

# 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

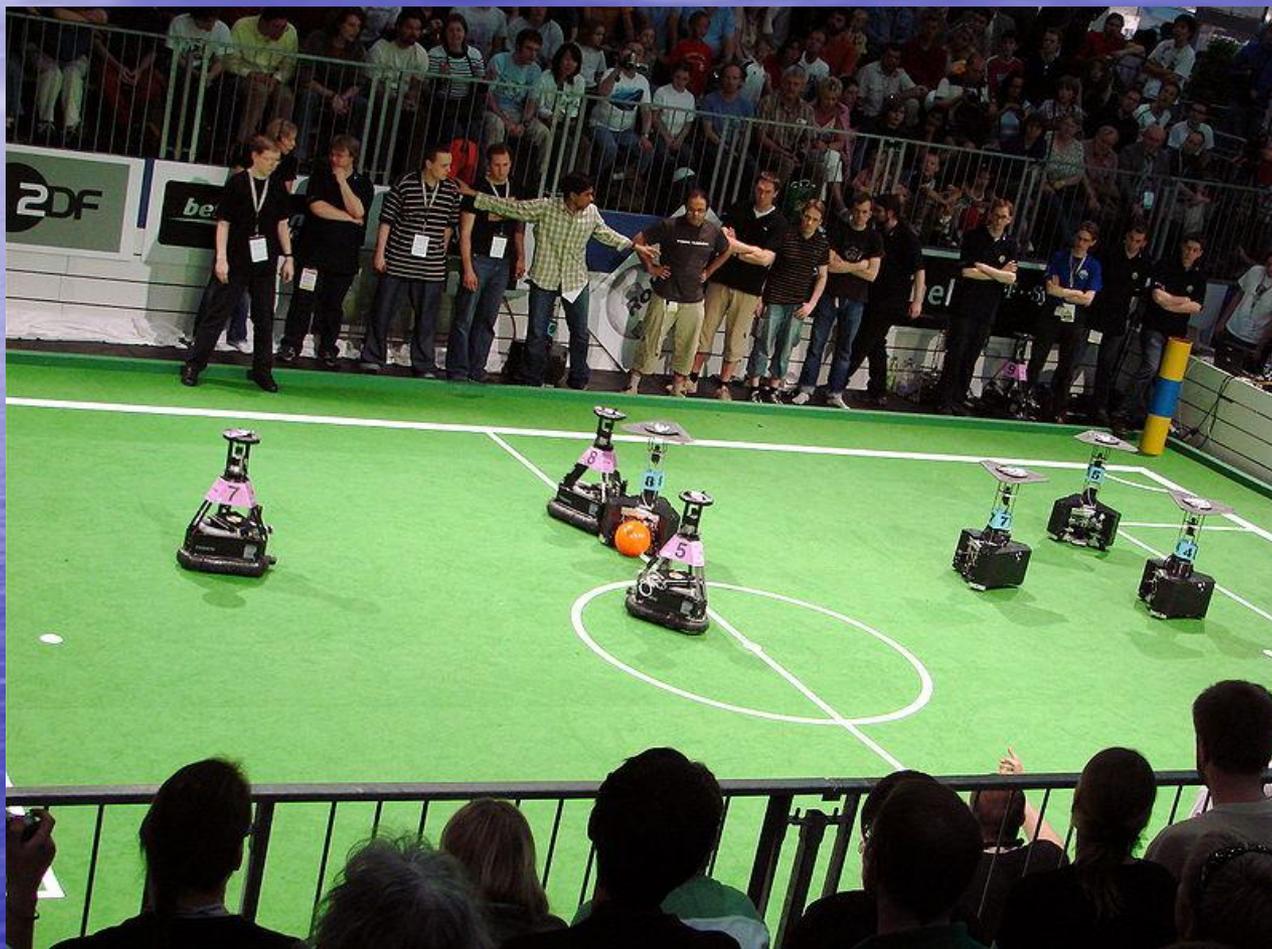


# Deep Blue – шахматный суперкомпьютер

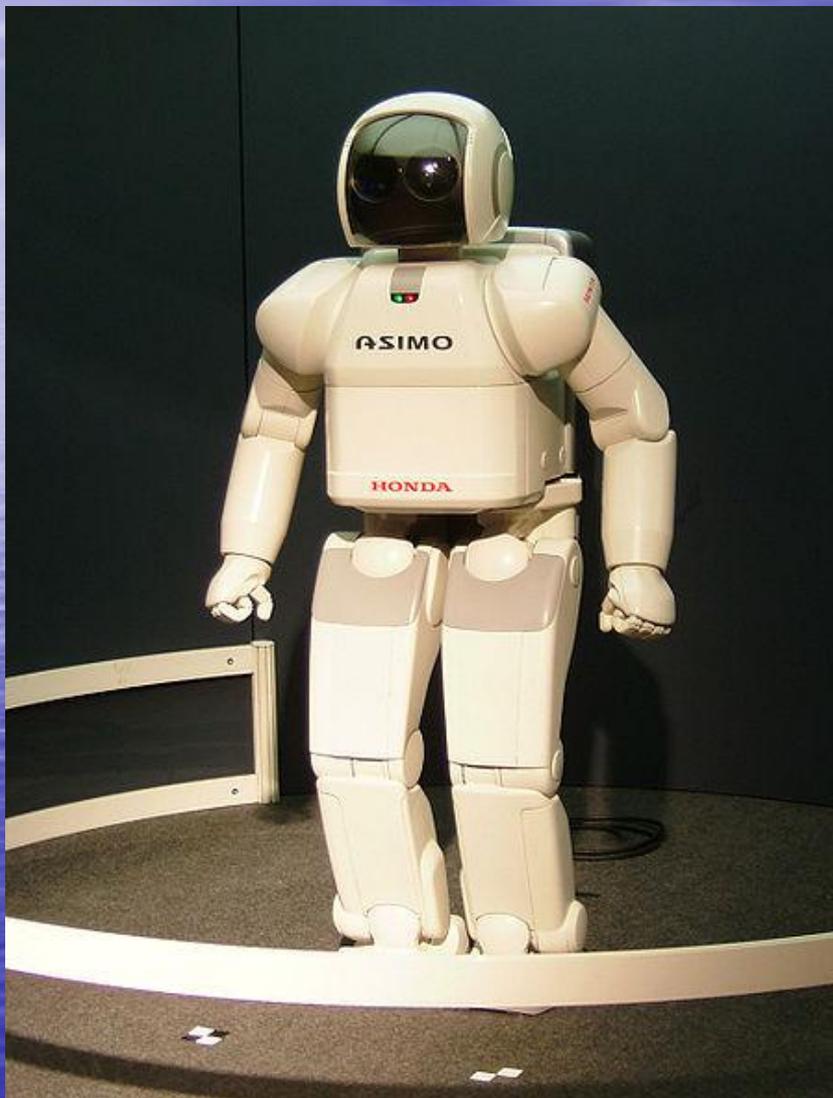


- 11 мая 1997 года одержал победу в матче из 6 партий с чемпионом мира по шахматам Гарри Каспаровым
- Разработан компанией IBM
- Название получил от «Deep Thought» (глубокая мысль) из романа Дугласа Адамса «Автостопом по галактике» и «клички» IBM: «Big Blue»
- После матча с чемпионом Deep Blue был разобран.
- 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

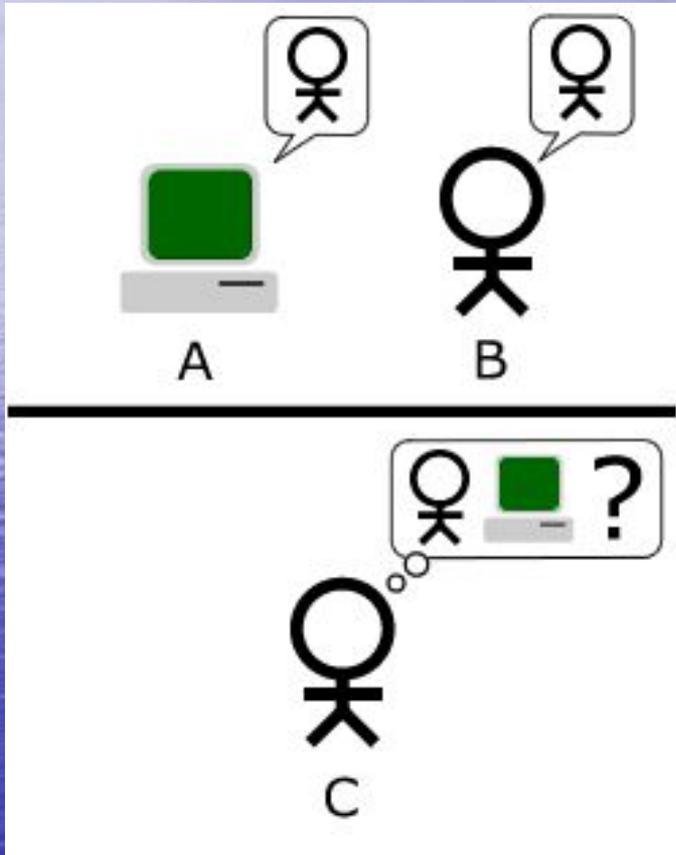
# Применение ИИ - турнир RoboCup



# Гуманоидный робот



# Тест Тьюринга



# Основные подходы 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. С)

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.

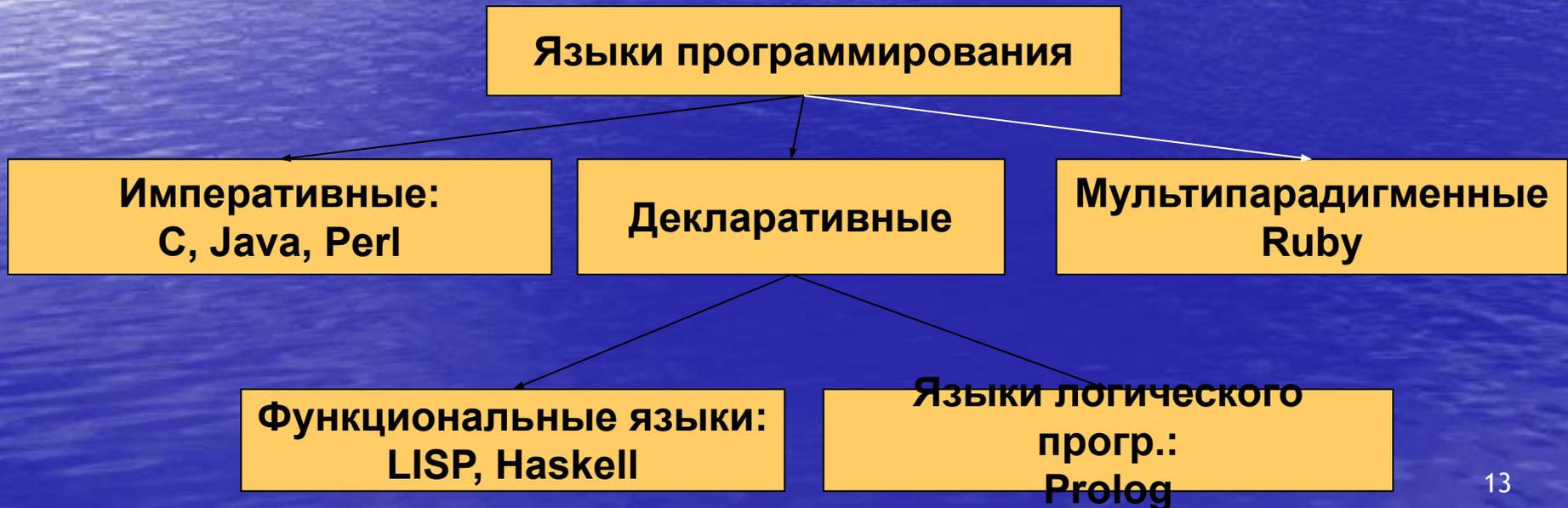
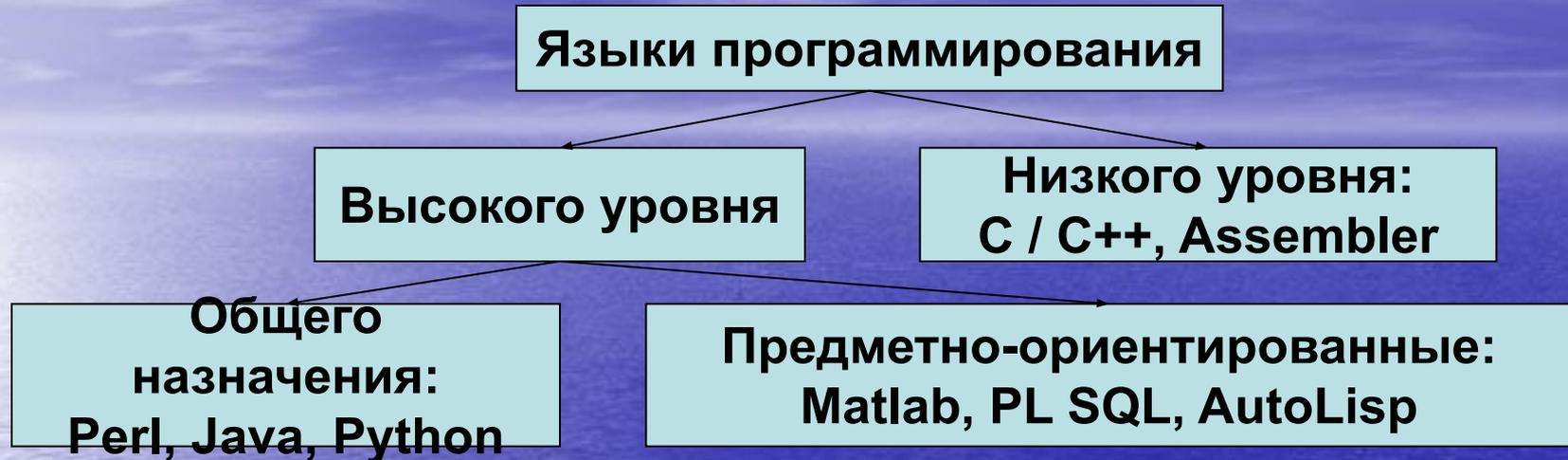
Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

# Парадигмы программирования

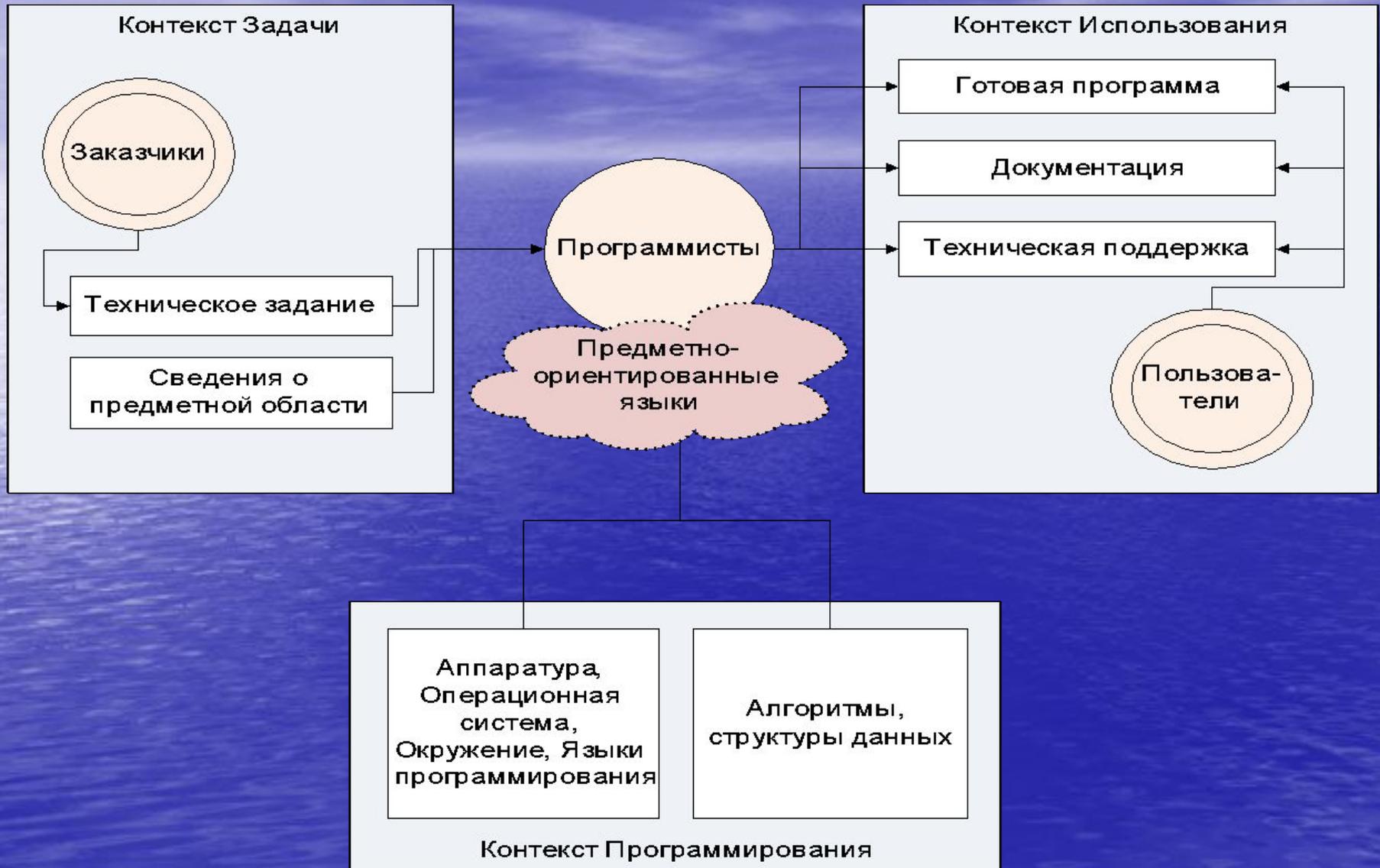
- \* Архитектура машины, ассемблер
- \* Формула-ориентированные
- \* Процедурное, структурное программирование
- \* Объектно-ориентированное программирование
- \* Языково-ориентированное программирование

**Сейчас**

# Классификации языков



# Программист как преобразователь контекста



# Предметно-ориентированный подход

Максимально приближена  
к описанию на ест. языке

Программа на  
предметно-  
ориентированном  
языке

Описание задачи  
на естественном  
языке

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

Реализация  
(трансляция)

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

Готовая  
исполняемая  
программа

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 С 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.

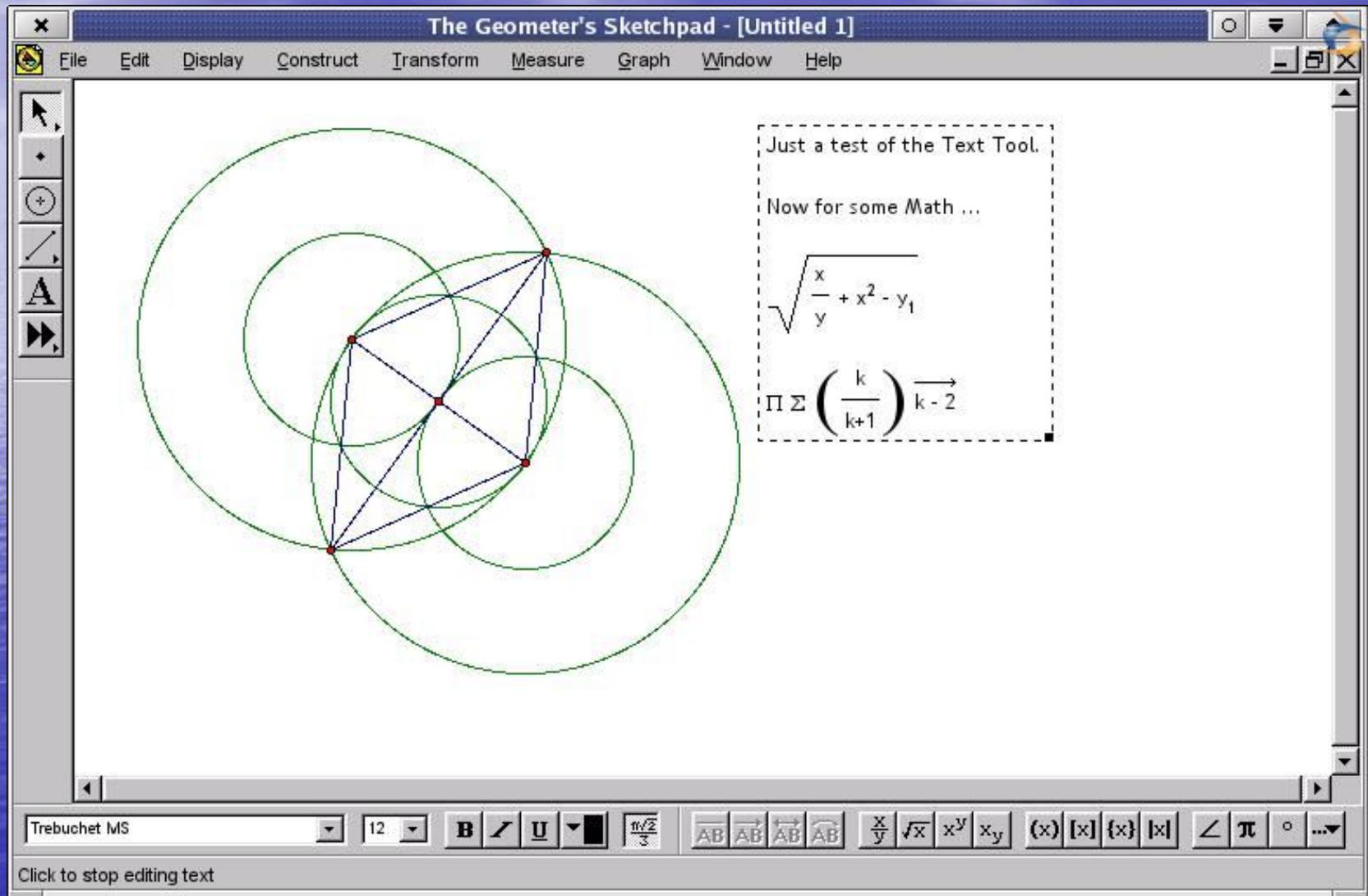
Тут возможно несколько подходов.

- \* 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

ГЛАВНАЯ

ЖУРНАЛ В ЖУРНАЛЕ

E-MAIL

## 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

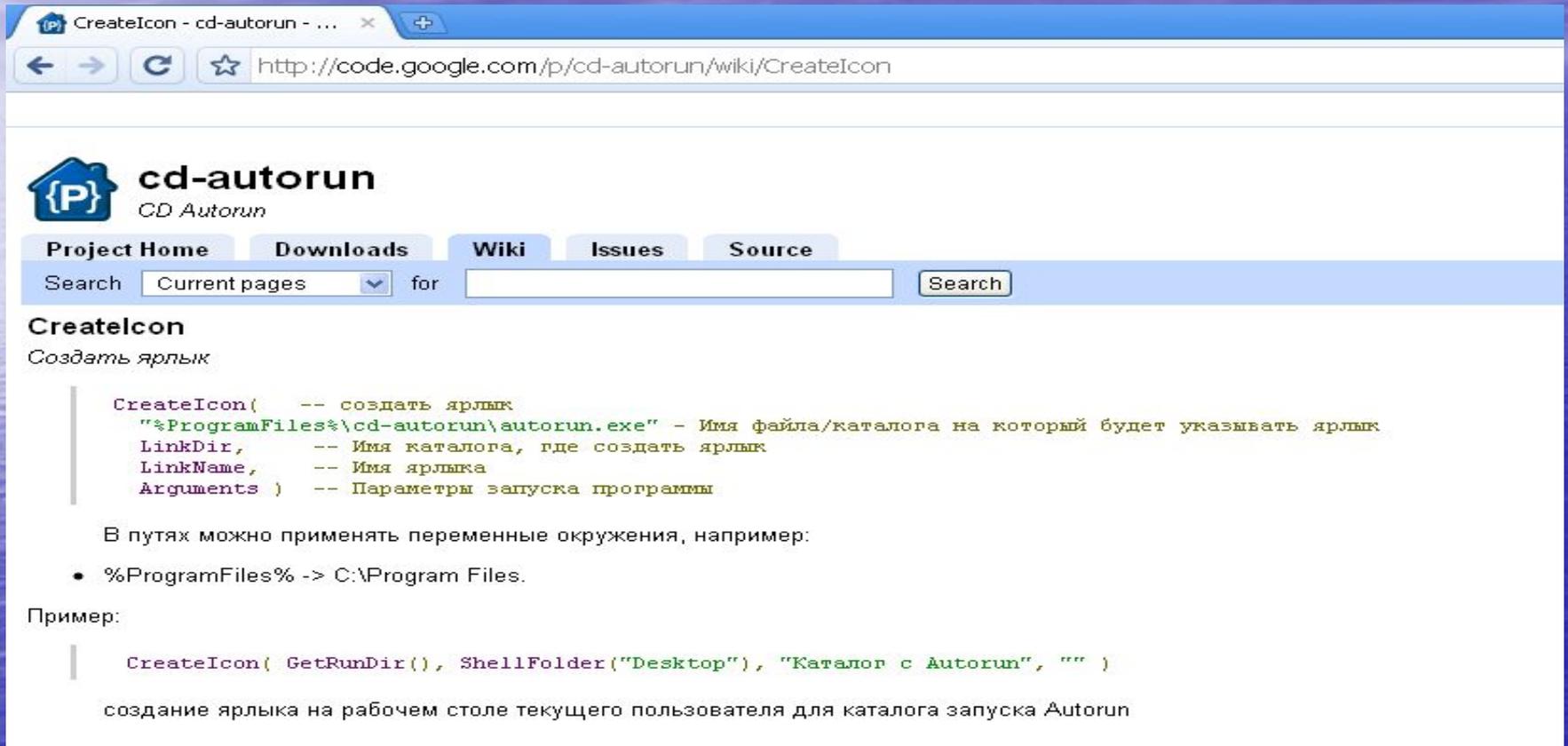
Кубистский паркет

## ИНСТРУМЕНТ

Система аспектно-ориентированного программирования

2

# Пример: Создание иконки



The screenshot shows a web browser window with the address bar containing the URL `http://code.google.com/p/cd-autorun/wiki/CreateIcon`. The page header features the project logo and name "cd-autorun" with the tagline "CD Autorun". Navigation tabs include "Project Home", "Downloads", "Wiki", "Issues", and "Source". A search bar is present with the text "Current pages" and a "Search" button.

## Createlcon

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:

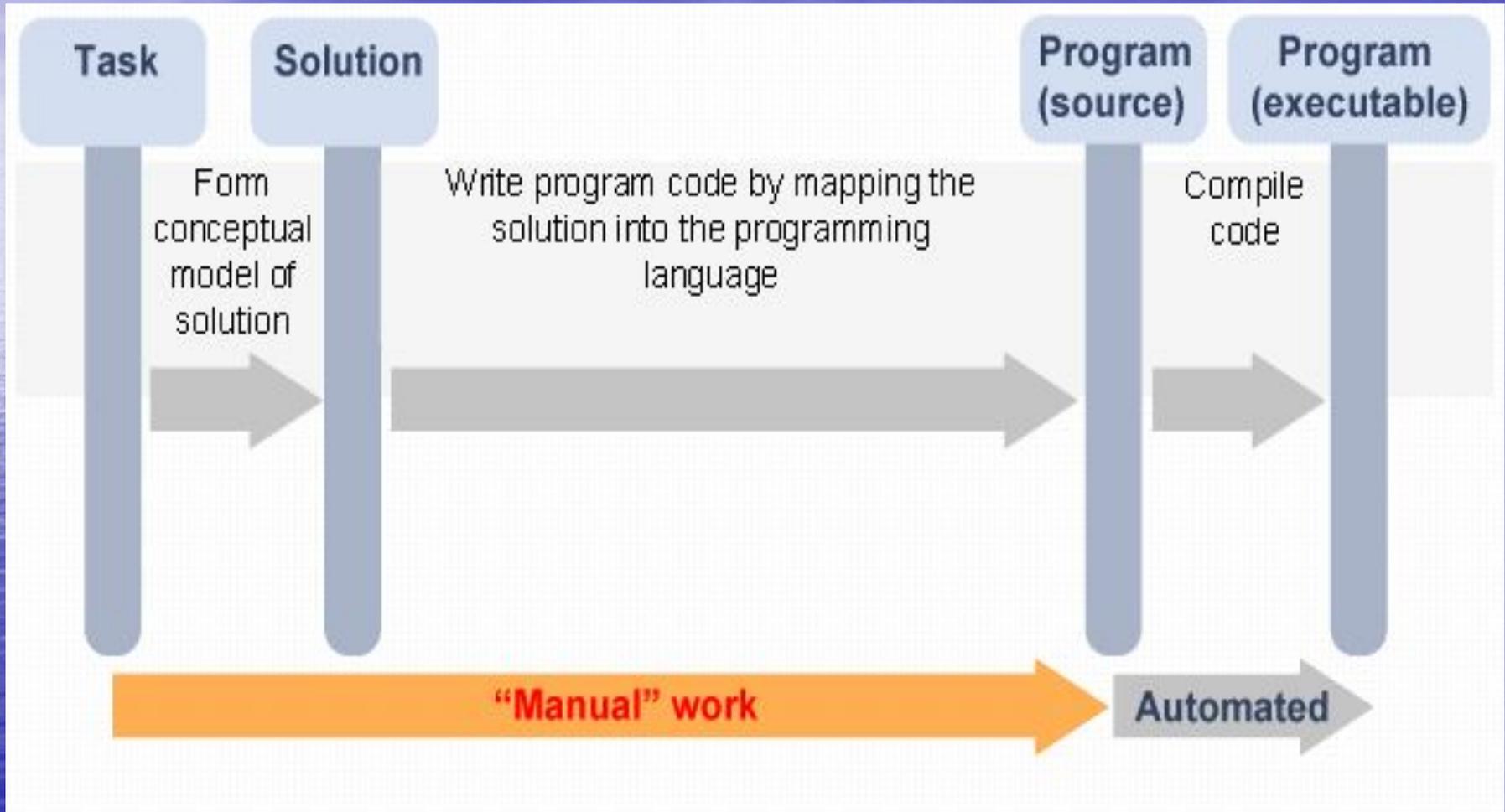
```
CreateIcon( GetRunDir(), ShellFolder("Desktop"), "Каталог с Autorun", "" )
```

создание ярлыка на рабочем столе текущего пользователя для каталога запуска Autorun

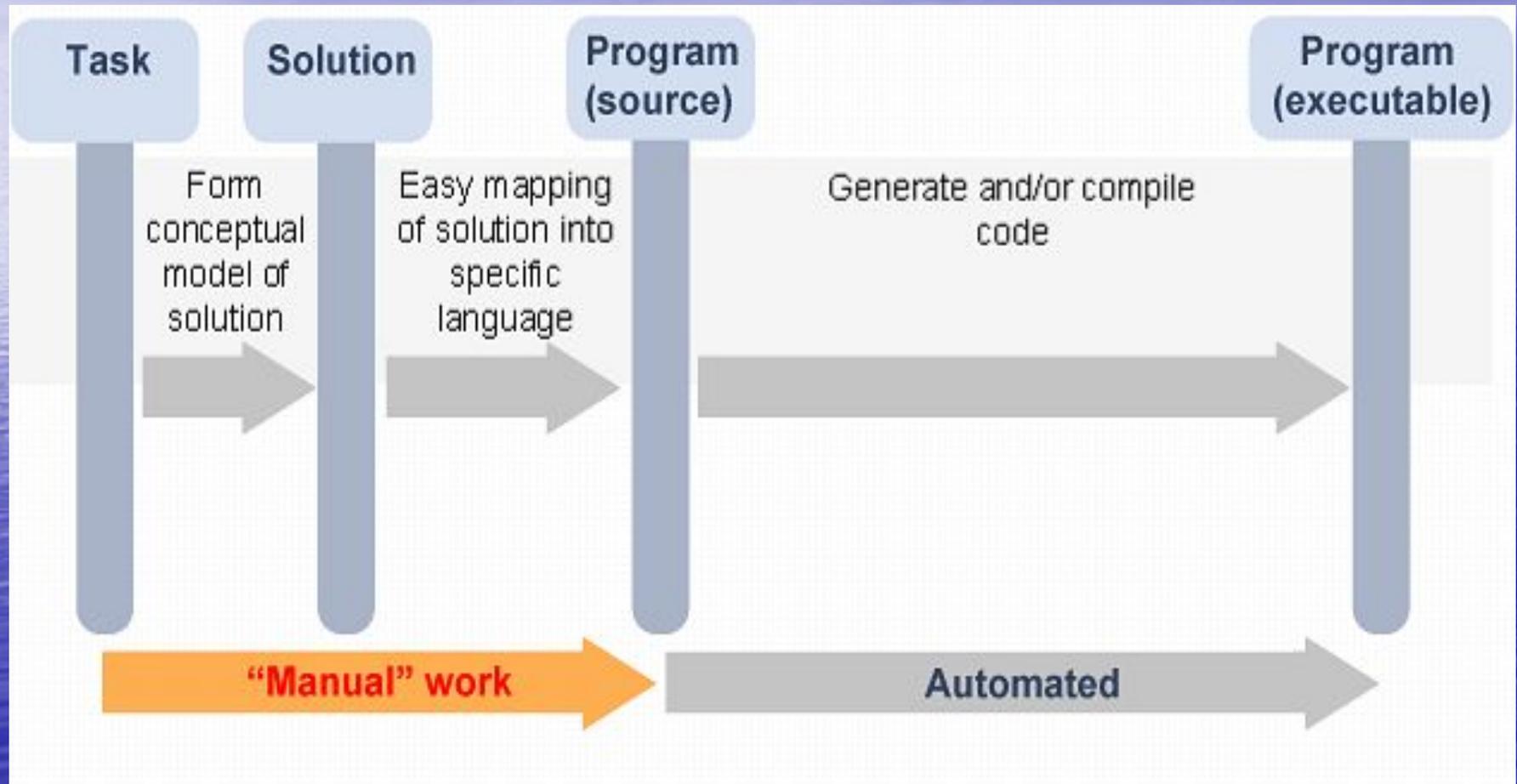
# Схема работы «Журнала в журнале»



# Программирование на языке общего назначения



# Программирование на основе ПОЯ



# Интерфейс учителя

Файл Правка Вид Настройки Инструменты Окно Справка

Переместить чертёж  
Перетащите чертёж или ось (с нажатой клавишей Shift)

Дерево задач

- Математика
  - Алгебра и геометрия
    - Абстрактная алгебра
    - Аналитическая геометрия
    - Векторная алгебра
    - Квадратичные формы, кривые и пов-ти 2-го поряд
    - Линейная алгебра
    - Линейные пространства и операторы
    - Матричная алгебра
    - Понятие функции
    - Собственные числа и вектора
  - Дискретная математика
    - Делимость целых чисел и многочленов
    - Комбинаторика
    - Мат. логика и теория алгоритмов
  - Дифференциальные уравнения
  - Комплексные числа
  - Математический анализ
    - Интегралы функций одной переменной
    - Кратные, криволинейные и поверхностные интегр
    - Пределы последовательностей и функций
    - Производная и ее применение**
  - Ряды
    - Ряды Фурье
    - Числовые и степенные ряды
    - Теория функций комплексной переменной
    - Формула и ряд Тэйлора

Панель объектов

Свободные объекты

- A = (6.64, 2.28)
- B = (2.46, 5.12)
- C = (6.62, 6.2)

Зависимые объекты

- D = (5.4, 4.61)
- F = (5.09, 5.8)
- G = (4.71, 3.59)
- a = 5.05
- b = 4.3
- c = 3.92
- d:  $-0.88x - 0.47y = -6.93$
- e:  $0.17x + 0.99y = 5.46$
- f:  $-0.02x + 3.09y = 14.15$
- g:  $3.3x - 2.24y = 7.47$
- h:  $-3.28x - 0.85y = -21.6$
- k:  $(x - 6.64)^2 + (y - 2.28)^2$
- p:  $(x - 2.46)^2 + (y - 5.12)^2$
- q:  $(x - 6.62)^2 + (y - 6.2)^2$

Полотно

Условие задачи

Построить окружность D доб. объект такую что (D касается p) и (D касается q) и (D касается k) доб. условие

Пользуясь инструментами:

- ✓ Циркуль
- ✓ Прямая по 2-м точкам **добавить инструмент**

Условие для ученика:

Построить окружность касающиеся 3-х данных. Пользуясь только циркулем линейкой.

Ввод:

# Интерфейс ученика

Файл Правка Вид Настройки Инструменты Окно Справка

Переместить чертёж  
Перетащите чертёж или ось (с нажатой клавишей Shift)

Полотно × Условие задачи

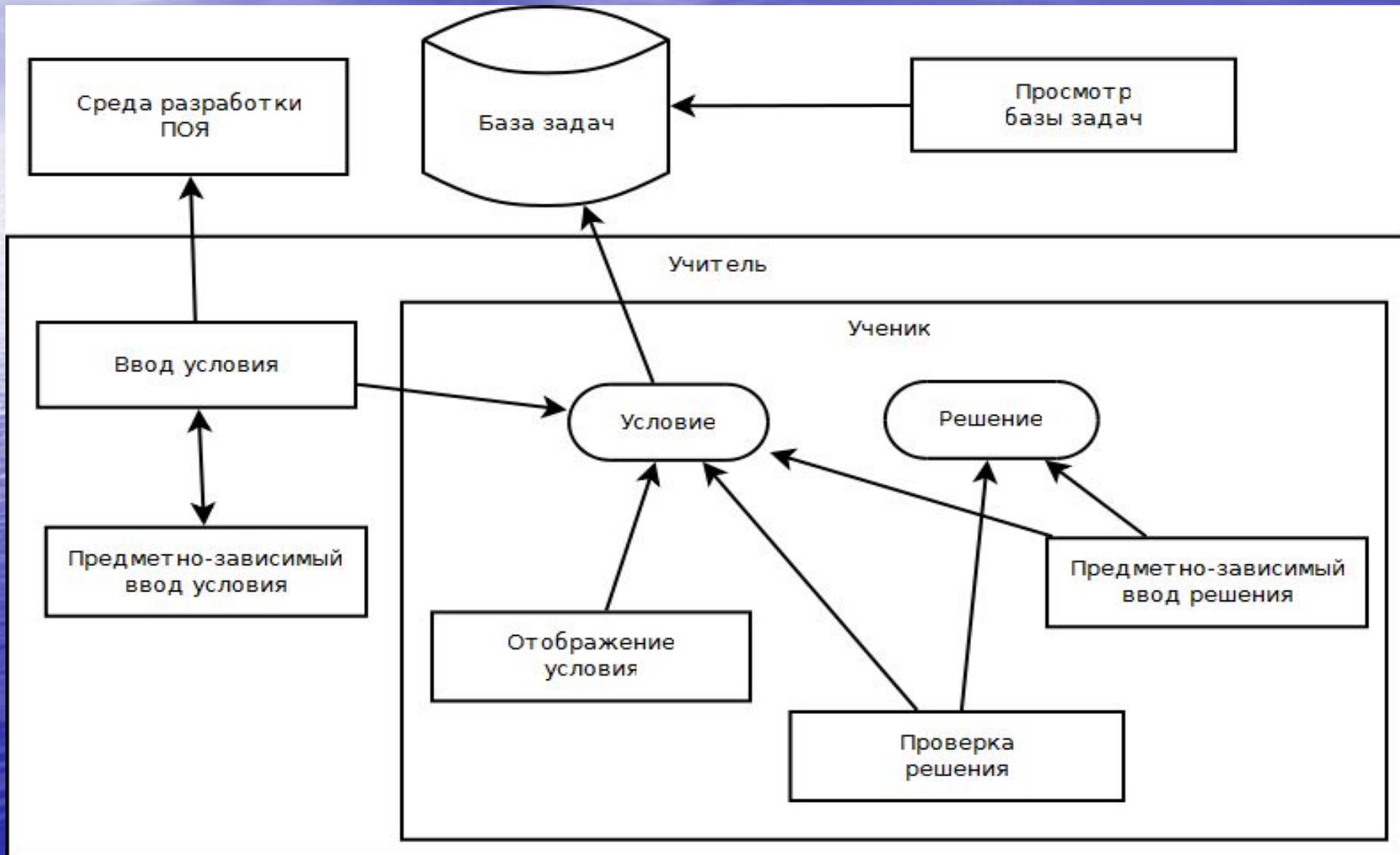
Построить окружность касающуюся 3-х данных используя только циркуль и линейку.

Проверить

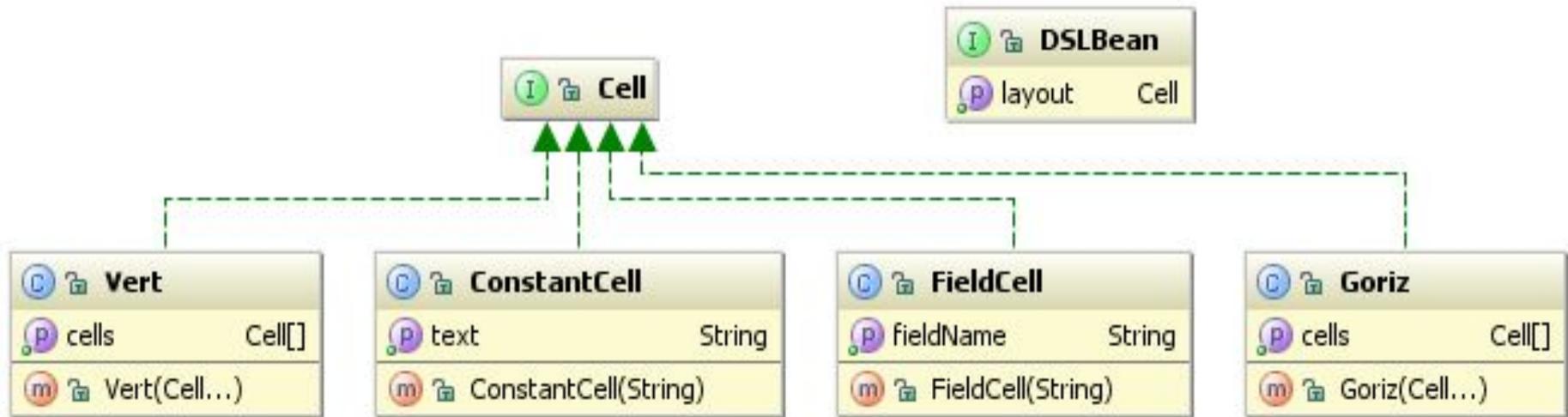
Последняя проверка:  
Условие не выполнено

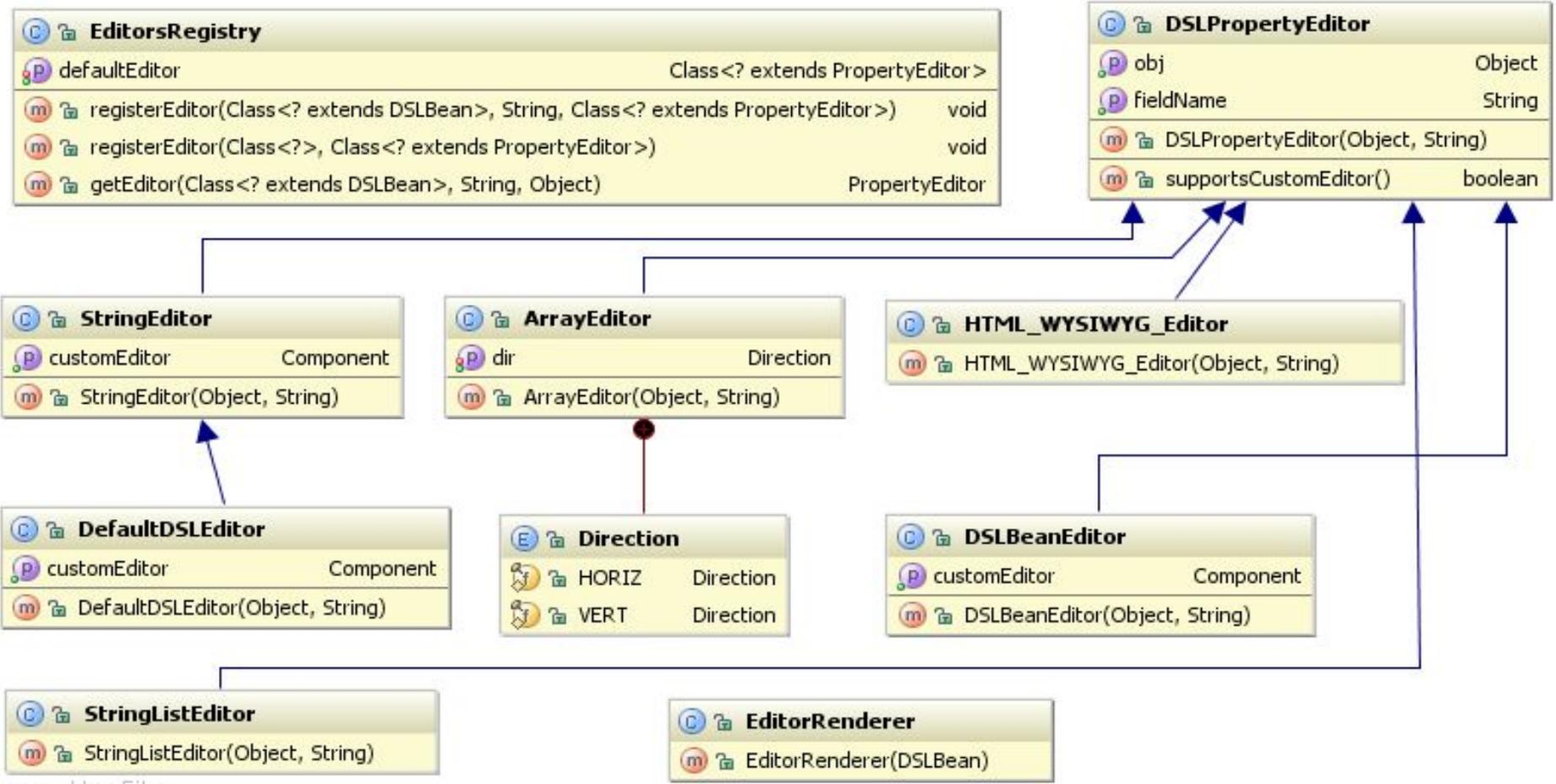
Ввод: z α

# Разбиение системы на модули



# Элементы редактора

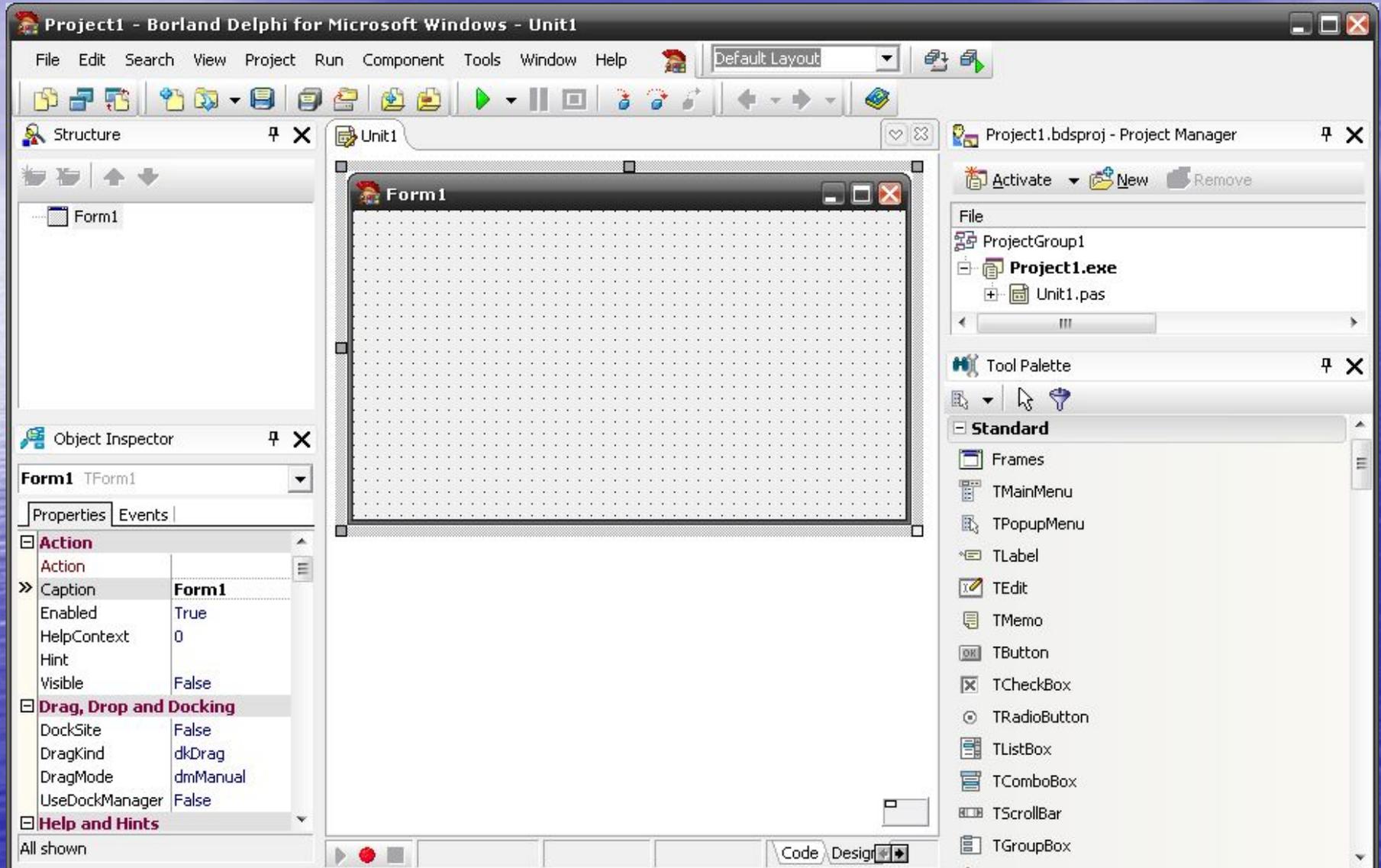


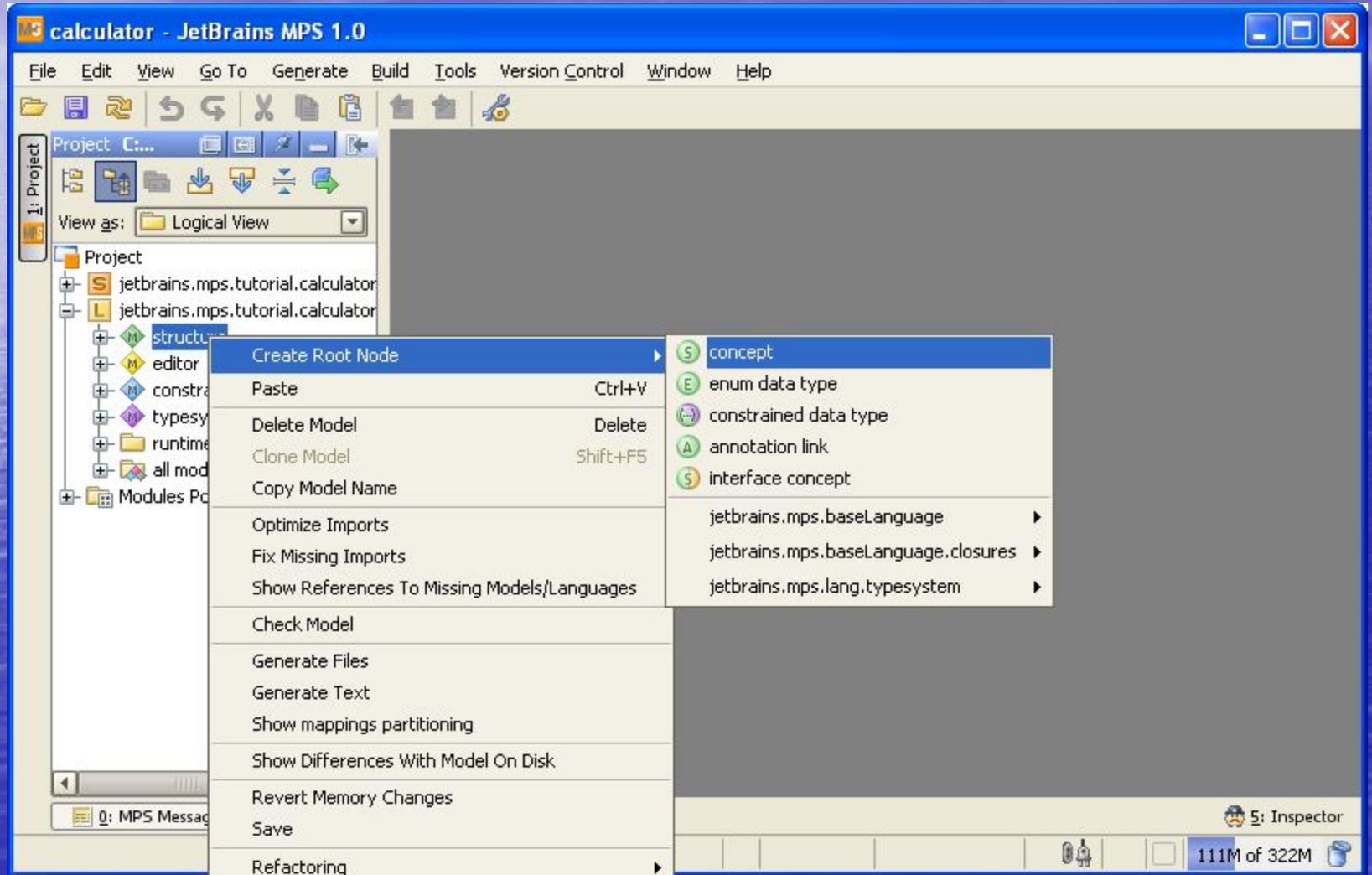


# Delphi – предметная ориентированность

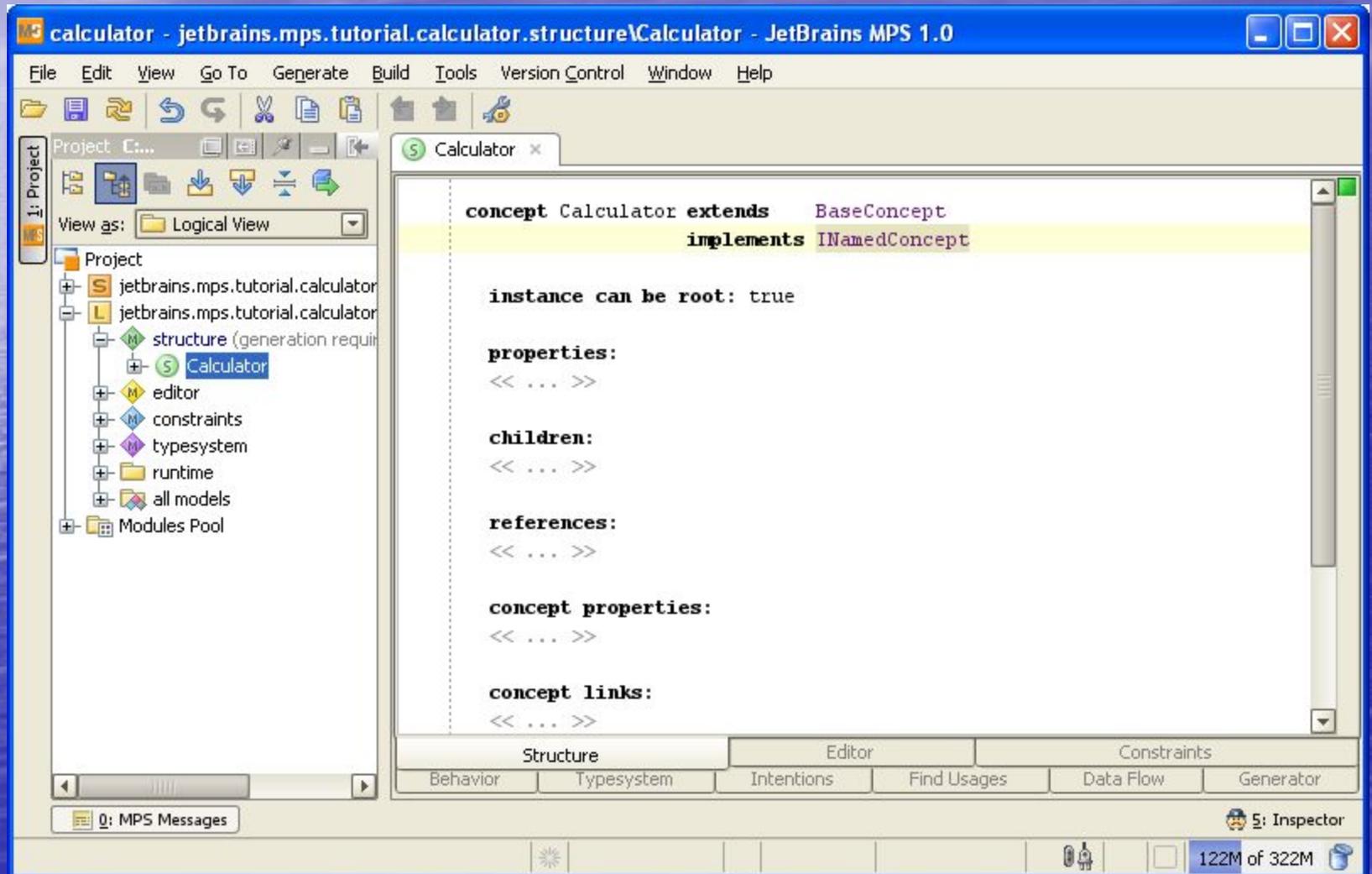
- Редактор интерфейса позволяет визуально (без программирования) нарисовать большую часть интерфейса. Процесс создания нагляден.
- Встроенные средства рефакторинга позволяют «переименовать» классы, методы, компоненты, модули в любой момент когда вы обнаружите несоответствие реального использования класса, 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

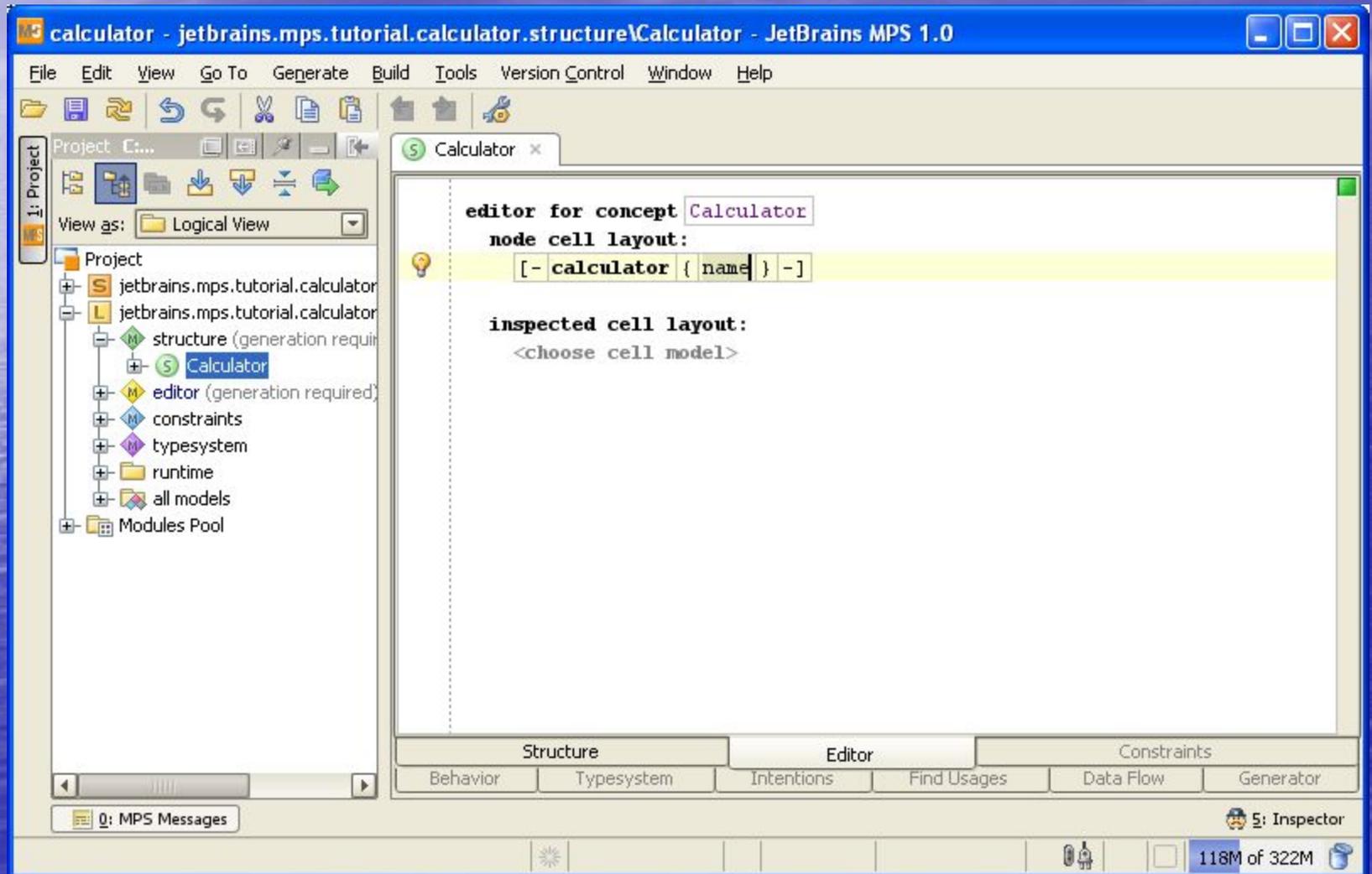




# Создание языка



# Создание редактора



**editor for concept** Calculator

**node cell layout:**

```
[ - calculator { name } - ]
```

**inspected cell layout:**

```
<choose cell model>
```

Structure

Editor

Constraints

Behavior

Typesystem

Intentions

Find Usages

Data Flow

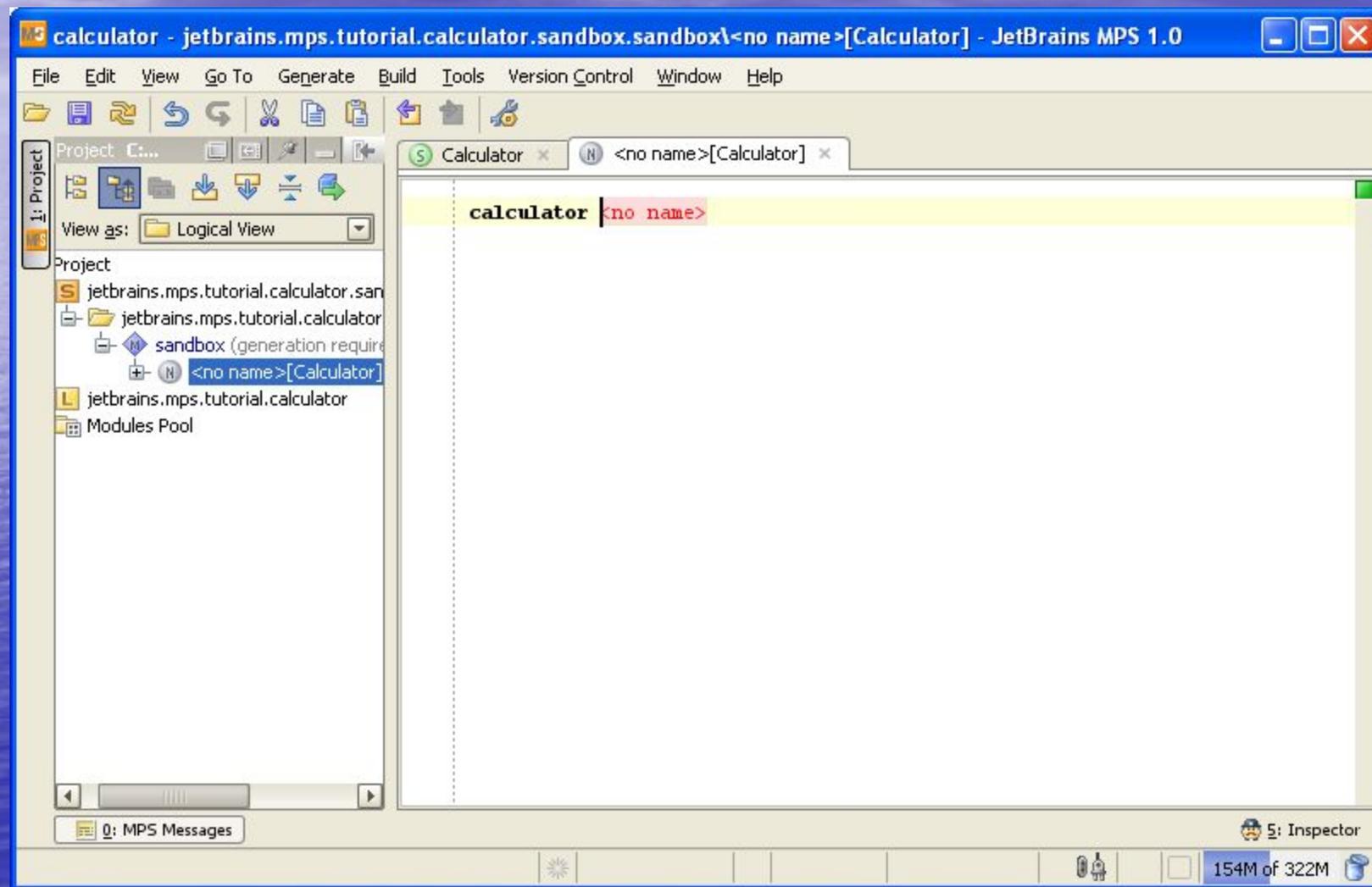
Generator

0: MPS Messages

5: Inspector

118M of 322M

# Использование - sandbox



# Создаём поле ввода данных

The screenshot displays the JetBrains MPS 1.0 IDE interface. The title bar reads "calculator - jetbrains.mps.tutorial.calculator.structure\InputField - JetBrains MPS 1.0". The menu bar includes File, Edit, View, Go To, Generate, Build, Tools, Version Control, Window, and Help. The toolbar contains various icons for file operations and development tools. On the left, the Project Explorer shows a tree structure for the "jetbrains.mps.tutorial.calculator" project, with "InputField" selected under the "structure" folder. The main editor window shows the following code:

```
concept InputField extends BaseConcept
    implements INamedConcept

instance can be root: false

properties:
<< ... >>

children:
<< ... >>

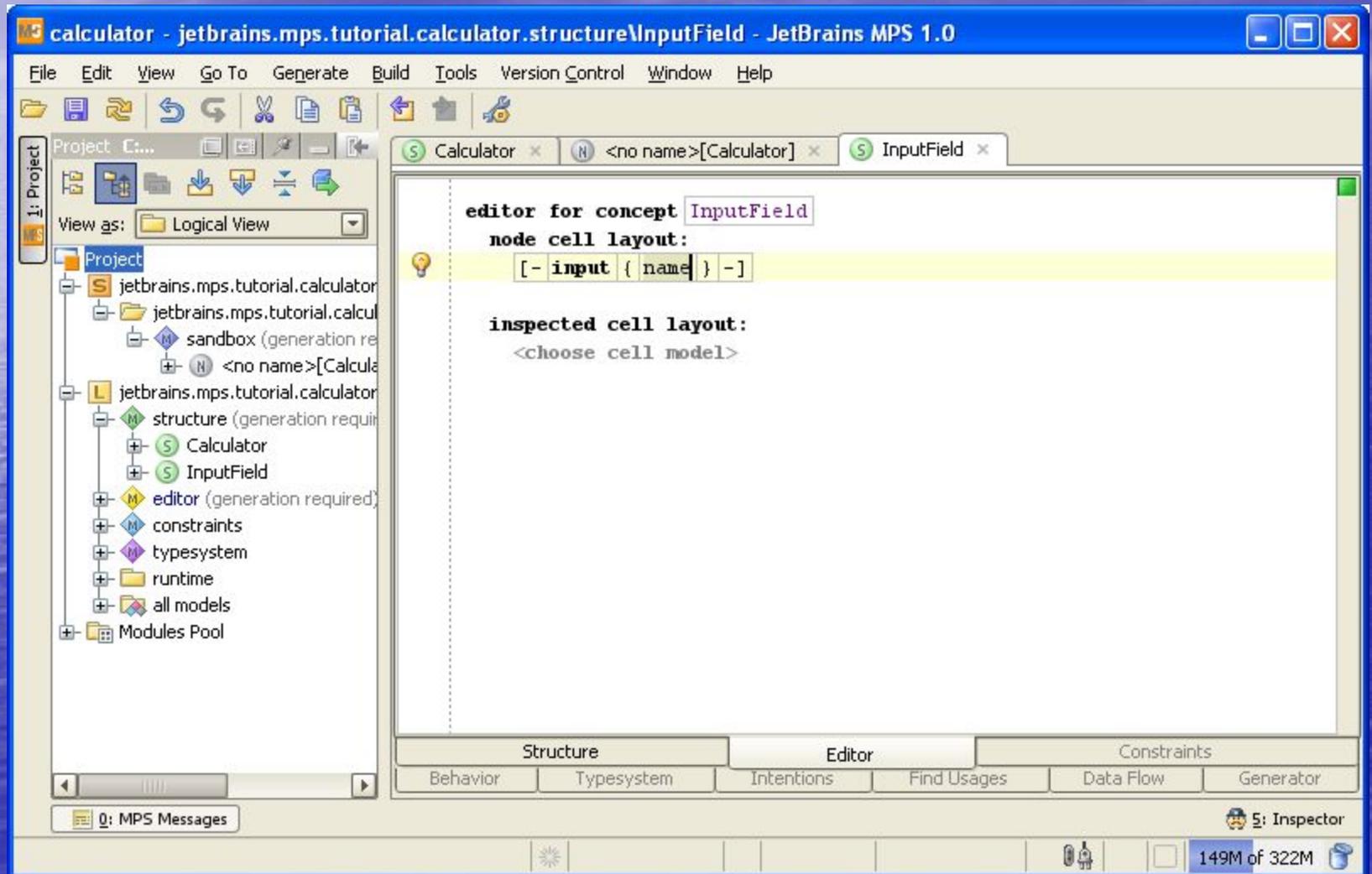
references:
<< ... >>

concept properties:
<< ... >>

concept links:
<< ... >>
```

At the bottom of the editor, there are tabs for "Structure", "Editor", and "Constraints". Below these are buttons for "Behavior", "Typesystem", "Intentions", "Find Usages", "Data Flow", and "Generator". The status bar at the bottom right shows "Inspector" and "141M of 322M".

# И редактор для него



# Добавляем поле в калькулятор

The screenshot displays the JetBrains MPS 1.0 IDE interface. The main editor window shows the following code for the `Calculator` concept:

```
concept Calculator extends BaseConcept
    implements INamedConcept

instance can be root: true

properties:
<< ... >>

children:
InputField inputField 0..1 specializes: <none>

references:
<< ... >>

concept properties:
<< ... >>

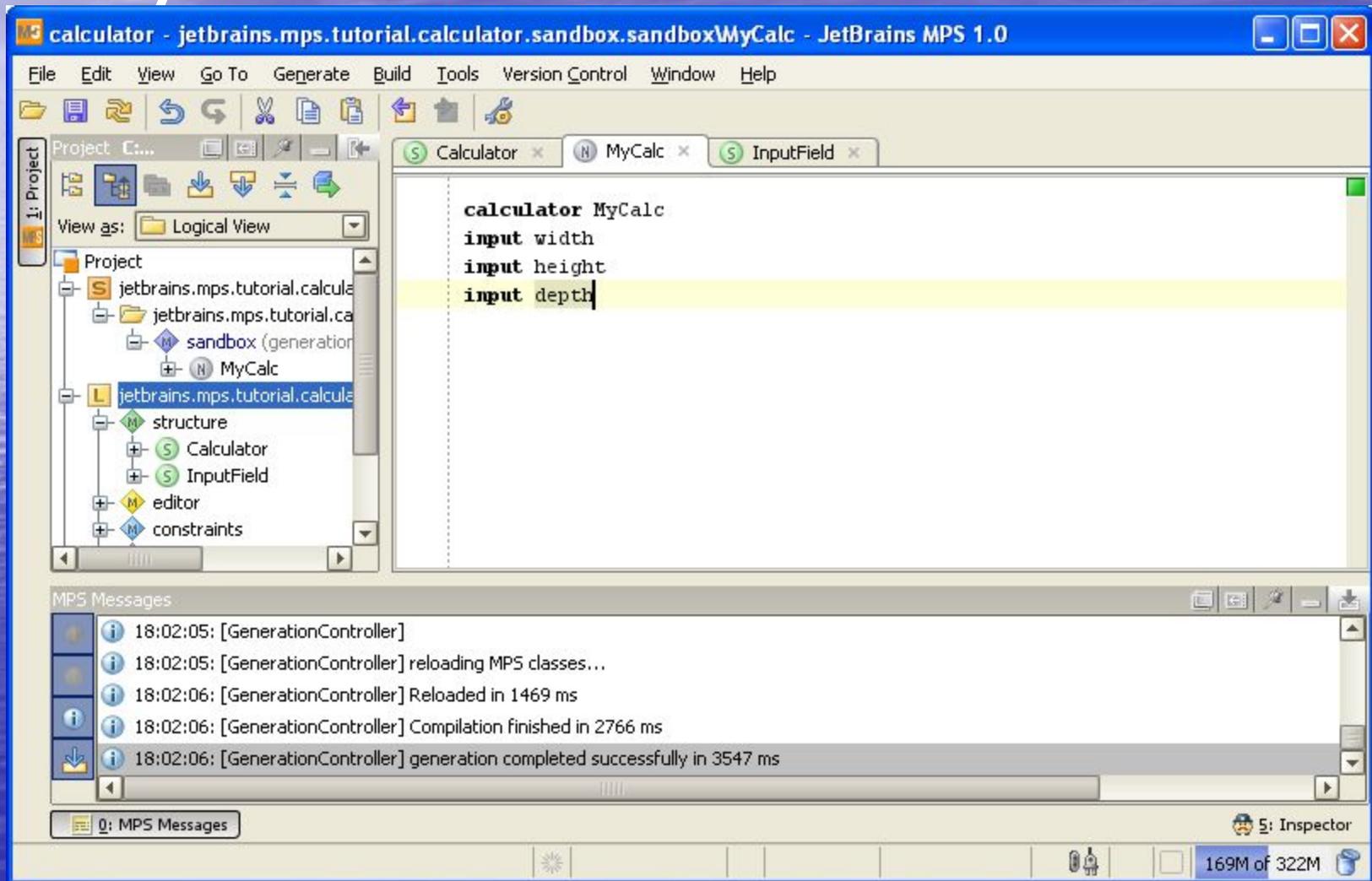
concept links:
<< ... >>
```

The `InputField` child is highlighted in yellow. The IDE's left sidebar shows a project tree with the following structure:

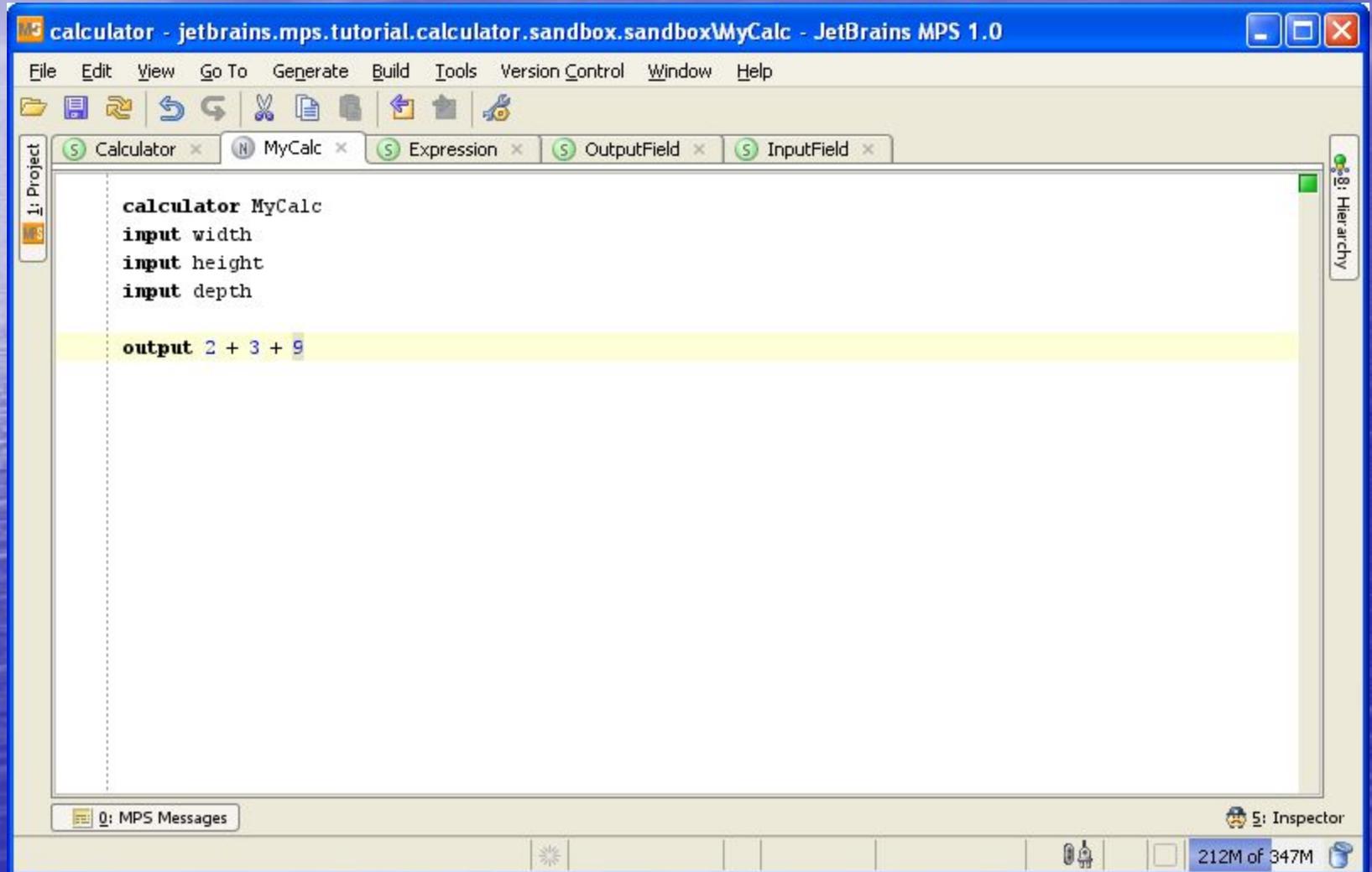
- Project
  - jetbrains.mps.tutorial.calculator
    - jetbrains.mps.tutorial.calculator
      - sandbox (generation required)
        - <no name>[Calculator]
jetbrains.mps.tutorial.calculator
      - structure (generation required)
        - Calculator
        - InputField
      - editor (generation required)
      - constraints
      - typesystem
      - runtime
      - all models
Modules Pool

The bottom of the IDE shows the `Inspector` tab active, displaying `155M of 322M` of memory usage.

# Генерируем и тестируем полученный язык



# Выходное поле и формула



# Можем использовать значение ВХОДНЫХ ПОЛЕЙ

The screenshot displays the JetBrains MPS 1.0 IDE interface. The title bar reads "calculator - jetbrains.mps.tutorial.calculator.sandbox.sandboxMyCalc - JetBrains MPS 1.0". The menu bar includes File, Edit, View, Go To, Generate, Build, Tools, Version Control, Window, and Help. The toolbar contains various icons for file operations and development actions.

The left sidebar shows a project tree with the following structure:

- Project
  - jetbrains.mps.tutorial.calculator
    - jetbrains.mps.tutorial.calculator.sandbox (generation root)
      - MyCalc
    - jetbrains.mps.tutorial.calculator.structure
      - Calculator
      - InputField
      - InputFieldReference
      - OutputField
    - editor
    - constraints
    - typesystem
    - runtime
    - all models
  - Modules Pool

# Создаем генератор кода

The screenshot displays the JetBrains MPS IDE interface. The main editor window shows the implementation of a `CalculatorImpl` class. The code includes a macro `<add members (ctrl+space)>` and two `$LOOP$` blocks for initializing `inputField` and `outputField`. The `public CalculatorImpl()` constructor sets the window title, layout, and default close operation.

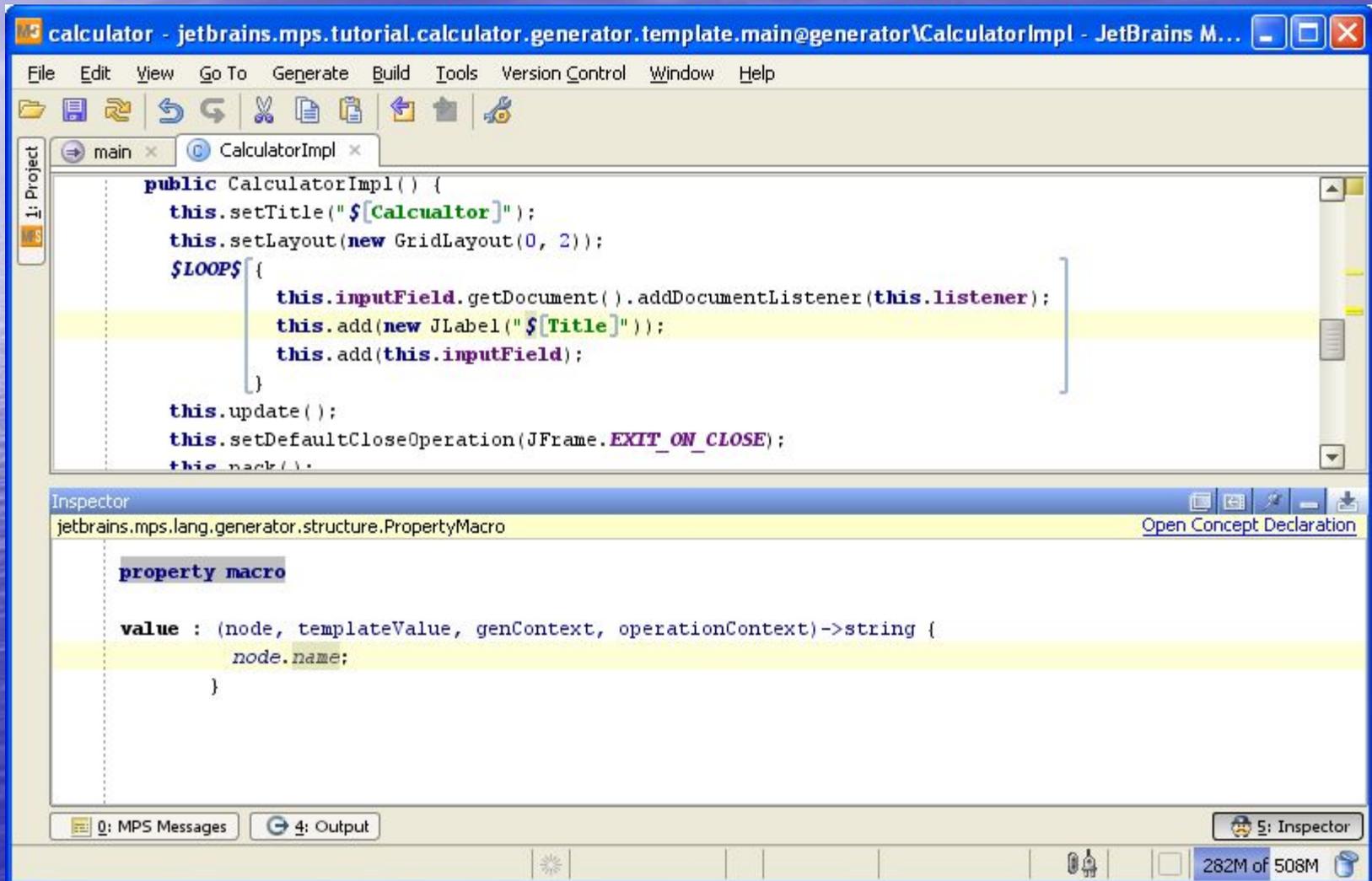
```
    }  
    <add members (ctrl+space)>  
  }:  
  $LOOP$private JTextField $[inputField] = new JTextField(); ]  
  $LOOP$private JTextField $[outputField] = new JTextField(); ]  
  <<properties>>  
  <<initializer>>  
  public CalculatorImpl() {  
    this.setTitle(" $[Calcualtor]");  
    this.setLayout(new GridLayout(0, 2));  
    this.update();  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.pack();  
  }
```

The Inspector panel at the bottom shows the `jetbrains.mps.lang.generator.structure.PropertyMacro` concept declaration:

```
property macro  
  
value : (node, templateValue, genContext, operationContext)->string {  
  genContext.unique name from ("outputField") in context (<no node>);  
}
```

The status bar at the bottom indicates 303M of 508M memory usage.

# Создание полей «в цикле»



The screenshot displays the JetBrains MPS IDE interface. The main editor window shows the implementation of the `CalculatorImpl` class. A loop macro, denoted by `$LOOP$`, is used to dynamically add components to the GUI. The code is as follows:

```
public CalculatorImpl() {  
    this.setTitle("${Calcualtor}");  
    this.setLayout(new GridLayout(0, 2));  
    $LOOP${  
        this.inputField.getDocument().addDocumentListener(this.listener);  
        this.add(new JLabel("${Title}"));  
        this.add(this.inputField);  
    }  
    this.update();  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.pack();  
}
```

The Inspector window at the bottom shows the definition of the `property macro` used in the code:

```
property macro  
  
value : (node, templateValue, genContext, operationContext)->string {  
    node.name;  
}
```

The status bar at the bottom indicates 282M of 508M memory usage.

# Ссылка на другой макрос

The screenshot shows an IDE window titled "calculator - jetbrains.mps.tutorial.calculator.generator.template.main@generator\CalculatorImpl - JetBrains M...". The main editor displays the following code:

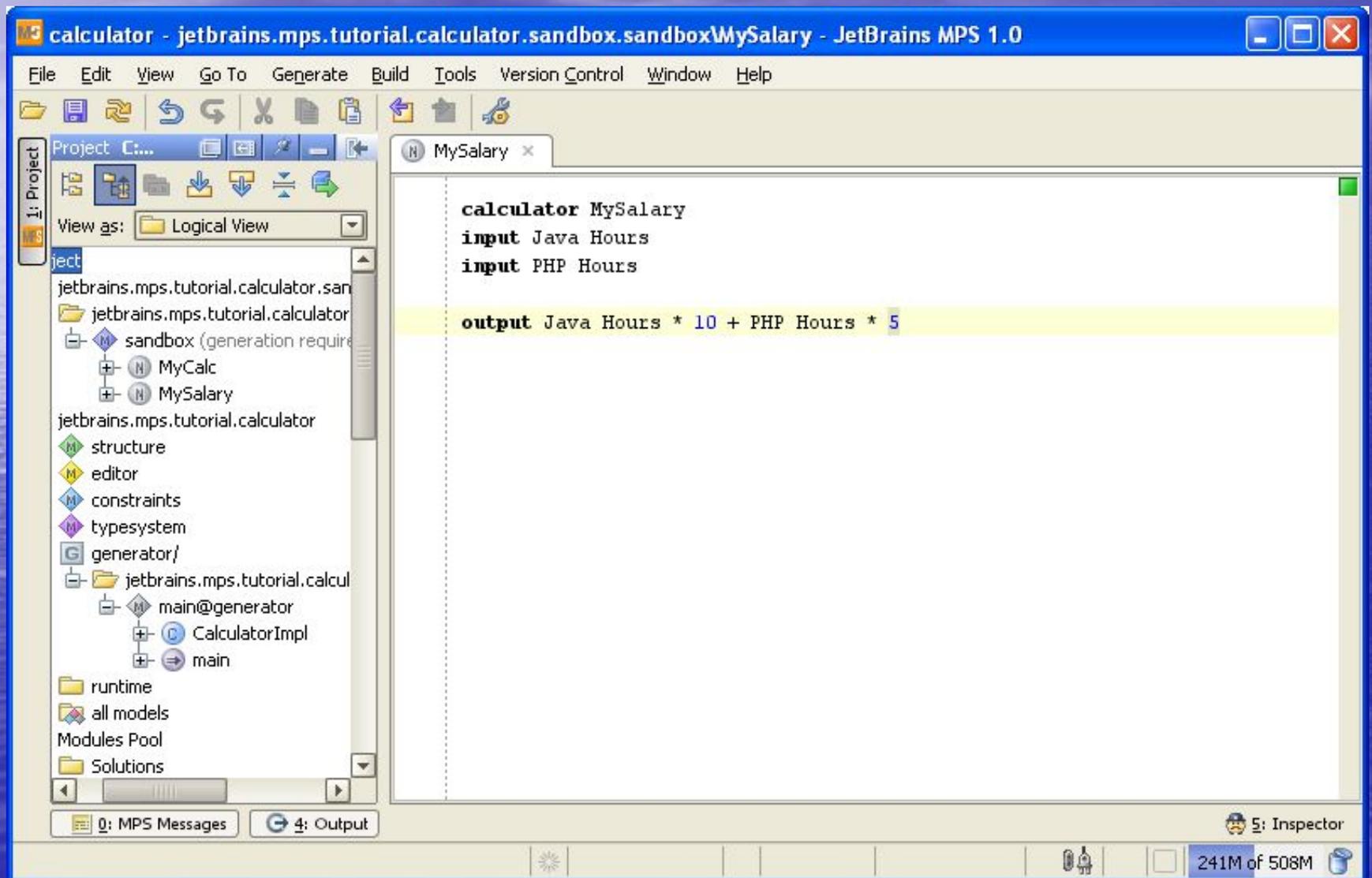
```
$LOOPS[private JTextField $[outputField] = new JTextField();]  
<<properties>>  
<<initializer>>  
public CalculatorImpl() {  
    this.setTitle("$[Calcualtor]");  
    this.setLayout(new GridLayout(0, 2));  
    $LOOPS[{  
        this->[$[inputField].getDocument().addDocumentListener(this.listener);  
        this.add(new JLabel("$[Title]"));  
        this.add(this.inputField);  
    }]  
    this.update();  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```

The Inspector panel at the bottom shows the definition of the `reference macro`:

```
referent : (node, outputNode, genContext, operationContext)->join(node<FieldDeclaration> | String) {  
    genContext.get output InputFieldDeclaration for (node);  
}
```

The Inspector also shows the path `jetbrains.mps.lang.generator.structure.ReferenceMacro` and a link to [Open Concept Declaration](#).

# Пример использования



# Готовая программа

The screenshot shows an IDE window titled "calculator - ...\solutions\jetbrains.mps.tutorial.calculator.sandbox\source\_gen\jetbrains\mps\tutorial\calculat...". The interface includes a menu bar (File, Edit, View, Go To, Generate, Build, Tools, Version Control, Window, Help), a toolbar, and a project explorer on the left. The project explorer shows a tree structure for the "calculator" project, with the "MySalary.java" file selected under the "source\_gen" directory. The main editor displays the following Java code:

```
private JTextField outputField1 = new JTextField();

public MySalary() {
    this.setTitle("MySalary");
    this.setLayout(new GridLayout(0, 2));
    this.inputField3.getDocument().addDocumentListener(this.listen
    this.add(new JLabel("Java Hours"));
    this.add(this.inputField3);
    this.inputField4.getDocument().addDocumentListener(this.listen
    this.add(new JLabel("PHP Hours"));
    this.add(this.inputField4);
    this.add(new JLabel("Output"));
    this.add(this.outputField1);
    this.update();
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.pack();
    this.setVisible(true);
}

public void update() {
    int i3 = 0;
```

The IDE status bar at the bottom shows "0: MPS Messages", "4: Output", "Inspector", "1:1", "Insert", "windows-1251", and "260M of 508M".

Спасибо за внимание!

Вопросы?