PROBLEMA 4: Algoritmos Genéticos

Se desea encontrar la mejor asignación entre una lista de **N personas** y una lista de **M proyectos**, **siendo N<=M,** de forma que se **minimice la suma total de incompatibilidades**. Cada persona debe tener asignado un proyecto diferente y, además, pueden quedar proyectos sin asignar, ya que el número de personas puede ser inferior o igual al número de proyectos. Para obtener tal solución, se pide resolver el problema mediante **Algoritmos Genéticos**.

Cada proyecto tiene la lista de requisitos mínimos que deben cumplirse y cada persona tiene la lista de requisitos que cumple. Por tanto, **el número de incompatibilidades entre un proyecto y una persona se define como el número de requisitos del proyecto incumplidos por dicha persona.**

EJEMPLO: Dada la **lista de requisitos** {A, B, C, D}.
Dada la siguiente lista de personas, los requisitos que cumple cada una son:

- **Persona 0** → {A, D}
- **Persona 1** → {B, C, D}

Dada la siguiente lista de proyectos, los requisitos que requiere cada proyecto son:

- **Proyecto 0** → {A, C, D}
- **Proyecto 1** → {A}
- **Proyecto 2** → {B, D}

La solución óptima sería la siguiente asignación con 0 incompatibilidades en total, que es el mínimo valor que se puede obtener para estos conjuntos de personas y proyectos:

- **Persona 0 → Proyecto 1** (0 incompatibilidades, persona 0 cumple todos los requisitos del proyecto 1)
- **Persona 1 → Proyecto 2** (0 incompatibilidades, persona 1 cumple todos los requisitos del proyecto 2)

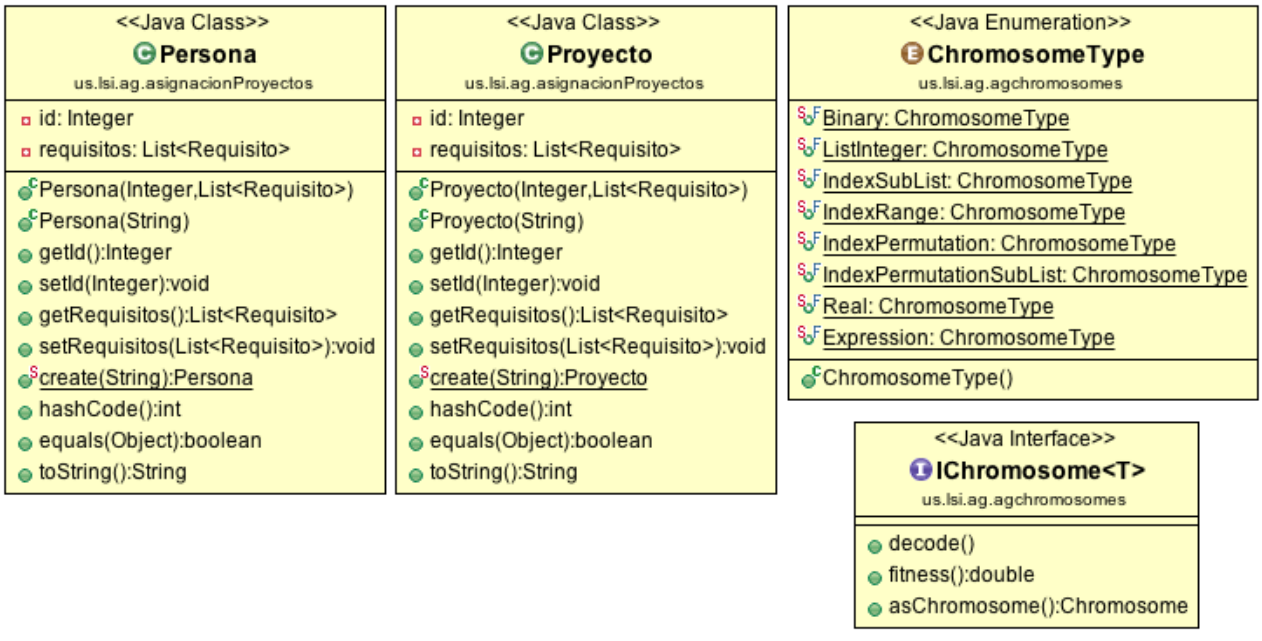
SE PIDE

1. ¿Qué tipo o tipos de cromosomas son los más adecuados para resolver el problema y por qué?
2. Defina matemáticamente cuál sería la función fitness para este problema de forma genérica.
3. Implemente los siguientes métodos pertenecientes a la clase **ProblemaProyectosAG** para la técnica **Algoritmos Genéticos**:

```
public Double fitnessFunction(… chromosome) { //TODO}
public Map<Persona,Proyecto> getSolucion(… chromosome){//TODO}
```

NOTAS:

- Tenga en cuenta que la implementación de Algoritmos Genéticos es de maximización por defecto.
- **ProblemaProyectosAG** tiene las propiedades: `List<Persona> personas, List<Proyecto> proyectos`.



Solución:

1. ¿Qué tipo o tipos de cromosomas son los más adecuados para resolver el problema y por qué?

   Respuesta
   El tipo de cromosoma más adecuado es el cromosoma *IndexPermutation* (de tamaño igual al número de proyectos) ya que se trata de una asignación donde importa el orden. Nos quedaríamos con los primeros n elementos, siendo n el número de personas, para obtener la solución asociada al cromosoma.
   También sería válido el cromosoma *IndexPermutationSubList*, siendo soluciones válidas aquellos cromosomas que seleccionen tantos proyectos como personas a asignar.

2. Defina matemáticamente cuál sería la función fitness.

   Respuesta
   La función fitness a maximizar sería la suma total del número de requisitos incumplidos por cada persona con respecto al proyecto que ha sido asignado a cada una de ellas.
   Matemáticamente se formularía de la siguiente forma:

$$fitness = -\sum_{i=0}^{|personas|-1} |p(decode(i)).getRequisitos() - a(i).getRequisitos()|$$

   $p(x) = proyectos.get(x)$
   $a(x) = personas.get(x)$

3. Implemente los siguientes métodos pertenecientes a la clase **ProblemaProyectosAG** para la técnica **Algoritmos Genéticos**:

```
public Double fitnessFunction(… chromosome) { //TODO}
```

```
public Double fitnessFunction(IndexChromosome chromosome) {
       List<Integer> ls = chromosome.decode();

       double incompatibilidades =IntStream.range(0,personas.size())
  .boxed().mapToDouble(i ->
              (double)proyectos.get(ls.get(i)).getRequisitos().stream()
              .filter(pt -
       > !personas.get(i).getRequisitos().contains(pt)).count()).sum();

       return -incompatibilidades;
}
```

```
public Map<Persona,Proyecto> getSolucion(… chromosome){//TODO}
```

```
public Map<Persona,Proyecto> getSolucion(IndexChromosome chromosome) {
       List<Integer> ls = chromosome.decode();
       Map<Persona,Proyecto> ms =  new HashMap<>();

       IntStream.range(0,personas.size())
                .boxed().forEach(i ->ms.put(personas.get(i),
                                            proyectos.get(ls.get(i))));
       return ms;


}
```