|  |
| --- |
| 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. **Пример оператора Span**

Так же есть вариант с указанием только верхнего или нижнего значение:

[from ..] или [.. to].

* Measure – оператор похожий на спан, указывается с какого элемента и сколько элементов. [from:length]
* In оператор дает возможность проверить, находится ли в коллекции определенный элемент, пример на картинке 3.1.5



3.1.5. att. **Пример оператора in**

### Выражения

Операторы в Ceylon выглядят вполне стандартно, вряд ли можно придумать что-то кардинально новое. Присутсвтуют всем давно известные операторы такие как:

* For – стандартный оператор, имеет несколько видов, for, foreach.
* If/else – так же не измененый оператор if, имеет дополнительные проверки на тип, существование и на пустоту, ключевыми словами, is – проверка на тип объекта, exists – не является ли объект null, nonempty – проверка на не пустоту коллекции.
* Switch – никаких значимых модификаций не претерпел, вместо команды “default” как в Java/C языках используется “else”, для выбора варианта, который не описан ни одним “case” оператором.
* While – выглядит и работает как и в популярных предшественниках – Java/C
* Try/Catch/Finally – так же не является новинкой, для людей использовавших или использующих по сей день Java/C. Try – попытаться выполнить какое-то действие, catch – поймать исключение которые было брошено в процессе блока try, Finally – выполнить тело блока finally после удачного или не удачного выполнения блока try.

## ООП

В этом разедле я бы хотел поговорить об особенностях ООП в Ceylon. В Ceylon есть классы и объекты, интерфейсы. Ceylon поддерживает множесвтенное наследование, но с определенными ограничениями, Red Hat называют его “mixin” наследование. В принципе, в Ceylon реализовано наследование как в Java. Нельзя наследовать больше 1 класса, но можно наследовать, или как пишется в самом языке “удовлетворять”, несколько интерфейсов одновремменно. Есть ещё одна интересная особенность в языке Ceylon. В отличии от С и Java, в Ceylon нет необходимости создавать класс, что бы писать код.

### Класс

В Ceylon класс представляет собой структуру, отражение реального предмета, механизма или понятия. Класс может наследовать другие классы, но только в одном экземпляре, стало быть, нельзя наследовать больше одного класса, но один класс может быть наследован бесконечное число раз. На наследование нет ограничения вглубь. Класс может иметь поля – переменные относящиеся непосредственно к классу, и методы – действия которые этот класс может выполнять. Классы могут быть абстрактными, что значит, что объект этого класса не может быть создан.

3.2.1 att. **Абстрактный и обычный класс**

На картинке 3.2.1 показан пример наследования. Класс Circle, наследует абстрактный класс Figure, передавая строку “Circle” как параметр. И переопределяет метод draw. Пример работы данного кода можно увидеть на картинке 3.2.2

### Интерфейс

3.2.2 att**. Абстрактный и обычный класс – Результат**

Интерфейс – это своего рода обязательство, которое обязан выполнить класс реализующий какой-либо интерфейс. Объект интерфейса так же как и абстрактного класса не может быть создан. В Ceylon через интерфейсы реализованно множественное наследование, так как один класс может реализовывать множество интерфейсов.

Как видно на картинке 3.1.8 интерфейс TwoDimensional имеет 2 метода: square и perimeter, которые должны вернуть дробное число. Принципе такой же и с абстрактным классом, методы переопределенны в классе Circle. Так же в класс Circle можно добавить реализацию интерфейса, например отвечающего за цвет, и обязать Circle, знать какого он цвета. Пример выполнения метода draw можно увидеть на картинке 3.2.3

3.2.3 att. **Реализация интерфейса**



3.2.4 att. **Результат реализации интерфейса**

## Особенности

В этой главе хотелось бы рассмотреть интересные особенности языка. Особенности которых нет в классических Java/C языках, которые характерны только новым возможно функциональным языками программирования. В Ceylon есть такие особенности, алиасы, модульность из-за которой уже в который раз откладывают выход Java 9 c проектом Jigsaw.

### Aliases

В Ceylon можно использовать псевдонимы – изменение название чего угодно. Это может значительно улучшить читабельность кода, особенно в случае с интерфейсами или коллекциями с большим количеством дженериков.



3.3.1 att. **Псевдонимы**

Как видно на картинке 3.1.1, псевдонимы в Ceylon можно использовать двумя способами. Первый способ использовать ключевое слово alias, в таком случае этому псевдониму можно присвоить, любое значение или несколько. Под alias Num подходит целые и дробные числа – типы Float и Integer. Alias PeopleMap это просто удобное сокращение записи. На этой картинке 2 раза присутствует надпись “given Value satisfies Object” это не совсем alias, но я отнес это к этой же категории. Эта строка означает, что параметр передаваемый на место Value, обязательно должен реализовывать интерфейс Object.

### Union Types

Union Types – это возможность установить несколько разных типов параметра. Вы можете сделать метод который будет принимать на вход или String, или Integer, что дает возможность уменьшить количество кода на не нужные проверки типов входящих параметров.

Как видно на картинке 3.3.2 метод printType принимает значение val, которое в свою очередь может быть любым из 3 типов: String, Integer, Float.

3.3.2 att. **Union Types**

### Enumerated Types

Особенность языка позволяющая задекларировать, какие классы наследуют данный интерфейс или абстрактный класс. Эта особенность позволяет использовать выражение switch, а так же не позволяет классу наследовать 2 интерфейса из одной группы. К примеру, если есть интерфейс CarPart, у него есть 3 наследника-интерфейса: AudiCarPart, BmwCarPart и MercedesCarPart, то допустим класс BrakeSystem может наследовать только один из этих классов, в противном случае программа выкинет исключение.



На картинке 3.3.3 показан пример использование Switch, на интерфейс. Например в Java использование String в Switch появилось в версии 1.7, последняя версия 1.8, не говоря об интерфейсах и прочем.

3.3.3 att. **Enumerated Types**

### Imports

В Ceylon как и во многих других языках, необходимо импортировать классы, чтобы их использовать. В Ceylon есть специальные механизмы, для импорта классов с одинаковым названием, есть 2 варианта избежать полного названия класса, включая в него название всех package.



3.3.4 att. **Imports**

Как видно на картинке 3.3.4 первый вариант – это использовать alias для названия импортируемого класса, как в случае с HashMap и Map из java.util, для которыъх alias является JMap и JHashMap. В случае с HashMap из ceylon.collection мы импортируем данный класс непосредственно в метод.

### Функции

В Ceylon присутствует множество особенностей связанных с функциями присущие новым или полностью функциональным языкам. В Ceylon все функции реализуют интерфейс Callable. И первая особенность языка – это функции высшего порядка – функция которая может в качестве аргумента принимать другие фукнции.

Как видно на картинке 3.3.5 метод repeat принимает на вход 2 апргумента, первый аргумент это целое число, второй аргумент это функция которая ничего не возвращает, метод repeat вызовет переданную ему функцию введенное количество раз, передаваемся ему функия не должна принимать никаких параметров.

3.3.5 att. **Функция высшего порядка**

Так же в Ceylon существуют так называемые Curried functions. A method or function may be declared in curried form, allowing the method or function to be partially applied to its arguments.

Как видно на картинке 3.3.6 метод adder умеет складывать 2 дробных числа. Используя особенность языка Ceylon мы можем сделать функцию, которая заменяет первый аргумент на единицу, тем самым мы создали функцию которая принимает только один аргумент – дробное число и всегда прибавляет к ниму единицу.

3.3.6. att. **Curried functions**

### Аргументы

Вы можете подумать, что такого в аргументах, но в Ceylon, есть несколько особенностей работы с аргументами. Первая из них это – названные аргументы. Это лишь упрощение чтения и никакой дополнительной функциональности это не несет.

3.3.7 att. **Названные аргументы**

Как видно на картинке 3.3.7 параметры метода someMethod при вызове, написаны не как в Cиобразных языках, через запятую, а несколько по дргуому, но читается такой вариант значительно проще.

### Работа с коллекциями

Практически ничего нового Ceylon не привнес в работе с коллекциями, но выполнено это гораздно лаконичнее.



3.3.8 att. **Создание потока и последовательности**

Как видно на картинке 3.3.8 создать поток – первая строка на картинке или последовательность – вторая строка на картинке, в Ceylon можно создать используя for в декларации. Так же эту же запись можно передавать в качестве аргумента, как видно на картинке 3.3.8.

### Особенности конструктора

Помимо стандартных конструкторов таких как в Java в Ceylon есть ещё 2 вида конструкторов. Первый это Named конструктор – конструктор который имеет название и может принимать абсолютно любые параметры. Второй вариант это Value конструктор – конструктор который не принимает никаких параметров и создает объект класса лишь однажды в данном контексте, создает так называемый Singleton[[19]](#footnote-19).

3.3.9 att. **Пример использование названного конструктора**

Как видно на картинке 3.3.9 названный конструктор получается своего рода статическим методом класса.

### Взаимодействие с базами данных

В Ceylon нет специально разработонного способа работы с базами данных, но по скольку возглавляет проект Гэвин Кинг, человек который является основателем такого проекта как Hibernate. Hibernate до боли знаком программистам которые используют Java. Hibernate - библиотека предназначенная для решения задач объектно-реляционного отображения. Но по скольку Ceylon имеет возможность использова любые Java библиотеки, то проблем с взаимодействием с базами данных не будет, ведь для Java есть уже готовые решения для этого. В случае с веб разработкой, для Ceylon нет специальных библиотек и фреймворков как например для Scala. Ceylon будет служить лишь каркасом, а не полноценной экосистемой для создание серверной части веб приложений.

# OBJEKTU IEKļAUšANA TEKSTĀ

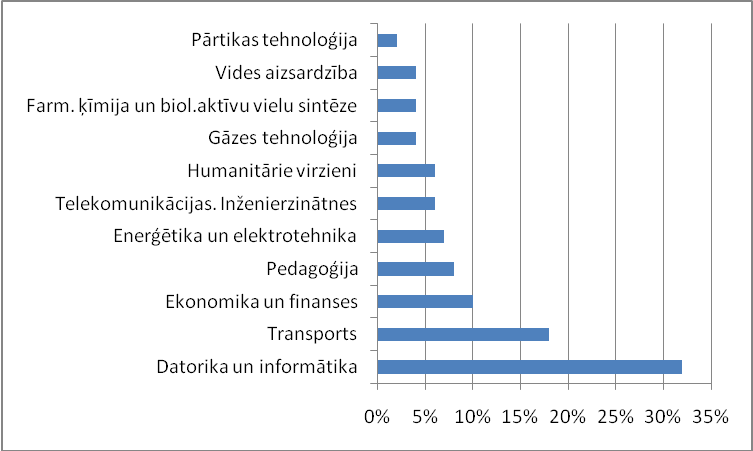
Par iekļautiem noslēguma darbā objektiem var būt uzskatīti attēli, tabulas un matemātiskās izteiksmes.

Pirmajā nodaļā tika aprakstīti sagatavošanās darbi objektu izmantošanai un automātiskai numurēšanai, skat. 1.1.2. punktu. Ja tā netika izlasīta un attiecīgie pasākumi netika veikti, tad tas ir jāizdara tagad.

Kā jau tika minēts, katru no objekta veidiem ieteicams noformēt ar eksistējošu piemēru kopēšanas palīdzību, vēlāk nomainot attiecīgās objekta saturu un virsrakstu. Atsauces ievietojamas ar opcijas *References→Cross-reference* palīdzību, no objektu saraksta izvēloties attiecīgo tipu (*Reference Type*). Lai ievietotu tikai attiecīgā objekta numuru, tad jāizvēlas opcija *Only label and number* no *Insert reference to* saraksta.

## Attēli

Darbā ievietoto ilustrāciju – fotogrāfiju, shēmu, grafiku, diagrammu un tml. – apzīmēšanai izmanto vienu un to pašu terminu "attēls". Attēli tiek numurēti nodaļas ietvaros, un katram no tiem ir jābūt savam nosaukumam. Attēla numuru un nosaukumu raksta simetriski zem ilustrācijas. Nosaukumu raksta treknrakstā ar maziem burtiem un lielo sākuma burtu, bez punkta nosaukuma beigās. Zemāk redzams 4.1. attēla piemērs.



4.1. att. **Bakalaurantu un maģistrantu stipendiju piešķīrumi pa nozarēm**

Ja attēlam nepieciešami paskaidrojumi, tos raksta zem nosaukuma centrā (paskaidrojumus var rakstīt, lietojot mazāka lieluma fontu), šajā gadījumā aiz nosaukuma liek kolu.

* Tekstā, kura ilustrēšanai izmantots attēls, attiecīgajā vietā uz to jānorāda atsauce, piemēram: “... uzskatāmi ilustrē 4.1. attēlā sniegtais zīmējums...” vai "4.1. att. sniegtais...". Tas ir viegli izdarāms ar automātisko atsauču sistēmu, par kuru minēts **Error! Reference source not found.**. nodaļas sākumā.
* Atsaucei uz ilustrāciju tekstā pirmoreiz jāparādās **pirms** attēla.
* Ievietojot atsauci, attēla **paskaidrojumam tekstā jāatšķiras no attēla virsraksta** (tas pats attiecas arī uz tabulām!).
* Attēlus ieteicams izveidot un iekļaut tekstā kā veselus objektus, proti, caur izvēlni *Insert→Object→Microsoft Visio* vai arī iekļaut attēlu no citiem grafiskajiem redaktoriem, lietojot opciju *Create from File*.
* Ja grafiskie objekti (attēli) tiek veidoti, lietojot opciju *Insert→Shapes*, tad visiem attēla elementiem jābūt sagrupētiem, lai rezultātā attēls būtu kā viens **vesels** bloks.
* Šāds attēls kā vesels bloks tiek ievietots darbā atsevišķajā rindā, piemēram, lietojot stilu Picture.
* Ja attēls (vai tabula) ir ņemts no kāda avota, tad tas jāiekļauj bibliogrāfijas sarakstā un attiecīgajā vietā obligāti jānorāda **atsauce uz oriģinālo** bibliogrāfijas avotu (pie paša attēla vai tabulas nosaukuma).

## Tabulas

Katrai darbā ievietotajai tabulai jābūt numurētai un ar savu nosaukumu. Tabulas numurē ar arābu cipariem **tikai nodaļas** ietvaros, piemēram: 1.1. tabula, 2.3. tabula, utt.

Tabulas numuru raksta labajā pusē virs tabulas nosaukuma. Tabulas nosaukumu raksta simetriski virs tabulas ar izceltiem maziem burtiem un lielo   
sākuma burtu (**Bold**, 12. lielums, novietojums − lapas **vidū**), bez punkta nosaukuma beigās.

Zemāk redzams tabulu noformēšanas paraugs ar bieži lietojamu saīsinājumu un to atšifrējumu aprakstu (4.1. tab.).

4.1. tabula

Tradicionālie saīsinājumi, kurus noslēguma darbā nav nepieciešams atšifrēt

| N.p.k. | Saīsinājums | Atšifrējums |
| --- | --- | --- |
| 1. | lpp. | lappuse |
| 2. | n.p.k. | numurs pēc kārtas |
| 3. | sk. vai skat. | skatīt |
| 4. | š.g. | šā gada; šī gada |
| 5. | t.i. | tas ir |
| 6. | u.c. | un citi |
| 7. | u.tml. | un tamlīdzīgi |
| 8. | utt. | un tā tālāk |

Tekstā attiecīgā vietā jādod atsauce uz tabulu, piemēram: “… kā rāda 4.1. tabulas dati...”

* Atsaucei uz tabulu tekstā pirmoreiz jāparādās **pirms** tabulas.
* Tabulas nosaukumu **nepasvītro**.
* Tabulas nosaukumā vēlams **izvairīties no komatiem**.
* Tabulas **galvenei** jābūt iezīmētai kā *Header Row*. Ja tabula aizņem vairākas lapaspuses, galveni atkārto.
* Tukšā tabulas rindā (ailē, šūniņā) ievelk svītriņu vai liek daudzpunktus.
* Nākamai pēc tabulas rindkopai jālieto stils „Normal pēc tab/form” vai „Normal pēc tab/form ar Tab”,, kas nodrošina attālumu pirms rindkopas sākuma.

## Formulas

Darbā ievietotās matemātiskās formulas iekļauj tekstā, taču katru no tām raksta savā atsevišķā rindiņā. Formulas numurē nodaļas ietvaros ar arābu cipariem, kurus raksta aiz formulas rindiņas labajā pusē, piemēram (1.1), (2.3) utt.

Numuru ieslēdz parastajās (apaļās) iekavās un raksta vienā rindā ar formulu lapas labajā malā. Aiz formulas kārtas numura punktu neliek, piemēram:

|  |  |
| --- | --- |
| , | (4.1) |

|  |  |  |  |
| --- | --- | --- | --- |
| kur |  | – | nepieciešamais materiāla daudzums gadā, kg; |
|  |  | – | materiāla patēriņa norma, kg/gab.; |
|  |  | – | gada ražošanas apjoms, gab. |

* Formulu labākam noformējumam ieteicams lietot **neredzamas tabulas**: pašai formulai – viena rinda ar divām kolonnām; eksplikācijai – tabula ar 4 kolonnām.
* Tekstā, atsaucoties uz kādu no formulām, tās numuru raksta tāpat kā aiz formulas – apaļajās iekavās, piemēram: "Izmantojot formulu (4.1) …".
* Atsaucei uz formulu pirmoreiz jāparādās **pirms** formulas.
* Ja formula aizņem vairākas rindiņas, numuru raksta pēdējās rindiņas labajā pusē.
* Formula parasti ir teikuma neatņemama daļa, un tajā teikumā korekti jālieto pieturzīmes.
* Formulā izmantotajiem simboliem jābūt atšifrētiem eksplikācijā, kuru raksta tūlīt aiz formulas, pirmo rindiņu sākot ar vārdu "kur". To raksta lapas kreisajā malā, kolu aiz tā neliek.
* Simboliem / atšifrējumiem tekstā / eksplikācijā **jāizskatās tāpat** kā pašā vienādojumā.
* Ja formulā izmantotiem elementiem ir zināmas mērvienības, tām jābūt iekļautām eksplikācijā.
* Formulu un vienādojumu rakstīšanai obligāti jāizmanto formulu **redaktors**, piemēram, *Microsoft Equation* vai *Math Type Equation*.

# 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[[20]](#footnote-20).

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*) [[21]](#footnote-21), 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**:**[[22]](#footnote-22)

Author**.** *Title* *(Italic[[23]](#footnote-23))***.** 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 - порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпля [↑](#footnote-ref-19)
20. Norādījumi noslēguma darba noformēšanai studiju programmā „Informācijas tehnoloģija” / Izstr. L. Aleksejeva, J. Grabis, J. Merkurjevs, I. Upīte. Rīga: RTU, 2016. 20 lpp. [↑](#footnote-ref-20)
21. 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-21)
22. Krāsas lietotas tikai, lai izceltu būtiskākos atribūtus; tās nav jāizmanto bibliogrāfijas noformēšanā. [↑](#footnote-ref-22)
23. *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-23)