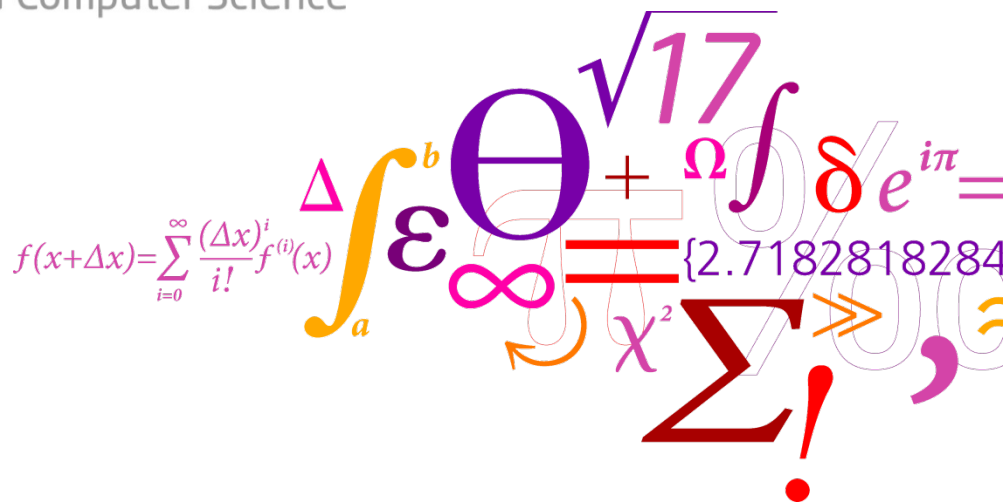


The ePNK: Hands-on / Project (Details)

Ekkart Kindler

DTU Compute

Department of Applied Mathematics and Computer Science



The screenshot shows the Eclipse IDE with a Petri net diagram for a program. The diagram consists of the following elements:

- Place p_0 : Contains the initial value $x = 0$.
- Transition t_0 : Labeled $x = 0$; it consumes p_0 and produces p_1 .
- Place p_1 : Contains the value 1.
- Transition t_1 : Labeled $x < 5$; it consumes p_1 and produces p_2 . It has an associated action: `System.out.println("x = " + x); x++;`
- Transition t_2 : Labeled `System.out.println("x = " + x);`; it consumes p_2 and produces p_1 .
- Transition t_4 : Unlabeled; it consumes p_2 and produces p_1 .

The IDE interface includes a Project Explorer on the left showing the file structure, a Palette on the right with Petri net symbols, and a Properties view at the bottom right showing details for the selected transition.

Core	Property	Value
Appearance	Label	◆ Action System.out.println("x = " + x);x++;
	Object	◆ Transition Code t1
	Text	▣ System.out.println("x = " + x);...

- Implement a new Petri net type extending PTNets (a PNTD for what we call PNCode) by adding
 - Action(label)s to transitions
 - Condition(label)s to transitions
 - Declaration(label)s to pages
- These concepts should be text in Java syntax
- For now, it is not necessary to check syntactical correctness of this syntax

- Install Eclipse (EMT package) and the ePNK
- Import the prepared PNCode PNTD project to your Eclipse workspace
 - Download the project from <http://www2.compute.dtu.dk/~ekki/teaching/external/MBSE-PN2019/code/pntd-plugin.initial.zip>
 - In your Eclipse (EMT) workspace import this project: File → Import... → Existing Project and select the file downloaded above
- Open the imported project `org.pnml.tools.epnk.tutorials.pn-mbse.pntd` and inspect it

Step 1a: Inspect

The screenshot shows the Eclipse IDE interface. The main window displays a UML class diagram for the 'pncode' package. The diagram includes the following classes and relationships:

- PTNet** (purple box) is a superclass of **PNetCode** (yellow box).
- Transition** (purple box) is a superclass of **TransitionCode** (yellow box).
- Page** (purple box) is a superclass of **PageCode** (yellow box).
- Label** (purple box) is a superclass of **Declaration** (yellow box).
- Declaration** (yellow box) has an attribute `text : EString`.
- PageCode** (yellow box) has a directed association to **Declaration** (yellow box) with the multiplicity `[0..*]` and the label `declarations`.

The **Model Explorer** on the left shows the project structure, with the **pncode class diagram** file circled in red. The **Properties** view at the bottom shows the package name **pncode** and its URI and prefix.

- If the workspace is not in the Modelling perspective switch to it
- Open the diagram for the PNCode PNTD (file pncode.aird in folder model) select:
Representation per category → Design → Entities in Class Diagram → pncode class diagram
- In this PNTD, the declaration extension is defined already (all the others are still missing)

- Add the missing concepts for transition (as labels) similarly to the declarations for pages
 - Actions
 - Conditions
- Validate and save and close diagram

Step 1c: Generate the EMF code for the PNTD

- Reload the `pncode.genmodel` and open it
- Generate the “Model” and the “Edit” code

Step 1c: Generate Code

ws - org.pnml.tools.epnk.tutorials.pn-mbse.pntd/model/pncode.genmodel - Eclipse IDE

File Edit Navigate Search Project Generator Run Window Help

Model Explorer

type filter text

- org.pnml.tools.epnk.tutorials.pn-mbse.pntd
 - JRE System Library [J2SE-1.5]
 - Plug-in Dependencies
 - src
 - META-INF
 - model
 - pncode.aird
 - pncode.ecore
 - pncode.genmodel

Properties

Property	Value
Model	
Array Accessors	false
Binary Compatible Reflective Methods	false
Class Name Pattern	
Containment Proxies	false
Feature Delegation	None
Generate Schema	false
Interface Name Pattern	
Minimal Reflective Methods	true
Model Directory	/org.pnml.tools.epnk.tutorials.pn-mbse.pntd/src
Model Plug-in Class	
Model Plug-in ID	org.pnml.tools.epnk.tutorials.pn-mbse.pntd
Model Plug-in Variables	
Operation Reflection	false
Suppress Containment	

Right-click:

- Generate Model Code
- Generate Edit Code

You do not need to generate the “Editor Code” and the “Test Code”.

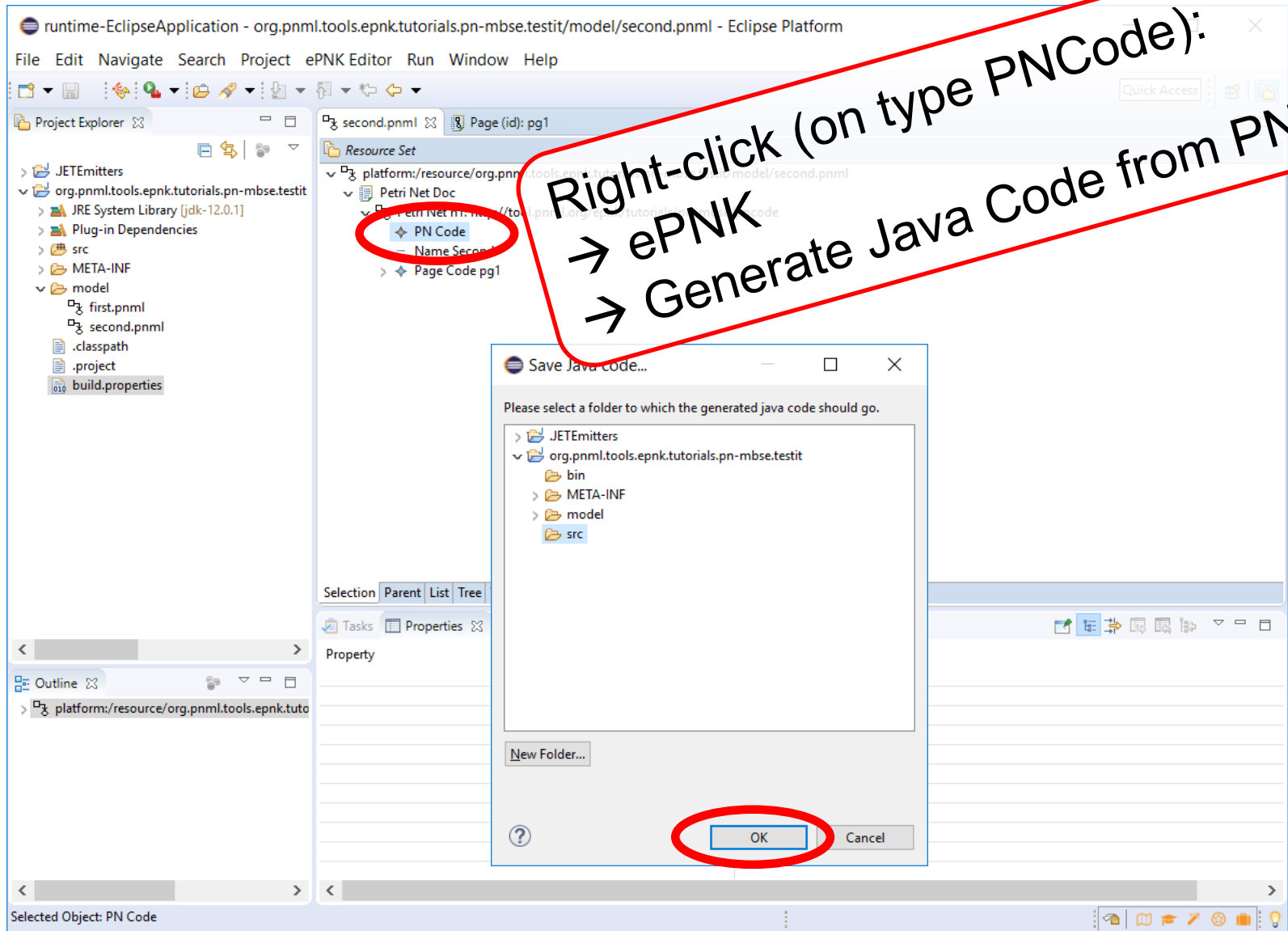
- Start the runtime workspace of your Eclipse (Run / Run Configurations)
- In the runtime workspace, create a new Java Project (since we will generate Java code from our Petri nets) and add a folder called “model”
- Create a new PNML document in this folder, create a new net of your new type PNCode and a name and add a page
- Double click on the page (opening the graphical editor) and see whether you edit a PNCode Petri net with all features: declarations, conditions and actions

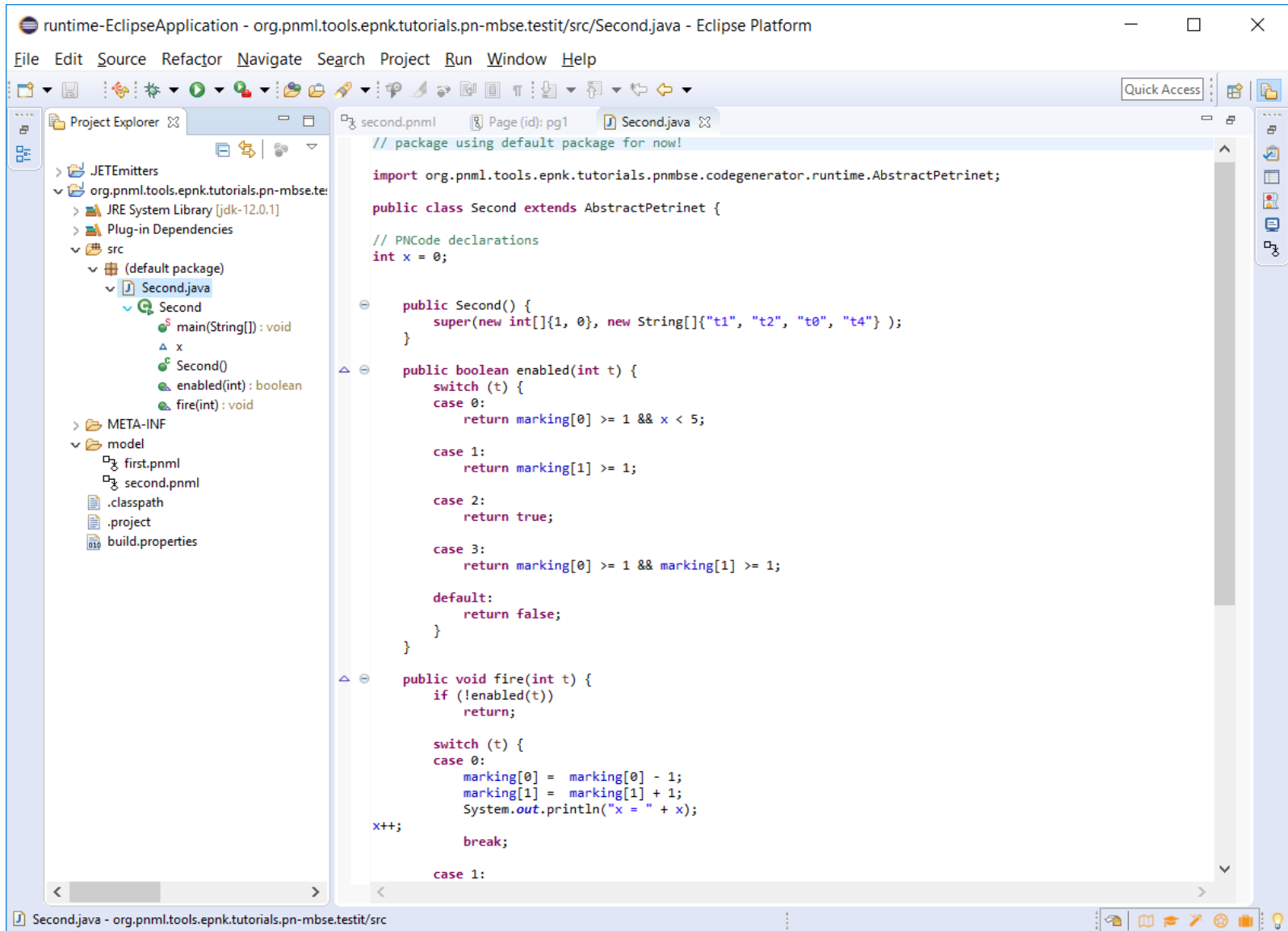
Step 1e: Test your PNTD

The screenshot shows the Eclipse IDE interface with a Petri net diagram open in the editor. The diagram consists of places t_0 , p_1 , t_1 , t_2 , and p_2 . Place t_0 contains one token and is connected to p_1 with the label $x = 0;$. Place p_1 contains one token and is connected to t_1 with the label $x < 5$. Place t_1 is connected to p_2 with the label $\text{System.out.println("x = " + x); x++;}$. Place p_2 is connected to t_2 with the label $\text{System.out.println("x = " + x);}$. Place t_2 is connected back to p_1 with the label t_4 .

The Properties view at the bottom shows the following table:

Core	Property	Value
	ePNK: Graphical information	
	Image	
	Misc	
	Id	pg1
	In	
	Out	





```
// package using default package for now!  
  
import org.pnml.tools.epnk.tutorials.pnmbse.codegenerator.runtime.AbstractPetrinet;  
  
public class Second extends AbstractPetrinet {  
  
    // PNCode declarations  
    int x = 0;  
  
    public Second() {  
        super(new int[]{1, 0}, new String[]{"t1", "t2", "t0", "t4"});  
    }  
  
    public boolean enabled(int t) {  
        switch (t) {  
            case 0:  
                return marking[0] >= 1 && x < 5;  
  
            case 1:  
                return marking[1] >= 1;  
  
            case 2:  
                return true;  
  
            case 3:  
                return marking[0] >= 1 && marking[1] >= 1;  
  
            default:  
                return false;  
        }  
    }  
  
    public void fire(int t) {  
        if (!enabled(t))  
            return;  
  
        switch (t) {  
            case 0:  
                marking[0] = marking[0] - 1;  
                marking[1] = marking[1] + 1;  
                System.out.println("x = " + x);  
                x++;  
                break;  
  
            case 1:  

```

- Implement an action that generates Java code from such PNCode running as a Java application
- The GUI and “runtime” environment (as well as all the set up of the code generation plugin is provided to you
- Also a generator template for the basic Petri net (not taking actions, conditions and declarations into account) is provided to you
- You can focus on extending the template and skeleton for generating the code for actions, conditions and declarations

- Import the prepared code generation project to your Eclipse workspace
 - Download the project from <http://www2.compute.dtu.dk/~ekki/teaching/external/MBSE-PN2019/code/coge-generator.initial.zip>
 - In your Eclipse (EMT) workspace import this project: File → Import... → Existing Project and select the file downloaded above
- Open the imported project `org.pnml.tools.epnk.tutorials.pn-mbse.codegenerator` and inspect it

Step 2a: Prepared Project

The screenshot shows the Eclipse IDE interface. On the left, the Model Explorer displays a project structure for 'org.pnml.tools.epnk.tutorials.pn-mbse.codegenerator'. The 'templates' folder is expanded, and 'PNCODE2Java.javajet' is circled in red. The main editor shows the content of this file, which is a template for generating Java code. A section of the code, starting from line 33, is circled in red. A large red callout bubble with white text points to this section, stating: "This is the part, where the code for the enabledness of transitions is generated: condition code needs to be added here!".

```
1 <% jet package="translated"
2 class="PNCODE2Java"
3 skeleton="PNCODE2JavaSkeleton.java"
4 %>
5 <%
6 PetriNet pn = (PetriNet)
7 initialize(pn);
8 %>
9 // package using default package for now
10
11 import org.pnml.tools.epnk.tutorials.pn-mbse.codegenerator.runtime;
12
13 public class <%= className %> extends AbstractPetriNet {
14
15     public <%= className %>() {
16         super(new int[] {
17             for (int i=0; i < noPlaces; i++) {
18                 <%= i!=0? ", ":"" %><%= getTransitionName(i) %>
19             }
20         });
21     }
22     for (int i=0; i < noTransitions; i++) {
23         <%= i!=0? ", ":"" %><%= getTransitionName(i) %>
24     }
25 }
26
27 public boolean enabled(int t) {
28     switch (t) {
29 <%
30     for (int i=0; i < noTransitions; i++) { %>
31         case <%= i %>:
32             return <%=
33             boolean first = true;
34             for (Arc arc: flatnet.getIn(getTransition(i))) {
35                 <%= !first? " && ":" " %> marking[<%= getPlaceNo(arc.getSource()) %>] >= 1
36                 first=false;
37             }
38             <%= first?"true":"" %>;
39         }
40     }
41 }
42 %>
43
44 default:
45     return false;
46 }
```


- Adjust `PNCode2JavaSkeleton.java` and `PNCode2Java.javajet` so that the code for declarations, conditions and actions is generated

Step 2b: Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Eksamen Kunder

You will find examples in the test-it.zip project (see next slide)

- Produce java code for a simple example net (you can start from the code generated by the example generator; it produces everything except for the code for declarations, conditions and actions)
- Think of methods, which will make it easy for you to get the additional code snippets easily into this code and implement them in the skeleton (some helper functions in the class **NetFunctions** of the `org.pnml.tools.epnk` project might make your life much easier)
- Add a few snippets to the JET template
- Test it (and proceed iteratively)

- You can start the runtime workbench right away, and work back and forth with the generator in the runtime workbench, and adjust the skeleton and template in the development workbench (without restarting the runtime workbench); this will speed up the process.
- In the runtime workbench, you can also import a prepared test project, which has some example nets and Java code generated from it:
<http://www2.compute.dtu.dk/~ekki/teaching/external/MBSE-PN2019/code/test-it.zip>

- I actually, use the generated generator code in the project “**.JETEmitters**” in the runtime workspace to program the additional methods in the skeleton (with IDE-support) and only later copy these methods to the skeleton in the development workspace.

This will be shown live in the hands-on session.