/\*\*

This class represents a database. There are many

competing threads wishing to read and write. It is

acceptable to have multiple processes reading at the

same time, but if one thread is writing then no other

process may either read or write.

\*/

public class Database

{

private int readers; // number of active readers

/\*\*

Initializes this database.

\*/

public Database()

{

this.readers = 0;

}

/\*\*

Read from this database.

@param number Number of the reader.

\*/

public void read(int number)

{

synchronized(this)

{

this.readers++;

System.out.println("Reader " + number + " starts reading.");

}

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

synchronized(this)

{

System.out.println("Reader " + number + " stops reading.");

this.readers--;

if (this.readers == 0)

{

this.notifyAll();

}

}

}

/\*\*

Writes to this database.

@param number Number of the writer.

\*/

public synchronized void write(int number)

{

while (this.readers != 0)

{

try

{

this.wait();

}

catch (InterruptedException e) {}

}

System.out.println("Writer " + number + " starts writing.");

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

System.out.println("Writer " + number + " stops writing.");

this.notifyAll();

}

}

Next, we present the reader.

/\*\*

This class represents a reader.

\*/

public class Reader extends Thread

{

private static int readers = 0; // number of readers

private int number;

private Database database;

/\*\*

Creates a Reader for the specified database.

@param database database from which to be read.

\*/

public Reader(Database database)

{

this.database = database;

this.number = Reader.readers++;

}

/\*\*

Reads.

\*/

public void run()

{

while (true)

{

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

this.database.read(this.number);

}

}

}

Next, we present the writer.

/\*\*

This class represents a writer.

\*/

public class Writer extends Thread

{

private static int writers = 0; // number of writers

private int number;

private Database database;

/\*\*

Creates a Writer for the specified database.

@param database database to which to write.

\*/

public Writer(Database database)

{

this.database = database;

this.number = Writer.writers++;

}

/\*\*

Writes.

\*/

public void run()

{

while (true)

{

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

this.database.write(this.number);

}

}

}

Finally, we present the simulator.

/\*\*

This app creates a specified number of readers and

writers and starts them.

\*/

public class Simulator

{

/\*\*

Creates the specified number of readers and writers and starts them.

@param args[0] The number of readers.

@param args[1] The number of writers.

\*/

public static void main(String[] args)

{

if (args.length < 2)

{

System.out.println("Usage: java Simulator <number of readers> <number of writers>");

}

else

{

final int READERS = Integer.parseInt(args[0]);

final int WRITERS = Integer.parseInt(args[1]);

Database database = new Database();

for (int i = 0; i < READERS; i++)

{

new Reader(database).start();

}

for (int i = 0; i < WRITERS; i++)

{

new Writer(database).start();

}

}

}

}