PROCDUCTOR CONSUMIDOR COLA CON SLEEP

```
public static void main(String[] args) {
   Cola cola = new Cola();
   Prod p = new Prod(cola, 1);
   Cons c = new Cons(cola, 1);
   p.start();
   c.start();
}
public class Prod extends Thread {
   private Cola cola; private int n;

   public Prod (Cola c, int n){ cola = c; this.n =n; }

   public void run() {
      for (int i = 0; i>=0; i++) {
         cola.put(i); //pone el número
         System.out.println(i+"=>Productor : " + n + ", produce: " + i);
         try { sleep(200); } catch (InterruptedException e) { }
         // duerme el hilo durante un tiempo aleatorio
      }
   }
}
public class Cons extends Thread {
   private Cola cola;  private int n;
   public Cons (Cola c, int n){ cola = c; this.n =n;
   }
   public void run() {
      int valor = 0;
      for (int i = 0; i>=0; i++) {
         try {
            sleep(100);
            valor = cola.get(); //recoge el numero
            System.out.println(i+"=>Consumidor : " + n + ", consume: " + valor+"----------");
            sleep(100);
         } catch (InterruptedException e) { throw new RuntimeException(e); }
      }
   }
}
public class Cola {
   private int numero;
   private boolean disponible = false; //inicialmente cola vacia
   public int get() {
      if (disponible) { //hay numero en la cola?
         disponible = false; //se pone cola vacia
         return numero; //se devuelve
      }
      return (-1); //no hay numero disponible, cola vacia
   }
   public void put (int valor) {
      numero = valor; //coloca valor en la cola
      disponible = true; //disponible para consumir, cola llena
   }
}
```

PRODUCTOR CONSUMIDOR COLA CON MONITORES

```
public static void main(String[] args) {
   Cola cola = new Cola();
   Prod p = new Prod(cola, 1);
   Cons c = new Cons(cola, 1);
```

```
    p.start();
    c.start();
}
public class Cons extends Thread {
    private Cola cola;
    private int n;
    public Cons (Cola c, int n){
        cola = c;
        this.n =n;
    }
    public void run() {
        int valor = 0;
        for (int i = 0; i<5000; i++) {
            valor = cola.get(); //recoge el numero
            System.out.println(i+"=>Consumidor : " + n + ", consume: "+ valor);
        }
    }
}
public class Prod extends Thread {
        private Cola cola;
        private int n;

        public Prod (Cola c, int n){
            cola = c;
            this.n =n;
        }

        public void run() {
            for (int i = 0; i<5000; i++) {
                cola.put(i); //pone el número
                System.out.println(i+"=>Productor : " + n + ", produce: " + i + "----------------------");
                try {
                    sleep(1000);
                } catch (InterruptedException e){            }
            }
        }
}
public class Cola {
    private int num;
    private boolean hay = false; //inicialmente cola vacia

    public synchronized int get() {
            while (!hay) { //hay numero en la cola?
                try{
                    wait(); //espera
                }catch (Exception e){}
            }
            hay = false; //se pone cola vacia
            notify(); //notifica a productor
            return num; //se devuelve

        //return (-1); //no hay numero disponible, cola vacia
    }

    public synchronized void put (int valor) {

            while (hay) { //hay numero en la cola?
                try{
```

```
            wait(); //espera
          }catch (Exception e){}
        }
        num = valor; //coloca valor en la cola
        hay = true; //disponible para consumir, cola llena
        notify(); //notifica a consumidor
    }
}
PRODUCTOR CONSUMIDOR COLA CON SEMAFOROS
public static void main(String[] args) {
    Semaphore prod = new Semaphore(1);
    Semaphore cons = new Semaphore(0);
    Semaphore mutex = new Semaphore(1);

    Cola cola = new Cola();
    Prod p = new Prod(cola, 1, prod, cons, mutex);
    Cons c = new Cons(cola, 1, prod, cons, mutex);
    p.start();
    c.start();
}
public class Prod extends Thread {
      private Cola cola;
      private int n;
      Semaphore prod;
      Semaphore cons;
      Semaphore mutex;

      public Prod (Cola c, int n, Semaphore prod, Semaphore cons, Semaphore mutex){
        cola = c;
        this.n =n;
        this.prod = prod;
        this.cons = cons;
        this.mutex = mutex;
      }

      public void run() {
        for (int i = 0; i < 20; i++) {
          try {
            prod.acquire(); // coge el semaforo de prod para producir y espera a que el consumidor lo libere
            mutex.acquire(); // coge el semaforo de mutex para acceder a la cola y espera a que el consumidor lo libere
            // cuando el consumidor libera el semaforo de mutex, el productor puede acceder a la cola
            cola.put(n); //pone el numero
            System.out.println("Productor : " + n + ", produce: " + n + "-----------------------------------");
            mutex.release(); // libera el semaforo de mutex para que el consumidor pueda acceder a la cola
            cons.release(); // libera el semaforo de cons para que el consumidor pueda consumir
          } catch (InterruptedException e) {
            throw new RuntimeException(e);
          }
        }
      }
}
public class Cons extends Thread {
    private Cola cola;
    private int n;
    Semaphore prod;
    Semaphore cons;
    Semaphore mutex;
```

```
  public Cons (Cola c, int n, Semaphore prod, Semaphore cons, Semaphore mutex){
    cola = c;
    this.n =n;
    this.prod = prod;
    this.cons = cons;
    this.mutex = mutex;
  }
  public void run() {
    for (int i = 0; i < 20; i++) {
      try {
        cons.acquire(); // coge el semaforo de cons para consumir y espera a que el productor lo libere
        mutex.acquire(); // coge el semaforo de mutex para acceder a la cola y espera a que el productor lo libere
        // cuando el productor libera el semaforo de mutex, el consumidor puede acceder a la cola
        n = cola.get(); //obtiene el numero
        System.out.println("Consumidor : " + n + ", consume: "+ n);
        mutex.release(); // libera el semaforo de mutex para que el productor pueda acceder a la cola
        prod.release(); // libera el semaforo de prod para que el productor pueda producir
      } catch (InterruptedException e) {
        throw new RuntimeException(e);
      }
    }
  }
}
public class Cola {
  private int num;
  private boolean hay = false; //inicialmente cola vacia

  public int get() {
    return num;
  }

  public void put (int valor) {
    num = valor;
  }

}
```

PING PONG SINCRONIZADO Y/O CON SEMAFOROS

```
public static void main(String[] args) {
  Cola cola = new Cola();
  Prod p = new Prod(cola);
  Cons c = new Cons(cola);
  p.start();
  c.start();

  // asi seria con semaforos
  /*
  Semaphore prod = new Semaphore(1);
  Semaphore cons = new Semaphore(0);
  Semaphore mutex = new Semaphore(1);
  Cola cola = new Cola();
  Prod p = new Prod(cola, prod, cons, mutex);
  Cons c = new Cons(cola, prod, cons, mutex);
  p.start();
  c.start();
  */
}
public class Prod extends Thread {
    private Cola cola;
```

```
    public Prod (Cola c){
      cola = c;
    }

    public void run() {
      for (int i = 0; i < 20; i++) {
        if(i%2==0){
          cola.put("PING");
        }else{
          cola.put("\tPONG");
        }
        try {
          sleep(500);
        } catch (InterruptedException e) {
          throw new RuntimeException(e);
        }
      }
    }

  // asi seria con semaforos
    /*
    Semaphore prod;
    Semaphore cons;
    Semaphore mutex;

    public Prod (Cola c, Semaphore prod, Semaphore cons, Semaphore mutex){
      cola = c;
      this.prod = prod;
      this.cons = cons;
      this.mutex = mutex;
    }
    public void run() {
      for (int i = 0; i < 20; i++) {
        try {
          prod.acquire();
          mutex.acquire();
          if(i%2==0){
            msg = "PING";
          }else{
            msg = "\tPONG";
          }
          cola.put(msg); //pone el numero
          mutex.release();
          cons.release();
          sleep(500);
        } catch (InterruptedException e) {
          throw new RuntimeException(e);
        }
      }
    }
    */
}
public class Cons extends Thread {
  private Cola cola;

  public Cons (Cola c){
    cola = c;
```

```
    }
    public void run() {
        for (int i = 0; i < 20; i++) {
            System.out.println(cola.get());
            try {
                sleep(500);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
    // asi seria con semaforos
    /*
    Semaphore prod;
    Semaphore cons;
    Semaphore mutex;

    public Cons (Cola c, Semaphore prod, Semaphore cons, Semaphore mutex){
        cola = c;
        this.prod = prod;
        this.cons = cons;
        this.mutex = mutex;
    }
    public void run() {
        for (int i = 0; i < 20; i++) {
            try {
                cons.acquire();
                mutex.acquire();
                msg = cola.get(); //obtiene el numero
                System.out.println(msg);
                mutex.release();
                prod.release();
                sleep(500);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
    */
}
public class Cola {
    private String msg;
    private boolean hay = false;

    public synchronized String get() {
        while(!hay){
            try{   wait();
            }catch (Exception e){} }
        hay = false;
        return msg;   }

    public synchronized void put (String valor) {
        while(hay){
            try{   wait();
            }catch (Exception e){}      }
        msg = valor;
        hay = true;
        notify();
```

```
    }
    // asi seria con semaforos
    /*
    public String get() {  return msg;    }
    public void put (String valor) {
      msg = valor;
    }
    */}
CUENTA BANCO SINCRONIZADO
public static void main(String[] args) {
    Cuenta cuenta = new Cuenta(0, 1000);
    Cliente pepe = new Cliente("Pepe", cuenta);
    Cliente juan = new Cliente("Juan", cuenta);
    Cliente maria = new Cliente("Maria", cuenta);
    Cliente ana = new Cliente("Ana", cuenta);

    pepe.start();
    juan.start();
    maria.start();
    ana.start();
}
public class Cuenta {
    private int saldo;
    private int maximo;
    private boolean fin = false;

    public Cuenta(int saldo, int maximo) {  this.saldo = saldo;
      this.maximo = maximo;   }
    public int getSaldo() {  return saldo;  }
    public int getMaximo() { return maximo;    }
    public boolean getFin() {  return fin;   }
    public void setFin(boolean fin) {   this.fin = fin;  }
    public synchronized void ingreso(int cantidad, String n) {
      System.out.println("\n"+n+"--intento de ingrerso de " + cantidad + "€");
      if (saldo + cantidad < maximo) {    saldo += cantidad;
        System.out.println( "Ingreso de " + cantidad + "€. Saldo actual: " + saldo + "€");
      }else{    System.out.println("Ingreso máximo superado");
        fin = true;
      }
    }

    public synchronized void reintegro(int cantidad, String n) {
      System.out.println("\n"+n+"--intento de retirada de " + cantidad + "€");
      if(cantidad < saldo) {  saldo -= cantidad;
        System.out.println( "Retirada de " + cantidad + "€. Saldo actual: " + saldo + "€");
      }else{   System.out.println("Saldo insuficiente");
        fin = true;
      }
    }
}
public class Cliente extends Thread{
    private String nombre;
    private Cuenta cuenta;
    private int cantidad;
    private boolean fin = false;

    public Cliente(String nombre, Cuenta cuenta){
      this.nombre = nombre;
```

```
    this.cuenta = cuenta;
  }
  public int aleatorio(){
    cantidad = (int) (random()*500+1);
    return cantidad;
  }
  public void run() {
      try {  for (int i = 0; i < 4; i++) {
          if(!cuenta.getFin()){
            if (i % 2 == 0) {   cuenta.ingreso(aleatorio(), nombre);
              sleep(500);
            } else {    cuenta.reintegro(aleatorio(), nombre);
              sleep(500);
            }
          }else {  break;   }
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
  }
}
```

GALLINERO SINCRONIZADO

```
public static void main(String[] args) {
  Gallinero gallinero = new Gallinero();
  Granjero granjero = new Granjero(gallinero);
  for (int i = 0; i < 5; i++) {
    Gallina gallina = new Gallina((i+1), gallinero);
    gallina.start();
  }
  granjero.start();
}
public class Granjero extends Thread {
  private Gallinero gallinero;
  private int huevos;
  public int rotos;

  public Granjero(Gallinero gallinero) {
    this.gallinero = gallinero;
    huevos = 0;
  }
  public synchronized void recogerHuevos(){
    for (int i = 0; i < gallinero.ponedero.length; i++) {
      if(gallinero.ponedero[i] == 1){
        huevos++;
        System.out.println("El granjero ha recogido un huevo");
        gallinero.ponedero[i] = 0;
      }
    }
  }
  public void run() {
    for (int i = 0; i < 365; i++) {
      try {
        sleep(12);
        recogerHuevos();
        sleep(12);

      } catch (InterruptedException ex) {
        System.out.println("Error en el granjero");
```

```
        }
      }
      rotos = gallinero.getRotos();
      System.out.println("El granjero ha recogido " + huevos + " huevos");
      System.out.println("El granjero ha perdido " + rotos + " huevos");
    }
}
public class Gallinero {
    public int[] ponedero = new int[5];
    private int rotos = 0;

    public void addhuevo(int n) {
      ponedero[n] = 1;
    }

    public int getRotos() {
      return rotos;
    }

    public void rompeHuevo(int n) {
      ponedero[n] = 0;
      rotos++;
    }
}
public class Gallina extends Thread{
    private int id = 0;
    private Gallinero gallinero;

    public Gallina(int id, Gallinero gallinero) {
      this.id = id;
      this.gallinero = gallinero;
    }
    public synchronized void ponerHuevo(){
        if(gallinero.ponedero[id] == 1){
          gallinero.rompeHuevo(id);
          System.out.println("La gallina "+id+" ha roto un huevo");
        }
        gallinero.addhuevo(id);
      System.out.println("La gallina "+id+" ha puesto un huevo");
    }
    public void run(){
      for(int i = 0; i < 365; i++){
        int random = (int) (Math.random() * 24)+1;
        try {
          sleep(random);
          ponerHuevo();
          sleep(24 - random);
          System.out.println("----------------------------------------");
        } catch (InterruptedException ex) {
          System.out.println("Error en la gallina");
        }
      }
    }
}
FIESTA CON MONITORES
public static void main(String[] args) {
      Fiesta fiesta = new Fiesta();
      Cliente[] clientes = new Cliente[50];
```

```
        for (int i = 0; i < 50; i++) {
            try{
                Thread.sleep(100);
                clientes[i] = new Cliente(fiesta, i);
                clientes[i].start();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        Barra barra_izq = new Barra(fiesta, 0);
        Barra barra_der = new Barra(fiesta, 1);
        barra_izq.start();
        barra_der.start();
    }
}
public class Fiesta {
    // Número de clientes en la cola de la barra izquierda
    private int cola_izq = 0;
    // Número de clientes en la cola de la barra derecha
    private int cola_der = 0;

    // Número de copas servidas
    private int total_der = 0;
    private int total_izq = 0;

    // Número de copas servidas en total
    private int total_copas = 0;

    // Número de turnos de la barra izquierda y derecha
    private int total_turnos_izq = 0;
    private int total_turnos_der = 0;

    // Método para pedir turno en la barra derecha
    public synchronized void pedir_der(int numero)
    {
        // Se incrementa la cola de la barra derecha
        cola_der++;
        System.out.println("--------El cliente " + numero + " va a la barra derecha.");
        notifyAll();
    }
    // Método para beber en la barra derecha
    public synchronized void beber_der(int numero)
    {
        // Se obtiene el turno del cliente y se incrementa el total de turnos
        int turno = total_der;
        total_der++;
        // Mientras el turno del cliente sea mayor que el total de turnos de la barra derecha se espera
        while( turno > total_turnos_der)
        {
            try
            {
                //MostrarEstado("-----------El Cliente " + numero + " ESPERA PARA RECOGER LA COPA.----------");
                wait();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        // Se incrementa el total de turnos de la barra derecha
```

```
    total_turnos_der++;
    notifyAll();
  }

  // Método para pedir turno en la barra izquierda
  public synchronized void pedir_izq(int numero)
  {
    // Se incrementa la cola de la barra izquierda
    cola_izq++;
    System.out.println("--------El Cliente" + numero + " va a la barra izquierda.");
    notifyAll();
  }

  // Método para beber en la barra izquierda
  public synchronized void beber_izq(int numero)
  {
    // Se obtiene el turno del cliente y se incrementa el total de turnos
    int turno = total_izq;
    total_izq++;
    // Mientras el turno del cliente sea mayor que el total de turnos de la barra izquierda se espera
    while(turno > total_turnos_izq)
    {
      try
      {
        //System.out.println("-----------El cliente " + numero + " ESPERA PARA RECOGER LA COPA.----------");
        wait();
      } catch (InterruptedException e) {
        throw new RuntimeException(e);
      }
    }
    // Se incrementa el total de turnos de la barra izquierda
    total_turnos_izq++;
    notifyAll();
  }

  // Método para obtener el número de clientes en la cola de la barra izquierda
  public synchronized int cola_barra_izq()
  {
    return cola_izq;
  }

  // Método para obtener el número de clientes en la cola de la barra derecha
  public synchronized int cola_barra_der()
  {
    return cola_der;
  }

  // Método para obtener el total de copas servidas
  public int obtener_total_copas()
  {
    return total_copas;
  }

  // Método para servir copas en la barra izquierda
  public synchronized void sirve_copas_izq()
  {
    // Se decrementa la cola de la barra izquierda y se incrementa el total de copas servidas
    cola_izq--;
```

```
    total_copas++;
  }
  // Método para servir copas en la barra derecha
  public synchronized void sirve_copas_der()
  {
    // Se decrementa la cola de la barra derecha y se incrementa el total de copas servidas
    cola_der--;
    total_copas++;
  }

}
public class Cliente extends Thread{
  // Número de copas que puede tomar un cliente
  private static int COPAS = 5;
  // Fiesta a la que asiste el cliente
  private Fiesta fiesta;
  // Número del cliente
  private int numero;

  public Cliente( Fiesta fiesta, int numero)  {
    this.fiesta = fiesta;
    this.numero = numero;
  }
  public void run()  {
    try    {
      // El cliente pide 5 copas
      for(int x = 0; x < COPAS; x++)       {
        // El cliente pide una copa en la barra que menos cola tenga y espera a que le sirvan la copa para beberla y bailar un rato
        System.out.println("--------En la cola izquierda hay: "+fiesta.cola_barra_izq()+ " en la cola derecha hay:
"+fiesta.cola_barra_der());
        if(fiesta.cola_barra_izq() < fiesta.cola_barra_der())          {
          fiesta.pedir_izq(numero);
          fiesta.beber_izq(numero);
        } else  {
          fiesta.pedir_der(numero);
          fiesta.beber_der(numero);
        }
        System.out.println("--------El cliente " + numero + " se pone a bailar.-");
        Thread.sleep((int)(Math.random() * 600) );
      }
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
}
public class Barra extends Thread{
  // Fiesta a la que asiste el cliente
  private Fiesta fiesta;
  // Número de la barra
  private int barra;
  // Variable para detener el hilo
  private boolean fin;

  public Barra(Fiesta fiesta, int barra){
    this.fiesta = fiesta;
    this.barra = barra;
    this.fin = false;
  }
```

```
  public void run()   {
    try     {
      // Mientras no se acabe la fiesta se sirven copas
      while(!fin)        {
        // Si la barra es 1 se sirve en la derecha
        if(barra == 1)           {
          synchronized(fiesta) {
            //si no hay cola o ya está sirviendo se espera
            while (fiesta.cola_barra_der() == 0) {
              try {
                fiesta.wait();
              } catch (InterruptedException e) {
                throw new RuntimeException(e);
              }
            }
            // Se sirve la copa y se notifica a los demás hilos que estén esperando
            Thread.sleep(100);
            fiesta.sirve_copas_der();
            System.out.println("--------El camarero DER sirve la copa " + fiesta.obtener_total_copas() + ".");
            fiesta.notifyAll();
          }
        }
        //mismo funcionamiento para la barra izquierda
        else {
          // Si la barra es 0 se sirve en la izquierda
          synchronized (fiesta) {
            //si no hay cola o ya está sirviendo se espera
            while (fiesta.cola_barra_izq() == 0) {
              try {
                fiesta.wait();
              } catch (InterruptedException e) {
                throw new RuntimeException(e);
              }
            }
            // Se sirve la copa y se notifica a los demás hilos que estén esperando
            Thread.sleep(100);
            fiesta.sirve_copas_izq();
            System.out.println("------- El camarero IZQ sirve la copa " + fiesta.obtener_total_copas() + ".");
            fiesta.notifyAll();
          }
        }
      }
    }
    catch(Exception error) {}
  }
  public void detener()   {
    this.fin=true;
  }
}
PARKING CON SEMAFOROS
public static void main(String[] args) {
  Semaphore capacidad = new Semaphore(50);
  Semaphore semaforoEntrada = new Semaphore(1);
  Semaphore semaforoSalida = new Semaphore(1);

  Parking parking = new Parking(semaforoEntrada, semaforoSalida, capacidad, 0);

  for (int i = 1; i <= 250; i++) {
```

```
        try {
            sleep((int) (Math.random() * 50));
            new Coche(parking, i).start();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
public class Coche extends Thread {
    private Parking parking;
    private int id;

    public Coche(Parking p, int id) {
        parking = p;
        this.id = id;
    }

    public void run() {
        try {
            parking.entrar(id);
            sleep((int) (Math.random() * 200));
            parking.salir(id);
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
public class Parking {
    private Semaphore semaforoCapacidad;
    private Semaphore semaforoEntrada;
    private Semaphore semaforoSalida;
    private int capacidad;
    private int coches;

    public Parking(Semaphore semaforoEntrada, Semaphore semaforoSalida, Semaphore semaforoCapacidad, int aparcados)      {
        this.capacidad = 50;
        this.coches = aparcados;
        this.semaforoEntrada = semaforoEntrada;
        this.semaforoSalida = semaforoSalida;
        this.semaforoCapacidad = semaforoCapacidad;
    }
    public void entrar ( int id){
        try {
            semaforoCapacidad.acquire(); // Pide permiso para entrar a la sección crítica
            // Si hay sitio en el parking, el coche entra
            semaforoEntrada.acquire(); // Pide permiso para entrar
            // Sección crítica (entrada) -> Solo un hilo puede entrar a la vez
                // Si hay sitio en el parking, el coche entra
                coches++;
                System.out.println("Coche " + id + " entra. Hay " + coches + " coches\n");
                if(coches == capacidad){
                    System.out.println("Parking lleno\n");
                }
            semaforoEntrada.release(); // Libera el semáforo de entrada
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
```

```
    public void salir (int id){
      try {
        semaforoSalida.acquire();// Pide permiso para salir
        // Sección crítica (salida) -> Solo un hilo puede salir a la vez
          // El coche sale del parking
          coches--;
          System.out.println("Coche " + id + " sale. Hay " + coches + " coches\n");
          if(coches == capacidad-1){
            System.out.println("PLAZAS DISPONIBLES\n");
          }
        semaforoSalida.release(); // Libera el semáforo de salida
        semaforoCapacidad.release(); // Libera el semáforo de capacidad
      } catch (Exception e) {
        System.out.println(e.getMessage());
      }
    }
}
```

PARKING LLENO (CLASES PARKING Y COCHE IGUALES)

```
public class CocheAparcado extends Thread{
    // añadimos esta clase para sacar los coches que ya hay dentro
    private Parking parking;
    private int id;

    public CocheAparcado(Parking p, int id) {
      parking = p;
      this.id = id;
    }

    public void run() {
      try {
        sleep((int) (Math.random() * 200));
        parking.salir(id);
      } catch (InterruptedException e) {
        System.out.println(e.getMessage());
      }
    }
}
        public class Main {
          public static void main(String[] args) {
        // iniciamos el semáforo de la capacidad a 0 para que ningún coche pueda entrar hasta que no slaga alguno y al paring le
        pasamos la cantidad de coches aparcados que hay (50)
            Semaphore semaforoCapacidad = new Semaphore(0);
            Semaphore semaforoEntrada = new Semaphore(1);
            Semaphore semaforoSalida = new Semaphore(1);
            Parking parking = new Parking(semaforoEntrada, semaforoSalida, semaforoCapacidad, 50);      for (int i = 1; i <= 250;
        i++) {
              try {
                if(i <= 50) {
                  int n = i + 250;
                  new CocheAparcado(parking, n).start();
                }
                sleep((int) (Math.random() * 50));
                new Coche(parking, i).start();
              } catch (InterruptedException e) {
                e.printStackTrace();
              }
            }   } }
```