Problema 3: Algoritmos Genéticos

Se desea construir un robot y para ello es necesario comprar un conjunto de piezas (`piezasNecesarias`). Las piezas no se venden de forma individual, sino como parte de un paquete de piezas de las cuales algunas pueden ser necesarias para la construcción del robot y otras no. Los paquetes que podemos comprar están almacenados en una lista (`paquetesDisponibles`) y solo se podrá comprar cada paquete una vez. Tenga en cuenta que al comprar dos o más paquetes distintos puede ser que una pieza se repita aunque solo nos hacía falta una vez.

Los objetos `PaqueteDePiezas` tendrán las siguientes propiedades:

- id: String **// Identificador del conjunto de piezas.**
- coste: Integer **// Coste del conjunto de piezas.**
- piezas: Set<Pieza> **// Conjunto con las piezas que forman parte del paquete.**

Los objetos `Pieza` tendrán las siguientes propiedades:

- id: Integer **// Identificación de la pieza.**
- nombre: String **// Nombre de la pieza.**

A partir de la información disponible, implemente un Algoritmo Genético que calcule los paquetes de piezas que debemos comprar para conseguir todas las piezas que necesitamos minimizando el coste total.

SE PIDE:

1. Completar los campos **TODO** de la siguiente ficha:

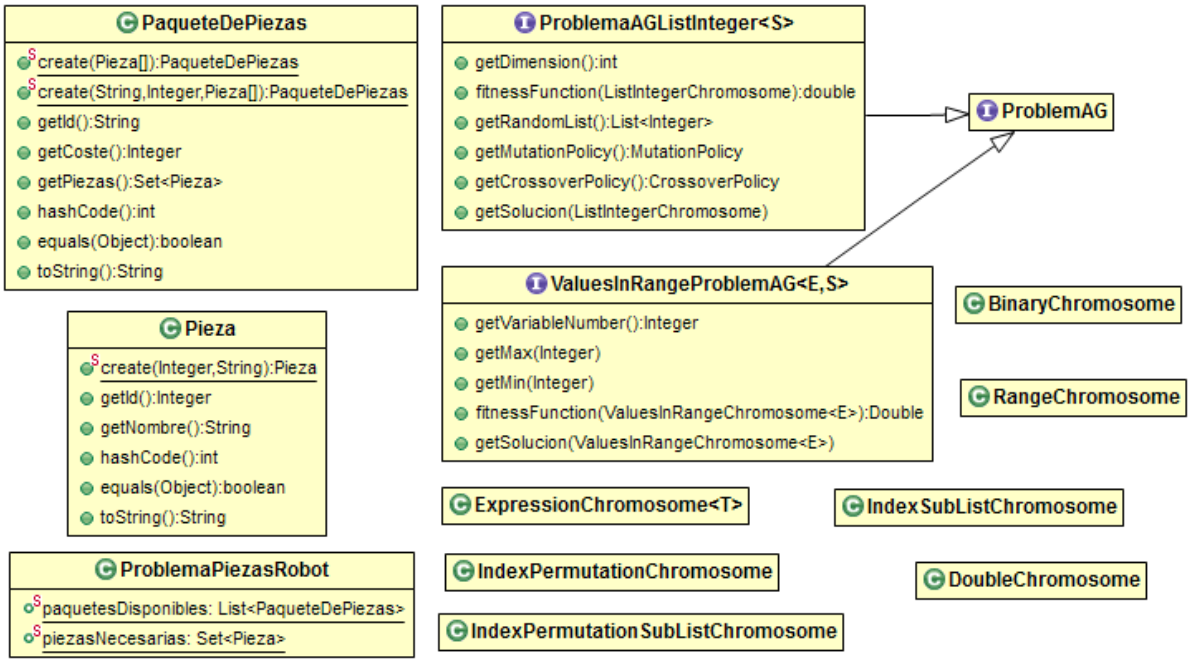
Problema de las Piezas del Robot: Algoritmos Genéticos	
Tipo de Problema: ValueInRangeProblemAG<E,S>	
Tipos	*E-Integer* *S – List<PaqueteDePiezas>*
Propiedades Compartidas	*paquetesDisponibles, List<PaqueteDePiezas>* *piezasNecesarias: Set<Pieza>*
Tipo de Cromosoma: **//TODO**	
Decode	*d: List<Integer>* *d.size(): Número de paquetes disponibles* *di:[0,1], entero –* **//TODO**
Fitness: **//TODO**	
Solución: **// TODO**	

2. Implemente los siguientes métodos para la clase **ProblemaRobotAG**:

```
public Integer getVariableNumber() { //TODO }
 public Integer getMax(Integer i) { //TODO }
 public Integer getMin(Integer i) { //TODO }
 public Double fitnessFunction(ValuesInRangeChromosome<Integer> cr) { //TODO }
 public List<PaqueteDePiezas>getSolucion(ValuesInRangeChromosome<Integer> cr){ //TODO }
```

NOTA: Tenga en cuenta que la implementación de Algoritmos Genéticos maximiza por defecto.

Ejemplo:	**Solución:**
PIEZAS NECESARIAS = [B, C, D, E, G] PAQUETES DE PIEZAS ID : PIEZAS -> COSTE ========== Set1:[B, C] -> 1 Set2:[A, B, D, E] -> 2 Set3:[B, E, F] -> 3 Set4:[B, F] -> 3 Set5:[B, D, E, F] -> 4 Set6:[A, B, D] -> 1 Set7:[A, B, H] -> 2 Set8:[E, F, G] -> 3	Paquetes de piezas seleccionados: ---------- Set1:[B, C] -> 1 Set6:[A, B, D] -> 1 Set8:[E, F, G] -> 3 Coste total: 5 Piezas compradas: [A, B, C, D, E, F, G] Piezas que faltan: 0



Solución:

Problema de las Piezas del Robot: Algoritmos Genéticos	
Tipo de Problema: ValueInRangeProblemAG<E,S>	
Tipos	*E-Integer* *S – List<PaqueteDePiezas>*
Propiedades Compartidas	*paquetesDisponibles, List<PaqueteDePiezas>* *piezasNecesarias: Set<Pieza>*
Tipo de Cromosoma: *BinaryChromosome*	
Decode	*d: List<Integer>* *d.size(): Número de paquetes disponibles* *di:[0,1], entero – 1 si se selecciona el paquete i , 0 en caso contrario.*
Fitness: *-costeTotal-piezasQueFaltan *100000L* • $costeTotal = \sum_{i \in A} paquetesDisponibles(i).getCoste()$ • $piezasQueFaltan = \left\| piezasNecesarias - \bigcup_{i \in A} paquetesDisponible(i).getPiezas() \right\|$ • $A = \{i \in [1,n) / d_i == 1\}$	
Solución: *s={}* *range(0, n - 1).filter(x->d_x ==1).forEach(x → s+paquetesDisponible(x))*	

```
public Integer getVariableNumber() {
	return ProblemaPiezasRobot.paquetesDisponibles.size();
}

public Integer getMax(Integer i) {
	return 1;
}

public Integer getMin(Integer i) {
	return 0;
}

public Double fitnessFunction(ValuesInRangeChromosome<Integer> cr) {
	List<PaqueteDePiezas>lps = getSolucion(cr);

	Double costeTotal = (double)lps.stream().mapToInt(sp-> sp.getCoste()).sum();
	Set<Pieza> piezasCompradas = lps.stream()
		.map(p -> p.getPiezas())
		.flatMap(sp ->sp.stream())
		.collect(Collectors.toSet());
	Double piezasKFaltan = (double)ProblemaPiezasRobot.piezasNecesarias.stream()
		.filter(p -> !piezasCompradas.contains(p))
		.count();

	return -costeTotal - piezasKFaltan * 100000L;
```

```
}

public List<PaqueteDePiezas>getSolucion(ValuesInRangeChromosome<Integer> cr) {
	List<Integer> ls = cr.decode();

	return IntStream.range(0, ls.size()).boxed()
		.filter(i -> ls.get(i)==1)
		.map(i -> ProblemaPiezasRobot.paquetesDisponibles.get(i))
			.collect(Collectors.toList());
}
```