Ejercicio 3:

El comité de oficiales de la prisión de Colditz se encarga de planificar las fugas de prisioneros. Sabiendo que para realizar una fuga es necesario reunir una serie de herramientas y que cada prisionero oculta un conjunto de herramientas que solo él sabe donde están. Se desea implementar un algoritmo que calcule todos los posibles grupos de prisioneros que podrían colaborar para realizar una fuga. El algoritmo también debe tener encuentra que los grupos de fugados pueden estar limitados por un número máximo de prisioneros, que dependerá de las peculiaridades de cada fuga. Por ejemplo dada la siguiente configuración de prisioneros:

P1	1 escalera, 1 pasaporte y plano
P2	1 escalera y 1 pala.
P3	1 cuerda y 1 escalera.
P4	1 plano y 1 ganzúa.

Si para realizar una fuga necesitamos disponer de 2 escaleras, 1 ganzúa, 1 plano y 1 cuerda y como máximo el grupo puede estar compuesto por 3 prisioneros, entonces los posibles grupos de prisioneros que podrían realizar la fuga serían: {P1, P3, P4} y {P2, P3, P4}.

Implemente los siguientes métodos TODO con ayuda del diagrama UML.

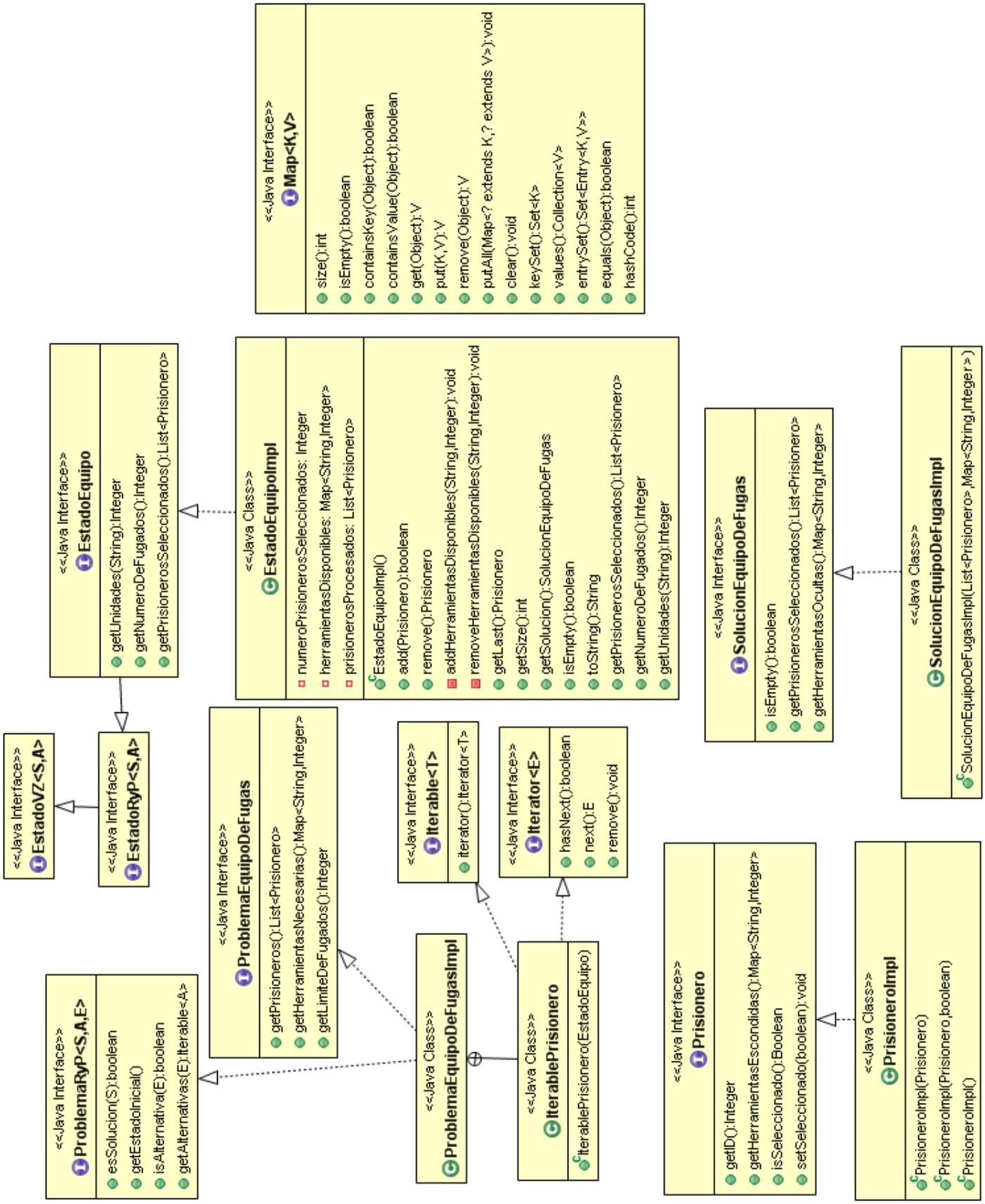
```
public class ProblemaEquipoDeFugasImpl implements … {
 private void ryp() { // TODO }
 public boolean isAlternativa(EstadoEquipo e) { // TODO  }
 public EstadoEquipo getEstadoInicial() { // TODO  }
 public boolean esSolucion(SolucionEquipoDeFugas s) { // TODO }
 // Devuelve un iterador con el siguiente prisionero a procesar. El siguiente
 // prisionero a procesar podrá o no estar seleccionado para la fuga, para ello
 // consulte el método isSeleccionado() del prisionero.
 public Iterable<Prisionero> getAlternativas(EstadoEquipo e){ //TODO  }
}

public class EstadoEquipoImpl implements EstadoEquipo {
 // Número de prisioneros que tiene que colaborar para fugarse.
 private Integer numeroPrisionerosSeleccionados = null;
 // Herramientas disponibles para la fuga. Las claves del mapa son cadenas que
 // representan las herramientas y los valores son enteros que representa
 // el número de unidades de disponible.
 private Map<String,Integer> herramientasDisponibles = null;
 // Prisioneros procesados por el estado, seleccionados o no para fugarse.
 private List<Prisionero> prisionerosProcesados = null;

 public boolean add(Prisionero prisionero) { // TODO  }
 public Prisionero remove() { // TODO }
 public SolucionEquipoDeFugas getSolucion() { // TODO }
 // Incrementa el número de herramientas disponibles para la fuga.
 private void addHerramientasDisponibles(String h, Integer cantidad) { // IMPLEMENTADA  }
 // Decrementa el número de herramientas disponibles para la fuga.
 private void removeHerramientasDisponibles(String h, Integer cantidad) { // IMPLEMENTADA}

}
```

Puntuación: 33’33% **Tiempo estimado: 45 minutos**


<<Java Interface>>
Map<K,V>
size():int
isEmpty():boolean
containsKey(Object):boolean
containsValue(Object):boolean
get(Object):V
put(K,V):V
remove(Object):V
putAll(Map<? extends K,? extends V>):void
clear():void
keySet():Set<K>
values():Collection<V>
entrySet():Set<Entry<K,V>>
equals(Object):boolean
hashCode():int
<<Java Interface>>
EstadoEquipo
getUnidades(String):Integer
getNumeroDeFugados():Integer
getPrisionerosSeleccionados():List<Prisionero>
<<Java Class>>
EstadoEquipoImpl
numeroPrisionerosSeleccionados: Integer
herramientasDisponibles: Map<String,Integer>
prisionerosProcesados: List<Prisionero>
EstadoEquipoImpl()
add(Prisionero):boolean
remove():Prisionero
addHerramientasDisponibles(String,Integer):void
removeHerramientasDisponibles(String,Integer):void
getLast():Prisionero
getSize():int
getSolucion():SolucionEquipoDeFugas
isEmpty():boolean
toString():String
getPrisionerosSeleccionados():List<Prisionero>
getNumeroDeFugados():Integer
getUnidades(String):Integer
<<Java Interface>>
SolucionEquipoDeFugas
isEmpty():boolean
getPrisionerosSeleccionados():List<Prisionero>
getHerramientasOcultas():Map<String,Integer>
<<Java Class>>
SolucionEquipoDeFugasImpl
SolucionEquipoDeFugasImpl(List<Prisionero>,Map<String,Integer>)
<<Java Interface>>
EstadoVZ<S,A>
<<Java Interface>>
EstadoRyP<S,A>
<<Java Interface>>
ProblemaEquipoDeFugas
getPrisioneros():List<Prisionero>
getHerramientasNecesarias():Map<String,Integer>
getLimiteDeFugados():Integer
<<Java Interface>>
Iterable<T>
iterator():Iterator<T>
<<Java Interface>>
Iterator<E>
hasNext():boolean
next():E
remove():void
<<Java Interface>>
ProblemaRyP<S,A,E>
esSolucion(S):boolean
getEstadoInicial()
isAlternativa(E):boolean
getAlternativas(E):Iterable<A>
<<Java Class>>
ProblemaEquipoDeFugasImpl
<<Java Class>>
IterablePrisionero
IterablePrisionero(EstadoEquipo)
<<Java Interface>>
Prisionero
getID():Integer
getHerramientasEscondidas():Map<String,Integer>
isSeleccionado():Boolean
setSeleccionado(boolean):void
<<Java Class>>
PrisioneroImpl
PrisioneroImpl(Prisionero)
PrisioneroImpl(Prisionero,boolean)
PrisioneroImpl()

Solución:

```
public class ProblemaEquipoDeFugasImpl implements ProblemaEquipoDeFugas,
        ProblemaRyP<SolucionEquipoDeFugas, Prisionero, EstadoEquipo>,
        AlgoritmoConUnaSolucion<SolucionEquipoDeFugas> {

 private void ryp() {
    SolucionEquipoDeFugas solucion = estado.getSolucion();
    if (problema.esSolucion(solucion)) {
        soluciones.add(solucion);
        fin = esFin();
    }
    if (!fin && problema.isAlternativa(estado)) {
        Iterable<Prisionero> it = problema.getAlternativas(estado);
        for (Prisionero alternativa : it) {
            estado.add(alternativa);
            ryp();
            estado.remove();
        }
    }
 }

 public boolean isAlternativa(EstadoEquipo e) {
    return e.getSize() < this.getPrisioneros().size();
 }

 public EstadoEquipo getEstadoInicial() {
    return new EstadoEquipoImpl();
 }

 public boolean esSolucion(SolucionEquipoDeFugas s) {
    boolean solucion = false;
    if (s.getPrisionerosSeleccionados().size() <= this.getLimiteDeFugados()) {
        solucion = true;
        Map<String,Integer> ho = s.getHerramientasOcultas();
        for (Map.Entry<String, Integer> e: this.getHerramientasNecesarias().entrySet()) {
            Integer h = ho.get(e.getKey());
             if (h==null || h<e.getValue()) { solucion = false; break; }
        }
    }
    return solucion;
 }

 public Iterable<Prisionero> getAlternativas(EstadoEquipo e) {
        return new IterablePrisionero(e);
 }

}


public class EstadoEquipoImpl implements EstadoEquipo {
 private Integer numeroPrisionerosSeleccionados = null;
 private Map<String,Integer> herramientasDisponibles = null;
 private List<Prisionero> prisionerosProcesados = null;


 public boolean add(Prisionero prisionero) {
    //System.out.println("Add: " + reina);
    this.prisionerosProcesados.add(prisionero);
```

```
    if (prisionero.isSeleccionado()) {
       Map<String,Integer> he = prisionero.getHerramientasEscondidas();
       for(String h: he.keySet()) {
          addHerramientasDisponibles(h, he.get(h));
       }
       this.numeroPrisionerosSeleccionados++;
    }
    return true;
 }


 public Prisionero remove() {
    Prisionero prisionero = this.prisionerosProcesados.remove(
                                this.prisionerosProcesados.size()-1);
    if (prisionero.isSeleccionado()) {
       Map<String,Integer> he = prisionero.getHerramientasEscondidas();
       for(String h: he.keySet()) {
          removeHerramientasDisponibles(h, he.get(h));
       }
        this.numeroPrisionerosSeleccionados--;
    }
    return prisionero;
 }

 public SolucionEquipoDeFugas getSolucion() {
    return new SolucionEquipoDeFugasImpl(
          this.prisionerosProcesados, herramientasDisponibles);
 }

}
```