**When to use an Abstract Class and an Interface**

**Abstract Class vs an Interface.**

Abstract classes allow for default default function definition. This means that whatever class extends the abstract class will have access to this. If we have a base class where all the classes will perform the same function, then we can define that in our Abstract class.

An interface is a list of functions or properties that if a class implements it, it will have to have those functions defined within it. It is a situation of “Is-A” vs “Can-Do-this”. Objects that extends an Abstract class “Is-A” base class. Objects that implement “Can-Do-This”.

**When to prefer an interface**

**public** **interface** Actor{

Performance say(Line l);

}

**public** **interface** Director{

Movie direct(**boolean** goodmovie);

}

In reality, **there are Actors who are also Directors**. If we are using interfaces rather than abstract classes, we can implement both Actor and Director. We could even define an ActorDirector interface that extends both like this:

**public** **interface** ActorDirector **:** Actor, Director{

...

}

**When to prefer an Abstract class**

If you plan on updating this base class throughout the life of your program, it is best to allow that base class to be an abstract class. Why? Because you can make a change to it and all of the inheriting classes will now have this new functionality. If the base class will be changing often and an interface was used instead of an abstract class, we are going to run into problems. Once an interface is changed, any class that implements that will be broken. Now if its just you working on the project, that’s no big deal. However, once your interface is published to the client, that interface needs to be locked down. At that point, you will be breaking the clients code.