Asignatura	**Análisis y Diseño de Algoritmos**
Convocatoria	2ª Alternativa
Curso	2010-2011

Enunciado

Dado un presupuesto ***PT*** de un proyecto para la contratación de personal y un sistema de categorías profesionales ***Categorias***, se pide invertir dicho presupuesto contratando el menor número de trabajadores posibles, ya que cuanto menor sea el número de trabajadores, más fácil será coordinar las tareas del proyecto.

El sistema de categorías profesionales está definido por un conjunto ***c*** de categorías diferentes. Cada categoría se representará por c_i, donde $i \in [0, c.size)$, y s_i como el sueldo que obtienen los trabajadores que pertenecen a dicha categoría. De esta manera, podemos representar a **una categoría profesional** de manera compacta como $c_i^{s,t}$, representando t_i el **número de trabajadores** de la **categoría** i cuyo **sueldo** es s_i.

Representaremos una **solución** del problema de contratación de personal como los trabajadores necesarios para cubrir el presupuesto total ***PT***. Una manera compacta de definirla es como sigue:

- $sol^{c,s,t}$: la solución con el multiconjunto de categorías utilizadas c cuyo cuyo presupuesto final (o lo que es lo mismo, la suma de los sueldos de todos los trabajadores que van a ser contratados) es s y el número total de trabajadores utilizados es t. Donde $\emptyset$ se corresponde a la solución con el multiconjunto vacío y $\perp$ que no hay solución.

Las **alternativas** para la contratación de personal son el número de trabajadores que podemos usar de la categoría c_i si el presupuesto del que se dispone es p.

$$A = \{0,1,2,\ldots,k\} \text{ con } k = p/s_i$$

El objetivo del problema es encontrar una solución cuyo presupuesto final sea el presupuesto total inicial ***PT*** y tenga el menor número de trabajadores posibles.

Los detalles se muestran en la Ficha siguiente:

Ficha Contratación de Personal	
Técnica*: Programación Dinámica*	
Representación: *Cada problema un objeto*	
Tamaño: *J*	
Propiedades Compartidas:	***Categorias:*** *List<Categoria>.* ***PT*** *: entero no negativo, presupuesto total*
Propiedades Individuales:	***p****: entero no negativo, presupuesto disponible.* ***J****: entero en [0,Categorias.Size)*
Solución: *SolucionCategoria*	
Objetivo: *Encontrar* $sol^{c,s,t}$ *tal que el número de trabajadores t tenga el menor valor posible.*	
Alternativas: $A_{p,j} = \{0,1,\ldots,k\}$, $k = p/s_j$	
pcp(c,p)=cp(PT,Categorias.Size-1) $cp(p,j) = \begin{cases} sol^{\{c_j\},t\cdot s_j,t}, & t = \frac{p}{s_j}, p = t\cdot s_j, j = 0 \\ \perp, & t = \frac{p}{s_j}, p \neq t\cdot s_j, j = 0 \\ cO_{a_i \in A_{p,j}}\left(cA\left(a_i, cp(p - a_i\, s_j, j-1)\right)\right), & p > 0, j > 0 \end{cases}$	
$cA(a_i,sol)$*: Añade* a_i *trabajadores de la categoría* c_i *a la solución sol*	
$cO_{ai \in Ap,j}(sol_{ai}^{c,s,t})$*: Elige la solución con el menor valor de t*	
Complejidad:	*O(p·Categorias.Size)*

Implemente de la clase *ProblemaCategoriaPDImpl* los métodos:

a) ***public boolean esCasoBase( ).***

b) ***public SolucionCategoria getSolucionCasoBase( )***.

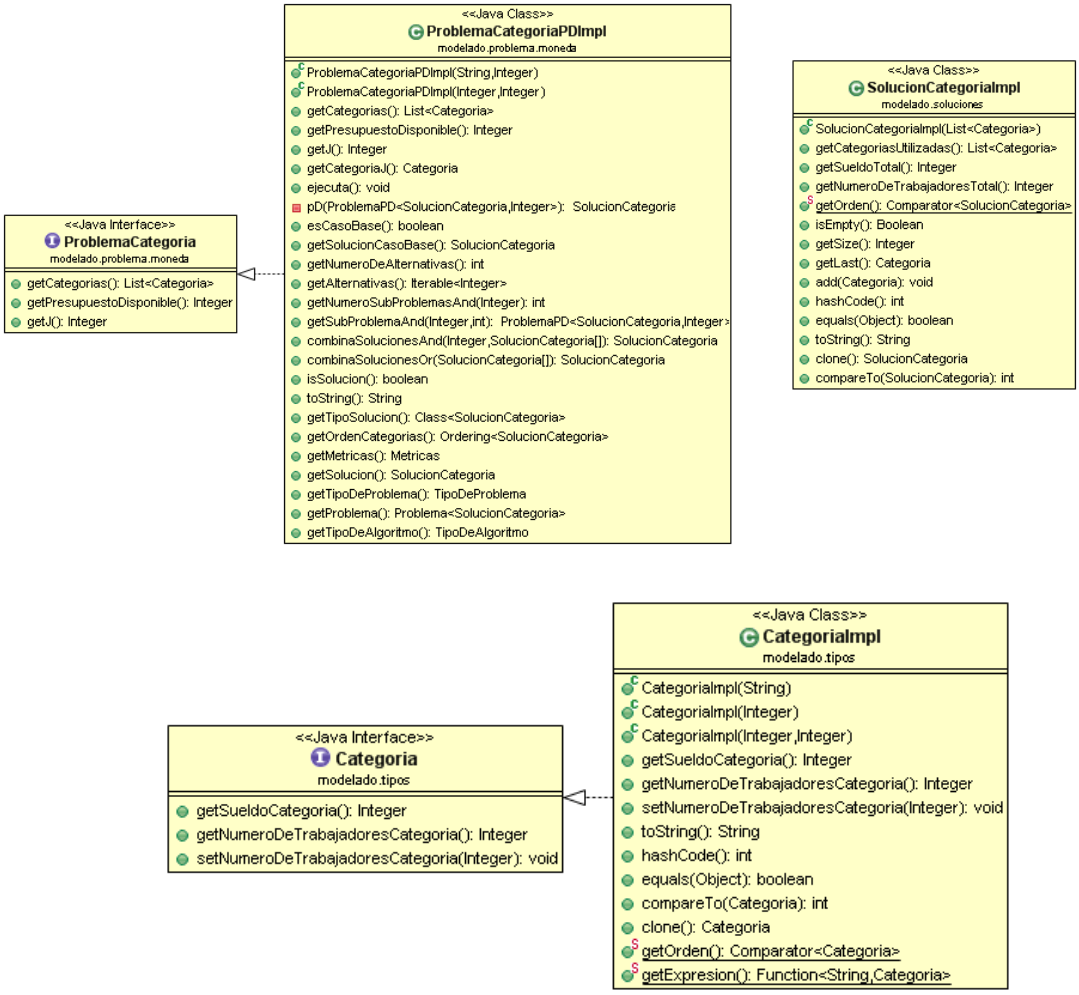
c) ***public int getNumeroDeAlternativas( ).***

d) ***public int getNumeroSubproblemasAnd(Integer a)***.

e) ***public ProblemaPD<SolucionCategoria, Integer> getSubProblemaAnd(Integer a,int i)***.

f) ***public SolucionCategoria combinaSolucionesAnd(Integer a,SolucionCategoria... ls)*** .

Nota: utilice los diagramas UML que se muestran a continuación para realizar las implementaciones.



Tiempo estimado: 60 min. Puntuación: 5 puntos

Solución

Implemente de la clase *ProblemaTrabajadoresPDImpl* los métodos:

g) ***public boolean esCasoBase().*** **(1 punto)**

```
public boolean esCasoBase() {
   return getJ() = = 0;
}
```

h) ***SolucionCategoria getSolucionCasoBase()***. **(2 punto)**

```
public SolucionCategoria getSolucionCasoBase() {
      SolucionCategoria s = null;

      Integer trabajadoresAUtilizar = getPresupuestoDisponible()
                        / getCategoriaJ().getSueldoCategoria();

      Integer presupuestoAUtilizar = getCategoriaJ().
             getSueldoCategoria() * trabajadoresAUtilizar;

      if (getPresupuestoDisponible().equals(
              presupuestoAUtilizar)) {

           getCategoriaJ().setNumeroDeTrabajadoresCategoria(
                              trabajadoresAUtilizar);

            Categoria copiaCategoriaJ = getCategoriaJ().clone();

            s = new SolucionCategoriaImpl(
                      Lists.newArrayList(copiaCategoriaJ));
            }

            return s;
}
```

i) ***int getNumeroDeAlternativas().*****(2 punto)**

```
 public int getNumeroDeAlternativas() {
         Integer posiblesTrabajadores = getPresupuestoDisponible()
                        / getCategoriaJ().getSueldoCategoria();

         return posiblesTrabajadores + 1;
   }
```

j) ***int getNumeroSubproblemasAnd(Integer a)***. **(1 punto)**

```
   public int getNumeroSubProblemasAnd(Integer a) {
         return 1;
   }
```

k) ***ProblemaPD<SolucionCategoria,Integer> getSubProblemaAnd(Integer a,int i)*** **(2 punto)**

```
 public ProblemaPD<SolucionCategoria, Integer>
        getSubProblemaAnd(Integer a, int i) {

     int nuevoPresupuesto = getPresupuestoDisponible()
                    - (a * getCategoriaJ().getSueldoCategoria());

     return new ProblemaCategoriaPDImpl(nuevoPresupuesto,getJ()-1);
 }
```

l) ***SolucionCategoria combinaSolucionesAnd(Integer a,SolucionCategoria... ls)*** . **(2 punto)**

```
public SolucionCategoria combinaSolucionesAnd(Integer a,
            SolucionCategoria... ls) {

      SolucionCategoria s = null;
      if (ls[0] != null) {

            s = ls[0].clone();

            Categoria m = getCategorias().get(getJ());

            m.setNumeroDeTrabajadoresCategoria(a);

            Categoria copia = m.clone();

            s.add(copia);

      }
      return s;
}
```