Below is a complete, self-contained implementation for Lab 2 using GraphStream. It contains **three files**: `Tools.java`, `TraversalAlgorithms.java`, and `App.java`. Paste them into the same package (or no package), ensure the DGS files are reachable via relative paths in `data/` (you can change those paths in `App`), add GraphStream dependencies, and run.

Tools.java

```
import org.graphstream.graph.*;
import org.graphstream.graph.implementations.*;
import org.graphstream.stream.file.FileSourceDGS;

import java.io.IOException;
import java.util.Locale;

public class Tools {
    public static final String BASE_STYLES = String.join("\n",
            "node {",
            "  size: 12px;",
            "  fill-color: #A9CCE3;",
            "  stroke-mode: plain;",
            "  stroke-color: #34495E;",
            "  stroke-width: 1px;",
            "  text-size: 12;",
            "  text-alignment: above;",
            "}",
            "edge {",
            "  fill-color: #95A5A6;",
            "  size: 1px;",
            "}",
            ".inQueue { fill-color: #F1C40F; size: 16px; }",
            ".current { fill-color: #E67E22; size: 18px; }",
            ".visited { fill-color: #2ECC71; size: 14px; }",
            ".stacked { fill-color: #9B59B6; size: 16px; }",
            ".treeEdge { fill-color: #2C3E50; size: 3px; }",
            ".redNode { fill-color: red; size: 30px; }"
    );

    public static Graph loadGraph(String path) {
        Graph g = new SingleGraph(path);
        g.setStrict(false);
        g.setAutoCreate(true);
        g.setAttribute("ui.stylesheet", BASE_STYLES);
        FileSourceDGS fs = new FileSourceDGS();
        fs.addSink(g);
        try {
            fs.readAll(path);
        } catch (IOException e) {
            throw new RuntimeException("Failed to load DGS: " + path, e);
        } finally {
            fs.removeSink(g);
        }
        return g;
    }

    public static void show(Graph g, String title) {
        g.setAttribute("ui.title", title);
        g.display();
    }

    public static void pause(long ms) {
        try { Thread.sleep(ms); } catch (InterruptedException ignored) {}
    }

    public static double averageDegree(Graph g) {
        double sum = 0.0;
        for (Node n : g) sum += n.getDegree();
        return g.getNodeCount() > 0 ? sum / g.getNodeCount() : 0.0;
    }

    public static double weight(Edge e) {
        if (e.hasNumber("weight")) return e.getNumber("weight");
        if (e.hasNumber("w")) return e.getNumber("w");
        return 1.0; // unweighted default
    }

    public static void labelNodesWithAttribute(Graph g, String attr) {
        for (Node n : g) {
            if (n.hasAttribute(attr)) {
                Object v = n.getAttribute(attr);
                n.setAttribute("ui.label", String.format(Locale.US, "%s=%.
3f", attr, toDouble(v)));
            }
        }
    }

    private static double toDouble(Object v) {
        if (v instanceof Number) return ((Number) v).doubleValue();
        try { return Double.parseDouble(String.valueOf(v)); } catch
(Exception e) { return Double.NaN; }
    }

    public static void resetNodeStyles(Graph g) {
        for (Node n : g) {
            n.removeAttribute("ui.class");
            n.removeAttribute("ui.style");
            n.removeAttribute("ui.label");
        }
    }
```

```java
    public static void resetEdgeStyles(Graph g) {
        for (Edge e : g.getEdgeSet()) {
            e.removeAttribute("ui.class");
            e.removeAttribute("ui.style");
        }
    }

    /**
     * Exercise 1 helper: style node red & size 30 if sum of neighbor 'cost' exceeds threshold.
     */
    public static void styleRedIfNeighborCostSumAbove(Graph g, double threshold) {
        for (Node u : g) {
            double sum = 0.0;
            for (Edge e : u.getEachEdge()) {
                Node v = e.getOpposite(u);
                if (v.hasNumber("cost")) sum += v.getNumber("cost");
            }
            if (sum > threshold) {
                u.setAttribute("ui.class", "redNode");
                u.setAttribute("ui.label", String.format(Locale.US, "∑nbr(cost)=%.2f", sum));
            }
        }
    }

    /** Linear blue→red gradient for a value in [min,max]. */
    public static String heatColor(double v, double min, double max) {
        if (Double.isNaN(v)) return "#7F8C8D";
        double t = (max > min) ? Math.max(0, Math.min(1, (v - min) / (max - min))) : 0.0;
        int r = (int) Math.round(255 * t);
        int b = (int) Math.round(255 * (1 - t));
        return String.format("#%02X00%02X", r, b);
    }
}
```

TraversalAlgorithms.java

```java
import org.graphstream.graph.*;

import java.security.SecureRandom;
import java.util.*;

public class TraversalAlgorithms {
    private static final SecureRandom RNG = new SecureRandom();
```

```java
// ===================
// Exercise 2: BFS EVOLUTION (frontier coloring)
// ===================
public static void BFS(Graph g, Node start, long delayMs) {
    Tools.resetNodeStyles(g); Tools.resetEdgeStyles(g);
    if (start == null) start = pickAnyNode(g);
    Queue<Node> q = new ArrayDeque<>();
    Set<Node> visited = new HashSet<>();
    start.setAttribute("ui.class", "current");
    q.add(start);
    Tools.pause(delayMs);

    while (!q.isEmpty()) {
        Node u = q.poll();
        u.setAttribute("ui.class", "current");
        Tools.pause(delayMs);

        if (!visited.contains(u)) {
            visited.add(u);
            u.setAttribute("ui.class", "visited");
            Tools.pause(delayMs);

            for (Edge e : u.getEachLeavingEdge()) {
                Node v = e.getTargetNode();
                if (!visited.contains(v)) {
                    if (!q.contains(v)) {
                        v.setAttribute("ui.class", "inQueue");
                        Tools.pause(delayMs);
                        q.add(v);
                    }
                }
            }
        }
    }
}

// ===================
// Exercise 3: DFS EVOLUTION (stack coloring)
// ===================
public static void DFS(Graph g, Node start, long delayMs) {
    Tools.resetNodeStyles(g); Tools.resetEdgeStyles(g);
    if (start == null) start = pickAnyNode(g);
    Deque<Node> st = new ArrayDeque<>();
    Set<Node> visited = new HashSet<>();
    st.push(start);
    start.setAttribute("ui.class", "stacked");
    Tools.pause(delayMs);

    while (!st.isEmpty()) {
        Node u = st.pop();
        u.setAttribute("ui.class", "current");
```

```
            Tools.pause(delayMs);

            if (!visited.contains(u)) {
                visited.add(u);
                u.setAttribute("ui.class", "visited");
                Tools.pause(delayMs);
                // push neighbors (you can randomize order for prettier
trees)
                List<Node> nbrs = new ArrayList<>();
                for (Edge e : u.getEachLeavingEdge())
nbrs.add(e.getTargetNode());
                Collections.shuffle(nbrs, RNG);
                for (Node v : nbrs) {
                    if (!visited.contains(v)) {
                        if (!st.contains(v)) {
                            v.setAttribute("ui.class", "stacked");
                            Tools.pause(delayMs);
                            st.push(v);
                        }
                    }
                }
            }
        }
    }

    // ===================
    // Exercise 6: Spanning trees via traversals (edge highlighting)
    // ===================
    public static void bfsSpanningTree(Graph g, Node start) {
        Tools.resetNodeStyles(g); Tools.resetEdgeStyles(g);
        if (start == null) start = pickAnyNode(g);
        Queue<Node> q = new ArrayDeque<>();
        Set<Node> seen = new HashSet<>();
        q.add(start); seen.add(start);
        while (!q.isEmpty()) {
            Node u = q.poll();
            for (Edge e : u.getEachEdge()) {
                Node v = e.getOpposite(u);
                if (!seen.contains(v)) {
                    seen.add(v);
                    q.add(v);
                    e.setAttribute("ui.class", "treeEdge");
                }
            }
        }
    }

    public static void dfsSpanningTree(Graph g, Node start) {
        Tools.resetNodeStyles(g); Tools.resetEdgeStyles(g);
        if (start == null) start = pickAnyNode(g);
        Deque<Node> st = new ArrayDeque<>();
        Set<Node> seen = new HashSet<>();
        st.push(start); seen.add(start);
        while (!st.isEmpty()) {
            Node u = st.pop();
            for (Edge e : u.getEachEdge()) {
                Node v = e.getOpposite(u);
                if (!seen.contains(v)) {
                    seen.add(v);
                    st.push(v);
                    e.setAttribute("ui.class", "treeEdge");
                }
            }
        }
    }

    // ===================
    // Exercise 4: Dijkstra (stores dist on nodes)
    // ===================
    public static void dijkstra(Graph g, Node source) {
        if (source == null) source = pickAnyNode(g);
        // init
        for (Node n : g) {
            n.setAttribute("dist", Double.POSITIVE_INFINITY);
            n.removeAttribute("prev");
        }
        source.setAttribute("dist", 0.0);

        // naive priority queue
        Comparator<Node> cmp = Comparator.comparingDouble(n ->
n.getNumber("dist"));
        PriorityQueue<Node> pq = new PriorityQueue<>(cmp);
        pq.add(source);
        Set<Node> settled = new HashSet<>();

        while (!pq.isEmpty()) {
            Node u = pq.poll();
            if (!settled.add(u)) continue;
            double du = u.getNumber("dist");
            for (Edge e : u.getEachLeavingEdge()) {
                Node v = e.getTargetNode();
                double w = Tools.weight(e);
                double nd = du + w;
                double cur = v.getNumber("dist");
                if (nd < cur) {
                    v.setAttribute("dist", nd);
                    v.setAttribute("prev", u.getId());
                    pq.remove(v);
                    pq.add(v);
                }
            }
        }
    }

    // ===================
    // Exercise 5: Eccentricities, diameter, radius + heatmap
    // ===================
    public static void computeEccentricities(Graph g) {
        // if graph is directed with asymmetric weights, this uses outgoing
edges only
        double diameter = 0.0;
        double radius = Double.POSITIVE_INFINITY;

        for (Node s : g) {
            dijkstra(g, s);
            double ecc = 0.0;
            for (Node t : g) {
                double d = t.getNumber("dist");
                if (Double.isFinite(d)) ecc = Math.max(ecc, d);
            }
            s.setAttribute("ecc", ecc);
            diameter = Math.max(diameter, ecc);
            radius = Math.min(radius, ecc);
        }

        g.setAttribute("diameter", diameter);
        g.setAttribute("radius", radius);

        // heatmap by ecc
        double min = Double.POSITIVE_INFINITY, max =
Double.NEGATIVE_INFINITY;
        for (Node n : g) {
            double v = n.getNumber("ecc");
            if (Double.isFinite(v)) { min = Math.min(min, v); max =
Math.max(max, v); }
        }
        for (Node n : g) {
            double v = n.getNumber("ecc");
            String color = Tools.heatColor(v, min, max);
            n.setAttribute("ui.style", String.format(Locale.US, "fill-color:
%s;", color));
            n.setAttribute("ui.label", String.format(Locale.US, "Ecc=%.2f",
v));
        }
    }

    private static Node pickAnyNode(Graph g) {
        return g.getNodeCount() > 0 ? g.getNode(0) : null;
    }
}
```

App.java

```java
import org.graphstream.graph.*;

import java.security.SecureRandom;

public class App {
    private static final SecureRandom RNG = new SecureRandom();

    // Adjust paths to your DGS files (from the Lab 2 handout names)
    private static final String DGS_FIRST = "data/
firstgraphlab2.dgs";                    // Ex.1
    private static final String DGS_BFS_EVOLUTION = "data/
completegrid_10.dgs";         // Ex.2
    private static final String DGS_DFS_EVOLUTION_1 = "data/
completegrid_30.dgs";       // Ex.3
    private static final String DGS_DFS_EVOLUTION_2 = "data/
uncompletegrid_50-0.12.dgs"; // Ex.3
    private static final String DGS_WEIGHTED_GRID = "data/
gridvaluated_10_12.dgs";      // Ex.4
    private static final String DGS_VON_NEUMANN = "data/
gridvonneumann_30.dgs";        // Ex.6 (+Ex.5 if desired)

    public static void main(String[] args) {
        System.setProperty("org.graphstream.ui", "swing");
        // === Exercise 1 ===
        exercise1_averageDegree_and_costStyling(35.0);

        // === Exercise 2 ===
        exercise2_bfsEvolution(40);

        // === Exercise 3 ===
        exercise3_dfsEvolution(35);

        // === Exercise 4 ===
        exercise4_dijkstraRandomSource();

        // === Exercise 5 ===
        exercise5_eccentricities_heatmap();

        // === Exercise 6 ===
        exercise6_spanningTrees();
    }

    // -------------------- Exercise 1 --------------------
    private static void exercise1_averageDegree_and_costStyling(double
threshold) {
        Graph g = Tools.loadGraph(DGS_FIRST);
        Tools.show(g, "Ex.1 – AvgDegree & Red if ∑nbr(cost) > " + threshold);
        double avgDeg = Tools.averageDegree(g);
        System.out.printf("[Ex1] Average degree = %.3f\n", avgDeg);
        Tools.styleRedIfNeighborCostSumAbove(g, threshold);
        Tools.pause(2000);
    }

    // -------------------- Exercise 2 --------------------
    private static void exercise2_bfsEvolution(long delayMs) {
        Graph g = Tools.loadGraph(DGS_BFS_EVOLUTION);
        Tools.show(g, "Ex.2 – BFS Evolution (frontier coloring)");
        Node start = pickRandomNode(g);
        System.out.println("[Ex2] BFS start node: " + start.getId());
        TraversalAlgorithms.BFS(g, start, delayMs);
        Tools.pause(2000);
    }

    // -------------------- Exercise 3 --------------------
    private static void exercise3_dfsEvolution(long delayMs) {
        Graph g1 = Tools.loadGraph(DGS_DFS_EVOLUTION_1);
        Tools.show(g1, "Ex.3 – DFS Evolution on completegrid_30");
        Node s1 = pickRandomNode(g1);
        System.out.println("[Ex3] DFS start #1: " + s1.getId());
        TraversalAlgorithms.DFS(g1, s1, delayMs);
        Tools.pause(1500);

        Graph g2 = Tools.loadGraph(DGS_DFS_EVOLUTION_2);
        Tools.show(g2, "Ex.3 – DFS Evolution on uncompletegrid_50-0.12");
        Node s2 = pickRandomNode(g2);
        System.out.println("[Ex3] DFS start #2: " + s2.getId());
        TraversalAlgorithms.DFS(g2, s2, delayMs);
        Tools.pause(2000);
    }

    // -------------------- Exercise 4 --------------------
    private static void exercise4_dijkstraRandomSource() {
        Graph g = Tools.loadGraph(DGS_WEIGHTED_GRID);
        Tools.show(g, "Ex.4 – Dijkstra on weighted grid (labels = dist)");
        Node s = pickRandomNode(g);
        System.out.println("[Ex4] Dijkstra source: " + s.getId());
        TraversalAlgorithms.dijkstra(g, s);
        // label distances
        for (Node n : g) {
            double d = n.getNumber("dist");
            if (Double.isFinite(d)) n.setAttribute("ui.label",
String.format("%.1f", d));
        }
        Tools.pause(2000);
    }

    // -------------------- Exercise 5 --------------------
    private static void exercise5_eccentricities_heatmap() {
        // You can switch to another graph if required by your handout
        Graph g = Tools.loadGraph(DGS_VON_NEUMANN);
        Tools.show(g, "Ex.5 – Eccentricities Heatmap (blue→red), diam &
radius");
        TraversalAlgorithms.computeEccentricities(g);
        double diam = g.getNumber("diameter");
        double rad = g.getNumber("radius");
        System.out.printf("[Ex5] Diameter=%.3f, Radius=%.3f\n", diam, rad);
        Tools.pause(2500);
    }

    // -------------------- Exercise 6 --------------------
    private static void exercise6_spanningTrees() {
        Graph gB = Tools.loadGraph(DGS_VON_NEUMANN);
        Tools.show(gB, "Ex.6 – BFS Spanning Tree (thick edges)");
        TraversalAlgorithms.bfsSpanningTree(gB, pickRandomNode(gB));
        Tools.pause(1500);

        Graph gD = Tools.loadGraph(DGS_VON_NEUMANN);
        Tools.show(gD, "Ex.6 – DFS Spanning Tree (thick edges)");
        TraversalAlgorithms.dfsSpanningTree(gD, pickRandomNode(gD));
        Tools.pause(2000);
    }

    private static Node pickRandomNode(Graph g) {
        if (g.getNodeCount() == 0) return null;
        return g.getNode(RNG.nextInt(g.getNodeCount()));
    }
}
```

How this maps to Lab 2 tasks

- **Ex.1**: computes **average degree** and styles nodes **red + bigger** when $\sum_{v\in N(u)} cost(v) > threshold$. Threshold is a parameter in `exercise1_...` – set to your lab's required value.
- **Ex.2**: `BFS(...)` shows *evolution*: current (orange), in-queue (yellow), visited (green), with per-step delays via `Tools.pause`.
- **Ex.3**: `DFS(...)` shows *evolution* with a stack metaphor: stacked (purple), current (orange), visited (green), on **two graphs** from the handout.
- **Ex.4**: `dijkstra(...)` reads `weight` (or `w`) from edges, labels nodes with distances from a **random source**.
- **Ex.5**: `computeEccentricities(...)` sets per-node `ecc`, graph `diameter` & `radius`, and paints a **blue→red heatmap** with labels `Ecc=...`.
- **Ex.6**: `bfsSpanningTree(...)` and `dfsSpanningTree(...)` **thicken** tree edges via class `treeEdge`.

  Tip: if your DGS files use different attribute names (e.g., `cost` vs `c`), tweak `styleRedIfNeighborCostSumAbove` and `weight()` accordingly.