|  |  |  |
| --- | --- | --- |
|  | **FACULTY OF COMPUTING, ENGINEERING and SCIENCE** | Final mark awarded:\_\_\_\_\_ |

**Assessment Cover Sheet and Feedback Form 2016/17**

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code:  CS2S564 | Module Title:  Event Driven and Graphical User Interface Programming | | Module Lecturer:  Keith Norris, Ben Daubney |
| Assessment Title and Tasks:  Real-Time Stock Price Viewer | | | Assessment No.  2 of 2 |
| No. of pages submitted in total including this page:  Completed by student | | | Word Count of submission  (if applicable): Completed by student |
| Date Set:  10/02/2017 | | Submission Date:  31/03/2017 | Feedback Date:  08/05/2017 |

|  |  |
| --- | --- |
| ***Part A: Record of Submission (to be completed by Student)*** | |
| **Extenuating Circumstances**  If there are any exceptional circumstances that may have affected your ability to undertake or submit this assignment, make sure you contact the Advice Centre on your campus prior to your submission deadline. | |
| **Fit to sit policy**:  The University operates a fit to sit policy whereby you, in submitting or presenting yourself for an assessment, are declaring that you are fit to sit the assessment. You cannot subsequently claim that your performance in this assessment was affected by extenuating factors. | |
| **Plagiarism and Unfair Practice Declaration:**  By submitting this assessment, you declare that it is your own work and that the sources of information and material you have used (including the internet) have been fully identified and properly acknowledged as required[[1]](#footnote-1). Additionally, the work presented has not been submitted for any other assessment. You also understand that the Faculty reserves the right to investigate allegations of plagiarism or unfair practice, which, if proven, could result in a fail in this assessment and may affect your progress. | |
| **Details of Submission:**  Note that all work handed in after the submission date and within 5 working days will be capped at 40%[[2]](#footnote-2). No marks will be awarded if the assessment is submitted after the late submission date unless extenuating circumstances are applied for and accepted (Advice Centre to be consulted).  **Submission**   1. Submit through blackboard a zipped file including all source code required to compile and run your program. 2. You must also submit a user guide to your **DataReceiver** class. This should be a word document and can be included in the zipped folder. 3. A code demo must be completed the week following the submission deadline. If you know you will be unavailable at this time, please make this known before the submission deadline. | |
| **You are required to acknowledge that you have read the above statements by writing your student number in the box:** | Student Number: |

|  |  |  |
| --- | --- | --- |
| **Part B: Marking and Assessment**  **(to be completed by Module Lecturer)** | | |
| This assignment will be marked out of 100%  This assignment contributes to 50% of the total module marks.  This assignment is bonded. Details: You must achieve 40% overall to pass the module. | | |
| **Assessment Task:**  **Introduction:**  You are part of a team working on a large scale project to develop a new stock market  trading platform. Your role as a junior developer is to develop a class that can listen to a stock market price feed through a specified IP address and port. Each time a stock price update is received the class should transmit the new price via a public event on the class.  To help test your application you have been provided with a stock price simulator that will simulate the data that the real feed will generate when operational.  To test and demonstrate your class you will need to write a simple GUI based application specified below.  **Transmitted Data Format:**  Initially, data for five stock prices will be sent, these are "BT Plc", "HSBC holdings Plc", "Admiral Group", "BAE Systems plc" and "Intertek Group plc". It has been agreed that you can assume that only data for these stocks will be sent.  The data will be sent in JSON format with the following fields:  struct StockDetails  {  public int StockId;  public string StockName;  public double StockPrice;  public double StockChange;  public DateTime Time;  }  The StockId field represents a unique id for the particular stock. The StockName field represents the name of the stock. The StockPrice field is a double representing the current price. The StockChange field represents the change in the price since trading opened. The Time field represents the time that the data was sent. The stock ID for each stock is shown in the table below:   |  |  | | --- | --- | | **StockId** | **StockName** | | 0 | BT Plc | | 1 | HSBC holdings Plc | | 2 | Admiral Group | | 3 | BAE Systems Plc | | 4 | Intertek Group Plc |   Only the price for one stock will be sent at a time, but the order they are sent in will be seemingly random as will the time interval between transmissions.  Each update in JSON format will not be more than 256 bytes in length.  The stock price simulator includes features to help you test your application, such as manually sending stock price updates.  **Proposed Solution:**  The solutions architect has given you the following specification for your listener class which you should adhere to:  The name of the class will be **DataReceiver**.  The name of the event to transmit stock price updates has not been specified but it should be suitably named. However, the delegate used by the event should accept as its only parameter a **StockDetails** object (see definition above). This object should be fully populated and contain the new item of data received (i.e. stock price value).  The class **DataReceiver** should contain two public methods, one to handle connecting to a given IP Address and port number (both passed in as arguments) and one that will listen for new stock price changes. Each of these are described in more detail in the table below:   |  | | --- | | **Description** | | The first method will open a connection with the given IP Address and port number. It can be assumed that the given details are correct and represent a valid IP Address. | | The second method will listen to the socket opened inthe above function.  Each time a stock price update is received it will broadcast this data through the public event on the class.  It can be assumed that the first method will have been invoked before this method is (i.e. the port is open). |   **Test Application:**  To test and demo the **DataReceiver** class the following Winforms application should be created (the GUI does not have to exactly the same as shown below though should have the functionality as described):    The “**Connection**” section of the form allows the user to enter an IP Address and Port number. When the “**Connect**” button is pressed the form should open a connection to the specified socket through the **DataReceiver** class and start listening for data to arrive (note this should be called on a worker thread so as not to make the application unresponsive). All components in the “**Connection**” section should then be disabled to prevent the user trying to reconnect to a different socket. You can assume the IP address and port number entered by the user is in the correct format and represents a valid socket.  The form then contains two separate sections: A “**Connection Summary**” section that shows how many price updates have been received and the time of the last update. A “**Current Prices**” section that displays a table containing each stock and the most recent price and price change (note that the price change is contained on the transmitted data, you do not have to calculate it).  Your manager has made it clear that each of these two sections should be updated via a separate call back function registered to the publicevent on the **DataReceiver** class. This is to ensure that the event on the class can handle multiple call back functions.  Finally, a menu should be provided so that the user can decide which data to display in the table. If the user selects a given stock, only that stock should be displayed. If the user selects “Show All” all data should be displayed.    As the application is only for the purpose of demoing your code it does not matter if the application exits cleanly (i.e. when closing your program don’t worry if you get unhandled exceptions).  **Additional Info:**  The stock market price generator will be available through blackboard.  You must ensure your solution is adequately documented and commented.  **User Guide:**  In addition to your code you must provide a user guide for your **DataReceiver** class explaining how to use it and some details on how it works.  You can assume the reader is a C# developer who is reading the documentation with the intention of integrating your class into their own project. You can assume they have a basic knowledge of C# but may not understand delegates, events, threading etc.  To explain how it can be used you might want to include code extracts/screen shots from your demo program or simple code examples of how to hook up to the events etc. | | |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Learning Outcomes to be assessed** (as specified in the validated module descriptor https://icis.southwales.ac.uk/):   |  |  | | --- | --- | | 1 | To Demonstrate an understanding of the prevailing technologies associated with the development and use of Event-driven Graphical User Interface applications. | | 2 | To demonstrate the application of the prevailing technologies associated with the design, development and testing of Event-driven Graphical User Interface applcations. | |  | | | |
| **Marking Scheme for Assignment** | **Marks Available** | **Marks Awarded** |
| **DataReceiver class:** |  |  |
| Implementation and documentation of event mechanism. Including declaration of delegate as specified in coursework description, correct invocation of event with fully populated argument. | **20** |  |
| Implementation and documentation of functionality to connect to a specified IP address and port and to read from that port. Each time new data is received it should be correctly decoded and transmitted through the event on the class. | **20** |  |
| **Demo App:** |  |  |
| Implementation of Connect button to correctly initialise the **DataReceiver** class and start it reading from the IP address and port specified by the user. GUI should not become inactive whilst awaiting data from socket connection.  Correct registration of delegates to the **DataReceiver**’s event.  The IP Address text box, Port text box and Connect button should be disabled once the Connect button has been pressed for the first time. | **15** |  |
| Implementation and use of call back functions to update “**Connection Summary**” and “**Current Prices**” sections. These must be separate functions and registered to the event using a delegate. | **15** |  |
| Implementation of menu to select which data should be shown on the table in “**Current Prices**” section. | **10** |  |
| **Additional Work:** |  |  |
| Document explaining how the **DataReceiver** class works and how to use it. | **10** |  |
| Code Demo | **10** |  |
| **Total:** | **100** |  |

1. University Academic Integrity Regulations [↑](#footnote-ref-1)
2. Information on exclusions to this rule is available from Campus Advice Centres [↑](#footnote-ref-2)