Notes.

(1) In Jenkins the project build timeout is set to 15 minutes. After this time, the task will stop executing and you will receive a message about aborted build.

(2) Demo class should be placed in the root package, it should demonstrate the functionality of all the subtasks (Jenkins will call this class).

(3) For solving each X subtask, it is guaranteed that several threads will be used:

\* a thread that executes the method PartX.main (it’s the same thread that executes the Demo.main);

\* a certain amount of child threads.

Terminate the child threads BEFORE the PartX.main method finishes its work (it’s required for the tests that Jenkins runs).

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

Task 1

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice5.Part1

———————————————————————————————-

Create a child thread that, during about 2 seconds, would print out its name every 1/2 seconds. The total count of outputs per child equals 4.

The total time of executing this task equals 4 seconds (2 seconds - executing the first child, 2 seconds - executing the second child)

The first output to console must occur immediately after launch. Each next output must occur after 1/2 seconds after the previous output.

Implement it in two ways:

\* using an extension of the Thread class;

\* using an implementation of the Runnable interface.

First, one implementation should be executed, then, after it has been finished, the other implementation should be executed.

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

Task 2

———————————————————————————————-

Package name: com.epam.rd.java.basic.practice5

Class names: Part2, Spam

———————————————————————————————-

Create Spam class that accepts an array of messages and a сoherent array of time intervals in milliseconds in its constructor, and simultaneously prints out the corresponding messages in the given time intervals. On Enter click, the application should terminate its work (this functionality should be placed in the Spam.main method).

Recommended structure of the Spam class:

————————————————

public class Spam {

  private Thread[] threads;

  public Spam(String[], int[]) {…}

  public void start() {…}

  public void stop() {…}

  private static class Worker extends Thread {…}

  public static void main(String[] args) {...}

}

————————————————

During demonstration of the functionality, mock Enter click in every 2 seconds (this functionality is to be placed in the Part2.main method).

Input data (an array of messages and an array of intervals) are to be written in the code of the Spam class. The quantity of the elements in each of the arrays should be at least 2, you can take them from the example:

Example of the input data (Spam class)

————————————————

String[] messages = new String[] { "@@@", "bbbbbbb" };

int[] times = new int[] { 333, 222 };

————————————————

Additional task information.

(1) Do not use daemon threads as the execution of Part2 goes along with the other tasks in the package, daemons threads will not terminate their work until the Demo.main method finishes its execution.

(2) In order to track the Enter click, it’s enough to read the console input and analyze the content. If the reading is implemented using Scanner / BufferedReader classes, then an empty line will be the sign of the Enter click that is returned, respectively, by Scanner#nextLine() / BufferedReader#readLine() methods.

(3) The algorithm of the console input mock (Part2.main method):

————————————————

(a) substitute the system input stream for your own:

System.setIn(YOUR\_OWN\_INPUT\_STREAM);

(b) call Spam.main in a separate thread:

Thread t = new Thread() { public void run() {Spam.main(null);}};

  t.start();

(c) wait until Spam.main terminates its work:

t.join();

(d) restore the system stream:

System.setIn(CAСHED\_VALUE\_OF\_SYSTEM\_IN)

————————————————

(4) For implementation of your own input stream it’s appropriate to create a class that extends java.io.InputStream abstract class. However, you will need to implement the only abstract method of this class:

————————————————

public abstract int read() throws IOException;  ————————————————

All the methods from the InputStream class (along with all its child classes methods) that read bytes from some data resource, in the end, call the ‘read’ method. It’s enough to implement a pause in this method during the first call that will make the execution thread, calling the ‘read’ method, wait. The ‘read’ method should sequentially return bytes that correspond to the line terminator, after that it needs to return -1 only (the sign that there is no data in the input stream anymore).

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

 Task 3

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice5.Part3

———————————————————————————————-

Create a class with two separate counters.

Create several equal threads, each of them repeats the following:

\* compares the value of the counters and prints out the result of the comparison;

\* increments the first counter;

\* sleeps for 100 milliseconds;

\* increments the second counter.

Compare the program functionality provided that the code is synchronized and not synchronized.

Implement the following scheme:

\* at first, a not synchronized code gets executed.

\* after its termination, the synchronized code gets executed.

The resulting output should be small, not more than a few dozen lines.

The recommended structure of the class:

 ————————————————

public class Part3 {

   private int counter;

   private int counter2;

  /\*\*

\* Should create {@code numberOfThreads} threads

\* that will {@code numberOfIterations} times

\* print "counter == counter2" and increase counters.

\* Between increasing first and second counter

\* must be delay equals 100 milliseconds.

\*

\* This method should wait until threads will finish their work.

\*/

   public void compare() {…}

  /\*\*

\* Should create {@code numberOfThreads} threads

\* that will {@code numberOfIterations} times **synchronously**

\* print "counter == counter2" and increase counters.

\* Between increasing first and second counter

\* must be delay equals 100 milliseconds.

\*

\* This method should wait until threads will finish their work.

\*/

   public void compareSync() {…}

   public static void main(String[] args) {...}

   public Part3(int numberOfThreads, int numberOfIterations) {...}

}

————————————————

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

Task 4

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice5.Part4

The input data should be uploaded from the ‘part4.txt’ file

———————————————————————————————-

Parallelize the task of the search of the maximum value in the matrix of the whole numbers MxN (upload from the file) using M threads. Additionally, resolve the task of search of the minimum value without parallelizing. Print out the result and the time of the code execution (in milliseconds) for both variants.

it is required to set a delay of 1 millisecond for every operation of comparison.

Stick to the following output format:

————————————————

MAX

TIME

MAX2

TIME2

————————————————

where MAX, TIME - the maximum value and time of the search in the multithreading implementation of the task; MAX2, TIME2 - the maximum value and time of the search in the single-threading implementation of the task.

The application functionality should be checked using a matrix 4x100 of random numbers.

The content of the ‘part4.txt’ file should be a readable matrix, the numbers are separated with a space, the lines are separated with a line terminator.

Notes.

(1) The line terminator should be platform independent (take it into account when reading data).

(2) The input file ‘part4.txt’ can be created in any convenient way before running Part4.main (for example, using some helper class). The application (Part4.main) does not create ‘part4.txt’ file and does not modify its content, the size of the matrix should be determined by the ‘part4.txt’ content.

Example of the ‘part4.txt’ content for M=5, N=20

————————————————

706 575 855 882 595 778 477 602 147 467 693 793 120 384 256 866 548 367 910 848

206 232 632 315 743 823 620 111 279 548 210 393 791 815 519 768 168 484 780 705

709 127 900 171 189 590 563 317 600 975 892 296 166 353 863 312 399 872 964 591

302 869 679 157 419 485 325 290 739 149 407 648 688 474 311 177 318 611 348 557

559 283 171 352 698 759 384 822 598 410 802 293 962 859 812 153 436 392 869 167

————————————————

Example of the resulting output

————————————————

975

26

975

115

————————————————

For the input data M=4, N=100, the time of the parallelized search should be approximately 4 times as little as the time of the single threaded search.

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

Task 5

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice5.Part5

The output data should be uploaded to the ‘part5.txt’ file

———————————————————————————————-

Create k threads that simultaneously write characters into the same file:

————————————————

the first thread writes a digit 0 exactly 20 times on the first line of the file;

the second thread writes a digit 1 exactly 20 times on the second line of the file;

…..

the tenth thread writes a digit 9 exactly 20 times on the tenth line of the file;

————————————————

Requirements to the implementation.

(1) It is required to set a 1 millisecond pause for writing of each digit.

(2) Use the RandomAccessFile for writing data to the file.

(3) You can use not more than one object of the RandomAccessFile class!

(4) If the file, where you are going to write data to, already exists, it should be deleted before starting the work.

(5) The main thread, after starting the child threads, should wait until they finish execution, after that, it should print the file content to the console.

Notes.

(1) The method RandomAccessFile#seek(long) allows to move the pointer inside the file. Each thread should know at what place of the file it should write the information. As in the the requirements there is a term ‘line’, every thread output should end with a line terminator that should be printed in a cross platform way.

(2) In order to write a certain digit, you can use an expression '0'+ n, where n is a digit from 0 to 9 inclusive.

Moving the pointer inside the file and writing the data should be synchronized.

Result of application work

————————————————

00000000000000000000

11111111111111111111

22222222222222222222

33333333333333333333

44444444444444444444

55555555555555555555

66666666666666666666

77777777777777777777

88888888888888888888

99999999999999999999

————————————————