|  |
| --- |
| 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.

# Языки программирования

## Новые языки программирования

В мире существует множество языков программировния, но все время создаются новые. Вполне закономерно будет задаться вопросом – зачем. Ведь существуют такие титаны индустрии как “Java”, “C”, “C++”,” Objective-C” и не на столько старые, но уже занявшие свою нишу “JavaScript”, “PHP”,”Ruby”. На это есть несколько причин и первая из них – это ориентированность. Согласитесь, что программировать низкоуровневые механизмы лучше на С, а интерактивые веб приложения на JavaScript. Второй причиной является архитектура языка. Не все языки подходят для высоконагруженных програм с большим количеством параллельных вычислений. Архитектурные решения такие как “null[[1]](#footnote-1)” и ”Generics”. Многие программисты считают null большой ошибкой в языках программирования. А за Generics создатели например Java, приносят извинения. Но по скольку многие языки программирования являются обратно совместимыми[[2]](#footnote-2) они по просту не могут изменить и улучшить эти части. По этому создаются новые языки, которые не повторяют ошибок своих предшественников и подстраиваются под новые тенденции в программировании. Ниже будут приведены несколько новых языков программирования появившихся в последние 3 года.

### Google Go

Go also known as golang, was launched in 2009, having been created by three Google employees - Robert Griesemer, Rob Pike, and Ken Thompson - in 2009. The open source language is viewed as faster and easier to use than more established languages such as Java and C, from which it is derived. It is used by a number of organisations, from the BBC to SoundCloud, and Facebook to the UK government award-winning GOV.UK site. It is also used by enterprise software startup du jour, Docker. “Go is an attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language,” its creators say.

### Swift

Swift revealed at Apple's WWDC conference in 2014, is intended as a replacement for the Objective-C language for OSX and iOS development. Apple made the language open source in December 2015 under the Apache license. That means . all of the source code will be available to edit and programs can be made without attributing them to Apple. Swift - which has similarities to more modern languages like Ruby and Python - has been enjoying “meteoric” growth since launching, Swift - which has similarities to more modern languages like Ruby and Python. “Swift adopts safe programming patterns and adds modern features to make programming easier, more flexible, and more fun,” says Apple.

### Scala

Scala functional and object-oriented language – which is highly scalable, hence its name – is continuing to gain ground with well-known organisations.

Also is running on JVM[[3]](#footnote-3). You can also rely on it for large mission critical systems, as many companies, including Twitter, LinkedIn, or Intel do.

### Rust

Created by Mozilla, Rust 1.0 was released in 2014, having been in development for a number of years. Close in some respects to C and C++, Mozilla desribes it as a “new programming language which focuses on performance, parallelisation, and memory safety. By building a language from scratch and incorporating elements from modern programming language design, the creators of Rust avoid a lot of backward-compatibility requirements that traditional languages have to deal with.

### Ceylon

Ceylon это – Объектно ориентированный язык высокого уровня со строгой статической типизацией и SDK[[4]](#footnote-4), разрабатываемые компанией Red Hat. Первая информация об этом языке появилась в 2011 году, когда разработчики языка начали вести блок в котором делились успехами и достижениями. Релиз первой полноценной версии был 12 ноября 2013 года. Этот язык может быть скомпилирован в byte-code для JVM или в JavaScript

## Специализированные и универсальные языки

Каждый язык программирования может быть использован в одной или нескольких областях. Основные области – это Web, Desktop, Mobile, Embedded.

* Web – Языки для аппликаций которые доступны через интернет и работают удаленном сервере. Картинка – расчерченный круг.
* Desktop – Языки для програм которые устанавливаются и работают на конкретном компьютере, сейчас это в основом научные программы. Картинка – монитор.
* Mobile – Языки для аппликаций которые работают на смартфонах. Картинка – смартфон.
* Embedded – Языки для программирование низкоуровневых устройств. Картинка – микрочип.



1.1.1. att. **Популярность и специализации языков**

Как можно увидеть на картинке 1.1 есть узкоспециализированные языки такие как R, есть достаточно универсальные такие как Java или C. В свою очередь Web раздел делиться на 2 части backend [[5]](#footnote-5)и frontend[[6]](#footnote-6). В основном backend и frontend пишутся на разных языках. Обычно это Java, C# или PHP для backend и JavaScript для frontend, но почти во всех языках есть фреймворк[[7]](#footnote-7) для написания front end. Язык о котором пойдет речь далее называется Ceylon. Очень молодой язык, который по словам разработчиков, должен избавить программистов от проблем, с которыми они столкнулись в Java.

# Ceylon и JVM

## Почему Ceylon

На JVM основано достаточно много языков а именно : Scala, Groovy, Clojure, Kotlin, Ceylon, Xtend, X10. Все эти языки имеют доступ ко всей, нажитой долгими годами, инфрастуктуре Java. Ко всем библиотекам, фреймворкам и прочему. Это одна из причин по которой я считаю, что языки основанные на JVM, будут иметь успех в одной из самых популярных индустрий. Речь идет о Web разработке, больших сложных систем, которые используются в больших предприятиях. Ведь Java сейчас занимает лидирущую позицию в этой области. Мой же выбор пал на Ceylon. Я считаю, что Ceylon самый перспективный язык из списка JVM языков. Ceylon не является функциональным языком программирования, что как я считаю, является плюсом, современные системы достаточно сложны и без функицональных элементов. Ceylon может быть скомпилирован в JavaScript, что является уникальной особенностью среди JVM языков. Это позволяет писать части программы которые будут выполняться непосредственно у пользователя в браузере, а не на сервере.

### Red Hat

Red Hat это самая крупная компания занимающаяся разработкой Open-Source Software[[8]](#footnote-8) продуктов, основонная в 1993 году. Это компания которая создала бесплатную и открытую операционную систему, которая в свою очередь используется по всему миру – Linux. Red Hat также занимается поддержкой и предоставлением услуг связанных со своими продуктами. Это ещё одна причина по которой я считаю, что Ceylon самый перспективный язык. Ведь, кто если не люди создавшие Linux, и множество других продуктов, люди который сотрудничают с ведущими компаниями в своих индустриях, могут создать лучший язык на основе JVM.

## Альтернативы JVM

Хотелось бы детальнее рассмотреть другие языки JVM и попробовать оценить их как альтернативу устоявшейся Java. Как уже было сказано выше Java является лидирующим языком для разработки серверной части тяжеловесных, больших и серьезных приложений для корпораций или банков.

### Scala

Scala is a statically typed programming language that fuses the object-oriented model and functional programming ideas. That means, in practice, that you can declare classes, create objects, and call methods just like you would typically do in Java. However, Scala also brings popular features from functional programming languages such as pattern matching[[9]](#footnote-9) on data structures, local type inference[[10]](#footnote-10), persistent collections[[11]](#footnote-11), and tuple[[12]](#footnote-12) literals. Я считаю, что Scala не очень подходит для больших проектов в которых работает много людей, только если требуется большая масштабируемость. Scala подойдет для проектов с очень высокой нагрузкой, и очень хорошим быстродействием. Имеет все возможности Java а так же возможность работать почти со всеми существующими библиотеками.

### Groovy

[Groovy](http://groovy.codehaus.org/) is a dynamically typed object-oriented language. Groovy’s dynamic nature lets you manipulate your code in powerful ways. For example, you can expand objects at runtime [[13]](#footnote-13) - by adding fields or methods. С моей точки зрения Groovy не подходит для больших проектов и комманд. Динамическая типизация не дает уверенности, когда много людей работает над проектом и каждый человек не знает всех частей. Зато на Groovy очень удобно покрывать тестами код. Тесты получаются лаконичнее. Имеет все возможности Java, некоторые из них даже дополнены, например работа с коллекциями.

### Clojure

[Clojure](http://www.clojure.org/) is a dynamically typed programming language that can be seen as a modern take on Lisp[[14]](#footnote-14). It is radically different from what object-oriented programmers might be used to. In fact, Clojure is a fully functional programming language, and as a result, it is centered on immutable data structures, recursion, and functions. Я считаю, что Clojure подходит только для научной деятельности, как и любой другой полностью функциональный язык программирования и абсолютно не применим в корпоративной разработке. Имеет доступ к Java экосистеме, по скольку язык не объектно – ориентированный, то функциональность может быть ограничена.

### Kotlin

[Kotlin](http://kotlin.jetbrains.org/) is a statically typed object-oriented language. Its main design goals are to be compatible with Java’s API, have a type system that catches more errors at compile time, and be less verbose than Java. Kotlin’s designers say that Scala is a close choice to match its design goals, but they dislike Scala’s complexity and long compilation time compared to Java. Kotlin aims to tackle these issues. По моему мнению Kotlin лучше всех предидущих языков подходит для больших проектов. Уменьшенное время компиляции на ряду с системой отлова большей части исключений во время компиляции кода, позволит допускать меньше ошибок. Разработчиком Kotlin является компания – Jetbrains, которая разработала ряд IDE[[15]](#footnote-15) для самых разных языков. IDE для Java не так давно стала самой популярной для этого Языка.

### Xtend

[Xtend](https://www.eclipse.org/xtend/) is a statically typed object-oriented language. One way it differs from other languages is that it compiles to pretty-printed Java code rather than bytecode. As a result, you can also work with the generated code.

Xtend supports two forms of method invocation: default Java dispatching and multiple dispatching. With multiple dispatching, an overloaded method is selected based on the runtime type of its arguments. Xtend provides many other popular features available in other languages such as operator overloading and type inference. Интересная особенность языка это - активные аннотации. Активные аннотации это способ использовать метапрограммирование[[16]](#footnote-16) во время компиляции кода. Мне кажется, главная особенность этого языка является его главным недостатком, зачем программировать на языке который в следствии переводится в Java код. Считаю, что этот язык будет сложно и не удобно использовать в большом корпоративном проекте.

### X10

[X10](http://x10-lang.org/) is an experimental object-oriented language that IBM developed. It supports features such as first-class functions[[17]](#footnote-17) and is designed to facilitate efficient programming for high-performance parallel computing. To this end, the language is based on a programming model called the “partitioned global address space”. In this model, each process shares a global address space, and slices of this space are allocated as private memory for local data and access. To work with this model, X10 offers specialized built-in language constructs to work with concurrency and distributed execution. По скольку IBM сами называют этот язык эксперементальным, использовать его в корпоративных проектах – глупо. Я считаю, что этот язык будут использовать сами IBM для создания какой-нибудь очень сложной системы на подобии IBM Watson[[18]](#footnote-18).

# Технические аспекты

## Основы

### Hello Word!

По традиции первая программа на новом языке, это приветствие мира. На Ceylon это выглядит следующим образом.

Как видно на картинке 3.1.1 Ceylon отошел от всем привычного класса Main со статическим методом main. В Ceylon метод run не принимает никаких параметров, ничего не возвращает и имеет модификатор видимости shared. Shared значит, что этот метод является публичным и может быть вызван из любой части аппликации.

3.1.1. att. **Hello World!**

### Типы

В отличие от Java в Ceylon, по умолчанию ни один тип не может быть null.

Как видно на картинке 3.1.2 необходимо добавлять знак ? возле типа, что бы в значение этой переменной можно было записать null. Это имеет очень большое значение в командой разработке. Ведь когда поле одного класса используется в нескольких десятках классов, очень сложно сразу понять, может ли метод вместо значения получить null. Необходимо ли проверять это значение. А когда метод принимает не одну переменную, а объект с 30 полями. В случае с Ceylon, IDE вам подскажет, если вы собираетесь использовать переменную которая может быть null или нет. Это экономит огромное количество времени. Так же Ceylon имеет вполне стандартные Class,Interface, Object. Но есть так же дополнительные типы, такие как:

3.1.2. att. **Не null типы**

* Anything/Nothing – Самый верхний уровень типов. Все типы, объекты и классы могут быть как Anything так и Nothing.
* Null – в Ceylon, Null это отдельный тип, который означает отсутсвие значения.
* Basic - is the default superclass of classes whose declaration lacks an extends clause.
* Sequence - is the type of non-empty sequences.
* Sequential -  is an enumerated type with subtypes [Sequence](https://modules.ceylon-lang.org/repo/1/ceylon/language/1.2.2/module-doc/Sequence.type.html) and [Empty](https://modules.ceylon-lang.org/repo/1/ceylon/language/1.2.2/module-doc/Empty.type.html).
* Empty - is the type of a [Sequential](https://modules.ceylon-lang.org/repo/1/ceylon/language/1.2.2/module-doc/Sequential.type.html) which contains no elements.
* Tuple - is a subclass of non-empty Sequence. It differs from Sequence in that it encodes the types of each of its elements individually.

### Операторы

Помимо стандартных операторов сложение вычитание и логических действий, есть новые или измененные операторы:

* Entry – позволяет создать объект “Entry<KeyType,ValueType>”,которые использются например в Java, как элементы коллекции Map, используя краткую запись которую видно на картинке 3.1.3.



3.1.3. att. **Краткое создание Entry**

* Compare – сравнивает 2 объекта которые должны наследовать “Comparable” и возвращает Comparison, выглядит как – “<=>”.
* Null – бесопасный оператор, указывает, может ли переменная являться null, выглядит как знак вопроса возле типа.
* Span – оператор позволяющий обращаться к нескольким элементам массива сразу, как видно на картинке 3.1.4, выглядит как [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.

## Серверная часть

Для Ceylon не существует специальных фреймворков для создания серверной части веб приложений. Базы данных являются ключевой частью любого маломальски серьезного приложения, а по скольку в Ceylon нет специально разработонного способа работы с базами данных. Но это не проблема ведь 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 таких как: REST[[20]](#footnote-20) или SOAP[[21]](#footnote-21) endpoints, авторизация, работа с кеш паматью и так далее.

# O

## Ceylon, плюсы

### Seamless web development

Ceylon is evolving into a compelling platform for web development, allowing you to reuse code written for a Java server on the JavaScript client and vice-versa, while still interoperating cleanly with native code on both sides. With our new serialization facility, you’ll be able to transparently move objects between client and server. The Ceylon SDK comes with a HTTP server module based on Undertow, and a transactions module based on JBoss Transactions, but if you’re looking for something “heavier”, Ceylon modules can already be deployed on RedHat’s [OpenShift](https://developers.openshift.com/en/ceylon-overview.html) cloud platform, on WildFly, or on [Vert.x 2](http://vertx.io/vertx2/core_manual_ceylon.html) (with Vert.x 3 support coming soon). Alternatively you could use Ceylon in Node.js

### Truly disciplined static typing

Chances are you have some experience writing code in a language with static typing. In combination with the right tooling, static typing makes code more robust, more understandable, and much more maintainable. But most languages go out of their way to include features and APIs which undermine their own static type system. A handful of languages such as ML and Haskell eschew this, offering a truly a principled, disciplined approach to static typing. And programmers working with these languages report a curious thing: that their programs have a strange tendency to work first or second time they run them. Sure, it takes a little longer to produce a program that the compiler accepts, but once the compiler is satisfied, so many common bugs have already been eliminated, that the program is often already correct or at least almost correct.

Ceylon follows this same philosophy and, even though it’s a very different sort of language to ML, our experience is that our programs have the very same tendency to just work, almost immediately. You surely won’t believe this until you experience it yourself.

Ceylon’s type system itself is state-of-the-art, including generic types with both use-site and declaration-site covariance and contravariance, mixin (multiple) inheritance, principal instantiation inheritance, sum types, disjointness analysis, experimental support for higer-order and higher-rank generics, and, best of all, union and intersection types.

Union and intersection types

Ceylon was the first modern language to introduce a full implementation of union and intersection types, and the first language to realize that union and intersection types make flow-typing and type inference, especially generic type inference, very much simpler, more predictable, and more elegant. As of today, it’s still the language with the most sophisticated support for unions and intersections.

What’s most difficult to explain to folks who’ve never written any significant amount of code in Ceylon is just how powerful union and intersection types really are. Far from being an exotic construct that you encounter occasionally in tricky library code, they form an essential part of the everyday experience of writing code in Ceylon, and there is almost no significant Ceylon program which doesn’t use union types somewhere. It’s no exaggeration to say they will totally change the way you think about types.

### Flow-sensitive typing

Ceylon was also the first modern language to embrace the notion of flow-sensitive typing, where the conditional logic in the body of a function affects the types inferred by the compiler. This approach eliminates a significant amount of verbosity and a significant number of runtime type casting errors. In combination with Ceylon’s powerful coverage and disjointness analysis, it also helps detect certain other kinds of logic errors.

### The cleanest solution to the problem of null

The null value has perplexed language designers for decades. For a long time the “best” solution to the problem was an ML- or Haskell-style Maybe type, which offered type safety and eliminated the hated “null pointer” error. That solution works quite well for languages without subtyping, but it’s not the best approach for a language with subtyping and union types. Ceylon’s approach to null is just as type safe, but, with flow-sensitive typing, is more user-friendly, and it doesn’t require the overhead of separate Option-style wrapper objects. Unlike some other languages, Null isn’t a special case in Ceylon’s type system. Rather, it’s just one further example of the power of union types.

### An elegant and powerful representation of tuple and function types

Like most other modern languages, Ceylon features tuples and function references. Unlike other languages Ceylon doesn’t represent these things as special cases in the type system, nor as an explosion of interface types, one for each tuple/function -arity. Instead, there is just one class Tuple, and one interface Callable which expresses function types in terms of tuple types. This approach allows the language to abstract over tuple or function -arity, and write higher-order functions that simply can’t be expressed in many other modern languages.

### Reified generics

Along with Gosu, Ceylon is one of only two languages to offer reified generics on the JVM. And it’s the only language to offer reified generics on JavaScript virtual machines. You’ll sometimes run into attempts to “backsplain” the lack of reified generics in Java or other JVM languages: claims that reified generics perform badly, aren’t useful, or don’t work correctly with variance or with higher-order generics. Ceylon is an existence proof that none of these assertions is true.

# 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[[22]](#footnote-22).

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*) [[23]](#footnote-23), 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**:**[[24]](#footnote-24)

Author**.** *Title* *(Italic[[25]](#footnote-25))***.** 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 - пустое значение в языках программирования. [↑](#footnote-ref-1)
2. Обратная совместимость - наличие в новой версии компьютерной программы или компьютерного оборудования интерфейса, присутствующего в старой версии, в результате чего другие программы (или человек) могут продолжать работать с новой версией без значительной переделки (или переучивания). [↑](#footnote-ref-2)
3. JVM – Java Virtual Machine исполняет [байт-код Java](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4_Java), предварительно созданный из [исходного текста](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) Java-программы [компилятором](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) Java. JVM может также использоваться для выполнения программ, написанных на других языках программирования.. [↑](#footnote-ref-3)
4. SDK – software development kit комплект средств разработки, который позволяет специалиистам по программному обеспечению создавать приложения для определенного пакета программ. [↑](#footnote-ref-4)
5. Backend - the [part](http://dictionary.cambridge.org/dictionary/english/part) of a [computer](http://dictionary.cambridge.org/dictionary/english/computer) [program](http://dictionary.cambridge.org/dictionary/english/program) or [system](http://dictionary.cambridge.org/dictionary/english/system) that [deals](http://dictionary.cambridge.org/dictionary/english/deal) with [data](http://dictionary.cambridge.org/dictionary/english/data) and that the [user](http://dictionary.cambridge.org/dictionary/english/user) does not see or [control](http://dictionary.cambridge.org/dictionary/english/control) directly [↑](#footnote-ref-5)
6. Frontend - the part of a computer system or application with which the user interacts directly. [↑](#footnote-ref-6)
7. Фреймворк - [программная платформа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0), определяющая структуру программной системы; [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), облегчающее разработку и объединение разных компонентов большого программного проекта. [↑](#footnote-ref-7)
8. Open-Source Software - is [computer software](https://en.wikipedia.org/wiki/Computer_software) with its [source code](https://en.wikipedia.org/wiki/Source_code) made available with a [license](https://en.wikipedia.org/wiki/Open-source_license) in which the [copyright](https://en.wikipedia.org/wiki/Copyright) holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a [collaborative public manner](https://en.wikipedia.org/wiki/Collaborative_software_development_model). [↑](#footnote-ref-8)
9. Pattern matching -  is the act of checking a given sequence of tokens for the presence of the constituents of some [pattern](https://en.wikipedia.org/wiki/Pattern). [↑](#footnote-ref-9)
10. Type inference - в программировании возможность компилятора самому логически вывести тип значения у выражения. [↑](#footnote-ref-10)
11. Persistent collections -  is a [collection](https://en.wikipedia.org/wiki/Data_structure) that always preserves the previous version of itself when it is modified. [↑](#footnote-ref-11)
12. Tuple – A tuple is a class that can contain a miscellaneous collection of elements. [↑](#footnote-ref-12)
13. Runtime - the period during which a computer program is executing [↑](#footnote-ref-13)
14. Lisp – Язык программирования появившийся в 1958 году, научный язык в большинстве своем используемый для разработки ИИ. [↑](#footnote-ref-14)
15. IDE – Integrated Development Environment is a [software application](https://en.wikipedia.org/wiki/Application_software) that provides comprehensive facilities to [computer programmers](https://en.wikipedia.org/wiki/Computer_programmer) for [software development](https://en.wikipedia.org/wiki/Software_development). [↑](#footnote-ref-15)
16. Метапрограммирование -  вид [программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), связанный с созданием [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), которые порождают другие программы как результат своей работы. в частности, на стадии [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) их [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), либо программ, которые меняют себя во время выполнения [↑](#footnote-ref-16)
17. First-class Functions -  В частности, это означает, что язык поддерживает передачу функций в качестве аргументов другим функциям, возврат их как результат других функций, присваивание их переменным или сохранение в структурах данных. [↑](#footnote-ref-17)
18. IBM Watson – суперкомпьютер оснащенный вопросно-ответной системой искусственного интеллекта. Основная задача Watson – понимать вопросы на людском языке, и находить на них ответы в базе данных. [↑](#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. REST - Representational State Transfer [архитектурный стиль](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) взаимодействия компонентов распределённого приложения в [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой [гипермедиа](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)-системы. В определённых случаях,[интернет-магазины](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BC%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD), [поисковые системы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), прочие системы, основанные на данных, это приводит к повышению производительности и упрощению архитектуры [↑](#footnote-ref-20)
21. SOAP - Simple Object Access Protocol [протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур [RPC](https://ru.wikipedia.org/wiki/Remote_Procedure_Call). Сейчас протокол используется для обмена произвольными сообщениями в формате [XML](https://ru.wikipedia.org/wiki/XML), а не только для вызова процедур.  [↑](#footnote-ref-21)
22. Norādījumi noslēguma darba noformēšanai studiju programmā „Informācijas tehnoloģija” / Izstr. L. Aleksejeva, J. Grabis, J. Merkurjevs, I. Upīte. Rīga: RTU, 2016. 20 lpp. [↑](#footnote-ref-22)
23. 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-23)
24. Krāsas lietotas tikai, lai izceltu būtiskākos atribūtus; tās nav jāizmanto bibliogrāfijas noformēšanā. [↑](#footnote-ref-24)
25. *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-25)