```java
package ejercicios;

import java.util.HashSet;


public class Ejercicio1 {

//  APARTADO A

    public static void apartadoA (SimpleDirectedGraph<Persona,Relacion> g, String file) {
        Graph<Persona,Relacion> vista = SubGraphView.of(g,
                x->padres(g,x).size()>1?
                        padres(g,x).stream().allMatch(y->y.anio().equals(padres(g,x).get(0).anio())
                                && y.ciudad().equals(padres(g,x).get(0).ciudad())):false,
                        null);

        GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoA" + ".gv",
                x->x.nombre() , x->" ",
                v->GraphColors.colorIf(Color.blue, Color.black, vista.containsVertex(v)),
                e->GraphColors.style(Style.solid));

        System.out.println(file + "apartadoA" + ".gv generado en " + "resultados/ejercicio1");
    }

    private static List<Persona> padres (SimpleDirectedGraph<Persona,Relacion> g,Persona p) {
        return Graphs.predecessorListOf(g, p);
    }


//  APARTADO B

    public static Set<Persona> apartadoB (SimpleDirectedGraph<Persona,Relacion> g,Persona p) {

        Graph<Persona,Relacion> vista = SubGraphView.of(g,x->ancestros(g,x,p),null);
        return vista.vertexSet();

    }

    public static Boolean ancestros (SimpleDirectedGraph<Persona,Relacion> g,Persona p,Persona v) {
        Boolean ac = false;
        if(p.equals(v)){
            ac = true;
        } else if(Graphs.vertexHasPredecessors(g, v)) {
            List<Persona> ancestros = Graphs.predecessorListOf(g, v);
            for(int i=0;i<ancestros.size();i++) {
                ac = ancestros(g,p,ancestros.get(i));
                if(ac==true)break;
            }
        }
        return ac;
    }

//  APARTADO C

    public static void apartadoC (SimpleDirectedGraph<Persona,Relacion> g,Persona p1,Persona p2) {

```

```
 72         son s = son.OTROS;
 73
 74         Set<Persona> abuelosP1 = abuelos(g,p1);
 75         Set<Persona> abuelosP2 = abuelos(g,p1);
 76
 77         if(sonHermanos(g,p1,p2)) {
 78             s = son.HERMANOS;
 79         } else if(abuelosP1.equals(abuelosP2)) {
 80             s = son.PRIMOS;
 81         }
 82
 83         System.out.println(p1.nombre() + " y " + p2.nombre() + " son " + s.name());
 84
 85     }
 86
 87     public static Boolean sonHermanos (SimpleDirectedGraph<Persona,Relacion> g,Persona
    p1,Persona p2) {
 88
 89         List<Persona> padresP1 = padres(g,p1);
 90         List<Persona> padresP2 = padres(g,p2);
 91
 92         Boolean ac = false;
 93         for (int i = 0; i<padresP1.size(); i++) {
 94             if(padresP2.contains(padresP1.get(i))) {
 95                 ac = true;
 96             }
 97         }
 98         return ac;
 99     }
100
101     public static Set<Persona> abuelos (SimpleDirectedGraph<Persona,Relacion> g,Persona p){
102
103         Set <Persona> abu = new HashSet<>();
104         List<Persona> padres = padres(g,p);
105         for (Persona per :padres) {
106             List<Persona> a = Graphs.predecessorListOf(g, per);
107             abu.addAll(a);
108         }
109         return abu;
110     }
111
112 //  APARTADO D
113
114     public static void apartadoD (SimpleDirectedGraph<Persona,Relacion> g, String file) {
115
116         Graph<Persona,Relacion> vista = SubGraphView.of(g,x->disPadres(g,x),null);
117
118         GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoD" + ".gv",
119                 x->x.nombre() , x->" ",
120                 v->GraphColors.colorIf(Color.blue, Color.black, vista.vertexSet().contains
    (v)),
121                 e->GraphColors.style(Style.solid));
122
123         System.out.println(file + "apartadoD" + ".gv generado en " +
    "resultados/ejercicio1");
124
125     }
126
127     public static Boolean disPadres (SimpleDirectedGraph<Persona,Relacion> g,Persona p) {
128
129         Boolean ac = false;
130         List<Persona> suc = Graphs.successorListOf(g, p);
```

```
131         if (suc.size()>=2) {
132             List<Persona> pad1 = Graphs.predecessorListOf(g, suc.get(0));
133             for (Persona per:suc) {
134                 if(!Graphs.predecessorListOf(g, per).equals(pad1)) {
135                     ac = true;
136                 }
137             }
138         }
139         return ac;
140     }
141 
142 //  APARTADO E
143 
144     public static void ApartadoE (Graph<Persona,Relacion> g,String file) {
145 
146         GreedyVCImpl<Persona,Relacion> vCover = new GreedyVCImpl<>(g);
147         Set<Persona> selected = vCover.getVertexCover();
148 
149         GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoE" + ".gv",
150                 x->x.nombre() , x->" ",
151                 v->GraphColors.colorIf(Color.blue, Color.black, selected.contains(v)),
152                 e->GraphColors.style(Style.solid));
153 
154         System.out.println(file + "apartadoE" + ".gv generado en " +
    "resultados/ejercicio1");
155 
156     }
157 
158 
159 
160 }
161 
```