Design Pattern, Parse, Search, Update

Singleton

```
namespace PracticePanther.models {
    public class ClientService{
        private IList<Client> clients;
        // private object lock;

        // Singleton
        public static ClientService Current {
            get {
                // lock(lock){ This is a simaphore, this ensures that the thread is not
creating duplicate lists, ensuring the singleton.
                    if(instance == null){
                        instance = new ClientServie();
                    }
                    return instance;
                }
            }
        }

        private ClientServie(/*Pass in Client List, If we dont Know We Own It*/){
            clients = new List<Client>();
        }
    }
}
// if we call the constructor with the list we can supply with variable types
// the singleton pattern can fix issues of duplicate deep copy date
```

This implementation is not thread safe, so it is not nessicarily a good use of the singleton. We can avoid this by using a lock object;
We also need to call this function in a different manner.

```
// how to call
var clientService = ClientService.Current;
```

Search

If we are searching through data, we should opt to use the `.where` keyword. This replaces the need `foreach`

```
return clients.Where(
    c=>
        c.Name.ToUpper().Contains(query ?? string.Empty)
        || c.Code.ToUpper().Contains(query ?? string.Empty);
        // return the found query or an empty string
)
// This makes reference, rather than deep copy (deep vs shallow)
```

GUID

```
// we may want to use Guid for an ID type
Guid Id {get;}
Guid ClientID {get; private set;}
```