|  |
| --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE |
| Datorzinātnes un informācijas tehnoloģijas fakultāte |
| Informācijas tehnoloģijas institūts |
| **<studējošā vārds, uzvārds>**  <bakalaura/maģistra akadēmisko/profesionālo, pirmā/otrā līmeņa studiju programmas> students, stud. apl. nr. <.......> |
| **<DARBA/PROJEKTA TEMATIKAS NOSAUKUMS>** |
| **< BAKALAURA/MAĢISTRA DARBS, DIPLOMDARBS, DIPLOMPROJEKTS, INŽENIERPROJEKTS,**  **KVALIFIKĀCIJAS DARBS>** |
| Zinātniskais vadītājs  <zinātniskais grāds, akadēmiskais amats>  <vārds, uzvārds> |
| RĪGA <20….> |

*Šī lapa netiek izdrukāta – tās vietā vēlāk gatavam darbam tiek pievienota uzdevumu lapa.*

|  |
| --- |
| RĪGAS TEHNISKĀ UNIVERSITĀTE |
| DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE |
| Informācijas tehnoloģijas institūts |
| INFORMĀCIJAS TEHNOLOĢIJAS ATTĪSTĪBA ELEKTRONISKAJĀ TIRDZNIECĪBĀ |
| Edvīns Mališevskis |
| Anotācija |

|  |
| --- |
| RIGA TECHNICAL UNIVERSITY |
| FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY |
| Institute of information technology |
| DEVELOPMENT OF INFORMATION TECHNOLOGY IN ELECTRONIC COMMERCE |
| Edvīns Mališevskis |
| Annotation |

Goals of the work, results and their estimations.

The work contains \_\_\_ p., \_\_\_ tables, \_\_\_\_ figures and \_\_\_ appendixes.

saturs

[Ievads 3](#_Toc444598137)

[1. SAGATAVOŠANĀS DARBI UN IETEIKUMI 3](#_Toc444598138)

[1.1. Sagatavošanās darbu secība 3](#_Toc444598139)

[1.1.1. Rezerves kopijas izveidošana 3](#_Toc444598140)

[1.1.2. Virsrakstu izveidošana objektiem 3](#_Toc444598141)

[1.2. Ieteikumi drukas teksta formatēšanai 3](#_Toc444598142)

[2. VISPĀRĒJIE NORĀDĪJUMI 3](#_Toc444598143)

[3. TEKSTA NOFORMĒŠANA 3](#_Toc444598144)

[3.1. Sadaļu numerācija 3](#_Toc444598145)

[3.2. Sadaļu virsraksti 3](#_Toc444598146)

[3.2.1. Nodaļas virsraksts 3](#_Toc444598147)

[3.2.2. Apakšnodaļas virsraksts 3](#_Toc444598148)

[3.2.3. Punktu virsraksts 3](#_Toc444598149)

[3.2.4. Apakšpunktu virsraksts 3](#_Toc444598150)

[4. OBJEKTU IEKļAUšANA TEKSTĀ 3](#_Toc444598151)

[4.1. Attēli 3](#_Toc444598152)

[4.2. Tabulas 3](#_Toc444598153)

[4.3. Formulas 3](#_Toc444598154)

[5. izmantotās literatūras SARAKSTA sakārtoŠANA un atsauces 3](#_Toc444598155)

[6. PIELIKUMI 3](#_Toc444598156)

[rezultāti un SECINĀJUMI 3](#_Toc444598157)

[Izmantotās literatūras saraksts 3](#_Toc444598158)

[pielikumi 3](#_Toc444598159)

[1. pielikums. Noslēguma darba vāks 3](#_Toc444598160)

[2. pielikums. Pielikumā iekļauto objektu noformēšanas paraugi 3](#_Toc444598161)

[3. pielikums. Atsauču noformēšana 3](#_Toc444598162)

Ievads

Šī veidne ir veidota saskaņā ar RTU Informācijas tehnoloģijas institūta noslēguma darbu izstrādes noteikumiem un ietver sevī dažādus automatizācijas elementus, kas vienkāršos darba noformēšanu un novērsīs problēmas darba satura izmaiņu gadījumos.

Dokumenta saturs ietver sevī arī pašus Noformēšanas norādījumus, kurus ir ieteicams izlasīt, lai izvairītos no pretrunām un nesaskaņām nākotnē, lietojot dažādas *Word* versijas un valodas uzstādījumus.

Lietotājam ieteicams glabāt neskartu veidnes versiju, jo tas satur dažādus elementus, kuru pavairošana veicama tikai ar kopēšanas palīdzību.

# Programmēšanas valodas

## Jaunās programmēšanas valodas

Pasaulē eksistē milzīgs daudzums programmēšanas valodu, taču tas netraucē pastāvīgi rasties jaunām valodām. Rodas pilnīgi dabisks jautājums- kādēļ tās rodas. Eksistē taču tādi industrijas titāni kā “Java”, “C”, “C++”,” Objective-C”, kā arī ne tik veci, taču ieņēmuši specifiskas nišas- “JavaScript”, “PHP”,”Ruby”. Tam ir vairāki iemesli un pirmais ir valodu fokuss. Ir jāpiekrīt, ka zemu līmeņu mehānismu programmēšanai labāk izmantot C, bet interaktīvus web elementu - Javascript. Otrs iemesls ir valodas arhitektūra. Ne visas valodas var pielietot smagi piekrautās programmās ar lielu paralēlu skaitļošanu daudzumu. Arhitektūras specifikas kā “null[[1]](#footnote-1)” un ”Generics”. Daudzi programmētāji uzskata, ka null ir milzīga programmēšanu valodu kļūda. Un, piemēram, Java veidotāji atvainojas par Generics ieviešanu. Taču tā kā lielākā daļa programmēšanas valodu ir atgriezeniski saderīgas(Backward compatibility)[[2]](#footnote-2) , nav iespējams izmainīt vai uzlabot šīs daļas. Tieši šis ir iemesls, kādēļ rodas jaunas programmēšanas valodas, kuras ne atkārto savu priekšnācēju kļūdas un optimizējas jaunajām programmēšanas tendencēm. Zemāk būs apskatītas dažas programmēšanas valodas, kuras izveidojās pēdējo trīs gadu laikā.

### Google Go

Go- arī pazīstams kā golang tika palaists 2009. Gadā. Tā veidotāji ir trīs Google darbinieki Robert Griesemer, Rob Pike, un Ken Thompson. Šī publiskās piekļuves(open source) valoda tika raksturota kā ātra un viegli izmantojama, salīdzinot ar eksistējošām valodām kā Java un C. Tieši šīs valodas tika izmantotas kā Go pamats. Tā tiek izmantota visdažādākās organizācijās sākot ar BBC, SoundCloud un Facebook, līdz pat Lielbritānijas valdības mājaslapai GOV.UK. Tā tiek arī izmantota Docker tehnoloģijās. Go veidotāji vēsta: „Go ir mēģinājums kombinēt programmēšanas vieglumu dinamiskā valodā ar statisku, kompilētu valodu efektivitāti un drošību”

### Swift

Swift tika atklāts Apple WWDC conferences laikā 2014. Gadā. Tās veidošanas nolūks bija Objective-C valodas aizvietošana priekš OSX un iOS attīstībā. Apple šo valodu publiskoja 2015. gada Decembrī zem Apache licenzijas. Tas nozīmē, ka viss avota kods(source code) ir atklāti pieejams izmaiņām un uz tā pamatiem var tikt veidotas programmas, nesaistītas ar Apple. Swift, kuram piemīt līdzība ar modernajām Ruby un Python valodām, tika pakļauts neizmērojamai izaugsmei kopš tā palaišanas.”Swift iekļauj sevī drošu programmēšanu un modernus uzlabojumus, lai padarītu programmēšanu vieglu, dinamisku un interesantu” saka Apple.

### Scala

Scala ir funkcionāla un objekt-orientēta valoda, no kuras vārda var secināt par tās mērogojamību(scalability). Tā tiek pielietota labi pazīstamās organizācijās. Tā ir cieši saistīta ar JVM[[3]](#footnote-3). Tā tiek pielietota lielās uzdevumu kritiskās sistēmās(large mission critical systems) kā piemēram Twitter, LinkedIn un Intel.

### Rust

Rust 1.0 tika izveidots 2014. gadā Mozilla kompānijas iniciatīvas dēļ un uzlabots šo dažu gadu laikā. Dažos aspektos Rust ir līdzīgs C un C++, un Mozilla apraksta to kā „ jaunu programmēšanas valodu, kura fokusējas uz procesu izpildi, paralelizēšanu un atmiņas drošību”. Veidojot jaunu valodu un pieliekot elementus no mūsdienu programmēšanas valodu dizaina, Rust veidotāji izvairās no liela daudzuma atgriezeniski saderības prasībām, kurām tradicionālās valodas velta lielu uzmanību.

### Ceylon

Ceylon ir Objekt-orientēta augsta līmeņa valoda ar prasīgu statisku tipizāciju un SDK[[4]](#footnote-4), kuru pilnveido Red Hat kompānija. Pirmā informācija par šo valodu parādījās 2011. gadā, kad tās veidotāji izveidoja blogu, kur dalījās ar valodas panākumiem un sasniegumiem. Pirmā pilnā versija tika laista klajā 2013. gada 12. Novembrī. Šī valoda var tikt kompilēta ar JVM vai arī JavaScript.

## Specificētas un universālas valodas

Katra programmēšanas valoda var tikt izmantota vienā vai vairākās nozarēs. Pamatnozares iekļauj sevī- Web, Desktop, Mobile,Embedded.

* Web – Valodas aplikācijām, kurām ir piekļuve caur internetu un kuras strādā attālinātā serverī. Simbols- aplis ar līniju noklājumu.
* Desktop – Valodas programmām, kuras tiek instalētas un izmantotas uz konkrēta datora. Pašlaik tās ir pārsvarā tikai zinātniskās programmas. Simbols- monitors
* Mobile – Valodas aplikācijām, kuras strādā uz telefoniem(smartfoniem). Simbols- smartfons.
* Embedded –Programmēšanas valodas zemu līmeņu ierīcēm. Simbols-mikročips.



1.1. att. Valodu specialitāte un popularitāte

Kā to var redzēt attēlā 1.1, eksistē valodas ar ļoti šauru specialitāti kā R un arī pietiekami universālas valodas kā Java un C. Savukārt Web nodaļa dalās 2 daļās backend [[5]](#footnote-5)un frontend[[6]](#footnote-6). Parasti backend un frontend tiek rakstīti dažādās valodās. Parasti backend programmēšanā izmanto Java, C# vai PHP, taču frontend programmēšanā izmanto Javascript, tomēr gandrīz visās valodā ir freimvorks(???)[[7]](#footnote-7) frontend programmēšanai. Programmēšanas valoda, kura tiks apskatīta pirmā, saucas Ceylon. To var uzskatīt par ļoti jaunu valodu, kura tiks vaļā no problēmām, kuras roda Java programmētājiem, vēsta tās izveidotāji.

# CEYLON

## Kādēļ Ceylon

Uz JVM pamatiem tika izveidots liels daudzums valoda- Scala, Groovy,Clojure, Kotlin, Ceylon, Xtend, X10. Visām šīm valodām ir tieša piekļuve ilgi krātai Java infrastruktūrai, bibliotēkām, struktūrām un citam. Šis ir viens no iemesliem, kādēļ es domāju, ka uz JVM balstītās valodas būs veiksmīgi pielietojamas vienā no vispopulārākajām industrijām. Runa iet par Web platformām, lielām un sarežģītām sistēmām, kuras tiek izmantotas lielos uzņēmumos. Java pašlaik ieņem līdera pozīciju šajā jomā. Manuprāt, Ceylon is visperspektīvākā valoda no JVM valodu skaita. Ceylon nav funkcionējoša programmēšanas valoda, ko uzskatu par plusu, jo mūsdienu sistēmas ir pietiekami sarežģītas pat bez funkcionāliem elementiem. Ceylon var tikt kompilēts JavaScript, kas ir kaut kas eksotisks starp JVM valodām. Tas ļauj programmētājiem rakstīt programmas, kuras tiks izmantotas lietotāja pārlūkprogrammā, nevis servera pusē.

### Red Hat

Red Hat ir vislielākā kompānija, kura nodarbojas ar Open-Source Software[[8]](#footnote-8) produktu izstrādi. Šī kompānija tika izveidota 1993. Gadā. Šī ir tā kompānija, kura izveidoja atklātu bezmaksas operētājsistēmu, kura tiek izmantota visa pasaulē- Linux. Red Hat arī nodarbojas ar savu produktu atbalstīšanu un dažādu servisu piedāvāšanu. Šis ir otrs iemesls, kādēļ es domāju, ka Ceylon ir visperspektīvākā valoda. Manuprāt, tieši cilvēki, kuri izgudroja Linux un milzīgu daudzumu citu produktu, kuri strādā kopā ar līderkompānijām savās industrijās, var izveidot vislabāko programmēšanas valodu, balstītu uz JVM.

## JVM Alternatīvas

Gribētos detalizētāk apskatīt citas JVM valodas un pamēģināt tās izvērtēt kā alternatīvu Java valodai. Kā tas bija minēts agrāk, Java ir līderis start valodām serveru izstrādē lielām un sarežģītām sistēmām korporācijām un bankām.

### Scala

Scala ir statistiski tipizēta programmēšanas valoda, kura savieno objektorientētu modeli kopā ar funkcionālām idejām. Tas nozīmē, ka praksē ir iespējams deklarēt klases, veidot objektus un izsaukt metodes tāpat kā ierastajā Java. Taču Scala ievieš arī modernās funkcionālās programmēšanas iezīmes kā piemēram pattern matching[[9]](#footnote-9) datu struktūrām, local type inference[[10]](#footnote-10), persistent collections[[11]](#footnote-11), un tuple[[12]](#footnote-12) literals. Es uzskatu, ka Scala ir nepietiekami piemērota lieliem projektiem, kuros strādā liels cilvēku daudzums. Scala ir labi pielietojama projektos ar ļoti augstu slodzi un ļoti labu ātrdarbību. Arī šī valoda var izmantot gandrīz visas Java bibliotēkas.

### Groovy

Groovy ir dinamiski tipizēta objektorientēta valoda. Groovy dinamiskais raksturs ļauj veikt grandiozas manipulācijas ar kodu. Piemēram, ir iespēja palielināt objektus izpildes laikā(runtime)[[13]](#footnote-13)- pievienojot tiem laukus un metodes. Manuprāt, Groovy neder lieliem projektiem un komandām. Dinamiskā tipizācija nav īpaši pārliecinoša, jo vairāki cilvēki strādā pie viena projekta un neviens nepārzin visas projekta daļas. Tomēr Groovy ir ļoti ērta koda testēšanai. Sanāk ļoti lakoniski testi. Groovy piemīt visas Java raksturīpašības, dažas pat ir uzlabotas, piemēram, darbs ar kolekcijām.

### Clojure

Clojure ir dinamiski tipizēta programmēšanas valoda, kuru var uzskatīt par modernu Lisp[[14]](#footnote-14) interpretāciju. Tā radikāli atšķiras no pieņemtā objekt-orientētās programmēšanas veida. Faktiski, Clojure ir pilnīgi funkcionāla programmēšanas valoda, un tādēļ tā ir centrēta ap nemainīgām datu struktūrām, rekursijām un funkcijām. Es uzskatu, ka Clojure ir izmantojama tikai zinātniskos nolūkos, kā jebkura cita pilnīgi funkcionāla programmēšanas valoda, un tā ir pilnīgi nelietojama korporatīvos nolūkos. Šai valodai piemīt piekļuve Java ekosistēmai, taču, tā kā valoda nav objekt-orientēta, tās funkcionalitāte var būt ierobežota.

### Kotlin

Kotlin ir statiski tipizēta objekt-orientēta programmēšanas valoda. Tās dizaina galvenie mērķi ir spēja apvienoties ar Java API, izveidot sistēmu, kura atrod vairāk kļūdu kompilācijas laikā un saturētu mazāk teksta nekā Java. Kotlin veidotāji uzskata, ka Scala ir tuvs piemērs viņu dizainam, taču tās sarežģītība un ilgais kompilācijas laiks, salīdzinot ar Java, ir milzīgs mīnuss. Kotlin mērķis ir pārvarēt šīs mīnusus. Manuprāt, Kotlin ir labākais kandidāts no iepriekš apskatītām valodām, kuru var intensīvi pielietot lielos projektos. Saīsinātais kompilācijas laiks un kļūdu atrašana kompilācijas laikā samazinās cilvēcisko kļūdu skaitu. Kotlin izstrādātāji ir Jetbrains kompānija, kura ir izstrādājusi lielu skaitu IDE[[15]](#footnote-15) visdažādākajām valodām. Viņu IDE Java valodai nesen tapusi par vispieprasītāko šai valodai.

### Xtend

Xtend ir statiski tipizēta objekt-orientēta valoda. Tās lielākā atšķirība no citā valodām ir tās kompilēšana, kura izmanto pretty-printed Java kodu nevis baitkodu. No tā seko, ka šajā valodā ir iespējams strādāt ar ģenerētu kodu.

Xtend atbalsta divu veidu metožu piesaukšanu : Noklusējama Java nosūtīšanu ( default Java dispatching) un daudzkārtīgu nosūtīšanu (multiple dispatching). Daudzkārtīgā nosūtīšanā tiek izvēlēta pārslogota metode, balstoties uz izpildlaika veidu tās argumentiem. Xtend nodrošina daudzas citas populāras funkcijas, kas pieejamas arī citās valodās, piemēram, operatora pārslodze un tipa secināšana(operator overloading and type inference). Interesanta šīs valodas iezīme ir aktīvās anotācijas. Aktīvās anotācijas ir veids, kā izmantot meta programmēšanu[[16]](#footnote-16) koda kompilācijas laikā. Manuprāt, galvenā šīs valoda iezīme ir arī tās galvenais trūkums. Rodas jautājums, kādēļ programmēt valodā, kura vēlāk pārrakstās Java kodā. Es uzskatu, ka izmantot šo valodu būs grūti un neērti lielos korporatīvos projektos.

### X10

X10 ir eksperimentāla objekt-orientēta programmēšanas valoda, kuru izstrādāja IBM. Tā atbalsta tādas funkcijas, kā pirmklasīgas funkcijas (first-class functions) [[17]](#footnote-17), kuras ir paredzētas, lai veicinātu efektīvu plānošanu augstas veiktspējas paralēlā skaitļošanā. Šī valoda ir balstīta uz programmēšanas modeli- “partitioned global address space”. Šajā modelī katrs process dala globālo adrešu telpu, un šīs telpas daļas tiek piešķirtas privātai atmiņai vietējiem datiem un piekļuvēm. Lai strādātu ar šo modeli, X10 izmanto specializētas iebūvētas valodas konstrukcijas, lai strādātu ar laiksakritība un izplatītu izpildi. Tā kā IBM paši nosauc šo valodu par eksperimentālu, ir muļķīgi izmantot šo valodu korporatīvos projektos. Es uzskatu, ka IBM paši izmantos šo valodu veidojot kādu ļoti sarežģītu sistēmu līdzīgu IBM Watson[[18]](#footnote-18).

# Tehniskie aspekti

## Pamati

### Hello Word!

Tradīciju ietvaros pirmā programma jaunajā programmēšanas valodā ir pasaules sveicināšana. Cylon valodā tas izskatās šādi.

Kā redzams attēlā 3.1.1 Ceylon neizmanto visiem pierasto klasi Main ar statisko metodi main. Ceylon valodā run metode ne izmanto nekādu parametru, neko neatgriež un izmanto shared modifikāciju. Shared nozīmē, ka šī metode ir publiska un var tikt izsaukta jebkurā aplikācijas vietā.



3.1.1. att. Hello World!

### Tipi

Atšķirībā no Java, Ceylon neviens tips nevar būt null pēc noklusējuma.



3.1.2. att. Ne null tipi

Kā redzams attēlā 3.1.2. ir nepieciešams pievienot „?” blakus tipam, lai šī mainīgā vērtība varētu būt null. Tam ir ļoti liela nozīme izstrādei komandā. Jo, kad pēc vienas klases izmantojas vēl citas klases, ir ļoti grūti uzreiz saprast, vai šī metode var atgriezt null vērtību. Ir svarīgi zināt, vai ir vajadzība vērtības pārbaudei. Vēl grūtāk ir kad metode pieņem objektus ar vismaz 30 laukiem nevis vienu vērtību. Ceylon gadījumā, IDE dos padomu, ja jūs gribēsiet izmantot mainīgo, kurš var vai nevar būt null. Tas ekonomē milzīgu laika daudzumu. Ceylon piemīt arī pilnīgi tipiskie Class, Interface, Object. Taču tam arī piemīt papildus tipi, kā:

* Anything/Nothing – Pats augstākais tipu līmenis. Visi tipi, objekti un klases var būt kā Anything tā arī Nothing.
* Null – Ceylon valodā, Null ir patstāvīgs tips, kurš nozīmē vērtības trūkumu.
* Basic – ir noklusējuma superklase klasēm, kuras deklarācijai trūkst izklāstījuma.
* Sequence – ir ne tukšu sekvenču tips.
* Sequential -ir sanumurēts tips ar apakštipiem Sequence un Empty.
* Empty – ir Sequential apakštips, kurš nesatur elementus.
* Tuple – ir ne-tukšo Sequence apakšklase. Tā atšķiras no Sequence ar savu īpašību kodēt katru savu elementu atsevišķi.

### Operatori

Papildus standarta saskaitīšanas, atņemšanas un loģiskajiem operatoriem, ir arī jaunie un izmainītie operatori:

* Entry – Ļauj veidot “Entry<KeyType,ValueType>” objektus, kuri izmantojas, piemēram, Java valodā kā Mar kolekcijas elementi, izmantojot īsu pierakstīšanu kā parādīts attēlā 3.1.3.



3.1.3. att. Īsa Entry veidošana

* Compare – Salīdzina divus objektus, kuri manto „Comparable” un atgriež Comparison, tas izskatās kā “<=>”.
* Null – drošs operators, norāda, vai mainīgais var būt null, izskatās kā jautājuma zīme pie specifiskā tipa.
* Span – operators, kurš dod iespēju pielietot vairākiem masīva elementiem vienlaicīgi, kā parādīts attēlā 3.1.4, un tas izskatās kā [from..to].

3.1.4. att. Span operatora piemērs

Tāpat eksistē ari variants, kad var norādit tikai augšējo vai lejējo elementu,

[from ..] vai [.. to].

* Measure – operators, kurš ir līdzīgs span operatoram, norādīdams, no kura elementa un cik elementu. [from:length]
* In-operators palīdz noskaidrot, vai noteiktais elements atrodas kolekcijā, piemērs attēlā 3.1.5. 

3.1.5. att. In operatora piemērs

### Loģiskie izteicieni

Operatori Ceylon valodā izskatās pietiekami normāli, diez vai ir iespējams izgudrot kaut ko kardināli citādu. Ceylon sagaida ar visiem pierastiem operatoriem:

* For – Standarta operators, kuram piemīt vairāki veidi, for, foreach.
* If/else – Arī nemainīgs operators, kuram piemīt papildus pārbaudes tipam, eksistencei un tukšumam, ar atslēgvārdiem, is- objekta tipa pārbade, exists-vai tips nav null, nonempty-pārbauda kolekcijas eksistenci.
* Switch – nekādu specifisku modifikāciku operatoram switch nepievienoja. Atšķirībā no Java/C “Default” komandas vietā izmantojas “else”, lai izvēlētos variantu, kurš nav aprakstīts nevienā “case” operatorā.
* While – izskatās un strādā tāpat kā populārajās valodās - Java/C
* Try/Catch/Finally – Arī nav jaunums cilvēkiem, kuri izmantoja vai pat tagad izmanto Java/C. Try- Cenšas izpildit kādu darbību, catch0 noķert izņēmumu, kurš tika izmests try bloka izpildes procesā, Finally = izpilda finally bloku pēc veiksmīgas vai neveiksmīgas try bloka izpildes.

## OOP

Šajā nodaļā es gribētu parunāt par OOP funkcijām Ceylon valodā. Ceylon piemīt klases un objekti, interfeisi.Ceylon atbalsta lielu skaitu mantojumu, taču ar specifiskiem ierobežojumiem, Red Hat sauc to par “mixin” mantošanu.Principa mantošana Ceylon valodā ir realizēta kā Java. Nav iespējams mantot vairāk kā 1 klasi, taču var mantot, vai kā ir rakstīts pašā valodā, “apmierināt” vairākus interfeisus vienlaicīgi. Ceyloin piemīt vel viena interesanta īpašība. Atšķirībā no C un Java, Ceylon valodā nav obligāti javeido klase, lai varētu rakstīt kodu.

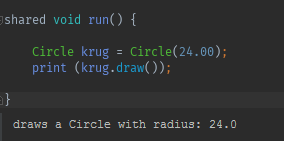
### Klase

Ceylon valodā par klasi sauc struktūru, kura atspoguļo reālu priekšmetu, mehanizmu vai konceptu. Klase var mantot citas klases, bet tikai vienā eksemplārī, tātad nav iespējams mantot vairāk kā vienu klasi, taču viena klase var but mantota bezgalīgi daudz reizes. Mantošanai nav dziļuma ierobežojumu. Klasei var piemist lauki – mainīgie, kuri attiecas tikai uz šo specifisko klasi, un metodes, darbības, ko šī klase var pildīt. Klases var būt arī abstraktas, kas nozīmē, ka nav iespējams izveidot šīs klases objektu.



3.2.1 att. Abstrakta un normāla klase

Attēlā 3.2.1 ir redzams mantošanas piemērs. Klase Circle, manto abstrakto klasi Figure, padodot “Circle” rindu kā parametru. Tā arī pārraksta draw metodi. Šāda koda darbības piemēru var redzēt attēlā 3.2.2.



3.2.2 att. Abstrakta un normāla klase-rezultāts

### Interfeiss

Interfeiss – ir savdabīgs pienākums, kuru ir jāizpilda klasei, kura realizē interfeisu. Interfeisa objekts ,tāpat kā abstraktas klases objekts, nevar but izveidots. Ceylon valodā caur interfeisiem tiek realizētas daudzkārtīgas mantošanas, tā kā viena klase var mantot vairākus interfeisus.

3.2.3 att. Interfeisa realizācija

Kā redzams attēlā 3.1.8, interfeisam TwoDimensional piemīt 2 metodes> square un perimeter, kurām ir jāatgriež daļskaitļi. Pēc tāda paša principa kā ar abstraktām klasēm, medotes tiek pārveidotas Circle klasē. Circle klasē var pievienot interfeisa realizācijas, piemēram atbildīgo par krāsu un likt Circle klasei zināt kādā krāsā tā ir. Draw metodes piemērs ir redzams attēlā 3.2.3.



3.2.4 att. Interfeisa realizācijas rezultāts

## Iezīmes

Šajā nodaļā gribētos apskatīt valodas specifiskās iezīmes. Iezīmes, kuru nav klasiskajās Java/C valodās, kuras ir raksturīgas tikai jaunajām, iespējams funkcionālajām programmēšanas valodām. Ceylon piemīt šādas specifikas- pseidonīmi, modularitāte, kuras dēļ jau kuro reizi atliek Java 9 izlaišanu ar Jigsaw projektu.

### Pseidonīmi

Ceylon valodā vai izmantot pseidonīmus- jebkuras lietas vārda mainīšana. Tie var uzlabot koda lasāmību, īpaši, kad interfeisos vai kolekcijās ir milzīgs daudzums generic vērtību.

3.3.1 att. **Pseidonīmi**



Kā redzams attēlā 3.1.1, pseidonīmus Ceylon valodā var izmatot divos veidos. Pirmais veids ir izmantot atslēgvārdu alias, kura gadījumā šim pseidonīmam var pieškirt jebkādu vienu vai vairākas nozīmes. Gan veselie Integer skaitļi, gan daļskaitļi Float var atrasties pseidonīmu Num. Pseidonīmu PeopleMap ir ērta ieraksta saīsināšana. Attēlā ir divas reizes izmantots uzraksts „given Value satisfies Object”, kas nav pseidonīms, taču es piedēvēju to tai pašai kategorijai. Šis uzraksts liecina, ka parametram, kurš tiek nodots Value vietā, obligāti jārealizē Object interfeisu.

### Union Tipi

Union Tipi – ir iespēja uzstādīt vairākus dažādus parametra tipus. Jūs varat izveidot metodi, kura pieņems String vai Integer vērtības, kas samazina koda daudzumu nevajadzīgām ienākošo parametru tipu pārbaudēm.



3.3.2 att. **Union Tipi**

Kā redzams attēlā 3.3.2., printType metode pieņem „val” vērtību, kas savukārt var piederēt jebkuram no 3 tipiem: String, Integer, Float.

### Enumerated Types (Uzskaitītie tipi)

Šī ir valodas īpašība, kura ļauj deklarēt, kuras klases manto doto interfeisu vai abstrakto klasi. Šī īpašība ļauj izmantot switch izteiksim, un arī ļauj klasei mantot divus interfeisus no vienas grupas. Piemēram, ja eksistē interfeiss CartPart, tam ir trīs mantinieku interfeisi: AudiCarPart, BmwCarPart un MercedesCarPart, tak, piemēram, BrakeSystem klase var mantot tikai vienu no šīm klasēm, citādi programma izmetīs izņēmumu.



Attēlā 3.3.3 parādīts Switch izmantošanas piemērs uz interfeisu. Java valodā, piemēram, String izmantošana Switch izteicienā parādījās tikai versijā 1.7, kamēr pēdējā versija ir 1.8, nerunājot par interfeisiem un citām struktūrām.

.

3.3.3 att. Enumerated Types

### Importēšana

Ceylon valodā, tāpat kā daudzās citās valodās, arī ir jāimportē klases, lai tās varētu izmantot. Ceylon piemēt speciāli mehānismi vienādu nosaukumu klašu importēšanai. Eksistē divi varianti, kā izvairīties no klases pilnā nosaukuma, pieliekot klāt visu tā package nosaukumus.



3.3.4 att. **Importēšana**

Kā redzams attēlā 3.3.4, pirmais variants- izmantot pseidonīmu, lai nosauktu importējamo klasi, kā situācijā ar HashMap un Map no java.util, kuru pseidonīmu nosaukumi ir Jmap un JhashMap. HashMap klase no ceylon.collection tiek importēta metodē tiešā veidā.

### Funkcijas

Ceylon valodai piemīt daudzums īpatnību, kuras raksturīgas jaunām un pulnībā funkcionālām valodām, saistībā ar funkcijām. Ceylon valodā visas funkcijas tiek realizētas izmantojot Callable interfeisu. Pirmā valodas īpatnība ir augstākās kārtības funkcijas. Tās ir funkcijas, kuras var argumenta vietā pieņemt citas funkcijas.



3.3.5 att. Augstākās kārtības funkcija

Kā redzams attēlā 3.3.5 repeat metode pieņem 2 argumentus, no kuriem viens ir vesels skaitlis un otrs ir funkcija, kura neko neatgriež. Repeat metode izsaukts tai nodoto funkciju vairākas reizes, kuras tiek noteiktas ar otru nopdoto argumentu. Šādā situācijā nodotā funkcija nedrīkst pati pieņemt nekādus parametrus.

Ceylon valodā eksistē arī, tā saucamās, Curried funkcijas. Metode vai funkcijavar tikt deklarēta curried formā, ļaujot šai metodei vai funkcijai būt daļēji pielietojamai tās argumentiem.



3.3.6. att. Curried functions

Kā redzams attēlā 3.3.6, adder metode var pieskaitī vienam daļskaitlim citu. Izmantojot Ceylon valodas īpašības, mēs varam izveidot funkciju, kura aizvieto pirmo argumentu ar vieninieku. Šādā veidā tiek izveidota funkcija, kura pieņem tikai vienu argumentu- daļskaitli. Šīs funkcijas nolūks ir pieskaitīt pie daļskaitļa vieninieku.

### Agrumenti

Jūs varat padomāt, kas tāds savādāks var būt argumentos, taču Ceylon valodai piemīt dažas īpatnības darbā ar argumentiem. Pirmā īpatnība-nosauktie argumenti. Šī īpatnība ir vaidota, lai atvieglotu koda lasīšanu un nenes sevī nekādas papildus funkcionalitātes.



3.3.7 att. Nosauktie argumenti

Kā redzams attēlā 3.3.7 someMethod metodes parametri to izsaukšanas vietā nav rakstīti kā C veidīgas valodas, ar atdalīšanas komatu, bet gan nedaudz savādāk, taču lasās tāds variants zināmi vieglāk.

### Darbs ar kolekcijām

Ceylon valoda neko jaunu darbā ar kolekcijām praktiski neieviesa, taču kolekciju saistītās izpilde ir veidota daudz lokaniskāk.

3.3.8 att. Straumes un secības veidošana

Kā redzams attēlā 3.3.8 straumes veidošana- pirmā rinda attēlā- vai secību- otra rinda attēlā- Veylon valodā izpildās, izmantojot for elementu deklarācijā. Šo pierakstu var arī nodot kā argumentu, kā parādīts attēlā 3.3.8.

### Konstruktora īpatnības

Papildus standarta konstruktoriem, tādiem kā Java valodā, Ceylon piemīt vēl 2 veidu konstruktori. Pirmais ir Named konstruktors- konstruktors, kuram ir nosaukums un kurš var pieņemt absolūti jebkādus parametrus. Otrs variants ir Value konstruktors- konstruktors, kurš nepieņem nekādus parametrus un veido klases objektu tikai vienreiz dotā kontekstā. Tas veido, tā saucamo, Singleton[[19]](#footnote-19).



3.3.9 att. Nosauktā konstruktora izmantošanas piemērs

Kā redzams attēlā 3.3.9, nosaukto konstruktoru var pieņemt kā sava veida statisku klasi.

### Mijiedarbība ar datubāzēm

Ceylon valodā nav speciāli izstrādātas metodes darbā ar datu bāzēm, taču projekta vadītājs ir Gevins Kings, cilvēks, kurš ir Hibernate projekta veidotājs.

Hibernate ir pazīstams programmētājiem, kuri izmanto Java valodu. Hibernate ir bibliotēka, kura ir veidota, lai izpildītu objekt-relāciju uzdevumus. Tā kā Ceylon var izmantot jebkādas Java bibliotēkas, mojiedarbībai ar datubāzēm problēmu nebūs, jo jau ir izveidoti dažādi risinājumi priekš Java valodas. Gadījumā ar web izstrādājumiem, Ceylon valodai nav specifisku bibliotēku un struktūru, kā piemēram, Scala valodā. Ceylon tiks izmantots kā rāmis, nevis pilnveidīga ekosistēma veb applikāciju servera daļas izstrādāšanā.

# Novērtējums

## Ceylon, plusi

### Seamless web development

Ceylon valoda kļūs par pārliecinošu platformu web izstrādē, ļaujot jums atkārtoti izmantot kodu, rakstītu Java serverim Javascript klientā, izmantojot tīru native kodu abās pusēs. un otrādi. Ar viņu jauno serializācijas iekārtu, būs iespējams nemanāmi pārvietot objektus no klienta uz serveri. Ceylon valodas SDK nāk kopā ar HTTP servera moduli, kurš ir balstīts uz Undertow, un transakciju moduli, balstītu uz JBoss Transactions, taču, ja ir vēlme pēc kaut kā „smagāka”, Ceylon moduļi ja var tikt izvietoti uz RadHat OpenShift mākoņa platformas un WildFly vai uz Vert x 2 (Vert x 3 atbalsts drīz nāks klajā). Kā alternatīva, Ceylon valodu var izmantot Node.js.

### Truly disciplined static typing

Kombinācijā ar pareiziem instrumentiem, statiskā tipizācija padara kodu spēcīgāku, labāk saprotamāku un vieglāk uzturamāku. Taču lielākā daļa valodu pievieno iezīmes un API, kuri grauj viņu statiskā tipa sistēmu. Pavisam neliels daudzum valodu, tādu kā ML un Haskell izvairās no tādām sekām, parādot īsteni principiālu un disciplinētu pieeju statiskajai tipizācijai. Programmētāji, kuri strādā ar šīm valodām ir secinājuši, ka viņu programmām ir tendence strādāt no pirmās vai otrās palaišanas reizes. Protams, nedaudz vairāk laika tiek patērēts, lai izveidotu programmu, kuru kompilators pieņemtu, taču, kad kompilators ir apmierināts, tik daudz izplatītu kļūdu ir novērsts, ka programma ir parasti gatava vai arī gandrīz gatava.

Ceylon seko šai filozofijai un , neskatoties uz to, ka Ceylon atšķiras no ML valodas, rezultāti rāda, ka Ceylon piemīt šī pati tendence, gandrīz nekavējoties strādāt.

Ceylon valodas tipu sistēma ir izplatīta, tajā ieskaitot, tipus ar abiem izmantošanas vietas un deklarācijas vietas kovariāciju un kontrkovariāciju, mantošanas maisīšanu, galveno instanču mantošanu, summu tipu, atdalības (disjunctivity) analīzēm, eksperimentālu atbalstu augstākās kārtas un ranka generics vērtībām un, protams, savienības un krustošanas tipiem.

### Savienošanas un krustošanas tipi

Ceylon bija pirmā modernā valoda, kura iepazīstināja ar pilno implementāciju savienošanas un krustošanas tipiem, un arī pirmā valoda, kura saprata, ka savienošanas un krustošanas tipi veido plūsmas rakstīšanu un tipu secināšanu, sevišķi generic tipu secināšana ir daudz vienkāršāka, prognozējamākā un elegantāka. Līdz pat šodienai, šī ir valoda ar visizsmalcinātāko atbalstu apvienošanām un krustošanām.

Ir grūti izskaidrot, cik īstenībā ir svarīga tipu apvienošana un krustošana. Tās abas ir ļoti tālas no eksotiskām konstrukcijām, kuras dažreiz var atrast savādās bibliotēkās. Tās abas veido būtisku daļu ikdienas pieredzē koda rakstīšanā Ceylon valodā. Ir grūti atrast ievērojamu Ceylon programmu, kura neizmanto savienotos tipus.

### Plūsmjūtīgā tipizācija (Flow-sensitive typing)

Ceylon bija arī pirmā modernā valoda, kura pieņēma flow-sensitive typing jēdzienu, kur nosacījumu loģika funkcijas pamattekstā ietekmē tipus, izsecinātus kompilatorā. Šī pieeja novērš ievērojamu daudzumu liekvārdības un ievērojamu daudzumu tipu lietošanas kļūdu. Kombinācijā ar Ceylon valodas spēcīgo pārklājumu un atdalības analīzi, tā palīdz atrast dažus citus loģisku kļūdu veidus.

### Tīrākais risinājums problēmām ar null

Null vērtība ir mulsinājusi valodu izstrādātājus jau vairākus gadu desmitus. Jau ilgu laiku „labākais” risinājums problēmai bija ML vai Haskell stipa Maybe tips, kurš piedāvāja tipa drošību un novērsa „null-pointer” kļūdu. Šis risinājums strādā pietiekami labi valodās bez apakštipizācijas, taču tā nav labākā pieeja valodām ar apakštipiem un vienotiem tipiem. Ceylon valodas pieeja null vērtībai ir tikpat droša, taču kopā ar plūsmjūtīgo tipizāciju tā ir draudzīgs priekš lietotāja un neprasa atsevišķa Option-stila wrapper objekta overhead vērtību. Atšķirībā no citām valodām, null nav nekas specifisks Ceylon valodas tipu sistēmā. Null vērtības ir tikai vēl viens pluss apvienoto tipu izmantošanai.

### Eleganta un spēcīgs vektoru un funkciju tipu reprezentācija

Kā vairums citu moderno valodu, Ceylon valodai arī ir raksturīgi vektora un funkciju atsauces. Taču, atšķirībā no citām valodām, Ceylon neattēlo šīs lietas kā speciālus gadījumus tipu sistēmā, nedz arī kā interfeisa tipu papildinājumu. Tā vietā eksistē tikai viena klase- Tuple (vektors) un viens interfeiss – Callable, kurš izsaka funkcijas tipus vektora tipos. Šis piegājiens ļauj abstrahēt vektoru un funkciju veidojumus un rakstīt augstākas kārtas funkcijas, kuras nav iespējams vaidot daudzās citās modernās programmēšanas valodās.

### Reified generics

Ceylon ir viena no divām valodām, kura piedāvā refied generics opciju uz JVM, kopā ar programmēšanas valoda Gosu. Ceylon ir arī vienīgā valoda, kura piedāvā refied generics uz JavaScript virtuālajām mašīnām. Bieži ir izteikti viedokļi par reified generics Java valodā un citās JVM valodās, ka reified generics piemīt slikts izpildījums, tie nav pietiekami noderīgi vai ka tie nepareizi strādā kopā ar izmaiņām un citām augstākās kārtas generics elementiem. Ceylon valoda parāda, ka vairums šo viedokļu nav pareizi.

## Ceylon mīnusi

### IDE atbalsts

Kā pirmo mīnusu uz doto laiku es gribētu izdalīt tādu momentu kā IDE priekš Ceylon. Kā arī priekš visām citām populārām JVM valodām, kā Scala, Kotlin un Groovy, Ceylon valodai izmantojas vai nu Intelij Idea vai arī Eclipse. Intelij Idea ir IDE, kuru izstrādāja JetBrains kompānija, kura arī ir Kotlin valodas veidotāja. Eclipse ir IDE, kura izveidoja IBM kompānija pašu vajadzībām, tajā pašā laikā programmas pamatkods bija pieejams publiski. Vēlāk Eclipse bija nodota citai kompānijai. Ceylon valodai un visām pārējām JVM valodām nav sava faktiska IDE, un papildinājumi, kuri ļauj strādāt ar Ceylon Eclipse un Intelij Idea programmās, nav vēl pietiekami attīstīti. Tas rada neērtības darbā ar Ceylon valodu, īpaši, ja šī valoda ir jaunums lietotājam un viņš vēl nepazīst visas Ceylon īpatnības. IDE atbalstīšanas pietiekamā līmeņa trūkuma dēļ, Ceylon izplatīšanās var kļūt grūtāka nekā par to spriež tās veidotāji. Es uzskatu, ka tas drīz tiks mainīts, taču pašlaik tas ir acīm redzams mīnuss.

### Projektu un attīstības trūkums

Es uzskatu, ka otrs mīnuss ir nopietnu projektu trūkums, kuros izmantojas Ceylon valoda. Šis mīnuss ļoti negatīvi iespaido Ceylon izplatību, cilvēki nav pārliecināti par šo valodu, viņi negrib riskēt un veidot projektus šajā valodā, jo neviens nevēlas pārrakstīt projektu sliktas valodas izvēles dēļ. Tas arī ir liels līdzekļu zudums. Lielās organizācijas ir absolūti pret riskiem un nepārbaudītām tehnoloģijām. Tādēļ ir nepieciešams, lai RedHat paši realizētu kādu masīvu projektu savā valodā, kā to izdarīja JetBrains, jo viņi taču izmantoja savu valodu Kotlin savos produktos. Kotlin ir Ceylon valodas vienaudzis, taču pareizas realizācijas dēļ, Kotlin ir populārāks un 20017. gada 17. maijā Google informēja, kaKotlin valoda ir kļuvusi par pilnībā atbalstītu Android platformā. Manuprāt, tā ir RedHat vaina viņu valodas attīstības un izplatīšanas jomās.

### Kopiena

Trešais mīnuss, kaut arī nav īpaši nopietns, es domāju izriet no otrā. Jo populārāka ir valoda, jo lielāka ir tās kopiena. Tā kā Ceylon kopiena ir relatīvi maza, atrast risinājumu kādai noteiktai problēmai būs problemātiski. No otras puses, Ceylon dod iespēju jebkurā momentā izmantot risinājumus līdzīgām problēmām Java valodā un Java klases no standarta bibliotēkām, kā arī papildus instrumentus rakstītus Java valodā. Es tomēr uzskatu to kā mīnusu, jo kādēļ izmantot citu valodu, ja vienalga vajadzēs izmantot “vecākā brāļa” Java valodas palīdzību.

### Unikalitātes trūkums

Diez vai mūsdienās var atrast unikālu programmēšanas valodu, īpaši orientētu uz veb aplikācijas servera daļas izstrādāšanu. Vienīgā, manuprāt, unikālā valoda mūsdienās ir JavaScript. Valodu, kura kādreiz bija domāta veb aplikāciju klienta daļas izstrādei, var atrast praktiski jebkādos projektos. Ceylon valoda nav unikāla un tai nav kaut kādu unikālu vai retu īpašību. Daudzas citas valodas piedāvā tādu pašu īpašību klāstu kā Ceylon valodā. Es uzskatu, ka Ceylon valoda ir nolemta palikt nepopulāra un tikt izmantota ļoti reti, jo tā ir ļoti līdzīga Kotlin valodai, kura, kā minēts agrāk, pavisam nesen kļuva par oficiālo Anroid valodu. Kotlin tāpat kā Ceylon der veb aplikāciju servera un klienta daļu rakstīšanai.

## Vai tā ir cienīga aizņemt Java vietu

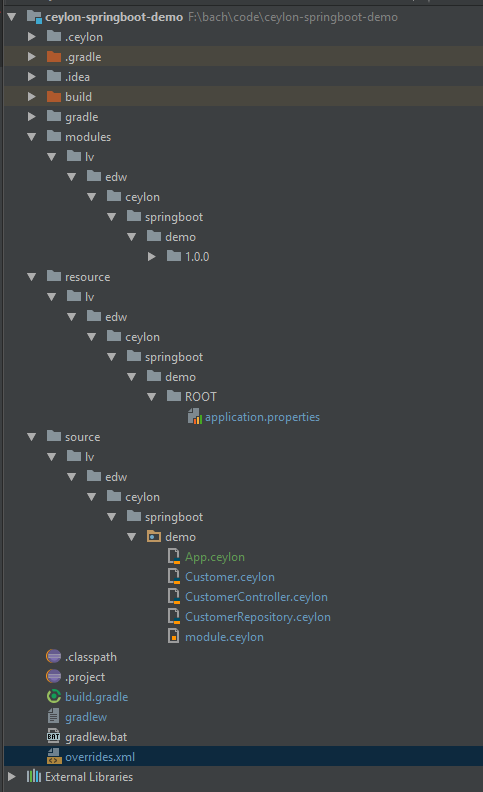
Te es gribētu izrādīt savu viedokli par to vai Ceylon valoda ir cienīgā kļūt par potenciālo substitūciju Java valodai nākotnē. Mans viedoklis ir tieši balstīts uz valodas plusiem un mīnusiem. Es uzskatu, ka, uz doto momentu, Ceylon nevarēs nomainīt Java valodu nevienā no aspektiem. No tehniskās puses valoda ir nenoliedzami labi izveidota, tai piemīt vairākas spēcīgas īpašības tehniskajā aspektā, taču tagad vairs nav tas laiks, kad tehnisko īpašību ir pietiekams, lai valoda aizņemtu tai paredzēto nišu. Mūsdienās ir nepieciešams vilināt lietotājus, ne tikai ar labu tehnisko valodas sastāvdaļu, bet, pirmkārt, ar tās vieglumu. Mūsdienās neviens negrib rakstīt milzīgas monolītas aplikācijas izmantojot sarežģītas darbības. Pašlaik programmatūras izstrādes pasaulē valda Agile, viss tiek maksimāli atvieglots. Es uzskatu, ka Ceylon valoda ir zaudējusi cīņā par popularitāti un izmantojamību savam vienaudzim Kotlin, tā kā Kotlin tagad ir kļuvis par oficiālo Android platformas valodu. Kotlin valodu izmanto tādas firmas un produkti kā Uber, Gradle, Coursera u.c. Pati populārākā mūsdienu struktūra veb aplikācijas servera daļai Java valodai ir Spring struktūra no Pitoval. Šī kompānija sola pilnveidīgu atbalstu Kotlin valodai no versijas 5.0, kura nāks klajā tuvākajā laikā.

## Prakstiskas pielietošanas piemērs

Šajā nodaļā es gribētu nodemonstrēt Ceylon izmantošanu kopā ar Spring, vebaplikācijas servera daļas veidošanai. Šī mazā aplikācija ļauj rakstīt un lasīt datus no datubāzes REST specifikācijā.

### Projekta struktūra

Rādot projekta struktūru, gribētos pieminēt, ka Ceylon projekta struktūra īpaši neatšķiras no Java, tā kā Ceylon ir sava veida pēctecis. Projekts tika sakopots, izmantojot Gradle, un kā darba IDE tika izmantots Intelij Idea.



4.4.1 att. Projekta struktūra

Pirmās trīs mapes ir individuālas katrai no ierīcēm un tām piemīt punkts pirms nosaukuma. Šīs mapes tiek automātiski izveidotas katrai ierīcei, uz kuras tiks palaists dotais projekts, parasti tās nepievieno pamatkodam versiju kontroles sistēmās. Pārējās mapes:

* Build – no nosaukuma ir saprotams, šajā mapē atrodas nokompilētie faili ar visām attiecībām. Parasti šī mape arī netiek pievienota versiju kontroles sistēmās, tā kā tās kontents tiek mainīts pēc katras projekta izveides.
* Gradle – šajā mapē atrodas Java arhīvs- Jar, kopā ar projekta veidotāju un viņa konfigurācijas failu. Tas tiek darīts tādēļ, lai varētu veidot projektu, neinstalējot veidotāju(Gradle), bet gan izmantojot šo Jar, tā saucamo wrapper.
* Modules – šī ir unikāla Ceylom mape, kurā atrodas divi arhīvi un divas hash atslēgas, lai pārbaudītu arhīvu derīgumu. Pirmajā arhīvā atrodas nokompilētie bait-kodā .class faili ar pamatkodu un mape ar metadatiem. Otrajā arhīvā atrodas nenokompilētie faili ar pamatkodu.
* Resource – mape, kurā atrodas projekta resursi un konfigurācijas faili, piemēram, Spring struktūras konfigurācijas fails- application.properties.
* Source – no nosaukuma var saprast, ka šajā mapē atrodas pamatkods, ar kuru notiek visas darbības un kurš ir sadalīts moduļos.
* .classpath/.project – tie ir konfigurācijas faili IDE Eclipse izmantošanai, bez kuriem projektu nebūs iespējams palaist dotajā IDE.
* Build.gradle – ir projekta veidotāja konfigurācija, kurā atrodas paša veidojuma, projekta atkarību, plaginu, u.c. elementu konfigurācija.
* Gradlew/gradlew.bat – tie ir faili, kuri pieder projekta veidotāja wrapper elementam, kurš atrodas Gradle mapē.
* Overrides.xml – Maven[[20]](#footnote-20) moduļa aprakstītāji ir nepārbaudāmi, un daudziem Maven moduļu aprakstītājiem trūkst informācijas par tiešajām atkarībām. Šie moduļi kompilējas un darbojas Maven izmantošanas laikā, nejaušības dēļ, izmantojot tās klašu atrašanās vietu, saturošo visas transitīvās moduļa atkarības. Kontrastam, Ceylon piemīt ClassLoader izolācija, kurai ir nepieciešama pareiza un pilnīga atkarību informācija. Maven arī neatbalsta koplietošanas moduļu importēšanas jēdzienu, tātad, ja modulis A veidos redzamus saviem lietotājiem tipus no sava importētā moduļa B, B importēšanai jābūt kopīgotai. Diemžēl, Maven moduļi bieži dod piekļuvi lietām, kurām parasti nav jābūt redzamām, piemēram, citi moduļi vai testi, kurus gribētos izslēgt no redzamo elementu skaita, kas norāda, ka veidot kopīgotas atkarības nav efektīvi. Visbeidzot Maven atbalsta moduļu versiju konflikta risināšanu ar dizaina, veiksmes vai override palīdzību, kamēr Ceylon piemīt strikta moduļu versiju importēšana.

Vienīgā atšķirība tieši tādam pašam projektam rakstītam Java valodā ir tikai module mape, jo viss nokompilētais Java projekts atradīsies build mapē.

### Module

Šis ir obligāts fails katrā modulī, rakstītam uz Ceylon. Šis fails apraksta kādas bibliotēkas un atkarības ir nepieciešamas dotajam modulim.

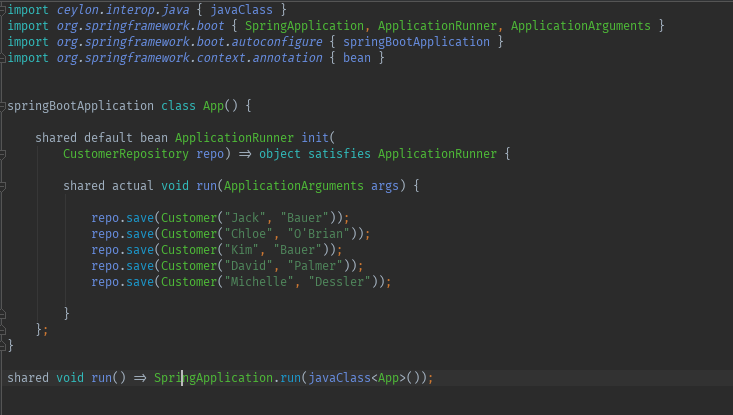


4.4.2 att. Module.ceylon

Attēlā 4.4.2 ir redzams dotā faila piemērs. Tā kā Ceylon var tikt nokompilēts ne tikai bait kodā priekš JVM, bet arī Javascript veidā, pirmais, ko norāda šī faila konfigurācijas ir, kādā valodā kods tiks translēts. Kā redzams piemērā, dotais modulis tiks nokompilēts JVM bait kodā. Tālāk tiek norādīts moduļa nosaukums, ieskaitot pilno atrašanās vietu, un moduļa versiju. Pēc atkarību norādīšanas, pelēkās krāsa komentāri, parasti tās visas dala kategorijās, lai zinātu, kur meklēt specifisku bibliotēku, un lai uzlabotu projekta strukturāciju, jo lielos projektos tiek izmantots daudz lielāks bibliotēku skaits, nekā manā pieticīgajā piemērā.

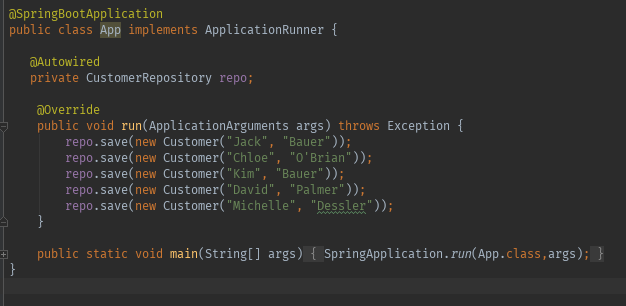
### Aplikācija

Šī ir standarta klase Spring Boot struktūrai, klase ieiešanai aplikācijā.



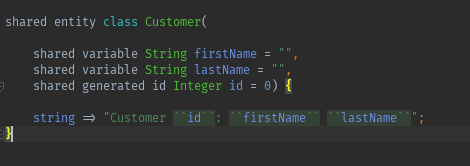
4.4.3 att. **App Ceylon**

Attēlā 4.4.3 ir parādīta klase, kura saucas App, virs kuras atrodas anotācija ”springBootApplication”, kas norāda struktūrai, kura klase darbojas kā ieeja aplikācijā. Tālāk notiek piecu Customer klases objektu veidošana un ierakstīšana datu bāzē, izmantojot Spring īpašības.



4.4.4 att. **App Java**

Kā redzams attēlā 4.4.4, šī pati klase Java valodā izskatās līdzīgi, taču, piemēram, anotācija “@SpringBootApplication” izskatās sarežģīta un grūti lasāma. Tas pats arī attiecas uz run metodi, kurš tiek izmantots, lai ierakstītu objektus datubāzē. IDE atbalsts ir Ceylon mīnuss, kura dēļ tā kods izskatās grūtāk lasāms, kā, piemēram, šajā gadījumā.



4.4.5 att. Сustomer Ceylon

### Сustomer

Šī klase ir aplikācijas “būtība”. Tā veb izstrādājumu pasaulē ir pieņemts saukt klases, kuras kalpo kā modeļi rakstīšanai un lasīšanai no datubāzēm.

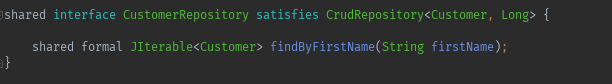
Kā redzams attēlā, tā ir parasta klase – datubāžu modelis, kurš satur trīs laukus: vārdu, uzvārdu un id numuru, ar kura palīdzību to var atrast un kurš ir unikāls. Kā redzams attēlā 4.4.6 Java variants izskatās daudz lielāks nekā tā alternatīva Ceylon valodā divu iemeslu dēļ. Pirmkārt, konstruktori, kuri, atšķirībā no Ceylon, ir nepieciešami Java izpildījumā. Otrkārt, lielais šablona koda daudzums, piemēram, getters[[21]](#footnote-21) un setters [[22]](#footnote-22) klašu laukiem, kas, manuprāt, jau skaitās kā novecojuši OOP likumi, kuri pielietojami, šādu lauku trūkumu gadījumā klasēm ar redzamības modifikāciju public. Šablona koda trūkums Ceylon valodā, parāda to izdevīgā gaismā, īpaši korporatīvas veb izstrādes jomā, kas, pēc autoru vārdiem, arī skaitās tā izstrādes mērķis. Protams, Java valodai ir liels daudzums specifisku bibliotēku, kas palīdz izvairītie no šablona koda, taču daudz patīkamāk ir, kad šo opciju piedāvā pati valoda.



4.4.6 att. Customer Java

### Repository

Repozitorija jeb glabātuve – ir pieņemtais to klašu nosaukums, kuras veic darbības ar datubāzēm. Pateicoties Spring tā izskatās ļoti vienkārši.

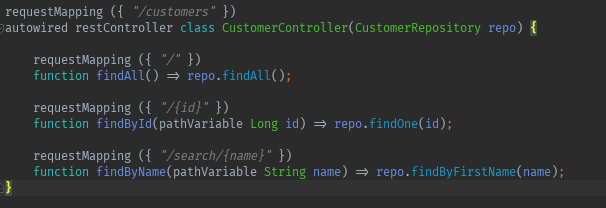


4.4.7 att. Glabātuve

Kā redzams attēlā 4.4.7, tas ir interfeiss, kurš manto citu interfeisu no Spring struktūras, kurš maksimāli atvieglo darbu ar datubāzēm mūsdienu veb aplikācijās. Java versijā šī klases izskatās identiski.

### Controller

Controller – ir klase, kura nosaka datu pieņemšanu un nosūtīšanu no aplikācijas pārējā pasaulē, izmantojot HTTP[[23]](#footnote-23). Dotajā piemērā tā izmanto REST specifikāciju, kura nodod un pieņem datus JSON[[24]](#footnote-24) formā.



4.4.8 att. **Controller Ceylon**

Kā redzams attēlā 4.4.8 visu darbu izpilda struktūra, kura tiek pieslēgta ar modifikatoru palīdzību: “requestMapping”,”autowired” un “restController”. Gadījumā ar Java, tie nav modifikatori, bet gan anotācijas, toties Ceylon valodā anotācijas neeksistē, tādēļ tie tiek deklarēti kā modifikatori.



4.4.9 att. **Controller Java**

Salīdzinot Ceylon variantu ar Java, pirmais izskatās kodolīgāks, tāpat kā situācijā ar Customer būtību. Tomēr, manuprāt, tas izskatās grūtāk lasāms. Tieši šeit parādās divas vissvarīgākās nianses korporatīvai veb izstrādei. Lasāmība un kodolīgums. Šajās lietās ir svarīgi atrast līdzsvaru. Ņemot vērā IDE atbalstu, Java kods izskatās lasāmāks.

# Secinājumi

Ceylon valoda noteikti ir uzmanības cienīga. Tomēr es uzskatu, ka pašlaik tā nevar konkurēt ar Java. Šis viedoklis ir balstīts uz vairāku faktoru sarakstu.

Sākšu ar tehniskajiem faktoriem, to ir pietiekami liels daudzums, taču praktiski visiem ir alternatīvas Java valodā, kaut arī mazattīstītas, vai arī tie nav izšķirošie. Protams, tādas īpašības kā null-safe, alias un agrāk apskatītie tipi, būtu ļoti noderīgi Java valodā, taču, sakarā ar atgriezenisko savienojamību, Java vienkārši nevar mainīt savu arhitektūru un pievienot tai šīs īpašības. Dažas no Ceylon īpašībām var tikt izmantotas Java dažādu bibliotēku vai struktūru veidā, taču tas neattiecas uz lielāko daļu īpašību. Ceylon tehniskās iespējas apstiprina tā veidotāju teikto, ka Ceylon tika veidota kā valoda lieliem projektiem un komandām. Ne tehniskās īpašības arī iespaido valodas pieprasījumu tirgū un ieinteresētību tajā. Pati galvenā no tām ir jebkādu lielu un nopietnu projektu trūkums, kuri būtu rakstīti Ceylon valodā. Otra problēma ir struktūru trūkums, kuras ir rakstītas Ceylon valodai. Jā, Spring tulkotājs ir izpildīts labi, toties ar to nepietiek pārliecībai šajā valodā. IDE atbalstam iekš Intelij Idea, vispopulārākajai IDE priekš Java, ir nepieciešami uzlabojumi, kas rod papildus grūtības darbam ar Ceylon, salīdzinot Ceylon ar modernākām JVM valodām, piemēram Kotlin vai Scala. Atgriezeniskās saderības dēļ Java velk sev līdzi visus arhitektūras lēmumus no 1995. gada. Scala ir funkcionāla JVM valoda, kas nozīmē, ka diez vai Ceylon varēs palielīties ar tehniskām īpašībām, kuru nav Scala valodā. Scala jau ir ieguvusi popularitāti daudzās jomās. Tai ir nopietnas struktūras, tādas kā Apache Spark, Play Framework 2.0. Pat Twitter platforma izmanto Scala, un Evolution Gaming kazino pārveda visu savu koda bāzi uz šo valodu, jo, būdama funkcionāla, tā mērogojas vieglāk, kas ir nepieciešams sistēmām ar lielu ikdienas lietotāju skaitu. Kotlin valodai, kuru var uzskatīt par Ceylon brāli, jo tika veidots tajā pašā 2011. gadā, piemīt tieši tādas pašas tehniskās īpašības kā Ceylon, taču popularitātes jomā, jau sen ir priekšā. Kotlin ir nodrošināta ar labāko IDE atbalstu un reāliem projektiem, kuri izmanto šo valodu, un, protams, Google lēmumu, padarīt Kotlin par pirmās klases valodu priekš Android platformas. Var secināt, ka Ceylon nav slikta valoda, taču pašlaik nav vajadzības tā izmantošanā. Tas ir tādēļ, ka neskaitot JVM eksistē vairums jaunu un ne pārāk valodu, kuri ir jau stabilizējušies savā nišā, vai tikai sāk kļūt populāri. Veb aplikāciju rakstīšanai var izmantot Google Go- jaunā valoda no Google, PHP, JavaScript, C# ar .NET platformu no Microsoft. Iespējams kādreiz Ceylon atradīs savu nišu un lietotājus, tomēr, kā jau minēts iepriekš, pašlaik nav nepieciešamības izmantot Ceylon valodu.

# izmantotās literatūras SARAKSTA sakārtoŠANA un atsauces

Diakopoulos, N., Cass,S. Interactive: *The Top Programming Languages 2016* [online].

IEEE SPECTRUM, 2016 [skatīts 2017. g. 3. majs.]. Available from: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

Finnegan, M. 10 up-and-coming programming languages developers should get to know [online].

Tech World, 2017 [skatīts 2017. g. 4. majs.]. Available from:

http://www.techworld.com/picture-gallery/apps/10-up-coming-programming-languages-for-developers-get-grips-with-3621455/

Raoul-Gabriel, U. Alternative Languages for the JVM [online].

Java Magazine 2014 [skatīts 2017. g. 5. majs.]. Available from:

http://www.oracle.com/technetwork/articles/java/architect-languages-2266279.html

Ja darbā ir atreferēts, citēts vai citādi tieši vai netieši izmantots materiāls no kādas grāmatas vai cita avota, tad tekstā ir jāatsaucas uz šo avotu. Darba autora pienākums ir precīzi norādīt informācijas avotu un autoru. Pareizi noformēta atsauce informē lasītāju par to, kādus avotus autors izmantojis rakstot darbu, un palīdz lasītājam sameklēt citēto materiālu. Atsauču noformēšanas pamatā ir divi elementi – atsauce tekstā un izmantotās literatūras saraksts jeb bibliogrāfija. Pastāv dažādas sistēmas (stili) bibliogrāfijas un atsauču noformēšanai, precīzāk par tām minēts RTU ITI noformēšanas norādījumos[[25]](#footnote-25).

Noslēguma darbos tiek lietojami šādi ieteikumi izmantotās literatūrassaraksta veidošanai un atsauču noformēšanai:

* Literatūras avotus apraksta **oriģinālvalodā** un tos sarindo alfabēta secībā pēc autora uzvārda / nosaukuma, ja autors nav minēts.
* Sarakstu kārto alfabēta secībā – **vispirms** uzrāda norādes **latīņu** cilmes valodās (latviešu, angļu, vācu utt. – latīņu alfabēta secībā) un pēc tam – norādes **slāvu** cilmes valodās (slāvu alfabēta secībā).
* Literatūras avotiem jābūt **numurētiem**.
* Literatūras noformēšanas **paraugi** sniegti 3. pielikumā.
* ***Izmantotās literatūras sarakstu*** ievieto darba beigās pēc „Rezultātiem un secinājumiem” un pirms pielikumiem.
* Uz katru no literatūras sarakstā minētajiem avotiem pamattekstā jābūt atsaucei. To izdara, tekstā aiz attiecīgā materiāla **kvadrātiekavās** ierakstot literatūras kārtējo numuru, piemēram, [1], [2], [2, 6] vai [3–5].
* Literatūras avota numurs kvadrātiekavās ir neatņemama teikuma daļa, un tam jāatrodas **pirms punkta** teikuma beigās.
* Veidojot literatūras sarakstu, tos ieteicams numurēt automātiski, lai izmaiņu gadījumā saglabātos pareiza secība, kā arī tiktu pareizi atjaunotas atsauces. Veidojot atsauces – jāveido *Cross Reference* uz numuru (*numbered item*) [[26]](#footnote-26), kas dotajā brīdī norāda konkrēto literatūras avotu (*Paragraph Number*), līdz ar to, mainoties literatūras sarakstam, automātiski mainīsies arī atsauces. Piemēram: teksts no literatūras avota A [1]; teksts no žurnāla B [3].
* Ja tabulas vai attēli tiek ņemti (pat ar savu nosaukumu) no kāda literatūras avota bez izmaiņām, tad obligāti atsauce tiek pievienota tabulas vai attēla nosaukuma beigās šādā veidā: [4, 56. lpp], ja literatūras avotā ir zināma attiecīgā objekta izvietošanas lappuse, vai parasti [3], piemēram, no tīmekļa resursiem, kur lappuses var būt nezināmas.

# PIELIKUMI

Dažādus palīgmateriālus, kas neiekļaujas darba pamatsaturā, pievieno darbam kā pielikumus ar kopīgu virsrakstu **PIELIKUMI** uz atsevišķas lapas (horizontāli un vertikāli pa vidu).

Katru pielikumu **sāk ar jaunu lapu**, lapas labajā augšējā stūrī uzrādot tā kārtas numuru, piemēram: 1. pielikums, 2. pielikums utt. Zem šī uzraksta, nākamās rindiņas vidū, raksta pielikuma nosaukumu. Pielikuma nosaukumu raksta ar izceltiem burtiem (**Bold**, lieliem vai maziem) simetriski tekstam.

* Ja pielikumā ir lietderīgi pievienot kādu svešvalodā izstrādātu informatīvi normatīvo materiālu, tad to nav nepieciešams tulkot valsts valodā (drīkst pievienot oriģinālvalodā).
* Tekstā attiecīgā vietā jādod **atsauce** uz pielikumu, piemēram: 1. pielikumā pievienoti dati par...
* Ja darbam ir viens pielikums, tad kopīgo virsrakstu PIELIKUMI neraksta un pielikumam numuru nepiešķir.
* Pielikumu virsraksti ir veidoti tā, lai tie saturā parādītos kā otrā līmeņa virsraksti. Papildus tie satur arī apslēptus laukus, kas nodrošina pareizu ilustrāciju numerāciju. Līdz ar to pielikuma virsrakstus nepieciešams kopēt un mainīt tikai pielikuma nosaukumu.
* Tā kā ilustrāciju numerācija nedaudz atšķiras, tad arī šajā gadījumā jāizmanto 2. pielikuma sniegtie piemēri, kurus iespējams nokopēt un mainīt saturu. Savukārt atsauču ievietošana neatšķiras no parastās.

rezultāti un SECINĀJUMI

Izmantotās literatūras saraksts

1. Literatūras avots A
2. Publikācija B
3. Žurnāls C

pielikumi

1. pielikums. Noslēguma darba vāks

|  |
| --- |
| **RĪGAS TEHNISKĀ UNIVERSITĀTE** |
| **<NOSLĒGUMA DARBA VEIDS>**  (no saraksta: bakalaura darbs, maģistra darbs, diplomdarbs, diplomprojekts, inženierprojekts, kvalifikācijas darbs) |
| RĪGA <20….> |

2. pielikums. Pielikumā iekļauto objektu noformēšanas paraugi

Pielikumā ievietoto objektu noformējums atšķiras no pamatsaturā ievietojamajiem tikai ar „P” burtu, kas liekams attiecīgā objekta numura priekšā. Zemāk ievietotos objektus jāizmanto kā paraugs saviem objektiem (jauna objekta ievietošana veicama ar kopēšanas palīdzību, nomainot attiecīgo saturu un virsrakstus).

P2.1. attēlā ir skatāms ...



P2.1. att. Attēla nosaukums

P2.1. tabulā ir redzami dati, kas atspoguļo...

P2.1. tabula

Tabulas nosaukums

| Nr. | Vērtība | Procenti | Anuitāte |
| --- | --- | --- | --- |
| 1. | ... |  |  |
| 2. | ... |  |  |

P2.1. tabulas dati atspoguļo...

No formulu iekļaušanas pielikumā labāk izvairīties, jo jebkādas matemātiskas izteiksmes var būt izskatītas kā neatņemama daļa teorētisko algoritmu izklāstam un tiem jābūt ievietotiem darba pamatdaļā.

3. pielikums. Atsauču noformēšana

**LVS ISO 690:2010** *[adaptēts]*

**1. ZINĀTNISKĀ MONOGRĀFIJA / MĀCĪBU GRĀMATA / LEKCIJU KONSPEKTS**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** *Nosaukums oriģinālvalodā* *(Slīprakstā)***.** Izdevums**.** Daļa (sējums)**.** Izdošanas vieta**:** Izdevējs**,** gads**.** Lappušu skaits**.** Pieejams: doi**:**[[27]](#footnote-27)

Author**.** *Title* *(Italic[[28]](#footnote-28))***.** Edition**.** Volume (if more than one)**.** Place**:** Publisher**,** date**.** Pages**.** Available from: doi**:**

* ***Ja autoru skaits ir lielāks par četriem, tad aiz ceturtā autora jāieraksta "u.c." vai "et al."***

**Piemēri:**

Šenfelde, M. *Makroekonomika.* 4. izd. Rīga: RTU izdevniecība, 2012. 244 lpp.

Barkans, J., Zalostiba, D. *On the Global Climate Change*. Riga: RTU Publishing House, 2010. 82 p.

Djukendžijevs, J. *Cilvēka balsta-kustību un manipulāciju aparāta protezēšana*. 1. daļa, 2. sēj., 6. grām. Rīga, 2000. 164 lpp.

Platais, I., Graudiņš, P. *Gāzapgāde.* 2. daļa. Dabasgāzes gāzapgādes sistēmu izveide, ierīkošana un apkalpe. Rīga: RTU, 2008. 219 lpp.

Маталин, А.А. *Технология машиностроения*. Ленинград: Машиностроение, 1985. 512 с.

**2. PROMOCIJAS DARBS**

Autora uzvārds**,** Vārds**.** *Nosaukums oriģinālvalodā* *(slīprakstā)***.** Promocijas darbs**.** Izdošanas vieta**:** Izdevējs**,** gads**.** Lappušu skaits**.**

**Piemērs:**

Poļaka, Inese. *Klašu blīvuma struktūras izmantošana lēmumu koku klasifikatoru ansambļu evolucionārajā indukcijā*. Promocijas darbs. Rīga: RTU, 2014. 141 lpp.

**3. PROMOCIJAS DARBA KOPSAVILKUMS**

Autora uzvārds**,** Vārds**.** *Nosaukums latviešu valodā* *(slīprakstā)***.** Promocijas darba kopsavilkums**.** Izdošanas vieta**:** Izdevējs**,** gads**.** Lappušu skaits**.**

Author**.** *Title of the theses* *(Italic)***.** Summary of Promotion Thesis**.** Place**:** Publisher**,** date**.** Pages**.**

**Piemēri:**

Poļaka, Inese. *Klašu blīvuma struktūras izmantošana lēmumu koku klasifikatoru ansambļu evolucionārajā indukcijā*. Promocijas darba kopsavilkums. Rīga: RTU Izdevniecība, 2014. 38 lpp.

Poļaka, Inese. *Evolutionary Induction of Decision Tree Classifier Ensembles using Class Density Structure*. Summary of Promotion Thesis. Riga: RTU Press, 2014. 37 p.

**4. RAKSTS KONFERENČU TĒŽU KRĀJUMĀ /  
RAKSTS PILNA TEKSTA KONFERENČU RAKSTU KRĀJUMĀ**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** Raksta nosaukums oriģinālvalodā**. No:** *Krājuma un konferences nosaukums***,** *Valsts***,** *Pilsēta***,** *Datums* *(slīprakstā)***.** Daļa (sējums)**.** Izdošanas vieta**:** Izdevējs**,** gads**,** lappuses**.** Pieejams: doi**:**

Author**.** Title of the conference paper**.** **In:** *Title of the conference Proceedings/Abstracts book (incl. conference name)*, *Place and date of conference* *(Italic)***.** Volume (if more than one)**.** Place**:** Publisher**,** date**,** page numbers**.** Available from: doi**:**

**Piemēri:**

Buliņš, Z., Šitikovs, V. Programmatūras paplašināšana, izmantojot MySQL piedāvātās iespējas. No: *Lietišķās datorsistēmas: 52. RTU studentu zinātniskās un tehniskās konferences rakstu krājums, Latvija, Rīga, 2011. g. aprīlis.* Rīga: RTU Izdevniecība, 2011, 84.−91. lpp.

Zicans, J., Kalnins, M., Bledzki, A.K., Jablonskis, I., et al. Tensile Properties of Irradiated Binary Heterogeneous Blends Based on Poly (ethylene terephtalate) and Polyethylene. In: *Materials Engineering & BALTTRIB* − *2001: Materials of the X–th International Baltic Conference*, *Latvia, Jurmala, 27*−*28 September 2001*. Riga: RTU Publishing House, 2001, pp. 120−121.

**5. PUBLIKĀCIJA ZINĀTNISKO RAKSTU KRĀJUMĀ**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** Raksta nosaukums oriģinālvalodā**. No:** *Krājuma nosaukums* *(slīprakstā)***.** Daļa (sējums)**.** Izdošanas vieta**:** Izdevējs**,** gads**,** lappuses**.** Pieejams: doi**:**

Author of the article**.** Article title**. In:** *Title of the book (Italic)***.** Volume (if more than one)**.** Place**:** Publisher**,** date**,** page numbers**.** Available from: doi**:**

**Piemēri:**

Zigmunde, A., Ķestere, I. Latvijas Universitātes Pedagoģijas nodaļas pirmsākumi, studiju process, mācībspēki un studenti. No: *Pedagoģijas vēsture: 15 jautājumi: Zinātnisko rakstu krājums*. Rīga: RaKa, 2010, 176.−203. lpp.

Počs, R. Regulations and Requirements for Development of Promotion Theses in Latvia. In: *Overcoming the Hindrance in Writing Doctoral Theses: Collection of Scientific Articles*. Riga: RTU Publishing House, 2009, pp. 7−25.

**6. NODAĻA ZINĀTNISKAJĀ MONOGRĀFIJĀ**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** Raksta nosaukums oriģinālvalodā**. No:** Monogrāfijas autora vai redaktora uzvārds**,** Iniciāļi**.** *Monogrāfijas nosaukums* *(slīprakstā)***.** Izdevums**.** Daļa (sējums)**.** Izdošanas vieta**:** Izdevējs**,** gads**,** lappuses**.** Pieejams: doi**:**

Author of chapter**.** Chapter title**. In:** Author or editor of book**.** *Title of book* *(Italic)***.** Edition**.** Volume (if more than one)**.** Place**:** Publisher**,** date**,** page numbers**.** Available from: doi**:**

**Piemēri:**

Ketners, K. Nodokļu evolūcija. No: Krastiņš, A., Andrējeva, V., Ketners, K. *Ievads nodokļu administrēšanas specialitātē*. Rīga: RTU Izdevniecība, 2007,   
10.−16. lpp.

Merkuryev, Yu., Burinskiene, A., Merkuryeva, G. Warehouse Order Picking Process. In: Yu. Merkuryev, G. Merkuryeva, eds. *Simulation-Based Case Studies in Logistics: Education and Applied Research.* London: Springer, 2009,   
pp. 147−165. Available from: doi: 10.1007/978-1-84882-187-3\_9.

**7. PUBLIKĀCIJAS ZINĀTNISKAJOS ŽURNĀLOS**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** Raksta nosaukums oriģinālvalodā**.** *Žurnāla nosaukums* *(slīprakstā)***.** Izdošanas gads**,** sējums (numurs)**,** lappuses**.** Pieejams: doi**:**

Author**.** Article title**.** *Journal title* *(Italic)***.** Date**,** volume number (issue)**,** page numbers**.** Available from: doi**:**

**Piemēri:**

Krēsliņš, A., Borodiņecs, A. Dzīvojamo ēku ventilācijas sistēmas. *Latvijas Būvniecība*. 2010, Nr. 1, 38.−39. lpp.

Haritonovs, V., Smirnovs, J., Naudžuns, J. Prediction of Rutting Formation in Asphalt Concrete Pavement. *The Baltic Journal of Road and Bridge Engineering*. 2010, vol. 5, no. 1, pp. 38−42. Available from: doi: 10.3846/bjrbe.2010.05.

* ***Iespējami 2 varianti numerācijas un lpp. atveidošanā:***

***2010, vol. 5, no. 1, pp. 38***−***42.***

***2010, 5(1), 38***−***42.***

**8. PUBLIKĀCIJAS IZDEVUMA „RTU ZINĀTNISKIE RAKSTI” ŽURNĀLOS**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** Raksta nosaukums oriģinālvalodā**.** *Žurnāla nosaukums* *(slīprakstā)***.** Izdošanas gads**,** sējuma numurs**,** lappuses**.** Pieejams: doi**:**

Author**.** Article title**.** *Journal title* *(Italic)***.** Date**,** volume number**,** page numbers**.** Available from: doi**:**

**Piemēri:**

Berziša, S., Grabis, J. Projekta fāžu atkarīga projektu vadības informācijas sistēmu konfigurācija. *Informācijas tehnoloģija un vadības zinātne.* 2012, 15. sēj., 105.−110. lpp. Pieejams: doi:10.2478/v10313-012-0011-x (angļu valodā).

Kirshners, A., Polaka, I., Aleksejeva, L. Gastric Cancer Risk Analysis in Unhealthy Habits Data with Classification Algorithms.  Information Technology and Management Science. 2015, vol. 18, pp. 97−102. Available from: doi:10.1515/itms-2015-0015.

**9. TĪMEKĻA RESURSI**

Autora uzvārds**,** Iniciāļi**.,** Nākamā autora uzvārds**,** Iniciāļi**.** *Nosaukums oriģinālvalodā* *(slīprakstā)* [tiešsaiste]**.** Izdevējs**,** gads [skatīts 2014. g. 21. febr.]. Pieejams: <URL>

Author**.** *Title* *(Italic)* [online]**.** Publisher**,** date [viewed 21 February 2014]. Available from: <URL>

* ***Bieži nav iespējams noteikt izdošanas vietu un izdevēju, tādēļ šie elementi ir fakultatīvi.***

**Piemēri:**

Ribickis, L. *VIEDIE TĪKLI – jaunās tehnoloģijas drošai elektroapgādei* [tiešsaiste]. Dienas Bizness, 2011 [skatīts 2014. g. 21. febr.]. Pieejams: http://konferences.db.lv/wp-content/uploads/2011/12/4\_Ribickis.pdf

Janusevskis, J., Le Riche, R. *Simultaneous Kriging-Based Sampling for Optimization and Uncertainty Propagation* [online]. CCSd, 2010 [viewed   
21 February 2014]. Available from: http://hal.archives-ouvertes.fr/hal-00506957

**10. LATVIJAS PATENTI**

PATENTA PIETEICĒJS vai ĪPAŠNIEKS**.** *Nosaukums (slīprakstā)***.** Vārds [*vai* Iniciālis] Uzvārds**,** Vārds [*vai* Iniciālis] Uzvārds (izgudrotāji)**.** Int. Cl.: [Starptautiskās klasifikācijas indekss]**.** Iesniegšanas datējums [gggg-mm-dd]**.** Patenti un Preču Zīmes [Avota nosaukums]**.** Patenta numurs ar valsts kodu**.** Publicēšanas datējums [gggg-mm-dd]**.**

**Piemērs:**

RĪGAS TEHNISKĀ UNIVERSITĀTE. *Zemtemperatūras keramika ar hidrauliskām īpašībām*. Laimonis Bīdermanis, Linda Krāģe, Andris Cimmers, Lauma Lindiņa, Ingunda Šperberga, Laimons Timma (izgudrotāji). Int. Cl.: C04B33/00. Iesniegšanas datējums 2012-06-01. Patenti un Preču Zīmes. LV14562B. 2013-01-20.

Darba izpildes un novērtējuma lapa

Es, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, bakalaura studiju programmas „Informācijas tehnoloģija” III RDBI0\_\_ grupas students(e), ar savu parakstu apstiprinu, ka esmu izstrādājis(-usi) doto bakalaura darbu, kas iesniegts Rīgas Tehniskajā universitātē inženierzinātņu bakalaura grāda datorvadībā un datorzinātnē iegūšanai.

Bakalaura darbs ir izpildīts pilnīgi patstāvīgi un satur visas nepieciešamās atsauces uz darbā izmantotajiem materiāliem.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Bakalaura darbs izstrādāts \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ katedrā

Darba autors: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_.g. \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Zinātniskais vadītājs: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_.g. \_\_\_. \_\_\_\_\_\_\_\_\_\_\_

Bakalaura darbs pielaists aizstāvēšanai:

\_\_\_\_\_\_\_ katedras vadītājs: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 201\_.g. \_\_\_ .\_\_\_\_\_\_\_

Bakalaura darbs aizstāvēts Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas 201\_. gada \_\_\_.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ sēdē, protokola Nr. \_\_\_\_\_ un novērtēts ar atzīmi \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Informācijas tehnoloģijas institūta bakalaura darbu aizstāvēšanas komisijas sekretāre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Dr.math. V.Minkēviča/

1. Null – Tukšuma apraksts programmēšanu valodās [↑](#footnote-ref-1)
2. Backward compatibility – programmu un interfeisu eksistēšana jaunākos datoru modeļos, kutas tika izmantotas vecākos modeļos, kas atvieglina programmu(vai cilvēku) darba turpināšanu bez drastiskas pārmācīšanās. [↑](#footnote-ref-2)
3. JVM – Java Virtual Machine veic bait kodu Java, kurš tika kompilēts no Java valodas koda. JVM var arī veikt darbības ar programmām, kuras tiek rakstītas citās programmēšanas valodās. [↑](#footnote-ref-3)
4. SDK – software development kit rīku komplekts, kurš ļauj speciālistiem veidot aplikācijas noteiktam programmu veidam [↑](#footnote-ref-4)
5. Backend – Datora programmas vai sistēmas daļa, kura strādā ar datiem un kurai lietotājam nav piekļuves. [↑](#footnote-ref-5)
6. Frontend – Datora sistēmas daļa vai aplikācija, kurai lietotājam ir tieša piekļuve. [↑](#footnote-ref-6)
7. Фреймворк – programmas platforma, kura nosaka programmas sistēmas struktūru; programmatūra, kura atvieglo dažādu liela projekta komponenšu izstrādi un apvienošanu. [↑](#footnote-ref-7)
8. Open-Source Software – ir datora programmatūra kopā ar tās pirmavota kodu, kura ir kļuvusi publiski atklāta un kuras veidotājs sniedz tiesības izmantot, mainīt un dalīties and šo programmatūru. Šī veida programmas var tikt veidotas publiskā manierē.  [↑](#footnote-ref-8)
9. Pattern matching – ir pārbaudes darbība, noteiktas secības žetonu pārbaudei uz dažādu modeļu klātbūtni.  [↑](#footnote-ref-9)
10. Type inference - Programmēšanā kopilatoram piemīt spēja pašam loģiski izspriest vērtību tipu. [↑](#footnote-ref-10)
11. Persistent collections - Kolekcija, kura vienmēr saglabā savu iepriekšējo versiju pēc modifikācijas. [↑](#footnote-ref-11)
12. Tuple – Tuple ir klase, kura var saturēt dažādu elementu kolekciju. [↑](#footnote-ref-12)
13. Runtime – Laika perios, kurā programma tiek izpildīta [↑](#footnote-ref-13)
14. Lisp – Programmēšanas valoda, kura parādijās 1958. gadā, zinātniska valoda, lielākoties izmantota Mākslīgā intelekta izstrādē. [↑](#footnote-ref-14)
15. IDE – Integrated Development Environment ir programmatūras aplikācija, kura nodrošina visaptverošas iespējas programmētājiem lietojumprogrammatūru izstrādē. [↑](#footnote-ref-15)
16. Meta programmēšana - programmēšana veids, kas saistīts ar programmu radīšanu, kas dod pamatu citām programmām savas izpildes rezultātā. It īpaši, pirmkoda kompilācijas laikā vai programmās, kuras maina pašas sevi izpildes laikā. [↑](#footnote-ref-16)
17. First-class Functions -  Tas nozīmē, ka valodas atbalsta funkcijas kā argumentus citās funkcijās, to atgriešanu kā rezultātu citās funkcijās, to piešķiršanu mainīgajiem vai uzglabāšanu datu struktūras. [↑](#footnote-ref-17)
18. IBM Watson – Superdators, kurš ir aprīkots ar jautājumu- atbilžu mākslīgā intelekta sistēmu Watson galvenais uzdevums - izprast jautājumus cilvēku valodā, un rast atbildes uz tiem datu bāzē. [↑](#footnote-ref-18)
19. Singleton – projektēšanas šablons, kurš garantē, ka vienprocesa aplikācijā eksistēs specifiskas klases vienīgais eksemplārs, un kurš nodrošina globālo piekļuvi šim eksemplārim. [↑](#footnote-ref-19)
20. Maven - Apache Maven ir programmatūras pārvaldīšanas un izpratnes rīks. Balstoties iz projekta objekta modeļa (POM) koncepta, Maven var pārvaldīt projekta veidošanu, ziņošanu un dokumentāciju no vienas centrālas informācijas daļas [↑](#footnote-ref-20)
21. Getter – metode, kura palīdz iegūt ne public redzamības modifikatora lauka vērtību [↑](#footnote-ref-21)
22. Setter – metode, kura palīdz nodot vērtību ne public redzamības modifikātora laukam [↑](#footnote-ref-22)
23. HTTP – HyperText Transfer Protocol – datu nosūtīšanas protokols, agrāk HTML veidā, taču tagad arī jebkuros datu veidos [↑](#footnote-ref-23)
24. JSON – JavaScript Object Notation – teksta formāts datu apmaiņai, balstīts uz JavaScript. [↑](#footnote-ref-24)
25. Norādījumi noslēguma darba noformēšanai studiju programmā „Informācijas tehnoloģija” / Izstr. L. Aleksejeva, J. Grabis, J. Merkurjevs, I. Upīte. Rīga: RTU, 2016. 20 lpp. [↑](#footnote-ref-25)
26. Lai saīsinātu numurēto ierakstu sarakstu un atvieglotu iespēju atsaukties uz literatūras avotiem, ***ieteicams izveidot iezīmes nosaukumu arī literatūras avotam***, analoģiski attēlu, tabulu vai formulu variantam. [↑](#footnote-ref-26)
27. Krāsas lietotas tikai, lai izceltu būtiskākos atribūtus; tās nav jāizmanto bibliogrāfijas noformēšanā. [↑](#footnote-ref-27)
28. *Slīprakstā* visos piemēros tiek atzīmēti ***augstākā līmeņa*** elementu nosaukumi – grāmatas, zinātniskās monogrāfijas, promocijas darba vai promocijas darba kopsavilkuma, rakstu krājuma, žurnāla, utt. nosaukumi. [↑](#footnote-ref-28)