LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

**WEBAPPOS .NET IZSAUKUMU ADAPTERIS UN C# STARPKODA ĢENERATORS**

KVALIFIKĀCIJAS DARBS

Autors:

Artis Pauniņš ap19122

Darba vadītājs: Sergejs Kozlovičs

RĪGA 2020

# ANOTĀCIJA

# ABSTRACT

# SATURA RĀDĪTĀJS

# 

# IEVADS

**Nolūks**

Kvalifikācijas darba nolūks ir aprakstīt programmatūras prasības izstrādājamajam adapterim un starpkoda ģeneratoram. Dokumentā tiks detalizēti aprakstītas sistēmas funkcijas un ierobežojumi.

Dokuments paredzēts visām ieinteresētajām pusēm, tostarp izstrādātājiem.

**Darbības sfēra**

WebAppOS operētājsistēmas izsaukumu adapteri ir paredzēti, lai tie spētu izsaukt operētājsistēmas funkcijas. Adapterus veido atsevišķi katrai programmēšanas valodai, tādējādi šie adapteri dos iespēju operētājsistēmas izstrādātājiem izstrādāt jaunas un esošās funkcijas vairākās programmēšanas valodās.

**Saistība ar citiem dokumentiem**

Dokumenta noformējumā ievērotas standarta LVS 68:1996 “Programmatūras prasību specifikācijas ceļvedis” vadlīnijas.

**Pārskats**

# APZĪMĒJUMU SARAKSTS

WebAppOS – operētājsistēma kura ir veidota palaišanai tīmeklī.

Atvērts kods jeb atvērta programmatūra – programmatūra, kas tiek izlaista saskaņā ar licenci, kurā autortiesību īpašnieks lietotājiem piešķir tiesības izmantot, pētīt, mainīt un izplatīt programmatūru un tās pirmkodu jebkuram nolūkam.

Konteksts – dati, kas glabā informāciju par pirmkoda sastāvdaļu.

# 1. VISPĀRĪGS APRAKSTS

## 1.1 Esošā stāvokļa apraksts

Kā jebkurai programmai un operētājsistēmai, arī WebAppOS ir savas funkcijas. Tā kā WebAppOS ir izstrādāts valodā JAVA, tad arī funkcijas ir rakstītas valodā JAVA. Tas ierobežo iespējas izstrādāt operētājsistēmas funkcijas citās programmēšanas valodās.

WebAppOS ir arī funkcijas, kuras ir rakstītas valodā lua. Lai operētājsistēma varētu palaist šīs funkcijas, tad tīmekļa procesorā ir adapteris, kurš ir veidots kādai konkrētai programmēšanas valodai un šie adapteri spēj atrast un izsaukt funkcijas konkrētā programmēšanas valodā. Izņemot JAVA un lua valodas, nav pieejami adapterī citām programmēšanas valodām, piemēram, C# vai Python.

Uzdevums ir izstrādāt tīmekļa procesoram adapteri, kas ļautu operētājsistēmai izsaukt funkcijas C# programmēšanas valodā.

## 1.2 Pasūtītājs

Sistēmas pasūtītājs ir Latvijas Universitātes Matemātikas un informātikas institūts.

## 1.3 Produkta perspektīva

WebAppOS apraksts

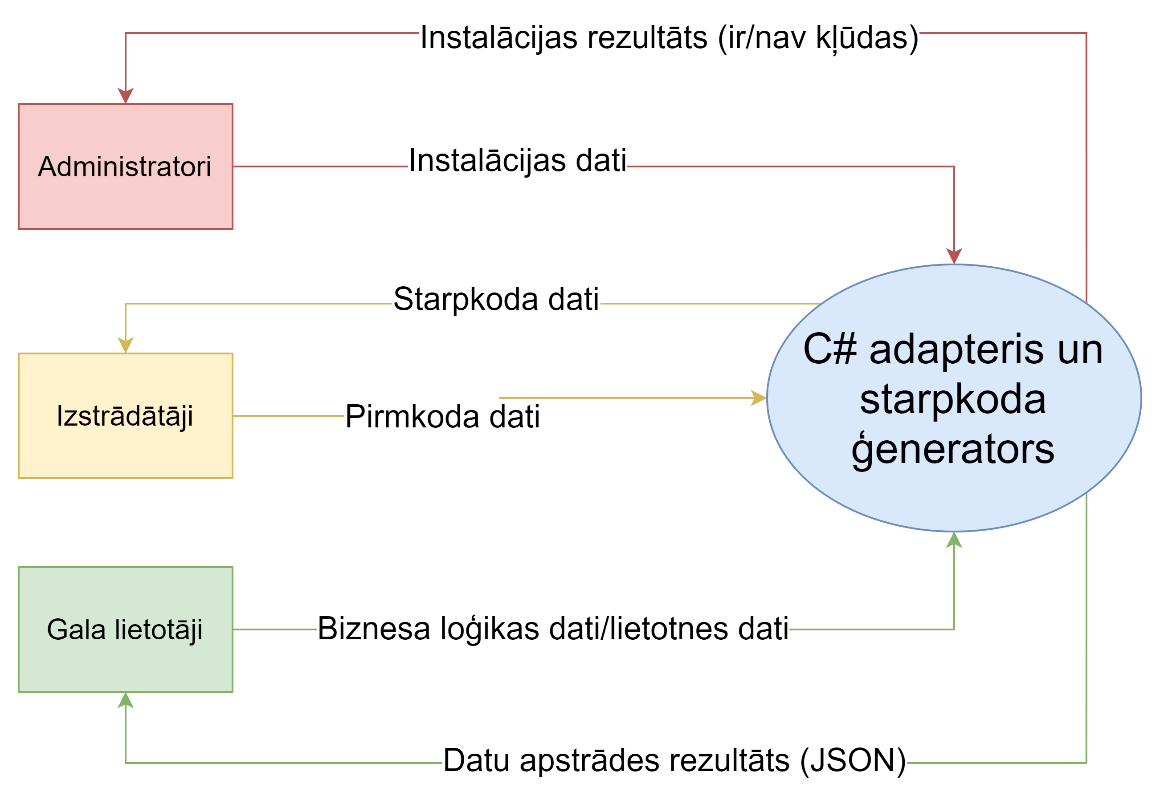
* Galvenās funkcijas
  + WebAppOS nodrošina iespēju izsaukt kodu, kas ir rakstīts dažādās programmēšanas valodās un kas atrodas dažādos tīkla mezglos
  + WebAppOS nodrošina tīmekļa atmiņas sinhronizāciju starp klientu un serveri.
  + WebAppOS ļauj veidot tīmekļa lietotnes tādā pašā veidā, ka veido darbvirsmas lietotnes, nedomājot par tīklu.
* Saskarnes
  + RAAPI – ļauj piekļūt tīmekļa atmiņai {https://webappos.org/dev/raapi/}
  + IWebCalls API – ļauj izsaukt funkcijas, kuras rakstītas citās programmēsanas valodās vai kas atrodas dažādos tīkla mezglos
    - Local – virzienā no WebAppOS uz .NET
    - Remote – virzienā no .NET uz WebAppOS
* Nepieciešams tīkla savienojums ar WebAppOS serveri un vēlamā operētājsistēma ir Linux Ubuntu

Adapteris un kodu ģenerators tiks iekļauti WebAppOS distribūcijā.

## 1.4 Sistēmas lietotāji

Sistēmu izmantos trīs lietotāju grupas:

* Administratori – administratoru uzdevums būs WebAppOS uzstādīšana.
* Izstrādātāji – izstrādātāji veidos programmatūru WebAppOS videi izmantojot izstrādāto adapteri un kodu ģeneratoru
* Gala lietotāji – gala lietotāji izmantos WebAppOS biznesa vajadzībām.



*1.1. att.* **0. līmeņa DPD diagramma**

## 1.5 Darījumprasības

* Dinamiski palaist funkcijas
* LINQ atbalsts
* Programmkodu ģenerēšana
* Vārdtelpu (namespace) atbalsts kodu ģenerēšanai
  + Vārdtelpas nosaukums sākas ar burtu vai apakš svītru un sastāv no burtiem, cipariem un apakš svītrām
* Rezervētie vārdi
  + class – klašu definēšanai
  + association – asociāciju definēšanai
  + Integer – veselo skaitļu definēšanai
  + String – simbolu virkņu definēšanai
  + Boolean – Būla vērtību glabāšanai
  + Real – reālo skaitļu glabāšanai
  + URL – url definēšanai
  + private – privāto lauku glabāšanai
  + public – publisko lauku glabāšanai
  + BaseObject – klase “BaseObject”, kas ir virsklase visām definētajām klasēm
* Klašu kļūdu pārbaude
  + Vai eksistē virsklase, no kuras grib mantot
  + Virsklase nevar būt pamatklase
  + Katrai klasei ir viena virsklase
  + Visām klasēm ir unikāli vārdi
  + Klašu definīcijai izmanto atslēgvārdu “class”
  + Klašu nosaukumi nevar būt rezervētie vārdi
* Lauku (mainīgo un funkciju) kļūdu pārbaude
  + Laukam ir datu tips “Integer”, “String”, “Boolean” vai “Real”
  + Lauku vārdi ir unikāli, kā arī nedrīkst sakrist ar klases vārdu un asociāciju lomas vārdiem, bet drīkst sakrist ar virsklases vārdu.
  + Lauku vārdi sākas ar burtu vai apakš svītru un sastāv no burtiem, cipariem un apakš svītrām
  + Lauku vārdi nevar būt rezervētie vārdi
  + Lauki ir atdalīti ar semikolu
  + Funkciju argumentu vārdi drīkst sakrist ar lauku vārdiem, kā arī ar klases vārdu
  + Funkciju arguments sastāv no datu tipa un vārda, un visi argumenti ir atdalīti ar komatu
  + Funkciju argumentu vārdi savā starpā ir unikāli
* Kļūdu pārbaude starp klases un tās virsklašu laukiem
  + Ja apakšklasē un virsklasē ir funkcija ar vienu un to pašu vārdu, tad to drīkst pieļaut, ja funkcijām ir identiski datu tipi un argumenti. URL drīkst būt dažādi
* Asociāciju kļūdu pārbaude
  + Asociācijā starp divām klasēm ir jābūt dažādiem lomu vārdiem skatoties gan no avota klases, gan no mērķa klases, ņemot vērā arī abu klašu virsklases un apakšklases.
  + Avotklasē asociācijas mērķa lomas vārds nedrīkst sakrist ar lauku vārdiem un avotklases vārdu, bet drīkst sakrist ar virsklases vārdu.
  + Mērķklasē asociācijas avota lomas vārds nedrīkst sakrist ar lauku vārdiem un mērķklases vārdu, bet drīkst sakrist ar virsklases vārdu.
  + Associācijas lomu vārdi nedrīkst būt rezervētie vārdi
  + Asociācijas lomu vārdi sākas ar burtu vai apakš svītru un sastāv no burtiem, cipariem un apakš svītrām.
* Ja ģenerējamajā pirmkodā tiek konstatēta kaut viena kļūda, tad kods netiek ģenerēts.

## 1.6 Vispārējie ierobežojumi

* Dažādi OS (Windows 10, Windows 11, Linux Ubuntu, MacOS 10, MacOS 11 )
* Dažādu procesoru arhitektūras (AMD64, ARM64)
* Java 11 (WebAppOS) un .NET 5.0 (Adapterim un kodu ģeneratoram)

## 1.7 Pieņēmumi un atkarības

* Izmanto trešo pušu kodu ģeneratorus ar atvērtu pirmkoda licenci
* Uzģenerētajam kodam jādarbojas iepriekš minētajās operētājsistēmās
* Izstrādātājam jāspēj iemācīties pašveidotā programmēšanas valoda
* Izstrādātāji var palaist programmas, izmantojot komandrindu
* Kods pašveidotajā programmēšanas valodā ir sintaktiski pareizs, piemēram, funkcijām nedrīkst likt divus vārdus, bet drīkst pieļaut gadījumus, kad kaut kas nav ievadīts, piemēram, funkcijai nav norādīts ne vārds, ne datu tips. Tādā gadījumā kompilatoram ir jāfiksē to kā kļūdu.

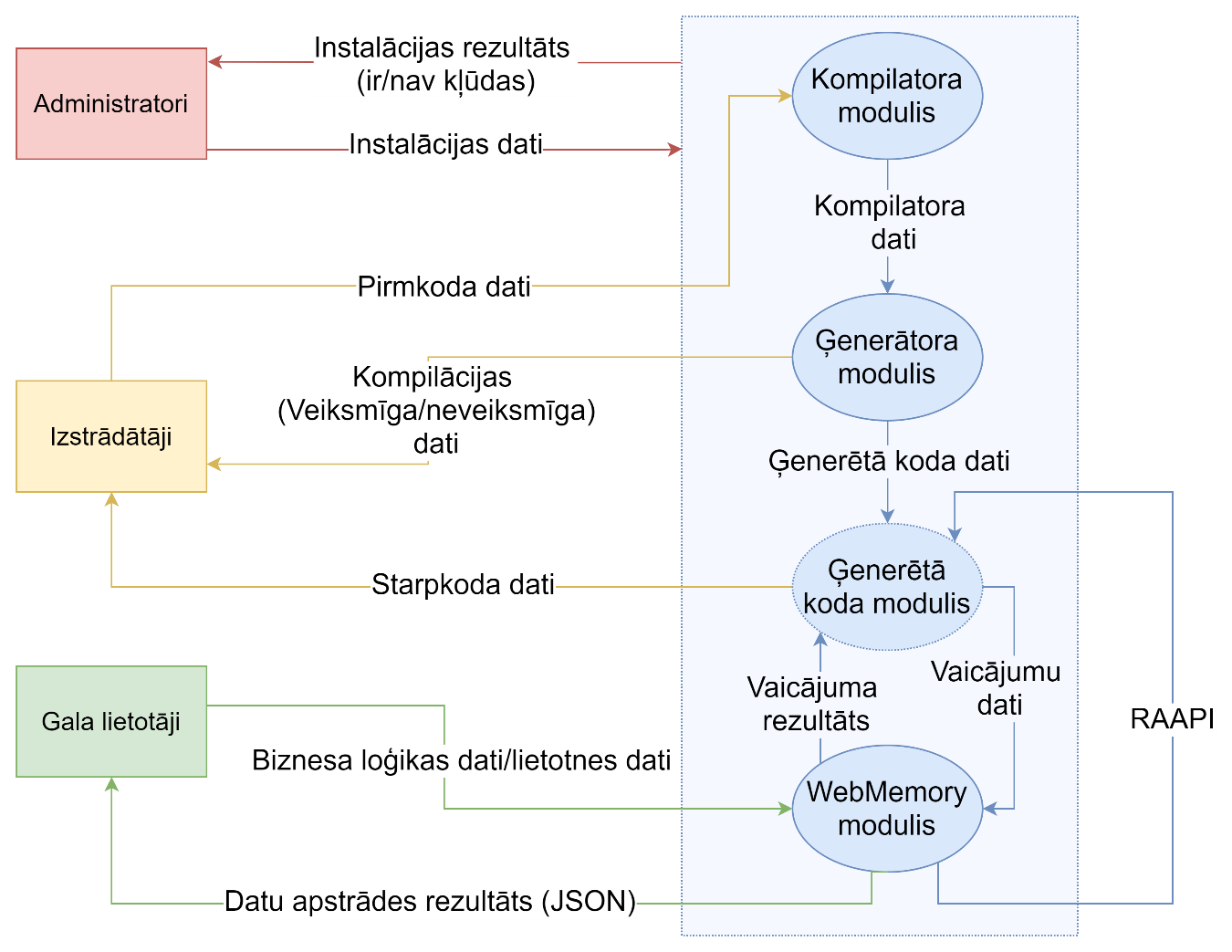
# 2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

## 2.2 Funkcionālās prasības

### 2.2.1 Funkciju sadalījums pa moduļiem/komponentiem

Sistēma sastāv no 4 moduļiem:

* WebMemory modulis – apstrādā klases, objektus, atribūtus un asociācijas, kuras glabājas šajā modulī.
* Ģenerētā koda modulis – uzģenerētā pakotne ar C# klasēm, kas savieno C# kodu ar WebMemory moduli. Tajā būs klases, kuras atbilst datiem, kas atrodas WebMemory modulī.
* Kompilatora modulis – atbild par pirmkoda apstrādi jeb kļūdu atrašanu pirmkodā un pirmkoda datu iegūšanu (klases un to īpašības, metodes, asociācijas).
* Ģeneratora modulis – atbild par pirmkoda ģenerēšanu C# programmēšanas valodā, izmantojot datus, kas iegūti no kompilatora moduļa.

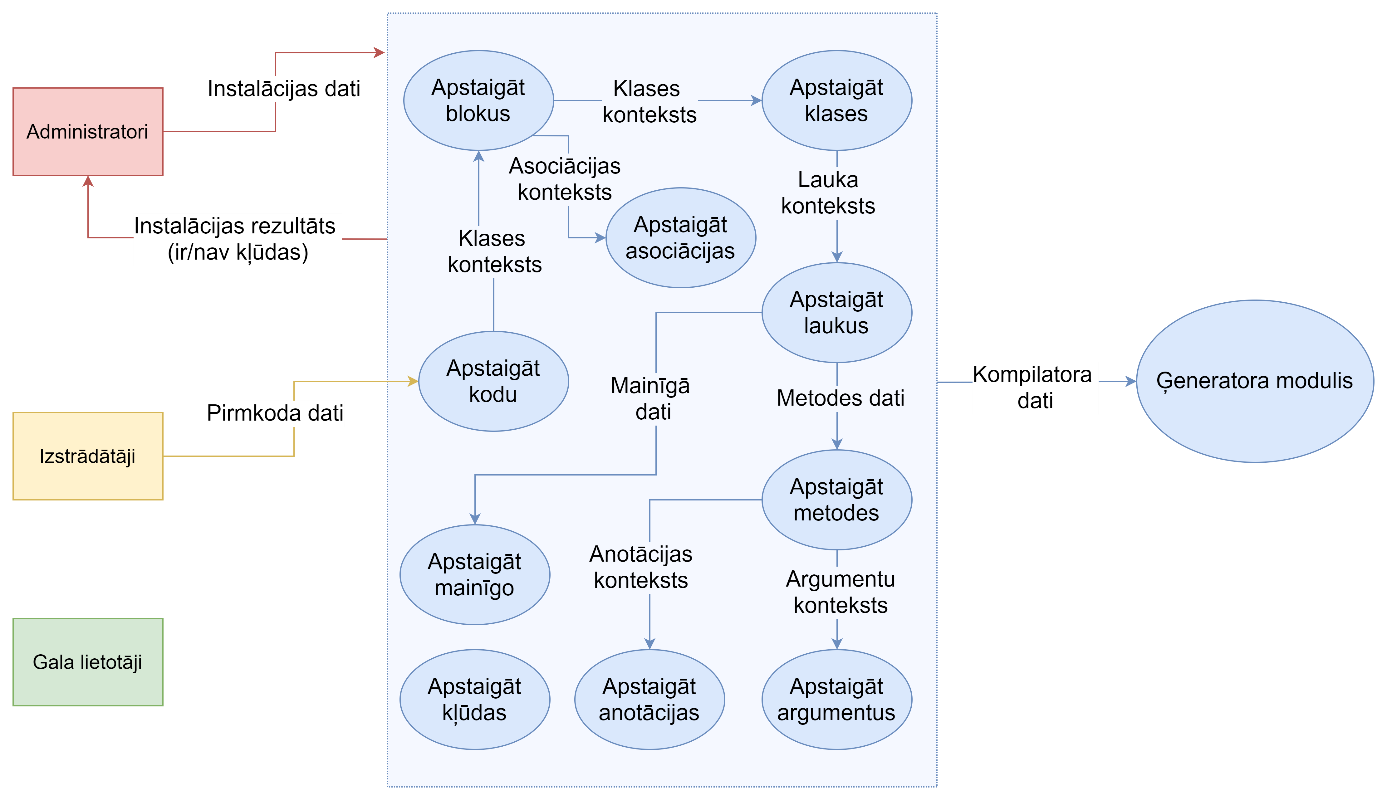
*2.1. att.* **1. līmeņa DPD diagramma**

### 2.2.2 Kompilatora modulis

Šo moduli izmantos tikai izstrādātāju grupa, jo tiek padots pirmkods, kurš tiek kompilēts. Uzdevums ir atrast kļūdas programmkodā. Visas kļūdas tiek saglabātas formātā

* At line ‘rindas numurs’: ‘kļūdas paziņojums’.

Pēc koda kompilēšanas, visi dati par klasēm un asociācijām tiek padoti ģeneratora modulim un tas veic koda ģenerēšanu, ja nav atrasta neviena kļūda. Visas funkcijas ir sava starpā saistītas, jo koda dati tiek glabāti kokā (skat. ? nodaļu). Visam funkcijām kā ievaddati tiek padoti konteksti.

* 2.2. att.* **2. līmeņa DPD diagramma kompilatora modulim**

#### 2.2.2.1 Apstaigāt kodu

**Nolūks**

Funkcija nepieciešama, lai uzsāktu koda kompilēšanu.

**Ievade**

Funkcijai ir šādi ievaddati:

* Koda konteksts jeb pirmkoda fails;

**Apstrāde**

* Netiek veiktas nekādas pārbaudes un papildus apstrādes

**Izvade**

* Tiek izveidoti sekojoši saraksti
  1. Klašu saraksts
  2. Asociāciju saraksts
  3. Kļūdu saraksts
* Katram koda blokam tiek izsaukta funkcija “Apstaigāt bloku”.

#### 2.2.2.2 Apstaigāt blokus

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu koda blokus.

**Ievade**

Funkcijai ir šādi ievaddati:

* Bloka konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai blokam ir ievadīts tips?
  + Vai blokam ir pareizs tips (klase vai asociācija)?
* Vai blokam ir ievadīta definīcija?
  + Kāds bloks tiek definēts (klases vai asociācijas)?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja blokam ir padots tips, tad tas tiek pārbaudīts.
  1. Ja bloka tips nav klase vai asociācija, tad tiek saglabāts kļūdas paziņojums “’padotais tips’ nav bloka tips. Izmantojiet klasi vai asociāciju”.
  2. Ja bloka tips nav padots, tad saglabājas kļūdas paziņojums “Trūkst atslēgvārda ‘class’” vai “Trūkst atslēgvārda ‘association’” atkarībā no tā, vai tiek definēta klase vai asociācija.
* Ja blokam ir definīcija tad pārbauda, vai blokam ir tips
  1. Ja tipa nav, tad nekas tālāk nenotiek
  2. Ja tips ir, tad skatās, kāda ir tipa un definīcijas kombinācija
     + Klases tips un klases definīcija – izsaucam funkciju “Apstaigāt klasi”
     + Asociācijas tips un asociācijas definīcija – izsaucam funkciju “Apstaigāt asociāciju”
     + Klases tips un asociācijas definīcija – saglabāts kļūdas paziņojums “Klasei ir dota asociācijas definīcija”.
     + Asociācijas tips un klases definīcija – saglabāts kļūdas paziņojums “Asociācijai ir dota klases definīcija”

#### 2.2.2.3 Apstaigāt asociāciju

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu asociācijas definīciju.

**Ievade**

Funkcijai ir šādi ievaddati:

* Asociācijas konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai asociācijai ir doti dati par avotu?
  + Vai avotam ir pareizs lomas vārds?
  + Vai avotam ir pareizs klases vārds?
* Vai asociācijai ir doti dati par mērķi?
  + Vai mērķim ir pareizs lomas vārds?
  + Vai mērķim ir pareizs klases vārds?
* Vai asociācijai ir bultas?

Vispirms ir jānoskaidro, vai asociācijai ir visi nepieciešamie dati: klases un lomu vārdi gan avotam, gan mērķim. Ja iztrūkst kaut viens no šiem datiem, tad dati par asociāciju netiks saglabāti.

**Izvade**

Izvade ir atkarīga no apstrādes:

* Klašu apstrāde:
  1. Ja klases vārds nav dots, tad saglabā kļūdas paziņojumu “trūkst klases”.
  2. Ja klase vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “klasi nevar saukt par ‘rezervētais vārds’”.
  3. Ja klase ar doto vārdu neeksistē, tad saglabā kļūdas paziņojumu “nav klases ‘klases vārds’, kuru izmantot kā avota/mērķa klasi”.
  4. Ja kļūdu nav, tad klasi saglabā.
* Lomas vārdu apstrāde:
  1. Ja lomas vārds nav dots, tad saglabā kļūdas paziņojumu “trūkst lomas vārda”.
  2. Ja lomas vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “lomas vārds nevar būt ‘rezervētais vārds’”.
  3. Ja avota/mērķa lomas vārds sakrīt ar kādu no mērķa/avota klases vai virsklases mainīgo, funkciju vai citu avota/mērķu vārdiem, tad saglabā kļūdas paziņojumu “klasē jau eksistē mainīgais/funkcija/asociācijas galapunkts ar nosaukumu ‘lomas vārds’”, norādot rindu, kur padotais vārds jau tiek izmantots.
  4. Ja kļūdu nav, tad lomas vārdu saglabā.
* Ja asociācijas definīcijā nav bultas, tad saglabā kļūdas paziņojumu “Trūkst bultu”.
* Ja avota vai mērķa definīcijā starp lomas vārdu un klases vārdu nav kola, tad saglabā kļūdas paziņojumu “Trūkst kola”.
* Ja nav sastapta neviena kļūda, tad asociācijas dati tiek saglabāti.

#### 2.2.2.4 Apstaigāt klasi

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu klases definīciju.

**Ievade**

Funkcijai ir šādi ievaddati:

* Klases konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai klasei ir dots pareizs klases vārds?
* Vai klasei ir dots pareizs virsklases vārds?
* Vai klasei ir dots ķermenis?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Klases vārda apstrāde:
  1. Ja klases vārds nav dots, tad saglabā kļūdas paziņojumu “trūkst klases vārda”.
  2. Ja klases vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “klasi nevar saukt par ‘rezervētais vārds’”.
  3. Ja klase ar doto vārdu eksistē, tad saglabā kļūdas paziņojumu “Klase ar vārdu ‘klases vārds’ jau eksistē”, norādot rindu, kur padotais vārds jau tiek izmantots.
  4. Ja kļūdu nav, tad klases vārdu saglabā.
* Virsklases apstrāde:
  1. Ja virsklases vārds nav dots, bet ir dots kols, tad saglabā kļūdas paziņojumu “trūkst virsklases vārda”.
  2. Ja virsklases vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “Virsklasi nevar saukt par ‘rezervētais vārds’”.
  3. Ja virsklase ar doto vārdu neeksistē, tad saglabā kļūdas paziņojumu “Klase ar vārdu ‘klases vārds’ neeksistē”, norādot rindu, kur padotais vārds jau tiek izmantots.
  4. Ja virsklases vārds sakrīt ar pamatklases vārdu, tad saglabā kļūdas paziņojumu “Nevar mantot no klases ar tādu pašu vārdu”.
  5. Ja kļūdu nav, tad klases vārdu saglabā.
* Ja klasei nav definīcijas, tad saglabā kļūdas paziņojumu “Trūkst klases ķermeņa”.
* Ja nav sastapta neviena kļūda, tad asociācijas dati tiek saglabāti.
* Ja klasei ir ķermenis, tad izsauc funkciju “Apstaigāt laukus”.

#### 2.2.2.5 Apstaigāt laukus

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu klases laukus.

**Ievade**

Funkcijai ir šādi ievaddati:

* Lauku konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai laukam ir semikols?
* Vai lauks ir mainīgais vai metode?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja lauks nebeidzas ar semikolu, tad saglabā kļūdas paziņojumu “Trūkst semikola”.
* Ja laukam ir definēta anotācija un/vai argumenti, tad lauks ir metode un tiek izsaukta funkcija “Apstaigāt metodi”. Citādi lauks tiek uzskatīts par mainīgo un tiek izsaukta funkcija “Apstaigāt mainīgo”.

#### 2.2.2.6 Apstaigāt mainīgos

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu klases mainīgos.

**Ievade**

Funkcijai ir šādi ievaddati:

* Lauka konteksts (pirms tam noskaidrots, ka tajā nav anotācijas konteksta un nav definēti argumenti);

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai mainīgajam ir aizsardzība?
* Vai mainīgajam ir pareizs datu tips?
* Vai mainīgajam ir pareizs vārds?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja mainīgajam ir aizsardzība, tad tā tiek saglabāta?
* Datu tipa apstrāde:
  1. Ja datu tips nav dots, tad saglabā kļūdas paziņojumu “mainīgajam trūkst datu tipa”.
  2. Ja nav dots pareizs datu tips, tad saglabā kļūdas paziņojumu “’padotais datu tips’ nav derīgs datu tips.”
  3. Ja nav kļūdu, tad datu tipu saglabā.
* Vārda apstrāde
  1. Ja vārds nav dots, tad saglabā kļūdas paziņojumu “mainīgajam trūkst vārda”.
  2. Ja vārds sakrīt ar klases vārdu, tad saglabā kļūdas paziņojumu “’padotais vārds’ nevar būt klases vārds”.
  3. Ja vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “mainīgo nevar saukt par ‘rezervētais vārds’”.
  4. Ja vārds jau tiek izmantots citā laukā vai kā asociācijas galapunkta lomas vārds, tad saglabā kļūdas paziņojumu “klasē jau eksistē lauks/asociācijas galapunkts ar vārdu ‘padotais vārds’”, norādot rindu, kur padotais vārds jau tiek izmantots.
  5. Ja nav kļūdu, tad vārdu saglabā.

#### 2.2.2.7 Apstaigāt metodes

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu klases metodes.

**Ievade**

Funkcijai ir šādi ievaddati:

* Lauku konteksts (pirms tam noskaidrots, ka tajā ir anotācijas konteksts un/vai ir definēti argumenti);

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai metodei ir aizsardzība?
* Vai metodei ir pareizs datu tips?
* Vai metodei ir pareizs vārds?
* Vai metodei ir definētas anotācijas?
* Vai metodei ir definēti argumenti?
* Metodes definīcija virsklasē

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja mainīgajam ir aizsardzība, tad tā tiek saglabāta?
* Datu tipa apstrāde:
  1. Ja datu tips nav dots, tad saglabā kļūdas paziņojumu “mainīgajam trūkst datu tipa”.
  2. Ja nav dots pareizs datu tips, tad saglabā kļūdas paziņojumu “’padotais datu tips’ nav derīgs datu tips.”
  3. Ja nav kļūdu, tad datu tipu saglabā.
* Vārda apstrāde
  1. Ja vārds nav dots, tad saglabā kļūdas paziņojumu “mainīgajam trūkst vārda”.
  2. Ja vārds sakrīt ar klases vārdu, tad saglabā kļūdas paziņojumu “’padotais vārds’ nevar būt klases vārds”.
  3. Ja vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “metodi nevar saukt par ‘rezervētais vārds’”.
  4. Ja vārds jau tiek izmantots citā laukā vai kā asociācijas galapunkta lomas vārds, tad saglabā kļūdas paziņojumu “klasē jau eksistē lauks/asociācijas galapunkts ar vārdu ‘padotais vārds’”, norādot rindu, kur padotais vārds jau tiek izmantots.
  5. Ja nav kļūdu, tad vārdu saglabā.
* Ja metodei ir definētas anotācijas, tad katrai anotācijai izsauc funkciju “Apstaigāt anotācijas”.
* Ja metodei nav definēta URL anotācija, tad saglabā kļūdas paziņojumu “Nav definēts metodes URL”.
* Ja metodei nav definēti argumenti, tad saglabā kļūdas paziņojumu “Trūkst argumentu definīcijas”.
* Ja metodei ir definēti argumenti, tad katram argumentam tiek palaista funkcija “Apstaigat argumentus”.
* Ja klasei, kurā metode tiek definēta, ir virsklase, tad pārbauda, vai metode ar padoto vārdu jau eksistē virsklasē. Ja eksistē tad tiek pārbaudītas šādas lietas:
  1. Ja abām metodes definīcijām ir atšķirīgs argumentu skaits, tad tiek saglabāts kļūdas paziņojums “Virsklases definīcijā metodei ir atšķirīgs argumentu skaits” , norādot rindu, kurā metode ir definēta virsklasē.
  2. Ja argumentu skaits ir vienāds, tad katram argumentam, kuras pozīcijas ir vienādas, pārbauda datu tipu. Ja datu tipi nesakrīt, tad tiek saglabāts kļūdas paziņojums “Arguemntam nr. ‘argumenta pozīcija’ nav tāds pats tips kā virsklasē”, norādot rindu, kurā arguments ir definēts virsklasē.

#### 2.2.2.8 Apstaigāt anotācijas

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu metodes anotācijas.

**Ievade**

Funkcijai ir šādi ievaddati:

* Anotācijas konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Kāds ir anotācijas tips?
* Ja tips ir URL, tad vai vērtība ir pareizajā formātā?
* Vai ir izmantotas iekavas un pēdiņas tam paredzētajās vietās?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja trūkst kādas iekavas vai pēdiņas, tad saglabā kļūdas paziņojumu “trūkst iekavas/pēdiņas”.
* Ja anotācijas tips ir URL un metodei vēl nav definēta URL anotācija, tad tiek skatīta anotācijas vērtība.
  1. Ja vērtībai nav URL specifiskie dati (programmēšanas valoda un lokācija), tad tiek saglabāts kļūdas paziņojums “trūkst URL specifisko datu”.
  2. Ja kļūdu nav, tad šo vērtību saglabā.
* Ja anotācijas tips ir URL un metodei jau ir definēta URL anotācija, tad saglabā kļūdas paziņojumu “URL jau ir definēts”.

#### 2.2.2.9 Apstaigāt argumentus

**Nolūks**

Funkcija nepieciešama, lai apstaigātu un pārbaudītu klases laukus.

**Ievade**

Funkcijai ir šādi ievaddati:

* Argumentu konteksts;

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai ir pareiza argumentu un komatu kārtība?
* Vai argumentam ir dots pareizs datu tips?
* Vai argumentam ir dots pareizs vārds?

Programmēšanas valodas gramatika pieļauj to, ka var ievadīt vai nu komatu, vai arī argumentu (datu tips un vārds), tāpēc ir jānoskaidro, kāds elements tiek lasīts un kādam elementam būtu jābūt. Visiem argumentiem ir jābūt atdalītiem ar vienu komatu. Ja metodei tiek definēti argumenti, tad jāsāk un jābeidz ar to, ka ir jābūt argumentam un nevis komatam.

**Izvade**

Izvade ir atkarīga no apstrādes:

* Elementu kārtības pārbaude:
  1. Ja pirms argumenta nav komata vai pēdējais argumentu elements ir komats, tad saglabā kļūdas paziņojumu “Trūkst komata”.
  2. Ja pirms komata nav argumenta, tad saglabā kļūdas paziņojumu “Tiek sagaidīts arguments”.
  3. Ja elements ir arguments, tad tas tiek pārbaudīts.
* Argumenta pārbaude:
  1. Ja argumentam nav dots datu tips, tad saglabā kļūdas paziņojumu “Trūkst datu tipa”.
  2. Ja argumentam trūkst vārda, tad tiek saglabāts kļūdas paziņojums “Trūkst vārda”.
  3. Ja nav dots pareizs datu tips, tad saglabā kļūdas paziņojumu “’padotais datu tips’ nav derīgs datu tips.”
  4. Ja argumenta vārds sakrīt ar rezervētajiem vārdiem, tad saglabā kļūdas paziņojumu “Argumentu nevar saukt par ‘rezervētais vārds’”.
  5. Ja argumenta vārds sakrīt ar kāda cita argumenta vārdu, tad saglabā kļūdas paziņojumu “Arguments ar vārdu ‘argumenta vārds’ jau eksistē.” Norādot rindu, kur padotais argumenta vārds jau tiek izmantots.
  6. Ja kļūdu nav, tad argumentu saglabā.

#### 2.2.2.10 Apstaigāt kļūdas

**Nolūks**

Funkcija nepieciešama, lai apstaigātu kļūdainos mezglus, kurus izķer ANTLR.

**Ievade**

Funkcijai ir šādi ievaddati:

* Kļūdas mezgls;

**Apstrāde**

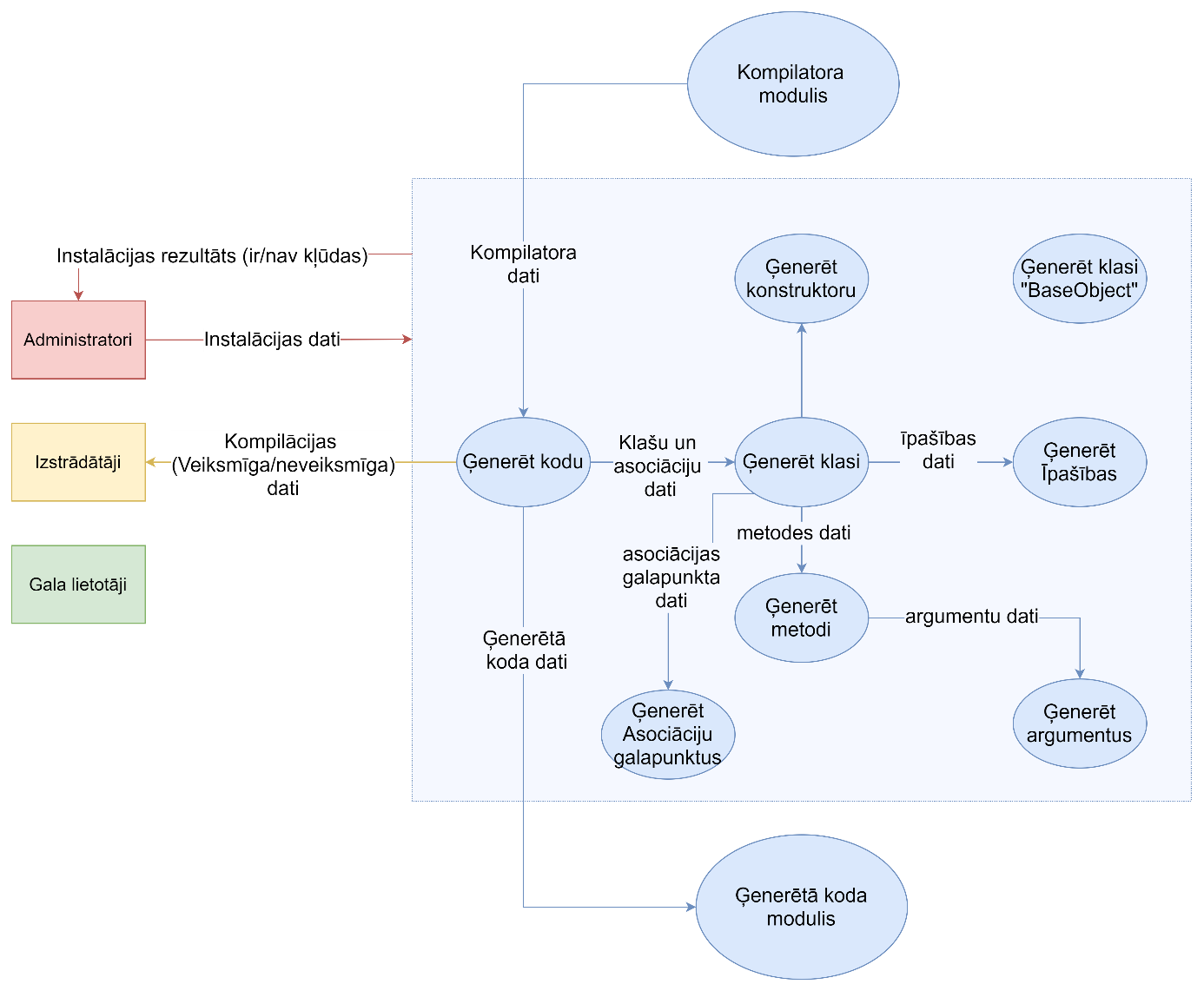
* Netiek veiktas nekādas pārbaudes un papildus apstrādes

**Izvade**

* Tiek saglabāta kļūdas paziņojums “Negaidīts ‘simbols’”, kur ‘simbols’ ir simbolu virkne, kuru ANTLR uzskata par kļūdainu.

### 2.2.3 Ģeneratora modulis

Pēc koda kompilēšanas, visi kompilēšanas dati (saraksti ar klasēm, asociācijām un kļūdām) tiek padoti ģenerēšanas modulim, kurš izmanto šos datus, lai noteiktu, vai kods ir jāģenerē un ja ir, tad kādas klases un to metodes, īpašības un asociācijas ir jāģenerē.

*2.3. att.* **2. līmeņa DPD diagramma ģeneratora modulim**

#### 2.2.2.1 Ģenerēt kodu

**Nolūks**

Funkcija nepieciešama, lai uzsāktu koda ģenerēšanu.

**Ievade**

Funkcijai ir šādi ievaddati:

* Vārdtelpa (namespace)
* Kompilatora dati (klases, asociācijas, kļūdas)

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai kompilēšanas rezultātā ir atrastas kļūdas?
* Vai ir dota pareiza vārdtelpa?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja kompilēšanas rezultātā ir atrastas kļūdas, tad tās visas tiek izdrukātas un ģenerēšana beidzas.
* Ja nav dota pareiza vārdtelpa, tad tiek izdrukāts kļūdas paziņojums “’Vārdtelpas nosaukums’ ir nepareizā formātā.” Un ģenerēšana beidzas.
* Ja kļūdas nav konstatētas, tad vispirms izsauc funkciju “Ģenerēt klasi ‘BaseObject’” un tad katrai klasei izsauc funkciju “Ģenerēt klasi”

#### 2.2.2.2 Ģenerēt klasi “BaseObject”

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klasi “BaseObject”, kurā ir klasēm nepieciešamās pamatfunkcijas.

**Ievade**

* Funkcijai nav nepieciešami ievaddati

**Apstrāde**

* Netiek veiktas nekādas pārbaudes un papildus apstrādes

**Izvade**

* Izvade ir fails “BaseObject.cs”, kurā ir uzģenerētā klase “BaseObject”[skat. ? nodaļu]

#### 2.2.2.3 Ģenerēt klasi

**Nolūks**

Funkcija nepieciešama, lai ģenerētu klasi.

**Ievade**

Funkcijai ir šādi ievaddati:

* Klases dati

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai klasei ir virsklase

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja klasei eksistē virsklase, tad ģenerētajā kodā tā tiek uzģenerēta. Citādi kā virsklasi ģenerē klasi “BaseObject”.
* Sekojošā secībā tiek izsauktas šādas funkcijas
  1. Ģenerēt konstruktoru
  2. Ģenerēt īpašības (ja klasei ir kaut viena īpašība)
  3. Ģenerēt asociācijas (ja klasei ir kaut viena asociācija)
  4. Ģenerēt metodes (ja klasei ir kaut viena metode)

#### 2.2.2.4 Ģenerēt konstruktoru

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klasei konstruktoru.

**Ievade**

Funkcijai ir šādi ievaddati:

* Klases vārds

**Apstrāde**

* Netiek veiktas nekādas pārbaudes un papildus apstrādes

**Izvade**

* Izvades failā tiek ierakstīts konstruktora kods. [skat. ? nodaļu]

#### 2.2.2.5 Ģenerēt īpašības

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klasei īpasības, ja tādas ir.

**Ievade**

Funkcijai ir šādi ievaddati:

* Īpašību dati

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai īpašībai ir aizsardzība

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja īpašībai ir aizsardzība, tad ģenerētajā kodā tā tiek uzģenerēta.
* Izvades failā tiek ierakstīts īpašības kods

#### 2.2.2.6 Ģenerēt asociācijas

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klasei asociācijas galapunktu, ja tādas ir.

**Ievade**

Funkcijai ir šādi ievaddati:

* Asociācijas dati

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Vai klase ir asociācijas avotklase vai merķklase?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja klase ir avotklase, tad funkcijai “checkAssociation” [skat. ? nodaļu] kā avotu padod asociācijas avotu un kā mērķi padod asociācijas mērķi. Pretējā gadīijumā kā avotu padod mērķi un kā mērķi padod avotu.
* Izvades failā tiek ierakstīts asociācijas galapunkta kods

#### 2.2.2.7 Ģenerēt metodes

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klasei metodes, ja tādas ir.

**Ievade**

Funkcijai ir šādi ievaddati:

* Metodes dati

**Apstrāde**

Tiek veiktas šādas pārbaudes:

* Metodes atgriežamā tipa pārbaude
* Vai metodei ir aizsardzība?

**Izvade**

Izvade ir atkarīga no apstrādes:

* Ja īpašībai ir aizsardzība, tad ģenerētajā kodā tā tiek uzģenerēta.
* Tiek izsaukta funkcija “Ģenerēt argumentus”.
* Atkarībā no metodes atgriežama tipa, tiek pareizi uzģenerēta atgriežamā vērtība
* Izvades failā tiek ierakstīts metodes kods

#### 2.2.2.8 Ģenerēt argumentus

**Nolūks**

Funkcija nepieciešama, lai uzģenerētu klases metodei argumentus, ja tādi ir.

**Ievade**

Funkcijai ir šādi ievaddati:

* Argumentu dati

**Apstrāde**

* Netiek veiktas nekādas pārbaudes un papildus apstrādes

**Izvade**

* Gadījumā, ja metodei nav neviena argumenta, tad nekas netiek uzģenerēts, citādi tiek uzģenerēti visi metodes argumenti.

### 2.2.4 Ģenerētā koda modulis

### 2.2.5 WebMemory modulis

## 2.3 Nefunkcionālās prasības

# 3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

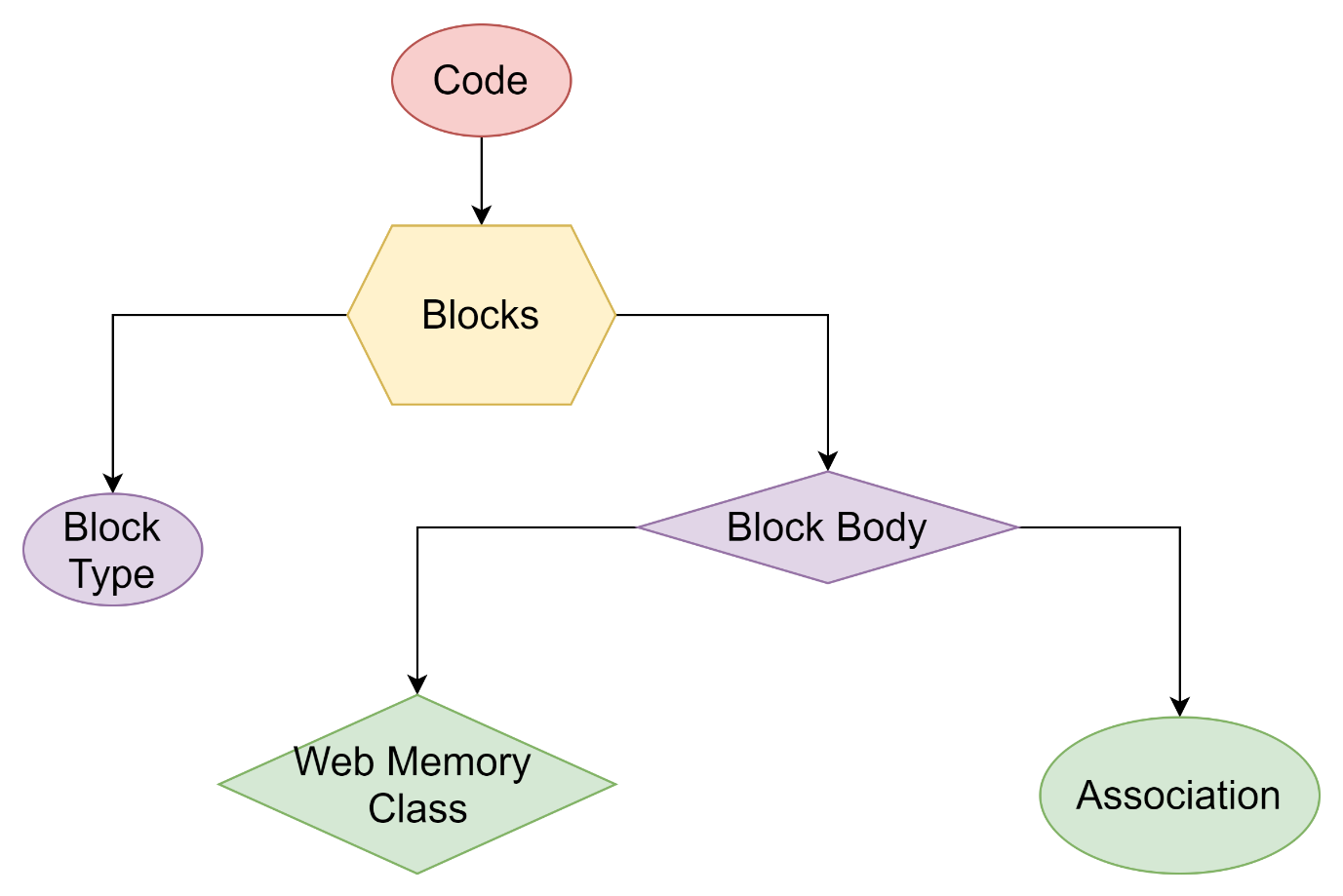
## 3.1 Programmēšanas valodas gramatikas projektējums

### 3.1.1 Koda projektējums

Šajā programmēšanas valoda programmkods tiek sadalīts pa vairākiem blokiem. Šajā gadījumā ir divu veidu bloki: klases un asociācijas. Katram blokam ir divas sastāvdaļas:

* Tips, kas apraksta, kāda veida bloks tas ir (Klasēm tiek izmantots atslēgvārds ‘class’ un asociācijām tiek izmantots atslēgvārds ‘association’).
* Ķermenis, kas apraksta pašu bloku.

Kodā var būt patvaļīgs skaits bloku, pat var nebūt neviena bloka. Katram blokam ir noteikti jānorāda vai nu tips, vai ķermenis (vismaz viena no sastāvdaļām var trūkt, bet noteikti vienai ir jābūt un kompilatoram to ir jāuztver kā kļūdu).

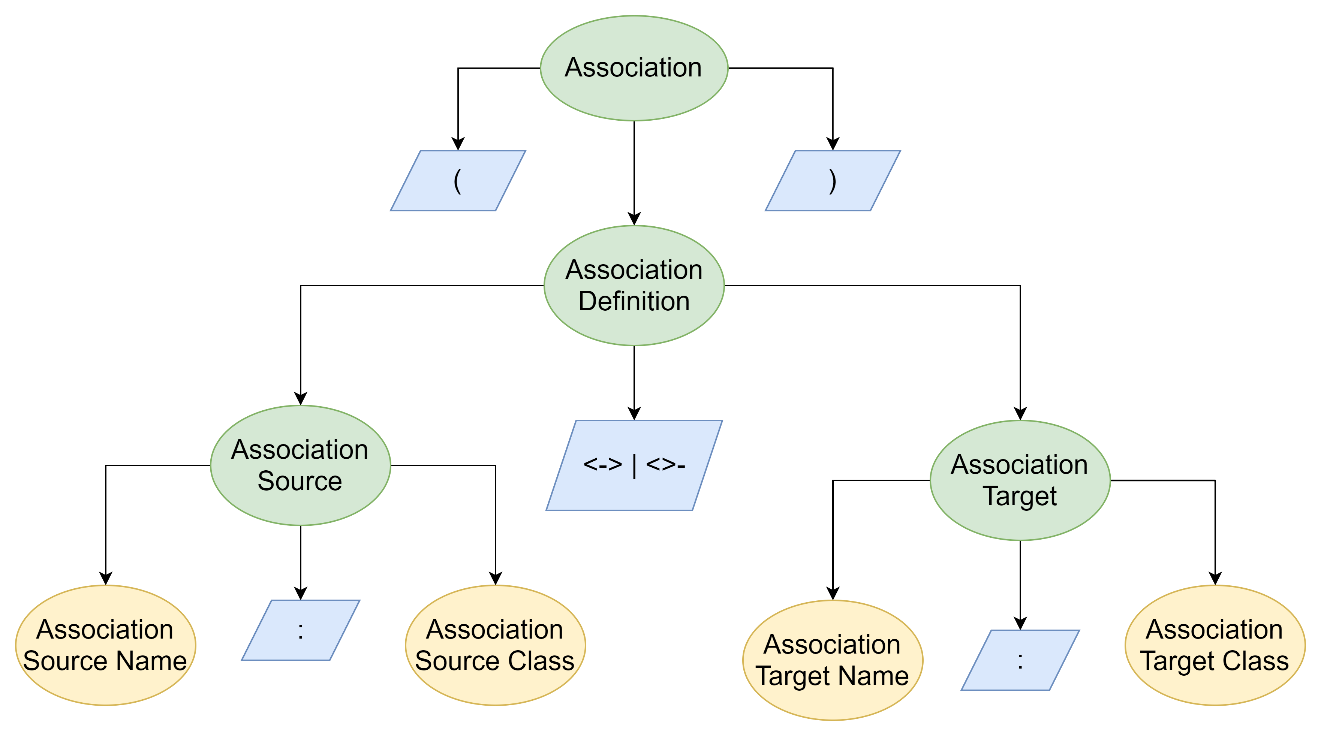


*3.1. att.* **gramatikas koks**

### 3.1.2 Asociācijas projektējums

Asociācijas ķermenis tiek rakstīts apaļajās iekavās. Tajās tiek norādīta informācija par asociācijas avotu un mērķi, kas ir atdalītas ar bultām. Avots un mērķis sastāv no lomas vārda un klases vārda, kas ir atdalīti ar kolu.

Lomas vārdi, klases vārdi, koli un bultas var netikt ierakstīta, bet kompilatoram to ir jāuztver kā kļūdu.



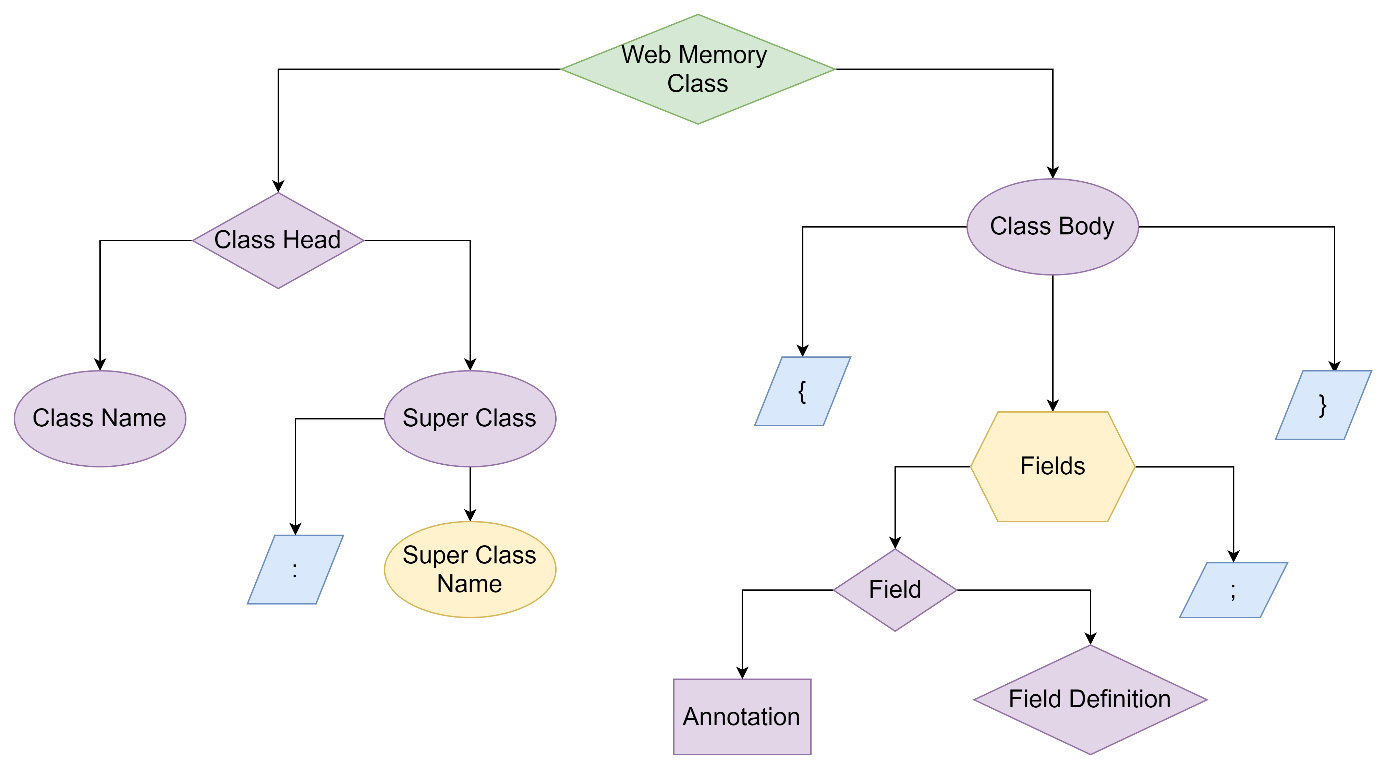
*3.2. att.* **asociācijas koks**

### 3.1.3 Klases projektējums

Klasei ir noteikti vajadzīgi dati par klases vārdu, tās virsklasi, vai dati par tās sastāvu, kas ir ierakstīts figūriekavās. Vieni no datiem var nebūt, bet kompilatoram to ir jāuztver ka kļūdu.

Ja tiek rakstīti dati par virsklasi, tad noteikti ir jābūt kolam pēc klases vārda.

Klases ķermenis (šajā gadījumā daļa, kas ir rakstīta figūriekavās) sastāv no laukiem (mainīgie un metodes), kas ir atdalīti ar semikolu, un katram laukam var definēt vai nu vairākas anotācijas (domātas metodēm), vai sniegt informāciju par pašu lauku (datu tips, vārds, aizsardzība un metodēm argumenti). Noteikti katram laukam tiek definētas anotācijas vai pati definīcija, bet kompilatoram to ir jāuztver kā kļūdu

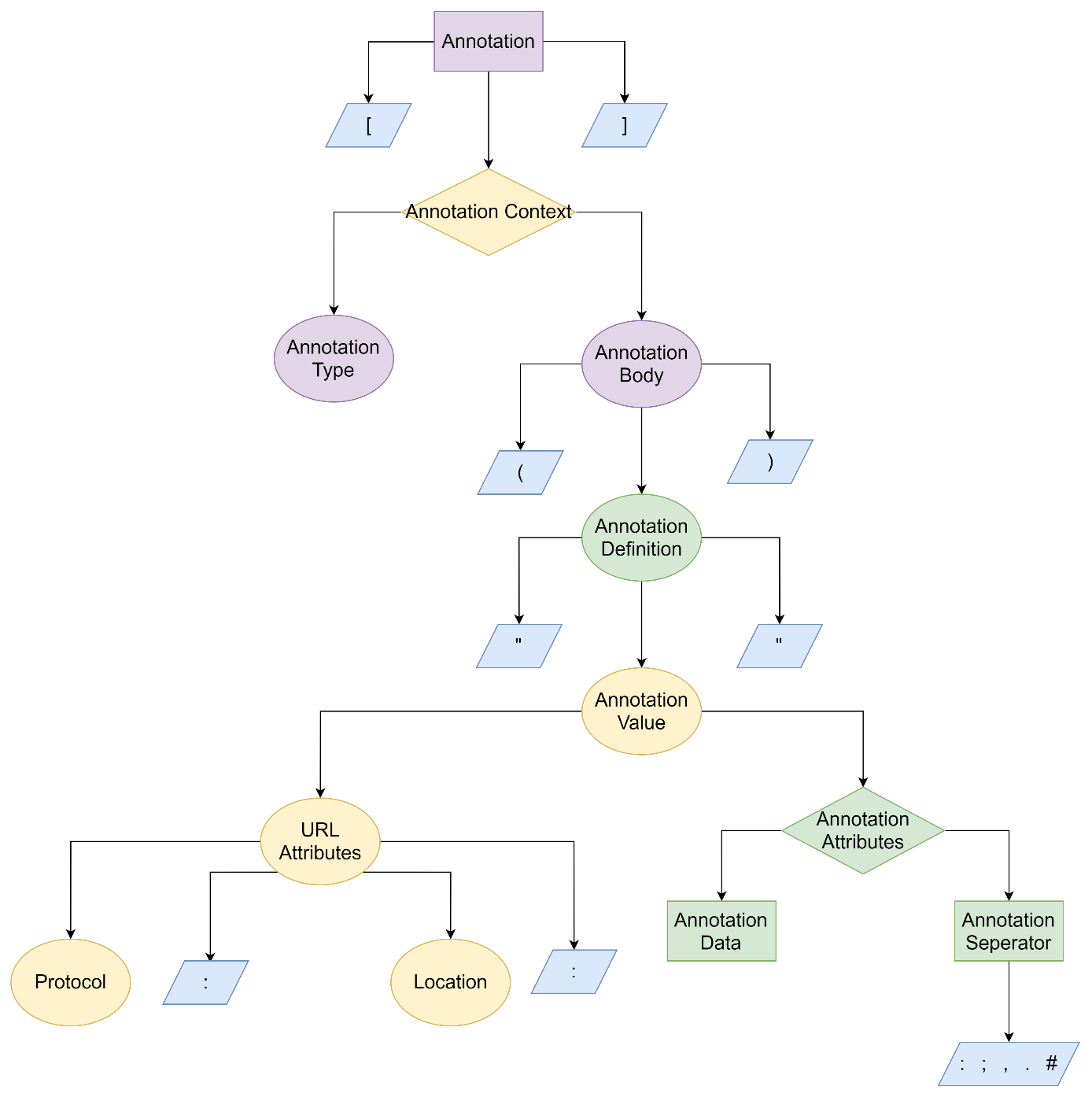


*3.3. att.* **klases koks**

### 3.1.4 Anotācijas projektējums

Anotācijas tiek definētas kvadrātiekavās. Tajās tiek ierakstīts anotācijas tips un tam blakus ir apaļas iekavas, kurās pēdiņās tiek rakstīta vērtība. Vērtība var būt jebkāda simbolu virkne, kas nesastāv no dubultajām pēdiņām un iekavām. Šīs simbolu virknes var tik atdalītas ar sekojošiem atdalītājiem: kolu, semikolu, punktu, komatu vai restīti. Ja anotācijas vērtība pirmie divi atdalītāji ir koli, tad teksts pirms otrā kola ar otro kolu ieskaitot tiek uzskatīts par URL anotācijas daļu, kura ir ierakstīta programmēšanas valoda un lokācija.

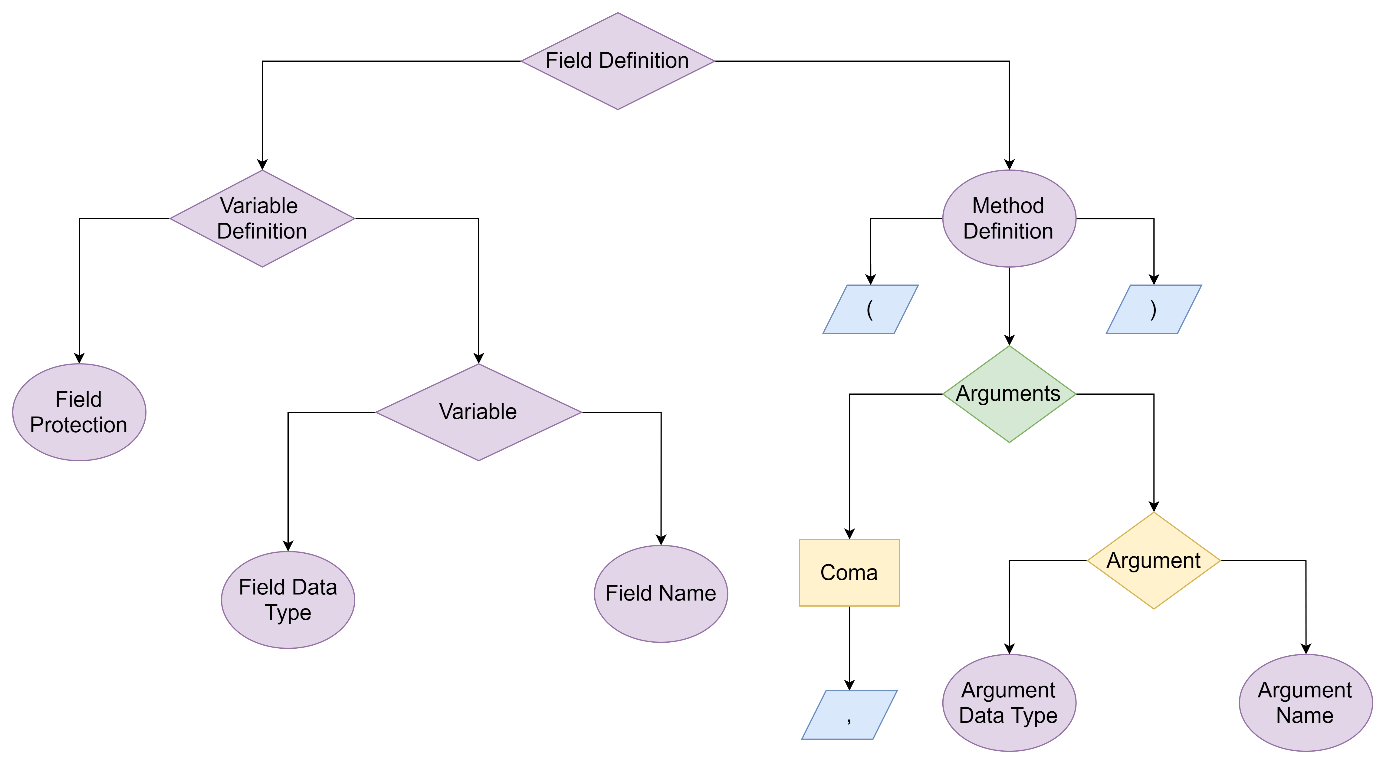
Drīkst nebūt jebkura no anotācijas sastāvdaļām, kas ir kvadrātiekavās, bet kompilatoram to ir jāuztver kā kļūdu.



*3.4. att.* **anotācijas koks**

### 3.1.5 Lauka definīcijas projektējums

Lauka definīcijā noteikti ir vajadzīgs definēt datu tipu, aizsardzību, lauka vārdu vai argumentus (metodēm). Var nebūt kāds no šiem datiem, bet kompilatoram to ir jāuztver kā kļūdu. Argumentiem ir jābūt definētiem apaļajās iekavās, kurās tiek rakstīts vai nu arguments (datu tips, vārds) vai tiek likts komats.

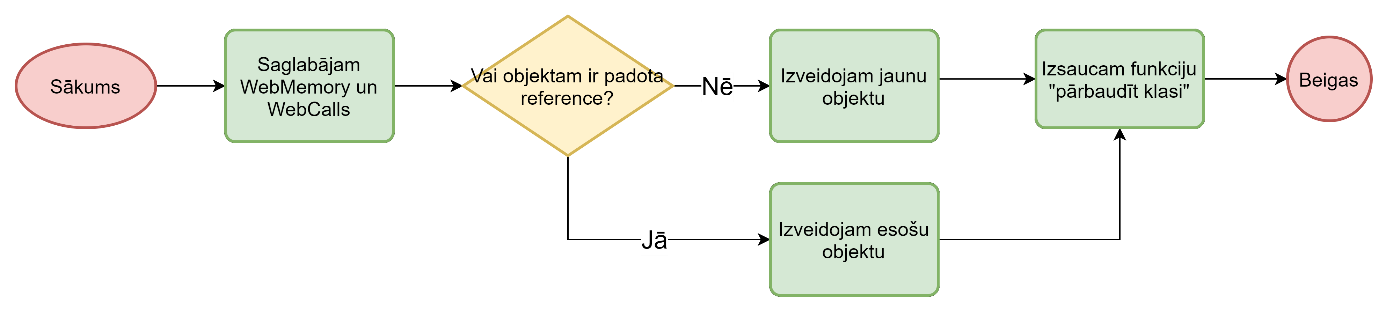
*3.5. att.* **lauka definīcijas koks**

## 3.2 Ģenerēto klašu lauku un funkciju projektējums

L

## 3.2.1 Konstruktora projektējums

Katrai klasei ir divi konstruktori. Pēc funkcionalitātes abi konstruktori ir vienādi. Vienīgā atšķirība ir tāda, vai konstruktoram tiek padota atsauce uz objektu. Ja tāda tiek padota, tad tiek veidots jau esošs objekts. Citādi tiek veidots pavisam jauns objekts. Pēc objekta izveidošanas tiek pārbaudīta klase [skat. 3.2.2 nodaļu].

*3.6. att.* **konstruktora secību diagramma**

## 3.2.2 Klases pārbaudes projektējums

Klases pārbaudei tiek veiktas divas lietas.

1. Cenšas atrast klasi pēc vārda
2. Cenšas atrast klases atribūtus pēc vārdiem (kad klase jau ir atrasta/izveidota).

Gadījumos, kad kaut kā nav (nav klases vai tās atribūta), tad to izveido. Kad tas ir izdarīts, tad tiek izveidots jauns objekts, ja tas ir nepieciešams.

Attēls, kurā ir teksts, ierīce, mērītājs, metrs

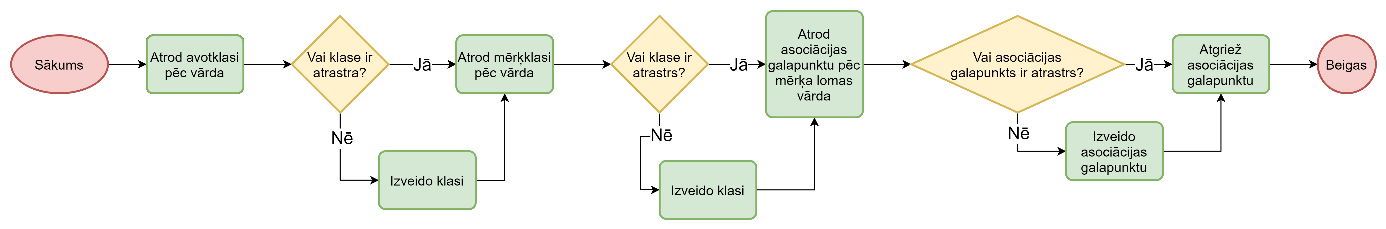
Apraksts ģenerēts automātiski*3.7. att.* **klases pārbaudes secību diagramma**

## 3.2.3 Asociācijas pārbaudes projektējums

Asociācijas pārbaudei tiek veiktas trīs lietas.

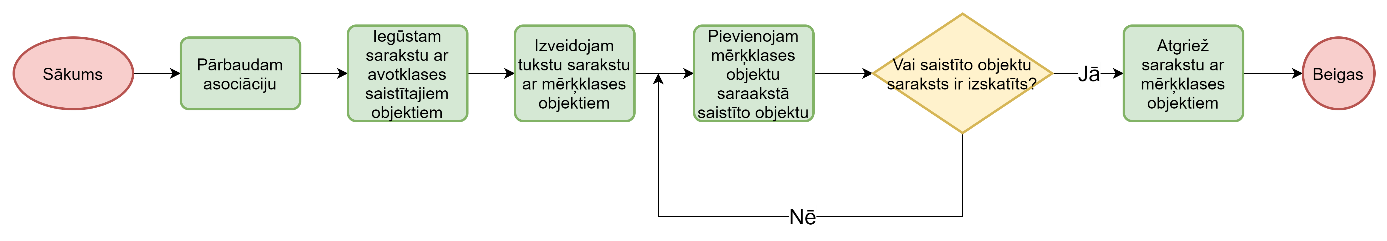
1. Cenšas atrast avotklasi pēc vārda
2. Cenšas atrast mērķklasi pēc vārda
3. Cenšas atrast asociācijas galapunktu avotklasē pēc mērķa lomas vārda.

Gadījumos, kad kaut kā nav (nav klases vai asociācijas), tad to izveido. Kad tas ir izdarīts, tad tiek atgriezts asociācijas galapunkta objekts, kuru izmanto klases asociāciju īpašībās [skat.3.2.4 un 3.2.5 nodaļas].

*3.8. att.* **asociācijas pārbaudes secību diagramma**

## 3.2.4 Asociācijas saraksta iegūšanas projektējums

Lai sāktu saraksta iegūšanu, vispirms ir jāpārbauda pati asociācija [skat. 3.2.3 nodaļu]. Tad no WebMemory iegūstam sarakstu ar objekta saistītajiem objektiem un izveidojam tukšu sarakstu ar mērķklašu objektiem, kurš ir rezultātsaraksts. Izejam cauri sarakstam, kuru iegūstam no WebMemory un katram saraksta elementam izveidojam objektus ar mērķklases tipu un pievienojam rezultātsarakstam. Kad visiem elementiem ir iziets cauri, tad rezultātsarakstu atgriež.

 *3.8. att.* **asociācijas saraksta iegūšanas secību diagramma**

## 3.2.5 Asociācijas saraksta uzstādīšanas projektējums

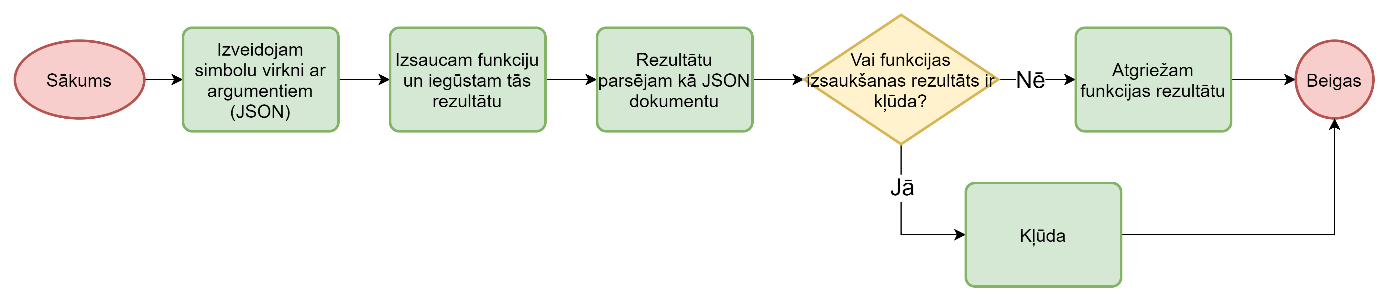
Lai sāktu saraksta iegūšanu, vispirms ir jāpārbauda pati asociācija [skat. 3.2.3 nodaļu]. Tad izmantojam padoto sarakstu ar mērķklases objektiem un izejam cauri katram tās elementam, kur katrā iterācijā objekts tiek saglabāts WebMemory.

Attēls, kurā ir teksts

Apraksts ģenerēts automātiski *3.8. att.* **asociācijas saraksta uzstādīšanas secību diagramma**

## 3.2.6 Metožu projektējums

Vispirms visus argumentus saglabājam kā JSON simbolu virkni. Tad tiek izsaukta funkcija [skat ? nodaļu], kuras rezultātā iegūstam simbolu virkni JSON formātā. Tad šo simbolu virkni parsējam kā JSON dokumentu, kurā glabājas vai nu funkcijas rezultāts, vai kļūda. Gadījumā, ja nav kļūdas, tad rezultātu atgriež noteiktā datu tipā.

 *3.8. att.* **metožu secību diagramma**