Fall 2015 CIS605 Semester-Long Project

Theme Park Management System

Overview

This is a semester-long project, with parts of the project due at various points during the term. The code and user-interface design will be looked at fairly closely in addition to checking for proper functioning of the running system.

The four deliverables, their due dates, and their brief coverage are summarized in the table below:

	Deliverable	Due	Points
1	User Interface	September 17	25
2	Class Design + Basic Code + Input Validation	October 8	50
3	Custom Events + hard coded test data	November 17	75
4	Full System with Arrays + File I/O	December 10	100
			250

For this project you will be developing a system in Visual Basic .Net, using Visual Studio 2012, 2013, or 2015, which implements some of the functionality that might exist with theme park management. Some functionality and data elements have been simplified in order to adequately scope the project for completion during the semester. As the semester progresses, you will be given more details on the project

The basic function of this system is to allow the purchase and use of theme park features. By the fourth project submission, you will have a completed working system that manages customers, features, and many logistics of theme park tracking.

The following use case diagram summarizes the various actors that may use the system. You do NOT need to create separate applications for each. In addition, you do NOT need to worry about user access, privileges, or passwords. You may assume that all actors will use the same program and that each actor will only use the parts of the UI that are applicable to them.

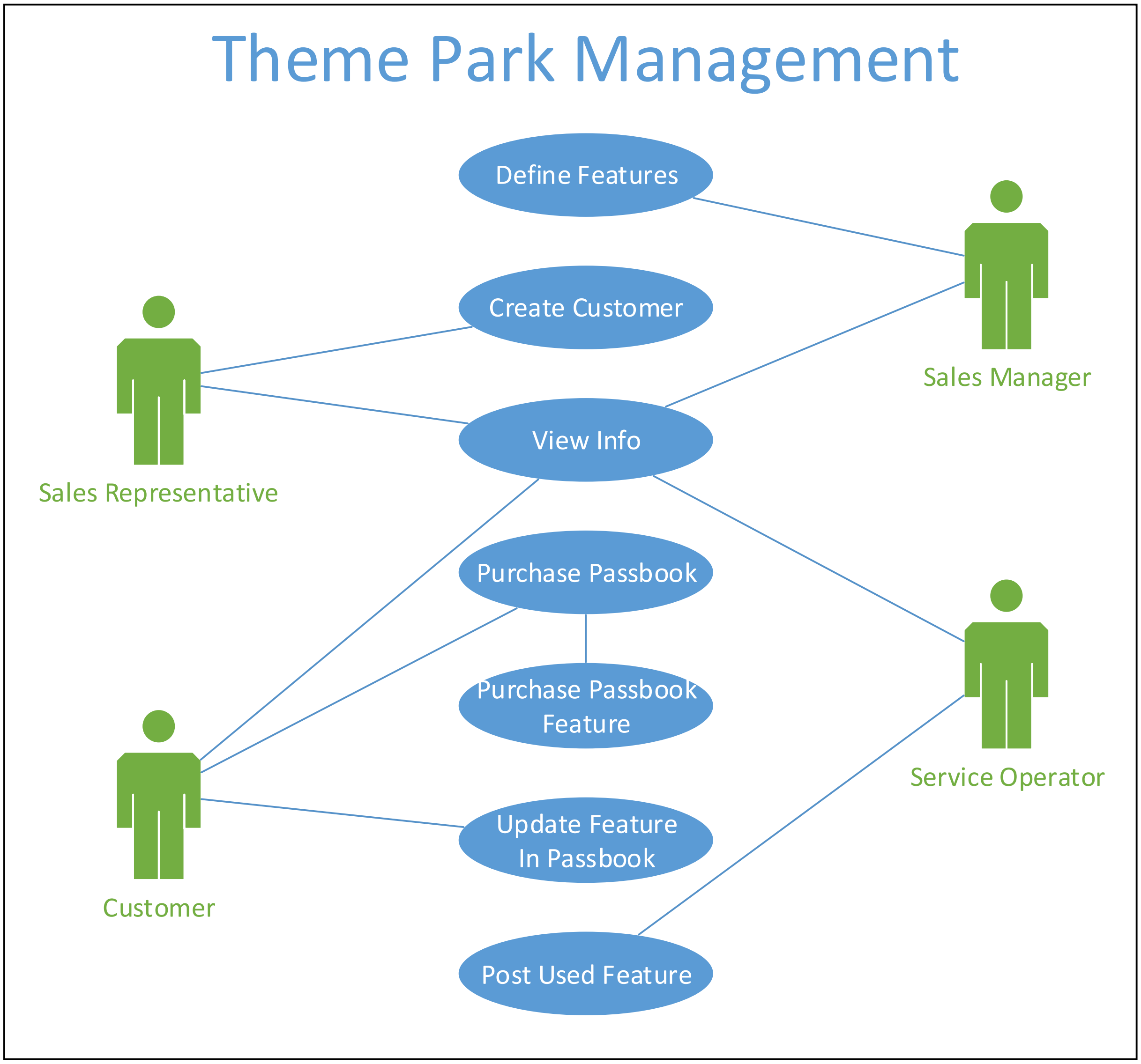


Figure 1: UML Use Case Diagram

The following high level use cases will help you to understand the user requirements better. This list represents functionality for the final project submission, and will be subject to change throughout the semester. Not all requirements are expected until the end of the semester; however, they are provided up front to keep you from going astray early. Carefully read each project assignment to understand the specific functionality required for that particular assignment. **NOTE: some modifications/refinements will likely be made throughout the semester as the project matures.**

<table>
<tr><td colspan="2">Use Case #1: Define a Feature</td></tr>
<tr><td>Description</td><td>A Sales Manager will periodically add features (services) that are sold to customers.</td></tr>
<tr><td>Main Success Scenario</td><td>Manager enters an alphanumeric identifier that is assigned externally and the feature name, a unit of measure (how the service is sold), and the price for Adults and Children separately.   To simplify the project, you can assume that the price will never change and no coupons or discounts will ever be used. You can also assume that features are never removed. The data is confirmed and added to the system.</td></tr>
<tr><td>Other</td><td>
<ul>
<li>Identifier must be unique.</li>
<li>Feature name and Unit of Measure must be a free form text entry.</li>
<li>Prices must be decimals.</li>
<li>Data Examples:</li>
</ul>
<table>
<tr><th>Feature Name</th><th>Unit of Measure</th><th>Price (Adult/Child)</th></tr>
<tr><td>Park Pass</td><td>Day</td><td>$100 / $80</td></tr>
<tr><td>Parking Lot Pass</td><td>Day</td><td>$15 / $15</td></tr>
<tr><td>Meal Plan</td><td>Meal</td><td>$30 / $20</td></tr>
<tr><td>Early Entry Pass</td><td>Day</td><td>$10 / $5</td></tr>
<tr><td>VIP Pass (cut in line at attraction)</td><td>Attraction</td><td>$...</td></tr>
<tr><td>Birthday Gift Package</td><td>Each</td><td>$...</td></tr>
<tr><td colspan="2">Etc… Free form text should allow any number of services and units.</td><td>$...</td></tr>
</table>
</td></tr>
</table>

Use Case #2: Create Customer	
Description	A Sales Rep will create a Customer when they purchase their first Passbook. The Customer is typically the head of the household or the trip planner. Only an ID and the customer name is required. (Other typical fields such as address, email, etc… are not required for this project)
Main Success Scenario	Manager enters an alphanumeric utility identifier for the customer that is assigned externally and the name of the customer. The data is confirmed and added to the system.
Other	• Identifier must be unique • Customer name must be free form text entry. (Given name and surname should be combined into just one entry for simplicity)

Use Case #3: Purchase Passbook	
Description	Customers can create any number of passbooks which will hold all the features that they buy. One passbook is created per visitor related to the customer. For example, one customer may be Joe with three of his passbooks belonging to Joe, Jen, and Jack. The Passbook is given an ID, a reference to the owner (by choosing the owner in a dropdown list), the date that the passbook was purchased (always defaults to the system date), the name of the person belonging to the passport, that person's birthdate. The birthdate is then used to calculate an age based on the system date and a determination if the visitor is an adult or a child. A child is defined as anyone under the age of 13.
Main Success Scenario	An alphanumeric number that is assigned externally is entered. A dropdown list is provided of customers and one is chosen as the person purchasing the passbook. A textbox should be provided to enter the visitor's name (given and surname can be combined into one field). A control should also be used to enter the visitor's birthdate. The data is confirmed and the passbook is added to the system.
Other	• Identifier must be unique. • The customer must already exist in the system. • Date Purchased does not need to be entered – this can default to the current system date. • Birthdate must be entered. Age and IsChild must be calculated based on the birthdate entered and the current system date.

Use Case #4: Purchase Passbook Feature	
Description	The Customer must be able to buy features and apply them to their passbooks. Customers can buy any quantity of any given feature. Examples of features are provided in Use Case #1. The amount should be calculated and stored in the object. The amount is the price of the feature (based on adult vs child prices of the passbook visitor's IsChild status) x quantity purchased
Main Success Scenario	An alphanumeric passbook feature ID that is assigned externally is entered. The customer selects the Passbook and Feature based on drop down lists. When a passbook is selected, the user should be able to see and verify the passbook visitor name, visitor age, IsChild status, and the customer name (which could be different than the visitor name). When a feature is selected, the user should be able to see and verify the feature name, unit of measure, and correct price of the feature based on the visitor's IsChild status. A quantity is then entered and the transaction is validated and added to the system.
Other	• Identifier must be unique. • The customer and the feature must already exist in the system. • Units must be numeric (decimal). • Quantity purchased should never be negative. • Price of the feature must match the visitor's IsChild status

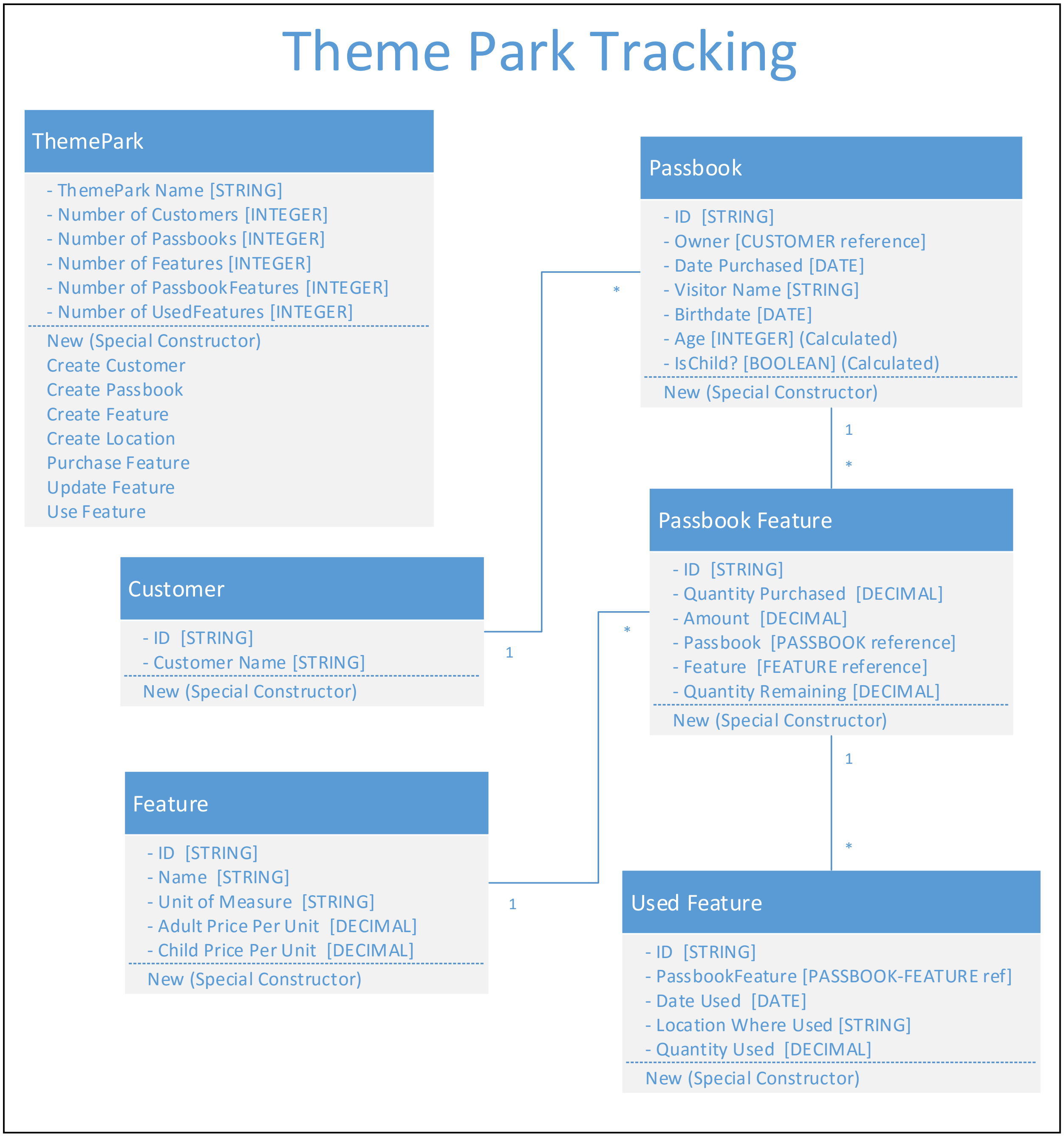
Use Case #5: Update a Passbook Feature	
Description	The customer must be able to increase or decrease any given feature quantity. For example, a customer may have originally purchased 4 days of park entrance feature, but would like to add 2 more for a total of 6.
Main Success Scenario	The sales rep enters or selects the alphanumeric ID that is assigned to the passbook feature being updated. When the ID is selected, the user should see passbook and customer information, feature information, units remaining, expiration date, and a list of all used features (if any). The sales rep then enters the new amount of total features (not the delta). The update textbox should default to the original quantity purchased. The data is validated and changed in the system: amount and quantity are adjusted.
Other	• Identifier must exist. • Units must be numeric (decimal). • Quantity purchased should never be negative. • Price adjustments must consider the visitor's IsChild status

Use Case #6: Post a Used Feature	
Description	The Park Employees must be able to register the use of a feature. For example, when the visitor arrives at the park, the system is checked that the Park Ticket feature is in the Passbook and that sufficient quantity exist to allow entrance. The location where the feature was used is indicated in a free-form text field along with the quantity used. Quantities could be decimal. For example, late arriving guests may only be charged a half day of park ticket. Or, a lunch may only be a half quantity of a meal ticket whereas dinner may be a full quantity. This does not need to be kept in the system, you can assume the employee will know and enter the correct quantity used.
Main Success Scenario	The employee enters an externally assigned alphanumeric ID that identifies the usage of the entitlement. The employee selects the Passbook and Passbook Feature being used. After the employee selects this information, they should see visitor information, feature information, and the number of features remaining. The employee then enters the number used. Finally, the employee enters the location where the feature was used (e.g. "The 80's Diner" or "Parking Lot A" or "Super Rollercoaster"): this field is always free form text. The date used always is set to the current date (no opportunity to modify it). After the employee enters all the information, the system should check if enough quantity exist to allow usage of the feature and then post the transaction.
Other	• Identifier must exist. • Rejected requests (not enough features left) should display an error. • Approved requests should automatically decrement the number of features left remaining.

Use Case #7: View Info and KPI's (Key Performance Indicators)	
Description	All users need to see key indicators and transaction logs.
Main Success Scenario	The user views correct calculations for various types of information. These calculations will be provided as the project progresses. The user also needs to see scrollable lists of all data in the system: customers, services, and entitlements. Finally, the user needs to see a transaction log: a scrollable list of all transactions that have occurred in the system, as they occur.
Other	• Calculations should be accurate at all times. • Be careful to not divide by zero when calculating averages. • More details on this requirement will be provided later.

Use Case #8: Test Button; Read / Write Files	
Description	The system should have a hardcoded test data button. The system must be able to import data files and export backups.
Main Success Scenario	Users should be able to load transactions from 1) UI per previous requirements, 2) hard coded data in a test button, 3) flat data files. Users should also be able to export all transactions in a flat data file.
Other	• Not required for Project 1 • Specific file formats will be provided later. • More details on this requirement will also be provided later.

The following **UPDATED** UML Class Diagram summarizes this structural information and the interrelationships between Classes. All data will be stored in memory in the application (i.e. no database management systems will be used). More details will be provided on the business logic classes as the project progresses, and the UML Diagram will be refined over the duration of the course.



Theme Park Tracking

ThemePark

- ThemePark Name [STRING]
- Number of Customers [INTEGER]
- Number of Passbooks [INTEGER]
- Number of Features [INTEGER]
- Number of PassbookFeatures [INTEGER]
- Number of UsedFeatures [INTEGER]

New (Special Constructor)
Create Customer
Create Passbook
Create Feature
Create Location
Purchase Feature
Update Feature
Use Feature

Passbook

- ID [STRING]
- Owner [CUSTOMER reference]
- Date Purchased [DATE]
- Visitor Name [STRING]
- Birthdate [DATE]
- Age [INTEGER] (Calculated)
- IsChild? [BOOLEAN] (Calculated)

New (Special Constructor)

Customer

- ID [STRING]
- Customer Name [STRING]

New (Special Constructor)

Passbook Feature

- ID [STRING]
- Quantity Purchased [DECIMAL]
- Amount [DECIMAL]
- Passbook [PASSBOOK reference]
- Feature [FEATURE reference]
- Quantity Remaining [DECIMAL]

New (Special Constructor)

Feature

- ID [STRING]
- Name [STRING]
- Unit of Measure [STRING]
- Adult Price Per Unit [DECIMAL]
- Child Price Per Unit [DECIMAL]

New (Special Constructor)

Used Feature

- ID [STRING]
- PassbookFeature [PASSBOOK-FEATURE ref]
- Date Used [DATE]
- Location Where Used [STRING]
- Quantity Used [DECIMAL]

New (Special Constructor)

Figure: UML Class Diagram

Semester Project: Deliverable 2 – Class Design & Initial Code

Due: Thursday, October 8, 2015 at 11:59 PM Mountain Time

Points: 50

Solution Name: Proj02-LLLL-FFFF (LLLL = your last name, FFFF = your first name)

Project Name: ThemePark

Copy your Proj01 directory and make a new copy in a directory called Proj02-LLLL-FFFF. Then rename the .sln and .suo files within it (to say Proj02 instead of Proj01) and then double click on the .sln file to open the new copy of your Solution. Do the same each time you start working on a new part of the project: Proj02, Proj03, Proj04.

For this increment of the project, you will be doing your Business Logic class design and developing some basic Business Logic class functionality by writing initial code for these classes. You will also be updating your User Interface design (if/as needed) and developing initial User Interface "functionality" by adding code for your UI class.

First, create business logic classes for all the classes listed in the class diagram in the project brief, and for a **ThemePark class (as updated in the UML class diagram)** using the standard naming conventions and software development practices that have been specified for this semester.

For each business logic class, be sure to include:

1. Attributes, per the Class Diagram.
2. Public and private property procedures, per the standard approach described in lectures.
3. Public and Private ToString methods **(even though these are not shown in the diagram)**.
4. Special constructor, with parameters for each of the attributes needed when the object is initially created.

For ThemePark class, be sure to include:

1. Attributes to count the number of each class created (Ex: Customer, Passbook, etc…), per the Class Diagram.
2. Public and private property procedures, per the Class Diagram.
3. Procedures to create each of the class objects.
4. Within each procedure, increment the appropriate count by 1 and create the appropriate class object by using the New method to invoke the special constructors written in #4 above.
5. Public and Private ToString methods **(even though these are not shown in the diagram)**.
6. Special constructor, with parameters for the attribute needed when a ThemePark object is initially created.

Write code in the main form that does basic functionality:

1. The initialize procedures
2. Exit button event
3. Transaction log scrolling
4. Each “Create” button click event:
    a. Validates user input if necessary (avoiding run time errors)
    b. Calls the appropriate Create procedure in the ThemePark class and passes the parameters necessary for that procedure to create the business object.
    c. Add a message to the transaction log.

Note: the “Update Feature” procedure does not need to be implemented for Project 2.

Code the button click event for “Test Data” to add hardcoded test data from within the application. The event will also call the same Create procedures in ThemePark and passing hard coded test data as parameters.

Finally, clean up any errors from Project 1, including tab stops, ToolTips, keyboard shortcuts, naming standards, and any other feedback that was given to you. Refer to the provided screen shots (i.e. the “Project 1 Answer Key”) to help you align on the requirements. Remember, you have some creative latitude – the UI does not have to match exactly, but the basic requirements should all be covered in some form or fashion.

A sample rubric for Project 2 is listed below.

CIS605 Project 2

Name: Sample Rubric

* Note: Feedback may refer to a specific code line number ("LN"). To view line numbers in Visual Studio, navigate as follows: (Main Menu) > Tools > Options > (Options Box) > Text Editor > All Languages > check "Line Numbers"

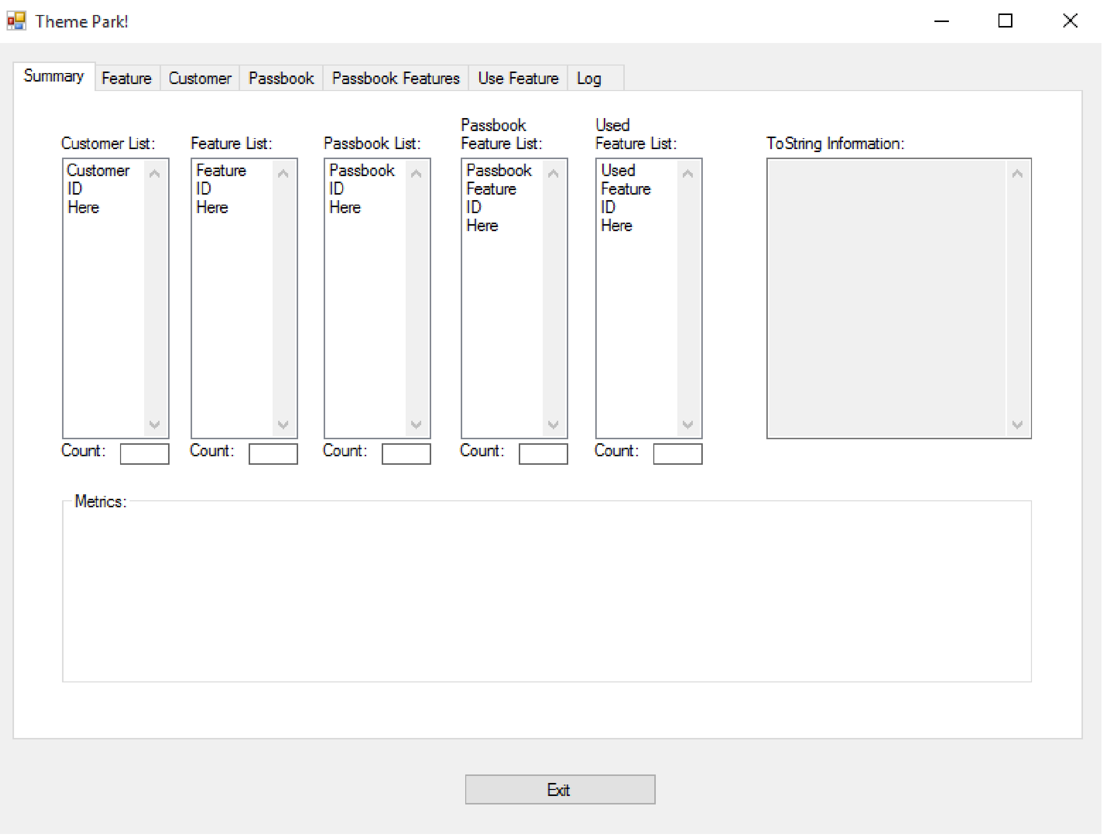
Topic	**Expectations**	**Points Possible**	**Points Received**	**Specific Feedback**
General Functionality *(applicable to all problem sets)*				
Zip File, Solution, Projects	Program should be zipped properly and include the solution and all the applicable file(s)	Mandatory to receive a grade		
Projects compile & bug-free	Each applicable project should compile cleanly (no compile errors, but compile warnings are sometimes okay)	-20% off of final point total		
	Program should not crash while running			
Flagrant errors impacting General Aesthetics or Supportability	Additional points may be deducted for errors such as no regard for naming convention, no comments, unusable interface, major spelling/grammar issues, or no template (among others).	Deduction assessed depending on severity		
General Aesthetics *(applicable to all problem sets)*				
Look and Feel	UI elements align properly on form	5		
	UI elements appropriately sized/spaced			
	Good creativity while maintaining professionalism			
	User friendly captions and messages with proper spelling/grammar			
	Understandable ToolTips as appropriate			
	Message log entries scroll and display the most recent entry			
	Justification as appropriate (e.g. numbers are right justified)			
Usability	All tab stops correct			
	Keyboard shortcuts as appropriate			
	Appropriate Accept/Cancel functionality			
	Cursor reset to fix user input errors			
	Form reset after valid input			

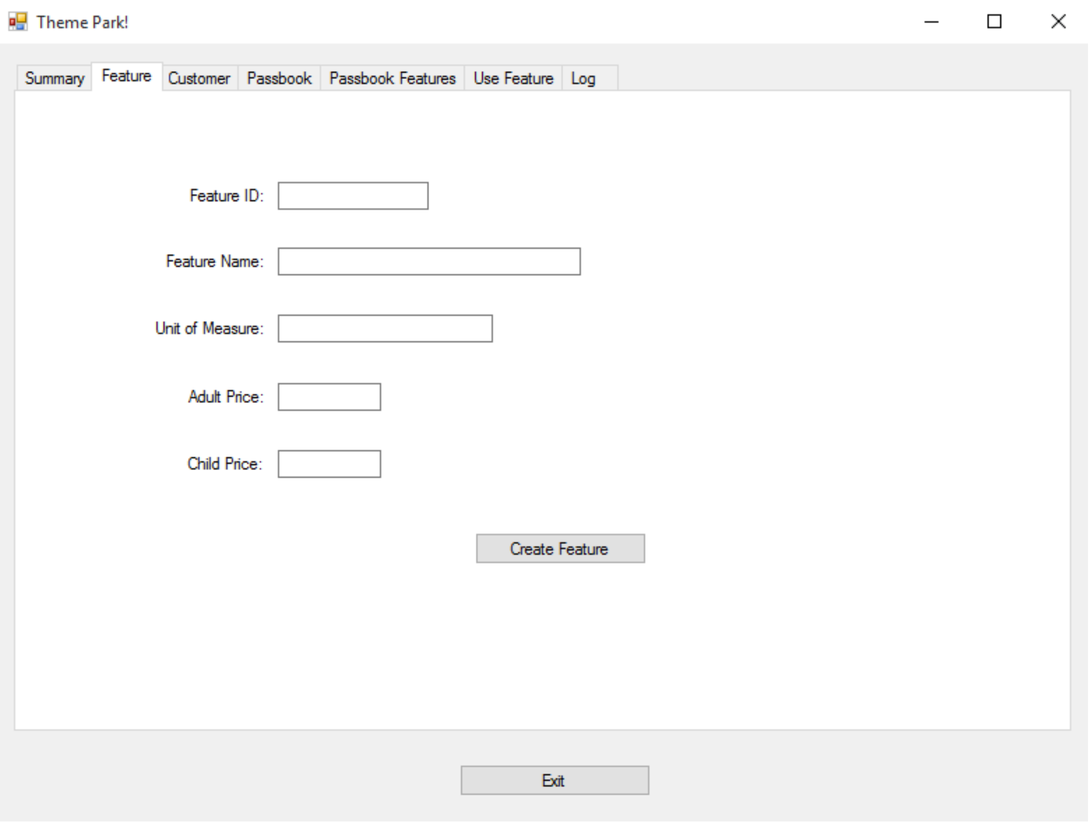
General Supportability *(applicable to all problem sets)*				
Template & Header	Official class template used in all files	5		
	Header completed for all files (LNs 3-15)			
	All code is in the proper template sections			
	Empty procedures are removed			
Internal Comments	All comments use clear business terms			
	Method comments exist			
	Section comments exist where longer blocks of code would benefit from them			
	Line-by-line code for very technical operations or where the code is not self-explanatory			
	End statements closed with a comment			
Naming convention	FrmMain and all other code files are named properly			
	UI elements are prefixed correctly and named in clear business terms (exception: elements not used in code, like many lables)			
	Variables and parameters are prefixed correctly and named in clear business terms			
	Methods and properties are named correctly			
Required Methods	_initializeUserInterface			
	_initializeBusinessLogic			
	ToString private and public override in classes			
Code Style	White space used effectively			
	Good logical blocks of code (e.g. local variables defined all together)			
	Good separation of UI, Business Logic, and Data functions			
Required End User Functionality (Graded as the project is running)				
RUN TIME	Data entry validated to prevent run time errors	10		
	"Add" buttons create correct output in Transaction Log			
	"Test Data" button creates correct output in Transaction Log			
	Exit button works			

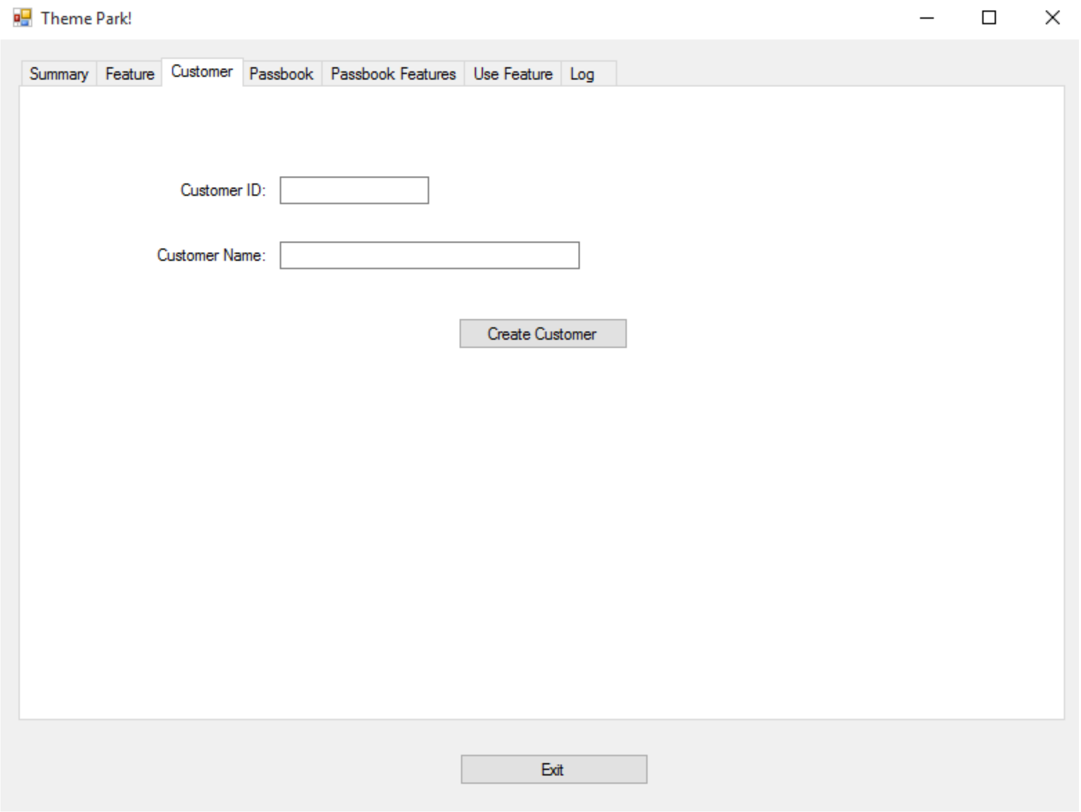
Required Code Elements (Graded as a code review)				
FrmMain	Module level attribute and private property for ParkSystem	8		
	Proper variable definition / scope			
	Click events for Create buttons			
	Pass data to business logic classes			
	Populate Transaction Log			
	Click event for "Test Data" button using hardcoded data			
ThemePark	Attributes	10		
	Public Properties			
	Private Properties			
	Create Procs for each Class			
	Private/Public ToString			
Classes	**Following Classes exist:**	12		
	ParkSystem			
	Customer			
	Passbook			
	Feature			
	PassbookFeature			
	UsedFeature			
	For each class (refer to Class Diagrams):			
	Attributes			
	Special Constructor			
	Public Properties			
	Private Properties			
	Private/Public ToString			
Overall Feedback and TOTAL				
Sample Rubric		**50**	**0**	

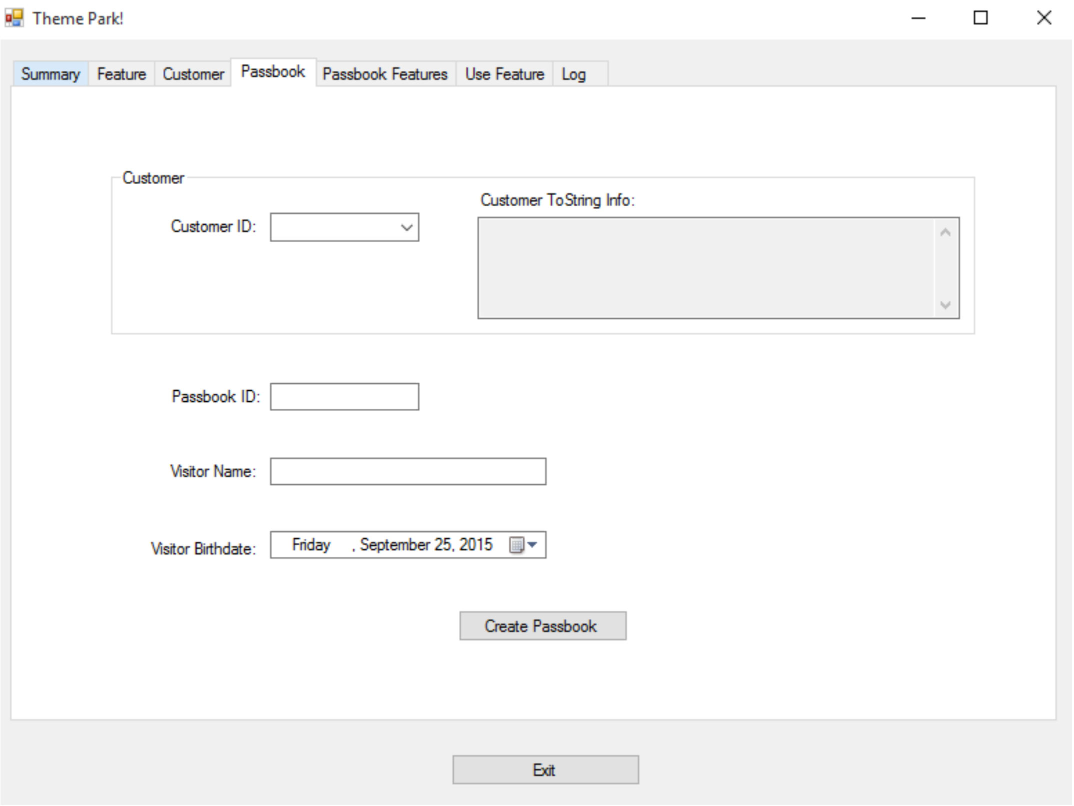
Screen Shots

Below are some sample screen shots on how the UI might be configured. The form is bare-bones and could be significantly improved upon with added professionalism. Nevertheless, it shows the basics and should give you some ideas on how to reconfigure your project one in some instances.

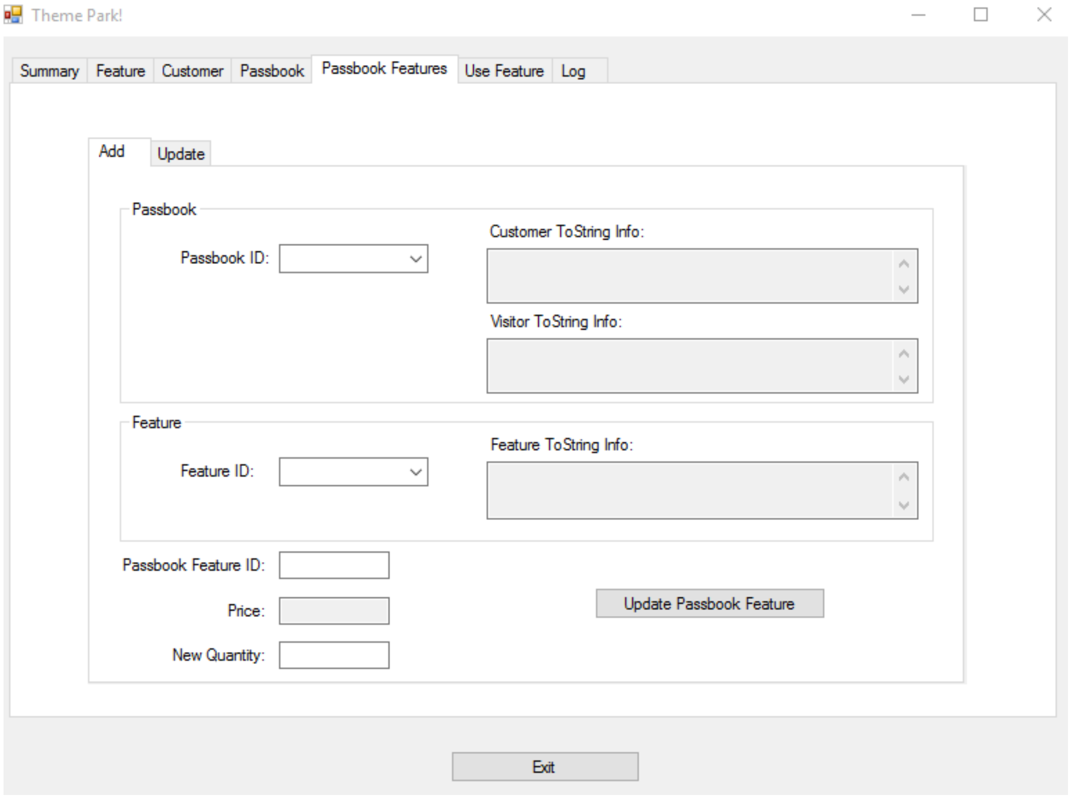

Theme Park!
Summary
Feature
Customer
Passbook
Passbook Features
Use Feature
Log
Customer List:
Feature List:
Passbook List:
Passbook Feature List:
Used Feature List:
ToString Information:
Customer ID Here
Feature ID Here
Passbook ID Here
Passbook Feature ID Here
Used Feature ID Here
Count:
Count:
Count:
Count:
Count:
Metrics:
Exit

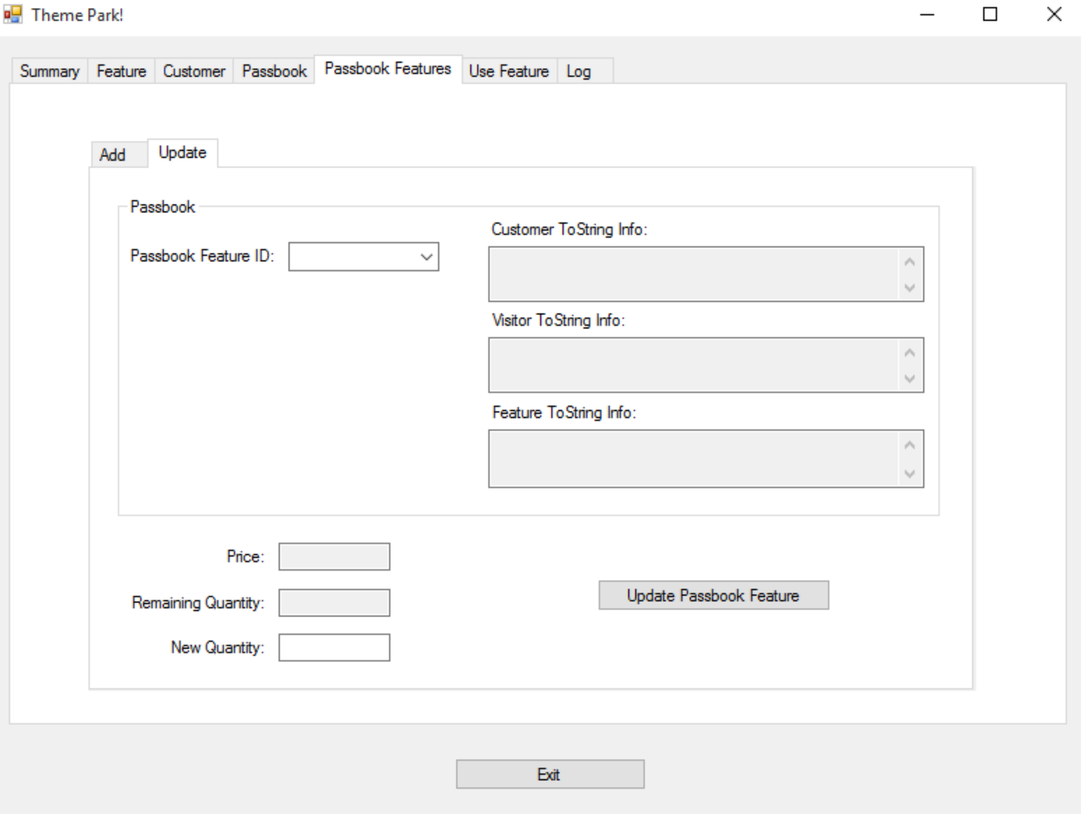

Theme Park!
Summary
Feature
Customer
Passbook
Passbook Features
Use Feature
Log
Feature ID:
Feature Name:
Unit of Measure:
Adult Price:
Child Price:
Create Feature
Exit

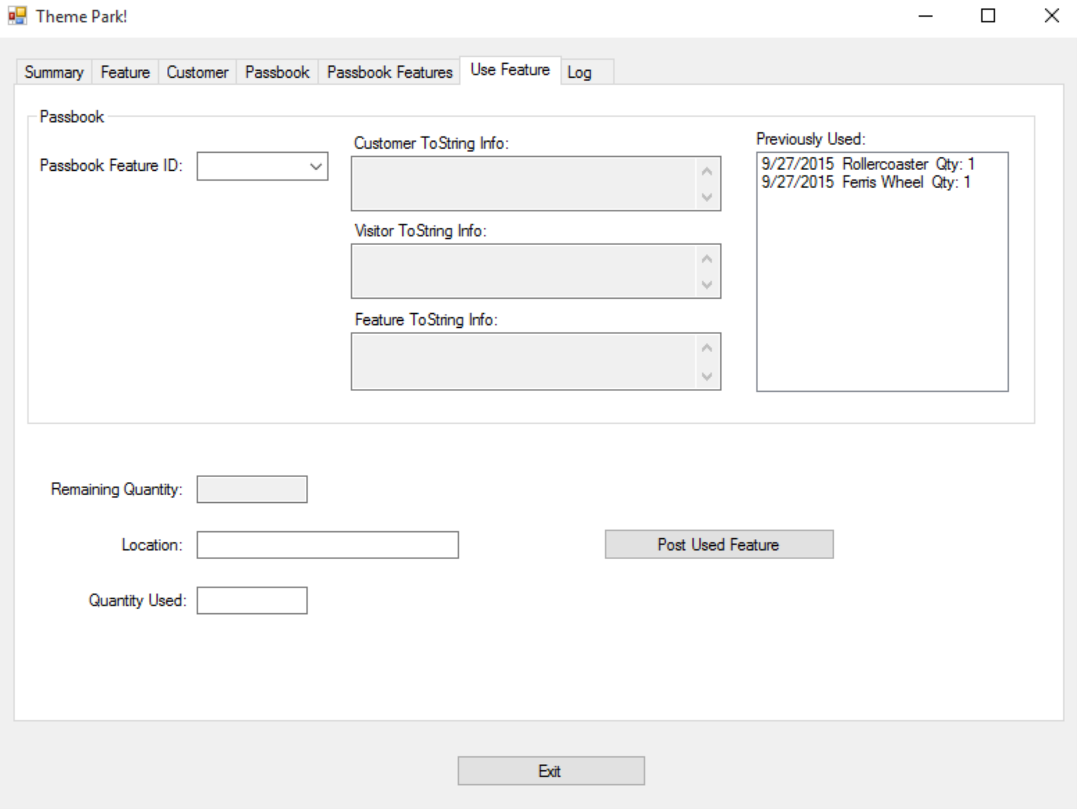




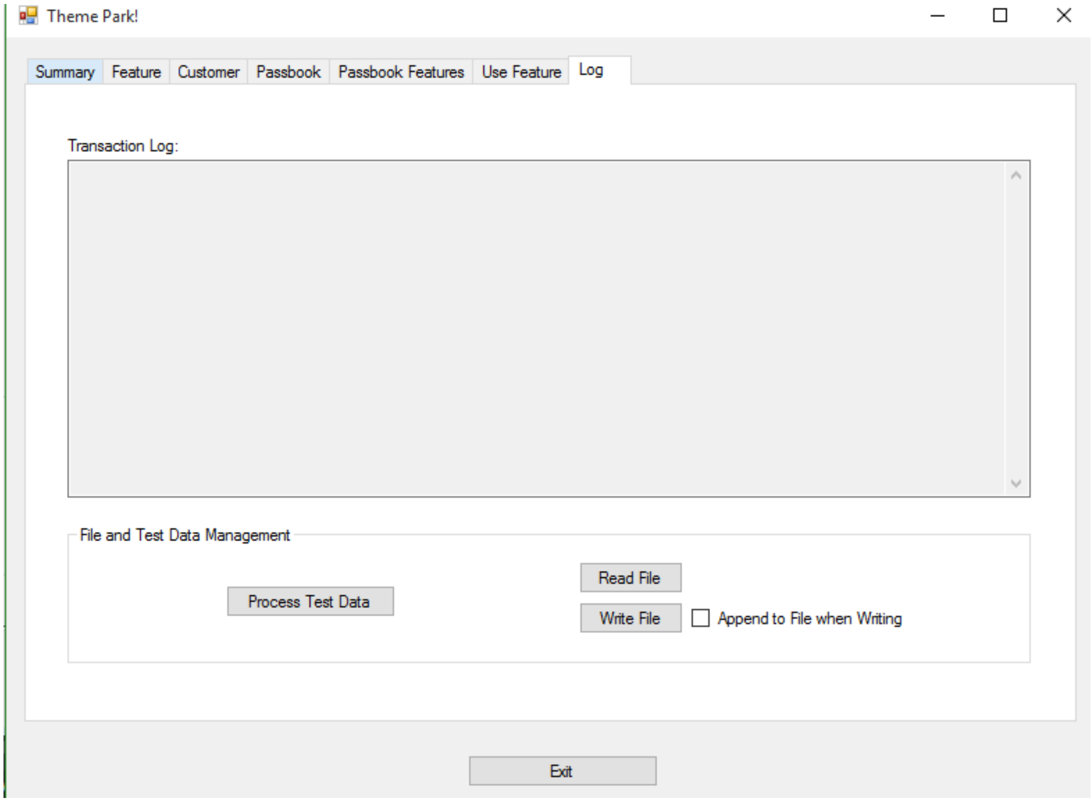
Visitor Birthdate: Friday , September 25, 2015


Theme Park!
Summary
Feature
Customer
Passbook
Passbook Features
Use Feature
Log
Add
Update
Passbook
Passbook ID:
Customer ToString Info:
Visitor ToString Info:
Feature
Feature ID:
Feature ToString Info:
Passbook Feature ID:
Price:
New Quantity:
Update Passbook Feature
Exit


Theme Park!
Summary
Feature
Customer
Passbook
Passbook Features
Use Feature
Log
Add
Update
Passbook
Passbook Feature ID:
Customer ToString Info:
Visitor ToString Info:
Feature ToString Info:
Price:
Remaining Quantity:
New Quantity:
Update Passbook Feature
Exit



```
9/27/2015  Rollercoaster  Qty: 1
9/27/2015  Ferris Wheel  Qty: 1
```


Theme Park!
Summary
Feature
Customer
Passbook
Passbook Features
Use Feature
Log
Transaction Log:
File and Test Data Management
Process Test Data
Read File
Write File
□ Append to File when Writing
Exit