|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации |
| Федеральное государственное автономное образовательное учреждение высшего образования |
| «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина» |
| Институт радиоэлектроники и информационных технологий – РтФ |
|  |
|  | |

**ОТЧЕТ**

О ВЫПОЛНЕНИИ

ЛАБОРАТОНОЙ РАБОТЫ № 6

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-130971 Е.В. Треглазов

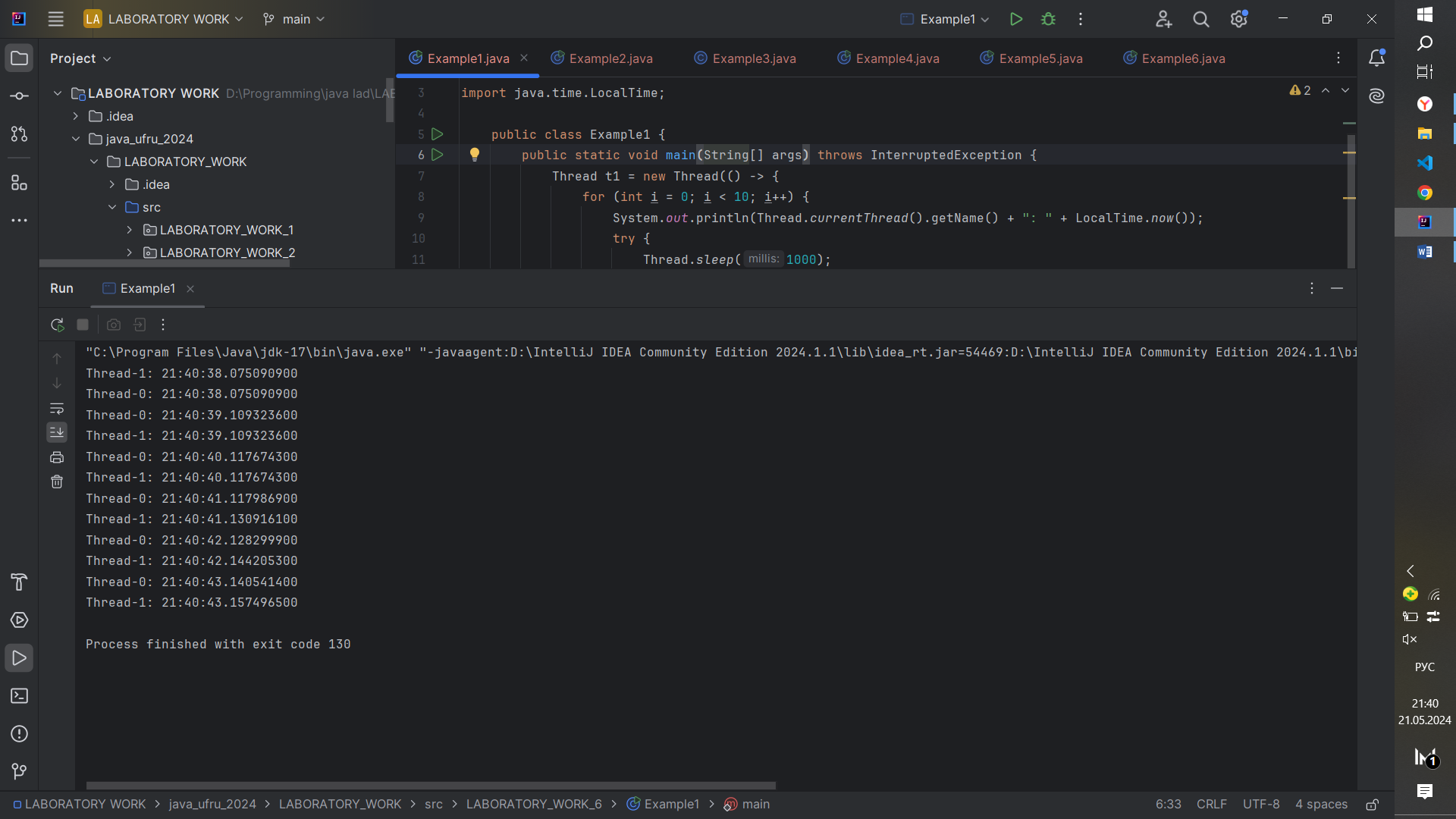
Екатеринбург 2024

Репозиторий: https://github.com/kuk86kuk/java\_ufru\_2024.

Example1

package LABORATORY\_WORK\_6;  
  
import java.time.LocalTime;  
  
 public class Example1 {  
 public static void main(String[] args) throws InterruptedException {  
 Thread t1 = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println(Thread.*currentThread*().getName() + ": " + LocalTime.*now*());  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 Thread t2 = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println(Thread.*currentThread*().getName() + ": " + LocalTime.*now*());  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 t1.start();  
 t2.start();  
 t1.join();  
 t2.join();  
 }  
 }

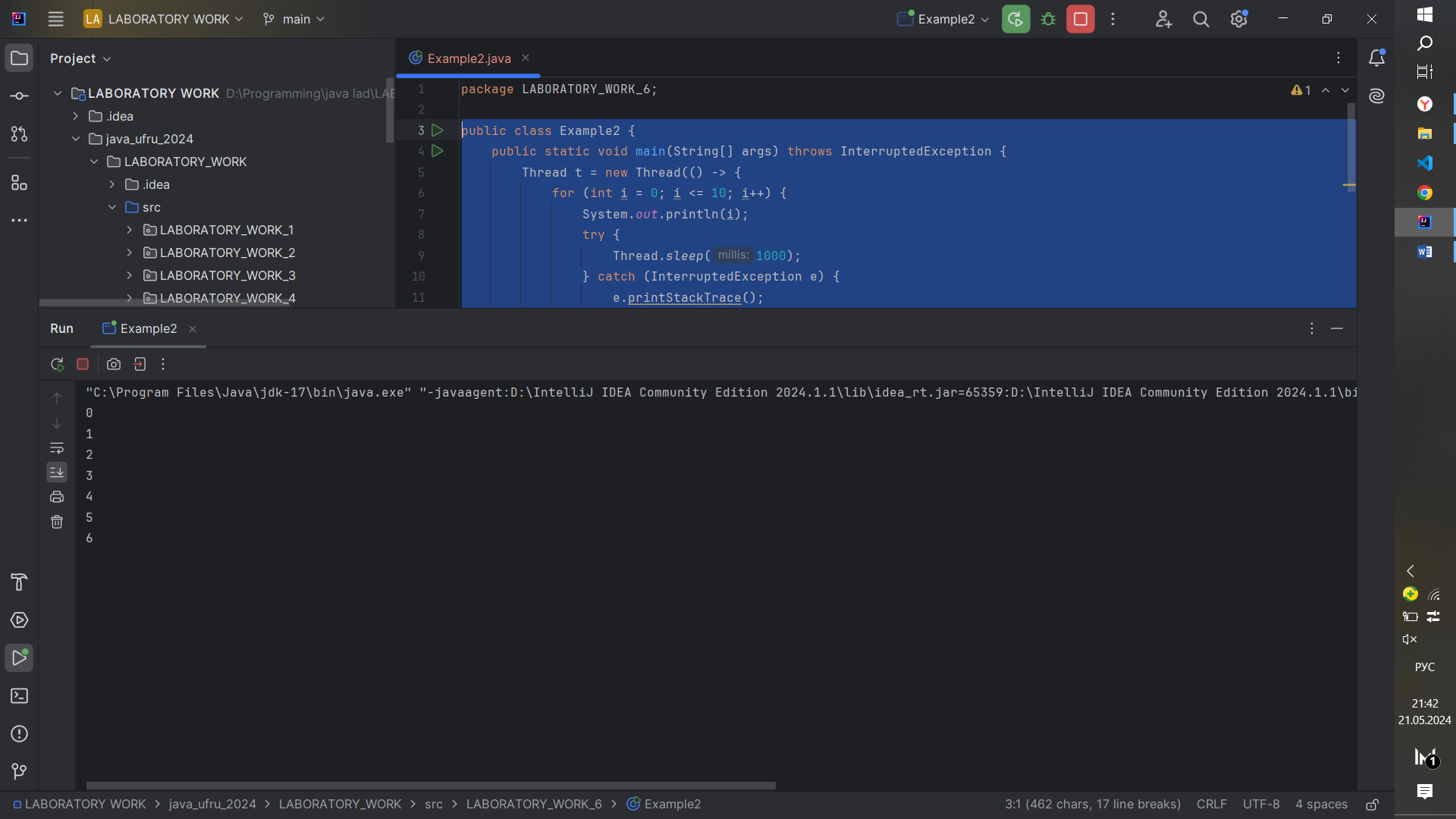
В коде демонстрируется использование многопоточности в Java для выполнения двух параллельных задач, каждая из которых выводит время на консоль с интервалом в одну секунду.



Example2

public class Example2 {  
 public static void main(String[] args) throws InterruptedException {  
 Thread t = new Thread(() -> {  
 for (int i = 0; i <= 10; i++) {  
 System.*out*.println(i);  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 t.start();  
 t.join();  
 }  
}

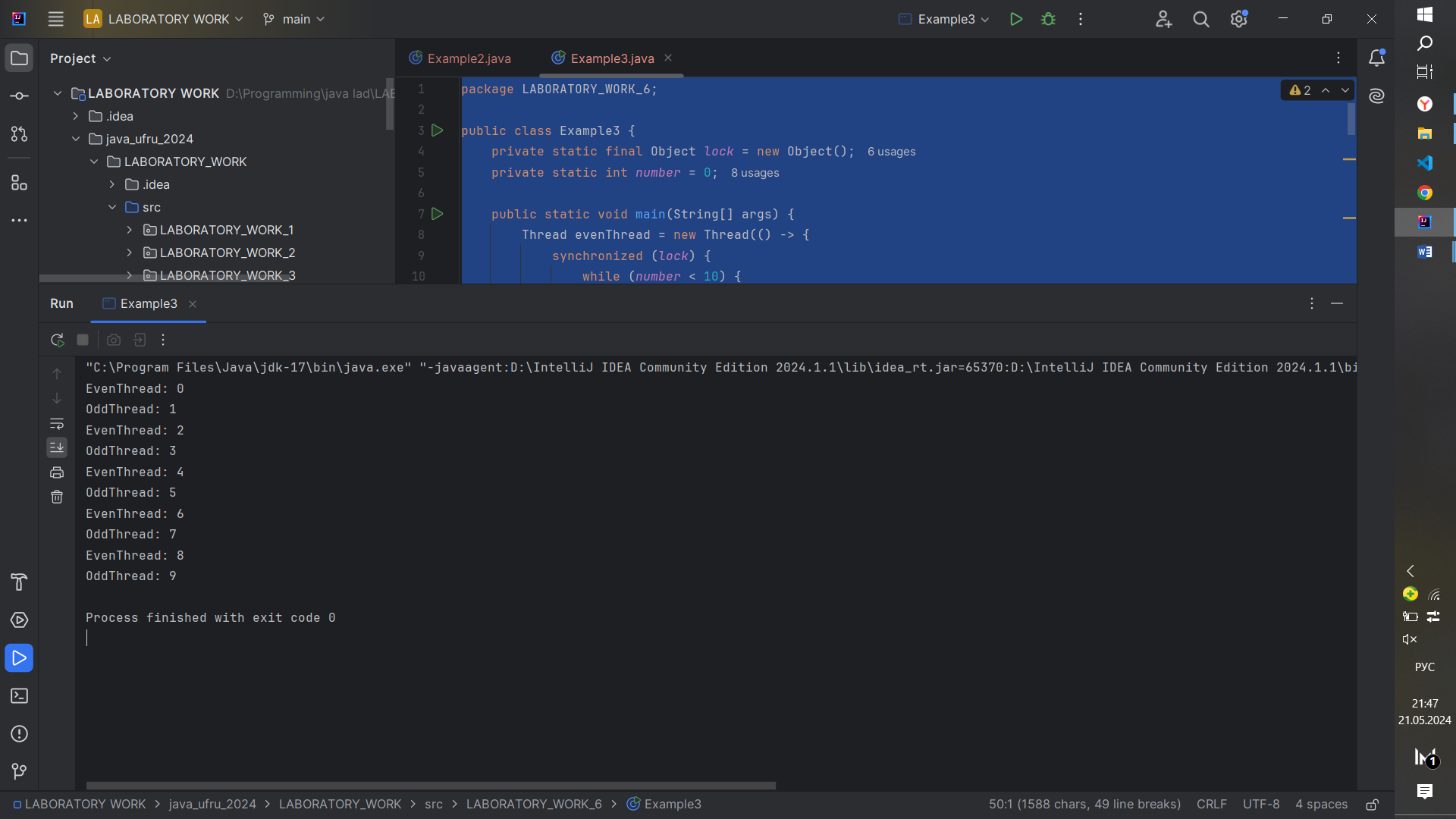
В коде представлено использования отдельного потока для выполнения задачи, которая заключается в последовательном выводе чисел от 0 до 10 на консоль с интервалом в одну секунду между каждым выводом.



Example3

package LABORATORY\_WORK\_6;  
  
public class Example3 {  
 private static final Object *lock* = new Object();  
 private static int *number* = 0;  
  
 public static void main(String[] args) {  
 Thread evenThread = new Thread(() -> {  
 synchronized (*lock*) {  
 while (*number* < 10) {  
 if (*number* % 2 == 0) {  
 System.*out*.println(Thread.*currentThread*().getName() + ": " + *number*);  
 *number*++;  
 } else {  
 try {  
 *lock*.wait();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 *lock*.notifyAll();  
 }  
 }  
 });  
  
 Thread oddThread = new Thread(() -> {  
 synchronized (*lock*) {  
 while (*number* < 10) {  
 if (*number* % 2 != 0) {  
 System.*out*.println(Thread.*currentThread*().getName() + ": " + *number*);  
 *number*++;  
 } else {  
 try {  
 *lock*.wait();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 *lock*.notifyAll();  
 }  
 }  
 });  
  
 evenThread.setName("EvenThread");  
 oddThread.setName("OddThread");  
 evenThread.start();  
 oddThread.start();  
 }  
}

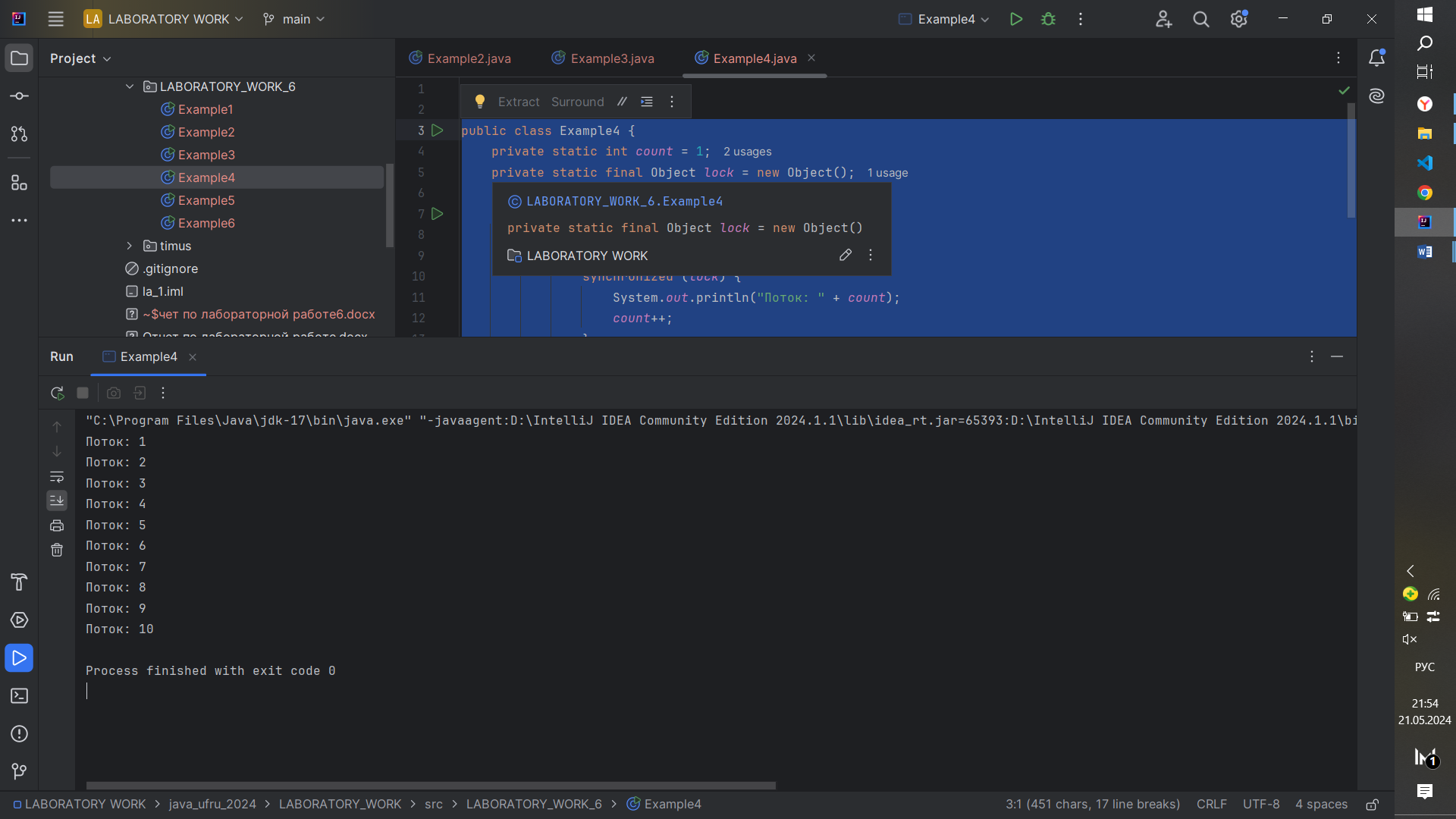
В коде представлен пример использования синхронизации и механизмов ожидания (wait) и уведомления (notifyAll) для управления выводом четных и нечетных чисел на консоль двумя потоками.



Example4

public class Example4 {  
 private static int *count* = 1;  
 private static final Object *lock* = new Object();  
  
 public static void main(String[] args) {  
 for (int i = 1; i <= 10; i++) {  
 Thread thread = new Thread(() -> {  
 synchronized (*lock*) {  
 System.*out*.println("Поток: " + *count*);  
 *count*++;  
 }  
 });  
  
 thread.start();  
 }  
 }  
}

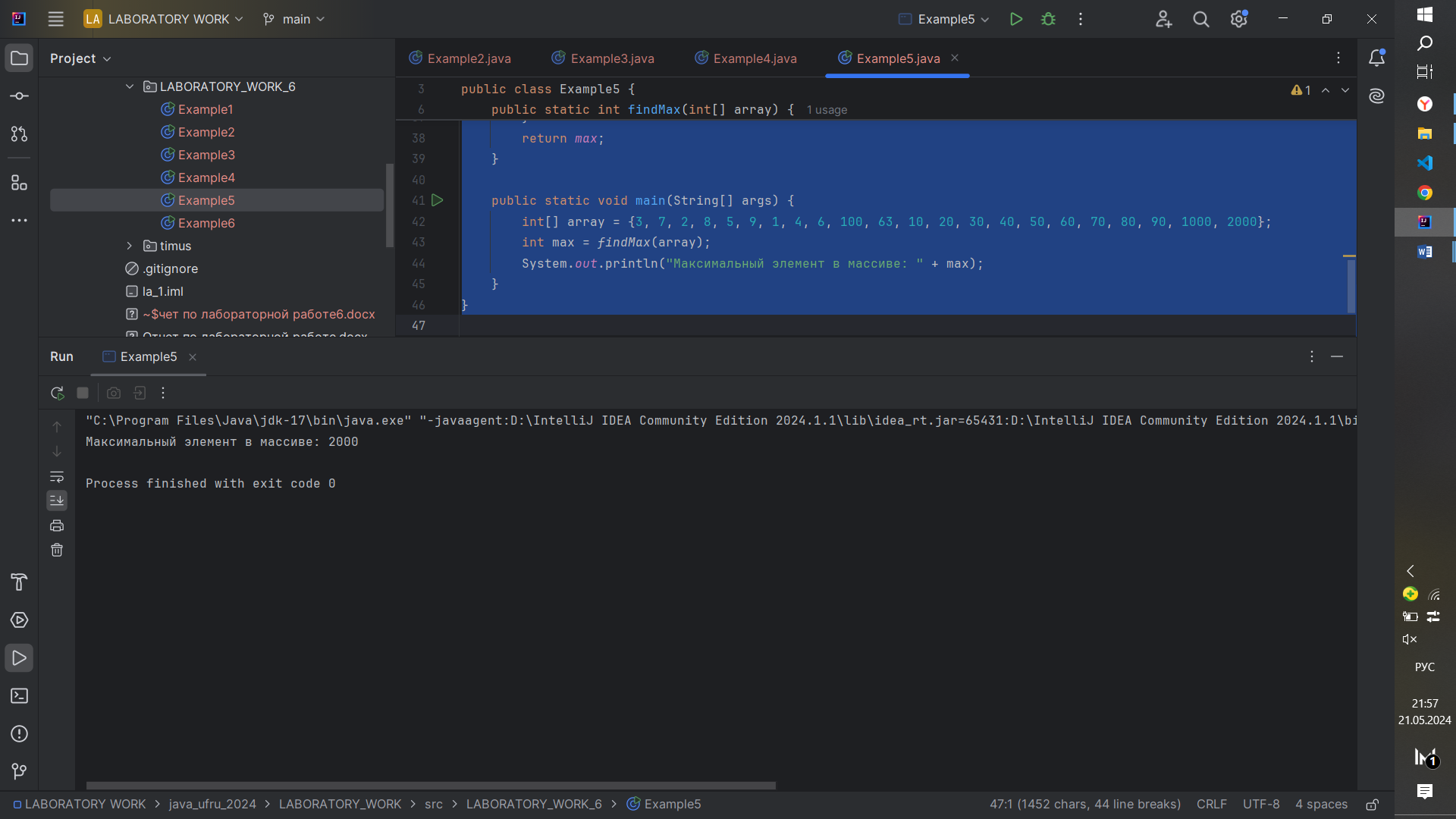
В данном коде представлен пример создания и запуска 10 потоков, каждый из которых выводит свой порядковый номер на консоль.



Example5

public class Example5 {  
 private static int *max* = Integer.*MIN\_VALUE*;  
  
 public static int findMax(int[] array) {  
 int numThreads = Runtime.*getRuntime*().availableProcessors();  
 Thread[] threads = new Thread[numThreads];  
 int size = array.length / numThreads;  
  
 for (int i = 0; i < numThreads; i++) {  
 final int start = i \* size;  
 final int end = (i == numThreads - 1) ? array.length : (i + 1) \* size;  
 threads[i] = new Thread(() -> {  
 int localMax = Integer.*MIN\_VALUE*;  
 for (int j = start; j < end; j++) {  
 if (array[j] > localMax) {  
 localMax = array[j];  
 }  
 }  
 synchronized (Example5.class) {  
 if (localMax > *max*) {  
 *max* = localMax;  
 }  
 }  
 });  
  
 threads[i].start();  
 }  
  
 for (Thread thread : threads) {  
 try {  
 thread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 return *max*;  
 }  
  
 public static void main(String[] args) {  
 int[] array = {3, 7, 2, 8, 5, 9, 1, 4, 6, 100, 63, 10, 20, 30, 40, 50, 60, 70, 80, 90, 1000, 2000};  
 int max = *findMax*(array);  
 System.*out*.println("Максимальный элемент в массиве: " + max);  
 }  
}

В данном коде представлен пример параллельного поиска максимального элемента в массиве с использованием нескольких потоков.



Example6

public class Example6 {  
 private static int *sum* = 0;  
  
 public static int sum(int[] array) {  
 int numThreads = Runtime.*getRuntime*().availableProcessors();  
 Thread[] threads = new Thread[numThreads];  
 int size = array.length / numThreads;  
  
 for (int i = 0; i < numThreads; i++) {  
 final int start = i \* size;  
 final int end = (i == numThreads - 1) ? array.length : (i + 1) \* size;  
 threads[i] = new Thread(() -> {  
 int localSum = 0;  
 for (int j = start; j < end; j++) {  
 localSum += array[j];  
 }  
 synchronized (Example6.class) {  
 *sum* += localSum;  
 }  
 });  
  
 threads[i].start();  
 }  
  
 for (Thread thread : threads) {  
 try {  
 thread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 return *sum*;  
 }  
  
 public static void main(String[] args) {  
 int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 int sum = *sum*(array);  
 System.*out*.println("Сумма элементов в массиве: " + sum);  
 }  
}

В данном коде представлен пример параллельного вычисления суммы элементов в массиве с использованием нескольких потоков.

