**Event Driven Programming Solutions (7630-414)**

**Assigment A – Pawel Gajewski**

**09.04.2019**

**Task 01**

**Event Driven Programming**

Event driven programming is a programming model in which flow of the program is determined by the events that occur in the system. There are few ways that events can happen some of them may be event listeners which would be mouse hover over something or keyboard inputs but events can be created not only by the user and this is very important part in event driven programming. Events can also be created without user interaction, they can be created by other things for example software in our personal computer or even other hardware. So basic thing in event driven programming are events and the way how the program will respond for them. This programming model is very popular because it can provide very high responsiveness for users inputs which directly translates to better user experience. Event driven programs could be written in any language but there are few languages that are specially designed for this type of programming, those would be languages like Visual Basic for example.

Below we can see few of event groups that can occur:

* Component events
  + This type of event is triggered when the instance of component is created. Components events are usually used in the system to communicate between multiple components.
  + User interaction with the web page can be used with component events. This would be clicking links on the web page or submitting forms.
* Hardware
  + Hardware events occur when user clicks a button on the keyboard or uses his mouse.
  + This can be printer connected to the computer and sending information that the document was successfully printed.
  + External speakers can have volume control know that controls the volume and some application can use this to respond to change the value and respectively would lower the volume or make it louder.
* Keyboard
  + Keyboard events are really easy to visualise and there are three types off keyboard events:
    - “KeyDown” event, which tells us when the key is pressed, and it can occur once.
    - “KeyPress” event, which would tell us that the button is pressed and this event could occur multiple times as the user, holds down the button.
    - “KeyUp” event tells us that the user released the button and this event is occurring only once like in key down event.
* Mouse
  + Mouse events occur when user interacts with mouse and there is few events triggering:
    - It could be mouse down, mouse up or simple click event
    - This could be double click
    - Mouse move is event as well
    - Mouse over and mouse out
* Sensor
  + Those events are very popular in mobile technology, there are many examples of sensor events and the basic events could be:
    - Using gyroscope in our smart phone
    - Or maybe map application constantly updating our GPS position
    - Recent updates on android let user choose that phone response to level of light that is read by light sensor and then can adjust brightness of our screen, which means at night our phone is not so bright and in a sunny day it would use 100% of brightness.
* System
  + System events are events coming from activities in our operating system and they help us understand of current state of our system. This could be:
    - System restarting
    - System shutting down
    - Or maybe system updating
* Touchscreen
  + Touchscreen events are relatively new type of events and mostly used in tablets and smart phones to provide response for touch based user interfaces. It can be touch with a finger or special “stylus” pen recognized by the device. In this type of events are also few states:
    - Touch start
    - Touch end
    - Touch move
    - Or touch cancel
* Windows
  + Windows events are mostly events triggered by the errors or warnings occurred in the system. Most common events are:
    - Error
    - Warning
    - Information

Discuss the following:

* Hardware Interrupts
  + Hardware interrupts is method that devices are using when they would need to tell the operating system about occurred event. Interrupts can be used to make asynchronous events. For example when writing to hard drive we could see notification that the device memory is full, that would be hard drive sending notification to the system. Referencing the interrupt is done by interrupt number which can be tracked back to the hardware that created the interruption.
* Polling
  + Event polling could be compared to gathering all the users inputs into a list called “event queue”. Polling is also continuously checking for change in state of programs or devices. For example multipoint communication where one main device controls other devices and sends a message to each one of them asking about any change.
* Event Dispatcher
  + The dispatcher is and object that holds a registry of event listeners and when some event is dispatched it also notifies all listeners registered to that event.
* Event listener
  + Event listener is an object that handles the event dispatched by event dispatcher. Listeners are programmed to respond to an input by calling event handler.
* Event handler (callback)
  + Event handler is a routine responsible for dealing with events and programmers can write code to handle the event when it occurs.

There are of course advantages of using event driven programming as well as disadvantages:

* Pros:
  + Event driven programming is very flexible and it follows logical order from start to finish. Using language like visual basic.net developers get help while writing the code and can see suggestions as the write. Its strictly drive on events so it is really easy to extend the functionality by adding new events to the system.
* Cons:
  + Event driven program constantly runs in background so it is taking up resources of the machine on which it is running, that mean we would need higher spec machine for all those tasks end events running in a loop asynchronously. EDP language is difficult to translate to different programming languages and in some cases even impossible.

Task 01 Word Count: 950

**Task 02**

**Design Specification Document**

**Description:**

Changing outdated simple login system allowing bank customer to log into the system and check their current balance. But because system is outdated and does not provide much response to the user’s current errors or invalid entries are not displayed to the users. A project has been commissioned to further this application to better inform the user about what is happening within the system.

The object we will be using are specified bellow and in Appendix A:

Class User:

* First Name
* Last Name
* User Id
* Password
* Role Method
* Account Number
* Email Address

Method to setup Role:

* Administration Full Access
* Administration Report Privileges
* Generate Audit Records
* View Audit Records
* Input Account Payments
* Authorise Account Payments
* Manage Account
* View Account Information
* View Account Balances

Class DataManager:

* dataTable

Class FileManager:

* LoadData
* SaveData

Class Password

* isPassword method

**Business Requirements:**

Application will be used by banks customers and bank employees, so the account number is not necessary for some users as they will be only employees. Validate the user privileges based on role of the user.

**Product & Technical Requirements:**

Login application will be reading from csv file and will be able to validate user inputs. Also when wrong characters are used there will be error handling system that will display the error messages. On screen help to guide user thru process of login in will also be available. On login window we will also have forgot password button that lets user to retrieve his password by mail. After successful login user will see new window with his details and will have chance to change his password . Backdoor login and password is provided and set up as “fred”. This account would grant full access to that user. If the user is not a member there is of course option to register within the application.

**Assumptions:**

Our application is reading csv files but usually for this kind of applications we should use database to have all the details there and we could read and write to it as needed.

**Functional Requirements**

Some of the features of this application:

* **Appendix A**: Password File Record Specification
* **Appendix B**: Class Diagrams
* **Appendix C**: Data File Structures
* **Appendix D**: End User Help Text (Guidance)
* **Appendix E**: End User Error Messages
* **Appendix F**: Screen Layout Sketches

**Dependencies**

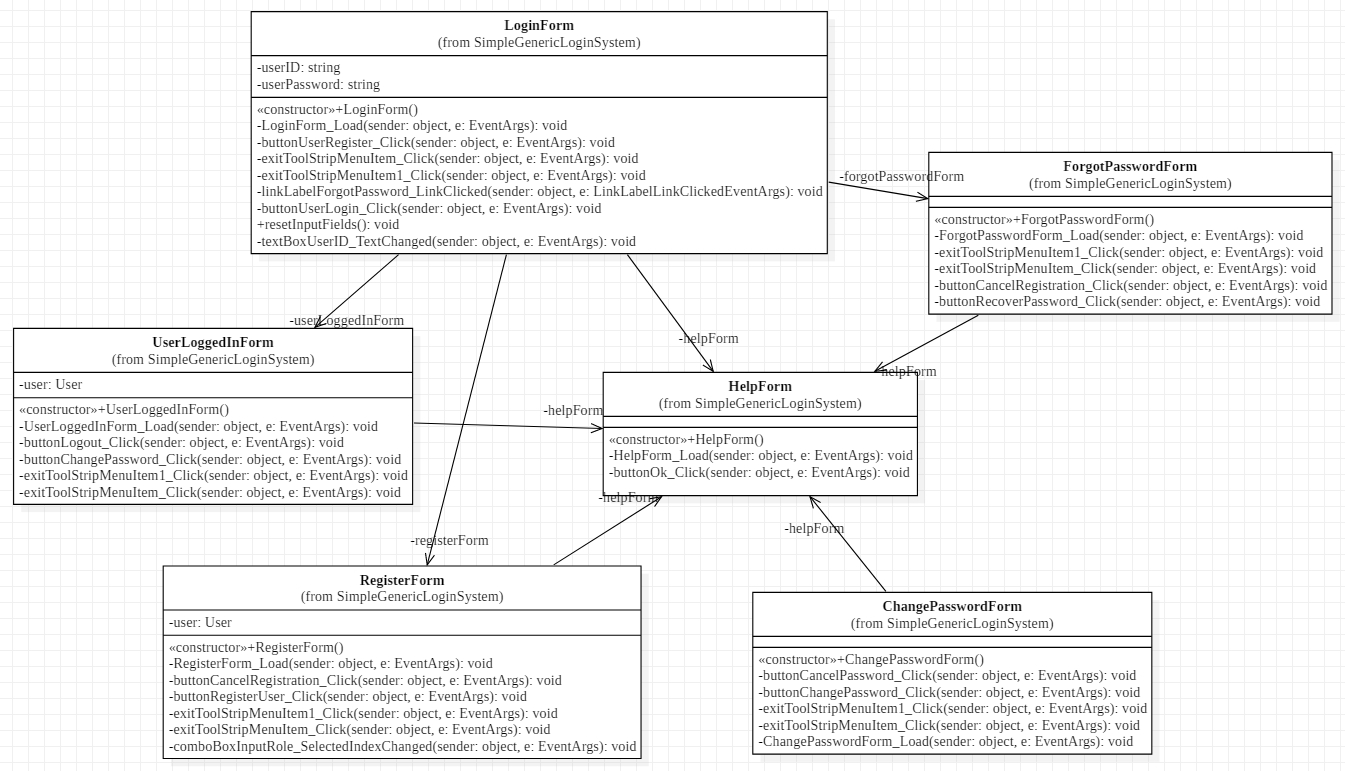
Simple login application is designed to work on any windows computer and the bank that will be using it has Windows computers in place.

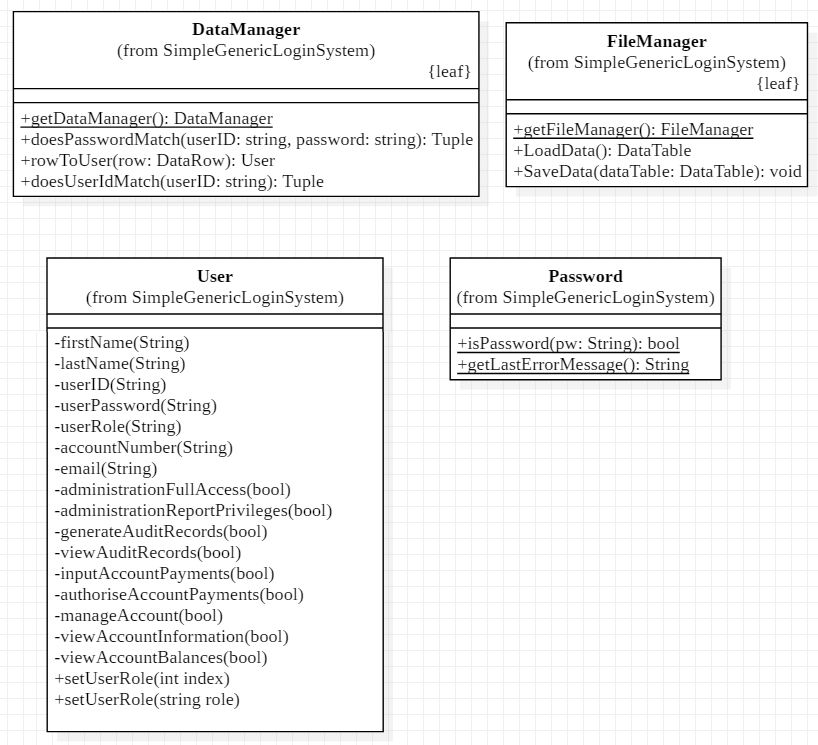
**Appendix A: Password File Record Specification**

**Filename: p\_words.csv**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sequence** | **Column** | **Required** | **Data Type** | **Min Chars** | **Max Chars** | **Min Value** | **Max Value** | **Default Value** |
| 1 | First Name | Yes | String (Alpha) | 3 | 25 |  |  |  |
| 2 | Last Name | Yes | String (Alpha) | 3 | 25 |  |  |  |
| 3 | User Id | Yes | String (Alpha Numeric) | 8 | 8 |  |  |  |
| 4 | Password | Yes | String (Alpha Numeric) | 8 | 16 |  |  |  |
| 5 | Role | Yes | String (Alpha) | 3 | 35 |  |  | Customer |
| 6 | Account Number | Yes, for Role: Customer No for other roles | String (Numeric) | 9 | 9 |  |  |  |
| 7 | Email Address | Yes | String (Alpha Numeric) | 15 | 35 |  |  |  |
| 8 | Administration Full Access | Yes | Boolean | N/A | N/A | No | Yes | No |
| 9 | Administration Report Privileges | Yes | Boolean | N/A | N/A | No | Yes | No |
| 10 | Generate Audit Records | Yes | Boolean | N/A | N/A | No | Yes | No |
| 11 | View Audit Records | Yes | Boolean | N/A | N/A | No | Yes | No |
| 12 | Input Account Payments | Yes | Boolean | N/A | N/A | No | Yes | No |
| 13 | Authorise Account Payments | Yes | Boolean | N/A | N/A | No | Yes | No |
| 14 | Manage Account | Yes | Boolean | N/A | N/A | No | Yes | No |
| 15 | View Account Information | Yes | Boolean | N/A | N/A | No | Yes | No |
| 16 | View Account Balances | Yes | Boolean | N/A | N/A | No | Yes | No |

# **Appendix B: Class Diagrams**

****

****

**Appendix C: Data File Structures**

In this project we will be using CSV data file.

CSV stands for Comma Separated Values and its nothing more than plain text file with list of data. Those type of files are very often used when data needs to be exchanged between the applications. CSV files are designed in the way that users can simply export data from one application to other application.

**Appendix D: End User Help Text (Guidance)**

User help window content:

1. **Login Window:**

Input user Id and user password and click Login. If successfully logged in the new window will show up. If incorrect values are entered in those fields or filed are left blank the error message will show up.

Option for users that forgot their password is also provided. To access password recovery window please click blue colour link “Forgot password?”

If the customer is not a member there is option to register new member and we can use “Register” to open new user register window.

1. **Forgot Password Window:**

Input existing user id to send email containing your password. If incorrect user ID is entered we can see the error message.

1. **Register Window:**

Input first name, last name user ID, password, account number, email and from dropdown list choose role for the user that is registering. NOTE: Different roles have different privileges. If the user is bank employee he wont need account number. All other fields have to be filled in.

1. **Login Successful:**

When user is successfully logged in we can see window with user details and from here we have the ability to change this user password or logout from the system.

1. **Change Password Window:**

Changing password is done by typing in new password in the filed and clicking “Change Password” button.

1. **Help Window**

Help window is available thru all the application in all of the windows. It will help users in navigating the application.

# **Appendix E: End User Error Messages**

**Message Number: Login error**

Message Text: “Please enter valid user ID and password"

Message Issued When: When user enters incorrect details into login fields and password.

**Message Number: Validation error**

Message Text: “Please enter valid user ID"

Message Issued When: When user enters incorrect details while recovering the password.

**Message Number: New User**

Message Text: “Fields required are: First Name:, Last Name:, User ID:, Password:, Email:"

Message Issued When: When user tries to register and has left the fields empty.

**Message Number: Password validation error**

Message Text: “Password must have 8 or more characters."

Message Issued When: When user enters less characters when creating new password for himself.

**Message Number: Password validation error**

Message Text: “Password must have at least one upper case (A - Z) character.”

Message Issued When: When user does not enter capital letter within new password field.

**Message Number: Password validation error**

Message Text: “Password must have at least one number (0 - 9).”

Message Issued When: When user does not enter number within new password field.

**Message Number: Password validation error**

Message Text: “Password must have at least one symbol or special character from this list”

"\*~! @ # $ % ^ & \* ( ) \_ + - = { } | [ ] \\ : \" ; \' < > ? , . /"

Message Issued When: When user does not enter symbol or special character within new password field.

# **Appendix F: Screen layout sketches**

* Login Window
* Login Successful
* Register Window
* Forgot Password Window
* Change Password Window
* Help Window

# 

Task 02 Word Count: 936

**Task 03**

Level 4 “Event Driven Programming Solutions (7630-414) Assessment A

C# application is located in folder named “SimpleGenericLoginSystem”.

<SimpleGenericLoginSystem>

**Task 04**

**Test Plan**

|  |  |
| --- | --- |
| **Software component name:**  **Application**  **Date: 26.03.2019** | Version number: 00001 **Tester name: Pawel Gajewski** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No:** | **Purpose / Test type** | **Input** | **Expected output** |
| **1** | Checking if the button does the right method | Click “Help” button | Open new window displaying users help text. |
| **2** | Checking if the button does the right method | Click “Send” button | Send message. Display window with success message and save message to a file. |
| **3** | Check the correct recipients number is inputted | Input incorrect value:  “absdefg” | Error window should come up with message about the error. |
| **4** | Check the correct recipients number is inputted | Input correct value:  “+3533434343434” | Should be no error displayed when submission is made by clicking the “send” button. |
| **5** | Check that text area contains any message to send. | Leave field blank | When submitting the error should come up that message is empty. |
| **6** | Check that text area contains any message to send. | Type in some message | When making submission there won’t be error message displayed. |
| **7** | Check if SMS and MMS radio group works | Pick “MMS” message | Should make visible file picker so user can choose the file attachment to send |
| **8** | Check if SMS and MMS radio group works | Pick “SMS” message | File picker button should not be visible for text message |

**Test Data**

|  |  |
| --- | --- |
| **Software component name:**  **Application**  **Date: 26.03.2019** | Version number: 00001 **Tester name: Pawel Gajewski** |

|  |  |
| --- | --- |
| **Test Data No: 1** | 1. **Help button** |
| **Field Name** | **Input Value** |
| 1. Help button | “Help” button clicked once |

|  |  |
| --- | --- |
| **Test Data No: 2** | 1. **Send button** |
| **Field Name** | **Input Value** |
| 1. Send button | “Send” button clicked once |

|  |  |
| --- | --- |
| **Test Data No: 3** | 1. **Phone validation** |
| **Field Name** | **Input Value** |
| 1. Phone validation | Input incorrect value: “absdefg” |

|  |  |
| --- | --- |
| **Test Data No: 4** | 1. **Phone validation** |
| **Field Name** | **Input Value** |
| 1. Phone validation | Input correct value: “+3533434343434” |

|  |  |
| --- | --- |
| **Test Data No: 5** | 1. **Message validation** |
| **Field Name** | **Input Value** |
| 1. Message validation | Leave field blank |

|  |  |
| --- | --- |
| **Test Data No: 6** | 1. **Message validation** |
| **Field Name** | **Input Value** |
| 1. Message validation | Type in some message |

|  |  |
| --- | --- |
| **Test Data No: 7** | 1. **Radio group validation** |
| **Field Name** | **Input Value** |
| 1. Radio group validation | Pick “MMS” message button from the radio group |

|  |  |
| --- | --- |
| **Test Data No: 8** | 1. **Radio group validation** |
| **Field Name** | **Input Value** |
| 1. Radio group validation | Pick “SMS” message button from the radio group |

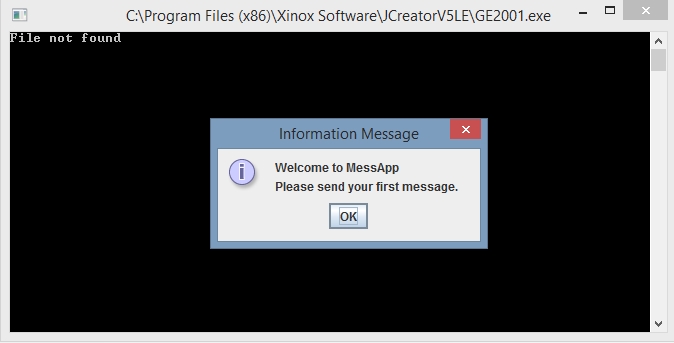
**Test Log**

|  |  |
| --- | --- |
| **Software component name:**  **Application**  **Date: 26.03.2019** | Version number: 00001 **Tester name: Pawel Gajewski** |

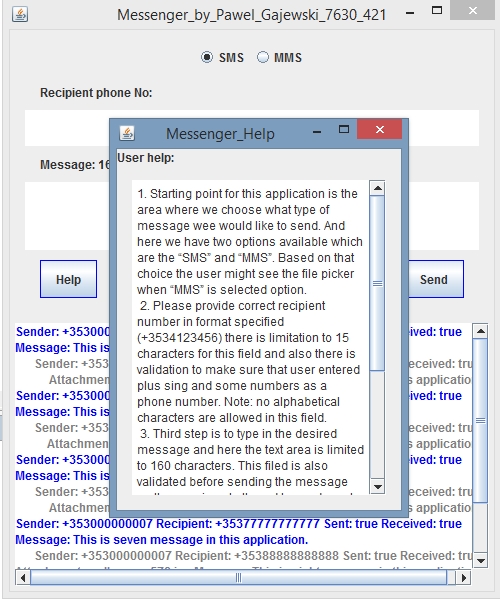
|  |  |  |  |
| --- | --- | --- | --- |
| **Test No:** | **Date** | **Actual output** | **Comments** |
| **1** | 26.03.2019 | Clicking “Help” button opened new window displaying users help text.  **Screen: “Test 1.jpg”** | Worked as expected. |
| **2** | 26.03.2019 | Clicking “Send” button validated and displayed success window that the message was sent. Also created the data file in the directory named “MessAppMessages.dat”  **Screen: “Test 2.jpg”** | Worked as expected |
| **3** | 26.03.2019 | Inputted incorrect value: “absdefg” caused program to display error message and only correct pattern is accepted.  **Screen: “Test 3.jpg”** | Validation working correctly. For future updates we could change the pattern required. |
| **4** | 26.03.2019 | Inputted correct value:“+3533434343434”  Lets the user send message without displaying error message | Worked as expected.  Note: no screen needed for this test |
| **5** | 26.03.2019 | Blank filed left in message area caused program to display error message that application will not accept blank message.  **Screen: “Test 5.jpg”** | Does not make sense to send empty message. |
| **6** | 26.03.2019 | Application accepts messages no longer that 160 characters.  And displays number of characters left to reach the limit. | Worked as expected  Note: we can see from previous screens that word counter is working correctly. |
| **7** | 26.03.2019 | Picking “MMS” message makes file picker button visible and lets user to choose desire attachment to send.  **Screen: “Test 7.jpg”** | Worked as expected |
| **8** | 26.03.2019 | Picking “SMS” message hides file picker and user can send sms message.  **Screen: “Test 8.jpg”** | Worked as expected |
| **9** | 26.03.2019 | Starting application without data file resulted in message displayed that we are using application for the first time.  **Screen: “Test 0.jpg”** | This test was not on the plan, but occurred at the start of application when running for the first time. |

**Test Evidence**

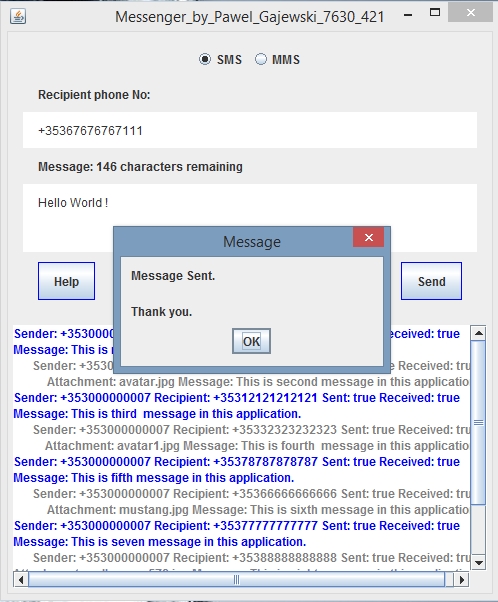
**Test 0.jpg**

****

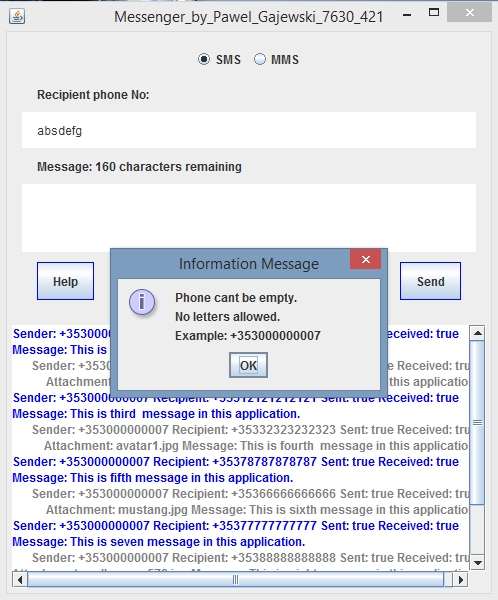
**Test 1.jpg**

****

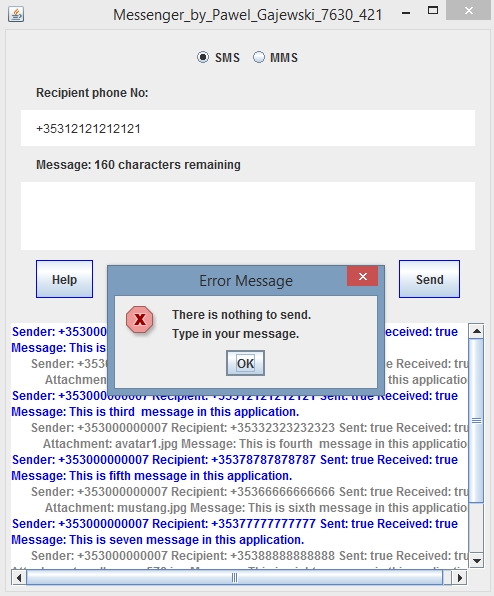
**Test 2.jpg**

****

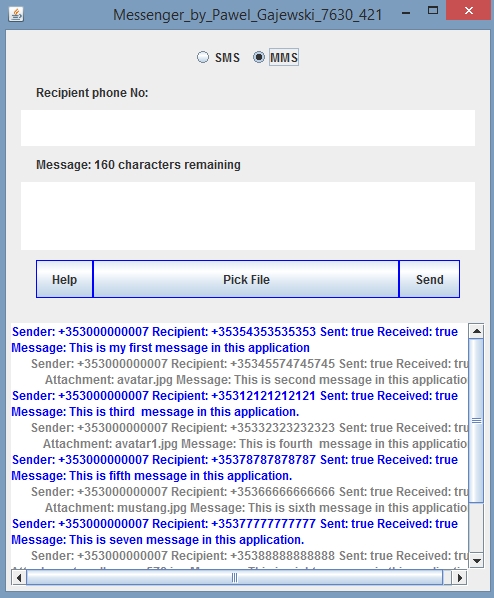
**Test 3.jpg**

****

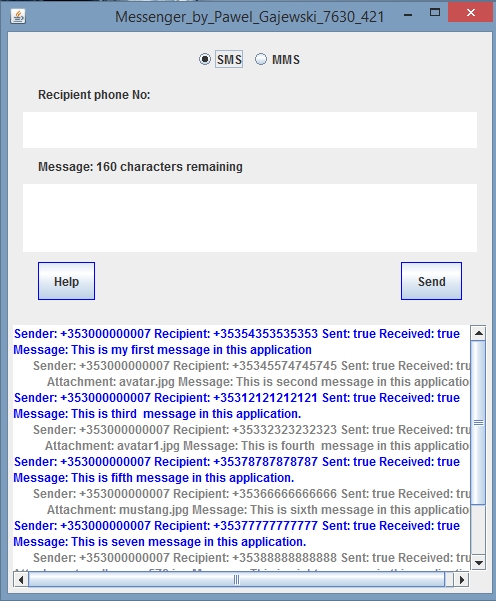
**Test 5.jpg**

****

**Test 7.jpg**

****

**Test 8.jpg**

****