Предметно - ориентированное программирование

Степулёнок Денис Олегович

Искусственный интеллект

- Автоматизация “рутинных” задач, “скучных” для человека
- Автономные системы
- “Быстрое” управление

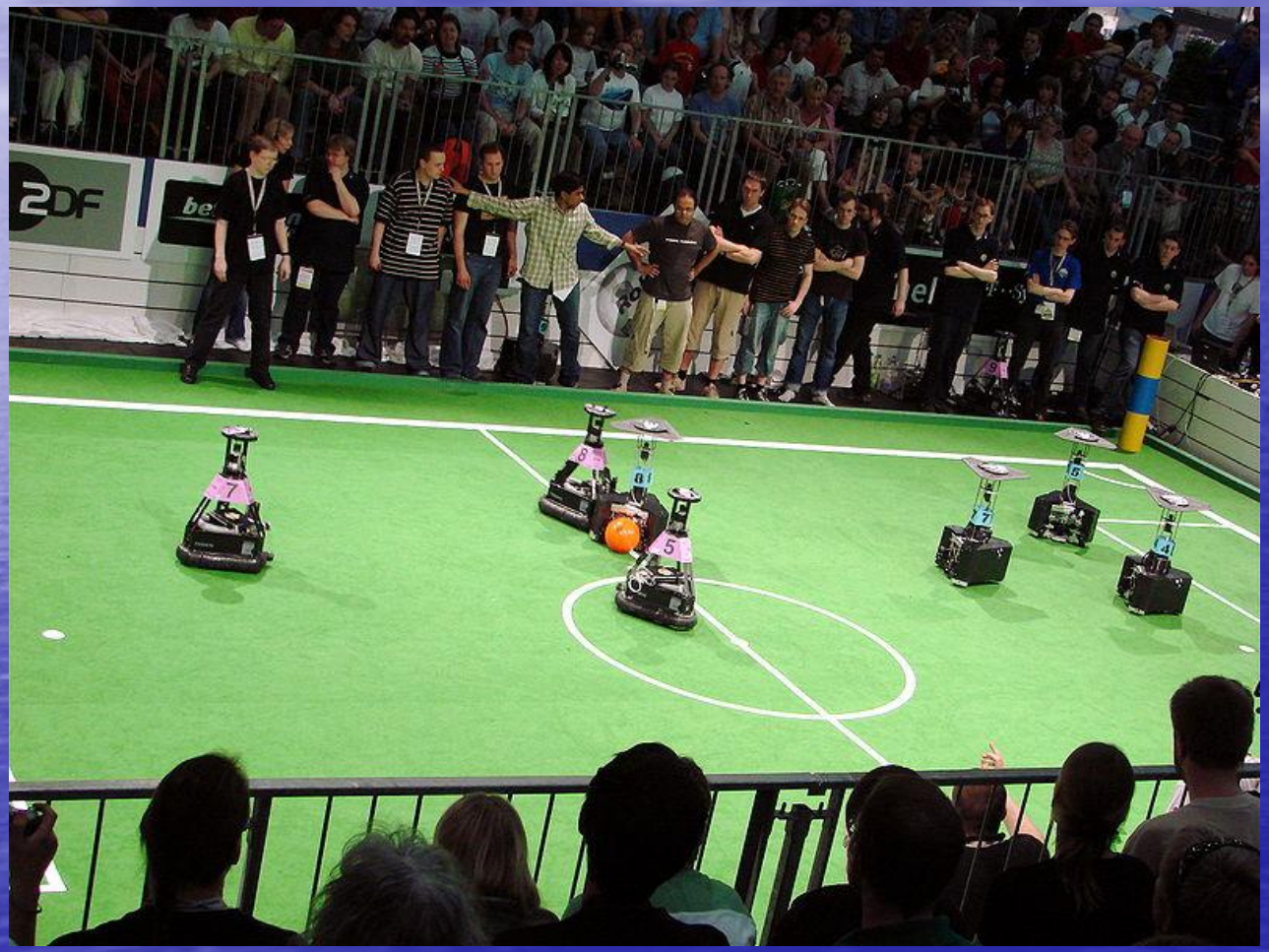
Фантастика: разговор с компьютером

«Идеальный» компьютер «понимает» естественный язык человека. «Программирование» на естественном языке

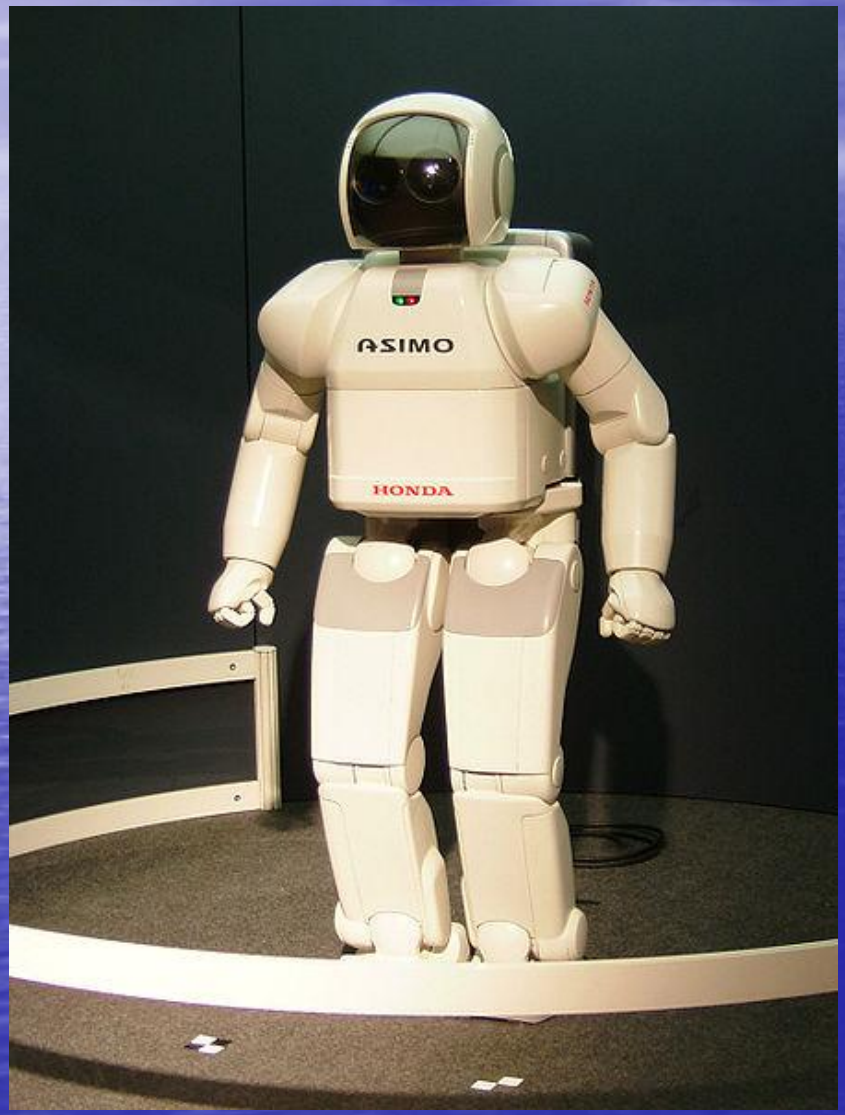
Deep Blue – шахматный суперкомпьютер

- 11 мая 1997 года одержал победу в матче из 6 партий с чемпионом мира по шахматам Гарри Каспаровым
- Разработан компанией IBM
- Название получил от «Deep Thought» (глубокая мысль) из романа Дугласа Адамса «Автостопом по галактике» и «клички» IBM: «Big Blue»
- После матча с чемпионом Deep Blue был разобран.
- В основе Deep Blue II находится сервер RS/6000 фирмы IBM, у которого имеется 31 процессор. Один процессор объявлен главным, а ему подчиняются 30 остальных. К каждому из этих 30 процессоров подключено 16 специализированных шахматных процессора. Таким образом всего имеется 480 шахматных процессоров

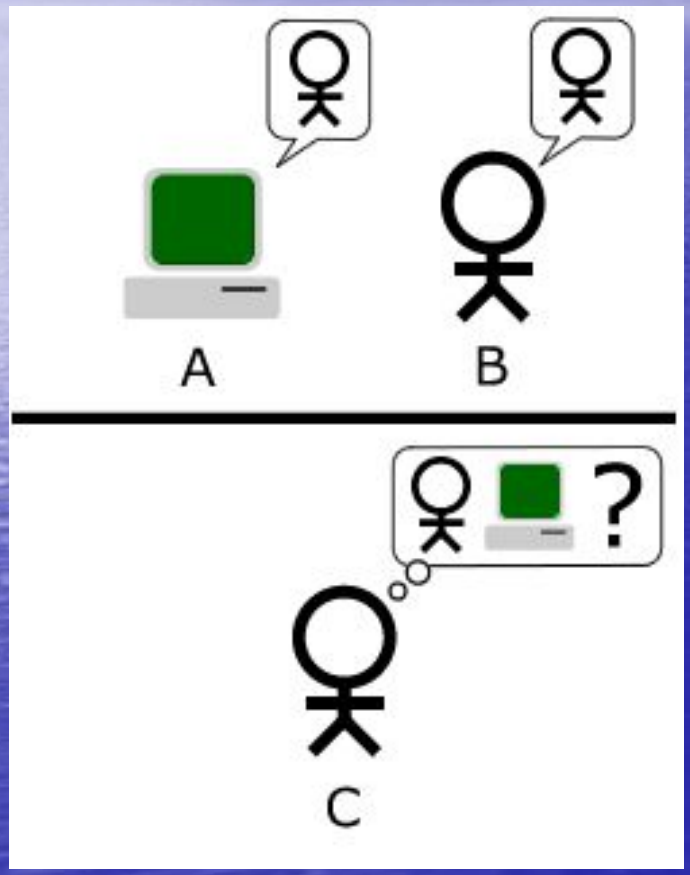
Применение ИИ - турнир RoboCup



Гуманоидный робот



Тест Тьюринга



Основные подхода к разработке ИИ:

- нисходящий (англ. Top-Down AI), семиотический — создание экспертных систем, баз знаний и систем логического вывода, имитирующих высокоуровневые психические процессы: мышление, рассуждение, речь, эмоции, творчество и т. д.;
- восходящий (англ. Bottom-Up AI), биологический — изучение нейронных сетей и эволюционных вычислений, моделирующих интеллектуальное поведение на основе биологических элементов, а также создание соответствующих вычислительных систем, таких как нейрокомпьютер или биокомпьютер.

Уровни языков программирования

Естественные языки
Терминология конкретной предметной области

Высокоуровневые языки программирования

Низкоуровневые языки программирования (напр. C)

Ассемблер (мнемокоды)
Машинный код (то, что исполняет процессор)

Языки высокого/низкого уровня

***Языки высокого уровня** – максимально приближены к задаче. Наиболее выражено в предметно ориентированных языках.*

*Приоритет – **что?***

***Языки низкого уровня** – в центре внимания не задача, а технология её реализации, связанная с языком / машиной. Привлекаются дополнительные понятия, не связанные с задачей.*

*Приоритет – **как?***

Разнообразие языков программирования

Со времени создания первых программируемых машин человечество придумало > 8500 языков программирования. Каждый год их число пополняется новыми.

Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей.

Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Парадигмы программирования

* Архитектура машины, ассемблер
* Формула-ориентированные
* Процедурное, структурное программирование
* Объектно-ориентированное программирование
* Языково-ориентированное программирование

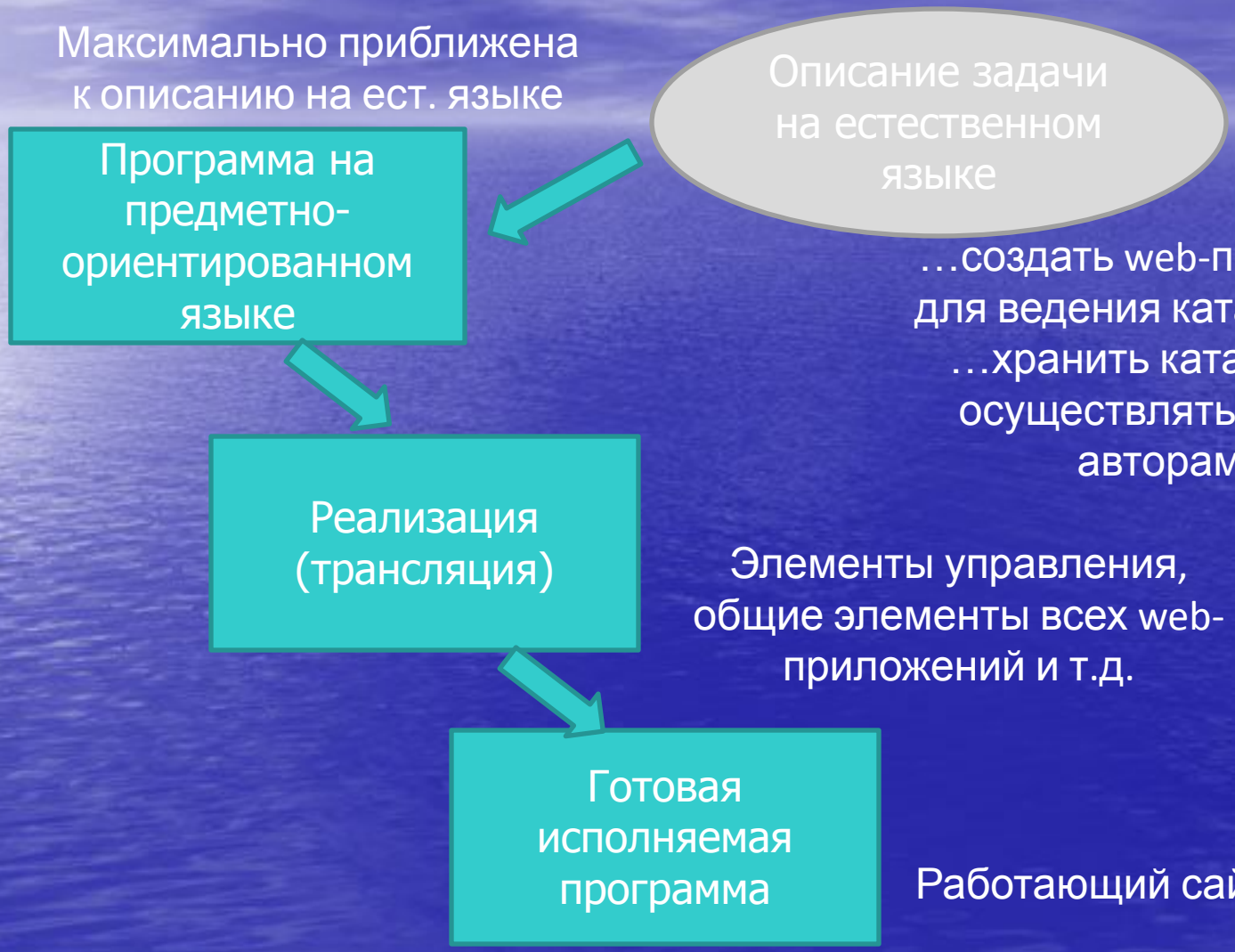
Сейчас

Классификации языков

Программист как преобразователь контекста

Предметно-ориентированный подход

Максимально приближена к описанию на ест. языке



…создать web-приложение для ведения каталога книг..
…хранить каталог книг..
осуществлять поиск по авторам …

Элементы управления, общие элементы всех web-приложений и т.д.

Работающий сайт

ПОЯ

Предметно-ориентированный язык программирования (англ. *domain-specific programming language, domain-specific language, DSL*) — язык программирования, специально разработанный для решения определённого круга задач, в отличие от *языков программирования общего назначения*, например C или Java, или *языков моделирования общего назначения* наподобие UML.

Требования к ПО

- **Функциональность:** программа должна выполнять ожидаемые функции. Функции нужно реализовывать в порядке их необходимости заказчику, чтобы это сделать, необходимо «вникнуть» в предметную область и определить, что действительно важно заказчику; выделить главные, вспомогательные и второстепенные функции.
- **Надежность:** необходимо обеспечить минимум ошибок, сбоев, защиту информации от непреднамеренной порчи. Программа должна разумно реагировать на ввод пользователем любых данных, иначе пользователи будут «бояться» ошибиться, что, как минимум, замедлит их работу с системой. Кроме того, программы, предназначенные для работы в компьютерных сетях, требуют защиты от разного рода вирусных и хакерских атак, – программа должна проверять все входящие данные, особенно тщательно те, которые приходят по сети.
- **Удобство:** программа должна иметь интуитивно-понятный интерфейс, удобный для пользователя. Нужно минимизировать количество действий, необходимых пользователю для выполнения задачи, но не в ущерб понятности самих действий.
- **Эффективность**: программа должна эффективно использовать память, процессор, «жёсткий» диск и другие ресурсы системы. В особых случаях (когда «медлительность» системы критична) это требование становится едва ли не самым важным, важнее надёжности, но чаще оно менее существенно, чем сопровождение.
- **Сопровождение:** программа должна быть понятной, гибкой и «простой в сопровождении, переносе на новые платформы и развитии» - это требование программистов, в отличие от предыдущих, которые являются требованиями заказчиков, пользователей. Удобство сопровождения, как правило, находится в противоречии с эффективностью и для большинства программ более важно. Понятность программы позволяет быстро её развивать, добавлять новые функции, исправлять ошибки.

Виды ПОЯ

ПОЯ могут быть:

- графическими (процесс программирования – «рисование» схемы в специальном редакторе);
- текстовыми (программирование – составление текста на некотором формальном языке).

Кроме того, ПОЯ можно разделить на:

- статические – языки, в которых не важно, в каком порядке программист рисует элементы схемы или составляет текст программы;
- динамические – имеет значение порядок действий (например, в Geometer's Sketchpad последовательность геометрического построения задаёт алгоритм).

Создание ПОЯ

- Проектирование языка программирования
- Реализация интерпретаторов (компиляторов) для выбранных платформ
- Создание редактора (среды разработки)

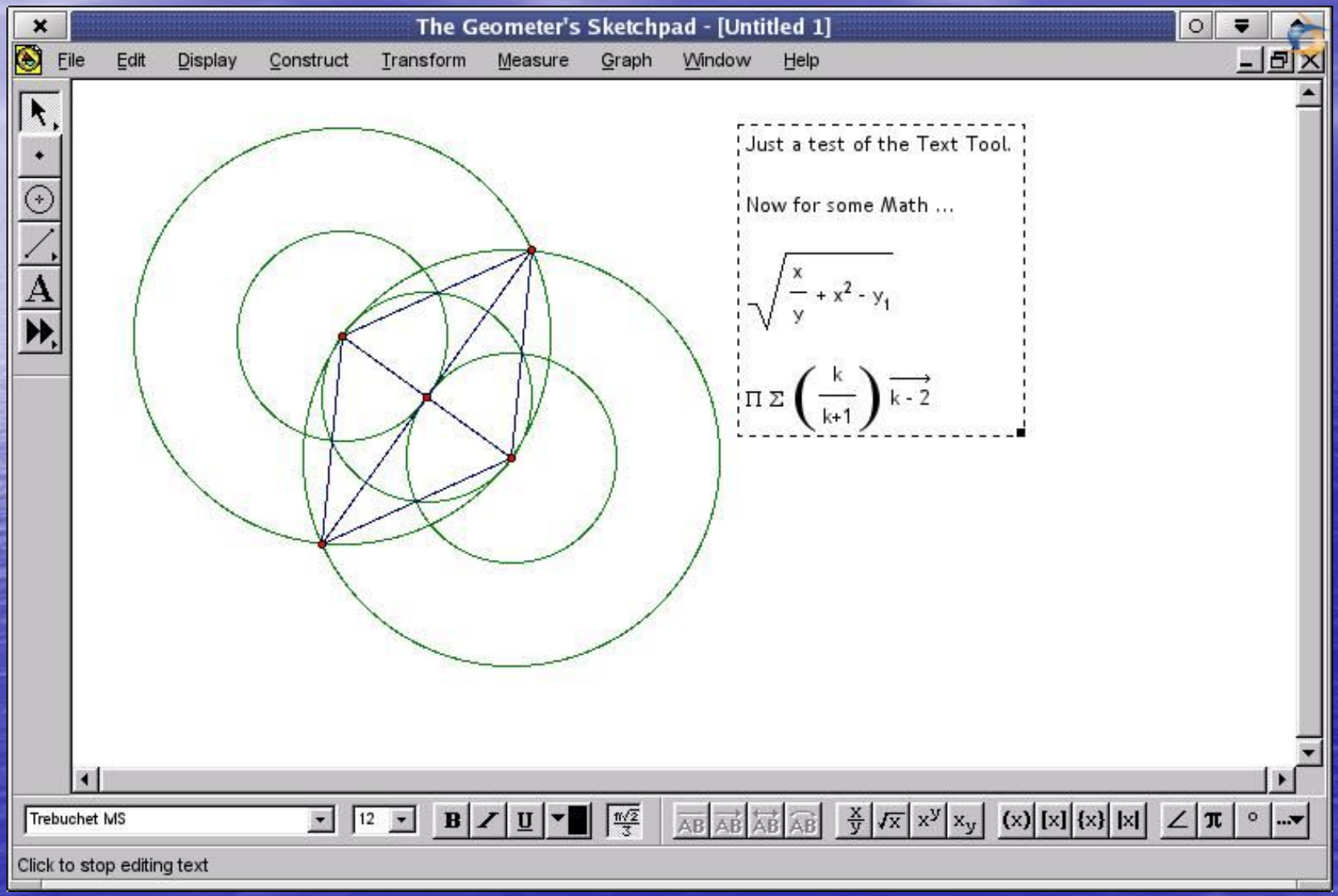
  При создании ПОЯ нужно разработать не только сам язык, но и среду программирования (IDE), удобный редактор кода, отладчик, профилировщик и т.д.

  Тут возможно несколько подходов.

* Использовать достаточно гибкий существующий универсальный Язык Программирования (C#, Java, Python) и добавить предметно-ориентированные возможности при помощи библиотеки компонентов (Framework'a);
* Использовать существующую систему для создания предметно-ориентированных языков (Meta Programming System от JetBrains или DSL от Microsoft – позволяет рисовать графические схемы и генерировать по ним код);
* Использовать среду программирования, в которой синтаксические конструкции языка можно модифицировать (настраивать процесс компиляции на определенную предметную область). Например, среда Phoenix, дополнение к компиляторам .Net;
* Создать собственный ЯП с компилятором, отладчиком и т.д. Для генерации лексического анализатора можно использовать продукты **lex** и **yacc**;

Процесс компиляции

Geometer's Sketchpad



КОМПЬЮТЕРНЫЕ ИНСТРУМЕНТЫ В ОБРАЗОВАНИИ

МАТЕМАТИКА

Лабораторные работы по алгебре и началам анализа 10-11 классы

ЛАБОРАТОРИЯ

«Качели»
Задача с конкурса КИО-2008

ГЕОМЕТРИЯ

1С Математический конструктор

HTML-АЛЬБОМ

Маршрутные огни трамваев Альбом «Санкт-Петербург»

ФИЛЬМ

Кубистский паркет

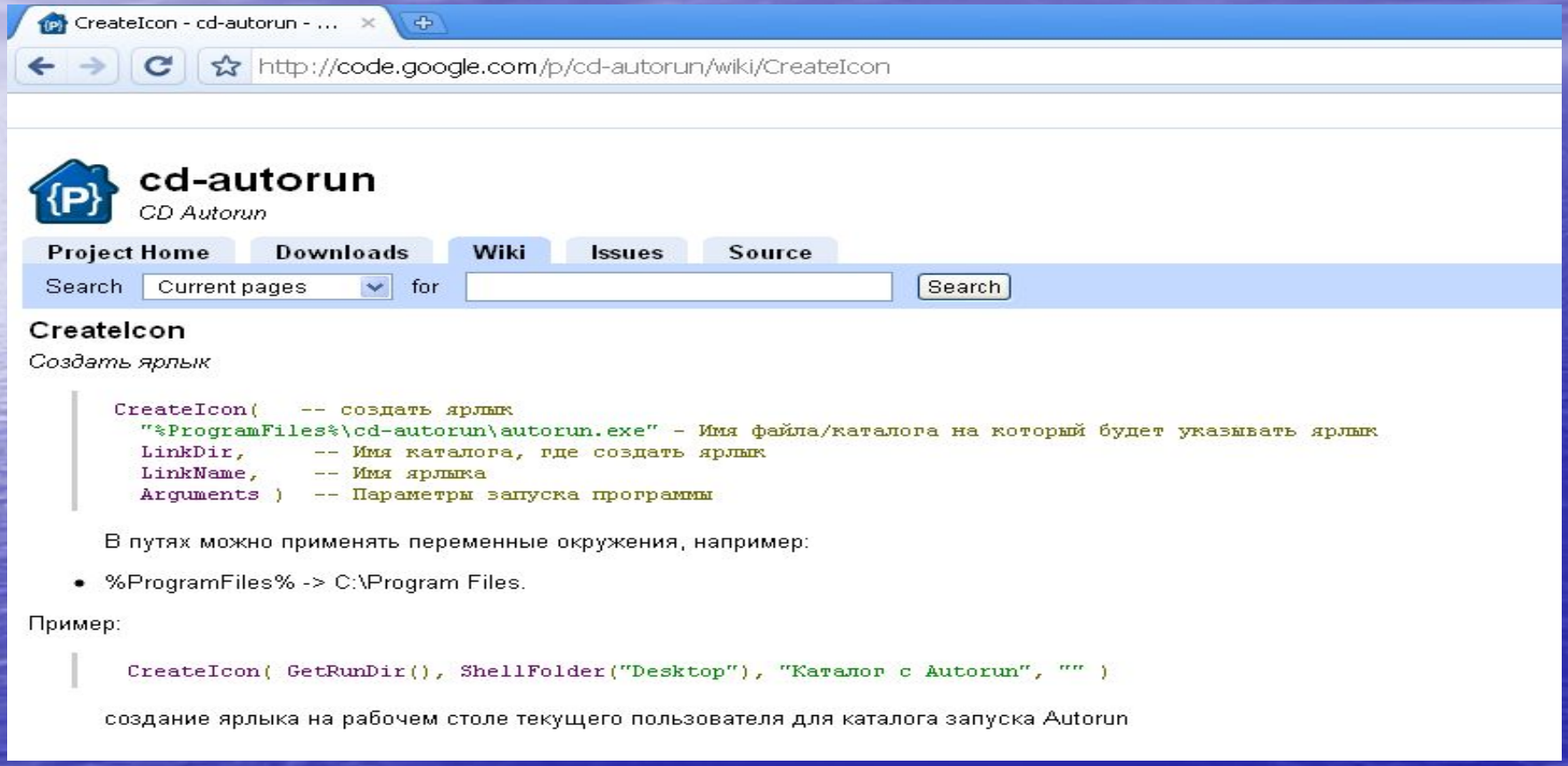
ИНСТРУМЕНТ

Система аспектно--ориентированного программирования

2

САНКТ-ПЕТЕРБУРГ 2008

Пример: Создание иконки



CreateIcon - cd-autorun - ...

http://code.google.com/p/cd-autorun/wiki/CreateIcon

cd-autorun

CD Autorun

Project Home | Downloads | Wiki | Issues | Source

Search | Current pages | for | Search

CreateIcon

Создать ярлык

```
CreateIcon(    -- создать ярлык
  "%ProgramFiles%\cd-autorun\autorun.exe" - Имя файла/каталога на который будет указывать ярлык
  LinkDir,      -- Имя каталога, где создать ярлык
  LinkName,     -- Имя ярлыка
  Arguments )   -- Параметры запуска программы
```

В путях можно применять переменные окружения, например:

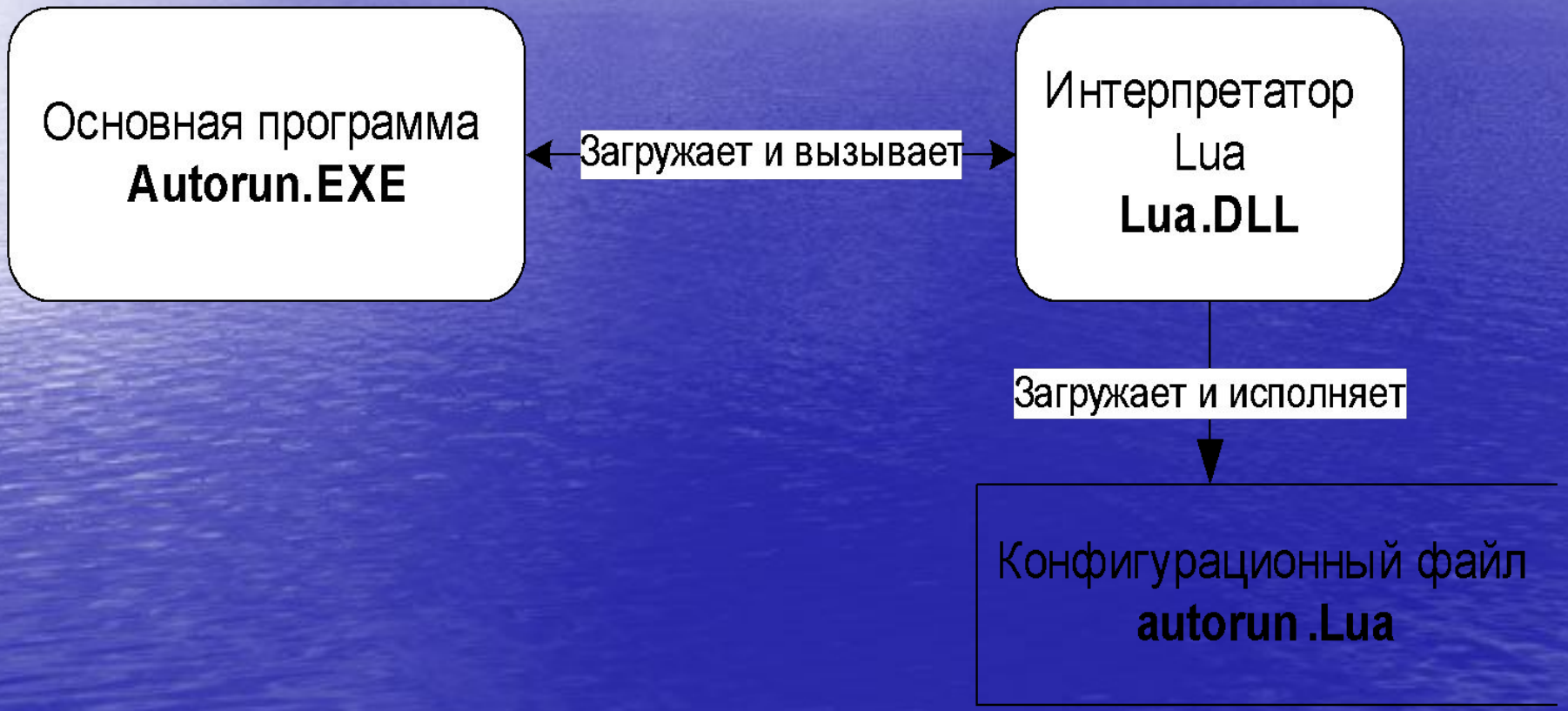
- %ProgramFiles% -> C:\Program Files.

Пример:

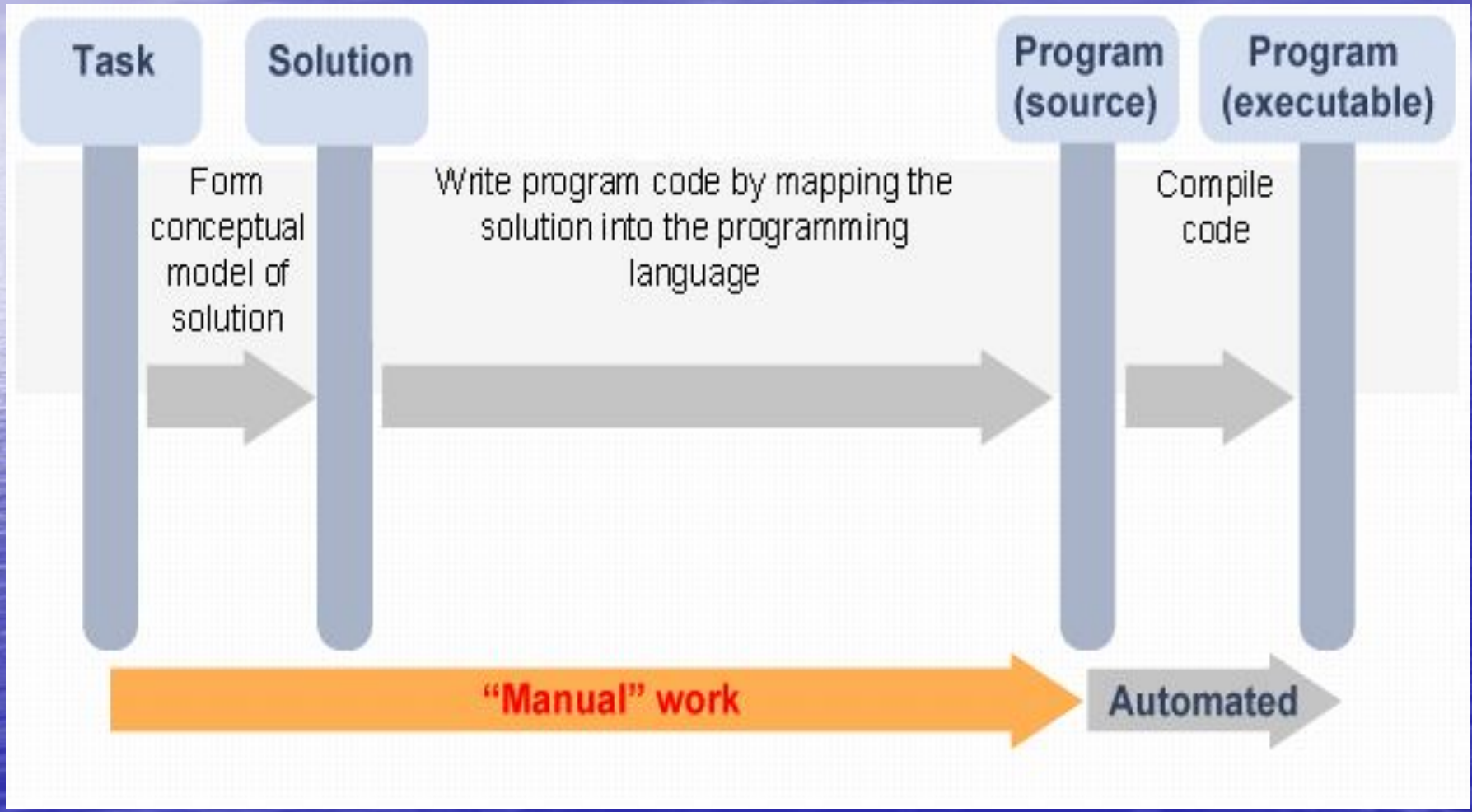
```
CreateIcon( GetRunDir(), ShellFolder("Desktop"), "Каталог с Autorun", "" )
```

создание ярлыка на рабочем столе текущего пользователя для каталога запуска Autorun

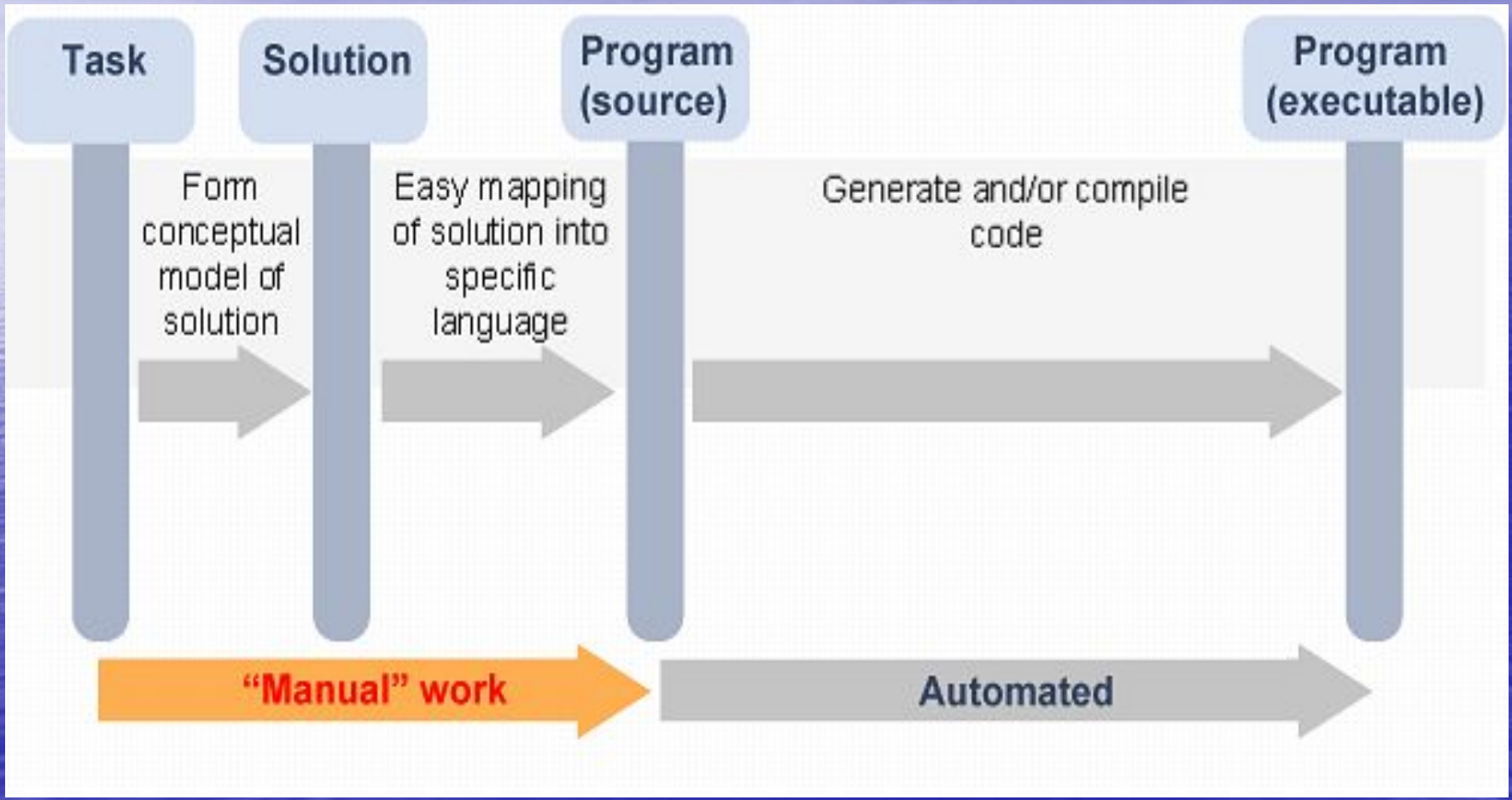
Схема работы «Журнала в журнале»



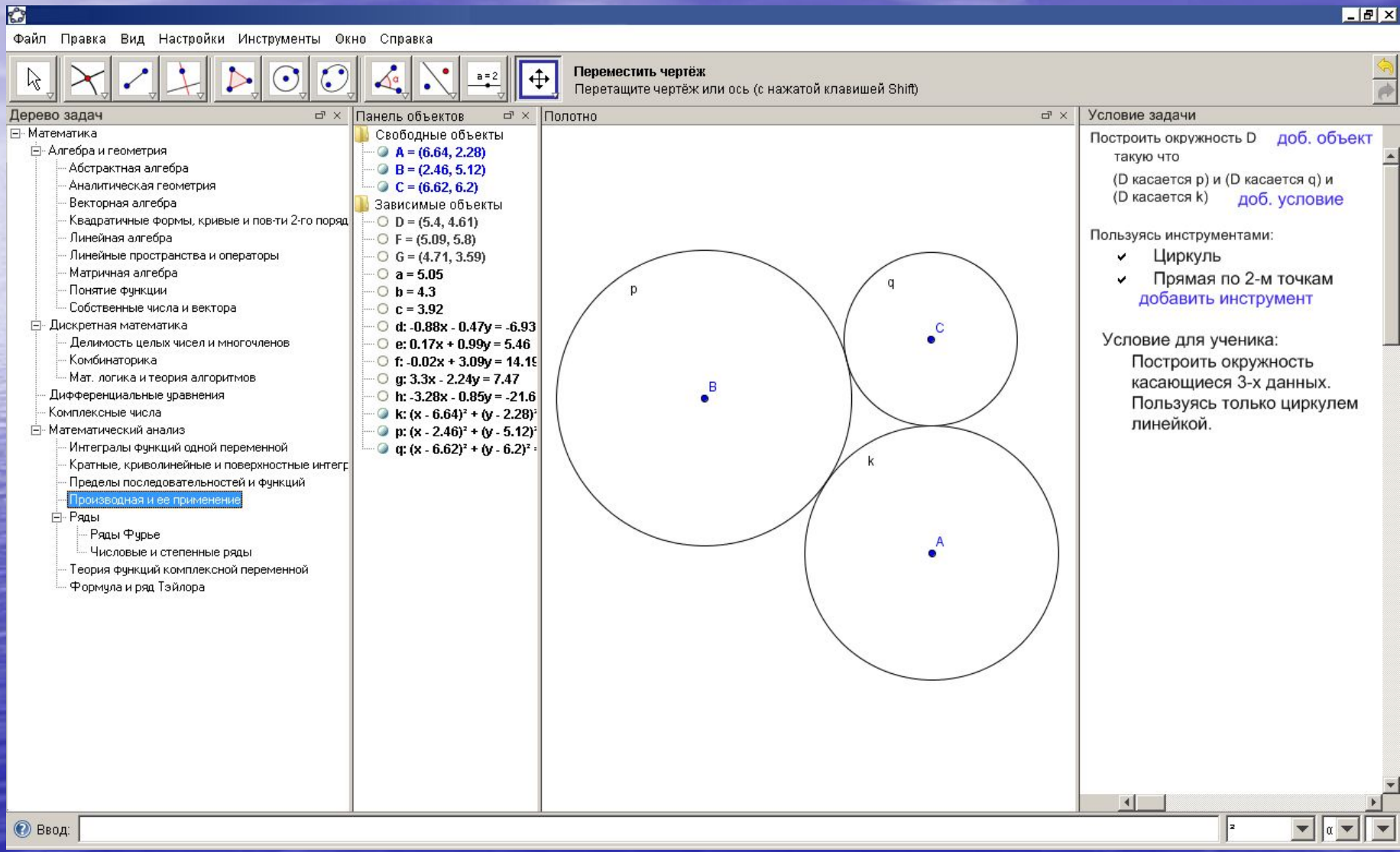
Программирование на языке общего назначения



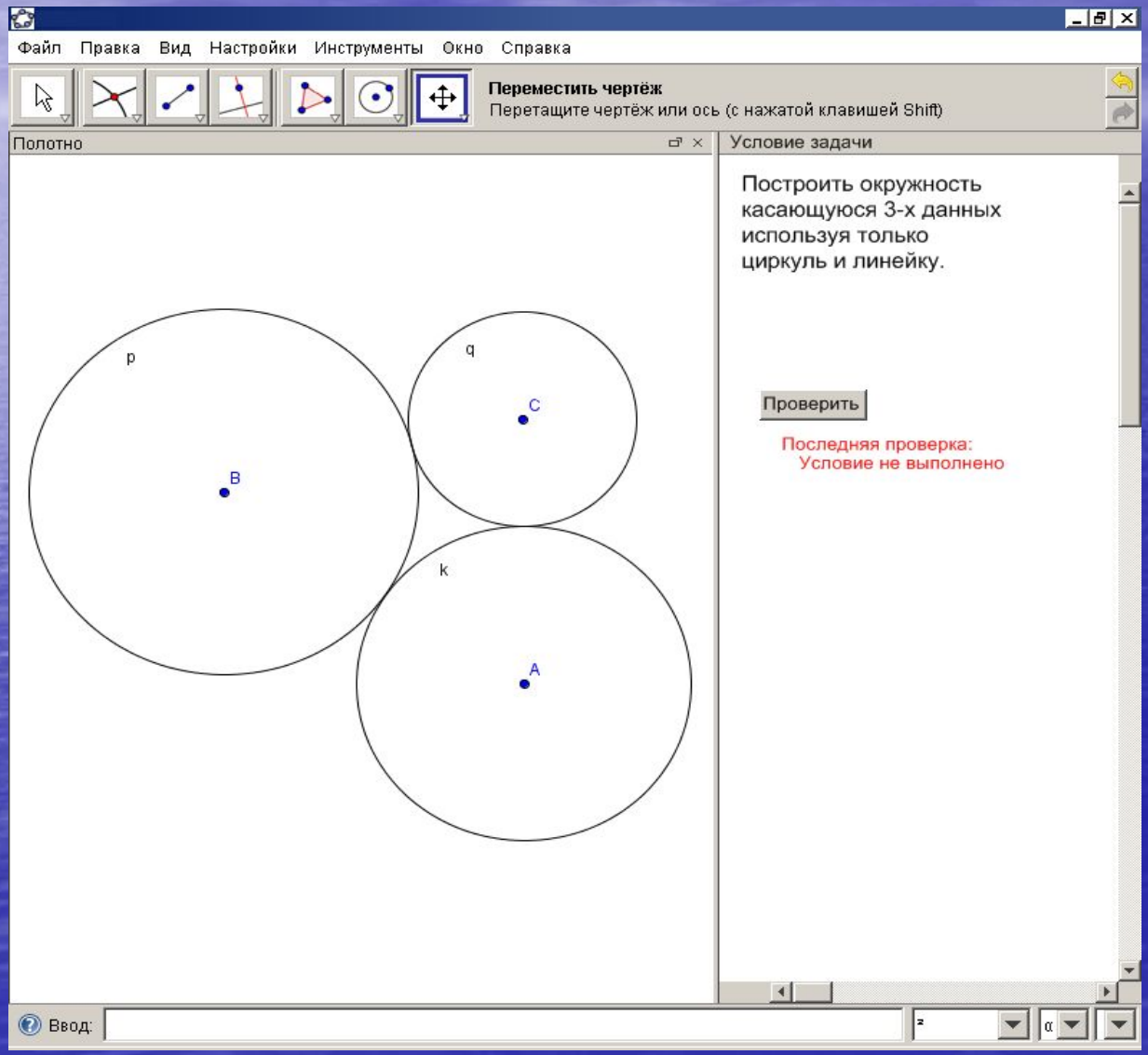
Программирование на основе ПОЯ



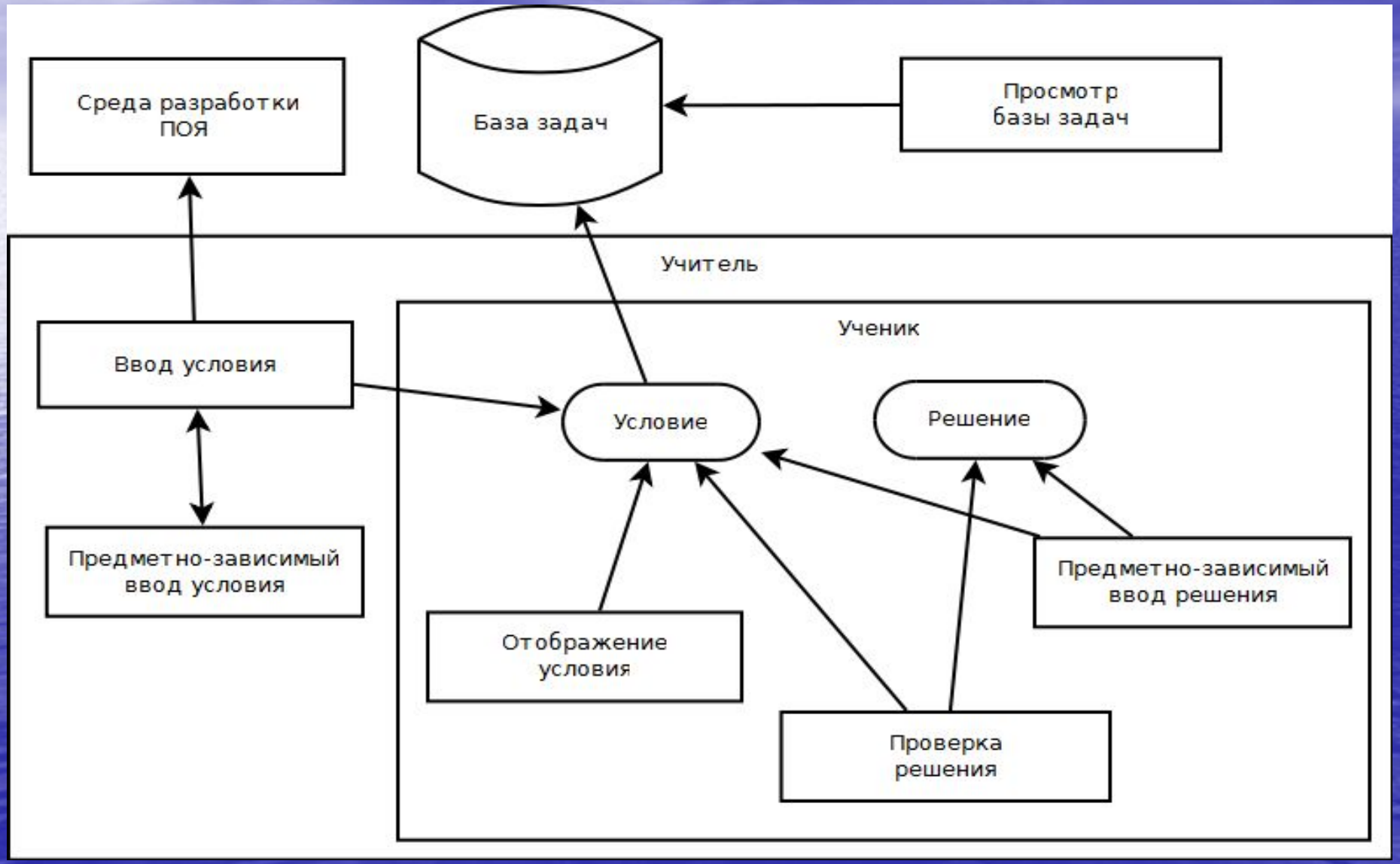
Интерфейс учителя



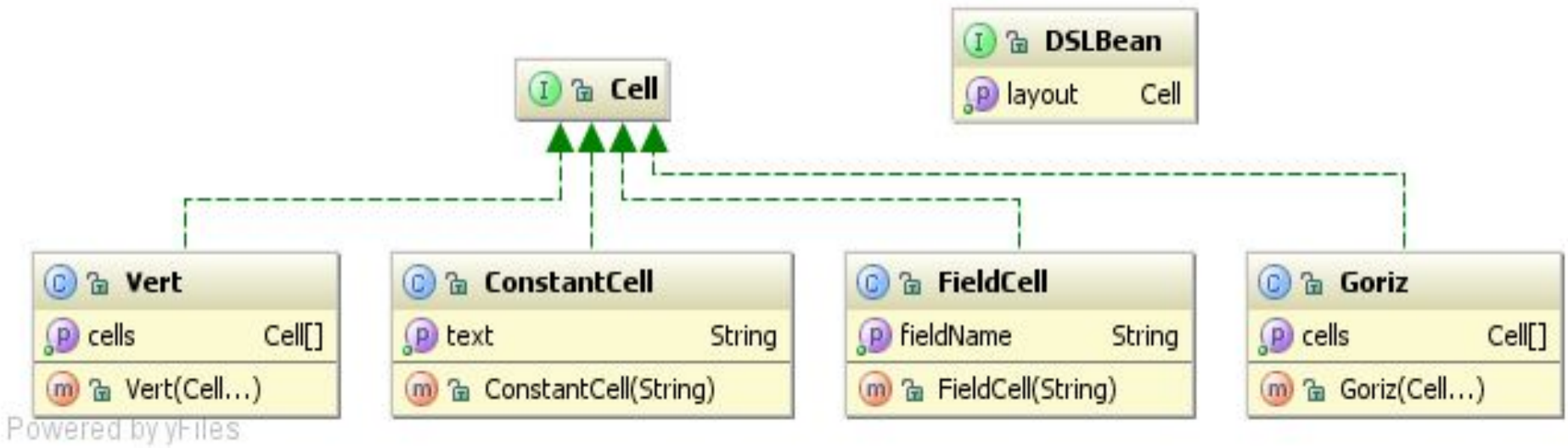
Интерфейс ученика

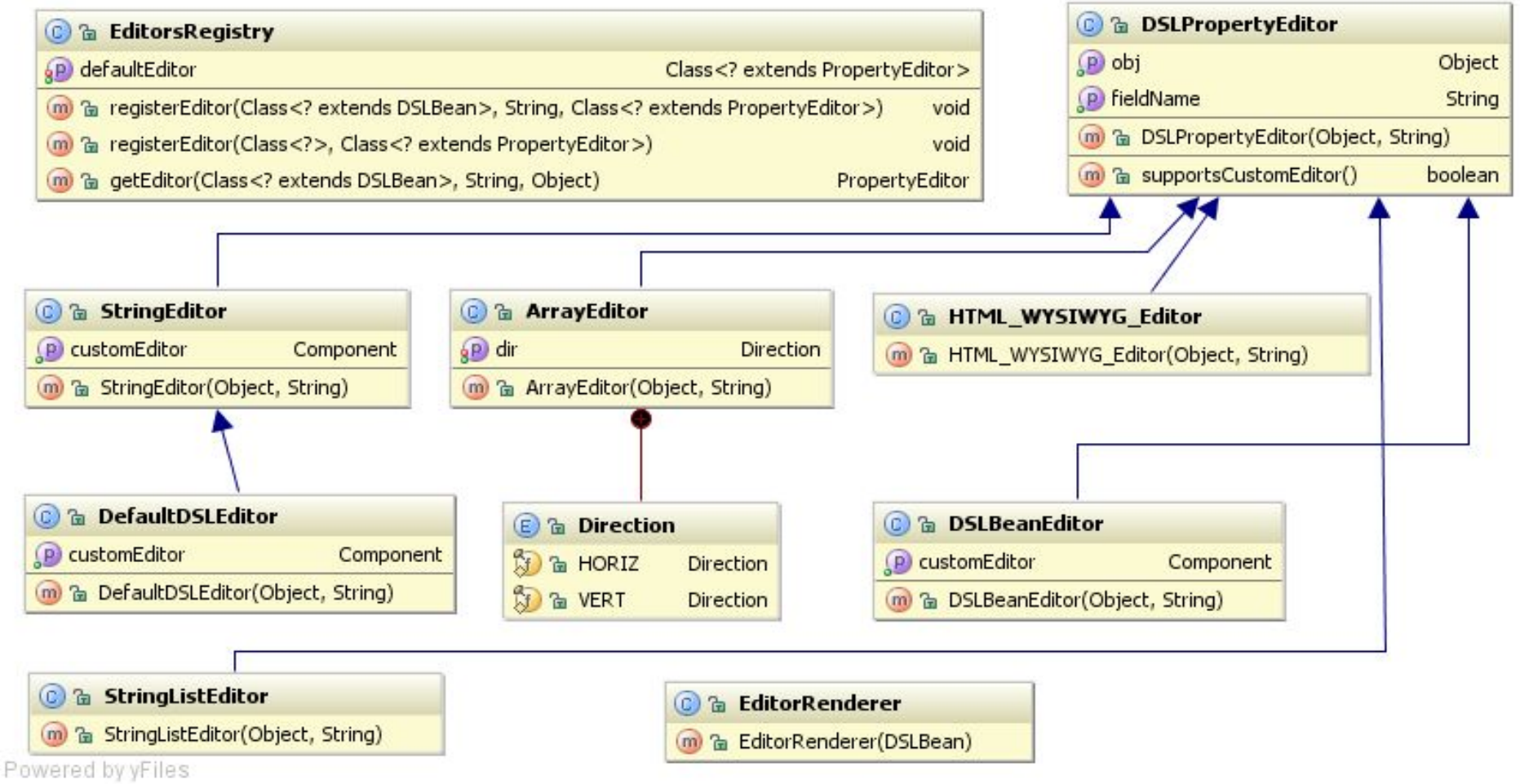


Разбиение системы на модули



Элементы редактора

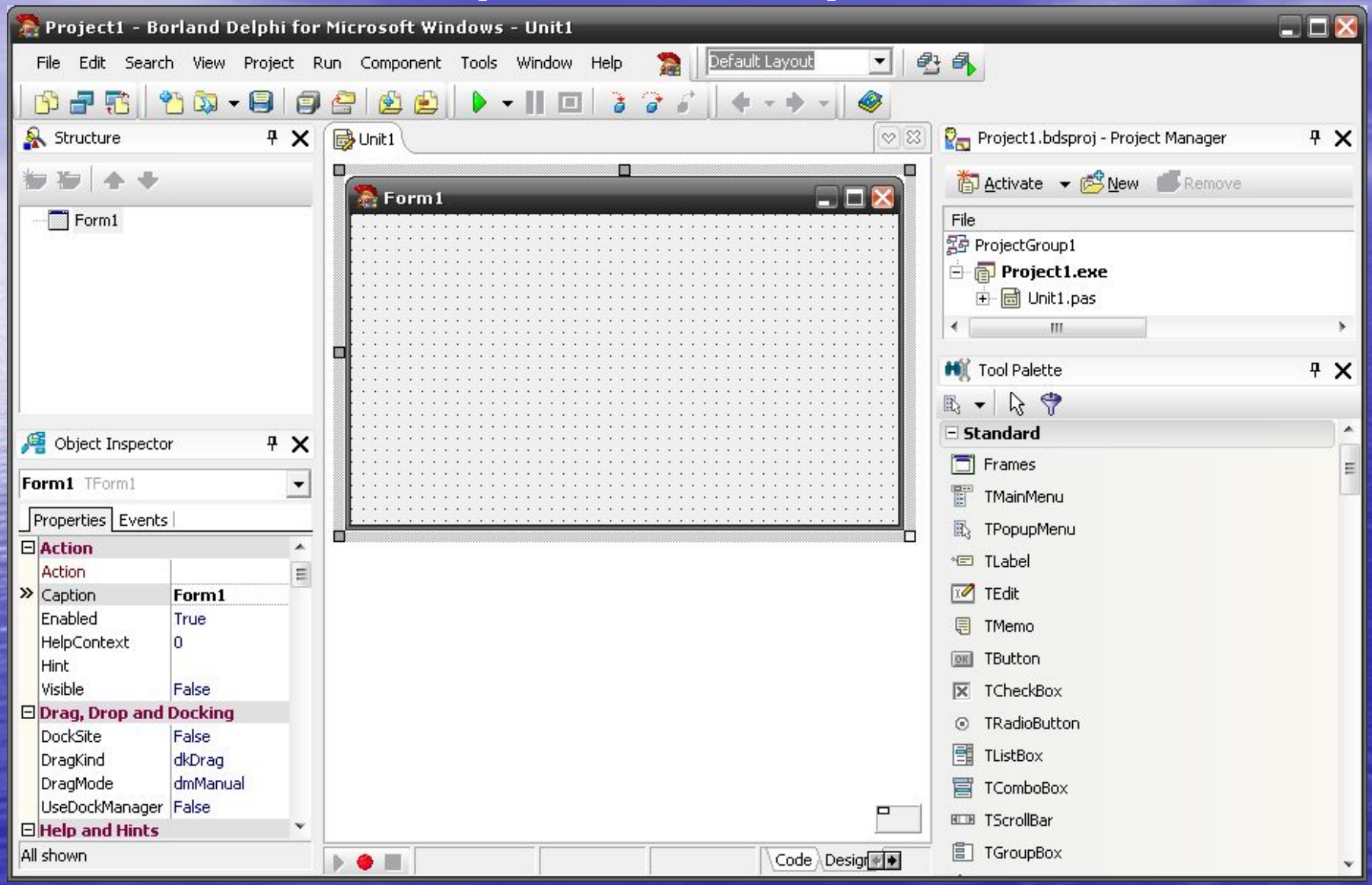



EditorsRegistry
defaultEditor Class<? extends PropertyEditor>
registerEditor(Class<? extends DSLBean>, String, Class<? extends PropertyEditor>) void
registerEditor(Class<?>, Class<? extends PropertyEditor>) void
getEditor(Class<? extends DSLBean>, String, Object) PropertyEditor
DSLPropertyEditor
obj Object
fieldName String
DSLPropertyEditor(Object, String)
supportsCustomEditor() boolean
StringEditor
customEditor Component
StringEditor(Object, String)
ArrayEditor
dir Direction
ArrayEditor(Object, String)
HTML_WYSIWYG_Editor
HTML_WYSIWYG_Editor(Object, String)
DefaultDSLEditor
customEditor Component
DefaultDSLEditor(Object, String)
Direction
HORIZ Direction
VERT Direction
DSLBeanEditor
customEditor Component
DSLBeanEditor(Object, String)
StringListEditor
StringListEditor(Object, String)
EditorRenderer
EditorRenderer(DSLBean)
Powered by yFiles

Delphi – предметная ориентированность

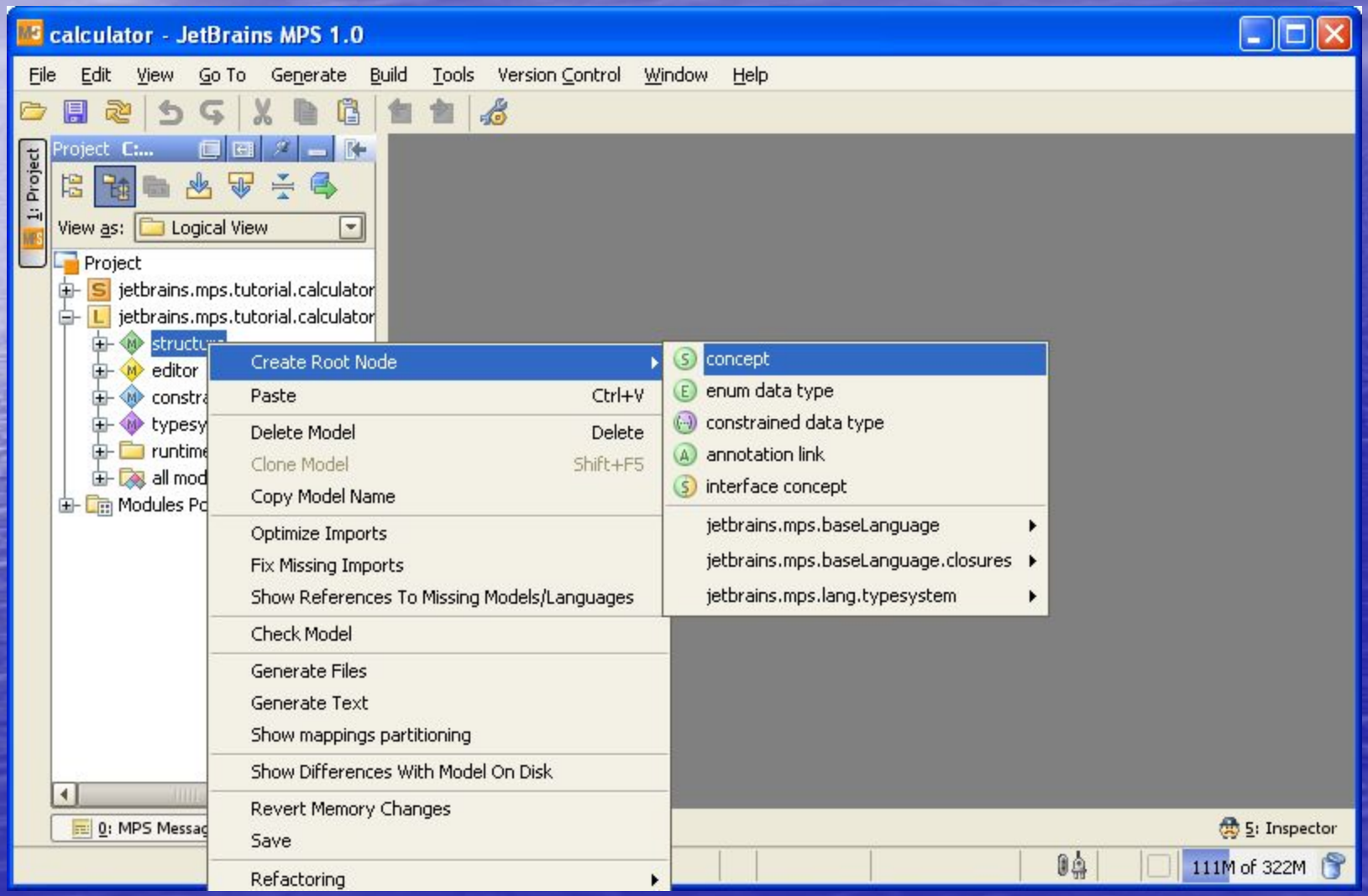
- Редактор интерфейса позволяет визуально (без программирования) нарисовать большую часть интерфейса. Процесс создания нагляден.
- Встроенные средства рефакторинга позволяют «переименовать» классы, методы, компоненты, модули в любой момент когда вы обнаружите несоответствие реального использования класса, метода, модуля и представления о нём.
- Среда разработки генерирует шаблон метода при выборе события в редакторе свойств, нужно писать только само тело обработчика.

Работа в среде Delphi

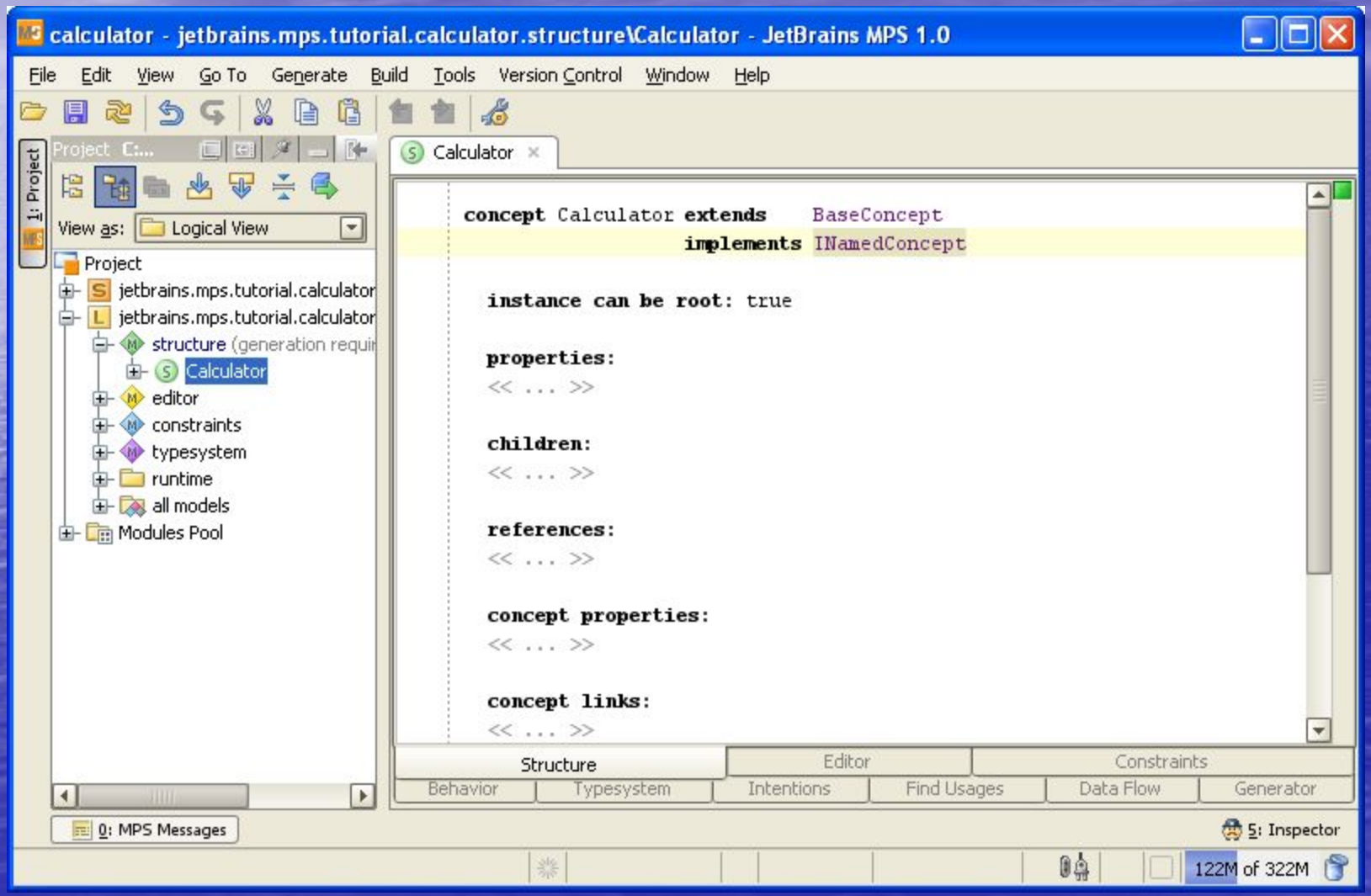


Meta Programming System

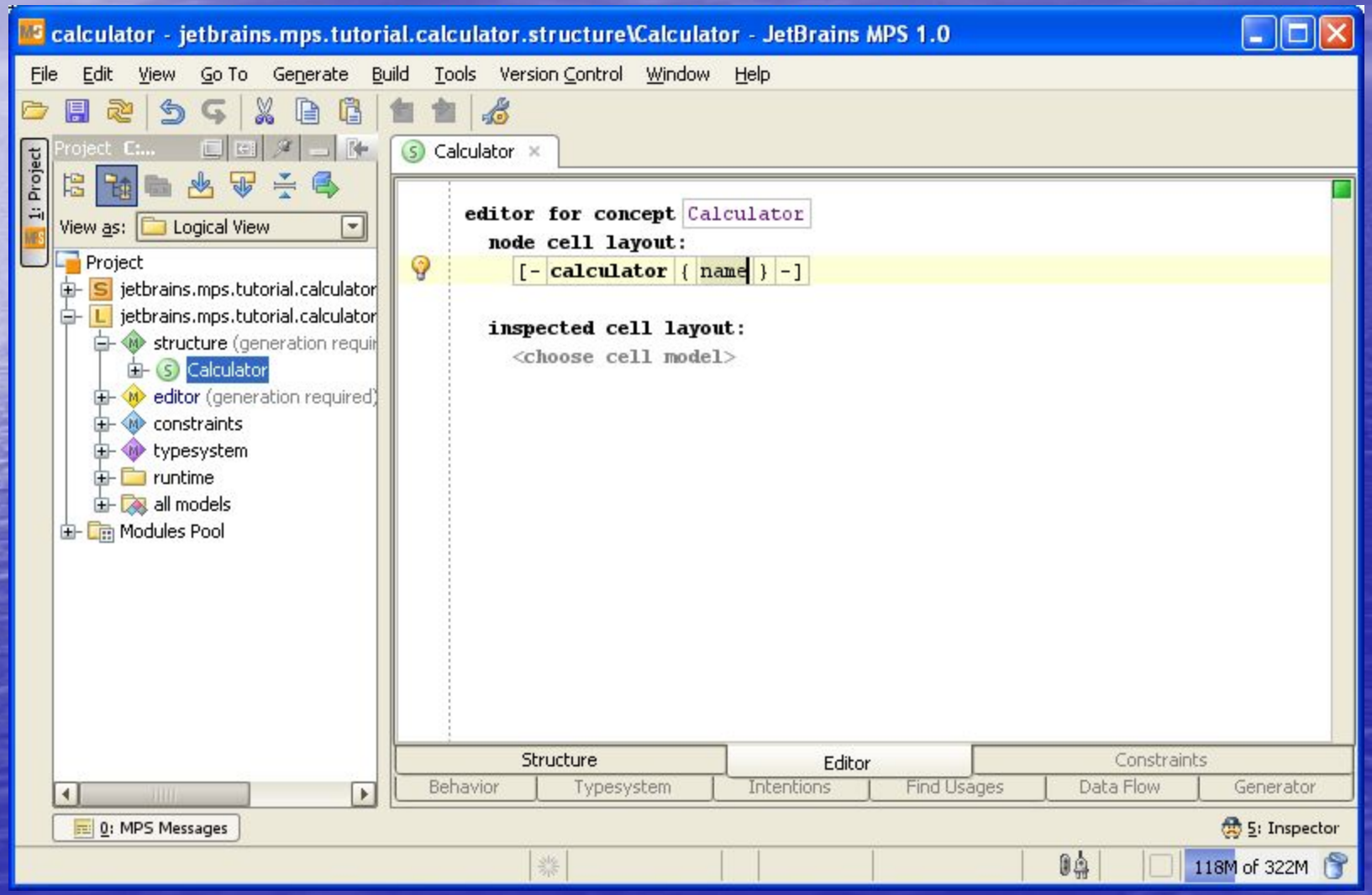
http://www.jetbrains.com/mps/



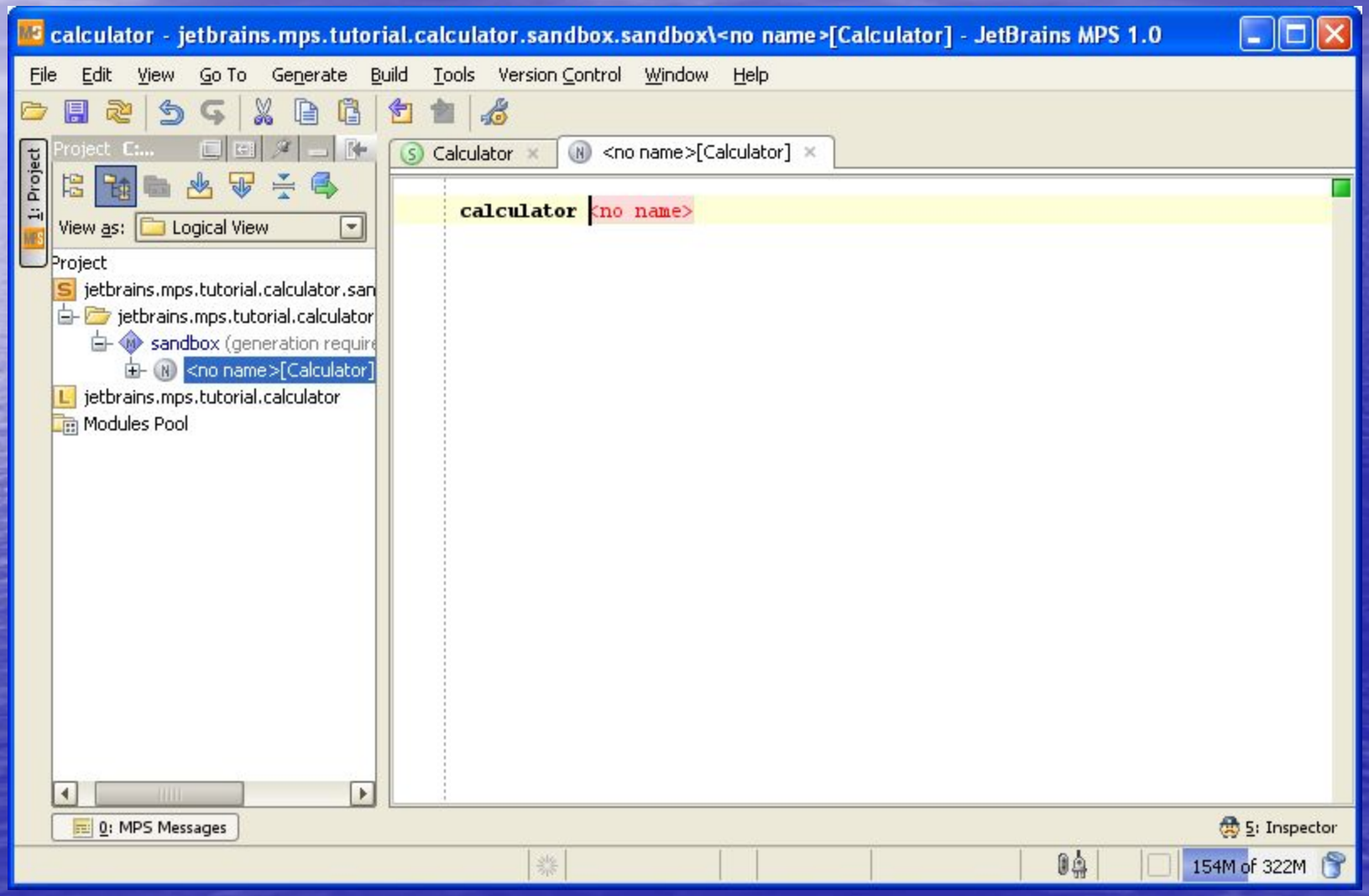
Создание языка



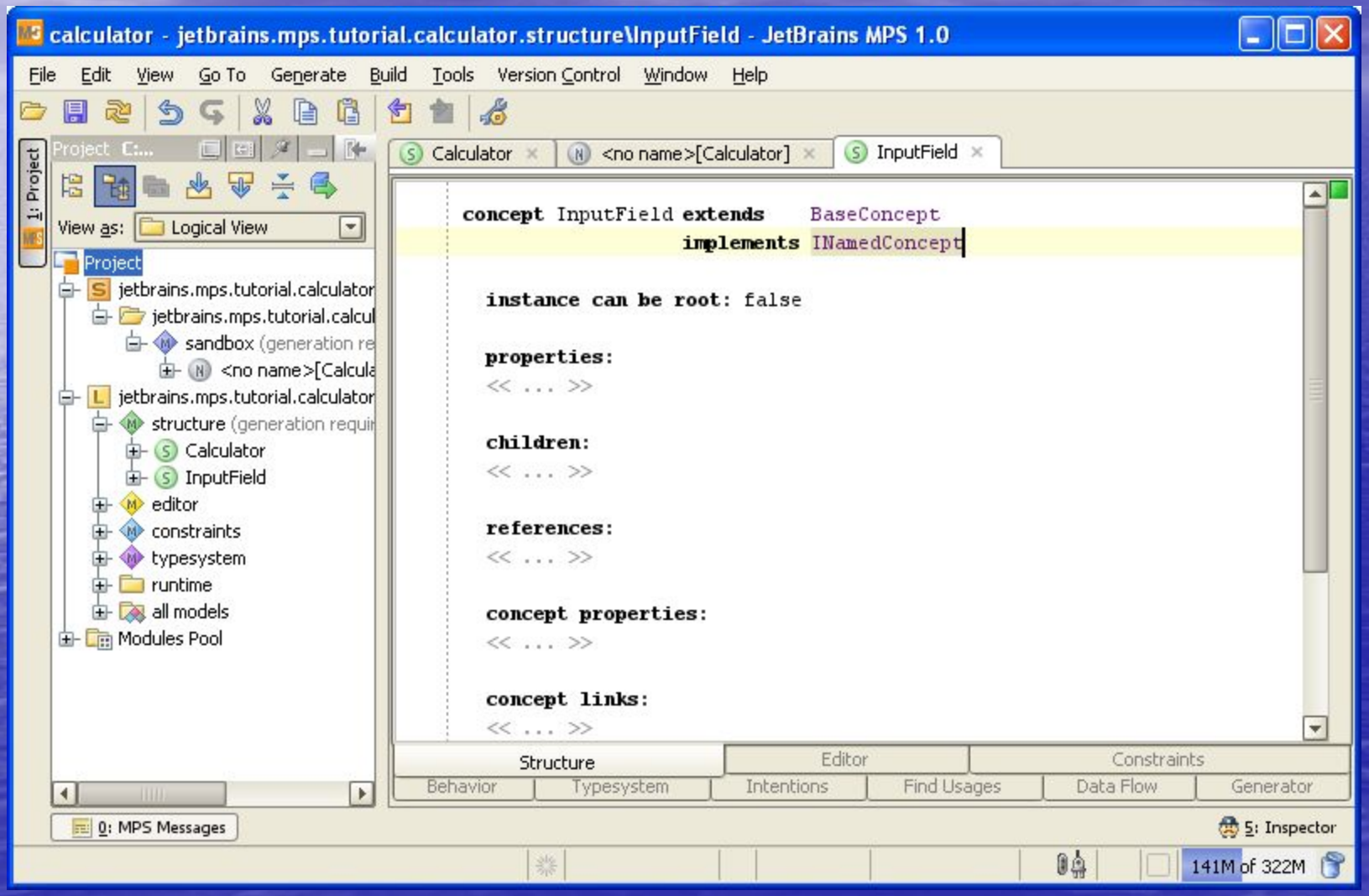
Создание редактора



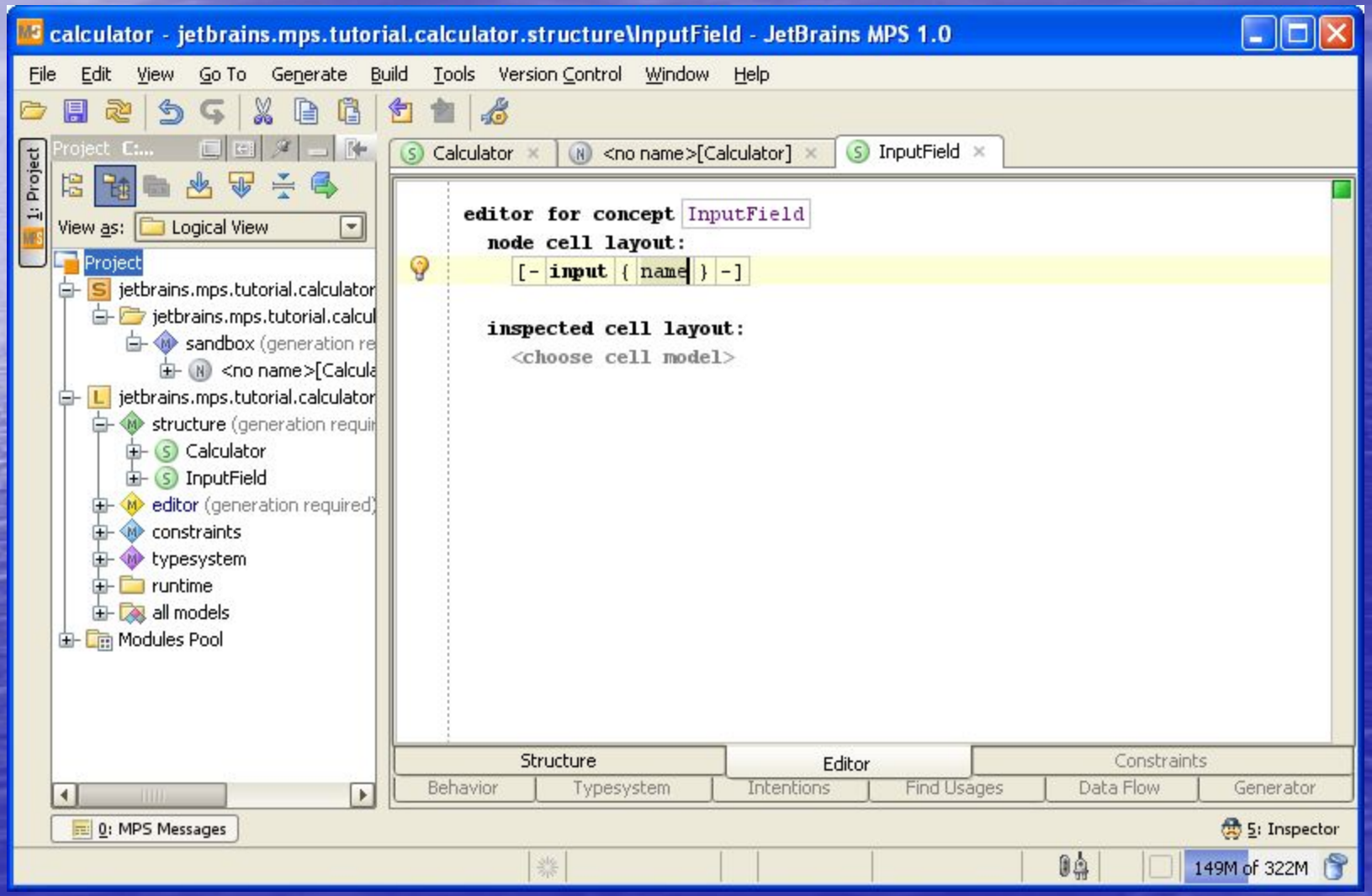
Использование - sandbox



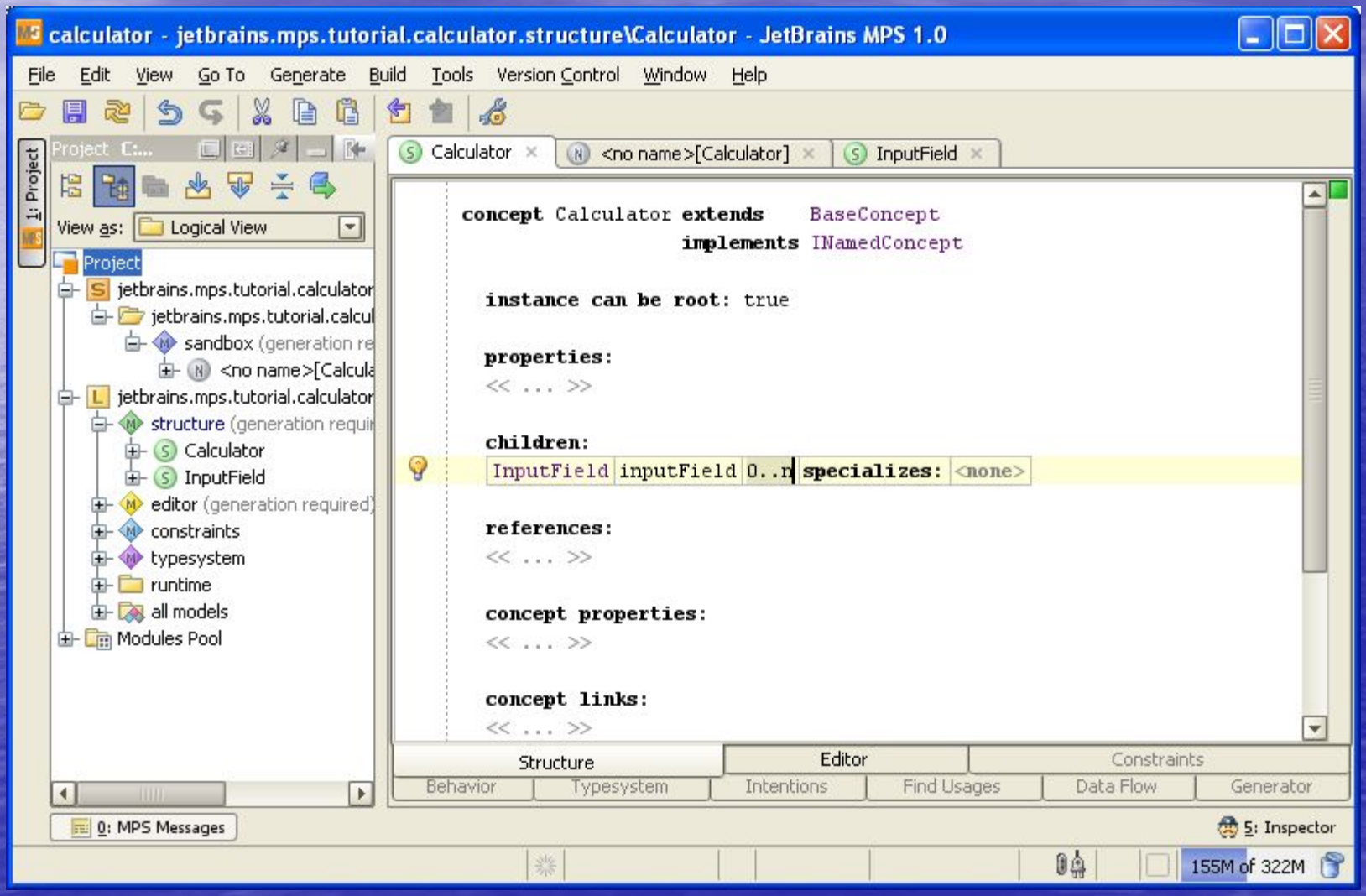
Создаём поле ввода данных



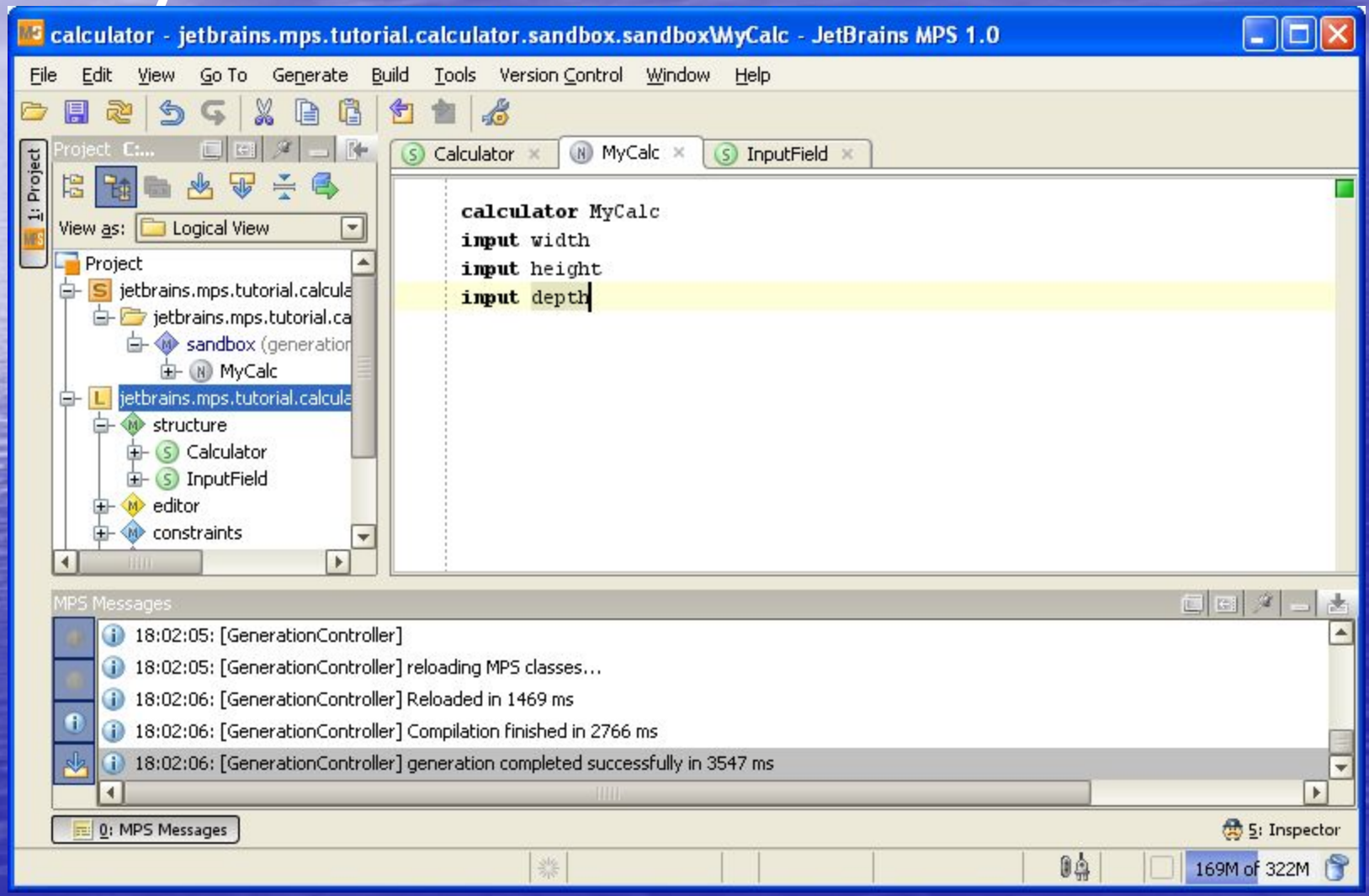
И редактор для него



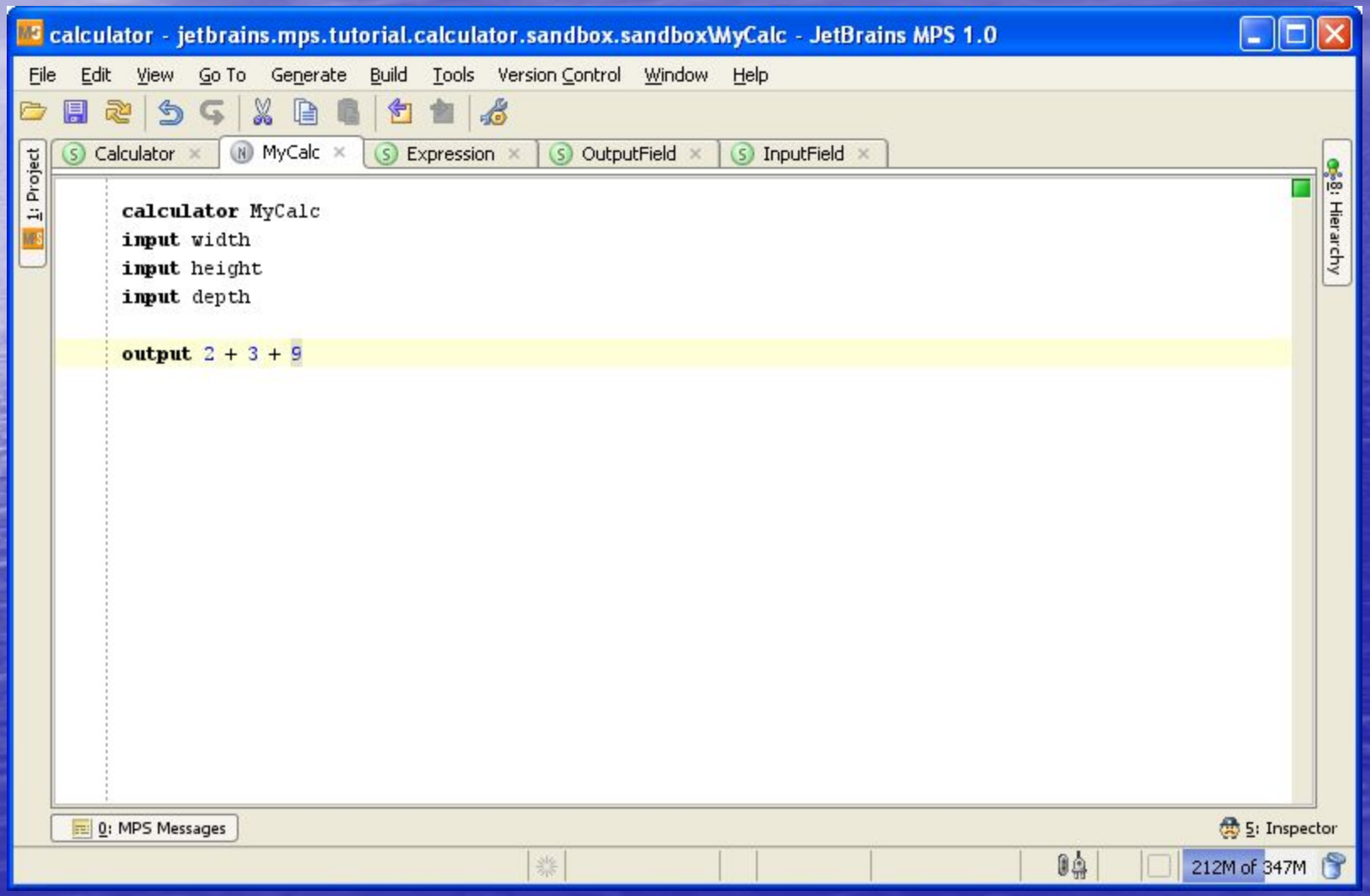
Добавляем поле в калькулятор



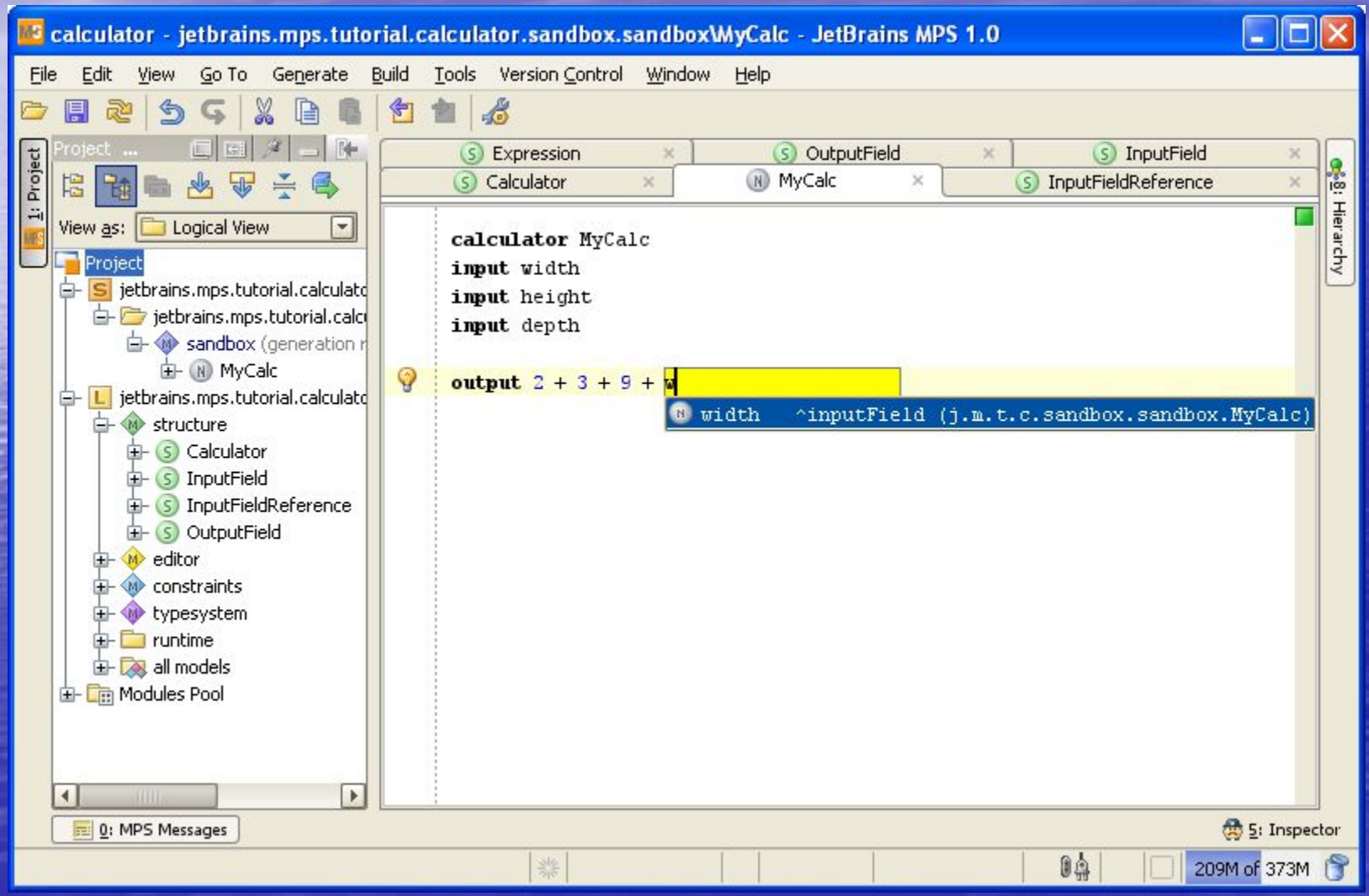
Генерируем и тестируем полученный язык



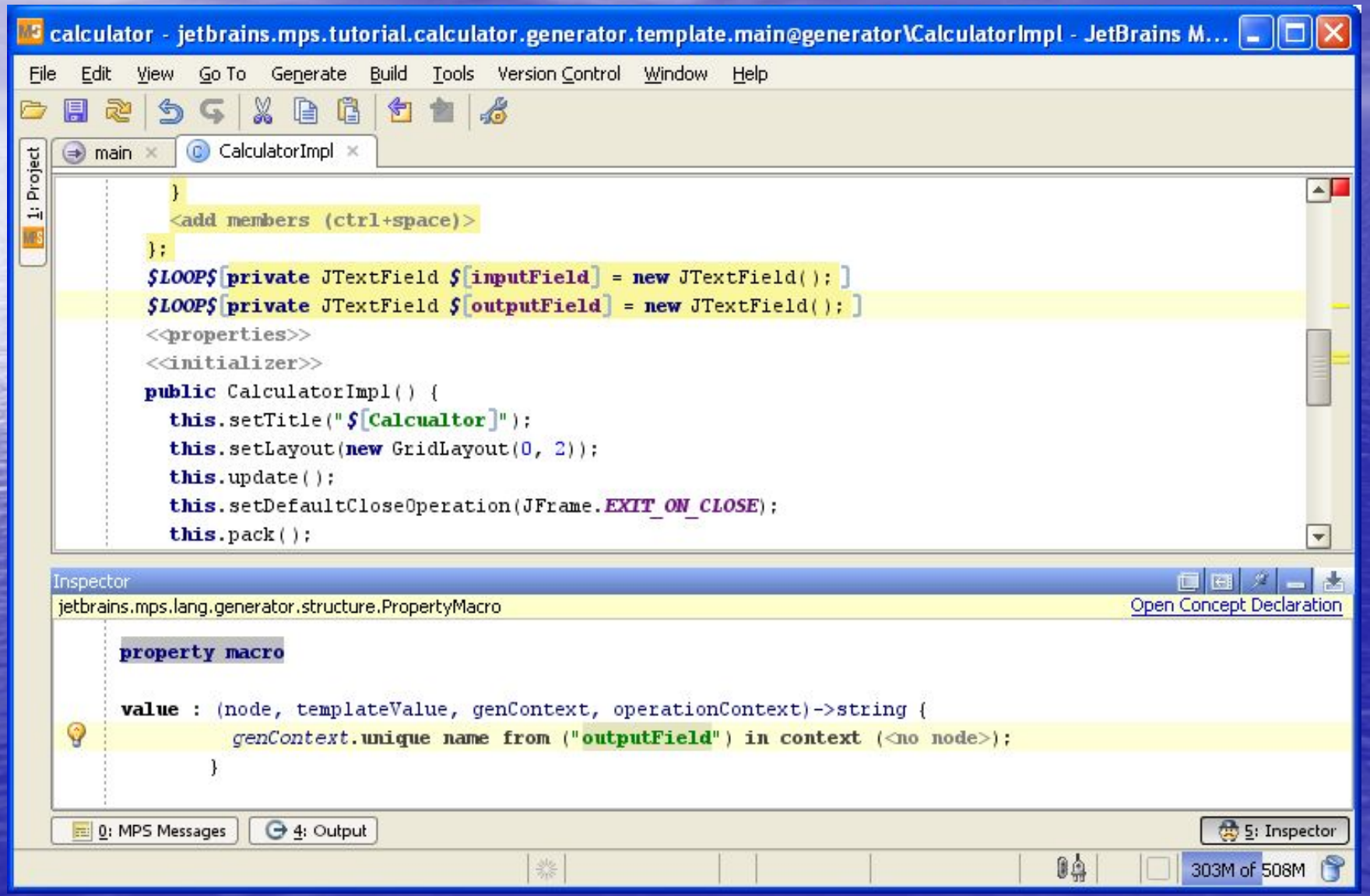
Выходное поле и формула



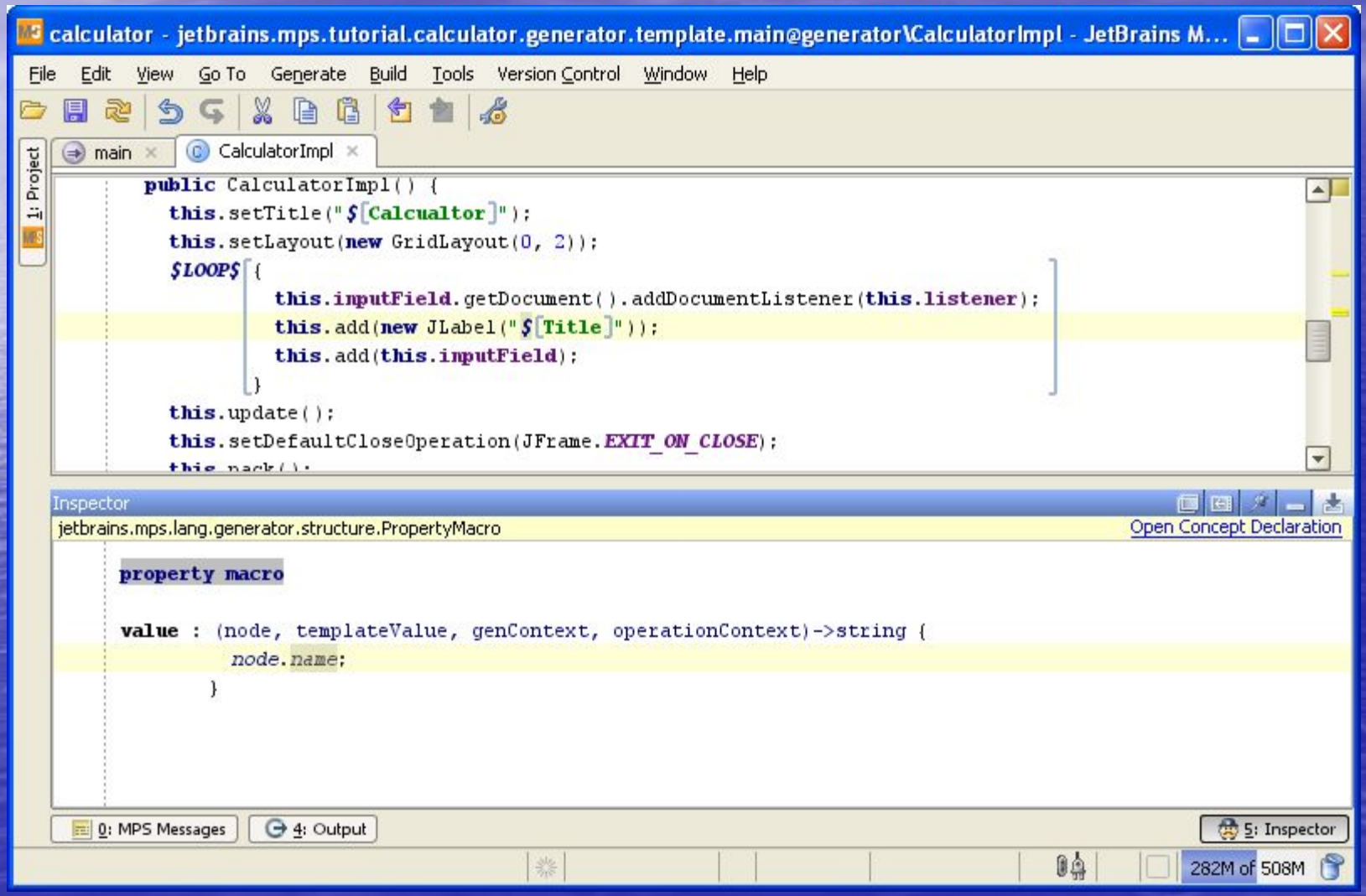
Можем использовать значение входных полей



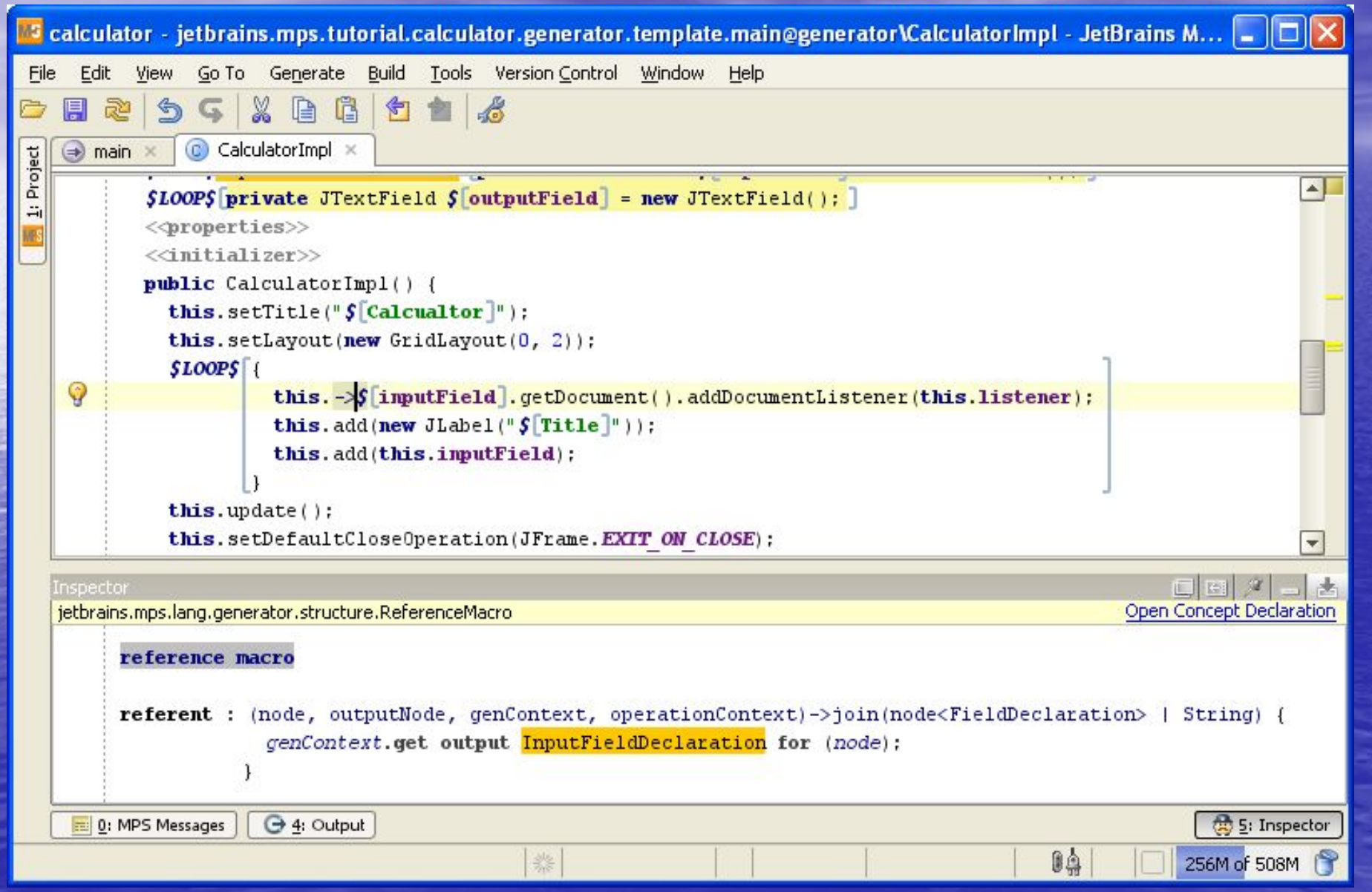
Создаем генератор кода



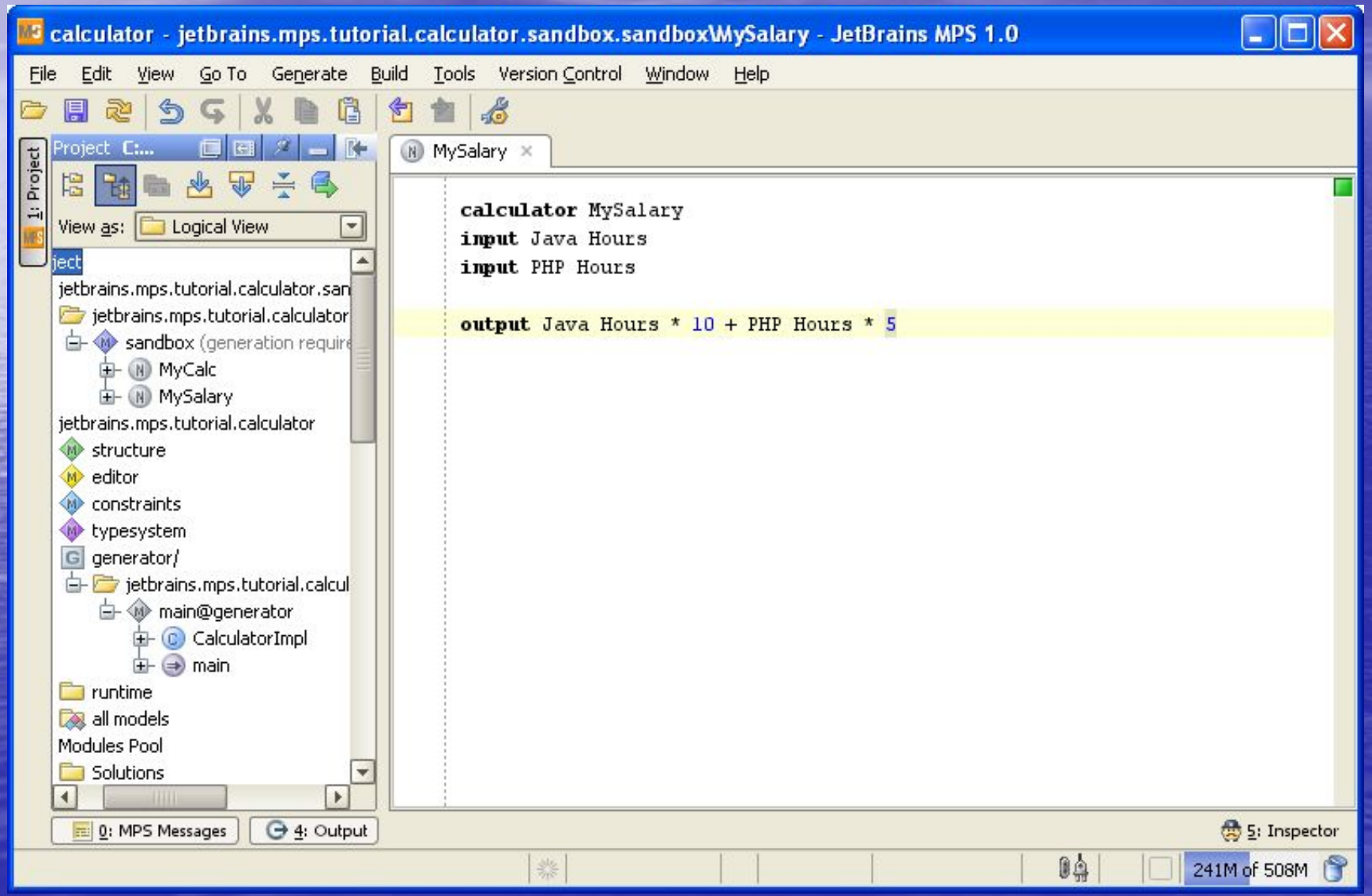
Создание полей «в цикле»



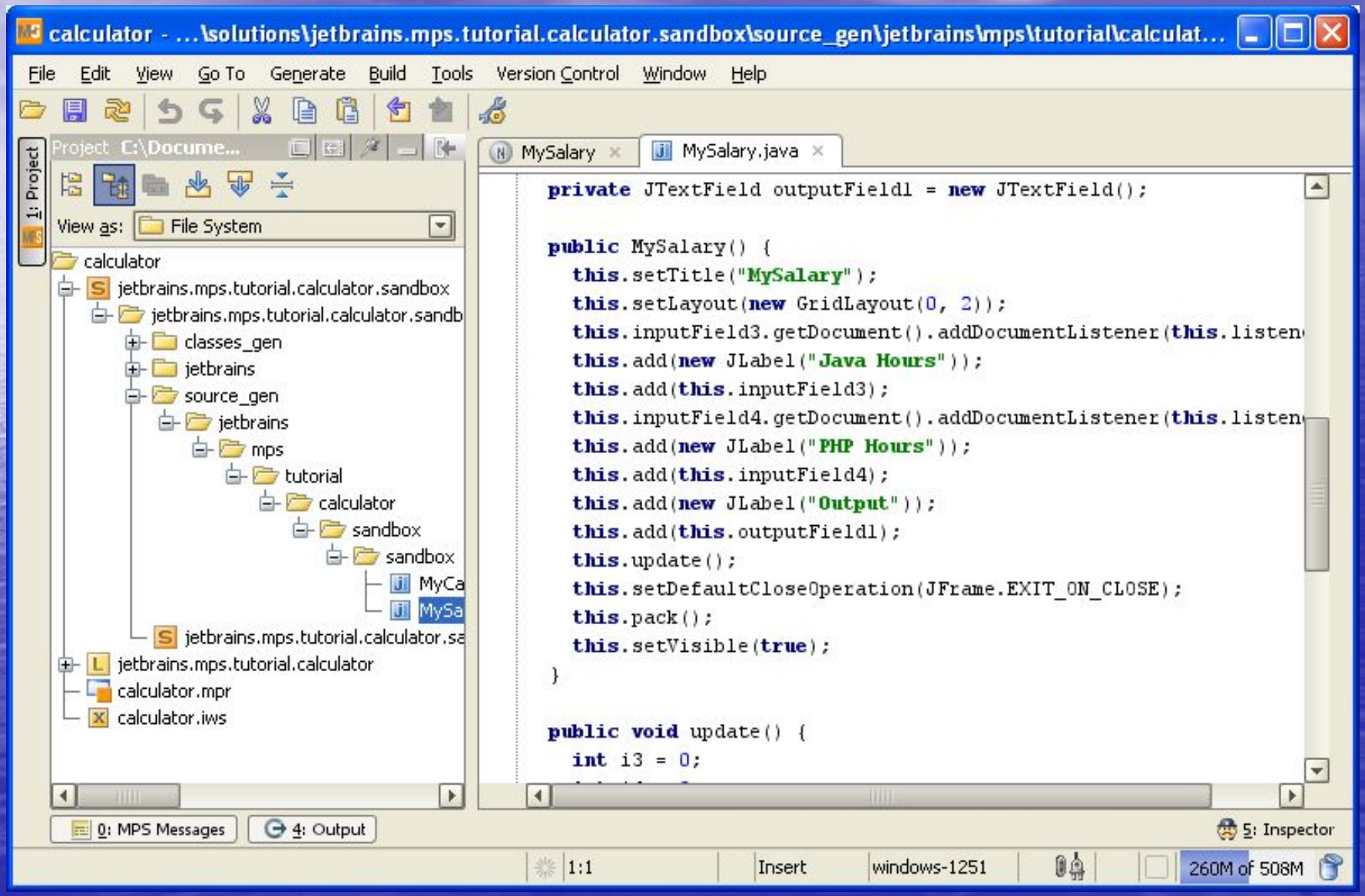
Ссылка на другой макрос



Пример использования



Готовая программа



Спасибо за внимание!

Вопросы?