31 August 2018 08:45

```
//steel.dll

class SteelForm: public UIForm{};
class SteelButton: public UIButton{};
class SteelTextBox: public UITextBox{};

class SteelUIFactory : public UIFactory {
    public:
        UIForm * CreateForm(){return new SteelForm; }
        UIButton * CreateButton(){return new SteelButton; }
        UITextBox * CreateTextBox(){return new SteelTextBox; }
};

declspec(dllexport) cdecl UIFactory * CreateFactory(){return new
SteelUIFactory; }
```

```
//Rubber.dll

class RubberForm: public UIForm{};
class RubberButton: public UIButton{};
class RubberTextBox: public UITextBox{};

class RubberUIFactory : public UIFactory {
    public:
        UIForm * CreateForm(){return new RubberForm; }
        UIButton * CreateButton(){return new RubberButton; }
        UITextBox * CreateTextBox(){return new RubberTextBox; }
};

UIFactory * CreateFactory(){return new RubberUIFactory; }
```

```
//core.dll
typedef UIFactory * (*FactoryCreator) ();

class UIFactoryProvider{
public:
    static UIFactory GetFactory( char *dllPath )
    {
        //load dll in memory
        HANDLE dll = LoadLibrary(dllPath); //  int dll = dlLoad(dllPath);
        if(dll==-1)
            return new DefaultFactory();

        //get a reference to CreateFactory function
        FactoryCreator creator = GetProcAddress(dll, "CreateFactory" ); // dlSym
        if(creator==NULL)
            return new DefaultFactory();

        //return the UIFactory object by calling the function
        return creator();

    }

}
```

# BookDataManager

31 August 2018 13:35

```
class BookDataManager {

    public List<Book> getAllBooks(){

        public List<Book> getAllBooks(){

            SqlConnection con=null;

            try{
                con=new SqlConnection(…);
                SqlCommand cmd=new
                SqlCommand(con);

                con.Open();

                cmd.CommandText="select * from
                books";
                List<Book> result=new List<Book>();

                SqlDataReader reader =
                cmd.ExecuteReader();

                while(reader.Read()){

                    Book book=new Book(
                            reader["Title"],
                            reader["Author"],
                            …);

                    result.add(book);
                }
                return result;
            }
            catch(Exception ex){
                Log(ex);
            }finally{

            con.Close();
            }

        }

    }
```

```
public void addBook(Book){

    SqlConnection con=null;

    try{
        con=new SqlConnection(…);
        SqlCommand cmd=new
        SqlCommand(con);

        con.Open();

        cmd.CommandText="insert into…";
        SqlDataReader reader =
        cmd.ExecuteQuery();
    }
    catch(Exception ex){
        Log(ex);
    }finally{

    con.Close();
    }

}
```

# BookDataManagerV2

31 August 2018 13:35

```
class DataManager {

        public T execute(CommandExecutor x){
            SqlConnection con=null;
            try{
                con=new SqlConnection(…);
                SqlCommand cmd=new
                SqlCommand(con);

                con.Open();

                //return x.executeCommand(cmd);
                return x(cmd);
            }
            catch(Exception ex){
                Log(ex);
            }finally{

            con.Close();
            }

        }

    }
```

```
    class BookAdder : CommandExecutor<int>{


        public int executeCommand(SqlCommand  cmd){

            cmd.CommandText="insert into…";

            return cmd.ExecuteUpdate();



        }
    }
```

```
interface CommandExecutor<T>{

    T executeCommand(SqlCommand cmd);

}

delegate T CommandExecutor<T>(SqlCommand cmd)


class BookLister implements CommandExecutor<List<Book>>{


List<Book>  execute(SqlCommand cmd){

    cmd.CommandText="select * from books";
    List<Book> result=new List<Book>();

    SqlDataReader reader = cmd.ExecuteReader();

    while(reader.Read()){

            Book book=new Book(
                reader["Title"],
                reader["Author"],
                …);

        result.add(book);
    }
    return result;

}

}
```

```
class BookDataManager{

    DataManager manager;

    public List<Book> GetAllBooks(){

        return manager.Execute(new BookLister());

    }

    public void AddBook(Book book){

        string qry=string.Format("insert into…", book.getTitle(),
                                    book.getAuthor(),…);


        manager.execute( cmd=>{
            cmd.CommandText=qry;
            cmd.ExecuteUpdate();
        });
```

```
            }


            }

        }
```