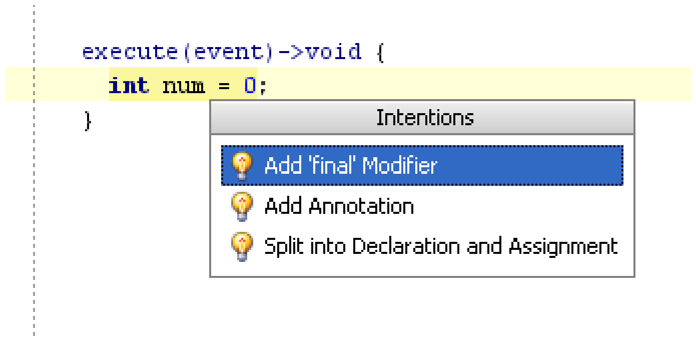
Intentions

Intentions are a very good example of how MPS enables language authors to smoothen the user experience of people using their language. Intentions provide fast access to the most used operations with syntactical constructions of a language, such as "negate boolean", "invert if condition," etc. If you've ever used IntelliJ IDEA's intentions or similar features of any modern IDEs, you will find MPS intentions very familiar.

Using intentions



```
execute(event)->void {
  int num = 0;
}
```

Like in IDEA, if there are avaliable intentions applicable to the code at the current position, a light bulb is shown. To view the list of avaliable intentions, press Alt+Enter or click the light bulb. To apply an intention, either click it or select it and press Enter. This will trigger the intention and alter the code accordingly.
Example: list of applicable intentions

Intention types

All intentions are "shortcuts" of a sort, bringing some operations on node structure closer to the user. Two kinds of intentions can be distinguished: regular intentions (possibly with parameters) and "surround with" intentions.
Generally speaking, there is no technical difference between these types of intentions. They only differ in how they are typically used by the user.

regular intentions are listed on the intentions list (the light bulb) and they directly perform transformations on a node without asking the user for parameters customizing the operations.

"surround with" intentions are used to implement a special kind of transformation - surrounding some node(s) with another construct (e.g. "surround with parenthesis"). These intentions are not offered to the users unless they press ctrl-alt-T (the surr ound with command) on a node. Neither they are shown in general intentions pop-up menu.

Universal Intention is a new experimental feature introduced in 3.4, which allows to unify intentions and parameterized intentions. In addition, it allows to add methods and other class members to intention and has a more java-like editor. As the feature is still in an experimental stage, we decided not to replace the old functionality fully. We still recommend you to use the old intentions, but those, who like the new editor better, can experiment with the new ones. The structure of the universal intention is very similar to the old intentions and using them is very straightforward.

Common Intention Structure

name	The name of an intention. You can choose any name you like, the only obvious constraint being that names must be unique in scope of the model.
for concept	Intention will be tested for applicability only to nodes that are instances of this concept and its subconcepts.
available in child nodes	Suppose N is a node for which the intention can be applied. If this flag is set to false, intention will be visible only when the cursor is over the node N itself. If set to true, it will be also visible in N's descendants (but still will be applied to N)
child filter	Used to show an intention only in some children. E.g. "make method final" intention is better not to be shown inside method's body, but preferrably to be shown in the whole header, including "public" child.
description	The value returned by this function is what users will see in the list of intentions.
isApplicable	Intentions that have passed the "for concept" test are tested for applicability to the current node. If this method returns "true," the intention is shown in the list and can be applied. Otherwise the intention is not shown in the list. The node argument of this method is guaranteed to be an instance of the concept specified in "for concept" or one of its subconcepts.
execute	This method performs a code transformation. It is guaranteed that the node parameter has passed the "for concept" and "is applicable" tests.

Regular Intentions

is error intention - This flag is responsible for an intention's presentation. It distinguishes two types of intentions - "error" intentions which correct some errors in the code (e.g. a missing 'cast') and "regular" intentions, which are intended to help the user perform some genuine code transformations. To visually distinguish the two types, error intentions are shown with a red bulb, instead of an orange one, and are placed above regular intentions in the applicable intentions list.

Parameterized regular intentions

Intentions can sometimes be very close to one another. They may all need to perform the same transformation with a node, just slightly differently. E.g. all "Add ... macro" intentions in the generator ultimately add a macro, but the added macro itself is different for different intentions. This is the case when parameterized intention is needed. Instead of creating separate intentions, you create a single intention and allow for its parametrization. The intention has a parameter function, which returns a list of parameter values. Based on the list, a number of intentions are created , each with a diferent parameter value. The parameter values can then be accessed in almost every intention's method.

> Note
> You don't have an access to the parameter in the isApplicable function. This is because of performance reasons. As isA pplicable is executed very often and delays would quickly become noticeable by the user, you should perform only base checks in isApplicable. All parameter-dependent checks should be performed in the parameter function, and if a check was not passed, this parameter should not be returned

Surround With - Intentions

This type of intentions is very similar to regular intentions and all the mentioned details apply to these intentions as well.

Where to store my intentions?

You can create intentions in any model by importing the intentions language. However, MPS collects intentions only from the In tentions language aspects. If you want your intentions to be used by the MPS intentions subsystem, they must be stored in the Intentions aspect of your language.

Previous Next