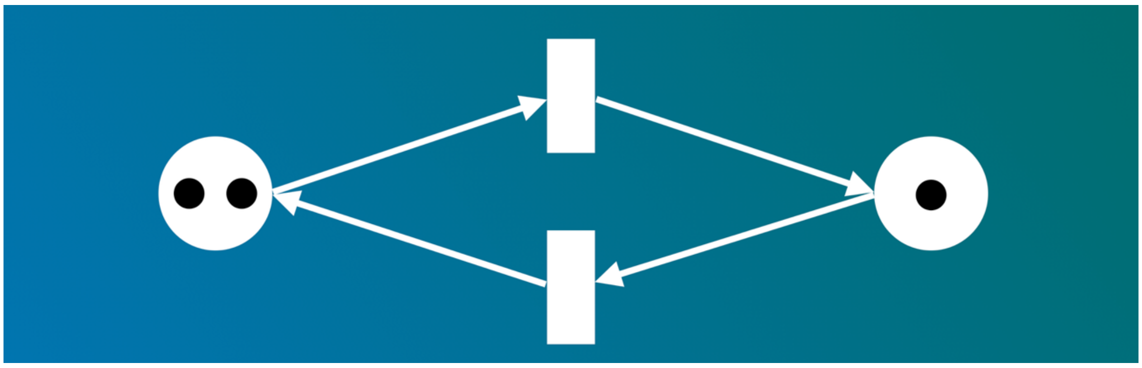

Petri Network

written by Philipp Klein
(PhilippKlein004 @ GitHub)

Version: 1.0
Languages: Java

TABLE OF CONTENTS

1. HOW TO USE THE SOFTWARE..1

2. HOW DO PETRI NETWORKS WORK?...3

3. HOW ARE THE COMPONENTS STRUCTURED? 4

1. How to use the software

When you launch the JAR file or run the main method in the source code, the main window will appear. Inside the window, a demo network is already set up for interaction.

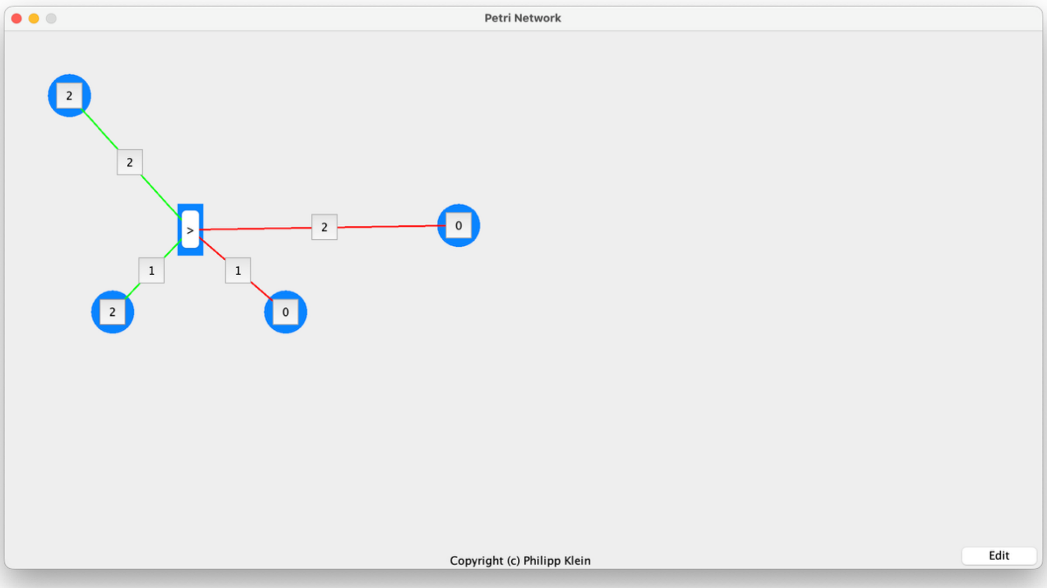

Figure 1: Main window with demo structure

A petri network has two essential components (see Figure 2):

1. Node/Place: stores the tokens and can give Transitions life.

2. Transitions: regulate the flow of tokens in the network. When it fires, it takes out tokens from the incoming Places and gives them to the outgoing Places.

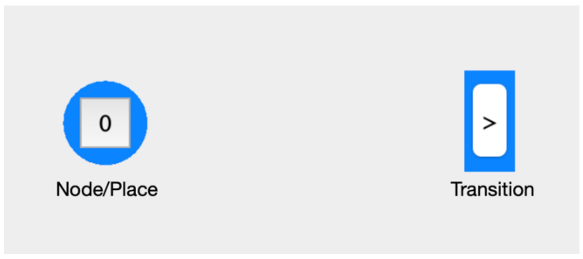

Figure 2: Elements of the petri network

Incoming Places are marked with a green Edge (line), outgoing Edges are red. The number of tokens can be changed by clicking the button inside the Places. The maximum number of tokens is 9. To edit an existing network and enter “Editmode”, press the “Edit” button in the bottom-left corner.

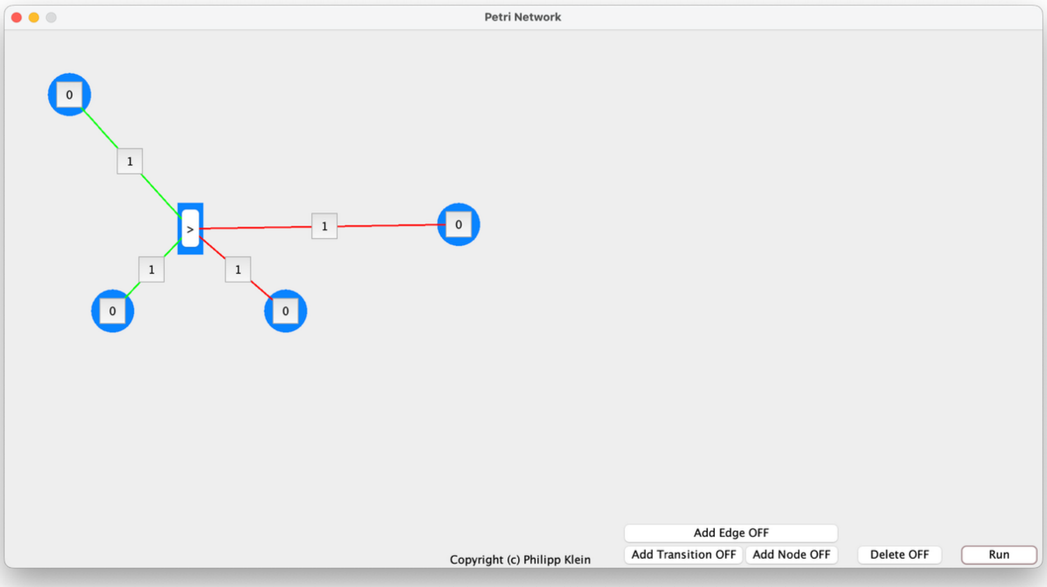

Figure 3: Active edit-mode in the window

In “Editmode” you have the following options:

1. **Delete Places and Transitions**: when you press the “Delete OFF” button, the “Deletion mode” will be activated. If you want to delete a specific Place, Transition, or Edge, just press its button to delete it.

2. **Add a New Node/Place**: to add a new Node, press the “Add Node OFF” button. When it’s activated, simply click anywhere on the screen to add a new Place at that position.

3. **Add a New Transition**: to add a new Transition, press the “Add Transition OFF” button. When it’s activated, simply click anywhere on the screen to add a new Transition at that position.

4. **Add a new Edge**: to create a new Edge between a Place and a Transition, click the button “Add Edge OFF.” When it’s activated, click either on a Place or a Transition first to set the direction. After that, click the next component to add a new Edge.

   If the first component is a Place, then the next one can only be a Transition. Therefore, the Edge is an incoming Edge for the Transition, and vice versa.

To interact with the newly created network press the “Run” button in the bottom-left corner to exit the “Editmode”.

2. How do Petri Networks work?

As stated before, a Petri network consists of Places, Transitions, and Edges. Let’s demonstrate how it works with the demo network from the application.

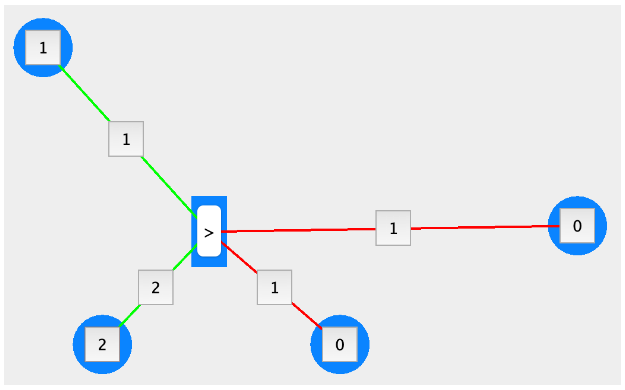

Figure 4: Demo network from the application

For a Transition to fire, all incoming Places (green Edge) must have a number of tokens greater than or equal to their Edge weight and the maximum Edge weight of the outgoing Edges (red).

In the given network, the first incoming place must have a weight equal to or greater than one. For the second one, its number must be equal to or greater than two. Since the maximum weight of the outgoing Edges is one, and the number of tokens in all incoming places is equal to or greater than one, the Transition is active.

If this were not the case, the Transition would be “dead”.

When the Transition fires, the number of tokens in the incoming Places is reduced according to the weight of their respective Edges. The outgoing Places receive tokens corresponding to the weight of their Edges.

3. How are the components structured?

The components of the network are extending the abstract class "PetriNetComponent". In that class, all the necessary properties are defined. The class structure looks like this:

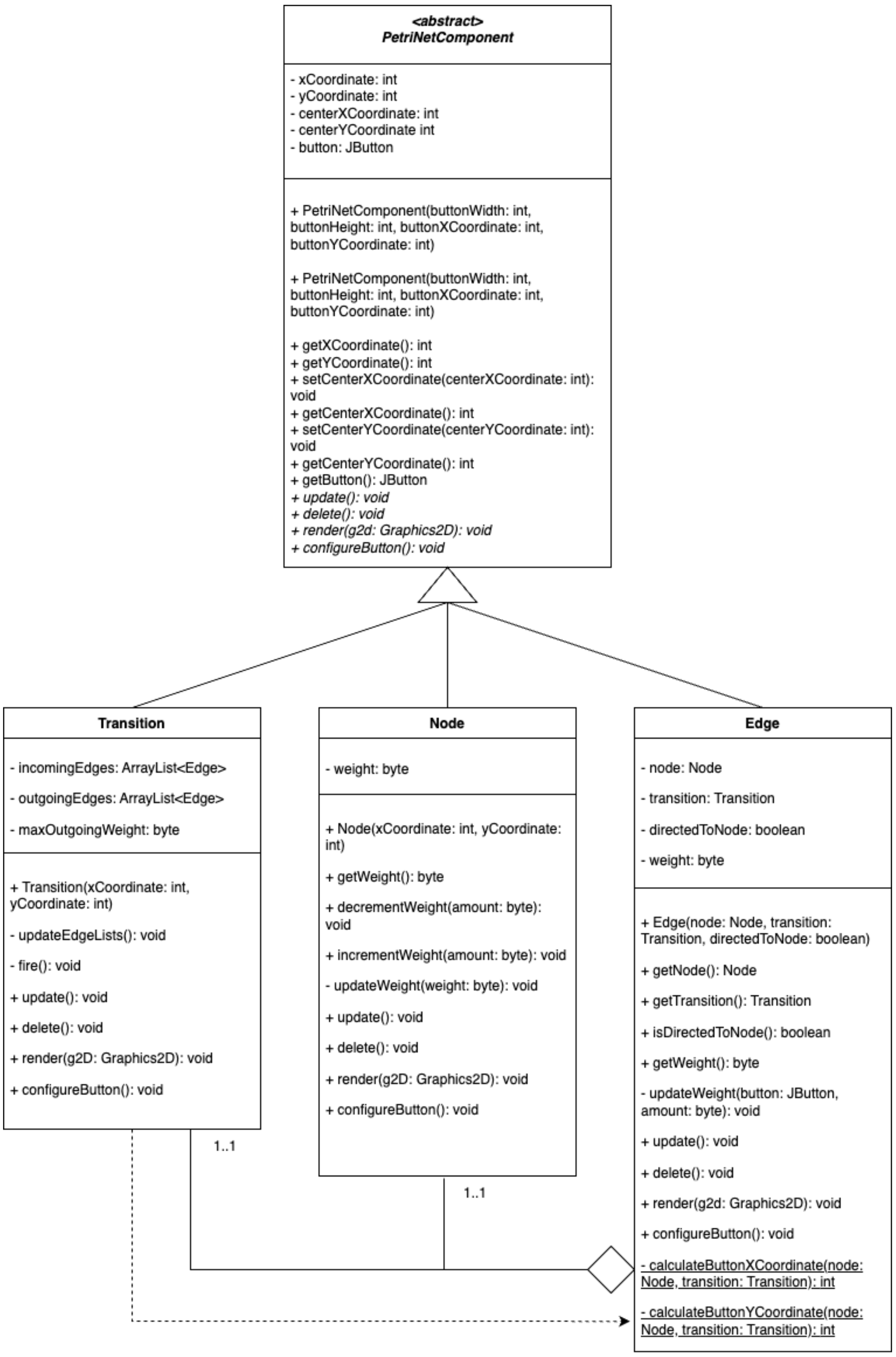
<abstract>
PetriNetComponent
- xCoordinate: int
- yCoordinate: int
- centerXCoordinate: int
- centerYCoordinate int
- button: JButton
+ PetriNetComponent(buttonWidth: int, buttonHeight: int, buttonXCoordinate: int, buttonYCoordinate: int)
+ PetriNetComponent(buttonWidth: int, buttonHeight: int, buttonXCoordinate: int, buttonYCoordinate: int)
+ getXCoordinate(): int
+ getYCoordinate(): int
+ setCenterXCoordinate(centerXCoordinate: int): void
+ getCenterXCoordinate(): int
+ setCenterYCoordinate(centerYCoordinate: int): void
+ getCenterYCoordinate(): int
+ getButton(): JButton
+ update(): void
+ delete(): void
+ render(g2d: Graphics2D): void
+ configureButton(): void
Transition
- incomingEdges: ArrayList<Edge>
- outgoingEdges: ArrayList<Edge>
- maxOutgoingWeight: byte
+ Transition(xCoordinate: int, yCoordinate: int)
- updateEdgeLists(): void
- fire(): void
+ update(): void
+ delete(): void
+ render(g2D: Graphics2D): void
+ configureButton(): void
Node
- weight: byte
+ Node(xCoordinate: int, yCoordinate: int)
+ getWeight(): byte
+ decrementWeight(amount: byte): void
+ incrementWeight(amount: byte): void
- updateWeight(weight: byte): void
+ update(): void
+ delete(): void
+ render(g2D: Graphics2D): void
+ configureButton(): void
Edge
- node: Node
- transition: Transition
- directedToNode: boolean
- weight: byte
+ Edge(node: Node, transition: Transition, directedToNode: boolean)
+ getNode(): Node
+ getTransition(): Transition
+ isDirectedToNode(): boolean
+ getWeight(): byte
- updateWeight(button: JButton, amount: byte): void
+ update(): void
+ delete(): void
+ render(g2d: Graphics2D): void
+ configureButton(): void
- calculateButtonXCoordinate(node: Node, transition: Transition): int
- calculateButtonYCoordinate(node: Node, transition: Transition): int
1..1
1..1

Figure 5: Class structure of the network components

Each component has its x and y coordinates for rendering in the window, as well as x and y coordinates for the center of the component. These center coordinates are used to align the given button within the component. The Edge class provides two static methods for calculating these center coordinates for the buttons. This is necessary due to the varying distances between Transitions and Places. It must also be ensured that if, for example, the Transition is positioned behind the Place, the calculation is adjusted accordingly to position the button correctly.

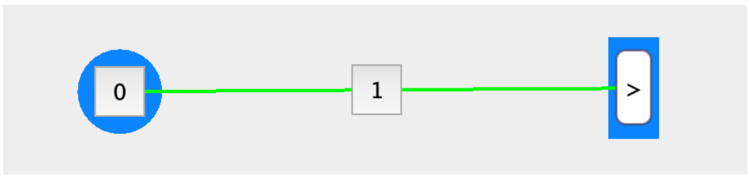

Figure 6: Edge between Place and Transition

An instance of the Transition class holds a list of incoming and outgoing Places, as well as the maximum weight of the outgoing Edges. Whenever the weight of an incoming Place changes, the Transition automatically updates itself to determine whether it can fire. These lists are essential for the firing process, as they help update the weights of the Places.

The Node class primarily serves as a data class, storing tokens and coordinates for each Place within the network.

The Edge class connects instances of Transition and Node/Place. It includes a Boolean value indicating whether it is directed toward a Node or Place, as well as a weight. When the weight of an Edge is updated, the connected Transition is also updated to verify if it can still fire.

If you are interested, you can view the code for this project on GitHub. Thank you for reading ☺