Grado en Ingeniería Informática

Estructura de Datos y Algoritmos, Grupo 84M, 2014/2015

12 de Marzo de 2015

Nombre y Apellidos:

……

PROBLEMA 1 (1 punto) – Programación Orientada a Objetos.

Una empresa de alquiler de vehículos solicita el desarrollo de una aplicación para las reservas y alquileres de sus vehículos.

En la primera versión de la aplicación solo se podrán gestionar a través de ella reservas y alquileres de vehículos tipo coche y motocicleta, pero debemos tener en cuenta que en siguientes versiones se extenderá la funcionalidad a otros tipos de vehículos (camiones, furgonetas, etc.).

Los datos que será necesario almacenar del vehículo son los siguientes:

- Marca del vehículo: String.
- Modelo del vehículo: String.
- Matrícula del vehículo: String.
- Disponible: Indica si el vehículo está disponible para el alquiler. Boolean
- Lista de las reservas: Un cliente puede apuntarse a una lista de reservas para poder alquilar un determinado vehículo si no está disponible. Almacena los Nombres y Apellidos de los clientes. El máximo de reservas para el coche es de 5 clientes y para la moto es de solo de 1 cliente.
- Número total de veces que ha sido alquilado un vehículo.

Además hay una serie de funciones en la aplicación que serán comunes para todos los tipos de vehículos (coches, motos, etc.) pero otras pueden implementarse de diferente forma en función del tipo de vehículo.

Se pide desarrollar la(s) estructura(s) de datos necesaria(s) para implementar la solución. En concreto, será necesario:

- Definición de clases (clases, clases abstractas, interfaces, etc.) necesarias para reflejar toda la semántica del problema.
- método(s) constructor(es) que reciba los parámetros necesarios para crear un vehículo (marca, modelo y matricula). Además, cada vez que un vehículo se crea se debe pone a cero el contador total de alquileres de ese vehículo y el vehículo se marca como disponible en la aplicación.
    - Es importante recordar que en el caso del vehículo tipo coche la lista de reservas es de 5 clientes como máximo y para el vehículo tipo moto sólo es de 1 cliente.
- Para todos los vehículos se pide implementar el siguiente método:
    - estaDisponible() devuelve el estado de un vehículo para saber si se puede alquilar.
    - devolverVehiculo() se actualiza el estado del vehículo para que vuelva a estar disponible
- Para el vehículo tipo coche se pide implementar los siguientes métodos:

- reservarVehiculo() recibe como parámetro el Nombre y Apellidos del cliente que lo quiere reservar y lo guarda en la lista de reservas del vehículo (si es posible). **NO** es necesario comprobar que si el cliente ya aparece en la lista de reservas de un mismo vehículo.
- alquilarVehiculo() recibe como parámetro el Nombre y Apellidos del cliente que va a alquilar el vehículo. Se comprueba si el cliente aparece en la lista de reservas y se libera la reserva que había realizado). Además, se actualiza el estado del vehículo para que deje de estar disponible y se incrementa el número total de veces que ha sido alquilado.

- Para el vehículo tipo moto se pide implementar los siguientes métodos:
  - reservarVehiculo() recibe como parámetro el Nombre y Apellidos del cliente que lo quiere reservar y lo guarda (si es posible). Es necesario comprobar que la moto no esté reservada previamente por otro cliente.
  - alquilarVehiculo() recibe como parámetro el Nombre y Apellidos del cliente que va a alquilar el vehículo. Solo se puede alquilar la moto por un cliente si la moto no tiene ninguna reserva previa y si la tuviera si es del mismo cliente. Además, se libera la reserva que había realizado. Además, se actualiza el estado del vehículo para que deje de estar disponible y se incrementa el número total de veces que ha sido alquilado.

La solución propuesta debe estar diseñada bajos los principios de la programación orientada a objetos, que permitan obtener software robusto, reutilizable y fácil de adaptar.

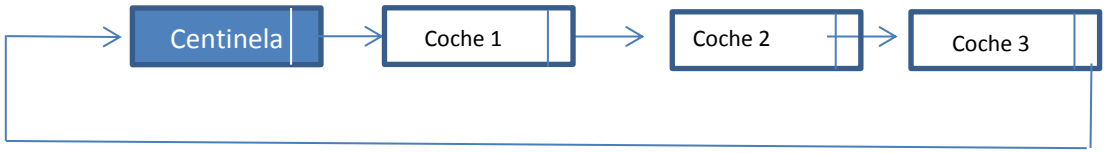
Notas:

- No es necesario implementar los métodos getters and setters.
- No es necesario implementar un método main o una clase Test para probar la aplicación, es suficiente con diseñar la(s) clase(s).

```java
public abstract class Vehiculo {

	protected String marca;
	protected String modelo;
	protected String matricula;
	protected boolean disponible;
	protected int totalAlquilado;
	protected String[] reservas;

	public boolean estaDisponible() {
		return this.disponible;
	}

	public void devolverVehiculo() {
		this.disponible = true;
	}

	public abstract void reservarVehiculo(String NombreApellidos);

	public abstract void alquilarVehiculo(String NombreApellidos);

}



public class Motocicleta extends Vehiculo {

	public Motocicleta(String marca, String modelo, String matricula) {
		this.marca = marca;
		this.modelo = modelo;
		this.matricula = matricula;
		this.disponible = true;
		this.totalAlquilado = 0;
		this.reservas = new String[1];
	}

	public void reservarVehiculo(String NombreApellidos) {
		if (this.reservas[0] != null)
			this.reservas[0] = NombreApellidos;
	}

	public void alquilarVehiculo(String NombreApellidos) {
		if (disponible) {
			if (this.reservas[0].equals(NombreApellidos)) {
				this.reservas[0] = null;
				this.disponible = false;
				this.totalAlquilado++;
			}
		}
	}

}
```

```java
public class Coche extends Vehiculo {

	public Coche(String marca, String modelo, String matricula) {
		this.marca = marca;
		this.modelo = modelo;
		this.matricula = matricula;
		this.disponible = true;
		this.totalAlquilado = 0;
		this.reservas = new String[5];
	}

	public void reservarVehiculo(String NombreApellidos) {
		for (int i = 0; i < this.reservas.length; i++) {
			if (this.reservas[i] == null) {
				this.reservas[i] = NombreApellidos;
				return;
			}
		}
	}

	public void alquilarVehiculo(String NombreApellidos) {
		if (disponible) {
			for (int i = 0; i < this.reservas.length; i++) {
				if (this.reservas[i].equals(NombreApellidos)) {
					this.reservas[i] = null;
					this.disponible = false;
					this.totalAlquilado++;
					return;
				}
			}
		}
	}

}
```

PROBLEMA 2 (1 punto)- Implementación de una lista circular de Vehículos tipo Coche, simplemente enlazada, usando un nodo centinela de la siguiente forma:



Esta lista se encarga de almacenar el los vehículos alquilados de tipo Coche (sin ningún tipo de ordenamiento). Como se puede observar en cada nodo se almacena un vehículo tipo coche.

```
public class SNodeVeh {

	Coche coche;
	public SNodeVeh nextNode = null;

	public SNodeVeh(Coche coche) {

		this.coche = coche;
	}
}
```

```
public class SListVeh {
		protected SNodeVeh centinela;

		public SListVeh(){
			centinela=new SNodeVeh(null);
			centinela.nextNode= centinela;
		}

		public boolean esVacia(){
			return (centinela.nextNode==centinela);
		}
}
```

Estas clases se corresponden con la implementación de una lista de vehículos tipo coche circular, simplemente enlazada, con un nodo centinela. Dicho de otra forma, la referencia **nextNode** del último Nodo de la lista circular apunta al nodo centinela (no al primer nodo), mientras que la referencia nextNode del nodo centinela apunta al primer nodo de la lista.

Se pide implementar los métodos insertaFinal(Coche coche) y borraFinal().

```
public class SNodeVeh {

	Coche coche;
	public SNodeVeh nextNode;

	public SNodeVeh(Coche coche) {
		this.coche = coche;
	}

}

public class SListVeh {

	protected SNodeVeh centinela;

	public SListVeh() {
		centinela = new SNodeVeh(null);
		centinela.nextNode = centinela;
	}

	public boolean esVacia() {
		return centinela.nextNode == centinela;
	}

	public void insertaFinal(Coche coche) {
		SNodeVeh newN = new SNodeVeh(coche);
		newN.nextNode = centinela;
		SNodeVeh last = centinela;
		for (SNodeVeh nodeIt = centinela.nextNode; nodeIt != centinela; nodeIt =
nodeIt.nextNode)
			last = nodeIt;
		last.nextNode = newN;
	}

	public void borraFinal() {
		SNodeVeh previous = centinela;
		for (SNodeVeh nodeIt = centinela.nextNode; nodeIt.nextNode != centinela;
nodeIt = nodeIt.nextNode)
			previous = nodeIt;
		previous.nextNode = centinela;
	}

}
```