|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования«Московский технологический университет»МИРЭА | | |

Институт информационных технологий

Кафедра вычислительной техники

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Отчет по лабораторным работам** | | | | |
| **по дисциплине** | | | | |
| **«Операционные системы»** | | | | |
|  | | | | |
| Студент группы ИКБО-6-15 | | Стась Денис Андреевич | | |
|  | |  | | |
| Работа представлена | | «\_\_»\_\_\_\_\_\_\_201\_\_\_ г. | | *(подпись студента)* |
|  | |  | |  |
| Принимающий работу | | «\_\_»\_\_\_\_\_\_\_201\_\_\_ г. | | *(подпись руководителя)* |

Москва, 2017

СОДЕРЖАНИЕ

1. Лабораторная работа №1………………………………………………………..2

2. Лабораторная работа №2………………………………………………………..4

3. Лабораторная работа №3………………………………………………………..5

4. Лабораторная работа №4………………………………………………………..6

5. Лабораторная работа №5………………………………………………………..8

6. Лабораторная работа №6……………………………………………………….18

7. Лабораторная работа №7………………………………………………………...21

**Лабораторная работа №1.**

**Задание:** Разработать приложение, осуществляющее ряд системных вызовов. Запротоколировать фактические вызовы, выполняемые приложением с помощью специального инструмента.

1. Создать в папке приложения файл с произвольным именем произвольного размера (не менее 10 КБ не более 10 МБ).
2. Считать у этого файла дату и время создания, последней модификации и доступа, атрибуты, размер и вывести их на экран.
3. Установить время создания файла на 11 января 1601 года, 18:06:00 (UTC+3)
4. Установить время модификации файла на 04:00:00 (UTC+3) послезавтра.

**Код программы:**

Класс CreateFile

**import** java.io.File;  
**import** java.io.IOException;  
**public class** CreateFile {  
 **public static void** main (String args [])**throws** IOException{  
 String absoluteFilePath = **"file.txt"**;  
 File file = **new** File(absoluteFilePath);  
 **if**(file.createNewFile()){  
 System.***out***.println(absoluteFilePath + **" Файл создан"**);  
 } **else** {  
 System.***out***.println(**"Файл "** + absoluteFilePath + **" уже существует"**);  
  
 }  
 }  
}

Класс Main

**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.LinkOption;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.nio.file.attribute.FileTime;  
**import** java.util.Calendar;  
**import** java.util.Date;  
**public class** Main {  
 **public static void** main(String[] args) {  
 *//Указываем путь к созданному файлу* Path path = Paths.*get*(**"D:\\ProgramFiles\\Programm\\OC\\lab1\\file.txt"**);  
  
 **try** {  
 System.***out***.println(**"Информация о файле: "**);  
 System.***out***.println(**"\t Имя файла: "** + path.getFileName());  
 System.***out***.println(**"\t Корневая папка: "** + path.getRoot());  
 System.***out***.println(**"\t Весь путь к файлу: "** + path.getParent());  
  
 Object object = Files.*getAttribute*(path, **"creationTime"**);  
 System.***out***.println(**"\t Время и дата создания: "** + object);  
  
 object = Files.*getAttribute*(path, **"lastModifiedTime"**,  
 LinkOption.***NOFOLLOW\_LINKS***);  
 System.***out***.println(**"\t Последняя модификация: "** + object);  
  
 object = Files.*getAttribute*(path, **"size"**);  
 System.***out***.println(**"\t Размер: "** + object);  
  
 object = Files.*getAttribute*(path, **"isDirectory"**);  
 System.***out***.println(**"\t Каталог: "** + object);  
  
 Calendar s = Calendar.*getInstance*();  
 s.set(1960,Calendar.***MARCH***,20);  
  
 object = Files.*setAttribute*(path,**"creationTime"**, FileTime.*fromMillis*(s.getTimeInMillis()));  
 object = Files.*getAttribute*(path, **"creationTime"**);  
 System.***out***.println(**"\t Время и дата создания: "** + object);  
  
 Calendar m = Calendar.*getInstance*();  
 m.set(2017,Calendar.***MARCH***,20);  
  
 object = Files.*setLastModifiedTime*(path, FileTime.*fromMillis*(m.getTimeInMillis()));  
 object = Files.*getAttribute*(path, **"lastModifiedTime"**);  
 System.***out***.println(**"\t Последняя модификация: "** + object);  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Скриншоты:**



**Лабораторная работа №2.**

**Задание:** Создать приложение, осуществляющее базовые операции с дочерним процессом и запрос расширенной информации о его состоянии.

Основные функции приложения:

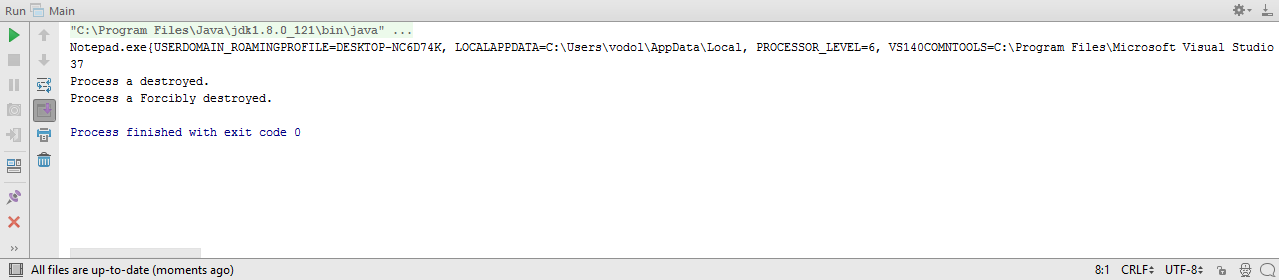
1. Создание процесса (Запуск процессов notepad.exe, stressload.exe)
2. Вывод информации о процессе (ID процесса, ID главного потока, приоритет)
3. Вывод информации о ресурсах (количество хэндлов, счетчики памяти)
4. Вывод статистики процесса (Счетчики времени и ввода-вывода)
5. Закрытие процесса (освобождение хэндлов)
6. Принудительное завершение процесса

**Код программы:**

Класс Main:

**import** java.io.IOException;  
**import** java.util.Map;  
**public class** Main {  
 **public static void** main(String[] args ) **throws** IOException {  
 Process a = **new** ProcessBuilder(**"notepad.exe"**).start();  
 ProcessBuilder ac = **new** ProcessBuilder();  
 Map<String, String> a1 = ac.environment();  
 System.***out***.println(**"Notepad.exe"** + a1);  
 System.***out***.println(a1.size());  
 a.destroy();  
 System.***out***.println(**"Process a destroyed."**);  
 a.destroyForcibly();  
 System.***out***.println(**"Process a Forcibly destroyed."**);  
  
  
  
  
  
 }  
}

**Скриншоты:**



**Лабораторная работа №3.**

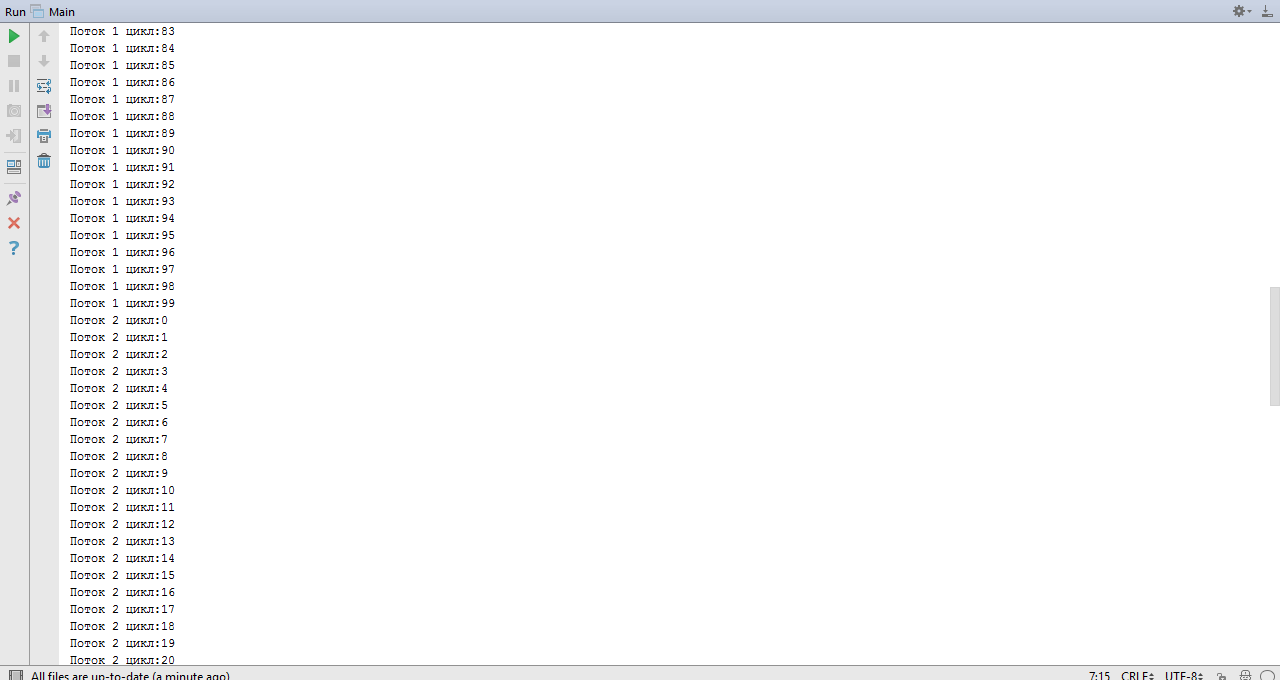
**Задание:** Составить программу, реализующую обработку 2х массивов целых чисел с использованием основного потока приложения или 1, 2, 3 или 4 вычислительных параллельных потоков.

**Код программы:**

Класс Main

**public class** Main {  
 **public static void** main(String args []){  
 Counter th = **new** Counter();  
 **new** Thread(th,**"Поток 1 "**).start();  
 **new** Thread(th,**"Поток 2 "**).start();  
 }  
}  
  
**class** Counter **implements** Runnable {  
 **synchronized public void** run() {  
 **int** i;  
 **for** (i=0; i<100; i++){  
 System.***out***.println(Thread.*currentThread*().getName() + **"цикл:"** + i);  
 }  
 }  
}

**Скриншоты:**



**Лабораторная работа №4.**

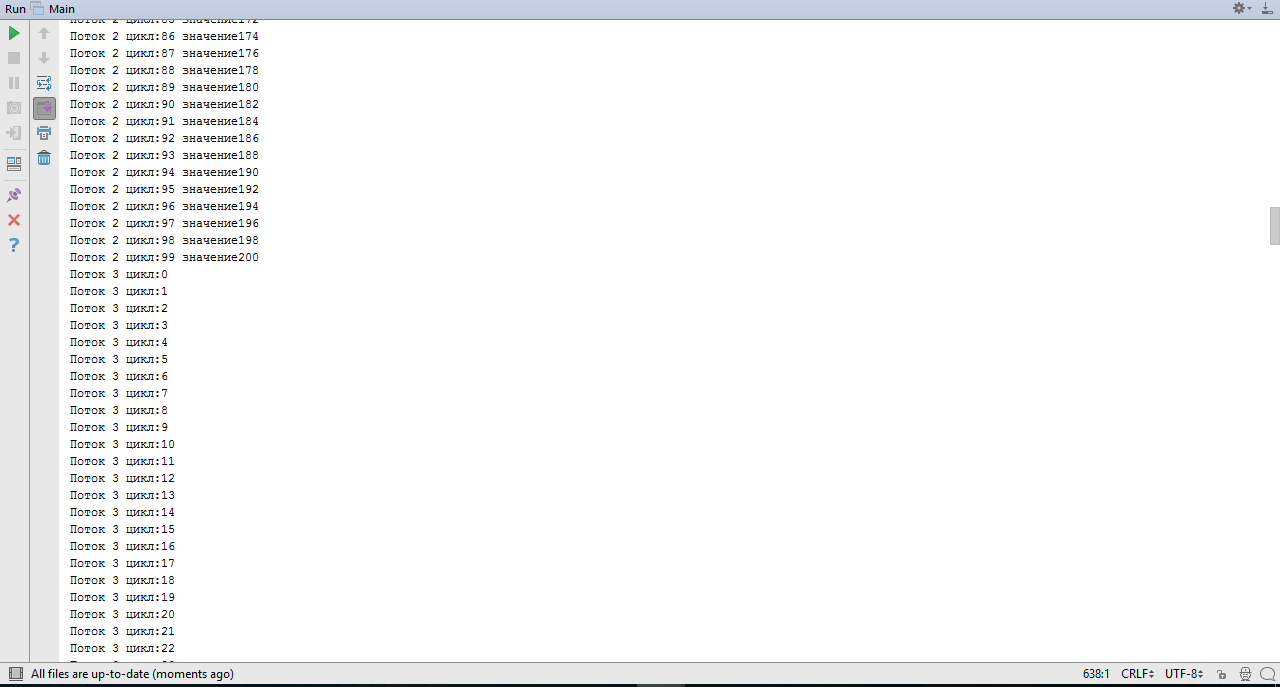
**Задание:** Составить программу, реализующую параллельную работу с общим ресурсом (целой переменной) с и без синхронизации. Использовать 4 потока, каждый из которых увеличивает значение переменной, делает работу (можно использовать задержку на 50-300 мс) и уменьшает значение переменной. Зафиксировать максимальное значение с включенной и выключенной синхронизацией.

**Код программы:**

Класс Main

**public class** Main {  
 **public static void** main(String args []) **throws** InterruptedException {  
 Counter1 th1 = **new** Counter1();  
 **new** Thread(th1,**"Поток 1 "**).*sleep*(100);  
 **new** Thread(th1,**"Поток 1 "**).start();  
 **new** Thread(th1,**"Поток 2 "**).start();  
  
 Counter3 th3 = **new** Counter3();  
 **new** Thread(th3,**"Поток 3 "**).*sleep*(250);  
 **new** Thread(th3,**"Поток 3 "**).start();  
  
 Counter4 th4 = **new** Counter4();  
 **new** Thread(th4,**"Поток 4 "**).start();  
  
  
 }  
}  
  
**class** Counter1 **implements** Runnable {  
 **synchronized public void** run() {  
 **int** i;  
 **int** a = 0;  
 **for** (i=0; i<100; i++){  
 a = a+2;  
 System.***out***.println(Thread.*currentThread*().getName() + **"цикл:"** + i + **" значение"** + a);  
 }  
 }  
}  
  
**class** Counter3 **implements** Runnable {  
 **public void** run() {  
 **int** i;  
 **for** (i=0; i<100; i++){  
 System.***out***.println(Thread.*currentThread*().getName() + **"цикл:"** + i);  
 }  
 }  
}  
  
**class** Counter4 **implements** Runnable {  
 **public void** run() {  
 **int** i;  
 **for** (i=0; i>-1000; i = i-3){  
 System.***out***.println(Thread.*currentThread*().getName() + **"цикл:"** + i);  
 }  
 }  
}

**Скриншоты:**

****

**Лабораторная работа №5.**

**Задание:** Реализовать приложение, позволяющее обмениваться текстовыми сообщениями между 2 запущенными экземплярами приложения. Использовать механизмы каналов (пайпов), сообщений windows, общей памяти через отображаемый файл.

**Код программы:**

Класс Main

**package** Main;

**import** client.Client;  
**import** pipe.Pipe;  
**import** server.Server;  
  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
  
 System.***out***.println(**"Запустить программу в режиме сервера или клиента? (S(erver) / C(lient)/ P(ipe)"**);  
 **while** (**true**) {  
 **char** answer = Character.*toLowerCase*(in.nextLine().charAt(0));  
 **if** (answer == **'s'**) {  
 **new** Server();  
 **break**;  
 } **else if** (answer == **'c'**) {  
 **new** Client();  
 **break**;  
 }**else if** (answer == **'p'**) {  
 **new** Pipe();  
 **break**;  
 } **else** {  
 System.***out***.println(**"Некорректный ввод. Повторите."**);  
 }  
 }  
 }  
  
}

Класс PipeExample

**import** Main.Const;  
  
**import** java.io.\*;  
**import** java.net.Socket;  
**import** java.util.Scanner;  
  
**public class** PipeExample {  
  
 **private static** BufferedReader *in*;  
 **private static** PrintWriter *out*;  
 **private static** Socket *socket*;  
  
 **public static void** main(String[] args) **throws** IOException {  
  
 **final** PipedOutputStream output = **new** PipedOutputStream();  
 **final** PipedInputStream input = **new** PipedInputStream(output);  
 Scanner scan = **new** Scanner(System.***in***);  
 **final** String[] str = {**""**};  
  
  
  
 Thread thread1 = **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 System.***out***.println(**"Введите IP для подключения к серверу."**);  
 System.***out***.println(**"Формат: xxx.xxx.xxx.xxx"**);  
 String ip = scan.nextLine();  
 **try** {  
  
 *socket* = **new** Socket(ip, Const.***Port***);  
 *in* = **new** BufferedReader(**new** InputStreamReader(*socket*.getInputStream()));  
 *out* = **new** PrintWriter(*socket*.getOutputStream(), **true**);  
  
 System.***out***.println(**"Введите свой ник:"**);  
 str[0] = scan.nextLine();  
 String a = str[0];  
 output.write(**"Input text"**.getBytes());  
 output.write(**"\n"**.getBytes());  
  
**while** (!str[0].equals(**"exit"**)) {  
 str[0] = scan.nextLine();  
 output.write(a.getBytes());  
 output.write(**":"**.getBytes());  
 output.write(str[0].getBytes());  
 output.write(**"\n"**.getBytes());  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **finally** {  
 **try** {  
 output.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 });  
  
  
 Thread thread2 = **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 **int** data = input.read();  
 **while**(data != -1){  
 System.***out***.print((**char**) data);  
 data = input.read();  
  
 }  
 } **catch** (IOException e) {  
 }  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
  
 }  
}

Класс Pipe

**package** pipe;  
  
**import** Main.Const;  
  
**import** java.io.\*;  
**import** java.net.Socket;  
**import** java.util.Scanner;  
  
**public class** Pipe {  
 **private** BufferedReader **PipedInputStream**;  
 **private** PrintWriter **PipedOutputStream**;  
 **private** Socket **socket**;  
  
 */\*\*  
 \* Запрашивает у пользователя ник и организовывает обмен сообщениями с  
 \* сервером  
 \*/* **public** Pipe() {  
 Scanner scan = **new** Scanner(System.***in***);  
  
 System.***out***.println(**"Введите IP для подключения к серверу."**);  
 System.***out***.println(**"Формат: xxx.xxx.xxx.xxx"**); *//127.0.0.1* String ip = scan.nextLine();  
  
 **try** {  
 *// Подключаемся в серверу и получаем потоки(in и out) для передачи сообщений* **socket** = **new** Socket(ip, Const.***Port***);  
 **PipedInputStream** = **new** BufferedReader(**new** InputStreamReader(**socket**.getInputStream()));  
 **PipedOutputStream** = **new** PrintWriter(**socket**.getOutputStream(), **true**);  
  
 System.***out***.println(**"Введите свой ник:"**);  
 **PipedOutputStream**.println(scan.nextLine());  
  
 *// Запускаем вывод всех входящих сообщений в консоль* Resender resend = **new** Resender();  
 resend.start();  
  
 *// Пока пользователь не введёт "exit" отправляем на сервер всё, что  
 // введено из консоли* String str = **""**;  
 **while** (!str.equals(**"exit"**)) {  
 str = scan.nextLine();  
 **PipedOutputStream**.println(str);  
 }  
 resend.setStop();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 close();  
 }  
 }  
  
 */\*\*  
 \* Закрывает входной и выходной потоки и сокет  
 \*/* **private void** close() {  
 **try** {  
 **PipedInputStream**.close();  
 **PipedOutputStream**.close();  
 **socket**.close();  
 } **catch** (Exception e) {  
 System.***err***.println(**"Потоки не были закрыты!"**);  
 }  
 }  
  
 */\*\*  
 \* Класс в отдельной нити пересылает все сообщения от сервера в консоль.  
 \* Работает пока не будет вызван метод setStop().  
 \*/* **private class** Resender **extends** Thread {  
  
 **private boolean stoped**;  
  
 */\*\*  
 \* Прекращает пересылку сообщений  
 \*/* **public void** setStop() {  
 **stoped** = **true**;  
 }  
  
 */\*\*  
 \* Считывает все сообщения от сервера и печатает их в консоль.  
 \* Останавливается вызовом метода setStop()  
 \*/* @Override  
 **public void** run() {  
 **try** {  
 **while** (!**stoped**) {  
 String str = **PipedInputStream**.readLine();  
 System.***out***.println(str);  
 }  
 } **catch** (IOException e) {  
 System.***err***.println(**"Ошибка при получении сообщения."**);  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

Класс Const

**package** Main;  
  
**public abstract class** Const {  
  
 **public static final int *Port*** = 8283;  
  
}

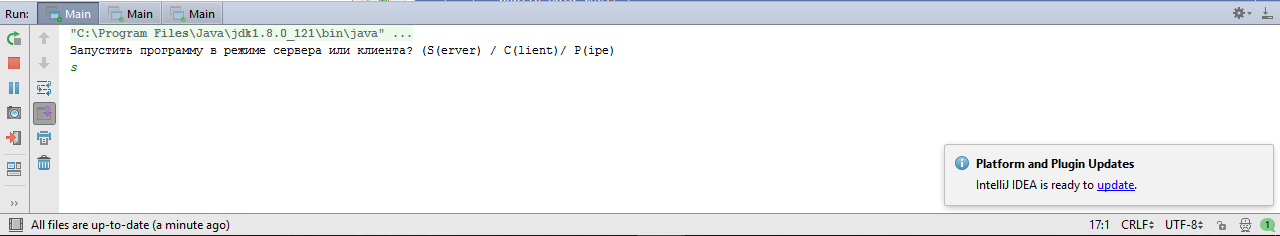
Класс Client

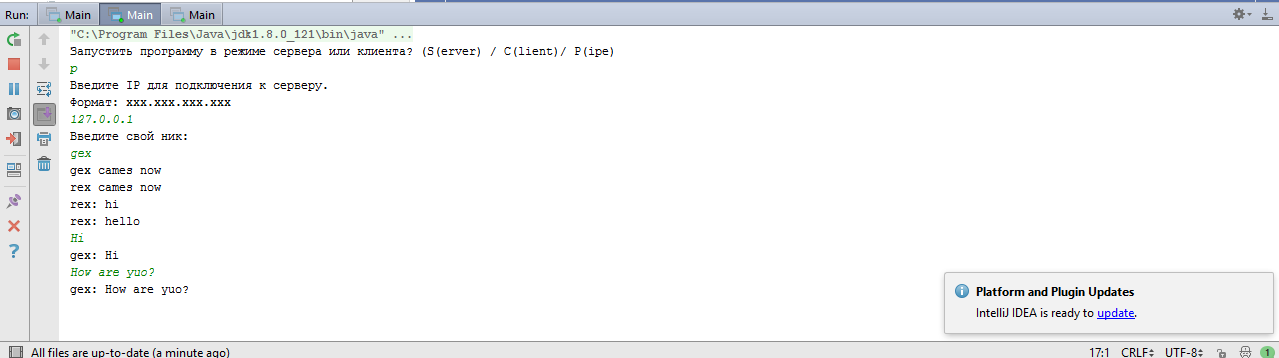
**package** client;  
  
**import** Main.Const;  
  
**import** java.io.\*;  
**import** java.net.Socket;  
**import** java.util.Scanner;  
  
**public class** Client {  
 **private** BufferedReader **in**;  
 **private** PrintWriter **out**;  
 **private** Socket **socket**;  
  
 */\*\*  
 \* Запрашивает у пользователя ник и организовывает обмен сообщениями с  
 \* сервером  
 \*/* **public** Client() {  
 Scanner scan = **new** Scanner(System.***in***);  
  
 System.***out***.println(**"Введите IP для подключения к серверу."**);  
 System.***out***.println(**"Формат: xxx.xxx.xxx.xxx"**); *//127.0.0.1* String ip = scan.nextLine();  
  
 **try** {  
  
 *// Подключаемся в серверу и получаем потоки(in и out) для передачи сообщений* **socket** = **new** Socket(ip, Const.***Port***);  
 **in** = **new** BufferedReader(**new** InputStreamReader(**socket**.getInputStream()));  
 **out** = **new** PrintWriter(**socket**.getOutputStream(), **true**);  
  
 System.***out***.println(**"Введите свой ник:"**);  
 **out**.println(scan.nextLine());  
  
 *// Запускаем вывод всех входящих сообщений в консоль* Resender resend = **new** Resender();  
 resend.start();  
  
 *// Пока пользователь не введёт "exit" отправляем на сервер всё, что  
 // введено из консоли* String str = **""**;  
 **while** (!str.equals(**"exit"**)) {  
 str = scan.nextLine();  
 **out**.println(str);  
 }  
 resend.setStop();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 close();  
 }  
 }  
  
 */\*\*  
 \* Закрывает входной и выходной потоки и сокет  
 \*/* **private void** close() {  
 **try** {  
 **in**.close();  
 **out**.close();  
 **socket**.close();  
 } **catch** (Exception e) {  
 System.***err***.println(**"Потоки не были закрыты!"**);  
 }  
 }  
  
 */\*\*  
 \* Класс в отдельной нити пересылает все сообщения от сервера в консоль.  
 \* Работает пока не будет вызван метод setStop().  
 \*/* **private class** Resender **extends** Thread {  
  
 **private boolean stoped**;  
  
 */\*\*  
 \* Прекращает пересылку сообщений  
 \*/* **public void** setStop() {  
 **stoped** = **true**;  
 }  
  
 */\*\*  
 \* Считывает все сообщения от сервера и печатает их в консоль.  
 \* Останавливается вызовом метода setStop()  
 \*/* @Override  
 **public void** run() {  
 **try** {  
 **while** (!**stoped**) {  
 String str = **in**.readLine();  
 System.***out***.println(str);  
 }  
 } **catch** (IOException e) {  
 System.***err***.println(**"Ошибка при получении сообщения."**);  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

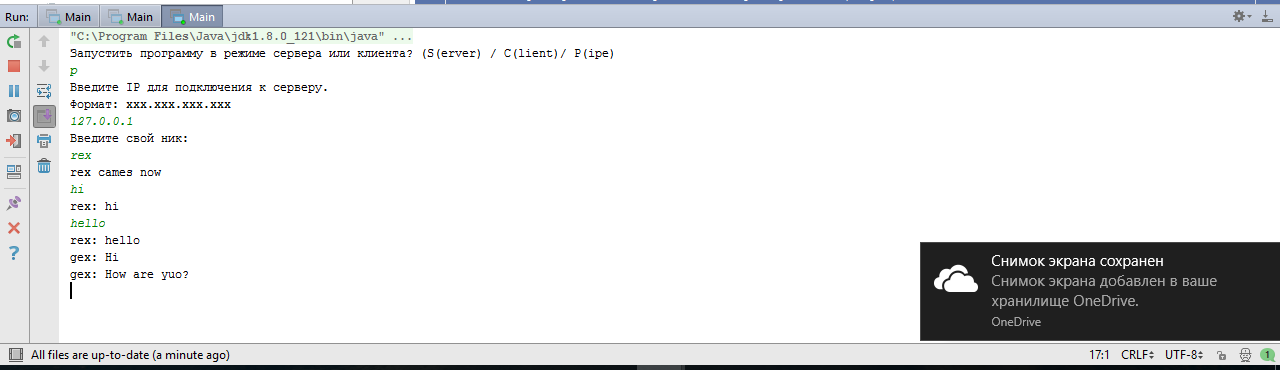
Класс Server

**package** server;  
  
**import** Main.Const;  
  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.io.PrintWriter;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.util.ArrayList;  
**import** java.util.Collections;  
**import** java.util.Iterator;  
**import** java.util.List;  
  
*//Обеспечивает работу программы в режиме сервера***public class** Server {  
  
  
 */\*\*  
 \* Специальная "обёртка" для ArrayList, которая обеспечивает доступ к  
 \* массиву из разных нитей  
 \*/* **private** List<Connection> **connections** =  
 Collections.*synchronizedList*(**new** ArrayList<Connection>());  
 **private** ServerSocket **server**;  
  
 */\*\*  
 \* Конструктор создаёт сервер. Затем для каждого подключения создаётся  
 \* объект Connection и добавляет его в список подключений.  
 \*/* **public** Server() {  
 **try** {  
 **server** = **new** ServerSocket(Const.***Port***);  
  
 **while** (**true**) {  
 Socket socket = **server**.accept();  
  
 *// Создаём объект Connection и добавляем его в список* Connection con = **new** Connection(socket);  
 **connections**.add(con);  
  
 *// Инициализирует нить и запускает метод run(),  
 // которая выполняется одновременно с остальной программой* con.start();  
  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 closeAll();  
 }  
 }  
  
 */\*\*  
 \* Закрывает все потоки всех соединений а также серверный сокет  
 \*/* **private void** closeAll() {  
 **try** {  
 **server**.close();  
  
 *// Перебор всех Connection и вызов метода close() для каждого. Блок  
 // synchronized {} необходим для правильного доступа к одним данным  
 // их разных нитей* **synchronized**(**connections**) {  
 Iterator<Connection> iter = **connections**.iterator();  
 **while**(iter.hasNext()) {  
 ((Connection) iter.next()).close();  
 }  
 }  
 } **catch** (Exception e) {  
 System.***err***.println(**"Потоки не были закрыты!"**);  
 }  
 }  
  
 */\*\*  
 \* Класс содержит данные, относящиеся к конкретному подключению:  
 \* <ul>  
 \* <li>имя пользователя</li>  
 \* <li>сокет</li>  
 \* <li>входной поток BufferedReader</li>  
 \* <li>выходной поток PrintWriter</li>  
 \* </ul>  
 \* Расширяет Thread и в методе run() получает информацию от пользователя и  
 \* пересылает её другим* **private class** Connection **extends** Thread {  
 **private** BufferedReader **in**;  
 **private** PrintWriter **out**;  
 **private** Socket **socket**;  
  
 **private** String **name** = **""**;  
  
 */\*\*  
 \* Инициализирует поля объекта и получает имя пользователя  
 \*  
 \** ***@param socket*** *\* сокет, полученный из server.accept()  
 \*/* **public** Connection(Socket socket) {  
 **this**.**socket** = socket;  
  
 **try** {  
 **in** = **new** BufferedReader(**new** InputStreamReader(  
 socket.getInputStream()));  
 **out** = **new** PrintWriter(socket.getOutputStream(), **true**);  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 close();  
 }  
 }  
  
 */\*\*  
 \* Запрашивает имя пользователя и ожидает от него сообщений. При  
 \* получении каждого сообщения, оно вместе с именем пользователя  
 \* пересылается всем остальным.  
 \*  
 \** ***@see*** *java.lang.Thread#run()  
 \*/* @Override  
 **public void** run() {  
 **try** {  
 **name** = **in**.readLine();  
 *// Отправляем всем клиентам сообщение о том, что зашёл новый пользователь* **synchronized**(**connections**) {  
 Iterator<Connection> iter = **connections**.iterator();  
 **while**(iter.hasNext()) {  
 ((Connection) iter.next()).**out**.println(**name** + **" cames now"**);  
 }  
 }  
  
 String str = **""**;  
 **while** (**true**) {  
 str = **in**.readLine();  
 **if**(str.equals(**"exit"**)) **break**;  
  
 *// Отправляем всем клиентам очередное сообщение* **synchronized**(**connections**) {  
 Iterator<Connection> iter = **connections**.iterator();  
 **while**(iter.hasNext()) {  
 ((Connection) iter.next()).**out**.println(**name** + **": "** + str);  
 }  
 }  
 }  
  
 **synchronized**(**connections**) {  
 Iterator<Connection> iter = **connections**.iterator();  
 **while**(iter.hasNext()) {  
 ((Connection) iter.next()).**out**.println(**name** + **" has left"**);  
 }  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 close();  
 }  
 }  
  
 */\*\*  
 \* Закрывает входной и выходной потоки и сокет  
 \*/* **public void** close() {  
 **try** {  
 **in**.close();  
 **out**.close();  
 **socket**.close();  
  
 *// Если больше не осталось соединений, закрываем всё, что есть и  
 // завершаем работу сервера* **connections**.remove(**this**);  
 **if** (**connections**.size() == 0) {  
 Server.**this**.closeAll();  
 System.*exit*(0);  
 }  
 } **catch** (Exception e) {  
 System.***err***.println(**"Потоки не были закрыты!"**);  
 }  
 }  
 }  
}

**Скриншоты:**

****

****

****

**Лабораторная работа №6.**

**Задание:** Составить приложение, использующее расширенные возможности современных файловых систем.

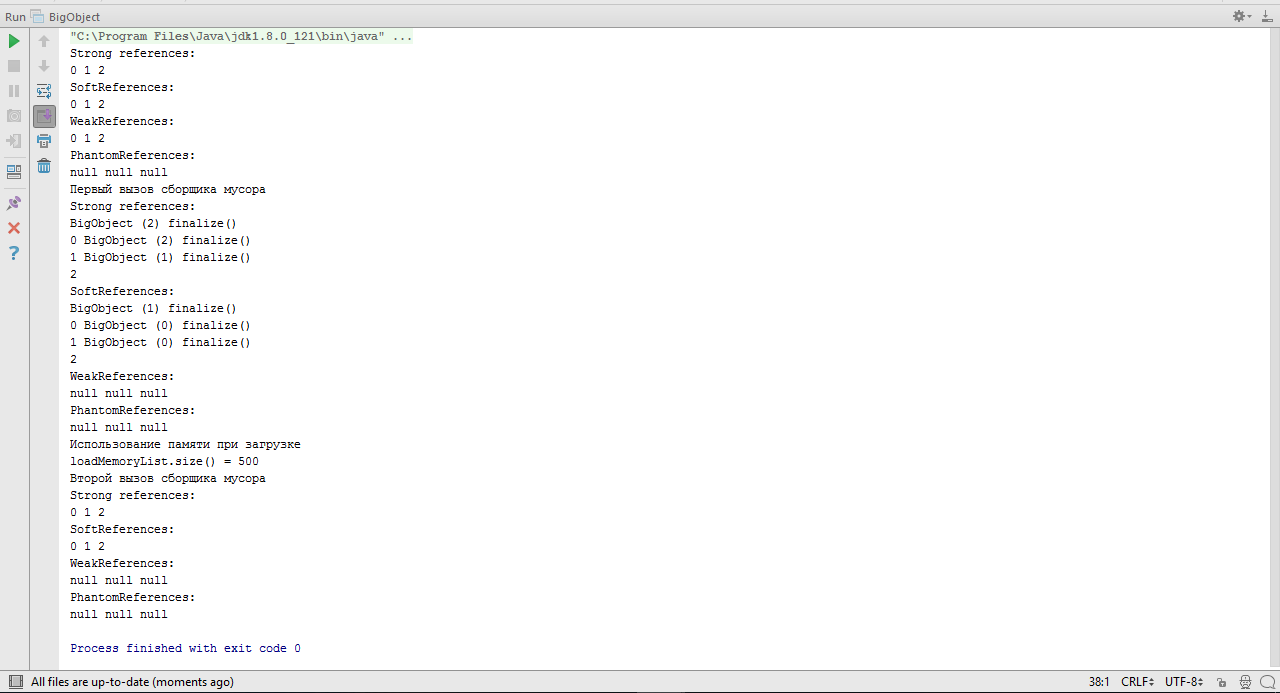
1. Создать файл со случайным содержимым, размером 256 МБ.
2. Вывести объем свободного места на всех доступных томах
3. Создать жесткую ссылку на этот файл. Посмотреть изменение свободного места
4. Создать символическую ссылку на этот файл. Посмотреть изменение свободного места
5. Получить доступ к файлу (прочитать часть содержимого) используя вторую жесткую и символическую ссылки
6. Удалить первое имя файла
7. Вывести объем свободного места. Объяснить результат.
8. Получить доступ к файлу (прочитать часть содержимого) используя вторую жесткую и символическую ссылки. Объяснить результат.
9. Удалить символическую ссылку.
10. Создать в файле по 2й жесткой ссылке 2й NTFS поток, записать туда 256 МБ данных.
11. Вывести размер файла. Вывести размер свободного места на томе.
12. Удалить файл полностью. Вывести размер свободного места на томе. Объяснить результат.

**Код программы:**

Класс Object

**import** java.lang.ref.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**class** BigObject {  
 **private int value**;  
  
  
  
 **public** BigObject(Integer pValue) {  
 **value** = pValue;  
 }  
  
 @Override  
 **protected void** finalize () **throws** Throwable {  
 System.***out***.println(**"BigObject ("** + **value** + **") finalize()"**);  
 }  
  
 @Override  
 **public** String toString () {  
 **return value** + **""**;  
 }  
  
  
 **static** List<BigObject> *strongList*; *// сильные ссылки* **static** List<Reference<BigObject>> *softList*; *// мягкие ссылки* **static** List<Reference<BigObject>> *weakList*; *// слабые ссылки* **static** List<Reference<BigObject>> *phantomList*; *// фантомные ссылки* **static** ReferenceQueue<BigObject> *queue*; *// очередь* **static** List<String> *loadMemoryList*; *// здесь будут храниться ссылки на строки, которые используются для загрузки памяти* **private static void** printLists() {  
 System.***out***.println(**"Strong references: "**);  
 **for** (BigObject bo : *strongList*) System.***out***.print(bo + **" "**);  
 System.***out***.println();  
 System.***out***.println(**"SoftReferences: "**);  
 *printList*(*softList*);  
 System.***out***.println(**"WeakReferences: "**);  
 *printList*(*weakList*);  
 System.***out***.println(**"PhantomReferences: "**);  
 *printList*(*phantomList*);  
 }  
  
 **private static void** printList(List<Reference<BigObject>> pList) {  
 **for** (Reference<BigObject> ref : pList)  
 System.***out***.print(ref.get() + **" "**);  
  
 System.***out***.println();  
 }  
  
 **private static void** init() {  
 *strongList* = **new** ArrayList<BigObject>();  
 *softList* = **new** ArrayList<Reference<BigObject>>();  
 *weakList* = **new** ArrayList<Reference<BigObject>>();  
 *phantomList* = **new** ArrayList<Reference<BigObject>>();  
 *loadMemoryList* = **new** ArrayList<String>();  
 *queue* = **new** ReferenceQueue<BigObject>();  
 **for** (**int** i = 0; i < 3; i++) {  
 *strongList*.add(**new** BigObject(i));  
 *softList*.add(**new** SoftReference<BigObject>(**new** BigObject(i)));  
  
 *weakList*.add(**new** WeakReference<BigObject>(**new** BigObject(i)));  
 *phantomList*.add(**new** PhantomReference<BigObject>(**new** BigObject(i), *queue*));  
 }  
 *printLists*();  
 }  
  
  
 **private static void** loadMemory() {  
 **for** (**int** i = 0; i < 500; i++) {  
 *loadMemoryList*.add(i + **""**);  
 }  
 }  
 **public static void** main(String[] args) {  
  
 *init*();  
 System.*gc*();  
 System.***out***.println(**"Первый вызов сборщика мусора"**);  
 *printLists*();  
 System.***out***.println(**"Использование памяти при загрузке"**);  
 *loadMemory*(); *// загрузка памяти* System.***out***.println(**"loadMemoryList.size() = "** + *loadMemoryList*.size());  
 System.*gc*();  
 System.***out***.println(**"Второй вызов сборщика мусора"**);  
 *printLists*();  
 }  
}

**Скриншоты:**

****

**Лабораторная работа №7.**

**Задание:** Составить приложение, использующее сервисы безопасности операционной системы.

1. Выбрать файл. Вывести имя владельца.
2. Вывести права на доступ к выбранному файлу.
3. Запретить любой доступ к файлу всем, кроме владельца
4. Запретить любой доступ к файлу вообще
5. Прочитать файл, к которому запрещен доступ
6. Выключить питание компьютера

**Код программы:**

Класс CreateFile

**import** java.io.File;  
**import** java.io.IOException;  
**public class** CreateFile {  
 **public static void** main (String args [])**throws** IOException{  
 String absoluteFilePath = **"file.txt"**;  
 File file = **new** File(absoluteFilePath);  
 **if**(file.createNewFile()){  
 System.***out***.println(absoluteFilePath + **" Файл создан"**);  
 } **else** {  
 System.***out***.println(**"Файл "** + absoluteFilePath + **" уже существует"**);  
  
 }  
 }  
}

Класс Main

**import** java.io.File;  
**import** java.io.IOException;  
**public class** Main {  
 **private static final** String ***FILENAME*** = **"D:\\ProgramFiles\\Programm\\OC\\lab7\\file.txt"**;  
  
 **public static void** main(String[] args) {  
  
 File file = **new** File(***FILENAME***);  
 System.***out***.println(**"Имя файла: "** + ***FILENAME***);  
  
 **if** (file.exists()) {  
 System.***out***.println(**"Файл существует."**);  
  
 file.setExecutable(**true**);  
 file.setReadable(**true**);  
 file.setWritable(**true**);  
  
 System.***out***.println(**"Файл доступен для выполнения: "** + file.canExecute());  
 System.***out***.println(**"Файл доступен для чтения: "** + file.canRead());  
 System.***out***.println(**"Файл доступен для записи: "** + file.canWrite());  
 }  
  
 System.***out***.println();  
 System.***out***.println(**"Установка прав на файл."**);  
  
 file.setExecutable(**true**);  
 file.setReadable(**true**);  
 file.setWritable(**false**);  
  
 System.***out***.println(**"Файл доступен для выполнения: "** + file.canExecute());  
 System.***out***.println(**"Файл доступен для чтения: "** + file.canRead());  
 System.***out***.println(**"Файл доступен для записи: "** + file.canWrite());  
  
 }  
}

Класс Turn off

**import** java.io.IOException;  
**public class** Turn\_off {  
 **public static void** main(String arg[]) **throws** IOException {  
 Runtime runtime = Runtime.*getRuntime*();  
 Process proc = runtime.exec(**"shutdown -s -t 0"**);  
 System.*exit*(0);  
 }  
}

**Скриншоты:**

