Chapter 1 and Introduction to Book Overview

Course Overview: contents of the book at a glance

**Source Code download**: <https://github.com/Apress/pro-csharp-7>

**Two goals of the book:**

1. teach syntax and semantics of C#
2. teach .NET APIs: ADO.NET, EF, WPF, WCF, desktop and web development

**Building blocks of the .NET platform**

platform and language independent run time execution engine and base class libraries

* MSIL (CIL)
* JIT
* CLR
* CTS
* CLS

figures 1-1, 1-2, 1-3

example on page 11 Calc.cs

review IL code with: ildasm.exe: C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools

review manifest and metadata on a DLL

multi language world of .net: c# .net, vb .net, f# .net, cobol.net

**History of .NET**

understanding CTS: class, interface, structure, enum, delegate

understanding intrinsic CTS data types: page 18

**Assembly/Namespace/Type**: pg 22 System.IO, System.Data keep names from conflicting

Sample of .NET namespaces

M$ root namespace – not safe for cross platform

using keyword

**GAC**: c:\windows\assembly\GAC

**Platform independent versions of .NET**

**Visual Studio Code**

Ch 2 Building C# Applications

Create a console application with VS – New Project Dialog

Running and Debugging

Solution Explorer

Adding references

Object Browser

Project Properties

Visual Studio IDE

**DEVELOPERS DEVELOPERS DEVELOPERS**

* VS Community
* VS Professional
* VS Enterprise

VS Community 2019 is out. Options are a bit different than in the book

C# language updates:

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1>

MSDN DOCS!

<https://docs.microsoft.com/en-us/dotnet>

.NET Framework API reference

Get familiar with API documentation layout

Core C# Programming Constructs

declaring variables – pg. 68

default values – pg. 71

everything is an object – pg. 72

Console write and read

Command line args

System.Environment

Ch 3. Hands-on:

Create a console application called Ch3SimpleConsole. The application should have the following interaction:

Welcome to my C# Program.

Please select an option:

1. Get the quote of the day.
2. The answer to the meaning of life.
3. Quit.

Enter Choice: 1

Money can’t buy me love.

Please select an option:

1. Get the quote of the day.
2. The answer to the meaning of life.
3. Quit.

Enter Choice:

Allow the user to continue to make selections until they choose option 3. Quit.

Provide 3 random quotes.

The code to get a random number between 0 and 2 is:

Random randomObject = new Random();

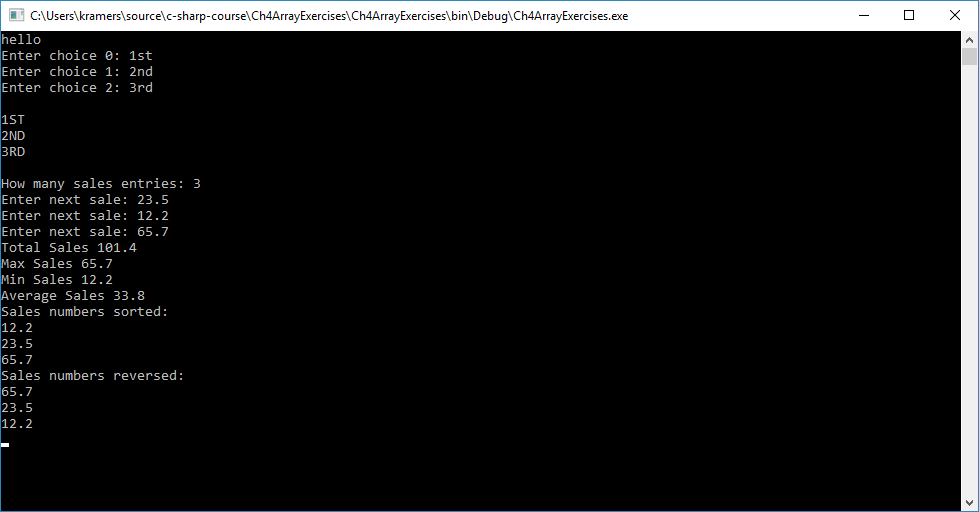
Int random = randomObject.Next(3);

Ch4 c# Programming Constructs Part II

Arrays

Ch4 Array Exercise:

* 1. create a char array using array initialization syntax (i.e. curly brackets) when the contents of ‘h’ ‘e’ ‘l’ ‘l’ ‘o’
  2. use a fori loop first then a foreach loop to write the contents of the array to the console
  3. Create an array strings of size 3
  4. Prompt the user for all three strings in a for loop
  5. Output the choices back to the user in a foreach loop. Upper case all the choices when you display them.
  6. Prompt user for how many sales entries they have
  7. Using a loop prompt them for each sale entry
  8. Print Total sales, Min Sales, Average Sales
  9. Print Sale numbers sorted
  10. Print Sales numbers reversed.



Parameter modifiers pg. 120 Table 4-2.

FunWithMethods example

Method overloading pg. 131

MethodOverloading example

Enum type

Struct type

Values types vs. Reference Types

ValueAndReferenceTypes c# example.

Nullable types

null coalescing operator

null conditional operator

# Battleship game

**Main** **Method**:

Print the options. Provide the intro and exit when done.

Initialize the board. 4x4 string jagged multidimensional array

**PrintBoard**:

Print the battleship grid to the console hiding where the ship is

**RevealBoard**:

Print the battleship grid to the console showing where the ship is

**InitializeBoard**:

Setup a 4x4 string array the will hold spaces for no ship and a tilde for the battleship

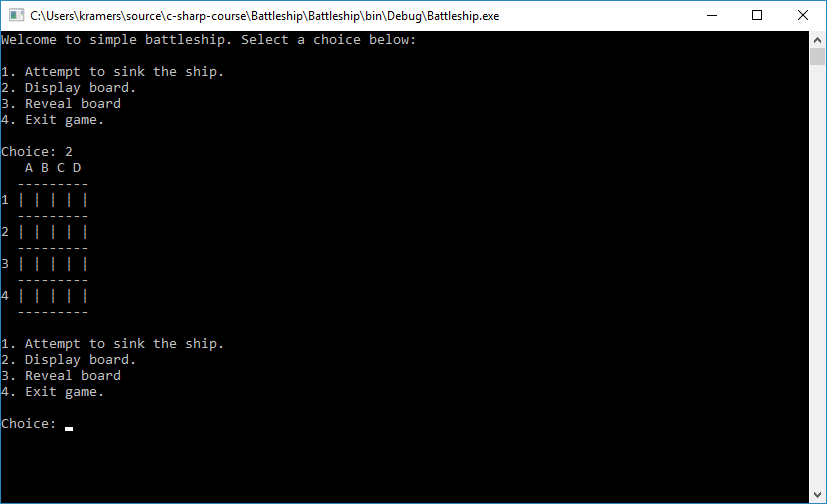
**Randomize Board:**

Generate a random column and a random row to put the ship in i.e. the tilde

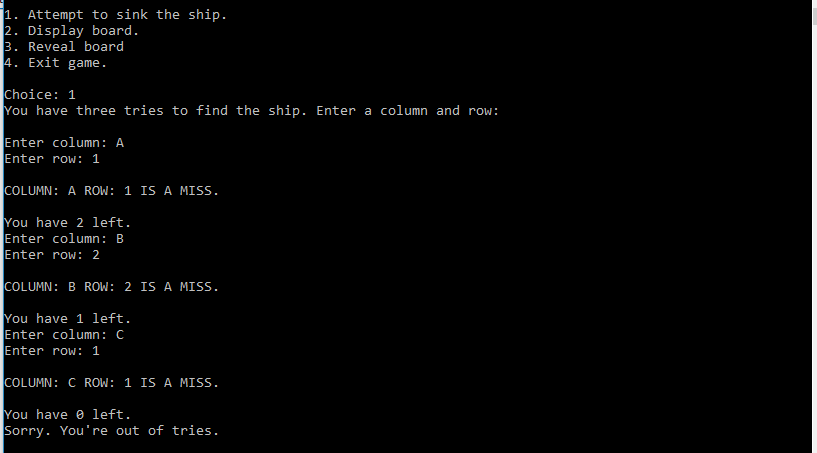
**ApplyCoordinates**:

Get the coordinates from the end user and see if they’ve won. Limit to three tries.

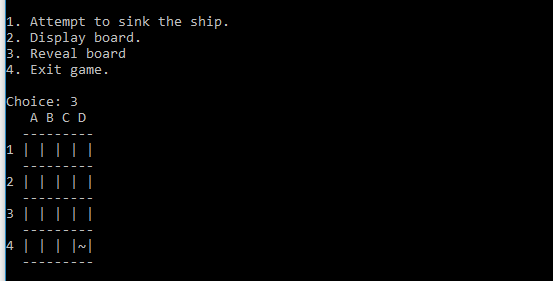
Here’s an example run:



Attempt to sink the ship:



Reveal the board:



Ch 5 Understanding Encapsulation

Classes and Objects

Constructors

Properties

Methods

c# access modifiers: pg. 188

Pillars of Object Orientated Programming pg. 184

Create classes for the following object domain:

**Employees**

id

name

salary

Methods:

Give Raise – takes an amount to raise salary to

**Company**

Properties:

id

name

departments

Methods:

Add department – takes a department object

**Department**

id

name

employees

Add employee – takes a employee object

**In your Program main method:**

Create a company and set its id and name

Create a Corporate, IT, and Loan Servicing departments and add them to your company.

Create a couple employees to add to each department.

Override the ToString method on all classes to that they print all the details of the object. The ToString method should then call the ToString method on each item within its collection.

Give a few random lucky employees a raise!

Chapter 6 Understanding Inheritance and Polymorphism

Understanding Inheritance and polymorphism

inheritance – promotes code reuse

1 base class

sealed keyword

another reason to choose classes over structures is c# structures are implicitly sealed

create a hierarchy of employees

call the base class ctor to set the base class properties

protected member variables in base classes

adding a sealed class

inner classes

virtual and override keywords

marking method abstract – Shape

using the as keyword to avoid InvalidCastException

is keyword to check for class type

**Console Paint Program**

Create a console paint program using Object Orientated Principles of encapsulation, inheritance, and polymorphism:

You will utilize the following class hierarchy:

A screenshot of a cell phone

Description generated with very high confidence

Shape class:

abstract base class for the Rectangle, Circle and Line class.

IBrush property – contains the brush to use to draw the shape.

Draw method - abstract method and therefore must be defined by all base classes

SetColor method - virtual protected method used to set the console color before the shape draws

ResetColor method - virtual protected method used to reset the console color back to the default after the shape draws. The default console color is ConsoleColor.Gray.

Rectangle, Circle, and Line classes extend Shape provide the draw method and prompt the user for any necessary variables.

IBrush interface:

Color – the console color that should be used for the drawing

Stroke – is the Unicode character that will be used to print the shape e.g. \_ or Ω, or Δ. A good resource to find Unicode characters and there numbers is here: <https://www.rapidtables.com/code/text/unicode-characters.html>

Resources

* for ascii shapes is here: shapes from here: <http://ascii.co.uk/art>
* Unicode <https://www.rapidtables.com/code/text/unicode-characters.html>

Example run:

c

Your main program should prompt for a shape and a brush. With that information you can create the appropriate Shape and Brush objects. Then you can call Draw on your shape.

Notice that each shape prompts for different parameters since they need different variables to draw. In the example above the Rectangle shape prompts for length and height whereas the circle shape prompts for a radius.

Chapter 9 Collections and Generics

Start with System.Collections namespace pg. 327

interfaces those collections implement pg. 328: ICollection, ICloneable, IDictionary, IEnumerable, IEnumerator, IList

generic collections are preferred because of type safety and better performance. Also generics prevent you from having to create your own type safe collection classes. pg. 337

generic placeholder <T> List<T> List of T pg. 338

System.Collections.Generic namespace: pg 342

Generic collection initialization syntax: pg. 344

FunWithGenericCollections pg. 345

FunWithObservableCollections pg. 350

GenericPoint pg. 357

Constraining Type Parameters pg. 360

**MyGeneric class.**

**Takes a generic of type G**

**create one property of type G called**

**GenericProperty**

**create one method on the class called**

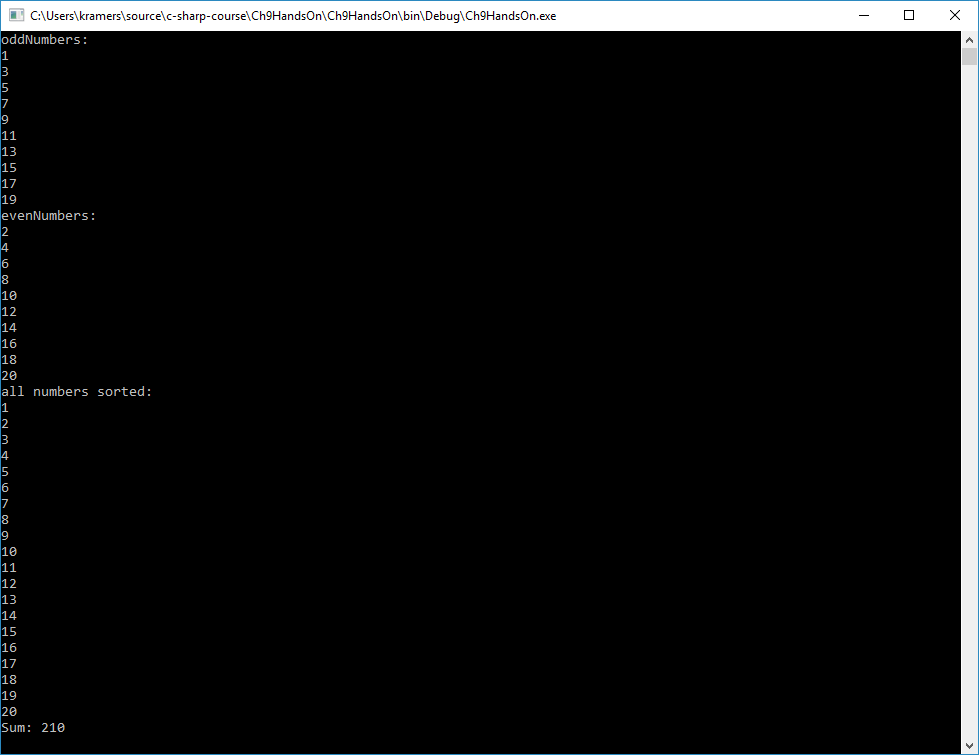
**PrintGeneric()**

**that method will print the type and the value of GenericProperty**

Utilize the Generic List<T> class. Pg. 345.

* + - 1. Create a generic list of integers called **oddNumbers** and populate it with 10 odd numbers. Utilize both the collection initialization syntax (pg. 343 – 344) and the Add method.
      2. Print the oddNumbers collection to the console.
      3. Create a generic list called **evenNumbers** and populated with 10 even numbers.
      4. Print the evenNumbers collection to the console.
      5. Create a generic list called **allNumbers** and populate with using AddRange with the values in both odd and even numbers lists.
      6. Print a the allNumbers collection to the console with the numbers sorted.
      7. Write a foreach loop to iterate over allNumbers to calculate the sum of all numbers. Print the sum to the console.

Example run:



Ch 10 Delegates, Events, and Lambda Expression

**delegates** provide a callback mechanism

**event** keyword – streamlines working with delegates

**lambdas** – can be used in place of delegates

delegate is also a function pointer

int x = 0;

delegate f = AddNumbers;

what is a delegate:

* address
* parameters
* return type

# Ch 10 project

Goals:

1. Understand and use delegates as function pointers
2. Understand and implement events
3. Understand Actions, Function and Lambdas

## Overview

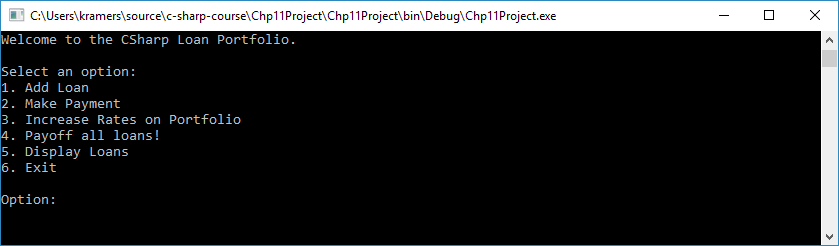
To exercise all these concepts, you’ll create the following classes/delegates:

A screenshot of a social media post

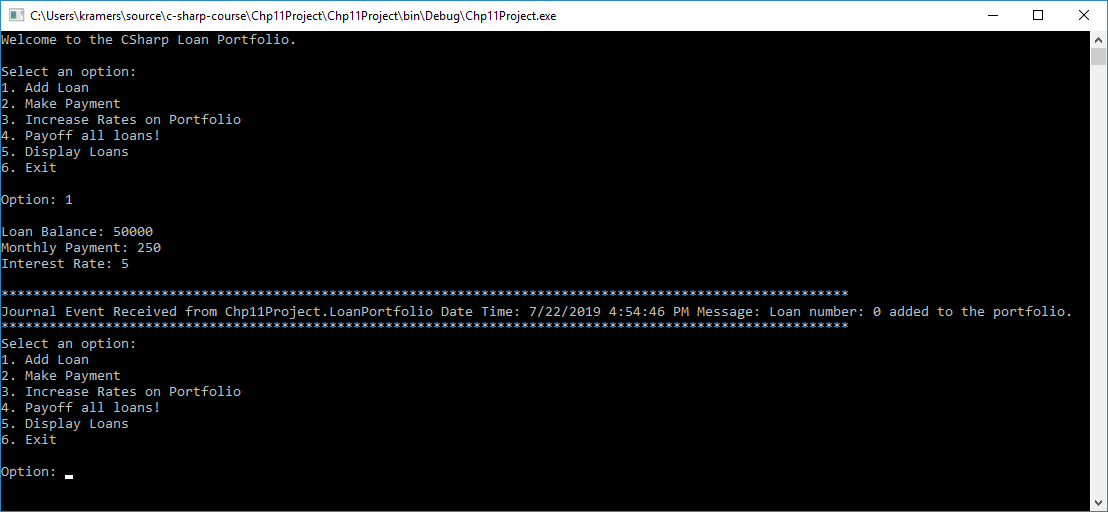
Description generated with very high confidence

The LoanPortfolio class maintains a generic collection of the Loan class. You’ll write a console application (i.e. the Program class in the diagram above) that will add loans, make payments, do bulk actions to every loan in the portfolio and log all events to the console.

Here’s a sample run of the application:

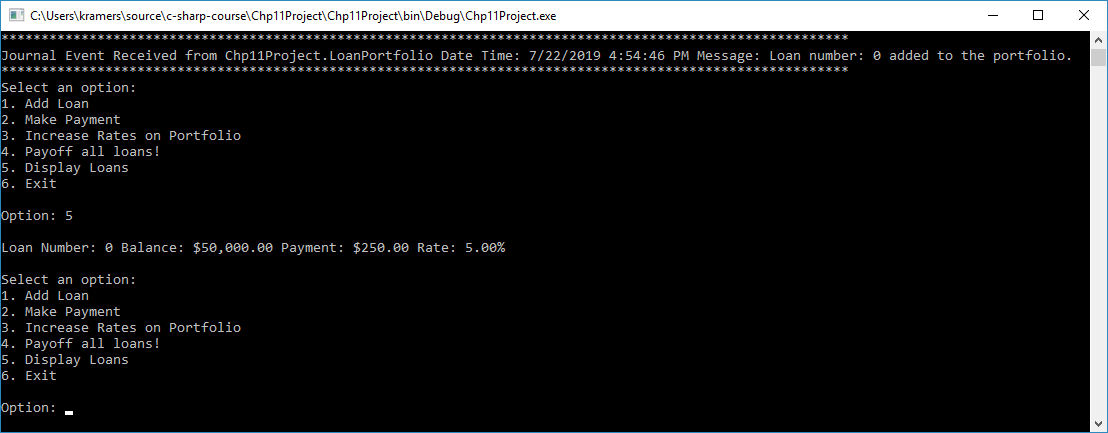


Enter option 1 Add Loan:

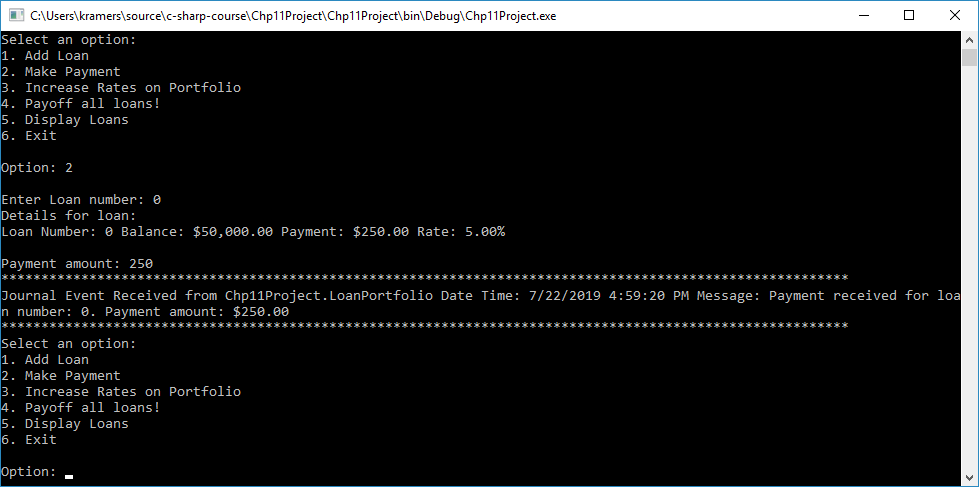


**Note: journal event is printed when a loan is added to the portfolio.**

Display all loans:



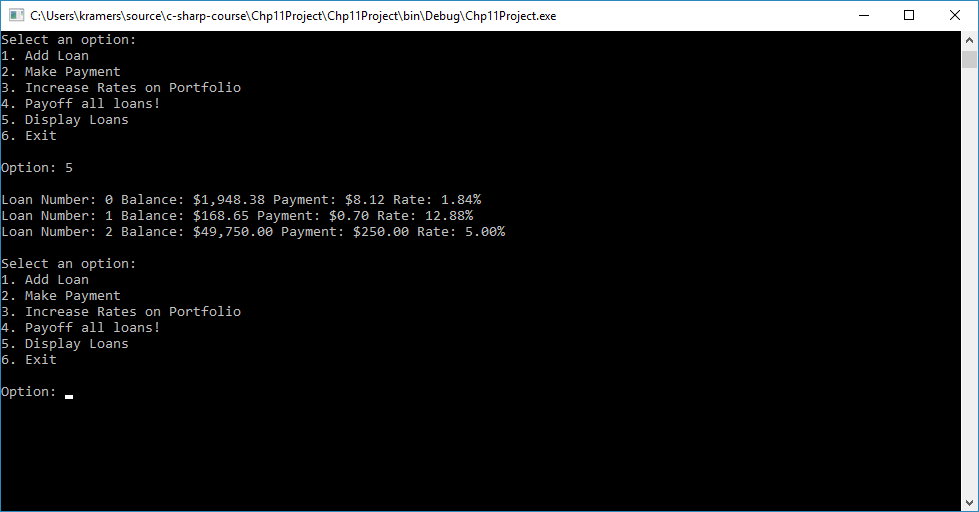
Make a payment:



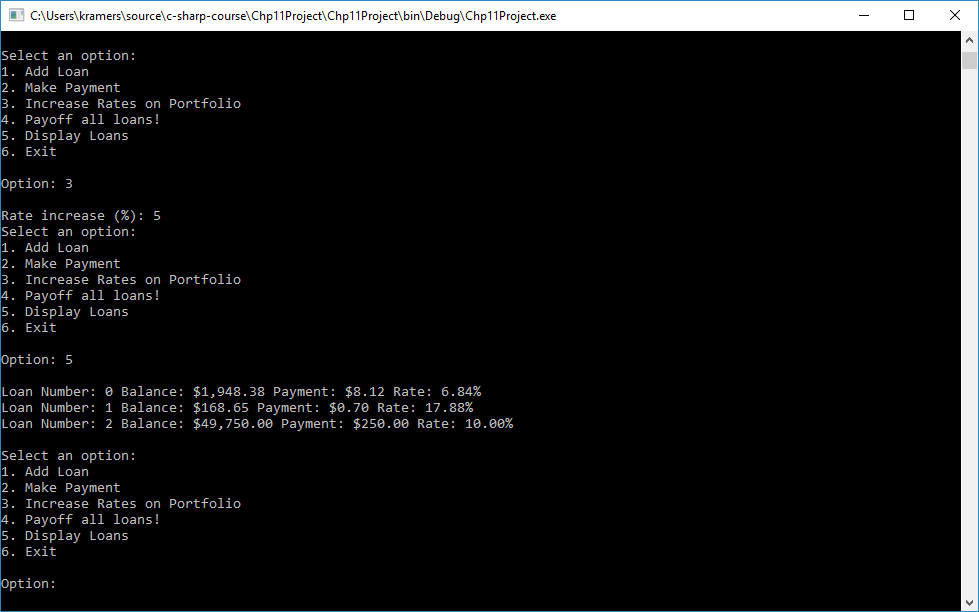
**Note: the journal event printed when a payment is made on a loan.**

I added a few more loans to show the increase rates on portfolio and payoff all loans functionality.

Display all loans:

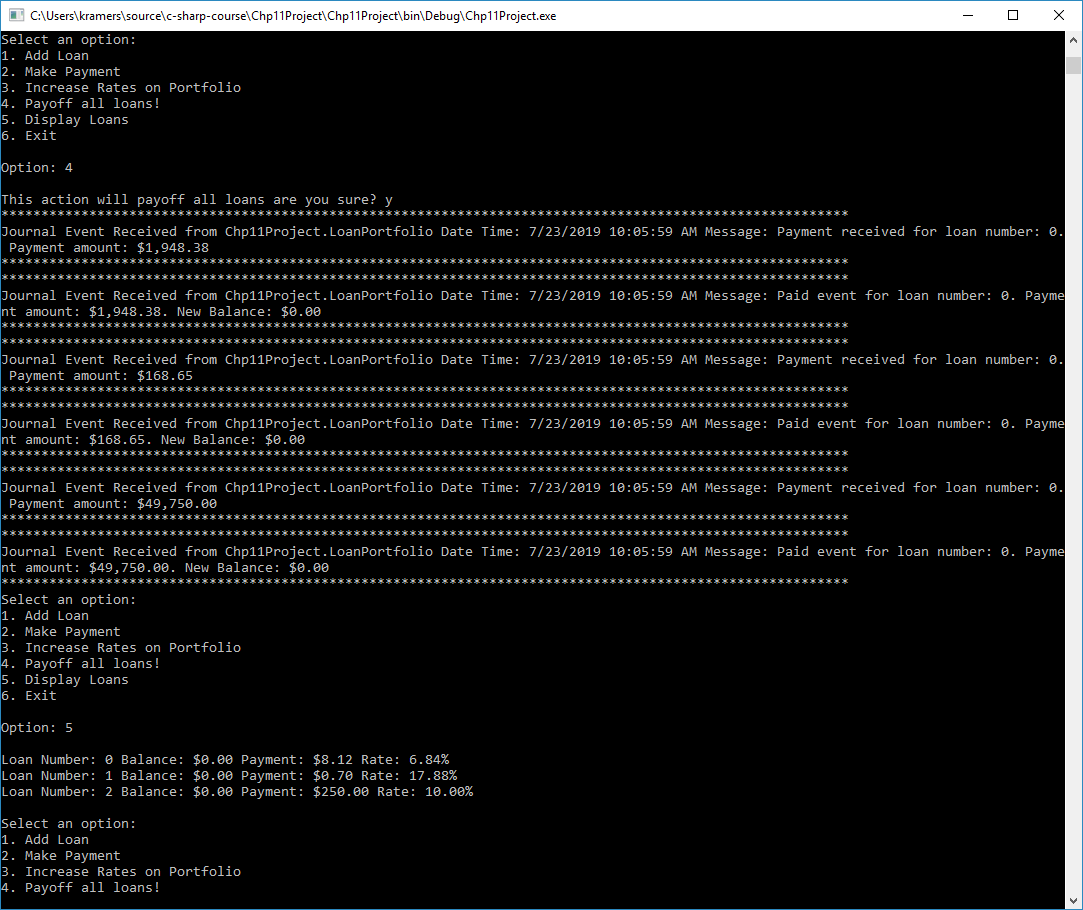


Increase Rates on Portfolio:



The rate on all loans in the portfolio has been increased by 5%.

Payoff all loans!



**Note: journal event printed when a loan is paid out.**

## Class overview

LoanPortfolio class:   
Contains a generic collection of the Loan class.

**AddLoan method:**Add the loan to the portfolio and call the EventJournal delegate to journal the loan number of the loan added. Register for PaidOut and ReceivedPayment events from the Loan class. When those events are received journal the loan number, payment amount, and new balance.

**FindLoans method:**Takes a Func<Loan, bool>. You will pass a lambda into this function to retrieve Loans by the criteria you need to find the loan by. For example to find all loans with a balance greater then 10000 you call FindLoans as follows:

IEnumerable<Loan> loans = portfolio.FindLoans((l) => l.Balance > 10000);

**PortfolioAction method:**Takes an Action<Loan> parameter. Runs the action for every loan in the portfolio.

**Journal Property:**Delegate property of type EventJournal.

Loan class:  
Contains the Rate, LoanNumber, Balance, and PaymentAmount for a loan.

**ReceivedPayment event:**This event is fired whenever a payment is made to the loan. Create a PaymentEventArgs with appropriate data and fire the event.

**PaidOut event:**This event is fired whenever the balance is less than or equal to 0 on a loan. Create a PaymentEventArgs with appropriate data and fire the event.

**ToString method:**Override the ToString method to return a string with Loan Number, Balance, Payment, and Rate. This is a good place for a StringBuilder.

Program class:  
This driver program will provide the menu prompt and accept input to add loans, make payments, increase rates on all loans and payoff all loans. In addition, the Program will provide the delegate function handler for the LoanPortfolio classes EventJournal delegate.

## Additional Instructions

All logging to the console takes place in the Program.cs file. No other class in the object model will do Console.WriteLine except the function in Program.cs that the EventJournal delegate points to.

Project: ASCII Art Store

concepts:

1. Given a problem statement provide a good object orientated design with
   1. classes needed
   2. events needed

problem statement:

ASCII Art store. Sell ASCII art. Provide an ASCII art store front in a console application.

Store

Inventory

ASCII Art

Admin functions

Sell Art

Manage Inventory

categories of ASCII art: animals, shapes, characters

Sale Specials

ArtRepository

Ch 11

**Call a method async using delegates**

Create a delegate

Invoke BeginInvoke on that method and store the IAsyncResult (calling BeginInvoke initializes and starts the background thread)

Invoke the EndInvoke method on the delegate (calling EndInvoke will wait for the delegate to complete and return the result if any)

Start with **AsyncDelegate**

starting a new thread with no callback

Then show **AsyncCallbackDelegate**

now invoke BeginInvoke with a callback

**Manually creating new threads**

ThreadStart and ParameterizedThreadStart

lock blocks of code

TimerCallback

**async await keywords**

Ch 11 Extension Methods

Add new methods or properties to a class or structure without adding them to the class directly or using inheritance

Check out the ExtensionMethods project under the books source folder.

You must import the namespace of extension methods to use them! Hint: Use using.

Intellisense help to indicate methods that are extensions.

Anonymous Types:

Allow for quick creation of data types to encapsulated properties for one-time use.

Extend from object

AnonymousTypes example project.

Ch. 12 Linq to objects

constructs needed to support LINQ – building blocks

* implicitly typed variables
* lambdas
* extensions methods
* anonymous types

Role of LINQ:



3 approaches (well really 2):

* + - 1. query expression – power tool with a laser light
      2. extension methods – power tool
      3. without linq (not really an approach to using linq) – hand tool

Deferred execution: I give you instructions on where to find what you need

Immediate execution: I give you what you need

101 LINQ Samples: <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>

ASP.NET Core:

Program.cs:

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/generic-host?view=aspnetcore-3.1>

Startup.cs:

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-3.1&tabs=windows>