ISMAEL FERNÁNDEZ HERRERUELA

EJERCICIO 1

```
public class Ejer1 extends Thread{
        protected int n;
        public Ejer1(int n) {this.n = n;}

        private void randomSleep() {
            try {Thread.sleep (Math.round (Math.random() * 5)); }
         catch(InterruptedException ie){ie.printStackTrace();}
        }

        public void run() {
            String name = Thread.currentThread().getName();
            for (int i=0; i<n; i++) {
                System.out.println(name +" doing iteration "+i);
                randomSleep();
            }
            System.out.println("End of thread "+ name);
        }

        public static void main(String[] argv) {
            System.out.println("---  Begin of execution ---- ");
            for (int i=0; i<10; i++)
                 new Ejer1(i).start();
            System.out.println("--- End of execution ---- ");
        }
 }
```

1.A) Se crean 10 hilos y se ejecutan los 10 hilos pero con orden diferente cada vez que lo ejecutemos porque el orden depende del planificador. Se asigna el nombre de Thread-X, siendo X el numero de cada hilo.

1.B) Por pantalla se mostrara un mensaje de la ejecución de cada iteración de cada hilo. Siempre aparecerá como primer mensaje "--- Begin of execution --- ", y al crearse todos los hilos se mostrara "--- End of execution ---", pero el resto de mensajes de cada hilo variara en cada ejecución ya que se ejecutaran en diferente orden.

2.A) Habría que añadir al método run el siguiente código:

```
String hilo = "Hilo" + n;
Thread.currentThread().setName(hilo);
System.out .println(hilo +" doing iteration "+i);
```

2.B)

```java
public class Ejer1 extends Thread{
    protected int n;
    public Ejer1(int n) {this.n = n;}

    private void randomSleep() {
        try {Thread.sleep (Math.round (Math.random() * 5)); }
        catch(InterruptedException ie) {ie.printStackTrace();}
    }

    public void run() {
        //String name = Thread.currentThread().getName();
        String hilo = "Hilo" + n;
        Thread.currentThread().setName(hilo);
        System.out.println(hilo + " Begin");

        for (int i=0; i<n; i++) {
            System.out .println(hilo +" doing iteration "+i);
            randomSleep();
        }
        System.out.println(hilo + " End");
    }

    Run | Debug
    public static void main(String[] argv) {
        System.out.println(x: "--- Begin of execution ---- ");
        for (int i=0; i<10; i++) {
            Ejer1 thread = new Ejer1(i);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(x: "--- End of execution ---- ");
    }
}
```

Modificaríamos el código de la siguiente manera utilizando locks.

EJERCICIO 2

```
public class Ejer2 extends Thread {
    private int label;
    private String name;
    public Ejer2(int n){
        label = n;
        name = "MidThread";
    }

    public void run() {
       label--;
       Ejer2 h = new Ejer2(label);
       h.setName(name + label);
       if (label >= 1)
          h.start();
       System.out.println("End of thread. I am:" + Thread.currentThread().getName());
    }
    public static void main(String[] argv) {
      new Thread( () -> {
          System.out.println("I am " + Thread.currentThread().getName());
          new Ejer2(4).start();
          },"BigThread").start();

      new Thread( () -> {
          System.out.println("I am " + Thread.currentThread().getName());
          new Ejer2(2).start();
          },"SmallThread");
    }
}
```

1.A) En este código hay 3 hilos, el main(Thread-0), “BigThread” y “SmallThread”, pero solo se utilizan el principal y “BigThread” ya que “SmallThread” no tiene .start(). El nombre de cada ejecución de los hilos diferentes al main es “MidThreadX” siendo X = label.

1.B) Por pantalla se mostrara “BigThread” primero de todo, y después se iran mostrando los nombres de cada hilo hasta que label sea igual o menor que 1. Siempre saldrán en orden descendente y empezando por “BigThread”.

2)

```
public class Ejer2 extends Thread {
private int label;
public Ejer2(int n) {
   label = n;
}
public void run() {
   System.out.println("I am " + Thread.currentThread().getName());
   if (label > 1) {
      Ejer2 h = new Ejer2(label - 1);
      h.start();
      try {
         h.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
   System.out.println("End of thread. I am: " + Thread.currentThread().getName());
}
Run | Debug
public static void main(String[] args) {
   new Ejer2(n:4).start();
}
```

EJERCICIO 3

1.) Al estar sincronizados con la palabra “synchronized” nunca se ejecutaran de forma concurrente ya que hasta que uno no se completa no se pasa al otro. Por lo que nunca se hará ninguna ejecución concurrente. Solo se utiliza un lock.

2.) Los métodos test1 y test2 estan sincronizados usando dos objetos diferentes(o1,o2) entonces solo 1 puede estar ejecutando la sección critica en test1 o test2. En este caso se emplean dos locks, uno para cada objeto. No es equivalente a MyExample ya que en este cada método tiene su monitor y por eso se utilizan 2 locks, además, es posible que 2 hilos ejecuten concurrentemente test1 y test2 siempre y cuando estén accediendo a objetos diferentes.

EJERCICIO 4

1.) Enter: se utiliza para que un hilo indique que quiere evolucionar a su pokemon en la región de galar. El parámetro isEevee indica si el pokemon es un Eevee o no. Si hay mas hilos esperando para que un pokemon del tipo opuesto evolucione el hilo actual espera con la condición “notalone”. Cuando es el único hilo esperando para evolucionar, incrementa el contador de su tipo y comprueba si se ha alcanzado el numero de pokemon necesarios para la evolución. Si esto ultimo se cumple, se notificara a los hilos que están esperando “grookeevolve” para que despierten.

Exit: se utiliz apara que un hilo indique que el pokemon ha terminado de evolucionar y esta abandonando la región galar. Decrementa el contador del pokemon correspondiente y comprueba si hay hilos esperando con la condición “notalone” y si es así, lo notifica para que despierte.

Evolve: se utiliza para que un hilo espere hasta que hayan alcanzado el numero necesario de pokemon para la evolución. Si el numero no se ha alcanzado todavía, el hilo espera en la condición “grookevolve” hasta que sea notificado por otro hilo.

2.) En el método wait en la condición “notalone”, el comportamiento será previsto en la variante de Hoare, pero puede llegar a producir problemas en Lampson-Redell. En la variante de Hoare cuando un hilo llama a este método, se libera el cerrojo permitiendo así que otros hilos entren en la sección critica. Pero por otro lado, en Lampson-Redell, el cerrojo permanece bloqueado por lo que otros hilos no pueden entrar en la sección critica hasta que el hilo que invocó al método wait sea notificado y libere el cerrojo. Si varios hilos llaman a este método pueden producirse deadlocks. Ejemplo de traza:

- Hilo A llama a enter(true) y espera en notalone
- Hilo B llama a enter(false) y espera en notalone
- Hilo C llama a enter(false) y espera en notalone

Pikachus=1

Eevees=1

Waiting=1

- Hilo D llama a exit(true) y notifica notalone
- Hilo A despierta y entra en la sección critica
- Hilo A eevees++ y comprueba si hay 5 eevees o pikachus
- Hilo A despierta a grookevolve y sale de la sección critica

Pikachus=1

Eevees=2

Waiting=1

-Hilo B intenta entrar en la sección critica pero no puede porque el cerrojo esta bloqueado

- Hilo C intenta entrar en la sección critica pero no puede porque el cerrojo esta bloqueado

De esta manera el monitor se quedaría permanentemente bloqueado en la condición notalone.

3.)

```
Monitor Galar {
int pikachus, eevees, waiting, evolution;
condition grookevolve, notalone_evee, notalone_pika;

public Galar() {
    pikachus = eevees = waiting = 0;
    evolution = 5;
}

entry void enter_Eevee() {
    waiting++;
    while (pikachus > 0) {
        notalone_evee.wait();
    }
    waiting--;
    eevees++;
    if (eevees >= evolution) {
        grookevolve.notify();
    }
}

entry void exit_Eevee() {
    eevees--;
    if (waiting > 0 && pikachus == 0) {
        notalone_evee.notify();
    }
}

entry void enter_Pikachu() {
    waiting++;
    while (eevees > 0) {
        notalone_pika.wait();
    }
    waiting--;
    pikachus++;
    if (pikachus >= evolution) {
        grookevolve.notify();
    }
}

entry void exit_Pikachu() {
    pikachus--;
    if (waiting > 0 && eevees == 0) {
        notalone_pika.notify();
    }
}

entry void evolve_grooky() {
    if (pikachus < evolution && eevees < evolution) {
        grookevolve.wait();
    }
```

EJERCICIO 5

1.) AddBlanca: se utiliza para agregar una unidad de articulo blanco al inventario. Si ya se cumple la capacidad máxima, el hilo actual se pone en espera utilizando la condición "hayHuecoBlancas". Una vez que se agrega un articulo, se notifica a cualquier hilo que este esperando la condición "hacerPedido".

AddAzul: igual que AddBlanca pero para azul.

SolicitarPedido: se utiliza para solicitar un pedido de cierta cantidad de artículos blancos y azules. Si ya hay un pedido en curso el hilo actual se pone en espera utilizando la condición "listoPedido". Si no hay suficientes artículos el hilo se pone en espera utilizando "hayHuecoBlancas" o "hayHuecoAzules". Cuando hayan suficientes artículos para cumplir con la solicitud, se actualiza el inventario y se notifica a los hilos en espera. Finalmente se establece "PedidoEnCurso" en falso para indicar que se ha completado el pedido y se notifica a cualquier hilo que este esperando con la condición "listoPedido".

2.) Pongamos que los hilos se ejecutan en este orden:

-Hilo A (encargado de proveer perlas blancas) invoca addblanca varias veces.

-Hilo B (encargado de perlas azules) invoca addazul varias veces

-Hilo C (encargado de pedidos) invoca solicitarpedido(10,10) varias veces

-Hilo D(encargado de pedidos) invoca solicitarpedido(5,5) varias veces

-Hilo E (encargado de pedidos) invoca solicitarpedido(20,20) varias veces

Para Hoare:

-Hilo A ejecuta

-Hilo B ejecuta

-Hilo C espera "listopedido"

-Hilo D espera "listopedido"

-Hilo E espera "listopedido"

Los hilos C,D y E se quedan en espera de manera indefinida ya que el pedido no se completa. Esto se debe a que los hilos no liberan de la espera en la condición "listopedido", por lo que seria necesario agregar una variable de condición adicional en solicitarpedido que permita notificar a los hilos en espera.

Para Lampson-Redell también se puede presentar una situación de interbloqueo. Supongamos que el Hilo A obtiene el cerrojo y entra al metodo addblanca. En ese momento el

Hilo B también intenta obtener el cerrojo para entrar en addazul, pero al no tenerlo se bloquea. Luego Hilo C, el cual es encargado de los pedidos entra en solicitarpedido, lo que activa hacerpedido.notify para que se despierten A y B que están esperando a las condiciones hayHuecoBlancas y hayHuecoAzules. Sin embargo como B no puede entrar a addazul, A se desbloquea y continua con addblanca. Como la condición nblancas==nmaxblancas no se cumple, no se espera y despierta al Hilo C. Este hilo intentara disminuir la cantidad de perlas blancas disponibles en una cantidad que no esta disponible, y el Hilo B sigue bloqueado en la condición hayhuecoazules por lo que se producirá un interbloqueo que no se podrá resolver.