Ejercicio 5: Backtracking

El servicio de organización docente de un instituto de formación necesita implementar un programa que calcule los horarios de clases maximizando las preferencias de los profesores y ajustándolos a las siguientes restricciones:

1) Cada clase será impartida por un profesor.
2) El profesor `i` puede impartir como máximo N_i clases.
3) La clase `i` será incompatible con el profesor `j` si el profesor `j` no puede impartir la clase `i`.

Para la resolución del problema dispone de la clase `ProblemaHorarios` que tiene las siguientes propiedades y funciones:

- `List<String> profesoresDisponibles`: Profesores disponibles para dar clases.
- `List<String> clasesLibres`: Clases que hay que impartir.
- `int getMaxClasesXProfesor(int p)`: Devuelve el número máximo de clases que puede impartir el profesor `p`.
- `int getPreferenciaProfesor(int p, int c)`: Devuelve la preferencia del profesor `p` por la clase `c`.
- `boolean getClaseIncompatible(int p, int c)`: Devuelve `true` si el profesor `p` *no* puede impartir la clase `c`, `false` en caso contrario.

También dispone de la clase `Asignación` que dispone de las siguientes funciones:

- `int add(int c, int p)`: Asigna el profesor `p` a la clase `c`.
- `int getPreferenciaTotal()`: Devuelve la preferencia total de la asignación.

Problema de los Horarios	
Tipos	S- Asignación A- Integer
Propiedades Compartidas	p_i, List<String>: Profesores disponibles c_i, List<String>: Clases libres $PREF_{ij}$: Preferencias del profesor i por la clase j N_i: Número máximo de clases que puede impartir el profesor i INC_{ij}: true si el profesor i no puede impartir la clase j, false en otro caso
Propiedades del Estado	index,Integer: Siguiente clase libre. gs,List<*Integer*>: Lista con la asignación de clases / profesores
Inicial: // TODO	
Caso Final: // TODO	
Alternativas: $A(index) = \{i \in [0, \lvert p \rvert) \ / \ H(i) < N(i) \wedge \neg INC(i, index)\} \quad H(i) = \sum_{j=0}^{index} (gs(j) = i \,?\, 1 : 0)$	
Avanza(a): // TODO **Retrocede(a):** // TODO **Solución (Para calcular *el* estado final):** // TODO	
Objetivo: //TODO	

SE PIDE:

1) Rellenar los campos TODO de la ficha anterior.

2) Implementar los siguientes métodos de la clase `EstadoHorariosBT`:

a) `void avanza(Integer a)`
b) `void retrocede(Integer a)`
c) `List<Integer> getAlternativas()`
d) `Bolean esCasoBase()`
e) `Asignacion getSolucion()`

3) Implementar el método `double getObjetivo()`la clase `Asignacion`:

EJEMPLO:

Profesores = [ProfesorA, ProfesorB, ProfesorC]
Clases = [Clase1, Clase2, Clase3, Clase4, Clase5]

Max clases por profesor
==========

ProfesorA	1
ProfesorB	2
ProfesorC	3

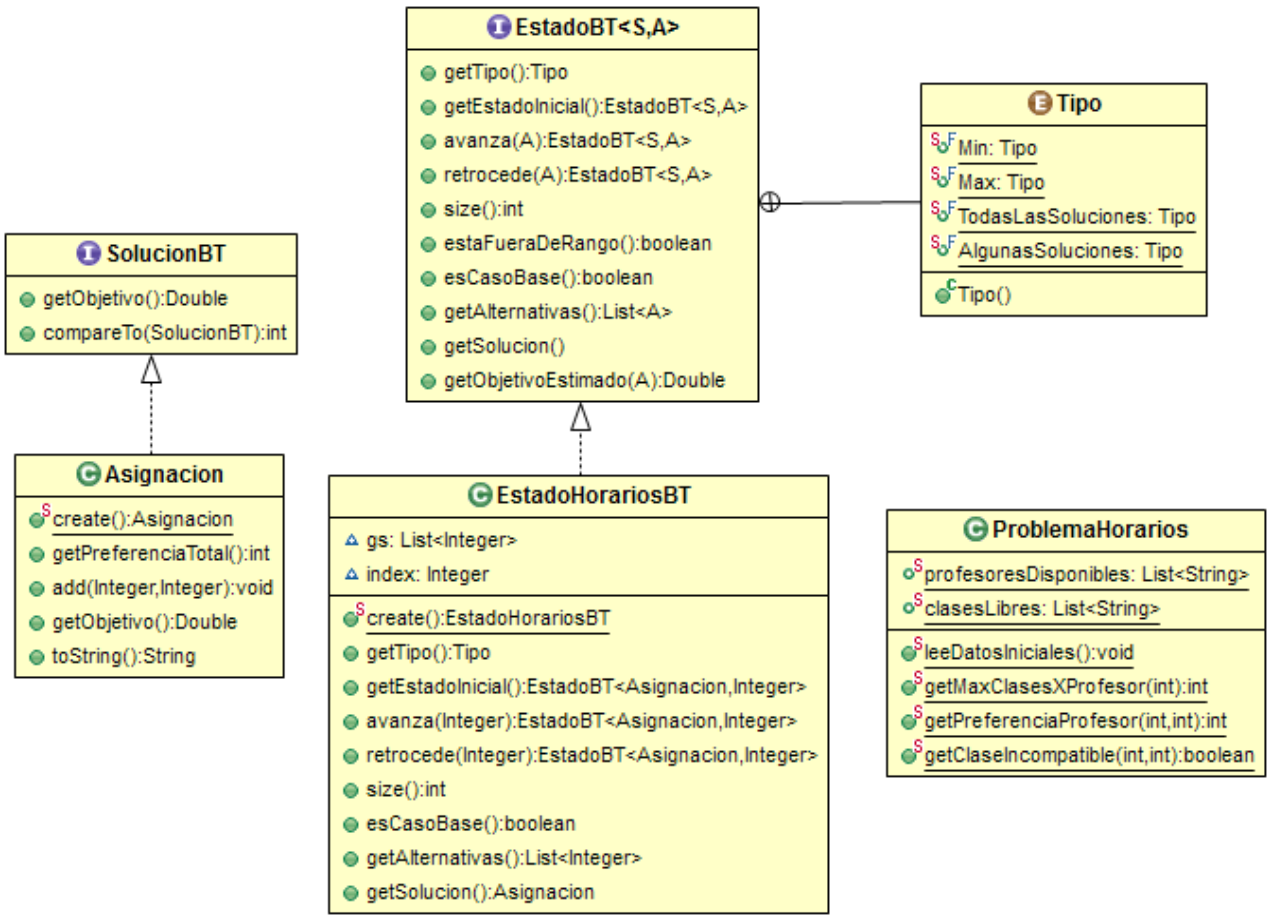
Preferencia por profesor
==========

	Clase1	Clase2	Clase3	Clase4	Clase5
ProfesorA	2	4	0	0	2
ProfesorB	2	0	3	0	3
ProfesorC	0	0	0	3	0

Incompatibilidades por profesor
==========

	Clase1	Clase2	Clase3	Clase4	Clase5
ProfesorA	true	false	false	true	false
ProfesorB	false	true	false	false	true
ProfesorC	false	false	false	false	false

Mejor solución = {Clase1=ProfesorB, Clase2=ProfesorA, Clase3=ProfesorB, Clase4=ProfesorC, Clase5=ProfesorC}
Preferencia total de la mejor solución = 12



SOLUCIÓN

Problema del Servicio de Impresión	
Tipos	`S- Asignación` `A- Integer`
Propiedades Compartidas	p_i, `List<String>: Profesores disponibles` c_i, `List<String>: Clases libres` $PREF_{ij}$`: Preferencias del profesor i por la clase j` N_i`: Número máximo de clases que puede impartir el profesor i` INC_{ij}`: true si el profesor i no puede impartir la clase j, false en otro caso`
Propiedades del Estado	`index,Integer`: Siguiente clase libre. `gs,List<Integer>`: Lista con la asignación de clases / profesores
Solución: `Asignación`	
Objetivo: Encontrar una asignación de clase/profesor que maximice la preferencia y cumpla las restricciones	
Inicial: (0, [])	
Caso Final: index == |c|	
Alternativas: $A(index) = \{i \in [0, \lvert p \rvert) / H(i) < N(i) \wedge \neg INC(i, index)\}$ $H(i) = \sum_{j=0}^{index} (gs(j) = i \,?\, 1 : 0)$	
Avanza(a): (index, gs) -> (index+1,gs[index]=a) **Retrocede(a):** (index, gs) -> (index-1,gs[index].remove(|gs|-1)) **Solución (Para calcular *el* estado final):** asig = Asignacion.create() range(0,gs.size()).forEach(i->asig.add(c_i,gs[i]))	
Objetivo: maximizar las preferencias de los profesores	

```
	public EstadoBT<Asignacion, Integer> avanza(Integer prof) {
		gs.add(prof);
		index++;
		return this;
	}

	public EstadoBT<Asignacion, Integer> retrocede(Integer a) {
		index--;
		gs.remove(gs.size()-1);
		return this;
	}

	public boolean esCasoBase() {
		return (ProblemaHorarios.clasesLibres.size() - index == 0)?true:false;
	}

	public List<Integer> getAlternativas() {
		return IntStream.range(0, ProblemaHorarios.profesoresDisponibles.size())
.filter(prof->getClasesXProfesor(prof)<ProblemaHorarios.getMaxClasesXProfesor(prof))
.filter(prof->!ProblemaHorarios.getClaseIncompatible(prof, index)).boxed()
.collect(Collectors.toList());
	}

	private int getClasesXProfesor(int prof) {
		return (int)gs.stream().filter(x->x==prof).count();
	}

	public Asignacion getSolucion() {
		Asignacion asig = Asignacion.create();
		IntStream.range(0,gs.size()).forEach(i->asig.add(i,gs.get(i)));
		return asig;
	}

	public Double getObjetivo() {
		return (double)getPreferenciaTotal();
	}
```