 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №5**

з дисципліни «Крос-платформне програмування»

на тему: " Асинхронне виконання дій. Малювання на полотні. Візуалізація медичних даних.

Варіант №12

**Виконав:**

студент гр. БС-81

Погребенко В.О.

**Перевірив:**

ас. каф. БМК Рисін С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2021

**Ціль**

Ціль – уяснити принципи дійсної та псевдопаралельності процесів та можливості організації асинхронної організації дій; усвідомити принципи відображення графічної інформації на двовимірному просторі; ознайомитися з представленням зображень у растровому та векторному вигляді; навчитися використовувати засоби для малювання на полотні та виконання асинхронних процесів, розібратися з використанням класів BufferedImage, Canvas, Timer та TimerTask; получити уявлення про стандарт DICOM та інформацію, з якою він дозволяє працювати; організувати показ відеоряду зображень з набору DICOM шляхом періодичного відображення послідовних кадрів.

**Завдання:**

1. Отримати з віддаленного сервера перелік файлів DICOM у наборі згідно з варіантом.

2. Завантажити з файлів наданого набору зображення з використанням класу BufferedImage та їх метадані.

2.1. Оскільки процес виборки файлів з зображеннями потребує деякий час, організувати завантаження таким чином, щоб при читанні набору відображалась інформація про файл, що завантажується в даний момент.

2.2. Для уникнення колізій на час, доки відбувається завантаження набору даних, інтерфейс користувача має бути заблоковано.

3. Відсортувати зображення по положенню зрізів, на яких вони отримані, та послідовності їх отримання.

4. Відобразити відсортований список всіх зрізів набору з вказанням положення кожного зрізу та кількості зображень на ньому.

5. При виборі довільного зрізу зі списку забезпечити показ зображень з цього зрізу з використанням класу Canvas.

6. Організувати показ відеоряду шляхом послідовноі зміни зображень вибраного зрізу, як його кадрів, з використанням класів Timer та TimerTask.

**Лістинг:**

**Весь код можна побачити на:**

https://github.com/PogrebenkoBS81/cross\_platform/tree/main/lab5

# **Package “pogrebenko.labfive.model”:**

## **DICOMImage.java:**

@Override  
public int compareTo(DICOMImage other) {  
 return Comparator.*comparing*((DICOMImage toCompare) -> toCompare.SliceLocation, Comparator.*reverseOrder*())  
 .thenComparingInt((DICOMImage toCompare) -> toCompare.SeriesNumber)  
 .thenComparingInt((DICOMImage toCompare) -> toCompare.AcquNumber)  
 .thenComparingInt((DICOMImage toCompare) -> toCompare.InstNumber)  
 .compare(this, other);  
}

## **DICOMImageSlicer.java:**

package pogrebenko.labfive.model;  
  
import pogrebenko.loggerwrapper.LoggerWrapper;  
  
import java.io.File;  
import java.util.\*;  
import java.util.logging.Logger;  
  
public class DICOMImageSlicer {  
 private static final Logger *LOGGER* = LoggerWrapper.*getLogger*();  
 private Map<Double, List<String>> slices = new HashMap<>();  
  
 public DICOMImageSlicer(List<String> DICOMFilePaths) throws Exception {  
 runSlicing(DICOMFilePaths);  
 }  
  
 private void runSlicing(List<String> DICOMFilePaths) throws Exception {  
 // Slices {"location" : [<file path>, <file path>]}  
 // Images loaded into memory only when they are required.  
 *LOGGER*.info("Creating images slices from listed files...");  
 Map<Double, List<String>> newSlices = new HashMap<>();  
 // Create slices by Slice location.  
 for (String filepath : DICOMFilePaths) {  
 DICOMImage newImage = getDICOMFromPath(filepath);  
  
 if (!newSlices.containsKey(newImage.SliceLocation)) {  
 newSlices.put(newImage.SliceLocation, new ArrayList<>());  
 }  
  
 newSlices.get(newImage.SliceLocation).add(filepath);  
 }  
  
 slices = newSlices;  
 *LOGGER*.info("Slices created: " + slices.size());  
 }  
  
 private DICOMImage getDICOMFromPath(String filepath) throws Exception {  
 *LOGGER*.finest("Retrieving DICOMImage from file: " + filepath);  
 File DICOMFile = new File(filepath);  
  
 String fileDir = DICOMFile.getAbsoluteFile().getParent();  
 String fileName = DICOMFile.getName();  
  
 return new DICOMImage(fileDir, fileName);  
 }  
  
 public Map<Double, List<String>> getSlices() {  
 *LOGGER*.finest("Retrieving slices... ");  
  
 return slices;  
 }  
  
 public List<DICOMImage> getSliceByLocation(Double sliceLocation) throws Exception {  
 *LOGGER*.info("Retrieving slice by its location... ");  
 List<String> slicePaths = slices.get(sliceLocation);  
 List<DICOMImage> sliceImages = new ArrayList<>();  
  
 for (String imagePath : slicePaths) {  
 sliceImages.add(getDICOMFromPath(imagePath));  
 }  
 // Sort images, so animation will be in the correct order.  
 Collections.*sort*(sliceImages);  
  
 return sliceImages;  
 }  
}

## **FTPClientLoader.java:**

package pogrebenko.labfive.model;  
  
import org.apache.commons.net.ftp.FTPClient;  
import org.apache.commons.net.ftp.FTPFile;  
import pogrebenko.loggerwrapper.LoggerWrapper;  
  
import java.io.Closeable;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.util.Arrays;  
import java.util.List;  
import java.util.logging.Logger;  
import java.util.stream.Collectors;  
  
public class FTPClientLoader implements Closeable {  
 private static final Logger *LOGGER* = LoggerWrapper.*getLogger*();  
 private FTPClient ftpClient;  
  
 public FTPClientLoader(String host, int port, String username, String pass) throws IOException {  
 setFTPConnection(host, port, pass, username);  
 }  
  
 public void setFTPConnection(String host, int port, String pass, String username) throws IOException {  
 *LOGGER*.info(  
 String.*format*("Creating ftp connection, host: %s; port: %s; username: %s", host, port, username)  
 );  
  
 ftpClient = new FTPClient();  
 ftpClient.connect(host, port);  
 ftpClient.login(username, pass);  
 // All connections are established by client.  
 ftpClient.enterLocalPassiveMode();  
 // Ensure connection to stay alive.  
 ftpClient.sendNoOp();  
 }  
  
 public List<String> listDirectoryFiles(String hostPath) throws IOException {  
 *LOGGER*.info("Listing files on host machine via FTP, host dir to list: " + hostPath);  
 List<String> listedFiles = Arrays.*stream*(ftpClient.listFiles(hostPath))  
 .filter(FTPFile::isFile)  
 .map(FTPFile::getName)  
 .collect(Collectors.*toList*());  
 *LOGGER*.info("Number of listed files on host machine: " + listedFiles.size());  
  
 return listedFiles;  
 }  
  
 public boolean downloadFile(String hostFile, String toSave, boolean isOverwrite) throws IOException {  
 *LOGGER*.finest(String.*format*(  
 "Downloading file '%s' via FTP; save location: '%s'; file overwrite: %b ",  
 hostFile,  
 toSave,  
 isOverwrite  
 )  
 );  
 File downloadedFile = new File(toSave);  
  
 if (downloadedFile.createNewFile() || isOverwrite) {  
 ftpClient.retrieveFile(hostFile, new FileOutputStream(toSave, false));  
  
 return true;  
 }  
  
 *LOGGER*.finest("File was not created: " + toSave);  
  
 return false;  
 }  
  
 public void close() throws IOException {  
 *LOGGER*.info("Closing ftp connection...");  
  
 if (!ftpClient.logout()) {  
 *LOGGER*.severe("Error during FTP logout!");  
 }  
  
 ftpClient.disconnect();  
 }  
}

# **Package “pogrebenko.labfive.controller”:**

## **Main.java:**

package pogrebenko.labfive.controller;  
  
import javafx.application.Application;  
import javafx.stage.Stage;  
import pogrebenko.labfive.util.JavaFxUtil;  
import pogrebenko.labfive.view.ViewRoutes;  
import pogrebenko.loggerwrapper.LoggerWrapper;  
  
import java.util.logging.Level;  
  
// *TODO: Refactor code. It was, as usual, written in a big hurry,*// *so, no architecture design were made.*public class Main extends Application {  
 private static final String *APP\_NAME* = "DICOM Viewer";  
 public static final String *MAIN\_SCREEN\_TITLE* = *APP\_NAME* + ": Main";  
 public static final String *INIT\_SCREEN\_TITLE* = *APP\_NAME* + ": Initialization";  
  
 // Initialize common logger.  
 static {  
 LoggerWrapper LOGGER\_WRAPPER = LoggerWrapper.*getWrapper*();  
 LOGGER\_WRAPPER.setLoggingLevel(Level.*INFO*);  
 LOGGER\_WRAPPER.setLogFormat();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 */\*\*  
 \* JavaFX entry method.  
 \*  
 \** ***@param*** *primaryStage - initial stage.  
 \*/* @Override  
 public void start(Stage primaryStage) throws Exception {  
 primaryStage.hide(); // Hide given window.  
 // And open initialization and log window.  
 // Alternatively i could just change scene of the primaryStage,  
 // But this approach with the new windows looks a little bit nicer.  
 JavaFxUtil.*openCleanWindow*(ViewRoutes.*APP\_INIT\_SCREEN*, *INIT\_SCREEN\_TITLE*);  
 }  
}

# **Package “pogrebenko.labfive.controller.initialization”:**

## **InitializationController.java:**

package pogrebenko.labfive.controller.initialization;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.control.Alert;  
import javafx.scene.control.ButtonType;  
import javafx.scene.control.TextField;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
import pogrebenko.labfive.controller.Main;  
import pogrebenko.labfive.controller.main.MainController;  
import pogrebenko.labfive.util.CommonUtil;  
import pogrebenko.labfive.util.JavaFxUtil;  
import pogrebenko.labfive.view.ViewRoutes;  
import pogrebenko.loggerwrapper.LoggerWrapper;  
  
import java.io.File;  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class InitializationController {  
 private static final Logger *LOGGER* = LoggerWrapper.*getLogger*();  
  
 // Settings for log file browse window.  
 private final static String[] *extensionsLogFile* = {"\*.txt", "\*.log"};  
 private final static String *filterLabelLogFile* = "Choose or create \*.txt or \*.log file";  
  
  
 @FXML // ResourceBundle that was given to the FXMLLoader  
 private ResourceBundle resources;  
  
 @FXML // URL location of the FXML file that was given to the FXMLLoader  
 private URL location;  
  
 @FXML // fx:id="tfLogFilePath"  
 private TextField tfLogFilePath; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfFTPHost"  
 private TextField tfFTPHost; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfFTPPort"  
 private TextField tfFTPPort; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfFTPUsername"  
 private TextField tfFTPUsername; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfFTPPassword"  
 private TextField pfFTPPassword; // Value injected by FXMLLoader  
  
 */\*\*  
 \* On log file browse event. Chooses log file to load in the app.  
 \*  
 \** ***@param*** *event browse event.  
 \*/* @FXML  
 void onBrowseLogFile(ActionEvent event) {  
 *LOGGER*.info("Browsing for log file...");  
 FileChooser fc = JavaFxUtil.*browseForFile*(*filterLabelLogFile*, *extensionsLogFile*);  
 File logFile = fc.showOpenDialog(new Stage());  
 // Check if any file has been selected.  
 if (logFile == null) {  
 *LOGGER*.info("No log file chosen...");  
  
 return;  
 }  
 // Get path to the selected file.  
 String absPath = logFile.getAbsolutePath();  
 tfLogFilePath.setText(logFile.getAbsolutePath());  
 *LOGGER*.info("Chosen log file is: " + absPath);  
 }  
  
 */\*\*  
 \* On continue button press event. Initializes the app, and opens the main window with loaded data.  
 \*  
 \** ***@param*** *event browse event.  
 \*/* @FXML  
 void onContinuePressed(ActionEvent event) {  
 *LOGGER*.info("Continue button pressed, validating input...");  
 // Check if the selected files exist.  
 if (!CommonUtil.*isFileExists*(getLogFilePath())) {  
 *LOGGER*.severe("Given log file path is invalid: " + getLogFilePath());  
 new Alert(Alert.AlertType.*ERROR*, "Invalid log file path!", ButtonType.*YES*).showAndWait();  
  
 return;  
 }  
  
 try {  
 getFTPPort();  
 } catch (NumberFormatException e) {  
 *LOGGER*.severe("Given DB port is invalid: " + getFTPPort());  
 new Alert(Alert.AlertType.*ERROR*, "Invalid DB port provided!", ButtonType.*YES*).showAndWait();  
  
 return;  
 }  
  
 // Open main page, and load data to it.  
 try {  
 prepareMainController(event);  
 } catch (IOException e) {  
 *LOGGER*.log(Level.*INFO*, e.getMessage(), e);  
 new Alert(Alert.AlertType.*ERROR*, "Cannot start app! " + e.getMessage(), ButtonType.*YES*).showAndWait();  
 }  
 }  
  
 private void prepareMainController(ActionEvent event) throws IOException {  
 *LOGGER*.info("Switching to the new window, loading scene...");  
 FXMLLoader loader = JavaFxUtil.*getSceneLoader*(ViewRoutes.*APP\_MAIN\_SCREEN*);  
 Parent root = loader.load();  
  
 *LOGGER*.info("Retrieving main controller and setting the init values...");  
 MainController mainController = loader.getController();  
  
 *LOGGER*.info("Setting main controller params...");  
 setMainControllerParams(mainController);  
  
 *LOGGER*.info("Preparing main controller...");  
 mainController.prepareController();  
  
 *LOGGER*.info("Opening new window...");  
 Stage mainStage = JavaFxUtil.*getNewStage*(root, Main.*MAIN\_SCREEN\_TITLE*);  
 mainStage.setOnHiding(mainController::onStageClosing);  
 mainStage.show();  
  
 *LOGGER*.info("Closing old window...");  
 JavaFxUtil.*closeWindow*(event);  
 }  
  
 private void setMainControllerParams(MainController mainController) {  
 *LOGGER*.info("Setting the init values for the main controller...");  
 mainController.setLogPath(getLogFilePath());  
  
 mainController.setFTPHost(getFTPHost());  
 mainController.setFTPPort(getFTPPort());  
 mainController.setFTPUsername(getFTPUsername());  
 mainController.setFTPPassword(getFTPPassword());  
 }  
  
 */\*\*  
 \* Exits the window when the "close" button is pressed.  
 \*  
 \** ***@param*** *event browse event.  
 \*/* @FXML  
 void onExitPressed(ActionEvent event) {  
 // Alternatively i could use: Stage stage = (Stage)btnExit.getScene().getWindow(); stage.close();  
 *LOGGER*.info("Exit button on init page pressed...");  
  
 JavaFxUtil.*closeWindow*(event);  
 }  
  
 @FXML  
 // This method is called by the FXMLLoader when initialization is complete  
 void initialize() {  
 *LOGGER*.info("Initialization...");  
 }  
  
 */\*\*  
 \* Return the path to the selected log file.  
 \*  
 \** ***@return*** *path to the selected log file.  
 \*/* public String getLogFilePath() {  
 *LOGGER*.finest("Retrieving log file path...");  
  
 return tfLogFilePath.getText();  
 }  
  
 public String getFTPHost() {  
 *LOGGER*.finest("Retrieving FTP connection host...");  
  
 return tfFTPHost.getText();  
 }  
  
 public int getFTPPort() throws NumberFormatException {  
 *LOGGER*.finest("Retrieving FTP connection port...");  
  
 return Integer.*parseInt*(tfFTPPort.getText());  
 }  
  
 public String getFTPUsername() {  
 *LOGGER*.finest("Retrieving FTP connection username...");  
  
 return tfFTPUsername.getText();  
 }  
  
 public String getFTPPassword() {  
 *LOGGER*.finest("Retrieving FTP connection password...");  
  
 return pfFTPPassword.getText();  
 }  
}

# **Package “pogrebenko.labfive.controller.main”:**

## **DICOMSliceRow.java:**

package pogrebenko.labfive.controller.main;  
  
import java.security.InvalidParameterException;  
  
*/\*\*  
 \* Represents the DICOM row value in listview.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*class DICOMSliceRow {  
 private String sliceLabel;  
 private Double sliceLocation;  
 private Integer sliceSize;  
  
 public DICOMSliceRow(Double sliceLocation, Integer sliceSize) {  
 setSliceLocation(sliceLocation);  
 setSliceSize(sliceSize);  
 setSliceLabel(String.*format*("Location: %f; Images: %d", getSliceLocation(), getSliceSize()));  
 }  
  
 public String getSliceLabel() {  
 return this.sliceLabel;  
 }  
  
 public void setSliceLabel(String sliceLabel) {  
 this.sliceLabel = sliceLabel;  
 }  
  
 public Double getSliceLocation() {  
 return this.sliceLocation;  
 }  
  
 public void setSliceLocation(Double sliceLocation) {  
 this.sliceLocation = sliceLocation;  
 }  
  
 public Integer getSliceSize() {  
 return this.sliceSize;  
 }  
  
 public void setSliceSize(Integer sliceSize) {  
 if (sliceSize < 0) {  
 throw new InvalidParameterException("Slice size cannot be below 0!");  
 }  
  
 this.sliceSize = sliceSize;  
 }  
}

## **PictureAnimationHelper.java:**

package pogrebenko.labfive.controller.main;  
  
import pogrebenko.labfive.model.DICOMImage;  
  
import javax.management.InvalidAttributeValueException;  
import java.util.List;  
  
*/\*\*  
 \* Represents current slide position in the slice of images.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*public class PictureAnimationHelper {  
 private List<DICOMImage> slice;  
 private int currSlide;  
  
 public PictureAnimationHelper(List<DICOMImage> currentSlice) throws InvalidAttributeValueException {  
 setCurrentSlice(currentSlice);  
 }  
  
 // Since configuration can be changed in the middle of the process, SYNCHRONIZED is required.  
 public synchronized void setCurrentSlice(List<DICOMImage> slice) throws InvalidAttributeValueException {  
 if (slice.size() < 1) {  
 throw new InvalidAttributeValueException("Slice must have at least 1 image.");  
 }  
  
 // If new slice were given - set it, and set curr idx to 0.  
 this.slice = slice;  
 this.currSlide = 0;  
 }  
  
 // Same as above.  
 public synchronized void incrementCurrentSlide() {

// TODO: Replace it with java util atomic.  
 this.currSlide++;  
 // If slice is ended - start from the beginning.  
 if (currSlide >= slice.size()) {  
 this.currSlide = 0;  
 }  
 }  
  
 // Same as above.  
 public synchronized DICOMImage getCurrentSlide() {  
 return slice.get(this.currSlide);  
 }  
}

## **PictureDrawer.java:**

package pogrebenko.labfive.controller.main;  
  
import javafx.application.Platform;  
import javafx.embed.swing.SwingFXUtils;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.canvas.GraphicsContext;  
import javafx.scene.control.Label;  
import javafx.scene.image.Image;  
import pogrebenko.labfive.model.DICOMImage;  
  
import javax.management.InvalidAttributeValueException;  
import java.util.List;  
import java.util.Timer;  
import java.util.TimerTask;  
  
  
*/\*\*  
 \* Represents the DICOM picture drawer.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*class PictureDrawer {  
 private final PictureAnimationHelper animationHelper;  
 Timer timer;  
 private Canvas canvasImage;  
 private Label taImageData;  
  
 PictureDrawer(List<DICOMImage> currentSlice, Canvas canvasImage, Label taImageData)  
 throws InvalidAttributeValueException {  
 animationHelper = new PictureAnimationHelper(currentSlice);  
 setCanvas(canvasImage);  
 setImageDataLabel(taImageData);  
 }  
  
 public void run() {  
 TimerTask timerTask = new TimerTask() {  
 @Override  
 public void run() {  
 Platform.*runLater*(() -> drawImage());  
 }  
 };  
 timer = new Timer(true);  
 timer.scheduleAtFixedRate(timerTask, 0, 27);  
 }  
  
 public void stop() {  
 timer.cancel();  
 // runLater to ensure that image will be clear.  
 Platform.*runLater*(() -> {  
 clearImage();  
 taImageData.setText("");  
 });  
 }  
  
 private double getXCenterCoords(Image image) {  
 return (canvasImage.getWidth() - image.getWidth()) / 2;  
 }  
  
 private double getYCenterCoords(Image image) {  
 return (canvasImage.getHeight() - image.getHeight()) / 2;  
 }  
  
 private void drawImage() {  
 // Since underlying helper is synchronized, worst possible scenario is  
 // changed configuration between getCurrentSlide() and incrementCurrentSlide()  
 // And this will only result in that new slide will start from it's next image  
 // (or from its 0, is slice size is 1)  
 DICOMImage currentImg = animationHelper.getCurrentSlide();  
 animationHelper.incrementCurrentSlide();  
  
 Image image = SwingFXUtils.*toFXImage*(currentImg.Image, null);  
 GraphicsContext ctx = canvasImage.getGraphicsContext2D();  
  
 setImageDataText(  
 "Image data:" + "\n" +  
 "File Name: " + currentImg.FileName + "\n" +  
 "Series Number: " + currentImg.SeriesNumber + "\n" +  
 "Acqu Number: " + currentImg.AcquNumber + "\n" +  
 "Inst Number: " + currentImg.InstNumber + "\n"  
 );  
  
 // Center the image  
 // *TODO: add check if image bigger than canvas.* ctx.drawImage(  
 image,  
 getXCenterCoords(image),  
 getYCenterCoords(image)  
 );  
 }  
  
 private void clearImage() {  
 GraphicsContext ctx = canvasImage.getGraphicsContext2D();  
 ctx.clearRect(0, 0, canvasImage.getWidth(), canvasImage.getHeight());  
 }  
  
 public void setCurrentSlice(List<DICOMImage> currentSlice) throws InvalidAttributeValueException {  
 animationHelper.setCurrentSlice(currentSlice);  
 }  
  
 public void setImageDataText(String data) {  
 taImageData.setText(data);  
 }  
  
 public void setCanvas(Canvas canvasImage) {  
 this.canvasImage = canvasImage;  
 }  
  
 public void setImageDataLabel(Label taImageData) {  
 this.taImageData = taImageData;  
 }  
}

## **MainController:**

package pogrebenko.labfive.controller.main;  
  
import javafx.application.Platform;  
import javafx.collections.FXCollections;  
import javafx.collections.transformation.SortedList;  
import javafx.concurrent.Task;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.Node;  
import javafx.scene.canvas.Canvas;  
import javafx.scene.control.\*;  
import javafx.scene.text.Text;  
import javafx.stage.DirectoryChooser;  
import javafx.stage.Stage;  
import javafx.stage.WindowEvent;  
import pogrebenko.labfive.model.DICOMImage;  
import pogrebenko.labfive.model.DICOMImageSlicer;  
import pogrebenko.labfive.model.FTPClientLoader;  
import pogrebenko.labfive.util.CommonUtil;  
import pogrebenko.loggerwrapper.LoggerWrapper;  
  
import javax.management.InvalidAttributeValueException;  
import java.io.File;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Paths;  
import java.util.\*;  
import java.util.function.BiConsumer;  
import java.util.function.Consumer;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
import java.util.stream.Collectors;  
  
public class MainController {  
 private static final Logger *LOGGER* = LoggerWrapper.*getLogger*();  
 FTPClientLoader ftpClientLoader;  
 DICOMImageSlicer dicomImageSlicer;  
 PictureDrawer pictureDrawer;  
 // Fields with required parameters for the app.  
 // Must be initialized before opening the page.  
 private String logPath = "";  
 // DB settings fields.  
 private String FTPHost;  
 private int FTPPort;  
 private String FTPUsername;  
 // *TODO: Secure password, move password to the main page and do not save it anywhere.* private String FTPPassword;  
 @FXML // ResourceBundle that was given to the FXMLLoader  
 private ResourceBundle resources;  
  
 @FXML // URL location of the FXML file that was given to the FXMLLoader  
 private URL location;  
  
 @FXML // fx:id="listViewImages"  
 private ListView<DICOMSliceRow> listViewImages; // Value injected by FXMLLoader  
  
 @FXML // fx:id="canvasImage"  
 private Canvas canvasImage; // Value injected by FXMLLoader  
  
 @FXML // fx:id="pbFilesLoading"  
 private ProgressBar pbFilesLoading; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfFilesDir"  
 private TextField tfFilesDir; // Value injected by FXMLLoader  
  
 @FXML // fx:id="tfSaveDir"  
 private TextField tfSaveDir; // Value injected by FXMLLoader  
  
 @FXML // fx:id="cbOverwriteSave"  
 private CheckBox cbOverwriteSave; // Value injected by FXMLLoader  
  
 @FXML // fx:id="btnLoadImg"  
 private Button btnLoadImg; // Value injected by FXMLLoader  
  
 @FXML // fx:id="btnSelectSaveDir"  
 private Button btnSelectSaveDir; // Value injected by FXMLLoader  
  
 @FXML // fx:id="textProgress"  
 private Text textProgress; // Value injected by FXMLLoader  
  
 @FXML // fx:id="lbImageData"  
 private Label lbImageData; // Value injected by FXMLLoader  
  
 private static boolean isDICOMFile(String filePath) {  
 *LOGGER*.finest("Checking is file is DICOM file...");  
  
 return CommonUtil.*getFileExtension*(filePath).equals("dcm");  
 }  
  
 @FXML  
 void onChooseSaveDir(ActionEvent event) {  
 *LOGGER*.info("Choosing save dir...");  
  
 DirectoryChooser directoryChooser = new DirectoryChooser();  
 File chosenDir = directoryChooser.showDialog(new Stage());  
  
 if (chosenDir == null) {  
 *LOGGER*.info("No save dir chosen...");  
  
 return;  
 }  
  
 String chosenDirStr = chosenDir.getAbsolutePath();  
 tfSaveDir.setText(chosenDirStr);  
 }  
  
 @FXML  
 private void onLoadImages() {  
 *LOGGER*.info("Starting files processing...");  
  
 disableUI(true);  
 stopPictureDrawer();  
 setEmptySlicesList();  
  
 Task<Void> loadTask = prepareProcessFilesTask(processFilesTask());  
 Thread th = new Thread(loadTask);  
  
 *LOGGER*.info("Starting file processing task...");  
 th.setDaemon(true);  
 th.start();  
 }  
  
 @FXML  
 // This method is called by the FXMLLoader when initialization is complete  
 void initialize() {  
 *LOGGER*.info("Initialization...");  
  
 initializeCellFactories();  
 initializeListClickListener();  
 }  
  
 private void initializeCellFactories() {  
 *LOGGER*.info("Initializing listview cell factories...");  
  
 listViewImages.setCellFactory(param -> new ListCell<>() {  
 @Override  
 protected void updateItem(DICOMSliceRow item, boolean empty) {  
 super.updateItem(item, empty);  
  
 if (empty || item == null || item.getSliceLabel() == null) {  
 setText(null);  
 } else {  
 setText(item.getSliceLabel());  
 }  
 }  
 });  
 }  
  
 private void initializeListClickListener() {  
 *LOGGER*.info("Initializing listview onclick listener");  
  
 listViewImages.getSelectionModel().selectedItemProperty().addListener(  
 (observableValue, oldValue, newValue) -> {  
 try {  
 if (newValue == null) {  
 *LOGGER*.info("Listview was updated...");  
  
 return;  
 }  
  
 if (oldValue == null) {  
 *LOGGER*.info(  
 "Listview, first image were chosen, starting picture drawer. " +  
 "Slice location: " + newValue.getSliceLocation()  
 );  
 runPictureDrawer(dicomImageSlicer.getSliceByLocation(newValue.getSliceLocation()));  
  
 return;  
 }  
  
 if (oldValue == newValue) {  
 *LOGGER*.info("Same item was selected...");  
  
 return;  
 }  
  
 *LOGGER*.info("Setting new slice to draw, slice location: " + newValue.getSliceLocation());  
 pictureDrawer.setCurrentSlice(dicomImageSlicer.getSliceByLocation(newValue.getSliceLocation()));  
 } catch (Exception e) {  
 *LOGGER*.log(Level.*SEVERE*, e.getMessage(), e);  
 new Alert(  
 Alert.AlertType.*ERROR*,  
 "Cannot show given slice! Please, contact the app developer.",  
 ButtonType.*YES* ).showAndWait();  
 }  
 });  
 }  
  
 private Task<Void> processFilesTask() {  
 *LOGGER*.info("Creating file processing task...");  
  
 return new Task<>() {  
 @Override  
 public Void call() throws Exception {  
 updateMessage("Listing files on the host machine...");  
 List<String> listedFiles = listDICOMFiles(ftpClientLoader.listDirectoryFiles(getFilesDir()));  
  
 updateMessage("Loading files from the host machine...");  
 List<String> loadedFiles = loadFTPFiles(  
 this::updateProgress,  
 this::updateMessage,  
 listedFiles,  
 getSaveDir(),  
 getOverwrite()  
 );  
  
 updateMessage("Creating slices from loaded files...");  
 dicomImageSlicer = new DICOMImageSlicer(loadedFiles);  
 SortedList<DICOMSliceRow> sortedRows = getSortedRows();  
  
 Platform.*runLater*(() -> listViewImages.setItems(sortedRows));  
 updateMessage("Done!");  
 *LOGGER*.info("Process files task finished");  
  
 return null;  
 }  
 };  
 }  
  
 private Task<Void> prepareProcessFilesTask(Task<Void> loadTask) {  
 *LOGGER*.info("Preparing file processing task...");  
  
 pbFilesLoading.progressProperty().bind(loadTask.progressProperty());  
 textProgress.textProperty().bind(loadTask.messageProperty());  
  
 loadTask.setOnSucceeded(e -> {  
 *LOGGER*.info("Data files were successfully loaded...");  
 new Alert(  
 Alert.AlertType.*INFORMATION*,  
 "Data successfully loaded",  
 ButtonType.*YES* ).showAndWait();  
  
 disableUI(false);  
 });  
  
 loadTask.setOnFailed(e -> {  
 Throwable exception = loadTask.getException();  
  
 *LOGGER*.log(Level.*SEVERE*, exception.getMessage(), exception);  
 new Alert(  
 Alert.AlertType.*ERROR*,  
 "Files download failed! Maybe, server closed connection." +  
 "Please, try to hit 'load' button again.",  
 ButtonType.*YES* ).showAndWait();  
  
 reconnectFTPClient();  
 disableUI(false);  
 });  
  
 return loadTask;  
 }  
  
 private SortedList<DICOMSliceRow> getSortedRows() {  
 *LOGGER*.info("Creating sorted rows for the listview...");  
  
 return new SortedList<>(  
 FXCollections.*observableArrayList*(formListViewRows()),  
 Collections.*reverseOrder*(  
 Comparator.*comparingDouble*(DICOMSliceRow::getSliceLocation)  
 )  
 );  
 }  
  
 private List<DICOMSliceRow> formListViewRows() {  
 *LOGGER*.info("Creating row values for the listview...");  
 List<DICOMSliceRow> rowValues = new ArrayList<>();  
  
 for (Map.Entry<Double, List<String>> entry : dicomImageSlicer.getSlices().entrySet()) {  
 rowValues.add(new DICOMSliceRow(entry.getKey(), entry.getValue().size()));  
 }  
  
 return rowValues;  
 }  
  
 private List<String> loadFTPFiles(  
 BiConsumer<Long, Long> progressBar,  
 Consumer<String> messageWindow,  
 List<String> listedDirFiles,  
 String toSave,  
 boolean overwrite  
 ) {  
 *LOGGER*.info("Loading files via ftp from the host machine...");  
 List<String> savePaths = new ArrayList<>();  
  
 for (int i = 0; i < listedDirFiles.size(); i++) {  
 String fileName = new File(listedDirFiles.get(i)).getName();  
  
 try {  
 String loadPath = Paths.*get*(getFilesDir(), listedDirFiles.get(i)).toString();  
 String savePath = Paths.*get*(toSave, fileName).toString();  
  
 messageWindow.accept("Loading file: " + loadPath);  
 boolean isSaved = !ftpClientLoader.downloadFile(loadPath, savePath, overwrite);  
  
 if (!overwrite && isSaved) {  
 *LOGGER*.finest(String.*format*("File '%s' already exist in path '%s': ", fileName, savePath));  
 }  
  
 savePaths.add(savePath);  
 progressBar.accept((long) i, (long) listedDirFiles.size());  
 } catch (IOException e) {  
 *LOGGER*.severe("Cannot load file: " + fileName);  
 new Alert(  
 Alert.AlertType.*WARNING*,  
 "Cannot load file: " + fileName +  
 "\nMaybe, server has fallen. If this file is important, please, try to run download again.",  
 ButtonType.*YES* ).showAndWait();  
 }  
 }  
  
 progressBar.accept((long) 1, (long) 1);  
 *LOGGER*.info("Number of Loading files from the host machine: " + savePaths.size());  
  
 return savePaths;  
 }  
  
 private List<String> listDICOMFiles(List<String> files) {  
 *LOGGER*.info("filtering DICOM files from the given files...");  
 List<String> listedDICOM = files.stream()  
 .filter(MainController::*isDICOMFile*)  
 .collect(Collectors.*toList*());  
 *LOGGER*.info("Number of DICOM files after filter: " + listedDICOM.size());  
  
 return listedDICOM;  
 }  
  
 private void runPictureDrawer(List<DICOMImage> selectedSlice) throws InvalidAttributeValueException {  
 *LOGGER*.info("Running picture drawer...");  
  
 pictureDrawer = new PictureDrawer(selectedSlice, canvasImage, lbImageData);  
 pictureDrawer.run();  
 }  
  
 private void stopPictureDrawer() {  
 *LOGGER*.info("Stopping picture drawer...");  
  
 if (pictureDrawer == null) {  
 *LOGGER*.info("Picture drawer already stopped.");  
  
 return;  
 }  
  
 pictureDrawer.stop();  
 pictureDrawer = null;  
 }  
  
 private void reconnectFTPClient() {  
 try {  
 ftpClientLoader = new FTPClientLoader(getFTPHost(), getFTPPort(), getFTPUsername(), getFTPPassword());  
 } catch (IOException exception) {  
 *LOGGER*.log(Level.*SEVERE*, exception.getMessage(), exception);  
 new Alert(  
 Alert.AlertType.*ERROR*,  
 "Error during FTP reconnection! Please, try again.",  
 ButtonType.*YES* ).showAndWait();  
 }  
 }  
  
 public void setNodesDisabled(boolean disable, Node... nodes) {  
 *LOGGER*.info("Disabling UI nodes: " + disable);  
  
 for (Node node : nodes) {  
 node.setDisable(disable);  
 }  
 }  
  
 // *TODO: Add graceful download thread stop.* public void onStageClosing(WindowEvent windowEvent) {  
 *LOGGER*.info("Stage is closing...");  
  
 try {  
 ftpClientLoader.close();  
 stopPictureDrawer();  
 } catch (IOException e) {  
 *LOGGER*.warning(e.getMessage());  
 }  
 }  
  
 public void prepareController() throws IOException {  
 *LOGGER*.info("Preparing controller...");  
  
 LoggerWrapper.*getWrapper*().addFileHandler(getLogFilePath());  
 ftpClientLoader = new FTPClientLoader(getFTPHost(), getFTPPort(), getFTPUsername(), getFTPPassword());  
 }  
  
 private void setEmptySlicesList() {  
 listViewImages.setItems(FXCollections.*observableArrayList*());  
 }  
  
 private void disableUI(boolean disable) {  
 *LOGGER*.info("Disable UI: " + disable);  
  
 setNodesDisabled(  
 disable,  
 btnLoadImg,  
 btnSelectSaveDir,  
 cbOverwriteSave,  
 tfFilesDir,  
 tfSaveDir,  
 cbOverwriteSave,  
 listViewImages  
 );  
  
 listViewImages.setMouseTransparent(disable);  
 listViewImages.setFocusTraversable(!disable);  
 }  
  
 public void setLogPath(String logPath) {  
 *LOGGER*.finest("Setting log file path...");  
  
 this.logPath = logPath;  
 }  
  
 public String getLogFilePath() {  
 *LOGGER*.finest("Retrieving log file path...");  
  
 return logPath;  
 }  
  
 public String getFTPHost() {  
 *LOGGER*.finest("Retrieving FTP connection host...");  
  
 return FTPHost;  
 }  
  
 public void setFTPHost(String FTPHost) {  
 *LOGGER*.finest("Setting FTP connection host...");  
  
 this.FTPHost = FTPHost;  
 }  
  
 public int getFTPPort() {  
 *LOGGER*.finest("Retrieving FTP connection port...");  
  
 return FTPPort;  
 }  
  
 public void setFTPPort(int FTPPort) {  
 *LOGGER*.finest("Setting FTP connection port...");  
  
 this.FTPPort = FTPPort;  
 }  
  
 public String getFTPUsername() {  
 *LOGGER*.finest("Retrieving FTP connection username...");  
  
 return FTPUsername;  
 }  
  
 public void setFTPUsername(String FTPUsername) {  
 *LOGGER*.finest("Setting FTP connection username...");  
  
 this.FTPUsername = FTPUsername;  
 }  
  
 public String getFTPPassword() {  
 *LOGGER*.finest("Retrieving FTP connection password...");  
  
 return FTPPassword;  
 }  
  
 public void setFTPPassword(String FTPPassword) {  
 *LOGGER*.finest("Setting FTP connection username...");  
  
 this.FTPPassword = FTPPassword;  
 }  
  
 public String getFilesDir() {  
 *LOGGER*.finest("Retrieving FTP connection files dir...");  
  
 return tfFilesDir.getText();  
 }  
  
 public String getSaveDir() {  
 *LOGGER*.finest("Retrieving save dir...");  
  
 return tfSaveDir.getText();  
 }  
  
 public boolean getOverwrite() {  
 *LOGGER*.finest("Retrieving save dir...");  
  
 return cbOverwriteSave.isSelected();  
 }  
}

# **Package “pogrebenko.labfive.util”:**

## **CommonUtil.java:**

package pogrebenko.labfive.util;  
  
import java.io.File;  
  
*/\*\*  
 \* Util methods that may be used in various labs.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*public final class CommonUtil {  
 */\*\*  
 \* Private constructor that throws an exception to prevent instantiation.  
 \*/* private CommonUtil() {  
 throw new UnsupportedOperationException();  
 }  
  
 */\*\*  
 \* Returns the extension of the given file name.  
 \*  
 \** ***@param*** *fileName file name to parse.  
 \** ***@return*** *file extension (without dot).  
 \** ***@throws*** *IllegalArgumentException if there is no "." in "fileName".  
 \*/* public static String getFileExtension(String fileName) throws IllegalArgumentException {  
 int pos = fileName.lastIndexOf('.');  
  
 if (pos < 0) {  
 throw new IllegalArgumentException(String.*format*("File %s doesn't have an extension!", fileName));  
 }  
  
 return fileName.substring(pos + 1);  
 }  
  
 */\*\*  
 \* Checks if the given file exists.  
 \*  
 \** ***@param*** *fileName file name to check.  
 \** ***@return*** *true if file exists, false otherwise.  
 \*/* @SuppressWarnings("BooleanMethodIsAlwaysInverted")  
 public static boolean isFileExists(String fileName) {  
 if (fileName == null) {  
 return false;  
 }  
  
 File f = new File(fileName);  
  
 return f.exists() && !f.isDirectory();  
 }  
}

## **JavaFxUtil.java:**

package pogrebenko.labfive.util;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
import pogrebenko.labfive.view.ViewRoutes;  
  
import java.io.IOException;  
  
*/\*\*  
 \* Frontend util methods that may be used in various labs.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*public class JavaFxUtil {  
 */\*\*  
 \* Private constructor that throws an exception to prevent instantiation.  
 \*/* private JavaFxUtil() {  
 throw new UnsupportedOperationException();  
 }  
  
 */\*\*  
 \* Creates "browse for file" dialog.  
 \*  
 \** ***@param*** *label label message of the dialog.  
 \** ***@param*** *extensions extensions to browse for.  
 \** ***@return*** *chosen file.  
 \*/* public static FileChooser browseForFile(String label, String... extensions) {  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter(label, extensions));  
  
 return fileChooser;  
 }  
  
 */\*\*  
 \* Returns FXML loader for given window.  
 \*  
 \** ***@param*** *route route of view to load.  
 \** ***@return*** *FXML loader for given window.  
 \*/* public static FXMLLoader getSceneLoader(ViewRoutes route) {  
 return new FXMLLoader(route.getURL());  
 }  
  
 public static Stage getNewStage(Parent root, String title) {  
 Stage stage = new Stage();  
 stage.setTitle(title);  
 stage.setScene(new Scene(root));  
 return stage;  
 }  
  
 */\*\*  
 \* Opens new window with given title, and sets it's scene to "root" scene.  
 \*  
 \** ***@param*** *root root scene to display in the new window.  
 \** ***@param*** *title title of the new window.  
 \*/* public static void openNewWindow(Parent root, String title) {  
 getNewStage(root, title).show();  
 }  
  
 */\*\*  
 \* Opens new window with view from given FXML route and given title.  
 \* No settings or fields is set here, so new window will be "clean".  
 \*  
 \** ***@param*** *route route of view to load.  
 \** ***@param*** *title title of the new window.  
 \*/* public static void openCleanWindow(ViewRoutes route, String title) throws IOException {  
 FXMLLoader loader = getSceneLoader(route);  
 openNewWindow(loader.load(), title);  
 }  
  
 */\*\*  
 \* Closes window via it's event.  
 \*  
 \** ***@param*** *event event of the window to close.  
 \*/* public static void closeWindow(ActionEvent event) {  
 ((Node) (event.getSource())).getScene().getWindow().hide();  
 }  
}

# **Package “pogrebenko.labfive.view”:**

## **app\_init.fxml**

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.text.\*?>  
<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="322.0" prefWidth="600.0"  
 style="-fx-background-color: #ffffff;" xmlns="http://javafx.com/javafx/15.0.1"  
 fx:controller="pogrebenko.labfive.controller.initialization.InitializationController">  
 <AnchorPane layoutX="15.0" layoutY="61.0" prefHeight="69.0" prefWidth="570.0"  
 style="-fx-background-color: #ffffff;">  
 <TextField fx:id="tfLogFilePath" editable="false" layoutX="14.0" layoutY="22.0" prefHeight="25.0"  
 prefWidth="435.0" />  
 <Label layoutX="14.0" prefHeight="19.0" prefWidth="130.0" text="Specify your log file:">  
 <font>  
 <Font name="System Bold Italic" size="13.0"/>  
 </font>  
 </Label>  
 <Button layoutX="466.0" layoutY="22.0" mnemonicParsing="false" onAction="#onBrowseLogFile" prefHeight="25.0"  
 prefWidth="90.0" text="Browse"/>  
 </AnchorPane>  
 <Button layoutX="308.0" layoutY="271.0" mnemonicParsing="false" onAction="#onContinuePressed" prefHeight="25.0"  
 prefWidth="70.0" text="Continue"/>  
 <Button layoutX="213.0" layoutY="271.0" mnemonicParsing="false" onAction="#onExitPressed" prefHeight="25.0"  
 prefWidth="70.0" text="Exit"/>  
 <Label layoutX="236.0" layoutY="26.0" prefHeight="25.0" prefWidth="169.0" style="-fx-background-color: #ffffff;"  
 text="APP INITIALIZATION">  
 <font>  
 <Font name="System Bold" size="16.0"/>  
 </font>  
 </Label>  
 <AnchorPane layoutX="20.0" layoutY="130.0" prefHeight="141.0" prefWidth="500.0">  
 <children>  
 <TextField fx:id="tfFTPHost" layoutX="92.0" layoutY="52.0" prefHeight="25.0" prefWidth="150.0"  
 text="195.3.158.22"/>  
 <Label layoutX="10.0" layoutY="14.0" prefHeight="25.0" prefWidth="540.0"  
 style="-fx-background-color: #ffffff;" text="Please, enter your FTP connection settings:">  
 <font>  
 <Font name="System Bold" size="15.0"/>  
 </font>  
 </Label>  
 <Label layoutX="21.0" layoutY="56.0" prefHeight="19.0" prefWidth="90.0" text="Host:">  
 <font>  
 <Font name="System Bold Italic" size="13.0"/>  
 </font>  
 </Label>  
 <Label layoutX="21.0" layoutY="93.0" prefHeight="19.0" prefWidth="90.0" text="Port:">  
 <font>  
 <Font name="System Bold Italic" size="13.0"/>  
 </font>  
 </Label>  
 <TextField fx:id="tfFTPPort" layoutX="93.0" layoutY="89.0" prefHeight="25.0" prefWidth="150.0" text="8000"/>  
 <TextField fx:id="tfFTPUsername" layoutX="386.0" layoutY="52.0" prefHeight="25.0" prefWidth="150.0"  
 text="81-12"/>  
 <Label layoutX="314.0" layoutY="55.0" prefHeight="19.0" prefWidth="70.0" text="Username:">  
 <font>  
 <Font name="System Bold Italic" size="13.0"/>  
 </font>  
 </Label>  
 <Label layoutX="314.0" layoutY="92.0" prefHeight="19.0" prefWidth="90.0" text="Password:">  
 <font>  
 <Font name="System Bold Italic" size="13.0"/>  
 </font>  
 </Label>  
 <PasswordField fx:id="pfFTPPassword" layoutX="387.0" layoutY="90.0"/>  
 </children>  
 </AnchorPane>  
</AnchorPane>

## **app\_main.fxml**

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.text.Font?>  
<?import javafx.scene.text.Text?>  
<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="655.0" prefWidth="881.0"  
 style="-fx-background-color: #ffffff;" xmlns="http://javafx.com/javafx/15.0.1"  
 fx:controller="pogrebenko.labfive.controller.main.MainController">  
 <Label layoutX="14.0" layoutY="14.0" prefHeight="19.0" prefWidth="116.0" style="-fx-background-color: #ffffff;"  
 text="DICOM Viewer">  
 <font>  
 <Font name="System Bold" size="16.0"/>  
 </font>  
 </Label>  
 <ListView fx:id="listViewImages" layoutX="14.0" layoutY="39.0" prefHeight="514.0" prefWidth="292.0"/>  
 <Button fx:id="btnLoadImg" layoutX="14.0" layoutY="582.0" mnemonicParsing="false" onAction="#onLoadImages"  
 text="Load Images"/>  
 <ProgressBar fx:id="pbFilesLoading" layoutX="116.0" layoutY="586.0" prefHeight="18.0" prefWidth="190.0"  
 progress="0.0"/>  
 <TextField fx:id="tfFilesDir" layoutX="395.0" layoutY="594.0" prefHeight="25.0" prefWidth="381.0" text="\\lab5"/>  
 <Label layoutX="317.0" layoutY="598.0" text="FTP Files dir:"/>  
 <TextField fx:id="tfSaveDir" editable="false" layoutX="395.0" layoutY="563.0" prefHeight="25.0" prefWidth="381.0"  
 text="D:\test"/>  
 <Label layoutX="317.0" layoutY="567.0" text="Save dir:"/>  
 <Button fx:id="btnSelectSaveDir" layoutX="791.0" layoutY="563.0" mnemonicParsing="false" onAction="#onChooseSaveDir"  
 prefHeight="25.0" prefWidth="68.0" text="Browse"/>  
 <Label layoutX="316.0" layoutY="627.0" prefHeight="17.0" prefWidth="106.0" text="Overwrite if exists:"/>  
 <CheckBox fx:id="cbOverwriteSave" layoutX="422.0" layoutY="627.0" mnemonicParsing="false" selected="true"/>  
 <Text fx:id="textProgress" layoutX="116.0" layoutY="622.0" strokeType="OUTSIDE" strokeWidth="0.0"  
 wrappingWidth="190.0"/>  
 <AnchorPane layoutX="315.0" layoutY="39.0" prefHeight="392.0" prefWidth="543.0" style="-fx-border-color: black;">  
 <children>  
 <Canvas fx:id="canvasImage" height="385.0" layoutX="6.0" layoutY="3.0" width="533.0"/>  
 </children>  
 </AnchorPane>  
 <AnchorPane layoutX="315.0" layoutY="440.0" prefHeight="113.0" prefWidth="543.0" style="-fx-border-color: black;">  
 <children>  
 <Label fx:id="lbImageData" alignment="CENTER" layoutX="5.0" layoutY="4.0" prefHeight="106.0"  
 prefWidth="533.0"/>  
 </children>  
 </AnchorPane>  
</AnchorPane>

## **ViewRoutes.java**

package pogrebenko.labfive.view;  
  
import java.net.URL;  
  
*/\*\*  
 \* Routes of the different windows of the app.  
 \* Enum is used to ensure that only given routes will be used.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.3.0  
 \** ***@since*** *1.2  
 \*/*public enum ViewRoutes {  
 // Initialize paths to the every view in the project.  
 *APP\_INIT\_SCREEN*("app\_init.fxml"),  
 *APP\_MAIN\_SCREEN*("app\_main.fxml");  
  
 private final URL url;  
  
 */\*\*  
 \* Constructs view url.  
 \*  
 \** ***@param*** *url url of the view.  
 \*/* ViewRoutes(final String url) {  
 this.url = getClass().getResource(url);  
 }  
  
 */\*\*  
 \* Returns the url of the view.  
 \*  
 \** ***@return*** *url of the view.  
 \*/* public URL getURL() {  
 return url;  
 }  
  
 */\*\*  
 \* Returns the path to the view.  
 \*  
 \** ***@return*** *string representation of the view url.  
 \*/* @Override  
 public String toString() {  
 return url.toString();  
 }  
}

# **Package “pogrebenko.loggerwrapper”**

package pogrebenko.loggerwrapper;  
  
import java.io.IOException;  
import java.util.logging.\*;  
  
*/\*\*  
 \* Wrapper for default logger.  
 \* Allows to simply log to a single file across different classes.  
 \*  
 \** ***@author*** *Pogrebenko Vasily, BS-81  
 \** ***@version*** *1.0  
 \** ***@since*** *1.0  
 \*/*public class LoggerWrapper {  
 // Init default logger.  
 private static final Logger *logger* = Logger.*getLogger*(LoggerWrapper.class.getName());  
 // Static wrapper variable for implementing singleton pattern.  
 private static LoggerWrapper *wrapper* = null;  
  
 */\*\*  
 \* Private constructor to ensure singleton.  
 \*/* private LoggerWrapper() {  
 }  
  
 */\*\*  
 \* Returns singleton instance of the logger wrapper.  
 \* No need in "synchronized" keyword for now,  
 \* but if app will be multithreaded, it will be necessary.  
 \*  
 \** ***@return*** *LoggerWrapper the singleton instance of the logger wrapper.  
 \*/* public static synchronized LoggerWrapper getWrapper() {  
 if (*wrapper* == null) {  
 *wrapper* = new LoggerWrapper();  
 }  
  
 return *wrapper*;  
 }  
  
 */\*\*  
 \* Returns singleton logger instance of the logger.  
 \*  
 \** ***@return*** *logger instance.  
 \*/* public static synchronized Logger getLogger() {  
 return *logger*;  
 }  
  
 */\*\*  
 \* Adds file handler to the logger.  
 \*  
 \** ***@param*** *LogName name of the log file.  
 \*/* public synchronized void addFileHandler(String LogName) throws IOException {  
 FileHandler fh = new FileHandler(LogName);  
 fh.setFormatter(new SimpleFormatter());  
 addHandler(fh);  
 }  
  
 */\*\*  
 \* Adds custom handler to the logger.  
 \*  
 \** ***@param*** *handler handler to add.  
 \*/* public synchronized void addHandler(Handler handler) {  
 *logger*.addHandler(handler);  
 }  
  
 */\*\*  
 \* Sets the logger logging level.  
 \*  
 \** ***@param*** *level logging level.  
 \*/* public synchronized void setLoggingLevel(Level level) {  
 *logger*.setLevel(level);  
 }  
  
 */\*\*  
 \* Sets the logger log format.  
 \*/* public synchronized void setLogFormat() {  
 // Set timestamp format for logger.  
 System.*setProperty*("java.util.logging.SimpleFormatter.format",  
 "%1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS %4$s %2$s %5$s%6$s%n");  
 }  
}

**Результат роботи програми:**

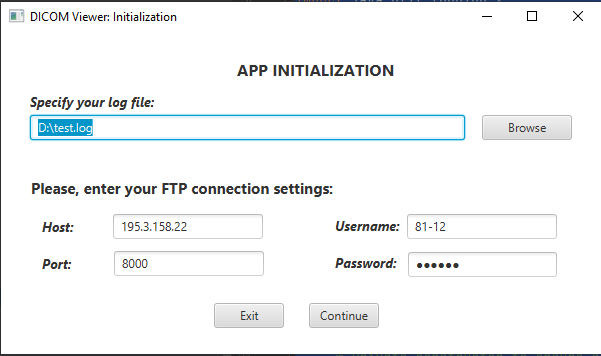


Рис. 1 – Ініціалізація додатку.

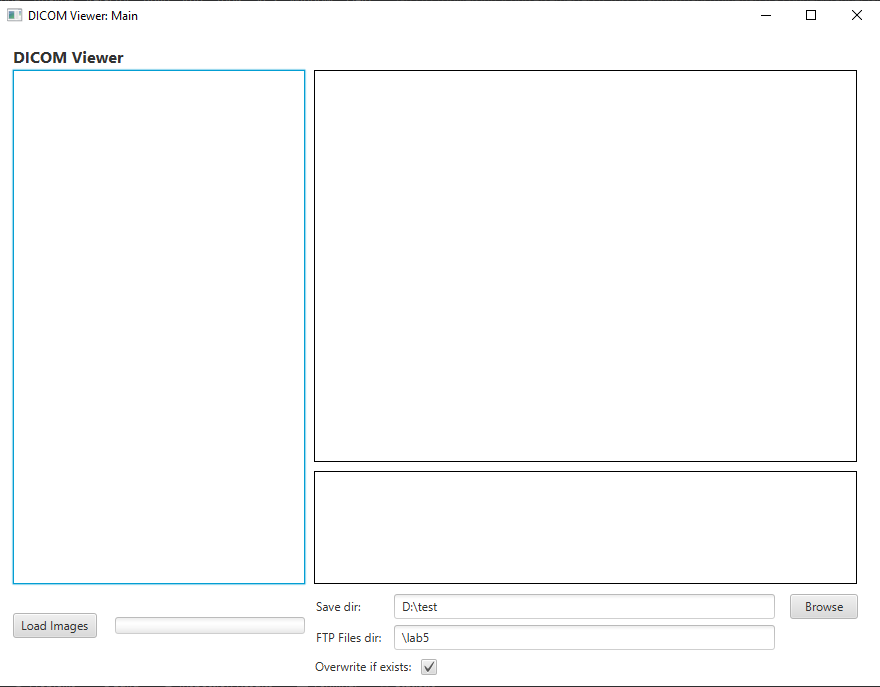


Рис. 2 – Початкове вікно. Можна вибрати директорію для завантаження, вибрати опцію збереження (перезапусувати вже існуючі чи ні), та інше.

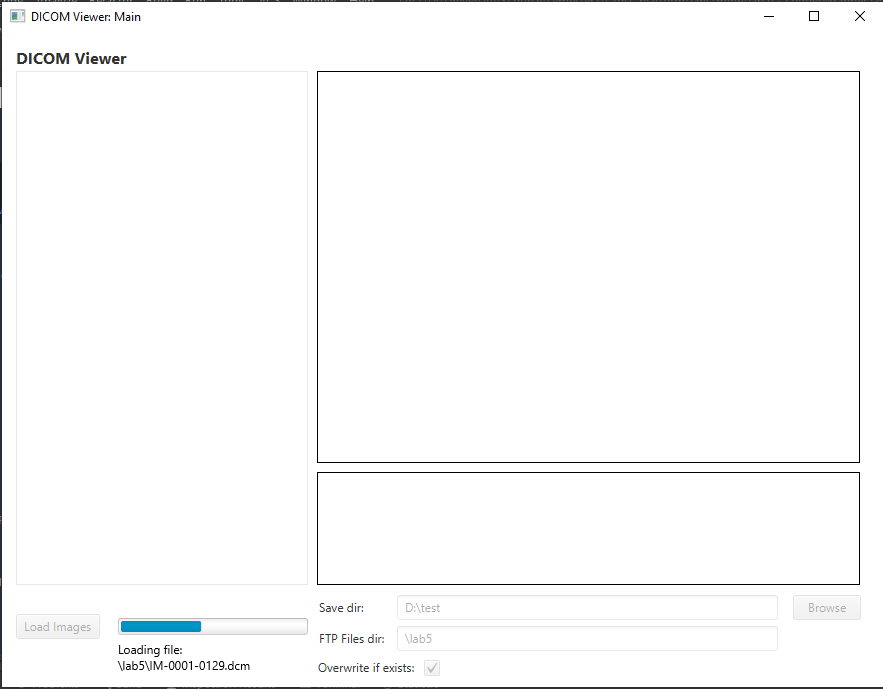


Рис. 3 – Процесс завантаження. У наявності progress bar та повідомлення про поточні дії. UI заблокований.

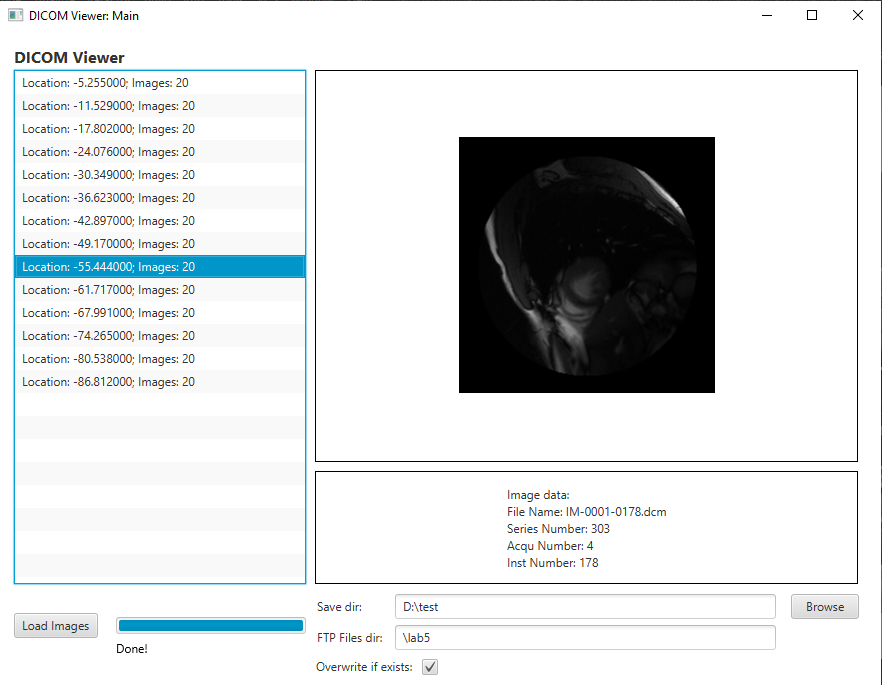


Рис. 4 – Приклад роботи створенних анимацій. UI розблокований після завантаження.

У випадку, якщо потрібні файли з іншої директоріі, а з’єднання з сервером було втрачено, буде виведено наступне повідомлення, та буде зроблена спроба відновлення з’єднання:

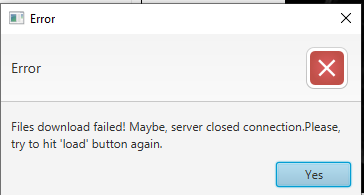
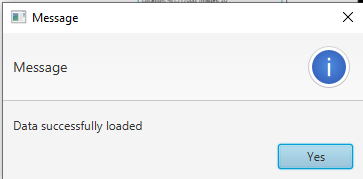
 

Рис. 5 – Відновленне з’єднання при натступному натисканні кнопки «завантажити».

**Контрольні запитання:**

1. **Опишіть класи FTPClient та BufferedImage.**

**FTPClient** – належить до пакету org.apache.commons.net, за його допомогою можна вести роботу з FTP – серверами. Наприклад, отримати список файлів у вказанній директорії, завантажити файл з FTP серверу, вказати режим роботи з сервером, етс.

**BufferedImage –** наслідник від абстракного класу Image, що дозволяє працювати з зображенням: отримувати його параметри (наприклад, висоту зображення), та змінювати зображення за допомогою, наприклад, setRGB().

1. **Для чого призначені методи paint() та update() класу Canvas та в чому полягає взаємозв'язок між ними?**

**update()** - оновлює данні Canvas.

**paint()**  - з використанням заданного GraphicsContext, «малює» дані Canvas.

**Взаємозв’язок** – зазвичай, після виклику paint() для зміни зображення, викликається update(), щоб візуально оновити дані.

1. **Які задачі і яким чином дозволяє вирішувати клас Timer? Що скасовує його метод cancel(), а що – ні?**

Засіб для тредів для планування завдань для подальшого виконання у фоновому потоці. Завдання можуть бути заплановані для одноразового виконання або для повторного виконання через рівні проміжки часу.

Кожному об'єкту таймера відповідає один фоновий потік, який використовується для послідовного виконання всіх завдань таймера.

**cancel()** – скасовує запланованні завдання, але не скасовує задачу, що вже запущена таймером.

1. **Надайте характеристику класу TimerTask та його властивостям.**

**TimerTask -** Завдання, яке може бути заплановано для одноразового або повторного виконання таймером.

Завдання таймера не можна використовувати багаторазово. Після того, як завдання було заплановано для виконання на таймері або скасовано, подальші спроби запланувати його для виконання призведуть до IllegalStateException.

Так як **TimerTask** реалізує інтерфейс Runnable, то він має методи run() и cancel().

1. **Розкрийте сутність асинхронної організації виконання дій та вкажіть, в чому полягяють її переваги над синхронною.**

Під час виконання асинхронного коду певні задачі можуть виконуватися паралельно, не очікуючи одна-одну (чк це відбувається під час виконання синхронного коду).

Може використовуватися як для параллельних обчислень, палалельного виконання дій, для прискорення виконання роботи, так і для UI, для того, щоб його поток не блокувався важкими задачами, та він залишвся активним.

1. **Чим відрізняється дійсна паралельність виконання від псевдопаралельності та при якій апаратній конфігурації перша з них неможлива і чому?**

Дії, що робить псевдопаралельна виконуються в одному потоці паралельно, але не одночасно. Це відбувається тим чином, що програма «переключається» з однієї задачі на іншу під час роботи.

А дісно паралельна програма виконується на різних потоках у багатопоточній системі,

1. **В чому полягають переваги багатопотоковості над послідовним виконанням дій та які є дві основні проблеми її використання? Що означає термін «потокобезпечний» («thread-safe»)?**

**Плюси:**

За допомогою багатопотоковості можливо:

- Дозволити UI програмі залишатися чутливою до введення під час виконання важких операцій.

- Пришвидшити розрахунки у важких задачах як, наприклад, рендер зображень.

В загалому це дозволяє використовувати ресурси системи більш ефективно.

**Мінуси:**

* Складність синхронізації та пов'язані з цим складнощі: використовуючи спільні ресурси, типові для різьбових програм, програміст повинен бути обережним, щоб уникати такого як race condition та data race. Потрібно синхронізувати програму м’ютексами, семаформаи, етс.
* Загалом, багатопотокові програми не є детермінованими. Деякі помилки можуть не з’явитися на тесті, а з’явитися на виробніцтві.

1. **Які засоби та механізми надає Java для паралельного виконання дій та яким класом вони забезпечуються?**

Можливі натсупні варіанти:

* Успадкування від класу Thread - можливо успадкувати свій клас від класу Thread і зробити override методу run () кодом, який потрібно виконати кількома потоками.
* Реалізувати інтерфейс Runnable (наприклад, Thread – реалізація Runnable).
* Реалізувати інтерфейс Callable. Callable схожий на Runnable, але повертає результат і може викликати виняток

1. **Що являє собою стандарт DICOM та для роботи з якою інформацією він призначений?**

Цифрова візуалізація та комунікація в медицині (DICOM) є стандартом для передачі та управління медичною інформаційною візуалізацією та відповідними даними. DICOM найчастіше використовується для зберігання та передачі медичних зображень, що дозволяє інтегрувати такі медичні пристрої візуалізації, як сканери, сервери, робочі станції, принтери, мережеве обладнання та системи архівування та комунікації зображень (PACS) від багатьох виробників.

1. **Сформулюйте своє розуміння щодо відповідального відношення до розробки технологій, які призначені для використання в медицині та інших важливих галузях, на прикладі ситуації з апаратом «Therac-25» та аналогічних випадків, що Вам відомі.**

Халатніть в процессі розробки критичних систем може призвести до трагедії, а, з ростом впливу IT на світ, критичні помилки можуть с кожним днем коштувати все більше. Так, наприклад, можна побачити, що в 1991-му році в Іспанії, в місті Сарагоса, на схожому до Therac-25 апараті Sagitar-35, протягом 10 днів піддалися передозуванні не менше 25 пацієнтів, з яких як мінімум троє померли.

Тому, у питаннях розробки продукту для медичних закладів, та інших важливих систем, весь софт повинен бути розвроблений та протестований з притримуванням усіх кращих практик, перед тим, як він буде переданий в експлуатацію.