Jacob Ellis

CIS 410

Portfolio

11/26/12

**Table of Contents**

1. Resume and Cover Letter
   1. Resume
   2. Cover Letter
   3. Technology skills
2. Business and Systems Analysis
   1. Vision and Scope Documents
   2. Charters
   3. System Requirements
   4. Gantt Chart
   5. Use Cases
   6. Sequence Diagrams
   7. Class Diagrams
   8. Entity Relationship Diagrams
3. Database Design and Application
   1. Database Creation
   2. Normalization
   3. Trigger
   4. Stored Procedure
   5. Table Creation
4. Programming Skills
   1. C#
      1. Console Application
      2. Graphical User Interface Application
      3. LINQ
5. Skill Set Sheet
   1. **Resume and Cover letter**

My resume depicts my hard work and leadership through my performance in school and work that are relevant to my field of work. Also, it shows my personality and willingness to help others through my extracurricular activities such as various organizations, community service and sports. It gives a summary of what I have done prior to this point to better prepare myself for the product owner position at Humana Inc.

**Jacob Ellis**

510 Colorado Ave. • Louisville, KY 40208 • 859-628-8413 • [jmelli12@gmail.com](mailto:jmelli12@gmail.com)

**Education**

University of Louisville, Louisville, KY Expected Graduation: May 2013

**Bachelor of Science in Business Administration**

**Major: College of Business: Computer Information Systems** 3.203 GPA

**Concentration: Information Security and Web Development**

* Skilled in Excel, Access, Word, and PowerPoint for both Mac OS and Windows OS
* Well trained with UML design
* Experienced in Model View Controller (MVC3), Windows Communications Foundation (WCF) services
* Team Foundation Server (TFS), ASP.NET(C#), HTML, Cascading Style Sheets (CSS)
* Test Driven Development, SQL Database

**Experience**

**Business Analyst / Product Owner**, Humana Inc., Louisville KY August 2012 – Present

Responsibilities:

* Identify potential system impacts for incoming projects
* Facilitate meetings in order to gather requirements for project estimation
* Gather future project estimates
* Evaluate future project budget
* Work with project owners and architects to develop user stories
* Send out estimation communication updates across many teams
* Evaluate project impacts
* Write and prioritize development stories
* Work with project managers to solve project issues

**Junior Software Developer**, Humana Inc., Louisville KY May 2012 – August 2012

Responsibilities:

* Fully develop an internal Quality Management Administrator Application
* Use MVC3, WCF services, TFS, .NET(C#), SQL, HTML, CSS, TDD
* Worked with 5 other team members in an Agile Scrum team

**QA Intern,** Papa Johns Inc., Louisville, KY November 2011 – May 2012

Responsibilities:

* Tested mobile website on numerous different devices.
* Tested the Papa Johns profit system
* Wrote and updated test scripts

**Achievements**

* Awarded full scholarship to attend the John O. Moseley Leadership School Summer 2011
* Dean’s list Fall 2009, Fall 2011 & Spring 2012

**Leadership**

**Sigma Alpha Epsilon Fraternity - University of Louisville**

* Vice President position of my pledge class- Fall 2009
* Initiated second best in pledge class
* Co-Pledge Educator- Fall 2011
* Pledge Educator (Executive Counsel) – Spring 2012

**Greeks Advocating the Mature Management of Alcohol (G.A.M.M.A)**

* Vice President Fall 2011- Spring 2012

**Community Service**

* Chi Omega Chili Cook off fundraiser
* Relay for Life fundraiser
* Special Olympics – Polar Plunge
* Kappa Delta Shamrock and Run 5k fundraiser
* University of Louisville Dance Marathon for Kosairs Children’s Hospital
* SAE Volleyball tournament for Kosairs Children’s Hospital
* Memory Walk for Alzheimer’s
* Wounded Warrior – Tough Mudder Race
* Habit for Humanity
  1. **Cover Letter**

My cover letter is used to further explain my skills and experiences listed on my resume, in a way that my resume could not. It provides details such as skills, experience and interests in the company, proving I am qualified for the job at Humana Inc.

Humana Inc.

321 West Main Street

Louisville, KY 40202

November 26, 2011

Dear Member of the Selection Committee,

I am applying for an IT Product Owner position at Humana Inc. I discovered this opportunity through the University of Louisville and I believe the occupational description sounds challenging although it is something I am very interested in and I feel this is the position best suited for me. I have a vast, yet balanced, knowledge of both the Information technology field and the business aspect, which is needed to work with project managers, architects and scrum masters. I am great with people and technology and I believe this position is the greatest way for me to express those skills. Throughout my experiences I have become aware that I learn at my greatest potential through hands on activities. I know once I begin working on a project or with a scrum team, I will understand and accomplish the task at hand with ease. I am not afraid to humble myself by asking for help when I am certain I need it, although I am confident when learning and completing tasks on my own.

The product owner must be able to create and groom stories, manage project release dates and find resources to fix problems for the designated scrum team. I have large amounts of researching experience during my career at U of L, gained by doing projects and essays for my Computer Information Systems courses. My superb interpersonal skills allow me to manage situations such as scrum meetings, planning and vetting sessions, or any communications that are needed to fix a problem. My skills have developed from various class projects and working through the summer Intern program here at Humana Inc. The Internship was three months long and my scrum team successfully completed an administrative quality management system in the allotted time period. My six person scrum team developed the application using C# MVC3, which allowed us to work with HTML, JavaScript and SQL programming languages. From beginning to end I learned the processes and culture of a large corporate organization and this experience helped me apply what I have in the classroom to the real world applications. I feel much more comfortable working across numerous platforms and often across numerous teams in order to complete a task.

My experience at Humana has provided me with drive and work ethic for the product owner position in order to benefit the organization as a whole. I know I can bring hard work and commitment into my job as well as precision in all the work I do.

I deeply appreciate your time and consideration. I look forward to hearing from you in the future. I can be contacted by email ([jmelli12@gmail.com](mailto:jmelli12@gmail.com)) or phone (859-628-8413).

Sincerely*,*

Jacob Ellis

510 Colorado Ave.

Louisville, KY 40208

* 1. **Technical Skills**

My technology skills encompass every piece of technology I am confident in my ability to successfully operate at this point in time. These skills vary from operating systems, programming languages, applications, database systems, and technical writing.

**Applications**

Word  
Excel

Access

PowerPoint

Visio

Visual Studio (2008, 2010)

SQL Server Management Studio

Enterprise Library 5.0

Text Wrangler

Firefox

Google Chrome

Safari

**Languages**

C#

SQL

**Database systems**

SQL server

MySQL

Access DB

**Operating Systems**

Mac OSX

Mac leopard

Mac Snow Leopard

Mac Lion

Windows Vista

Windows XP

Windows 7

Android

**Technical Writing**

Use Cases

Development Stories

Development tasks

Documentation

1. **Business and Systems Analysis**

Business and systems analysis is the process of evaluating systems from the perspective of business as well as technology, in order to understand the structure, design and implementation of potential opportunities.

* 1. **Vision and Scope Documents**

The Vision and scope document explains and defines the business case for the project in the nontechnical language. It represents a high level overview of the requirements, features and proposed solutions.

PRINT VISION DOCUMENT AND INSERT HERE

Vision Document 1.0.docx

* 1. **Charters**

A charter is a statement of the project’s scope, objectives and participants in a project. It provides a preliminary explanation of the roles and responsibilities, outlines the project objectives, identifies the main stakeholders, and defines the authority of the project manager. It serves as a reference of authority for the future of the project.

**Team Charter**

**Team Goals**

Our main goal as a team is to develop an efficient and effective project plan for the Louisville Urban League’s Youth Development and Education Program. The plan should centralize user data in order to generate quality reports, which allows for the opportunity to receive more grants. We also would like to develop team-building skills and develop our communication skills. Becoming active listeners, good public speakers, and enhancing documentation skills are just few specific goals we have as a team. Another goal, of course, is to help out a non-profit organization in the way that they have helped our community.

**Team Meetings**

Our team meetings are not a scheduled day of the week; however, we try to meet at least twice a week, often before and directly after class because we are all in the College of Business for this class. Also, due to the way all of our schedules work out, we meet for about four hours on Monday and Wednesday afternoons but these times and days are subject to change. Some reasons that these meetings could possibly be postponed or cancelled are due to conflicting work schedules, exam or homework needs for other U of L classes, or unexpected incidents. This also, sometimes leads to meetings where not all of the members can attend so anything that was accomplished during that meeting can be found on Blackboard and are distributed through e-mail. Meeting arrangements are announced through face-to-face communication, electronic mail, phone, text, or our blackboard group discussion board. We have a group leader, but we try to decide as a team when and where meetings should take place in order to meet everyone’s needs. Our meetings are casual and relaxed, but we try to use every minute wisely and productively. We plan ahead what meetings will cover and each member takes their own notes on the topics we plan to cover prior to the meeting. During the meeting, each member takes notes on what they feel is important, however we also collaborate member ideas into the Iteration documents.

**Team Communications**

Among our team members, communication is not as structured as most other group’s communication might be. Fortunately we have a team that works well together due to our similar personalities, similar interests, common work ethics and common scheduling. Our favored communication medium is e-mail, because it allows for a paper trail and an easily accessible referencing mechanism. Decisions will be made through group discussions and during team meetings. We have decided to make Josh Wachsman our group leader and he is in charge of being the form of direct communication with Kevin Fields. If someone has a question they need to ask Mr. Fields, Josh will then send him an e-mail and then post the answers on the group discussion board, file exchange, or e-mail all group members. The reason Wachsman has the only direct contact with Mr. Fields is so he is not overloaded with the same questions and so that he can create a more personal relationship with him. It will be easier for Mr. Fields to make a more personal relationship with one member of our team rather than all five. This allows, hopefully, for quicker email and phone responses back to Wachsman with answers that our team may need.

**Team Decisions**

All of our team decisions will be made by majority rule. We came up with this idea at our first team meeting in order to limit disputes. Since there are an uneven number of members in our group, there should never be a situation where a tie can occur. Although a tie is not possible, everyone will have a chance to state their case if they don’t agree with the majority in order to try and persuade others to favor their opinion. If major conflict arises, we will first try and resolve it ourselves through a meeting with the group or just with our group leader. If conflict persists we will resort to addressing a higher authority such as Dr. Barker or another faculty/staff member at the College of Business.

**Project Repository**

Our project documentation will be maintained in a variety of ways. One way is electronically through our group file exchange on blackboard. Also, each member of our team will keep copies of all documentation on a flash drive. We also have a team project binder that will be kept with the team leader, Josh. The project binder will be organized by each iteration or phase of the project and dividers will be used to separate the phases. No one member will have the responsibility of creating the formal documents for the binder; this will be a shared responsibility among group members.

* 1. **System Requirements**

Requirements are the needs or the foundation of every project. They must be defined in order to understand exactly what the project entails and to deliver a complete and successful project.

**System Requirements**

**Functional Requirements**

1. The LUL Youth Development and Education Program system shall give the ability to apply for grants.
   1. The LUL Youth Development and Education Program system shall provide the ability to research available grants.
   2. The LUL Youth Development and Education Program system shall provide the means to fill out grant forms.
   3. The LUL Youth Development and Education Program system shall provide the means to send off grant applications.
2. The LUL Youth Development and Education Program system shall provide the ability to process grants received.
   1. The LUL Youth Development and Education Program system shall allow for evaluation of grants received.
   2. The LUL Youth Development and Education Program system shall give the ability to record grants.
   3. The LUL Youth Development and Education Program system shall provide the ability to generate receipts for grants received.
3. The LUL Youth Development and Education Program system shall give the ability to generate program enrollment.
   1. The LUL Youth Development and Education Program system shall provide the capability to obtain enrollment forms.
   2. The LUL Youth Development and Education Program system shall give the ability to process enrollment forms.
4. The LUL Youth Development and Education Program system shall provide the ability to conduct programs.
   1. The LUL Youth Development and Education Program system shall allow the directors to make the schedule.
   2. The LUL Youth Development and Education Program system shall provide

the ability to take attendance for the programs.

* 1. The LUL Youth Development and Education Program system shall permit the program coordinators to run the educational programs.
  2. The LUL Youth Development and Education Program system shall allow program directors to enter attendance into the system.

1. The LUL Youth Development and Education Program system shall give the ability of tracking students to the Senior and Assistant directors.
   1. The LUL Youth Development and Education Program system shall permit the directors to input attendance.
   2. The LUL Youth Development and Education Program system shall provide the ability to input satisfaction numbers.
2. The LUL Youth Development and Education Program system shall provide the opportunity to generate reports.
   1. The LUL Youth Development and Education Program system shall allow the directors to input requirements for the requested reports.
   2. The LUL Youth Development and Education Program system shall permit the directors to run the reports requested by the donors.
3. The LUL Youth Development and Education Program system shall allow the directors to send reports.

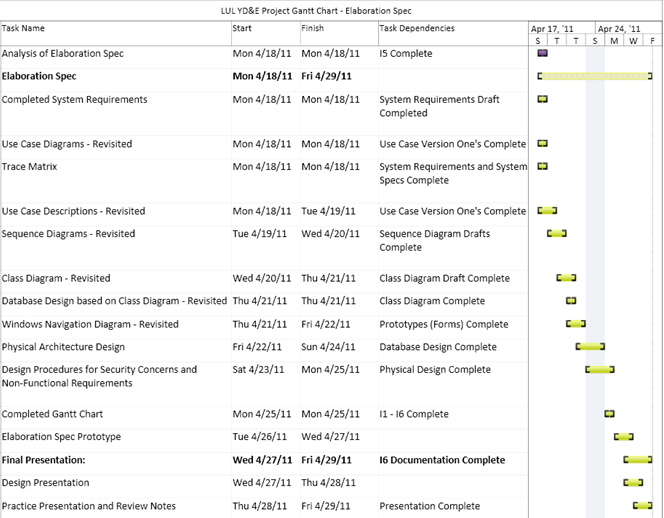
7.1 The LUL Youth Development and Education Program system shall allow the directors to print reports and file reports.

7.2 The LUL Youth Development and Education Program system shall permit the directors to send the generated reports to the donors that had requested them through email.

**Non-Functional Requirements**

1. The LUL Youth Development and Education Program system shall be available during business hours at minimum.
2. The LUL Youth Development and Education Program system shall be as efficient as excel.
3. The LUL Youth Development and Education Program system shall reduce the amount of printed documents.
4. The LUL Youth Development and Education Program system shall reduce the overall process time.
5. The LUL Youth Development and Education Program system shall provide means to centralize data.
   1. **Gantt Chart**

A Gantt chart depicts the start and end dates for a project. It can be broken down for individual sprints or even different features of the project. It allows the team to track their progress and make any changes if necessary.



* 1. **Use Cases**

A use case defines the functional requirements such as how the feature is operated or interacted with, by a user. It is made of interactions across the system’s boundaries in order to understand how the requirements affect the entities involved.

USE CASE:

1. **Use-Case Name – Generate Street Academy Enrollment**
   1. **Brief Description**
      1. The Parent/Guardian, Program Assistant, or Youth Specialist enrolls student into the Street Academy Program.
2. **Flow of Events**
   1. **Basic Flow** 
      1. The use case starts when a Parent/Guardian wants to enroll their child into the program.
      2. If Parent/Guardian decides to enroll online
         1. The system requests youth demographics, parent/guardian information, and additional youth information.
         2. The Parent/Guardian provides the youth demographics, parent/guardian information, and additional youth information.
         3. The system records the youth demographics, parent/guardian information, and additional youth information.
      3. If Parent/Guardian decides to enroll on-site
         1. Program Assistant/Youth Specialist requests youth demographics, parent/guardian information, and additional youth information.
         2. The Parent/Guardian provides the youth demographics, parent/guardian information, and additional youth information.
         3. The Program Assistant/Youth Specialist inputs into system youth demographics, parent/guardian information, and additional youth information.
         4. The system records the youth demographics, parent/guardian information, and additional youth information.
   2. Use case ends
3. **Alternative Flows**
   1. None.
4. **Special Requirements**
   1. None.
5. **Pre-conditions**
   1. **Pre-condition** 
      1. The Street Academy Program has been created within the system.
6. **Post-conditions**
   1. **Post-condition One**
      1. The enrollee’s information has been recorded.
   2. **Post-condition Two**
      1. The Parent/Guardian receives confirmation their child has been enrolled, along with a schedule of the program.
7. **Extension Points**
   1. **Name of Extension Point**
      1. Extend: Update existing youth demographics, parent/guardian information, and additional youth information.
   2. **Sequence Diagrams**

A sequence diagram is a visual aid used in depicting the interactions between objects in the sequential order that the interactions occur. This can be used to communicate how various business processes are accomplished as well as how functional requirements will interact within the current system.



* 1. **Class Diagrams**

A class diagram shows a set of classes, interfaces and associations and generalizations. It describes the attributes and operations of a class and also the constraints imposed on the system.



* 1. **Entity Relationship Diagrams**

Enterprise relationship diagrams illustrate the conceptual and logical design of a new or existing system. This is especially beneficial to an organization because it allows an easy way to communicate the design and structure of a particular system to a large amount of people.



1. **Database Design and Application**

Database design and application is the process of creating and implementing a database schema from start to finish. The design begins with understanding and fulfilling the business need in the most efficient and effective manor. Once the design is the complete the database is created and implemented.

In this section I have covered database normalization, table creation, triggers and stored procedures using SQL scripts.

* 1. Normalization

Normalization is the process of organizing data in a database by eliminating redundancy and ensuing data dependencies between tables and elements inside the database.

**First Normal Form**

ISBN, AUTHOR\_NUM, BOOKTITLE, LASTNAME, PUBLISHER, ROYALTY, EDITION

**Second normal form**

ISBN table (ISBN, BOOKTITLE, PUBLISHER, EDITION)

AUTHOR table (AUTHOR\_NUM, LASTNAME)

BOOK ROYALTY table (ISBN, AUTHOR\_\_NUM, ROYALTY)

**Third normal form**

BOOK table (BOOKTITLE, PUBLISHER)

AUTHOR table (AUTHOR\_NUM, LASTNAME)

BOOK ROYALTY table (ISBN, AUTHOR\_NUM, ROYALTY)

ISBN table (ISBN, BOOKTITLE, EDITION)

* 1. Trigger

A database trigger is code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mainly used to insert, delete or update as I display in my code in this section.

USE [CIS31008]

GO

ALTER TRIGGER A9\_TRIGGER ON ASSIGNMENT

AFTER INSERT, DELETE, UPDATE

AS

BEGIN

DECLARE @EMP\_TOTAL\_HOURS FLOAT

UPDATE EMPLOYEE

SET EMP\_TOTAL\_HOURS = 0

END

BEGIN

DECLARE JME CURSOR FOR

SELECT EMP\_NUM, SUM(ASSIGN\_HOURS) AS "SUM"

FROM ASSIGNMENT

GROUP BY EMP\_NUM

DECLARE @SUM FLOAT

DECLARE @EMP\_NUM FLOAT

SELECT 'OLD VALUES'

SELECT \*

FROM DELETED

SELECT 'NEW VALUES'

SELECT \*

FROM INSERTED

OPEN JME

FETCH NEXT FROM JME INTO @EMP\_NUM, @SUM

WHILE(@@FETCH\_STATUS=0)

BEGIN

PRINT(@EMP\_NUM +''+ @SUM )

UPDATE EMPLOYEE

SET EMP\_TOTAL\_HOURS = @SUM

WHERE EMP\_NUM = @EMP\_NUM

FETCH NEXT FROM JME INTO @EMP\_NUM, @SUM

END

CLOSE JME

DEALLOCATE JME

END

* 1. Stored Procedure

A stored procedure is a section of queries that are used to improved efficiency and usability of database applications. The benefits of using a stored procedure are the substantial performance gains from precompiled execution and code reuse.

ALTER PROCEDURE A9

AS

----- #4-1 --------

BEGIN

DECLARE @EMP\_TOTAL\_HOURS FLOAT

UPDATE EMPLOYEE

SET EMP\_TOTAL\_HOURS = 0

END

----- END #4-1 --------

------ #4-2 --------

BEGIN

DECLARE JME CURSOR FOR

SELECT EMP\_NUM, SUM(ASSIGN\_HOURS) AS "SUM"

FROM ASSIGNMENT

GROUP BY EMP\_NUM

DECLARE @SUM FLOAT

DECLARE @EMP\_NUM FLOAT

OPEN JME

FETCH NEXT FROM JME INTO @EMP\_NUM, @SUM

WHILE(@@FETCH\_STATUS=0)

BEGIN

PRINT(@EMP\_NUM +''+ @SUM )

UPDATE EMPLOYEE

SET EMP\_TOTAL\_HOURS = @SUM

WHERE EMP\_NUM = @EMP\_NUM

FETCH NEXT FROM JME INTO @EMP\_NUM, @SUM

END

CLOSE JME

DEALLOCATE JME

END

---- END #4-2 -----

* 1. Table Creation

Table creation is the process of using SQL scripts to generate a new table inside the database that can accept and hold various types of information for a specified purpose. Each table is equipped with primary and foreign keys that are used to create relationships between other tables in the same database.

My SQL code creates two tables named Assignment and employees and their respective primary and foreign keys.

CREATE TABLE ASSIGNMENT

(

ASSIGN\_NUM INT NOT NULL ,

ASSIGN\_DATE SMALLDATETIME,

PROJ\_NUM INT,

EMP\_NUM INT,

ASSIGN\_JOB INT,

ASSIGN\_CHG\_HR FLOAT,

ASSIGN\_HOURS FLOAT,

ASSIGN\_CHARGE FLOAT

)

ALTER TABLE ASSIGNMENT

ADD CONSTRAINT PK\_ASSIGNMENT PRIMARY KEY (ASSIGN\_NUM)

ALTER TABLE ASSIGNMENT

ADD CONSTRAINT FK\_ASSIGNMENT FOREIGN KEY (EMP\_NUM) REFERENCES EMPLOYEE

CREATE TABLE EMPLOYEE

(

EMP\_NUM INT NOT NULL,

EMP\_LNAME VARCHAR(15),

EMP\_FNAME VARCHAR (15),

EMP\_INITIAL VARCHAR(1),

EMP\_HIREDATE SMALLDATETIME,

JOB\_CODE INT,

EMP\_YEARS INT,

EMP\_TOTAL\_HOURS FLOAT

)

1. **Programming Skills**

My programming skills exemplify the knowledge and experience I have learned from programming in these specific languages. My domain languages are C#, SQL DB, HTML, JavaScript and CSS.

* 1. **C#**

C# is a language from the ASP.NET platform, which uses object oriented programming. This is the University of Louisville’s main programming language in the computer information system major within the business school.

* + 1. C# Console applications

Console Applications are applications that display information to the computer’s console, unlike graphical user interface applications. This console application uses user input to calculate the total square feet of a room and the amount of paint needed to paint the room, in gallons.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Program\_1

{

    class Program

    {

        static void Main(string[] args)

        {

            double length; // Length of the walls

            double height; // The height of the walls

            int num\_doors; // The number of doors

            int num\_windows; // The number of windows

            double rough\_square\_feet; // rough Square feet of the room

            double total\_sq\_feet; // Total squate feet of the room

            double gallons\_paint; // Gallons of paint needed

            const double SQ\_FT\_DOOR = 20; // Square feet per door

            const double SQ\_FT\_WINDOWS = 15; // Square feet per window

            const double sq\_feet\_gal\_paint = 350; // square feet per gallon of paint

            int num\_coats; // Number of coats of paint needed

            Console.Out.WriteLine("Enter the total Length of all of the walls to be painted in feet (add all wall lengths together):");

            length = double.Parse(Console.In.ReadLine());

            Console.Out.WriteLine("Enter the total height of all of the walls to be painted in feet (Add all wall heights together):");

            height = double.Parse(Console.In.ReadLine());

            Console.Out.WriteLine("Enter the total number of doors in the room:");

            num\_doors = int.Parse(Console.In.ReadLine());

            Console.Out.WriteLine("Enter the total number of windows in the room:");

            num\_windows = int.Parse(Console.In.ReadLine());

            Console.Out.WriteLine("Enter the number of coats of paint needed: ");

            num\_coats = int.Parse(Console.In.ReadLine());

            rough\_square\_feet = (height \* length);

            total\_sq\_feet = ((rough\_square\_feet) - (num\_doors \* SQ\_FT\_DOOR) - (num\_windows \* SQ\_FT\_WINDOWS));

            gallons\_paint = (total\_sq\_feet / sq\_feet\_gal\_paint) \* num\_coats;

            Console.Out.WriteLine("Total square feet of the room: " + total\_sq\_feet);

            Console.Out.WriteLine("Gallons of paint needed: " + gallons\_paint.ToString("0.0"));

}

}

}

* + 1. Graphical User Interface Applications

Graphical user interface applications create a form that the user is able to interact with by using mouse clicks and key strokes in order to enter and receive data. This is different than a console application because the interface can integrate drop down boxes, text boxes, buttons, and even interact with other programs on the user’s computer.

This GUI application I have written is a piece of a larger application, which emulates a library. The code creates a form allowing a library patron to be created and added to the library. This allows the patron to checkout a library book, magazine, journal, movie, and music item.

// Program 3

// CIS 200-75

// Spring 2012

// Due: 11/9/2012

// By: Jacob Ellis

// File: PatronForm.cs

// This class creates the Patron dialog box form GUI. It performs validation

// and provides String properties for each field.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace LibraryItems

{

    public partial class PatronForm : Form

    {

        // Precondition:  None

        // Postcondition: The form's GUI is prepared for display.

        public PatronForm()

        {

            InitializeComponent();

        }

        public String PatronName

        {

            // Precondition:  None

            // Postcondition: The text of form's name field has been returned

            get

            {

                return patronNameTxt.Text;

            }

            // Precondition:  None

            // Postcondition: The text of form's name field has been set to the specified value

            set

            {

                patronNameTxt.Text = value;

            }

        }

        public String PatronID

        {

            // Precondition:  None

            // Postcondition: The text of form's ID field has been returned

            get

            {

                return patronIdTxt.Text;

            }

            // Precondition:  None

            // Postcondition: The text of form's ID field has been set to the specified value

            set

            {

                patronIdTxt.Text = value;

            }

        }

        // Precondition:  Focus is shifting from patronNameTxt

        // Postcondition: If text is invalid, focus remains and error provider

        //                highlights the field

        private void patronNameTxt\_Validating(object sender, CancelEventArgs e)

        {

            if (patronNameTxt.TextLength == 0) // Empty field

            {

                e.Cancel = true;

                errorProvider.SetError(patronNameTxt, "Must provide Name");

            }

        }

        // Precondition:  Validating of patronNameTxt not cancelled, so data OK

        // Postcondition: Error provider cleared and focus allowed to change

        private void patronNameTxt\_Validated(object sender, EventArgs e)

        {

            errorProvider.SetError(patronNameTxt, "");

        }

        // Precondition:  Focus is shifting from patronIdTxt

        // Postcondition: If text is invalid, focus remains and error provider

        //                highlights the field

        private void patronIdTxt\_Validating(object sender, CancelEventArgs e)

        {

            if (patronIdTxt.TextLength == 0) // Empty field

            {

                e.Cancel = true;

                errorProvider.SetError(patronIdTxt, "Must provide ID");

            }

        }

        // Precondition:  Validating of patronIdTxt not cancelled, so data OK

        // Postcondition: Error provider cleared and focus allowed to change

        private void patronIdTxt\_Validated(object sender, EventArgs e)

        {

            errorProvider.SetError(patronIdTxt, "");

        }

        // Precondition:  User pressed on cancelBtn

        // Postcondition: Form closes and sends Cancel result

        private void cancelBtn\_MouseDown(object sender, MouseEventArgs e)

        {

            // This handler uses MouseDown instead of Click event because

            // Click won't be allowed if other field's validation fails

            if (e.Button == MouseButtons.Left) // Was it a left-click?

                this.DialogResult = DialogResult.Cancel;

        }

        // Precondition:  User clicked on okBtn

        // Postcondition: If invalid field on dialog, keep form open and give first invalid

        //                field the focus. Else return OK and close form.

        private void okBtn\_Click(object sender, EventArgs e)

        {

            if (ValidateChildren()) // If all controls validate

                this.DialogResult = DialogResult.OK; // Causes form to close and return OK result

* + 1. LINQ

LINQ is the integration of querying capabilities in the C# and .NET framework. It has become a well-known type of C# programming element that uses the SQL statement formation in order to query information is a specified format as I display in this section.

//Jake Ellis

//Due: 9/5/12

//CIS200-75

//Lab 1

//Description: Using linq to query data and foreach statements to display assortments of specific data from the Invoices array such as invoice value, description, Price, quantity, etc,

// in order to become more familiar with C# and linq in general

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab1

{

    public class LinqTest

    {

        public static void Main(string[] args)

        {

            // initialize array of invoices

            Invoice[] invoices = {

                new Invoice( 83, "Electric sander", 7, 57.98M ),

                new Invoice( 24, "Power saw", 18, 99.99M ),

                new Invoice( 7, "Sledge hammer", 11, 21.5M ),

                new Invoice( 77, "Hammer", 76, 11.99M ),

                new Invoice( 39, "Lawn mower", 3, 79.5M ),

                new Invoice( 68, "Screwdriver", 106, 6.99M ),

                new Invoice( 56, "Jig saw", 21, 11M ),

                new Invoice( 3, "Wrench", 34, 7.5M ) };

            //a) Use LINQ to sort the Invoice objects by PartDescription

                //PRE: The invoice array must be created and populated with valid information in all fields

                //POST: All of the invoice objects will be displayed and ordered in ascending order by PartDescription

            //sortedDescription: a query from the invoices array, which selects all fields and orders by the PartDescription field

            //the variable "query" represents the fields of the invoice array that are being queried and displayed

            Console.WriteLine("A) Sorted by PartDescription: ");

            var sortedDescription = from query in invoices

                                    orderby query.PartDescription

                                    select query;

            //The foreach runs through the sortedDescription query and finds each of the desired rows and then displays them

            //element is a variable for the items in the query which are being displayed

            foreach (var element in sortedDescription)

            {

                Console.WriteLine("{0}", element);

            }

            //Used to create a line of seperation between each result set

            Console.WriteLine();

//---------------------------------------------------------------------------------------------------------------------------------------------------

                        //c) Use LINQ to select the Part Description and Quantity and sort the results by Quantity

                //PRE: The invoice array must be created and populated correctly

                //POST: The query will display the Part Description and Quantity being sorted by the Quantity in ascending order

            //sortedQuantity: a query from the invoices array which selects Quantity and PartDescription and orders by the Quantity field

            //the variable "query" represents the fields of the invoice array that are being queried and displayed

            Console.WriteLine("C) Select Part Description and Quantity, and sorted by Quantity: ");

        var sortedQuantity = from query in invoices

                             orderby query.Quantity

                             select new { query.Quantity, query.PartDescription };

        //The foreach runs through the sortedQuantity query and finds each of the desired rows and then displays them

        //e is a variable for the items in the query which are being displayed

            foreach (var e in sortedQuantity)

            {

                Console.WriteLine(" {0}", e);

            }

            //Used to create a line of seperation between each result set

            Console.WriteLine();

//-------------------------------------------------------------------------------------------------------------------------------------------------------------

            //d) Use LINQ to select from each Invoice the PartDescription and the value of the Invoice Name the calculated column InvoiceTotal. Order the results by Invoice value.

            //PRE: The invoice array must be fully populated with correct information for all fields

            //POST: The Invoice value must be calculated correctly (quantity \* Price) and the results will display

                //Part description and value being ordered by value

            Console.WriteLine("D) Select Part Description and Value, and sorted by Invoice value: ");

            var sortedValue = from query in invoices

                                 let InvoiceTotal = (query.Quantity \* query.Price)

                                 orderby InvoiceTotal

                                 select new { InvoiceTotal = InvoiceTotal, query.PartDescription };

            //The foreach runs through the sortedValue query and finds each of the desired rows and then displays them

            //e is a variable for the items in the query which are being displayed

            foreach (var e in sortedValue)

            {

                Console.WriteLine(" {0}", e);

            }

            //Used to create a line of seperation between each result set

            Console.WriteLine();

//-----------------------------------------------------------------------------------------------------------------------------------------------------------

            //e) Using the results of the LINQ query in Part d, select the InvoiceTotals in the range $200 to $500.

                //PRE: The invoice array must be created and populated and the sortedValue query is completed correctly

                //POST: The sortInvoiceTotals query displays the values in the rang of 200-500 and displays them correctly

                    //sorted by the value from lowest to highest

            Console.WriteLine("E) Select Invoice totals from part D in the range of $200 - $500");

            var sortInvoiceTotals = from query in sortedValue

                                    orderby query.InvoiceTotal

                                    where (query.InvoiceTotal >= 200 && query.InvoiceTotal <= 500)

                                    select new { query.InvoiceTotal, query.PartDescription };

            //The foreach runs through the sortedInvoiceTotals query and finds each of the desired rows and then displays them

            //e is a variable for the items in the query which are being displayed

            foreach (var e in sortInvoiceTotals)

            {

                Console.WriteLine(" {0}", e);

            }

            //Used to create a line of seperation between each result set

            Console.WriteLine();

            Console.ReadLine();

1. **Skills Set Sheet**

Student Name: Jacob Ellis Term: Fall, 2012

Portfolio Image: Evaluation:

Portfolio Review: Jacob Ellis organized his portfolio to demonstrate knowledge and skills in three areas: Business and Systems Analysis; Project Management; Database; and Programming. He responded to all questions directly. He was open to suggestions throughout the review.

|  |  |  |
| --- | --- | --- |
| Portfolio Learning Outcome | Included in Portfolio  (note if not applicable) | Identification and/or Description of Appropriate Items |
| Analytical skills – problem solving. | yes or no;  somewhat;  not explicitly |  |
| CIS technical knowledge – web page development, relational database, object-oriented programming, telecommunications, systems analysis and design methodology. | yes or no;  somewhat;  not explicitly |  |
| CIS technical skills – HTML, Java programming, systems analysis and design modeling, relational database management software use, development tool use (ColdFusion, ASP, VB, etc.) | yes or no;  somewhat;  not explicitly |  |
| Communication skills – technical or business writing, oral presentation, face-to-face communication, and listening. | yes or no;  somewhat;  not explicitly |  |
| Team building – included The Strength Assessment, which is a Gallup Poll Study from the book Now, Discover Your Strengths. | yes or no;  somewhat;  not explicitly |  |
| Initiative – leadership or another means of beginning or following through with a plan of activity | yes or no;  somewhat;  not explicitly | [XXX’ role as team leader is not evident. This is one of his/her strengths, and its inclusion would improve his portfolio]. |