

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 + hard coded test data	October 8	50
3	Custom Events + More code + hard coded test data	November 12	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.**

Use Case #1: Define a Feature																										
Description	A Sales Manager will periodically add features (services) that are sold to customers.																									
Main Success Scenario	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.																									
Other	<ul style="list-style-type: none"> • Identifier must be unique. • Feature name and Unit of Measure must be a free form text entry. • Prices must be decimals. • Data Examples: <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>		Feature Name	Unit of Measure	Price (Adult/Child)	Park Pass	Day	\$100 / \$80	Parking Lot Pass	Day	\$15 / \$15	Meal Plan	Meal	\$30 / \$20	Early Entry Pass	Day	\$10 / \$5	VIP Pass (cut in line at attraction)	Attraction	\$...	Birthday Gift Package	Each	\$...	Etc... Free form text should allow any number of services and units.		\$...
Feature Name	Unit of Measure	Price (Adult/Child)																								
Park Pass	Day	\$100 / \$80																								
Parking Lot Pass	Day	\$15 / \$15																								
Meal Plan	Meal	\$30 / \$20																								
Early Entry Pass	Day	\$10 / \$5																								
VIP Pass (cut in line at attraction)	Attraction	\$...																								
Birthday Gift Package	Each	\$...																								
Etc... Free form text should allow any number of services and units.		\$...																								

Use Case #2: Create Customer	
Description	A Sales Rep will create a Customer before 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Identifier must be unique. • The customer must already exist in the system. • Date Purchased does not need to be entered – in the GUI this can default to the current system date; in hard-coded test data this can be specified. • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<p>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).</p> <p>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.</p>
Other	<ul style="list-style-type: none"> • 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	<p>The user views correct calculations for various types of information. These calculations will be provided as the project progresses.</p> <p>The user also needs to see scrollable lists of all data in the system: customers, services, and entitlements.</p> <p>Finally, the user needs to see a transaction log: a scrollable list of all transactions that have occurred in the system, as they occur.</p>
Other	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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.

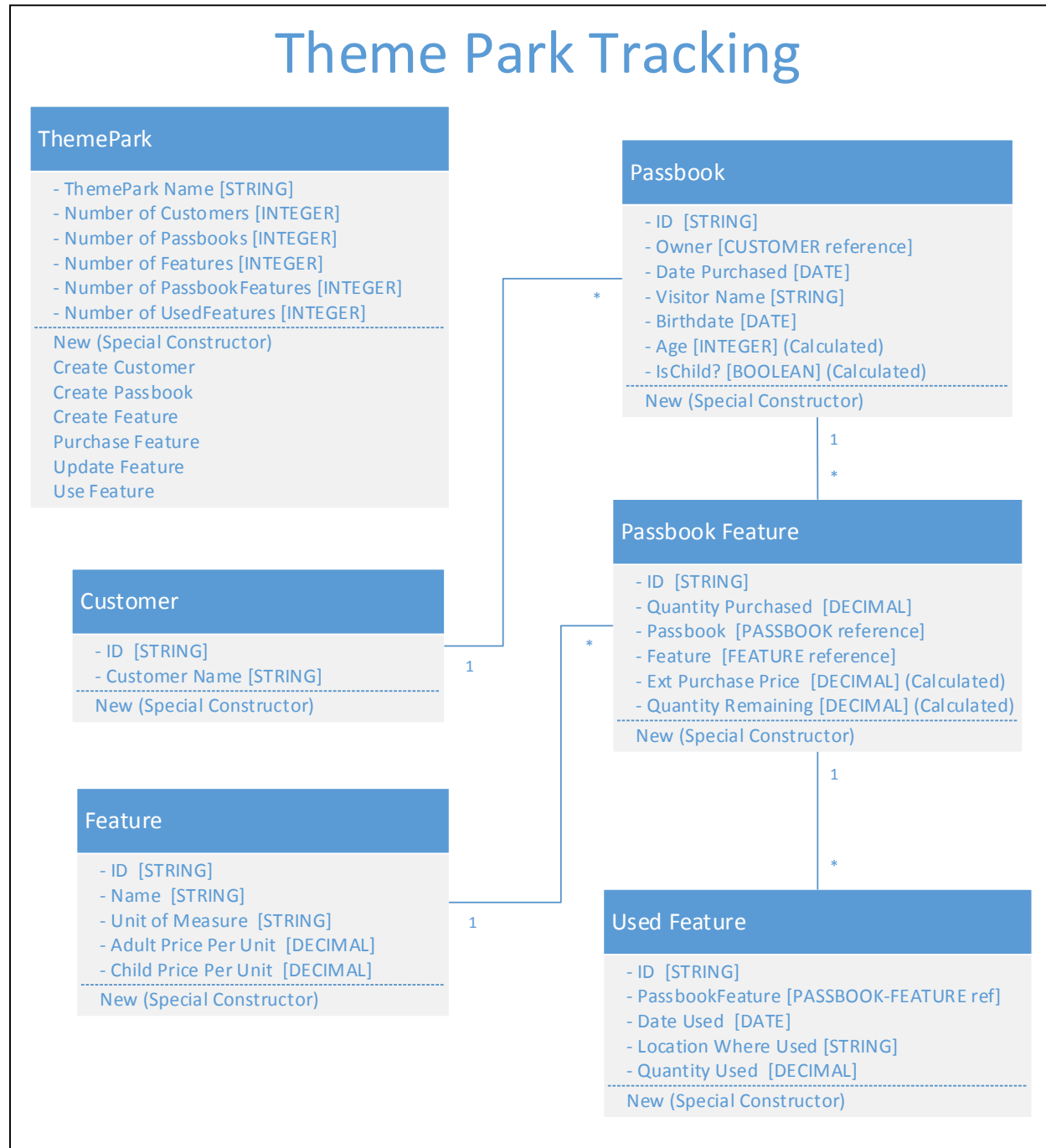


Figure: UML Class Diagram

Semester Project: Deliverable 3 – Custom Events, Etc.

Due: Thursday, November 12, 2015 at 11:59 PM Mountain Time

Points: 75

Solution Name: Proj03-LLLL-FFFF (LLLL = your last name, FFFF = your first name)

Project Name: ThemePark

Copy your Proj02 directory and make a new copy in a directory called Proj03-LLLL-FFFF. Then rename the .sln and .suo files within it (to say Proj03 instead of Proj02) 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 continue building business logic behaviors, including custom events, for the following:

1. Create Feature
2. Create Customer
3. Create Passbook
4. Purchase Feature (i.e. Create PassbookFeature)
5. Use Feature (i.e. Create UsedFeature)
6. Update Passbook Feature

For each business logic behavior, be sure to include:

1. Implement a behavioral method in the ThemePark class with the appropriate parameters. (In general the create methods should have parameters for each of the attributes that will be created in the object, and should return a reference to the created object, the purchase, use, and update methods should have the relevant ID(s) passed to them so that in Project04 we can look up the actual information. For example, in createCustomer() you should pass the ID and name, not first create a Customer object and pass the reference to it. In Project 04 we will be able to look things up internally within these behavioral methods to do some cross-referencing and error-checking that we can't do until we have learned about arrays and loops.)
 - a. Create the object.
 - b. Keep track of the "number of" objects created in the ThemePark class.
 - c. Raise a corresponding custom event from ThemePark.
 - d. Return a reference to the created object if successful, or Nothing if unsuccessful. ("Nothing" is the keyword in VB.Net that indicates "null", "nil", "reference/pointer to nothing", ...)
2. Implement a descriptively-named event procedure in FrmMain that responds to each custom event.

- a. Have all data presented in combo/list boxes and the transaction log populate from the custom events (not from the button clicks).
3. Further UI validation and feedback including:
 - a. Blank string values
 - b. Birthday's in future
 - c. When an item in a combo/list box is selected, information about that selection should be displayed in the proper label / textbox / list. (Eventually, after we learn about arrays and loops, for Project 04 we will be able to look up complete information and display it; for before that time we will only be able to display ID numbers and hard-coded messages.)
4. Populate the event data:
 - a. In the list boxes and labels on the Summary tab.
 - b. In the combo/list boxes on the appropriate functional tabs (e.g. Passbook, PassbookFeature, UsedFeature, ...)
 - c. As a record in the transaction log.

For "Update Feature": Create the procedure in ThemePark with the necessary parameters. This procedure will NOT need to increment any "NumberOf" variables. Also, for project 3, you will NOT need to actually update any data – because we will not, until Project04, be able to look up and find the PassbookFeature to actually update. But, you will have to raise a relevant custom event and update the transaction log with a temporary, hard-coded message.

Your behavioral methods should work when called in "batch" mode from the Process Test Data button, as well as when called "interactively" from the UI functional tabs. NOTE: we will still NOT store data long term in this iteration, so you will not be able to validate some interactive data that should be passed with object references. In these cases, you can elect to create a "dummy" reference to pass instead. You must follow the techniques and designs that we have been discussing in class.

Clean up any errors from Project 2 based on feedback that was given to you.

Make it so that any data entered from the GUI as well as data entered from your "process test data" method is tracked together. In other words, there should be a single module-level variable on your form for the main ThemePark. You could create additional local ThemePark variables if you wish, but for the test data given below it should update the same ThemePark as the GUI updates.

Test data to use in your "process test data" method is given below. By using this exact test data, we will be able to easily tell if your program is working properly without having to come up with our own test data when grading your assignment.

- Create Theme Park, ID "CIS605 Theme Park"
- Create Feature, ID "F01", Description "Park Pass", Units "Day", Adult Price \$100, Child Price \$80

- Create Feature, ID "F02", Description "Early Entry Pass", Units "Day", Adult Price \$10, Child Price \$5
- Create Feature, ID "F03", Description "Meal Plan", Units "Meal", Adult Price \$30, Child Price \$20
- Create Customer, ID "C01", Name "CName01"
- Create Customer, ID "C02", Name "CName02"
- Create Customer, ID "C03", Name "Customer Name 03"
- Create Passbook, ID "PB01", Customer "C01" reference, DatePurch 9/15/2015, Visitor Name "self", Visitor BDay 1/1/1980
- Create Passbook, ID "PB02", Customer "C02" reference, DatePurch 9/16/2015, Visitor Name "self", Visitor BDay 6/1/1985
- Create Passbook, ID "PB03", Customer "C02" reference, DatePurch 9/17/2015, Visitor Name "C02 Visitor", Visitor BDay 12/1/2003
- Create Passbook, ID "PB04", Customer "C03" reference, DatePurch 8/15/2015, Visitor Name "self", Visitor BDay 1/1/1975
- Create Passbook, ID "PB05", Customer "C03" reference, DatePurch 9/15/2015, Visitor Name "C03 Visitor 1", Visitor BDay 10/7/2002
- Create Passbook, ID "PB06", Customer "C03" reference, DatePurch 10/15/2015, Visitor Name "C03 Visitor 2", Visitor BDay 10/8/2002
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF01", Qty 1, Passbook "PB01" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF02", Qty 2, Passbook "PB02" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF03", Qty 3, Passbook "PB03" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF04", Qty 1, Passbook "PB04" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF05", Qty 1, Passbook "PB05" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF06", Qty 1, Passbook "PB06" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF07", Qty 3, Passbook "PB03" reference, Feature "F02" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF08", Qty 9, Passbook "PB03" reference, Feature "F03" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF09", Qty 1, Passbook "PB04" reference, Feature "F01" reference
- Purchase Feature (i.e. Create PassbookFeature), ID "PBF10", Qty 3, Passbook "PB04" reference, Feature "F01" reference
- Use Feature (i.e. Create UsedFeature), ID "UF01", PBFeature "PBF01" reference, DateUsed 10/20/2015, LocationUsed "Epcot Center", QtyUsed 1

- Use Feature (i.e. Create UsedFeature), ID “UF02”, PBFeature “PBF02” reference, DateUsed 20/20/2015, LocationUsed “West Parking”, QtyUsed 1
- Use Feature (i.e. Create UsedFeature), ID “UF03”, PBFeature “PBF03” reference, DateUsed 10/20/2015, LocationUsed “France”, QtyUsed 2
- Use Feature (i.e. Create UsedFeature), ID “UF04”, PBFeature “PBF03” reference, DateUsed 10/20/2015, LocationUsed “American Pavilion”, QtyUsed 1
- Update Passbook Feature, PBFeatureID “PBF03”, DateUpdated 10/21/2015, QtyUpdated 1

A sample rubric for Project 3 is listed below.

CIS605 Project 3

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		For all sections: · Fix any issues from previous project submissions. · Refresh comments for all code blocks that have changed. · Ensure best practices have been applied to all new content too. · Follow directions provided in the assignment and in class lectures.
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			
General Aesthetics (applicable to all problem sets)				
Look and Feel	UI elements align properly on form	7		
	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	8		
	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	Correct use of test data:	10		
	Button clicks on the content tabs			
	Process data button (hard coded tests)			
	Functional Tabs:			
	List boxes, combo boxes, populated with data consistent with test data and interactive data			
	Summary Tab:			
	List Boxes and Total counts should be populated.			
Required Code Elements (Graded as a code review)				
FrmMain	Button clicks:	12		

	Validate input fields with if/then and try/catch			
	Call behavioral methods in ThemePark			
	Updates to the form are made only by responding to custom events that are raised in the ThemePark behavioral methods.			
	Hard coded test data:			
	Appropriate objects used			
	Call behavioral methods in ThemePark			
	Updates to the form are made only by responding to custom events that are raised in the ThemePark behavioral methods.			
	Required Custom Event Handlers:			
	Listens for events from ThemePark			
	Adds information to all applicable lists, combos, textboxes, and labels			
	Updates the transaction log.			
ThemePark	Correct attributes and properties	10		
	Total counts for each object			
	Methods:			
	Specified business process methods			
	Creates other primary objects in the process methods (e.g. AddCustomer creates the Customer object)			
	Calculations as possible			
	Correct parameters and logic			
	Custom Events:			
	Use proper EventArgs			
	Fire at appropriate places			
Structured correctly				
Classes	Correct attributes and properties	10		
	Constructor			
	Behavioral Methods			
	ToString			
EventArgs Classes	Created for every data transaction	18		
	Inherits System.EventArgs			
	Correct attributes and properties			
	Constructor			
	ToString			
Overall Feedback and TOTAL				
Sample Rubric	75	0		