Getting Started with Banjos4Hire

Rob Miles

Department of Computer Science

Data Objects

- There are a number of objects that you will need to keep track of in the program
  - Banjo
  - Customer
  - Rental
- You can use the Account and Bank classes from the course and Yellow Book as good starting points
- Make sure that you have included all the required behaviours
- One way is to check them off the specification as you complete them

How to get started

- You don't create the entire application all in one
- The best approach is to build up the behaviours, one simple step at a time
- This is how we can create any kind of complex software system
- So, to start you must decide the sequence of items you are going to build

My Suggested Sequence

1. Create the Banjo class
2. Make the Load and Save behaviours for a single Banjo
3. Create the Shop class
4. Make the Load and Save behaviours for a Shop
5. Make the User Interface to find a particular Banjo in the Shop (use a ListView)
6. Make the User Interface to allow the Banjo to be manipulated
7. Repeat the process for Customer and Rental objects
8. Add the search behaviours

1. Create the Banjo Class

1. Read through the specification and identify the properties that the banjo should have:
   - A description of the banjo as a string of text
   - The price per day of hiring that banjo
   - The value of the banjo
   - The state of the banjo (either in the shop, being repaired or out on hire)
   - A unique banjo ID
2. Add these to the Banjo class that you are building
   - Create a constructor
3. Identify how the properties should be used
   - Create methods for each change

2. Make the Load and Save behaviours

1. Create a Save method that will save the Banjo to a stream
   - There is some sample code to do this in the lecture materials
2. Create a Load method that will fetch the Banjo from a stream
3. Create Load and Save methods that will work with files
4. Test these methods by loading and saving banjos

UNIVERSITY OF Hull

3. Create the Shop class

1. Create a Shop class that will contain everything in the shop
2. Initially just put a collection of banjos in the shop.
3. This can be a List or an array
4. Add methods to allow a program to add new banjos and search the shop for banjos
   - There is some sample code to do this in the lecture materials for Accounts

UNIVERSITY OF Hull

4. Make Save and Load behaviours for the Shop

1. Write the Save behaviour
   - To save the banjos the shop must write out the number of banjos and then write out each in turn
2. Write the Load behaviour
   - To load the banjos the shop must read in how many banjos and then read each one in
   - The Shop will use the Load behaviour from the Banjo to load each one
   - There is some sample code in the lecture notes

UNIVERSITY OF Hull

5. Make the User Interface to find a Banjo

1. Create the code to find a banjo
   - It can do this from the description of the banjo
   - The user will enter the search information and then trigger a search of the banjos
2. Populate the display with the banjo information when it is found
3. Alternatively you can use a ListView to show all the items and then detect when one is selected

6. Add the Banjo User Interface

1. Add TextBoxes and Buttons to allow the required functions to be performed
   - The functions will all be selected from the single page
   - This will keep the user interface simple
   - All the actions will be performed on the currently active banjo – the one that has been selected
2. Test each action, and make sure that the changes that are made are persisted in the stored version of the banjo

A Simple Banjo Class

```
class Banjo
{
    string Description;

    public Banjo(string inDescription)
    {
        Description = inDescription;
    }

    public override string ToString()
    {
        return Description ;
    }
}
```

- The `Banjo` class holds information about a banjo
- This version just holds the description

Banjos and Bank Accounts

- You can treat a Banjo just like we treat Bank Accounts
- You need to be able to add a banjo to the store and find a banjo by its ID (the banjo ID is just like an account number)
- You also need to save and load the banjo data to a file
- You can use the same save and load behaviours that you created for accounts

A Simple Shop Class

```
class Shop
{
    string ShopName;

    public List<Banjo> Banjos;

    public Shop(string inName)
    {
        ShopName = inName;
        Banjos = new List<Banjo>();
    }
}
```

- The Shop class holds all the shop information
- This includes a list of banjos

Making Test Data

```
public void MakeTestData()
{
    Random r = new Random(1);
    AddBanjo("The Strummer", r.Next(500, 10000));
    AddBanjo("The Woodsman", r.Next(500, 10000));
    AddBanjo("Deep Twang", r.Next(500, 10000));
    AddBanjo("Clawhammer Grip", r.Next(500, 10000));
}
```

- This method is held in the Shop class
- It makes a few banjos and stores them
  - The `AddBanjo` method accepts a name and a value for the banjo
- This is much easier for testing than typing them in

Create some Test Banjos

```
string[] banjoNames = new string[] {
"The Strummer", "The Woodsman", "Deep Twang", "Clawhammer Grip",
"Beard Tangler", "Old Smokey", "Vera", "Front Porch Special",
"Plain White", "The Trumpet", "Mr Parse", "Big Earl"  };
```

- This is an array of banjo names that you can use in your program if you like
- We could write a loop that creates a set of banjos with those names and random values
- If you want to you can add more names to the list

The Rental class

- A customer will contain a list of rentals that the customer has made
- Each rental will contain the start and end of the rental and the banjo that was rented
- The relationship between Rental and Customer is very similar to the one between BanjoStore and Banjo
- One contains a collection of the other

Designing your Solution

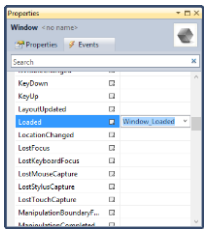
- You may be wondering where does the data "live" when the program is running?
- When the program is active the main page will hold a reference to the shop that holds the data the program is working on
- When the program starts the shop data will be loaded, and when the program ends the shop data will be stored

Events in Windows Presentation Foundation (WPF)

- In a WPF application parts of the program run in response to events that are produced by the WPF framework
- We have seen that we can make buttons on the screen cause events in the program
- There are also events that fire when the program starts and when it ends
- The program will use these events to trigger when the data is loaded and saved

Window Loaded Event

- Visual Studio will connect window events to methods for us
- This event occurs when the program loads the main window and displays it
- That is when we want to load our data



Setting up the Shop in Window_Loaded

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    Shop ActiveShop;
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        ActiveShop = new Shop("Test Shop");
        ActiveShop.MakeTestData();
    }
}
```

- This makes a new shop when the program starts
- In the finished program it will load the data from a file

Saving in the Window Closing event

- There is a corresponding event which is fired when the user leaves the program
- This is when your application should save the data to a file
- Note that this means the user never actually saves and loads the information
  - The data is automatically saved and loaded when the program is started and stopped
- This makes the program much easier to use

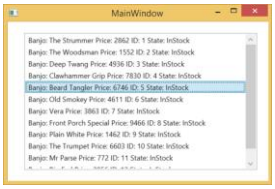
Showing Lists using the ListBox component

```
<ListBox Height="109" HorizontalAlignment="Left"
Margin="87,142,0,0" Name="BanjosListView"
VerticalAlignment="Top" Width="326" />
```

```
BanjosListView.ItemsSource = ActiveShop.Banjos;
```

- The ListBox is a XAML component that can display lists
- We can add it to the XAML in our page very easily
- It will display a collection of items if we set the ItemsSource property to the collection we want to see

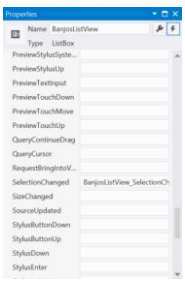
ItemsSource in ListView



- The ListView uses the ToString method of the object to get the string to display in the box
- In my Banjo class this returns all the banjo details

Selection Changed Events

- Our program can connect to an event which fires when the user selects a different item in the ListView
- This is how we can get our users to select banjos from the list



Dealing with Selection Changed Events

```
private void BanjosListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    MessageBox.Show(e.AddedItems[0].ToString() +
                    " was selected");
}
```

- The event handler for the selection changed event is given a list of items that were selected
- We can use this in our program identify which items is required

Problem Solving

- Keep each individual step a small one so that it is easy to debug
- Don't "run away" from problems you can't solve to other problems that you can't solve either
- Get help when you are stuck, but make sure that you ask "How do I do this?" rather that "What do I do?"