package Command;

import State.StateHolder;

/\*\*

\* This command changes the state of the statemachine

\* @author rob

\*

\*/

public class ChangeStateCommand implements Command {

@Override

public void execute() {

StateHolder.next();

}

}

package Command;

/\*\*

\* Abstract that defines how commands work

\* @author rob

\*

\*/

public interface Command {

void execute();

}

package Command;

/\*\*

\* Command for moving selected images to the left

\* @author rob

\*

\*/

public class LeftCommand extends ReindexCommand implements Command{

@Override

public int direction() {

return -1;

}

}

/\*\*

\* Handles reseting the index of the currently selected images

\*/

package Command;

import Director.Director;

import State.\*;

import Study.\*;

public abstract class ReindexCommand implements Command {

@Override

public void execute() {

Study currentStudy = Director.getStudy();

int currentIndex = currentStudy.getIndex();

int newIndex;

if(direction() == -1 && (currentIndex - StateHolder.images()) < 0){

newIndex = 0;

}

else{

newIndex = currentIndex + (StateHolder.images() \* direction());

}

currentStudy.setIndex(newIndex);

}

abstract public int direction();

//This determines the direction the command goes in. -1 is to the left, 1 is to the right

}

package Command;

/\*\*

\* Command for moving selected images to the right

\* @author rob

\*

\*/

public class RightCommand extends ReindexCommand implements Command {

@Override

public int direction() {

return 1;

}

}

package Command;

import Director.Director;

/\*\*

\* Command that forces the Study to save it's state

\* @author rob

\*

\*/

public class SaveStudyCommand implements Command {

@Override

public void execute() {

Director.getStudy().saveState();

}

}

package Director;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import State.StateHolder;

import State.State;

import Study.NoValidStudiesFoundException;

import Study.Study;

import Study.StudyBuilder;

import Study.StudyBuilder.StudyType;

/\*\*

\* Simplifys talking to and stores the currently opened Study

\* @author rob

\*

\*/

public class Director {

private static List<Study> availStudies = null;

private static String root = "";

private static Study study = null;

/\*\*

\* Gets the current study

\* @return The current study loaded in

\*/

public static Study getStudy(){

return study;

}

/\*\*

\* Sets the study to a new value

\* @param newStudy the new study to replace the current one with

\*/

public static void setStudy(Study newStudy){

study = newStudy;

}

/\*\*

\* Returns a list of images to load into the gui based on the currentstate

\* @return the list of images to load

\*/

public static List<String> getImages(){

List<String> images = null;

if(StateHolder.images() == 1){

images = study.getImgAddresses().subList(study.getIndex(), (study.getIndex() + StateHolder.images()));

}

else if(StateHolder.images() == 4){

if((study.getIndex() + 4) > (study.getImgAddresses().size()-1)){

images = study.getImgAddresses().subList(study.getIndex(), (study.getImgAddresses().size()));

}

else{

images = study.getImgAddresses().subList(study.getIndex(), (study.getIndex() + StateHolder.images()));

}

}

return images;

}

/\*\*

\* Set the root search directory to a different path

\* @param newRoot new Root path

\*/

public static void setRoot(String newRoot){

root = newRoot;

}

/\*\*

\* Gets all of the valid directories that are studies

\* @return All valid directories

\* @throws NoValidStudiesFoundException

\*/

public static List<String> getAvailStudies() throws NoValidStudiesFoundException{

availStudies = Arrays.asList(StudyBuilder.getAvailableStudies(root, StudyType.local));

List<String> stringStudies = new ArrayList<String>();

for(Study curr: availStudies){

stringStudies.add(curr.getMyPath());

}

return stringStudies;

}

/\*\*

\* Chooses a study based on an index of the getAvailStudies

\* @param Index The index of the chosen study

\*/

public static void choseStudy(int Index){

study = availStudies.get(Index);

State s = StudyBuilder.readState(study.getMyPath());

if (s == null) {

StateHolder.empty();

StateHolder.next();

}

else {

while (StateHolder.images() != s.images()) {

StateHolder.next();

}

}

}

/\*\*

\* Indicates if there is anything to the "left"

\* @return boolean that indicates whether it's possible to move to the left

\*/

public static boolean isLeft(){

if(study == null){

return false;

}

int currentIndex = study.getIndex();

int step = StateHolder.images();

return !((currentIndex - 1) < 0);

}

/\*\*

\* Indicates if there is anything to the "right"

\* @return boolean that indicates whether it's possible to move to the right

\*/

public static boolean isRight(){

if(study == null){

return false;

}

int currentIndex = study.getIndex();

int step = StateHolder.images();

int maxIndex = (study.getImgAddresses().size() - 1);

return !((currentIndex + step) > maxIndex);

}

package State;

/\*\*

\* A UI state that displays four images.

\* @author msd7734

\*

\*/

public class FourState implements State {

@Override

public int images() {

return 4;

}

}

package State;

/\*\*

\* A UI state which displays one image.

\* @author msd7734

\*

\*/

public class OneState implements State {

@Override

public int images() {

return 1;

}

}

package State;

/\*\*

\* An interface to define a UI state. The purpose of this state

\* is to describe how many images the UI should expect to display

\* at once.

\* @author msd7734

\*

\*/

public interface State {

/\*\*

\* How many images the UI should expect to display at once.

\* @return An int - the number of images

\*/

public int images();

}

package State;

/\*\*

\* A class that encapsulates UI State operations.

\* @author msd7734

\*

\*/

public class StateHolder {

private static State currentState = new ZeroState();

/\*\*

\* Cycle the current state to the next logical state.

\*/

public static void next(){

if(currentState.images() == 0){

//Switch to 1

currentState = new OneState();

}else if(currentState.images() == 1){

//switch to 4

currentState = new FourState();

}else if(currentState.images() == 4){

//switch to 1

currentState = new OneState();

}

}

/\*\*

\* Force the state to display 4 images.

\*/

public static void setFour(){

currentState = new FourState();

}

/\*\*

\* Wrap the images() method of State

\* @return Return the number of images the state wants to display

\*/

public static int images(){

return currentState.images();

}

/\*\*

\* Force the state to display an empty state.

\*/

public static void empty(){

currentState = new ZeroState();

}

}

package State;

/\*\*

\* A UI State which is used when there are no images to display.

\* @author msd7734

\*

\*/

public class ZeroState implements State {

@Override

public int images() {

return 0;

}

}

package Study;

import State.\*;

import java.util.List;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileStudy implements Study {

private List<String> imgAddresses;

private String name;

private String myPath;

private int curIndex;

private int bufferSize;

public FileStudy(List<String> imgAddresses, String name, String myPath, int startIndex) {

this.imgAddresses = imgAddresses;

this.myPath = myPath;

this.name = name;

this.curIndex = startIndex;

//something may have to intervene to help set this properly

bufferSize = 1;

}

@Override

public List<String> getImgAddresses() {

return imgAddresses;

}

@Override

public String[] getCurImgAddress() {

return imgAddresses.subList(curIndex, curIndex + bufferSize)

.toArray(new String[]{});

}

@Override

public void saveState() {

System.out.println("Beginning to save state in dir: " + myPath);

try {

File save = new File(myPath + File.separator + "0.sav");

if (!save.exists()) {

System.out.println("No save file found, creating new.");

save.createNewFile();

}

FileOutputStream out = new FileOutputStream(save, false);

byte[] indexEntry = new String("index=" +

this.curIndex + "\n")

.getBytes();

System.out.println(new String(indexEntry));

byte[] stateEntry = new String("state=" +

stateToString(StateHolder.images()) + "\n")

.getBytes();

System.out.println(new String(stateEntry));

out.write(indexEntry);

out.write(stateEntry);

System.out.println("Save file written.");

out.close();

}

catch (IOException ioe) {

System.err.println("IOException in saveState()");

}

}

@Override

public void saveState(int index) {

//because we don't know if the Study should manage its index yet,

//take an index int in case it's handled externally

try {

File save = new File(myPath + File.separator + "0.sav");

if (!save.exists()) {

save.createNewFile();

}

FileOutputStream out = new FileOutputStream(save, false);

//don't forget to implement reading ZeroState in studybuilder

byte[] indexEntry = new String("index=" +

index + "\n")

.getBytes();

byte[] stateEntry = new String("state=" +

stateToString(StateHolder.images()) + "\n")

.getBytes();

out.write(indexEntry);

out.write(stateEntry);

out.close();

}

catch (IOException ioe) {

System.err.println("IOException in saveState()");

}

}

private String stateToString(int imgs) {

if (imgs == 1) {

return "one";

}

else if (imgs == 4) {

return "four";

}

else {

return "zero";

}

}

@Override

public String getName() {

return name;

}

@Override

public String getMyPath() {

return myPath;

}

@Override

public int getIndex() {

return curIndex;

}

@Override

public void setIndex(int newIndex) {

if (newIndex + (bufferSize-1) >= imgAddresses.size()) {

System.err.println("Attempted to set study current image index out of bounds.");

throw new IndexOutOfBoundsException();

}

else {

curIndex = newIndex;

}

}

}

package Study;

/\*\*

\* An Exception to throw when a StudyBuilder is given a root directory

\* from which no valid Studies can be built.

\* @author msd7734

\*

\*/

public class NoValidStudiesFoundException extends Exception {

NoValidStudiesFoundException(String dir) {

super("No valid studies could be found in the root directory " + dir);

}

}

package Study;

import java.util.List;

/\*\*

\* A shell for an implementation of a Study that is built from

\* a remote address.

\* @author msd7734

\*

\*/

public class RemoteStudy implements Study {

public RemoteStudy() { }

@Override

public List<String> getImgAddresses() {

return null;

}

@Override

public String[] getCurImgAddress() {

return null;

}

@Override

public int getIndex() {

// TODO Auto-generated method stub

return 0;

}

@Override

public void setIndex(int newIndex) {

// TODO Auto-generated method stub

}

@Override

public String getName() {

// TODO Auto-generated method stub

return null;

}

@Override

public String getMyPath() {

return null;

}

@Override

public void saveState() {

return;

}

@Override

public void saveState(int index) {

return;

}

}

package Study;

import java.util.List;

/\*\*

\* Defines the methods common to local and remote Study objects.

\* A study is mean to hold a collection of addresses to allow

\* a UI to load images. Also provides a way to save its settings

\* to a .sav file.

\* @author msd7734

\*

\*/

public interface Study {

/\*\*

\* Get a collection of Strings representing absolute pathnames to .jpg

\* images in a study.

\* @return List of Strings

\*/

public List<String> getImgAddresses();

/\*\*

\* Get the absolute path for the current image(s) being looked at

\* @return

\*/

public String[] getCurImgAddress();

/\*\*

\* Get the name of this study (usually the directory name)

\* @return Name String

\*/

public String getName();

/\*\*

\* Return the location of this Study

\* @return A String address

\*/

public String getMyPath();

/\*\*

\* Save this Study's information on current image address and

\* the display state of the application.

\*/

public void saveState();

public void saveState(int index);

/\*\*

\* Get the index of the current image to display

\* @return An integer index

\*/

public int getIndex();

/\*\*

\* Set the index of the current image to display

\* @param newIndex The new index

\*/

public void setIndex(int newIndex);

}

package Study;

/\*\*

\* StudyBuilder.java

\*

\* @author Matthew Dennis

\*

\* Responsible for constructing a list of valid Study objects from a given address,

\* local or remote. For this project, only local studies are fully accessible. The

\* CmdBuilder or some other intermediary should be responsible for managing the

\* returned list of Studies and exposing what is necessary.

\*

\* 4/29/13 Matt Dennis

\* - Initial implementation

\*

\*/

import State.\*;

import java.util.List;

import java.util.ArrayList;

import java.util.Arrays;

import java.io.File;

import java.io.FilenameFilter;

import java.util.Scanner;

import java.io.FileNotFoundException;

import java.util.regex.\*;

public class StudyBuilder {

public enum StudyType { remote, local; }

private StudyBuilder() { }

/\*\*

\* Find all the studies at a given address.

\* @param rootDir Some String for accessing a directory or remote address.

\* @param studyType Determines the kind of study, local or remote, that is returned

\* @return An array of Study objects

\* @throws NoValidStudiesFoundException If no valid Study objects can be built

\* at the given address

\*/

public static Study[] getAvailableStudies(String rootDir, StudyType studyType)

throws NoValidStudiesFoundException {

if (! new File(rootDir).exists())

throw new NoValidStudiesFoundException(rootDir);

if (studyType == StudyType.remote) {

//dummy object to pay lip service to concept of remote study

return new RemoteStudy[]{};

}

else if (studyType == StudyType.local) {

return findLocalStudies(rootDir, studyType);

}

else {

//default behavior

//return empty local study

return new FileStudy[]{};

}

}

/\*\*

\* Given an address, read the state of a Study from a .sav file

\* @param studyDir An address where a 0.sav can be found and read

\* @return The UI State read

\*/

public static State readState(String studyDir) {

File f = new File(studyDir + File.separator + "0.sav");

try {

Scanner sc = new Scanner(f);

while (sc.hasNextLine()) {

String[] pair = sc.nextLine().split("=");

if (pair[0].equals("state")) {

if (pair[1].equals("one")) {

return new OneState();

}

else if (pair[1].equals("four")) {

return new FourState();

}

else if(pair[1].equals("zero")) {

return new ZeroState();

}

else {

System.err.println("Invalid state read.");

return null;

}

}

}

sc.close();

}

catch(FileNotFoundException fnfe) {

System.out.println("No save file found.");

return null;

}

System.err.println("Found no state information in study .sav file");

return null;

}

private static FileStudy[] findLocalStudies(String rootDir, StudyType studyType)

throws NoValidStudiesFoundException {

//This should probably return paths, not FileStudys. If it's given to the UI

//we don't want the UI to be dealing with a Study object

//-Rob

ArrayList<FileStudy> studies = new ArrayList<FileStudy>();

File root = new File(rootDir);

//For each directory in root

for (File f : root.listFiles()) {

if (f.isDirectory()) {

File studyDir = new File(f.getPath());

String[] jpgs = studyDir.list(

new FilenameFilter() {

@Override

public boolean accept(File dir, String name) {

Pattern p = Pattern.compile(".+\\.jpg");

Matcher m = p.matcher(name);

return m.matches();

}

}

);

File save = new File(studyDir.getPath() + File.separator + "0.sav");

int studyStart = 0;

//Format:

//name=val

try {

Scanner sc = new Scanner(save);

while (sc.hasNextLine()) {

String[] pair = sc.nextLine().split("=");

if (pair[0].equals("index")) {

studyStart = Integer.parseInt(pair[1]);

}

}

sc.close();

}

catch(FileNotFoundException fnfe) {

studyStart = 0;

}

catch(NumberFormatException nfe) {

studyStart = 0;

}

if (jpgs == null) {

//no jpgs found

}

else if (jpgs.length > 0) {

studies.add(new FileStudy(

Arrays.asList( getAbsolutePaths(studyDir, jpgs) ),

studyDir.getName(),

studyDir.getAbsolutePath(),

studyStart

));

}

}

}

//we have no studies, up to the caller to decide what to do

if (studies.size() == 0)

throw new NoValidStudiesFoundException(rootDir);

return studies.toArray(new FileStudy[]{});

}

/\*\*

\* Gets the path as "absolutely" as the original root directory you gave to

\* this StudyBuilder. Meant to be called on the .jpgs of a study.

\* @param dir The directory of the Study

\* @param paths Path names to Study images

\* @return An array of path strings as a copy of paths, but with "absolute"

\* paths

\*/

private static String[] getAbsolutePaths(File dir, String[] paths) {

String[] result = new String[paths.length];

for (int i=0; i<paths.length; i++) {

result[i] = dir.getAbsolutePath() + File.separator + paths[i];

//result[i] = new File(paths[i]).getAbsolutePath();

}

return result;

}

}

package View;

import Command.ChangeStateCommand;

import Command.LeftCommand;

import Command.RightCommand;

import Command.SaveStudyCommand;

import Director.Director;

import State.StateHolder;

import Study.NoValidStudiesFoundException;

import java.awt.BorderLayout;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.BorderFactory;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextPane;

import javax.swing.SwingUtilities;

import javax.swing.UIManager;

import javax.swing.border.Border;

import javax.swing.plaf.basic.BasicArrowButton;

import javax.swing.text.DefaultStyledDocument;

import javax.swing.text.MutableAttributeSet;

import javax.swing.text.StyleConstants;

import javax.swing.text.StyleContext;

import javax.swing.text.StyledDocument;

public class Frame extends JFrame {

private JPanel mainLayout;

private JPanel centerScreen;

private JMenuItem fourTile;

private JMenuItem singleTile;

private BasicArrowButton rightArrow;

private BasicArrowButton leftArrow;

private int curMode;

private List<String> images;

private final JFileChooser fc = new JFileChooser();

private LeftCommand left;

private RightCommand right;

private ChangeStateCommand changeState;

private SaveStudyCommand saveStudy;

private JList<Object> listOfStudies;

public Frame()

{

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch(Exception e) {

System.out.println("Error setting native LAF: " + e);

}

System.setProperty("apple.laf.useScreenMenuBar", "true");

fc.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

fc.setAcceptAllFileFilterUsed(false);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

mainLayout = new JPanel(new BorderLayout(20, 20));

curMode = 1;

listOfStudies = null;

left = new LeftCommand();

right = new RightCommand();

changeState = new ChangeStateCommand();

saveStudy = new SaveStudyCommand();

setSize(900, 600);

buildMenuBar();

setResizable(false);

startUpScreen();

add(mainLayout);

setVisible(true);

}

public void availableStudies()

{

final JFrame test = new JFrame();

test.setSize(500, 500);

test.setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

try {

listOfStudies = new JList<Object>(Director.getAvailStudies().toArray());

} catch (NoValidStudiesFoundException e1) {

System.err.println("No Available Studies");

JFrame errorFrame = new JFrame();

JOptionPane.showMessageDialog(errorFrame, "No Available Studies, please select a different directory");

return;

}

JScrollPane listScroller = new JScrollPane(listOfStudies);

JPanel availableStudyFrame = new JPanel(new BorderLayout());

JPanel buttonFlow = new JPanel(new FlowLayout());

JButton select = new JButton("Select");

JButton cancel = new JButton("Cancel");

buttonFlow.add(select);

buttonFlow.add(cancel);

availableStudyFrame.add(buttonFlow, BorderLayout.SOUTH);

availableStudyFrame.add(listScroller);

test.add(availableStudyFrame);

select.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

int index = listOfStudies.getSelectedIndex();

Director.choseStudy(index);

if(StateHolder.images() == 4)

{

fourTileMode();

}

else

{

singleTileMode();

}

images = Director.getImages();

fillScreen(images);

test.dispose();

}

});

cancel.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

test.dispose();

}

});

test.setVisible(true);

}

public void buildMenuBar()

{

//create the menu bar

JMenuBar menu = new JMenuBar();

//create menu bar items

JMenuItem file = new JMenu("File");

JMenuItem view = new JMenu("View");

JMenuItem info = new JMenu("Info");

//create file menu items

JMenuItem selectDirectory = new JMenuItem("Select Directory");

JMenuItem openStudy = new JMenuItem("Open Study");

JMenuItem save = new JMenuItem("Save");

JMenuItem saveAs = new JMenuItem("Save As");

JMenuItem close = new JMenuItem("Close Study");

JMenuItem exit = new JMenuItem("Exit");

saveAs.setEnabled(false);

//create view menu items

fourTile = new JMenuItem("Four Tile Mode");

singleTile = new JMenuItem("Single Tile Mode");

//create info menu items

JMenuItem about = new JMenuItem("About");

// add to the file menu

file.add(selectDirectory);

file.add(openStudy);

file.add(save);

file.add(saveAs);

file.add(close);

file.add(exit);

//add to the view menu

view.add(fourTile);

view.add(singleTile);

//add to the info menu

info.add(about);

// add action listeners

// file menu

selectDirectory.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

//open();

if (fc.showOpenDialog(null) == JFileChooser.APPROVE\_OPTION) {

Director.setRoot(fc.getSelectedFile().getAbsolutePath());

availableStudies();

}

}

});

openStudy.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

availableStudies();

}

});

save.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

saveStudy.execute();

}

});

saveAs.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

}

});

close.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

promptSave(0);

}

});

exit.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

promptSave(1);

}

});

// view menu

fourTile.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

changeState.execute();

curMode = StateHolder.images();

images = Director.getImages();

System.out.println(images.size());

fourTileMode();

fillScreen(images);

}

});

singleTile.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

changeState.execute();

curMode = StateHolder.images();

images = Director.getImages();

singleTileMode();

fillScreen(images);

}

});

// info menu

about.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

about();

}

});

//add to the menu bar

menu.add(file);

menu.add(view);

menu.add(info);

//set the menu bar for the frame

this.setJMenuBar(menu);

}

public void startUpScreen()

{

mainLayout.removeAll();

// layouts for the startup screen which is empty

centerScreen = new JPanel(new GridLayout(1,1));

// JPanel topScreen = new JPanel(new FlowLayout());

JPanel leftScreen = new JPanel(new GridLayout(3,1));

JPanel rightScreen = new JPanel(new GridLayout(3,1));

JPanel bottomScreen = new JPanel(new FlowLayout());

// arrow Buttons

BasicArrowButton rightArrow = new BasicArrowButton(BasicArrowButton.EAST);

BasicArrowButton leftArrow = new BasicArrowButton(BasicArrowButton.WEST);

// action listeners

//left and right button padding

Border leftPadding = BorderFactory.createEmptyBorder(0, 5, 0, 0);

leftScreen.setBorder(leftPadding);

Border rightPadding = BorderFactory.createEmptyBorder(0, 0, 0, 5);

rightScreen.setBorder(rightPadding);

// enable left and right buttons

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

// add to layouts

leftScreen.add(new JLabel(""));

leftScreen.add(leftArrow);

rightScreen.add(new JLabel(""));

rightScreen.add(rightArrow);

// mainLayout.add(topScreen, BorderLayout.NORTH);

mainLayout.add(centerScreen, BorderLayout.CENTER);

mainLayout.add(bottomScreen, BorderLayout.SOUTH);

mainLayout.add(rightScreen, BorderLayout.EAST);

mainLayout.add(leftScreen, BorderLayout.WEST);

//disable menu item

fourTile.setEnabled(false);

singleTile.setEnabled(false);

SwingUtilities.updateComponentTreeUI(this);

}

public void singleTileMode()

{

mainLayout.removeAll();

// layouts for the startup screen which is empty

centerScreen = new JPanel(new GridLayout(1,1));

// JPanel topScreen = new JPanel(new FlowLayout());

JPanel leftScreen = new JPanel(new GridLayout(3,1));

JPanel rightScreen = new JPanel(new GridLayout(3,1));

JPanel bottomScreen = new JPanel(new FlowLayout());

// arrow Buttons

rightArrow = new BasicArrowButton(BasicArrowButton.EAST);

leftArrow = new BasicArrowButton(BasicArrowButton.WEST);

// action listeners

rightArrow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

right.execute();

images = Director.getImages();

fillScreen(images);

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

}

});

leftArrow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

left.execute();

images = Director.getImages();

fillScreen(images);

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

}

});

//left and right button padding

Border leftPadding = BorderFactory.createEmptyBorder(0, 5, 0, 0);

leftScreen.setBorder(leftPadding);

Border rightPadding = BorderFactory.createEmptyBorder(0, 0, 0, 5);

rightScreen.setBorder(rightPadding);

// enable left and right buttons

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

// add to layouts

leftScreen.add(new JLabel(""));

leftScreen.add(leftArrow);

rightScreen.add(new JLabel(""));

rightScreen.add(rightArrow);

// mainLayout.add(topScreen, BorderLayout.NORTH);

mainLayout.add(centerScreen, BorderLayout.CENTER);

mainLayout.add(bottomScreen, BorderLayout.SOUTH);

mainLayout.add(rightScreen, BorderLayout.EAST);

mainLayout.add(leftScreen, BorderLayout.WEST);

//disable menu item

fourTile.setEnabled(true);

singleTile.setEnabled(false);

SwingUtilities.updateComponentTreeUI(this);

}

public void fourTileMode()

{

mainLayout.removeAll();

// layouts for the startup screen which is empty

centerScreen = new JPanel(new GridLayout(2,2, 10, 10));

JPanel leftScreen = new JPanel(new GridLayout(3,1));

JPanel rightScreen = new JPanel(new GridLayout(3,1));

JPanel bottomScreen = new JPanel(new FlowLayout());

// arrow Buttons

rightArrow = new BasicArrowButton(BasicArrowButton.EAST);

leftArrow = new BasicArrowButton(BasicArrowButton.WEST);

// action listeners

//left and right button padding

Border leftPadding = BorderFactory.createEmptyBorder(0, 5, 0, 0);

leftScreen.setBorder(leftPadding);

Border rightPadding = BorderFactory.createEmptyBorder(0, 0, 0, 5);

rightScreen.setBorder(rightPadding);

// enable left and right buttons

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

leftScreen.add(new JLabel(""));

leftScreen.add(leftArrow);

rightScreen.add(new JLabel(""));

rightScreen.add(rightArrow);

rightArrow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

right.execute();

images = Director.getImages();

fillScreen(images);

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

}

});

leftArrow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

left.execute();

images = Director.getImages();

fillScreen(images);

rightArrow.setEnabled(Director.isRight());

leftArrow.setEnabled(Director.isLeft());

}

});

// mainLayout.add(topScreen, BorderLayout.NORTH);

mainLayout.add(centerScreen, BorderLayout.CENTER);

mainLayout.add(bottomScreen, BorderLayout.SOUTH);

mainLayout.add(rightScreen, BorderLayout.EAST);

mainLayout.add(leftScreen, BorderLayout.WEST);

//disable menu button

fourTile.setEnabled(false);

singleTile.setEnabled(true);

SwingUtilities.updateComponentTreeUI(this);

}

public void promptSave(final int flag)

{

final JFrame savePrompt = new JFrame();

savePrompt.setSize(350,100);

savePrompt.setResizable(false);

JPanel promptLayout = new JPanel(new BorderLayout());

JPanel centerFlow = new JPanel(new FlowLayout());

JPanel buttonFlow = new JPanel(new FlowLayout());

JLabel notSaved = new JLabel("Warning! Your current state is not saved");

JButton closeAnyway = new JButton("Close Anyway");

JButton saveNow = new JButton("Save");

JButton cancelNow = new JButton("Cancel");

centerFlow.add(notSaved);

buttonFlow.add(closeAnyway);

buttonFlow.add(saveNow);

buttonFlow.add(cancelNow);

promptLayout.add(centerFlow, BorderLayout.CENTER);

promptLayout.add(buttonFlow, BorderLayout.SOUTH);

savePrompt.add(promptLayout);

closeAnyway.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

savePrompt.dispose();

if(flag == 0)

{

startUpScreen();

}

else

{

System.exit(EXIT\_ON\_CLOSE);

}

}

});

saveNow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

saveStudy.execute();

if(flag == 0)

{

startUpScreen();

}

else

{

System.exit(EXIT\_ON\_CLOSE);

}

savePrompt.dispose();

}

});

cancelNow.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

savePrompt.dispose();

}

});

savePrompt.setVisible(true);

}

public JLabel openImage(String filePath)

{

JLabel icon = new JLabel(new ImageIcon(filePath));

return icon;

}

public void fillScreen(List<String> listOfImages)

{

centerScreen.removeAll();

for(String paths: listOfImages)

{

centerScreen.add(openImage(paths));

}

SwingUtilities.updateComponentTreeUI(this);

}

public void about()

{

final JFrame aboutFrame = new JFrame();

aboutFrame.setSize(300, 200);

aboutFrame.setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

JPanel aboutLayout = new JPanel(new BorderLayout());

JPanel aboutInfo = new JPanel(new FlowLayout());

JPanel aboutButtons = new JPanel(new FlowLayout());

JButton cool = new JButton("Thats Awesome!");

StyledDocument document = new DefaultStyledDocument();

MutableAttributeSet defaultStyle = document.getStyle(StyleContext.DEFAULT\_STYLE);

StyleConstants.setAlignment(defaultStyle, StyleConstants.ALIGN\_CENTER);

JTextPane info = new JTextPane(document);

info.setEditable(false);

info.setText("\n First Implementation of the \n " +

"Medical Image Viewing Console \n" +

" by Team Petulant-Batman");

info.setSize(200, 200);

aboutButtons.add(cool);

aboutInfo.add(info);

aboutLayout.add(aboutButtons, BorderLayout.SOUTH);

aboutLayout.add(info, BorderLayout.CENTER);

cool.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

aboutFrame.dispose();

}

});

aboutFrame.add(aboutLayout);

aboutFrame.setVisible(true);

}

public static void main(String [] args)

{

Frame test = new Frame();

}

}