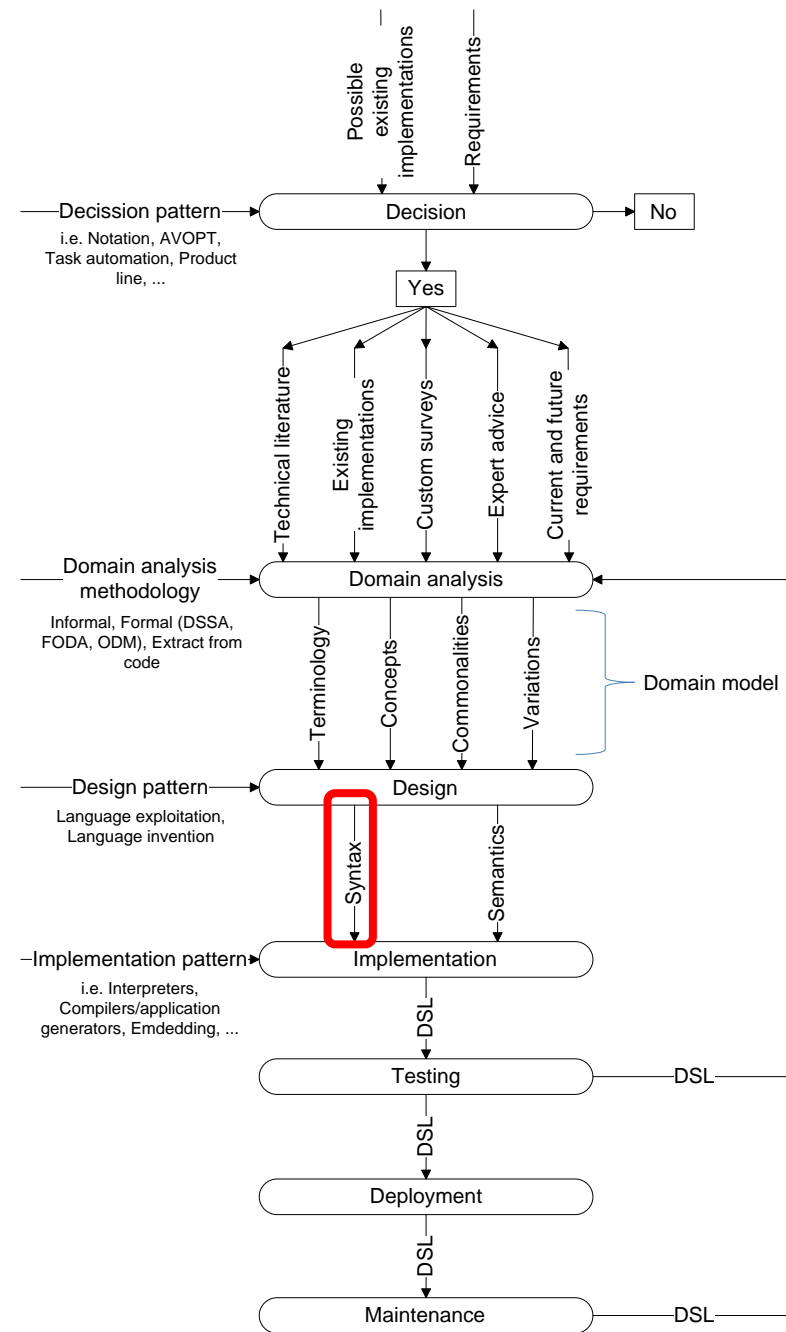


# DOMENSKO SPECIFIČNI MODELIRNI JEZIKI 2024/2025

4. predavanje

# Življenjski cikel



# Načrtovanje DSL – sintaksa

## ■ Različni zapisi sintakse

- **Kontestno proste gramatike (CFG)**
- Abstraktno sintaksno drevo (AST)
- Konceptualni razredni diagram oz. metamodeli
- DTD in XML sheme
- itd.

# Načrtovanje DSL – sintaksa (ponovitve)

- $G = (T, N, S, P)$ ,  $P \subseteq N \times (N \cup T)^*$ ,  $S \in N$ 
  - $T$  = terminal,  $N$  = netermina,  $S$  = startni terminal,  $P$  = produkcije
- **Produkcija** je naslednje oblike  $A \rightarrow \alpha$  ;  $\alpha \in (N \cup T)^*$
- Sintentična oblika  $\gamma$   $G$  je katerikoli **niz terminalnih in neterminalnih simbolov**, ki jih lahko dobimo iz  $S$ .
  - $\gamma_1$  dobimo iz  $\gamma_2$  ( $\gamma_1 \Rightarrow \gamma_2$ ): if  $\gamma_1 = \sigma\alpha\tau$  in  $\gamma_2 = \sigma\beta\tau$  in  $\alpha \rightarrow \beta$
  - $\gamma_1$  dobimo iz  $\gamma_2$  ( $\gamma_1 \Rightarrow^* \gamma_2$ ): if  $\gamma_1 \Rightarrow \sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \gamma_2$

# Načrtovanje DSL – sintaksa (ponovitve)

- **Kontekstno prost jezik**  $L(G)$  za gramatiko  $G$ , je **množica vseh nizov**, ki so **sestavljani samo iz terminalnih simbolov**, ki jih lahko dobimo iz začetnega simbola  $S$  s sekvenčno uporabo produkcijskih pravil.
- $L(G) = \{x \mid S \Rightarrow^* x \wedge x \in T^*\}$

# Načrtovanje DSL - sintaksa

## ■ Vloga neterminalnih simbolov v kontekstno prosti gramatiki je dvojna:

- Na višjem nivoju abstrakcije se neterminalni simboli uporabljajo za **opis konceptov** (na primer v splošnonamenskih programskih jezikih: izraz, deklaracija).
- Na konkretni ravni se za **opis strukture koncepta** uporabijo neterminalni in terminalni simboli (npr. izraz je sestavljen iz dveh operandov, ločenih s simbolom operatorja, spremenljivka je sestavljena iz deklaracije tipa in imena spremenljivke).



# Načrtovanje DSL – sintaksa (konkretna, dekorirana)

## ■ Primer gramatike – jezik FDL

FDL	→ FeatureDef+ Constraint*
FeatureDef	→ featureName : FeatureExpr
FeatureList	→ FeatureExpr , FeatureList   FeatureExpr
FeatureExpr	→ all (FeatureList)   one-of (FeatureList)   more-of(FeatureList)   OptFeature   Feature
OptFeature	→ Feature ?
Constraint	→ Feature require Feature   Feature exclude Feature
Feature	→ featureName

## ■ Program

Car: all(carBody, Transmission, Engine, HorsePower, pullsTrailer?)  
Transmission: one-of(automatic, manual)  
Engine: more-of(electric, gasoline)  
HorsePower: one-of(lowPower, mediumPower, highPower)



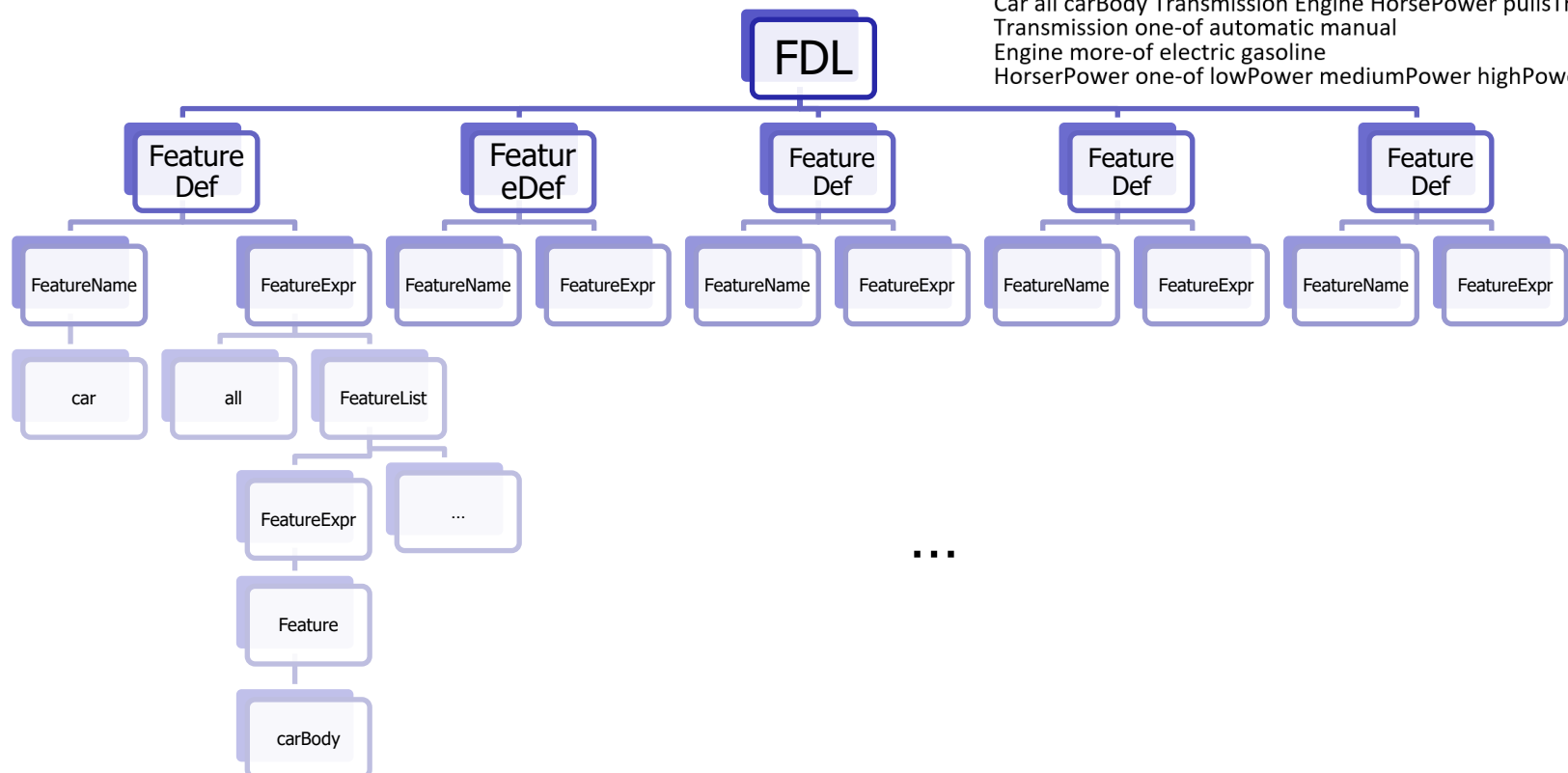
# Načrtovanje DSL – abstraktno sintaksno drevo (nedekorirana)

Primer gramatike – jezik FDL

FDL	→ FeatureDef+ Constraint*
FeatureDef	→ featureName FeatureExpr
FeatureList	→ FeatureExpr+
FeatureExpr	→ all (FeatureList)   one-of (FeatureList)   more-of (FeatureList)   OptFeature   Feature
OptFeature	→ Feature ?
Constraint	→ Feature require Feature   Feature exclude Feature
Feature	→ featureName

Program

Car all carBody Transmission Engine HorsePower pullsTrailer  
Transmission one-of automatic manual  
Engine more-of electric gasoline  
HorsePower one-of lowPower mediumPower highPower





# Abstraktna vs. konkretna sintaksa

- **Abstraktna sintaksa** je abstraktna, ker izpušča „nepomembne“ podrobnosti jezika
  - Kaj je „nepomembno“?
- Vsak jezik L ima **vsaj eno konkretno sintakso**
- Programerji, ki uporabljajo jezik L uporabljajo njegovo konkretno sintakso

# Abstraktna vs. konkretna sintaksa

## ■ Primer

### ■ Abstraktna sintaksa

`MyArdoLang`  $\rightarrow$  `Program Setup+ Loop+`

### ■ Konkretna sintaksa

`MyArdoLang`  $\rightarrow$  `'program' Program`  
`'{' 'init' Setup+ '}'`  
`'{' 'start' Loop+ '}'`

# BNF

- Okrajšava za Backus-Naur Form
- Formalni, matematični način za zapis kontekstno proste gramatike
- Natančen in nedvoumen

# EBNF

- EBNF prinaša nekaj **preprostih razširitev** za BNF, ki omogočijo lažje izražanje gramatike
- EBNF bolj zgoščen od BNF
- EBNF je **enako "močen" zapis** kot BNF;
  - vse, kar se lahko izrazi v EBNF se lahko izrazi tudi v BNF
- EBNF se uporablja kot **standard** za zapis programskih jezikov

# EBNF razširitve

## ■ Razširitve izpeljane iz **regularnih izrazov**

- \*: pomeni 0 ali več ponovitev (včasih se uporablja {...})
- +: pomeni 1 ali več ponovitev
- ?: pomeni 0 ali 1 ponovitev (včasih se uporablja [...])
- () : združevanje

# BNF vs. EBNF

## ■ Gramatika realnih števil

### ■ BNF

$\text{expr} ::= '-' \text{ num} \mid \text{ num}$

$\text{num} ::= \text{ digits} \mid \text{ digits} '.' \text{ digits}$

$\text{digits} ::= \text{ digit} \mid \text{ digit digits}$

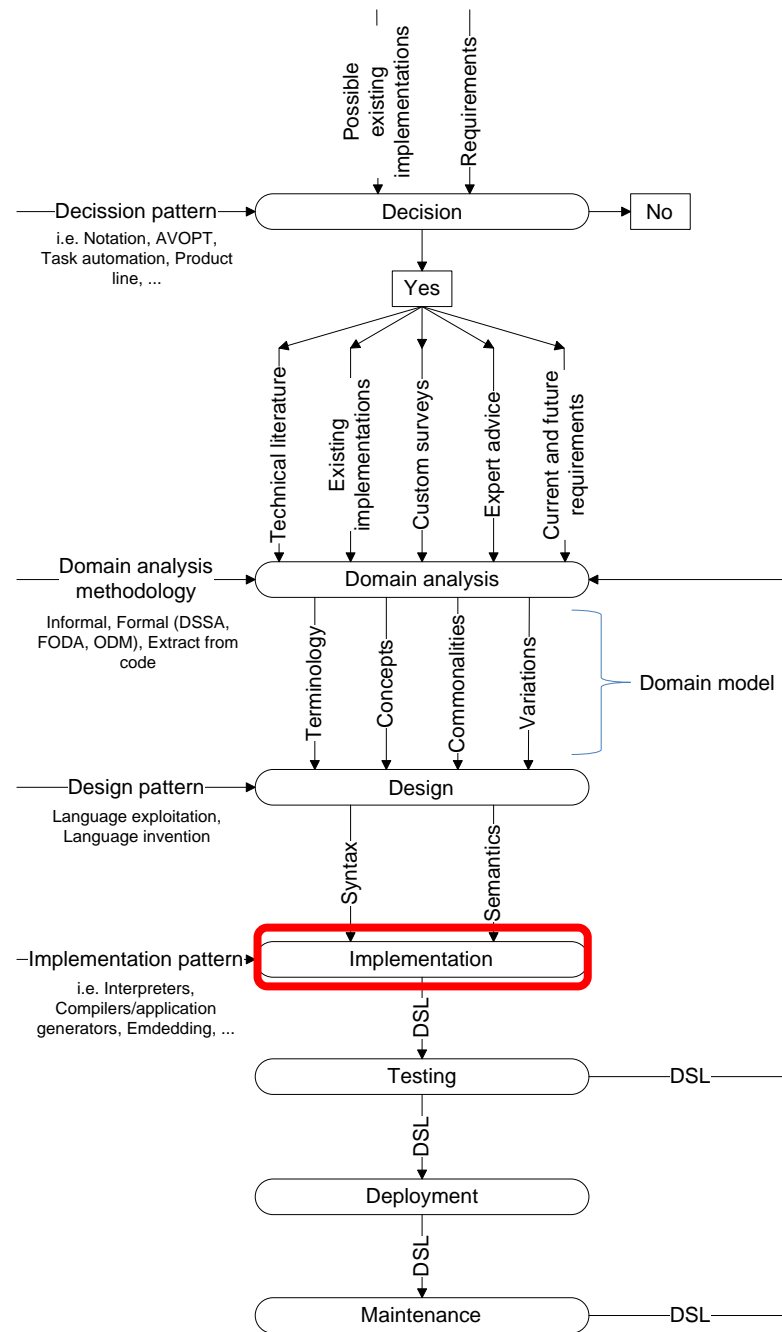
$\text{digit} ::= '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$ ,

### ■ EBNF

$\text{num} ::= '-'? \text{ digit}^+ ('.' \text{ digit}^+)?$

$\text{digit} ::= '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$

# Življenjski cikel



# Jetbrains MPS

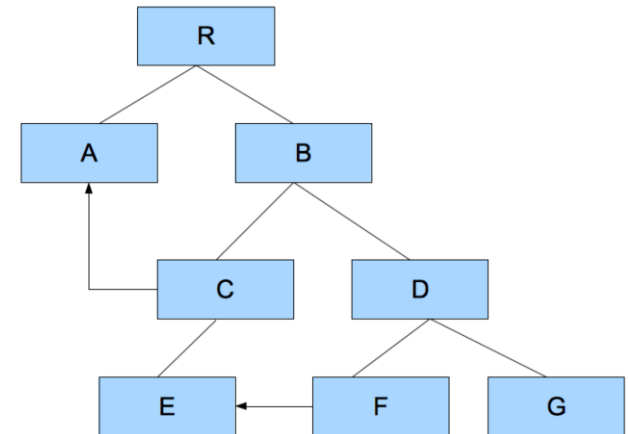


- **Orodje za razvoj jezikov**  
(angl. Language Workbench)
  - Prenos orodja: [link](#)
- **Jetbrains**
  - IntelliJ IDEA, PhpStorm, PyCharm, CLion (orodja)
  - Kotlin, Ktor, **MPS**, Compose for Desktop (jeziki in ogrodja)
- **Temelji na uporabi abstraktne sintaksnega drevesa (AST)**
  - Za urejanje drevesa se uporabljajo **projekcijski urejevalniki besedila**
- **2009 (verzija 1.0), trenutna verzija 2024.1.1 ([link](#))**
- **Alternativa**
  - Xtext (Eclipse, sintaksa podobna EBNF)



# Abstraktno sintaksno drevo (AST)

- Program vedno predstavlja AST

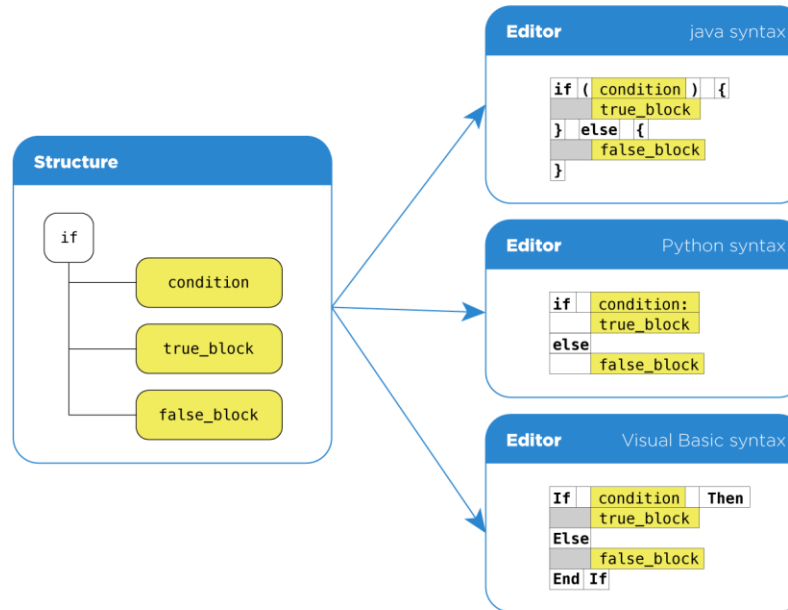


Vir: [link](#)

- Kodo **urejamo** kot AST, shranimo v AST ter tudi prevedemo
- Omogoča, da se **izognemo definiranju gramatike** in gradnji razpoznavalnika (angl. parser)
- Jezik definiramo s pomočjo **vozlišč** AST in definiramo **medsebojne odnose**

# Projekcijski urejevalnik

- Projekcijski urejevalniki neposredno **manipulirajo z AST**
- Razpoznavanje (angl. parsing) ni potrebno, saj je koda **vedno v AST**
- **Urejevalnik predstavi AST** tako, kot smo oblikovali jezik
- Za AST je možno naredi **različne predstavitve**



Vir: [link](#)

# Vozlišča

- **Vozlišča** (angl. nodes) sestavljajo abstraktno sintaksno drevo
- **Korensko** vozlišče najvišje v hierarhiji drevesa
- Vsako vozlišče ima starševsko vozlišče (razen korenskega vozlišča)
- Vozlišča vsebujejo:
  - **Lastnosti** (angl. properties)
  - **Vozlišča potomcev** (angl. children)
  - **Reference** na druga vozlišča (angl. references)
- Vozlišča predstavljajo **neterminalne simbole** iz EBNF, lastnost pa **terminalne** simbole (razen rezerviranih besed)

# Koncepti v MPS

- Ko programiramo jezik v MPS, se v ozadju gradi AST, katerega posamezne „delčke“ definiramo s **koncepti**
- **Koncept** je deklaracija vozlišča
- Povezuje z **vrsto povezana vozlišča** v abstraktnem sintaktičnem drevesu

# Structure language

- Ker MPS ne pozna gramatike, potrebujemo **drug način za definiranje strukture**
- Za definiranje konceptov uporabljamo **Structure language**
- Structure language:
  - **Concept**
  - Concept interface
  - Constrained data type
  - **Enumeration**
  - Attribute (annotation)

# Structure language: Concept

## ■ Koncept sestavljajo:

### ■ **Properties**

- **Vrednost** znotraj primerka koncepta

### ■ **Children**

- Omogočajo sestavljati koncept s pomočjo drugih in tako gradimo drevo

### ■ **References**

- Reference na druga vozlišča (ime, vrsta in kardinalnost)



# Structure language: Concept

- Konzept

- Poimenovan koncept

- Lastnosti (properties)

```
concept PrintingUnit extends BaseConcept  
                        implements INamedConcept
```

```
    instance can be root: false  
    alias: <no alias>  
    short description: <no short description>
```

```
properties:  
    prints : string
```

```
children:  
<< ... >>
```

```
references:  
<< ... >>
```



# Korenski koncept

## ■ Korenski koncept

## ■ Alias:

```
concept Printer extends BaseConcept
    implements INamedConcept
    instance can be root: true
    alias: printer
    short description: <no short description>

    properties:
    product : integer

    children:
    listOfPrintingUnits : PrintingUnit[0..n]

    references:
    << ... >>
```



# Structure language: Enumeration

- **Enumeration** omogoča definicijo lastnosti iz seznama članov
- Člani so podani z:
  - name (angl. ime) in
  - vrednost (angl. value).
- Name je obvezen, value opcijski.
- Privzeta lastnost:
  - Izberemo člana

```
E WeekDays
enumeration WeekDays
members:
  • Mon monday (default)
  • Tue tuesday
  • Wed wednesday
  • Thu thursday
  • Fri friday
  • Sat saturday
  • Sun sunday
default member: Mon
```

# Privzeta projekcija

- Po zapisu strukture, lahko takoj preverimo privzeto projekcijo
  - Peskovnik (angl. sandbox)

```
printer test {  
  product : 30  
  
  list of printing units :  
    printing unit test {  
      prints : a  
    }  
}
```

Privzeta

```
printer test {  
  product : 30  
  unit test : a  
  unit unit1 : b  
  unit unit2 : c  
}
```

Želena

# Definiranje projekcije koncepta

- V projektu direktorij „editor“ lahko najdemo primere naslednjih definicij

```
<default> editor for concept PrintingUnit
```

```
node cell layout:
```

```
[- unit { name } : { prints } -]
```

```
inspected cell layout:
```

```
<choose cell model>
```

```
<default> editor for concept Printer
```

```
node cell layout:
```

```
[-
```

```
[- printer {name} {-]
```

```
[- product : {product} -]
```

```
[/
```

```
(/ % listOfPrintingUnits % /)
```

```
/empty cell: <default>
```

```
}
```

```
/]
```

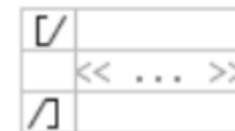
```
-]
```

# Definiranje projekcije koncepta

■ Editor je razdeljen na celice (ang. cell)

■ Razporeditev

- Vertikalna
- Horizontalna
- Z zamikom (angl. indent)



■ Celice z informacijami

- Lastnosti
- Potomci
- Reference
- Lista potomcev

{ property }

% child %

( % *reference* % -> { name } )

(> % child % /empty cell: <default> <)

■ Dodatno zamikanje



# Testiranje

- Ponovno prevedemo jezik (angl. Rebuild language)
- Editor v peskovniku (angl. sandbox)
- Z želeno notacijo

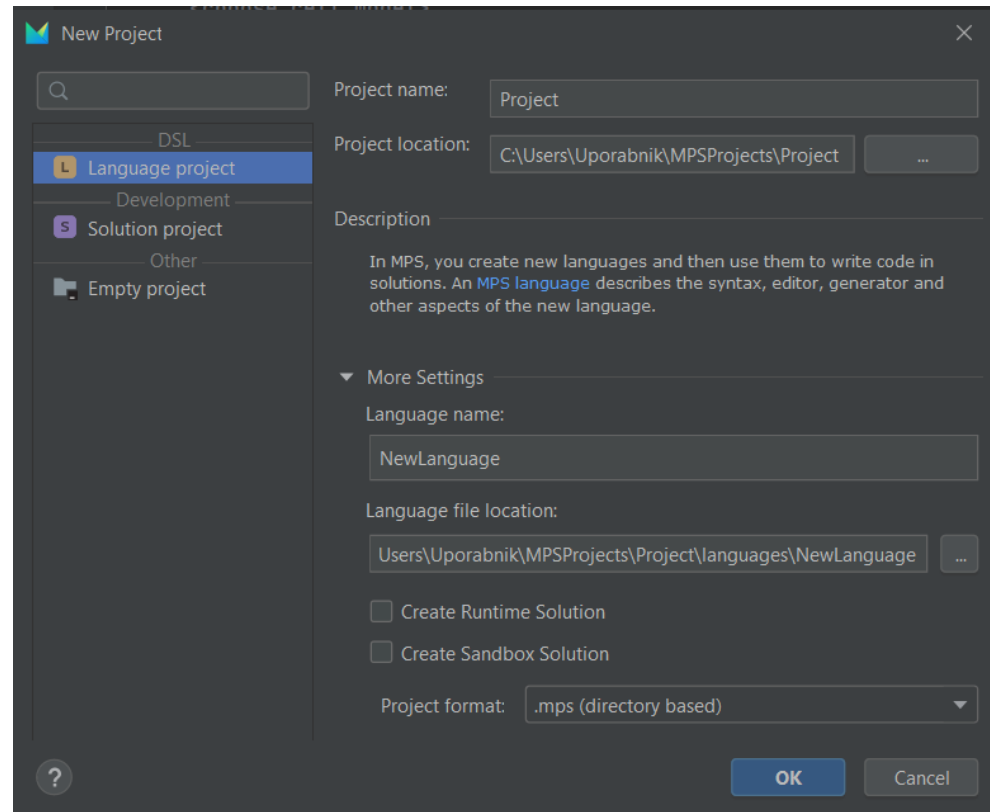


# Uporaba orodja JetBrains MPS



# Orodje JetBrains MPS

- Naloži JetBrains MPS  
<https://www.jetbrains.com/mps/>
- Ustvari nov Project
  - File -> New -> Project



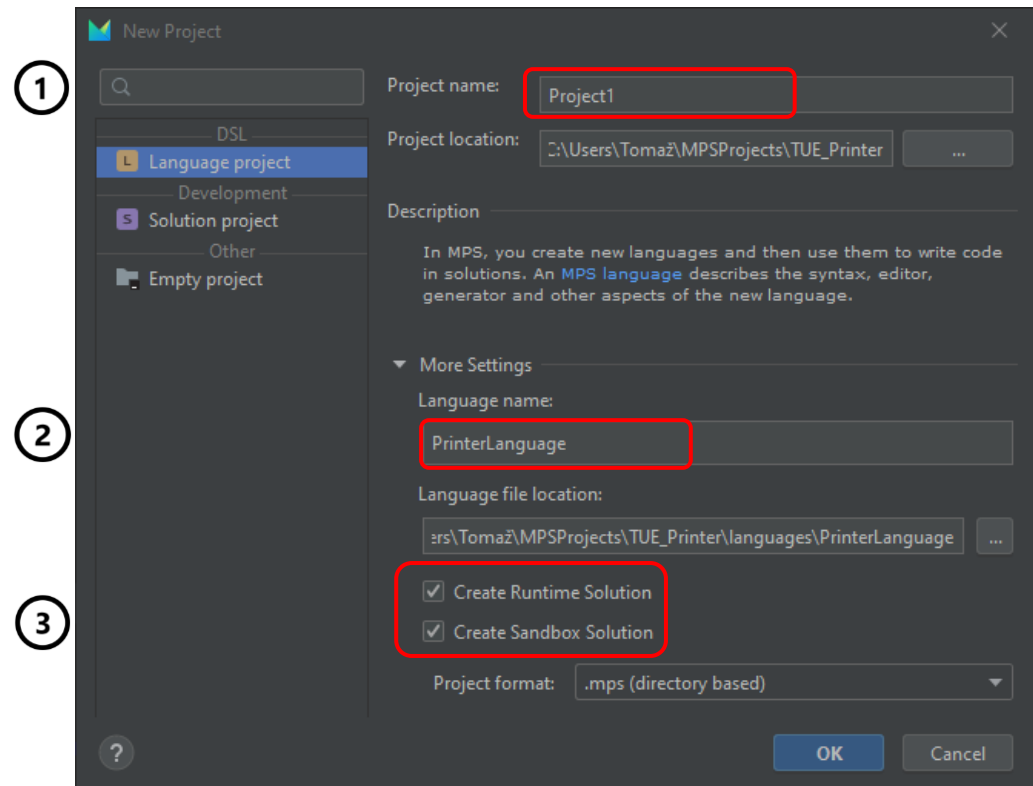


# Projekt za nov jezik

1. Ime projekta

2. Ime jezika

3. Ustvarimo peskovnik

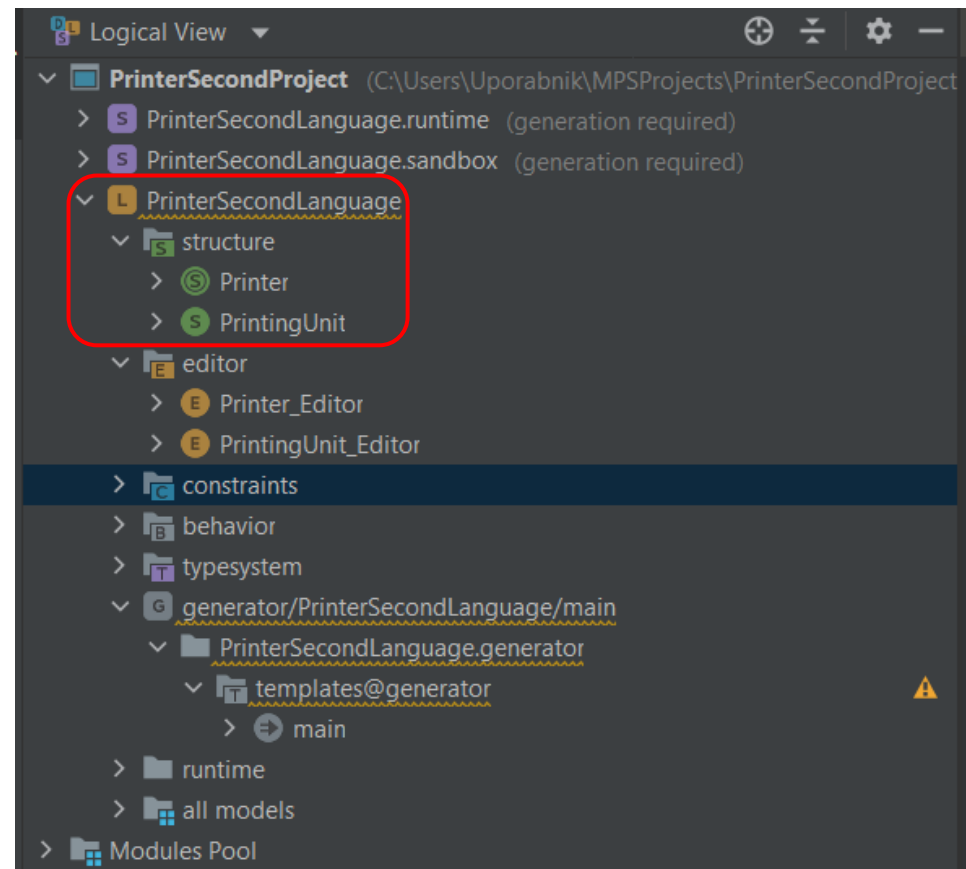






# Struktura jezika

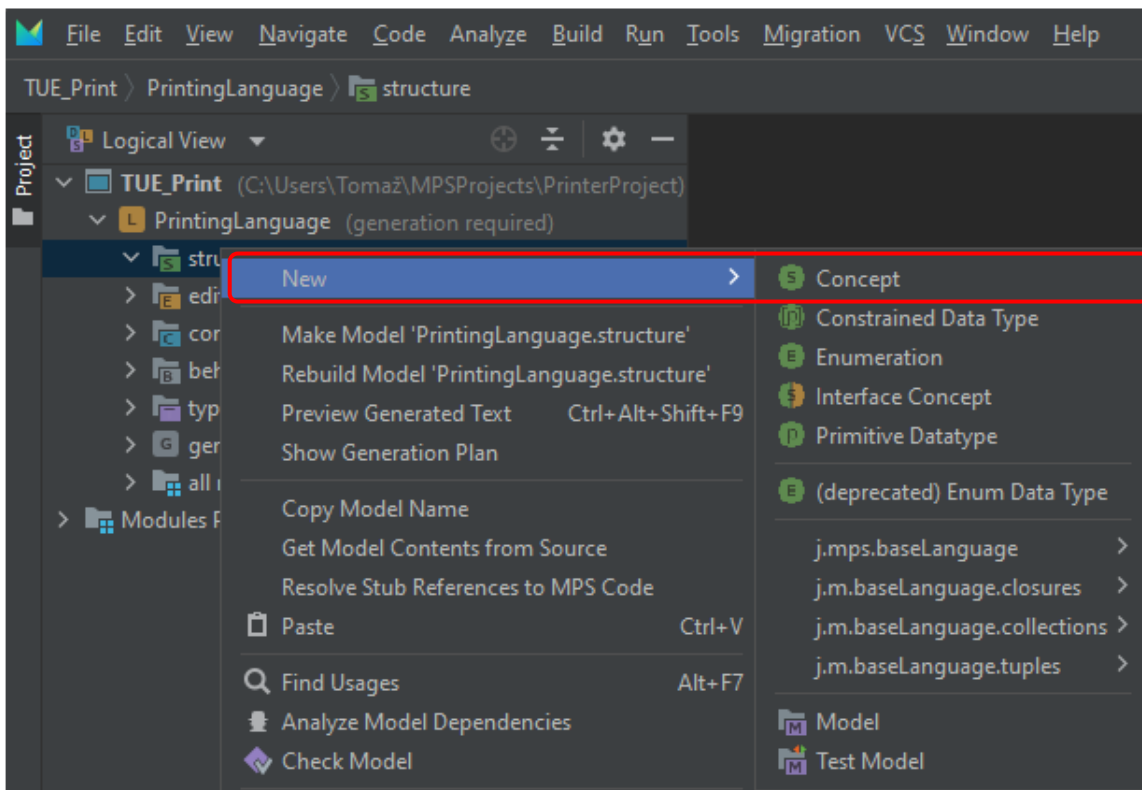
- Vsaka datoteka v mapi „structure“ definira koncepte iz AST.





# Nov koncept

- Ko jo ustvarimo, datoteka že vsebuje osnovno strukturo.



```
concept <no name> extends BaseConcept
    implements <none>

instance can be root: false
alias: <no alias>
short description: <no short description>

properties:
<< ... >>

children:
<< ... >>

references:
<< ... >>
```



# Nov koncept

```
concept Printer extends BaseConcept
                    implements INamedConcept

instance can be root: true
alias: printer
short description: <no short description>

properties:
product : integer

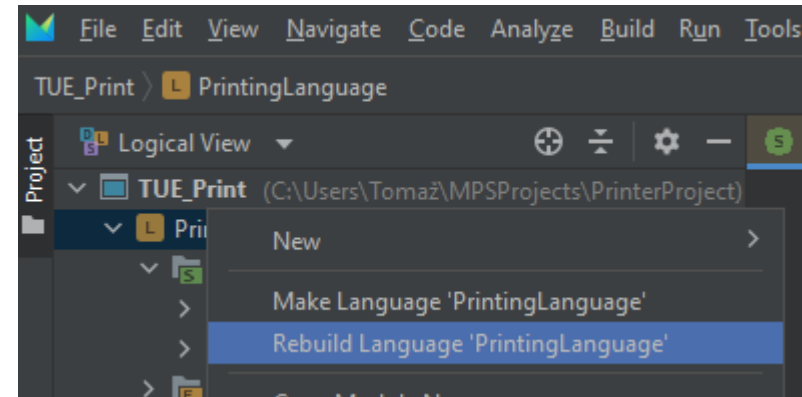
children:
listOfPrintingUnits : PrinterUnit[0..n]

references:
<< ... >>
```



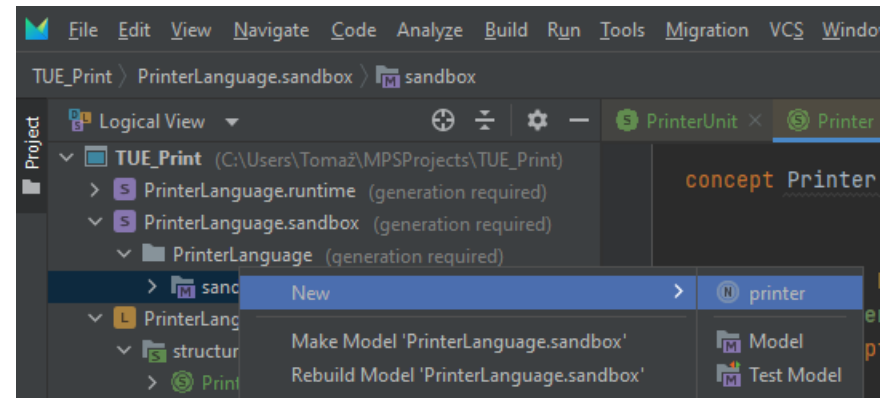
# Testiranje s privzeto projekcijo

## ■ Rebuild language

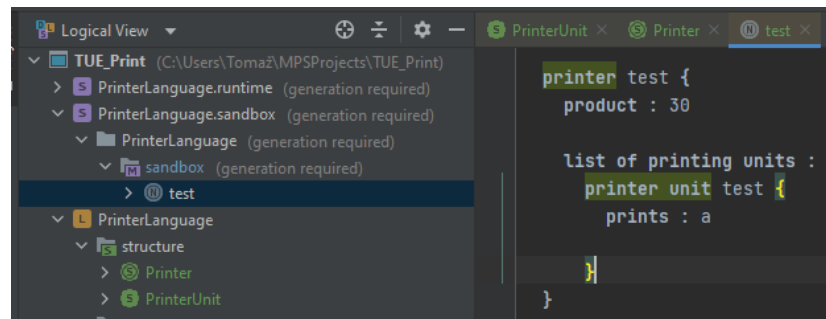


## ■ Sandbox urejevalnik

### ■ Nov program



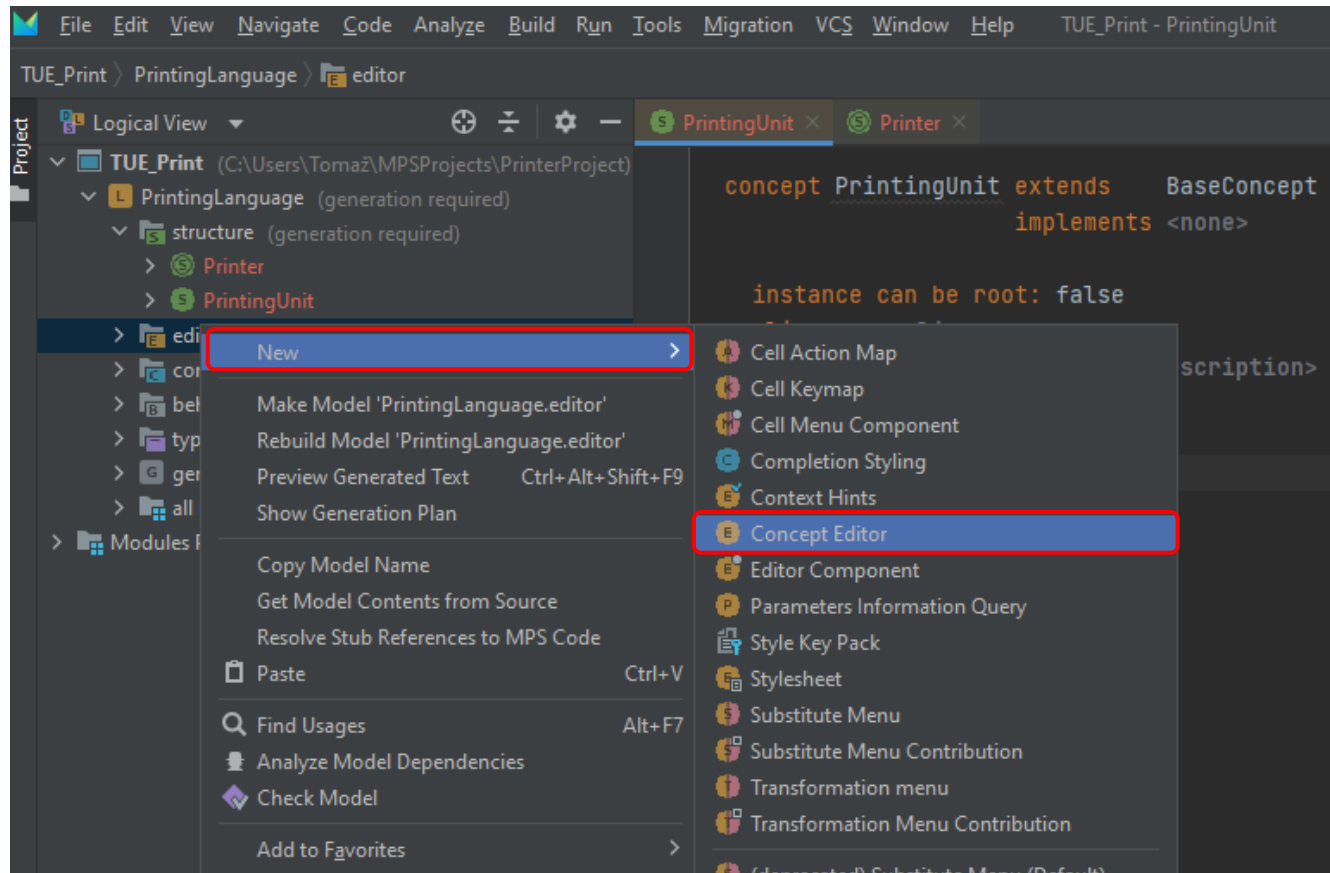
## ■ Privzeti pogled





# Definiranje projekcije

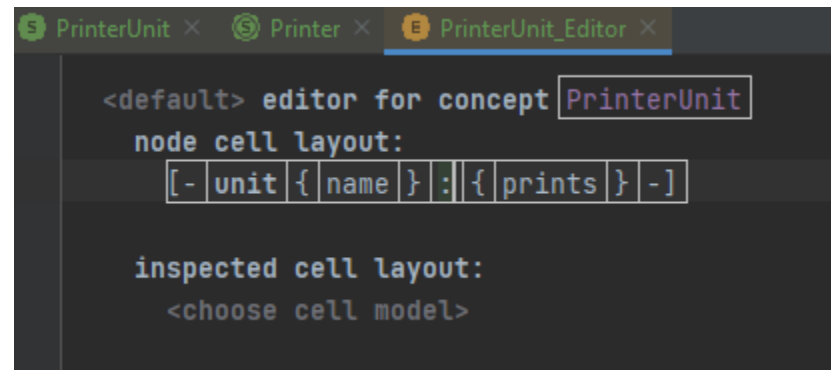
## ■ Mapa Editor



# Definiranje projekcije koncepta

## ■ Kaj vidimo spodaj?

- Razporeditev z znakom: [-
- Dodatna rezervirana beseda (**unit**) in znake ({})
- Lastnosti (**prints**)



```
<default> editor for concept PrinterUnit
node cell layout:
  [- unit { name } : { prints } -]

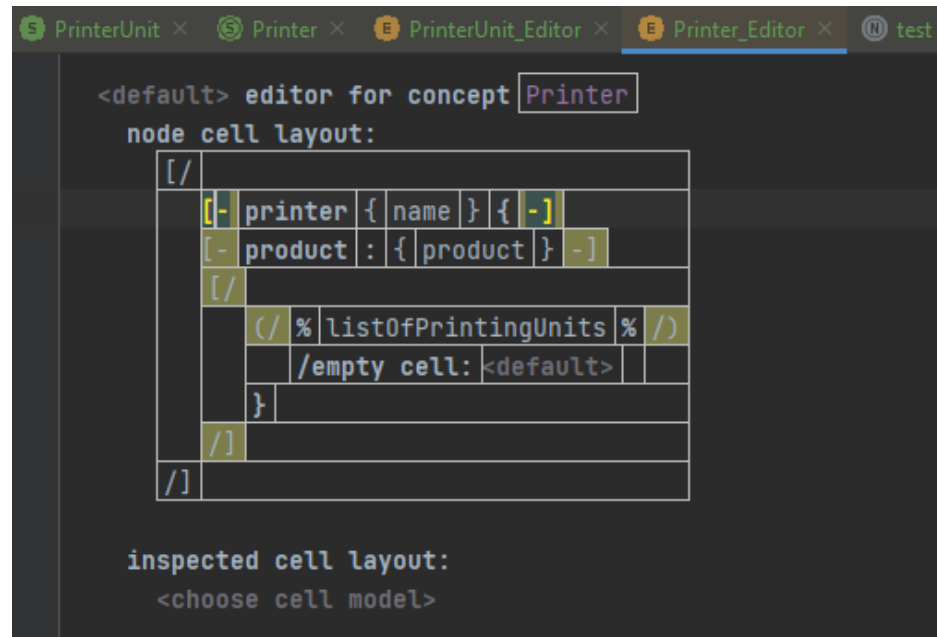
inspected cell layout:
<choose cell model>
```

- S **shift+enter** lahko vstavimo novo pred trenutno celico.

# Definiranje projekcije koncepta

## ■ Kaj vidimo spodaj?

- Vertikalna razporeditev: [/
- Dodatno rezervirane besede (**printer**) in zanke
- Seznam potomcev: (/

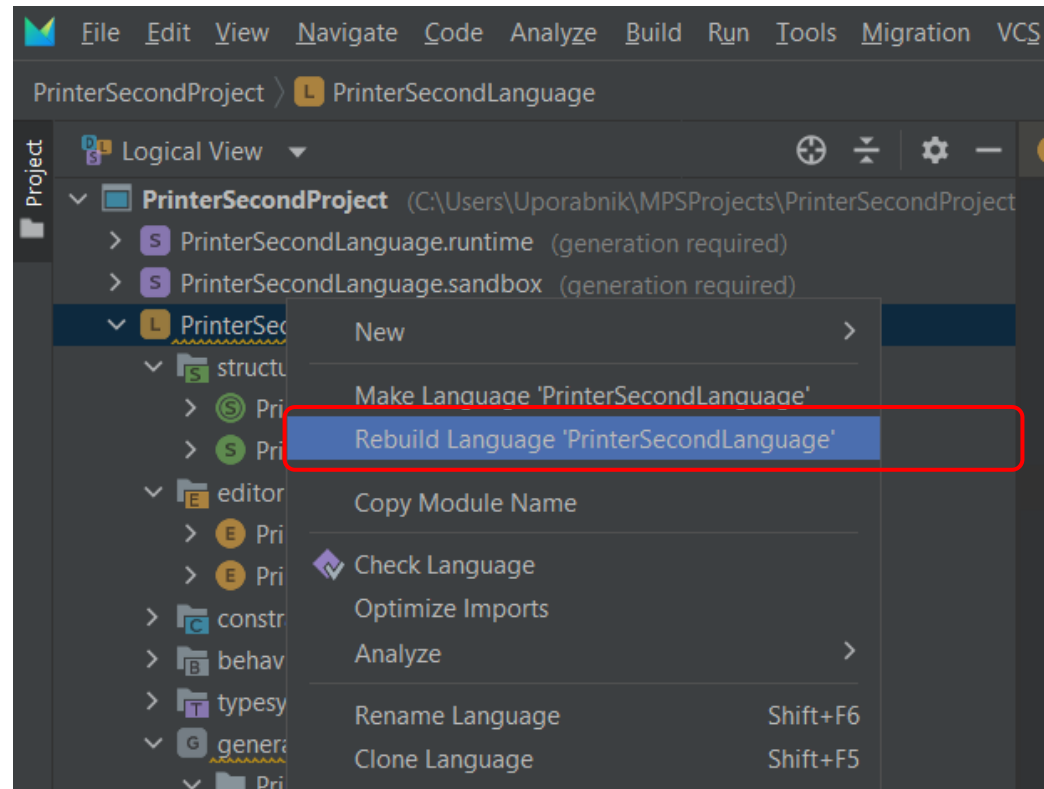


```
PrinterUnit x Printer x PrinterUnit_Editor x Printer_Editor x test x
<default> editor for concept Printer
node cell layout:
[/
  [- printer { name } { -]
  [- product : { product } -]
[/
  (/ % listOfPrintingUnits % /)
  /empty cell: <default>
  }
/]
/]

inspected cell layout:
<choose cell model>
```

# Prevajanje jezika

- Prevajanje (angl. "Compile"):  
Desni klik na mapo jezika
- „Rebuild language ...“
- Če želimo testirati jezik, je potrebno strukturo jezika po vsaki spremembi **ponovno prevesti**.

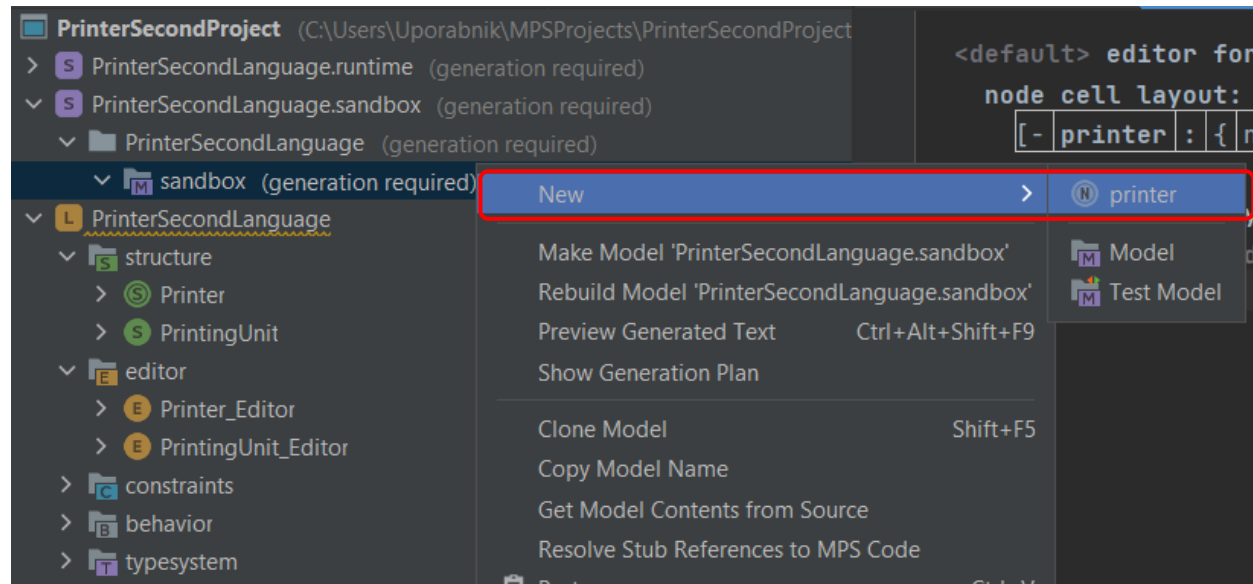






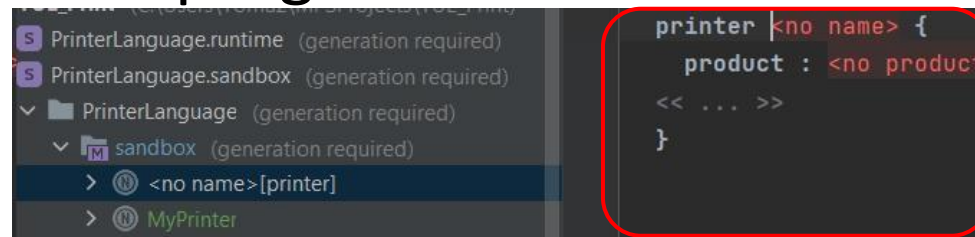
# Testiranje jezika

- V peskovniku (angl. sandbox) odpremo nov testni program.

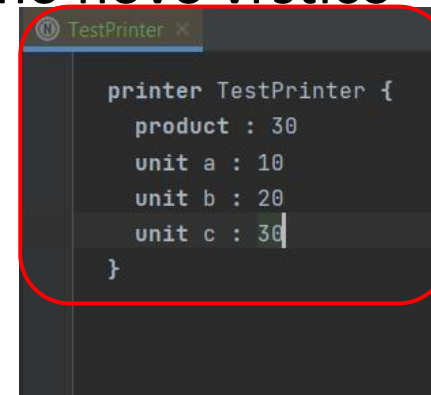


# Testiranje jezika (2)

- Pričnemo s testiranjem našega jezika v „pripravljenem“ programu.



- Urejevalnik nas opozarja (rdeča barva) kje je potrebno vstaviti podatke.
- Z **enter** na koncu vrstice vstavimo novo vrstico
- S **ctrl+space** lahko odpremo seznam opcij, ki so podprte v našem DSLju.





# DEMO CarModelLang



# Primer CarModelLang (1. korak - structure)

Car → name CarBody Color Roof? Wheels

CarBody → Suv | Limo | Minivan

Color → BodyColor RoofColor?

Roof → Rails? sunroof?

Wheel → 195/60R16 | 205/60R16 | 215/60R16

BodyColor → white | black | red

RoofColor → white | black

Rails → silver | black

The screenshot displays a software development environment with two main panels. The left panel, titled 'Logical View', shows a hierarchical tree structure of the project. The 'structure' folder is expanded, revealing a list of components: BodyColor, BodyColorEnum, Car (highlighted with a red box), CarBody, CarBodyEnum, Color, Rails, RailsEnum, Roof, RoofColor, RoofColorEnum, Sunroof, Wheels, and WheelsEnum. The right panel shows a code editor for the 'Car' concept. The code defines the 'Car' concept as extending 'BaseConcept' and implementing 'INamedConcept'. It includes metadata such as 'instance can be root: true', 'alias: <no alias>', and 'short description: <no short description>'. The 'properties' section is empty. The 'children' section lists 'body' (CarBody[1]), 'color' (Color[1]), 'roof' (Roof[0..1]), and 'wheels' (Wheels[1]). The 'references' section is also empty.

```
Logical View
├── CarLanguage
│   └── structure
│       ├── BodyColor
│       ├── BodyColorEnum
│       └── Car
│           ├── CarBody
│           ├── CarBodyEnum
│           ├── Color
│           ├── Rails
│           ├── RailsEnum
│           ├── Roof
│           ├── RoofColor
│           ├── RoofColorEnum
│           ├── Sunroof
│           ├── Wheels
│           └── WheelsEnum
└── editor
    ├── BodyColor_Editor
    ├── Car_Editor
    ├── CarBody_Editor
    ├── Color_Editor
    ├── Rails_Editor
    ├── Roof_Editor
    ├── RoofColor_Editor
    └── Wheels_Editor
```

```
concept Car extends BaseConcept
    implements INamedConcept

    instance can be root: true
    alias: <no alias>
    short description: <no short description>

    properties:
    << ... >>

    children:
    body : CarBody[1]
    color : Color[1]
    roof : Roof[0..1]
    wheels : Wheels[1]

    references:
    << ... >>
```



# Primer CarModel (2. korak – editor)

CarModelLang → 'car' name 'carBody' CarBody Color Roof? 'wheels' Wheels

CarBody → 'suv' | 'Limo' | 'Minivan'

Color → 'color' BodyColor ('roof' RoofColor)?

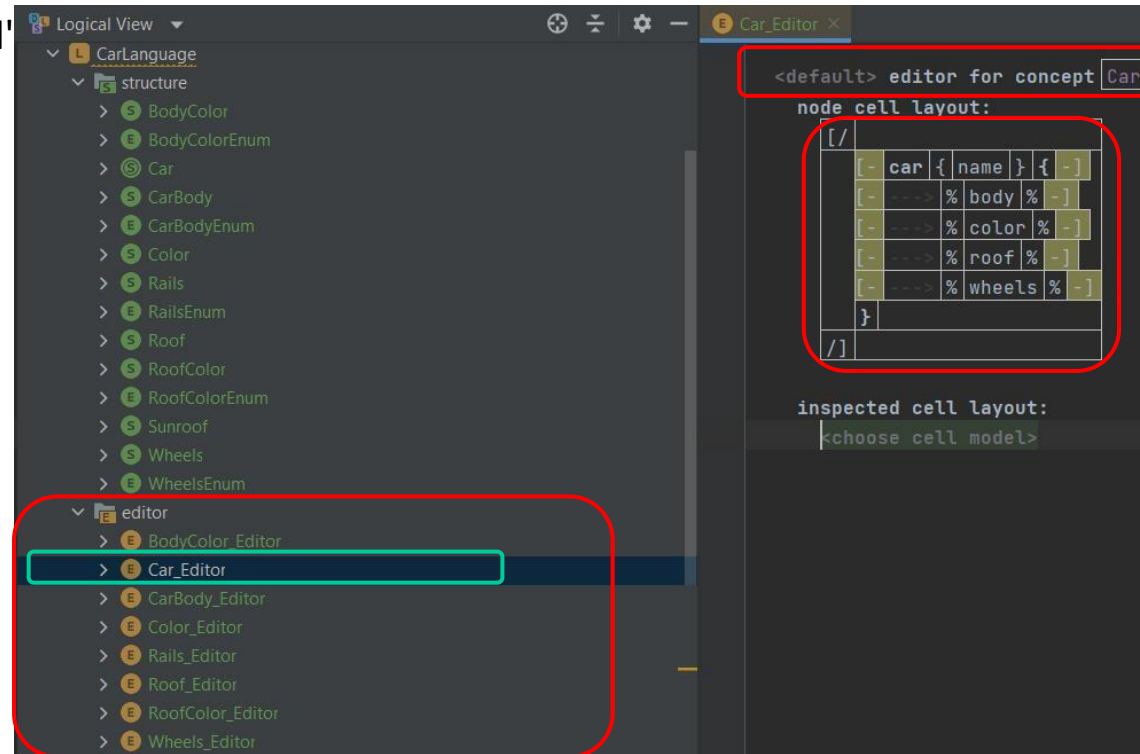
Roof → ('roof' 'rails' Rails)? 'sunroof'?

Wheels → '195/60R16' | '205/60R16' | '215/60R16'

BodyColor → 'white' | 'black' | 'red'

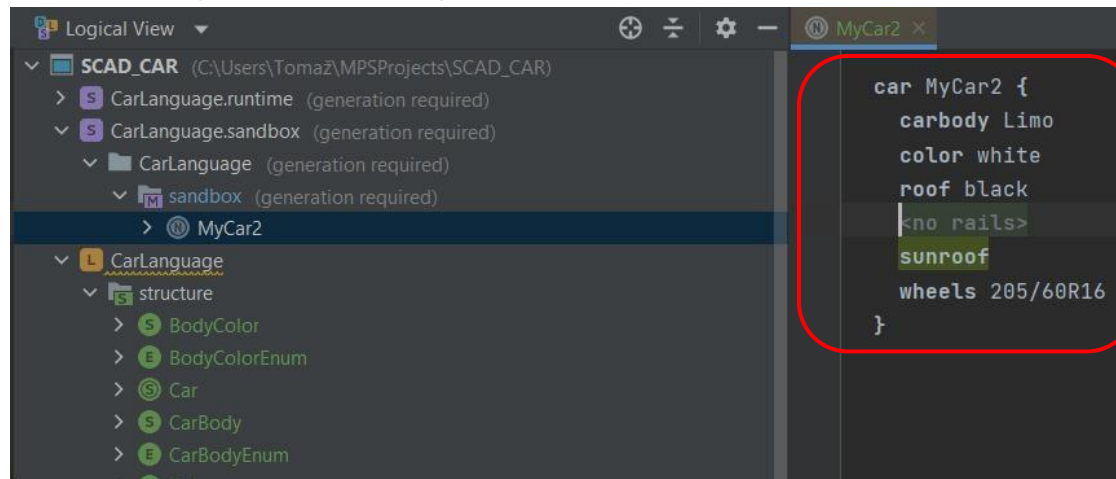
RoofColor → 'white' | 'black'

Rails → 'silver' | 'black'



# Primer CarModel: testiranje

## ■ Testiranje sintakse jezika



# Odprta vprašanja

- Generacija kode
- Syntax highlighting
- Validacija
- Kako do nove predstavitev





