Using Dynamic Dispatch with Linked Lists: a Review

Remember your good friend dynamic dispatch? Abstract classes? The 'virtual' keyword? I hope so!

Consider the power of Dynamic Dispatch and Inheritance when used with a linked list. You can have a linked list of one type of class, and then use inheritance to create separate nodes any number of different derived classes underneath it.

As an example, consider (hmm) an Airport. All the airplanes at the airport are different, but they all share the common trait of being airplanes. Now, we need to make different TYPES of airplanes—does this smell like the perfect inheritance scenario or what?

Here's an example of Dynamic Dispatch used with linked lists. Consider the following psuedo-code. Assume anything that's left out has been implemented.

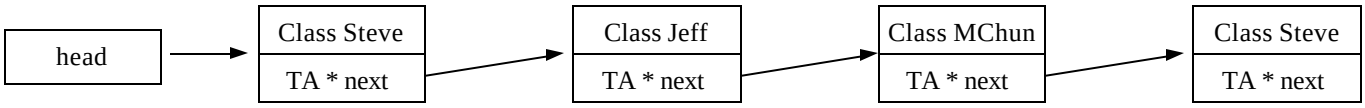
```
class TA {
        public:
        virtual void talk( ) = 0;
        TA * next;
};

class Steve : public TA {
        public:
        virtual void talk() { cout << "I like pie" << endl; }
};

class Jeff : public TA {
        public:
        virtual void talk() { cout << "I smell like tacos" << endl; }
};

class MChun : public TA {
        public:
        virtual void talk() { cout << "I enjoy peanut butter" << endl; }
};
```

Using Dynamic Dispatch with Linked Lists, part 2

Now, if we made a linked list using the following syntax:

TA * head;

We could make our list out of any of these derived classes. For example:



Now, we can go through and run the virtual methods of each node, which can do different things.

For example, if we run the talk() method on each node in this list, we would get the following. . .

"I like pie
I smell like tacos
I enjoy peanut butter
I like pie"

. .Because TA is an abstract class with virtual methods.