base class to derive other classes. You cannot create objects of an abstract class, since it is not a complete class definition. An abstract class is a partial class definition because it can contain other member functions that are not pure virtual functions. An abstract class is also a type, so you can write code with parameters of the abstract class type and it will apply to all objects of classes that are descendants of the abstract class" (658).

03) Can you create instances of abstract classes?

No instances of abstract classes cannot be created, as mentioned in the answer to question 02.

04) What is the effect on a class, if it is derived from an abstract class, of the presence of pure virtual functions in the abstract class?

The derived class will also be abstract unless it provides function definitions for all inherited pure virtual functions. According to our textbook: "If you derive a class from an abstract class, the derived class will itself be an abstract lass unless you provide definitions for all the inherited pure virtual functions (and also do not introduce any new pure virtual functions). If you do provide definitions for all the inherited pure virtual functions (and also do not introduce any new pure virtual functions), the resulting class is not an abstract class, which means you can create objects of the class" (658, 3rd edition).

05) What is the purpose of declaring destructors of base class virtual? Why/when is this important?

Declaring a base class destructor as virtual ensures that the correct destructor function is executed when polymorphism is being used in the program: during a deletion an instance of a derived class assigned to a pointer of the base class will have the derived type's destructor called only if the base class had its destructor function designated as virtual. If the function of the base class was not marked as virtual then the destructor for the base class will be called when the instance is deleted and the unique destructor of the derived class will never be executed.

Declaring destructor functions as virtual is important because it ensures that the correct memory cleanup operations are performed when an object is deleted. This will prevent memory leaks from occurring when an instance that contains dynamic data members assigned to a parent pointer is deleted.

06) Of the classes you implemented for today's challenge, which are the abstract classes?

The class Object is the only abstract class because it is the only class that contains a pure virtual function. Although the other classes in the programming challenge are descendents of the Object class, none of them declare new pure virtual functions and each includes a function definition for the pure virtual function—this

ensures that they cannot be abstract classes.

07) Of the classes you implemented for today's challenge, for which is it critical that its base classes have virtual destructors, and why?

Both the Object class and the Item class must have destructors marked as virtual because they both serve as parent classes to the derived classes MagicItem, KeyItem and FoodItem. The derived classes will not have their unique destructor functions executed unless the destructors for both parent classes are designated as virtual.