MPS Workshop

Till Schallau (till.schallau@tu-dortmund.de) 14. September 2020

The following exercise are designed to give you an in-depth understanding for the creation of languages using the MPS language workbench.

1 Setup - Console

1. Download the latest version of MPS for your operating system from `https://www.jetbrains.com/de-de/mps/download/`
2. Clone the *mps-workshop* repository using `git@github.com:tillschallau/mps-workshop.git` or `https://github.com/tillschallau/mps-workshop.git`
3. Open the project in MPS with File ≫ Open
4. When prompted: migrate the project
5. Right-click on the language de.tudo.cs.ls14.aqua.mps.workshop.java ≫ Rebuild Language to make the language available for the sandbox solution
6. Right-click on the project mps-workshop ≫ Rebuild project

2 Setup - IDE

1. Download the latest version of MPS for your operating system from `https://www.jetbrains.com/de-de/mps/download/`
2. Open MPS
3. Select Get from Version Control
4. Enter `https://github.com/tillschallau/mps-workshop.git` into the URL field, select a directory and press clone
5. When prompted: migrate the project
6. Right-click on the language de.tudo.cs.ls14.aqua.mps.workshop.java ≫ Rebuild Language to make the language available for the sandbox solution
7. Right-click on the project mps-workshop ≫ Rebuild project

MPS Workshop, AQUA, TU Dortmund

3 MPS Hands-On

These exercises are designed to give you a feeling of how the projectional editor of MPS is working. For the following tasks, work on the ↪ master branch. Navigate to the model sandbox under the solution de.tudo.cs.ls14.aqua.mps.workshop.java and create a new JavaInMPSTest node as shown in Figure 1.

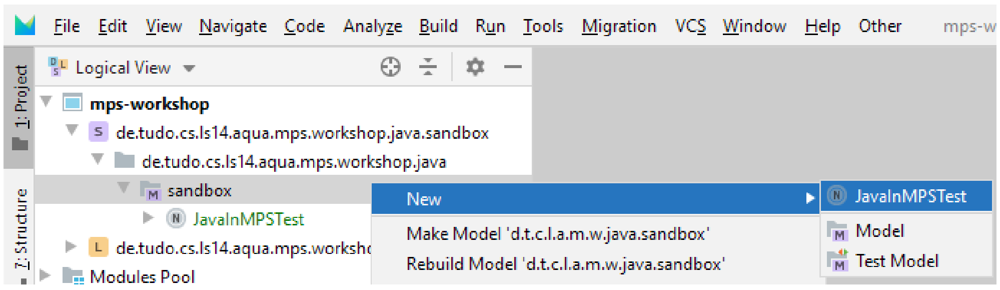
File Edit View Navigate Code Analyze Build Run Tools Migration VCS Window Help Other mps-w
1: Project
7: Structure
Logical View
mps-workshop
de.tudo.cs.ls14.aqua.mps.workshop.java.sandbox
de.tudo.cs.ls14.aqua.mps.workshop.java
sandbox
JavaInMPSTest
de.tudo.cs.ls14.aqua.mps.worksho
Modules Pool
New
Make Model 'd.t.c.l.a.m.w.java.sandbox'
Rebuild Model 'd.t.c.l.a.m.w.java.sandbox'
JavaInMPSTest
Model
Test Model

Figure 1: Navigation menu for creation of a new JavaInMPSTest node

Exercise 3.1. Implement the code shown in Listing 1 in your JavaInMPSTest node. Notice the change in the predefined code fragment while editing the name of the method. After completion rebuild the model by right-clicking sandbox 》 Rebuild model. When there are no more syntax and compilation errors, run the code by right-clicking onto JavaInMPSTest 》 ▶ Run 'Node de.tudo.cs.ls14...'.

Note: You can check the generated code by right-clicking onto the editor and select Preview Generated Text.

```
public static void readLine(){
    String name = "";
    try{
        System.out.println("Hello! Please enter your name."
            ↪ );
        BufferedReader br = new BufferedReader(new
            ↪ InputStreamReader(System.in));
        name = br.readLine();
        System.out.println("Hello " + name);
    } catch(Exception e){
        throw e;
    }
}
```

Listing 1: Java code to read a line from the console and print the result

$$\begin{array}{lcl@{\qquad}lcl}
\text{Class} & ::= & \text{Function}^* & \text{BExp} & ::= & \text{BExp \&\& BExp} \\
 & & & & \mid & \text{BExp} \mid\mid \text{BExp} \\
\text{Function} & ::= & \text{ID} = \text{AExp} & & \mid & \text{AExp} == \text{AExp} \\
 & \mid & \text{ID} = \text{BExp} & & \mid & \text{AExp} \; != \text{AExp} \\
 & & & & \mid & \text{AExp} \geq \text{AExp} \\
\text{AExp} & ::= & \text{AExp} + \text{AExp} & & \mid & \text{AExp} > \text{AExp} \\
 & \mid & \text{AExp} - \text{AExp} & & \mid & \text{AExp} \leq \text{AExp} \\
 & \mid & \text{AExp} * \text{AExp} & & \mid & \text{AExp} < \text{Aexp} \\
 & \mid & \text{AExp} \; / \; \text{AExp} & & \mid & \text{True} \\
 & \mid & \text{INT} & & \mid & \text{False} \\
 & \mid & \text{ID //variable} & & \mid & \text{ID //variable}
\end{array}$$

Table 1: BNF of arithmetic and boolean expressions

Exercise 3.2. Be creative! Add a new JavaInMPSTest node and add your own code.

Note: If your desired package does not exist in your current context, you can add *Accessories Models* by right-clicking the language de.tudo.cs.ls14.aqua.mps.workshop.java 〉 Module Properties. Navigate to Runtime 〉 Accessories Models and add the required models using the + button, e.g. `java.time@java_stub`. You have to rebuild the language de.tudo.cs.ls14.aqua.mps.workshop.java 〉 Rebuild Language to apply your change.

4 Expressions

This section is designed to show you the most important core aspects of MPS. Each sub-exercise can be pulled from their respective branch, such that each task can be done based on a shared code base.

Exercise 4.1. Create a new language named
`de.tudo.cs.ls14.aqua.mps.workshop.expressions` by right-clicking onto mps-workshop 〉 New 〉 Language. Check the Create Sandbox Solution in the process. You have to build the language de.tudo.cs.ls14.aqua.mps.workshop.expressions 〉 Rebuild Language to remove errors.

Exercise 4.2. Translate the BNF of Table 1 into concepts. You may use abstract concepts to simplify the task and add internal structure.
Give each concept a meaningful **name** and **short description**.

For the next exercise you can checkout the branch ↪ expression-0-concepts to work with the shared solution, or work with your own concepts.

Exercise 4.3. Add editors for your concepts, such that the order of operations is clearly visible.

Note: For some visual modifications you may need the Inspector. To show it, click on the respective button in the lower right corner of your IDE.

Intermediate solution: $\boxed{\hookrightarrow \text{expression-1-editors}}$

Exercise 4.4. Add intentions for the `and`- and `or`-Expression which transforms a given node into the other and vice versa.

Intermediate solution: $\boxed{\hookrightarrow \text{expression-2-intentions}}$

Exercise 4.5. Add checks to ensure that each function name is unique. Add checks to ensure that each variable is unique for each function.

Intermediate solution: $\boxed{\hookrightarrow \text{expression-3-checks}}$

Exercise 4.6. Add transformations that automatically append a node into the AST after typing a specific keyword. Write transformations for the `and`- and `or`-Expression for both directions (left and right).

Intermediate solution: $\boxed{\hookrightarrow \text{expression-4-transformations}}$

Exercise 4.7. Add generators for at least one of `AExp` or `BExp`. Your code construct for the expression `x >= 2` should look like the following Listing 2.

The final solution with all steps can be found under $\boxed{\hookrightarrow \text{expression-5-generations}}$

```java
package de.tudo.cs.ls14.aqua.mps.workshop.expressions.sandbox;

/*Generated by MPS */

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class ExpressionTest {
    public static void main(String[] args) {
        test();
    }
    public static void test() {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        int x = -1;

        try {
            System.out.println("Please enter value for int-variable " + "x");
            String line = reader.readLine();
            x = Integer.parseInt(line);
        } catch (Exception e) {
        }
        boolean z = (x >= 2);
        System.out.println("The result of your function test is: " + z);
    }
}
```

Listing 2: Generated Java code that parses the variables and returns the result of the defined expression