



# The jGraphT library

Tecniche di Programmazione – A.A. 2018/2019



#### Summary

- ▶ The JGraphT library
- Creating graphs

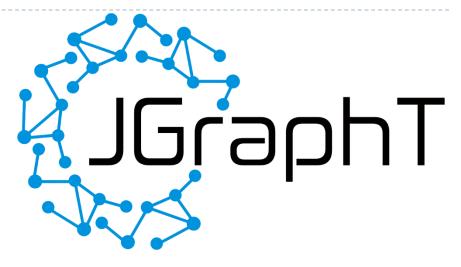


## Introduction to jGraphT

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-1.3.0.jar to your project





# JGraphT structure

Packages	
org.jgrapht	The front-end API's interfaces and classes, including <b>Graph</b> .
org.jgrapht.alg	Algorithms provided with <b>JGraphT</b> .
org.jgrapht.demo	Demo programs that help to get started with <b>JGraphT</b> .
org.jgrapht.event	Event classes and listener interfaces, used to provide a change notification mechanism on graph modification events.
org.jgrapht.ext	Extensions and integration means to other products.
org.jgrapht.generate	Generators for graphs of various topologies.
org.jgrapht.graph	Implementations of various graphs.
org.jgrapht.graph.builder	Various builder for graphs.
org.jgrapht.graph.specifics	Implementations of specifics for various graph types.
org.jgrapht.traverse	Graph traversal means.
org.jgrapht.util	Non-graph-specific data structures, algorithms, and utilities used by <b>JGraphT</b> .

Packages org.jgrapht.alg	
.clique	Clique related algorithms.
.color	Graph coloring algorithms.
.connectivity	Algorithms dealing with various connectivity aspects.
.cycle	Algorithms related to graph cycles.
.decomposition	Algorithms for computing decompositions.
.flow	Flow related algorithms.
.mincost	Algorithms for minimum cost flow
.independentset	Algorithms for Independent Set in a graph.
.interfaces	Algorithm related interfaces.
.isomorphism	Algorithms for (sub)graph isomorphism.
.lca	Algorithms for computing lowest common ancestors.
.matching	Algorithms for the computation of matchings.
.partition	Algorithm for computing partitions.
.scoring	Vertex and/or edge scoring algorithms.
.shortestpath	Shortest-path related algorithms.
.spanning	Spanning tree and spanner algorithms.
.tour	Graph tours related algorithms.
.transform	Package for graph transformers
.vertexcover	Vertex cover algorithms.

## 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>
+ contains Vertex(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

Graph

org.jgrapht.graph SimpleGraph SimpleWeightedGraph DefaultDirectedGraph SimpleDirectedGraph DefaultDirectedWeightedGraph SimpleDirectedWeightedGraph simple graph multigraph pseudograph DirectedMultigraph DirectedPseudograph Multigraph Directed Weighted Pseudograph **Pseudograph** DirectedWeightedMultigraph WeightedMultigraph WeightedPseudograph

Class Name	Edges	Self-loops	Multiple edges	Weighted
Default Undirected Weighted Graph	undirected	yes	no	yes
SimpleGraph	undirected	no	no	no
Multigraph	undirected	no	yes	no
Pseudograph	undirected	yes	yes	no
Default Undirected Graph	undirected	yes	no	no
SimpleWeightedGraph	undirected	no	no	yes
Weighted Multigraph	undirected	no	yes	yes
Weighted Pseudograph	undirected	yes	yes	yes
Default Undirected Weighted Graph	undirected	yes	no	yes
SimpleDirectedGraph	directed	no	no	no
Directed Multigraph	directed	no	yes	no
DirectedPseudograph	directed	yes	yes	no
Default Directed Graph	directed	yes	no	no
SimpleDirectedWeightedGraph	directed	no	no	yes
DirectedWeightedMultigraph	directed	no	yes	yes
DirectedWeightedPseudograph	directed	yes	yes	yes
Default Directed Weighted Graph	directed	yes	no	yes

https://jgrapht.org/guide/UserOverview#graph-structures

SimpleGraph	SimpleWeighted	SimpleDirected	SimpleDirected
	Graph	WeightedGraph	Graph
DefaultUndirected	DefaultUndirected	DefaultDirected	DefaultDirected
Graph	WeightedGraph	WeightedGraph	Graph
Multigraph	WeightedMultigraph	DirectedWeighted Multigraph	DirectedMultigraph
Pseudograph	Weighted	DirectedWeighted	Directed
	Pseudograph	Pseudograph	Pseudograph

SimpleGraph	SimpleWeighted Graph	SimpleDirected WeightedGraph	SimpleDirected Graph	
DefaultUndirected Graph	DefaultUndirected WeightedGraph	DefaultDirected WeightedGraph	DefaultDirected Graph	
Undire	ected	Directed		
Multigraph	WeightedMultigraph	DirectedWeighted Multigraph	DirectedMultigraph	
Pseudograph	Weighted Pseudograph	DirectedWeighted Pseudograph	Directed Pseudograph	

SimpleDirected **SimpleWeighted** SimpleDirected SimpleGraph WeightedGraph Graph Graph **DefaultUndirected DefaultUndirected DefaultDirected DefaultDirected** Graph Not Graph Not WeightedGraph WeightedGraph Weighted weighted weighted **DirectedWeighted** DirectedMultigraph Multigraph WeightedMultigraph Multigraph Weighted **DirectedWeighted** Directed Pseudograph Pseudograph Pseudograph Pseudograph

SimpleGraph	SimpleWeighted Graph <b>Sim</b>	SimpleDirected PleeightedGraph	SimpleDirected Graph
DefaultUndirected Graph	DefaultUndirected Simplerwith	Default Directed  Self Noops	DefaultDirected Graph
Multigraph	WeightedMultigram	DirectedWeighted  JITI Multigraph	DirectedMultigraph
Pseudograph	Pseudo (N	1 Directed Weighted 1 Light Self	Directed Pseudograph



# Creating graphs

The jGraphT library

# Creating graphs (1/2)

Decide what is the vertex class \( \bigvee \)

Class Name	Edges	Self-loops	Multiple edges	Weighted
DefaultUndirectedWeightedGraph	undirected	yes	no	yes
SimpleGraph	undirected	no	no	no
Multigraph	undirected	no	yes	no
Pseudograph	undirected	yes	yes	no
DefaultUndirectedGraph	undirected	yes	no	no
SimpleWeightedGraph	undirected	no	no	yes
WeightedMultigraph	undirected	no	yes	yes
WeightedPseudograph	undirected	yes	yes	yes
DefaultUndirectedWeightedGraph	undirected	yes	no	yes
SimpleDirectedGraph	directed	no	no	no
DirectedMultigraph	directed	no	yes	no
DirectedPseudograph	directed	yes	yes	no
DefaultDirectedGraph	directed	yes	no	no
SimpleDirectedWeightedGraph	directed	no	no	yes
DirectedWeightedMultigraph	directed	no	yes	yes
DirectedWeightedPseudograph	directed	yes	yes	yes
DefaultDirectedWeightedGraph	directed	yes	no	yes

- 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
  - For the second is a second in the second is a second in the second

#### Alternative: GraphTypeBuilder

Use the GraphTypeBuilder class, and specify the variants of the graph you need:

```
Foraph<Integer, DefaultEdge> g =
    GraphTypeBuilder
    .<Integer, DefaultEdge> undirected()
    .allowingMultipleEdges(false)
    .allowingSelfLoops(false)
    .edgeClass(DefaultEdge.class)
    .weighted(false)
    .buildGraph();
```

See:

https://jgrapht.org/javadoc/org/jgrapht/graph/builder/GraphType Builder.html

# Creating graphs (2/2)

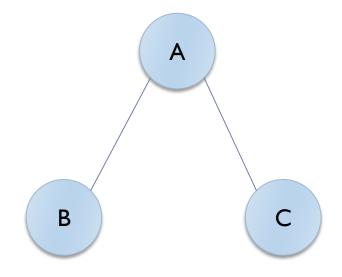
- Add vertices
  - boolean addVertex(V v)
- Add edges
  - E addEdge(V sourceVertex,V targetVertex)
  - boolean addEdge(V sourceVertex, V targetVertex, E e)
  - void setEdgeWeight(E e, double weight)
- Print graph (for debugging)
  - toString()
- Remember: E and V should correctly implement .equals()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

## Example

```
Graph<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>
+ contains Vertex(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)

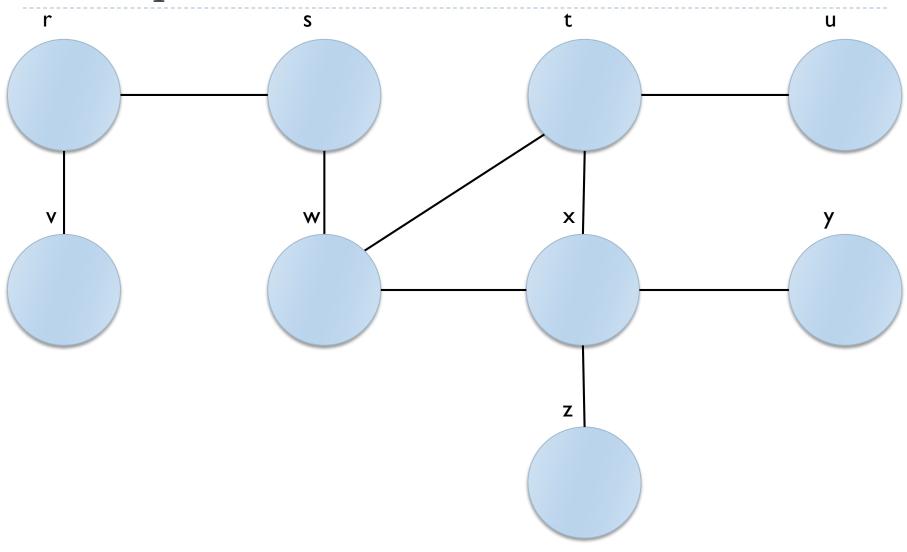
## Utility functions

- Static class org.jgrapht.Graphs
- 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(
                                                Α
                   Graphs.getOppositeVertex(
                   graph, e, s));
                                         В
```

# 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/





