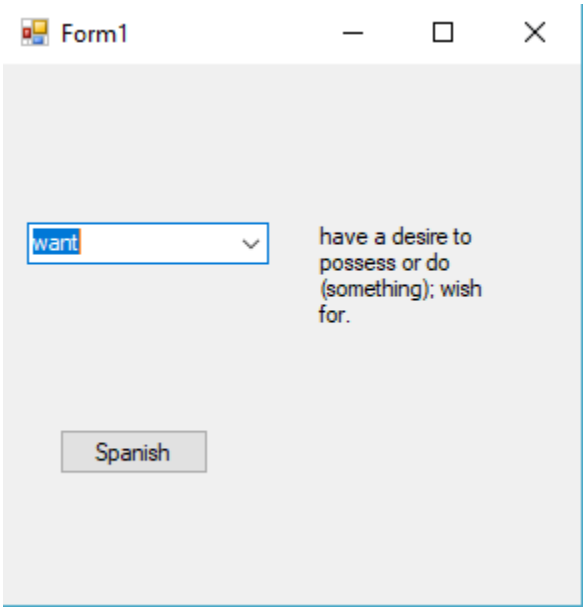
Adapter Pattern

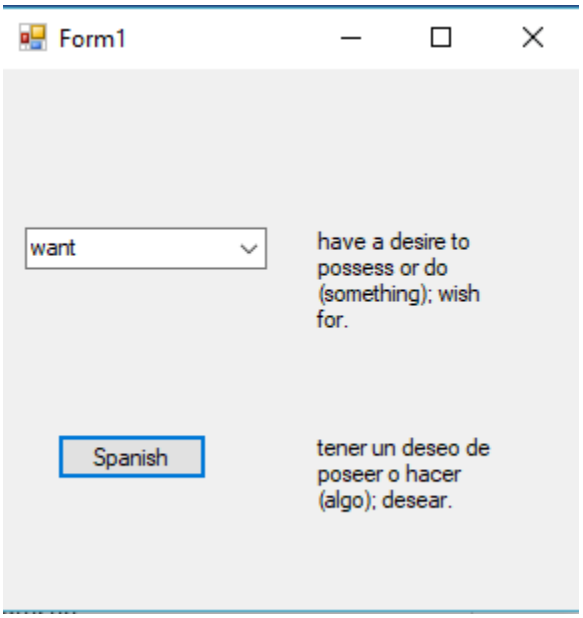
Introduction:

For this assignment, I was asked to make an app that demonstrated the adapter design pattern as described in the GO4 book. According to the DOFactory the adapter design pattern can be described as: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



Narrative:

For my demo, I created an app that takes a word in English and gives the definition, I adapted this app to also give the definition in Spanish.



The English class acts as the adaptee, and the Spanish class acts as the target. Both of these classes are extremely similar and there's really no need for an adapter,(I didn't really think about it until I was halfway through writing it) because the Spanish class could literally implement the same interface in the same way, but for the purpose of the demo I thought that a translator was a good example of something that needed adapted. So, the adaptee class has an interface called English, and the adapter class implements the English interface.

```
public interface english {
            string getDefinition(string word);
        }
```

```
    class adapter : english
     {
         spanish span = new
spanish();
         public string
getDefinition(string word)
         {
             return
span.getDefinition(word);
         }
     }
```

```
  {
        private string wantDefinition = "have a desire to possess or do (something);
wish for.";
        private string anyDefinition = "used to refer to one or some of a thing or
number of things, no matter how much or many.";
        private string becauseDefinition = "for the reason that; since.";
        private string theseDefinition = "plural form of this";
        public string getDefinition(string word)
        {
            switch (word)
            {
                case "want": return wantDefinition;
                case "any": return anyDefinition;
                case "because": return becauseDefinition;
                case "these": return theseDefinition;
                default: return " ";

            }
        }
    }
```

Conclusion:

I think that this design pattern could actually be really useful, but it took me longer than it probably should have to understand how to implement it. I may have just picked a really bad example because of how similar the class I was adapting was, but I didn't really understand the pattern at first. When the pattern did finally click though it took me about a half an hour to finish my app, so it was relatively simple. I'm pretty sure that I implemented it correctly but I'm just worried about how the class I was adapting didn't actually need to be adapted. I'm sure that if I ever have to use this pattern in a real life situation it would be a lot more beneficial.