IBM Carbon TreeView

Overview

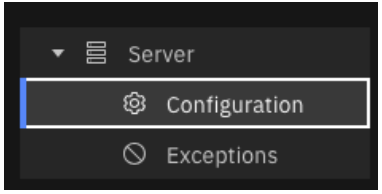
- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT**

IBM Carbon TreeView

Overview

- **Tree View … First View** (Static TreeView)
- Dynamic TreeView
- Approaches to provide TreeData dynamically
- Fundamentals of Graph Theory
- Server Sided Java-Frameworks to handle Graphs
- JGraphT

Static TreeView




```
import React, { Component } from 'react';

import { TreeView, TreeNode } from '@carbon/react';
import { BareMetalServer, Settings, Error } from '@carbon/icons-react';

render() {

   . . .

 <TreeView
   label=""
   onSelect={() => { }}
 >
    <TreeNode
      id="1"
      label="Server"
      value=""
      isExpanded={true}
      renderIcon={BareMetalServer}>

       <TreeNode
         id="1-1"
         label="Configuration"
         value=""
         renderIcon={Settings}
         onSelect={this.toggleUi.bind(this, 'config')}
       />

       <TreeNode
         id="1-2"
         label="Exceptions"
         value=""
         renderIcon={Error}
         onSelect={this.toggleUi.bind(this, 'exceptions')}
       />

    </TreeNode>

</TreeView>
```

IBM Carbon TreeView

Overview

- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT**

Dynamic TreeView

Reasons for Dynamic

- **Data displayed in the TreeView is not known at the time of UI programming**
  but is provided externally (e.g. via REST-Endpoint).

The following illustration shows **various tree structure** – typical examples:

- *Company Organization*
- Hierarchical bill of materials for a product (similar to an exploded view)
- Directory Tree (directories, subdirectories, files)

IBM Carbon's TreeView - A look ahead to dynamic behavior

The TreeView expects the following data structure

```
type TypeTreeNode = {
  id: string;
  label: string;
  children?: TreeNode[];
};

type TypeTreeNodeList = TypeTreeNode[]
```

… this can be populated via JSON in an easy way:

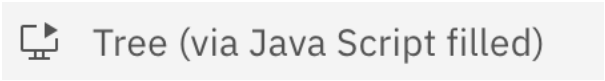
```
function captureTreeContentExample()
 {
  let data = [

  {
    id: "1",
    label: "First Node",
    children: [
        { id: "1-1", label: "Child 1"},
        { id: "1-2", label: "Child 2"},
    ],
  },

  {
    id: "2",
    label: "Second Node",
},

  ];
  return data;
}
```

Dilemma:

- It is not the task of a backend implementation (e.g. REST endpoint) to meet all the specific requirements of a front end.
- There may be many front ends on different end devices with different requirements.

IBM Carbon‘s TreeView - A look ahead to dynamic behavior

Take a look at the example:



IBM Carbon TreeView

Overview

- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT**

Dynamic Tree-Data – Approach 1: Tree-Data is per default a Tree

Example: Directory Tree

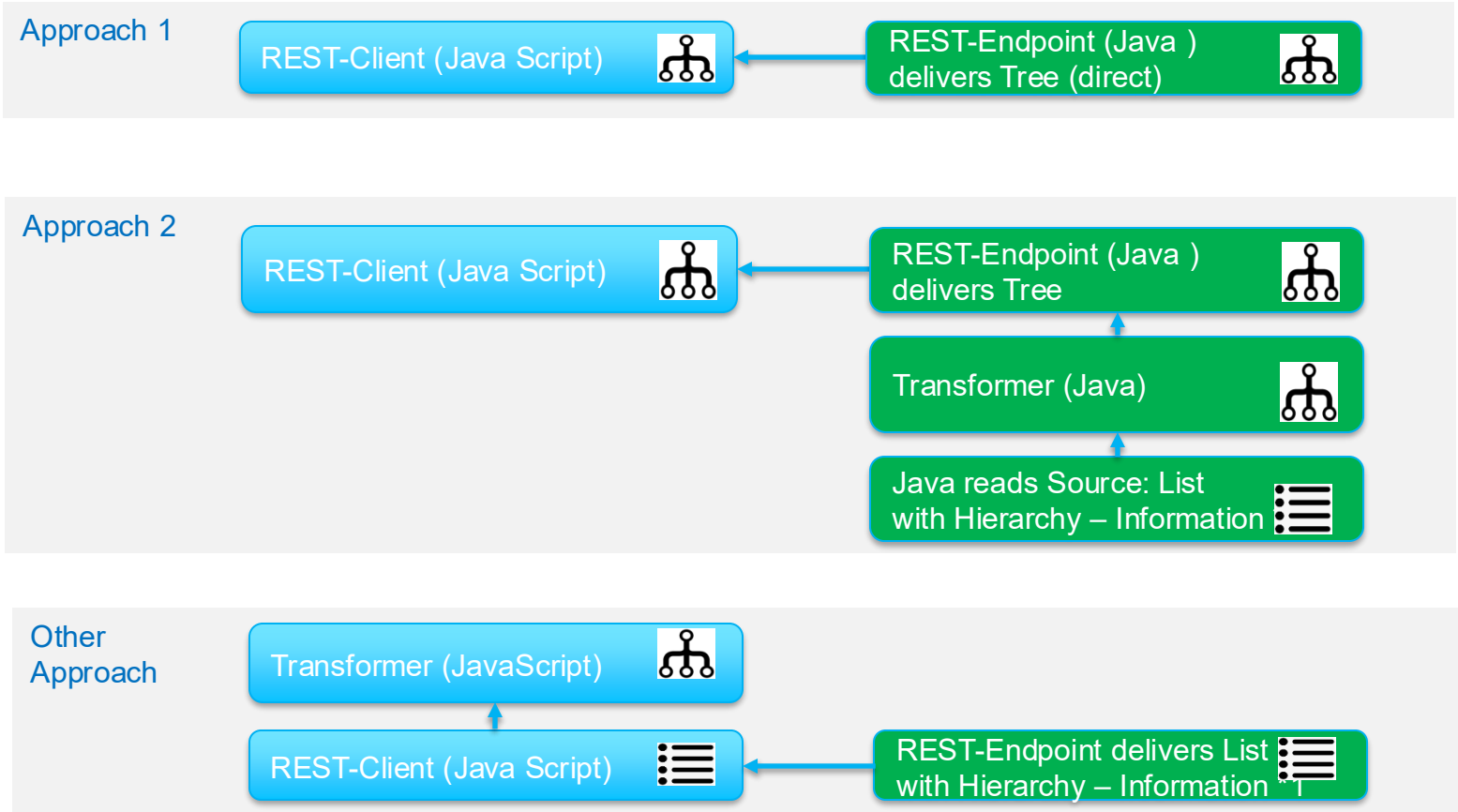
```
.
├── explain_packages
│   ├── EXAMPLE_TREE
│   │   └── placeholder.txt
│   ├── EXAMPLE_TREE_BACKEND_FILLED_1
│   │   └── placeholder.txt
│   ├── EXAMPLE_TREE_BACKEND_FILLED_2
│   │   └── placeholder.txt
│   └── EXAMPLE_TREE_SELF_FILLED
│       └── placeholder.txt
└── i18n
    └── locales
        ├── de
        │   └── translation.properties
        └── en
            └── translation.properties
```

Dynamic Tree-Data – Approach 2: Tree-Data is List with hierarchical Information

Example: Content of a relational Database

A-Z file	A-Z mimetype
2001:db8:85a3:0:0:8a2e:370:7334.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/Datei3-FolderA1.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/FolderA2/Datei3-FolderA2.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/Datei3-FolderA3.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/Datei3-FolderA4.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/FolderA6/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/FolderA6/Datei3-FolderA6.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/FolderA5/Datei3-FolderA5.txt	text/plain
Test2.zip_unzip/Test2/FolderA1/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
Test2.zip_unzip/Test2/FolderA1/FolderA2/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/FolderA4.zip_unzip/FolderA4/FolderA5/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
Test2.zip_unzip/Test2/FolderA1/FolderA2/FolderA3/OSS.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation

Dynamic Tree-Data – Approaches



IBM Carbon TreeView

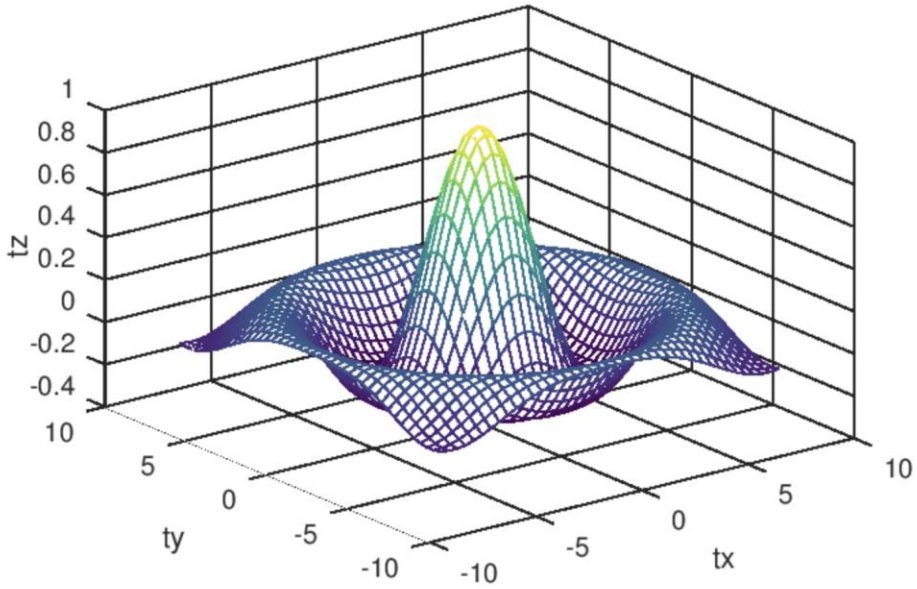
Overview

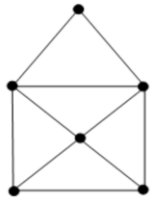
- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT**

What is a Graph

THEORY

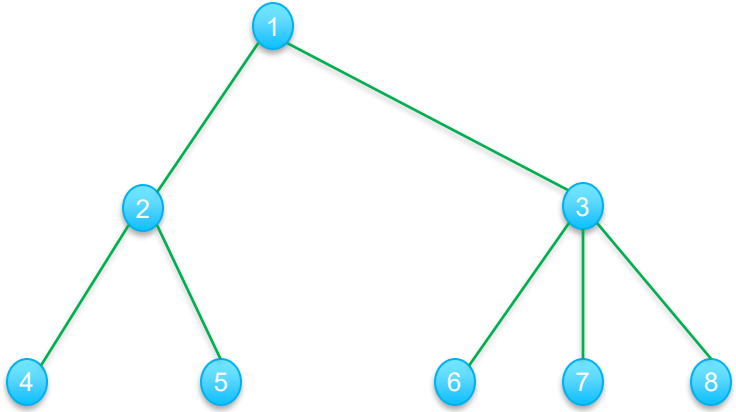
Any object consisting of Nodes and connections (= Edges)





How to understand a Tree as Graph

THEORY

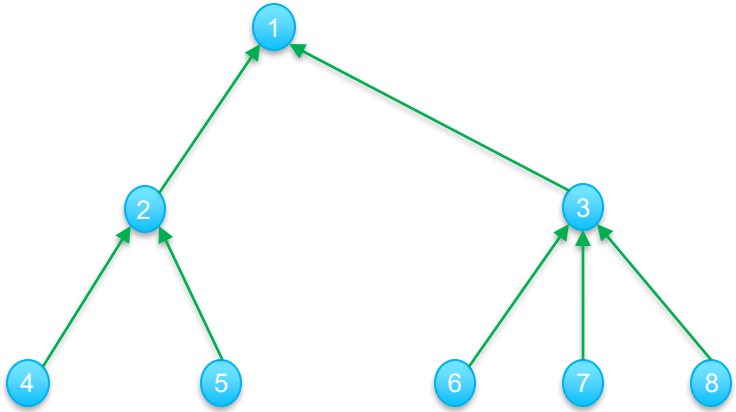


Node with an ID n

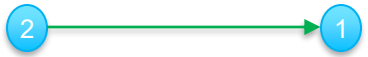
Edge

A Tree is a directed Graph

THEORY



A directed Edge

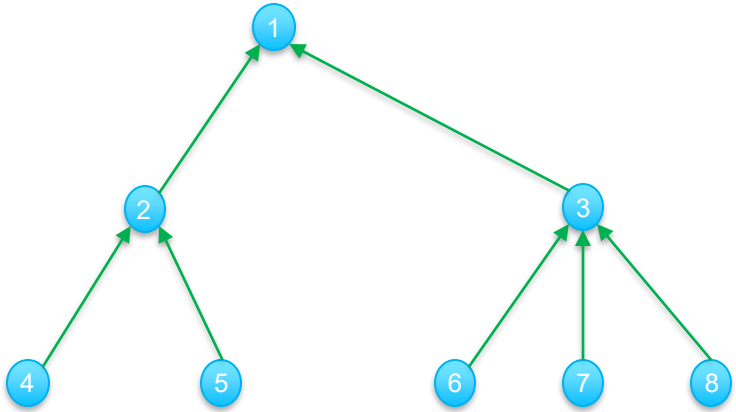


… points from one Node (2) to another Node (1)

… ParentID of (2) is (1)

A Node within a Tree can have Children[]

THEORY



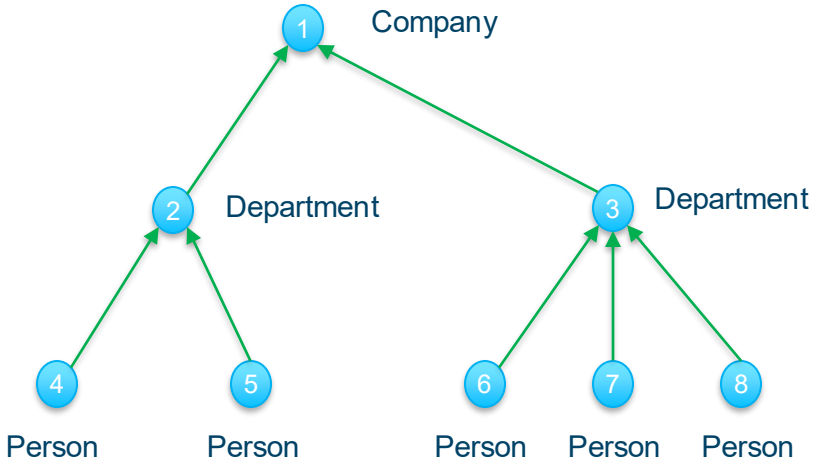
1 children[] : 2 3

2 children[] : 4 5

3 children[] : 6 7 8

Typically a Node within a Tree represents Objects of normal Life

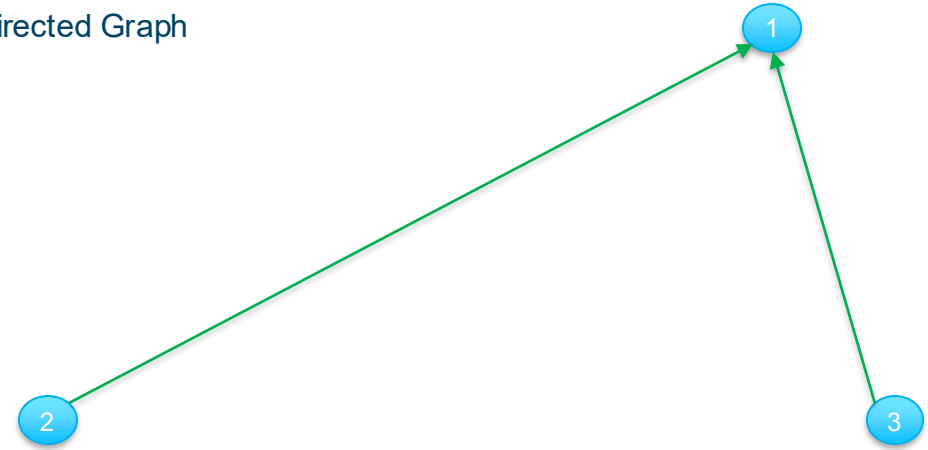
PRAXIS



What helpes: Each Objekt is defined by Attributes (we say: Payload)

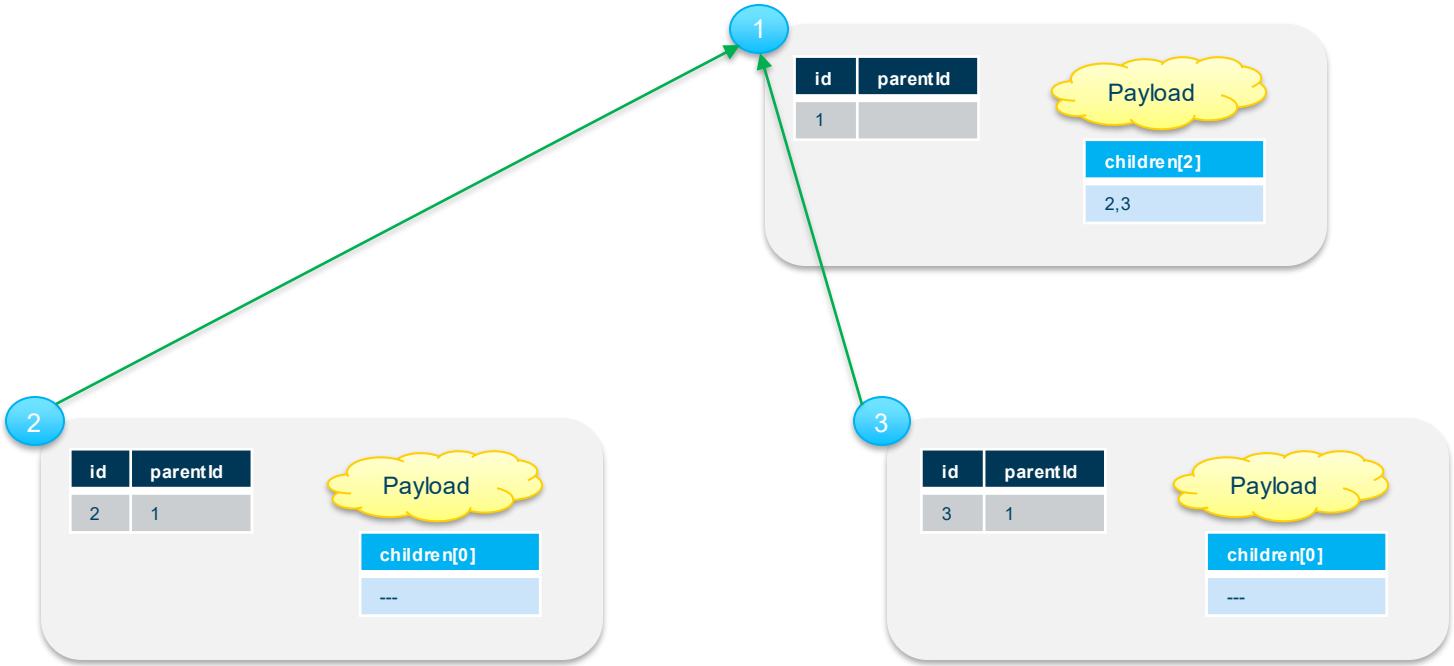
A Tree as Graph - Summary

Directed Graph



A Tree as Graph – Helpful Attributes

PRAXIS



IBM Carbon TreeView

Overview

- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT**

Available Java-Framworks to hande Graphs

ChatGPT:

License Experience

Core Java Graph Libraries

JGraphT

A robust and widely-used library offering a rich collection of graph types and advanced algorithms (shortest paths, spanning trees, matching, flows, subgraph isomorphism, NP-hard approximations, and more) arXiv. Excellent for general-purpose graph algorithm work with well-documented APIs.

EPL
LGPL

JUNG (Java Universal Network/Graph Framework)

Focused on network analysis and visualization, JUNG supports a variety of graph types, offers layout algorithms, centrality measures, clustering, and interactive exploration tools Wikipedia. Note: Its latest stable release was in 2016; a preview exists from 2020 Wikipedia.

BSD

Graph4J

A newer Java graph library focusing on performance and memory efficiency, using primitive arrays rather than object-based structures, excelling in speed and low memory overhead arXiv.

GPL

...

Selected: JGraphT

- Is established in the market (actively maintained and frequently used)
- Helps on particular Topics, to avoid a lot of manual work
    - Avoid Loops in Trees
    - Avoid unclean trees (e.g., 1 node with multiple parents)
- No License Issues
- Is flexible to handle own Java Data Structure

=> Therefore: JGraphT was selected

IBM Carbon TreeView

Overview

- **Tree View … First View** (Static TreeView)
- **Dynamic TreeView**
- **Approaches to provide TreeData dynamically**
- **Fundamentals of Graph Theory**
- **Server Sided Java-Frameworks to handle Graphs**
- **JGraphT (Results asking Ai)**

General Features of JGraphT – a *‚short‘* Overview provided by Ai

Flexible Graph Modeling

- **Generic types for vertices and edges**: JGraphT lets you use *any Java object*—strings, URLs, XML documents, or even other graphs—for vertices and edges, with type safety provided by generics.
  jgrapht.org Open Hub
- **Wide range of graph structures**:
  - Directed and undirected
  - Weighted and unweighted
  - Simple graphs, multigraphs, pseudographs jgrapht.org java-source.net
- **Advanced graph variants**:
  - **Unmodifiable graphs** for safe, read-only usage
  - **Listenable graphs** that allow external monitoring of graph changes
  - **Live subgraph views**, adapters for combining and converting graphs (e.g., treating a directed graph as undirected) jgrapht.org

---

Algorithmic Power

- **Rich traversal support**: DFS, BFS, along with specialized iterators jgrapht.org
- **Extensive algorithm library**, including:
  - Path finding (e.g., shortest paths)
  - Clique detection, graph isomorphism
  - Coloring, cycle detection, connectivity
  - Matching, flows, cuts, centrality measures
  - NP-hard problem approximations (e.g., independent set, traveling salesman)
  - Exotic algorithms like Berge graph detection jgrapht.org Eindhoven Tech Research Portal
- **Importers and Exporters**: Support for formats like GraphViz, GraphML, adjacency lists/matrices, plus adapters for integration with libraries like JGraphX and Guava jgrapht.org coe.psu.ac.th
  Eindhoven Tech Research Portal

More prompting in Ai about JGraphT

Has JGraphT classes for Nodes and Trees

You're asking whether JGraphT includes dedicated "Node" and "Tree" classes. It doesn't have generic classes named `Node` or `Tree`, but it does offer a range of **graph**, **tree-specific algorithms**, and **utility node containers**.

. . .

Summary Table

Concept	JGraphT Support
Generic `Node` class	None — vertices are user-defined objects
Generic `Tree` class	None — but many tree-specific algorithms and data structures

JGraphT's Vocabulary:

Vertex & Edge

JGraphT – Example for an Undirected Graph


```
import org.jgrapht.*;
import org.jgrapht.graph.*;

public class ExampleUndirectedGraph {

   public static void main(String[] args) {

      Graph<String, DefaultEdge> g = new SimpleGraph<>(DefaultEdge.class);

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

      g.addEdge("A", "B");
      g.addEdge("B", "C");
      g.addEdge("C", "A");

      System.out.println(g);

   }
}
```

JGraphT's Vocabulary:

Vertex & Edge

Simplest class to implement a Vertex in JGraphT

In **JGraphT**, vertices can be *any* Java object — so you can make a simple class for them like this:

java Copy Edit

```java
public class Vertex {
    private String id;
    private String label;

    public Vertex(String id, String label) {
        this.id = id;
        this.label = label;
    }

    public String getId() { return id; }
    public String getLabel() { return label; }

    @Override
    public String toString() {
        return label; // How it will appear when printing the graph
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Vertex)) return false;
        Vertex v = (Vertex) o;
        return id.equals(v.id);
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }
}
```

Example usage with JGraphT:

java Copy Edit

```java
Graph<Vertex, DefaultEdge> g = new SimpleGraph<>(DefaultEdge.class);
Vertex v1 = new Vertex("1", "A");
Vertex v2 = new Vertex("2", "B");
g.addVertex(v1);
g.addVertex(v2);
g.addEdge(v1, v2);
System.out.println(g);
```

Key points for JGraphT vertices:

- `equals` and `hashCode` must be consistent — graphs use these to identify vertices.
- `toString()` controls how vertices appear in console output.
- You can store extra data (like coordinates, weights, etc.) as fields.

You will see this in our Example Packages:

are there libraries on top of JGraphT wihich handle Trees

There are **no widely-used, standalone libraries that are built *on top* of JGraphT specifically to handle trees**, but there are a few directions worth exploring:

✅ 1. Libraries or Tools Using JGraphT Internally

Some projects use JGraphT internally and may provide tree utilities as part of their domain-specific features. However, these are usually:

- **Not general-purpose tree libraries**

✅ 2. JGraphT + Tree Abstractions (DIY / Helper Wrappers)

Because of JGraphT's flexibility, many developers write their own **lightweight wrapper or utility layer** on top of `DirectedGraph` or `DirectedAcyclicGraph` to enforce tree semantics.

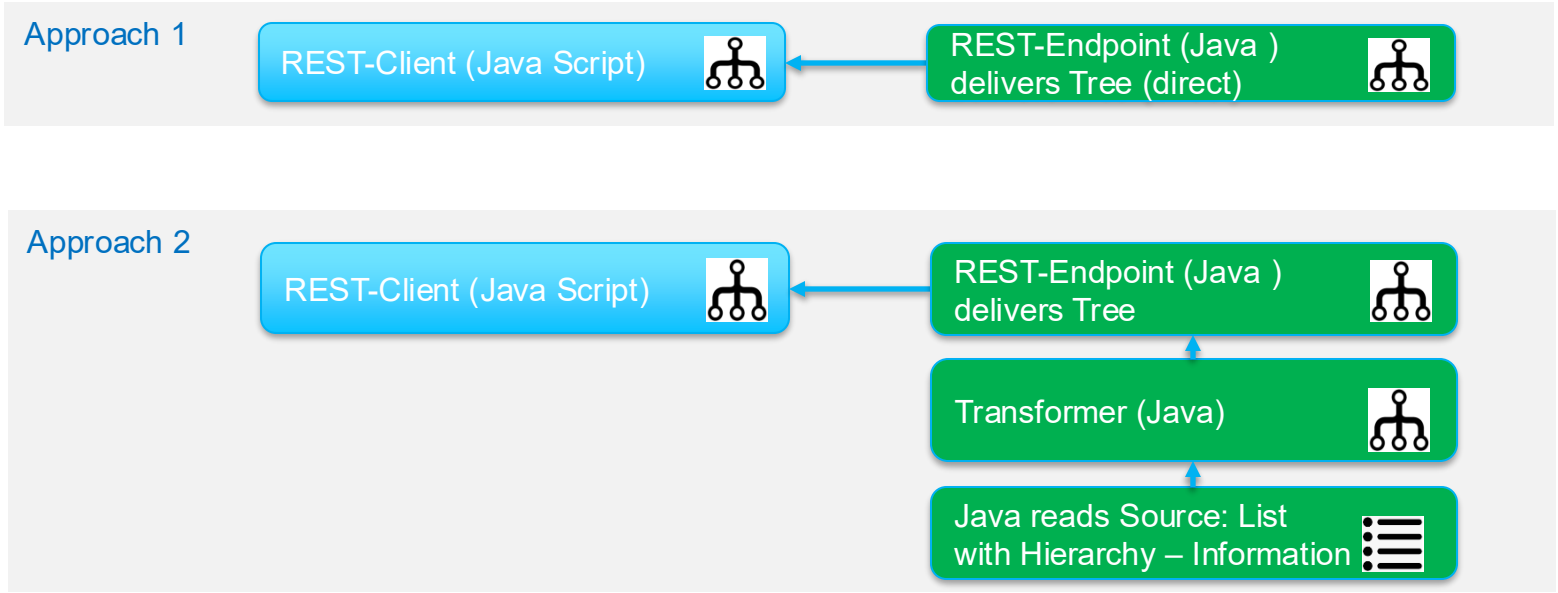
You can create:

- A `TreeNode` wrapper around vertices
- A helper class that tracks parent/child relationships
- Traversal and structural validation methods

There are GitHub projects and blog posts with such helpers, but again, not formal libraries.

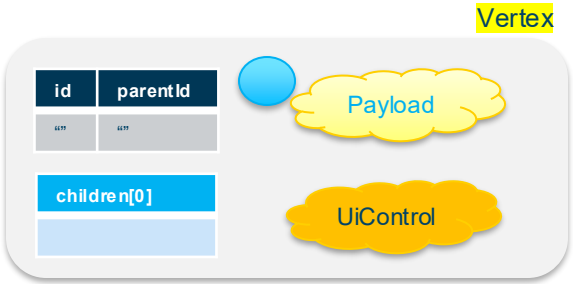
We do so in our Example Packages

Remember: Our Approaches

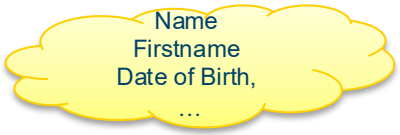


Technical Requirement: Consistency of Result (independent from Approach)

Central Object



Types of PayLoad



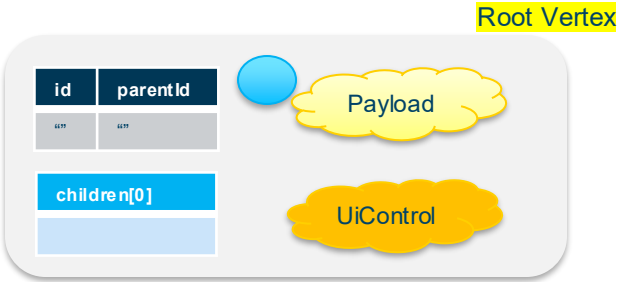
UiControl

IBM Carbon TreeView

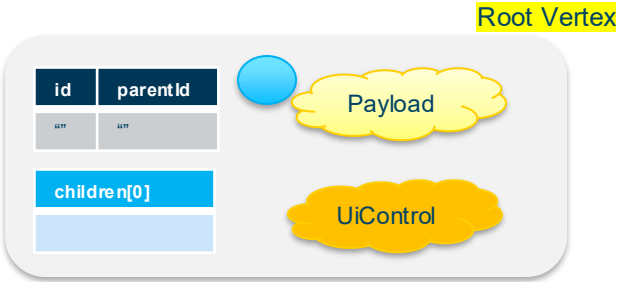
Approach 1

Approach 1: Recursion

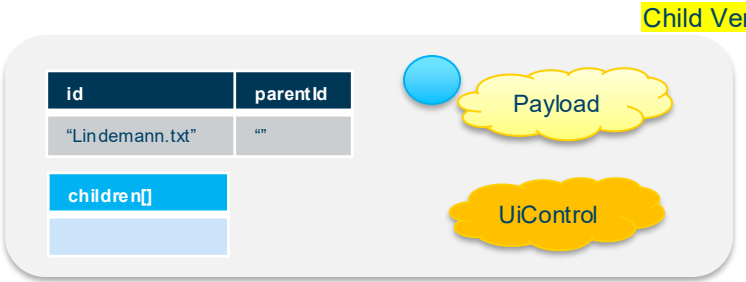
// Handle Root



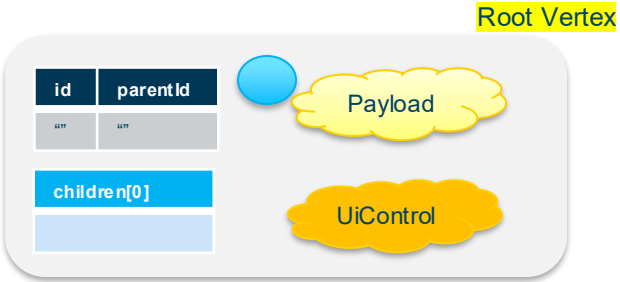
Approach 1: Recursion



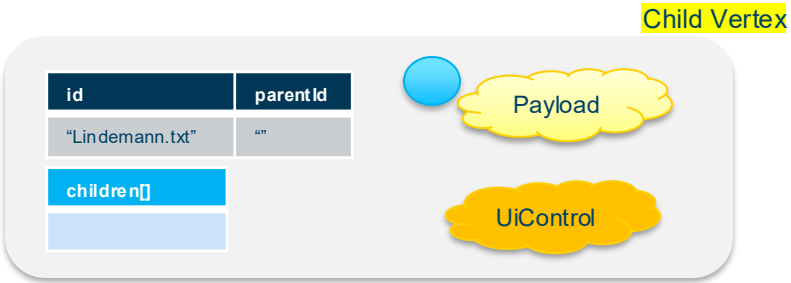
// Handle Child



Approach 1: Recursion

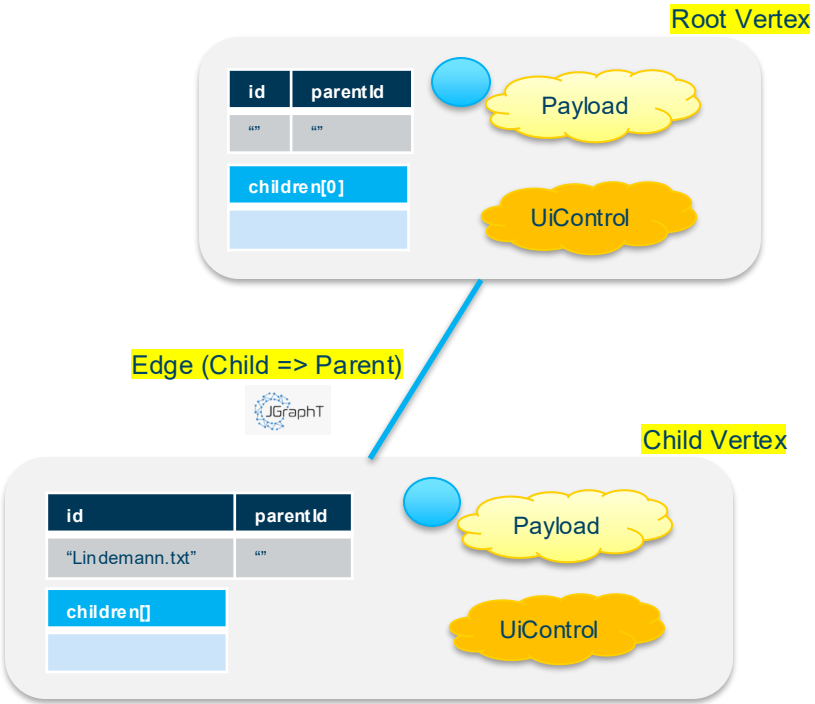


// Hierarchy Handling



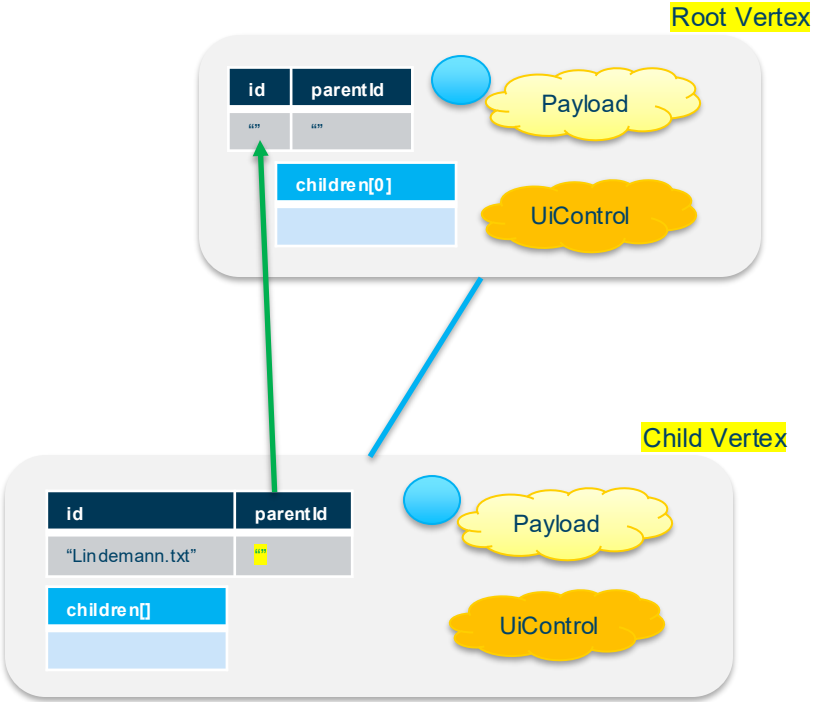
Approach 1: Recursion

// Hierarchy Handling



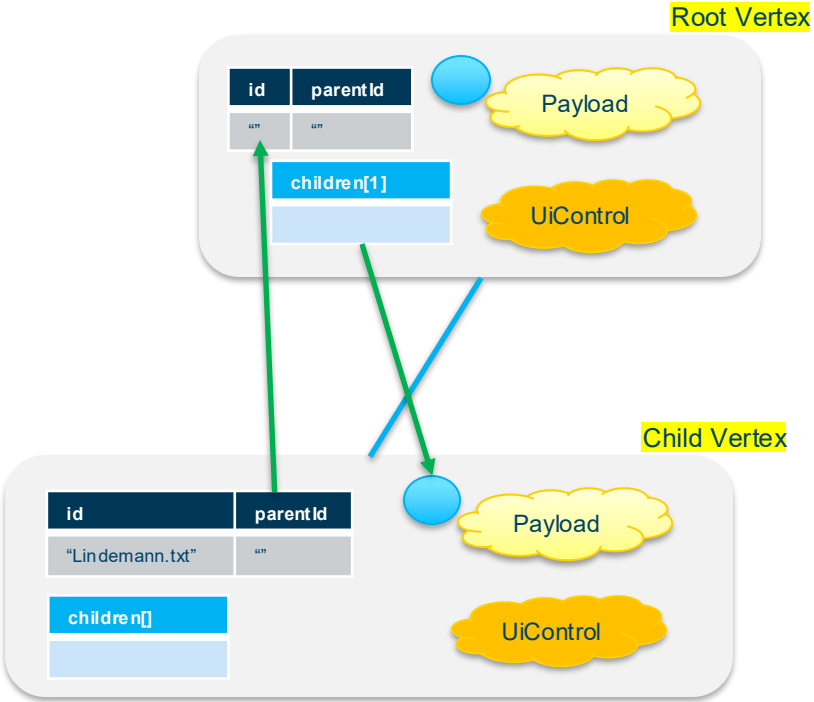
Approach 1: Recursion

// Hierarchy Handling



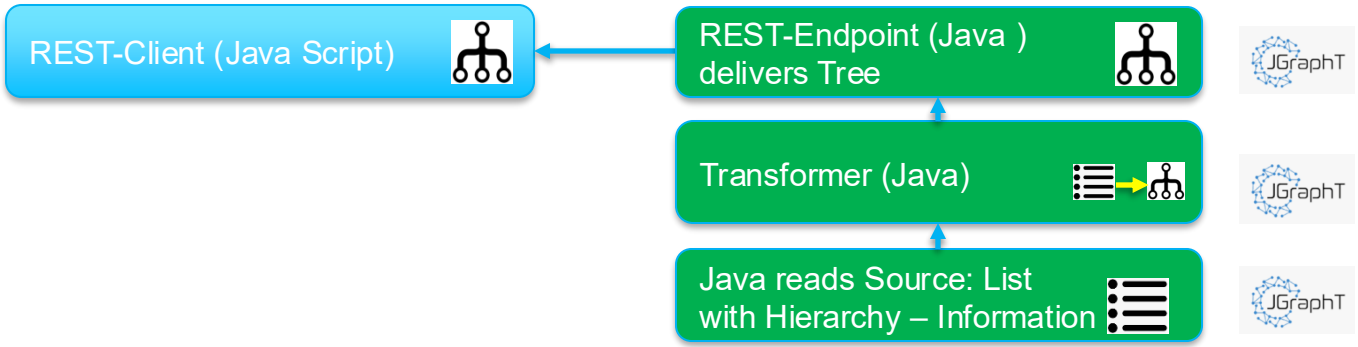
Approach 1: Recursion

// Hierarchy Handling



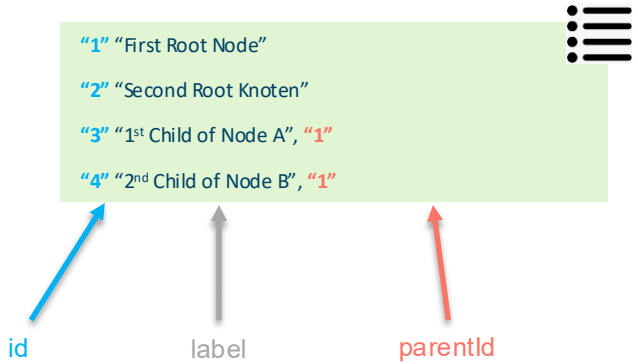
IBM Carbon TreeView

Approach 2



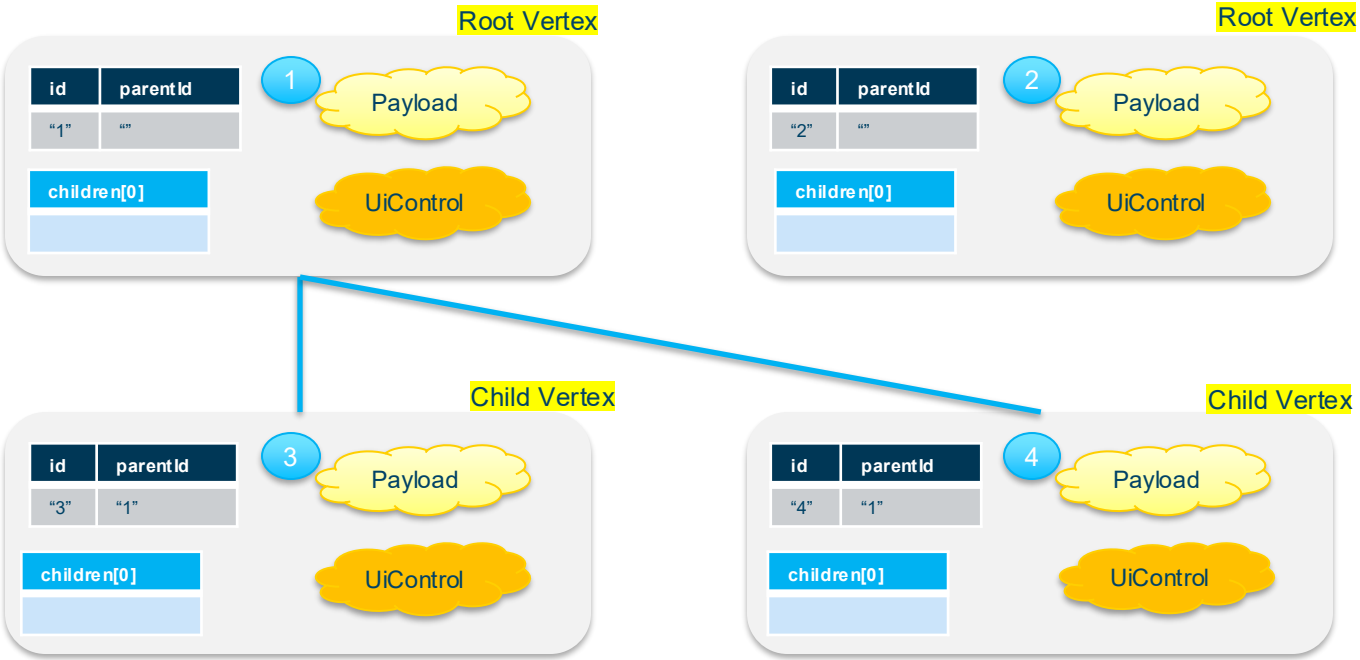
Approach 2: List with hierarchical information

Java reads Source: List with Hierarchy – Information

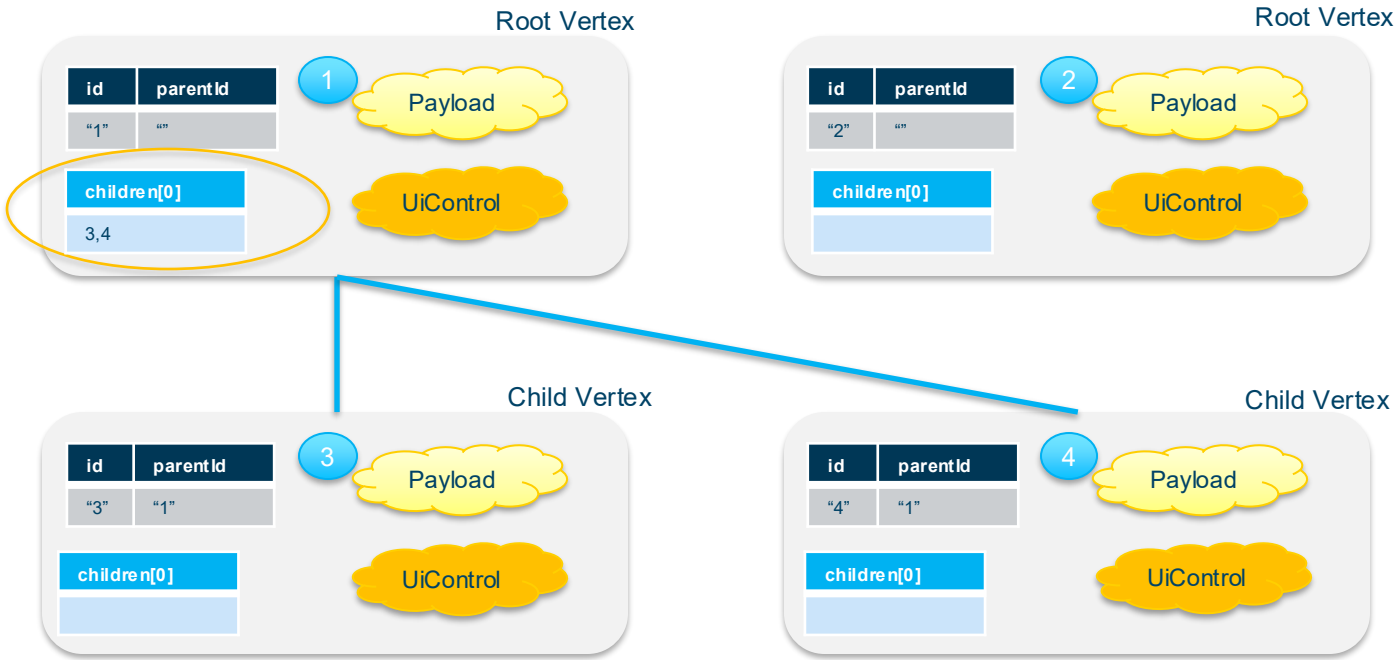


Approach 2: List with hierarchical information

Approach 2: List with hierarchical information

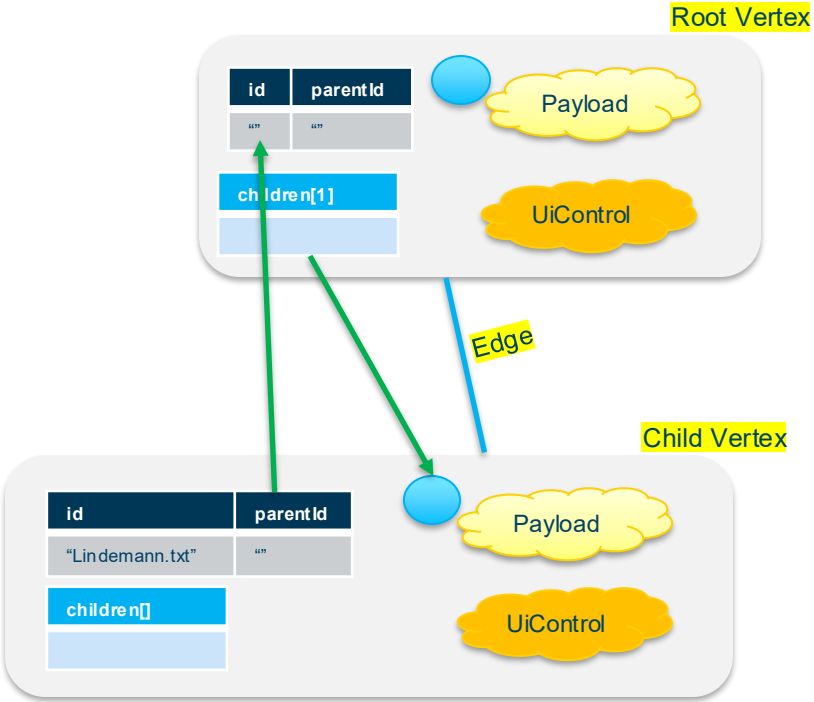


Approach 2: List with hierarchical information



Approach 1 and 2 provide the same consistent Data Structure

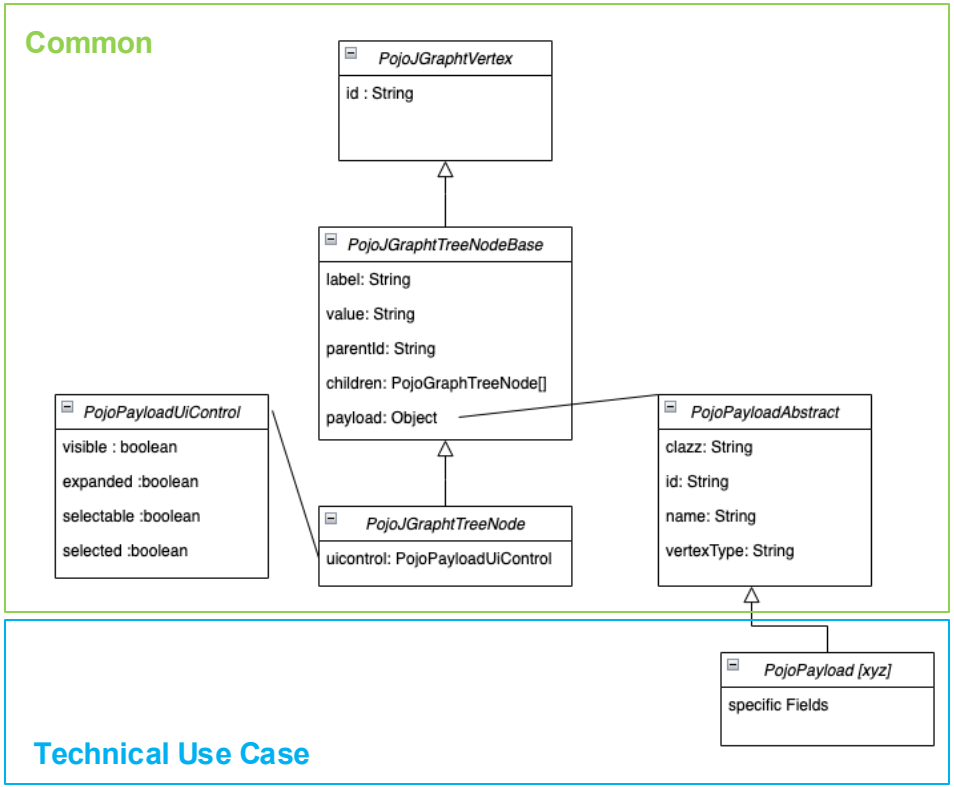
// Hierarchy Handling



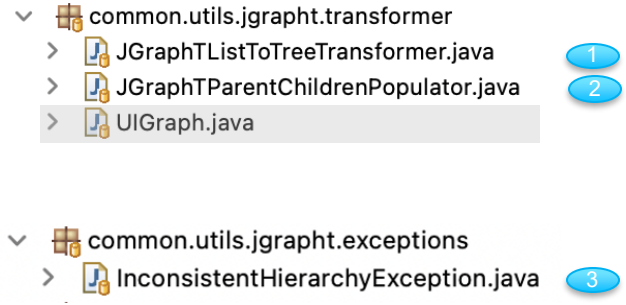
Pojo: Data to deal with and to hand over to IBM Carbon Client

Pojo = Plain Old Java Object

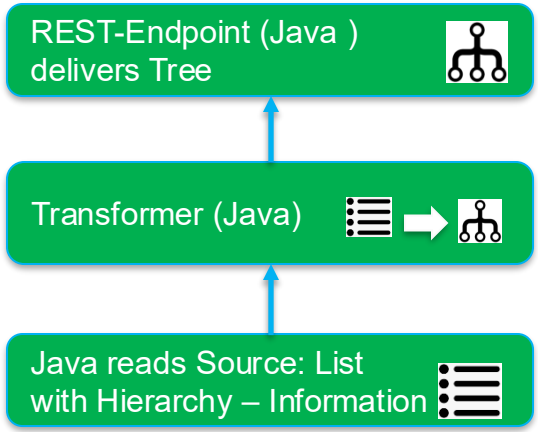
- Attributes
- Getter & Setter



Pojo: Data to deal with and to hand over to IBM Carbon Client



1 Performs this Job in Approach 2:



can throw 3

2 Supports unique Data Structure in Approach 1

Overview about the Examples:

Implementation without Server & Endpoints

GET	/api/carbonplay/Example/TreeViaRecursion	Deliver Explain Directory (recursiv
GET	/api/carbonplay/Example/TreeViaList	Deliver an example Tree to populate a Car
GET	/api/carbonplay/Example/Tree/Direct/Example	Deliver an example Tree to p

The Basic Sequence Diagram

All Examples follow this Sequence Diagram

… more or less with Variations