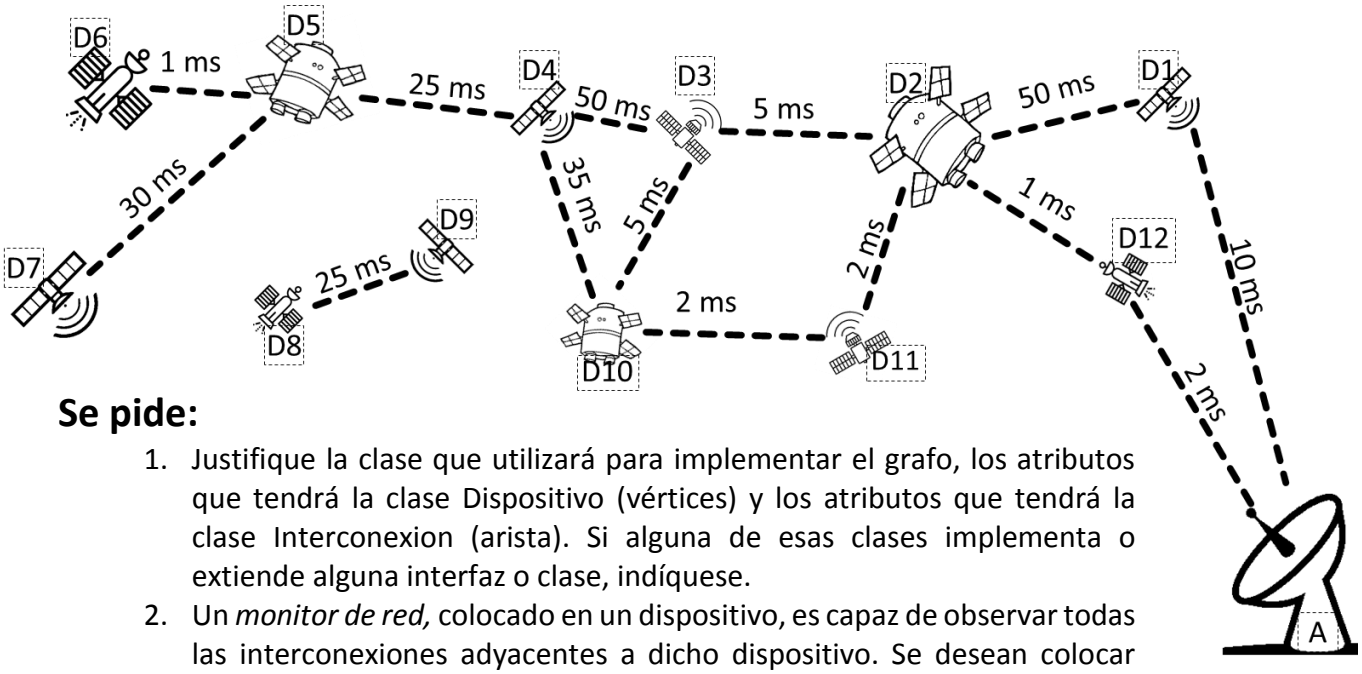
Ejercicio 4:

Se requiere desarrollar un sistema que modele un conjunto de ***Dispositivos*** que están interconectados los unos a los otros inalámbricamente. Los dispositivos tienen un *nombre* (por ejemplo, D1, A, D3). La ***Interconexión*** entre cada dos dispositivos es bidireccional y tiene una *latencia* (tiempo en establecer la comunicación entre dos dispositivos medido en ms). Además, sólo puede existir una interconexión entre cada dos dispositivos.



Se pide:

1. Justifique la clase que utilizará para implementar el grafo, los atributos que tendrá la clase Dispositivo (vértices) y los atributos que tendrá la clase Interconexion (arista). Si alguna de esas clases implementa o extiende alguna interfaz o clase, indíquese.
2. Un *monitor de red,* colocado en un dispositivo, es capaz de observar todas las interconexiones adyacentes a dicho dispositivo. Se desean colocar monitores de red en el menor número de dispositivos posible de manera que todas las interconexiones de la red puedan ser observadas. Para ello, implemente un método con la siguiente signatura:

Set<Dispositivo> getMonitorizaciónMenor(Graph<Dispositivo,Interconexion> red)

Para el ejemplo de la figura, una posible solución sería: [D2, D5, D10, D13, D8, D4]

3. Se debe conocer cuál es la ruta óptima para realizar las comunicaciones entre los dispositivos, es decir, aquella ruta que acumule menos latencia. Para ello, implemente un método que dados dos dispositivos, devuelva la lista de dispositivos que suponga la ruta con menor latencia acumulada entre los dos. Deberá seguir la siguiente signatura:

List<Dispositivo> getRutaMenorLatencia(
Graph<Dispositivo,Interconexion> red, Dispositivo d1, Dispositivo d2)

Para el ejemplo de la figura, la ruta menor entre D6 y D3 sería [D6, D5, D4, D10, D3] con una latencia acumulada de 66 ms.

4. Implemente un método que devuelva cuántos grupos aislados forman lo dispositivos. Deberá seguir la siguiente signatura:

int getGruposAislados(Graph<Dispositivo,Interconexion> red)

Para el ejemplo de la figura, hay 2 grupos aislados.

5. El dispositivo *antena*, etiquetado como 'A' en el ejemplo, es el que permite gestionar los datos de la red desde tierra. Se desea conocer qué dispositivos son accesibles desde la antena. Para ello, implemente un método con la siguiente signatura:

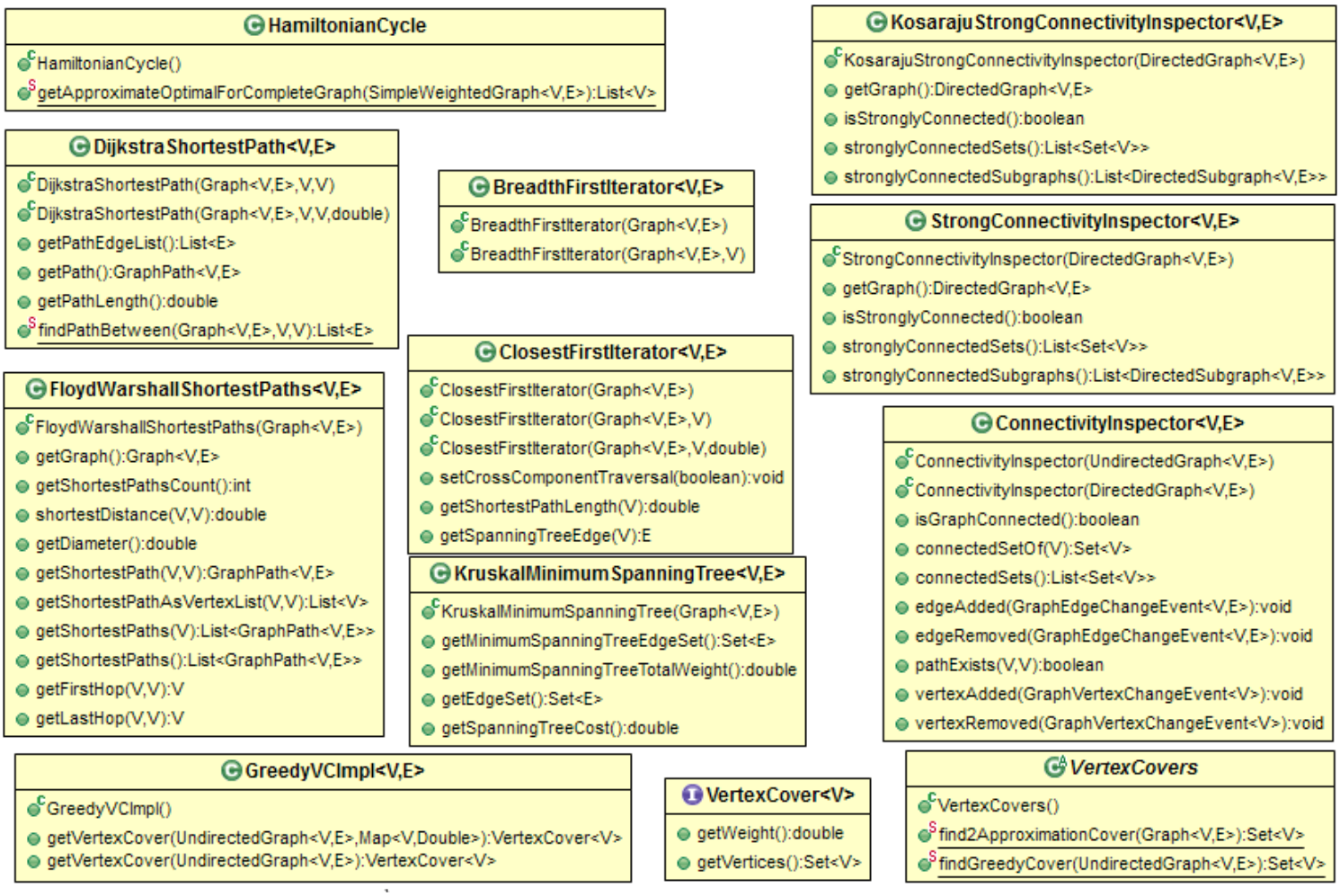
Set<Dispositivo> getAccesibles(Graph<Dispositivo,Interconexion> red)

Para el ejemplo de la figura, serían accesibles:[A, D1, D2, D3, D4, D5, D6, D7, D10, D11, D12]

6. Para ahorrar batería, los dispositivos pueden apagar las interconexiones que deseen. Cree un método que indique cuál es el conjunto de interconexiones que pueden apagarse para que, sin romper la conectividad de la red, la suma de las latencias de las interconexiones a mantener sea mínima. Siga la siguiente signatura:

Set<Interconexion> getInterconexionesInnecesarias(Graph<Dispositivo,Interconexion> red)

Para el ejemplo de la figura, las interconexiones innecesarias serían: [(D3 : D10), (D3 : D4), (D1 : D2)]



Sol

1.

- SimpleWeightedGraph
- Dispositivo: nombre
- Interconexión: latencia, además extiende de DefaultWeightedEdge

2. (Directamente VertexCovers)

```
public static Set<Dispositivo> getMonitorizacionMenor(Graph<Dispositivo,
                                                    Interconexion> red){
      return new GreedyVCImpl<Dispositivo, Interconexion>()
            .getVertexCover((UndirectedGraph<Dispositivo, Interconexion>)red)
            .getVertices();
}
```

Opción "deprecated" válida:

```
public static Set<Dispositivo> getMonitorizacionMenor(Graph<Dispositivo,
                                                    Interconexion> red){
      return VertexCovers.
            .find2ApproximationCover(red);
}
```

3. (Directamente Dijkstra)

```
public static List<Dispositivo> getRutaMenorLatencia(Graph<Dispositivo,
                        Interconexion> red, Dispositivo d1, Dispositivo d2){
      return new DijkstraShortestPath<>(red, d1, d2)
            .getPath()
            .getVertexList();//OK si recorre la lista de aristas
}
```

4. (Directamente ConnectivityInspector)

```
public static int getGruposAislados(Graph<Dispositivo, Interconexion> red){
            return new ConnectivityInspector<>((UndirectedGraph<Dispositivo,
                                                    Interconexion>)red)
                  .connectedSets()
                  .size();
      }
```

5. (ConnectivityInspector desde vértice)

```
public static Set<Dispositivo> getAccesibles(Graph<Dispositivo, Interconexion>
red){
      return new ConnectivityInspector<>((UndirectedGraph<Dispositivo,
                                    Interconexion>)red)
            .connectedSetOf(
                  new Dispositivo("A","A")//OK si busca la antena en el grafo
                  );
}
```

6. (El complementario a Kruskal)

```
public static Set<Interconexion> getInterconexionesInnecesarias(Graph<Dispositivo,
                                                      Interconexion> red){
      Set<Interconexion> ret= new HashSet<>(red.edgeSet());
      ret.removeAll(
            new KruskalMinimumSpanningTree<>(red)
                  .getMinimumSpanningTreeEdgeSet());

      return ret;
}
```