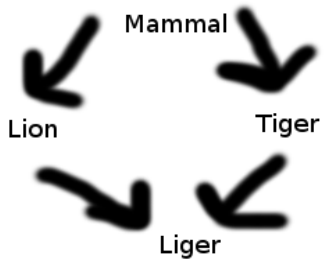
Matthew Koken
COEN 171
Quiz 6

1. Explain why multiple inheritance can be a problem.



Multiple inheritance presents us an issue of disambiguation, as seen in the diamond problem. A class can inherit properties from a parent class. Given two parent classes (multiple inheritance) there is the possibility of two different versions of the same function or variable being inherited. If the liger wants to use an inherited method hunt, and has inherited a hunt method from each parent, it does not know which version of hunt to use.

2. Explain and give examples about how C++ solves the problem of multiple inheritance

C++ solves the problem with virtual virtual inheritance. When declaring the child class, you define the parent class as a virtual. This guarantees that there will only be a single instance of the base class, instead of multiple. This clears up the ambiguity issue of the diamond problem.

So

```
class Mammal{
        ...
        virtual hunt();
        ...
};
class Lion : public virtual Mammal {
        ...
        hunt();
        ...
};
class Tiger : public virtual Mammal{
        ...
        hunt();
```

```
    ...
};
class Liger : public Lion, public Tiger{
    ...
    ...
};
```

3. How does Java solves the problem of multiple inheritance? Is there any work around? and if so explain it

Java solves the problem using interfaces. A child class may implement multiple parent classes, or can extend and implement another class. However, it cannot implement multiple interfaces. This means that, liger implementing tiger and lion, liger must provide its own implementation for the function.

```
Class Liger implements Lion, Tiger{
    hunt() {
        //liger specific implementation
    }
}
```

4. Explain how a class is implemented. give example

In C++, a class is implemented as a sort of data structure, with its own included variables and functions. The functions and variables can have multiple levels of privacy – public (everything has access), protected (child classes have access), and private (only the class and friends have access). A class, at minimum, has a name, and a constructor and destructor (though those may be created by default/automatically by the compiler). This handles the creation and destruction of the class. The functions and variables may be added.

```
class Foo {
    private:
        int *var1;
    public:
        Foo(){ var1 = new int(5);}//constructor
        ~Foo(){ delete(var1);} //destructor
        Bar() { cout << var1<< endl;} //function
};
```

5. Define what is an Abstract Data Type.

An abstract data type is a form of data structure, where implementation does not really matter. A user should expect the given functions to perform the same all the time – they will return the expected data as intended or perform the operations as intended. But how that happens does not matter for the user. Only that the operations are performed reliably. This removal from implementation is key for an ADT – it is abstract. A stack ADT for instance will likely provide features such as push and pop – typical operations for the stack. One who uses that class should only care that those operations are performed as expected. The stack itself, however, may have one of several different implementations at its core.

6. What is the aliasing problem in Java

Aliasing is handled by Java during run time. Java variables are objects – you make a reference to the object. Assignment, in the same way, passes the reference to the variable. At runtime, the actual reference is used to decide the object, what method to use, etc. This can cause issues with calling references and modifying objects. If an object is modified by an alias, it may not be apparent. The program may compile fine, but there may be errors with calling a method if a variable is assigned to refer to a different type of object. This makes it difficult to debug, and you need to be careful when passing references. So given b extends a

A[] a = new A[10];

B[] b = a; //doing any chance to b will effect a since they refer to the same object.