**Actividad UF1-2.**

**Multitarea. Centro de exámenes**

**Ivan Nuñez Rodriguez**

2º DAM

Programación de Procesos y Servicios

**Índice**

1. Código.
2. Resultado.

**Código**

**Clase BufferExamenes**

import java.util.LinkedList;  
import java.util.Queue;  
  
public class BufferExamenes {  
 private Queue<String> colaExamenes;  
  
 public BufferExamenes() {  
 colaExamenes = new LinkedList<>();  
 }  
  
 public synchronized void fabricarNuevoExamen(String codigo) {  
 colaExamenes.add(codigo);  
 System.*out*.println("Producido examen " + codigo);  
 notify(); // Despierta un hilo en espera  
 }  
  
 public synchronized String consumirExamen() {  
 while (colaExamenes.isEmpty()) {  
 try {  
 wait(); // Espera hasta que haya un examen disponible  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 return null;  
 }  
 }  
 return colaExamenes.poll(); // Extrae y devuelve el código del examen  
 }  
}

**Clase Examinado**

import java.util.Random;  
  
public class Examinado implements Runnable {  
 private Thread hilo;  
 private BufferExamenes buffer;  
  
 public Examinado(String alumno, BufferExamenes generador) {  
 this.buffer = generador;  
 this.hilo = new Thread(this, alumno);  
 this.hilo.start();  
 }  
  
 @Override  
 public void run() {  
 String codigoExamen = this.buffer.consumirExamen();  
 if (codigoExamen != null) {  
 Random random = new Random();  
 String[] respuestas = {"A", "B", "C", "D", "-"};  
 for (int i = 1; i <= 10; i++) {  
 String respuesta = respuestas[random.nextInt(respuestas.length)];  
 System.*out*.println(codigoExamen + ";" + hilo.getName() + "; Pregunta " + i + ";" + respuesta);  
 try {  
 Thread.*sleep*(100); // Simula el tiempo de respuesta  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 } else {  
 System.*out*.println("Agotado tiempo de espera y no hay más exámenes");  
 }  
 }  
}

**Clase Principal**

public class Principal {  
 public static void main(String[] args) throws InterruptedException {  
 BufferExamenes generador = new BufferExamenes();  
  
 new ProductorExamenes(generador);  
 new Examinado("Rosa", generador);  
  
 new ProductorExamenes(generador);  
 new Examinado("Miguel", generador);  
  
 new ProductorExamenes(generador);  
 new Examinado("Carlos", generador);  
 }  
}

**Clase ProductorExamenes**

import java.time.LocalDateTime;  
  
public class ProductorExamenes implements Runnable {  
 private BufferExamenes buffer;  
 private static int *numeroExamen* = 0;  
 private Thread hilo;  
  
 public ProductorExamenes(BufferExamenes buffer) {  
 *numeroExamen*++;  
 this.buffer = buffer;  
 this.hilo = new Thread(this, "E" + *numeroExamen*);  
 this.hilo.start();  
 }  
  
 @Override  
 public void run() {  
 int aa = LocalDateTime.*now*().getYear();  
 String codigo = hilo.getName() + "-" + aa;  
 buffer.fabricarNuevoExamen(codigo);  
 }  
}

**Ejecución**

Ejecucion PrincipalCaptura de pantalla de computadora

Descripción generada automáticamente