POLITECNICO DI TORINO

e-Lite

The jGraphT library

Tecniche di Programmazione – A.A. 2017/2018

Summary

- The JGraphT library
- Creating graphs

Introduction to jGraphT

The jGraphT library

JGraphT

- http://jgrapht.org
  - (do not confuse with jgraph.com)
- Free Java graph library that provides graph objects and algorithms
- Easy, type-safe and extensible thanks to <generics>
- Just add `jgrapht-core-1.1.0.jar` to your project

Latest News

JGraphT source code has moved from the old svn repository to github. Please send all new contributions there in the form of pull requests!

Release 0.8.3 is now available. As of 0.8.0, JGraphT now requires Java 1.6 to build and run. If you are stuck on an earlier JVM/JDK version, you can keep using older releases of JGraphT.

If you'd like to help out with project maintenance, click the Like button for our "Adopt-a-Highway" initiative on Facebook!

JGraphT Adopt-a-Highway

Like

215 people like JGraphT Adopt-a-Highway.

JGraphT structure

Packages	
org.jgrapht	The front-end API's interfaces and classes, including Graph, DirectedGraph and UndirectedGraph.
org.jgrapht.alg.*	Algorithms provided with JGraphT.
org.jgrapht.event	Event classes and listener interfaces, used to provide a change notification mechanism on graph modification events.
org.jgrapht.generate	Generators for graphs of various topologies.
org.jgrapht.graph	Implementations of various graphs.
org.jgrapht.traverse	Graph traversal means.

http://jgrapht.org/javadoc/

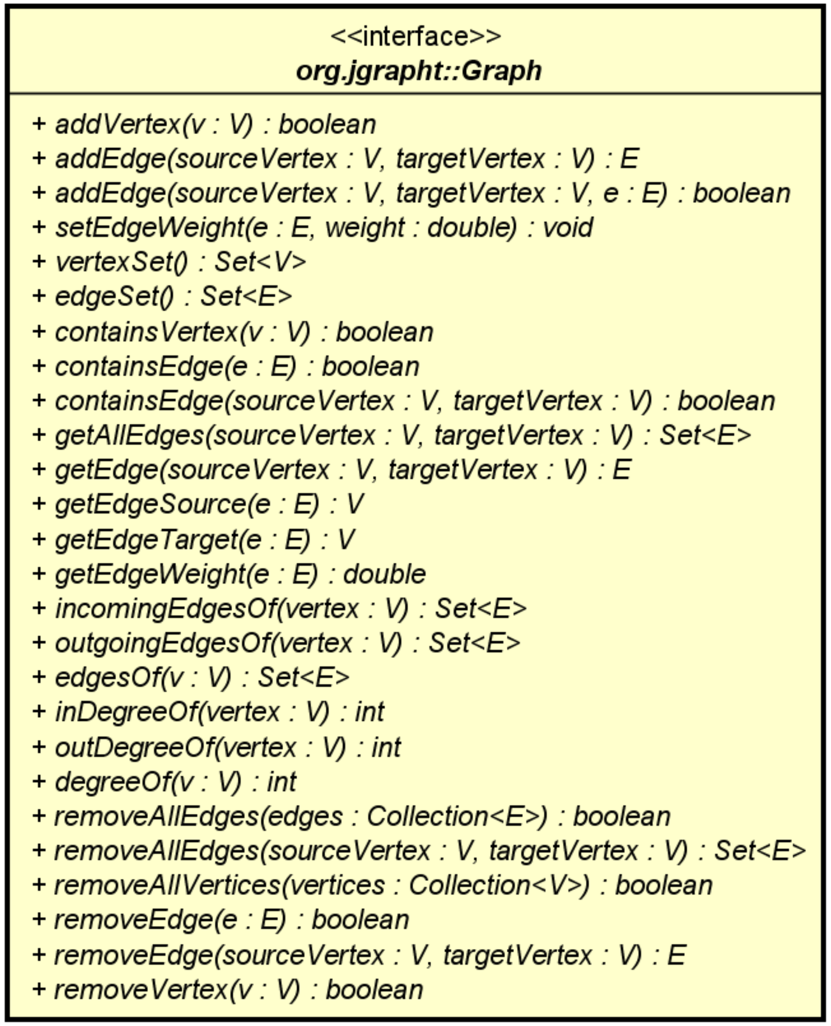
Graph objects

- All graphs derive from:
  - Interface *org.jgrapht.*`Graph<V,E>`
- V = type of vertices
  - Any class
- E = type of edges
  - *org.jgrapht.graph.*`DefaultEdge`
  - *org.jgrapht.graph.*`DefaultWeightedEdge`
  - Your own custom subclass

<V, E>

- User-defined objects, depending on the problem
- Must properly define hashCode and equals
  - The Graph implementation and many graph algorithms use HashSet and HashMap internally!
- Vertex type V
  - Your own object
  - Define hashCode and equals
- Edge type E
  - Subclass of DefaultEdge or DefaultWeightedEdge
  - Do not redefine (override) the provided hashCode and equals

What is a Graph?



```
<<interface>>
org.jgrapht::Graph

+ addVertex(v : V) : boolean
+ addEdge(sourceVertex : V, targetVertex : V) : E
+ addEdge(sourceVertex : V, targetVertex : V, e : E) : boolean
+ setEdgeWeight(e : E, weight : double) : void
+ vertexSet() : Set<V>
+ edgeSet() : Set<E>
+ containsVertex(v : V) : boolean
+ containsEdge(e : E) : boolean
+ containsEdge(sourceVertex : V, targetVertex : V) : boolean
+ getAllEdges(sourceVertex : V, targetVertex : V) : Set<E>
+ getEdge(sourceVertex : V, targetVertex : V) : E
+ getEdgeSource(e : E) : V
+ getEdgeTarget(e : E) : V
+ getEdgeWeight(e : E) : double
+ incomingEdgesOf(vertex : V) : Set<E>
+ outgoingEdgesOf(vertex : V) : Set<E>
+ edgesOf(v : V) : Set<E>
+ inDegreeOf(vertex : V) : int
+ outDegreeOf(vertex : V) : int
+ degreeOf(v : V) : int
+ removeAllEdges(edges : Collection<E>) : boolean
+ removeAllEdges(sourceVertex : V, targetVertex : V) : Set<E>
+ removeAllVertices(vertices : Collection<V>) : boolean
+ removeEdge(e : E) : boolean
+ removeEdge(sourceVertex : V, targetVertex : V) : E
+ removeVertex(v : V) : boolean
```

Graph classes

org.jgrapht

I Graph

org.jgrapht.graph

DefaultDirectedGraph

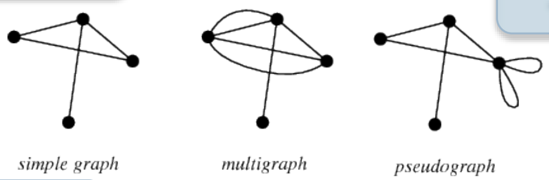
DefaultDirectedWeightedGraph

SimpleGraph

SimpleWeightedGraph

SimpleDirectedGraph

SimpleDirectedWeightedGraph



DirectedMultigraph

Multigraph

DirectedWeightedMultigraph

WeightedMultigraph

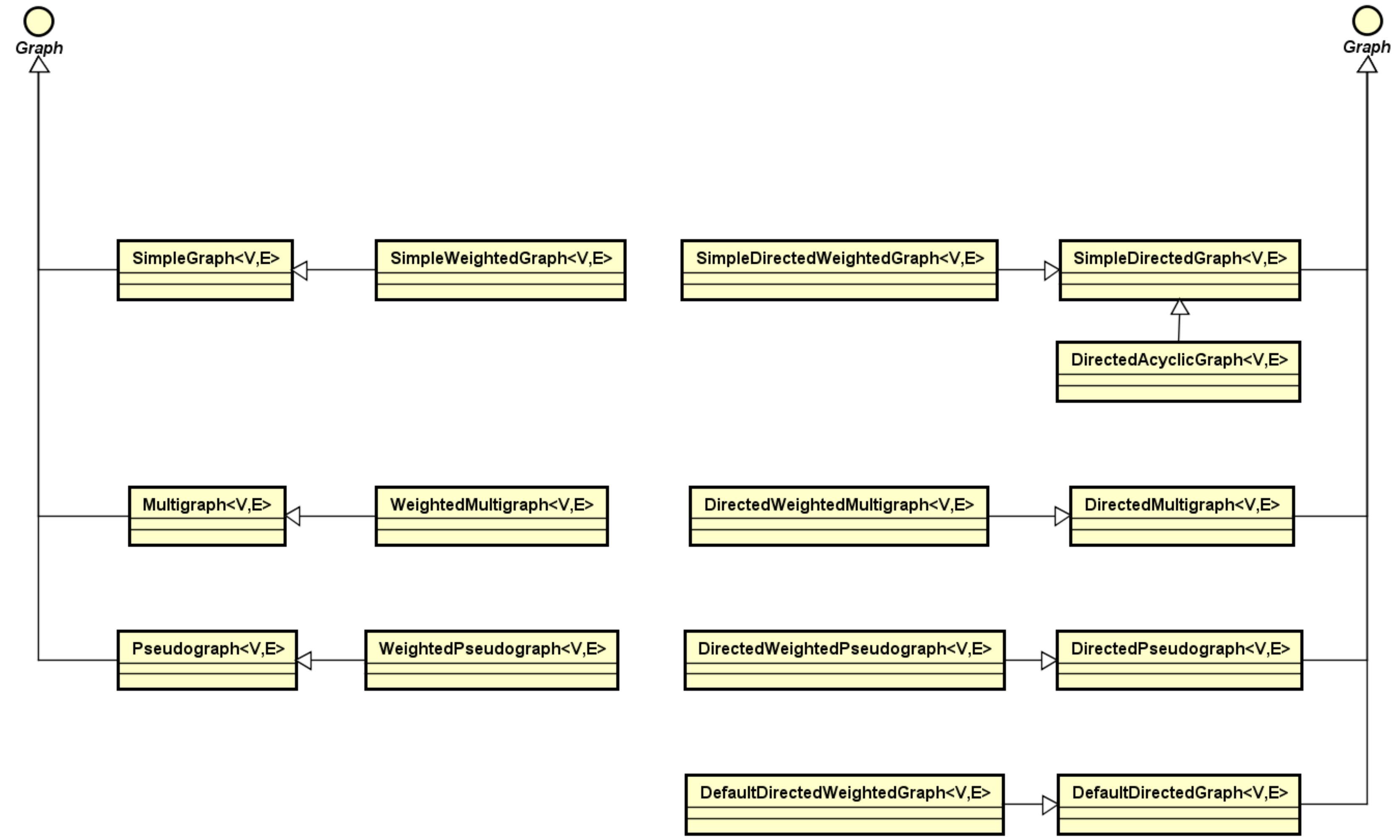
DirectedPseudograph

DirectedWeightedPseudograph

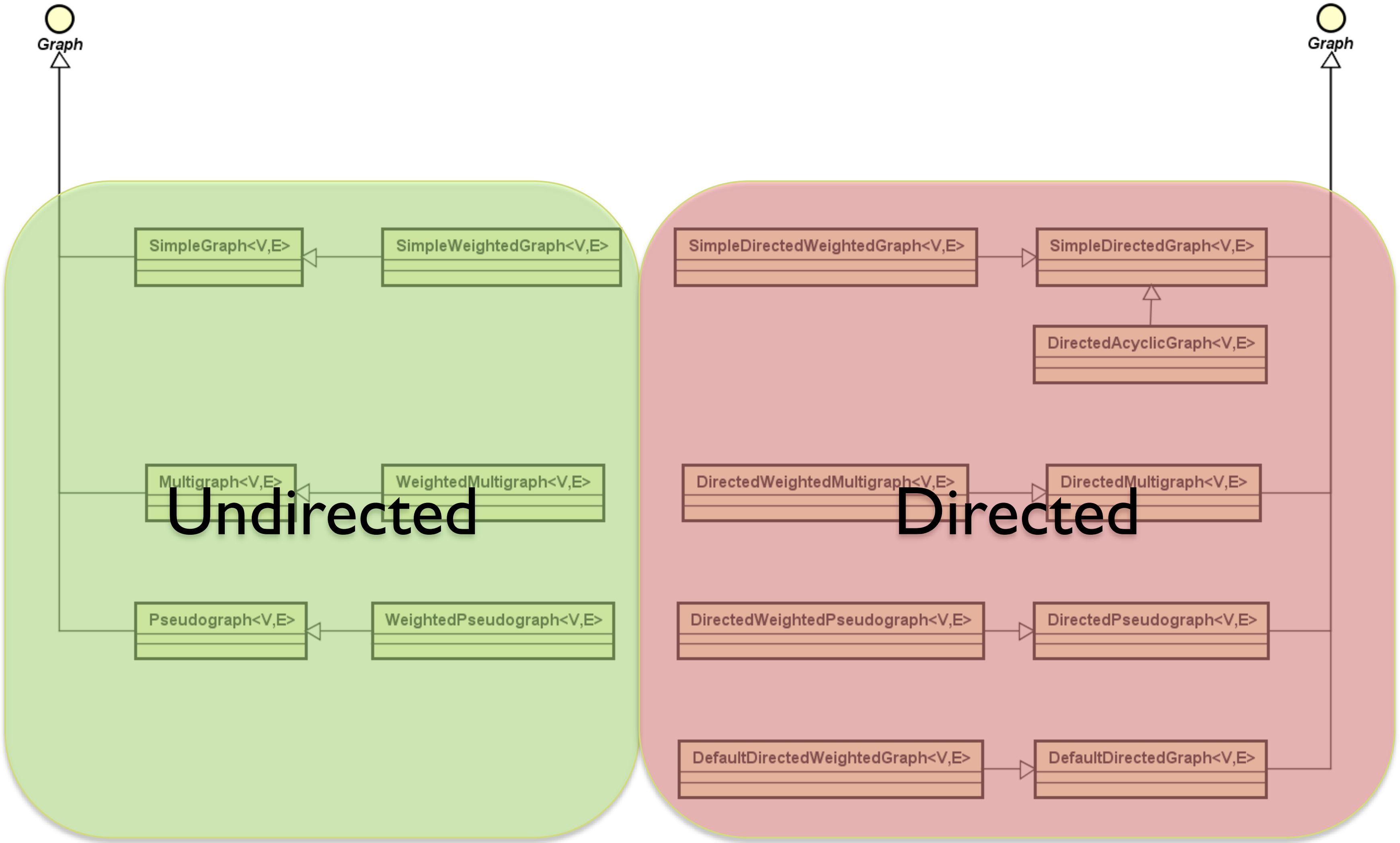
Pseudograph

WeightedPseudograph

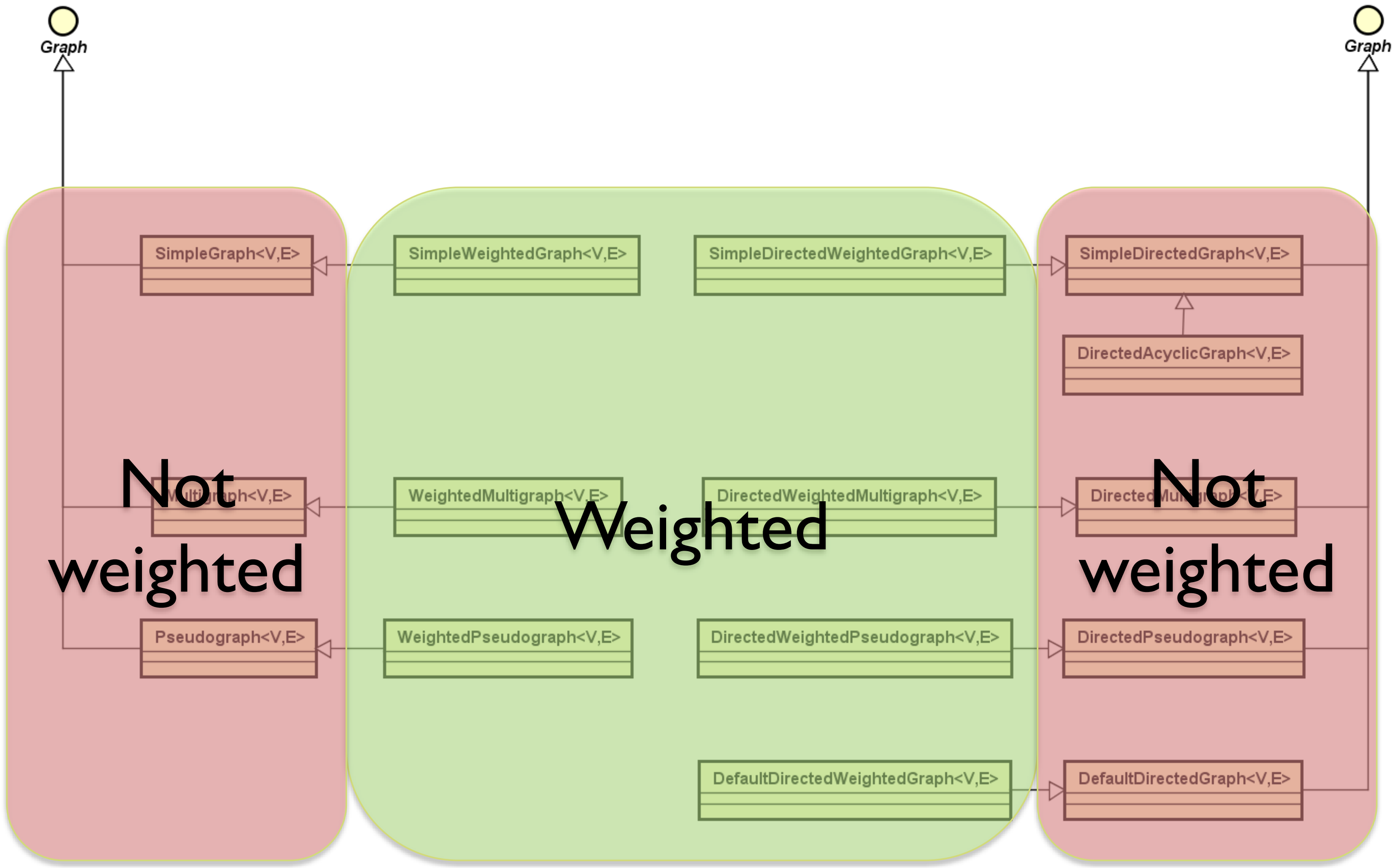
Graph classes (in `org.jgrapht.graph`)



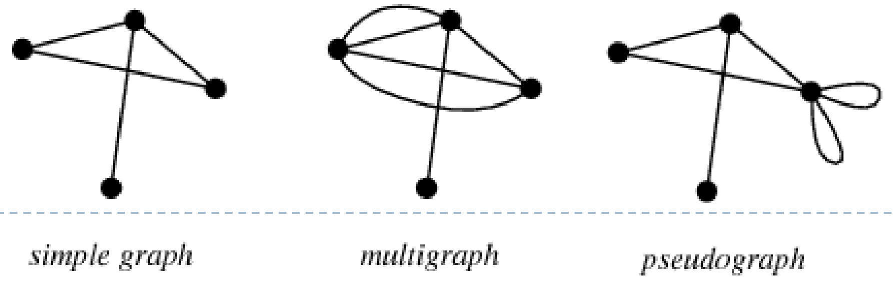
Graph classes

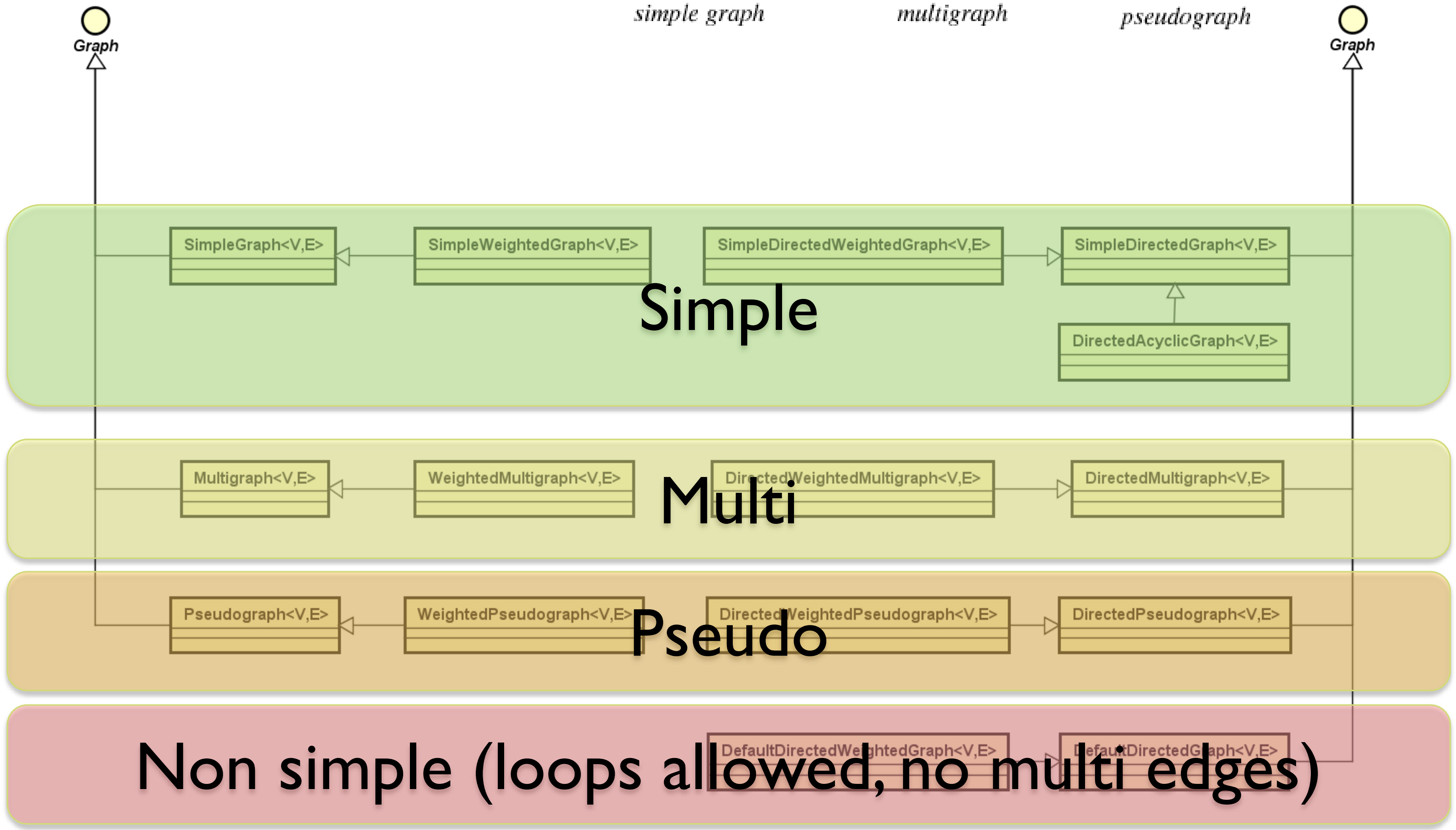


Graph classes



Graph classes

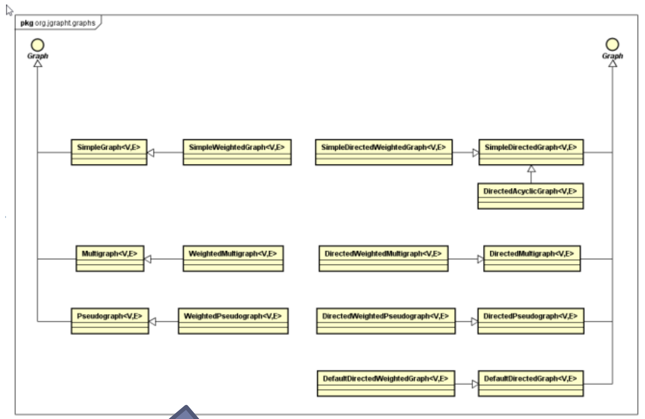



Graph
Graph
SimpleGraph<V,E>
SimpleWeightedGraph<V,E>
SimpleDirectedWeightedGraph<V,E>
SimpleDirectedGraph<V,E>
DirectedAcyclicGraph<V,E>
Simple
Multigraph<V,E>
WeightedMultigraph<V,E>
DirectedWeightedMultigraph<V,E>
DirectedMultigraph<V,E>
Multi
Pseudograph<V,E>
WeightedPseudograph<V,E>
DirectedWeightedPseudograph<V,E>
DirectedPseudograph<V,E>
Pseudo
DefaultDirectedWeightedGraph<V,E>
DefaultDirectedGraph<V,E>
Non simple (loops allowed, no multi edges)

Creating graphs

The jGraphT library

Creating graphs (1/2)



- Decide what is the vertex class V
- Decide which graph class suits your needs
  - For unweighted graphs, use DefaultEdge as E
  - For weighted graphs, use DefaultWeightedEdge as E
- Create the graph object

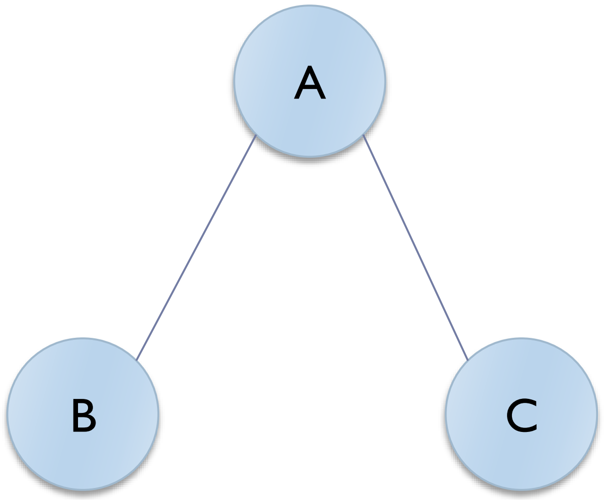
  -
    ```
    Graph<V,E> graph = new
    SimpleGraph<V,E>(E.class) ;
    ```

Creating graphs (2/2)

- Add vertices
  - boolean **addVertex**(V v)
- Add edges
  - E **addEdge**(V sourceVertex, V targetVertex)
  - boolean **addEdge**(V sourceVertex, V targetVertex, E e)
  - void **setEdgeWeight**(E e, double weight)
- Print graph (for debugging)
  - toString()
- Remember: E and V should correctly implement .equals() and .hashCode()

Example

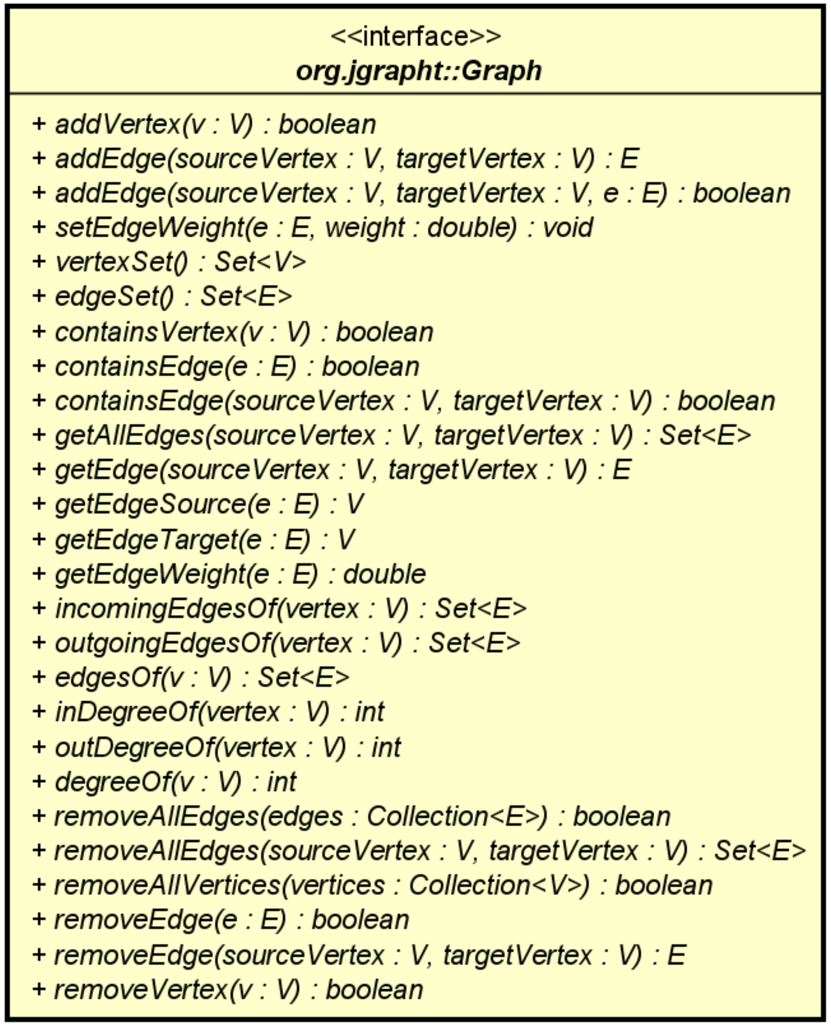
```
UndirectedGraph<String, DefaultEdge> graph = new
SimpleGraph<>(DefaultEdge.class) ;

graph.addVertex("A") ;
graph.addVertex("B") ;
graph.addVertex("C") ;

graph.addEdge("A", "B") ;
graph.addEdge("A", "C") ;
```



Querying graph structure

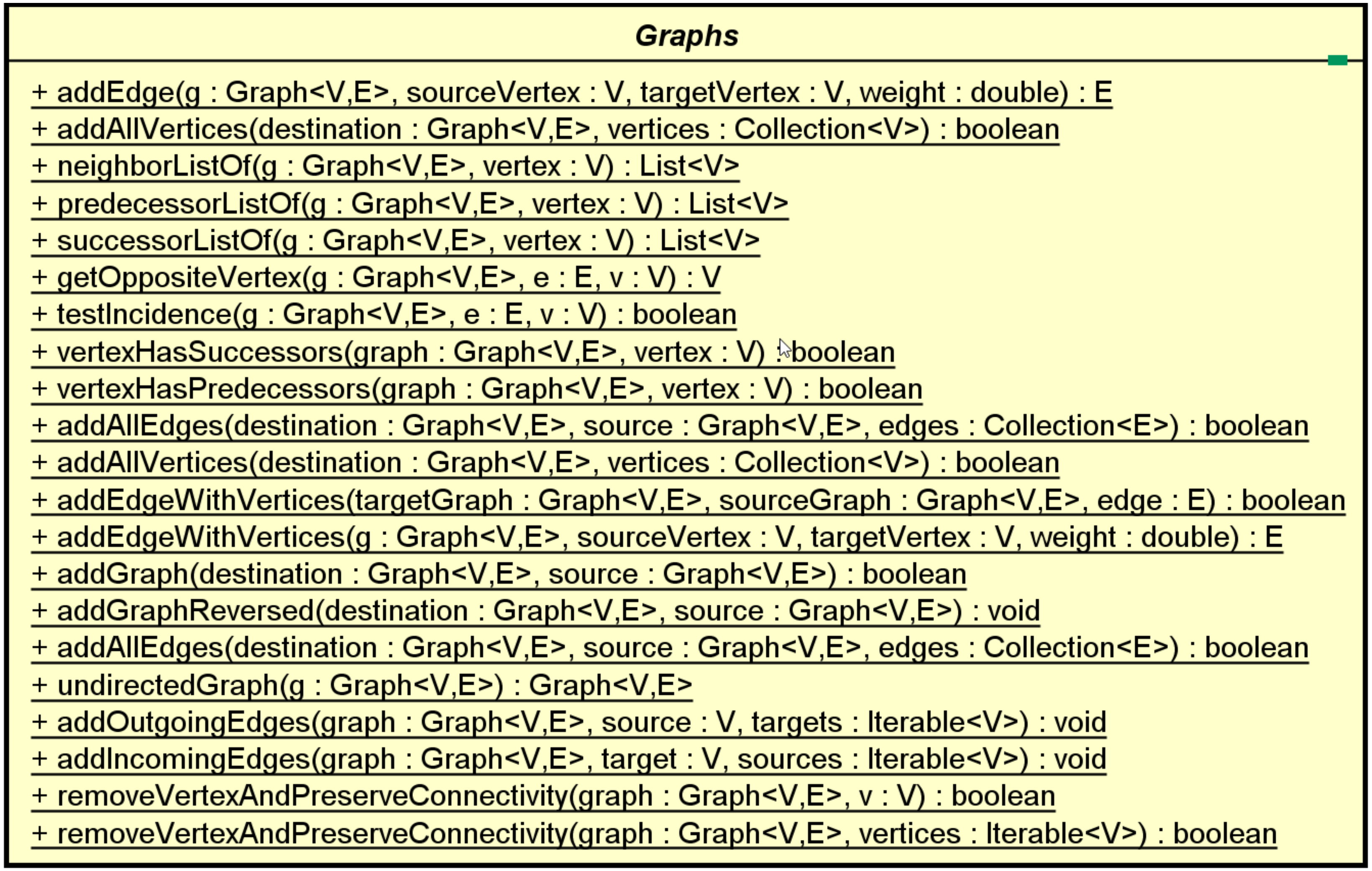
- Navigate structure
  - java.util.Set<V> **vertexSet**()
  - boolean **containsVertex**(V v)
  - boolean **containsEdge**(V sourceVertex, V targetVertex)
  - java.util.Set<E> **edgesOf**(V vertex)
  - java.util.Set<E> **getAllEdges**(V sourceVertex, V targetVertex)
- Query Edges
  - V **getEdgeSource**(E e)
  - V **getEdgeTarget**(E e)
  - double **getEdgeWeight**(E e)

Graph manipulation functions



```
<<interface>>
org.jgrapht::Graph

+ addVertex(v : V) : boolean
+ addEdge(sourceVertex : V, targetVertex : V) : E
+ addEdge(sourceVertex : V, targetVertex : V, e : E) : boolean
+ setEdgeWeight(e : E, weight : double) : void
+ vertexSet() : Set<V>
+ edgeSet() : Set<E>
+ containsVertex(v : V) : boolean
+ containsEdge(e : E) : boolean
+ containsEdge(sourceVertex : V, targetVertex : V) : boolean
+ getAllEdges(sourceVertex : V, targetVertex : V) : Set<E>
+ getEdge(sourceVertex : V, targetVertex : V) : E
+ getEdgeSource(e : E) : V
+ getEdgeTarget(e : E) : V
+ getEdgeWeight(e : E) : double
+ incomingEdgesOf(vertex : V) : Set<E>
+ outgoingEdgesOf(vertex : V) : Set<E>
+ edgesOf(v : V) : Set<E>
+ inDegreeOf(vertex : V) : int
+ outDegreeOf(vertex : V) : int
+ degreeOf(v : V) : int
+ removeAllEdges(edges : Collection<E>) : boolean
+ removeAllEdges(sourceVertex : V, targetVertex : V) : Set<E>
+ removeAllVertices(vertices : Collection<V>) : boolean
+ removeEdge(e : E) : boolean
+ removeEdge(sourceVertex : V, targetVertex : V) : E
+ removeVertex(v : V) : boolean
```

The Graphs utility class



Graphs

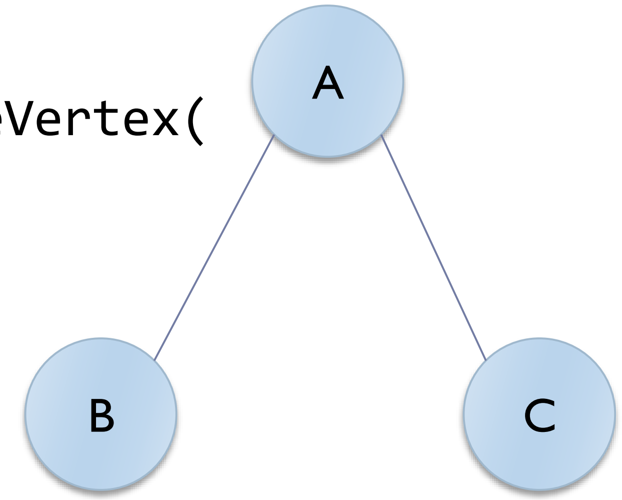
```
+ addEdge(g : Graph<V,E>, sourceVertex : V, targetVertex : V, weight : double) : E
+ addAllVertices(destination : Graph<V,E>, vertices : Collection<V>) : boolean
+ neighborListOf(g : Graph<V,E>, vertex : V) : List<V>
+ predecessorListOf(g : Graph<V,E>, vertex : V) : List<V>
+ successorListOf(g : Graph<V,E>, vertex : V) : List<V>
+ getOppositeVertex(g : Graph<V,E>, e : E, v : V) : V
+ testIncidence(g : Graph<V,E>, e : E, v : V) : boolean
+ vertexHasSuccessors(graph : Graph<V,E>, vertex : V) : boolean
+ vertexHasPredecessors(graph : Graph<V,E>, vertex : V) : boolean
+ addAllEdges(destination : Graph<V,E>, source : Graph<V,E>, edges : Collection<E>) : boolean
+ addAllVertices(destination : Graph<V,E>, vertices : Collection<V>) : boolean
+ addEdgeWithVertices(targetGraph : Graph<V,E>, sourceGraph : Graph<V,E>, edge : E) : boolean
+ addEdgeWithVertices(g : Graph<V,E>, sourceVertex : V, targetVertex : V, weight : double) : E
+ addGraph(destination : Graph<V,E>, source : Graph<V,E>) : boolean
+ addGraphReversed(destination : Graph<V,E>, source : Graph<V,E>) : void
+ addAllEdges(destination : Graph<V,E>, source : Graph<V,E>, edges : Collection<E>) : boolean
+ undirectedGraph(g : Graph<V,E>) : Graph<V,E>
+ addOutgoingEdges(graph : Graph<V,E>, source : V, targets : Iterable<V>) : void
+ addIncomingEdges(graph : Graph<V,E>, target : V, sources : Iterable<V>) : void
+ removeVertexAndPreserveConnectivity(graph : Graph<V,E>, v : V) : boolean
+ removeVertexAndPreserveConnectivity(graph : Graph<V,E>, vertices : Iterable<V>) : boolean
```

Utility functions

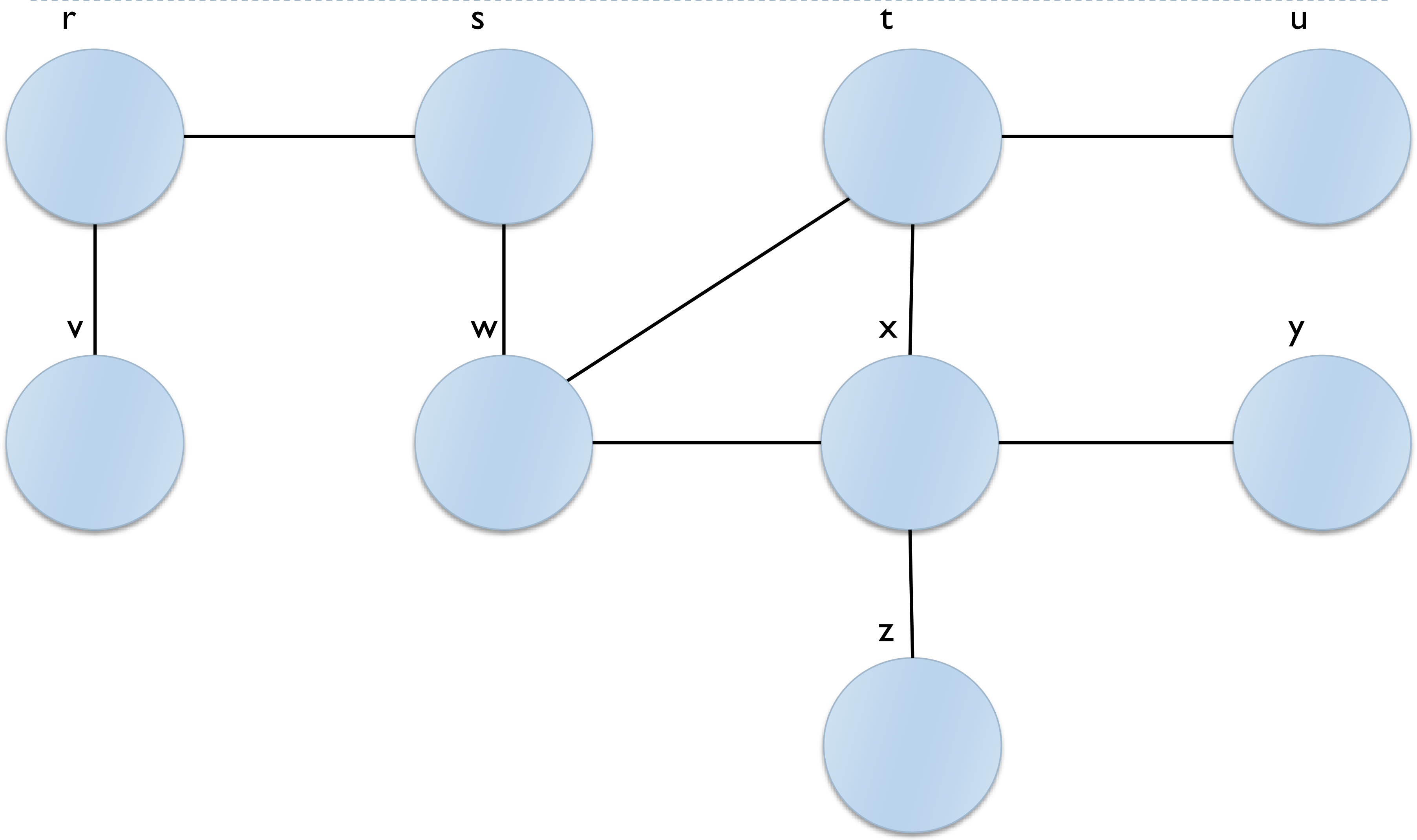
- Static class **org.jgrapht.Graphs**
- Easier creation
  - public static <V,E> E **addEdge**(Graph<V,E> g,V sourceVertex, V targetVertex, double weight)
  - public static <V,E> E **addEdgeWithVertices**(Graph<V,E> g, V sourceVertex,V targetVertex)
- Easier navigation
  - public static <V,E> java.util.List<V> **neighborListOf**(Graph<V,E> g, V vertex)
  - public static String **getOppositeVertex**(Graph<String, DefaultEdge> g, DefaultEdge e, String v)
  - public static <V,E> java.util.List<V> **predecessorListOf**(DirectedGraph<V,E> g,V vertex)
  - public static <V,E> java.util.List<V> **successorListOf**(DirectedGraph<V,E> g,V vertex)

Example

```
for( String s: graph.vertexSet() ) {
      System.out.println("Vertex "+s) ;
      for( DefaultEdge e: graph.edgesOf(s) ) {
            System.out.println("Degree: “
                  +graph.degreeOf(s)) ;
            System.out.println(
                  Graphs.getOppositeVertex(
                  graph, e, s)) ;
      }
}
```



Example



Licenza d'uso

- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera
- Alle seguenti condizioni:
  - Attribuzione — Devi attribuire la paternità dell'opera agli autori origina e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - Non commerciale — Non puoi usare quest'opera per fini commerciali.
  - Condividi allo stesso modo — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- http://creativecommons.org/licenses/by-nc-sa/3.0/