CIS 400 HW3

General Information:

- Adam Giaccaglia
- Windows 7 64 bit
- Interpreter type:
    - Java: IntelliJ IDEA and SDK 9.0
    - C++ and C#: Visual Studio 2013

Question 1

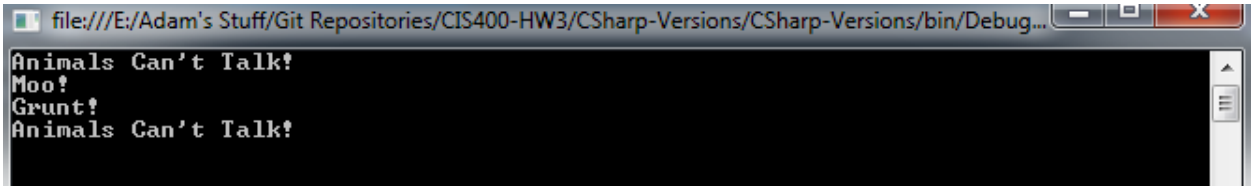
```
/*
    Java
    Question1
    Adam Giaccaglia

 */

public class Question1 {
    public static void main( String[] args ) {

        Animal[] animals = new Animal[4];

        animals[0] = new Animal();
        animals[1] = new Cow();
        animals[2] = new Pig();
        animals[3] = new Snake();

        for (int i = 0; i < animals.length; i++)
        {
            animals[i].talk();
        }

    }
}

class Animal
{
    private int Leg;
    public Animal()
    {
        Leg = 4;
    }

    public void talk()
    {
        System.out.println("Animals Can't Talk!");
    }
}

class Cow extends Animal {
    // functions are virtual by default
    public void talk() {
      System.out.println("Moo!");
  }
}
```

```java
class Pig extends Animal {
    public void talk() {
        System.out.println("Grunt!");
    }
}
class Snake extends Animal {

}
```

```
"C:\Program Files\Java\jdk-9.0.1\bin\java" "-ja
Animals Can't Talk!
Moo!
Grunt!
Animals Can't Talk!

Process finished with exit code 0
```

```csharp
/*
 * C#
 * Question 1
 * Adam Giaccaglia
 *
 */

namespace CSharp_Versions
{
    class Question1
    {
        static void Main(string[] args)
        {
            Animal[] animals = new Animal[4];

            animals[0] = new Animal();
            animals[1] = new Cow();
            animals[2] = new Pig();
            animals[3] = new Snake();

            for (int i = 0; i < animals.Length; i++)
            {
                animals[i].talk();
            }

            //pause
            Console.ReadKey();

        }
    }

    public class Animal
    {
        private int Leg;
        public Animal()
        {
            Leg = 4;
        }
```

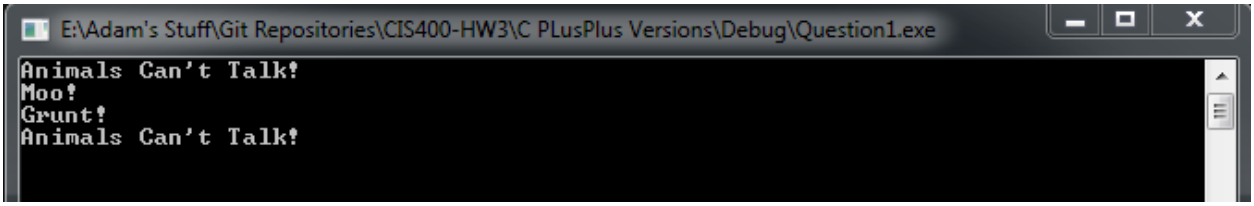
```
        public virtual void talk()
        {
            Console.WriteLine("Animals Can't Talk!");
        }
    }

    public class Cow : Animal
    {
        public Cow()
            : base()
        {

        }
        public override void talk()
        {
            Console.WriteLine("Moo!");
        }

    }

    public class Pig : Animal
    {
        public Pig()
            : base()
        {

        }
        public override void talk()
        {
            Console.WriteLine("Grunt!");
        }
    }

    public class Snake : Animal
    {
        public Snake()
            : base()
        {

        }

    }
}
```




```
/*

       Question 1
       C++
```

```
	Adam Giaccaglia

*/
#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;


class Animal
{
private:
	int Leg;

public:
	Animal()
	{
		Leg = 4;
	}
	virtual void talk()
	{
		cout << ("Animals Can't Talk!");
	}
};

class Cow : public Animal
{
public:
	void talk()
	{
		cout << ("Moo!");
	}

};

class Pig : public Animal
{
public:
	void talk()
	{
		cout << ("Grunt!");
	}

};

//public to override default private inheritance
class Snake : public Animal
{
public:

};

int _tmain(int argc, _TCHAR* argv[])
{
	//pointer for dyanmic binding and allocation
	Animal* animals[4];
```

```
        animals[0] = new Animal();
        animals[1] = new Cow();
        animals[2] = new Pig();
        animals[3] = new Snake();


        for (int i = 0; i < 4; i++)
        {
                animals[i]->talk();
                cout << endl;
                delete animals[i];
        }

        cin.get();
}
```



The difference between C# and Java is that functions are virtual by default in Java. This means no need to do anything special for dynamic binding. In C# you need the virtual and override key word. For C++, I had to do the same thing with C# and make the function virtual but no need for override. In C++ I had to use an Animal pointer to enable dynamic binding whereas C# and Java are smart enough to know the correct class.

Question 2:

```
/*
    Question2
    Java
    Adam Giaccaglia

 */

public class Question2 {
    public static void main( String[] args ) {
        Musician guy = new Musician("billy");

        guy.sing();
        guy.pianoPlay();
        guy.violinPlay();
    }
}

class Pianist
```

```java
{
    private String name;
    public Pianist(String x)
    {
        name = x;
    }
    public void pianoPlay(){
        System.out.println("Play a piano!");
    }
}

class Violinist implements IViolinist{
    public void violinPlay() {
            System.out.println("Play a violin!");
    }

}

interface IViolinist
{
    void violinPlay();
}


class Musician extends Pianist implements IViolinist{

    public Musician(String nm) {
             super(nm);
    }
    public void sing() {
            System.out.println("Sing a song!");
    }

    // makes having a violinist class redundant
    public void violinPlay() {
            System.out.println("Play a violin!");
    }
}
```

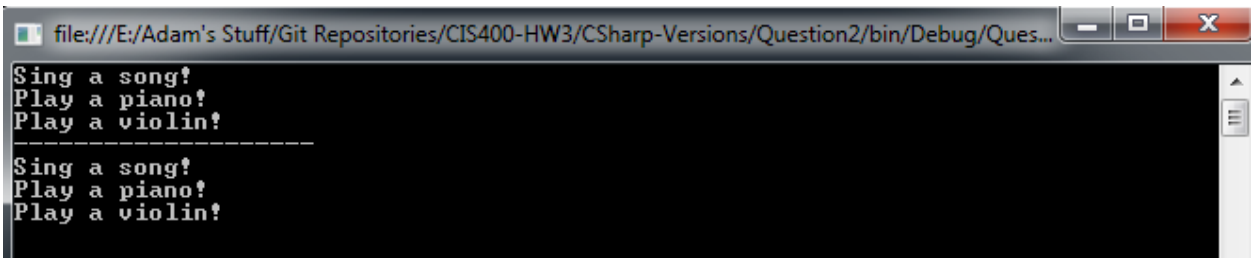
```
"C:\Program Files\Java\jdk-9.0.1\bin\java" "-javaagent
Sing a song!
Play a piano!
Play a violin!

Process finished with exit code 0
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
/*
 * Question2
 * C#
 * Adam Giaccaglia
 */

namespace Question2
{

    class Pianist
    {
        private string name;
        public Pianist(string x)
        {
            name = x;
        }
        public void pianoPlay(){
            Console.WriteLine("Play a piano!");
        }
    }

    class Violinist: IViolinist
    {
        public void violinPlay()
        {
            Console.WriteLine("Play a violin!");
        }
    }

    interface IViolinist
    {
        void violinPlay();
    }


    class Musician: Pianist, IViolinist
    {

        public Musician(string nm) : base(nm){

        }
        public void sing()
        {
            Console.WriteLine("Sing a song!");
        }

        // makes having a violinist class redundant
        public void violinPlay()
        {
            Console.WriteLine("Play a violin!");
        }
    }

    /*****************************************/
    //          Updated to fit C# better with some slight changes
    /*****************************************/
    class Pianist2 : IInterstrument
```

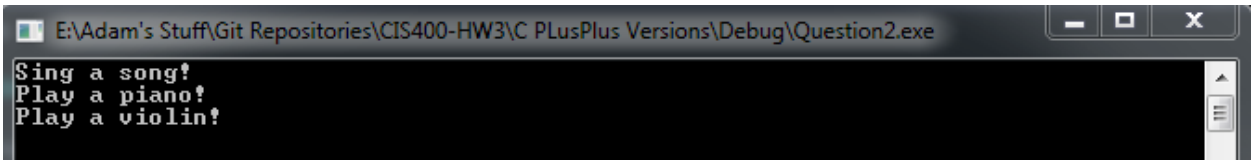
```
    {
        public void plays()
        {
            Console.WriteLine("Play a piano!");
        }
    }

    class Violinist2 : IInterstrument
    {
        public void plays()
        {
            Console.WriteLine("Play a violin!");
        }
    }

    // comman attribute that all
    public interface IInterstrument
    {
        void plays();
    }

    class Musicianv2
    {
        // uses IInterstrument instead of separate variables for each type
        private List<IInterstrument> interPlays;
        private string name;

        public Musicianv2(string nm){
            name = nm;
            interPlays = new List<IInterstrument>();
        }
        public void addinter(IInterstrument i){
            interPlays.Add(i);
        }

        public void sing()
        {
            Console.WriteLine("Sing a song!");
        }

        public void playAllInter()
        {
            // since we are using an interface we don't need to use specific functions
for each type like in previous example
            foreach (IInterstrument i in interPlays)
            {
                i.plays();
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Musician guy = new Musician("billy");

            guy.sing();
```

```
            guy.pianoPlay();
            guy.violinPlay();

            Console.WriteLine("--------------------");

            Musicianv2 bob = new Musicianv2("bob");
            IInterstrument piano = new Pianist2();
            IInterstrument violin = new Violinist2();
            bob.sing();
            bob.addinter(piano);
            bob.addinter(violin);
            bob.playAllInter();

            Console.ReadKey();
        }
    }
}
```




```
// Question2.cpp : Defines the entry point for the console application.
//

/*
       Question 2
       C++
       Adam Giaccaglia
*/

#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;

class Pianist
{
//protected to allow pianist to inherit
protected:
       string name;
public:
       Pianist(string x)
       {
              name = x;
       }
```

```
 void pianoPlay(){
        cout << ("Play a piano!") << endl;
       }
};

class Violinist
{
public:
       void violinPlay()
       {
              cout << ("Play a violin!") << endl;
       }
};

class Musician: public Pianist, public Violinist
{
public:
       Musician(string str) :Pianist(str), Violinist(){

       }
       void sing()
       {
              cout << ("Sing a song!") << endl;
       }
};

int _tmain(int argc, _TCHAR* argv[])
{

       Musician * guy = new Musician("Bob");
       guy->sing();
       guy->pianoPlay();
       guy->violinPlay();

       return 0;
}
```



C++ supports multiple inheritance so having musician inherit from pianist and violinist is simply adding a comma to add another base class.

Java and C# do not support multiple inheritance so an interface must be used. However using an interface without changing any of the classes makes the violinist class redundant. This is because Musician must implement IViolinist and therefore must have the same line as the violinist class.

So as an alternate implementation for C# I made using a new interface IInterstrument. All violinist, pianists, etc play an instrument. This is the common functionality the interface and every class implementing IInterstrument must have. Musicianv2 then uses and accepts IInterstrument instead of

using individual classes. This allows us to add more types of instruments like cellist or flautists without having to rewrite the Musician class.

For Java, I did not include the alternate implantation as it's the same C# code with minor syntax changes.

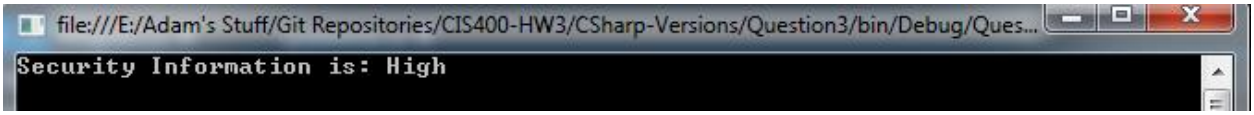
Question 3:

```
/*
    Question 3
    Java
    Ada Giaccaglia

 */


package Corporation;
public class Question3 {
    public static void main( String[] args ) {
        Bank bank1 = new Bank();
        Manager boss = new Manager();
        boss.securityAccess(bank1);
    }
}

class Bank
{
    private String name;
    protected String securityInfo;
    //so securityInfo is not null
    Bank(){
        securityInfo = "High";
    }
    public void display(){
        System.out.println("This is a bank!");
    }

}
class Manager
{
    private int id;
    public void display()
    {
        System.out.println("I am a manager!");
    }
    public void securityAccess(Bank x){
        System.out.println("Security Information is: " + x.securityInfo);
    }
}
```

```
"C:\Program Files\Java\jdk-9.0.1\bin\java" "-javaagent:C:
Security Information is: High

Process finished with exit code 0
```

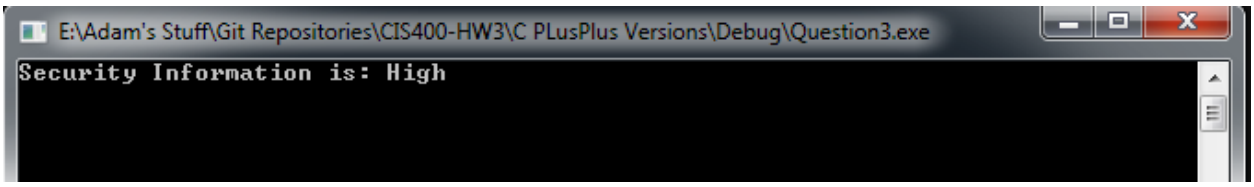
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


/*
 * Question3
 * C#
 * Adam Giaccaglia
 */

namespace Question3
{
    class Program
    {

        class Bank
        {
            private string name;
            protected internal string securityInfo;
            public Bank()
            {
                securityInfo = "High";
            }
            public void display(){
                Console.WriteLine("This is a bank!");
            }

        }
        class Manager
        {
            private int id;
            public void display()
            {
                Console.WriteLine("I am a manager!");
            }
            public void securityAccess(Bank x){
                Console.WriteLine("Security Information is: " + x.securityInfo);
            }
        }

        static void Main(string[] args)
        {

            Bank bank1 = new Bank();
            Manager boss = new Manager();
            boss.securityAccess(bank1);
```

```
            //pause
            Console.ReadKey();
        }
    }
}
```




```
// Question3.cpp : Defines the entry point for the console application.
//
/*

       Question3
       C++
       Adam Giaccaglia

*/
#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;

class Bank
{
       friend class Manager;
       private:
              string name;
       protected:
              string securityInfo;
       public:
              Bank(){
                     securityInfo = "High";
              }
              void display(){
                     cout <<("This is a bank!");
              }

};

class Manager
{

private:
       int id;
public:
       void display()
       {
              cout <<("I am a manager!");
       }
       void securityAccess(Bank x){
              cout <<("Security Information is: " + x.securityInfo);
```

```
	}
};

int _tmain(int argc, _TCHAR* argv[])
{
	Bank bank1;
	Manager boss;
	boss.securityAccess(bank1);

	return 0;
}
```



C# it won't compile unless securityInfo is also internal as protected stops a nonderived class from accessing. Making the classes internal changes northing. SecurityInfo and name are also never set. In Java, the complier didn't complain about protected and ran normally. For C++, I made the Bank class a friend of the Manager class giving the Manager class access to the Bank class's protected and private members

Friend class only works for specific classes labeled friend whereas package and internal work on a more general level.