Act 2.9.- Puente doble sentido

Main

```
public class Main {

    public static void main(String[] args) {
        Puente puente = new Puente();
        Random r = new Random();

        Persona[] personas = new Persona[15];
        for (byte i = 0; i < 15; i++) {
            personas[i] = new Persona("Persona " + i, puente);
        }

        for (Persona persona : personas) {
            try {
                int tiempoLlegada = r.nextInt(origin: 1000,bound: 30001);
                System.out.printf(format: "%sTiempo de llegada de una nueva persona es %d segundos\n",args: "\u001B[31m", tiempoLlegada/1000);
                Thread.sleep(millis: tiempoLlegada);
                persona.start();
            } catch (InterruptedException ex) {
                System.out.printf(format: "ERROR: %s\n", args: ex.getMessage());
            }
        }

    }

}
```

Yo he limitado las personas que cruzan el puente a 15 para la simulación.

Persona

```
public class Persona extends Thread {

    private final String nombre;
    private final Puente puente;
    private final int peso;

    public Persona(String nombre, Puente puente) {
        this.nombre = nombre;
        this.puente = puente;
        this.peso = generarPeso();
    }

    public String getNombre() {
        return nombre;
    }

    public int getPeso() {
        return peso;
    }

    private int generarPeso() {
        Random r = new Random();
        return r.nextInt(origin: 40, bound: 121);
    }
```

```java
    @Override
    public void run() {

        Random r = new Random();

        try {
            if (r.nextBoolean()) {
                puente.cruzarPuenteDerecha(p: this);
            } else {
                puente.cruzarPuenteIzquierda(p: this);
            }

            long t = r.nextInt(origin: 10000, bound: 50001);

            while (t > 100) {
                Thread.sleep(millis: 10000);
                System.err.println(getNombre() + " me quedan " + t / 1000 + " segundos minutos para terminar de cruzar");
                t -= 10000;
            }
            puente.terminarCruzarPuente(p: this);
        } catch (InterruptedException ex) {
            System.out.printf(format: "ERROR: %s\n", args: ex.getMessage());
        }

    }

}
```

Puente

```java
public class Puente {

    private int cantidadPersonas = 0;
    private int cantidadPesoActual = 0;

    public Puente() {
    }

    public synchronized void cruzarPuenteDerecha(Persona p) throws InterruptedException {
        while (cantidadPersonas >= 4 || cantidadPesoActual + p.getPeso() > 250) {
            System.out.printf(format: "[%s] con peso %skg. Espera a cruzar el puente.\n", args: p.getNombre(), args: p.getPeso());
            wait();
        }

        cantidadPersonas++;
        cantidadPesoActual += p.getPeso();

        System.out.printf(format: "[%s] cruza el puente desde la derecha.\n", args: p.getNombre());
        System.out.printf(format: "Personas en el puente: %d - Peso en el puente: %d\n", args: cantidadPersonas, args: cantidadPesoActual);
    }
```

```java
    public synchronized void cruzarPuenteIzquierda(Persona p) throws InterruptedException {
        while (cantidadPersonas >= 4 || cantidadPesoActual + p.getPeso() > 250) {
            System.out.printf(format: "[%s] con peso %skg. Espera a cruzar el puente.\n", args: p.getNombre(), args: p.getPeso());
            wait();
        }

        cantidadPersonas++;
        cantidadPesoActual += p.getPeso();

        System.out.printf(format: "[%s] cruza el puente desde la izquierda.\n", args: p.getNombre());
        System.out.printf(format: "Personas en el puente: %d - Peso en el puente: %d\n", args: cantidadPersonas, args: cantidadPesoActual);
    }

    public synchronized void terminarCruzarPuente(Persona p) {
        cantidadPersonas--;
        cantidadPesoActual -= p.getPeso();
        System.out.printf(format: "[%s] con peso %skg. Termino de cruzar el puente.\n", args: p.getNombre(), args: p.getPeso());
        System.out.printf(format: "Personas en el puente: %d - Peso en el puente: %d\n", args: cantidadPersonas, args: cantidadPesoActual);
        notifyAll();
    }

}
```

Resultado

```
run:
Tiempo de llegada de una nueva persona es 1 segundos
Tiempo de llegada de una nueva persona es 21 segundos
[Persona 0] cruza el puente desde la derecha.
Personas en el puente: 1 - Peso en el puente: 92
Persona 0 me quedan 14 segundos minutos para terminar de cruzar
Persona 0 me quedan 4 segundos minutos para terminar de cruzar
[Persona 0] con peso 92kg. Termino de cruzar el puente.
Personas en el puente: 0 - Peso en el puente: 0
Tiempo de llegada de una nueva persona es 15 segundos
[Persona 1] cruza el puente desde la izquierda.
Personas en el puente: 1 - Peso en el puente: 71
Persona 1 me quedan 39 segundos minutos para terminar de cruzar
Tiempo de llegada de una nueva persona es 23 segundos
[Persona 2] cruza el puente desde la izquierda.
Personas en el puente: 2 - Peso en el puente: 143
Persona 1 me quedan 29 segundos minutos para terminar de cruzar
Persona 2 me quedan 41 segundos minutos para terminar de cruzar
Persona 1 me quedan 19 segundos minutos para terminar de cruzar
Persona 2 me quedan 31 segundos minutos para terminar de cruzar
Tiempo de llegada de una nueva persona es 15 segundos
[Persona 3] cruza el puente desde la izquierda.
Personas en el puente: 3 - Peso en el puente: 233
Persona 1 me quedan 9 segundos minutos para terminar de cruzar
[Persona 1] con peso 71kg. Termino de cruzar el puente.
Personas en el puente: 2 - Peso en el puente: 162
Persona 2 me quedan 21 segundos minutos para terminar de cruzar
Persona 3 me quedan 18 segundos minutos para terminar de cruzar
Tiempo de llegada de una nueva persona es 8 segundos
[Persona 4] cruza el puente desde la derecha.
Personas en el puente: 3 - Peso en el puente: 250
Persona 2 me quedan 11 segundos minutos para terminar de cruzar
Persona 3 me quedan 8 segundos minutos para terminar de cruzar
```

Act 2.9.- Parking Camiones

Main

```java
public class Main {

    public static void main(String[] args) {
        Parking parking = new Parking();

        for (byte i = 0; i < 50; i++) {
            if (i % 13 < 10) {
                new Thread(new Coche(parking)).start();
            } else {
                new Thread(new Camion(parking)).start();
            }

            try {
                Thread.sleep(millis: 1500);
            } catch (InterruptedException ex) {
                System.out.printf(format: "ERROR: %s\n", args: ex.getMessage());
            }
        }

    }

}
```

He realizado la simulación con 50 vehículos.

Vehículo

```
class Vehiculo {

    protected final String matricula;
    private int posicionParking;

    public Vehiculo() {
        this.matricula = asignarMatricula();
        this.posicionParking = 0;
    }

    public String getMatricula() {
        return matricula;
    }

     public int getPosicionParking() {
        return posicionParking;
    }

    public void setPosicionParking(int posicionParking) {
        this.posicionParking = posicionParking;
    }
```

```
    private String asignarMatricula() {
        Random r = new Random();
        String ret = "[";
        ret += String.format(format: "%04d", args: r.nextInt(bound: 10000));
        ret += " ";

        for (byte i = 0; i < 3; i++) {
            char c = (char) ('A' + r.nextInt(bound: 26));
            ret += c;
        }
        ret += "]";
        return ret;
    }
}
```

Camion

```
class Camion extends Vehiculo implements Runnable {
    private Parking parking;

    public Camion(Parking parking) {
        super();
        this.parking = parking;
    }

    @Override
    public void run() {
        try {
            Random r = new Random();
            parking.accederParking(v: this);
            Thread.sleep(millis: r.nextInt(origin: 20000, bound: 30001));
            parking.salirParking(v: this);
        } catch (InterruptedException ex) {
            System.out.printf(format: "ERROR: %s\n", args: ex.getMessage());
        }
    }
}
```

Coche

```
class Coche extends Vehiculo implements Runnable {
    private Parking parking;

    public Coche(Parking parking) {
        super();
        this.parking = parking;
    }

    @Override
    public void run() {
        try {
            Random r = new Random();
            parking.accederParking(v: this);
            Thread.sleep(millis: r.nextInt(origin: 20000, bound: 30001));
            parking.salirParking(v: this);
        } catch (InterruptedException ex) {
            System.out.printf(format: "ERROR: %s\n", args: ex.getMessage());
        }
    }
}
```

Parking

```java
public class Parking {

    private final Map<Integer, Vehiculo> aparcamientos = new HashMap<>(initialCapacity:10);
    private final LinkedList<Integer> posicionesLiberadas = new LinkedList<>();
    private int plazasLibres;

    public Parking() {
        plazasLibres = 10;
    }

    public Map<Integer, Vehiculo> getMap() {
        return aparcamientos;
    }

    public synchronized void accederParking(Vehiculo v) throws InterruptedException {
        while (plazasLibres == 0) {
            System.out.printf(format: "%s con matricula%s espera a que queden plazas libres\n", args: TipoVehiculo(v), args: v.getMatricula());
            wait();
        }

        if (v.getClass().equals(obj:Coche.class) && plazasLibres > 1) {
            aparcarCoche(v);
        } else if (v.getClass().equals(obj:Camion.class) && plazasLibres > 2) {
            aparcarCamion(v);
        }

    }
```

```java
    public synchronized void salirParking(Vehiculo v) throws InterruptedException {
        int posicion = v.getPosicionParking();
        this.aparcamientos.remove(key:posicion);
        System.out.printf(format: "%s con matricula %s deja la plaza %d \n", args: TipoVehiculo(v), args: v.getMatricula(), posicion + 1);
        plazasLibres++;
        posicionesLiberadas.offer(e: posicion);
        notifyAll();

    }

    public synchronized void aparcarCoche(Vehiculo v) {
        int posicion;
        if (!posicionesLiberadas.isEmpty()) {
            posicion = posicionesLiberadas.poll();
        } else {
            posicion = aparcamientos.size();
        }

        while (aparcamientos.containsKey(key:posicion) && posicion < 10) {
            posicion++;
        }

        if (posicion < 10) {
            this.aparcamientos.put(key:posicion, value: v);
            v.setPosicionParking(posicionParking:posicion);
            plazasLibres--;

            System.out.printf(format: "%s con matricula %s aparca en la plaza %d -> Quedan %d plazas libres\n", args: TipoVehiculo(v), args: v.getMatricula(), posicion + 1,
                    args: plazasLibres);
        } else {
            System.out.printf(format: "%s con matricula %s no pudo aparcar, no hay plazas disponibles\n", args: TipoVehiculo(v), args: v.getMatricula());
        }
    }
```

```java
    public synchronized void aparcarCamion(Vehiculo v) {
        int posicionPar;
        int posicionImpar;

        if (!posicionesLiberadas.isEmpty()) {
            posicionPar = posicionesLiberadas.poll();
            posicionImpar = posicionesLiberadas.poll();
        } else {
            posicionPar = aparcamientos.size();
            posicionImpar = posicionPar + 1;
        }

        while ((aparcamientos.containsKey(key:posicionPar) || aparcamientos.containsKey(key:posicionImpar)
                || aparcamientos.containsKey(posicionPar + 1) || aparcamientos.containsKey(posicionImpar + 1))
                && (posicionPar < 10 && posicionImpar < 10)) {
            posicionPar += 2;
            posicionImpar += 2;
        }

        if ((posicionPar < 10 && posicionImpar < 10)) {
            this.aparcamientos.put(key:posicionPar, value: v);
            this.aparcamientos.put(posicionPar + 1, value: v);
            v.setPosicionParking(posicionParking:posicionPar);

            plazasLibres -= 2;

            System.out.printf(format: "%s con matricula %s aparca en las plazas %d y %d -> Quedan %d plazas libres\n", args: TipoVehiculo(v),
                    args: v.getMatricula(), posicionPar + 1, posicionPar + 2, args: plazasLibres);
        } else {
            System.out.printf(format: "%s con matricula %s no pudo aparcar, no hay plazas disponibles\n", args: TipoVehiculo(v), args: v.getMatricula());
        }
    }
```

```java
    public synchronized String TipoVehiculo(Vehiculo v) {
        if (v.getClass().equals(obj:Coche.class)) {
            return "Coche";
        } else {
            return "Camion";
        }
    }
}
```

Resultado

```
Coche con matricula [8949 TJP] aparca en la plaza 1 -> Quedan 9 plazas libres
Coche con matricula [5634 RCI] aparca en la plaza 2 -> Quedan 8 plazas libres
Coche con matricula [8786 WYK] aparca en la plaza 3 -> Quedan 7 plazas libres
Coche con matricula [6306 WIK] aparca en la plaza 4 -> Quedan 6 plazas libres
Coche con matricula [5621 IQW] aparca en la plaza 5 -> Quedan 5 plazas libres
Coche con matricula [2743 MMY] aparca en la plaza 6 -> Quedan 4 plazas libres
Coche con matricula [5596 RLN] aparca en la plaza 7 -> Quedan 3 plazas libres
Coche con matricula [8157 GEA] aparca en la plaza 8 -> Quedan 2 plazas libres
Coche con matricula [8499 MKJ] aparca en la plaza 9 -> Quedan 1 plazas libres
Coche con matricula [5634 RCI] deja la plaza 2
Coche con matricula [6299 JMA] aparca en la plaza 2 -> Quedan 1 plazas libres
Coche con matricula [8786 WYK] deja la plaza 3
Coche con matricula [0119 ECA] aparca en la plaza 3 -> Quedan 1 plazas libres
Coche con matricula [6306 WIK] deja la plaza 4
Coche con matricula [2476 EQO] aparca en la plaza 4 -> Quedan 1 plazas libres
Coche con matricula [8949 TJP] deja la plaza 1
Coche con matricula [5552 TJQ] aparca en la plaza 1 -> Quedan 1 plazas libres
Coche con matricula [2743 MMY] deja la plaza 6
Coche con matricula [0632 RFL] aparca en la plaza 6 -> Quedan 1 plazas libres
Coche con matricula [5621 IQW] deja la plaza 5
Coche con matricula [6465 ZOE] aparca en la plaza 5 -> Quedan 1 plazas libres
Coche con matricula [5596 RLN] deja la plaza 7
Coche con matricula [8157 GEA] deja la plaza 8
Camion con matricula [4067 GMU] no pudo aparcar, no hay plazas disponibles
Camion con matricula [3798 RJA] no pudo aparcar, no hay plazas disponibles
Camion con matricula [4497 ZSR] no pudo aparcar, no hay plazas disponibles
Coche con matricula [2484 UUG] deja la plaza 1
Coche con matricula [8499 MKJ] deja la plaza 9
```