CityUniversity of Seattle

Syllabus

SCHOOL OF TECHNOLOGY AND COMPUTING

IS 382: C# - Intermediate

5 Credits
Effective: Fall 2011

Access to the Internet is required.
All written assignments must be in Microsoft-Word-compatible formats.
See the library's APA Style Guide tutorial for a list of resources that can help you use APA style.

FACULTY

Faculty Name: FACULTY NAME

Contact Information: CONTACT INFORMATION

[INSTRUCTOR MAY INSERT PERSONAL MESSAGE IF DESIRED]

COURSE DESCRIPTION

This course focuses on the intermediate object-oriented programming concepts in C#, such as creating interfaces and abstract classes, garbage collection, resource management, implementing properties, using indexers, operator overloading, creating interfaces, interrupting program flow and handling events, introducing generics, and querying data using query expressions. The students will also get introduced to building applications with Windows Presentation Foundation. The course prepares the students to study advanced programming in C#.

COURSE RESOURCES

Required and recommended resources to complete coursework and assignments are available from the Course Document Lookup.

CITYU LEARNING GOALS

This course supports the following City University learning goals:

- Professional competency and professional identity
- Critical thinking and information literacy

COURSE OUTCOMES

In this course, learners:

- Demonstrate the use of delegates and events (both synchronously and asynchronously) in C#
- Demonstrate knowledge of generics and collections in C#
- Demonstrate the power of inheritance by creating interfaces and abstract classes
- Demonstrate the power and purpose of LINQ in C#
- Develop C# applications using Windows Presentation Foundation

CORE CONCEPTS, KNOWLEDGE, AND SKILLS

- Analyze the proper C# programming techniques and constructs to apply in a challenging situation.
- Become familiar with key Industry Standards and be able to apply them in context.
- Demonstrate the ability to effectively employ an Integrated Development Environment (IDE).
- Demonstrate exception handling in program
- Design and implement, or trace a detailed, complete and appropriate Program Flow
- Distinguish the best Object Oriented Programming Techniques to apply to solve problems.
- Prepare appropriate documentation of programs

- Understand and be able to apply standard Programming Constructs.
- Use code management techniques

OVERVIEW OF COURSE GRADING

The grades earned for the course will be derived using City University of Seattle's decimal grading system, based on the following:

Overview of Required Assignments	*% of Final Grade*
Interface Programming	15%
Event Programming	15%
Temperature Conversion Programming	15%
User Interface Programming	15%
Code Programming Project	20%
Instructor Determined Assignments (includes Participation)	20%
TOTAL	**100%**

SPECIFICS OF COURSE ASSIGNMENTS

The instructor will provide grading rubrics that will provide more detail as to how this assignment will be graded.

Interface Programming

The student will write a class Square by implementing ICloneable interface to provide an implementation to the Clone method that will allow copying these Square objects. The student will also write Perimeter, Area and Side properties to make them available. The same is done with the Circle class implementing the ICloneable interface exposing the radius, circumference, and area properties. The student will write a test class to test the accuracy of the calculations. The students should include an error checking and validation routine by catching and handling exceptions in C#.
Create a class named Square implementing the ICloneable interface. This class will have the following members:
• A private member variable named _side of type double. (Note that it is common practice to name class member variables with a preceding underscore to distinguish them from local variables.)
• A constructor that takes a double and saves it by calling Side. (This property will be implemented in the next step).
• A read/write property named Side that allows access to _side. This property returns a double.
• A read-only property named Perimeter that returns the perimeter of the square. To compute the perimeter: Side * 4. This property returns a double.
• A read-only property named Area that returns the area of the square. To compute the area: Side * Side. This property returns a double.
• The public Clone() method that returns an object. This method will do two things.
o It will create a new Square instance passing in this.Side into the constructor.
o It will return the new Square instance.
Create a class named Circle implementing the ICloneable interface. This class will have the following

members:
• A private member variable named _radius of type double.
• A constructor that takes a double and saves it by calling Radius. (This property will be implemented in the next step).
• A read/write property named Radius that allows access to _radius. This property returns a double.
• A read-only property named Circumference that returns the circumference of the circle. To compute the circumference: (Radius*2) * Math.PI. This property returns a double.
• A read-only property named Area that returns the area of the circle. To compute the area: (Radius*Radius) * Math.PI. This property returns a double.
• The public Clone() method that returns an object. This method will do two things.
o It will create a new Circle instance passing in this.Radius into the constructor.
o It will return the new Circle instance.
Create a class named TestCloning, which will contain the Main() method. This class will have the following members:
• A private static method named Copy that takes a single argument of type ICloneable and returns an object. This method will call the Clone() method on the argument that was passed in. The Clone() method returns an object. Then this method will return the object back to the caller.
• Implement the Main() method:
o Create a Square object providing the size of one of its sides.
o Display the square's side, perimeter, and area.
o Clone the square by calling the Copy() method. This will return an object. You will need to cast this object to a new Square().
o Display the copied square's side, perimeter, and area.
o Change the side of one of the squares and display the data of both again.
o Following the same procedure for creating two Circle objects. Remember to cast the object returned from Copy() to a new Circle.
Notice that the Copy() method will take any object that implements the ICloneable interface. You can create more classes that implement this interface and pass them all to Copy().

Components	***% of Grade***
Delegate/Events	40%
Documentation and completeness of the code	30%
Readability, Reusability and Efficiency	30%
TOTAL	**100%**

Event Programming

Create an application that will write messages to the console and to a log file whenever a timer raises an event.

1. Create a class called TimerEventArgs. It will contain a constructor that will get the current time from the system and save it in a local variable. It will also have a public read-only property that will format the time into HH:MM:SS.mmm and return this new format as a string.
2. Create a class called CustomTimerClass. It will contain a public delegate called TimerAlarmHandler and a public event called OnTimerAlarm. It will also contain a private member variable called stopFlag of type bool and set it to false.
3. The CustomTimerClass will contain three methods. The first is a method called Run() that will sleep for 1 second and then call a second method to raise an event. The second method called RaiseTimerAlarm() will check to see if there are any subscribers to the event. If so, it will create a new TimerEventArgs object and raise the event. The final method is actually a read/write property called StopFlag the sets the stopFlag member or returns its value.

4. Create a class called DisplayMessageClass. It will have two member variables. The first is a private reference to the CustomTimerClass and the second is a static counter of type int. Also, create a constructor that accepts a CustomTimerClass argument and save that argument in the private member variable. Finally, create a method called DisplayMessage that takes two arguments – a sender object and a TimerEventArgs reference and displays the information from the TimerEventArgs object. Increment the counter and then check it. If the counter is >= 0, set the StopFlag property of the CustomTimerClass object to true.
5. Create a class called LogMessageClass. It will have one member variable, a private reference to the CustomTimerClass object. Create a constructor that takes a CustomTimerClass argument and saves that argument in the private member variable. Create a method called LogMessage() that takes two arguments – a sender object and a TimerEventArgs reference and displays the information from the TimerEventArgs object.
6. Create a class called TestClass that will contain the Main() method. Create an object of type CustomTimerClass. Then create an object of type DisplayMessageClass and an object of type LogMessageClass. Call the Run() method in the CustomTimerClass object.

Components	***% of Grade***
Readability, Reusability and Efficiency	30%
Generics/Collections	40%
Documentation and completeness of the code	30%
TOTAL	**100%**

Temperature Conversion Programming

Students will create a program that converts between Celsius and Fahrenheit temperatures:
Create a base class called Temperature that contains a protected float variable that will contain the temperature value passed in through the class's constructor. Also, create a constructor that accepts a float and assigns it to the protected variable, and a get property that returns the value of the protected variable.
Create a Celsius class [inherit from Temperature if it exists]. Its constructor should just call the base constructor. Implement three methods. The first would convert a float to a Celsius. The second would convert a Fahrenheit to a Celsius by performing the following calculation on the value stored in Fahrenheit: (((temperature - 32) / 9) * 5). The third would convert a Celsius to a float.
Create a Fahrenheit class [inherit from Temperature if it exists]. Its constructor should just call the base constructor. Implement three methods. The first would convert a float to a Fahrenheit. The second would convert a Celsius to a Fahrenheit by performing the following calculation on the value stored in Celsius: (((temperature * 9) / 5) + 32). The third would convert a Fahrenheit to a float.
In a new class called Tester, create the Main() method and implement code to test the various conversions. You may want to create a loop so that the user can try it multiple times and a menu to allow users to easily choose which conversion they would like to perform.

Components	***% of Grade***
Documentation and completeness of the code	30%
Readability, Reusability and Efficiency	30%
Inheritance	40%
TOTAL	**100%**

User Interface Programming

Students will re-visit the temperature conversion program and build a user interface to it:
• Create the user interface and set the properties.
• Create the Celsius and Fahrenheit classes.

• Complete the application by providing all validation for user inputs.

Components	*% of Grade*
Documentation and completeness of the code	30%
WPF	40%
Readability, Reusability and Efficiency	30%
TOTAL	**100%**

Code Programming Project

Students will create program that is a simple compact disc management tool. It uses a text file to store data and a Windows Forms application to view the data. When this exercise is complete, the application will support the following operations.
• Allow the user to open a text file that contains a list of compact discs with associated data. The format for this file is as follows:
<index>|<title>|<artist>|<stock_number>|<price>|<genre>|<production_year>
Each element in the "record" is separated by the vertical bar. This is done so that each line of the file can be read in as a unit (using the File class), and then the code can parse the record into its individual fields. The fields that make up the record are:
<index>: string representing a unique identifier for the record.
<title>: string representing the title of the compact disc.
<artist>: string representing the person or group that created the contents that appears on the compact disc.
<stock_number>: string representing the catalog number as assigned by the record company.
<price>: double representing the price that was paid for the compact disc. (The price is stored as a string in the file, but should be used in the program as a double.)
<genre>: string representing the type of music (rock, dance, jazz, classical, etc.)
<production_year>: string representing the year the compact disc was produced.
• Display a subset of the data for each record in a list box control.
• Allow the user to click on a single line-item in the list box control. When this occurs, text boxes (with associated labels) will be populated with ALL the data that makes up the selected compact disc record.
• Provide status of the various operations in a status strip control.
• Provide a menu that allows the user to open a data file and exit the application. This will be done through a menu strip control.
• Provide a control box at the form level that only allows exiting the application. The minimize and maximize buttons will be removed.
• The form cannot be resized.
• Provide an icon that shows the type of the application running.

Components	*% of Grade*
Documentation and completeness of the code	30%
WPF	30%
LINQ	10%
Readability, Reusability and Efficiency	30%
TOTAL	**100%**

Instructor Determined Assignments (includes Participation)

Students will participate in activities as defined by the instructor. Whether in class, online, or in a mixed mode setting, students will be graded on their participation in discussions. Students will be graded on their ability to present, explain or defend alternative viewpoints; and the degree to which they have

mastered the concepts and principles inherent in the course. Other instructor-determined assignments may include, but are not limited to research tasks, outlines and drafts of required assessments, peer reviews, journals, and readings. Written work will be assessed not only on relevance to the subject presented, but also on adherence to good written form and professional presentation. The instructor may modify the grading components for his or her assignments as appropriate.

Components	***% of Grade***
Writing Mechanics	15%
Analysis	25%
Content	30%
Key Topic Support	30%
TOTAL	**100%**

COURSE POLICIES

Late Assignments

LATE ASSIGNMENT

Participation

PARTICIPATION

Professional Writing

Assignments require error-free writing that uses standard English conventions and logical flow of organization to address topics clearly, completely, and concisely. CityU requires the use of APA style.

UNIVERSITY POLICIES

You are responsible for understanding and adhering to all of City University of Seattle's academic policies. The most current versions of these policies can be found in the University Catalog that is linked from the CityU Web site.

Scholastic Honesty

Scholastic honesty in students requires the pursuit of scholarly activity that is free from fraud, deception and unauthorized collaboration with other individuals. You are responsible for understanding CityU's policy on scholastic honesty and adhering to its standards in meeting all course requirements. A complete copy of this policy can be found in the University Catalog in the section titled *Scholastic Honesty* under *Student Rights & Responsibilities*.

Attendance

Students taking courses in any format at the University are expected to be diligent in their studies and to attend class regularly.

Regular class attendance is important in achieving learning outcomes in the course and may be a valid consideration in determining the final grade. For classes where a physical presence is required, a student has attended if s/he is present at any time during the class session. For online classes, a student has

attended if s/he has posted or submitted an assignment. A complete copy of this policy can be found in the University Catalog in the section titled *Attendance Policy for Mixed Mode, Online and Correspondence Courses*.

SUPPORT SERVICES

Disability Resources

If you are a student with a disability and you require an accommodation, please contact the Disability Resource Office as soon as possible. For additional information, please see the section in the University Catalog titled *Students with Special Needs* under *Student Rights & Responsibilities*.

Library Services

CityU librarians are available to help you find the resources and information you need to succeed in this course. Contact a CityU librarian through the *Ask a Librarian* service, or access library resources and services online, 24 hours a day, seven days a week.

Smarthinking

As a CityU student, you have access to 10 free hours of online tutoring offered through Smarthinking, including writing support, from certified tutors 24 hours a day, seven days a week. Contact CityU's Student Support Center at help@cityu.edu to request your user name and password.