Console

Console is a tool which allows developers to conveniently run DSL code directly in the MPS environment against the active models. It enables you to quickly query the model and change it. You can trigger actions against your models or study statistics about your project.

For example, you can quickly get all (or some) instances of a (deprecated) concept and migrate them to a new concept:

```
#instances(TryStatement).where({~it => it.catchClause.isNotEmpty; }).refactor({~node =>
  if (node.body.statement.size > 5) {
    node.replace with new(TryCatchStatement);
  }
})
```

The Console tool window allows line-by-line execution of any DSL construction in the realtime. After the command is written in the console, it is generated by the MPS generator and executed in the IDE's context. This way the code in the console can access and modify the program's AST, display project statistic, execute IDE actions, launch code generation or initiate classes reloading.

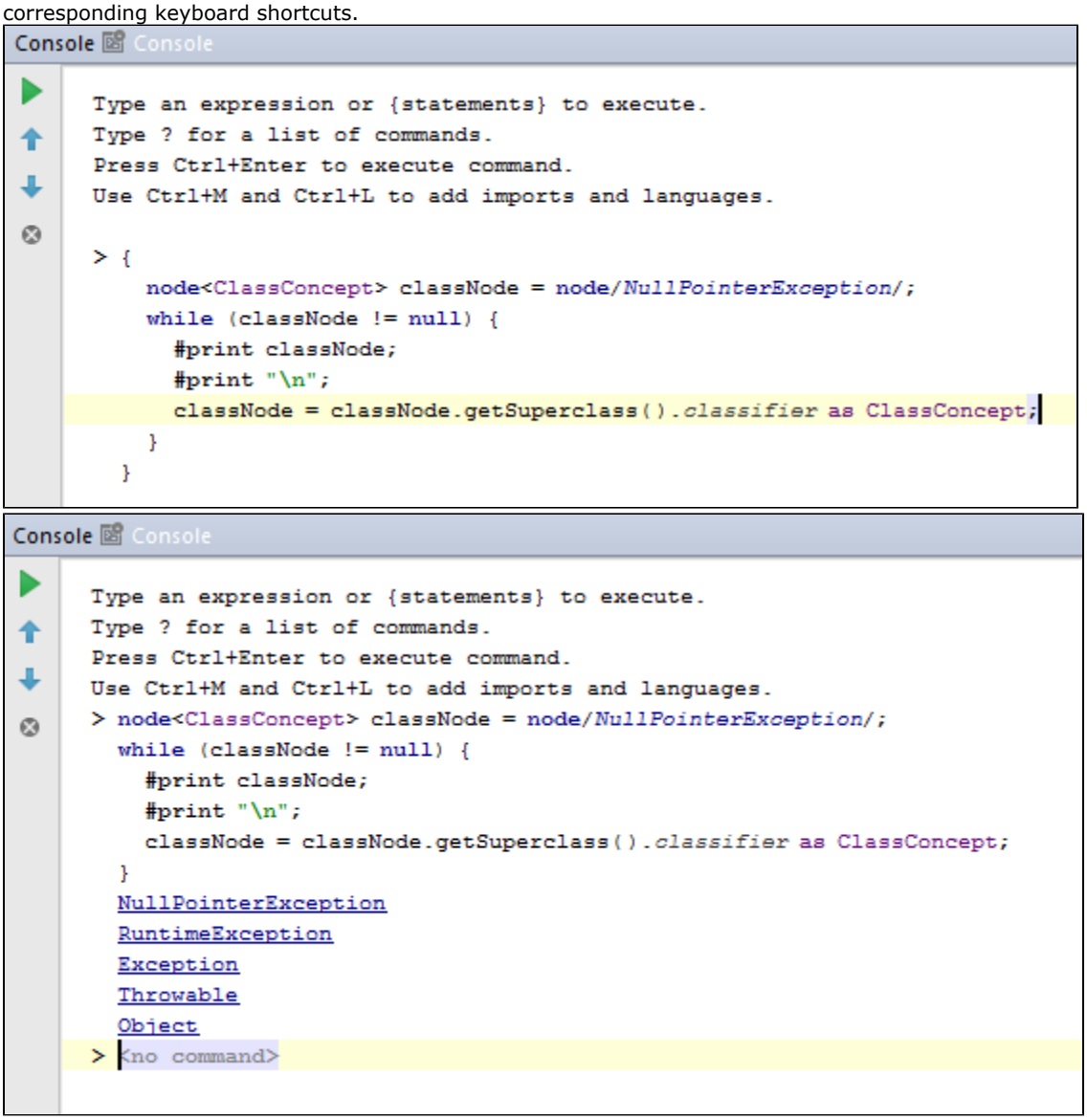
For discoverability reasons, most of the console-specific DSL constructs start with symbol '#'. Code-completion (Control + Space) assists developers to insert code in the console.

We shot a short informative screen-cast about using the MPS Console to investigate and update the user models. Check it out!

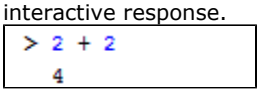
In general, there are 3 kinds of commands:

1. BaseLanguage statement lists. These commands can contain any BaseLanguage constructions. If some construction or class is not available in completion, it may not have been imported. Missing imports can easily be added as in the normal editor, using actions 'Add model import', 'Add model import by root', 'Add language import', or by the

corresponding keyboard shortcuts.



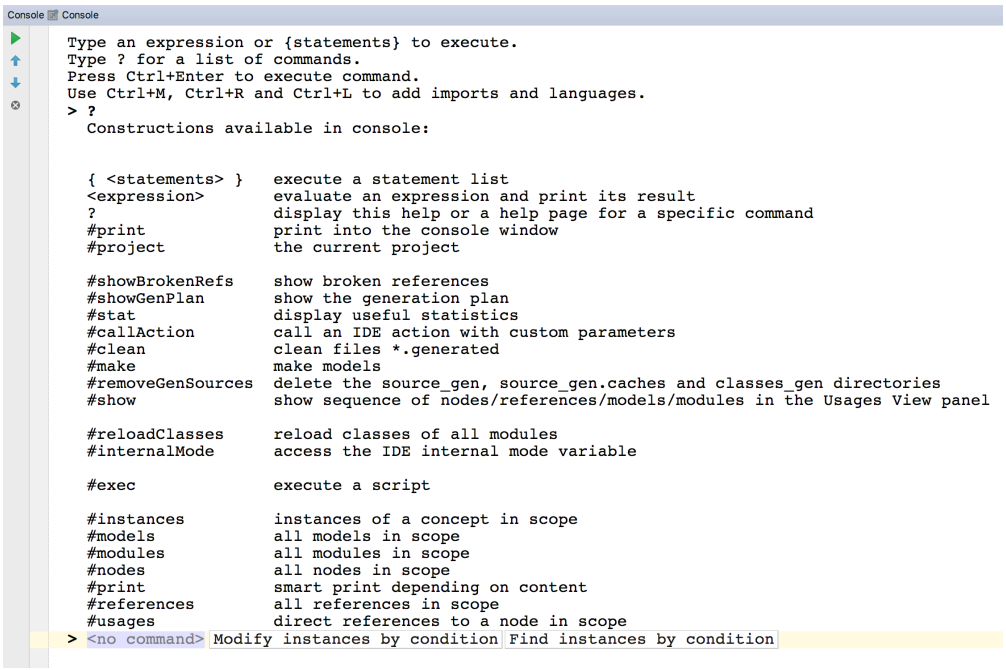
2. BaseLanguage expressions. Expression is evaluated and, if its type is not void, printed in console as text, AST, or interactive response.



3. Non-BaseLanguage commands. These are simple non-customizable commands, such as #reloadClasses.

Console commands

There is also a set of languages containing the console commands and BaseLanguage constructions, which allow developers to easily make custom refactorings, complex usages search etc.



1. BaseLanguage constructions for iterating over IDE objects (#nodes, #references, #models, #modules). These expressions are lazy sequences, including all nodes/references/models/modules in project or in custom scope.

```
> #nodes<scope = models:[model/jetbrains.mps.baseLanguage.structure/], r/o+>.si
  783
> #models<scope = global, r/o+>.where({~it => it.name.startsWith("jetbrains");
  1581 models
```

> To inspect read-only modules and models, such as imported libraries and used languages, you need to include the r/o+ parameter to the desired search scope.

2. BaseLanguage constructions for usages searching (#usages, #instances). These expressions are also sequences, which can be iterated over, but not lazy. When these expressions are evaluated, find usages mechanism is called, so it runs faster then iterating over all nodes or references and then filtering by concept/target.

```
> #usages<scope = project>(node/SwingUtilities/ -> invokeLater)
  49 references
> #instances(DotExpression).where({~it => it.operation.isInstanceOf(AllOperation); })
  50 nodes
```

3. Commands for quering data from the IDE (#stat, #showBrokenRefs, #showGenPlan)
4. Commands for interacting with the IDE (#reloadClasses, #make, #clean, #removeGenSources)

> Te initiate a rebuild of a model, first invoke #clean followed by #make.

5. BaseLanguage constructions for showing results to user
   - #show expression opens usages view and shows there nodes, models or modules from sequence passed to the expression as a parameter.
   - #print expression writes result to the console. There are also specialized versions of this construction:
     - #printText converts result to string and add it to the response.
     - #printNode is applicable only to nodes. This construction adds to response the the whole node and its subnodes. Since the response is also part of the AST, the node is displayed with its normal editor.
     - #printNodeRef makes sense with only nodes locates in the project models. This construction prints to the console an interactive response, which can be clicked on in order to open the node in the editor.
     - #printSeq is applicable to collections of nodes, models or modules. This command prints to the console an interactive response, which describes the size of the collection. When the response is clicked on, the usage view opens to show the nodes or the models.
     - #print expression is a universal construction, which tries to choose the most appropriate way of displaying its argument, according to its type and value
6. refactor operation. This operation applies a function to sequence of nodes (like forEach operation), but before that it opens the found nodes in the usages view, where user can review the nodes before the refactoring is started and manually select the nodes to include/exclude in the refactoring and then apply or cancel the refactoring.

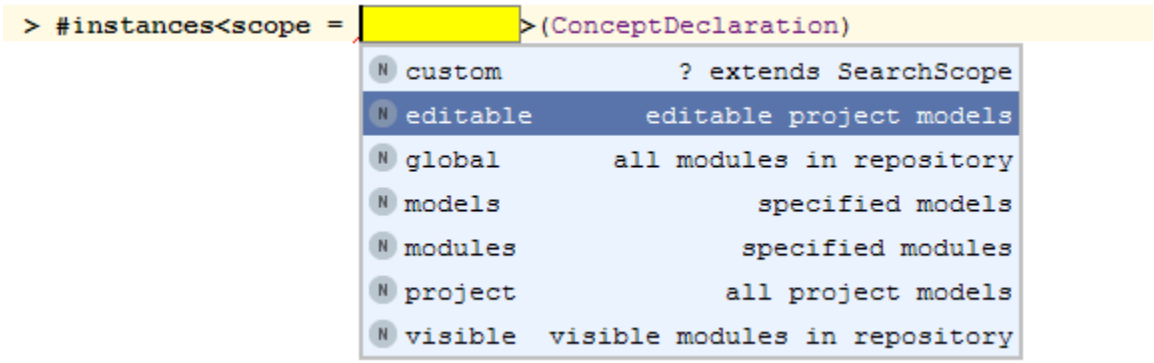
```
> #instances(ClassConcept).where({~node =>
    node.name.isEmpty || Character.isLowerCase(node.name.charAt(0));
  }).refactor({~node =>
    node.name = Character.toUpperCase(node.name.charAt(0)) + node.name.substring(1, node.na
  })
```

Additionally, the console languages can be extended by the user, if needed.

> The model-querying commands used in the Console are defined in the jetbrains.mps.lang.smodel.query language. After importing the language you can use the commands in code to programmatically access and query the models, as well. Details can be found in the smodel queries documentation.

Scopes of Console queries

Queries can have scope specified to constraint the area of the repository to search. The scope is specified in pointy brackets:



- project - only the modules in the current project
- editable - only editable models in the current project (this is the default scope used when no scope is specified explicitly)
- global - the whole module repository
- visible - only visible modules from global repository
- modules - restrict to the listed modules
- models - restrict to the listed models
- custom - use the provided Java Scope class to filter the modules

Copying nodes into Console

In order to point to a concrete node in project from the console, this node can be copied from the editor and then pasted into the console. The node will be pasted as a special construction, called nodeRef, with is a BaseLanguage expression of type node

<>, with value of the pasted node. If there is a necessity to paste the piece of code as is, the 'Paste Original Node' action is available from the context menu.