ADDA	2º Parcial	Curso 2017/2018

Ejercicio 2

Un paciente de una determinada enfermedad tiene una cierta carga patológica inicial. Cuando dicha carga es igual o menor que cero se considera que el paciente está curado. La carga patológica se puede reducir mediante la aplicación de uno o varios tratamientos médicos. Existe un límite respecto al número de veces que un mismo tratamiento se puede aplicar a un paciente; dicho límite es igual para todos los tratamientos. La aplicación de un tratamiento consume un determinado número de días, y no se puede detener una vez iniciada.

Se desea construir un programa mediante *Programación Dinámica* que calcule el número de veces que cada tratamiento se puede aplicar a un paciente de manera que éste se cure en el menor tiempo posible, teniendo en cuenta que el orden de los tratamientos en la solución no importa.

SE PIDE:

a) Rellene los apartados de la ficha marcados con **TODO** (no escriba en el enunciado; utilice otra hoja para ello).

b) Escriba el código de los siguientes métodos:

- List<Integer> **getAlternativas** ()
- ProblemaPD<Multiset<TratamientoMedico>, Integer> **getSubproblema** (Integer a, int np)
- Sp<Integer> **getSolucionParcialPorAlternativa** (Integer a, List<Sp<Integer>> ls)

Problema de los Tratamientos Médicos	
***Técnica**: Programación Dinámica*	
Tipos	• $S - Multiset<TratamientoMedico>$ • $A - Integer$
Propiedades Compartidas	*cargaInicial, Integer, carga patológica inicial del paciente, básica.* *maxVeces, Integer, máximo número de veces que un mismo tratamiento se puede aplicar, básica.* *listaTM, List<TratamientoMedico>, tratamientos disponibles, básica.* *nTrat, Integer, número de tratamientos médicos disponibles, derivada.* *reduccionCarga(i), Integer, reducción de carga patológica que se consigue al aplicar el tratamiento i-ésimo, i en* [*0, nTrat*)*, derivada.* *duración(i), Integer, días necesarios para el tratamiento i-ésimo, i en* [*0, nTrat*)*, derivada.*
Propiedades Individuales	*cargaRestante, Integer, carga patológica restante para el paciente.* *k, Integer en* [*0, nTrat*]*, siguiente tratamiento a considerar.*
***Solución:** Multiset<TratamientoMedico>*	
***Solución Parcial (TODO)** (a, d)* *Siendo d el número de días que lleva el paciente bajo tratamientos.*	
Alternativas (TODO): $A = [0, maxVeces]$ *Otra opción: A=[0, p] donde:* *p= min (maxVeces, (cargaRestante % reduccionCarga(k) == 0 ?* *cargaRestante / reduccionCarga(k) : cargaRestante / reduccionCarga(k) + 1))*	
Instanciación (TODO) $Inicial = (cargaInicial, 0)$ *Asumimos que el problema inicial con cargaInicial=0 tiene la solución ()*	

Casos base (TODO)

$cargaRestante \leq 0 \;\; || \;\; k = nTrat$

Solución casos base (TODO):

$$\begin{cases} (?, 0.0), & si\ cargaRestante \leq 0 \\ \perp, & e.o.c. \end{cases}$$

Número de subproblemas (TODO): 1

Subproblemas (TODO)

$p = (cargaRestante, k) \xrightarrow{a}$

$p_a = (cargaRestante - a * reduccionCarga(k), k + 1)$

Esquema recursivo (TODO)

$$sp(cargaRestante, k) = \begin{cases} (?, 0.0), & cargaRestante \leq 0 \\ \perp, & cargaRestante > 0 \; y \; k = nTrat \\ \min\limits_{a \in A} \{sA(a, sp(p_a))\}, & e.o.c. \end{cases}$$

sA (TODO) $\quad sA(a, (a', d)) = (a, d + a * duracion(k))$

sP (TODO): *elige la solución con menor d*

Solución reconstruida (TODO)

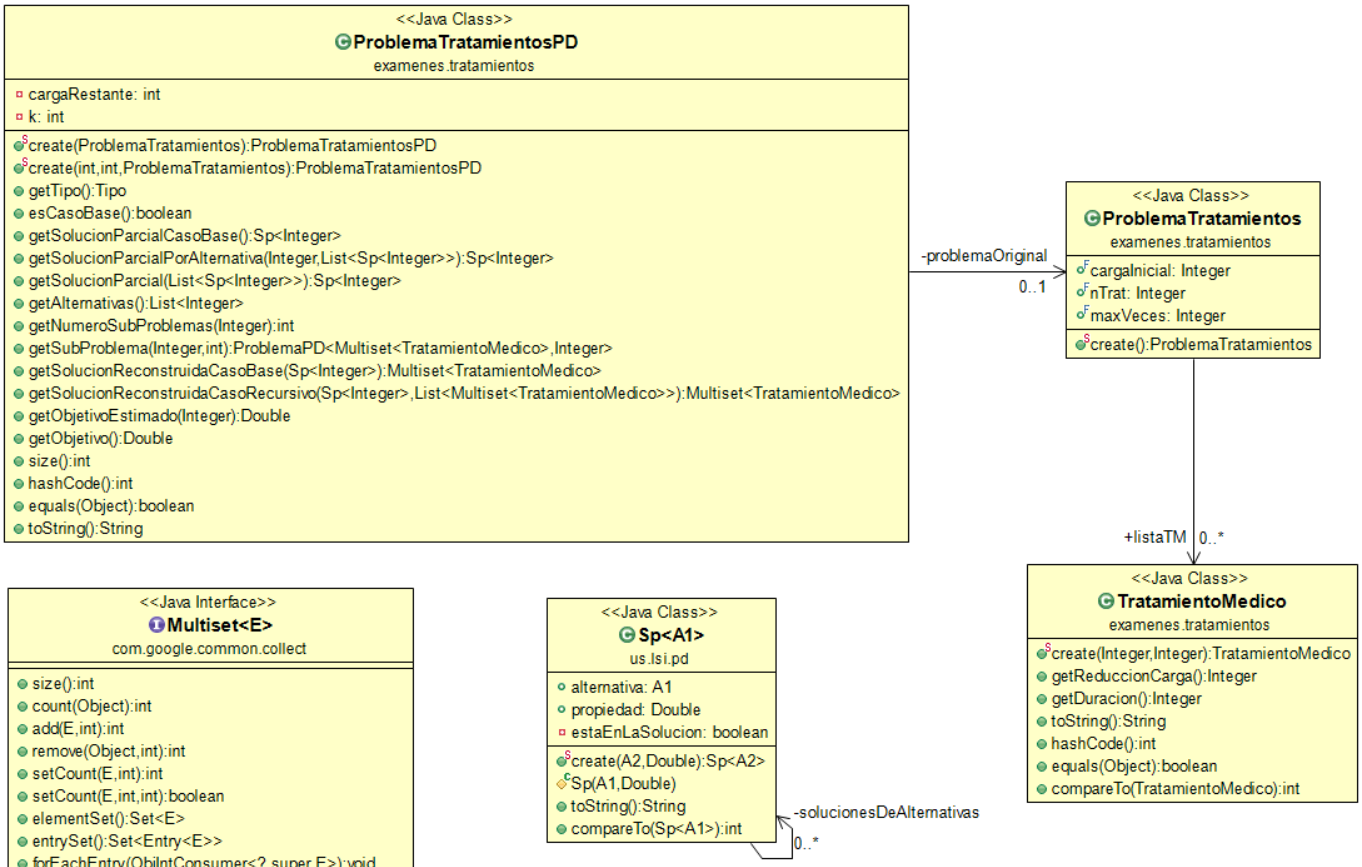
$sr((a, d)) = \{\}\ (caso\ base)$

$sr((a, d), s)) = s + (listaTM_k, a)$ *(caso recursivo)*

Los métodos create de la clase ProblemaTratamientosPD tienen los siguientes perfiles:

```
public static ProblemaTratamientosPD create (ProblemaTratamientos
                                problemaOriginal)

public static ProblemaTratamientosPD create (int cargaRestante, int k,
      ProblemaTratamientos problemaOriginal)
```


<<Java Class>>
ProblemaTratamientosPD
examenes.tratamientos
cargaRestante: int
k: int
create(ProblemaTratamientos):ProblemaTratamientosPD
create(int,int,ProblemaTratamientos):ProblemaTratamientosPD
getTipo():Tipo
esCasoBase():boolean
getSolucionParcialCasoBase():Sp<Integer>
getSolucionParcialPorAlternativa(Integer,List<Sp<Integer>>):Sp<Integer>
getSolucionParcial(List<Sp<Integer>>):Sp<Integer>
getAlternativas():List<Integer>
getNumeroSubProblemas(Integer):int
getSubProblema(Integer,int):ProblemaPD<Multiset<TratamientoMedico>,Integer>
getSolucionReconstruidaCasoBase(Sp<Integer>):Multiset<TratamientoMedico>
getSolucionReconstruidaCasoRecursivo(Sp<Integer>,List<Multiset<TratamientoMedico>>):Multiset<TratamientoMedico>
getObjetivoEstimado(Integer):Double
getObjetivo():Double
size():int
hashCode():int
equals(Object):boolean
toString():String
-problemaOriginal
0..1
<<Java Class>>
ProblemaTratamientos
examenes.tratamientos
cargaInicial: Integer
nTrat: Integer
maxVeces: Integer
create():ProblemaTratamientos
+listaTM
0..*
<<Java Class>>
TratamientoMedico
examenes.tratamientos
create(Integer,Integer):TratamientoMedico
getReduccionCarga():Integer
getDuracion():Integer
toString():String
hashCode():int
equals(Object):boolean
compareTo(TratamientoMedico):int
<<Java Interface>>
Multiset<E>
com.google.common.collect
size():int
count(Object):int
add(E,int):int
remove(Object,int):int
setCount(E,int):int
setCount(E,int,int):boolean
elementSet():Set<E>
entrySet():Set<Entry<E>>
forEachEntry(ObjIntConsumer<? super E>):void
equals(Object):boolean
hashCode():int
toString():String
iterator():Iterator<E>
contains(Object):boolean
containsAll(Collection<?>):boolean
add(E):boolean
remove(Object):boolean
removeAll(Collection<?>):boolean
retainAll(Collection<?>):boolean
forEach(Consumer<? super E>):void
spliterator():Spliterator<E>
lambda$forEach$1(Consumer,Entry):void
lambda$forEachEntry$0(ObjIntConsumer,Entry):void
<<Java Class>>
Sp<A1>
us.lsi.pd
alternativa: A1
propiedad: Double
estaEnLaSolucion: boolean
create(A2,Double):Sp<A2>
Sp(A1,Double)
toString():String
compareTo(Sp<A1>):int
-solucionesDeAlternativas
0..*

```
@Override
public List<Integer> getAlternativas() {
	List<Integer> ls;
	ls= IntStream
			.rangeClosed(0, problemaOriginal.maxVeces)
			.boxed()
			.collect(Collectors.toList());
	return ls;
}


/* Otra opción : */

@Override
public List<Integer> getAlternativas() {
	List<Integer> ls;


	int p= Math.min(problemaOriginal.maxVeces,
	cargaRestante % problemaOriginal.listaTM.get(k).getReduccionCarga() == 0 ?
	cargaRestante / problemaOriginal.listaTM.get(k).getReduccionCarga()
	: cargaRestante / problemaOriginal.listaTM.get(k).getReduccionCarga() + 1);


	ls= IntStream
			.rangeClosed(0, p)
			.boxed()
			.collect(Collectors.toList());



	return ls;

}


@Override
public ProblemaPD<Multiset<TratamientoMedico>,Integer>
   getSubProblema(Integer a, int np) {
	ProblemaPD<Multiset<TratamientoMedico>,Integer> p;
	p= create(cargaRestante-a*problemaOriginal.listaTM.get(k).getReduccionCarga(),
			k+1,
			problemaOriginal);
	return p;
}


@Override

public Sp<Integer>

     getSolucionParcialPorAlternativa (Integer a,
		List<Sp<Integer>> ls) {

	Sp<Integer> spHijo= ls.get(0);

	double d= problemaOriginal.listaTM.get(k).getDuracion();

	return Sp.create(a, a*d+spHijo.propiedad);

}
```