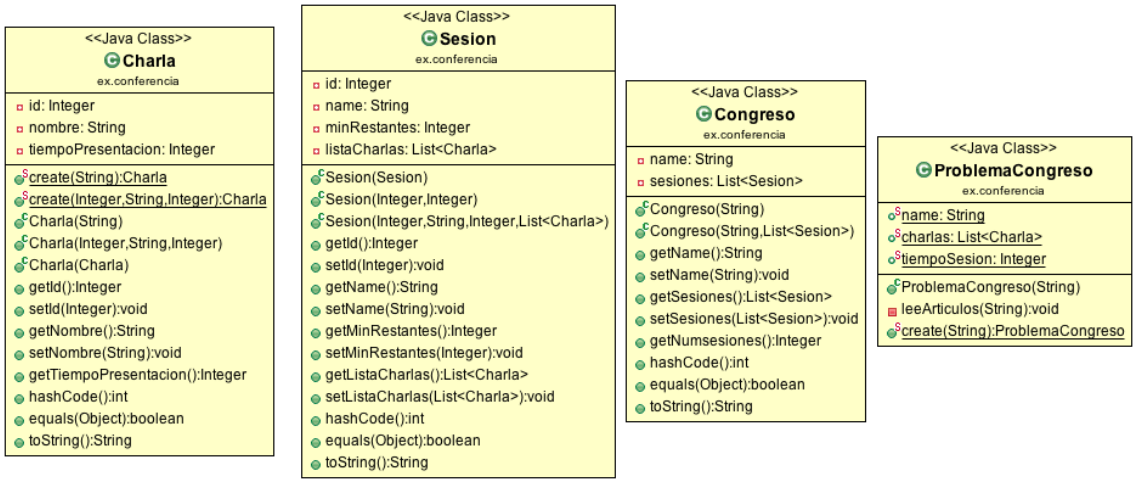
PROBLEMA 5: Backtracking

La empresa *OrgTuConferencia* está organizando el congreso ADDA 2017. Se dispone de una lista de *m* Charlas aceptadas que hay que agrupar en Sesiones. Cada charla tiene asignado un tiempo de presentación (en minutos), y cada sesión consta de 60 min. El objetivo de *OrgTuConferencia* es buscar la asignación de las distintas charlas a sesiones de forma que el número de sesiones sea mínimo.

Tenga en cuenta que todas las charlas tienen que estar asignadas a alguna sesión. A la hora de elegir la sesión más adecuada para una charla, se podrá elegir entre abrir una nueva sesión o asignarla a alguna de las sesiones ya abiertas (siempre que quepa).

Dispone de las siguientes clases:



SE PIDE

1. Rellenar la ficha adjunta. Se debe entregar la hoja de la ficha rellena.
2. Implemente los siguientes métodos de la clase *EstadoCongreso* que implementa *EstadoBT<Congreso, Sesion>*:
   a. *Boolean isFinal()*
   b. *void avanza(Sesion s)*
   c. *void retrocede(Sesion s)*
   d. *List<Sesion> getAlternativas()*
   e. *Congreso getSolucion()*
   f. *Double getObjetivo()*

EJEMPLO:

Problema:

```
Congreso: name=ADDA2017, tiempoSesion =60
------
Articulo [id=1, nombre="Art1", 15]
Articulo [id=2, nombre="Art2", 20]
Articulo [id=3, nombre="Art3", 15]
Articulo [id=4, nombre="Art4", 20]
Articulo [id=5, nombre="Art5", 15]
Articulo [id=6, nombre="Art6", 20]
Articulo [id=7, nombre="Art7", 15]
```

Solución:

Congreso [name=ADDA2017, numsesiones=2, sesiones= [Sesion [id=0, name=Sesion0, minRestantes=0, listaArticulos=["Art1", "Art3", "Art5", "Art7]], Sesion [id=1, name=Sesion1, minRestantes=0, listaArticulos=["Art2","Art4","Art6"]]]

Apellidos, Nombre:
Titulación/Grupo:

Problema *OrgTuConferencia*	
Técnica: BT	
Tamaño:	
Prop. Compartidas:	
Prop. del Estado:	
Solución:	
E. Inicial:	
E. Final:	
Objetivo:	
Alternativas:	
Avanza(a):	
Retrocede(a):	

Solución:

1. Rellenar la ficha adjunta. Se debe entregar la hoja de la ficha rellena.

Problema *OrgTuConferencia*	
Técnica: BT	
Tamaño: *indiceCharlaActual*	
Prop. Compartidas:	*charlas: List<Charla>* *tiempoSesion: Integer*
Prop. del Estado:	*indiceCharlaActual: Integer[0, charlas.size())* *sesiones: List<Sesion>*
Solución:	*Congreso*
E. Inicial:	*indiceCharlaActual = 0* *sesiones = []*
E. Final:	*indiceCharlaActual == charlas.size()*
Objetivo:	*Minimizar el número de sesiones*
Alternativas:	*A = nuevaSesion + ∀s ∈ sesiones || s.tiempoRestante ≤charlas.get(indiceCharlaActual).getTiempoPresentacion()*
Avanza(a):	*Si !sesiones.contains(a) sesiones.add(a)* *a.getCharlas().add(charlas.get(índiceCharlaActual))* *a.setMinRestantes(a.getMinRestantes() -* *charlas.get(índiceCharlaActual).getTiempoPresentacion())* *indiceCharlaActual ++*
Retrocede(a):	*indiceCharlaActual--* *a.getCharlas().remove(charlas.get(indiceCharlaActual))* *a.setMinRestantes(a.getMinRestantes() +* *charlas.get(índiceCharlaActual).getTiempoPresentacion())* *Si !sesiones.contains(a) sesiones.remove(a)*

2. Implemente los siguientes métodos de la clase *EstadoCongreso* que implementa *EstadoBT<Congreso, Sesion>*:
   a. *Boolean isFinal()*

```
public boolean isFinal() {
      return indiceCharlaActual == ProblemaCongreso.charlas.size();
}
```

   b. *void avanza(Sesion s)*

```
public void avanza(Sesion s) {
      if (!sesiones.contains(s)) {
            sesiones.add(s);
      }
      Charla charla =
            ProblemaCongreso.charlas.get(indiceCharlaActual);
      s.getListaCharlas().add(charla);
      s.setMinRestantes(s.getMinRestantes() -
            charla.getTiempoPresentacion());
      this.indiceCharlaActual++;
}
```

   c. *void retrocede(Sesion s)*

```
public void retrocede(Sesion s) {
      indiceCharlaActual--;
      Charla charla =
            ProblemaCongreso.charlas.get(indiceCharlaActual);
      s.getListaCharlas().remove(charla);
      s.setMinRestantes(s.getMinRestantes() +
            charla.getTiempoPresentacion());
      if (s.getListaCharlas().isEmpty()) {
            this.sesiones.remove(s);
      }
}
```

   d. *List<Sesion> getAlternativas()*

```
public List<Sesion> getAlternativas() {
      Charla charla =
            ProblemaCongreso.charlas.get(indiceCharlaActual);
      List<Sesion> alternativas = new LinkedList<Sesion>();
      Sesion sesionnueva = new Sesion(this.sesiones.size(),
            ProblemaCongreso.tiempoSesion);
      alternativas.add(sesionnueva);
      sesiones.stream()
         .filter(s ->
            s.getMinRestantes() >= charla.getTiempoPresentacion())
         .forEach(s -> alternativas.add(s));
      return alternativas;
}
```

e. Congreso getSolucion()

```
public Congreso getSolucion() {
      return new Congreso(ProblemaCongreso.name, this.sesiones);
}
```

f. Double getObjetivo()

```
public Double getObjetivo() {
      return this.sesiones.size() * 1.0;
}
```