

Software Implementation of a PC-Based Home Surveillance System

by

Murni Binti Masri

Final Report submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

JUNE 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

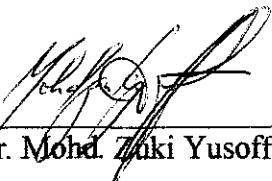
Software Implementation of a PC-Based Home Surveillance System

by

Murni Binti Masri

A project final report submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,



(Mr. Mohd. Zuki Yusoff)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



MURNI BINTI MASRI

ABSTRACT

This report is written upon the completion of the two semesters of Final Year Project course in Electrical & Electronic Engineering programme. This project is entitled as “Software Implementation of a PC-Based Home Surveillance System”. The main objective of this project is to design and build software system, emphasizing on GUI for the PC-based home surveillance system. The developed software is called the “Home Guard System”.

The “Home Guard System” is designed to scan and display a number of input signals using sensory interface and represents them on the computer interface. The input and output modules are interfaced with the computer via a serial port. Microcontroller is also implemented to translate and manipulate the data received from the serial port into meaningful functions for prototype demonstration.

The scope of study for this report mainly involves the software development process; the tools, services and packages available in Java that can be used for this project; and implementation of microcontrollers, specifically the PIC16F84 to be used in the prototype development and construction.

This project development is divided into three main modules, which are window module, hardware module and prototype module. In the window module, the GUI of this application is designed. The hardware module involves the communication establishment with the computer’s serial port, while the prototype module consists of a circuit that implement PIC16F84 as its processor.

This project has been successfully completed. Several recommendations are proposed to improve the current system, and come up with a better presentable, secure and reliable system with additional functions.

ACKNOWLEDGEMENTS

In the development of this 2-semester Final Year Project, it seems that an infinite number of people have provided immeasurable amount of guidance, idea and assistance. While the writer's gratitude goes out to all those that had assisted her, she could only mention a few of many benefactors here.

My greatest gratitude, thankfulness and appreciation to my supervisor, Mr Mohd. Zuki Yusoff for his great support, guidance and concern. Thank you for the continuous motivation that is given towards the development and completion of this project.

My deepest thanks to all my colleagues, who have always been there from the beginning till the end, and through ups and down. Thank you all for your words of encouragement to keep on going and overcome all the hurdles in making this project a success.

Special thanks is also conveyed to Miss Siti Hawa Talib, one of the most cooperative laboratory technician in UTP. Thank you for your cooperation and assistance in developing the project prototype.

And last but not least, my heartiest gratitude and appreciation to my family for their never-ending support and concern. They have given me the warmest helping-hand and inspired me the will to try my best for this project. My deepest thanks again to them and I apologize for all the lost time together.

It would be impossible to complete this project without the help from those mentioned above and the blessing of Allah SWT. Thank you and may Allah SWT bless us all.

TABLE OF CONTENTS

CERTIFICATION OF APPROVAL	ii
CERTIFICATION OF ORIGINALITY	iii
ABSTRACT	iv
ACKNOWLEDGEMENTS	v
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xi
FONT STANDARD	xii
CHAPTER 1: INTRODUCTION	1
1.1 Background of Study.....	1
1.2 Problem Statement.....	2
1.3 Objective of Study.....	3
1.4 Scope of Study.....	4
CHAPTER 2: LITERATURE REVIEW AND/OR THEORY	6
2.1 Programming Fundamentals	6
2.2 Java Foundation Classes.....	7
2.3 Java Communications API.....	8

2.4	Serial Ports	8
2.5	The PIC16F84 Fundamentals.....	10
CHAPTER 3:	METHODOLOGY/PROJECT WORK.....	11
3.1	Procedure Identification.....	11
3.1.1	Programming Process.....	11
3.1.2	Project Process Flow.....	13
3.1.3	Communicating with a Port.....	18
3.1.4	The PIC16F84 Implementation.....	21
3.2	Tools Required.....	22
3.2.1	Java 2 Software Development Kit (J2SDK).....	22
3.2.2	Java Integrated Development Environments (IDEs) And Other Softwares	24
3.2.3	The PIC16F84 circuit board	25
CHAPTER 4:	RESULTS AND DISCUSSION.....	27
4.1	GUI Development	27
4.1.1	Access Pad Frame	27
4.1.1.1	Password Field	28
4.1.1.2	Background Image	28
4.1.2	Main Frame	30
4.1.2.1	Tree Navigator Panel	32
4.1.2.2	Module Display Panel	34
4.2	Serial Port Programming	37
4.3	PIC16F84 Programming	38
4.4	Module Integration	42
CHAPTER 5:	CONCLUSION AND RECOMMENDATION.....	43
REFERENCES.....		47
APPENDIX A.....		49

APPENDIX B..... 50

APPENDIX C..... 51

APPENDIX D..... 52

APPENDIX E..... 57

APPENDIX F..... 60

APPENDIX G..... 65

LIST OF TABLES

Table A: Font Standard.....	xii
Table 2.1: D-Type 9 Pin and D-Type 25 Pin Connectors.....	9
Table 3.1: Swing Lightweight Controls	15
Table 3.2: Categories, Events and Interfaces	17
Table 3.3: Several Issues in Establishing Communication with Ports	20
Table 3.4: Packages of the Java API	23
Table 3.5: PIC16F84 Circuit Board Components	25
Table D.1: User Action, Source Object, and Event Type	53
Table D.2: Events, Event Listeners, and Listener Methods	55

LIST OF FIGURES

Figure 2.1: Pinout position of a male DB9 connector.....	9
Figure 2.2: Pin configuration of PIC16F84	10
Figure 3.1: The Iteration cycle programming	11
Figure 3.2: Project Process Flow.....	14
Figure 3.3: The hierarchy of event objects	16
Figure 3.4: Schematic of prototype circuit	26
Figure 4.1: Access Pad Frame	27
Figure 4.2: Invalid password pop up.....	28
Figure 4.3: Valid password pop up.....	28
Figure 4.4: The original floor plan	29
Figure 4.5: Background image generated.....	29
Figure 4.6: Access pad with truncated password field.....	29
Figure 4.7: Access pad with fully visible password field.....	30
Figure 4.8: Draft of the main frame design of the application.....	30
Figure 4.9: An initial design of the upper panel.....	32
Figure 4.10: The tree navigator panel.....	33
Figure 4.11: The modified tree design.....	34
Figure 4.12: Initial design of the module view	35
Figure 4.13: Final design of the module view	36
Figure 4.14: The main frame design	36
Figure 4.15: A test module for serial port communication programming	38
Figure 4.16: PIC serial data processing	40
Figure 5.1: Recommended schematic for prototype module using 2 PICs	45
Figure 5.2: Serial data processing using 2 PICs	46
Figure D.1: An event is an object of the <i>EventObject</i> class	53
Figure D.2: Event-handling	54
Figure E.1: Define a thread class by implementing the <i>Runnable</i> interface	57
Figure E.2: Thread states	58

LIST OF ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
API	Application Programmer Interface
AWT	Abstract Window Toolkit
CD	Carrier Detect
CTS	Clear To Send
DCE	Data Communication Equipment
DSR	Data Set Ready
DTE	Data Terminating Equipment
DTR	Data Terminal Ready
EEPROM	Electrically Erasable Programmable Read-Only Memory
GUI	Graphical User Interface
IDE	Integrated Development Environment
JDK	Java Development Kit
JFC	Java Foundation Classes
J2SDK	Java 2 Software Development Kit
LED	Light Emitting Diode
OSC	Oscillator
PC	Personal Computer
PIC	Peripheral Interface Controller
RD	Receive Data
RI	Ring Indicator
RTS	Request To Send
SG	Signal Ground
TD	Transmit Data
XT	External

FONT STANDARD

This report will be using many types of Java programming anatomy, such as packages, classes, methods and etc. Therefore, to differentiate the types of anatomy represented by a word, different font standard is utilized to help the reader in understanding the whole project process flow that is conveyed in this report. Table A below summarized the font standard for this report.

Table A: Font Standard

Anatomy	Font Standard	Examples
Methods/ Packages	<ul style="list-style-type: none">- Times New Roman- Size 12- Bold	<ul style="list-style-type: none">- open()- java.util
Terms	<ul style="list-style-type: none">- Times New Roman- Size 12- Italic	<ul style="list-style-type: none">- <i>callback</i>- <i>leaf</i>
Classes/ Components/ Objects	<ul style="list-style-type: none">- Times New Roman- Size 12- Bold & Italic	<ul style="list-style-type: none">- <i>CommPort</i>- <i>JComponent</i>- <i>mLivingView</i>
Codes	<ul style="list-style-type: none">- Courier New- Size 8	<pre>- try { SerialPort modem = (SerialPort) cpi.open(); } catch (PortInUseException e) {}</pre>

CHAPTER 1

INTRODUCTION

1.1 Background of Study

Smart House, Smart Home, Intelligent House and Home Automation are all referring to the same thing. In this particular project, the term chosen is the Home Guard System. Is this new or "experimental" technology? Certainly not, in fact, the equipment used has been used in office buildings, shopping malls and luxury homes for over a decade. According to the general analysis done, the Smart House System is already well known and implemented in the world especially in the United States of America (USA) and United Kingdom (UK) with various configurations. However, the level of implementation in Malaysia is still remaining as a question mark.

This project, entitled as "Software Implementation of a PC-Based Home Surveillance System", is designated for two semesters of Final Year Project course in Electrical & Electronic Engineering programme. Since this project emphasize on the Graphical User Interface (GUI) design, programming skill is essential in delivering better software for user. Development and improvement of programming skills are expected throughout the learning process of this project. The study is concentrated more on developing codes for GUI and how to handle each component of the GUI to perform security tasks by monitoring and controlling the inputs and outputs from the home appliances and devices connected to one of the computer port. The study will be based on Internet resources, programmer community resources, web-documentation, journals and books. Consultation with people who have software development background is also necessary throughout the project development.

This project presents a platform for the students to dive further in the world of programming and software development, which will be one of an extra requirement as a Computer System Engineer. The experience during industrial training at Motorola has given a basic knowledge in establishing communication with the serial port using C++. The knowledge obtained and experience gained contributes in assisting the development of this project.

1.2 Problem Statement

Software systems are used and implemented almost anywhere. Most electrical equipment now includes software as part of their interfaces, controlling or operating system. Better solutions and software designs are developed to provide better user-friendly application for users as well as satisfying the market demand.

The “Home Guard System” is designed to scan and display a number of input signals using sensory interface and represents them on the computer interface. The PC-based surveillance system consists of a number of sensors designed to monitor physical parameters and a number of devices controlled by the computer. The input and output modules are interfaced with the computer via a serial port. Examples of input and output modules are:

- a) Temperature sensor and display.
- b) Door/window sensor, which checks for any damages or failure.
- c) Smoke sensor.
- d) Magnetic card reader, which allows entry of authorized persons only.
- e) Fan, which is turned on and off by the computer depending on the temperature.
- f) Search light, which is turned on and off by the computer, at a specific time given by the user
- g) Lock, which is opened by the computer, when a valid card is inserted in the magnetic card reader.

A GUI application is needed to constantly monitor the status of all the modules existed in the systems as well as control the operation on part of these modules.

At the early stage of this project, the student has to come up with a suitable GUI design concept for the application and construct it using the resources available and methods discussed in previous semester. The design level of this application depends on the consideration to the user needs, the programmer's expertise and their experience using computers.

The basic principles considered in designing the GUI concept are:

- a) Friendliness - the GUI should use terms and input techniques that are known to the users.
- b) Consistency - similar operations should produce similar results. This helps to minimize memory requirement for the user and reduces surprises.
- c) Error response - the GUI should be able to recover from user error.
- d) User guidance - the GUI should provide feedback, meaning that the user should know that his commands have been accepted. The GUI visual presentation should be obvious to the user and instruct the user how to use the program.

1.3 Objective of Study

The early main objective of this project is to design and build software system, emphasizing on GUI for the PC-based home surveillance system only. However, several extended features are added throughout the development process this project. Ultimately, the objectives of this project are summarized into:

- a) To design and build software system for the PC-based surveillance system. Study on several different approaches involved in each stage of the software development process, which are from choosing an appropriate programming language up to releasing the final product of the software system are conducted and implemented. The study conducted is centred intensely on the services

available in Java to assist and support the project design. The Swing components are the basic components that will be used for the development of this application.

- b) Implementing the Java Communications API 2.0. It is a standard extension that is used to establish communication with ports available on the computer. In this project, the study is concentrated on establishing communication with a serial port only.
- c) Constructing a prototype, basically built up of a simple circuit board to demonstrate the workability of the software to communicate with the serial port.
- d) Implementing the use of microcontrollers to translate and manipulate the data received from the serial port into meaningful functions for prototype demonstration.

1.4 Scope of Study

A computer application can be developed using various ranges of available programming languages, depending on the needs and complexity of the application, as well as the preference of the programmer or developer themselves. A research on the programming language is conducted to have a better view on the advantages of associated programming languages and the potentially developed skill of the students upon the completion of this project.

A study on the software development process, the tools involved in the development process, and other suitable integrated development environments (IDEs) are also conducted. Further study is carried out regarding the tools, services and packages available in Java that can be used for this project.

This project is scheduled to be completed within 2 semesters. The first semester is used for research and learning process involved in Java software development and the services available for the development process such as the Java 2 Documentation of J2SDK SE v 1.3.1. The second half of the 2-scheduled semesters will involve implementation and manipulation of the developed skills to design and construct the application GUI, control assignment to the components of the GUI and the establishment of communication between the computer, serial port and the microprocessor in order to monitor and control the home appliances.

The scheduled tasks and milestones for the first semester and second semester of the 2-scheduled semesters of the final year project are summarized in the Gantt Chart of **Appendix A** and **Appendix B** respectively.

CHAPTER 2

LITERATURE REVIEW AND/OR THEORY

2.1 Programming Fundamentals

To construct any computer-based system, some processes are needed to translate the idea for the use of the computer into lines of source code which can be compiled and executed. This process typically includes the tasks of:

- a) requirements gathering - what we would like the system to do;
- b) analysis - finding out how the system should behave;
- c) design - deciding the structure of the system to be constructed;
- d) implementation - writing the source code;
- e) testing, verification and validation - making sure the system does what we claim.

The tasks of implementation and testing also include the task of debugging, which is the finding and removing of errors in the program.

The design of a language like Java is based on principles that are the result of both many years of research and of the practical use of earlier generations of programming languages. The research addresses not only the best ways of making the computer behave as we want it to, but also how best to avoid the errors that human beings, being imperfect, introduce into the systems they are developing. Thus, the programming language and the development tools used for constructing programs try to prevent the programmer making errors in the first place and, if errors are introduced, help finding and eradicating them quickly and efficiently. The features of Java and the tools for developing Java programs support these principles. (Russel Winder & Graham Roberts, 1998)

2.2 Java Foundation Classes

The Java Foundation Classes (JFC) is a new set of GUI-related classes created to solve the AWT problem of platform peculiarity. JFC also supports:

- a) A pluggable look and feel, meaning that when the program is run, user can choose whether he want it to look like Windows GUI, a Macintosh GUI, or some other style.
- b) An accessibility API for things like larger text for the visually impaired.
- c) The Java 2D drawing an API.
- d) A drag-and-drop library and an “undo last command” library.
- e) The Swing components set.

The Swing components (scrollbar, button, textfield, label, etc.) replace the AWT versions of these components. The AWT is still used for other areas of GUI functionality, like layout control and printing. The AWT are simpler than the Swing components, but more basic and more bug-prone.

In AWT, all components are based on peer components. A Java AWT button really is a Win32 button on Windows. This is termed a heavyweight component. A lightweight component, like all the Swing *JComponents*, is one which doesn't use a peer or native component. Instead, it is drawn by Java code on a piece of the screen that already belongs to Java. It is drawn onto its container in fact. The most important differences are:

- a) Lightweight components can have transparent areas in them, so they don't have to look rectangular in shape.
- b) Mouse events on the lightweight component are delivered to its container.
- c) When they overlap, lightweight components are never drawn on top of heavyweight components. This is because we can't draw half of a lightweight component on one component and the other half on another. Lightweights exist wholly within their parent heavyweight component.

Poor behaviour when overlapping is the main reason JavaSoft gives for recommending that we should not mix Swing *JComponents* with AWT components.

2.3 Java Communications API

The Java Communications API 2.0 is a standard extension available in Java 1.1 and later that allows Java applications (but not applets) to send and receive data to and from the serial and parallel ports of the host computer. The Java Communications API operates at a very low level. It only understands how to send and receive bytes by these ports. It does not understand anything about what these bytes mean. Doing useful work generally requires not only understanding the Java Communications API, but also the protocols spoken by the devices connected to the ports.[7]

Because the Java Communications API is a standard extension, it is not installed by default with the JDK. It has to be downloaded from <http://java.sun.com/products/javacomm/index.html> and installed separately.

2.4 Serial Ports

Strictly the RS232 standard specifies the names and functions of signals between Data Terminating Equipment (DTE) and Data Communication Equipment (DCE), and the gender of the connectors used. It does not specify the connector type or the communication protocol employed. The interface is now widely used for connecting instruments to computers and "RS232" is used to imply specific connector patterns and communication protocols.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. **Figure 2.1** is the view looking into a male DB9 connector.

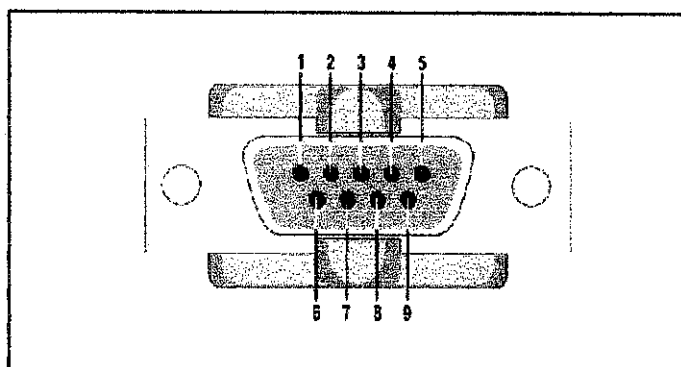


Figure 2.1: Pinout position of a male DB9 connector.

Table 2.1 shows the signal names and their corresponding pins to the maximum of 9 wires used for instrumentation applications. Not all wires are required in all applications. All signals are named from the viewpoint of the DTE. Thus the TD line is used by the DTE to transmit data to the DCE, whilst it uses the RD line to receive data from the DCE. (Hence the DCE is transmitting on the RD line and receiving on the TD line).

Table 2.1: D-Type 9 Pin and D-Type 25 Pin Connectors

D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

A logic 0 on the TD and RD lines, or a control signal "on" on the RTS, CTS, DSR, DCD and DTR lines is represented by a voltage in the range +5V to +15V at the source end, and must be > +3V at the receiving end of the cable. The converse signal must be in the range -5V to -15V at the source and must be < -3V at the receiving end.

2.5 The PIC16F84 Fundamentals

The PIC16F84 is an 18-pin 14-bit embedded micro featuring electronically erasable programmable read-only memory (EEPROM), as shown in **Figure 2.2**. The PIC16F84 features two ports named A and B having five and eight digital lines respectively. Any line can be configured to be an input or output. The pinout description of PIC16F8X is listed in **Appendix C**. The PIC16F84 has no serial port but with some hardware and programming, PIC-to-PC serial communication can be established. The PIC can send or receive 8-bit values at prescribed intervals (baud rate).

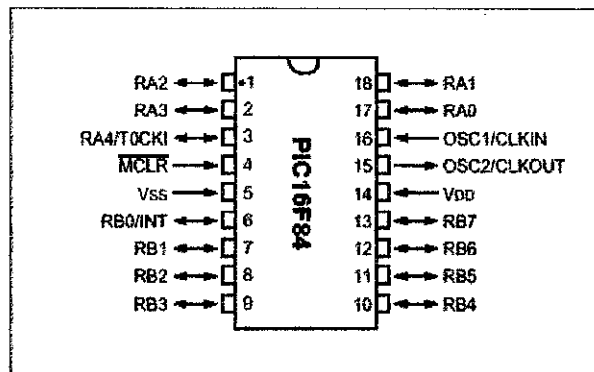


Figure 2.2: Pin configuration of PIC16F84

PIC16F84 perfectly fits many uses, from automotive industries and controlling home appliances to industrial instruments, remote sensors, electrical door locks and safety devices. It is also ideal for smart cards as well as for battery supplied devices because of its low consumption.

In System Programmability of this chip (along with using only two pins in data transfer) makes possible the flexibility of a product, after assembling and testing have been completed. This capability can be used to create assembly-line production, to store calibration data available only after final testing, or it can be used to improve programs on finished products.

CHAPTER 3

METHODOLOGY/PROJECT WORK

3.1 Procedure Identification

3.1.1 Programming Process

A small-scale program is typically under a thousand lines of source code (excluding comments). We need to distinguish small-scale from large-scale as, although the key object-oriented ideas remain just as important, large-scale programs are typically developed by a team of people and require considerably more design effort.

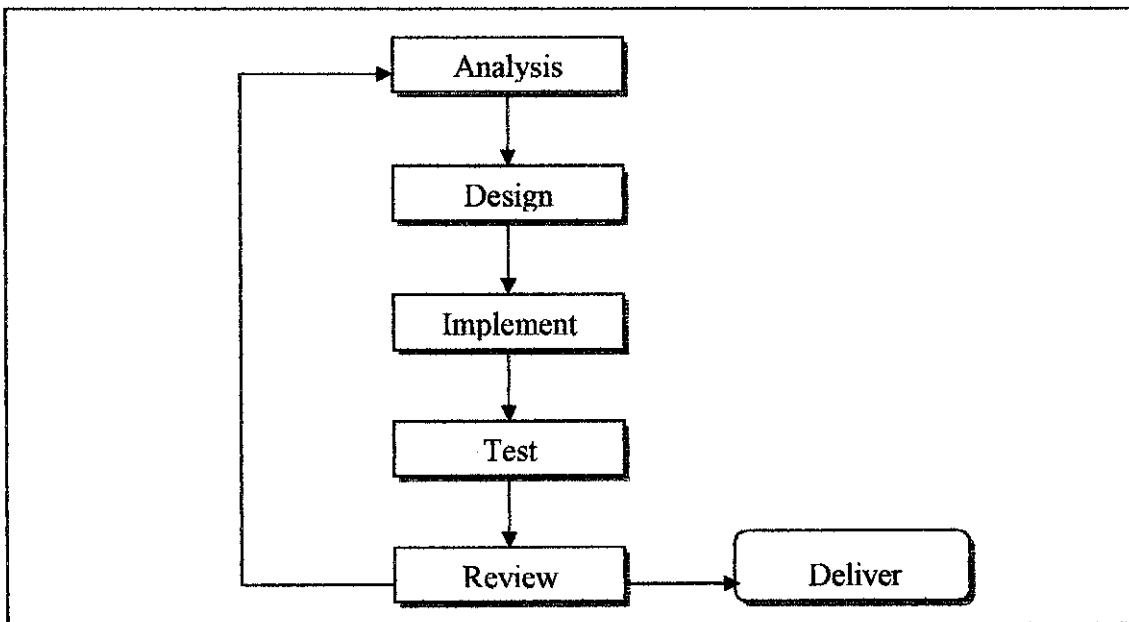


Figure 3.1: The iteration cycle programming

An overall process is needed to order the steps, and determine what to do when. An *iterative* approach is normally adopted. This process is often characterized as “Analyze a little, Design a little, Program a little, Review and Repeat until finished” as illustrated in **Figure 3.1**.

Iteration is important as it is very hard to get the design of a program correct first time. In particular, it is difficult to identify all the key abstractions right at the start. Often they only become apparent as understanding of the program developed with each stage of iteration.

There are dangers with iteration in that it can be difficult to control the quality and scope of the program, and also it can be hard to know when it is finished since there is always a temptation to add more features to the design. Problems can be avoided by pausing regularly to review progress, usually after having implemented some new aspect of the program.

A further consequence of iteration and prototyping is that, although the various development stages are listed separately, they may be merged together or omitted. In particular, for small programs analysis and design can be treated as essentially the same thing, with the design itself being developed by actually writing and commenting Java code.

At the early stage of this project, the student has to come up with a suitable GUI design concept for the application and construct it using the resources available and methods discussed in previous semester. The design level of this application depends on the consideration to the user needs, the programmer’s expertise and their experience using computers.

The basic principles considered in designing the GUI concept are:

- a) Friendliness - the GUI should use terms and input techniques that are known to the users.
- b) Consistency - similar operations should produce similar results. This helps to minimize memory requirement for the user and reduces surprises.
- c) Error response - the GUI should be able to recover from user error.
- d) User guidance - the GUI should provide feedback, meaning that the user should know that his commands have been accepted. The GUI visual presentation should be obvious to the user and instruct the user how to use the program.

3.1.2 Project Process Flow

This project development is divided into three main modules, which are window module, hardware module and prototype module. These modules are divided into different timeframes of the overall time allocated for this project. However, the development process might overlap with each other due to time constraint and design problems. The main project process flow is summarized in **Figure 3.2**.

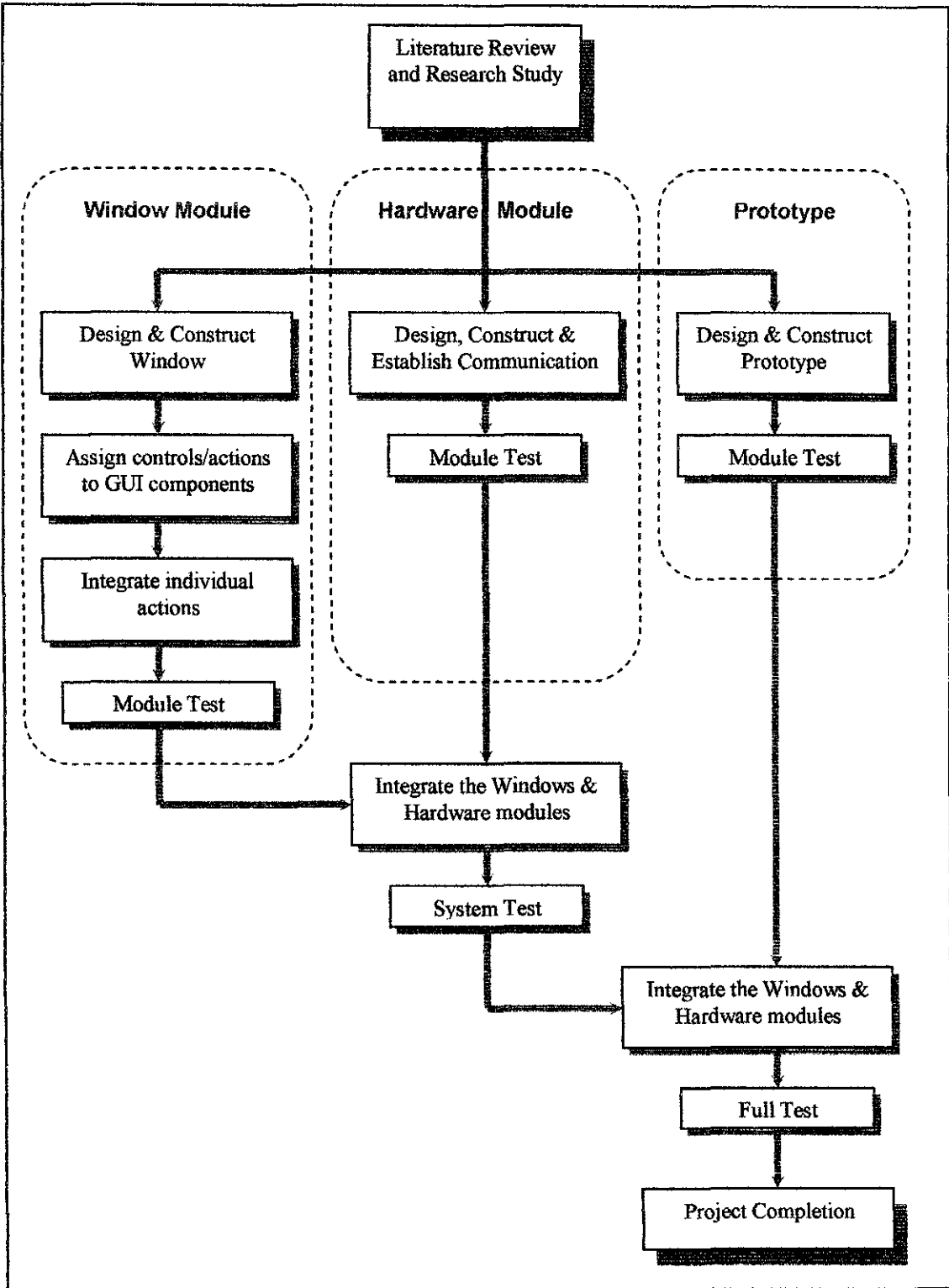


Figure 3.2: Project Process Flow

The window module will be constructed first, followed by the hardware module and prototype module. In the window module, the GUI of this application shall be design, constructed and integrated. This module will be built up using Swing components. Swing components are no longer peer-based, but are written in Java and are thus consistent on all platforms. The main Swing lightweight controls that can be implemented in the design are as listed in **Table 3.1**. The behaviour and appearance of each specific control is one level down in the subclasses of *JComponent*. These classes are the controls or building blocks from which GUIs are created. To use these components, the steps taken are:

- a) Add them to the content pane of a container.
- b) Register an event-handler using the **addSomeListener()** method of the control.

Table 3.1: Swing Lightweight Controls

GUI Category	Control	Swing Class Name
Basic Controls	Button	<i>JButton, JCheckBox, JRadioButton</i>
	Combo box	<i>JComboBox</i>
	List	<i>JList</i>
	Menu	<i>JMenu, JMenuBar, JMenuItem</i>
	Slider	<i>JSlider</i>
	Toolbar	<i>JToolBar</i>
	Text field	<i>JTextField, JPasswordField, JTextArea</i>
Uneditable Displays	Label	<i>JLabel</i>
	Tooltip	<i>JToolTip</i>
	Progress bar	<i>JProgressBar</i>
Editable Displays	Table	<i>JTable</i>
	Text	<i>JTextPane, JTextArea, JEditorPane</i>
	Tree	<i>JTree</i>
	Color chooser	<i>JColorChooser</i>
	File chooser	<i>JFileChooser</i>
Space-Saving Containers	Scroll pane	<i>JScrollPane, JScrollBar</i>
	Split pane	<i>JSplitPane</i>
	Tabbed pane	<i>JTabbedPane</i>
Top-Level Containers	Frame	<i>JFrame</i>
	Applet	<i>JApplet</i>
	Dialog	<i>JDialog, JOptionPane</i>
Other Containers	Panel	<i>JPanel</i>
	Internal frame	<i>JInternalFrame</i>
	Layered pane	<i>JLayeredPane</i>
	Root pane	<i>JRootPane</i>

The window concept is design by drawing the basic features used for the main page. This draft shall be used as guideline in developing the window using Swing components.

Then, each component in the constructed window shall be assign to appropriate controls or actions. In event-driven programming, the logic of the codes is inverted. Instead of one flow of control from beginning to end, the runtime system sits in a “window main loop” simply waiting for user input. When the user clicks the mouse, the operation system passes it to the window manager, which turns it into an *event* and passes it on to a handler supplied earlier. This is known as a *callback*. Our handler is the *callback routine*, because the window system calls back to it when the event happens. Our event handler will deal with the graphics event and any work that is associated with it.

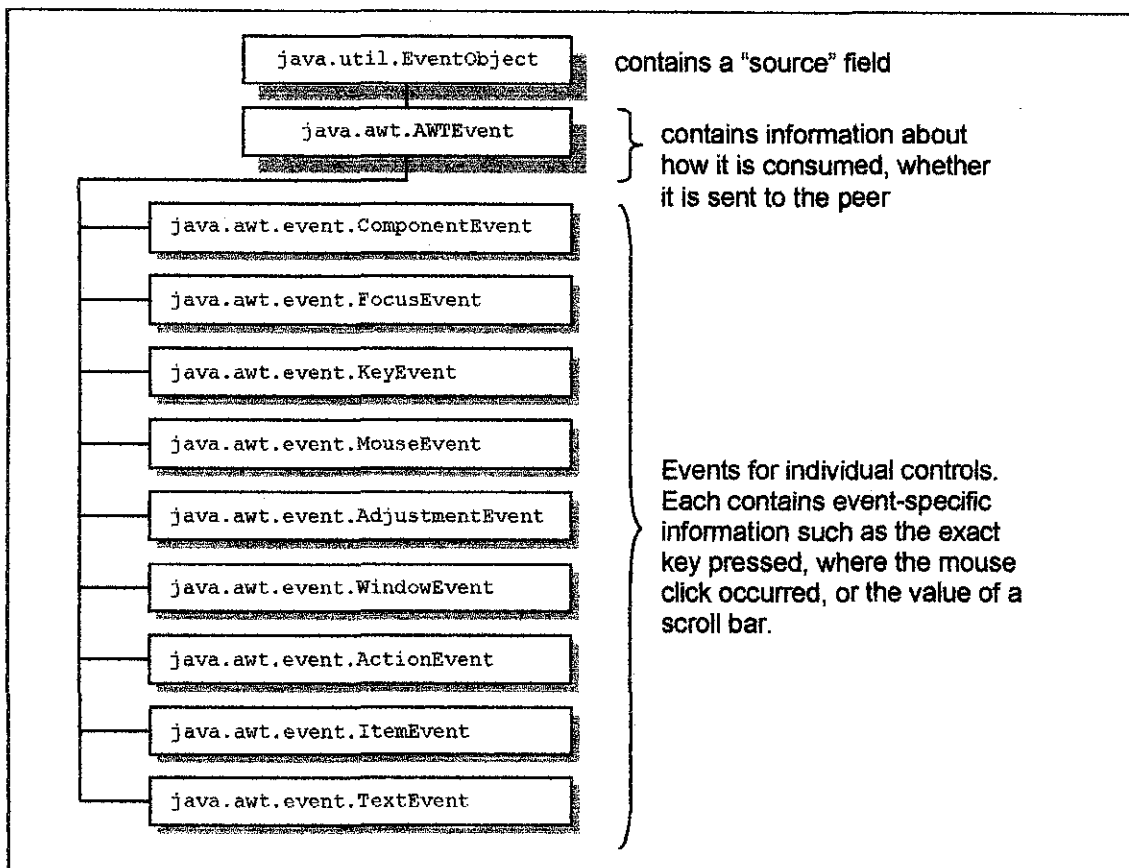


Figure 3.3: The hierarchy of event objects.

An event object has several data fields holding certain information. There is a general `java.util.EventObject` type, and all AWT events are children of that as shown in **Figure 3.3** above.

Table 3.2 summarizes the relations between existing event, their interfaces and the methods that *Listener* interfaces use. A more detail description about event-driven programming is discussed in **Appendix D**.

Table 3.2: Categories, Events and Interfaces.

General Category	Events That It Generates	Interface That The Event-Handler Implements
Mouse	Dragging, moving mouse causes a <i>MouseEvent</i>	<i>MouseMotionListener</i>
	Clicking, selecting, releasing causes a <i>MouseEvent</i> .	<i>MouseListener</i>
Keyboard	Key press or release causes a <i>KeyEvent</i> .	<i>KeyListener</i>
Selecting (an item from a list, checkbox, etc.)	When item is selected causes an <i>ItemEvent</i> .	<i>ItemListener</i>
Text Input Controls	When newline is centered causes a <i>TextEvent</i> .	<i>TextListener</i>
Scrolling Controls	When a scrollbar slider is moved causes an <i>AdjustmentEvent</i> .	<i>AdjustmentListener</i>
Other Controls (button, menu, etc.)	When pressed causes an <i>ActionEvent</i> .	<i>ActionListener</i>
Window Changes	Open, close, iconify, etc., causes a <i>WindowEvent</i> .	<i>WindowListener</i>
Keyboard Focus Changes	Tabbing to next field or requesting focus causes a <i>FocusEvent</i> . A component must have the focus to generate key events.	<i>FocusListener</i>
Component Change	Resizing, hiding, revealing, or moving a component causes a <i>ComponentEvent</i> .	<i>ComponentListener</i>
Component Change	Adding or removing a component to a container causes a <i>ContainerEvent</i> .	<i>ContainerListener</i>

The hardware module involves serial communication establishment via a serial port on the computer. In addition, microprocessor will also be added as part of the hardware module. The microprocessor will be utilized to interact with the software constructed through serial port on the computer. PIC16F84 perfectly fits the usage for serial communication and controlling home appliances and safety devices.

3.1.3 Communicating with a Port

The main thing to do before programming any Java files for port communication is to download the extension and install it on the computer. The steps taken to install the Java Communications API are:

- a) Unzip the file javacomm20-win32.zip file. This will produce a hierarchy with a top level directory commapi folder.
- b) Copy win32com.dll file to <JDK>\bin directory.
- c) Copy comm.jar file to <JDK>\lib directory.
- d) Copy javax.comm.properties file to <JDK>\lib directory.
- e) The javax.comm.properties file must be installed. If it is not, no ports will be found by the system.
- f) Add comm.jar file to classpath.

The *javax.comm.SerialPort* class is an abstract subclass of *CommPort* that provides various methods and constants useful for working with RS232 serial ports and devices. The main purposes of the class are to allow the programmer to inspect, adjust, and monitor changes in the settings of the serial port. Simple input and output is accomplished with the methods of the super class, *CommPort*. *SerialPort* has a public constructor, but shouldn't be used by applications. Instead, one should call the `open()` method of a *CommPortIdentifier* that maps to the port he wants to communicate with, then cast the result to *SerialPort*.

For example:

```
CommPortIdentifier cpi = CommPortIdentifier.getPortIdentifier("COM1");
if (cpi.getType() == CommPortIdentifier.PORT_SERIAL) {
    try {
        SerialPort modem = (SerialPort) cpi.open();
    }
    catch (PortInUseException e) {}
}
```

javax.comm is divided into high-level and low-level classes. High-level classes are responsible for controlling access to and ownership of the communication ports and performing basic I/O. The *CommPortIdentifier* class lets us find and open the ports available on the system. The *CommPort* class provides input and output streams connected to the ports. Low-level classes, *javax.comm.SerialPort* and *javax.comm.ParallelPort* for example, manage interaction with particular kinds of ports and help us read and write the control wires on the ports. They also provide event-based notification of changes to the state of the port.

There are several issues in establishing communication with ports using the **javax.comm** package. Main concern involve in establishing a communication with a port are identifying ports, finding the ports, getting information about a port, opening ports, waiting for a port with a port ownership events, and registering ports. **Table 3.3** summarizes these issues with brief description and also listed the packages, classes or methods that can be used to perform those jobs.

There are five basic steps to communicating with a port:

- a) Open the port using **open()** method of *CommPortIdentifier*. If the port is available, this returns a *CommPort* object. Otherwise, a **PortInUseException** is thrown.
- b) Get the port's output stream using the **getOutputStream()** method of *CommPort*.
- c) Get the port's input stream using the **getInputStream()** method of *CommPort*.
- d) Read and write data onto those streams as desired.
- e) Close the port using the **close()** method of *CommPort*.

Table 3.3: Several Issues in Establishing Communication with Ports

Event Issue	Packages/Classes/Methods
<p>Identifying Ports</p> <ul style="list-style-type: none"> - lists the available ports, figure out which program owns them, take control of a port, and open a port so that I/O can be perform with it. 	<ul style="list-style-type: none"> - javax.comm.CommPortIdentifier - javax.comm.CommPort
<p>Finding Ports</p> <ul style="list-style-type: none"> - find and create the right port using a port identifier. 	<ul style="list-style-type: none"> - javax.comm.CommPortIdentifier <ul style="list-style-type: none"> • public static Enumeration getPortIdentifiers() • public static CommPortIdentifier getPortIdentifier(String portName) throws NoSuchPortException • public static CommPortIdentifier getPortIdentifier(CommPort port) throws NoSuchPortException
<p>Getting Information About a Port</p> <ul style="list-style-type: none"> - once a particular port is identified by CommPortIdentifier, the information about the port can be obtained by calling several accessor methods. 	<ul style="list-style-type: none"> - CommPortIdentifier <ul style="list-style-type: none"> • public String getName () • public int getPortType() • public String getCurrentOwner() • public boolean isCurrentlyOwned()
<p>Opening Ports</p> <ul style="list-style-type: none"> - a port has to be open before it can be read from or written to. - opening a port gives application exclusive access to the port, until the port is given up or the program ends. 	<ul style="list-style-type: none"> - CommPortIdentifier <ul style="list-style-type: none"> • public synchronized CommPort open(String name, int timeout) throws PortInUseException
<p>Waiting for a Port with Port Ownership Events</p> <ul style="list-style-type: none"> - two methods used to receive notification of changes in ownership of the port. - port ownership events are fired to signal that : <ul style="list-style-type: none"> • a port has been opened. • a port has been closed. • another application wants to take control of the port. - listener must be registered to listen for ownership changes on a particular port. 	<ul style="list-style-type: none"> - CommPortIdentifier - javax.comm.CommPortOwnershipListener <ul style="list-style-type: none"> • public void addPortOwnershipListener (CommPortOwnershipListener listener) • public void removePortOwnershipListener (CommPortOwnershipListener listener) • public abstract void ownershipChange(int type)
<p>Registering Ports</p> <ul style="list-style-type: none"> - register a particular name, type, and driver with the Comm API so that it can be returned by CommPortIdentifier.getPortIdentifiers() 	<ul style="list-style-type: none"> - CommPortIdentifier <ul style="list-style-type: none"> • public static void addPortName(String portName, int portType, CommDriver driver)

3.1.4 The PIC16F84 Implementation

A website that provides tutorial on PIC16F84 fundamentals; <http://www.boondog.com//tutorials/pic16F84/pic16f84.html> was studied. Learning a microprocessor's capabilities often demands such a setup where light emitting diodes (LEDs) turn on/off, blink at desired rates and respond to switches. This tutorial is very focused because its purpose is to rapidly acquaint us with the fundamentals needed to develop PIC16F84-based applications. This is achieved with focused hands-on exercises exploring:

- a) PIC input/output (I/O) ports: LEDs are turned on and off and switches are read
- b) PIC timer: An LED is blinked at a desired rate
- c) PIC serial communication: ASCII characters are sent between a PC and PIC

The PIC16f84 is an 18-pin 14-bit embedded micro featuring electronically erasable programmable read-only memory (EEPROM). The essential steps in the development cycle of this microchip are:

- a) On a PC, type the program, successfully compile it and then generate the .HEX file.
- b) Using a PIC16F84 device programmer, upload the .HEX file into the PIC16F84. This step is often called *burning*.
- c) Insert your PIC16F84 into your circuit, power up and verify the program works as expected. This step is often called *dropping* the chip. If it isn't, you must go to Step 1 and debug your program and repeat burning and dropping.

The PIC C Compiler program kit installed in the laboratory computer is used to burn the chips. The steps to burn a chip are:

- a) Open a file written in C language.
- b) Compile the file.
- c) Select *Program Chip*, and check the *XT* radio button under *OSC Options* field, indicating an external oscillator is used.
- d) Erase the chip placed on the burning board.

- e) Then, execute this process in sequence: *READ*, *BLANK*, *VERIFY*.
- f) Load the compiled *.c file.
- g) Execute *PROGRAM*.

3.2 Tools Required

3.2.1 Java 2 Software Development Kit (J2SDK)

To develop software, computer serves as an essential tool for this project. In order to use Java and get some programming done, developments tools are need. Java is distributed as the Java Development Kit (JDK). The basic JDK from SunSoft provides the tools as command line versions, meaning that they are generally used by typing in commands to a command interpreter (in MSDOS window if using Windows, an xterm if using UNIX, or something equivalent).

The following are some of the tools in the JDK:

- i) **javac** is the Java compiler. It is run to create the bytecode for applications and applets.
- ii) **java** is the Java interpreter. To run an application, the name of a class that contains the entry point of the application is supplied.
- iii) **jre** is also a Java interpreter, but is packaged separately from the JDK. It provides a run-time environment so that developers can include it with their application code without requiring users to install the complete JDK.

Java contains many predefined classes that are grouped into categories of related classes, called packages. These packages are referred as Java applications programming interface (Java API), or the Java class library. **import** statements is used to specify the class required to compile a Java program. For example, a program uses the statement

```
import javax.swing.JApplet;
```

to tell the compiler to load the *JApplet* class from the `javax.swing` package. One of the great strengths of Java is the large number of classes in the packages of the Java API that can be reused rather than reinventing the codes structure. Table 3.4 lists a subset of the many packages in the Java API and provides a brief description of each package.

The set of packages available in the Java 2 Software Development Kit (J2SDK) is quite large. In addition to the packages summarized in Table 3.4, the J2SDK includes packages for complex graphics, advanced graphical user interfaces, printing, advanced networking, security, database processing, multimedia, accessibility (for people with disabilities) and many other functions. Packages that are related for the development process of this project shall be studied more thoroughly and will be implemented in the project design.

Table 3.4: Packages of the Java API

Package	Description
<code>java.applet</code>	<i>The Java Applet Package</i> Contains the <code>Applet</code> class and several interfaces that enable the creation of applets, interaction of applets with the browser and playing audio clips. In Java 2, class <code>javax.swing.JApplet</code> is used to define an applet that uses the <i>Swing GUI components</i> .
<code>java.awt</code>	<i>The Java Abstract Windowing Toolkit Package</i> Contains the classes and interfaces required to create and manipulate graphical user interfaces in Java 1.0 and 1.1. In Java 2, these classes can still be used, but the <i>Swing GUI components</i> of the <code>javax.swing</code> packages are often used instead.
<code>java.awt.event</code>	<i>The Java Abstract Windowing Toolkit Event Package</i> Contains classes and interfaces that enable event handling for GUI components in both the <code>java.awt</code> and <code>javax.swing</code> packages.
<code>java.io</code>	<i>The Java Input/Output Package</i> Contains classes that enable programs to input and output data.
<code>java.lang</code>	<i>The Java Language Package</i> Contains classes and interfaces required by many Java programs and is automatically imported by the compiler into all programs
<code>java.net</code>	<i>The Java Networking Package</i> Contains classes that enable programs to communicate via networks.

Package	Description
java.text	<i>The Java Text Package</i> Contains classes and interfaces that enable a Java program to manipulate numbers, dates, characters and strings. It provides many of Java's internationalizing capabilities I.e., features that enable a program to be customized to a specific locale.
java.util	<i>The Java Utilities Package</i> Contains utility classes and interfaces, such as: date and time manipulations, random-number processing capabilities, storing and processing large amounts of data, breaking strings into smaller pieces called tokens and other capabilities.
java.swing	<i>The Java Swing GUI Components Package</i> Contains classes and interfaces for Java's Swing GUI components that provide support for portable GUIs.
java.swing.event	<i>The Java Swing Event Package</i> Contains classes and interfaces that enable event handling for GUI components in the javax.swing package.

3.2.2 Java Integrated Development Environments (IDEs) and Other Softwares

Many other vendors are now supplying Java integrated development environments (IDEs), usually with tools integrated in sophisticated graphical environments. A number of Java IDE products are available from IBM, Sun, Symantec, Borland, Microsoft, and other companies. These products will ease the programming and debugging process, especially programming related to GUI. JBuilder6 from Borland is used in this project. JBuilder 6 is used as part of the tools in this project. JBuilder is released by Borland. It is Windows-based application software. It increases productivity and provides output to the user. A JBuilder project organizes the files used and maintains the properties set. JBuilder stores projects with a **.jpx** or **.jpr** extension.

Apart from JBuilder, JCreator is also used in the software development process. JCreator™ is a trademark of Xinox Software. JCreator LE is a simple Java IDE for Windows. This freeware version of JCreator can be downloaded from <http://www.jcreator.com/>. JCreator runs on Windows machines and requires Sun's Java SDK installed. The system requirements for running JCreator 2.x are Microsoft Windows

95, 98, ME, XP, NT, or 2000, and a 133-MHz CPU with 32 MB of RAM. JCreator is written entirely in C++, which makes it fast and efficient compared to the Java based IDEs.

Ulead Photo Express 2.0 SE and Microsoft Paint are used for developing images and logos. Photo Express helps us to create projects quickly with an easy-to-understand visual interface. Additionally, there are numerous edges, frames, textures and backgrounds, making it more inspired and easy to create a wide variety of image projects.

3.2.3 The PIC16F84 circuit board

The PIC16F84 features two ports named A and B having five and eight digital lines respectively. Any line can be configured to be an input or output. The parts list used to build the prototype circuit board in order to demonstrate the workability of the designed software is given in Table 3.5.

Table 3.5: PIC16F84 Circuit Board Components

Part Description	Quantity
PIC16F84-04/P	1
PUSHBUTTON SWITCH	5
4 MHZ CRYSTAL CLOCK OSCILLATOR	1
0.1 UF CAP	1
0.1 INCH HEADERS	1
LED	5
100 OHM RESISTOR	1
10 KILO OHM RESISTOR	1
220 OHM RESISTOR	5
6 INCH PROTOTYPING CIRCUIT BOARD	1
DB9 RIGHT ANGLE FEMALE CONNECTOR	1
SERIAL CABLE MALE/FEMALE DB9	1
MAX233CPP RS-232 DRIVER/RECEIVER	1

The schematic of the circuit built is as shown in Figure 3.4. The circuit is built on a breadboard first before transferring it to a prototyping circuit board.

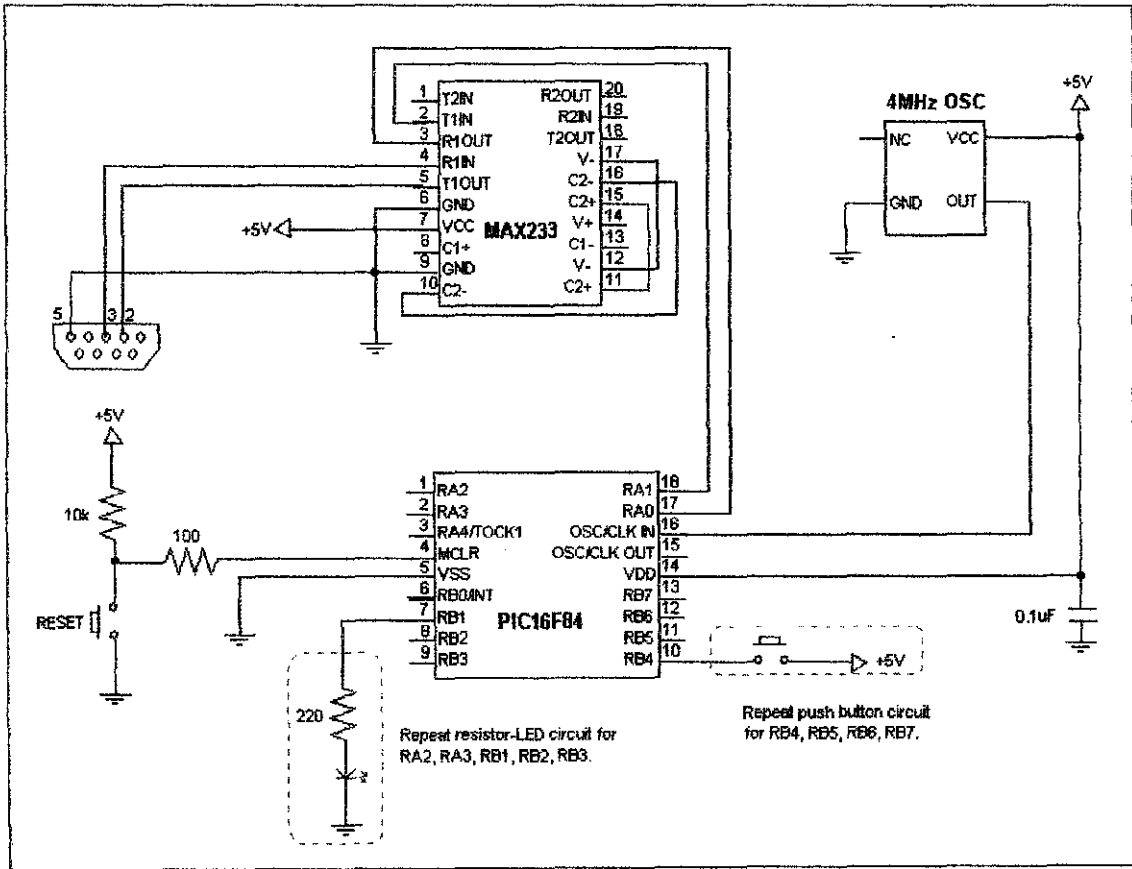


Figure 3.4: Schematic of prototype circuit.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 GUI Development

The GUI for this project is developed in several stages. The results and discussion are break down into several sections, in accordance with the design flow of the GUI. Mainly, it is categorized into Access Pad Frame and Main Frame.

4.1.1 Access Pad Frame

In order to access the program, the user must enter a password for security purposes. Hence, an access pad as shown in **Figure 4.1**, is created to obtain the right password from the user, before permitting any access to the program.

MyAccessPad is an extends of a *JFrame*. Its design mainly consists of 2 components:

- a) a password field
- b) a background image.

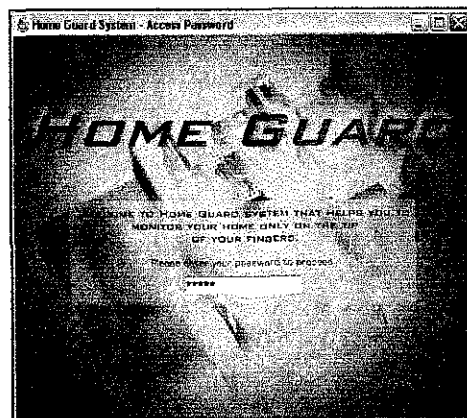


Figure 4.1: Access Pad Frame.

4.1.1.1 Password Field

Swing's password field conceals its text by displaying an '*' for every character entered in the field. The asterisk is referred to as an echo character and can be set after a password field is constructed. *JPassword* provides the same set of constructors as its superclass, *JTextField*. The *JPasswordField* constructors invoke the superclass constructors and set the echo character to the '*'. The `getPassword()` method is used to obtain the password instead of the `getText()` methods inherited from *JTextField*, which are deprecated in *JPasswordField* to provide a compile-time warning.

The `getPassword()` method is used to obtain the password instead of the `getText()` methods inherited from *JTextField*, which are deprecated in *JPasswordField* to provide a compile-time warning.

If the password is incorrect, a message dialog box displays "Your password is not valid. Please try again." as shown in **Figure 4.2**. The user will be prompt back to the access pad, until the right password is gained. If the password is correct, the message dialog box displays "Welcome to Home Guard system." as shown in **Figure 4.3**. The access pad will be hidden, and the user will then have access to the program through the main frame visible.

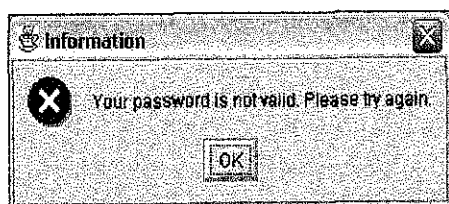


Figure 4.2: Invalid password popup.

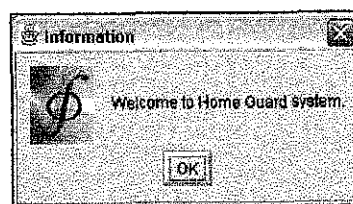


Figure 4.3: Valid password popup.

4.1.1.2 Background Image

A floor plan image obtained from the internet is used to design a background image for the access pad. The image in **Figure 4.5** is generated from the floor plan image shown in **Figure 4.4**, using Ulead Photo Express 2.0 SE. The rest of the images used in the

development of this application are mainly designed using Ulead Photo Express 2.0 SE and Microsoft Paint.

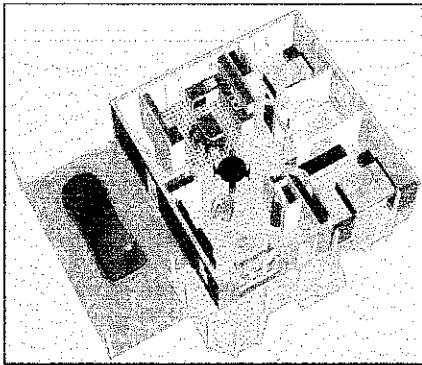


Figure 4.4: The original floor plan.

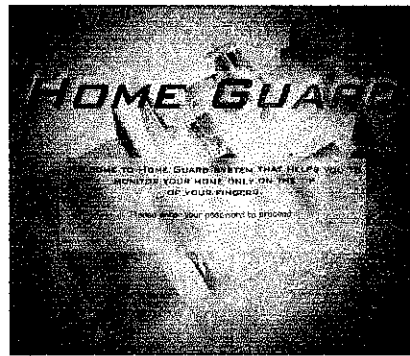


Figure 4.5: Background image generated.

At first, the image and the password field are added to the access pad using the `add()` method. The result is not as expected, wherein the password field is truncated when the access pad is initialized, as shown in **Figure 4.6**. The problem was then overcome by using the `paintComponent()` method.

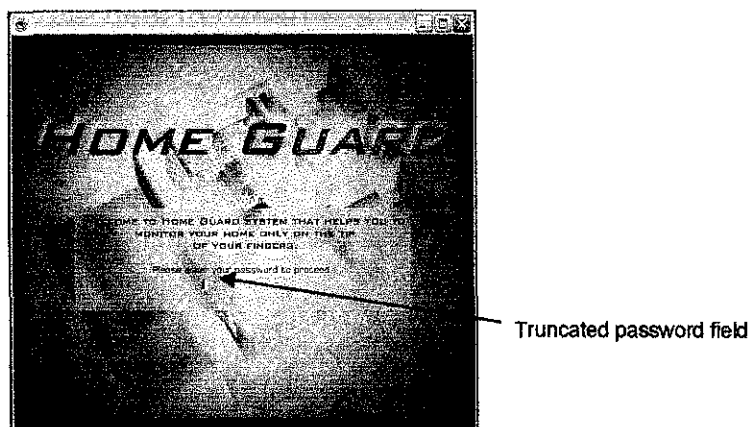


Figure 4.6: Access pad with truncated password field.

Swing programs should override `paintComponent()` instead of overriding `paint()`. Although the API allows it, there is generally no reason to override `paintBorder()` or `paintComponents()`. This factoring makes it easier for programs to override only the portion of the painting which they need to extend. For example, this solves the AWT

problem mentioned previously where a failure to invoke `super.paint()` prevented any lightweight children from appearing. The solution result is shown in **Figure 4.7**.

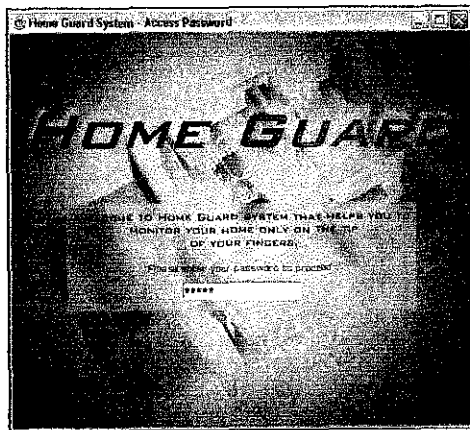


Figure 4.7: Access pad with fully visible password field.

4.1.2 Main Frame

The design concept for this application is to enable users to easily monitor the modules or devices involved in the security system of the house. The draft of the main frame design for the application is displayed in **Figure 4.8**. The main frame consists of two panels, which are the upper panel and the main panel. The upper panel will be consists of logo, icons, buttons, and date and time display. The main panel will be consists of two main panels, which are the tree navigator and the module display.

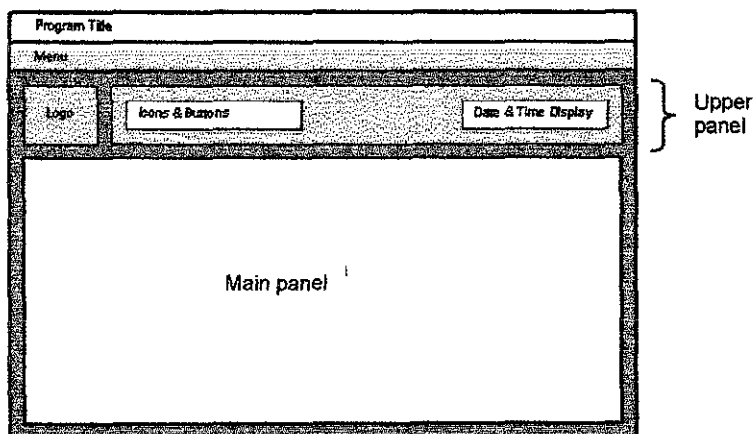


Figure 4.8: Draft of the main frame design of the application.

For the date and time display, *SimpleDateFormat* class is used. *SimpleDateFormat* is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date to text), parsing (text to date), and normalization. *SimpleDateFormat(String pattern, Locale locale)* creates a date formatter using the specified pattern, with the default *DateFormatSymbols* for the given locale. It allows us to start by choosing any user-defined patterns for date-time formatting. Each of these class methods can return a date/time formatter initialized with a default format pattern. The format pattern was then modified using the *applyPattern()* methods as desired.

After the format pattern of the date/time is modified, the date/time must be displayed to the user. The displayed date/time must be updated from time to time so that the user is kept informed of the current time. To update the displayed time, threading is used. The class created for this function is *TimerThread*. It is an extension of a *Thread* class that implements *Runnable* interface. *Thread* and *Runnable* are defined in the *java.lang* package. The *TimerThread* prompt the thread to halt for 1000 milliseconds, before updating the current time displayed. In other words, the current time displayed is updated every 1 second. For a more detail description about threads, you can refer **Appendix E**. The *TimerThread* codes are listed below.

```
class TimerThread extends Thread implements Runnable
{
    UpperPanel myclock;

    public TimerThread(UpperPanel myclock)
    {
        this.myclock = myclock;
    }

    public void run()
    {
        while(true)
        {
            try
            {
                this.sleep(1000);
            }
            catch(InterruptedException e){}
            myclock.setTime();
        }
    }
}
```

```
}  
}  
}
```

The date and time was displayed as text in a *JTextField* object. However, flickering occurs when the program is running. The problem was overcome when the text field is replaced by *JLabel* instead. **Figure 4.9** shows an initial design of the upper panel with only a sample icon and the date and time display. The design was then improved before combining it with the main panel design.

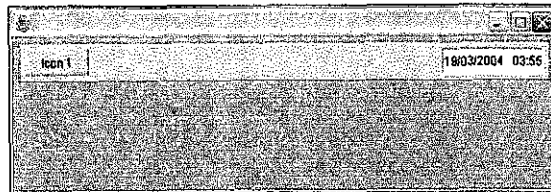


Figure 4.9: An initial design of the upper panel.

The main panel mainly consists of two separate panels:

- a) tree navigator panel,
- b) module display panel.

4.1.2.1 Tree Navigator Panel

Swing trees display hierarchical data by using a well-known paradigm of folders and leaf items. The most widely used tree is indeed Windows Explorer, which contains a tree component for navigating directories. The tree component is represented by the *JTree* class, which resides in the swing package.

Trees are composed of nodes, which can be either folders or leaves. Folders can have child nodes, and all nodes but a tree's root node has a single parent node. Empty folders can be differentiated from leaves by whether they allow children. Folders and leaves are represented by different icons that look-and-feel dependent. Folders can be expanded and collapsed, either by double-clicking on the folder or by clicking on the folder's handle. The visibility of the root node's handle can be set.

In addition to parent and child nodes, tree nodes also have a user object. User objects are of type `Object` and therefore provide a way to associate any object with a node. Trees have a simple model, and each *JTree* instance maintains references to a renderer and an editor that are used for all nodes in the tree.

The *JTree* class also provides constructors for creating trees with `Object` arrays, hash tables, and vectors. Since the order in which objects are added to a hash table has no correlation to the manner in which the objects are stored, trees created with hash tables exhibit an unpredictable node order.

The tree navigator design consists of 5 nodes representing the rooms to be monitored in the house, which are Bedrooms, Kitchen, Living Room, Dining Room, and General. Since the design assumed to have 3 bedrooms, these bedrooms are created as child nodes under Bedrooms node. The leaves under each node represent devices to be monitored in related rooms. The tree navigator constructed is shown in **Figure 4.10**.

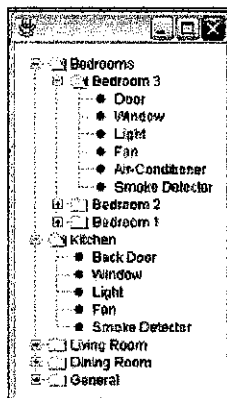


Figure 4.10: The tree navigator panel.

Tree nodes are often quantified by tree paths. When a tree node is selected, the selection is identified by an instance of *TreePath*. The *TreePath* class identified a set of nodes that form a path from one node to another. This class is a simple extension of `Object` that maintains an array of objects representing a path.

The tree navigator is equipped with a selection listener that identifies the path of the recently selected node in order to link it to the status of the devices, according to the right rooms. The selection listener obtains a reference to the path associated with the selection by invocation of `TreeSelectionEvent.getNewLeadSelectionPath()`.

The `swing.tree` package provides a default renderer in the form of the `DefaultTreeCellRenderer` class. This class maintains three `Icon` references for leaf nodes and open/closed folder nodes. Colours for text, background, and the renderer's border are also maintained. `DefaultTreeCellRenderer` can also be used to customize the colours, icons, and font associated with tree nodes. An instance of this class is used to modify leaf, open and closed icons, and font of the tree navigator panel. The modified tree is shown in **Figure 4.11**.

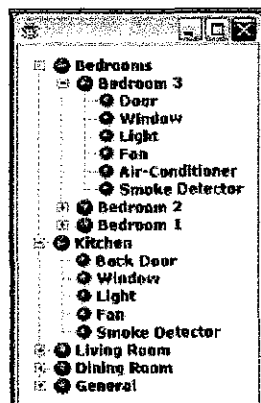


Figure 4.11: The modified tree design.

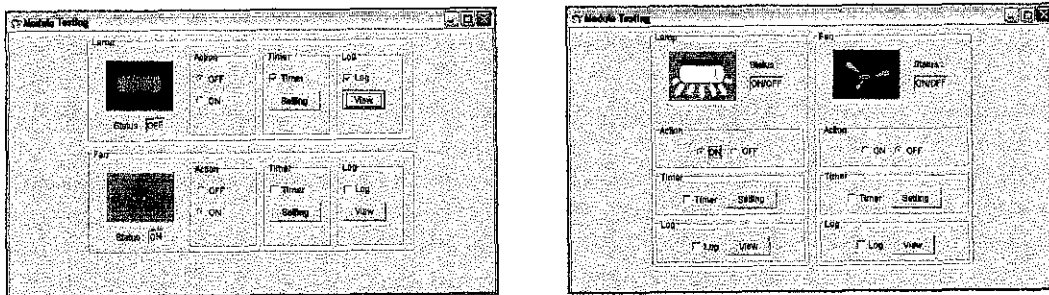
4.1.2.2 Module Display Panel

Tabbed panes are a common user interface component that provides convenient access to more than one panel. Swing's tabbed pane is implemented by `JTabbedPane`. The tabs contained in an instance of `JTabbedPane` have a single component associated with them that is displayed below the tab. Tabs can display both icon and text and can have their background colour set. It allows its tabs to be placed along the right, left, bottom, and top

of the tabbed pane. Tab placement can be specified when instances of *JTabbedPane* are constructed or after construction with the *JTabbedPane.setTabPlacement()* method.

The module display panel is design to support two choices of view, which are module view and list view. The module view will display control panel of individual devices which are grouped according to the rooms.

Each module generally contains 4 fields, which are status, action, timer and log. The status field consists of an image that change according to the status of the devices, and a label that states out the device status. The action field is made up of 2 choices of radio buttons. The two radio buttons are mutually exclusive, that is when one of the radio buttons is selected, the other will be deselected. The timer field and log field are additional features to set the time when a device can be switch on or off, and to log any activities took place by the device as a report. The initial design of the module view is as shown in **Figure 4.12**.



(a) Horizontal arrangement of the fields. (b) Vertical arrangement of the fields.

Figure 4.12: Initial design of the module view.

To simplify the module view, the timer field and log field are hidden by replacing them with a single button, "Option". The images designed are also enhanced and the vertical arrangement is chosen for the final module view design, as shown in **Figure 4.13**.

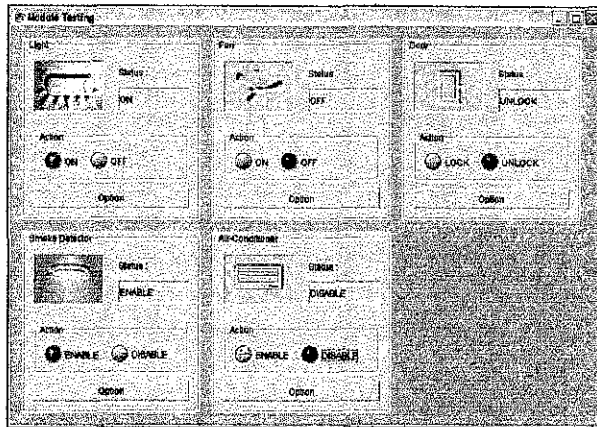


Figure 4.13: Final design of the module view.

The results of individual design panels are merged and shown in **Figure 4.14**. More features should be revised, either added or removed from the design, depending on the necessity, time constraint and hardware constraint. Some of the components only existed on the base only without any control functions, and some of the controls are still not functioning as required.

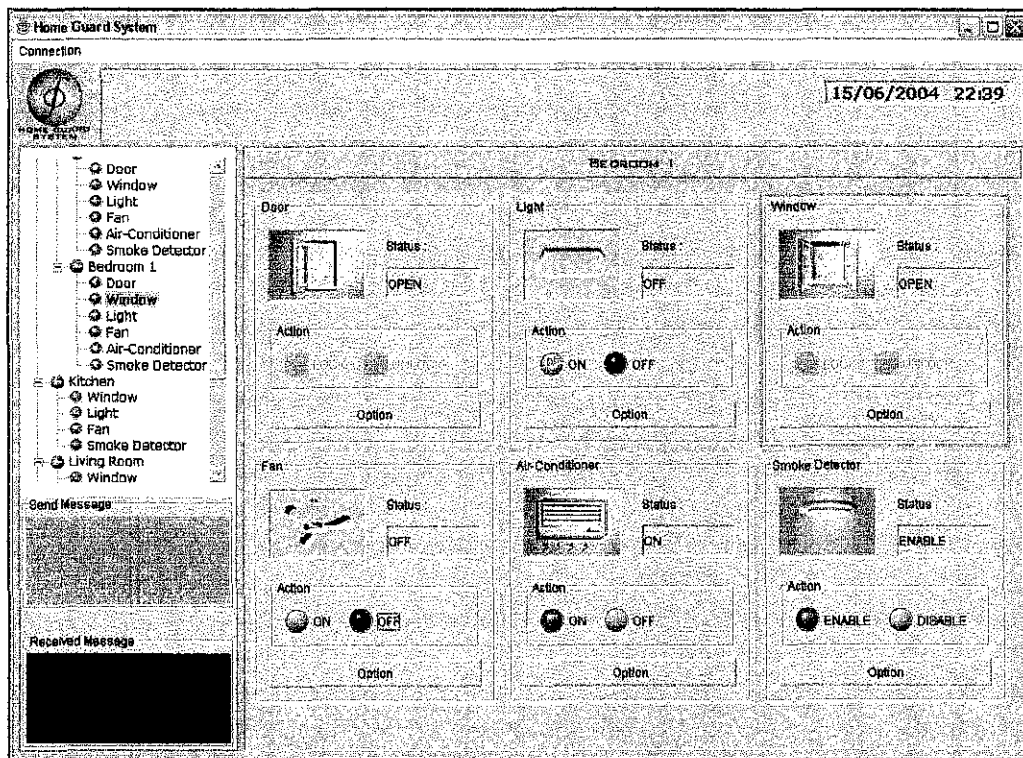


Figure 4.14: The main frame design.

4.2 Serial Port Programming

After downloading and installing the Java Communication API in C: partition, a basic sample program codes, *NamedPortLister*, is compiled and run to test for the functionality of the installed extension package. This program basically looks for serial and parallel ports and lists them down. It is successfully compiled and run, but the program did not find any port on the designated computer. Troubleshooting is carried out, by altering a few lines of coding. At last, it is detected that the compiled was saved in a different partition, which is in D: partition. When this program is compiled in the same partition as the installed Java Communication API and JDK, the program is successfully run and lists down several serial and parallel ports in that computer. Therefore, it is concluded that all programs which applied the Java Communication API must be saved, compiled, and run in C: partition onwards. After that, more basic sample programs are compiled and run to test the methods available in the extension package.

A sample program from Java Communication API, *SerialDemo*, was compiled, run, and studied. This program basically provides functions to set parameters of an available serial port on the computer, and open the port with the selected parameters value. The classes involved in this program was studied, changed, and adapted in order to gain understanding on how to construct coding that is necessary to establish serial communication. Classes that are essential for setting and opening the serial port are *SerialParamateres* and *SerialConnection*.

MySerial1 was constructed with a few functions available in the sample program, with several additional functions such as registering listeners to notify events such as *setRTS()* and *isCTS()*. Figure 4.15 shows “My Serial 1” application. This program is tested by connecting TD pin to RD pin, and RTS pin to CTS pin, of COM1. Hence, data that are transmitted will be received back by the same program, and whenever the value of RTS is checked or unchecked, the program will state it by displaying a line of messages such as, “RTS is checked” or “RTS is Unchecked”. When RTS is set to

high, the CTS pin will detect it and state it by displaying a line of messages such as, “CTS is cut OFF” or “CTS is received” inside the received data text field. These functions basically constructed to test the codes manipulation on the six pins of DTR, RTS, CTS, DSR, RI, and CD.

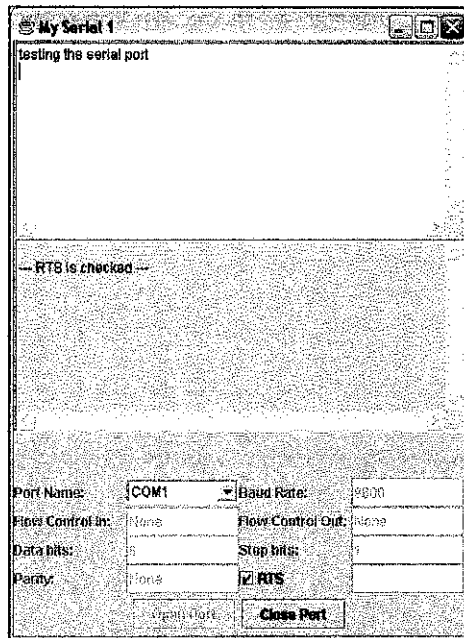


Figure 4.15: A test module for serial port communication programming.

4.3 PIC16F84 Programming

The prototype circuit discussed in previous chapter was built and tested. When the circuit is tested and proven to be working perfectly, the next step is to program the PIC. Several examples available on the internet were studied to have a better understanding on how to program the PIC16F84, using suitable function for the demonstration. Some of the examples do not work when it is compiled and burn, while the other working examples are designated to different types of PIC. Hence, adjustment and manipulation of these examples is used to come up with a program that can be used on PIC16F84, and can establish serial communication with the computer’s serial port.

The PIC must be set to the required speed of the processor. The speed used must be the same with the external crystal clock oscillator's frequency, which in this case is 4MHz.

The command to invoke this function is:

```
#use delay(clock=4000000) //4MHz OSC
```

It tells the compiler the speed of the processor and enables the use of the built-in functions: **delay_ms()** and **delay_us()**.

The PIC must also be set to use the RS232 built-in function. The command to invoke this function is:

```
#use rs232(baud=9600, xmit=PIN_A1, rcv=PIN_A0)
```

This directive tells the compiler the baud rate and pins used for serial I/O. This directive takes effect until another RS232 directive is encountered. The **#use delay** directive must appear before this directive can be used. This directive enables use of built-in functions such as **getc()**, **putc()**, and **printf()**. The baud rate set must be the same with the received data baud rate, in this case is the computer's serial port baud rate. If not, there will be error in the received data

To transmit serial data from the PIC, the built-in function of **printf()** can be used directly, in example:

```
printf("F0");
```

But to handle received serial data from the computer, a more complex function must be used, especially when the received serial data is in terms of a stream. This is because **getc()** can only read one character per time. It waits for a character to come in over the RS232 RCV pin and returns the character.

Since we do not want to hang forever waiting for an incoming character, **kbhit()** is used to test for a character available. If the RS232 is under software control, this function returns "true" if the start bit of a character is being sent on the RS232 RCV pin. If the RS232 is hardware, this function returns "true" if a character has been received and is

waiting in the hardware buffer for `getc()` to read. This function may be used to poll for data without stopping and waiting for the data to appear.

An example of codes to handle this problem was found, named as *mikeslib.c*, and is used for handling the received serial data. It monitors the RS232 RCV pin for any incoming data, and then catches this data byte-by-byte and stores it temporarily in a buffer. If the first data is not handled immediately, the data will be lost and the buffer will be replaced by the second received byte, and the same goes with each incoming data.

Therefore, another function is created to handle streams of received data. Looping is used to read streams of received data, so that the data is not lost. The streams is read byte-by-byte and stored in a buffer. After that, the stored data is then called and interpreted to trigger designated pins. In example, if the received data buffer is "L1", pin A2 is set to high, and if it is "L0", pin A2 is set to low. In this case, the computer can send streams of meaningful data to the PIC, and the PIC will process the data to trigger any designated pins that represents the home appliances. The process flow for serial data processing on the PIC is summarized in **Figure 4.16**.

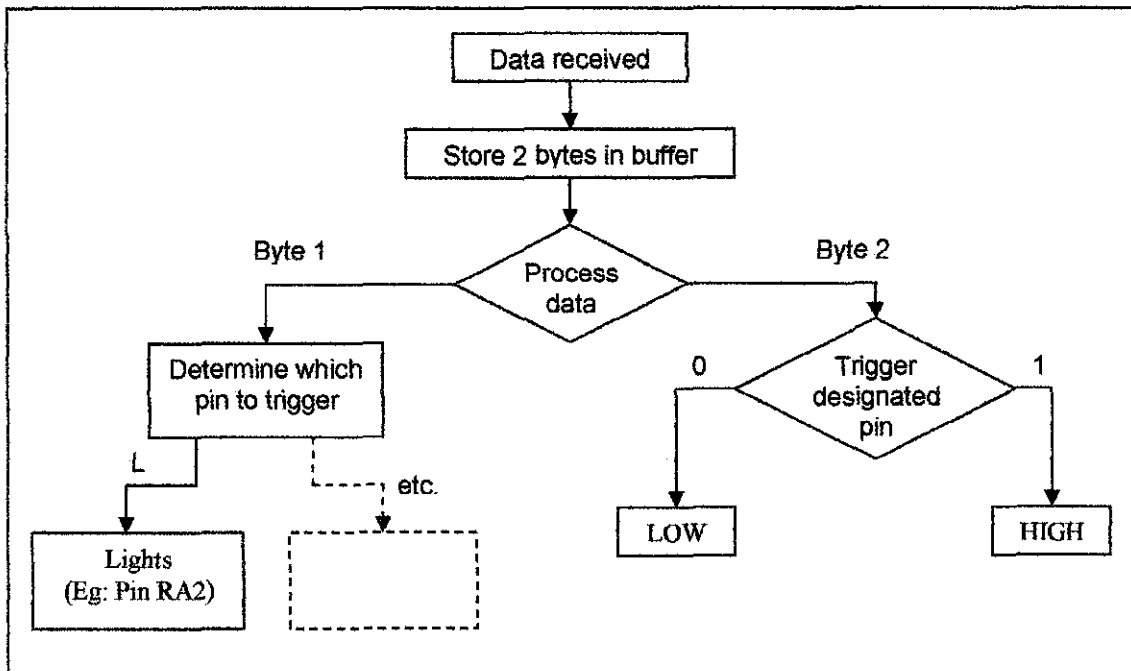


Figure 4.16: PIC serial data processing.

However, the program still cannot detect any other input besides the serial data from the computer. In real situation, manual switch is also used to turn the appliances on and off. Therefore, the program should be able detect switches or button press. Interrupts is used to handle this situation.

Built-in function to perform this task is **enable_interrupts()**. It enables the interrupt at a given level. An interrupt procedure should be defined for the indicated interrupt. The GLOBAL level will not enable any of the specific interrupts but will allow any of the specific interrupts previously enabled to become active. To detect a button press, RB interrupt is applied. The RB interrupt will happen when there is any change (input or output) on pins B4-B7. There is only one interrupt and the PIC does not tell which pin changed. Which pin changes must be determined based on the previously known value of the port. Furthermore, a single button press may cause several interrupts due to bounce in the switch. A debounce algorithm is needed to be used. A simplest way is to set a delay after the first interrupt, to eliminate possible debounce, before executing any function and waiting for the second interrupt. Example of interrupt usage is, when a button press triggers pin B4, it will toggles pin A2, and transmit streams to inform the changes to the computer. If pin A2 is high, it will be set to low, and vice versa. In this case, the user can control home appliances using manual switches in parallel with the software, and the software is aware of the changes.

After further alteration and enhancement, the program should be able to perform these tasks:

- a) Triggers designated pins using the serial received data.
- b) Triggers designated pins using button press.
- c) Inform the computer software about the changes made using button press.

The codes involves in the PIC programming are listed in **Appendix F**.

4.4 Module Integration

Individual working modules are integrated to come up with a single working system. The window module, hardware module and prototype module is integrated to come up with the Home Guard System, with a prototype for demonstration. A lot of problems are encountered when integrating these modules, since changes must be made to enable the modules to work with each other.

Most problems arise when integrating window module with the hardware module. The GUI developed was not working properly with the serial communication classes constructed. To overcome this problem, some of the serial communication classes constructed are combined with the GUI classes. This will eliminate problems arises from multi-level properties modifications, and accessor methods. The problems are successfully overcome and the integration results in a single working software system. All the classes involve in this integrated module are listed under **Appendix G**.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

A software system for the PC-based surveillance system is successfully designed and built. The Swing components are used as the basic components for the development of this application. However, more features needed to be revised, and some of them are either added or removed from the design due to the necessity, time constraint and hardware constraint.

The Java Communications API 2.0 is successfully implemented. The software can successfully communicate with the serial port on the computer. The software can then sends and receives serial data through the serial port. These serial data is processed to enable the monitoring and controlling of home appliances.

A prototype, basically built up of a simple circuit board is successfully built. It demonstrates the workability of the software to communicate with the serial port. A microcontroller, specifically PIC16F84 is successfully used to translate and manipulate the data received from the serial port into meaningful functions for prototype demonstration.

Apart from the successfully working program, this project is still weak in terms of producing a complete and safe surveillance system. It emphasises more on enhancing the GUI design and looks. It can only monitor and control pins available on the PIC16F84. Further enhancement on this project can be made to improve the system, and come up with a better presentable, secure and reliable system with additional functions. Improvement on the Home Guard System can be recommended in these two modules, the window module and prototype module.

The window module basically consists of the GUI development and the management of the control functions. The GUI can be improved by designing better images, create better arrangement of components and using more sophisticated Java Look-And-Feel utilities. Apart from viewing the home appliances in the available module display, list or table display can be added. This display can list down all the appliances' states and control components in one table to make it easy for the user to monitor and control all the appliances only in one frame. Multimedia utilities can also be added to the system, such as video monitoring and audio assignment in case of emergency. A more user friendly software can also be created by adding a wizard that helps user on how to use and configure this software.

As for the management of the control functions, themes can be used to set several appliances to behave in certain ways. Besides monitoring and controlling the home appliances individually, a theme can be design to set desired behaviour of desired appliances at a desired time, and these properties can be saved and uploaded when need by the user. The software system can also be extended to function over the internet services, since Java programming language already provides an easy path to program a server/client application. A distance home surveillance monitoring and controlling is possible via the internet network. The user can then install the software at the office, and still can monitor his home via the internet network.

For the prototype module, several enhancements can be carried out. Besides triggering LEDs, the PIC pins can be connected to trigger real home appliances. In this case, power relays is needed to be considered and applied. The PIC can also be used to trigger external alarm system.

Since the PIC pins are limited, the appliances to be monitored and controlled are also limited. Therefore, the number of inputs and outputs to be monitored and controlled can be increased by increasing the amount of PICs used. An example design circuit for using 2 PICs are shown in **Figure 5.1**. This will enables the system to control more home appliances.

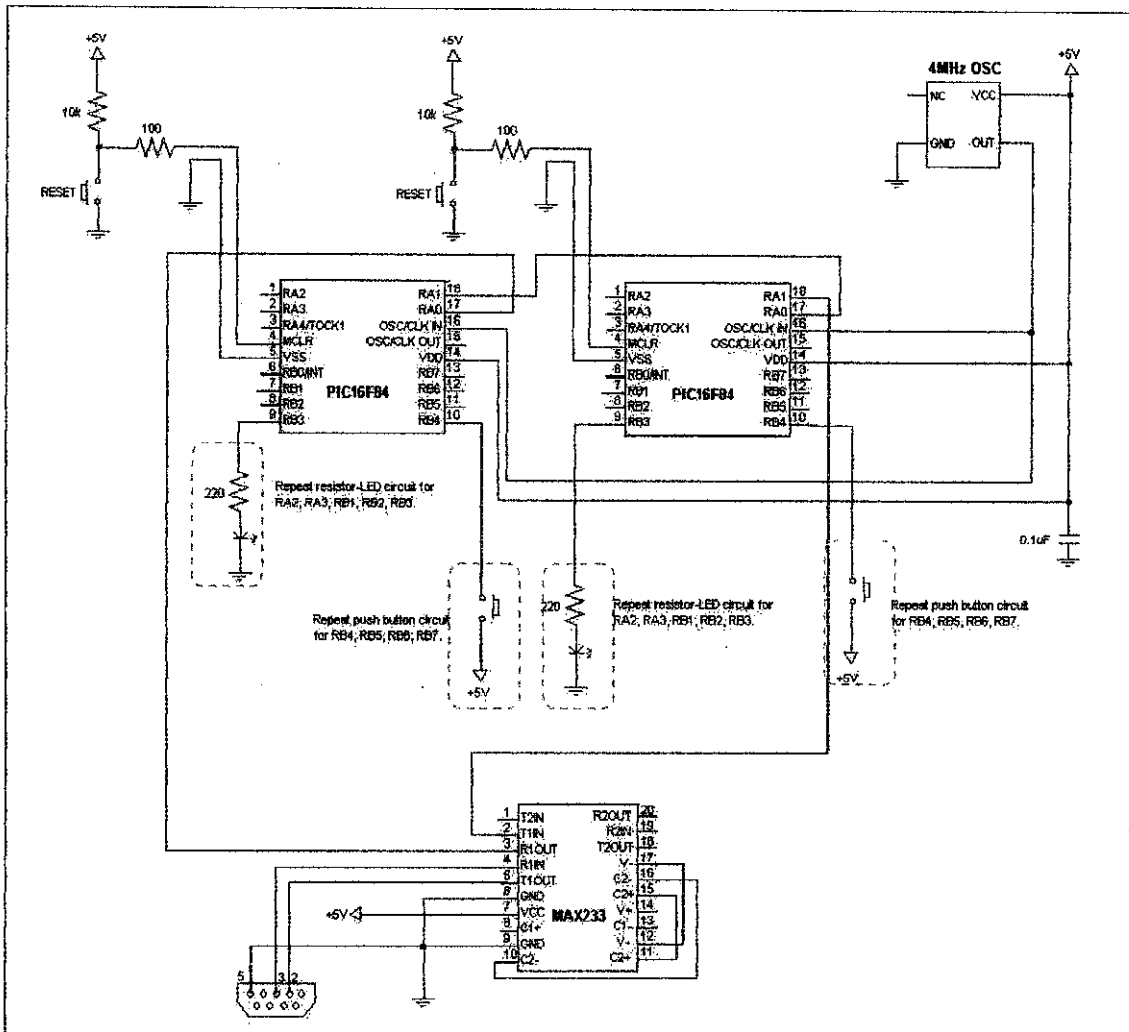


Figure 5.1: Recommended schematic for prototype module using 2 PICs.

An extended protocol on the serial data can also be developed to enable the usage of more than one PIC. For example, instead of storing 2 bytes of data to be processed, the buffer can be set to store 3 bytes of received serial data. The first byte will determine whether the command is intended for the first PIC or the second PIC. The following 2 bytes will function the same as discussed in the previous chapter, which is to determine which pin to trigger and how to trigger it. An extended protocol for the recommended schematic in Figure 5.1 can be summarized as shown in Figure 5.2.

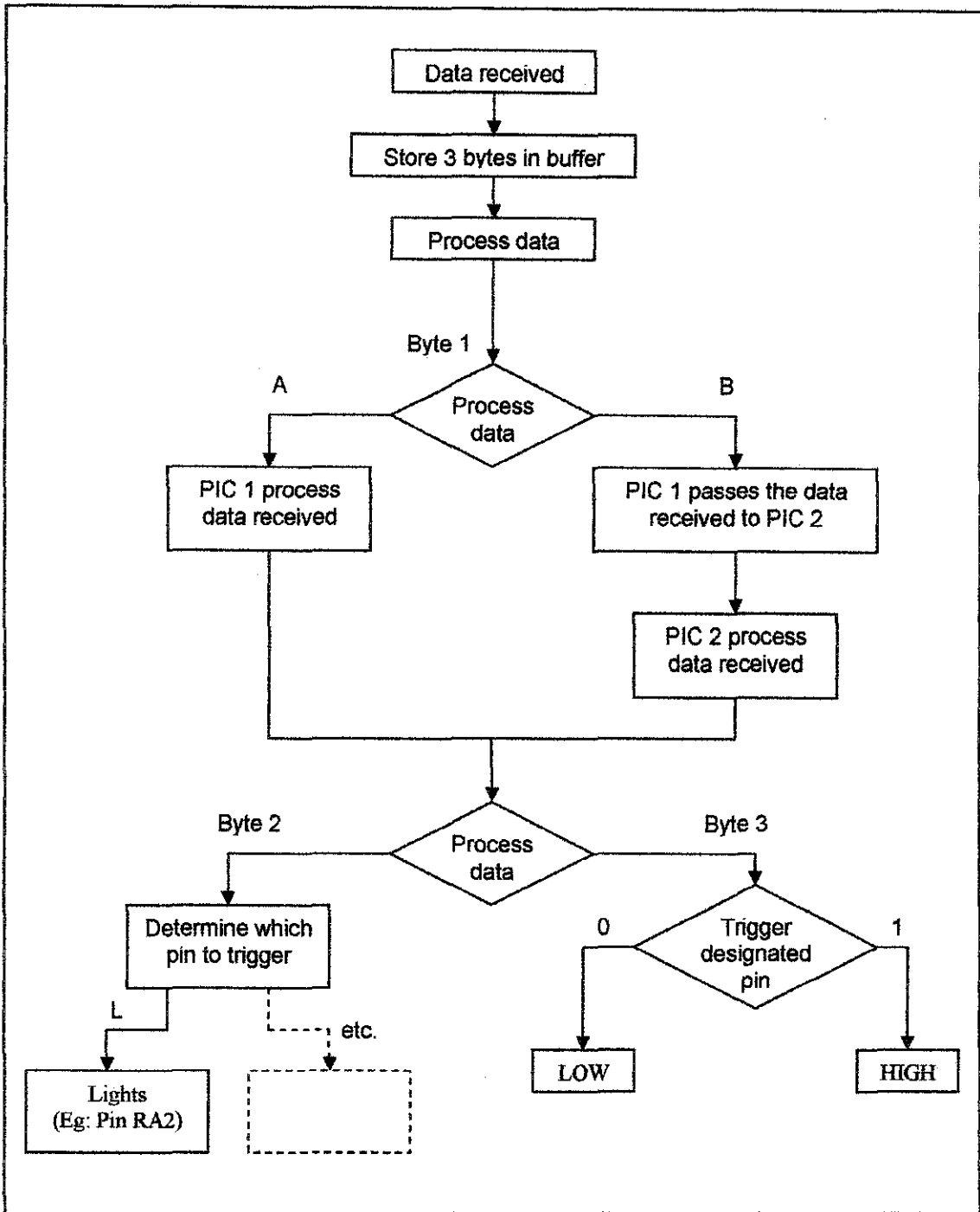


Figure 5.2: Serial data processing for 2 PICs.

REFERENCES

1. H. M. Deitel and P. J. Deitel; *Java: How To Program*; Prentice Hall; 4th Edition; USA; 2002.
2. Russel Winder and Graham Roberts; *Developing Java Software*; John Wiley & Sons; 1st Edition; UK; 1998.
3. Ian Sommerville; *Software Engineering*; Addison-Wesley; 6th Edition; USA; 2001.
4. Joe Wigglesworth and Paula Lumby; *Java Programming: Making the Move from C++*; Course Technology; 1st Edition; USA; 1999.
5. Peter Van Der Linden; *Just Java2*; Sun Microsystems; 4th Edition; USA; 1999.
6. David M. Geary; *Graphic Java 2: Mastering the JFC, Volume II: Swing*; Sun Microsystems; 3rd Edition; USA; 1999.
7. Elliotte Rusty Harold; *Java I/O*; O'Reilly; 1st Edition; Beijing; 1999.
8. Y. Daniel Liang; *Introduction to Java Programming with JBuilder 4/5/6*; Prentice Hall; 2th Edition; USA; 2002.
9. www.sun.com
10. www.softwaredev.earthweb.com
11. www.developer.java.sun.com
12. www.javaworld.com/index.html
13. www.jguru.com
14. www.bakson.com

15. www.smarthome.com
16. <http://java.sun.com/getjava/help.html>
17. <http://java.sun.com/j2se/1.3/docs/api/index.html>
18. <http://www.arcelect.com/rs232.htm>
19. <http://www.boondog.com//tutorials/pic16F84/pic16f84.html>
20. <http://www.mikroelektronika.co.yu/english/product/books/picbasicbook/06.htm>
21. <http://www.picant.com/c2c/examples.html>

NAME : Mumi Binti Masri
 ID NUMBER : 616
 FYP TITLE : Software Implementation of a PC-Based Home Surveillance System

49

APPENDIX A

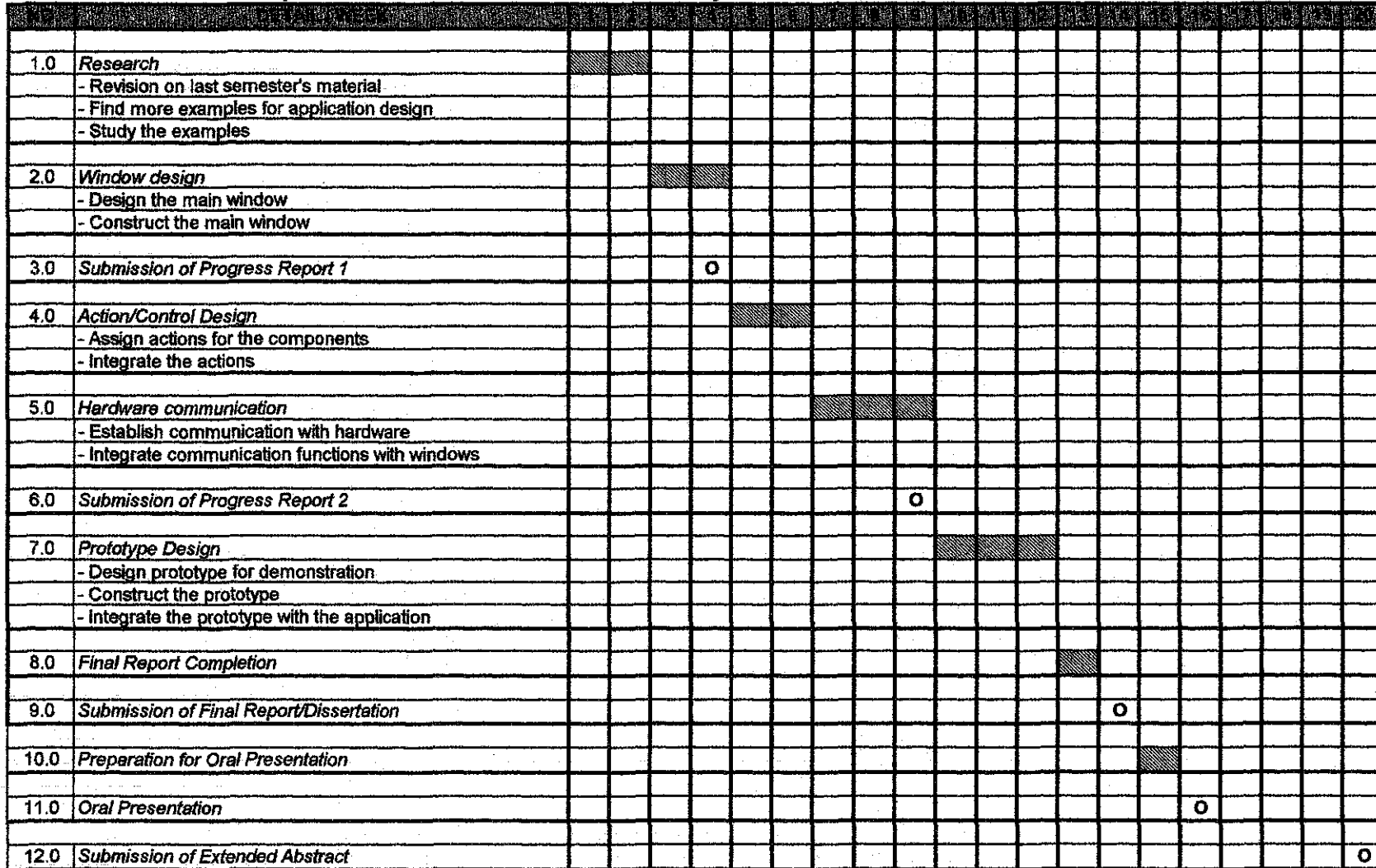
		1	2	3	4	5	6	7	8	9	10	11	12
1.0	<i>Selection of Project Topic</i>												
	- FYP Briefing, approval on Project Title and Synopsis	■											
	- Selection and Prioritisation of Project Titles		■										
	- Allocation of Approved Project Titles			■									
2.0	<i>Preliminary Research work</i>												
	- Introduction												
	- Objective												
	- List of References / Literature												
	- Project Planning												
3.0	<i>Submission of Preliminary Report</i>												
4.0	<i>Project Work</i>												
	- Reference / Literature												
	- Research / Programming												
5.0	<i>Submission of Progress Report</i>												
6.0	<i>Project Work Continue</i>												
	- Research / Programming												
7.0	<i>Submission of Interim report Final Draft</i>												
8.0	<i>Oral Presentation</i>												
9.0	<i>Submission of Interim Report</i>												

NOTE : ○ Milestone ■ Process

APPENDIX A : GANTT CHART FOR THE FIRST SEMESTER OF 2 SEMESTER OF FINAL YEAR PROJECT

NAME : Murni Binti Masri
ID NUMBER : 616
FYP TITLE : Software Implementation of a PC-Based Home Surveillance System

50



APPENDIX B

NOTE : ○ Milestone ■ Process

APPENDIX B : GANTT CHART FOR THE SECOND SEMESTER OF 2 SEMESTER FINAL YEAR PROJECT

APPENDIX C

Pin Name	DIP No.	SOIC No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	I	ST/CMOS ⁽³⁾	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	O	---	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR	4	4	I/P	ST	Master clear (reset) input/programming voltage input. This pin is an active low reset to the device.
RA0	17	17	I/O	TTL	<p>PORTA is a bi-directional I/O port.</p> <p>Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.</p>
RA1	18	18	I/O	TTL	
RA2	1	1	I/O	TTL	
RA3	2	2	I/O	TTL	
RA4/T0CKI	3	3	I/O	ST	
RB0/INT	6	6	I/O	TTL/ST ⁽¹⁾	<p>PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.</p> <p>RB0/INT can also be selected as an external interrupt pin.</p> <p>Interrupt on change pin.</p> <p>Interrupt on change pin.</p> <p>Interrupt on change pin. Serial programming clock.</p> <p>Interrupt on change pin. Serial programming data.</p>
RB1	7	7	I/O	TTL	
RB2	8	8	I/O	TTL	
RB3	9	9	I/O	TTL	
RB4	10	10	I/O	TTL	
RB5	11	11	I/O	TTL	
RB6	12	12	I/O	TTL/ST ⁽²⁾	
RB7	13	13	I/O	TTL/ST ⁽²⁾	
Vss	5	5	P	---	Ground reference for logic and I/O pins.
VDD	14	14	P	---	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = Input/Output P = power
 --- = Not used TTL = TTL input ST = Schmitt Trigger input

- Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
 Note 2: This buffer is a Schmitt Trigger input when used in serial programming mode.
 Note 3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

Appendix C: PIC16F8X Pinout Description.

APPENDIX D

EVENT-DRIVEN PROGRAMMING^[8]

a) Event and Event Source

When Java graphics programs are run, the program interacts with the user and the events drive its execution. An *event* can be defined as a signal to the program that something has happened. The event is generated either by external user actions, such as mouse movements, mouse button clicks, and keystrokes, or by the operating system, such as a timer. The program can choose to respond to or ignore the event.

The GUI component on which an event is generated is called the *source object*. For example, a button is the source object for a clicking-button action event. An event is an instance of an event class. The root class of the event classes is *java.util.EventObject*. The hierarchical relationship of the event classes are shown in **Figure D.1**.

An event object contains whatever properties are pertinent to the event. The source object of the event can be identified by using the `getSource()` instance method in the *EventObject* class. The subclasses of *EventObject* deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes. **Table D.1** lists external user actions, source objects, and event types generated.

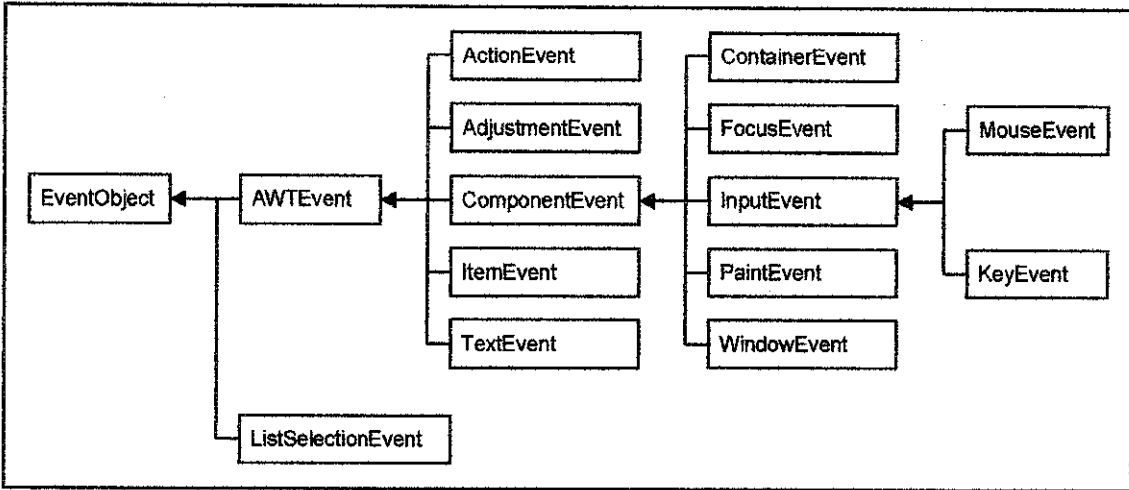


Figure D.1: An event is an object of the *EventObject* class.

Table D.1: User Action, Source Object, and Event Type

User Action	Source Object	Event Type Generated
Click a button	JButton	ActionEvent
Change text	JTextComponent	TextEvent
Press return on a text field	JTextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Select item(s)	JList	ListSelectionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Select a menu item	JMenuItem	ActionEvent
Move the scroll bar	JScrollBar	AdjustmentEvent
Window opened, closed, iconified, deiconified, or closing	Window	WindowEvent
Component added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden, or shown	Component	ComponentEvent
Component gained or lost focus	Component	ComponentEvent
Key released or pressed	Component	KeyEvent
Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent
Mouse moved or dragged	Component	MouseEvent

b) Event Registration, Listening, and Handling

Java uses a delegation-based model for event handling. An external user action on a source object triggers an event. An object interested in the event receives the event. Such an object is called a *listener*. Not all objects can receive events. To become a listener, an object must be registered as a listener by the source object. The source object maintains a list of listeners and notifies all the registered listeners by invoking the event-handling method, known as the *handler*, on the listeners object to respond to the event, as shown in Figure D.2.

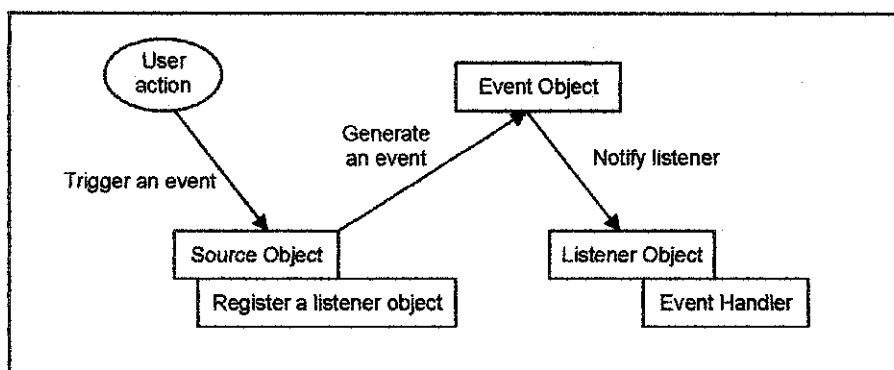


Figure D.2: Event-handling.

For example, if a *JFrame* object is interested in the external events on a *JButton* source object, it must register with the *JButton* object. The registration is done by invoking a method from the *JButton* object to declare that the *JFrame* object is a listener for the *JButton* object. When the button is clicked, the *JButton* object generates an *ActionEvent* and notifies the listener by invoking a method defined in the listener to handle the event.

Registration methods are dependent on event type. For *ActionEvent*, the method is `addActionListener`. In general, the method is named `addXListener` for *XEvent*.

To become a listener, the listener must implement the standard handler. The handler is defined in the corresponding event-listener interface. Java provides a listener interface for every type of graphics event. For example, the corresponding listener interface for

ActionEvent is *ActionListener*, each listener for *ActionEvent* should implement the *ActionListener* interface.

Table D.2 lists event types, the corresponding listener interfaces, and the methods defined in the listener interfaces.

Table D.2: Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed (ActionEvent e)
ItemEvent	ItemListener	itemStateChanged (ItemEvent e)
WindowEvent	WindowListener	windowClosing (WindowEvent e)
		windowOpened (WindowEvent e)
		windowIconified (WindowEvent e)
		windowDeiconified (WindowEvent e)
		windowClosed (WindowEvent e)
		windowActivated (WindowEvent e)
		windowDeactivated (WindowEvent e)
ContainerEvent	ContainerListener	componentAdded (ContainerEvent e)
		componentRemoved (ContainerEvent e)
ComponentEvent	ComponentListener	componentMoved (ComponentEvent e)
		componentHidden (ComponentEvent e)
		componentResized (ComponentEvent e)
		componentShown (ComponentEvent e)
FocusEvent	FocusListener	focusGained (FocusEvent e)
		focusLost (FocusEvent e)
TextEvent	TextListener	textValueChanged (TextEvent e)
KeyEvent	KeyListener	keyPressed (KeyEvent e)
		keyReleased (KeyEvent e)
		keyTyped (KeyEvent e)
MouseEvent	MouseListener	mousePressed (MouseEvent e)
		mouseReleased (MouseEvent e)
		mouseEntered (MouseEvent e)
		mouseExited (MouseEvent e)
		mouseClicked (MouseEvent e)

	MouseMotionListener	mouseDragged (MouseEvent e)
		mouseMoved (MouseEvent e)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent e)

c) Handling Events

A listener object must implement the corresponding listener interface. For example, a listener for a *JButton* source object must implement the *ActionListener* interface. The *ActionListener* interface contains the `actionPerformed(ActionEvent e)` method. This method must be implemented in the listener class. Upon receiving notification, it is executed to handle the event.

An event is passed to the handling method. The event object contains information pertinent to the event type. Useful data values can be obtain from the event object for processing the event. For example, `e.getSource()` can be used to obtain the source object in order to determine whether it is a button, a check box, a radio button, or a menu item.

APPENDIX E

THREADS^[81]

A *thread* is a flow of execution, with a beginning and an end, of a task in a program. When program executes as an application, the Java interpreter starts a thread for the **main()** method. Additional threads can be created to run concurrent tasks in the program. Each new thread is an object of a class that implements the *Runnable* interface or extends a class that implements the *Runnable* interface. This new object is referred to as a *runnable* object. Threads can be created either by extending the *Thread* class or implementing the *Runnable* interface. Both *Thread* and *Runnable* are defined in the **java.lang** package. **Thread** actually implements *Runnable*.

The *Runnable* interface is rather simple. It contains just the **run()** method. However, this approach works well if the user thread class inherits only from the *Thread* class, but not if it inherits multiple classes, as in the case of an applet. We need to implement this method to tell the system how our thread is going to run. **Figure E.1** illustrates the key elements of a thread class that implements the *Runnable* interface, and how to use it to create a thread in a class.

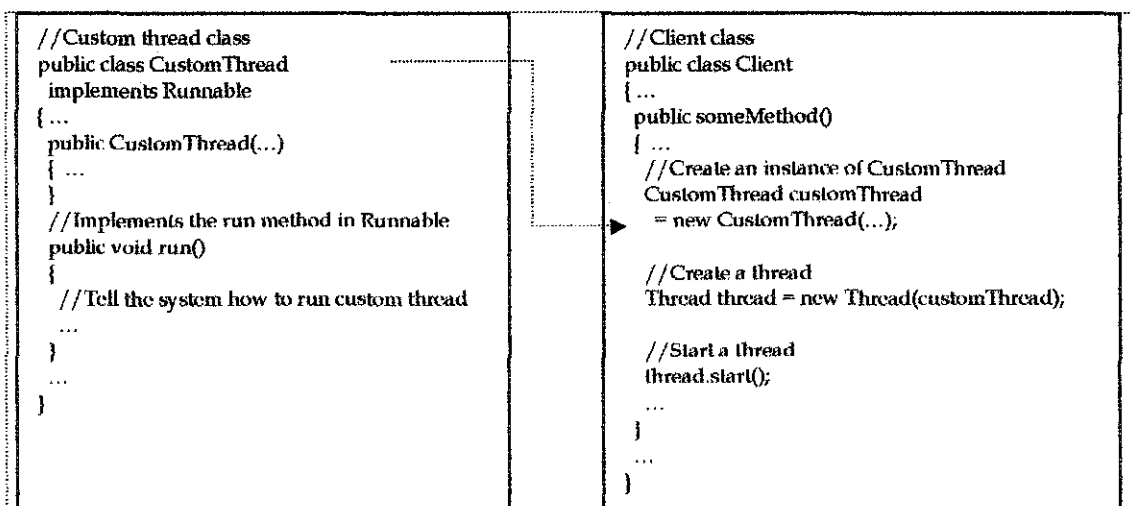


Figure E.1: Define a thread class by implementing the *Runnable* interface.

Threads can be in one of five states: new, ready, running, blocked, or finished, as described in Figure E.2. When a thread is newly created, it enters the *new state*. After a thread is started by calling its **start()** method, it enters the *ready state*. A ready thread is runnable but may not be running yet. The operating system has to allocate CPU time to it.

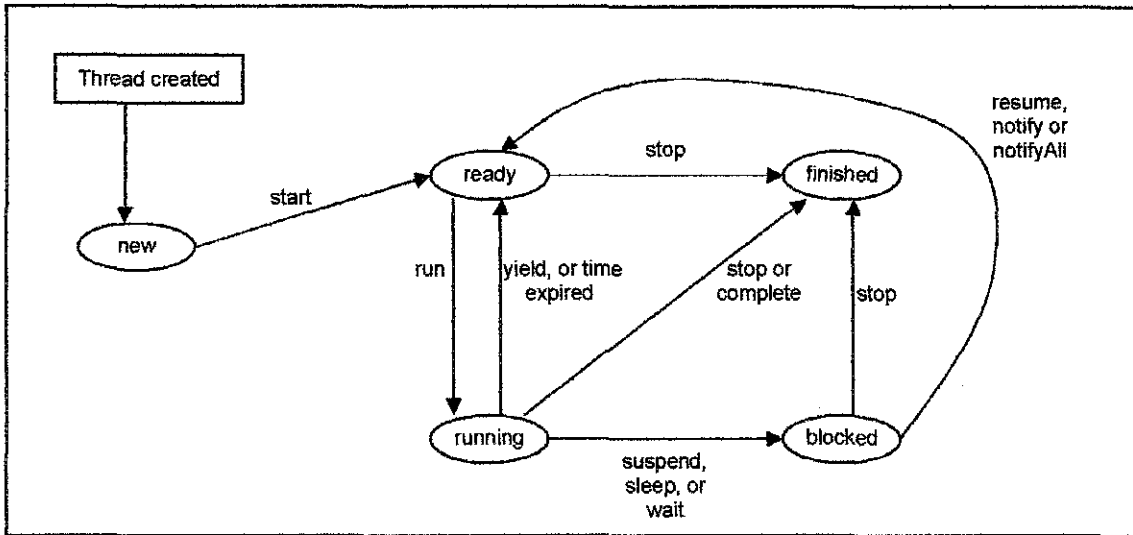


Figure E.2: Thread states.

When a ready thread begins executing, it enters the *running state*. A running thread may enter the ready state if its given CPU time expires or its **yield()** method is called. A thread may enter the *blocked state* (i.e., become inactive) for several reasons. It may have invoked the **sleep()**, **wait()**, or **suspend()** method, or some other thread may have invoked its **sleep()** or **suspend()** method. It may be waiting for an I/O operation to finish. A blocked thread may be reactivated when the action inactivating it is reversed. For example, if a thread has been put to sleep and the sleep time has expired, the thread is reactivated enters the ready state. Finally, a thread is finished if it completes the execution of its **run()** method or if its **stop()** is invoked.

The **isAlive()** method is used to find out the state of a thread. It returns *true* if a thread is in the ready, inactive, or running state; it returns *false* if a thread is new and has not started or if it is finished.

Java assigns every thread a priority. By default, a thread inherits the priority of the thread that spawned it. You can increase or decrease the priority of any thread by using the **setPriority** method, and you can get the thread's priority by using the **getPriority** method. Priorities are numbers ranging from 1 to 10. The Thread class has int constants **MIN_PRIORITY**, **NORM_PRIORITY**, and **MAX_PRIORITY**, representing 1, 5, and 10, respectively. The priority of the main thread is **Thread.NORM_PRIORITY**.

The Java runtime system always picks the currently runnable with the highest priority. If several runnable threads have equally high priorities, the CPU is allocated to all of them in round-robin fashion. A lower-priority thread runs only when no higher-priority threads are running.

APPENDIX F

C MAIN PROGRAM CODE FOR PIC16F84

```
//=====
//Author      : Murni Masri
//File Name   : pushbutton1.c
//Description : This program uses PIC16F84 to handle serial data input. The data is
//             translated and used to trigger designated pins. It monitors changes on
//             pins to trigger other designated pins, and transmit serial data.
//=====

// mikeslib.c

#ifndef MikesLibrary
#define MikesLibrary

// Following is the Software Driven Serial driver written by HI-TECH

#ifdef SERIALSOFTWARE
#ifdef TxPort
#define TxTris TRISA
#define TxPort PORTA
#define TxBit 1
#endif

#ifdef RxPort
#define RxTris TRISA
#define RxPort PORTA
#define RxBit 0
#endif

#ifdef XTAL
#define XTAL 4000000
#endif

#ifdef BRATE
#define BRATE 9600
#endif

#define DLY      3
#define TX_OHEAD 13
#define RX_OHEAD 12
#define RSDelay(oh) ((XTAL/4/BRATE) - (oh))/DLY

// Serial Initialisation Routine

static bit      TxData @ PORTBIT(TxPort, TxBit); //TXD Pin
static bit      RxData @ PORTBIT(RxPort, RxBit); //RXD Pin
static bank1 bit TxTRIS @ PORTBIT(TxTris, TxBit);
static bank1 bit RxTRIS @ PORTBIT(RxTris, RxBit);
bit TxRxInit = 0;

void InitSerial (void)
{
    TxData = 1;      // set pin high to start with

    TxTRIS = 0;
    RxTRIS = 1;

    TxRxInit = 1;
}

void putch (char c)
{
    unsigned char dly, bitno;
```



```

    CLRWDT();

    bitno = 11;

    if (TxRxInit == 0)
    {
        InitSerial();
    }

    TxData = 0;    // start bit
    bitno = 12;
    do
    {
        dly = RSDELAY(TX_OHEAD);    // wait one time
        do
        { //nix
            }while(--dly);
            if(c & 1) TxData = 1;
            if(!(c & 1)) TxData = 0;
            c = (c >> 1) | 0x80;
        } while (--bitno);
    }

    // Software Getch Routine

char getch(void)
{
    unsigned char c, bitno, dly;
    if (TxRxInit == 0)
    {
        InitSerial();
    }
    for(;;)
    {
        while(RxData) { CLRWDT(); continue; }    // wait for start bit
        dly = RSDELAY(3) / 2;
        do ; /*nix*/
        while(--dly);
        if(RxData) continue;    // was just noise
        bitno = 8;
        c = 0;
        do
        {
            dly = RSDELAY(RX_OHEAD);
            do ; //nix
            while(--dly);
            c = (c >> 1) | (RxData << 7);
        } while (--bitno);
        return c;
    }
}
#endif    // End of Serial Software routines
#endif

// main.c -----
#include <16F84.H>
#define RS232_XMIT PIN_A1
#define RS232_RCV PIN_A0
#define delay(clock=4000000) //4MHz OSC
#define rs232(baud=9600, xmit=PIN_A1, rcv=PIN_A0)

#include <ctype.h>

// Serial Port Settings - must be before MikesLib
#define SERIALSOFTWARE
#define TxPort PORTA
#define RxPort PORT
#define TxBit
#define RxBit 1
#define TxTris TRISA

```

```

#define RxTris TRISA

#define BUFFER_SIZE 32
byte buffer[BUFFER_SIZE];
byte next_in = 0;
byte next_out = 0;

#int_ext
void serial_isr() {

}

#int_rb
void checkrb()
{
    if(INPUT(PIN_B4))
    {
        if(INPUT(PIN_A2))
        {
            OUTPUT_LOW(PIN_A2);
            printf("L0");
        }
        else if(!INPUT(PIN_A2))
        {
            OUTPUT_HIGH(PIN_A2);
            printf("L1");
        }
        else
            break;
    }

    else if(INPUT(PIN_B5))
    {
        if(INPUT(PIN_A3))
        {
            OUTPUT_LOW(PIN_A3);
            printf("F0");
        }
        else if(!INPUT(PIN_A3))
        {
            OUTPUT_HIGH(PIN_A3);
            printf("F1");
        }
        else
            break;
    }

    else if(INPUT(PIN_B6))
    {
        if(INPUT(PIN_B1))
        {
            OUTPUT_LOW(PIN_B1);
            printf("R0");
        }
        else if(!INPUT(PIN_B1))
        {
            OUTPUT_HIGH(PIN_B1);
            printf("R1");
        }
        else
            break;
    }

    else if(INPUT(PIN_B7))
    {
        if(INPUT(PIN_B2))
        {
            OUTPUT_LOW(PIN_B2);
            printf("S0");
        }
        else if(!INPUT(PIN_B2))

```

```

        {
            OUTPUT_HIGH(PIN_B2);
            printf("S1");
        }
        else
            break;
    }

    else
        break;
    delay_ms(200);
}

#define bkbhit (next_in!=next_out)

byte bgetc() {
    byte c;

    while(!bkbhit) ;
    c=buffer[next_out];
    next_out=(next_out+1) % BUFFER_SIZE;
    return(c);
}

void cut (byte long_array[2])
{
    int e;
    byte part[2];

    for(e=0; e<2; e++)
    {
        memcpy(&part[e], &long_array[e],1);
    }

    if(part[0] == 'L')
    {
        if(part[1] == '1')
            OUTPUT_HIGH(PIN_A2);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_A2);
    }
    else if(part[0] == 'F')
    {
        if(part[1] == '1')
            OUTPUT_HIGH(PIN_A3);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_A3);
    }
    else if(part[0] == 'R')
    {
        if(part[1] == '1')
            OUTPUT_HIGH(PIN_B1);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_B1);
    }
    else if(part[0] == 'S')
    {
        if(part[1] == '1')
            OUTPUT_HIGH(PIN_B2);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_B2);
    }
    else if(part[0] == 'D')
    {
        if(part[1] == '1')
            OUTPUT_HIGH(PIN_B3);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_B3);
    }
    else if(part[0] == 'W')
    {

```

```

        if(part[1] == '1')
            OUTPUT_HIGH(PIN_B0);
        else if(part[1] == '0')
            OUTPUT_LOW(PIN_B0);
    }

    else
        break;
}

main()
{
    int a, b;
    char c;

    byte read[2];

    // 0 = outputs ; 1 = inputs
    set_tris_a(0x00); //0000 0100
    set_tris_b(0xF0); //1111 0000

    OUTPUT_LOW(PIN_A2);
    OUTPUT_LOW(PIN_A3);
    //OUTPUT_LOW(PIN_A4);
    OUTPUT_LOW(PIN_B1);
    OUTPUT_LOW(PIN_B2);
    OUTPUT_LOW(PIN_B3);
    OUTPUT_LOW(PIN_B4);
    OUTPUT_LOW(PIN_B5);
    OUTPUT_LOW(PIN_B6);
    OUTPUT_LOW(PIN_B7);

    OUTPUT_HIGH(PIN_B0);
    delay_ms(1000);
    printf("PIC connected.\r\n");
    OUTPUT_LOW(PIN_B0);
    delay_ms(1000);

    enable_interrupts(global);
    enable_interrupts(RB_CHANGE);

    while(c = getch())
    {
        while(c != '~')
        {
            for( a=0; a < 2; a++)
            {
                read[a] = c;
                // cannot have any other command to avoid delay in receiving next character
                c = getch();
                if(c == '~')
                    break;
            }
            c = '~';
        }

        putc(read[0]);
        putc(read[1]);

        cut(read);
        c = 'e';

        printf("\n");
    }
}

```

APPENDIX G

JAVA CLASSES PROGRAM CODE

```
//=====
//Author      : Murni Masri
//File Name   : MainFrame.java
//Description  : This program is an application with advance GUI.
//             : It can communicate with a serial port and handle signals and data
//             : available on the serial port.
//=====

import java.awt.*;
import java.awt.event.*;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import java.io.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.ImageIcon;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.tree.*;
import javax.swing.tree.TreePath;

import java.util.*;
import java.util.TooManyListenersException;
import java.util.StringTokenizer;
import java.util.Properties;
import java.util.Enumeration;
import javax.comm.*;

public class MainFrame extends JFrame implements ActionListener
{
    final int HEIGHT = 550;
    final int WIDTH = 410;
    private Container container;

    private JButton jbOpen;
    private JButton jbClose;
    private JPanel jpButton;
    private JMenuBar menubar;
    private JMenu connectionMenu;
    private JMenuItem jmiOpenPort, jmiClosePort;

    private SerialParameters parameters;
    private SerialConnection connection;
    private JPanel jpMessage;
    private JTextArea messageAreaOut;
    private JTextArea messageAreaIn;

    private MyButton myButton;
    private Status status;

    private JPanel jpCenter;
    private JPanel jpTab;
    private JPanel jpTree;
    private JTabbedPane jtpTabPane;
    private JPanel jpModule;
}
```

```

private JPanel jpList;
private JTree jtTree;

private IntroView introview;
private GeneralView mGeneralView;
private KitchenView mKitchenView;
private LivingView mLivingView;
private DiningView mDiningView;
private BedroomView1 mBedroomView1;
private BedroomView2 mBedroomView2;
private BedroomView3 mBedroomView3;

private JScrollPane jspSend;
private JScrollPane jspReceive;
private TreePanel tree;
private UpperPanel clock;
private TimerThread timerthread;
private JPanel jpNorth;
private JLabel jlLogo;

String general = "[root, General]";
String living = "[root, Living Room]";
String dining = "[root, Dining Room]";
String kitchen = "[root, Kitchen]";
String bedroom1 = "[root, Bedroom 1]";
String bedroom2 = "[root, Bedroom 2]";
String bedroom3 = "[root, Bedroom 3]";

CardLayout cardlayout;
MyAccessPad access;

Dimension tabsize = new Dimension(600,700);
Dimension messagesize = new Dimension(150,250);
Color lightgray = new Color(225,225,225);
Color selected = new Color(130,130,253);
Color lightgreen = new Color(0,255,64);

public MainFrame()
{
    setTitle("Home Guard System");
    container = getContentPane();

    menubar = new JMenuBar();
    connectionMenu = new JMenu("Connection");
    jmiOpenPort = new JMenuItem("Open Serial Port");
    jmiClosePort = new JMenuItem("Close Serial Port");
    jmiClosePort.setEnabled(false);
    connectionMenu.add(jmiOpenPort);
    connectionMenu.add(jmiClosePort);
    menubar.add(connectionMenu);
    setJMenuBar(menubar);

    jmiOpenPort.addActionListener(this);
    jmiClosePort.addActionListener(this);

    messageAreaOut = new JTextArea();
    messageAreaOut.setBackground(selected);
    messageAreaOut.setForeground(Color.white);
    messageAreaOut.add(new JScrollPane());

    messageAreaIn = new JTextArea();
    messageAreaIn.setBackground(Color.black);
    messageAreaIn.setForeground(lightgreen);

    messageAreaIn.setEditable(false);

    jspSend = new JScrollPane(messageAreaOut);
    jspSend.setBorder(new TitledBorder("Send Message"));

    jspReceive = new JScrollPane(messageAreaIn);
    jspReceive.setBorder(new TitledBorder("Received Message"));
}

```

```

jpMessage = new JPanel();
jpMessage.setLayout(new GridLayout(2,1,20,20));
jpMessage.setPreferredSize(messagesize);
jpMessage.add(jspSend);
jpMessage.add(jspReceive);

parameters = new SerialParameters();
connection = new SerialConnection(this, parameters, messageAreaOut, messageAreaIn);

status = new Status();
introview = new IntroView();
mGeneralView = new GeneralView(this, connection, status);
mGeneralView.setDoubleBuffered(true);
mKitchenView = new KitchenView(this, connection, status);
mKitchenView.setDoubleBuffered(true);
mLivingView = new LivingView(this, connection, status);
mLivingView.setDoubleBuffered(true);
mDiningView = new DiningView(this, connection, status);
mDiningView.setDoubleBuffered(true);
mBedroomView1 = new BedroomView1(this, connection, status);
mBedroomView1.setDoubleBuffered(true);
mBedroomView2 = new BedroomView2(this, connection, status);
mBedroomView2.setDoubleBuffered(true);
mBedroomView3 = new BedroomView3(this, connection, status);
mBedroomView3.setDoubleBuffered(true);

jpModule = new JPanel();
jpModule.setMaximumSize(tabsize);
cardlayout = new CardLayout();
jpModule.setLayout(cardlayout);
jpModule.setBorder(new BevelBorder(BevelBorder.LOWERED));

jpModule.add(mKitchenView, "Kitchen");
jpModule.add(mGeneralView, "General");
jpModule.add(mLivingView, "Living Room");
jpModule.add(mDiningView, "Dining Room");
jpModule.add(mBedroomView1, "Bedroom 1");
jpModule.add(mBedroomView2, "Bedroom 2");
jpModule.add(mBedroomView3, "Bedroom 3");
jpModule.add(introview, "Welcome to Home Guard System");

introview.setVisible(true);
mKitchenView.setVisible(false);
mGeneralView.setVisible(false);
mLivingView.setVisible(false);
mDiningView.setVisible(false);
mBedroomView1.setVisible(false);
mBedroomView2.setVisible(false);
mBedroomView3.setVisible(false);

jpCenter = new JPanel();
jpCenter.setLayout(new BorderLayout(5,5));
jpCenter.setBorder(new EmptyBorder(5,5,5,5));
jpCenter.setBackground(new Color(180,180,194));

tree = new TreePanel(this);
jpTree = new JPanel();
jpTree.setLayout(new BorderLayout(5,5));
jpTree.setBackground(Color.white);
jpTree.setBorder(new LineBorder(Color.gray, 2));

jpTree.add(tree, BorderLayout.NORTH);
jpTree.add(jpMessage, BorderLayout.SOUTH);

clock = new UpperPanel();
jlLogo = new JLabel();
jlLogo.setIcon(new ImageIcon("image/logo_company.jpg"));
jlLogo.setVerticalAlignment(SwingConstants.TOP);
jlLogo.setVerticalTextPosition(SwingConstants.BOTTOM);
jpNorth = new JPanel();

```

```

    jpNorth.setLayout(new BorderLayout(10,10));
    jpNorth.setBackground(new Color(180,180,194));
    jpNorth.add(jlLogo, BorderLayout.WEST);
    jpNorth.add(clock, BorderLayout.CENTER);

    jpCenter.add(jpNorth, BorderLayout.NORTH);
    jpCenter.add(jpModule, BorderLayout.CENTER);
    jpCenter.add(jpTree, BorderLayout.WEST);
    container.setLayout(new BorderLayout());
    container.add(jpCenter, BorderLayout.CENTER);

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int WIDTH = screenSize.width;
    int HEIGHT = screenSize.height - 30;
    setLocation(0,0);
    setSize(WIDTH, HEIGHT);

    timerthread = new TimerThread(clock);
    timerthread.start();

    access = new MyAccessPad(this);
    access.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setVisible(false);
    access.setVisible(true);
}

public static void main(String args[])
{
    try
    {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    MainFrame frame = new MainFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(false);
}

//----- Action Performed -----

public void actionPerformed(ActionEvent e)
{
    String cmd = e.getActionCommand();

    if (cmd.equals("Open Serial Port"))
    {
        portOpened();
    }

    if (cmd.equals("Close Serial Port"))
    {
        portClosed();
    }
}

//----- Methods -----

public void portOpened()
{
    jmiOpenPort.setEnabled(false);
    try {
        connection.openConnection();
    }
    catch (SerialConnectionException e2) {

```



```

        JOptionPane.showMessageDialog (this,
            "Error opening port," +
            e2.getMessage() + "." +
            "Select new settings, try again.",
            "Port Error",
            JOptionPane.ERROR_MESSAGE);

        jmiOpenPort.setEnabled(true);
    }

    jmiClosePort.setEnabled(true);
}

public void portClosed()
{
    connection.closeConnection();
    jmiOpenPort.setEnabled(true);
    jmiClosePort.setEnabled(false);
}

class CloseHandler extends WindowAdapter
{
    MainFrame sd;

    public CloseHandler(MainFrame sd)
    {
        this.sd = sd;
    }

    public void windowClosing(WindowEvent e)
    {
        sd.shutdown();
    }
}

private void shutdown()
{
    connection.closeConnection();
    System.exit(1);
}

public void setLivingView()
{
    this.cardlayout.show(jpModule, "Living Room");
}

public void setDiningView()
{
    this.cardlayout.show(jpModule, "Dining Room");
}

public void setKitchenView()
{
    this.cardlayout.show(jpModule, "Kitchen");
}

public void setGeneralView()
{
    this.cardlayout.show(jpModule, "General");
}

public void setBedroomView1()
{
    this.cardlayout.show(jpModule, "Bedroom 1");
}

public void setBedroomView2()
{
    this.cardlayout.show(jpModule, "Bedroom 2");
}

```

```

}

public void setBedroomView3()
{
    this.cardlayout.show(jpModule, "Bedroom 3");
}

// ===== End of methods involved directly with MainFrame =====

//=====Child classes of MainFrame =====

//----- MyButton -----

class MyButton extends JPanel implements ActionListener
{
    private MainFrame parent;
    private SerialConnection connection;
    private JRadioButton buttonOn;
    private JRadioButton buttonOff;
    private Icon icon;
    final int HEIGHT = 100;
    final int WIDTH = 230;

    public MyButton(MainFrame parent, SerialConnection connection)
    {
        this.parent = parent;
        this.connection = connection;

        buttonOn = new JRadioButton("ON");
        buttonOn.setIcon(new ImageIcon("image/none.jpg"));
        buttonOn.setRolloverIcon(new ImageIcon("image/blurmetal_green.jpg"));
        buttonOn.setSelectedIcon(new ImageIcon("image/alien_green.jpg"));

        buttonOff = new JRadioButton("OFF");
        buttonOff.setIcon(new ImageIcon("image/none.jpg"));
        buttonOff.setRolloverIcon(new ImageIcon("image/blurmetal_red.jpg"));
        buttonOff.setSelectedIcon(new ImageIcon("image/alien_red.jpg"));

        ButtonGroup btg = new ButtonGroup();
        btg.add(buttonOn);
        btg.add(buttonOff);

        buttonOn.addActionListener(this);
        buttonOff.addActionListener(this);

        setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
        add(buttonOn);
        add(buttonOff);
    }

    public void actionPerformed(ActionEvent e)
    {
        String cmd = e.getActionCommand();

        if (connection.isOpen())
        {
            if (cmd.equals("ON")) // either one can
                //if (e.getSource() == buttonOn)
                {
                    connection.settingRTS(true);
                }

            if (cmd.equals("OFF"))
                //if (e.getSource() == buttonOff)
                {
                    connection.settingRTS(false);
                }
        }
    }
}

```

```

    }
}

} // end of MyButton -----

//----- SerialConnection -----

class SerialConnection implements SerialPortEventListener,CommPortOwnershipListener
{
    private MainFrame parent;
    private JTextArea messageAreaOut;
    private JTextArea messageAreaIn;
    private SerialParameters parameters;
    private OutputStream os;
    private InputStream is;
    private KeyHandler keyHandler;
    private CommPortIdentifier portId;
    private SerialPort sPort;
    private boolean open;
    private MyButton myButton;

    public SerialConnection(MainFrame parent, SerialParameters parameters,
        JTextArea messageAreaOut, JTextArea messageAreaIn)
    {
        this.parent = parent;
        this.parameters = parameters;
        this.messageAreaOut = messageAreaOut;
        this.messageAreaIn = messageAreaIn;
        open = false;
        myButton = new MyButton(parent, this);
    }

    public void openConnection() throws SerialConnectionException
    {
        try {
            portId = CommPortIdentifier.getPortIdentifier(parameters.getPortName());
        } catch (NoSuchPortException e) {
            throw new SerialConnectionException(e.getMessage());
        }

        try {
            sPort = (SerialPort)portId.open("MainFrame", 5000);
        } catch (PortInUseException e) {
            throw new SerialConnectionException(e.getMessage());
        }

        try {
            setConnectionParameters();
        } catch (SerialConnectionException e) {
            sPort.close();
            throw e;
        }

        try {
            os = sPort.getOutputStream();
            is = sPort.getInputStream();
        } catch (IOException e) {
            sPort.close();
            throw new SerialConnectionException("Error opening i/o streams");
        }

        keyHandler = new KeyHandler(os);
        messageAreaOut.addKeyListener(keyHandler);

        try {
            sPort.addEventListener(this);

```

```

    } catch (TooManyListenersException e) {
        sPort.close();
        throw new SerialConnectionException("too many listeners added");
    }

    sPort.notifyOnDataAvailable(true);
    sPort.notifyOnCTS(true);
    sPort.notifyOnDSR(true);
    sPort.notifyOnRingIndicator(true);
    sPort.notifyOnCarrierDetect(true);
    sPort.notifyOnBreakInterrupt(true);

    try {
        sPort.enableReceiveTimeout(30);
    } catch (UnsupportedCommOperationException e) {
    }

    portId.addPortOwnershipListener(this);

    sPort.setRTS(false);
    sPort.setDTR(true);

    open = true;
}

public void setConnectionParameters() throws SerialConnectionException
{
    int oldBaudRate = sPort.getBaudRate();
    int oldDatabits = sPort.getDataBits();
    int oldStopbits = sPort.getStopBits();
    int oldParity = sPort.getParity();
    int oldFlowControl = sPort.getFlowControlMode();

    try {
        sPort.setSerialPortParams(parameters.getBaudRate(),
                                   parameters.getDatabits(),
                                   parameters.getStopbits(),
                                   parameters.getParity());
    } catch (UnsupportedCommOperationException e) {
        parameters.setBaudRate(oldBaudRate);
        parameters.setDatabits(oldDatabits);
        parameters.setStopbits(oldStopbits);
        parameters.setParity(oldParity);
        throw new SerialConnectionException("Unsupported parameter");
    }

    try {
        sPort.setFlowControlMode(parameters.getFlowControlIn() |
                                 parameters.getFlowControlOut());
    } catch (UnsupportedCommOperationException e) {
        throw new SerialConnectionException("Unsupported flow control");
    }
}

public void closeConnection()
{
    if (!open) { return; }

    messageAreaOut.removeKeyListener(keyHandler);

    if (sPort != null)
    {
        try {
            os.close();
            is.close();
        } catch (IOException e) {
            System.err.println(e);
        }

        sPort.close();
    }
}

```

```

        portId.removePortOwnershipListener(this);
    }

    open = false;
}

public void sendBreak()
{
    sPort.sendBreak(1000);
}

public boolean isOpen()
{
    return open;
}

//-----Setting the RTS,DTR and serial data to transmit--

public void settingRTS(boolean rts)
{
    sPort.setRTS(rts);
}

public void settingDTR(boolean dtr)
{
    sPort.setDTR(dtr);
}

public void setlight(boolean light)
{
    byte[] on = {'L','1','1'};
    byte[] off = {'L','0','0'};

    try
    {
        if(light)
        {
            os.write(on,0,3);
        }
        else
        {
            os.write(off,0,3);
        }
    }catch (IOException ex)
    {
        System.err.println(ex);
        return;
    }
}

public void setDoor(boolean door)
{
    byte[] on = {'D','1','1'};
    byte[] off = {'D','0','0'};

    try
    {
        if(door)
        {
            os.write(on,0,3);
        }
        else
        {
            os.write(off,0,3);
        }
    }catch (IOException ex)
    {
        System.err.println(ex);
        return;
    }
}

```

```

    }
}

public void setWindow(boolean window)
{
    byte[] on = {'R','1','1'};
    byte[] off = {'R','0','0'};

    try
    {
        if(window)
        {
            os.write(on,0,3);
        }
        else
        {
            os.write(off,0,3);
        }
    }catch (IOException ex)
    {
        System.err.println(ex);
        return;
    }
}

public void setAircond(boolean aircond)
{
    byte[] on = {'R','1','1'};
    byte[] off = {'R','0','0'};

    try
    {
        if(aircond)
        {
            os.write(on,0,3);
        }
        else
        {
            os.write(off,0,3);
        }
    }catch (IOException ex)
    {
        System.err.println(ex);
        return;
    }
}

public void setFan(boolean fan)
{
    byte[] on = {'F','1','1'};
    byte[] off = {'F','0','0'};

    try
    {
        if(fan)
        {
            os.write(on,0,3);
        }
        else
        {
            os.write(off,0,3);
        }
    }catch (IOException ex)
    {
        System.err.println(ex);
        return;
    }
}

public void setSmoke(boolean smoke)
{

```

```

byte[] on = {'D','1','1'};
byte[] off = {'D','0','0'};

try
{
    if(smoke)
    {
        os.write(on,0,3);
    }
    else
    {
        os.write(off,0,3);
    }
}catch (IOException ex)
{
    System.err.println(ex);
    return;
}
}

```

//-----End of setting the RTS,DTR and serial data to transmit-----

//-----Serial port Event handling-----

```

public void serialEvent(SerialPortEvent e)
{
    StringBuffer inputBuffer = new StringBuffer();
    int newData = 0;

    switch (e.getEventType())
    {
        case SerialPortEvent.DATA_AVAILABLE:
            while (newData != -1)
            {
                try {
                    newData = is.read();
                    if (newData == -1) {
                        break;
                    }
                    if ('\r' == (char)newData) {
                        inputBuffer.append('\n');
                    } else {
                        inputBuffer.append((char)newData);
                    }
                } catch (IOException ex) {
                    System.err.println(ex);
                    return;
                }
            }

            messageAreaIn.append(new String(inputBuffer));
            break;
        case SerialPortEvent.BI:
            messageAreaIn.append("\n--- BREAK RECEIVED ---\n");
            break;
        case SerialPortEvent.CTS:
            if (sPort.isCTS() == false)
                messageAreaIn.append("\n--- CTS = OFF ---");
            if (sPort.isCTS() == true)
                messageAreaIn.append("\n--- CTS = ON ---");
            break;
        case SerialPortEvent.DSR:
            if (sPort.isDSR() == false)
            {
                messageAreaIn.append("\n--- DSR = OFF ---");
            }
            if (sPort.isDSR() == true)

```

```

        {
            messageAreaIn.append("\n--- DSR = ON ---");
        }
        break;

    case SerialPortEvent.RI:
        if (sPort.isCTS() == false)

            messageAreaIn.append("\n--- RI = OFF ---");
        if (sPort.isCTS() == true)
            messageAreaIn.append("\n--- RI = ON ---");
            break;

    case SerialPortEvent.CD:
        if (sPort.isCTS() == false)
        {
            messageAreaIn.append("\n--- CD = OFF ---");

        }
        if (sPort.isCTS() == true)
        {
            messageAreaIn.append("\n--- CD = ON ---");
            parent.setGeneralView();
            parent.mGeneralView.mLivingsmoke.setAlarm();

        }
        break;

    }

}

//-----end of Serial port Event handling-----

public void ownershipChange(int type)
{
    if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED)
    {
        PortRequestedDialog prd = new PortRequestedDialog(parent);
    }
}

class KeyHandler extends KeyAdapter
{
    OutputStream os;

    public KeyHandler(OutputStream os)
    {
        super();
        this.os = os;
    }

    public void keyTyped(KeyEvent evt)
    {
        char newCharacter = evt.getKeyChar();
        try {
            os.write((int)newCharacter);
        } catch (IOException e) {
            System.err.println("OutputStream write error: " + e);
        }
    }
}

} // end of SerialConnection -----

//----- Status -----

class Status
{
    MainFrame parent;
}

```



```

boolean frontdoor = true,
backdoor = false;
boolean dininglight = false,
    diningfan = false,
    diningsmoke = false;
boolean livingwindow = false,
    livinglight = true,
    livingfan = true,
    livingaircond = true,
    livingsmoke = false;
boolean kitchenwindow = false,
    kitchenlight = false,
    kitchenfan = false,
    kitchensmoke = false;
boolean bedroomdoor1 = true,
    bedroomwindow1 = true,
    bedroomlight1 = true,
    bedroomfan1 = true,
    bedroomaircond1 = true,
    bedroomsmoke1 = true;
boolean bedroomdoor2 = true,
    bedroomwindow2 = true,
    bedroomlight2 = true,
    bedroomfan2 = true,
    bedroomaircond2 = true,
    bedroomsmoke2 = true;
boolean bedroomdoor3 = false,
    bedroomwindow3 = false,
    bedroomlight3 = false,
    bedroomfan3 = false,
    bedroomaircond3 = false,
    bedroomsmoke3 = false;
int temperature = 25;

public Status()
{
}

public boolean getStatus(String name)
{
    if (name == "frontdoor")
        return this.frontdoor;
    else if (name == "backdoor")
        return backdoor;
    else if (name == "dininglight")
        return this.dininglight;
    else if (name == "diningfan")
        return this.diningfan;
    else if (name == "diningsmoke")
        return this.diningsmoke;
    else if (name == "livingwindow")
        return this.livingwindow;
    else if (name == "livinglight")
        return this.livinglight;
    else if (name == "livingfan")
        return this.livingfan;
    else if (name == "livingaircond")
        return this.livingaircond;
    else if (name == "livingsmoke")
        return this.livingsmoke;
    else if (name == "kitchenwindow")
        return this.kitchenwindow;
    else if (name == "kitchenlight")
        return this.kitchenlight;
    else if (name == "kitchenfan")
        return this.kitchenfan;
    else if (name == "kitchensmoke")
        return this.kitchensmoke;
    else if (name == "bedroomdoor1")

```

```

        return this.bedroomdoor1;
    else if (name == "bedroomwindow1")
        return this.bedroomwindow1;
    else if (name == "bedroomlight1")
        return this.bedroomlight1;
    else if (name == "bedroomfan1")
        return this.bedroomfan1;
    else if (name == "bedroomaircond1")
        return this.bedroomaircond1;
    else if (name == "bedroomsmoke1")
        return this.bedroomsmoke1;
    else if (name == "bedroomdoor2")
        return this.bedroomdoor2;
    else if (name == "bedroomwindow2")
        return this.bedroomwindow2;
    else if (name == "bedroomlight2")
        return this.bedroomlight2;
    else if (name == "bedroomfan2")
        return this.bedroomfan2;
    else if (name == "bedroomaircond2")
        return this.bedroomaircond2;
    else if (name == "bedroomsmoke2")
        return this.bedroomsmoke2;
    else if (name == "bedroomdoor3")
        return this.bedroomdoor3;
    else if (name == "bedroomwindow3")
        return this.bedroomwindow3;
    else if (name == "bedroomlight3")
        return this.bedroomlight3;
    else if (name == "bedroomfan3")
        return this.bedroomfan3;
    else if (name == "bedroomaircond3")
        return this.bedroomaircond3;
    else if (name == "bedroomsmoke3")
        return this.bedroomsmoke3;

    else
        return false;
}

public void setStatus(String name, boolean state)
{
    if (name == "frontdoor")
        this.frontdoor = state;
    else if (name == "backdoor")
    {
        JOptionPane optionPane = new JOptionPane("Back door changes state.",
            JOptionPane.ERROR_MESSAGE );
        JDialog dialog = optionPane.createDialog(parent, "Port Error");
        dialog.show();
        this.backdoor = state;
    }
    else if (name == "dininglight")
        this.dininglight = state;
    else if (name == "diningfan")
        this.diningfan = state;
    else if (name == "diningsmoke")
        this.diningsmoke = state;
    else if (name == "livingwindow")
        this.livingwindow = state;
    else if (name == "livinglight")
        this.livinglight = state;
    else if (name == "livingfan")
        this.livingfan = state;
    else if (name == "livingaircond")
        this.livingaircond = state;
    else if (name == "livingsmoke")
        this.livingsmoke = state;
    else if (name == "kitchenwindow")
        this.kitchenwindow = state;
    else if (name == "kitchenlight")

```

```

        this.kitchenlight = state;
    else if (name == "kitchenfan")
        this.kitchenfan = state;
    else if (name == "kitchensmoke")
        this.kitchensmoke = state;
    else if (name == "bedroomdoor1")
        this.bedroomdoor1 = state;
    else if (name == "bedroomwindow1")
        this.bedroomwindow1 = state;
    else if (name == "bedroomlight1")
        this.bedroomlight1 = state;
    else if (name == "bedroomfan1")
        this.bedroomfan1 = state;
    else if (name == "bedroomaircond1")
        this.bedroomaircond1 = state;
    else if (name == "bedroomsmoke1")
        this.bedroomsmoke1 = state;
    else if (name == "bedroomdoor2")
        this.bedroomdoor2 = state;
    else if (name == "bedroomwindow2")
        this.bedroomwindow2 = state;
    else if (name == "bedroomlight2")
        this.bedroomlight2 = state;
    else if (name == "bedroomfan2")
        this.bedroomfan2 = state;
    else if (name == "bedroomaircond2")
        this.bedroomaircond2 = state;
    else if (name == "bedroomsmoke2")
        this.bedroomsmoke2 = state;
    else if (name == "bedroomdoor3")
        this.bedroomdoor3 = state;
    else if (name == "bedroomwindow3")
        this.bedroomwindow3 = state;
    else if (name == "bedroomlight3")
        this.bedroomlight3 = state;
    else if (name == "bedroomfan3")
        this.bedroomfan3 = state;
    else if (name == "bedroomaircond3")
        this.bedroomaircond3 = state;
    else if (name == "bedroomsmoke3")
        this.bedroomsmoke3 = state;

    else
    {
        JOptionPane optionPane = new JOptionPane("No Valid Status Assigned",
            JOptionPane.ERROR_MESSAGE );
        JDialog dialog = optionPane.createDialog(parent, "Port Error");
        dialog.show();
    }
}
} // end of Status class -----

```

//----- Module -----

```

class Module extends JPanel implements ActionListener
{
    private MainFrame parent;
    private SerialConnection connection;

    private Status status;
    private String title;
    private String iconName;
    private String on = "ON";
    private String off = "OFF";
    private String open = "OPEN";
    private String close = "CLOSE";
    private String lock = "LOCK";
    private String unlock = "UNLOCK";
    private String disable = "DISABLE";
    private String enable = "ENABLE";
}

```

```

private String alert = "ALERT";
private String name;
boolean state;
boolean locked;
boolean detector;

private JLabel jlImage;
private ImageIcon iconOn;
private ImageIcon iconOff;
private ImageIcon iconOther;
private ImageIcon doorlock = new ImageIcon("image/door_lock.jpg");
private ImageIcon windowlock = new ImageIcon("image/window_lock.jpg");

private JLabel jlStatus;
private JLabel jlCurrentStatus;

private JPanel jpStatus;
private JPanel jpAction;
private JRadioButton jrbOn;
private JRadioButton jrbOff;
private JButton jbOption;

private JPanel jpSouth;
private JPanel jpNorth;
private JPanel jpBase;

public Module(SerialConnection connection, Status status, String title,
              String iconName, String name)
{
    this.connection = connection;
    this.status = status;
    this.name = name;

    this.state = status.getStatus(name);
    this.iconName = iconName;
    String sIconOn = "image/" + iconName + "_on1.jpg";
    String sIconOff = "image/" + iconName + "_off1.jpg";
    this.iconOn = new ImageIcon(sIconOn);
    this.iconOff = new ImageIcon(sIconOff);

    jpBase = new JPanel();
    jpBase.setLayout(new BorderLayout(10,10));
    jpBase.setBorder(new TitledBorder(title));

    jlImage = new JLabel();
    jlStatus = new JLabel("Status :");
    jlCurrentStatus = new JLabel();

    jrbOn = new JRadioButton();
    jrbOn.setIcon(new ImageIcon("image/nonel.jpg"));
    jrbOn.setRolloverIcon(new ImageIcon("image/blurmetal_green1.jpg"));
    jrbOn.setSelectedIcon(new ImageIcon("image/alien_green1.jpg"));
    this.setjrbOn();
    jrbOn.addActionListener(this);
    jrbOff = new JRadioButton();
    jrbOff.setIcon(new ImageIcon("image/nonel.jpg"));
    jrbOff.setRolloverIcon(new ImageIcon("image/blurmetal_red1.jpg"));
    jrbOff.setSelectedIcon(new ImageIcon("image/alien_red1.jpg"));
    this.setjrbOff();
    jrbOff.addActionListener(this);
    jpAction = new JPanel();
    jpAction.setBorder(new TitledBorder("Action"));
    jpAction.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 5));
    jpAction.add(jrbOn);
    jpAction.add(jrbOff);
    ButtonGroup btgAction = new ButtonGroup();
    btgAction.add(jrbOn);
    btgAction.add(jrbOff);

    if (state)
    {

```

```

j1Image.setIcon(iconOn);
if (iconName == "door" || iconName == "window")
{
    j1CurrentStatus.setText(open);
    this.disableAction();
}
else if (iconName == "fan" || iconName == "aircond" || iconName == "light")
{
    j1CurrentStatus.setText(on);
    jrbOn.setSelected(true);
}
else if (iconName == "smoke")
{
    j1CurrentStatus.setText(enable);
    jrbOn.setSelected(true);
}
else
    j1CurrentStatus.setText(on);
}
else
{
    j1Image.setIcon(iconOff);

    if (iconName == "door" || iconName == "window")
    {
        j1CurrentStatus.setText(close);
        this.enableAction();
        jrbOff.setSelected(true);
    }
    else if (iconName == "fan" || iconName == "aircond" || iconName == "light")
    {
        j1CurrentStatus.setText(off);
        jrbOff.setSelected(true);
    }
    else if (iconName == "smoke")
    {
        j1CurrentStatus.setText(disable);
        jrbOff.setSelected(true);
    }
    else
        j1CurrentStatus.setText(off);
}

j1Image.setBorder(new BevelBorder(BevelBorder.LOWERED));
Dimension dImage = new Dimension(iconOn.getIconWidth(),
    iconOn.getIconHeight());
j1Image.setSize(dImage);
j1CurrentStatus.setBorder(new BevelBorder(BevelBorder.LOWERED));

jpStatus = new JPanel();
jpStatus.setLayout(new GridLayout(2,1,5,5));
jpStatus.add(j1Status);
jpStatus.add(j1CurrentStatus);

jpNorth = new JPanel();
jpNorth.setLayout(new BorderLayout(20,10));
jpNorth.setBorder(new EmptyBorder(10,10,10,10));
jpNorth.add(j1Image, BorderLayout.WEST);
jpNorth.add(jpStatus, BorderLayout.CENTER);

jbOption = new JButton("Option");
jbOption.addActionListener(this);

jpSouth = new JPanel();
jpSouth.setLayout(new BorderLayout(10,10));
jpSouth.setBorder(new EmptyBorder(10,10,10,10));
jpSouth.add(jpAction, BorderLayout.CENTER);
jpSouth.add(jbOption, BorderLayout.SOUTH);

jpBase.add(jpNorth, BorderLayout.NORTH);

```

```

jpbBase.add(jpbSouth, BorderLayout.SOUTH);

setLayout(new BorderLayout(10,10));
setBorder(new LineBorder(lightgray, 3));
add(jpbBase);
}

public void actionPerformed(ActionEvent e)
{
    if (!connection.isOpen())
    {
        JOptionPane.showMessageDialog (this,
            "The serial port is not opened. ",
            "Port Error",
            JOptionPane.ERROR_MESSAGE);
    }

    else
    {
        if (jrbOn.isSelected())
        {
            jlImage.setIcon(iconOn);

            if (iconName == "door")
            {
                jlCurrentStatus.setText(lock);
                jlImage.setIcon(doorlock);

                if (name == "backdoor")
                {
                    connection.settingRTS(true);
                    connection.setDoor(true);
                }
            }

            else if (iconName == "window")
            {
                jlCurrentStatus.setText(lock);
                jlImage.setIcon(windowlock);

                if (name == "kitchenwindow")
                {
                    connection.setWindow(true);
                }
            }

            else if (iconName == "fan" || iconName == "aircond" ||
                iconName == "light")
            {
                jlCurrentStatus.setText(on);

                if (name == "kitchenfan")
                {
                    connection.setFan(true);
                }

                if (name == "kitchenlight")
                {
                    connection.setLight(true);
                }
            }

            else if (iconName == "smoke")
            {
                jlCurrentStatus.setText(enable);

                if (name == "livingsmoke")
                {
                    //connection.settingDTR(true);
                }
            }
        }
    }
}

```

```

        if (name == "kitchensmoke")
        {
            connection.setSmoke(true);
        }
    }
    else
        jlCurrentStatus.setText(on);
}

else if (jrbOff.isSelected())
{
    jlImage.setIcon(iconOff);

    if (iconName == "door" || iconName == "window")
    {
        jlCurrentStatus.setText(close);

        if (name == "backdoor")
        {
            connection.settingRTS(false);
            connection.setDoor(false);
        }

        if (name == "kitchenwindow")
        {
            connection.setWindow(false);
        }
    }
}
else if (iconName == "fan" || iconName == "aircond" ||
        iconName == "light")
{
    jlCurrentStatus.setText(off);

    if (name == "kitchenfan")
    {
        connection.setFan(false);
    }

    if (name == "kitchenlight")
    {
        connection.setLight(false);
    }
}
else if (iconName == "smoke")
{
    jlCurrentStatus.setText(disable);

    if (name == "livingsmoke")
    {
        //connection.settingDTR(false);
    }

    if (name == "kitchensmoke")
    {
        connection.setSmoke(false);
    }
}
else
    jlCurrentStatus.setText(off);
}
}

}

public void setJlImageOther()
{
    this.jlImage.setIcon(iconOther);
}
}

```

```

public void setIconOther(String icon)
{
    this.iconOther = new ImageIcon(icon);
}

public boolean getState()
{
    return this.state;
}

public void setjlcSAAlert()
{
    this.jlcCurrentStatus.setText(alert);
}

public void setjlcSDisable()
{
    this.jlcCurrentStatus.setText(disable);
}

public void setjlcSEnable()
{
    this.jlcCurrentStatus.setText(enable);
}

public void disableAction()
{
    this.jrbOn.setEnabled(false);
    this.jrbOff.setEnabled(false);
}

public void enableAction()
{
    this.jrbOn.setEnabled(true);
    this.jrbOff.setEnabled(true);
}

public void setjrbOnLock()
{
    this.jrbOn.setText(lock);
}

public void setjrbOffUnlock()
{
    this.jrbOn.setText(unlock);
}

public void setjrbOnEnable()
{
    this.jrbOn.setText(enable);
}

public void setjrbOffDisable()
{
    this.jrbOn.setText(disable);
}

public void setjrbOn()
{
    if (iconName == "door" || iconName == "window")
    {
        jrbOn.setText(lock);
    }
    else if (iconName == "fan" || iconName == "aircond" || iconName == "light")
    {
        jrbOn.setText(on);
    }
    else if (iconName == "smoke")
    {
        jrbOn.setText(enable);
    }
}

```



```

        else
            jrbOn.setText(on);
    }

    public void setjrbOff()
    {
        if (iconName == "door" )} iconName == "window")
        {
            jrbOff.setText(unlock);
        }
        else if (iconName == "fan" || iconName == "aircond" || iconName == "light")
        {
            jrbOff.setText(off);
        }
        else if (iconName == "smoke")
        {
            jrbOff.setText(disable);
        }
        else
            jrbOff.setText(off);
    }

    public void setSerialConnection(SerialConnection connection)
    {
        this.connection = connection;
    }

    public void setAlarm()
    {
        setDoubleBuffered(true);

        for (int a=0; a<6; a++)
        {
            setBorder(new LineBorder(Color.red, 3));
            for (int j=0; j<50000000; j++ ) {}
            setBorder(new LineBorder(lightgray, 3));
            for (int j=0; j<50000000; j++ ) {}
            setBorder(new LineBorder(Color.red, 3));
            for (int j=0; j<50000000; j++ ) {}
            setBorder(new LineBorder(lightgray, 3));
            for (int j=0; j<50000000; j++ ) {}
            setBorder(new LineBorder(Color.red, 3));
            for (int j=0; j<50000000; j++ ) {}
            setBorder(new LineBorder(lightgray, 3));
            for (int j=0; j<50000000; j++ ) {}
        }
    }

} // end of Module class -----

//----- IntroView -----

class IntroView extends JPanel
{
    final int WIDTHH = 549;
    final int HEIGHT = 485;
    Image image = new ImageIcon("image/intropane.jpg").getImage();
    JPanel jpRoom;
    JLabel jlRoom;

    public IntroView()
    {
        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
    }
}

```

```

        jlRoom = new JLabel("Welcome to Home Guard System");
        jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);
        this.add(jpRoom);

        setBackground(Color.black);
        setSize(WIDTH, HEIGHT);

        this.validate();

    }
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        g.drawImage(image, 150, 100, this);
        revalidate();
    }
} //end of IntroView -----

//----- GeneralView -----
class GeneralView extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mFrontdoor;
    private Module mBackdoor;
    private Module mLivingsmoke;

    private String[] sfd = {"Front Door","door","frontdoor"};
    private String[] sbd = {"Back Door","door","backdoor"};
    private String[] sls = {"Smoke Detector","smoke","livingsmoke"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "General";

    public GeneralView(MainFrame parent, SerialConnection connection, Status status)
    {
        this.parent = parent;
        this.connection = connection;
        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mFrontdoor = new Module (connection, status, this.getTitle(sfd),
this.getIconName(sfd), this.getName(sfd));
        mBackdoor = new Module (connection, status, this.getTitle(sbd),
this.getIconName(sbd),this.getName(sbd));
        mLivingsmoke = new Module (connection, status, this.getTitle(sls),
this.getIconName(sls),this.getName(sls));

        mFrontdoor.setPreferredSize(maxSize);
        mBackdoor.setPreferredSize(maxSize);
        mLivingsmoke.setPreferredSize(maxSize);

        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
        jlRoom = new JLabel(room);
        jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);

```

```

        jpInnerModule = new JPanel();
        jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
        jpInnerModule.setBackground(new Color(200,200,210));
        jpInnerModule.add(mFrontdoor);
        jpInnerModule.add(mBackdoor);
        jpInnerModule.add(mLivingSmoke);

        setLayout(new BorderLayout(10,10));
        add(jpRoom, BorderLayout.NORTH);
        add(jpInnerModule, BorderLayout.CENTER);

        setBackground(new Color(200,200,210));

    }

    public String getTitle(String[] module)
    {
        return module[0];
    }

    public String getIconName(String[] module)
    {
        return module[1];
    }

    public String getName(String[] module)
    {
        return module[2];
    }

    public void setModuleClear()
    {
        mFrontdoor.setBorder(new LineBorder(lightgray, 3));
        mBackdoor.setBorder(new LineBorder(lightgray, 3));
        mLivingSmoke.setBorder(new LineBorder(lightgray, 3));
    }

} // end of GeneralView class -----
//----- KitchenView -----

class KitchenView extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mKitchenwindow;
    private Module mKitchensmoke;
    private Module mKitchenfan;
    private Module mKitchenlight;
    private String[] sbd = {"Back Door", "door", "backdoor"};
    private String[] skw = {"Window", "window", "kitchenwindow"};
    private String[] skf = {"Fan", "fan", "kitchenfan"};
    private String[] sks = {"Smoke Detector", "smoke", "kitchensmoke"};
    private String[] skl = {"Light", "light", "kitchenlight"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "Kitchen";

    public KitchenView(MainFrame parent, SerialConnection connection, Status status)
    {
        this.parent = parent;
        this.connection = connection;
    }

```

```

        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mKitchenwindow = new Module (connection, status, this.getTitle(skw),
this.geticonName(skw),this.getName(skw));
        mKitchensmoke = new Module (connection, status, this.getTitle(sks),
this.geticonName(sks),this.getName(sks));
        mKitchenfan = new Module (connection, status, this.getTitle(skf),
this.geticonName(skf),this.getName(skf));
        mKitchenlight = new Module (connection, status, this.getTitle(skl),
this.geticonName(skl), this.getName(skl));

        mKitchenwindow.setPreferredSize(maxSize);
        mKitchensmoke.setPreferredSize(maxSize);
        mKitchenfan.setPreferredSize(maxSize);
        mKitchenlight.setPreferredSize(maxSize);

        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
        jlRoom = new JLabel(room);
        jpRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);
        jpInnerModule = new JPanel();
        jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
        jpInnerModule.setBackground(new Color(200,200,210));

        jpInnerModule.add(mKitchenlight);
        jpInnerModule.add(mKitchenwindow);
        jpInnerModule.add(mKitchenfan);
        jpInnerModule.add(mKitchensmoke);

        setLayout(new BorderLayout(10,10));
        add(jpRoom, BorderLayout.NORTH);
        add(jpInnerModule, BorderLayout.CENTER);

        setBackground(new Color(200,200,210));
    }

    public String getTitle(String[] module)
    {
        return module[0];
    }

    public String geticonName(String[] module)
    {
        return module[1];
    }

    public String getName(String[] module)
    {
        return module[2];
    }

    public void setModuleClear()
    {
        mKitchenwindow.setBorder(new LineBorder(lightgray, 3));
        mKitchensmoke.setBorder(new LineBorder(lightgray, 3));
        mKitchenfan.setBorder(new LineBorder(lightgray, 3));
        mKitchenlight.setBorder(new LineBorder(lightgray, 3));
    }

} // end of KitchenView class -----
//----- LivingView -----
class LivingView extends JPanel

```

```

{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mLivinglight;
    private Module mLivingwindow;
    private Module mLivingsmoke;
    private Module mLivingfan;
    private Module mLivingaircond;
    private String[] sll = {"Light", "light", "livinglight"};
    private String[] slw = {"Window", "window", "livingwindow"};
    private String[] slf = {"Fan", "fan", "livingfan"};
    private String[] sla = {"Air-Conditioner", "aircond", "livingaircond"};
    private String[] sls = {"Smoke Detector", "smoke", "livingsmoke"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "Living Room";

    public LivingView(MainFrame parent, SerialConnection connection, Status status)
    {
        this.parent = parent;
        this.connection = connection;
        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mLivinglight = new Module (connection, status, this.getTitle(sll),
this.getIconName(sll), this.getName(sll));
        mLivingwindow = new Module (connection, status, this.getTitle(slw),
this.getIconName(slw), this.getName(slw));
        mLivingsmoke = new Module (connection, status, this.getTitle(sls),
this.getIconName(sls), this.getName(sls));
        mLivingfan = new Module (connection, status, this.getTitle(slf),
this.getIconName(slf), this.getName(slf));
        mLivingaircond = new Module (connection, status, this.getTitle(sla),
this.getIconName(sla), this.getName(sla));

        mLivinglight.setPreferredSize(maxSize);
        mLivingwindow.setPreferredSize(maxSize);
        mLivingsmoke.setPreferredSize(maxSize);
        mLivingfan.setPreferredSize(maxSize);
        mLivingaircond.setPreferredSize(maxSize);

        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
        jlRoom = new JLabel(room);
        jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);
        jpInnerModule = new JPanel();
        jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
        jpInnerModule.setBackground(new Color(200,200,210));

        jpInnerModule.add(mLivinglight);
        jpInnerModule.add(mLivingwindow);
        jpInnerModule.add(mLivingfan);
        jpInnerModule.add(mLivingaircond);
        jpInnerModule.add(mLivingsmoke);

        setLayout(new BorderLayout(10,10));
        add(jpRoom, BorderLayout.NORTH);
        add(jpInnerModule, BorderLayout.CENTER);
    }
}

```

```

        setBackground(new Color(200,200,210));
    }

    public String getTitle(String[] module)
    {
        return module[0];
    }

    public String geticonName(String[] module)
    {
        return module[1];
    }

    public String getName(String[] module)
    {
        return module[2];
    }

    public void setModuleClear()
    {
        mLivinglight.setBorder(new LineBorder(lightgray, 3));
        mLivingwindow.setBorder(new LineBorder(lightgray, 3));
        mLivingsmoke.setBorder(new LineBorder(lightgray, 3));
        mLivingfan.setBorder(new LineBorder(lightgray, 3));
        mLivingaircond.setBorder(new LineBorder(lightgray, 3));
    }
} // end of LivingView class -----

//----- DiningView -----
class DiningView extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mDininglight;
    private Module mDiningfan;
    private Module mDiningsmoke;
    private String[] sdl = {"Light","light","dininglight"};
    private String[] sdf = {"Fan","fan","diningfan"};
    private String[] sds = {"Smoke Detector","smoke","diningsmoke"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "Dining Room";

    public DiningView(MainFrame parent, SerialConnection connection, Status status)
    {
        this.parent = parent;
        this.connection = connection;
        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mDininglight = new Module (connection, status, this.getTitle(sdl),
this.geticonName(sdl), this.getName(sdl));
        mDiningfan = new Module (connection, status, this.getTitle(sdf),
this.geticonName(sdf),this.getName(sdf));
        mDiningsmoke = new Module (connection, status, this.getTitle(sds),
this.geticonName(sds),this.getName(sds));

        mDininglight.setPreferredSize(maxSize);
        mDiningfan.setPreferredSize(maxSize);
    }
}

```

```

mDiningsmoke.setPreferredSize(maxSize);

jpRoom = new JPanel();
jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
jpRoom.setBackground(new Color(200,200,210));
jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
jlRoom = new JLabel(room);
jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
jpRoom.add(jlRoom);
jpInnerModule = new JPanel();
jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
jpInnerModule.setBackground(new Color(200,200,210));

jpInnerModule.add(mDininglight);
jpInnerModule.add(mDiningfan);
jpInnerModule.add(mDiningsmoke);

setLayout(new BorderLayout(10,10));
add(jpRoom, BorderLayout.NORTH);
add(jpInnerModule, BorderLayout.CENTER);

setBackground(new Color(200,200,210));
}

public String getTitle(String[] module)
{
    return module[0];
}

public String geticonName(String[] module)
{
    return module[1];
}

public String getName(String[] module)
{
    return module[2];
}

public String getRoom()
{
    return this.room;
}

public void setModuleClear()
{
    mDininglight.setBorder(new LineBorder(lightgray, 3));
    mDiningfan.setBorder(new LineBorder(lightgray, 3));
    mDiningsmoke.setBorder(new LineBorder(lightgray, 3));
}

} // end of DiningView class -----

//----- BedroomView1 -----
class BedroomView1 extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mBedroomdoor;
    private Module mBedroomlight;
    private Module mBedroomwindow;
    private Module mBedroomsmoke;
    private Module mBedroomfan;
    private Module mBedroomaircond;
    private String[] sbd = {"Door", "door", "bedroomdoor1"};
    private String[] sbl = {"Light", "light", "bedroomlight1"};
    private String[] sbw = {"Window", "window", "bedroomwindow1"};
}

```

```

private String[] sbf = {"Fan", "fan", "bedroomfan1"};
private String[] sba = {"Air-Conditioner", "aircond", "bedroomaircond1"};
private String[] sbs = {"Smoke Detector", "smoke", "bedroomsmoke1"};

private Dimension maxSize = new Dimension(250,250);
final int HEIGHT = 700;
final int WIDTH = 550;

private JPanel jpRoom;
private JPanel jpInnerModule;
private JLabel jlRoom;
private String room = "Bedroom 1";

public BedroomView1(MainFrame parent, SerialConnection connection, Status
status)
{
    this.parent = parent;
    this.connection = connection;
    this.status = status;

    enableEvents(AWTEvent.WINDOW_EVENT_MASK);

    mBedroomdoor = new Module (connection, status, this.getTitle(sbd),
this.getIconName(sbd), this.getName(sbd));
    mBedroomlight = new Module (connection, status, this.getTitle(sbl),
this.getIconName(sbl), this.getName(sbl));
    mBedroomwindow = new Module (connection, status, this.getTitle(sbw),
this.getIconName(sbw), this.getName(sbw));
    mBedroomsmoke = new Module (connection, status, this.getTitle(sbs),
this.getIconName(sbs), this.getName(sbs));
    mBedroomfan = new Module (connection, status, this.getTitle(sbf),
this.getIconName(sbf), this.getName(sbf));
    mBedroomaircond = new Module (connection, status, this.getTitle(sba),
this.getIconName(sba), this.getName(sba));

    mBedroomdoor.setPreferredSize(maxSize);
    mBedroomlight.setPreferredSize(maxSize);
    mBedroomwindow.setPreferredSize(maxSize);
    mBedroomsmoke.setPreferredSize(maxSize);
    mBedroomfan.setPreferredSize(maxSize);
    mBedroomaircond.setPreferredSize(maxSize);

    jpRoom = new JPanel();
    jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
    jpRoom.setBackground(new Color(200,200,210));
    jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
    jlRoom = new JLabel (room);
    jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
    jpRoom.add(jlRoom);
    jpInnerModule = new JPanel();
    jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
    jpInnerModule.setBackground(new Color(200,200,210));

    jpInnerModule.add(mBedroomdoor);
    jpInnerModule.add(mBedroomlight);
    jpInnerModule.add(mBedroomwindow);
    jpInnerModule.add(mBedroomfan);
    jpInnerModule.add(mBedroomaircond);
    jpInnerModule.add(mBedroomsmoke);

    setLayout(new BorderLayout(10,10));
    add(jpRoom, BorderLayout.NORTH);
    add(jpInnerModule, BorderLayout.CENTER);

    setBackground(new Color(200,200,210));
}

public String getTitle(String[] module)
{

```



```

    return module[0];
}

public String geticonName(String[] module)
{
    return module[1];
}

public String getName(String[] module)
{
    return module[2];
}

public void setModuleClear()
{
    mBedroomdoor.setBorder(new LineBorder(lightgray, 3));
    mBedroomlight.setBorder(new LineBorder(lightgray, 3));
    mBedroomwindow.setBorder(new LineBorder(lightgray, 3));
    mBedroomsmoke.setBorder(new LineBorder(lightgray, 3));
    mBedroomfan.setBorder(new LineBorder(lightgray, 3));
    mBedroomaircond.setBorder(new LineBorder(lightgray, 3));
}

} // end of BedroomView1 class -----
//----- BedroomView2 -----
class BedroomView2 extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mBedroomdoor;
    private Module mBedroomlight;
    private Module mBedroomwindow;
    private Module mBedroomsmoke;
    private Module mBedroomfan;
    private Module mBedroomaircond;
    private String[] sbd = {"Door", "door", "bedroomdoor2"};
    private String[] sbl = {"Light", "light", "bedroomlight2"};
    private String[] sbw = {"Window", "window", "bedroomwindow2"};
    private String[] sbf = {"Fan", "fan", "bedroomfan2"};
    private String[] sba = {"Air-Conditioner", "aircond", "bedroomaircond2"};
    private String[] sbs = {"Smoke Detector", "smoke", "bedroomsmoke2"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "Bedroom 2";

    public BedroomView2(MainFrame parent, SerialConnection connection, Status
status)
    {
        this.parent = parent;
        this.connection = connection;
        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mBedroomdoor = new Module (connection, status, this.getTitle(sbd),
this.geticonName(sbd), this.getName(sbd));
        mBedroomlight = new Module (connection, status, this.getTitle(sbl),
this.geticonName(sbl), this.getName(sbl));
        mBedroomwindow = new Module (connection, status, this.getTitle(sbw),
this.geticonName(sbw), this.getName(sbw));

```

```

        mBedroomsmoke = new Module (connection, status, this.getTitle(sbs),
this.geticonName(sbs),this.getName(sbs));
        mBedroomfan = new Module (connection, status, this.getTitle(sbf),
this.geticonName(sbf),this.getName(sbf));
        mBedroomaircond = new Module (connection, status, this.getTitle(sba),
this.geticonName(sba),this.getName(sba));

        mBedroomdoor.setPreferredSize(maxSize);
        mBedroomlight.setPreferredSize(maxSize);
        mBedroomwindow.setPreferredSize(maxSize);
        mBedroomsmoke.setPreferredSize(maxSize);
        mBedroomfan.setPreferredSize(maxSize);
        mBedroomaircond.setPreferredSize(maxSize);

        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
        jlRoom = new JLabel(room);
        jlRoom.setFont(new Font("BankGothic Lt BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);
        jpInnerModule = new JPanel();
        jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
        jpInnerModule.setBackground(new Color(200,200,210));

        jpInnerModule.add(mBedroomdoor);
        jpInnerModule.add(mBedroomlight);
        jpInnerModule.add(mBedroomwindow);
        jpInnerModule.add(mBedroomfan);
        jpInnerModule.add(mBedroomaircond);
        jpInnerModule.add(mBedroomsmoke);

        setLayout(new BorderLayout(10,10));
        add(jpRoom, BorderLayout.NORTH);
        add(jpInnerModule, BorderLayout.CENTER);

        setBackground(new Color(200,200,210));
    }

    public String getTitle(String[] module)
    {
        return module[0];
    }

    public String geticonName(String[] module)
    {
        return module[1];
    }

    public String getName(String[] module)
    {
        return module[2];
    }

    public void setModuleClear()
    {
        mBedroomdoor.setBorder(new LineBorder(lightgray, 3));
        mBedroomlight.setBorder(new LineBorder(lightgray, 3));
        mBedroomwindow.setBorder(new LineBorder(lightgray, 3));
        mBedroomsmoke.setBorder(new LineBorder(lightgray, 3));
        mBedroomfan.setBorder(new LineBorder(lightgray, 3));
        mBedroomaircond.setBorder(new LineBorder(lightgray, 3));
    }

} // end of BedroomView2 class -----
//----- BedroomView3 -----

```

```

class BedroomView3 extends JPanel
{
    private MainFrame parent;
    private SerialConnection connection;
    private Status status;

    private Module mBedroomdoor;
    private Module mBedroomlight;
    private Module mBedroomwindow;
    private Module mBedroomsmoke;
    private Module mBedroomfan;
    private Module mBedroomaircond;
    private String[] sbd = {"Door", "door", "bedroomdoor3"};
    private String[] sbl = {"Light", "light", "bedroomlight3"};
    private String[] sbw = {"Window", "window", "bedroomwindow3"};
    private String[] sbf = {"Fan", "fan", "bedroomfan3"};
    private String[] sba = {"Air-Conditioner", "aircond", "bedroomaircond3"};
    private String[] sbs = {"Smoke Detector", "smoke", "bedroomsmoke3"};

    private Dimension maxSize = new Dimension(250,250);
    final int HEIGHT = 700;
    final int WIDTH = 550;

    private JPanel jpRoom;
    private JPanel jpInnerModule;
    private JLabel jlRoom;
    private String room = "Bedroom 3";

    public BedroomView3(MainFrame parent, SerialConnection connection, Status
status)
    {
        this.parent = parent;
        this.connection = connection;
        this.status = status;

        enableEvents(AWTEvent.WINDOW_EVENT_MASK);

        mBedroomdoor = new Module (connection, status, this.getTitle(sbd),
this.geticonName(sbd),this.getName(sbd));
        mBedroomlight = new Module (connection, status, this.getTitle(sbl),
this.geticonName(sbl),this.getName(sbl));
        mBedroomwindow = new Module (connection, status, this.getTitle(sbw),
this.geticonName(sbw),this.getName(sbw));
        mBedroomsmoke = new Module (connection, status, this.getTitle(sbs),
this.geticonName(sbs),this.getName(sbs));
        mBedroomfan = new Module (connection, status, this.getTitle(sbf),
this.geticonName(sbf),this.getName(sbf));
        mBedroomaircond = new Module (connection, status, this.getTitle(sba),
this.geticonName(sba),this.getName(sba));

        mBedroomdoor.setPreferredSize(maxSize);
        mBedroomlight.setPreferredSize(maxSize);
        mBedroomwindow.setPreferredSize(maxSize);
        mBedroomsmoke.setPreferredSize(maxSize);
        mBedroomfan.setPreferredSize(maxSize);
        mBedroomaircond.setPreferredSize(maxSize);

        jpRoom = new JPanel();
        jpRoom.setLayout(new FlowLayout(FlowLayout.CENTER, 5,5));
        jpRoom.setBackground(new Color(200,200,210));
        jpRoom.setBorder(new BevelBorder(BevelBorder.RAISED));
        jlRoom = new JLabel(room);
        jlRoom.setFont(new Font("BankGothic It BT", Font.PLAIN, 15));
        jpRoom.add(jlRoom);
        jpInnerModule = new JPanel();
        jpInnerModule.setLayout(new FlowLayout(FlowLayout.LEFT, 5,5));
        jpInnerModule.setBackground(new Color(200,200,210));

        jpInnerModule.add(mBedroomdoor);
        jpInnerModule.add(mBedroomlight);
        jpInnerModule.add(mBedroomwindow);
    }
}

```

```

        jpInnerModule.add(mBedroomfan);
        jpInnerModule.add(mBedroomaircond);
        jpInnerModule.add(mBedroomsmoke);

        setLayout(new BorderLayout(10,10));
        add(jpRoom, BorderLayout.NORTH);
        add(jpInnerModule, BorderLayout.CENTER);

        setBackground(new Color(200,200,210));
    }

    public String getTitle(String[] module)
    {
        return module[0];
    }

    public String geticonName(String[] module)
    {
        return module[1];
    }

    public String getName(String[] module)
    {
        return module[2];
    }

    public void setModuleClear()
    {
        mBedroomdoor.setBorder(new LineBorder(lightgray, 3));
        mBedroomlight.setBorder(new LineBorder(lightgray, 3));
        mBedroomwindow.setBorder(new LineBorder(lightgray, 3));
        mBedroomsmoke.setBorder(new LineBorder(lightgray, 3));
        mBedroomfan.setBorder(new LineBorder(lightgray, 3));
        mBedroomaircond.setBorder(new LineBorder(lightgray, 3));
    }

} // end of BedroomView3 class -----

//----- TreePanel -----
class TreePanel extends JPanel
{
    private JScrollPane jspTree;
    public JTree jtTree;
    final int WIDTH = 500;
    final int HEIGHT = 500;

    Hashtable htMain;
    Hashtable htBedrooms;
    Object[] objGeneral, objLiving, objDining, objKitchen, objBathroom,
            objBedroom1, objBedroom2, objBedroom3;

    Icon open = new ImageIcon("image/right.jpg");
    Icon close = new ImageIcon("image/down.jpg");
    Icon leaf = new ImageIcon("image/leaf.jpg");

    MainFrame mainframe;
    String general = "[root, General]";
    String living = "[root, Living Room]";
    String dining = "[root, Dining Room]";
    String kitchen = "[root, Kitchen]";
    String bedrooms = "[root, Bedrooms]";
    String bedroom1 = "[root, Bedrooms, Bedroom 1]";
    String bedroom2 = "[root, Bedrooms, Bedroom 2]";
    String bedroom3 = "[root, Bedrooms, Bedroom 3]";
}

```

```

GeneralView mGeneralView;
KitchenView mKitchenView;
LivingView mLivingView;
DiningView mDiningView;
BedroomView1 mBedroomView1;
BedroomView2 mBedroomView2;
BedroomView3 mBedroomView3;

public TreePanel(MainFrame mainframe)
{
    this.mainframe = mainframe;

    htMain = new Hashtable();
    htBedrooms = new Hashtable();
    objGeneral = new Object[] { "Front Door", "Back Door", "Smoke Detector"};
    objLiving = new Object[] {"Window", "Light", "Fan", "Air-Conditioner",
        "Smoke Detector"};
    objDining = new Object[] {"Light", "Fan", "Smoke Detector"};
    objKitchen = new Object[] {"Window", "Light", "Fan", "Smoke Detector"};
    objBathroom = new Object[] {"Light", "Smoke Detector"};
    objBedroom1 = new Object[] {"Door", "Window", "Light", "Fan",
        "Air-Conditioner", "Smoke Detector"};
    objBedroom2 = new Object[] {"Door", "Window", "Light", "Fan",
        "Air-Conditioner", "Smoke Detector"};
    objBedroom3 = new Object[] {"Door", "Window", "Light", "Fan",
        "Air-Conditioner", "Smoke Detector"};

    htBedrooms.put("Bedroom 1", objBedroom1);
    htBedrooms.put("Bedroom 2", objBedroom2);
    htBedrooms.put("Bedroom 3", objBedroom2);

    htMain.put("Kitchen", objKitchen);
    htMain.put("Dining Room", objDining);
    htMain.put("Living Room", objLiving);
    htMain.put("General", objGeneral);
    htMain.put("Bedrooms", htBedrooms);

    DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
    renderer.setClosedIcon(close);
    renderer.setOpenIcon(open);
    renderer.setLeafIcon(leaf);

    jtTree = new JTree(htMain);
    jtTree.setFont(new Font("Verdana", Font.PLAIN, 12));
    jtTree.setCellRenderer(renderer);
    jtTree.setEditable(true);
    jtTree.setBackground(Color.white);

    jspTree = new JScrollPane(jtTree);
    jspTree.setBackground(Color.white);
    jspTree.setBorder(new EmptyBorder(10,10,10,10));
    setLayout(new BorderLayout(10,10));
    add(jspTree, BorderLayout.CENTER);

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

    jtTree.addTreeSelectionListener(
        new TreeSelectionListener(){
            public void valueChanged(TreeSelectionEvent e){
                TreePath path = e.getNewLeadSelectionPath();

                mGeneralView = getGeneralView();
                mGeneralView.setModuleClear();
                mKitchenView = getKitchenView();
                mKitchenView.setModuleClear();
                mLivingView = getLivingView();
                mLivingView.setModuleClear();
                mDiningView = getDiningView();
                mDiningView.setModuleClear();
                mBedroomView1 = getBedroomView1();
                mBedroomView1.setModuleClear();
            }
        }
    );
}

```

```

mBedroomView2 = getBedroomView2();
mBedroomView2.setModuleClear();
mBedroomView3 = getBedroomView3();
mBedroomView3.setModuleClear();

if(path==null){}
else {
    TreePath parentPath = path.getParentPath();

    String parent = parentPath.toString();
    String child = path.toString();
    TreeNode treenode =
(TreeNode)path.getLastPathComponent();
    String node = treenode.toString();

                                if(parent.equals(general) ||
child.equals(general))
                                {
                                    setGeneralView();

                                    if (node.equals("Back Door"))
                                    {
                                        mGeneralView.mBackdoor.setBorder(new
LineBorder(selected, 3));
                                    }

                                    if (node.equals("Front Door"))
                                    {
                                        mGeneralView.mFrontdoor.setBorder(new
LineBorder(selected, 3));
                                    }

                                    if (node.equals("Smoke Detector"))
                                    {
                                        mGeneralView.mLivingsmoke.setBorder(new LineBorder(selected, 3));
                                    }
                                }
                                else if(parent.equals(living) ||
child.equals(living))
                                {
                                    setLivingView();

                                    if (node.equals("Window"))
                                    {
                                        mLivingView.mLivingwindow.setBorder(new LineBorder(selected, 3));
                                    }

                                    if (node.equals("Smoke Detector"))
                                    {
                                        mLivingView.mLivingsmoke.setBorder(new LineBorder(selected, 3));
                                    }

                                    if (node.equals("Light"))
                                    {
                                        mLivingView.mLivinglight.setBorder(new LineBorder(selected, 3));
                                    }

                                    if (node.equals("Fan"))
                                    {
                                        mLivingView.mLivingfan.setBorder(new
LineBorder(selected, 3));
                                    }

                                    if (node.equals("Air-Conditioner"))
                                    {

```

```

mLivingView.mLivingaircond.setBorder(new LineBorder(selected, 3));
    }
}
else if(parent.equals(dining) ||
child.equals(dining))
{
    setDiningView();
    if (node.equals("Smoke Detector"))
    {
mDiningView.mDiningsmoke.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Light"))
    {
mDiningView.mDininglight.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Fan"))
    {
mDiningView.mDiningfan.setBorder(new
LineBorder(selected, 3));
    }
}
else if(parent.equals(kitchen) ||
child.equals(kitchen))
{
    setKitchenView();
    if (node.equals("Window"))
    {
mKitchenView.mKitchenwindow.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Smoke Detector"))
    {
mKitchenView.mKitchensmoke.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Light"))
    {
mKitchenView.mKitchenlight.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Fan"))
    {
mKitchenView.mKitchenfan.setBorder(new LineBorder(selected, 3));
    }
}
else if(child.equals(bedrooms))
{
    setBedroomView1();
}
else if(parent.equals(bedroom1) ||
child.equals(bedroom1))
{
    setBedroomView1();
    if (node.equals("Window"))
    {

```

```

mBedroomView1.mBedroomwindow.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Smoke Detector"))
    {
mBedroomView1.mBedroomsmoke.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Light"))
    {
mBedroomView1.mBedroomlight.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Fan"))
    {
mBedroomView1.mBedroomfan.setBorder(new LineBorder(selected, 3));
    }
    if (node.equals("Air-Conditioner"))
    {
mBedroomView1.mBedroomaircond.setBorder(new LineBorder(selected, 3));
    }
    }
    else if(parent.equals("bedroom2") ||
child.equals("bedroom2"))
    {
        setBedroomView2();
        if (node.equals("Window"))
        {
mBedroomView2.mBedroomwindow.setBorder(new LineBorder(selected, 3));
        }
        if (node.equals("Smoke Detector"))
        {
mBedroomView2.mBedroomsmoke.setBorder(new LineBorder(selected, 3));
        }
        if (node.equals("Light"))
        {
mBedroomView2.mBedroomlight.setBorder(new LineBorder(selected, 3));
        }
        if (node.equals("Fan"))
        {
mBedroomView2.mBedroomfan.setBorder(new LineBorder(selected, 3));
        }
        if (node.equals("Air-Conditioner"))
        {
mBedroomView2.mBedroomaircond.setBorder(new LineBorder(selected, 3));
        }
    }
    else if(parent.equals("bedroom3") ||
child.equals("bedroom3"))
    {
        setBedroomView3();
        if (node.equals("Window"))
        {

```



```

mBedroomView3.mBedroomwindow.setBorder(new LineBorder(selected, 3));
    }

    if (node.equals("Smoke Detector"))
    {

mBedroomView3.mBedroomsmoke.setBorder(new LineBorder(selected, 3));
    }

    if (node.equals("Light"))
    {

mBedroomView3.mBedroomlight.setBorder(new LineBorder(selected, 3));
    }

    if (node.equals("Fan"))
    {

mBedroomView3.mBedroomfan.setBorder(new LineBorder(selected, 3));
    }

    if (node.equals("Air-Conditioner"))
    {

mBedroomView3.mBedroomaircond.setBorder(new LineBorder(selected, 3));
    }
    }
    else
    {
        setGeneralView();
    }
    }
    }
});

}

public JTree getTree()
{
    return this.jtTree;
}

public void setLivingView()
{
    this.mainframe.setLivingView();
}

public void setDiningView()
{
    this.mainframe.setDiningView();
}

public void setKitchenView()
{
    this.mainframe.setKitchenView();
}

public void setGeneralView()
{
    this.mainframe.setGeneralView();
}

public void setBedroomView1()
{
    this.mainframe.setBedroomView1();
}
}

```

```

public void setBedroomView2()
{
    this.mainframe.setBedroomView2();
}

public void setBedroomView3()
{
    this.mainframe.setBedroomView3();
}

public GeneralView getGeneralView()
{
    return this.mainframe.mGeneralView;
}

public KitchenView getKitchenView()
{
    return this.mainframe.mKitchenView;
}

public LivingView getLivingView()
{
    return this.mainframe.mLivingView;
}

public DiningView getDiningView()
{
    return this.mainframe.mDiningView;
}

public BedroomView1 getBedroomView1()
{
    return this.mainframe.mBedroomView1;
}

public BedroomView2 getBedroomView2()
{
    return this.mainframe.mBedroomView2;
}

public BedroomView3 getBedroomView3()
{
    return this.mainframe.mBedroomView3;
}

} // end of TreePanel class -----
//=====End of Child classes in MainFrame=====

} // end of MainFrame class-----
//=====End of MainFrame=====

// Another class to handle thread
class TimerThread extends Thread implements Runnable
{
    UpperPanel myclock;

    public TimerThread(UpperPanel myclock)
    {
        this.myclock = myclock;
    }

    public void run()
    {

```

```
while(true)
{
    try
    {
        this.sleep(1000);
    }
    catch(InterruptedException e){}
    myclock.setTime();
}
}
```

```

//=====
//Author      : Murni Masri
//File Name   : MyAccessPad.java
//=====

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.Image;

class MyAccessPad extends JDialog
{
    final int WIDTH = 549;
    final int HEIGHT = 485;
    MainFrame parent;
    private String pass = "murni";

    public MyAccessPad(MainFrame parent)
    {
        this.parent = parent;
        setTitle("Home Guard System - Access Password");

        Container container = getContentPane();
        container.setLayout(new GridLayout());
        container.add(new ImagePanel(parent, this));

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(screenSize.width/2 - WIDTH/2, screenSize.height/2 - HEIGHT/2);
        setSize(WIDTH, HEIGHT);

        this.validate();
    }
}

class ImagePanel extends JPanel implements ActionListener
{
    private JPasswordField jpfPassword;
    Image image = new ImageIcon("image/intro.jpg").getImage();
    private String pass = "murni";
    Icon icon = new ImageIcon("image/logo_kecikl.jpg");
    MainFrame parent;
    MyAccessPad frame;

    public ImagePanel(MainFrame parent, MyAccessPad frame)
    {
        this.parent = parent;
        this.frame = frame;
        setLayout(new FlowLayout(FlowLayout.LEFT, 200, 285));
        jpfPassword = new JPasswordField(20);
        jpfPassword.addActionListener(this);

        add(jpfPassword);
        setOpaque(true);
    }

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.drawImage(image, 0, 0, this);
        revalidate();
    }

    public void actionPerformed(ActionEvent e)
    {
        String enter = new String(jpfPassword.getPassword());

        if(enter.equals(pass))

```

```

{
    JOptionPane.showMessageDialog (this,
                                   "Welcome to Home Guard system.",
                                   "Information",
                                   JOptionPane.INFORMATION_MESSAGE,
                                   icon);

    this.frame.setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
    parent.setVisible(true);
}

else
{
    JOptionPane.showMessageDialog (this,
                                   "Your password is not valid. Please try again.",
                                   "Information",
                                   JOptionPane.ERROR_MESSAGE);

    this.frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    this.frame.addWindowListener(new WindowAdapter() {
        public void windowClosed(WindowEvent e) {
            System.exit(0);
        }
    });
}
}
}

```

```

//=====
//Author      : Murni Masri
//File Name   : UpperPanel.java
//=====

import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.text.*;
import java.util.*;

public class UpperPanel extends JPanel
{
    SimpleDateFormat formatter;
    Date currentTime;
    public String dateString;
    JLabel jlClock, jlLogo;
    JButton jbIcon, jbIcon2;

    JPanel inner, jpClock, jpIcon, jpLogo;

    public UpperPanel()
    {
        String time = getCurrentTime();
        jlClock = new JLabel();
        jlClock.setText(time);
        jlClock.setDoubleBuffered(true);
        jlClock.setFont(new Font("Verdana", Font.BOLD, 15));
        jlClock.setBorder(new BevelBorder(BevelBorder.LOWERED));

        jpClock = new JPanel();
        jpClock.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        jpClock.add(jlClock);
        jpIcon = new JPanel();
        jpIcon.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));

        jpLogo = new JPanel();
        jlLogo = new JLabel();
        jlLogo.setIcon(new ImageIcon("image/logo_bulat_kecik5.jpg"));

        inner = new JPanel();
        inner.setLayout(new BorderLayout());
        inner.setBorder(new EmptyBorder(5, 5, 5, 5));
        inner.add(jpClock, BorderLayout.EAST);
        inner.add(jpIcon, BorderLayout.CENTER);

        this.setLayout(new BorderLayout(10, 10));
        this.setBorder(new BevelBorder(BevelBorder.LOWERED));
        this.add(jpLogo, BorderLayout.WEST);
        this.add(inner, BorderLayout.CENTER);
    }

    public void setTime()
    {
        String time = getCurrentTime();
        jlClock.setText(time);
    }

    public String getCurrentTime()
    {
        int s = 0, m = 10, h = 10;
        Date currentTime = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("s", Locale.getDefault());

        try
        {
            s = Integer.parseInt(formatter.format(currentTime));
        }
        catch (NumberFormatException n)
        {
        }
    }
}

```

```

    s = 0;
}

formatter.applyPattern("m");
try
{
    m = Integer.parseInt(formatter.format(currentTime));
}
catch (NumberFormatException n)
{
    m = 10;
}

formatter.applyPattern("h");
try
{
    h = Integer.parseInt(formatter.format(currentTime));
}
catch (NumberFormatException n)
{
    h = 10;
}

formatter.applyPattern(" dd/MM/yyyy HH:mm ");
String dateString = formatter.format(currentTime);
return dateString;
}
}

```

```

//=====
//Author      : Murni Masri
//File Name   : SerialParameters.java
//=====

import javax.comm.*;

public class SerialParameters
{
    private String portName;
    private int baudRate;
    private int flowControlIn;
    private int flowControlOut;
    private int databits;
    private int stopbits;
    private int parity;

    public SerialParameters ()
    {
        this("COM1",
            9600,
            SerialPort.FLOWCONTROL_NONE,
            SerialPort.FLOWCONTROL_NONE,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE );
    }

    public SerialParameters(String portName,int baudRate,int flowControlIn,
        int flowControlOut,int databits,int stopbits, int parity)
    {
        this.portName = portName;
        this.baudRate = baudRate;
        this.flowControlIn = flowControlIn;
        this.flowControlOut = flowControlOut;
        this.databits = databits;
        this.stopbits = stopbits;
        this.parity = parity;
    }

    public void setPortName(String portName)
    {
        this.portName = portName;
    }

    public String getPortName()
    {
        return portName;
    }

    public void setBaudRate(int baudRate)
    {
        this.baudRate = baudRate;
    }

    public void setBaudRate(String baudRate)
    {
        this.baudRate = Integer.parseInt(baudRate);
    }

    public int getBaudRate()
    {
        return baudRate;
    }

    public String getBaudRateString()
    {
        return Integer.toString(baudRate);
    }

    public void setFlowControlIn(int flowControlIn)

```



```

    {
        this.flowControlIn = flowControlIn;
    }

    public void setFlowControlIn(String flowControlIn)
    {
        this.flowControlIn = stringToFlow(flowControlIn);
    }

    public int getFlowControlIn()
    {
        return flowControlIn;
    }

    public String getFlowControlInString()
    {
        return flowToString(flowControlIn);
    }

    public void setFlowControlOut(int flowControlOut)
    {
        this.flowControlOut = flowControlOut;
    }

    public void setFlowControlOut(String flowControlOut)
    {
        this.flowControlOut = stringToFlow(flowControlOut);
    }

    public int getFlowControlOut()
    {
        return flowControlOut;
    }

    public String getFlowControlOutString()
    {
        return flowToString(flowControlOut);
    }

    public void setDatabits(int databits)
    {
        this.databits = databits;
    }

    public void setDatabits(String databits)
    {
        if (databits.equals("5"))
        {
            this.databits = SerialPort.DATABITS_5;
        }
        if (databits.equals("6"))
        {
            this.databits = SerialPort.DATABITS_6;
        }
        if (databits.equals("7"))
        {
            this.databits = SerialPort.DATABITS_7;
        }
        if (databits.equals("8"))
        {
            this.databits = SerialPort.DATABITS_8;
        }
    }

    public int getDatabits()
    {
        return databits;
    }

    public String getDatabitsString()
    {

```

```

        switch(databits)
        {
            case SerialPort.DATABITS_5:
                return "5";
            case SerialPort.DATABITS_6:
                return "6";
            case SerialPort.DATABITS_7:
                return "7";
            case SerialPort.DATABITS_8:
                return "8";
            default:
                return "8";
        }
    }

    public void setStopbits(int stopbits)
    {
        this.stopbits = stopbits;
    }

    public void setStopbits(String stopbits)
    {
        if (stopbits.equals("1"))
        {
            this.stopbits = SerialPort.STOPBITS_1;
        }
        if (stopbits.equals("1.5"))
        {
            this.stopbits = SerialPort.STOPBITS_1_5;
        }
        if (stopbits.equals("2"))
        {
            this.stopbits = SerialPort.STOPBITS_2;
        }
    }

    public int getStopbits()
    {
        return stopbits;
    }

    public String getStopbitsString()
    {
        switch(stopbits)
        {
            case SerialPort.STOPBITS_1:
                return "1";
            case SerialPort.STOPBITS_1_5:
                return "1.5";
            case SerialPort.STOPBITS_2:
                return "2";
            default:
                return "1";
        }
    }

    public void setParity(int parity)
    {
        this.parity = parity;
    }

    public void setParity(String parity)
    {
        if (parity.equals("None"))
        {
            this.parity = SerialPort.PARITY_NONE;
        }
        if (parity.equals("Even"))
        {
            this.parity = SerialPort.PARITY_EVEN;
        }
    }

```

```

        if (parity.equals("Odd"))
        {
            this.parity = SerialPort.PARITY_ODD;
        }
    }

    public int getParity()
    {
        return parity;
    }

    public String getParityString()
    {
        switch(parity)
        {
            case SerialPort.PARITY_NONE:
                return "None";
            case SerialPort.PARITY_EVEN:
                return "Even";
            case SerialPort.PARITY_ODD:
                return "Odd";
            default:
                return "None";
        }
    }

    private int stringToFlow(String flowControl)
    {
        if (flowControl.equals("None"))
        {
            return SerialPort.FLOWCONTROL_NONE;
        }
        if (flowControl.equals("Xon/Xoff Out"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_OUT;
        }
        if (flowControl.equals("Xon/Xoff In"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_IN;
        }
        if (flowControl.equals("RTS/CTS In"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowControl.equals("RTS/CTS Out"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }

    String flowToString(int flowControl)
    {
        switch(flowControl)
        {
            case SerialPort.FLOWCONTROL_NONE:
                return "None";
            case SerialPort.FLOWCONTROL_XONXOFF_OUT:
                return "Xon/Xoff Out";
            case SerialPort.FLOWCONTROL_XONXOFF_IN:
                return "Xon/Xoff In";
            case SerialPort.FLOWCONTROL_RTSCTS_IN:
                return "RTS/CTS In";
            case SerialPort.FLOWCONTROL_RTSCTS_OUT:
                return "RTS/CTS Out";
            default:
                return "None";
        }
    }
}

```

```

//=====
//Author      : Murni Masri
//File Name   : PortRequestedDialog.java
//=====

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PortRequestedDialog extends JDialog implements ActionListener
{
    private MainFrame parent;
    Container container;
    private JPanel jpInfo, jpButton;
    private JButton yesButton, noButton;
    private String info = "Your port has been requested by another application. " ;
    private String ask = "Do you want to give up your port?";

    public PortRequestedDialog(MainFrame parent)
    {
        super(parent, "Port Requested!", true);
        this.parent = parent;

        container = getContentPane();

        jpInfo = new JPanel();
        jpInfo.add(new JLabel(info, JLabel.CENTER));
        jpInfo.add(new JLabel(ask, JLabel.CENTER));

        yesButton = new JButton("Yes");
        noButton = new JButton("No");
        jpButton = new JPanel();
        jpButton.add(yesButton);
        jpButton.add(noButton);

        yesButton.addActionListener(this);
        noButton.addActionListener(this);

        container.add(jpInfo, BorderLayout.CENTER);
        container.add(jpButton, BorderLayout.SOUTH);

        int width = 200;
        setSize(400, 150);
        setLocation(parent.getLocationOnScreen().x + 30,
            parent.getLocationOnScreen().y + 30);
        setVisible(true);

        this.pack();
    }

    public void actionPerformed(ActionEvent e)
    {
        String cmd = e.getActionCommand();

        if (cmd.equals("Yes"))
        {
            parent.portClosed();
        }

        setVisible(false);
        dispose();
    }
}

```

```
//=====
//Author      : Murni Masri
//File Name   : SerialConnectionException.java
//=====

public class SerialConnectionException extends Exception
{
    public SerialConnectionException(String str) {
        super(str);
    }

    public SerialConnectionException() {
        super();
    }
}
```