## Ejercicio 3: Programación Dinámica

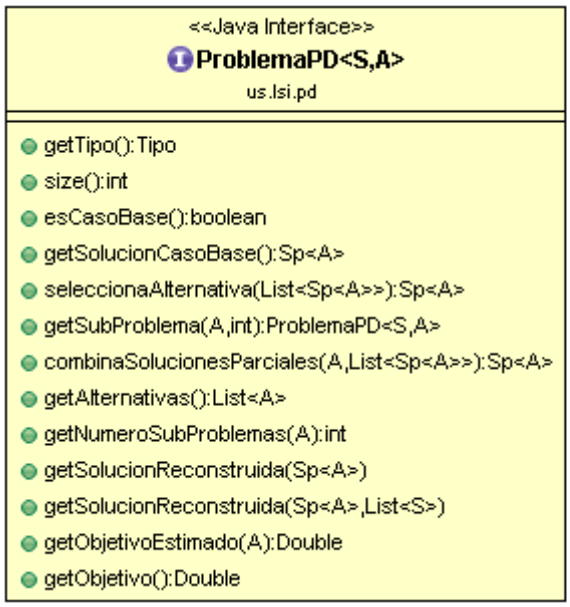
Para la renovación de la flota de autobuses de una empresa se han presentado diferentes ofertas de vehículos con un coste y una capacidad de viajeros. La empresa tiene un presupuesto que debe gastar por completo (presupuesto exacto). Desea maximizar la capacidad de transporte de viajeros y que los vehículos adquiridos sean todos diferentes. **Se pretende diseñar un algoritmo para escoger aquellos vehículos que hagan máximo el número de viajeros, usando exactamente el presupuesto establecido.** A través de un algoritmo de programación dinámica se debe encontrar cuáles son los vehículos que dan lugar al máximo número de viajeros usando exactamente el presupuesto establecido.

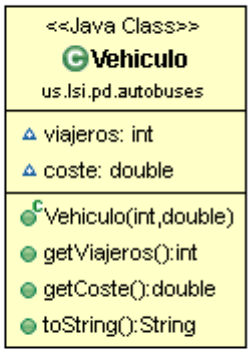
Un **Vehículo** permite almacenar toda la información sobre una propuesta. Los métodos más importantes son:

- Integer getViajeros(): Devuelve el número de viajeros de capacidad.
- Double getCoste() : Devuelve el coste asociado al vehículo.

El problema se modelará a través de la clase **ProblemaAutobusesPD**, que implementa la interfaz ProblemaPD <S,A>(ver ficha adjunta). Suponga que dicha clase contiene al menos los siguientes atributos:

- Double **presupuesto**: presupuesto exacto a gastar.
- List<Vehículo> **listaVehículos**: Lista que almacena todos los tipos de vehículos que existen.




**SE PIDE:**

a) Completar la ficha del problema.

b) Escribir el código completo de los métodos *esCasoBase*, *getSolucionCasoBase*, *getAlternativas*, *getSubProblema*, *combinaSolucionesParciales y seleccionaAlternativa* de la clase ProblemaAutobusesPD, teniendo en cuenta la ficha proporcionada.

**Puntuación: 33,33 %**

**Ejemplo**: La siguiente tabla muestra un ejemplo de datos de entrada. Para un presupuesto de 250 unidades la mejor opción es incluir en las propuestas 1 y 4, que agruparían 600 viajeros.

| Vehículo | Viajeros | Coste |
|---|---|---|
| 1 | 150 | 110 |
| 2 | 250 | 130 |
| 3 | 250 | 120 |
| 4 | 450 | 140 |

**Presupuesto: 250**

| ***Problema Áutobuses*** | |
|---|---|
| *Técnica: Programación Dinámica* | |
| *Propiedades Compartidas* | *listaVehículos, List<Vehiculo>*<br>*presupuesto, double* |
| *Propiedades Individuales* | *c, double*<br>*j, integer* |
| *Solución: List<Vehiculo> Lista con los vehículos seleccionados* | |
| *Alternativas:*<br>$A_{c,j}$ = c >= listaVehículos.get(j).getCoste()? {true, false} : {false} | |
| *Instanciación:*<br>*pa(presupuesto, listaVehiculos)=pv(presupuesto, listaVehículos.size() - 1)* | |

$$pv(c, j) = \begin{cases} (false, 0) & j = 0 \wedge c = 0 \\ (true, \text{listaVehículos[j].getPasajeros()}) & j = 0 \wedge \text{c} = \text{listaVehículos[j].coste} \\ \perp & j = 0 \wedge c \neq 0 \wedge \text{c} \neq \text{listaVehículos[j].coste} \\ sA_{a \in A_{c,j}}(cS(a, pv(\text{c} - (\text{a?listaVehículos[j].coste: } 0.0), j-1)) & j > 0 \end{cases}$$

cS(a,($s_1$,$v_1$)):
(a, (a?listaVehículos[j].viajeros:0) + $v_1$);

$sA_{a \in A_{c,j}}(r_s)$:
Elige la solución parcial con mayor valor de la propiedad a optimizar (más vehículos)

```
public class ProblemaAutobusesPD implements ProblemaPD<List<Vehiculo>,
Boolean> {

public boolean esCasoBase() {
	return j == 0;
}

public Sp<Boolean> getSolucionCasoBase() {
	Sp<Boolean> ret = null;
	if (c == 0){
		ret = Sp.create(false, 0.0);
	}else if (c == listaVehiculos.get(j).getCoste()){
		ret = Sp.create(true,
(double)listaVehiculos.get(j).getViajeros());
	}else{
		ret = null;
	}
	return ret;
}

public List<Boolean> getAlternativas() {
	List<Boolean> ls = new ArrayList<Boolean>();
	if (c >= listaVehiculos.get(j).getCoste()){
		ls.add(true);
	}
	ls.add(false);
	return ls;
}

public ProblemaPD<List<Vehiculo>, Boolean> getSubProblema(Boolean a, int np)
{
	return ProblemaAutobusesPD.create(c -
(a?listaVehiculos.get(j).getCoste():0.0), j - 1);
}

public Sp<Boolean> combinaSolucionesParciales(Boolean a, List<Sp<Boolean>>
ls) {
	Sp<Boolean> r = ls.get(0);
	return Sp.create(a, (a?listaVehiculos.get(j).getViajeros():0) +
r.propiedad);
}

public Sp<Boolean> seleccionaAlternativa(List<Sp<Boolean>> ls) {
	Sp<Boolean> r =ls.stream().filter(x -> x.propiedad != null)
			.max(Comparator.naturalOrder()).orElse(null);
	return r;
}
```