Ejercicio 3. Backtracking

En el puerto de Algeciras hay un conjunto de contenedores dispuestos para ser embarcados. Se nos pide hacer uso de la técnica de backtracking para seleccionar qué contenedores deberán ser embarcados en el siguiente carguero y en qué orden embarcarlos teniendo en cuenta las siguientes restricciones:

1) Los contenedores seleccionados no pueden superar la capacidad del carguero.
2) El tiempo de embarque de los contenedores seleccionados no puede superar el tiempo disponible para tal fin.
3) La diferencia entre el número de contenedores embarcados de tipo 1 y tipo 2 será menor o igual a 3.
4) El orden en que se seleccionen los contenedores influye en el beneficio obtenido. Dicho beneficio se calculará utilizando la siguiente formula:

$$\text{beneficio\_contenedor} = \text{precio\_ofertado} * \left(1 + \frac{1}{1 + \text{orden\_en\_que\_se\_embarca}}\right)$$

Se pide:

a) Completar la siguiente ficha:

Embarque de Contenedores	
Técnica: Vuelta atrás	
Propiedades Compartidas	contenedoresDisponibles: List<Contenedor> capacidadDelCargero: Integer tiempoParaEmbarcar: Integer
Propiedades del Estado	ordenDeEmbarque: List<Integer> capacidadRestante: Integer tiempoRestante: Integer tipo1, tipo2: Integer beneficio: Double
Solución: // **TODO**	
Objetivo: Encontrar una solución que obtenga el máximo beneficio sin superar la capacidad del carguero y el tiempo para embarcar, de forma qué la diferencia entre el número de contenedores embarcados de tipo 1 y tipo 2 sea menor o igual a 3.	
Definiciones: c(k)=contenedoresDisponibles.get(k) csize()=contenedoresDisponbles.size() oesize()=ordenDeEmbarque.size()	
Alternativas: $A = \left\{ k \in [0..csize()-1] \middle/ \begin{array}{l} k \notin ordenDeEmbarque \;\&\; c(k).getPeso() \le capacidadRestante \;\& \\ c(k).getTiempoDeCarga() \le tiempoRestante \end{array} \right\}$	
Estado Inicial: ([],capacidadDelCargero,tiempoParaEmbarcar,0,0,0)	
Estado Final: // **TODO**	
Avanza(k): // **TODO**	

b) Implementar los siguientes métodos de la clase EstadoContenedores:

b.1) **public** List<Integer> getAlternativas()
b.2) **public void** avanza(Integer contenedor)
b.4) **public boolean** isFinal()
b.5) **public** List<Contenedor> getSolucion()
b.6) **public** Double getObjetivo()

Ejemplo:

Contenedores Disponibles:

ID	Peso	Tiempo De Embarque	Tipo	Precio Oferta
0	10	30	1	2
1	10	15	1	3
2	25	20	2	4
3	30	10	1	4
4	15	5	1	5
5	20	10	2	6
6	25	10	1	5

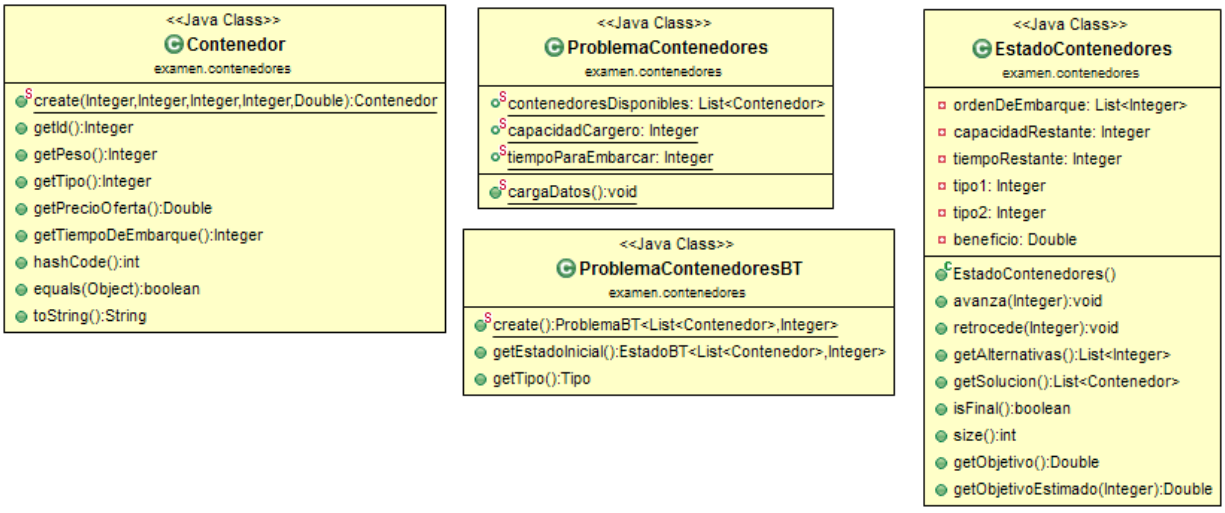
Capacidad del carguero: 70
Tiempo para embarcar contenedores: 100

Orden de carga:

ID	Peso	Tiempo	Tipo	Precio	Beneficio
5	20	10	2	6	12
1	10	15	1	3	4.5
4	15	5	1	5	6,6
6	25	10	1	5	6,25

Beneficio orden de carga: 29.35
Peso orden de carga: 70
Tiempo orden de carga: 40
Diferencia entre tipos de contenedores: 2

Diagrama de Clases:



SOLUCIÓN

a)

Embarque de Contenedores	
Técnica: Vuelta atrás	
Propiedades Compartidas	contenedoresDisponibles: List<Contenedor> capacidadDelCargero: Integer tiempoParaEmbarcar: Integer
Propiedades del Estado	ordenDeEmbarque: List<Integer> capacidadRestante: Integer tiempoRestante: Integer tipo1: Integer tipo2: Integer beneficio: Double
Solución: **List<Contenedor>**	
Objetivo: Encontrar una solución que obtenga el máximo beneficio sin superar la capacidad del carguero y el tiempo para embarcar, de forma que la diferencia entre el número de contenedores de tipo 1 y tipo 2 sea menor o igual a 3.	
Definiciones: c(k)=contenedoresDisponibles.get(k) csize()=contenedoresDisponbles.size() oesize()=ordenDeEmbarque.size()	
Alternativas: $A=\left\{k\in[0..csize()-1] \middle/ \begin{array}{l} k\notin ordenDeEmparque \,\&\, c(k).getPeso() \le capacidadDeEmbarque \,\& \\ c(k).getTiempoDeCarga() \le tiempoRestante \end{array}\right\}$	
Estado Inicial: ([],capacidadDelCargero,tiempoParaEmbarcar,0,0,0)	
Estado Final: $A.size() == 0$	
Avanza(k): **(ordenDeEmbarque+[k], capacidadRestante-c(k).getPeso(), / tiempoRestante-c(k).getTiempoDeCarga(), tipo1+(c(k).getTipo()=1?1:0), tipo2+(c(k).getTipo()=2?1:0), beneficio + c(k).getPrecioOferta()*** $\left(1+\frac{1}{1+oesize()}\right)$ **)**	

b.1)

```
public List<Integer> getAlternativas() {
   List<Integer>la =
          IntStream.range(0,ProblemaContenedores.contenedoresDisponibles.size())
    .boxed()
    .filter(pos -> ordenDeEmbarque.indexOf(pos)<0)
    .filter(pos -> capacidadRestante>=
ProblemaContenedores.contenedoresDisponibles.get(pos).getPeso())
    .filter(pos -> tiempoRestante>=
ProblemaContenedores.contenedoresDisponibles.get(pos).getTiempoDeEmbarque())
    .collect(Collectors.toList());
    return la;
}
```

b.2)

```
public void avanza(Integer contenedor) {

   Contenedor c = ProblemaContenedores.contenedoresDisponibles.get(contenedor);
   capacidadRestante-=c.getPeso();
   tiempoRestante-=c.getTiempoDeEmbarque();
   if (c.getTipo()==1) tipo1++;
   if (c.getTipo()==2) tipo2++;
   beneficio+= c.getPrecioOferta() * (1+(1/(1+ordenDeEmbarque.size())));
   ordenDeEmbarque.add(contenedor);

}
```

b.3)

```
public boolean isFinal() {
   return getAlternativas().isEmpty();
}
```

b.4)

```
public List<Contenedor> getSolucion() {
    List<Contenedor> sol = null;
    if (Math.abs(tipo1-tipo2)<=3) {
       sol = this.ordenDeEmbarque.stream()
         .map(pos-> ProblemaContenedores.contenedoresDisponibles.get(pos))
        .collect(Collectors.toList());
      }
    return sol;
}
```

b.5)

```
public Double getObjetivo() {
    return beneficio;
}
```