Ejercicio 2:

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}.

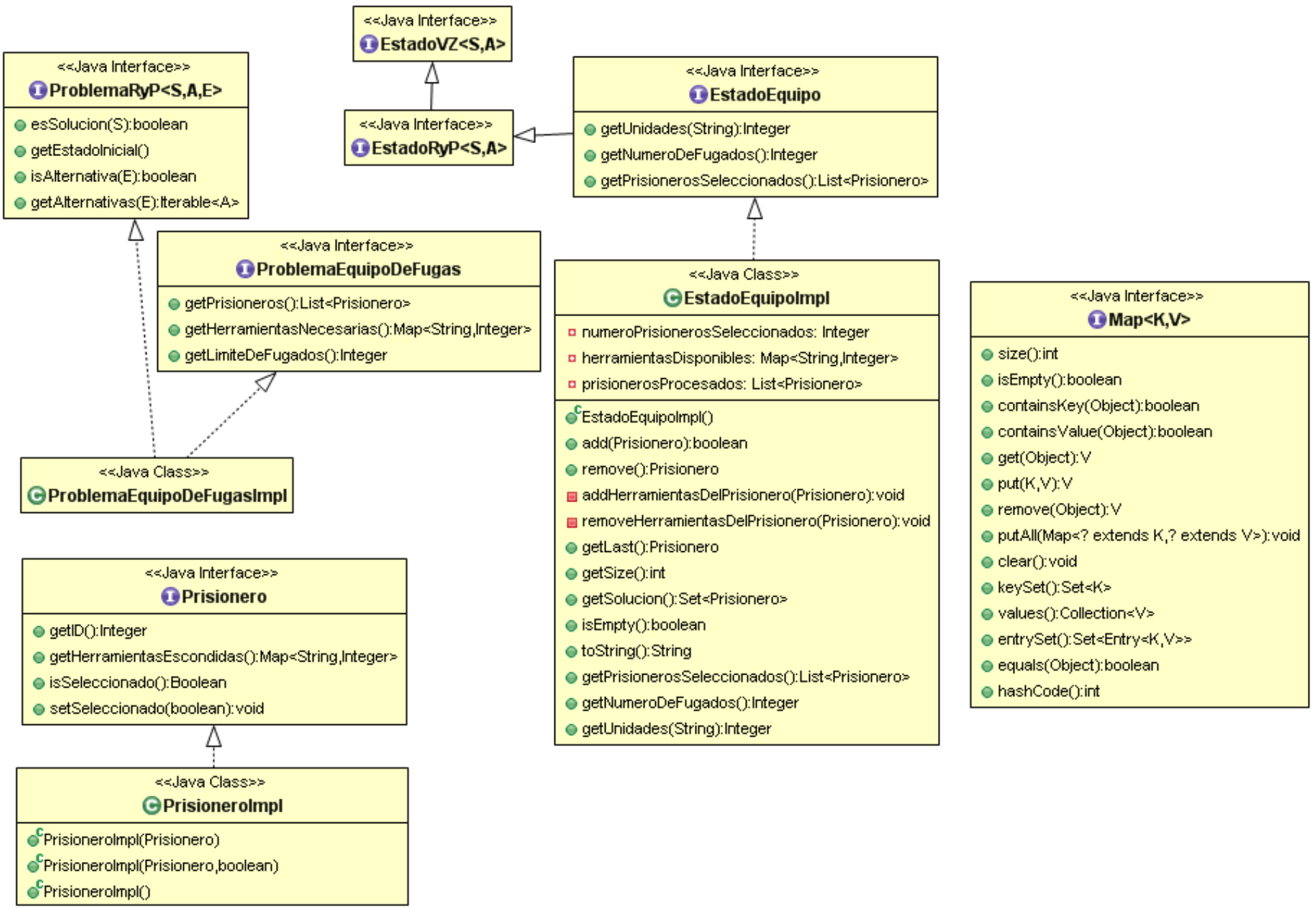
Implemente los siguientes métodos TODO con ayuda del diagrama UML.

```
public class ProblemaEquipoDeFugasImpl implements … {
 private void ryp() { // TODO }
 public boolean isAlternativa(EstadoEquipo e) { // TODO  }
 public EstadoEquipo getEstadoInicial() { // TODO  }
 // Devuelve una lista con el siguiente prisionero a procesar duplicado. En el primer
 // duplicado el prisionero estará seleccionado para fugarse y en el segundo duplicado
 // el prisionero estará seleccionado para NO fugarse.
 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.
 private Map<String,Integer> herramientasDisponibles = null;
 // Prisioneros procesados por el estado, seleccionados o no para fugarse.
 private List<Prisionero> prisionerosProcesados = null;

 // El prisionero añadido o eliminado del estado podrá o no estar seleccionado para la
 // fuga, para ello consulte el método isSeleccionado() del objeto prisionero.
 // En caso de estar seleccionado deberá añadir y eliminar las herramientas del
 // prisionero a las herramientas disponibles del estado. Así como incrementar o
 // decrementar el número de prisioneros seleccionados y actualizar la lista de
 // prisioneros procesados.
 public boolean add(Prisionero prisionero) { // TODO  }
 public Prisionero remove() { // TODO }
 // Devuelve un conjunto con los prisionero seleccionados para la fuga incluidos en
 // la lista de prisioneros procesados.
 public Set<Prisionero> getSolucion() { // TODO }
 // Añade todas las herramientas del prisionero al mapa de herramientas disponibles.
 private void addHerramientasDelPrisionero(Prisionero p) { // IMPLEMENTADA }
// Elimina todas las herramientas del prisionero del mapa de herramientas disponibles.
 private void removeHerramientasDelPrisionero(Prisionero p) { // IMPLEMENTADA }
}
```

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


<<Java Interface>>
EstadoVZ<S,A>
<<Java Interface>>
ProblemaRyP<S,A,E>
esSolucion(S):boolean
getEstadoInicial()
isAlternativa(E):boolean
getAlternativas(E):Iterable<A>
<<Java Interface>>
EstadoRyP<S,A>
<<Java Interface>>
EstadoEquipo
getUnidades(String):Integer
getNumeroDeFugados():Integer
getPrisionerosSeleccionados():List<Prisionero>
<<Java Interface>>
ProblemaEquipoDeFugas
getPrisioneros():List<Prisionero>
getHerramientasNecesarias():Map<String,Integer>
getLimiteDeFugados():Integer
<<Java Class>>
EstadoEquipoImpl
numeroPrisionerosSeleccionados: Integer
herramientasDisponibles: Map<String,Integer>
prisionerosProcesados: List<Prisionero>
EstadoEquipoImpl()
add(Prisionero):boolean
remove():Prisionero
addHerramientasDelPrisionero(Prisionero):void
removeHerramientasDelPrisionero(Prisionero):void
getLast():Prisionero
getSize():int
getSolucion():Set<Prisionero>
isEmpty():boolean
toString():String
getPrisionerosSeleccionados():List<Prisionero>
getNumeroDeFugados():Integer
getUnidades(String):Integer
<<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 Class>>
ProblemaEquipoDeFugasImpl
<<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<Set<Prisionero>> {

 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 Iterable<Prisionero> getAlternativas(EstadoEquipo e) {
    List<Prisionero> selecciones = Lists.newArrayListWithExpectedSize(2);
    Prisionero prisionero = prisioneros.get(estado.getSize());
    selecciones.add(new PrisioneroImpl(prisionero, true));
    selecciones.add(new PrisioneroImpl(prisionero, false));
    return selecciones;
 }

}

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()) {
        addHerramientasDelPrisionero(prisionero);
       this.numeroPrisionerosSeleccionados++;
    }
    return true;
 }




 public Prisionero remove() {
    Prisionero prisionero = this.prisionerosProcesados.remove(
                                this.prisionerosProcesados.size()-1);
    if (prisionero.isSeleccionado()) {
```

```
            removeHerramientasDelPrisionero(prisionero);
            this.numeroPrisionerosSeleccionados--;
        }
        return prisionero;
    }

    public Set<Prisionero> getSolucion() {
        Set<Prisionero> set = Sets.newHashSet();
        for(Prisionero p: prisionerosProcesados) {
            if(p.isSeccionado()) {
                set.add(p);
            }
        }
        return p;
    }

}
```