Ejercicio 4

Una red social desea maximizar el número de veces que sus miembros acceden a la misma. Una técnica para ello es enviar a los usuarios correos cada día animándoles a visitar la red. Por cada usuario se dispone de una tabla estadística que refleja, de lunes a domingo, el número de veces promedio que un usuario accedió a la red cada día *d* de la semana durante 2017 si dicho día recibió *x* correos, $d \in \{Lunes,...,Domingo\}$, $x \in [0, maxPorDia]$. El máximo número de correos que la red social puede enviar a un usuario por día es *maxPorDia* y, por semana, *maxPorSem*, siendo *maxPorDia* y *maxPorSem* dos valores dados. El número de correos que se envía cada día de la semana puede ser diferente.

Dados los datos estadísticos de 2017 para el usuario *Acme*, se desea aplicar la técnica de *Algoritmos Genéticos* para calcular el número de correos que hay que enviar a *Acme* cada día de la semana de 2018 para maximizar su número de visitas semanales a la red, teniendo en cuenta que el máximo número de correos que se puede enviar por día es *maxPorDia* y, por semana, *maxPorSem*.

Ejemplo: en la siguiente tabla (datos de entrada del algoritmo) podemos apreciar que aquellos lunes de 2017 en los que *Acme* recibió 2 correos, éste accedió a la red 4 veces (de media). Aquellos viernes de 2017 en los que no recibió correos, *Acme* accedió a la red 7 veces (de media). En esta tabla, *maxPorDia*=4 (dato de entrada del algoritmo). Otro dato de entrada del algoritmo es *maxPorSem*, al que daremos un valor de 18.
Una posible solución se muestra a la derecha de la tabla.

Número promedio de visitas del usuario Acme por día de la semana durante 2017							
Correos recibidos	L	M	Mi	J	V	S	D
0	1	2	4	5	7	7	2
1	2	3	7	8	11	12	6
2	4	7	8	9	15	20	10
3	10	9	8	9	20	35	11
4	6	8	6	7	14	20	5

Una posible solución es enviar los lunes 3 correos, los martes 3, los miércoles 2, los jueves 2, los viernes 3, los sábados 3 y los domingos 2. Esta solución conseguiría que *Acme* visitara la red 101 veces cada semana de 2018, si se cumplieran las estadísticas de 2017.

Se pide:

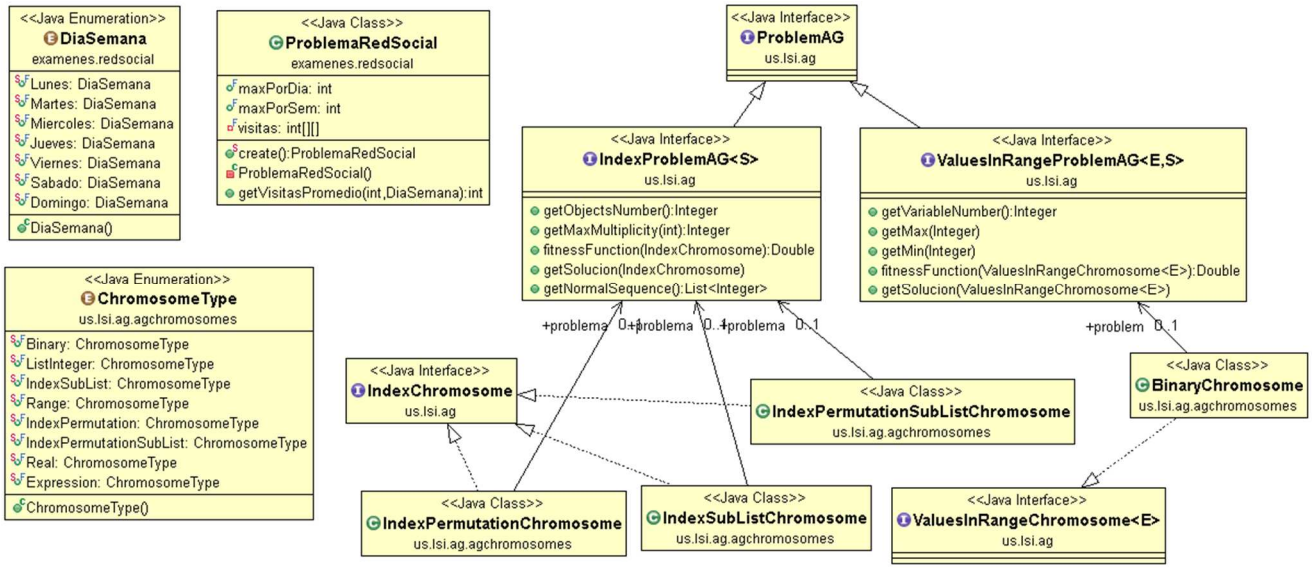
a) Una explicación razonada del tipo de cromosoma a utilizar.
b) El tipo que se va a utilizar para la solución del problema.
c) La clase **`ProblemaRedSocialAG`** (sólo aquellos atributos y métodos necesarios para implementar la interfaz que corresponda).

Nota 1: la función de *fitness* debe seguir la expresión que se ha estudiado en teoría.
Nota 2: para el acceso a la tabla puede utilizar el siguiente método de la clase **`ProblemaRedSocial`**:

```
int getVisitasPromedio (int x, DiaSemana d)
```

que devuelve el número de visitas promedio que el usuario *Acme* efectuó durante 2017 el día de la semana *d* si dicho día recibió *x* correos.
Nota 3: El método estático *`values()`* de un tipo enumerado devuelve un array con los valores del tipo.

Por detrás de esta hoja puede encontrar un diagrama con las interfaces y clases necesarias.


<<Java Enumeration>>
DiaSemana
examenes.redsocial
Lunes: DiaSemana
Martes: DiaSemana
Miercoles: DiaSemana
Jueves: DiaSemana
Viernes: DiaSemana
Sabado: DiaSemana
Domingo: DiaSemana
DiaSemana()
<<Java Class>>
ProblemaRedSocial
examenes.redsocial
maxPorDia: int
maxPorSem: int
visitas: int[][]
create():ProblemaRedSocial
ProblemaRedSocial()
getVisitasPromedio(int,DiaSemana):int
<<Java Interface>>
ProblemAG
us.lsi.ag
<<Java Interface>>
IndexProblemAG<S>
us.lsi.ag
getObjectsNumber():Integer
getMaxMultiplicity(int):Integer
fitnessFunction(IndexChromosome):Double
getSolucion(IndexChromosome)
getNormalSequence():List<Integer>
<<Java Interface>>
ValuesInRangeProblemAG<E,S>
us.lsi.ag
getVariableNumber():Integer
getMax(Integer)
getMin(Integer)
fitnessFunction(ValuesInRangeChromosome<E>):Double
getSolucion(ValuesInRangeChromosome<E>)
<<Java Enumeration>>
ChromosomeType
us.lsi.ag.agchromosomes
Binary: ChromosomeType
ListInteger: ChromosomeType
IndexSubList: ChromosomeType
Range: ChromosomeType
IndexPermutation: ChromosomeType
IndexPermutationSubList: ChromosomeType
Real: ChromosomeType
Expression: ChromosomeType
ChromosomeType()
+problema 0..1
+problema 0..1
+problema 0..1
+problem 0..1
<<Java Interface>>
IndexChromosome
us.lsi.ag
<<Java Class>>
IndexPermutationSubListChromosome
us.lsi.ag.agchromosomes
<<Java Class>>
BinaryChromosome
us.lsi.ag.agchromosomes
<<Java Class>>
IndexPermutationChromosome
us.lsi.ag.agchromosomes
<<Java Class>>
IndexSubListChromosome
us.lsi.ag.agchromosomes
<<Java Interface>>
ValuesInRangeChromosome<E>
us.lsi.ag

SOLUCIÓN

a) Para encontrar el tipo de cromosoma adecuado, comencemos estudiando qué tipo de solución queremos. Buscamos una asociación de cada día de la semana con un número de correos a enviar, que es simplemente un valor en el rango [0,*maxPorDia*]. Por supuesto, la solución debe satisfacer unas restricciones y ser la mejor en el sentido de maximización, pero esto no cambia su estructura. Esta estructura de la solución sugiere el uso del cromosoma *ValuesInRange*.

b) Como hemos explicado en el apartado anterior, la solución es una asociación de cada día de la semana con un número de correos. Por tanto, el tipo solución es Map<DiaSemana,Integer>.

c) El código de la clase ProblemaRedSocialAG se muestra a continuación. Aunque no se pedía en el ejercicio, incluimos también el constructor y el método create.

```
public class ProblemaRedSocialAG
    implements ValuesInRangeProblemAG<Integer,
                                  Map<DiaSemana,Integer>> {


      ProblemaRedSocial problemaOriginal;

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

      private ProblemaRedSocialAG () {
            problemaOriginal= ProblemaRedSocial.create();
      }

      @Override
      public Map<DiaSemana,Integer> getSolucion
            (ValuesInRangeChromosome<Integer> chromosome) {

            List<Integer> ls = chromosome.decode();
            Map<DiaSemana,Integer> result= new
                                     TreeMap<DiaSemana,Integer>();
            IntStream.range(0, 7)
            .forEach(i-> result.put(DiaSemana.values()[i], ls.get(i)));

            return result;
      }




      @Override
      public Integer getVariableNumber() {
             return DiaSemana.values().length;
      }

      @Override
      public Integer getMax(Integer index){
            return problemaOriginal.maxPorDia;
      }

      @Override
      public Integer getMin(Integer index){
            return 0;
      }
```

```
	@Override
	public Double fitnessFunction(ValuesInRangeChromosome<Integer>
	                                        crom) {

		List<Integer> d= crom.decode();

		long v = IntStream.range(0,7)

		    .mapToLong(i ->
		        problemaOriginal.getVisitasPromedio(d.get(i),
		             DiaSemana.values()[i]))
		    .sum();

		int n = problemaOriginal.maxPorDia;
		int K = (int) Math.pow(n, 2);

		long correosPorSemana= IntStream.range(0,7)
		            .mapToLong(i -> d.get(i))
		            .sum();

		long R1= correosPorSemana<=problemaOriginal.maxPorSem? 0 :
		            correosPorSemana-problemaOriginal.maxPorSem;

		double fitness= v - K*R1;

		return fitness;
	}

}
```