МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет  
информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Проектная работа

Построение клиент-серверной архитектуры

Выполнил студент группы № М3307:  
Егоров Николай Валерьевич

Проверил:  
Рябчиков Игорь …

Санкт-Петербург  
2019

# Введение

Требуется реализовать два независимых компонента клиент-сервера на языке java: клиент и сервер с возможностью запуска на разных узлах. Решение должно иметь следующий функционал:

* коммуникация между клиентом и сервером для приема-передачи изображений;
* сервер должен хранить изображения в базе на диске;
* возможность установить одно из доступных на клиентском узле изображений в качестве рабочего стола системы.

# Архитектура

При реализации были разработаны следующие основные классы:

* *ServerBase*

Является непосредственно ответсвенным за серверную часть, причем в целях многозадачности реализована система потоков, каждый из которых запускает класс-обработчик запросов индивидуально для каждого из клиентов.

* QueryHandler

Класс-обработчик серверной стороны запросов клиента. Таким образом, многопоточность достигается простым сопоставлением каждому экземпляра класса клиента экземлпяром класса обработчика, который вызывается извне класса ServerBase. Так же, у каждого экземпляра данного класса реализована возможность логирования событий: подключения, отключения клиента, загрузка/отправка изображения.

* Client

Класс-клиент, функционал соответствует требованиям задания – коммуникация и возможность установки классом WallChanger доступного изображения как фона рабочего стола.

* ClientUI

Представляет собой оболочку с графическим интерфейсом над классом Client, инкапсулирует его.

* WallChanger

Класс для решения задачи программной установки фона рабочего стола для операционных систем Windows OS и Mac OS.

# Структура

Для наглядного представления об устройстве решения основные классы с методами и подклассами могут быть внесены в таблицы

|  |  |  |
| --- | --- | --- |
|  |  | Создает сервер для класса Client. |
| **Класс:** ServerBase | | Использует класс QueryHandler для обработки всех запросов. Имеет верхнюю границу числа подключенных потоков. |
|  |  |  |
|  | main(*String[] args*) | Создан в целях отладки, инициирует запуск сервера по loopback порт: 6000, лимит по потокам клиентов – 5. |
|  |  |
|  |  |
|  |  |  |
|  | ServerBase*(int port,* | Конструктор, открывает сокет для подключения |
|  | *int upperThreadLimit*) | И создает пул потоков обработки. |
|  |  |  |
| **Подкласс:** ServerConnectionLoop | | Создает пул потоков. Запускается при инициализации сервера, поле threadLimit определяет границу подключений. |
|  |  | Ожидает подключения клиента. При подключении клиента |
|  | run() | определяет новый поток обработки. |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Является протоколом для обслуживания сообщения | |
|  |  | клиента и сервера. Вызывается классом ServerBase | |
| **Класс:** QueryHandler | |  | |
| Спроектирован так, что база помещается по относительному пути от места запуска: | |
|  |  |
|  |  |  | /images/ |
|  |  |  |  |
|  |  |  |  |
|  | QueryHandler(*Socket* | Конструктор | |
|  | *client*) |
|  |  |  |
|  |  |  | |
|  | Send(*String message*) |  | |
|  | Посылает сообщение message клиенту по потоку и выводит сообщение в консоль. | |
|  |  |
|  |  |  |  |
|  | writeToLog(*String* | Дополняет файл serverlog.txt в следующем формате: | |
|  | *functionRequested,* |
|  | date : time : ipAddress : request | |
|  | *String filename*) |
|  |  |  | |
|  |  | Реагирует на запрос со стороны клиента, имеющие один из | |
|  |  | видов: | |
|  | handleQuery() |  | requesting::[filename] – запрос на отправку |
|  |  | sending::[filename] – запрос на загрузку |
|  |  |
|  |  |  | requestlist:: - запрос на список доступных |
|  |  |  | |
|  |  |  | |
|  | ScanImagesFromDisk() |  | |
|  | Индексирует собственную базу и выводит содержимое на консоль | |
|  |  |
|  |  |  | |
|  | uploadToUser(*String* |  | |
|  | *fileName*) | Отправляет файл клиенту после получения команды requesting .Выводит информацию о действиях в терминал | |
|  |  |  | |
|  | downloadFromUser( |  | |
|  | *String fileName*) | Осуществляет загрузку конкретного файла со стороны клиента | |
|  |  |  | |
|  | sendImageList() | Посылает содержимое проиндексированной базы в виде динамического массива типа String | |
|  |  |  | |
|  | run() |  | |
|  | Класс-стартер | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** Client | |  | |
| Клиент для подключения. В данной реализации выступает и сервером при отправке/загрузке изображений. | |
|  |  |
|  |  |  | |
|  |  |  | |
|  | main(*String[] args*) | Реализован для нужд откладки, подключается автоматически по loopback – интерфейсу к серверу. | |
|  |  |  | |
|  |  |  | |
|  | ImageClient(*String* |  | |
|  | Конструктор. Подключается к серверу по сокету и далее вызывает метод cacheImages()update чтобы определить имеющиеся в доступе изображения. | |
|  | *host, int port*) |
|  |  | |
|  |  |
|  |  |  | |
|  |  |  | |
|  | getImageList() | Посылает запрос серверу об его базе изображений, выводит эту информацию в консоль, а также заводит информацию в список | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  | cacheImages() | Считывает свои изображения в папке /images и хранит информацию об их именах в списке | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  | Считывает запросы из консоли, определен известный формат: | |
|  | queryTaker() |  | * requesting::[filename] – запрос на загрузку * sending::[filename] – запрос на отправку * requestlist:: - запрос на список доступных |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  | sendToServer(*String* |  | |
|  | *fileName*) | Посылает на сервер конкретный файл. Выводит информацию о действиях в консоль. | |
|  |  |  | |
|  | downloadFromServer( | Загружает файл с сервера. Выводит информацию о действиях в консоль. | |
|  | *String fileName*) |  | |

# Исходные файлы

ServerBase.java

import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ThreadPoolExecutor;  
  
public class ServerBase {  
  
 private ServerSocket serverSocket = null;  
 private Integer threadLimit = null;  
  
 public ServerBase(int port, int threadLimit) {  
 this.threadLimit= threadLimit;  
 try {  
 serverSocket= new ServerSocket(port);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Thread sclThread = new Thread(new ServerConnectLoop());  
 sclThread.start();  
 }  
  
 public class ServerConnectLoop implements Runnable  
 {  
 ThreadPoolExecutor pool = (ThreadPoolExecutor) Executors.*newFixedThreadPool*(threadLimit);  
  
 public void run()  
 {  
 try  
 { while (true)  
 { Socket clientSocket = serverSocket.accept();  
 QueryHandler q = new QueryHandler(clientSocket);  
 pool.execute(q);  
 System.*out*.println("[SERVER] Client connected. " + pool.getActiveCount() + " client(s) connected.");  
 }  
 }  
 catch (IOException e)  
 { e.printStackTrace();  
 }  
 }  
 }  
  
  
 public static void main(String[] args) {  
 ServerBase s= new ServerBase(6000,5);  
 System.*out*.println("Server is running. Terminate the process for shut down.");  
 }  
}

QueryHandler.java

import javax.imageio.ImageIO;  
import java.awt.image.BufferedImage;  
import java.io.\*;  
import java.net.InetAddress;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
  
public class QueryHandler implements Runnable {  
  
 private Scanner reader = null;  
 private PrintWriter writer = null;  
 private String message = null;  
 private InetAddress clientAddress = null;  
 public ArrayList<String> imageList = null;  
 public File[] fileList = null;  
 private String dirPath = null;  
  
  
  
 public QueryHandler(Socket client) {  
 clientAddress = client.getInetAddress();  
 dirPath = System.*getProperty*("user.dir");  
 try {  
 reader = new Scanner(client.getInputStream());  
 writer = new PrintWriter(client.getOutputStream(), true);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 private void Send(String message) {  
 this.writer.println(message);  
 System.*out*.println("[SERVER RESPOND] " + message);  
 }  
  
 private void handleQuery() {  
 String fileName = "";  
 // Retrieve filename from terminal query.  
 int separatorPos = this.message.lastIndexOf("::");  
 if ((separatorPos > 0) ) {  
 if (!this.message.endsWith("::")) {  
 fileName = this.message.substring(separatorPos + 2);  
 }  
 }  
  
 if (this.message.startsWith("requesting::")) {  
 SendFileToUser(fileName);  
 } else if (this.message.startsWith("sending::")) {  
 DownloadFromUser(fileName);  
 } else if (this.message.startsWith("requestlist::")) {  
 SendImageList();  
 } else {  
 Send("Invalid query: " + this.message);  
 }  
  
 }  
  
  
 private void ScanImagesFromDisk() {  
  
 System.*out*.println("Indexing own base...");  
 File dir = new File(this.dirPath + "/images");  
 this.fileList = dir.listFiles((dir1, name) -> name.toLowerCase().endsWith(".jpg")  
 ||name.toLowerCase().endsWith(".png") ||name.toLowerCase().endsWith(".jpeg")  
 || name.toLowerCase().endsWith(".bmp") || name.toLowerCase().endsWith(".gif")  
 || name.toLowerCase().endsWith(".png") ||name.toLowerCase().endsWith(".tif"));  
 this.imageList = new ArrayList<String>(this.fileList.length);  
 for (File file : this.fileList) {  
 if (file.isFile()) {  
 System.*out*.println(" adding: " + file.getName());  
 this.imageList.add(file.getName());  
 }  
 }  
 System.*out*.println();  
  
 }  
  
 public void SendFileToUser(String fileName) {  
 try {  
 int dotLocation = fileName.lastIndexOf(".");  
 String fileType = "";  
 if (dotLocation > 0) {  
 fileType = fileName.substring(dotLocation + 1);  
 }  
 System.*out*.println("Detected file type: " + fileType);  
 System.*out*.println("Analyzing file: /images/" + fileName);  
 BufferedImage img = ImageIO.*read*(new File(this.dirPath + "/images/" + fileName));  
 System.*out*.println("Converting image to byte array output stream.");  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 ImageIO.*write*(img, fileType, baos);  
 baos.flush();  
 byte[] bytes = baos.toByteArray();  
 baos.close();  
 System.*out*.println("Byte array with " + bytes.length + " length created.");  
  
  
 System.*out*.println("Opening new socket to connect to user.");  
 Socket soc = new Socket(this.clientAddress, 51712);  
 System.*out*.println("Opening streams with user.");  
 OutputStream outStream = soc.getOutputStream();  
 DataOutputStream dos = new DataOutputStream(outStream);  
  
 System.*out*.println("Writing to user stream.");  
 dos.writeInt(bytes.length);  
 dos.write(bytes, 0, bytes.length);  
 System.*out*.println("Closing streams/socket.");  
 dos.close();  
 outStream.close();  
 soc.close();  
 writeToLog("sent image", fileName);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
  
 public void SendImageList() {  
 ScanImagesFromDisk();  
 Send(Integer.*toString*(imageList.size()));  
 for (String anImageList : this.imageList) {  
 Send(anImageList);  
 }  
 Send("stop::");  
 }  
  
 private void writeToLog(String functionRequested, String filename) {  
 File log = new File("servlog.txt");  
  
 if (!log.exists()) {  
 try {  
 PrintWriter writer = new PrintWriter("servlog.txt", "UTF-8");  
 writer.write("");  
 writer.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 // Create a new Date object and read date and time from it as strings.  
 Date currentDateTime = new Date();  
 DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");  
 DateFormat timeFormat = new SimpleDateFormat("HH:mm:ss");  
 String request = functionRequested + " " + filename;  
 String date = dateFormat.format(currentDateTime);  
 String time = timeFormat.format(currentDateTime);  
 String toLog = date + " : " + time + " : " + this.clientAddress + " : " + request + "\n";  
 // Write to serverlog.txt  
 try {  
 Files.*write*((Paths.*get*("servlog.txt")), toLog.getBytes(), StandardOpenOption.*APPEND*);  
 System.*out*.println("Event has been logged.");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 public void DownloadFromUser(String fileName) {  
 try {  
 System.*out*.println("Opening new socket for client to connect...");  
 ServerSocket server = new ServerSocket(51712);  
 Socket socket = server.accept();  
 System.*out*.println(" Accepted connection with client.\nRetrieving input stream...");  
 InputStream inStream = socket.getInputStream();  
 DataInputStream dis = new DataInputStream(inStream);  
  
 int dataLength = dis.readInt();  
 byte[] data = new byte[dataLength];  
 dis.readFully(data);  
 dis.close();  
 inStream.close();  
 System.*out*.println(" Finished receiving input stream.\nConverting to file...");  
  
 InputStream bais = new ByteArrayInputStream(data);  
 System.*out*.println(this.dirPath);  
 String filePath = this.dirPath + "/images/" + fileName;  
 OutputStream toFile = new FileOutputStream(filePath);  
 byte[] buffer = new byte[1024];  
 int bytesRead;  
 while ((bytesRead = bais.read(buffer)) != -1) {  
 System.*out*.println(" Bytes read of length: " + bytesRead);  
 toFile.write(buffer, 0, bytesRead);  
 }  
 bais.close();  
 toFile.flush();  
 toFile.close();  
 server.close();  
 System.*out*.println(" ...Finished!\n");  
 writeToLog("received image", fileName);  
  
 ScanImagesFromDisk();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 this.message = reader.nextLine();  
 while (this.message != null) {  
 System.*out*.println("Server read: " + this.message + "\n");  
 handleQuery();  
 System.*out*.println("Listening for message...");  
 this.message = reader.nextLine();  
 }  
 } catch (NoSuchElementException nsee) {  
 reader.close();  
 writer.close();  
 System.*out*.println("Client disconnected.");  
 writeToLog("client disconnected", "");  
 break;  
 }  
 }  
 }  
  
  
}

Client.java

package client;  
  
import javax.imageio.ImageIO;  
import java.awt.image.BufferedImage;  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.net.UnknownHostException;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Client {  
  
 private String hostServ;  
 private Scanner socketInbound;  
 private PrintWriter socketOut = null;  
 private Scanner keyIn = null;  
 private String messageIn = null;  
 public ArrayList<String> serverImageList = null;  
 private String dirPath;  
 public ArrayList<String> imageList = null;  
 public File[] fileList = null;  
  
 private WallChanger changer;  
  
  
 public Client(String host, int port) {  
 hostServ = host;  
 dirPath = System.*getProperty*("user.dir");  
 changer = new WallChanger();  
 try {  
 Socket socket = new Socket(host, port);  
 socketInbound = new Scanner(socket.getInputStream());  
 socketOut = new PrintWriter(socket.getOutputStream(), true);  
 keyIn = new Scanner(System.*in*);  
 cacheImages();  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
  
 public void getImageList() {  
 socketOut.println("requestlist::");  
 int length = Integer.*parseInt*(socketInbound.nextLine());  
 this.serverImageList = new ArrayList<String>(length);  
 this.messageIn = socketInbound.nextLine();  
 System.*out*.println("Available images to download:");  
 while (!this.messageIn.equals("stop::")) {  
 System.*out*.println(" - " + this.messageIn);  
 this.serverImageList.add(this.messageIn);  
 this.messageIn = socketInbound.nextLine();  
 }  
 System.*out*.println("");  
  
 }  
  
 public void downloadFromServer(String fileName) {  
 socketOut.println("requesting::" + fileName);  
 String filePath = this.dirPath + "/images/" + fileName;  
 try (ServerSocket server = new ServerSocket(51712);  
 OutputStream toFile = new FileOutputStream(filePath);  
 ) {  
  
 System.*out*.println("Opening the new socket for connection...");  
  
 Socket socket = server.accept();  
 System.*out*.println(" Connection with server is established.\nRetrieving input stream...");  
 InputStream inStream = socket.getInputStream();  
 DataInputStream dis = new DataInputStream(inStream);  
 int dataLength = dis.readInt();  
 byte[] data = new byte[dataLength];  
 dis.readFully(data);  
 dis.close();  
 inStream.close();  
 System.*out*.println(" Finished receiving input stream.\nConverting to file...");  
  
 InputStream bais = new ByteArrayInputStream(data);  
 byte[] buffer = new byte[1024];  
 int bytesRead = 0;  
 while ((bytesRead = bais.read(buffer)) != -1) {  
 System.*out*.println(" Bytes read of length: " + bytesRead);  
 toFile.write(buffer, 0, bytesRead);  
 }  
 bais.close();  
 toFile.flush();  
 toFile.close();  
 System.*out*.println(" ...Finished!\n");  
 cacheImages();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
  
 public void sendToServer(String fileName) {  
  
 for (String item : serverImageList)  
 if (item.equals(fileName)) {  
 return;  
 }  
 socketOut.println("sending::" + fileName);  
  
  
 try {  
 int dot = fileName.lastIndexOf(".");  
 String fileType = "";  
 if (dot > 0) {  
 fileType = fileName.substring(dot + 1);  
 }  
 System.*out*.println("Detected file type: " + fileType);  
 System.*out*.println("Reading file: /images/" + fileName);  
 BufferedImage img = ImageIO.*read*(new File(this.dirPath + "/images/" + fileName));  
 System.*out*.println("Converting image to byte array output stream.");  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 ImageIO.*write*(img, fileType, baos);  
 baos.flush();  
 byte[] bytes = baos.toByteArray();  
 baos.close();  
 System.*out*.println("Byte array of length " + bytes.length + " created.");  
  
 System.*out*.println("Opening new socket to connect to server.");  
 Socket soc = new Socket(this.hostServ, 51712);  
 System.*out*.println("Opening streams with server.");  
 OutputStream outStream = soc.getOutputStream();  
 DataOutputStream dos = new DataOutputStream(outStream);  
  
 System.*out*.println("Writing to server stream.");  
 dos.writeInt(bytes.length);  
 dos.write(bytes, 0, bytes.length);  
 System.*out*.println("Closing streams/socket.");  
 dos.close();  
 outStream.close();  
 soc.close();  
 } catch (Exception e) {  
 System.*out*.println("Exception: " + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
  
 private void cacheImages() {  
 File dir = new File(this.dirPath + "/images");  
 this.fileList = dir.listFiles();  
 this.imageList = new ArrayList<String>(this.fileList.length);  
 System.*out*.println("Finding files in /images directory...");  
 boolean found = false;  
 for (File file : this.fileList) {  
 if (file.isFile()) {  
 String extension = "";  
 int i = file.toString().lastIndexOf('.');  
 if (i > 0) {  
 extension = file.toString().substring(i + 1);  
 }  
 System.*out*.println(" Found: " + file.getName());  
 found = true;  
 if (!extension.equals("db"))  
 this.imageList.add(file.getName());  
 }  
 }  
 if (!found) {  
 System.*out*.println("No files found!");  
 }  
 System.*out*.println();  
 }  
  
 */\*\*  
 \* Reads input from terminal, which should be one of the following:  
 \* requesting::[filename], sending::[filename], requestlist::  
 \* Calls appropriate method if input is valid. Does not send  
 \* query to server if e.g. file does not exist.  
 \*\*/* protected void queryTaker() {  
 String messageOut;  
 while ((messageOut = keyIn.nextLine()) != null) {  
 String fileName = "";  
 boolean error = true;  
 if (messageOut.contains("::")) {  
 int separatorLocation = messageOut.lastIndexOf("::");  
 if (!messageOut.endsWith("::")) {  
 fileName = messageOut.substring(separatorLocation + 2);  
 }  
  
 if (messageOut.equals("requestlist::")) {  
 error = false;  
 System.*out*.println("[CLIENT OUTPUT] " + messageOut + "\n");  
 System.*out*.println(" Available images to download:");  
 getImageList();  
 }  
  
 if (messageOut.startsWith("requesting::") && serverImageList.contains(fileName)) {  
 error = false;  
 System.*out*.println("[CLIENT OUTPUT] " + messageOut);  
 downloadFromServer(fileName);  
 }  
  
 if (messageOut.startsWith("sending::") && imageList.contains(fileName)) {  
 error = false;  
 System.*out*.println("Ohoh");  
 sendToServer(fileName);  
 }  
 }  
  
 if (error) {  
 System.*out*.println("\n Invalid requests. Expected one of the following:");  
 System.*out*.println(" \* requesting::[filename] ");  
 System.*out*.println(" \* sending::[filename]");  
 System.*out*.println(" \* requestlist::");  
 }  
 }  
 }  
  
 protected void setWallPaper(String fileName) throws IOException {  
 System.*out*.println("Setting " + fileName + " as the main wallpaper... ");  
 changer.changeWallpaper(dirPath + "/images/" + fileName);  
 System.*out*.println("Successfully set");  
 }  
  
  
 */\*\*  
 \* For debug and test purposes  
 \*/* public static void main(String[] args) {  
 String host = "127.0.0.1"; //args[0]  
 Client ic = new Client(host, 6000);  
 ic.queryTaker();  
 }  
  
  
}

OsDefiner.java

public final class OsDefiner  
{  
 private static String *OS* = null;  
 public static String getOsName()  
 {  
 if(*OS* == null) { *OS* = System.*getProperty*("os.name"); }  
 return *OS*;  
 }  
 public static boolean isWindows()  
 {  
 return *getOsName*().startsWith("Windows");  
 }  
  
 public static boolean isUnix()  
 {  
 return *getOsName*().startsWith("Unix");  
 }  
  
 public static boolean isMac()  
 {  
 return *getOsName*().startsWith("Mac");  
 }  
}

WallChanger.java

import com.sun.jna.Library;  
import com.sun.jna.win32.W32APIOptions;  
import com.sun.jna.Native;  
import java.io.IOException;  
  
public class WallChanger {  
  
 public void setMac(String path) throws IOException {  
 String as[] = {  
 "osascript",  
 "-e", "tell application \"Finder\"",  
 "-e", "set desktop picture to POSIX file \"" + path + "\"",  
 "-e", "end tell"  
 };  
 Runtime runtime = Runtime.*getRuntime*();  
 runtime.exec(as);  
 }  
  
 public static interface User32 extends Library {  
 User32 *INSTANCE* = (User32) Native.*loadLibrary*("user32",User32.class, W32APIOptions.*DEFAULT\_OPTIONS*);  
 boolean SystemParametersInfo (int one, int two, String s ,int three);  
 }  
  
 */\*\*  
 \* For Wallpaper changer which is uiAction == 20 and uiParam is obligatory == 0  
 \** ***@param*** *path - path to the wallpaper file  
 \** ***@return*** *\*/* public void changeWallpaper(String path) throws IOException {  
 if (OsDefiner.*isWindows*())  
 User32.*INSTANCE*.SystemParametersInfo(0x0014, 0, path, 1);  
 else if (OsDefiner.*isMac*() || OsDefiner.*isUnix*())  
 setMac(path);}}