Hacer un programa compuesto por cuatro clases:
• Clases hilos HinchaGlobos (HG) (encargada de hinchar los globos)
• Clase hilo PinchaGlobos (PG) (encargada de pinchar los globos)
• Clase Globos (almacén de globos)
• Clase Inicio (contiene el código main e instancia al resto)

En relación a la clase Globos (tendremos 1 instancia de esta clase)
• En el almacén habrá un máximo de 10 globos .
• Se irán entregando de uno en uno numerándolos desde 1 a 10.
• Una vez entregados se podrán hinchar hasta que estallen o los pinchen.
• Sólo podrá haber 3 globos hinchándose a la vez.
• Los globos tendrán volumen de 1 a 5. Si se llena más de 5 estallan.
• El globo se entrega con un volumen inicial de 1.
• Los globos pueden ser pinchados mientras se están hinchando.
• Se escribirá un mensaje con el número de globo cada vez que:
o Se entregue un globo (indicaremos el nombre del hilo ejem: GLOBO 5 ENTREGADO A HG3) o Se hinche un globo e indicaremos el nuevo volumen (ejem: GLOBO 5 VOLUMEN 5)
o Se estalle un globo por inflar de más (ejem : GLOBO 5 ESTALLA)
o Un PG pinche un globo (indicaremos el nombre del hilo ejem : GLOBO 5 PINCHADO POR PG3)

En relación a la clase HinchaGlobos (tendremos 5 instancias de esta clase)
• (0,5) Obtendrá un globo de la clase globos cada vez y en orden salvo que no queden más.
• (2) Si ya hubiera tres hinchándose se esperará hasta que uno se pinche o estalle .
• Los tendremos que numerar y nombrar con HG seguido de

su número. (ejem: HG3)
• (1) Intentará hincharlo hasta estallarlo salvo que sea pinchando.
• (1) Cada segundo aumentará el volumen del globo desde 1 hasta estallarlo.
• (1) Si estalla o se pincha volverá a por otro globo hasta que no queden más.

En relación a la clase PinchaGlobos (tendremos 5 instancias de esta clase)
• (1) Intentará pinchar uno de los globos que se está hinchando cada un tiempo aleatorio de 1 a 10 segundos .
• (1) Si no hay globos que pinchar se quedará en espera.
• (0,5) Dejará de pinchar cuando no queden globos que pinchar.
• Sus instancias las tendremos que numerar y nombrar con PG seguida de su número (ejem : PG 2)

Los distintos apartados serán comprobados por la impresión de los mensajes detallados en la clase Globos .

Se deberá tener en cuenta que aquellos dormidos deberán ser despertados convenientemente:
(1) Caso de HG dormidos
(1) Caso de PG dormidos

```
public class Inicio {

    public static void main(String[] args) {

        Globos g = new Globos();    //Instanciamos Globos

        for (int i=1;i<=5;i++) new HinchaGlobo(g,i);
//Bucle que crea 5 hilos de HinchaGlobo, le pasamos los
globos e i para saber el número del hilo
```

```
        for (int i=1;i<=5;i++) new PinchaGlobo(g,i);
//Bucle que crea 5 hilos de PinchaGlobo, le pasamos los
globos e i para saber el número del hilo
    }
}
public class Globos {

    private int maxHinchando=3;     // Número máximo de
globos hinchando a la vez
    private int maxGlobos=10;        // Número máximo de
globos en el almacén
    private int maxVolumen=5;          // Volumen máximo del
globo
    private int nGlobo=1;             // Número del globo
para dar, inicialmente 1
    private int hinchandoAhora=0;     // Cuántos se están
hinchando ahora mismo
    private int globos[];
    // Si vale 0 el globo no está dado
    // Si vale 1 a maxVolumen está dado e indica su volumen
    // Si vale maxVolumen+1 está roto
    // Si vale maxVolumen+2 está pinchado

    public Globos() {    // Constructor de Globos
        globos=new int[maxGlobos];
        for (int i=0;i<maxGlobos;i++) globos[i]=0; //
Rellenamos el almacén a 0, (ninguno dado).
    }

    public synchronized int dameGlobo() { // Método que
devuelve el siguiente globo del almacen, si no quedan
devuelve -1

        while (hinchandoAhora==maxHinchando &&
nGlobo!=maxGlobos+1)  { // Me espero si ya hay maxHinchando
y quedan globos por dar
            try {wait();} catch (Exception e) {}
        }
```

```
        if (nGlobo==maxGlobos+1) return -1; // Retorno un -1
si no quedan globos por dar
        globos[nGlobo-1]=1;                 // Cambio el 0
del almacen de globos por 1 (entregado a HinchaGlobos)
        System.out.println("GLOBO "+nGlobo+" ENTREGADO A
"+Thread.currentThread().getName());    //Informo por
consola a que hilo se le da el globo
        hinchandoAhora++;     //Sumo 1 al hinchaAhora
        notifyAll();        //Notifico a todos que hay un
cambio
        return nGlobo++;    //Retorno el globo
    }

    public synchronized boolean pincho() { // Método que
pincha un globo, si no quedan devuelve true

        while (hinchandoAhora==0 && nGlobo!=maxGlobos+1){
// Me espero si no hay hinchando y quedan por pinchar
            try {wait();} catch (Exception e) {}
        }

        if (nGlobo==maxGlobos+1) return true;    // Me
aseguro de salir porque ha cambiado hinchando

        for (int i=0;i<maxGlobos;i++)    // Busco un globo
para pinchar dentro del almacen
            if (globos[i]>0 && globos[i]<=maxVolumen) {
                System.out.println("GLOBO"+(i+1)+" LO PINCHA
"+Thread.currentThread().getName());
                globos[i]=maxVolumen+2;
                hinchandoAhora--;
                notifyAll();
                break;
            }

        return hinchandoAhora!=0;
    }
```

```
    public synchronized boolean hincho(int num) {

        if (globos[num-1]<=maxVolumen) globos[num-1]++;
// Puede que ya esté estallado, se comprueba
        else return true;                                   //
si estuviera estallado, pinchado o estallase dev true

        if (globos[num-1]==maxVolumen+1){    // Si lo he
estallado lo notifico
            hinchandoAhora--;
            System.out.println("GLOBO "+num+ " ESTALLA");
            notifyAll();
            return true;
        }
        else {
            System.out.println("GLOBO "+num+" VOLUMEN
"+globos[num-1]);
            return false;
        }
    }
}
public class HinchaGlobo extends Thread {
    //Los globos estallan cuando pasan de 10
    private Globos g;
    private int numero;

    public HinchaGlobo(Globos pg,int pnumero) {
        g=pg;
        numero=pnumero;
        setName("HG"+numero);
        start();
    }

    @Override
    public void run(){
        int manejado;
        boolean estalla; // cierto si se pincha o estalla
```

```
        while (true) {
            if ((manejado=g.dameGlobo())==-1) break ; // me
da un globo o -1 si no hay mas
            do{
                try {Thread.sleep(1000);} catch (Exception
e) {}
                estalla=g.hincho(manejado);
            } while (!estalla);

        } // while(true)
    } // public run
} // public class
public class PinchaGlobo extends Thread{
    private Globos g;
    private int numero;
    public PinchaGlobo(Globos pg,int pnumero){
        g=pg;
        numero=pnumero;
        setName("PG"+numero);
        start();
    }

    @Override
    public void run(){
        int num;
        boolean nohaymas;
        do{
            try {Thread.sleep((int)(Math.random()*5000));}
catch (Exception e) {}
            // Si no quedan globos tengo que dejar de
pinchar
            nohaymas=g.pincho();
        } while (!nohaymas); // while true
    } // public run
} // public class
```

Ejercicio 1: Implementación de Threads

Enunciado:

Crea un programa en Java que simule un banco de pruebas de vehículos. En este escenario, tienes tres vehículos que necesitan ser sometidos a pruebas en tres pistas diferentes. Cada pista solo puede manejar un vehículo a la vez. El programa debe utilizar threads para simular que los vehículos acceden a las pistas, permanecen en ellas por un tiempo aleatorio, y luego se liberan. Cada thread debe imprimir un mensaje indicando cuándo el vehículo entra en la pista, cuánto tiempo pasa en ella, y cuándo la deja.

Código:

```
import java.util.Random;

class Vehiculo extends Thread {
    private String nombre;
    private int pista;

    public Vehiculo(String nombre, int pista) {
        this.nombre = nombre;
        this.pista = pista;
    }

    @Override
    public void run() {
        try {
            System.out.println(nombre + " entrando en la pista " + pista);
            Random random = new Random();
            int tiempoPrueba = random.nextInt(5000) + 2000; // Tiempo de prueba entre 2 y 5 segundos
            Thread.sleep(tiempoPrueba);
            System.out.println(nombre + " completó la prueba en la pista " + pista + " en " + (tiempoPrueba / 1000) + "
segundos.");
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

```
}

public class BancoPruebas {
    public static void main(String[] args) {
        Vehiculo vehiculo1 = new Vehiculo("Vehiculo 1", 1);
        Vehiculo vehiculo2 = new Vehiculo("Vehiculo 2", 2);
        Vehiculo vehiculo3 = new Vehiculo("Vehiculo 3", 3);

        vehiculo1.start();
        vehiculo2.start();
        vehiculo3.start();
    }
}
```

Ejercicio 2: Uso de Semáforos

Enunciado:

Crea un programa en Java que simule un laboratorio de computación con cuatro computadoras. Hay seis estudi necesitan usar las computadoras, pero solo cuatro pueden hacerlo al mismo tiempo. Utiliza semáforos para con acceso de los estudiantes a las computadoras. Cada estudiante debe imprimir un mensaje cuando comienza a u computadora y otro cuando termina de usarla, indicando qué computadora estaba usando.

Código:

```
import java.util.concurrent.Semaphore;

class Estudiante extends Thread {
    private String nombre;
    private Semaphore semaforo;

    public Estudiante(String nombre, Semaphore semaforo) {
        this.nombre = nombre;
        this.semaforo = semaforo;
    }

    @Override
    public void run() {
        try {
            semaforo.acquire();
            int computadora = semaforo.availablePermits() + 1; // Para identificar la computadora utilizada
            System.out.println(nombre + " comenzó a usar la computadora " + computadora);
            Thread.sleep((long) (Math.random() * 3000) + 2000); // Simula el tiempo de uso entre 2 y 5 segundos
            System.out.println(nombre + " terminó de usar la computadora " + computadora);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        } finally {
            semaforo.release();
        }
    }
}

public class LaboratorioComputacion {
    public static void main(String[] args) {
```

```
}

public class LaboratorioComputacion {
    public static void main(String[] args) {
        Semaphore semaforo = new Semaphore(4); // 4 computadoras disponibles

        Estudiante estudiante1 = new Estudiante("Estudiante 1", semaforo);
        Estudiante estudiante2 = new Estudiante("Estudiante 2", semaforo);
        Estudiante estudiante3 = new Estudiante("Estudiante 3", semaforo);
        Estudiante estudiante4 = new Estudiante("Estudiante 4", semaforo);
        Estudiante estudiante5 = new Estudiante("Estudiante 5", semaforo);
        Estudiante estudiante6 = new Estudiante("Estudiante 6", semaforo);

        estudiante1.start();
        estudiante2.start();
        estudiante3.start();
        estudiante4.start();
        estudiante5.start();
        estudiante6.start();
    }
}
```

Adaptación propuesta del examen:

- Escenario: “N usuarios” (hilos) que quieren acceder a una impresora, pero además cada usuario tiene un trabajo que genera “X páginas” que deben

imprimirse en “la impresora” y simultáneamente registrar estadísticas en un objeto compartido (contador total de páginas impresas).

- Requisitos:
  - Cada hilo-usuario debe pedir acceso a la impresora (recurso único) mediante semáforo.
  - Una vez que accede, imprime sus páginas (simulación con `sleep()`) y actualiza un contador global compartido de “páginas totales impresas” (requiere sincronización para evitar condiciones de carrera).
  - Después libera la impresora.
  - El programa debe funcionar para cualquier N ≥ 2.
  - Asegurar que no haya interbloqueo (aunque al haber una sola impresora no es clásico de deadlock, se puede pedir evitar “inanición” de algún hilo: que ningún hilo quede eternamente sin imprimir).
  - Documentar y explicar el mecanismo de sincronización: semáforo para la impresora, `synchronized` o `Lock` para el contador compartido.
- Parte extra (opcional de mayor dificultad): modifica el escenario para que haya **M impresoras** (M < N) y cada hilo decida aleatoriamente cuál impresora usar, de modo que haya varios recursos; esto se acerca más al problema de múltiples tenedores del “filósofos”.

```
import java.util.concurrent.Semaphore;


class Impresora {
  private Semaphore semaforo = new Semaphore(1); // 1 impresora disponible
  private int totalPaginas = 0;


  // Imprimir un trabajo
  public void imprimir(String nombreUsuario, int paginas) {
    try {
      semaforo.acquire(); // Pedir acceso a la impresora
      System.out.println(nombreUsuario + " está imprimiendo " + paginas + "
páginas.");
      Thread.sleep(paginas * 100); // Simulación de impresión


      // Actualización segura del contador compartido
```

```
            synchronized (this) {
                totalPaginas += paginas;
                System.out.println("Total de páginas impresas: " + totalPaginas);
            }

            System.out.println(nombreUsuario + " terminó de imprimir.");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            semaforo.release(); // Liberar impresora
        }
    }
}

// Hilo que representa a un usuario
class Usuario extends Thread {
    private Impresora impresora;
    private int paginas;
    private String nombre;

    public Usuario(String nombre, Impresora impresora, int paginas) {
        this.nombre = nombre;
        this.impresora = impresora;
        this.paginas = paginas;
    }

    @Override
    public void run() {
```

```
        impresora.imprimir(nombre, paginas);
    }
}

// Clase principal
public class ExamenImpresora {
    public static void main(String[] args) {
        int N = 5; // Número de usuarios
        Impresora impresora = new Impresora();

        // Crear hilos de usuarios
        Thread[] usuarios = new Thread[N];
        for (int i = 0; i < N; i++) {
            usuarios[i] = new Usuario("Usuario" + (i+1), impresora, (i+1)*2);
            usuarios[i].start();
        }

        // Esperar a que terminen todos
        for (int i = 0; i < N; i++) {
            try {
                usuarios[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Impresión completa. Total de páginas impresas: " +
impresora.totalPaginas);
```

```
    }
}
```