

Stage C: The Program

```

/*=====
Program / Class: Dossier, DigiMonopoly      Author: Ross Chan
Purpose of Class: Main Monopoly class that carries out all functions of the program.
Date of This Revision: March 28, 2007     Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0   Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP   IDE: Eclipse 3.2
=====*/

```

```

import javax.swing.*;

import java.io.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Random;
import java.awt.Color;
import java.awt.FlowLayout;

public class DigiMonopoly
{
    public static void main( String[] args )
    {
        askFrame = new JFrame();
        boardFrame = new JFrame();
        boardFrame.setVisible( false );
        desktop = new JDesktopPane();

        askFrame.getContentPane().add( desktop );

        JMenuBar menuBar = new JMenuBar();
        askFrame.setJMenuBar( menuBar );

        JMenu fileMenu = new JMenu( "File" );
        JMenu helpMenu = new JMenu( "Help" );
        menuBar.add( fileMenu );
        menuBar.add( helpMenu );

        final JMenuItem newGame = new JMenuItem( "New Game" );
        final JMenuItem loadGame = new JMenuItem( "Load Game" );
        final JMenuItem next = new JMenuItem( "Select Number of Players" );
        final JMenuItem userS = new JMenuItem( "Define User Settings" );
        final JMenuItem exit = new JMenuItem( "Exit" );

        next.setEnabled( false ); userS.setEnabled( false );

        fileMenu.add( newGame ); fileMenu.add( loadGame );
        fileMenu.addSeparator(); fileMenu.add( next );
        fileMenu.add( userS ); fileMenu.addSeparator();
    }
}

```

```

fileMenu.add( exit );

final JMenuItem helpGen = new JMenuItem( "Interface Help" );
final JMenuItem helpRule = new JMenuItem( "Rules of the Game" );
final JMenuItem helpFAQ = new JMenuItem( "Frequently Asked Questions" );
final JMenuItem helpAbo = new JMenuItem( "About Digital Monopoly" );

helpMenu.add( helpGen ); helpMenu.addSeparator();
helpMenu.add( helpRule ); helpMenu.add( helpFAQ );
helpMenu.addSeparator(); helpMenu.add( helpAbo );

class AddNewListener implements ActionListener
{
    public void actionPerformed((ActionEvent event)
    {
        next.setEnabled( true );
    }
}

ActionListener listenN = new AddNewListener();
newGame.addActionListener( listenN );

class LoadListener implements ActionListener
{
    public void actionPerformed((ActionEvent event)
    {
        while( loadFile() )
        {
            int number = loadSettings();

            Info.setPlayerNumber( number );
            setStatus( file );
            break;
        }
    }
}

ActionListener listenL = new LoadListener();
loadGame.addActionListener( listenL );

class AddActionListener implements ActionListener
{
    public void actionPerformed((ActionEvent event)
    {
        Info info = new Info();
        info.setVisible( true );
        desktop.add( info );
        userS.setEnabled( true );
    }
}

```

```

ActionListener listen1 = new AddActionListener();
next.addActionListener( listen1 );

class AddUserListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        final UserInfo infi = new UserInfo();
        infi.setVisible( true );
        desktop.add( infi );
    }
}

ActionListener listen2 = new AddUserListener();
userS.addActionListener( listen2 );

askFrame.setSize( 408, 315 );
askFrame.setTitle("Welcome to Digital Monopoly");
askFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
askFrame.setVisible( true );
askFrame.setResizable( false );
askFrame.setLocation( 300, 105 );

topdesk = new JDesktopPane();
boardFrame.getContentPane().add( topdesk );

go = new JButton();
purple1 = new JButton();
chest1 = new JButton();
purple2 = new JButton();
tax1 = new JButton();
rr1 = new JButton();
lblue1 = new JButton();
chance1 = new JButton();
lblue2 = new JButton();
lblue3 = new JButton();
jail = new JButton();
pink1 = new JButton();
utility1 = new JButton();
pink2 = new JButton();
pink3 = new JButton();
rr2 = new JButton();
orange1 = new JButton();
chest2 = new JButton();
orange2 = new JButton();
orange3 = new JButton();
fPark = new JButton();
red1 = new JButton();
chance2 = new JButton();
red2 = new JButton();
red3 = new JButton();

```

```

rr3 = new JButton();
yellow1 = new JButton();
yellow2 = new JButton();
utility2 = new JButton();
yellow3 = new JButton();
toJail = new JButton();
green1 = new JButton();
green2 = new JButton();
chest3 = new JButton();
green3 = new JButton();
rr4 = new JButton();
chance3 = new JButton();
blue1 = new JButton();
tax2 = new JButton();
blue2 = new JButton();
rollDice = new JButton("Roll Dice");

ImageIcon picture = new ImageIcon("Board.png");
JLabel backgroundBoard = new JLabel( picture );

WIDTH = picture.getIconWidth();
HEIGHT = picture.getIconHeight();

backgroundBoard.setBounds( FRAME_WIDTH - ( WIDTH + 7 ),
    0, WIDTH, HEIGHT );
boardFrame.getLayeredPane().add( backgroundBoard,
    new Integer( Integer.MIN_VALUE ) );
topdesk.add( backgroundBoard );

go.setBounds( 359, 503, 80, 80 );
go.setLayout( new FlowLayout() );
topdesk.add( go );
purple1.setBounds( 359, 455, 63, 48 );
purple1.setLayout( new FlowLayout() );
topdesk.add( purple1 );
chest1.setBounds( 359, 408, 80, 48 );
chest1.setLayout( new FlowLayout() );
topdesk.add( chest1 );
purple2.setBounds( 359, 360, 63, 49 );
purple2.setLayout( new FlowLayout() );
topdesk.add( purple2 );
tax1.setBounds( 359, 313, 80, 48 );
tax1.setLayout( new FlowLayout() );
topdesk.add( tax1 );
rr1.setBounds( 359, 267, 80, 47 );
rr1.setLayout( new FlowLayout() );
topdesk.add( rr1 );
lblue1.setBounds( 359, 220, 64, 48 );
lblue1.setLayout( new FlowLayout() );
topdesk.add( lblue1 );
chance1.setBounds( 359, 173, 81, 49 );

```

```
chance1.setLayout( new FlowLayout() );
topdesk.add( chance1 );
lblue2.setBounds( 360, 126, 64, 49 );
lblue2.setLayout( new FlowLayout() );
topdesk.add( lblue2 );
lblue3.setBounds( 360, 80, 64, 48 );
lblue3.setLayout( new FlowLayout() );
topdesk.add( lblue3 );
jail.setBounds( 361, 2, 81, 79 );
jail.setLayout( new FlowLayout() );
topdesk.add( jail );
pink1.setBounds( 440, 2, 48, 62 );
pink1.setLayout( new FlowLayout() );
topdesk.add( pink1 );
utility1.setBounds( 488, 2, 47, 78 );
utility1.setLayout( new FlowLayout() );
topdesk.add( utility1 );
pink2.setBounds( 535, 2, 47, 62 );
pink2.setLayout( new FlowLayout() );
topdesk.add( pink2 );
pink3.setBounds( 581, 2, 48, 62 );
pink3.setLayout( new FlowLayout() );
topdesk.add( pink3 );
rr2.setBounds( 628, 3, 48, 79 );
rr2.setLayout( new FlowLayout() );
topdesk.add( rr2 );
orange1.setBounds( 674, 3, 48, 62 );
orange1.setLayout( new FlowLayout() );
topdesk.add( orange1 );
chest2.setBounds( 720, 3, 48, 79 );
chest2.setLayout( new FlowLayout() );
topdesk.add( chest2 );
orange2.setBounds( 767, 3, 48, 64 );
orange2.setLayout( new FlowLayout() );
topdesk.add( orange2 );
orange3.setBounds( 814, 3, 47, 64 );
orange3.setLayout( new FlowLayout() );
topdesk.add( orange3 );
fPark.setBounds( 859, 5, 80, 80 );
fPark.setLayout( new FlowLayout() );
topdesk.add( fPark );
red1.setBounds( 876, 84, 64, 47 );
red1.setLayout( new FlowLayout() );
topdesk.add( red1 );
chance2.setBounds( 859, 130, 80, 47 );
chance2.setLayout( new FlowLayout() );
topdesk.add( chance2 );
red2.setBounds( 876, 177, 64, 47 );
red2.setLayout( new FlowLayout() );
topdesk.add( red2 );
red3.setBounds( 876, 223, 64, 47 );
```

```
red3.setLayout( new FlowLayout() );
topdesk.add( red3 );
rr3.setBounds( 859, 270, 80, 46 );
rr3.setLayout( new FlowLayout() );
topdesk.add( rr3 );
yellow1.setBounds( 876, 315, 64, 47 );
yellow1.setLayout( new FlowLayout() );
topdesk.add( yellow1 );
yellow2.setBounds( 876, 361, 64, 48 );
yellow2.setLayout( new FlowLayout() );
topdesk.add( yellow2 );
utility2.setBounds( 859, 407, 81, 48 );
utility2.setLayout( new FlowLayout() );
topdesk.add( utility2 );
yellow3.setBounds( 876, 454, 64, 48 );
yellow3.setLayout( new FlowLayout() );
topdesk.add( yellow3 );
toJail.setBounds( 859, 501, 81, 80 );
topdesk.add( toJail );
green1.setBounds( 813, 519, 48, 64 );
green1.setLayout( new FlowLayout() );
topdesk.add( green1 );
green2.setBounds( 767, 519, 48, 64 );
green2.setLayout( new FlowLayout() );
topdesk.add( green2 );
chest3.setBounds( 720, 503, 48, 80 );
chest3.setLayout( new FlowLayout() );
topdesk.add( chest3 );
green3.setBounds( 673, 519, 48, 64 );
green3.setLayout( new FlowLayout() );
topdesk.add( green3 );
rr4.setBounds( 626, 502, 48, 80 );
rr4.setLayout( new FlowLayout() );
topdesk.add( rr4 );
chance3.setBounds( 579, 502, 48, 80 );
chance3.setLayout( new FlowLayout() );
topdesk.add( chance3 );
blue1.setBounds( 532, 520, 48, 64 );
blue1.setLayout( new FlowLayout() );
topdesk.add( blue1 );
tax2.setBounds( 485, 502, 48, 81 );
tax2.setLayout( new FlowLayout() );
topdesk.add( tax2 );
blue2.setBounds( 438, 520, 48, 64 );
blue2.setLayout( new FlowLayout() );
topdesk.add( blue2 );
rollDice.setBounds( 740, 460, 95, 25 );
topdesk.add( rollDice );

go.setContentAreaFilled( false );
purple1.setContentAreaFilled( false );
```

```

chest1.setContentAreaFilled( false );
purple2.setContentAreaFilled( false );
tax1.setContentAreaFilled( false );
rr1.setContentAreaFilled( false );
lblue1.setContentAreaFilled( false );
chance1.setContentAreaFilled( false );
lblue2.setContentAreaFilled( false );
lblue3.setContentAreaFilled( false );
jail.setContentAreaFilled( false );
pink1.setContentAreaFilled( false );
utility1.setContentAreaFilled( false );
pink2.setContentAreaFilled( false );
pink3.setContentAreaFilled( false );
rr2.setContentAreaFilled( false );
orange1.setContentAreaFilled( false );
chest2.setContentAreaFilled( false );
orange2.setContentAreaFilled( false );
orange3.setContentAreaFilled( false );
fPark.setContentAreaFilled( false );
red1.setContentAreaFilled( false );
chance2.setContentAreaFilled( false );
red2.setContentAreaFilled( false );
red3.setContentAreaFilled( false );
rr3.setContentAreaFilled( false );
yellow1.setContentAreaFilled( false );
yellow2.setContentAreaFilled( false );
utility2.setContentAreaFilled( false );
yellow3.setContentAreaFilled( false );
toJail.setContentAreaFilled( false );
green1.setContentAreaFilled( false );
green2.setContentAreaFilled( false );
chest3.setContentAreaFilled( false );
green3.setContentAreaFilled( false );
rr4.setContentAreaFilled( false );
chance3.setContentAreaFilled( false );
blue1.setContentAreaFilled( false );
tax2.setContentAreaFilled( false );
blue2.setContentAreaFilled( false );

JMenuBar menu = new JMenuBar();
boardFrame.setJMenuBar( menu );

JMenu file = new JMenu( "File" );
JMenu help = new JMenu( "Help" );
menu.add( file ); menu.add( help );

final JMenuItem saveGame = new JMenuItem( "Save Game" );
final JMenuItem quitGame = new JMenuItem( "Quit Game" );

file.add( saveGame ); file.addSeparator();
file.add( quitGame );

```

```
final JMenuItem helpG = new JMenuItem( "Interface Help" );
final JMenuItem helpR = new JMenuItem( "Rules of the Game" );
final JMenuItem helpF = new JMenuItem( "Frequently Asked Questions" );
final JMenuItem helpA = new JMenuItem( "About Digital Monopoly" );
```

```
help.add( helpG ); help.addSeparator();
help.add( helpR ); help.add( helpF );
help.addSeparator(); help.add( helpA );
```

```
purple1.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 0 );
    }
} );
```

```
purple2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 1 );
    }
} );
```

```
lblue1.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 3 );
    }
} );
```

```
lblue2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 4 );
    }
} );
```

```
lblue3.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 5 );
    }
} );
```

```
pink1.addActionListener( new ActionListener()
{
```

```
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 6 );
    }
} );
```

```
pink2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 8 );
    }
} );
```

```
pink3.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 9 );
    }
} );
```

```
orange1.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 11 );
    }
} );
```

```
orange2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 12 );
    }
} );
```

```
orange3.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 13 );
    }
} );
```

```
red1.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 14 );
    }
} );
```

```
    }  
  } );  
  
red2.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 15 );  
    }  
} );  
  
red3.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 16 );  
    }  
} );  
  
yellow1.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 18 );  
    }  
} );  
  
yellow2.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 19 );  
    }  
} );  
  
yellow3.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 21 );  
    }  
} );  
  
green1.addActionListener( new ActionListener()  
{  
    public void actionPerformed( ActionEvent e )  
    {  
        propertyFrame( 22 );  
    }  
} );
```

```
green2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 23 );
    }
} );
```

```
green3.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 24 );
    }
} );
```

```
blue1.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 26 );
    }
} );
```

```
blue2.addActionListener( new ActionListener()
{
    public void actionPerformed((ActionEvent e)
    {
        propertyFrame( 27 );
    }
} );
```

```
class QuitGameListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        closeFile();
        System.exit( 0 );
    }
}
```

```
ActionListener listenQ = new QuitGameListener();
exit.addActionListener( listenQ );
quitGame.addActionListener( listenQ );
```

```
class RollDiceListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        String name = rollDice.getText();
```

```

if( name == "Roll Dice" || name == "Roll Again" )
{
    rollDice();

    if( doubles == 0 )
        rollDice.setText( "Done" );
    else
    {
        rollDice.setText( "Roll Again" );
        roller.setText( "" );
    }
}

else if ( name == "Done" )
{
    int nextPlayer = player + 1;

    if( nextPlayer == Info.playerNumber() )
        nextPlayer = 0;

    rollDice.setText( "Roll Dice" );
    roller.setText( "" );
    playerTurn.setText( "It is "
        + playerPlace[ nextPlayer ].getText() + "'s turn" );

    if( player + 1 == Info.playerNumber() )
        player = -1;
    player++;
}
}
}

```

```

ActionListener roller = new RollDiceListener();
rollDice.addActionListener( roller );

```

```

class AddSaveListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        playerM.saveFile();
    }
}

```

```

ActionListener tester = new AddSaveListener();
saveGame.addActionListener( tester );

```

```

boardFrame.setSize( FRAME_WIDTH, FRAME_HEIGHT );
boardFrame.setTitle("Digital Monopoly");
boardFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
boardFrame.setResizable( false );
boardFrame.setLocation( 40, 45 );

```

```

}

/**
 * Closes the files and terminates the application
 * @exception IOException
 * @pre The files must be open first
 * @post The files are closed
 */
public static void closeFile()
{
    try
    {
        if( file != null )
            file.close();

        System.exit( 0 );
    }

    catch( IOException ioException )
    {
        JOptionPane.showMessageDialog( null, "The File Cannot be Closed", "Error",
            JOptionPane.ERROR_MESSAGE );
        System.exit( 1 );
    }
}

private static void rollDice()
{
    int choice = 0;

    if( playerPos[ player ].getJail() == true )
    {
        if( playerPos[ player ].getinJailCount() < 3 )
            choice = JOptionPane.showConfirmDialog( null,
                "Click yes to pay $50, no to attempt to roll doubles",
                "You're in jail", JOptionPane.YES_NO_OPTION );

        if( choice == 0 || playerPos[ player ].getinJailCount() == 3 )
        {
            playerPos[ player ].setMoney( playerPos[ player ].getMoney() - 50 );
            playerM.setCash( player, playerPos[ player ].getMoney(), file );

            playerPos[ player ].inJail( false );
            playerPos[ player ].inJailCount( 0 );

            JOptionPane.showMessageDialog( null,
                "You paid $50 and are now oout of jail!", "Out of Jail",
                JOptionPane.INFORMATION_MESSAGE );
        }
    }
}

```

```

Random random = new Random();
int diceRoll1, diceRoll2, currentPlace, newPlace;
diceRoll1 = random.nextInt(6) + 1;
diceRoll2 = random.nextInt(6) + 1;
currentPlace = playerPos[ player ].getPosition();
currentdiceRoll = diceRoll1 + diceRoll2;
newPlace = currentPlace + currentdiceRoll;

roller = new JLabel();
roller.setText( "You rolled a " + diceRoll1 + " and a " + diceRoll2 );
roller.setBounds( 725, 430, 120, 25 );
topdesk.add( roller );

if( playerPos[ player ].getJail() == true && diceRoll1 != diceRoll2 )
{
    playerPos[ player ].inJailCount( playerPos[ player ]
        .getinJailCount() + 1 );

    newPlace = 10;
}

if ( diceRoll1 == diceRoll2 )
{
    doubles++;
    JOptionPane.showMessageDialog( null, "You Rolled Doubles!",
        "Doubles!", JOptionPane.INFORMATION_MESSAGE );

    if( playerPos[ player ].getJail() == true )
    {
        playerPos[ player ].inJail( false );
        playerPos[ player ].inJailCount( 0 );

        JOptionPane.showMessageDialog( null, "You got out of jail!",
            "Doubles!", JOptionPane.INFORMATION_MESSAGE );
    }
}

if ( doubles == 3 )
{
    JOptionPane.showMessageDialog( null,
        "You have rolled 3 consecutive doubles. Go to JAIL!",
        "JAILED!", JOptionPane.INFORMATION_MESSAGE );

    newPlace = 10;
    doubles = 0;
}

if( diceRoll1 != diceRoll2 )
    doubles = 0;

if ( newPlace > 39 ) //if end of board, reset + pass Go handling

```

```

{
    newPlace = newPlace - 40;
    playerPos[ player ].setPosition( newPlace );

    JOptionPane.showMessageDialog( null,
        "The Bank pays you a dividend of $200",
        "You Passed Go!", JOptionPane.INFORMATION_MESSAGE );

    playerPos[ player ].setMoney( playerPos[ player ].getMoney() + 200 );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
}

if( newPlace == 30 ) //Go to jail handling
{
    JOptionPane.showMessageDialog( null, "Go Directly to Jail", "Go to Jail",
        JOptionPane.INFORMATION_MESSAGE );

    newPlace = 10; //Set position to jail
    playerPos[ player ].setPosition( newPlace );
    playerPos[ player ].inJail( true );
}

removeLabel( currentPlace ); //remove player label
currentPlace = newPlace;
landed( currentPlace );

playerPos[ player ].setPosition( currentPlace );
playerPlace[ player ].setVisible( true );
}

private static void landed( int place )
{
    switch( place )
    {
        case 0:
            go.add( playerPlace[ player ] ); break;
        case 1:
            purple1.add( playerPlace[ player ] );
            if( bank[ 0 ] == 1 )
                buyIt( 0 );
            else
                payIt( 0 );
            break;
        case 2:
            chest1.add( playerPlace[ player ] ); break;
        case 3:
            purple2.add( playerPlace[ player ] );
            if( bank[ 1 ] == 1 )
                buyIt( 1 );
            else
                payIt( 1 );
    }
}

```

```

    break;
case 4:
    tax1.add( playerPlace[ player ] ); break;
case 5:
    rr1.add( playerPlace[ player ] );
    if( bank[ 2 ] == 1 )
        buyIt( 2 );
    else
        payIt( 2 );
    break;
case 6:
    lblue1.add( playerPlace[ player ] );
    if( bank[ 3 ] == 1 )
        buyIt( 3 );
    else
        payIt( 3 );
    break;
case 7:
    chance1.add( playerPlace[ player ] ); break;
case 8:
    lblue2.add( playerPlace[ player ] );
    if( bank[ 4 ] == 1 )
        buyIt( 4 );
    else
        payIt( 4 );
    break;
case 9:
    lblue3.add( playerPlace[ player ] );
    if( bank[ 5 ] == 1 )
        buyIt( 5 );
    else
        payIt( 5 );
    break;
case 10:
    jail.add( playerPlace[ player ] ); break;
case 11:
    pink1.add( playerPlace[ player ] );
    if( bank[ 6 ] == 1 )
        buyIt( 6 );
    else
        payIt( 6 );
    break;
case 12:
    utility1.add( playerPlace[ player ] );
    if( bank[ 7 ] == 1 )
        buyIt( 7 );
    else
        payIt( 7 );
    break;
case 13:
    pink2.add( playerPlace[ player ] );

```

```

    if( bank[ 8 ] == 1 )
        buyIt( 8 );
    else
        payIt( 8 );
    break;
case 14:
    pink3.add( playerPlace[ player ] );
    if( bank[ 9 ] == 1 )
        buyIt( 9 );
    else
        payIt( 9 );
    break;
case 15:
    rr2.add( playerPlace[ player ] );
    if( bank[ 10 ] == 1 )
        buyIt( 10 );
    else
        payIt( 10 );
    break;
case 16:
    orange1.add( playerPlace[ player ] );
    if( bank[ 11 ] == 1 )
        buyIt( 11 );
    else
        payIt( 11 );
    break;
case 17:
    chest2.add( playerPlace[ player ] ); break;
case 18:
    orange2.add( playerPlace[ player ] );
    if( bank[ 12 ] == 1 )
        buyIt( 12 );
    else
        payIt( 12 );
    break;
case 19:
    orange3.add( playerPlace[ player ] );
    if( bank[ 13 ] == 1 )
        buyIt( 13 );
    else
        payIt( 13 );
    break;
case 20:
    fPark.add( playerPlace[ player ] ); break;
case 21:
    red1.add( playerPlace[ player ] );
    if( bank[ 14 ] == 1 )
        buyIt( 14 );
    else
        payIt( 14 );
    break;

```

```

case 22:
    chance2.add( playerPlace[ player ] ); break;
case 23:
    red2.add( playerPlace[ player ] );
    if( bank[ 15 ] == 1 )
        buyIt( 15 );
    else
        payIt( 15 );
    break;
case 24:
    red3.add( playerPlace[ player ] );
    if( bank[ 16 ] == 1 )
        buyIt( 16 );
    else
        payIt( 16 );
    break;
case 25:
    rr3.add( playerPlace[ player ] );
    if( bank[ 17 ] == 1 )
        buyIt( 17 );
    else
        payIt( 17 );
    break;
case 26:
    yellow1.add( playerPlace[ player ] );
    if( bank[ 18 ] == 1 )
        buyIt( 18 );
    else
        payIt( 18 );
    break;
case 27:
    yellow2.add( playerPlace[ player ] );
    if( bank[ 19 ] == 1 )
        buyIt( 19 );
    else
        payIt( 19 );
    break;
case 28:
    utility2.add( playerPlace[ player ] );
    if( bank[ 20 ] == 1 )
        buyIt( 20 );
    else
        payIt( 20 );
    break;
case 29:
    yellow3.add( playerPlace[ player ] );
    if( bank[ 21 ] == 1 )
        buyIt( 21 );
    else
        payIt( 21 );
    break;

```

```

case 30:
    jail.add( playerPlace[ player ] ); break;
case 31:
    green1.add( playerPlace[ player ] );
    if( bank[ 22 ] == 1 )
        buyIt( 22 );
    else
        payIt( 22 );
    break;
case 32:
    green2.add( playerPlace[ player ] );
    if( bank[ 23 ] == 1 )
        buyIt( 23 );
    else
        payIt( 23 );
    break;
case 33:
    chest3.add( playerPlace[ player ] ); break;
case 34:
    green3.add( playerPlace[ player ] );
    if( bank[ 24 ] == 1 )
        buyIt( 24 );
    else
        payIt( 24 );
    break;
case 35:
    rr4.add( playerPlace[ player ] );
    if( bank[ 25 ] == 1 )
        buyIt( 25 );
    else
        payIt( 25 );
    break;
case 36:
    chance3.add( playerPlace[ player ] ); break;
case 37:
    blue1.add( playerPlace[ player ] );
    if( bank[ 26 ] == 1 )
        buyIt( 26 );
    else
        payIt( 26 );
    break;
case 38:
    tax2.add( playerPlace[ player ] ); break;
case 39:
    blue2.add( playerPlace[ player ] );
    if( bank[ 27 ] == 1 )
        buyIt( 27 );
    else
        payIt( 27 );
    break;
}

```

```

}

private static void removeLabel( int pos )
{
    switch( pos )
    {
        case 0:
            playerPlace[ player ].setVisible( false );
            go.remove( playerPlace[ player ] );
            break;
        case 1:
            playerPlace[ player ].setVisible( false );
            purple1.remove( playerPlace[ player ] );
            break;
        case 2:
            playerPlace[ player ].setVisible( false );
            chest1.remove( playerPlace[ player ] );
            break;
        case 3:
            playerPlace[ player ].setVisible( false );
            purple2.remove( playerPlace[ player ] );
            break;
        case 4:
            playerPlace[ player ].setVisible( false );
            tax1.remove( playerPlace[ player ] );
            break;
        case 5:
            playerPlace[ player ].setVisible( false );
            rr1.remove( playerPlace[ player ] );
            break;
        case 6:
            playerPlace[ player ].setVisible( false );
            lblue1.remove( playerPlace[ player ] );
            break;
        case 7:
            playerPlace[ player ].setVisible( false );
            chance1.remove( playerPlace[ player ] );
            break;
        case 8:
            playerPlace[ player ].setVisible( false );
            lblue2.remove( playerPlace[ player ] );
            break;
        case 9:
            playerPlace[ player ].setVisible( false );
            lblue3.remove( playerPlace[ player ] );
            break;
        case 10:
            playerPlace[ player ].setVisible( false );
            jail.remove( playerPlace[ player ] );
            break;
        case 11:

```

```

    playerPlace[ player ].setVisible( false );
    pink1.remove( playerPlace[ player ] );
    break;
case 12:
    playerPlace[ player ].setVisible( false );
    utility1.remove( playerPlace[ player ] );
    break;
case 13:
    playerPlace[ player ].setVisible( false );
    pink2.remove( playerPlace[ player ] );
    break;
case 14:
    playerPlace[ player ].setVisible( false );
    pink3.remove( playerPlace[ player ] );
    break;
case 15:
    playerPlace[ player ].setVisible( false );
    rr2.remove( playerPlace[ player ] );
    break;
case 16:
    playerPlace[ player ].setVisible( false );
    orange1.remove( playerPlace[ player ] );
    break;
case 17:
    playerPlace[ player ].setVisible( false );
    chest2.remove( playerPlace[ player ] );
    break;
case 18:
    playerPlace[ player ].setVisible( false );
    orange2.remove( playerPlace[ player ] );
    break;
case 19:
    playerPlace[ player ].setVisible( false );
    orange3.remove( playerPlace[ player ] );
    break;
case 20:
    playerPlace[ player ].setVisible( false );
    fPark.remove( playerPlace[ player ] );
    break;
case 21:
    playerPlace[ player ].setVisible( false );
    red1.remove( playerPlace[ player ] );
    break;
case 22:
    playerPlace[ player ].setVisible( false );
    chance2.remove( playerPlace[ player ] );
    break;
case 23:
    playerPlace[ player ].setVisible( false );
    red2.remove( playerPlace[ player ] );
    break;

```

```
case 24:
    playerPlace[ player ].setVisible( false );
    red3.remove( playerPlace[ player ] );
    break;
case 25:
    playerPlace[ player ].setVisible( false );
    rr3.remove( playerPlace[ player ] );
    break;
case 26:
    playerPlace[ player ].setVisible( false );
    yellow1.remove( playerPlace[ player ] );
    break;
case 27:
    playerPlace[ player ].setVisible( false );
    yellow2.remove( playerPlace[ player ] );
    break;
case 28:
    playerPlace[ player ].setVisible( false );
    utility2.remove( playerPlace[ player ] );
    break;
case 29:
    playerPlace[ player ].setVisible( false );
    yellow3.remove( playerPlace[ player ] );
    break;
case 30:
    playerPlace[ player ].setVisible( false );
    toJail.remove( playerPlace[ player ] );
    break;
case 31:
    playerPlace[ player ].setVisible( false );
    green1.remove( playerPlace[ player ] );
    break;
case 32:
    playerPlace[ player ].setVisible( false );
    green2.remove( playerPlace[ player ] );
    break;
case 33:
    playerPlace[ player ].setVisible( false );
    chest3.remove( playerPlace[ player ] );
    break;
case 34:
    playerPlace[ player ].setVisible( false );
    green3.remove( playerPlace[ player ] );
    break;
case 35:
    playerPlace[ player ].setVisible( false );
    rr4.remove( playerPlace[ player ] );
    break;
case 36:
    playerPlace[ player ].setVisible( false );
    chance3.remove( playerPlace[ player ] );
```

```

        break;
    case 37:
        playerPlace[ player ].setVisible( false );
        blue1.remove( playerPlace[ player ] );
        break;
    case 38:
        playerPlace[ player ].setVisible( false );
        tax2.remove( playerPlace[ player ] );
        break;
    case 39:
        playerPlace[ player ].setVisible( false );
        blue2.remove( playerPlace[ player ] );
        break;
    }
}

private static boolean loadFile()
{
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
    int pick = fileChooser.showOpenDialog( null );
    File fileName = fileChooser.getSelectedFile();

    if( pick == JFileChooser.CANCEL_OPTION )
        return false;

    if( fileName == null || fileName.getName().equals( "" ) )
    {
        JOptionPane.showMessageDialog( null, "Invalid File Name",
            "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
        return false;
    }

    else
    {
        try
        {
            file = new RandomAccessFile( fileName, "r" );
        }

        catch( IOException iox )
        {
            JOptionPane.showMessageDialog( null, "Cannot Load Settings",
                "Invalid File Name", JOptionPane.ERROR_MESSAGE );
        }

        return true;
    }
}

public static void setStatus( RandomAccessFile file )

```

```

{
    playerM = new PlayerMenu( Info.playerNumber() );
    topdesk.add( playerM );

    playerPos = new PlayerStatus[ Info.playerNumber() ];

    playerPlace = new JLabel[ Info.playerNumber() ];
    String[] playerName = new String[ Info.playerNumber() ];
    int[] playerCash = new int[ Info.playerNumber() ];
    int[] playerColour = new int[ Info.playerNumber() ];

    playerM.readFile( file );

    playerName = playerM.getPlayerName();
    playerCash = playerM.getPlayerCash();
    playerColour = playerM.getPlayerColour();

    //confirmation that game has just started
    if( playerCash[ 0 ] == 1500 && playerCash[ 1 ] == 1500 )
    {
        for( int s = 0; s < Info.playerNumber(); s++ )
        {
            playerPos[ s ] = new PlayerStatus( 0, 1500, playerName[ s ] );
        }
    }
    else
    {
        switch( Info.playerNumber() )
        {
            case 2:
            {
                playerPos[ 0 ] = new PlayerStatus( 0, playerCash[ 0 ], playerName[ 0 ] );
                playerM.setCash( 0, playerCash[ 0 ], file );
                playerPos[ 1 ] = new PlayerStatus( 0, playerCash[ 1 ], playerName[ 1 ] );
                playerM.setCash( 1, playerCash[ 1 ], file );
                break;
            }
            case 3:
            {
                playerPos[ 0 ] = new PlayerStatus( 0, playerCash[ 0 ], playerName[ 0 ] );
                playerM.setCash( 0, playerCash[ 0 ], file );
                playerPos[ 1 ] = new PlayerStatus( 0, playerCash[ 1 ], playerName[ 1 ] );
                playerM.setCash( 1, playerCash[ 1 ], file );
                playerPos[ 2 ] = new PlayerStatus( 0, playerCash[ 2 ], playerName[ 2 ] );
                playerM.setCash( 2, playerCash[ 2 ], file );
                break;
            }
            case 4:
            {
                playerPos[ 0 ] = new PlayerStatus( 0, playerCash[ 0 ], playerName[ 0 ] );
                playerM.setCash( 0, playerCash[ 0 ], file );
            }
        }
    }
}

```

```

    playerPos[ 1 ] = new PlayerStatus( 0, playerCash[ 1 ], playerName[ 1 ] );
    playerM.setCash( 1, playerCash[ 1 ], file );
    playerPos[ 2 ] = new PlayerStatus( 0, playerCash[ 2 ], playerName[ 2 ] );
    playerM.setCash( 2, playerCash[ 2 ], file );
    playerPos[ 3 ] = new PlayerStatus( 0, playerCash[ 3 ], playerName[ 3 ] );
    playerM.setCash( 3, playerCash[ 3 ], file );
    break;
}
case 5:
{
    playerPos[ 0 ] = new PlayerStatus( 0, playerCash[ 0 ], playerName[ 0 ] );
    playerM.setCash( 0, playerCash[ 0 ], file );
    playerPos[ 1 ] = new PlayerStatus( 0, playerCash[ 1 ], playerName[ 1 ] );
    playerM.setCash( 1, playerCash[ 1 ], file );
    playerPos[ 2 ] = new PlayerStatus( 0, playerCash[ 2 ], playerName[ 2 ] );
    playerM.setCash( 2, playerCash[ 2 ], file );
    playerPos[ 3 ] = new PlayerStatus( 0, playerCash[ 3 ], playerName[ 3 ] );
    playerM.setCash( 3, playerCash[ 3 ], file );
    playerPos[ 4 ] = new PlayerStatus( 0, playerCash[ 4 ], playerName[ 4 ] );
    playerM.setCash( 4, playerCash[ 4 ], file );
    break;
}
case 6:
{
    playerPos[ 0 ] = new PlayerStatus( 0, playerCash[ 0 ], playerName[ 0 ] );
    playerM.setCash( 0, playerCash[ 0 ], file );
    playerPos[ 1 ] = new PlayerStatus( 0, playerCash[ 1 ], playerName[ 1 ] );
    playerM.setCash( 1, playerCash[ 1 ], file );
    playerPos[ 2 ] = new PlayerStatus( 0, playerCash[ 2 ], playerName[ 2 ] );
    playerM.setCash( 2, playerCash[ 2 ], file );
    playerPos[ 3 ] = new PlayerStatus( 0, playerCash[ 3 ], playerName[ 3 ] );
    playerM.setCash( 3, playerCash[ 3 ], file );
    playerPos[ 4 ] = new PlayerStatus( 0, playerCash[ 4 ], playerName[ 4 ] );
    playerM.setCash( 4, playerCash[ 4 ], file );
    playerPos[ 5 ] = new PlayerStatus( 0, playerCash[ 5 ], playerName[ 5 ] );
    playerM.setCash( 5, playerCash[ 5 ], file );
}
}
}

bank = new int[ 28 ];
for( int a = 0; a < 28; a++ )
{
    bank[ a ] = 1;
}

first = new int[ 28 ];
for( int a = 0; a < 28; a++ )
{
    first[ a ] = 0;
}

```

```

second = new int[ 28 ];
for( int a = 0; a < 28; a++ )
{
    second[ a ] = 0;
}

switch( Info.playerNumber() )
{
case 3:
    third = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        third[ a ] = 0;
    } break;
case 4:
    third = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        third[ a ] = 0;
    }
    fourth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        fourth[ a ] = 0;
    } break;
case 5:
    third = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        third[ a ] = 0;
    }
    fourth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        fourth[ a ] = 0;
    }
    fifth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        fifth[ a ] = 0;
    } break;
case 6:
    third = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        third[ a ] = 0;
    }
    fourth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        fourth[ a ] = 0;
    }
}

```

```

    }
    fifth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        fifth[ a ] = 0;
    }
    sixth = new int[ 28 ];
    for( int a = 0; a < 28; a++ )
    {
        sixth[ a ] = 0;
    }
}

for( int i = 0; i < Info.playerNumber(); i++ )
{
    playerPlace[ i ] = new JLabel( playerPos[ i ].getName() );
    playerPlace[ i ].setOpaque( true );

    if( playerColour[ i ] == 1 )
        playerPlace[ i ].setForeground( Color.MAGENTA );
    else if( playerColour[ i ] == 2 )
        playerPlace[ i ].setForeground( Color.BLUE );
    else if( playerColour[ i ] == 3 )
        playerPlace[ i ].setForeground( Color.RED );
    else if( playerColour[ i ] == 4 )
    {
        playerPlace[ i ].setForeground( Color.ORANGE );
        playerPlace[ i ].setBackground( Color.GRAY );
    }
    else if( playerColour[ i ] == 5 )
    {
        playerPlace[ i ].setForeground( Color.GREEN );
        playerPlace[ i ].setBackground( Color.GRAY );
    }
    else if( playerColour[ i ] == 6 )
    {
        playerPlace[ i ].setForeground( Color.YELLOW );
        playerPlace[ i ].setBackground( Color.GRAY );
    }
}

player = 0;

for( int i = 0; i < Info.playerNumber(); i++ )
    go.add( playerPlace[ i ] );

playerTurn = new JLabel();
playerTurn.setText( "It is " + playerPlace[ player ].getText() + "'s turn" );
playerTurn.setBounds( 450, 80, 130, 25 );
topdesk.add( playerTurn );
}

```

```

private static int loadSettings()
{
    RandomAccessUserRecord record = new RandomAccessUserRecord();
    int pNumber = 0, count = 1;

    try
    {
        boolean load = true;

        while( load )
        {
            file.seek( ( count - 1 ) * record.size() );
            record.read( file );
            if( count > 6 )
                load = false;

            if( record.getToken() != 0 )
            {
                pNumber++;
                count++;
            }
            else
                count++;
        }
    }
    catch( IOException iox ) {
    }

    askFrame.setVisible( false );
    boardFrame.setVisible( true );
    return pNumber;
}

private static String getPropertyName( int pos )
{
    switch( pos )
    {
        case 0: return "Mediterranean Avenue";
        case 1: return "Baltic Avenue";
        case 2: return "Reading Railroad";
        case 3: return "Oriental Avenue";
        case 4: return "Vermont Avenue";
        case 5: return "Connecticut Avenue";
        case 6: return "St Charles Place";
        case 7: return "Electric Company";
        case 8: return "States Avenue";
        case 9: return "Virginia Avenue";
        case 10: return "Pennsylvania Railroad";
        case 11: return "St James Place";
        case 12: return "Tennessee Avenue";
        case 13: return "New York Avenue";
    }
}

```

```

    case 14: return "Kentucky Avenue";
    case 15: return "Indiana Avenue";
    case 16: return "Illinois Avenue";
    case 17: return "B & O Railroad";
    case 18: return "Atlantic Avenue";
    case 19: return "Ventnor Avenue";
    case 20: return "Water Works";
    case 21: return "Marvin Gardens";
    case 22: return "Pacific Avenue";
    case 23: return "North Carolina Avenue";
    case 24: return "Pennsylvania Avenue";
    case 25: return "Short Line Railroad";
    case 26: return "Park Place";
    case 27: return "Boardwalk";
    }
    return "";
}

private static void buyIt( int position )
{
    String p = getProperty( position );

    Properties properties = new Properties( position, 0 );
    if( properties.getCost() < playerPos[ player ].getMoney() )
    {
        int choice = JOptionPane.showConfirmDialog( null,
            "Would you like to buy " + p + "?", "Buy Property",
            JOptionPane.YES_NO_OPTION );

        if( choice == 0 )
        {
            playerPos[ player ].setMoney( playerPos[ player ].getMoney()
                - properties.getCost() );
            playerM.setCash( player, playerPos[ player ].getMoney(), file );
            bank[ position ] = 0;

            switch( player )
            {
                case 0: first[ position ] = 1; break;
                case 1: second[ position ] = 1; break;
                case 2: third[ position ] = 1; break;
                case 3: fourth[ position ] = 1; break;
                case 4: fifth[ position ] = 1; break;
                case 5: sixth[ position ] = 1; break;
            }
        }
    }
    else
    {
        JOptionPane.showMessageDialog( null,
            "You need more money to purchase this property",

```

```

        "Cannot Purchase Property", JOptionPane.ERROR_MESSAGE );
    }
}

private static void payIt( int position )
{
    boolean found = false;
    int houses = 0, current = 0;
    String p = getProperty( position );

    switch( player )
    {
        case 0:
            if( first[ position ] > 0 )
            {
                JOptionPane.showMessageDialog( null, "You currently own " + p,
                    "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
                current = 1;
            }
            break;
        case 1:
            if( second[ position ] > 0 )
            {
                JOptionPane.showMessageDialog( null, "You currently own " + p,
                    "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
                current = 2;
            }
            break;
        case 2:
            if( third[ position ] > 0 )
            {
                JOptionPane.showMessageDialog( null, "You currently own " + p,
                    "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
                current = 3;
            }
            break;
        case 3:
            if( fourth[ position ] > 0 )
            {
                JOptionPane.showMessageDialog( null, "You currently own " + p,
                    "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
                current = 4;
            }
            break;
        case 4:
            if( fifth[ position ] > 0 )
            {
                JOptionPane.showMessageDialog( null, "You currently own " + p,
                    "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
                current = 5;
            }
    }
}

```

```

        break;
    case 5:
        if( sixth[ position ] > 0 )
        {
            JOptionPane.showMessageDialog( null, "You currently own " + p,
                "Property Ownership", JOptionPane.INFORMATION_MESSAGE );
            current = 6;
        }
        break;
}

while( !found )
{
    for( int i = 0; i < Info.playerNumber(); i++ )
    {
        switch( i )
        {
            case 0:
                if( first[ position ] > 0 )
                {
                    houses = first[ position ];
                    found = true;
                }
                break;
            case 1:
                if( second[ position ] > 0 )
                {
                    houses = second[ position ];
                    found = true;
                }
                break;
            case 2:
                if( third[ position ] > 0 )
                {
                    houses = third[ position ];
                    found = true;
                }
                break;
            case 3:
                if( fourth[ position ] > 0 )
                {
                    houses = fourth[ position ];
                    found = true;
                }
                break;
            case 4:
                if( fifth[ position ] > 0 )
                {
                    houses = fifth[ position ];
                    found = true;
                }
        }
    }
}

```

```

        break;
    case 5:
        if( sixth[ position ] > 0 )
        {
            houses = sixth[ position ];
            found = true;
        }
        break;
    }
}
}

```

```

Properties properties = new Properties( position, houses );

```

```

if( first[ position ] > 0 && current != 1 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 0 ].setMoney( playerPos[ 0 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 0, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 0 ].getName(),
        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
}
else if( second[ position ] > 0 && current != 2 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 1 ].setMoney( playerPos[ 1 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 1, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 1 ].getName(),
        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
}
else if( third[ position ] > 0 && current != 3 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 2 ].setMoney( playerPos[ 2 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 2, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 2 ].getName(),

```

```

        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
    }
else if( fourth[ position ] > 0 && current != 4 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 3 ].setMoney( playerPos[ 3 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 3, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 3 ].getName(),
        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
}
else if( fifth[ position ] > 0 && current != 5 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 4 ].setMoney( playerPos[ 4 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 4, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 4 ].getName(),
        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
}
else if( sixth[ position ] > 0 && current != 6 )
{
    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - properties.getRent() );
    playerPos[ 5 ].setMoney( playerPos[ 5 ].getMoney()
        + properties.getRent() );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );
    playerM.setCash( 5, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You paid $"
        + properties.getRent() + " to " + playerPos[ 5 ].getName(),
        "Rent Pay", JOptionPane.INFORMATION_MESSAGE );
}
}

private static void mortgageProperty( int position )
{
    //sets the property array to 8, the mortgaged state
    switch( player )
    {
        case 0: first[ position ] = 8; break;
        case 1: second[ position ] = 8; break;
        case 2: third[ position ] = 8; break;
    }
}

```

```

case 3: fourth[ position ] = 8; break;
case 4: fifth[ position ] = 8; break;
case 5: sixth[ position ] = 8; break;
}

Properties property = new Properties( position, 0 );
int mortgage = property.getCost() / 2;
playerPos[ player ]
    .setMoney( playerPos[ player ].getMoney() + mortgage );
playerM.setCash( player, playerPos[ player ].getMoney(), file );

JOptionPane.showMessageDialog( null, "You got $" + mortgage + " by"
    + " mortgaging the " + getProperty( position )
    + " property.", "Mortgaged Property",
    JOptionPane.INFORMATION_MESSAGE );
}

private static void unmortgageProperty( int position )
{
    //sets the property array to 1, the unmortgaged state
    switch( player )
    {
    case 0: first[ position ] = 1; break;
    case 1: second[ position ] = 1; break;
    case 2: third[ position ] = 1; break;
    case 3: fourth[ position ] = 1; break;
    case 4: fifth[ position ] = 1; break;
    case 5: sixth[ position ] = 1; break;
    }

    Properties property = new Properties( position, 0 );
    int unmortgage = ( ( property.getCost() / 2 ) * 110 ) / 100;
    playerPos[ player ].setMoney( playerPos[ player ].getMoney() - unmortgage );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "It cost $" + unmortgage + " to"
        + " unmortgage the " + getProperty( position )
        + " property.", "Unmortgaged Property",
        JOptionPane.INFORMATION_MESSAGE );
}

private static void propertyFrame( int pos )
{
    boolean ownedbyPlayer = false;
    int checker = 0;
    JLabel ownership = new JLabel();
    JLabel propertyName = new JLabel();
    final int position = pos;

    properties = new int[ 28 ];
    final JButton buyHouse = new JButton( "Buy House" );

```

```
final JButton sellHouse = new JButton( "Sell House" );
final JButton mortgage = new JButton( "Mortgage" );
final JButton unmortgage = new JButton( "Unmortgage" );
```

```
buyHouse.setEnabled( false );
sellHouse.setEnabled( false );
mortgage.setEnabled( false );
unmortgage.setEnabled( false );
```

```
JFrame frame = new JFrame();
```

```
if( bank[ position ] == 1 )
    ownership.setText( "Currently Owned by Bank" );
else
{
    checker = 0;
    boolean found = false;

    while( !found )
    {
        for( int s = 0; s < Info.playerNumber(); s++ )
        {
            switch( s )
            {
                {
                case 0:
                    if( first[ position ] > 0 )
                    {
                        checker = 0;
                        found = true;
                    }
                    break;
                case 1:
                    if( second[ position ] > 0 )
                    {
                        checker = 1;
                        found = true;
                    }
                    break;
                case 2:
                    if( third[ position ] > 0 )
                    {
                        checker = 2;
                        found = true;
                    }
                    break;
                case 3:
                    if( fourth[ position ] > 0 )
                    {
                        checker = 3;
                        found = true;
                    }
                }
            }
        }
    }
}
```

```

        break;
    case 4:
        if( fifth[ position ] > 0 )
        {
            checker = 4;
            found = true;
        }
        break;
    case 5:
        if( sixth[ position ] > 0 )
        {
            checker = 5;
            found = true;
        }
        break;
    }
}

ownership.setText( "Currently Owned by " + playerPos[ checker ].getName() );
}

propertyName.setText( getProperty( position ) );

frame.getContentPane().setLayout( new FlowLayout() );
frame.getContentPane().add( propertyName );
frame.getContentPane().add( ownership );
frame.getContentPane().add( buyHouse );
frame.getContentPane().add( sellHouse );
frame.getContentPane().add( mortgage );
frame.getContentPane().add( unmortgage );

if( playerPos[ player ].getName() == playerPos[ checker ].getName() )
    ownedbyPlayer = true;

int mortgageNo = 0;
boolean mortgageStatus = false;

if( ownedbyPlayer )
{
    if( checker == 0 )
    {
        mortgageNo = first[ position ];
        properties = first;
    }
    else if( checker == 1 )
    {
        mortgageNo = second[ position ];
        properties = second;
    }
    else if( checker == 2 )

```

```

    {
        mortgageNo = third[ position ];
        properties = third;
    }
else if( checker == 3 )
    {
        mortgageNo = fourth[ position ];
        properties = fourth;
    }
else if( checker == 4 )
    {
        mortgageNo = fifth[ position ];
        properties = fifth;
    }
else if( checker == 5 )
    {
        mortgageNo = sixth[ position ];
        properties = sixth;
    }
}

if( mortgageNo == 8 )
    mortgageStatus = true;

if( mortgageStatus )
    unmortgage.setEnabled( true );
else if( !mortgageStatus && ownedbyPlayer )
    mortgage.setEnabled( true );

class mortgageListener implements ActionListener
{
    public void actionPerformed((ActionEvent e )
    {
        mortgageProperty( position );
        unmortgage.setEnabled( true );
        mortgage.setEnabled( false );
    }
}
ActionListener mortgageListen = new mortgageListener();
mortgage.addActionListener( mortgageListen );

class unmortgageListener implements ActionListener
{
    public void actionPerformed((ActionEvent e )
    {
        unmortgageProperty( position );
        mortgage.setEnabled( true );
        unmortgage.setEnabled( false );
    }
}
ActionListener unmortgageListen = new unmortgageListener();

```

```

unmortgage.addActionListener( unmortgageListen );

if( checkMonopoly( position, properties ) )
{
    int houseCost = houseCost( position );
    if( playerPos[ player ].getMoney() > houseCost
        && properties[ position ] < 7 )
        // if player has enough cash and not yet at house limit to buy
        buyHouse.setEnabled( true );
}

else if( checkMonopoly( position, properties ) )
{
    if( properties[ position ] > 0 )
        // if player has houses to sell
        sellHouse.setEnabled( true );
}

class buyHouseListener implements ActionListener
{
    public void actionPerformed((ActionEvent e)
    {
        buyHouse( position ); // builds a house
        if( properties[ position ] == 5
            || houseCost( position ) > playerPos[ player ]
                .getMoney() )
            buyHouse.setEnabled( false );
        if( properties[ position ] > 0 )
            sellHouse.setEnabled( true );
    }
}
ActionListener buyHouseListen = new buyHouseListener();
buyHouse.addActionListener( buyHouseListen );

class sellHouseListener implements ActionListener
{
    public void actionPerformed((ActionEvent e)
    {
        sellHouse( position ); // sells a house
        if( properties[ position ] == 0 ) // if no more houses to sell
            sellHouse.setEnabled( false );
        if( houseCost( position ) > playerPos[ player ].getMoney() )
            buyHouse.setEnabled( true );
    }
}
ActionListener sellHouseListen = new sellHouseListener();
sellHouse.addActionListener( sellHouseListen );

frame.setTitle( "Property Settings" );
frame.setSize( 265, 125 );
frame.show();

```

```

frame.setResizable( false );
frame.setLocation( 550, 250 );
frame.setDefaultCloseOperation( JFrame.DISPOSE_ON_CLOSE );
}

private static int houseCost( int pos )
{
    if( ( pos == 0 ) || ( pos == 1 ) || ( pos == 3 ) || ( pos == 4 )
        || ( pos == 5 ) )
        return 50;
    else if( ( pos == 6 ) || ( pos == 8 ) || ( pos == 9 ) || ( pos == 11 )
        || ( pos == 12 ) || ( pos == 13 ) )
        return 100;
    else if( ( pos == 14 ) || ( pos == 15 ) || ( pos == 16 )
        || ( pos == 18 ) || ( pos == 19 ) || ( pos == 21 ) )
        return 150;
    else if( ( pos == 22 ) || ( pos == 23 ) || ( pos == 24 )
        || ( pos == 26 ) || ( pos == 27 ) )
        return 200;
    else
        return 0;
}

private static boolean checkMonopoly( int pos, int[] currentPlayer )
{
    switch( pos )
    {
    case 0:
        // If player owns both Mediterranean and Baltic
        if( currentPlayer[ 0 ] == 1 && currentPlayer[ 1 ] == 1 )
            return true;
        else
            return false;
    case 1:
        // If player in question owns both Mediterranean and Baltic
        if( currentPlayer[ 0 ] == 1 && currentPlayer[ 1 ] == 1 )
            return true;
        else
            return false;
    case 3:
        // If player in question owns Oriental, Vermont & Connecticut
        if( currentPlayer[ 3 ] == 1 && currentPlayer[ 4 ] == 1
            && currentPlayer[ 5 ] == 1 )
            return true;
        else
            return false;
    case 8:
        // If player in question owns Oriental, Vermont & Connecticut
        if( currentPlayer[ 3 ] == 1 && currentPlayer[ 4 ] == 1
            && currentPlayer[ 5 ] == 1 )
            return true;
    }
}

```

```

else
    return false;
case 9:
    // If player in question owns Oriental, Vermont & Connecticut
    if( currentPlayer[ 3 ] == 1 && currentPlayer[ 4 ] == 1
        && currentPlayer[ 5 ] == 1 )
        return true;
    else
        return false;
case 11:
    // If player in question owns St Charles, States & Virginia
    if( currentPlayer[ 6 ] == 1 && currentPlayer[ 8 ] == 1
        && currentPlayer[ 9 ] == 1 )
        return true;
    else
        return false;
case 13:
    // If player in question owns St Charles, States & Virginia
    if( currentPlayer[ 6 ] == 1 && currentPlayer[ 8 ] == 1
        && currentPlayer[ 9 ] == 1 )
        return true;
    else
        return false;
case 14:
    // If player in question owns St Charles, States & Virginia
    if( currentPlayer[ 6 ] == 1 && currentPlayer[ 8 ] == 1
        && currentPlayer[ 9 ] == 1 )
        return true;
    else
        return false;
case 16:
    // If player in question owns St James, Tennessee & New York
    if( currentPlayer[ 11 ] == 1 && currentPlayer[ 12 ] == 1
        && currentPlayer[ 13 ] == 1 )
        return true;
    else
        return false;
case 18:
    // If player in question owns St James, Tennessee & New York
    if( currentPlayer[ 11 ] == 1 && currentPlayer[ 12 ] == 1
        && currentPlayer[ 13 ] == 1 )
        return true;
    else
        return false;
case 19:
    // If player in question owns St James, Tennessee & New York
    if( currentPlayer[ 11 ] == 1 && currentPlayer[ 12 ] == 1
        && currentPlayer[ 13 ] == 1 )
        return true;
    else
        return false;

```

```

case 21:
    // If player in question owns Kentucky, Indiana & Illinois
    if( currentPlayer[ 14 ] == 1 && currentPlayer[ 15 ] == 1
        && currentPlayer[ 16 ] == 1 )
        return true;
    else
        return false;
case 23:
    // If player in question owns Kentucky, Indiana & Illinois
    if( currentPlayer[ 14 ] == 1 && currentPlayer[ 15 ] == 1
        && currentPlayer[ 16 ] == 1 )
        return true;
    else
        return false;
case 24:
    // If player in question owns Kentucky, Indiana & Illinois
    if( currentPlayer[ 14 ] == 1 && currentPlayer[ 15 ] == 1
        && currentPlayer[ 16 ] == 1 )
        return true;
    else
        return false;
case 26:
    // If player in question owns Atlantic, Ventnor & Marvin Gardens
    if( currentPlayer[ 18 ] == 1 && currentPlayer[ 19 ] == 1
        && currentPlayer[ 21 ] == 1 )
        return true;
    else
        return false;
case 27:
    //If player in question owns Atlantic, Ventnor & Marvin Gardens
    if( currentPlayer[ 18 ] == 1 && currentPlayer[ 19 ] == 1
        && currentPlayer[ 21 ] == 1 )
        return true;
    else
        return false;
case 29:
    //If player in question owns Atlantic, Ventnor & Marvin Gardens
    if( currentPlayer[ 18 ] == 1 && currentPlayer[ 19 ] == 1
        && currentPlayer[ 21 ] == 1 )
        return true;
    else
        return false;
case 31:
    /*If player in question owns Pacific, North Carolina &
    Pennsylvania */
    if( currentPlayer[ 22 ] == 1 && currentPlayer[ 23 ] == 1
        && currentPlayer[ 24 ] == 1 )
        return true;
    else
        return false;
case 32:

```

```

    /*If player in question owns Pacific, North Carolina &
    Pennsylvania */
    if( currentPlayer[ 22 ] == 1 && currentPlayer[ 23 ] == 1
        && currentPlayer[ 24 ] == 1 )
        return true;
    else
        return false;
case 34:
    /*If player in question owns Pacific, North Carolina &
    Pennsylvania */
    if( currentPlayer[ 22 ] == 1 && currentPlayer[ 23 ] == 1
        && currentPlayer[ 24 ] == 1 )
        return true;
    else
        return false;
case 37:
    //If player in question owns both Park Place & Boardwalk
    if( currentPlayer[ 26 ] == 1 && currentPlayer[ 27 ] == 1 )
        return true;
    else
        return false;
case 39:
    //If player in question owns both Park Place & Boardwalk
    if( currentPlayer[ 26 ] == 1 && currentPlayer[ 27 ] == 1 )
        return true;
    else
        return false;
}
return false;
}

private static void buyHouse( int pos )
{
    int current = 0;

    switch( player )
    {
    case 0:
        if( first[ pos ] > 0 )
            current = 0;
    case 1:
        if( second[ pos ] > 0 )
            current = 1;
    case 2:
        if( third[ pos ] > 0 )
            current = 2;
    case 3:
        if( fourth[ pos ] > 0 )
            current = 3;
    case 4:
        if( fifth[ pos ] > 0 )

```

```

        current = 4;
    case 5:
        if( sixth[ pos ] > 0 )
            current = 5;
    }

    if( current == 0 )
    {
        first[ pos ]++;
    }
    else if( current == 1 )
    {
        second[ pos ]++;
    }
    else if( current == 2 )
    {
        third[ pos ]++;
    }
    else if( current == 3 )
    {
        fourth[ pos ]++;
    }
    else if( current == 4 )
    {
        fifth[ pos ]++;
    }
    else if( current == 5 )
    {
        sixth[ pos ]++;
    }

    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        - houseCost( pos ) );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "You Paid $" + houseCost( pos )
        + " for a house on " + getPropertyname( pos ) + ".",
        "Bought a House", JOptionPane.INFORMATION_MESSAGE );
}

/**
 * Method sellHouse sells a house for the current player at property of
 * choice
 */
private static void sellHouse( int pos )
{
    int current = 0;

    switch( player )
    {
    case 0:

```

```

        if( first[ pos ] > 0 )
            current = 0;
    case 1:
        if( second[ pos ] > 0 )
            current = 1;
    case 2:
        if( third[ pos ] > 0 )
            current = 2;
    case 3:
        if( fourth[ pos ] > 0 )
            current = 3;
    case 4:
        if( fifth[ pos ] > 0 )
            current = 4;
    case 5:
        if( sixth[ pos ] > 0 )
            current = 5;
    }

    if( current == 0 )
    {
        first[ pos ]--;
    }
    else if( current == 1 )
    {
        second[ pos ]--;
    }
    else if( current == 2 )
    {
        third[ pos ]--;
    }
    else if( current == 3 )
    {
        fourth[ pos ]--;
    }
    else if( current == 4 )
    {
        fifth[ pos ]--;
    }
    else if( current == 5 )
    {
        sixth[ pos ]--;
    }

    playerPos[ player ].setMoney( playerPos[ player ].getMoney()
        + houseCost( pos ) / 2 );
    playerM.setCash( player, playerPos[ player ].getMoney(), file );

    JOptionPane.showMessageDialog( null, "Got $" + houseCost( pos ) / 2
        + " for selling a house on " + getPropertyNames( pos ) + ".",
        "Sold a House", JOptionPane.INFORMATION_MESSAGE );

```

```

}

private static PlayerMenu playerM;
private static int FRAME_WIDTH = 950, FRAME_HEIGHT = 643, WIDTH, HEIGHT;
private static JDesktopPane desktop, topdesk;
private static JFrame boardFrame, askFrame;
private static JButton go, purple1, chest1, purple2, tax1, rr1, lblue1,
    chance1, lblue2, lblue3, jail, pink1, utility1, pink2, pink3, rr2,
    orange1, chest2, orange2, orange3, fPark, red1, chance2, red2,
    red3, rr3, yellow1, yellow2, utility2, yellow3, toJail, green1,
    green2, chest3, green3, rr4, chance3, blue1, tax2, blue2, rollDice;
private static JLabel roller, playerTurn, playerPlace[];
private static int doubles = 0, player, currentdiceRoll, properties[];
private static PlayerStatus playerPos[];
private static RandomAccessFile file;
private static int first[], second[], third[], fourth[], fifth[], sixth[], bank[];
}

/*=====
Program / Class: Dossier, Info          Author: Ross Chan
Purpose of Class: Asks and reads in the number of players.
Date of This Revision: March 28, 2007   Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0   Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP   IDE: Eclipse 3.2
=====*/

```

```

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*.*;

public class Info extends JFrame
{
    public Info()
    {
        JPanel panel = new JPanel();
        JLabel players = new JLabel( "How many players? ( Two Minimum - Six Maximum )" );
        selectPlayers = new JTextField( 10 );
        panel.add( players );
        panel.add( selectPlayers );
        playerNumber = 2;

        JButton okButton = new JButton( "OK" );
        class AddActionListener implements ActionListener
        {
            public void actionPerformed( ActionEvent event )
            {
                try
                {

```

```

String read = selectPlayers.getText();
playerNumber = Integer.parseInt( read );

if( playerNumber < 2 || playerNumber > 6 )
    JOptionPane.showMessageDialog( null, "Must be between 2 and 6",
        "Incorrect Player Number", JOptionPane.ERROR_MESSAGE );

}
catch( NumberFormatException nfx )
{
    if( nfx == null )
        setVisible( false );
    else
        JOptionPane.showMessageDialog( null, "Must be a number between 2 and 6",
            "Incorrect Player Number", JOptionPane.ERROR_MESSAGE );
}

setVisible( false );
}
}

ActionListener listen = new AddActionListener();
okButton.addActionListener( listen );

panel.add( okButton );
add( panel, BorderLayout.CENTER );

setSize( FRAME_WIDTH, FRAME_HEIGHT );
setDefaultCloseOperation( HIDE_ON_CLOSE );
setTitle( "Player Selection" );
setVisible( false );
setClosable( true );
}

public static int playerNumber()
{
    return playerNumber;
}

public static void setPlayerNumber( int a )
{
    playerNumber = a;
}

private static int FRAME_WIDTH = 320, FRAME_HEIGHT = 100;
private static JTextField selectPlayers;
private static int playerNumber;
}

```

```

/*=====
Program / Class: Dossier, UserInfo      Author: Ross Chan
Purpose of Class: Main window that reads in player names and token colour.
Date of This Revision: March 28, 2007   Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0   Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP   IDE: Eclipse 3.2
=====*/

```

```

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

import java.io.*;

public class UserInfo extends JFrame
{
    public UserInfo()
    {
        loop = Info.playerNumber(); //get the number of players

        JPanel panel = new JPanel();
        JPanel startPane = new JPanel();
        JLabel[] ask = new JLabel[ 6 ];

        for( int a = 0; a < 6; a++ )
        {
            ask[ a ] = new JLabel( "Player Name" ); //create 6 JLabels
        }

        name = new JTextField[ 6 ]; //Initialize 6 different JTextFields
        name[ 0 ] = new JTextField( 13 );
        name[ 1 ] = new JTextField( 13 );
        name[ 2 ] = new JTextField( 13 );
        name[ 3 ] = new JTextField( 13 );
        name[ 4 ] = new JTextField( 13 );
        name[ 5 ] = new JTextField( 13 );

        player = new String[ loop ]; //create new player name strings

        //initialization of JComboBoxes
        selectToken1 = new JComboBox();
        selectToken1.addItem( "-----" );
        selectToken1.addItem( "Purple" );
        selectToken1.addItem( "Blue" );
        selectToken1.addItem( "Red" );
        selectToken1.addItem( "Orange" );
        selectToken1.addItem( "Green" );
        selectToken1.addItem( "Yellow" );
    }
}

```

```

selectToken2 = new JComboBox();
selectToken2.addItem( "-----" );
selectToken2.addItem( "Purple" );
selectToken2.addItem( "Blue" );
selectToken2.addItem( "Red" );
selectToken2.addItem( "Orange" );
selectToken2.addItem( "Green" );
selectToken2.addItem( "Yellow" );

selectToken3 = new JComboBox();
selectToken3.addItem( "-----" );
selectToken3.addItem( "Purple" );
selectToken3.addItem( "Blue" );
selectToken3.addItem( "Red" );
selectToken3.addItem( "Orange" );
selectToken3.addItem( "Green" );
selectToken3.addItem( "Yellow" );

selectToken4 = new JComboBox();
selectToken4.addItem( "-----" );
selectToken4.addItem( "Purple" );
selectToken4.addItem( "Blue" );
selectToken4.addItem( "Red" );
selectToken4.addItem( "Orange" );
selectToken4.addItem( "Green" );
selectToken4.addItem( "Yellow" );

selectToken5 = new JComboBox();
selectToken5.addItem( "-----" );
selectToken5.addItem( "Purple" );
selectToken5.addItem( "Blue" );
selectToken5.addItem( "Red" );
selectToken5.addItem( "Orange" );
selectToken5.addItem( "Green" );
selectToken5.addItem( "Yellow" );

selectToken6 = new JComboBox();
selectToken6.addItem( "-----" );
selectToken6.addItem( "Purple" );
selectToken6.addItem( "Blue" );
selectToken6.addItem( "Red" );
selectToken6.addItem( "Orange" );
selectToken6.addItem( "Green" );
selectToken6.addItem( "Yellow" );

//listener for all siz JComboBoxes
class BoxListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        String name1 = (String)selectToken1.getSelectedItem();

```

```
String name2 = (String)selectToken2.getSelectedItemAt();
String name3 = (String)selectToken3.getSelectedItemAt();
String name4 = (String)selectToken4.getSelectedItemAt();
String name5 = (String)selectToken5.getSelectedItemAt();
String name6 = (String)selectToken6.getSelectedItemAt();
```

```
if( name1 == "Purple" )
    token1 = 1;
else if( name1 == "Blue" )
    token1 = 2;
else if( name1 == "Red" )
    token1 = 3;
else if( name1 == "Orange" )
    token1 = 4;
else if( name1 == "Green" )
    token1 = 5;
else if( name1 == "Yellow" )
    token1 = 6;
```

```
if( name2 == "Purple" )
    token2 = 1;
else if( name2 == "Blue" )
    token2 = 2;
else if( name2 == "Red" )
    token2 = 3;
else if( name2 == "Orange" )
    token2 = 4;
else if( name2 == "Green" )
    token2 = 5;
else if( name2 == "Yellow" )
    token2 = 6;
```

```
if( name3 == "Purple" )
    token3 = 1;
else if( name3 == "Blue" )
    token3 = 2;
else if( name3 == "Red" )
    token3 = 3;
else if( name3 == "Orange" )
    token3 = 4;
else if( name3 == "Green" )
    token3 = 5;
else if( name3 == "Yellow" )
    token3 = 6;
```

```
if( name4 == "Purple" )
    token4 = 4;
else if( name1 == "Blue" )
    token4 = 4;
else if( name1 == "Red" )
    token4 = 4;
```

```

else if( name1 == "Orange" )
    token4 = 4;
else if( name1 == "Green" )
    token4 = 4;
else if( name1 == "Yellow" )
    token4 = 4;

if( name5 == "Purple" )
    token5 = 1;
else if( name5 == "Blue" )
    token5 = 2;
else if( name5 == "Red" )
    token5 = 3;
else if( name5 == "Orange" )
    token5 = 4;
else if( name5 == "Green" )
    token5 = 5;
else if( name5 == "Yellow" )
    token5 = 6;

if( name6 == "Purple" )
    token6 = 1;
else if( name6 == "Blue" )
    token6 = 2;
else if( name6 == "Red" )
    token6 = 3;
else if( name6 == "Orange" )
    token6 = 4;
else if( name6 == "Green" )
    token6 = 5;
else if( name6 == "Yellow" )
    token6 = 6;
    }
}
BoxListener listener = new BoxListener();
selectToken1.addActionListener( listener );
selectToken2.addActionListener( listener );
selectToken3.addActionListener( listener );
selectToken4.addActionListener( listener );
selectToken5.addActionListener( listener );
selectToken6.addActionListener( listener );

//loop for setting enabled state, depending on how many players
//the user picked
switch ( loop )
{
    case 2:
    {
        ask[5].setEnabled( false ); name[5].setEnabled( false );
        selectToken6.setEnabled( false );
        ask[4].setEnabled( false ); name[4].setEnabled( false );
    }
}

```

```

        selectToken5.setEnabled( false );
        ask[3].setEnabled( false ); name[3].setEnabled( false );
        selectToken4.setEnabled( false );
        ask[2].setEnabled( false ); name[2].setEnabled( false );
        selectToken3.setEnabled( false ); break;
    }
    case 3:
    {
        ask[5].setEnabled( false ); name[5].setEnabled( false );
        selectToken6.setEnabled( false );
        ask[4].setEnabled( false ); name[4].setEnabled( false );
        selectToken5.setEnabled( false );
        ask[3].setEnabled( false ); name[3].setEnabled( false );
        selectToken4.setEnabled( false ); break;
    }
    case 4:
    {
        ask[5].setEnabled( false ); name[5].setEnabled( false );
        selectToken6.setEnabled( false );
        ask[4].setEnabled( false ); name[4].setEnabled( false );
        selectToken5.setEnabled( false ); break;
    }
    case 5:
    {
        ask[5].setEnabled( false ); name[5].setEnabled( false );
        selectToken6.setEnabled( false ); break;
    }
}

```

```

panel.add( ask[0] ); panel.add( name[0] ); panel.add( selectToken1 );
panel.add( ask[1] ); panel.add( name[1] ); panel.add( selectToken2 );
panel.add( ask[2] ); panel.add( name[2] ); panel.add( selectToken3 );
panel.add( ask[3] ); panel.add( name[3] ); panel.add( selectToken4 );
panel.add( ask[4] ); panel.add( name[4] ); panel.add( selectToken5 );
panel.add( ask[5] ); panel.add( name[5] ); panel.add( selectToken6 );

```

```

final JButton okButton = new JButton( "Start Game" );
okButton.setEnabled( false );

```

```

//ok button disables welcome window and switches to game board
class AddActionListener implements ActionListener

```

```

{
    public void actionPerformed((ActionEvent event)
    {
        setVisible( false );
        DigiMonopoly.boardFrame.setVisible( true );
        DigiMonopoly.askFrame.setVisible( false );
        DigiMonopoly.setStatus( file );
    }
}

```

```

ActionListener listen = new AddActionListener();
okButton.addActionListener( listen );

JButton save = new JButton( "Save Player Settings" );

//saves the RandomAccessFile
class AddSaveListener implements ActionListener
{
    public void actionPerformed( ActionEvent event )
    {
        saveFile();
        okButton.setEnabled( true );
    }
}

ActionListener listenS = new AddSaveListener();
save.addActionListener( listenS );

startPane.add( save );
startPane.add( okButton );

setLayout( new BorderLayout() );
add( panel, BorderLayout.CENTER );
add( startPane, BorderLayout.SOUTH );

setSize( FRAME_WIDTH, FRAME_HEIGHT );
setDefaultCloseOperation( HIDE_ON_CLOSE );
setTitle( "Player Settings" );
setVisible( false );
setClosable( true );
}

//method used to save player settings using a RandomAccessFile
public boolean saveFile()
{
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
    int pick = fileChooser.showSaveDialog( this );
    File fileName = fileChooser.getSelectedFile();
    setNames();

    if( pick == JFileChooser.CANCEL_OPTION )
        return false;

    if( fileName == null || fileName.getName().equals( "" ) )
    {
        JOptionPane.showMessageDialog( this, "Invalid File Name",
            "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
        return false;
    }
}

```

```

else
{
    try
    {
        RandomAccessUserRecord record = new RandomAccessUserRecord();

        file = new RandomAccessFile( fileName + ".dat", "rw" );
        switch( loop )
        {
            case 2:
            {
                record.setToken( token1 );
                record.setName( player[0] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                record.setToken( token2 );
                record.setName( player[1] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                break;
            }
            case 3:
            {
                record.setToken( token1 );
                record.setName( player[0] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                record.setToken( token2 );
                record.setName( player[1] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                record.setToken( token3 );
                record.setName( player[2] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                break;
            }
            case 4:
            {
                record.setToken( token1 );
                record.setName( player[0] );
                record.setScore( 1500 );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                record.setToken( token2 );
                record.setName( player[1] );

```

```

record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token3 );
record.setName( player[2] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token4 );
record.setName( player[3] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
}
case 5:
{
record.setToken( token1 );
record.setName( player[0] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token2 );
record.setName( player[1] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token3 );
record.setName( player[2] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token4 );
record.setName( player[3] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( token5 );
record.setName( player[4] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
}
case 6:
{
record.setToken( token1 );
record.setName( player[0] );
record.setScore( 1500 );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );

```

```

        record.setToken( token2 );
        record.setName( player[1] );
        record.setScore( 1500 );
        file.seek( ( record.getToken() - 1 ) * record.size() );
        record.write( file );
        record.setToken( token3 );
        record.setName( player[2] );
        record.setScore( 1500 );
        file.seek( ( record.getToken() - 1 ) * record.size() );
        record.write( file );
        record.setToken( token4 );
        record.setName( player[3] );
        record.setScore( 1500 );
        file.seek( ( record.getToken() - 1 ) * record.size() );
        record.write( file );
        record.setToken( token5 );
        record.setName( player[4] );
        record.setScore( 1500 );
        file.seek( ( record.getToken() - 1 ) * record.size() );
        record.write( file );
        record.setToken( token6 );
        record.setName( player[5] );
        record.setScore( 1500 );
        file.seek( ( record.getToken() - 1 ) * record.size() );
        record.write( file );
    }
}
}
catch( IOException iox )
{
    JOptionPane.showMessageDialog( this, "Cannot Save Settings",
        "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
    return false;
}

return true;
}
}

//gets the player's names from the JTextField
protected void setNames(){
    for( int i = 0; i < loop; i++ )
        player[ i ] = name[ i ].getText();
}

private static int FRAME_WIDTH = 400, FRAME_HEIGHT = 260;
private static int loop;
public static int token1, token2, token3, token4, token5, token6;
private static JComboBox selectToken1, selectToken2, selectToken3,
    selectToken4, selectToken5, selectToken6;
private static RandomAccessFile file;

```

```

    public static String player[];
    private static JTextField name[];
}

/*=====
Program / Class: Dossier, LoadFile      Author: Ross Chan
Purpose of Class: Loads a previously saved RandomAccessFile.
Date of This Revision: March 28, 2007   Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0   Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP   IDE: Eclipse 3.2
=====*/

import javax.swing.*;

import java.io.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoadFile extends JFrame
{
    public LoadFile()
    {
        JPanel panel = new JPanel();
        loadField = new JTextField( 16 );
        JButton browse = new JButton( "Browse" );
        JButton OKButton = new JButton( "Done" );

        panel.add( loadField ); panel.add( browse );
        panel.add( OKButton );

        add( panel );

        class BrowseListener implements ActionListener
        {
            public void actionPerformed( ActionEvent event )
            {
                loadFile();
            }
        }

        ActionListener listen = new BrowseListener();
        browse.addActionListener( listen );

        class DoneListener implements ActionListener
        {
            public void actionPerformed( ActionEvent event )
            {
                setVisible( false );
                DigiMonopoly.askFrame.setVisible( false );
            }
        }
    }
}

```

```

        DigiMonopoly.boardFrame.setVisible( true );
    }
}

ActionListener listener = new DoneListener();
OKButton.addActionListener( listener );

setSize( FRAME_WIDTH, FRAME_HEIGHT );
setDefaultCloseOperation( HIDE_ON_CLOSE );
setTitle( "Load Saved File" );
setVisible( false );
setClosable( true );
}

public boolean loadFile()
{
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setSelectionMode( JFileChooser.FILES_ONLY );
    int pick = fileChooser.showOpenDialog( this );
    loadField.setText( fileChooser.getName() );
    File fileName = fileChooser.getSelectedFile();

    if( pick == JFileChooser.CANCEL_OPTION )
        return false;

    if( fileName == null || fileName.getName().equals( "" ) )
    {
        JOptionPane.showMessageDialog( this, "Invalid File Name",
            "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
        return false;
    }

    else
    {
        try
        {
            file = new RandomAccessFile( fileName, "r" );
        }
        catch( IOException iox )
        {
            JOptionPane.showMessageDialog( this, "Cannot Load Settings",
                "Invalid File Name", JOptionPane.ERROR_MESSAGE );
        }

        return true;
    }
}

public RandomAccessFile setFile()
{
    return file;
}

```

```

    }

    private static RandomAccessFile file;
    private static int FRAME_WIDTH = 300, FRAME_HEIGHT = 100;
    private static JTextField loadField;
}

/*=====
Program / Class: Dossier, PlayerMenu          Author: Ross Chan
Purpose of Class: Displays the current information of every player.
Date of This Revision: March 28, 2007      Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0    Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP    IDE: Eclipse 3.2
=====*/

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

import java.awt.Color;
import java.io.*;

public class PlayerMenu extends JFrame
{
    public PlayerMenu( int a )
    {
        loop = a;

        JPanel panel = new JPanel( new GridLayout( 6,1 ) );

        player = new JLabel[ 6 ];
        playerName = new String[ 6 ];
        playerCash = new int[ 6 ];
        playerColour = new int[ 6 ];

        player[0] = new JLabel(); player[1] = new JLabel();
        player[2] = new JLabel(); player[3] = new JLabel();
        player[4] = new JLabel(); player[5] = new JLabel();

        money = new JLabel[ 6 ];
        money[0] = new JLabel(); money[1] = new JLabel();
        money[2] = new JLabel(); money[3] = new JLabel();
        money[4] = new JLabel(); money[5] = new JLabel();

        JButton record1 = new JButton( "Record" );
        JButton record2 = new JButton( "Record" );
        JButton record3 = new JButton( "Record" );
        JButton record4 = new JButton( "Record" );

```

```

JButton record5 = new JButton( "Record" );
JButton record6 = new JButton( "Record" );

switch ( loop )
{
    case 2:
    {
        player[5].setVisible( false ); record6.setVisible( false );
        money[5].setVisible( false );
        player[4].setVisible( false ); record5.setVisible( false );
        money[4].setVisible( false );
        player[3].setVisible( false ); record4.setVisible( false );
        money[3].setVisible( false );
        player[2].setVisible( false ); record3.setVisible( false );
        money[2].setVisible( false );
        break;
    }
    case 3:
    {
        player[5].setVisible( false ); record6.setVisible( false );
        money[5].setVisible( false );
        player[4].setVisible( false ); record5.setVisible( false );
        money[4].setVisible( false );
        player[3].setVisible( false ); record4.setVisible( false );
        money[3].setVisible( false );
        break;
    }
    case 4:
    {
        player[5].setVisible( false ); record6.setVisible( false );
        money[5].setVisible( false );
        player[4].setVisible( false ); record5.setVisible( false );
        money[4].setVisible( false );
        break;
    }
    case 5:
    {
        player[5].setVisible( false ); record6.setVisible( false );
        money[5].setVisible( false );
    }
}

panel.add( player[0] ); panel.add( money[0] );
panel.add( record1 );
panel.add( player[1] ); panel.add( money[1] );
panel.add( record2 );
panel.add( player[2] ); panel.add( money[2] );
panel.add( record3 );
panel.add( player[3] ); panel.add( money[3] );
panel.add( record4 );
panel.add( player[4] ); panel.add( money[4] );

```

```
panel.add( record5 );
panel.add( player[5] ); panel.add( money[5] );
panel.add( record6 );
```

```
class RecordListener implements ActionListener
{
    public void actionPerformed((ActionEvent event)
    {

    }
}
```

```
ActionListener roller = new RecordListener();
record1.addActionListener( roller );
record2.addActionListener( roller );
record3.addActionListener( roller );
record4.addActionListener( roller );
record5.addActionListener( roller );
record6.addActionListener( roller );
```

```
add( panel );
```

```
setSize( FRAME_WIDTH, FRAME_HEIGHT );
setTitle( "Player Window" );
setVisible( true );
}
```

```
public void setCash( int a, int b, RandomAccessFile file )
{
    RandomAccessUserRecord record = new RandomAccessUserRecord();
    money[ a ].setText( "$" + b );

    try
    {
        file.seek( ( a ) * record.size() );
        record.setScore( b );
    }
    catch( IOException iox )
    {
        JOptionPane.showMessageDialog( this, "Cannot Save Settings",
            "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
    }
}
```

```
public void readFile( RandomAccessFile fil )
{
    RandomAccessUserRecord record = new RandomAccessUserRecord();
    file = fil;

    try
    {
```

```

file.seek( 0 );
for( int count = 0; count < loop; count++ )
{
    do
    {
        record.read( file );
    } while( record.getToken() == 0 );
    player[ count ].setText( record.getName().trim() );
    money[ count ].setText( "$" + record.getScore() );
    playerName[ count ] = record.getName().trim();
    playerCash[ count ] = record.getScore();
    playerColour[ count ] = record.getToken();

    //set colours
    if( record.getToken() == 1 )
        player[ count ].setForeground( Color.MAGENTA );
    else if( record.getToken() == 2 )
        player[ count ].setForeground( Color.BLUE );
    else if( record.getToken() == 3 )
        player[ count ].setForeground( Color.RED );
    else if( record.getToken() == 4 )
        player[ count ].setForeground( Color.ORANGE );
    else if( record.getToken() == 5 )
        player[ count ].setForeground( Color.GREEN );
    else if( record.getToken() == 6 )
        player[ count ].setForeground( Color.YELLOW );
    }
}
catch( IOException iox )
{
    JOptionPane.showMessageDialog( this, "Cannot Save Settings",
        "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
}
}

public void saveFile()
{
    RandomAccessUserRecord record = new RandomAccessUserRecord();

    try
    {
        switch( loop )
        {
            case 2:
                record.setToken( playerColour[ 0 ] );
                record.setName( player[ 0 ].getText() );
                record.setScore( Integer.parseInt( money[ 0 ].getText() ) );
                file.seek( ( record.getToken() - 1 ) * record.size() );
                record.write( file );
                record.setToken( playerColour[ 1 ] );
                record.setName( player[ 1 ].getText() );

```

```

record.setScore( Integer.parseInt( money[ 1 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
case 3:
record.setToken( playerColour[ 0 ] );
record.setName( player[ 0 ].getText() );
record.setScore( Integer.parseInt( money[ 0 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.setToken( playerColour[ 1 ] );
record.setName( player[ 1 ].getText() );
record.setScore( Integer.parseInt( money[ 1 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 2 ] );
record.setName( player[ 2 ].getText() );
record.setScore( Integer.parseInt( money[ 2 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
case 4:
record.setToken( playerColour[ 0 ] );
record.setName( player[ 0 ].getText() );
record.setScore( Integer.parseInt( money[ 0 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 1 ] );
record.setName( player[ 1 ].getText() );
record.setScore( Integer.parseInt( money[ 1 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 2 ] );
record.setName( player[ 2 ].getText() );
record.setScore( Integer.parseInt( money[ 2 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 3 ] );
record.setName( player[ 3 ].getText() );
record.setScore( Integer.parseInt( money[ 3 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
case 5:
record.setToken( playerColour[ 0 ] );
record.setName( player[ 0 ].getText() );
record.setScore( Integer.parseInt( money[ 0 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 1 ] );
record.setName( player[ 1 ].getText() );
record.setScore( Integer.parseInt( money[ 1 ].getText() ) );

```

```

file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 2 ] );
record.setName( player[ 2 ].getText() );
record.setScore( Integer.parseInt( money[ 2 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 3 ] );
record.setName( player[ 3 ].getText() );
record.setScore( Integer.parseInt( money[ 3 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 4 ] );
record.setName( player[ 4 ].getText() );
record.setScore( Integer.parseInt( money[ 4 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
case 6:
record.setToken( playerColour[ 0 ] );
record.setName( player[ 0 ].getText() );
record.setScore( Integer.parseInt( money[ 0 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 1 ] );
record.setName( player[ 1 ].getText() );
record.setScore( Integer.parseInt( money[ 1 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 2 ] );
record.setName( player[ 2 ].getText() );
record.setScore( Integer.parseInt( money[ 2 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 3 ] );
record.setName( player[ 3 ].getText() );
record.setScore( Integer.parseInt( money[ 3 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 4 ] );
record.setName( player[ 4 ].getText() );
record.setScore( Integer.parseInt( money[ 4 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
record.setToken( playerColour[ 5 ] );
record.setName( player[ 5 ].getText() );
record.setScore( Integer.parseInt( money[ 5 ].getText() ) );
file.seek( ( record.getToken() - 1 ) * record.size() );
record.write( file );
break;
}

```

```

        JOptionPane.showMessageDialog( null, "Game Has Been Saved",
            "Save Game", JOptionPane.INFORMATION_MESSAGE );
    }
    catch( IOException iox )
    {
        JOptionPane.showMessageDialog( null, "Cannot Save Settings",
            "Incorrect File Name", JOptionPane.ERROR_MESSAGE );
    }
}

public static void deleteRecord(RandomAccessFile file, int user)
{
    RandomAccessUserRecord record = new RandomAccessUserRecord();
    try
    {
        int currentUser = user + 1; //add 1 due to use of arrays
        file.seek( ( currentUser - 1 ) * record.size() );
        record.setToken( 0 ); //set to zero, making record unreachable
        record.write( file );
    }
    catch( IOException iox )
    {
        JOptionPane.showMessageDialog( null, "Problem with Player Deletion",
            "Player Bankruptcy", JOptionPane.ERROR_MESSAGE );
    }
}

public String[] getPlayerName()
{
    return playerName;
}

public int[] getPlayerCash()
{
    return playerCash;
}

public int[] getPlayerColour()
{
    return playerColour;
}

private static int FRAME_WIDTH = 356, FRAME_HEIGHT = 587;
private static JLabel player[], money[];
private static int loop;
private static String playerName[];
private static int playerCash[], playerColour[];
private static RandomAccessFile file;
}

```

```
/*=====
Program / Class: Dossier, PlayerRecord      Author: Ross Chan
Purpose of Class: Record that reads in the user input.
Date of This Revision: March 28, 2007      Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0    Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP    IDE: Eclipse 3.2
=====*/
```

```
import java.io.Serializable;
```

```
/**
 * This program creates an inventory of the hardware in a store.
 * @version Version 1.0
 * @author Ross Chan
 */
```

```
public class PlayerRecord implements Serializable
{
    public PlayerRecord()
    {
        this( 0, "", 0 );
    }

    public PlayerRecord( int tok, String name, int amount )
    {
        setToken( tok );
        setName( name );
        setScore( amount );
    }

    public void setToken( int rec )
    {
        record = rec;
    }

    public int getToken()
    {
        return record;
    }

    public void setName( String name )
    {
        playerName = name;
    }

    public String getName()
    {
        return playerName;
    }
}
```

```

public void setScore( int amount )
{
    score = amount;
}

public int getScore()
{
    return score;
}

private static int record, score;
private static String playerName;
}

/*=====
Program / Class: Dossier, PlayerStatus      Author: Ross Chan
Purpose of Class: Defines the current users wealth, position, etc.
Date of This Revision: March 28, 2007      Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0    Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP    IDE: Eclipse 3.2
=====*/

public class PlayerStatus
{
    /**
     * Creates a new instance of Player, by taking in the player name and
     * calling the private constructor to initialize the player object
     */

    public PlayerStatus(String nam)
    {
        this( 0, 1500,nam ); //Instantiate player to new game stats
    }

    public PlayerStatus()
    {
        this( 0, 0, "" ); // Instantiate player to new game stats
    }

    /**
     * Constructor Player takes in all the values of each player and sets them
     * to the starting position
     */
    PlayerStatus( int pos, int cash, String n )
    {
        setMoney( cash );
        setPosition( pos ); // sets player position
        setName( n );
        inJail( false ); // sets player not in jail
    }
}

```

```

        inJailCount( 0 ); // set number of times in jail (in a row) to 0
    }

    /**
     * Sets the cash of the player
     * @param m the amount of money the player currently has
     */
    public void setMoney( int m )
    {
        cash = m; //sets cash to parameter m
    }

    /**
     * Returns the cash the specific player currently has
     * @return the amount of cash the player has
     */
    public int getMoney()
    {
        return cash; //returns the player's cash
    }

    /** Method setName sets the player name. */
    public void setName( String n )
    {
        name = n; // sets name to parameter n
    }

    /** Method getName returns the player name. */
    public String getName()
    {
        return name; // returns player name
    }

    /** Method setPosition sets the position of the player on the board */
    public void setPosition( int p )
    {
        position = p; // sets player position to parameter p
    }

    /**
     * Method getPosition returns the square a player is on. Go = 0, Boardwalk
     * =39, all other squares between those numbers in clockwise order
     */
    public int getPosition()
    {
        return position; // returns position integer
    }

    /**
     * Method setJail sets a boolean to indicate whether or not the player is in
     * jail

```

```

    */
public void inJail( boolean b )
{
    inJail = b; // sets jail boolean to parameter b
}

/** Method getJail returns a boolean indicating the player's jail status */
public boolean getJail()
{
    return inJail; // returns current jail status (in or out)
}

/**
 * Method setInJailCount sets an int to count how many turns a player has
 * been in jail
 */
public void inJailCount( int j )
{
    jailCount = j; // sets jail count to parameter j
}

/**
 * Method getInJailCount gets the int that counts how many turns a player
 * has been in jail
 */
public int getinJailCount()
{
    return jailCount; // returns how many consecutive turns player is in
                    // jail
}

private int cash; // Variable to control player's cash
private String name; // Variable to control player's name
private int position; // Variable to control player's position
private boolean inJail; // Boolean whether or not player is in jail
private int jailCount; // Variable how many turns player has been in jail
}

/*=====
Program / Class: Dossier, Properties          Author: Ross Chan
Purpose of Class: Settings for all the properties on the board.
Date of This Revision: March 28, 2007      Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0    Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP    IDE: Eclipse 3.2
=====*/

public class Properties
{
    public Properties( int place, int houses )

```

```

{
switch( place )
{
case 0:
    Mediterranean( houses );
    break;
case 1:
    Baltic( houses );
    break;
case 2:
    ReadingRR( houses );
    break;
case 3:
    Oriental( houses );
    break;
case 4:
    Vermont( houses );
    break;
case 5:
    Connecticut( houses );
    break;
case 6:
    StCharles( houses );
    break;
case 7:
    ElectricCompany( houses );
    break;
case 8:
    States( houses );
    break;
case 9:
    Virginia( houses );
    break;
case 10:
    PennsylvaniaRR( houses );
    break;
case 11:
    StJames( houses );
    break;
case 12:
    Tennessee( houses );
    break;
case 13:
    NewYork( houses );
    break;
case 14:
    Kentucky( houses );
    break;
case 15:
    Indiana( houses );
    break;
}
}

```

```

case 16:
    Illinois( houses );
    break;
case 17:
    BandORR( houses );
    break;
case 18:
    Atlantic( houses );
    break;
case 19:
    Ventnor( houses );
    break;
case 20:
    WaterWorks( houses );
    break;
case 21:
    Marvin( houses );
    break;
case 22:
    Pacific( houses );
    break;
case 23:
    NorthCarolina( houses );
    break;
case 24:
    Pennsylvania( houses );
    break;
case 25:
    ShortLineRR( houses );
    break;
case 26:
    ParkPlace( houses );
    break;
case 27:
    Boardwalk( houses );
    break;
}

}

private void Mediterranean( int house )
{
    setCost( 60 );
    switch( house )
    {
        case 1:
            setRent( 2 );
            break;
        case 2:
            setRent( 10 );
            break;
    }
}

```

```

    case 3:
        setRent( 30 );
        break;
    case 4:
        setRent( 90 );
        break;
    case 5:
        setRent( 160 );
        break;
    case 6:
        setRent( 250 );
        break;
    case 7:
        setRent( 4 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Baltic takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Baltic( int house )
{
    setCost( 60 );
    switch( house )
    {
    case 1:
        setRent( 4 );
        break;
    case 2:
        setRent( 20 );
        break;
    case 3:
        setRent( 60 );
        break;
    case 4:
        setRent( 180 );
        break;
    case 5:
        setRent( 320 );
        break;
    case 6:
        setRent( 450 );
        break;
    case 7:
        setRent( 8 ); // owns entire monopoly, but undeveloped
    }
}
}

```

```

/**
 * Method Oriental takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Oriental( int house )
{
    setCost( 100 );
    switch( house )
    {
        case 1:
            setRent( 6 );
            break;
        case 2:
            setRent( 30 );
            break;
        case 3:
            setRent( 90 );
            break;
        case 4:
            setRent( 270 );
            break;
        case 5:
            setRent( 400 );
            break;
        case 6:
            setRent( 550 );
            break;
        case 7:
            setRent( 12 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Vermont takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Vermont( int house )
{
    setCost( 100 );
    switch( house )
    {
        case 1:
            setRent( 6 );
            break;
        case 2:
            setRent( 30 );
            break;
    }
}

```

```

    case 3:
        setRent( 90 );
        break;
    case 4:
        setRent( 270 );
        break;
    case 5:
        setRent( 400 );
        break;
    case 6:
        setRent( 550 );
        break;
    case 7:
        setRent( 12 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Connecticut takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Connecticut( int house )
{
    setCost( 120 );
    switch( house )
    {
    case 1:
        setRent( 8 );
        break;
    case 2:
        setRent( 40 );
        break;
    case 3:
        setRent( 100 );
        break;
    case 4:
        setRent( 300 );
        break;
    case 5:
        setRent( 450 );
        break;
    case 6:
        setRent( 600 );
        break;
    case 7:
        setRent( 16 ); // owns entire monopoly, but undeveloped
    }
}

```

```

/**
 * Method StCharles takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void StCharles( int house )
{
    setCost( 140 );
    switch( house )
    {
        case 1:
            setRent( 10 );
            break;
        case 2:
            setRent( 50 );
            break;
        case 3:
            setRent( 150 );
            break;
        case 4:
            setRent( 450 );
            break;
        case 5:
            setRent( 625 );
            break;
        case 6:
            setRent( 750 );
            break;
        case 7:
            setRent( 20 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method States takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void States( int house )
{
    setCost( 140 );
    switch( house )
    {
        case 1:
            setRent( 10 );
            break;
        case 2:
            setRent( 50 );
            break;
    }
}

```

```

    case 3:
        setRent( 150 );
        break;
    case 4:
        setRent( 450 );
        break;
    case 5:
        setRent( 625 );
        break;
    case 6:
        setRent( 750 );
        break;
    case 7:
        setRent( 20 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Virginia takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Virginia( int house )
{
    setCost( 160 );
    switch( house )
    {
    case 1:
        setRent( 12 );
        break;
    case 2:
        setRent( 60 );
        break;
    case 3:
        setRent( 180 );
        break;
    case 4:
        setRent( 500 );
        break;
    case 5:
        setRent( 700 );
        break;
    case 6:
        setRent( 900 );
        break;
    case 7:
        setRent( 24 ); // owns entire monopoly, but undeveloped
    }
}

```

```

/**
 * Method StJames takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void StJames( int house )
{
    setCost( 180 );
    switch( house )
    {
        case 1:
            setRent( 14 );
            break;
        case 2:
            setRent( 70 );
            break;
        case 3:
            setRent( 200 );
            break;
        case 4:
            setRent( 550 );
            break;
        case 5:
            setRent( 750 );
            break;
        case 6:
            setRent( 950 );
            break;
        case 7:
            setRent( 28 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Tennessee takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Tennessee( int house )
{
    setCost( 180 );
    switch( house )
    {
        case 1:
            setRent( 14 );
            break;
        case 2:
            setRent( 70 );
            break;
    }
}

```

```

case 3:
    setRent( 200 );
    break;
case 4:
    setRent( 550 );
    break;
case 5:
    setRent( 750 );
    break;
case 6:
    setRent( 950 );
    break;
case 7:
    setRent( 28 ); // owns entire monopoly, but undeveloped
}
}

/**
 * Method NewYork takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void NewYork( int house )
{
    setCost( 200 );
    switch( house )
    {
    case 1:
        setRent( 16 );
        break;
    case 2:
        setRent( 80 );
        break;
    case 3:
        setRent( 220 );
        break;
    case 4:
        setRent( 600 );
        break;
    case 5:
        setRent( 800 );
        break;
    case 6:
        setRent( 1000 );
        break;
    case 7:
        setRent( 32 ); // owns entire monopoly, but undeveloped
    }
}
}

```

```

/**
 * Method Kentucky takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Kentucky( int house )
{
    setCost( 220 );
    switch( house )
    {
        case 1:
            setRent( 18 );
            break;
        case 2:
            setRent( 90 );
            break;
        case 3:
            setRent( 250 );
            break;
        case 4:
            setRent( 700 );
            break;
        case 5:
            setRent( 875 );
            break;
        case 6:
            setRent( 1050 );
            break;
        case 7:
            setRent( 36 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Indiana takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Indiana( int house )
{
    setCost( 220 );
    switch( house )
    {
        case 1:
            setRent( 18 );
            break;
        case 2:
            setRent( 90 );
            break;
    }
}

```

```

    case 3:
        setRent( 250 );
        break;
    case 4:
        setRent( 700 );
        break;
    case 5:
        setRent( 875 );
        break;
    case 6:
        setRent( 1050 );
        break;
    case 7:
        setRent( 36 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Illinois takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Illinois( int house )
{
    setCost( 240 );
    switch( house )
    {
        case 1:
            setRent( 20 );
            break;
        case 2:
            setRent( 100 );
            break;
        case 3:
            setRent( 300 );
            break;
        case 4:
            setRent( 750 );
            break;
        case 5:
            setRent( 925 );
            break;
        case 6:
            setRent( 1100 );
            break;
        case 7:
            setRent( 40 ); // owns entire monopoly, but undeveloped
    }
}

```

```

/**
 * Method Atlantic takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Atlantic( int house )
{
    setCost( 260 );
    switch( house )
    {
        case 1:
            setRent( 22 );
            break;
        case 2:
            setRent( 110 );
            break;
        case 3:
            setRent( 330 );
            break;
        case 4:
            setRent( 800 );
            break;
        case 5:
            setRent( 975 );
            break;
        case 6:
            setRent( 1150 );
            break;
        case 7:
            setRent( 44 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Ventnor takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Ventnor( int house )
{
    setCost( 260 );
    switch( house )
    {
        case 1:
            setRent( 22 );
            break;
        case 2:
            setRent( 110 );
            break;
    }
}

```

```

    case 3:
        setRent( 330 );
        break;
    case 4:
        setRent( 800 );
        break;
    case 5:
        setRent( 975 );
        break;
    case 6:
        setRent( 1150 );
        break;
    case 7:
        setRent( 44 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Marvin takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Marvin( int house )
{
    setCost( 280 );
    switch( house )
    {
    case 1:
        setRent( 24 );
        break;
    case 2:
        setRent( 120 );
        break;
    case 3:
        setRent( 360 );
        break;
    case 4:
        setRent( 850 );
        break;
    case 5:
        setRent( 1025 );
        break;
    case 6:
        setRent( 1200 );
        break;
    case 7:
        setRent( 48 ); // owns entire monopoly, but undeveloped
    }
}

```

```

/**
 * Method Pacific takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Pacific( int house )
{
    setCost( 300 );
    switch( house )
    {
        case 1:
            setRent( 26 );
            break;
        case 2:
            setRent( 130 );
            break;
        case 3:
            setRent( 390 );
            break;
        case 4:
            setRent( 900 );
            break;
        case 5:
            setRent( 1100 );
            break;
        case 6:
            setRent( 1275 );
            break;
        case 7:
            setRent( 52 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method NorthCarolina takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void NorthCarolina( int house )
{
    setCost( 300 );
    switch( house )
    {
        case 1:
            setRent( 26 );
            break;
        case 2:
            setRent( 130 );
            break;
    }
}

```

```

    case 3:
        setRent( 390 );
        break;
    case 4:
        setRent( 900 );
        break;
    case 5:
        setRent( 1100 );
        break;
    case 6:
        setRent( 1275 );
        break;
    case 7:
        setRent( 52 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method Pennsylvania takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void Pennsylvania( int house )
{
    setCost( 320 );
    switch( house )
    {
    case 1:
        setRent( 28 );
        break;
    case 2:
        setRent( 150 );
        break;
    case 3:
        setRent( 450 );
        break;
    case 4:
        setRent( 1000 );
        break;
    case 5:
        setRent( 1200 );
        break;
    case 6:
        setRent( 1400 );
        break;
    case 7:
        setRent( 56 ); // owns entire monopoly, but undeveloped
    }
}

```

```

/**
 * Method ParkPlace takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void ParkPlace( int house )
{
    setCost( 350 );
    switch( house )
    {
        case 1:
            setRent( 35 );
            break;
        case 2:
            setRent( 175 );
            break;
        case 3:
            setRent( 500 );
            break;
        case 4:
            setRent( 1100 );
            break;
        case 5:
            setRent( 1300 );
            break;
        case 6:
            setRent( 1500 );
            break;
        case 7:
            setRent( 70 ); // owns entire monopoly, but undeveloped
    }
}

private void Boardwalk( int house )
{
    setCost( 400 );
    switch( house )
    {
        case 1:
            setRent( 50 );
            break;
        case 2:
            setRent( 200 );
            break;
        case 3:
            setRent( 600 );
            break;
        case 4:
            setRent( 1400 );
            break;
    }
}

```

```

    case 5:
        setRent( 1700 );
        break;
    case 6:
        setRent( 2000 );
        break;
    case 7:
        setRent( 100 ); // owns entire monopoly, but undeveloped
    }
}

/**
 * Method ElectricCompany takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void ElectricCompany( int utility )
{
    setCost( 150 );
    switch( utility )
    {
    case 1:
        setRent( 4 );
        break;
    case 2:
        setRent( 10 );
    }
}

private void WaterWorks( int utility )
{
    setCost( 150 );
    switch( utility )
    {
    case 1:
        setRent( 4 );
        break;
    case 2:
        setRent( 10 );
    }
}

private void ReadingRR( int RRs )
{
    setCost( 200 );
    switch( RRs )
    {
    case 1:
        setRent( 25 );
        break;

```

```

    case 2:
        setRent( 50 );
        break;
    case 3:
        setRent( 100 );
        break;
    case 4:
        setRent( 200 );
    }
}

/**
 * Method PennsylvaniaRR takes in
 *
 * @param nHouse as the number of houses, and sets rent and cost according
 * to the property's current state
 */
private void PennsylvaniaRR( int RRs )
{
    setCost( 200 );
    switch( RRs )
    {
    case 1:
        setRent( 25 );
        break;
    case 2:
        setRent( 50 );
        break;
    case 3:
        setRent( 100 );
        break;
    case 4:
        setRent( 200 );
    }
}

/** Method BandORR takes in @param nHouse as the number of houses,
 and sets rent and cost according to the property's current state */
private void BandORR( int RRs )
{
    setCost( 200 );
    switch( RRs )
    {
    case 1:
        setRent( 25 );
        break;
    case 2:
        setRent( 50 );
        break;
    case 3:
        setRent( 100 );

```

```

        break;
    case 4:
        setRent( 200 );
    }
}

private void ShortLineRR( int RRs )
{
    setCost( 200 );
    switch( RRs )
    {
    case 1:
        setRent( 25 );
        break;
    case 2:
        setRent( 50 );
        break;
    case 3:
        setRent( 100 );
        break;
    case 4:
        setRent( 200 );
    }
}

public void setCost( int c )
{
    cost = c;
}

public int getCost()
{
    return cost;
}

public void setRent( int r )
{
    rent = r;
}

public int getRent()
{
    return rent;
}

private int cost, rent;
}

```

```

/*=====
Program / Class: Dossier, RandomAccessUserRecord          Author: Ross Chan
Purpose of Class: Main program that is used to work with RandomAccessFiles.
Date of This Revision: March 28, 2007    Teacher: Gerry Donaldson
School: Sir Winston Churchill High School, Calgary, Alberta, Canada
Text: Horstmann, Cay. BIG JAVA 2nd Edition, 2006 ISBN: 0-471-69703-6
Language: Java J2SE 5.0    Target Operating System: Java Virtual Machine
System: Pentium IV 2.5 GHz running under Windows XP    IDE: Eclipse 3.2
=====*/

```

```

import java.io.*;

/**
 * This program helps create a RandomAccessFile
 * @author Ross Chan
 * @version 1.0
 */
public class RandomAccessUserRecord extends PlayerRecord
{
    // no-argument constructor calls other constructor
    // with default values
    public RandomAccessUserRecord()
    {
        this( 0, "", 0 );
    }

    /**
     * Initializes a RandomAccessHardwareRecord
     * @param record The tool identification number
     * @param toolname The name of the tool
     * @param amount The quantity of the tool
     * @param price The price of the tool
     */
    public RandomAccessUserRecord( int record, String toolname,
        int amount )
    {
        super( record, toolname, amount );
    }

    /**
     * Read a record from a specified RandomAccessFile
     * @param file The given file
     * @throws IOException
     * @pre The file must be a RandomAccessFile
     * @post The RandomAccessFile is read
     */
    public void read( RandomAccessFile file ) throws IOException
    {
        setToken( file.readInt() );
        setName( padName( file ) );
        setScore( file.readInt() );
    }
}

```

```

}

/**
 * Returns the name of the tool to the array
 * @param file The RandomAccessFile class
 * @return The name of the tool
 * @throws IOException
 */
private String padName( RandomAccessFile file ) throws IOException
{
    char name[] = new char[ 12 ], temp;

    for( int count = 0; count < name.length; count++ )
    {
        temp = file.readChar();
        name[ count ] = temp;
    }

    return new String( name ).replace( '\0', ' ' );
}

private void writeName( RandomAccessFile file, String name )
    throws IOException
{
    StringBuffer buffer = null;

    if( name != null )
        buffer = new StringBuffer( name );
    else
        buffer = new StringBuffer( 12 );

    buffer.setLength( 12 );
    file.writeChars( buffer.toString() );
}

public void write( RandomAccessFile file ) throws IOException
{
    file.writeInt( getToken() );
    writeName( file, getName() );
    file.writeInt( getScore() );
}

public int size()
{
    return 32;
}
}

```