10/10 - Good job on the assignment, the Class Diagram and the impressions writeup.

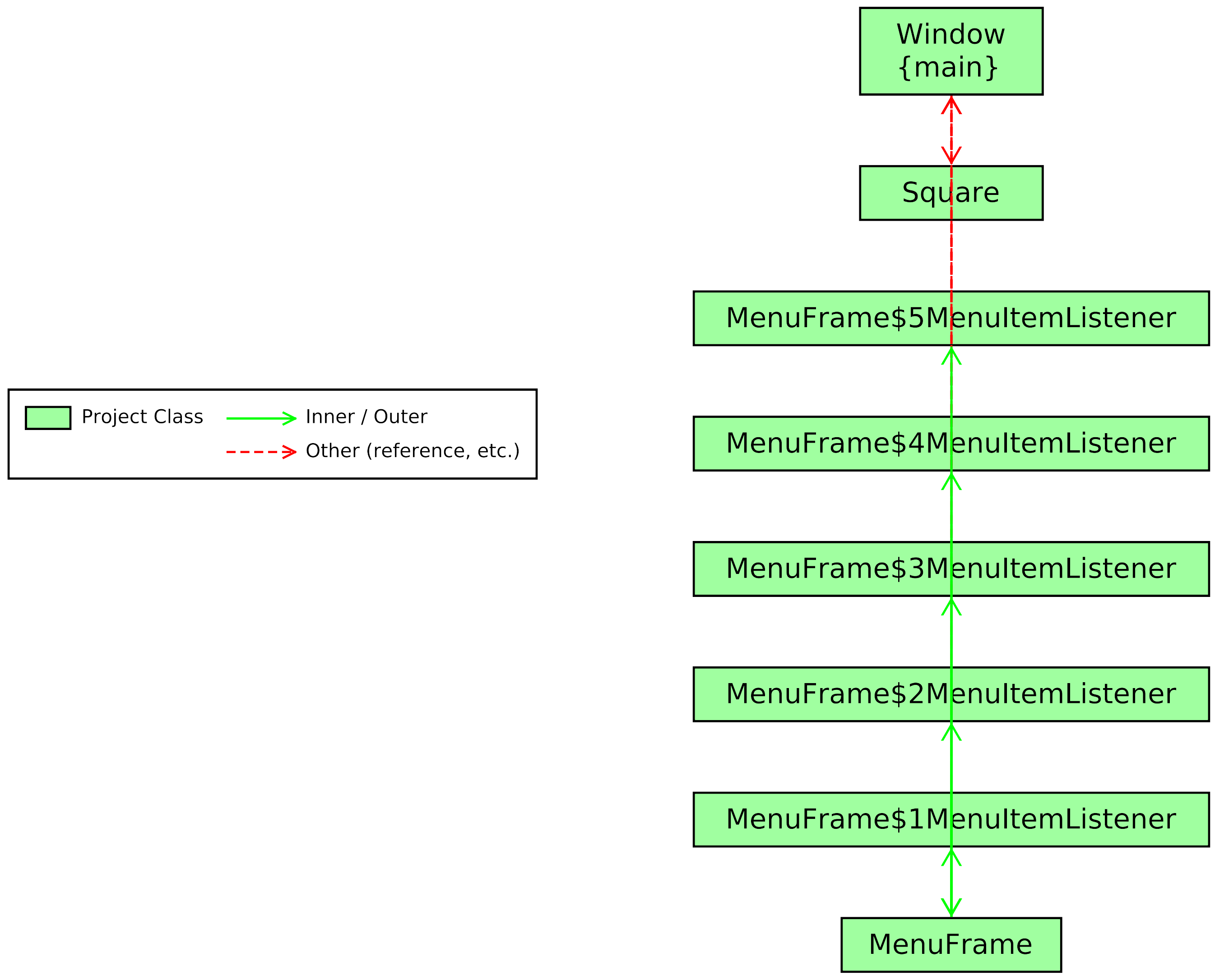
Again, good job David.

David Pick
CM 2403

It seems as though jGRASP was designed to be an IDE that would help programmers understand the code they were working on better. In order to do this it provides tools to generate UML diagrams of the java code, as well as control structure diagrams. The control struture diagrams are designed to improve the readability of code, while I can understand how this tool would be useful to developers. I found that they mostly hindered the readability of the code. I much prefer the systems implemented by many modern IDE's (eclipse, visual studio, etc.) of allowing a user to right-click on a method call or variable and jump to any other place in the code where it is referenced.

While I did not like the control structure diagram that was produced, I did like the UML diagram that was created. Often times when looking through a new project I find myself lost as to which class to start reading through. A tool that generates a UML diagram gives a new developer a clear idea of how the code is connected and where to look in the code for a specific piece of functionality.

While reading through some of the jGRASP documentation I did discover that it has the ability to show the interals of linked lists, trees, and other data structures. I was not able to test this feature, however I have often wished other development tools I have used had this feature, and I believe that once I had it working it would be an invaluable to tool for development.


Window
{main}
Square
MenuFrame$5MenuItemListener
MenuFrame$4MenuItemListener
MenuFrame$3MenuItemListener
MenuFrame$2MenuItemListener
MenuFrame$1MenuItemListener
MenuFrame
Project Class
Inner / Outer
Other (reference, etc.)

```
import java.awt.Color;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;

import java.lang.Math;

public class Square extends JButton implements ActionListener , KeyListener, MouseList
ener {
    /*Some predefined, useful constants for the class*/
    protected Square [][] board;
    protected int difficulity;
    protected int numbombs;
    protected int row;
    protected int col;
    protected int numBombsAround = 0;
    static final int SPECIAL_KEY = 'c';
    protected boolean isBomb = false;
    protected boolean isFlag = false;
    protected boolean isQMark = false;
    protected boolean revealed = false;
    protected ImageIcon bomb;
    protected ImageIcon flag;
    protected ImageIcon Qmark;
    protected Color background = Color.WHITE;
    protected Color nobombsaround = Color.BLUE;
    /**
     * Square class constructor
     * @param row The number of rows the board the square is on will have
     * @param col The number of columns the board the square is on will have
     * @param diff The difficulty of the game
     * @param board The board the square will appear on.
     */
    public Square(int row, int col, int diff, Square [][] board, int numbombs) {
        this.addKeyListener(this);
        this.addMouseListener(this);
        this.bomb = new ImageIcon("bomb.jpg");
        this.flag = new ImageIcon("flag.jpg");
        this.Qmark = new ImageIcon("question_mark.gif");
        this.setBackground(background);
        this.row = row;
        this.col = col;
        this.difficulity = diff;
        this.board = board;
        this.numbombs = numbombs;
    }
    /**
     * When a square is clicked, it reveals itself and the number of bombs it has around it
.
     * If no bombs are around the clicked square, adajcent squares are revealed until they
have
     * bombs near them.
     * @param board the board that the square is on
     */
    public void sweep(Square [][] board) {
        if (this.numBombsAround != 0) {
            if (!this.isFlag) {
                this.setText(this.numBombsAround);
                this.revealed = true;
            }
        }
```

```
         else if (this.revealed == false && this.isBomb == false) {
            this.setBackground(nobombsaround);
            this.revealed = true;
            for (int i = Math.max(0, this.row - 1); i <= Math.min(this.difficulity - 1, th
is.row + 1); i++) {
               for (int j = Math.max(0, this.col - 1); j <= Math.min(this.difficulity - 1,
 this.col + 1); j++) {
                  if (i == this.row && j == this.col){}
                  else {
                     board[i][j].sweep(board);
                  }
               }
            }
         }
      }
   /**
   * When a square is clicked, sets it as a bomb if it is a bomb, or it calculates
   * the number of bombs adjacent to it if it is not a bomb.
   *
   */
      public void setBomb() {
         if (this.isBomb == false){
            this.isBomb = true;
         }

         for (int i = Math.max(0, this.row - 1); i <= Math.min(this.difficulity - 1, this.
row + 1); i++) {
            for (int j = Math.max(0, this.col - 1); j <= Math.min(this.difficulity - 1, th
is.col + 1); j++) {
               if (i == this.row && j == this.col && board[i][j].isBomb == true){}
               else {
                  board[i][j].numBombsAround++;
               }
            }
         }
      }

      public void bombsAround(int bombs) {
         this.numBombsAround = bombs;
      }

   /**
   * Lets the squares listen for a key press of 'c'; used to implement the cheat function
   * found it Window (showbombs).
   */
      public void keyPressed(KeyEvent e) {
         if (e.getKeyChar()== SPECIAL_KEY){
            Window.showbombs(Window.easy, Window.easyboard);
            Window.showbombs(Window.medium, Window.mediumboard);
            Window.showbombs(Window.expert, Window.expertboard);

         }

      }
   /**
   * Lets them hear for a release of 'c', used to implement the cheat function
   * found in Window (showbombs).
   */
      public void keyReleased(KeyEvent e) {
         if (e.getKeyChar()== SPECIAL_KEY){
            Window.hidebombs(Window.easy, Window.easyboard);
            Window.hidebombs(Window.medium, Window.mediumboard);
            Window.hidebombs(Window.expert, Window.expertboard);

         }

      }
   /**
    * Sets the flag variable of each square and displays an icon; used in mouseClicked met
```

```
hod
    * when right-click is detected.
    */
    public void setFlag(){
      if(this.isFlag){
         this.isFlag = false;
         this.revealed = false;
         this.setIcon(null);
      }
      else {
         this.isFlag = true;
         this.revealed = true;
         this.setIcon(this.flag);
      }
    }
   /**
    * Sets a square as a Question mark.
    *
    */
    public void setQmark() {
      if (this.isQMark) {
         this.isQMark = false;
         this.revealed = false;
         this.setIcon(null);
      }
      else {
         this.isQMark = true;
         this.revealed = true;
         this.setIcon(Qmark);
      }
    }
   /**
    * Shows the number of bobs around a square, used as a function
    * to easily change the colors of the numbers shown on the square
    * @param numbombsaround the number of bombs for each square.
    */
    public void setText(int numbombsaround) {
      numbombsaround = this.numBombsAround;
      if (numbombsaround == 1)
         this.setForeground(Color.BLUE);
      else if (numbombsaround == 2)
         this.setForeground(Color.GREEN);
      else if (numbombsaround == 3)
         this.setForeground(Color.RED);
      else if (numbombsaround == 4)
         this.setForeground(new Color(0, 50, 0));
      else if (numbombsaround == 5)
         this.setForeground(Color.PINK);

      this.setText("" + numbombsaround);
    }

   /**
   * Checks for clicks - if a left click on a square, runs the sweep function,
   * or if it's a bomb, reveeals all the bombs,
   * or, if a right-click is detected, runs setFlag to change flag or question mark status
   *
   * Also checks for a win or a lose after each click; a VERY important method.
   */
    public void mouseClicked(MouseEvent e) {
      if (!Window.won) {
         if(e.getButton()!=3 && !this.isFlag){
            if(!this.isBomb && this.numBombsAround != 0){
               this.setText(this.numBombsAround);
            }
            else if(!this.isBomb && this.isQMark) {
               this.setText(this.numBombsAround);
            }
            else if(this.isBomb){
                Window.showbombs(Window.easy, Window.easyboard);
                Window.showbombs(Window.medium, Window.mediumboard);
                Window.showbombs(Window.expert, Window.expertboard);
                Window.won = true;

                String [] you = new String[4];
                String [] lose = new String[4];
                you[0] = "Y";
                you[1] = "o";
                you[2] = "u";
                you[3] = " ";
                lose[0] = "L";
                lose[1] = "o";
                lose[2] = "s";
                lose[3] = "e";

                for (int i = 0; i < this.difficulity; i++) {
                   for (int j = 0; j < this.difficulity; j++) {
                      board[i][j].setText("");
                      board[i][j].setBackground(Color.WHITE);
                   }
                }
                for (int i = 0; i < 4; i++) {
                   this.board[(int) Math.floor(this.difficulity / 2)][i].setText(you[i])
;
                   this.board[(int) Math.floor(this.difficulity / 2)][i].setIcon(null);
                   this.board[(int)Math.floor(this.difficulity / 2)][i + 4].setText(lose
[i]);
                   this.board[(int)Math.floor(this.difficulity / 2)][i + 4].setIcon(null
);
                }
                System.out.println("lose");
             }
             else if (this.numBombsAround == 0)
                this.sweep(board);

             this.revealed = true;
          }

          if (e.getButton() == 3 && !this.isBomb && !this.isFlag && !this.revealed && !t
his.isQMark)
          {
             Window.flagCount++;
             this.setFlag();
          }
          else if (e.getButton()== 3 && !this.isBomb && this.isFlag && this.revealed &&
!this.isQMark){
             Window.flagCount--;
             this.setFlag();
             this.setQmark();
          }
          else if (e.getButton()==3 && this.isBomb && !this.isFlag && !this.revealed &&
!this.isQMark){
             Window.flagCount++;
             Window.toWinCount++;
             this.setFlag();
          }
          else if (e.getButton()==3 && this.isBomb && this.isFlag && this.revealed && !t
his.isQMark){
             Window.flagCount--;
             Window.toWinCount--;
             this.setFlag();
             this.setQmark();
          }
          else if (e.getButton() == 3 && this.revealed && this.isQMark) {
             this.setQmark();
          }

          if (Window.checkwinflags(this.numbombs) || Window.checkwinrevealed(this.numbom
bs, this.board, this.difficulity * this.difficulity)) {
                String [] you = new String[3];
                String [] win = new String[3];
                you[0] = "Y";
                you[1] = "o";
                you[2] = "u";
                win[0] = "W";
                win[1] = "i";
                win[2] = "n";
                for (int i = 0; i < this.difficulity; i++) {
                   for (int j = 0; j < this.difficulity; j++) {
                      board[i][j].setText("");
                      board[i][j].setIcon(null);
                      board[i][j].setBackground(Color.WHITE);
                   }
                }
                for (int i = 0; i < 3; i++) {
                   this.board[(int) Math.floor(this.difficulity / 2)][i].setText(you[i]);
                   this.board[(int)Math.floor(this.difficulity / 2)][i + 4].setText(win[i])
;
                }
                System.out.println("win");
             }
          }

       }

      public void actionPerformed(ActionEvent e) {}

      public void keyTyped(KeyEvent e) {}

      public void mouseEntered(MouseEvent arg0) {}

      public void mouseExited(MouseEvent arg0) {}

      public void mousePressed(MouseEvent e) {}

      public void mouseReleased(MouseEvent arg0) {}

   }
```

```
import java.awt.Color;
import java.awt.GridLayout;


import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class Window {
/* some special constants*/
   static final int SPECIAL_KEY = 'c';

   protected static int flagCount = 0;
   protected static int toWinCount = 0;

   static int EASY_FRAME_WIDTH = 500;
   static int EASY_FRAME_HEIGHT = 500;

   static int MEDIUM_FRAME_WIDTH = 775;
   static int MEDIUM_FRAME_HEIGHT = 775;

   static int EXPERT_FRAME_WIDTH = 1100;
   static int EXPERT_FRAME_HEIGHT = 1100;

   static int easy = 9;
   static int medium = 16;
   static int expert = 25;

   static int easybombs = 9;
   static int mediumbombs = 40;
   static int expertbombs = 99;

   static Square [][] easyboard = new Square[easy][easy];
   static Square [][] mediumboard = new Square[medium][medium];
   static Square [][] expertboard = new Square[expert][expert];

   static JFrame frame_Easy = new MenuFrame();
   static JFrame frame_Medium = new MenuFrame();
   static JFrame frame_Expert = new MenuFrame();

   static ImageIcon bomb = new ImageIcon("bomb.jpg");
   static ImageIcon flag = new ImageIcon("flag.jpg");

   static boolean won = false;

/**
* Populates a 2D array with Squares to reprpesent the board.
*
* @param frame the particular frame that the board will be created in; used for difficu
lty implementation
* @param pieces the number of squares in each row/column of a difficulty board (made sq
uares for simplicity)
* @param frame_width the width of the window frame for the corresponding difficulty
* @param frame_height the height of the window frame for the corresponding difficulty
* @param board the particular board to populate with squares
*/
   public static void createboard(JFrame frame, int pieces, int frame_width, int frame
_height, Square[][] board, int numbombs) {
      frame.setSize(frame_width, frame_height);
      frame.setTitle("Minesweeper");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();

      panel.setFocusable(false);

      for (int i=0; i<pieces; i++) {
         for (int j=0; j<pieces; j++) {
            board[i][j] = new Square(i, j, pieces, board, numbombs);
```

```
                board[i][j].addActionListener(board[i][j]);
                panel.add(board[i][j]);
            }
        }

        panel.setLayout(new GridLayout(pieces, pieces));
        frame.add(panel);
        frame.setVisible(false);
    }

  /**
   *  Randomly populates the Squares in the board with the proper of number of bombs.
   * @param boardSize the row and column size of the board the bombs are set in
   * @param numBombs the number of bombs to set
   * @param board the specific board to set bombs in
   */
    public static void setBombs (int boardSize, int numBombs, Square[][] board){
        boardSize = boardSize - 1;
        Square [] Bombs = new Square [numBombs];
        int setBombs = 1;
        while(setBombs <= numBombs){
            double indexRow = boardSize*Math.random();
            double indexColumn = boardSize*Math.random();
            Square square = board[(int)java.lang.Math.round(indexRow)][(int)java.lang.Math
.round(indexColumn)];
            if(square.isBomb)
                continue;
            square.setBomb();
            setBombs++;
        }
    }
  /**
   *
   * @param boardSize the size of the board to be reset by NewGame
   * @param numBombs the number of bombs the board will be populated with
   * @param board the actual board being reset by NewGame
   */
    public static void newGame (int boardSize ,int numBombs, Square[][] board){
        for (int i = 0; i < boardSize;i++){
            for (int j = 0; j < boardSize; j++){
                board[i][j].isFlag = false;
                board[i][j].isBomb = false;
                Window.flagCount = 0;
                Window.toWinCount = 0;
                board[i][j].numBombsAround = 0;
                board[i][j].setIcon(null);
                board[i][j].setText(null);
                board[i][j].setBackground(Color.WHITE);
                Window.won = false;
                board[i][j].revealed = false;
            }
        }
        Window.setBombs(boardSize, numBombs, board);
    }
  /**
   * Reveals all the bombs on the board; used in cheat function and lose.
   * @param difficulity the rows/columns of the board for array index looping
   * @param board the board where bombs are to be shown.
   */
    public static void showbombs(int difficulity, Square [][] board) {
        for (int i=0; i<difficulity; i++) {
            for (int j=0; j<difficulity; j++) {
                if(board[i][j].isBomb && !board[i][j].isFlag)
                    board[i][j].setIcon(bomb);
                else if (board[i][j].isBomb && board[i][j].isFlag)
                    board[i][j].setIcon(flag);
            }
        }
    }
```

```
/**
 * Hides all the bombs on the board; used in cheat function and lose.
 * @param difficulity the rows/columns of the board for array index looping
 * @param board the board where bombs are to be hid.
 */
 public static void hidebombs(int difficulity, Square [][] board) {
    for (int i=0; i<difficulity; i++) {
       for (int j=0; j<difficulity; j++) {
          if(board[i][j].isBomb && !board[i][j].isFlag)
             board[i][j].setIcon(null);
          else if (board[i][j].isBomb && board[i][j].isFlag)
             board[i][j].setIcon(flag);
       }
    }
 }
/**
 * Checks for a win by seeing if all of the squares that are bombs are covered by flags
 * See Window class for more details
 * @param difficulitybombs the number of bombs that flags has to be equal to
 * @return true or false depending on the condition of the game
 */
 public static boolean checkwinflags(int difficulitybombs) {
    if (difficulitybombs == Window.toWinCount) {
       return true;
    }
    else {
       return false;
    }
 }
/**
 * Checks for a win based on whether or not all squares are revealed except those that
are
 * bombs
 * @param difficulitybombs the number of bombs on the board
 * @param board the board that will be checked
 * @param difficulity the number of squares in the board’s row and column, used to find
 * total squares
 * @return true or false depending on whether or not the number of squares reveealed is

 * equal to the number of squares on the board minus the bombs.
 */
 public static boolean checkwinrevealed(int difficulitybombs, Square [][] board, int
difficulity) {
    int revealedcount = 0;

    for (int i = 0; i < Math.sqrt(difficulity); i++) {
       for (int j = 0; j < Math.sqrt(difficulity); j++) {
          if (board[i][j].revealed && !board[i][j].isBomb && !board[i][j].isFlag && !
board[i][j].isQMark)
             revealedcount++;
       }
    }
    System.out.println(revealedcount);
    System.out.println(difficulity - difficulitybombs);
    if (revealedcount == (difficulity - difficulitybombs)) {
    //       System.out.println("true");

       return true;
    }
    else {
    //System.out.println("false");
       return false;
    }
 }
/**
 * Creates all three boards for difficulties and sets bombs, allowing the game to be pl
ayed.
 * @param args
 */
```

```
   │   ▀ public static void main(String [] args){
   │   ├── createboard(frame_Easy, easy, EASY_FRAME_WIDTH, EASY_FRAME_HEIGHT, easyboard, eas
ybombs);
   │   ├── createboard(frame_Medium, medium, MEDIUM_FRAME_WIDTH, MEDIUM_FRAME_HEIGHT, medium
board, mediumbombs);
   │   ├── createboard(frame_Expert, expert, EXPERT_FRAME_WIDTH, EXPERT_FRAME_HEIGHT, expert
board, expertbombs);
   │   │
   │   ├── setBombs(easy, easybombs, easyboard);
   │   ├── setBombs(medium, mediumbombs, mediumboard);
   │   ├── setBombs(expert, expertbombs, expertboard);
   │   │
   │   ├── frame_Easy.show();
   │   │
   │   │//showbombs(easy, easyboard);
   │   └}
   └}
   import java.awt.BorderLayout;
   import java.awt.Font;
   import java.awt.GridLayout;
   import java.awt.event.ActionEvent;
   import java.awt.event.ActionListener;
   import java.awt.event.KeyEvent;

   import javax.swing.ButtonGroup;
   import javax.swing.JButton;
   import javax.swing.JCheckBox;
   import javax.swing.JComboBox;
   import javax.swing.JFrame;
   import javax.swing.JLabel;
   import javax.swing.JMenu;
   import javax.swing.JMenuBar;
   import javax.swing.JMenuItem;
   import javax.swing.JPanel;
   import javax.swing.JRadioButton;
   import javax.swing.border.EtchedBorder;
   import javax.swing.border.TitledBorder;

/**
   This frame has a menu with commands to change the font
   of a text sample.
*/
    public class MenuFrame extends JFrame
   {
   /**
      Constructs the frame.
   */


      public MenuFrame()
      {

      // Construct menu
         JMenuBar menuBar = new JMenuBar();
         setJMenuBar(menuBar);
      //      menuBar.setLayout(new GridLayout(2, 1));
         menuBar.add(createFileMenu());

         setSize(FRAME_WIDTH, FRAME_HEIGHT);
      }

   /**
      Creates the File menu.
      @return the menu
   */
      public JMenu createFileMenu()
      {
         JMenu menu = new JMenu("File");
         menu.add(createEasyItem());
         menu.add(createMediumItem());
         menu.add(createExpertItem());
         menu.add(createNewGameItem());
         menu.add(createFileExitItem());
         menu.setMnemonic(KeyEvent.VK_F);
         return menu;
      }

   /**
      Creates the File->Exit menu item and sets its action listener.
      @return the menu item
   */
      public JMenuItem createFileExitItem()
      {
         JMenuItem item = new JMenuItem("Exit");
          class MenuItemListener implements ActionListener
         {
          public void actionPerformed(ActionEvent event)
         {
            System.exit(0);
         }
      }
      ActionListener listener = new MenuItemListener();
      item.addActionListener(listener);
      item.setMnemonic(KeyEvent.VK_X);
      return item;
   }

    public JMenuItem createEasyItem()
   {
      JMenuItem item = new JMenuItem("Easy");
       class MenuItemListener implements ActionListener
      {
          public void actionPerformed(ActionEvent event)
         {
            Window.newGame(Window.easy, Window.easybombs, Window.easyboard);
            Window.frame_Easy.show();
            Window.frame_Medium.hide();
            Window.frame_Expert.hide();

         }
      }
      ActionListener listener = new MenuItemListener();
      item.addActionListener(listener);
      item.setMnemonic(KeyEvent.VK_E);
      return item;
   }

    public JMenuItem createMediumItem()
   {
      JMenuItem item = new JMenuItem("Medium");
       class MenuItemListener implements ActionListener
      {
          public void actionPerformed(ActionEvent event)
         {
            Window.newGame(Window.medium, Window.mediumbombs, Window.mediumboard);
            Window.frame_Easy.hide();
            Window.frame_Medium.show();
            Window.frame_Expert.hide();
         }
      }
      ActionListener listener = new MenuItemListener();
      item.addActionListener(listener);
      item.setMnemonic(KeyEvent.VK_M);
      return item;
   }

    public JMenuItem createExpertItem()
   {
      JMenuItem item = new JMenuItem("Expert");
       class MenuItemListener implements ActionListener
      {
          public void actionPerformed(ActionEvent event)
         {
            Window.newGame(Window.expert, Window.expertbombs, Window.expertboard);
            Window.frame_Easy.hide();
            Window.frame_Medium.hide();
            Window.frame_Expert.show();
         }
      }
      ActionListener listener = new MenuItemListener();
      item.addActionListener(listener);
      item.setMnemonic(KeyEvent.VK_T);
      return item;
   }
   public JMenuItem createNewGameItem()
   {
      JMenuItem item = new JMenuItem("New Game");
      class MenuItemListener implements ActionListener
      {
         public void actionPerformed(ActionEvent event)
         {
            Window.newGame(Window.easy, Window.easybombs, Window.easyboard);
            Window.newGame(Window.medium, Window.mediumbombs, Window.mediumboard);
            Window.newGame(Window.expert, Window.expertbombs, Window.expertboard);

         }
      }
      ActionListener listener = new MenuItemListener();
      item.addActionListener(listener);
      item.setMnemonic(KeyEvent.VK_N);
      return item;
   }

   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 400;
}
```