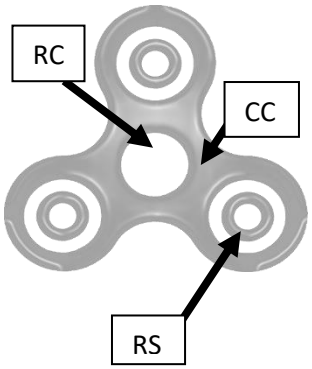
Un Fidget Spinner es un tipo de juguete antiestrés, hecho de plástico, acero u otros materiales y constituido por un eje central con dos, tres brazos, los cuales terminan en unos aros con rodamientos. La ingeniera Catherine Hettinger ha decidido crear el mejor spinner que pueda girar el máximo tiempo posible para un coste de fabricación determinado. Para ello, dispone de distintas piezas de distintos materiales que le aportan una mayor duración de giro al spinner. Nótese que la facilidad de giro del spinner viene dado por la suma de la facilidad de giro aportada por cada pieza.



A modo de resumen, las restricciones del problema serían:

El spinner ha de tener concretamente 1 cuerpo central (CC), 1 rodamiento principal (RC) y 3 rodamientos secundarios (RS).

Cada tipo de pieza puede ser de madera, plástico o metal.

Hay que maximizar el sumatorio de la facilidad de giro de las piezas respetando un coste máximo.

Catherine, está trabajando con los siguientes tipos de piezas, facilidades de giro asociados y costes por tipo de rodamiento.

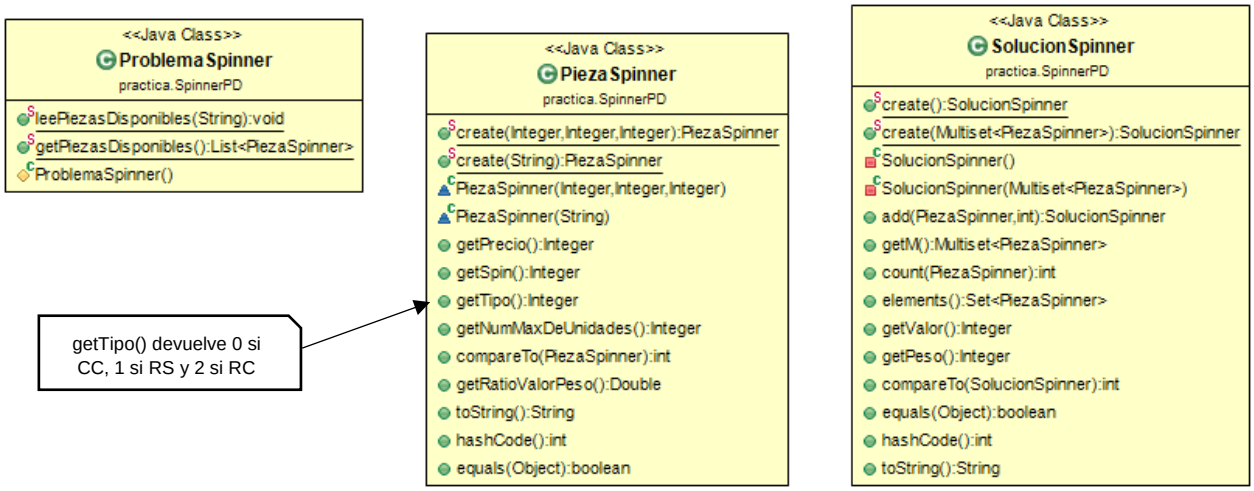
Material	Tipo de pieza	Facilidad de giro	Coste de la pieza
Metal	CC	10	10 u.m
Plástico	CC	5	4 u.m
Metal	RS	4	6 u.m
Madera	RS	3	1 u.m
Plástico	RS	2	1 u.m
Metal	RC	20	5 u.m
Plastico	RC	15	3 u.m

Un posible spinner (no optimo) sería: CC= metal; RS1=Metal; RS2=Plástico; RS3=Plástico; RC=Metal con un giro total de 38 y un coste de 23 u.m.

Se pide ayudar a Catherine a desarrollar el spinner perfecto para un coste total de 30 unidades monetarias (u.m) usando el esquema de programación dinámica.

1) Completar la ficha adjunta del problema (//TODO)
2) Implemente los métodos:
   a) getAlternativas
   b) getSolucionCasoBase
   c) esCasoBase
   d) getSubProblema
   e) combinaSolucionesParciales

NOTA: Preste atención a que los contadores de cada tipo de pieza no han de superar su cantidad máxima.



PD	
Problema Spinner	
Técnica: programación Dinámica	
Propiedades Compartidas	Piezas: List<PiezaSpinner>// lista de piezas MaxNumCC: int // 1 CC MaxNumRS: int // 3 RS MaxNumRC: int // 1 RC MaxCosteTotal:int
Propiedades individuales	Index: int // [0,…,|Piezas|] costeAcumulado: double contadorCC:int // para asegurarnos de comprar 1 CC contadorRS:int // para asegurarnos de comprar 3 RS contadorRC:int // para asegurarnos de comprar 1 RC
Propiedades derivadas	
Solución: Multiset<PiezaSpinner> // Piezas de cada tipo	
Alternativas://TODO	
Instanciación: //TODO	
psg(p)= //TODO	

Soluciones

1) Completar la ficha adjunta del problema

PD	
Problema Spinner	
Técnica: programación Dinámica	
Propiedades Compartidas	Piezas: List<PiezaSpinner>// lista de piezas MaxNumCC: int // 1 CC MaxNumRS: int // 3 RS MaxNumRC: int // 1 RC MaxCosteTotal:int

Propiedades individuales	index (i): int // [0,…,|piezas|]
	costeAcumulado(c): double
	contadorCC(cc):int // para asegurarnos de comprar 1 CC
	contadorRS(rs):int // para asegurarnos de comprar 3 RS
	contadorRC(rc):int // para asegurarnos de comprar 1 RC

Propiedades derivadas	
Solución: Multiset<PiezaSpinner> // Piezas de cada tipo	
Alternativas: $A_{i,c,cc,rs,rc} = \{a{:}\,k..0\}, k = \min(\frac{MaxCosteTotal-c}{p(i)}, m_i))$ / $cc + k \leq maxNumCC \; si \; t(i) == 0$ $rs + k \leq maxNumRS \; si \; t(i) == 1$ $rc + k \leq maxNumRC \; si \; t(i) == 2$ p(i)=piezas.get(i).getPrecio() m(i)=piezas.get(i).getMaxUnidades() t(i)=piezas.get(i).getTipo()	
Instanciación: ps(piezas, maxNumCC, MaxNumRS, MaxNumRC, MaxCosteTotal) =psg(0,0,0,0,0)	

psg(p)=

(null,0), i=|piezas|; cc=MaxCC; rs=MaxRS; rc=MaxRC

$\perp$ i=|piezas|; cc!=MaxCC|| rs!=MaxRS|| rc!=MaxRC

$sA_{a \in A_{i,c,cc,rs,rc}} c\big(p, a, psg(sp)\big)$, i<|piezas|

p = (i,ca,cc,rs,rc)

sp=(i+1,c+a*p(i),cc+a,rs,rc) si t(i)==0

sp=(i+1,c+a*p(i),cc,rs+a,rc) si t(i)==1

sp=(i+1,c+a*p(i),cc,rs,rc+a) si t(i)==2

2) Implemente los métodos:

a. getAlternativas

```
@Override
public List<Integer> getAlternativas() {
	List<Integer> ls = IntStream.rangeClosed(0,
piezas.get(this.index).getNumMaxDeUnidades() )
				.filter(x->this.constraints(x))
				.boxed()
				.collect(Collectors.toList());
	Collections.reverse(ls);
	return ls;
}
private Boolean constraints(Integer x) {
	boolean res=x*piezas.get(index).getPrecio() <= dineroRestante;
	res=res&& verificaCompra(piezas.get(index).getTipo(),x);
	return res;
}
	//Verifica que no me paso de los limites de cada pieza
private boolean verificaCompra(Integer tipo, Integer a2) {
	boolean res=false;
	if(tipo==0){
		res=contadorCC+a2<=MaxNumCC;
	}else if(tipo==1){
		res=contadorRS+a2<=MaxNumRS;
	}else if(tipo==2){
		res=contadorRC+a2<=MaxNumRC;
	}
	return res;
}
```

b. getSolucionCasoBase

```
@Override
public Sp<Integer> getSolucionCasoBase() {

	//si no he comprado todas las piezas
	if (ProblemaSpinnerPD.MaxNumCC-contadorCC !=0 ||
ProblemaSpinnerPD.MaxNumRC-contadorRC!=0 ||ProblemaSpinnerPD.MaxNumRS-
contadorRS !=0){
	return null;
	}
return Sp.create(null, 0.0);
}
```

c. esCasobase

```
@Override
public boolean esCasoBase() {
	return index == ProblemaSpinnerPD.piezas.size();
}
```

d. getSubProblema

```
@Override
public ProblemaPD<Multiset<PiezaSpinner>, Integer> getSubProblema(Integer
a, int np) {
	Preconditions.checkArgument(np==0);
	int coste=a*piezas.get(index).getPrecio();
	Integer dineroRestante = this.dineroRestante-coste;
	Double spinAcumulado =
this.spinAcumulado+a*piezas.get(index).getSpin();
	int contadorCCtmp=contadorCC;
	int contadorRCtmp=contadorRC;
	int contadorRStmp=contadorRS;

	if(piezas.get(index).getTipo() ==0){
		contadorCCtmp+=a;
	}else if(piezas.get(index).getTipo() ==1){
		contadorRStmp+=a;
	}else if(piezas.get(index).getTipo() ==2){
		contadorRCtmp+=a;
	}
	return
ProblemaSpinnerPD.create(index+1,dineroRestante,spinAcumulado,contadorCCtm
p,contadorRStmp,contadorRCtmp);
	}
```

e. combinaSolucionesParciales

```
@Override
	public Sp<Integer> combinaSolucionesParciales(Integer a,
List<Sp<Integer>> ls) {
		Sp<Integer> r = ls.get(0);
		Double valor = a*piezas.get(index).getSpin()+r.propiedad;
		return Sp.create(a, valor);
	}
```