Other languages

Here we introduce some handy BaseLanguage extensions

Checked dots

Language: jetbrains.mps.baseLanguage.checkedDots

A Checked Dot Expression is an dot expression extended with null checks on the operand.

If the operand is null, the value of the whole checked dot expression becomes null, otherwise it evaluates to the value of corresponding dot expression.

Ways to create a Checked Dot Expression

- The Make dot expression checked intention
- Enter "?" after dot, e.g. customer.?address.?street
- Left transform of operation with "?"

You can transform checked dot expressions to the usual dot expressions using the Make dot expression not checked intention

Overloaded operators

Language: jetbrains.mps.baseLanguage.overloadedOperators

This language provides a way to overload binary operators.

Overloaded operator declarations are stored in an OverloadedOperatorContainer.

If there are several overloaded versions for one operator the most relevant is chosen.

Note that if an overloaded operators' usage is in the other model than its declaration, overloadedOperators language should be added to "languages engaged on generation" of usage's model.

Examples

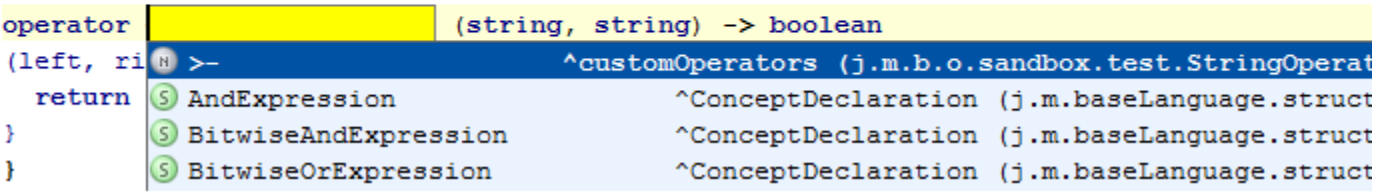
Overloading plus operator for class Complex:

```
operator + (Complex, Complex) -> Complex
(left, right)->Complex {
  Complex res = new Complex();
  res.set(left.getRe() + right.getRe(), left.getIm() + right.getIm());
  res;
}
```

Also, you can define your own custom operators. Assume we want to create a binary boolean operator for strings, which tells if one string contains another:

```
overloaded operators StringOperators {

custom operator >-
```



Now, we can simply use this operator:

```
if (str >- "abc") {

}
```

Custom constructors (since 1.5.1)

Language: jetbrains.mps.baseLanguage.constructors

Custom constructors provide a simple way to create complex objects. They are stored in a special root node - CustomConstructorsContainer.

Example

Assume we need a faster way to create rectangle.

```
custom constructors Rectangle

custom constructor Point
short description: <no shortDescription>
( double x, double y ) -> Point2D
()->Point2D {
  return new Point2D.Double(x, y);
}
separator: ,

custom constructor Rectangle
short description: <no shortDescription>
[ Point2D topleft, Point2D bottomright ] -> Rectangle2D
()->Rectangle2D {
  double x = topleft.getX();
  double y = topleft.getY();
  return new Rectangle2D.Double(x, y, bottomright.getX() - x, bottomright.getY() - y
}
separator: >
```

Now, let's create a rectangle:

```
Rectangle2D r = [(2, 3)> (4, 6)];
```

Previous Next