Ejercicio 2: Voraz

Una determinada empresa de trabajo temporal desea encontrar la mejor asignación entre la lista de posibles candidatos y la lista de ofertas de trabajo disponibles.

Cada candidato ha informado a la empresa sobre los requisitos que satisface para las ofertas de trabajo a las que desearía optar (*idiomas, disponibilidad para viajar, experiencia superior a un año y formación universitaria*). Por ejemplo, un candidato quiere optar a una oferta que requiera conocimientos altos de idiomas y formación universitaria, pero no desea viajar ni tiene experiencia previa superior a un año.

La empresa desea asignar una oferta diferente a cada uno de los candidatos de forma que se **minimice la suma total de incompatibilidades** para todos los candidatos.

Por ejemplo, dado el siguiente conjunto de candidatos:

Candidatos	Idiomas	Disponibilidad para viajar	Experiencia superior a 1 año	Formación universitaria
C1	Si	No	No	Si
C2	No	Si	Si	Si

Y las siguientes ofertas de trabajo disponibles:

Ofertas	Idiomas	Disponibilidad para viajar	Experiencia superior a 1 año	Formación universitaria
O1	Si	No	Si	Si
O2	Si	No	No	No
O3	No	Si	No	Si

En este ejemplo, la solución óptima sería asignar la oferta O2 al candidato C1 (0 incompatibilidades, ya que C1 cumple todos los requisitos mínimos de la oferta O2) y la oferta O3 al candidato C2 (también 0 incompatibilidades), lo que hace un total de 0 incompatibilidades que es el mínimo valor que se puede obtener para estos conjuntos de candidatos y ofertas. **Tenga en cuenta que el número de incompatibilidades entre una oferta y un candidato se define como el número de requisitos que una oferta requiere y no posee el candidato.**

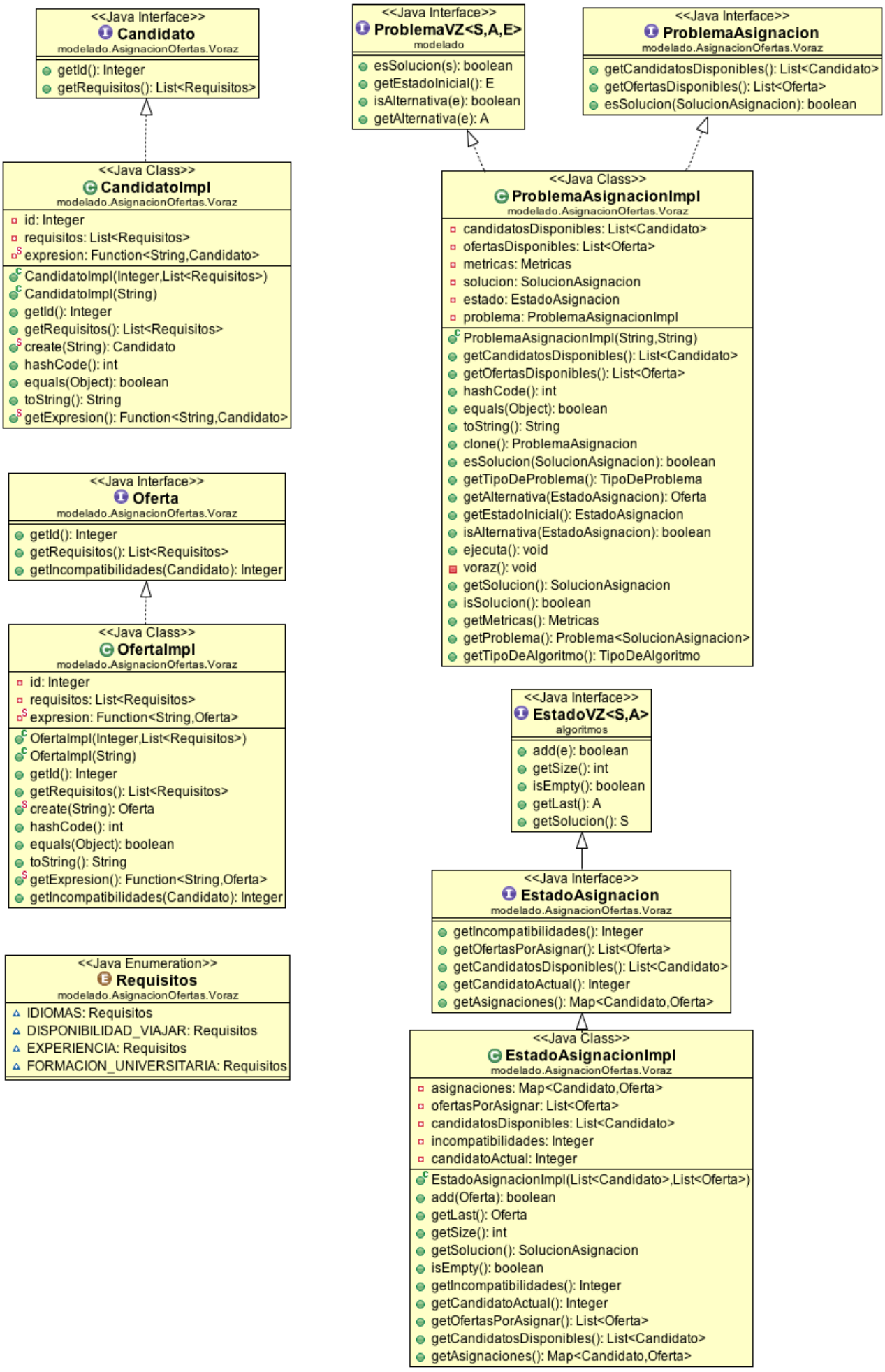
El objetivo del problema será encontrar la solución que contenga la asignación entre candidatos y ofertas de trabajo que **minimice la suma total de incompatibilidades**. Para obtener tal asignación, se pide resolver el problema mediante la técnica **Voraz**.

La alternativa se modelará como la mejor oferta elegida para el candidato correspondiente a la posición `candidatoActual`. Tenga en cuenta que cada candidato tiene que tener asignado **una oferta diferente** y además pueden quedar ofertas sin asignar.

SE PIDE:

1) Implemente los siguientes métodos pertenecientes a:
   - Clase `OfertaImpl`:
     a. `public Integer getIncompatibilidades(Candidato c)`
   - Clase `ProblemaAsignacionImpl`:
     b. `public boolean isAlternativa(EstadoAsignacion e)`
     c. `public EstadoAsignacion getEstadoInicial()`
     d. `public Oferta getAlternativa(EstadoAsignacion e)`
   - Clase `EstadoAsignacionImpl`:
     e. `public boolean add(Oferta o)`

Tiempo estimado: 45 minutos.


<<Java Interface>>
Candidato
modelado.AsignacionOfertas.Voraz
getId(): Integer
getRequisitos(): List<Requisitos>
<<Java Class>>
CandidatoImpl
modelado.AsignacionOfertas.Voraz
id: Integer
requisitos: List<Requisitos>
expresion: Function<String,Candidato>
CandidatoImpl(Integer,List<Requisitos>)
CandidatoImpl(String)
getId(): Integer
getRequisitos(): List<Requisitos>
create(String): Candidato
hashCode(): int
equals(Object): boolean
toString(): String
getExpresion(): Function<String,Candidato>
<<Java Interface>>
Oferta
modelado.AsignacionOfertas.Voraz
getId(): Integer
getRequisitos(): List<Requisitos>
getIncompatibilidades(Candidato): Integer
<<Java Class>>
OfertaImpl
modelado.AsignacionOfertas.Voraz
id: Integer
requisitos: List<Requisitos>
expresion: Function<String,Oferta>
OfertaImpl(Integer,List<Requisitos>)
OfertaImpl(String)
getId(): Integer
getRequisitos(): List<Requisitos>
create(String): Oferta
hashCode(): int
equals(Object): boolean
toString(): String
getExpresion(): Function<String,Oferta>
getIncompatibilidades(Candidato): Integer
<<Java Enumeration>>
Requisitos
modelado.AsignacionOfertas.Voraz
IDIOMAS: Requisitos
DISPONIBILIDAD_VIAJAR: Requisitos
EXPERIENCIA: Requisitos
FORMACION_UNIVERSITARIA: Requisitos
<<Java Interface>>
ProblemaVZ<S,A,E>
modelado
esSolucion(s): boolean
getEstadoInicial(): E
isAlternativa(e): boolean
getAlternativa(e): A
<<Java Interface>>
ProblemaAsignacion
modelado.AsignacionOfertas.Voraz
getCandidatosDisponibles(): List<Candidato>
getOfertasDisponibles(): List<Oferta>
esSolucion(SolucionAsignacion): boolean
<<Java Class>>
ProblemaAsignacionImpl
modelado.AsignacionOfertas.Voraz
candidatosDisponibles: List<Candidato>
ofertasDisponibles: List<Oferta>
metricas: Metricas
solucion: SolucionAsignacion
estado: EstadoAsignacion
problema: ProblemaAsignacionImpl
ProblemaAsignacionImpl(String,String)
getCandidatosDisponibles(): List<Candidato>
getOfertasDisponibles(): List<Oferta>
hashCode(): int
equals(Object): boolean
toString(): String
clone(): ProblemaAsignacion
esSolucion(SolucionAsignacion): boolean
getTipoDeProblema(): TipoDeProblema
getAlternativa(EstadoAsignacion): Oferta
getEstadoInicial(): EstadoAsignacion
isAlternativa(EstadoAsignacion): boolean
ejecuta(): void
voraz(): void
getSolucion(): SolucionAsignacion
isSolucion(): boolean
getMetricas(): Metricas
getProblema(): Problema<SolucionAsignacion>
getTipoDeAlgoritmo(): TipoDeAlgoritmo
<<Java Interface>>
EstadoVZ<S,A>
algoritmos
add(e): boolean
getSize(): int
isEmpty(): boolean
getLast(): A
getSolucion(): S
<<Java Interface>>
EstadoAsignacion
modelado.AsignacionOfertas.Voraz
getIncompatibilidades(): Integer
getOfertasPorAsignar(): List<Oferta>
getCandidatosDisponibles(): List<Candidato>
getCandidatoActual(): Integer
getAsignaciones(): Map<Candidato,Oferta>
<<Java Class>>
EstadoAsignacionImpl
modelado.AsignacionOfertas.Voraz
asignaciones: Map<Candidato,Oferta>
ofertasPorAsignar: List<Oferta>
candidatosDisponibles: List<Candidato>
incompatibilidades: Integer
candidatoActual: Integer
EstadoAsignacionImpl(List<Candidato>,List<Oferta>)
add(Oferta): boolean
getLast(): Oferta
getSize(): int
getSolucion(): SolucionAsignacion
isEmpty(): boolean
getIncompatibilidades(): Integer
getCandidatoActual(): Integer
getOfertasPorAsignar(): List<Oferta>
getCandidatosDisponibles(): List<Candidato>
getAsignaciones(): Map<Candidato,Oferta>

Solución

Implemente los siguientes métodos pertenecientes a:

- Clase `OfertaImpl`:

a. `public Integer getIncompatibilidades(Candidato c)`

```
public Integer getIncompatibilidades(Candidato c) {
      Integer res = 0;
      for (Requisitos r : requisitos) {
            if (!c.getRequisitos().contains(r)) {
                  res++;
            }
      }
      return res;

}
```

- Clase `ProblemaAsignacionImpl`:

b. `public boolean isAlternativa(EstadoAsignacion e)`

```
public boolean isAlternativa(EstadoAsignacion e) {
      return e.getCandidatoActual() <
                        e.getCandidatosDisponibles().size()
                        && !e.getOfertasPorAsignar().isEmpty();

}
```

c. `public EstadoAsignacion getEstadoInicial()`

```
public EstadoAsignacion getEstadoInicial() {
   return new EstadoAsignacionImpl( this.getCandidatosDisponibles(),
                                    this.getOfertasDisponibles());

}
```

d. `public Oferta getAlternativa(EstadoAsignacion e)`

```
public Oferta getAlternativa(EstadoAsignacion e) {
      Candidato c = this.getCandidatosDisponibles().get(
                                    e.getCandidatoActual());
      List<Oferta> ofertasDisponibles = e.getOfertasPorAsignar();
      Integer incompatibilidades = 0;
      Integer min = Integer.MAX_VALUE;
      Oferta ofertaElegida = null;
      for (Oferta o : ofertasDisponibles) {
            incompatibilidades = o.getIncompatibilidades(c);
            if (incompatibilidades < min) {
                  min = incompatibilidades;
                  ofertaElegida = o;
            }
      }
      return ofertaElegida;

}
```

- Clase `EstadoAsignacionImpl`:

e. **`public boolean`** *`add`*`(Oferta o)`

```
public boolean add(Oferta a) {
	boolean res = false;
	if (a != null) {
		Candidato candidato = getCandidatosDisponibles().get(
							candidatoActual);
		ofertasPorAsignar.remove(a);
		asignaciones.put(candidato, a);
		incompatibilidades += a.getIncompatibilidades(candidato);
		candidatoActual++;
		res = true;
	}
	return res;
}
```