Ejercicios 1 y 2 – Programación Lineal y Algoritmos Genéticos

Una panadería dispone de una cantidad de kilos de harina y litros de agua para elaborar sus distintos tipos de productos. Teniendo en cuenta que cada tipo de producto reporta un beneficio distinto, y que cada uno requiere una cantidad distinta de harina y agua para fabricarse, es necesario determinar cuántas unidades elaborar de cada uno de los productos de forma que el beneficio sea el máximo.

Por ejemplo, para 50 kilos de harina y 40 litros de agua, y los siguientes 4 productos:

Tipo de producto	Kg de harina por unidad	Litros de agua por unidad	Beneficio por unidad
P1	0,4	0,6	2
P2	0,6	0,3	1,5
P3	0,5	0,4	1
P4	0,7	0,7	2,5

La mejor opción sería: 35 unidades de P1, 54 de P2, ninguna de P3 y 4 de P4.

SE PIDE:

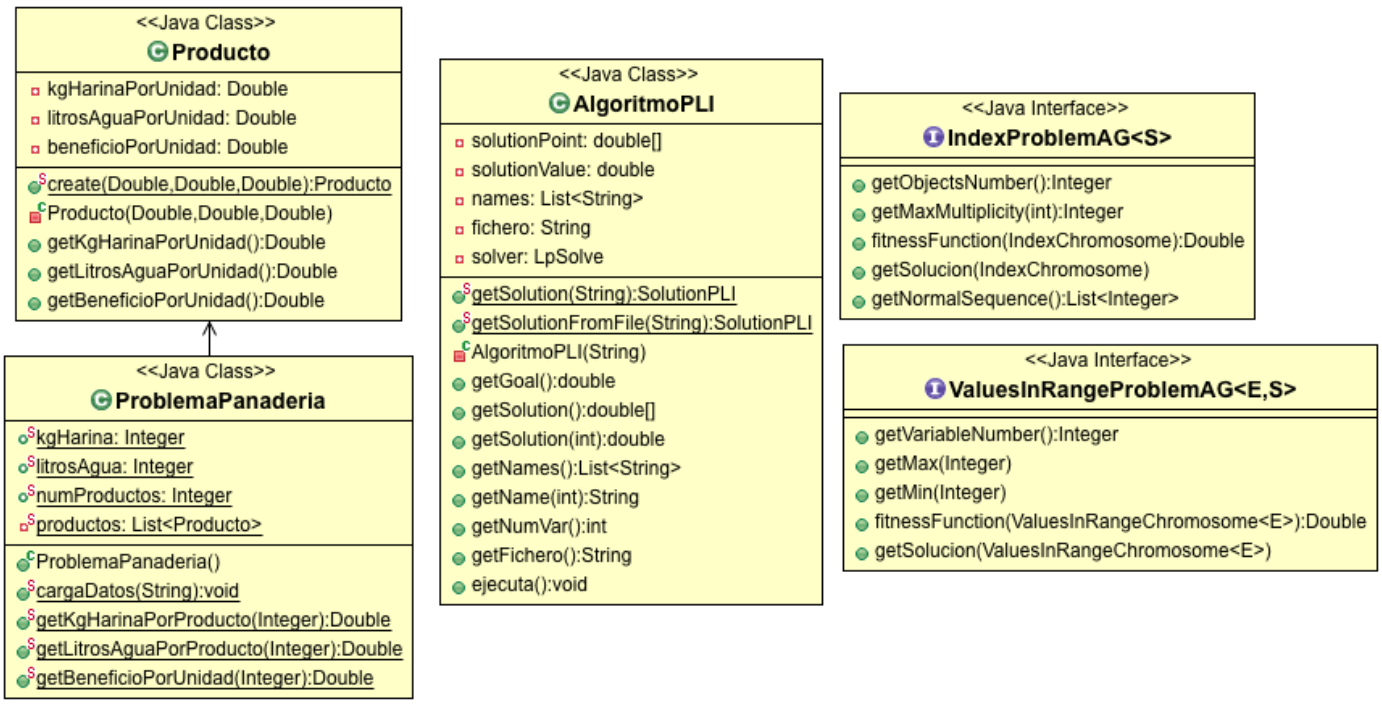
Ejercicio 1. Resolver el problema mediante **Programación Lineal**. Para ello:

a. Redacte el fichero *LpSolve* para resolver el ejemplo concreto.
b. Implemente el código necesario para resolver automáticamente el problema genérico, invocando al método *getSolution(String)* de la clase *AlgoritmoPLI*.

Ejercicio 2. Resolver el problema mediante **Algoritmos Genéticos**. Para ello:

c. Indique, razonadamente, la selección del tipo de cromosoma.
d. Implemente la clase *ProblemaPanaderiaAG* acorde al tipo de cromosoma seleccionado.

Para la resolución de ambos ejercicios, suponga que los datos de entrada vienen dados en un fichero a partir del cual es posible crear un *ProblemaPanaderia*.



Ejercicio 1. Programación Lineal

a)

```
max: B;

B = 2p1 + 1.5p2 + 1p3 + 2.5p4;

H >= 0.4p1 + 0.6p2 + 0.5p3 + 0.7p4;
A >= 0.6p1 + 0.3p2 + 0.4p3 + 0.7p4;

p1 >= 0;
p2 >= 0;
p3 >= 0;
p4 >= 0;

H = 50;
A = 40;

int p1, p2, p3, p4;
```

b)

```
private static String getConstraints() {
	String res = "max: B;\n";

	res += IntStream.range(0, ProblemaPanaderia.numProductos)
		.boxed()
		.map(i -> String.format("%sp%d",
			ProblemaPanaderia.getBeneficioPorUnidad(i).toString(
			).replace(",","."), i))
		.collect(Collectors.joining("+", "B = ", ";\n"));
	res += "\n";

	res += IntStream.range(0, ProblemaPanaderia.numProductos)
		.boxed()
		.map(i -> String.format("%sp%d",
			ProblemaPanaderia.getKgHarinaPorProducto(i).toString
			().replace(",","."), i))
		.collect(Collectors.joining("+", "H >= ", ";\n"));

	res += IntStream.range(0, ProblemaPanaderia.numProductos)
		.boxed()
		.map(i -> String.format("%sp%d",
			ProblemaPanaderia.getLitrosAguaPorProducto(i).toStri
			ng().replace(",","."), i))
		.collect(Collectors.joining("+", "A >= ", ";\n"));

	res += "H = " + ProblemaPanaderia.kgHarina + ";\n";
	res += "A = " + ProblemaPanaderia.litrosAgua + ";\n";

	res += IntStream.range(0, ProblemaPanaderia.numProductos)
		.boxed()
		.map(i -> String.format("p%d", i))
		.collect(Collectors.joining(",", "int ", ";\n"));

	return res;
}
```

Ejercicio 2. Algoritmos Genéticos

Es posible resolverlo tanto con ValuesInRangeChromosome (ChromosomeType.***Range***) como con IndexChromosome (ChromosomeType.***IndexSubList***). Se aporta la solución con ValuesInRangeChromosome por ser la más común.

```
public class ProblemaPanaderiaAG implements
            ValuesInRangeProblemAG<Integer, SolucionPanaderia> {

public Integer getVariableNumber() {
      return ProblemaPanaderia.numProductos;
}

public Integer getMax(Integer index) {
      return (int) Math.min(ProblemaPanaderia.kgHarina/
            ProblemaPanaderia.getKgHarinaPorProducto(index),
            ProblemaPanaderia.litrosAgua/
            ProblemaPanaderia.getLitrosAguaPorProducto(index));
}

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

public SolucionPanaderia getSolucion(ValuesInRangeChromosome<Integer>
      cromosoma) {
      List<Integer> decode = cromosoma.decode();
      Double beneficio = calculaBeneficio(decode);
      SolucionPanaderia solucion = SolucionPanaderia.create(decode,
            beneficio);
      return solucion;
}

private Double calculaBeneficio(List<Integer> l) {
      return IntStream.range(0, l.size())
                  .boxed()
                  .mapToDouble(i -> l.get(i) *
                        ProblemaPanaderia.getBeneficioPorUnidad(i))
                  .sum();
}

public Double fitnessFunction(ValuesInRangeChromosome<Integer>
      cromosoma) {
      SolucionPanaderia solucion = getSolucion(cromosoma);

      Double factorPenalizacion = calculaFactorPenalizacion();
      Double kilosH = kilosUtilizadosHarina(solucion);
      Double litrosA = litrosUtilizadosAgua(solucion);

      Double penalH = kilosH>ProblemaPanaderia.kgHarina?kilosH -
            ProblemaPanaderia.kgHarina:.0;
      Double penalA = litrosA > ProblemaPanaderia.litrosAgua?litrosA -
            ProblemaPanaderia.litrosAgua:.0;

      return solucion.getBeneficio() -
            factorPenalizacion*(penalH+penalA);
}
private Double calculaFactorPenalizacion() {
      return Math.sqrt(IntStream.range(0,
            ProblemaPanaderia.numProductos)
```

```
            .boxed()
            .map(i -> getMax(i) *
            ProblemaPanaderia.getBeneficioPorUnidad(i))
            .reduce(0.0, Double::sum));
      }

private Double litrosUtilizadosAgua(SolucionPanaderia solucion) {
      return IntStream.range(0, ProblemaPanaderia.numProductos)
                  .boxed()
                  .mapToDouble(i ->
                        solucion.getCantidadProducida().get(i) *
                        ProblemaPanaderia.getLitrosAguaPorProducto(i))
                  .sum();
}

private Double kilosUtilizadosHarina(SolucionPanaderia solucion) {
      return IntStream.range(0, ProblemaPanaderia.numProductos)
                  .boxed()
                  .mapToDouble(i ->
                        solucion.getCantidadProducida().get(i) *
                        ProblemaPanaderia.getKgHarinaPorProducto(i))
                  .sum();
}}
```

NO ES NECESARIO DESARROLLAR UNA CLASE SOLUCIÓN, AUNQUE ESTA SOLUCIÓN SÍ LA UTILIZA:

```
public class SolucionPanaderia {

private List<Integer> cantidadProducida;
private Double beneficio;

public static SolucionPanaderia create(List<Integer> l, Double b) {
      return new SolucionPanaderia(l, b);
}

protected SolucionPanaderia(List<Integer> l, Double b) {
      cantidadProducida = new ArrayList<Integer>(l);
      beneficio = b;
}

public List<Integer> getCantidadProducida() {
      return cantidadProducida;
}

public Double getBeneficio() {
      return beneficio;
}

public String toString() {
      return "SolucionPanaderia [cantidadProducida=" +
cantidadProducida + ", beneficio=" + beneficio + "]";
}

public int hashCode() {
      …
}

public boolean equals(Object obj) {
      …
}}
```