Bloque 4. Paradigmas de la Programación

Tarea 2: Corral de gallinas con Semáforos

A tener en cuenta:

- La clase Main contiene los 28 pollos que serán los hilos.
- Los pollos realizarán 4 actividades: comer, beber, pasear y dormir, las cuales tendrán una limitación de pollos por actividad que regularemos mediante semáforos de contador.
- Los pollos nunca finalizan el ciclo (las 4 actividades se ejecutarán en un bucle while)
- Tendremos 5 clases:
    - Main: lanza los 28 hilos
    - Pollo: contiene las 4 opciones/actividades que realizarán los pollos. Primero aseguramos que todos empiezan paseando y tras esto, de forma aleatoria, realizarán otras actividades.
    - Comedero/Bebedero/Cama: cada clase tendrá un semáforo contador y realizará la actividad.

Código Clase Main:

```
public class Main {

    public static void main(String[] args) {
        //Variables compartidas, hay que proteger la sección crítica con Semáforos
        Comedero comedero = new Comedero();
        Bebedero bebedero = new Bebedero();
        Cama cama = new Cama();
        // Los 28 hilos representan los pollos
        for (int i = 1; i <= 28; i++) {
            Pollo pollo = new Pollo(comedero, bebedero, cama, i);
            pollo.start();
        }
    }

}
```

Código Clase Pollo:

```
import java.util.logging.Level;
import java.util.logging.Logger;

public class Pollo extends Thread {


    private Comedero comedero;
    private Bebedero bebedero;
    private Cama cama;
    private int id;
    //El pollo tiene un identificador y una actividad aleatoria en cada momento.
    public Pollo(Comedero comedero, Bebedero bebedero, Cama cama, int id) {
        this.comedero = comedero;
        this.bebedero = bebedero;
        this.cama = cama;
        this.id = id;
```

```
    }

    public void run() {
        //Primero, aseguraremos que la actividad inicial de todos los pollos/hilos será la de pasear
        int accion;
        accion = 1;
        while (true) {
            //En un bucle infinito, se realizarán las actividades aleatorias
            switch (accion) {
                case 1:
                    System.out.println("El pollo " + id + " está paseando");
                    int tiempo = (9 + (int) (-5 * Math.random()));
                     {
                        try {
                            sleep(tiempo);
                        } catch (InterruptedException ex) {
                            Logger.getLogger(Pollo.class.getName()).log(Level.SEVERE, null, ex);
                        }
                    }
                    break;
                case 2: {
                    try {
                        comedero.comer(id);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(Pollo.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                break;
                case 3: {
                    try {
                        bebedero.beber(id);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(Pollo.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                break;
                case 4: {
                    try {
                        cama.dormir(id);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(Pollo.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                break;
            }
            //Aquí generaremos un valor aleatorio entre 1 y 4, que decidirá la siguiente acción del pollo
            accion = (4 + (int) (-4 * Math.random()));

        }
    }

}
```

Código Clase Comedero:

```
 import static java.lang.Thread.sleep;
import java.util.concurrent.Semaphore;

public class Comedero {
   //  El semáforo nos ayudará a asegurar la exclusión mutua de la sección crítica
   private Semaphore exclusionMutua = new Semaphore(4);
   //Constructor vacío
   public Comedero() {
   }

   public void comer(int id) throws InterruptedException {

      try {
         exclusionMutua.acquire();
         System.out.println("El pollo: " + id + " está comiendo");
         int tiempo = (6 + (int) (-2 * Math.random()));
         sleep(tiempo);

      } finally {
         exclusionMutua.release(4);
      }
   }
}
```

Código Clase Bebedero:

```
        import static java.lang.Thread.sleep;
import java.util.concurrent.Semaphore;

public class Bebedero {

   private Semaphore exclusionMutua = new Semaphore(8);

   public Bebedero() {
   }

   public void beber(int id) throws InterruptedException {

      try {
         exclusionMutua.acquire();
         System.out.println("El pollo: " + id + " está bebiendo");
         int tiempo = (3 + (int) (-1 * Math.random()));
         sleep(tiempo);

      } finally {
         exclusionMutua.release(8);
      }
   }
}
```

Código Clase Cama:

```
        import static java.lang.Thread.sleep;
import java.util.concurrent.Semaphore;

public class Cama {

   private Semaphore exclusionMutua = new Semaphore(10);

   public Cama() {
   }

   public void dormir(int id) throws InterruptedException {

      try {
         exclusionMutua.acquire();
         System.out.println("El pollo: " + id + " está durmiendo");
         int tiempo = (19 + (int) (-15 * Math.random()));
         sleep(tiempo);

      } finally {
         exclusionMutua.release(10);
      }
   }

}
```

Resultado:

Observamos que los pollos empiezan paseando y que los que han acabado, como el caso del pollo 2, pasan a realizar otras actividades

```
run:
El pollo: 1 está paseando
El pollo: 2 está paseando
El pollo: 3 está paseando
El pollo: 4 está paseando
El pollo: 5 está paseando
El pollo: 7 está paseando
El pollo: 6 está paseando
El pollo: 8 está paseando
El pollo: 9 está paseando
El pollo: 2 está bebiendo
El pollo: 10 está paseando
El pollo: 11 está paseando
El pollo: 12 está paseando
El pollo: 13 está paseando
El pollo: 14 está paseando
El pollo: 3 está durmiendo
El pollo: 5 está durmiendo
El pollo: 15 está paseando
El pollo: 4 está paseando
```