```
// $Header: /home/cvs/t21617/samuel.riedo/Hello/src/Magrathea.java,v 1.6 2017-03-27 07:29:45 samuel.riedo Exp $

import java.util.Arrays;
import java.util.Collections;
import java.util.concurrent.Semaphore;

/**
 * Simulation of the interactions between Bezerkis and Vogons. Bezerkis and
 * Vogons are simulated via processes and synchronization is provided by
 * Semaphores. Each Bezerki process wait to meet one Vogon, whereas each Vogon
 * process must meet two Bezerkis. If there isn't enough Bezerkis or Vogons so
 * everyone can meet the correct number of the other race, processes are
 * interrupted.
 *
 * @param string[],
 *           in the following order: number of Bezerki threads, number of
 *           Bezerki threads iterations, number of Vogon threads, number of
 *           Vogon threads iterations.
 */
public class Magrathea {
    private static int vogonIterations      = 17;     // Number of Vogon iterations
    private static int vogonNumber          = 41;     // Number of Vogon threads
    private static int bezerkiIterations    = 37;     // Number of Bezerki iterations
    private static int bezerkiNumber        = 43;     // Number of Bezerki threads

    private static boolean  programTerminate        = false;    // True when simulation is finish.
    private static int      activeVogonThreads      = 0;        // Number of active Vogon threads.  Should be volatile
    private static int      activeBezerkiThreads    = 0;        // Number of active Bezerki threads.
    private static boolean  oneBezerkiMet           = false;    // Indicates whether a Vogon is waiting
                                                                // to meet a second Bezerki.
    private static Thread[] bezerki = new Thread[bezerkiNumber];            // Countain all Bezerki threads.
    private static Thread[] vogon   = new Thread[vogonNumber];              // Countain all Vogon threads.

    private static Semaphore mutex                 = new Semaphore(1, true); // Universal Mutex
    private static Semaphore secondBezerki         = new Semaphore(0, true); // Semaphore blocking a Vogon thread
                                                                             // waiting to meet a second Bezerki.
    private static Semaphore waitForVogon          = new Semaphore(0, true); // Semaphore blocking Bezerki thread
                                                                             // waiting to meet a Vogon.
    private static Semaphore waitForBezerki        = new Semaphore(0, true); // Semaphore blocking Vogon thread
                                                                             // waiting to meet a Bezerki.
    private static Semaphore detectEnd             = new Semaphore(0, true); // Semaphore used in main method to stop
                                                                             // the simulation if there is only active
                                                                             // threads from one race.
    /**
     * main method. Created and start Bezerki and Vogon threads. If there isn't
     * enough Bezerki or Vogon to terminate the simulation, interrupt the remaining
     * threads.
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Program start.");
        int argsl = args.length;
        switch (argsl) {
            case 4:
                vogonIterations = Integer.valueOf(args[--argsl]);
            case 3:
                vogonNumber = Integer.valueOf(args[--argsl]);
            case 2:
                bezerkiIterations = Integer.valueOf(args[--argsl]);
            case 1:
                bezerkiNumber = Integer.valueOf(args[--argsl]);
        }

        createThreads();
        startThreads();
        waitOnThreads();    In addition, you should join() each thread...
        terminateThreads();

        System.out.println("-----------------------------------------");      // End of simulation.
        System.out.println("Simulation successfully ended.");
    }

    /**
     * Create all threads in vogon[] and bezerki[].
     */
    private static void createThreads(){
        System.out.println("Creating threads...");                          // Create threads.

        for (int i = 0; i < vogon.length; i++) {
            vogon[i] = new Vogon(i);
        }
        for (int i = 0; i < bezerki.length; i++) {
            bezerki[i] = new Bezerki(i);
        }
    }

    /**
     * Shuffle and start all threads in vogon[] and bezerki[].
     */
    private static void startThreads(){
        System.out.println("Shuffling threads...");                         // Shuffle threads.
        Collections.shuffle(Arrays.asList(bezerki));
        Collections.shuffle(Arrays.asList(vogon));

        System.out.println("Starting threads...");                          // Start threads.
        System.out.println("-----------------------------------------");
        for (int i = 0; i < Math.max(bezerkiNumber, vogonNumber); i++) {
            if (i < bezerkiNumber)
                bezerki[i].start();
            if (i < vogonNumber)
                vogon[i].start();
        }
    }

    /**
     * Terminate all threads in vongon[] and bezerki[].
     * @throws InterruptedException
     */
    private static void terminateThreads() throws InterruptedException {
        for (int i = 0; i < Math.max(bezerkiNumber, vogonNumber); i++) {          // Interrupt all remaining Bezerki
            if (i < bezerkiNumber){                                               // and Vogon threads.
                bezerki[i].interrupt();
            }
            if (i < vogonNumber){
                vogon[i].interrupt();
            }
        }
    }

    /**
     * Wait on all threads in vongon[] and bezerki[].
     * If there is zero active threads from one race, exit.
     */
    private static void waitOnThreads(){
        do {                                                                      // Wait for all threads to terminate.
            detectEnd.acquireUninterruptibly();                                   // If there is only one race active
            mutex.acquireUninterruptibly();                                       // threads, break.
            if (activeBezerkiThreads == 0 && activeVogonThreads > 0) {
                programTerminate=true;
                mutex.release();
                break;
            }
            if (activeVogonThreads == 0 && activeBezerkiThreads > 0) {
                programTerminate=true;
                mutex.release();
                break;
            }
            mutex.release();
        } while (activeVogonThreads > 0 && activeBezerkiThreads > 0);
    }

    /**
     * Simulation the behavior of an alien race called Vogon. This alien must
     * go on a planet called Magrathea and meet two Bezerkis aliens before
     * leaving.
     */
    static class Vogon extends Thread {

        private int id;                                                           // Vogon thread unique ID.

        public Vogon(int id) {
            this.id = id;
        }
        /**
         * Simulate meeting between this thread and two Bezerki thread.
         * This processus is done vogonIterations's time.
         */
        @Override
        public void run() {   You need to do random sleeps (in nanoseconds) to mix up thread execution
            try{
                mutex.acquire();
                activeVogonThreads++;
                mutex.release();

                for (int i = 0; i < vogonIterations; i++) {
                    System.out.printf("Vogon %d strolling on Magrathea \n", id);
                    waitForVogon.release();
                    waitForBezerki.acquire();                                    // Wait for a Berzerki.

                    mutex.acquire();
                    System.out.printf("Vogon %d met one bezerki.\n", id); Why is this considered a critical section ?
                    waitForVogon.release();
                    mutex.release();

                    detectEnd.release();<--- this is wrong.  You're activating a process when there is no end to
                    secondBezerki.acquire();                                     // Wait for another Berzerki. detect !

                    System.out.printf("Vogon %d met two bezerki.\n", id);
                    System.out.printf("Vogon %d leaving Magrathea.\n", id);
                }

                mutex.acquire();                                                 // All iterations done.
                activeVogonThreads--;
                mutex.release();
                detectEnd.release(); <--- Here, it's OK.
            }
            catch(InterruptedException e){
                if(programTerminate && activeVogonThreads>1)
                    System.out.println("Thread "+this.id+" interrupted, no enough Bezerki to continue.");
            }
        }
    }

    /**
     * Simulation the behavior of an alien race called Bezerki. This alien must
     * go on a planet called Magrathea and meet one Vogon aliens before
     * leaving.
     */
    static class Bezerki extends Thread {

        private int id;                                                          // Bezerki thread unique ID.

        public Bezerki(int id) {
            this.id = id;
        }

        /**
         * Simulate meeting between this thread and one Vogon thread.
         * This processus is done bezerkiIterations's time.
         */
        @Override
        public void run() {
            try{
                mutex.acquire();
                activeBezerkiThreads++;
                mutex.release();

                for (int i = 0; i < bezerkiIterations; i++) {
                    System.out.printf("Bezerki %d strolling on Magrathea\n", id);

                    waitForVogon.acquire();
                    mutex.acquire();
                                                   +----- simpler to say "if (oneBezerkiMet)"
                    if (oneBezerkiMet == true) {<--+                               // Check if there is a Vogon
                        oneBezerkiMet = false;                                      // waiting to meet a second Bezerki.
                        secondBezerki.release();
                    } else {
                        waitForBezerki.release();
                        oneBezerkiMet = true;
                    }

                    System.out.printf("Bezerki %d met one Vogon.\n", id);
                    System.out.printf("Bezerki %d leaving Magrathea.\n", id);
                    mutex.release();
                }

                mutex.acquire();                                                // All iterations done.
                activeBezerkiThreads--;
                mutex.release();
                detectEnd.release();
            }
            catch(InterruptedException e){
                if(programTerminate && activeBezerkiThreads>1)
                    System.out.println("Thread "+this.id+" interrupted, no enough Vogon to continue.");
            }
        }
    }
}

/**
 * $Log: Magrathea.java,v $                 Please put ** in front of $Log, as indicated in 10 commandments
 * Revision 1.6  2017-03-27 07:29:45  samuel.riedo
 * Comments grammar correction.
 *
 * Revision 1.5  2017-03-27 07:10:52  samuel.riedo
 * Update all semaphore to use aquire instead of aquireUninterruptibly().
 * The programme can now stop without a system.exit()
 *
 * Revision 1.4  2017-03-26 20:36:54  samuel.riedo
 * Typography
 *                                                      I expect more comments in the future.  Beware next lab !
 * Revision 1.3  2017-03-26 17:56:39  samuel.riedo
 * Delete unused variable. (randomThreadsSleep)
 *
 * Revision 1.2  2017-03-26 17:47:45  samuel.riedo
 * Split main method in Magrathea into several sub methods.
 *
 * Revision 1.1  2017-03-26 17:35:40  samuel.riedo
 * Move to default package.
 *
 * Revision 1.4  2017-03-26 12:28:38  samuel.riedo
 * Updating comments.
 * Revision 1.3 2017-03-25 12:08:58 samuel.riedo
 * Functional version, only need to add a way to terminate program when there
 * isn't enough Bezerki to meet all Vogon or vice versa.
 * Revision 1.2 2017-03-20 09:38:24 samuel.riedo
 * Maybe first functional version. Need more deep tests.
 * Revision 1.1 2017-03-06 10:17:06 samuel.riedo File created
 */
```

```
// $Header: /home/cvs/t21617/samuel.riedo/Hello/src/Main.java,v 1.1 2017-02-27 10:21:17 samuel.riedo Exp $

public class Main {

   public static void main(String... args){
      System.out.println("Hello World");
   }
}
/*
** $Log: Main.java,v $
** Revision 1.1  2017-02-27 10:21:17  samuel.riedo
** clique droit -> team -> commit
**
*/

```

Obviously a mistake

```
java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server,
                  because you are running on a server-class machine.


    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  include/exclude user private JREs in the version search
    -? -help      print this help message
    -X            print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions with specified granularity
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g. -agentlib:hprof
                  see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
                  show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for more details.
########################################..... 0 lines skipped ......
########################################
Because of 2 main(String args[]) in directory...
```