Ejercicio 4 – Algoritmos Genéticos

Se tiene una colección de libros que deben ser colocados en una estantería compuesta por distintos estantes. Cada libro tiene una altura y una anchura determinada, mientras que todos los estantes tienen la misma anchura, pero distinta altura. Determinar una distribución de libros en los estantes de forma que se maximice el número de libros colocados.

EJEMPLO: para una estantería de anchura 7 compuesta por dos estantes *E0* y *E1*, de alturas 15 y 18 respectivamente, los siguientes libros:

Libro	Altura	Anchura
L0	15	2,5
L1	20	1,5
L2	10	4
L3	12	4,5
L4	18	3
L5	17	1
L6	16	1,2

deberían ubicarse de la siguiente forma:

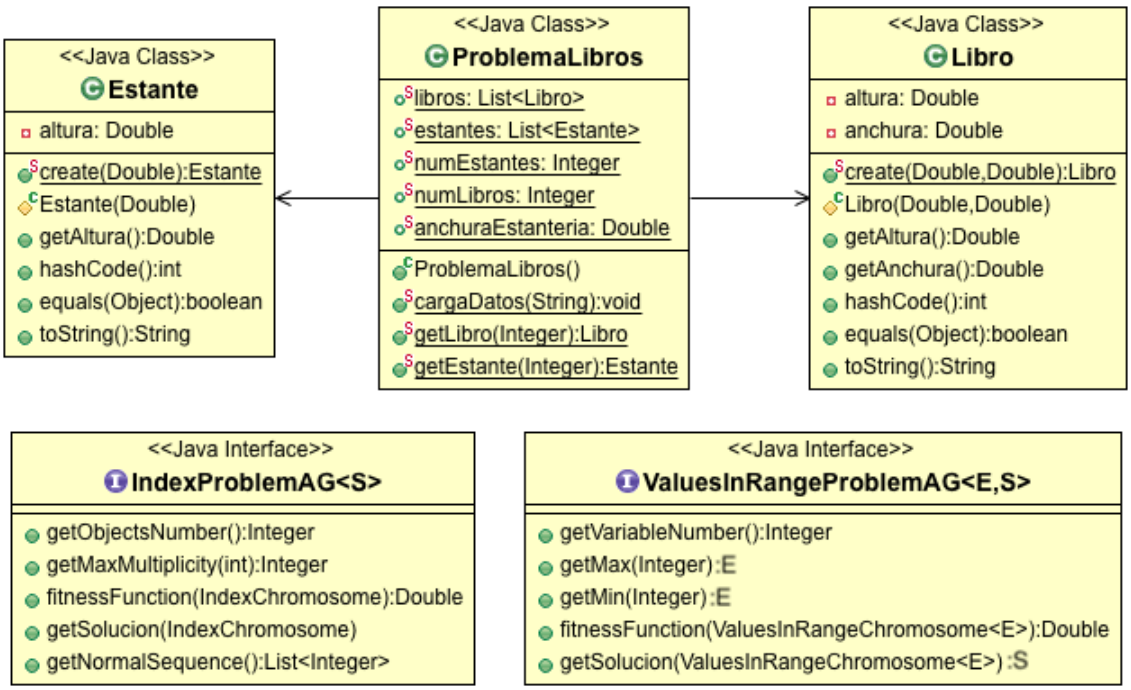
E0 = {*L0*, *L2*}
E1 = {*L3*, *L5*, *L6*}

de manera que quede colocado el mayor número posible de libros.
Véase que no es necesario ubicar todos los libros en la estantería.

SE PIDE: Resolver el problema mediante **Algoritmos Genéticos**.

a) Indique, razonadamente, el tipo de cromosoma seleccionado.
b) Realice la implementación de la clase *ProblemaLibrosAG*.

NOTA: Suponga que los datos del problema, contenidos en un fichero, ya han sido cargados en *ProblemaLibros*.



Algoritmos Genéticos (SE ADMITE CUALQUIER TIPO DE SOLUCION QUE INDIQUE EL MAPEO ESTANTES-LIBROS)

```
public class ProblemaLibrosAG implements
	ValuesInRangeProblemAG<Integer,SolucionLibros>{

...

public Integer getVariableNumber() {
	return ProblemaLibros.numLibros;
}

public Integer getMax(Integer i) {
	return ProblemaLibros.numEstantes;
}

public Integer getMin(Integer i) {
	return -1;
}

public Double fitnessFunction(ValuesInRangeChromosome<Integer> cr) {
	List<Integer> list = cr.decode();
	SolucionLibros sol = getSolucion(cr);

	//cuántos libros hay ubicados?
	long librosEnEstanteria = list.stream().filter(x -> x != -1).count();

	//todos caben por altura?
	long librosQueNoCabenAltura = IntStream.range(0, list.size())
		.boxed()
		.filter(i -> list.get(i)!= -1)
		.filter(i -> ProblemaLibros.getLibro(i).getAltura() >
			ProblemaLibros.getEstante(list.get(i)).getAltura())
		.count();

	//algún estante sobrepasado en anchura?
	long estantesSobrepasadosAnchura =
		IntStream.range(0, ProblemaLibros.numEstantes)
		.boxed()
		.filter(est ->
		sol.existeEstanteEnLaSolucion(ProblemaLibros.getEstante(est)))
		.filter(est ->
		sol.getLibrosEnEstante(ProblemaLibros.getEstante(est))
			.stream()
			.map(libro -> libro.getAnchura())
			.reduce(0.0, Double::sum) >
				ProblemaLibros.anchuraEstanteria)
		.count();

	return (double) (librosEnEstanteria -
		(ProblemaLibros.numLibros * ProblemaLibros.numLibros *
		(librosQueNoCabenAltura + estantesSobrepasadosAnchura)));
}

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

	IntStream.range(0,ls.size())
		.filter(i -> ls.get(i)!=-1)
		.forEach(i -> s.add(ProblemaLibros.getLibro(i),
			ProblemaLibros.getEstante(ls.get(i))));

	return s;}}
```

```java
public class SolucionLibros {

	private Map<Estante,List<Libro>> map;

	public static SolucionLibros create() {
		return new SolucionLibros();
	}

	private SolucionLibros() {
		super();
		this.map = new HashMap<Estante,List<Libro>>();
	}

	public void add(Libro lib, Estante est) {
		if(map.containsKey(est)) {
			map.get(est).add(lib);
		}else {
			List<Libro> listLib = new ArrayList<Libro>();
			listLib.add(lib);
			map.put(est,listLib);
		}
	}

	public List<Libro> getLibrosEnEstante(Estante est) {
		return map.get(est);
	}

	public boolean existeEstanteEnLaSolucion(Estante estante) {
		return map.containsKey(estante);
	}

	@Override
	public String toString() {
		return "SolucionLibros [map=" + map + "]";
	}


}
```