

Kotlin DSL: Теория и Практика

Ivan Osipov
Haulmont

Бэкграунд

- Software Developer в Haulmont
- CUBA.platform
- Планирование образовательной деятельности
- Последние 3 года с системами планирования
- Тестируем с Kotlin уже год



HAULMONT



CUBA.platform

Бэкграунд

- Software Developer в Haulmont
- CUBA.platform
- Планирование образовательной деятельности
- Последние 3 года с системами планирования
- Тестируем с Kotlin уже год



HAULMONT



CUBA.platform

Бэкграунд

- Software Developer в Haulmont
- CUBA.platform
- Планирование образовательной деятельности
- Последние 3 года с системами планирования
- Тестируем с Kotlin уже год



HAULMONT



CUBA.platform

Бэкграунд

- Software Developer в Haulmont
- CUBA.platform
- Планирование образовательной деятельности
- Последние 3 года с системами планирования
- Тестируем с Kotlin уже год



HAULMONT



CUBA.platform

Бэкграунд

- Software Developer в Haulmont
- CUBA.platform
- Планирование образовательной деятельности
- Последние 3 года с системами планирования
- Тестируем с Kotlin уже год



HAULMONT



CUBA.platform

name Ivan Osipov

telegram ivan_osipov

twitter _osipov_

viber
whatsapp /dev/null



i-osipov.ru

github.com/ivan-osipov/kotlin-dsl-example

О докладе

- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

О докладе

- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

О докладе

- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

О докладе

- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

О докладе

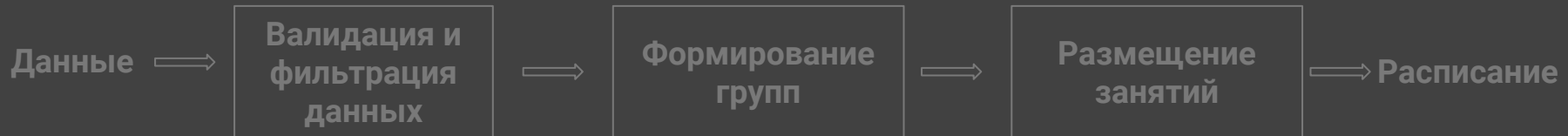
- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

О докладе

- DSL и EDSL;
- построение расписания для образовательного учреждения;
- тестирование с помощью DSL;
- инструменты Kotlin для разработки DSL;
- демо;
- минусы использования подхода.

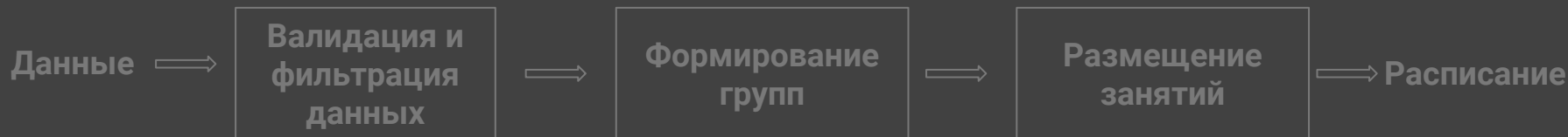
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



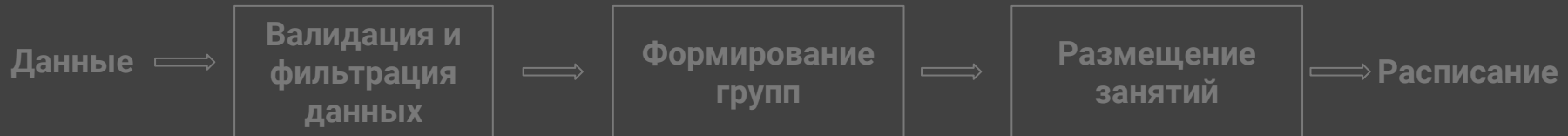
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



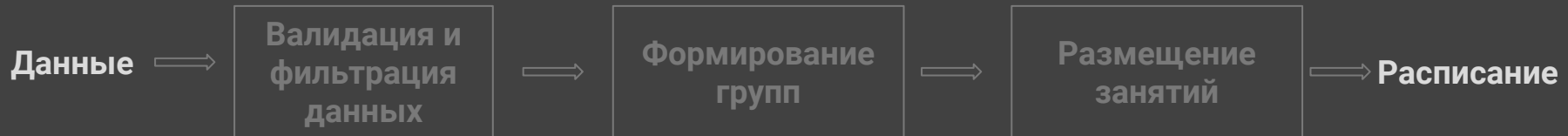
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- **общая модель данных.**



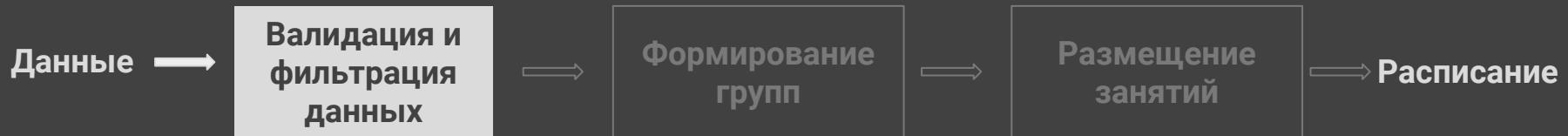
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



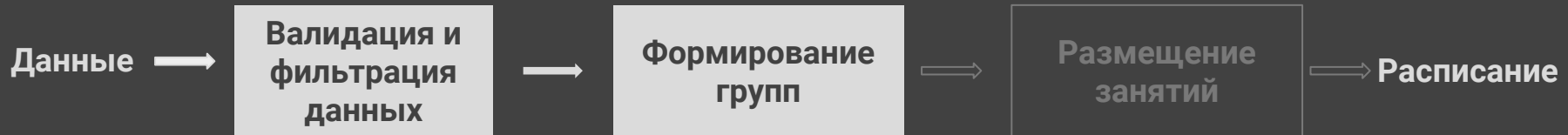
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



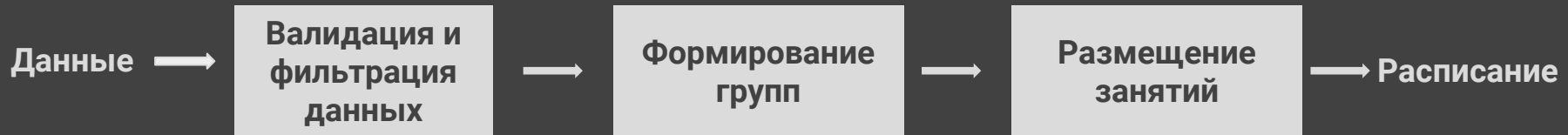
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



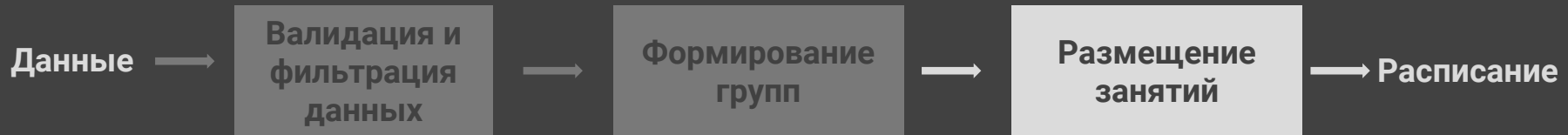
Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



Модуль планирования учебных занятий

- построение расписания состоит из нескольких этапов;
- каждый этап нужно тестировать;
- общая модель данных.



Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Тестирование планировщика

- разные этапы тестируются отдельно;
- стандартный процесс тестирования;
- широкий спектр функциональности;
- модель: 1 сущность $\approx$ 5 доп. сущностей;
- суммарно много кода для создания тестов;
- поддержка тестов занимает много времени;
- обновление модели = обновление кода тестов.

Напишем тест

Тест бобра-энтузиаста

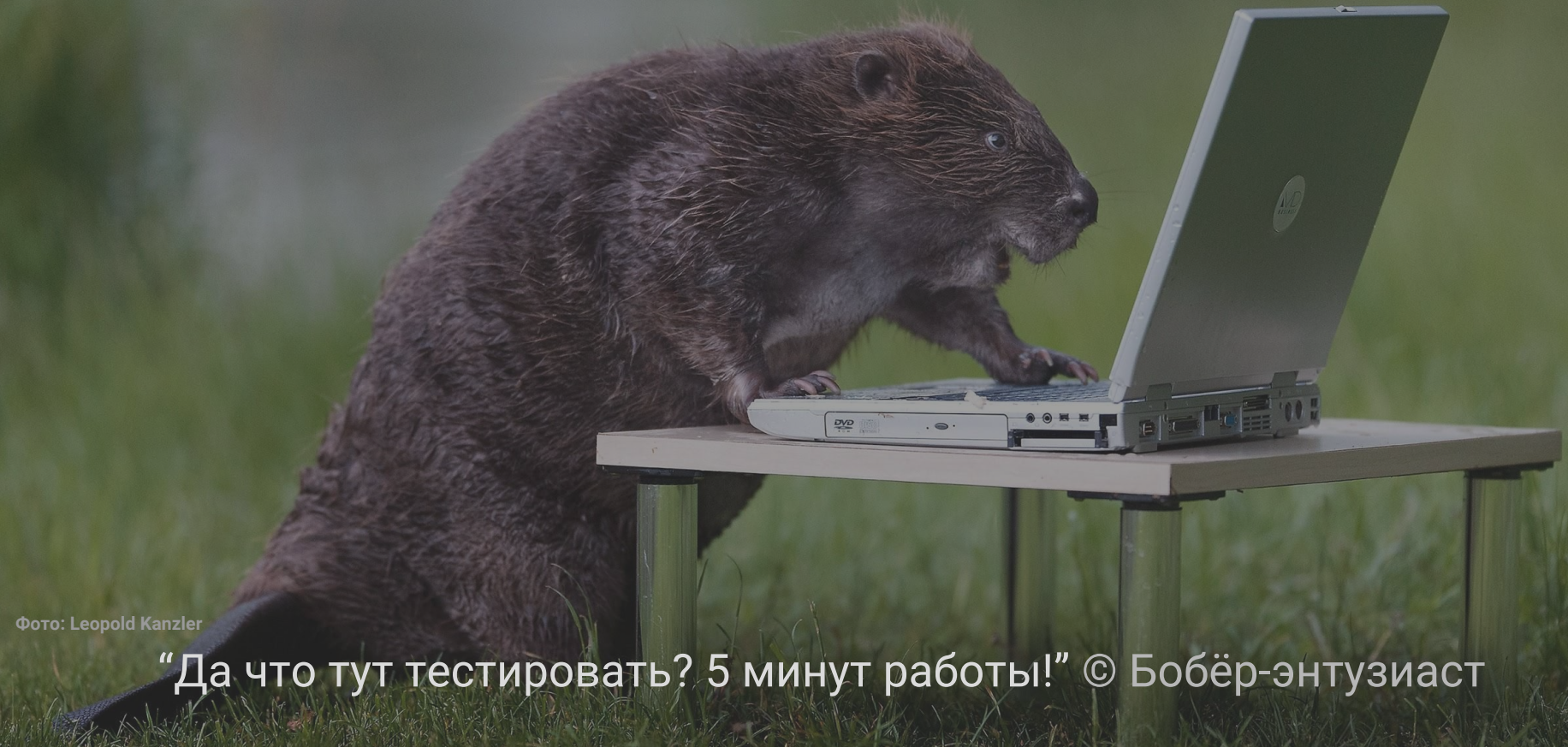


Фото: Leopold Kanzler

“Да что тут тестировать? 5 минут работы!” © Бобёр-энтузиаст

Тест бобра-энтузиаста

```
@Test
public void someTest() {
    Student studentA = new Student();
    studentA.setName("Default name");
    studentA.setProfile(/*...*/)

    Teacher teacherA = new Teacher();
    //аналогично

    Scheduler scheduler = new Scheduler();
    scheduler.initData(/*...*/);
    SchedulerResult schedule = scheduler.run();

    //куча assertions
}
```

Тест бобра-энтузиаста

```
@Test
public void someTest() {
    Student studentA = new Student();
    studentA.setName("Default name");
    studentA.setProfile(/*...*/)

    Teacher teacherA = new Teacher();
    //аналогично

    Scheduler scheduler = new Scheduler();
    scheduler.initData(/*...*/);
    SchedulerResult schedule = scheduler.run();

    //куча assertions
}
```


Тест бобра-энтузиаста

```
@Test
public void someTest() {
    Student studentA = new Student();
    studentA.setName("Default name");
    studentA.setProfile(/*...*/)

    Teacher teacherA = new Teacher();
    //аналогично

    Scheduler scheduler = new Scheduler();
    scheduler.initData(/*...*/);
    SchedulerResult schedule = scheduler.run();

    //куча assertions
}
```

Тест бобра-энтузиаста спустя 5 минут



Фото: Leopold Kanzler

“Надо вынести дублирование кода в статические функции” © Бобёр-энтузиаст

Тест бобра-энтузиаста спустя 5 минут

```
@Test
public void someTest() {
    Student studentA = createStudent();
    //проставляем только специфичные значения

    Teacher teacherA = createTeacher();
    //аналогично

    Scheduler scheduler = new Scheduler();
    scheduler.initData(/*...*/);
    SchedulerResult schedule = scheduler.run();

    //куча assertions
}
```


Тест бобра-строителя



Фото: Leopold Kanzler

“Напишем каждому классу в модели свой builder” © Бобёр-строитель

Тест бобра-строителя

```
@Test
```

```
public void someTest() {  
    Student studentA = new StudentBuilder()  
        .age(42)  
        .build();
```

уже лучше



```
Scheduler scheduler = new Scheduler();  
scheduler.initData(/*...*/);  
SchedulerResult schedule = scheduler.run();
```

```
//куча assertions
```

```
}
```

Тест бобра-строителя

```
@Test
public void someTest() {
    Student studentA = new StudentBuilder();
    studentA.age(42)
    studentA.build();

    Scheduler scheduler = new Scheduler();
    scheduler.initData(/*...*/);
    SchedulerResult schedule = scheduler.run();

    //куча assertions
}
```



boilerplate



Тест бобра-строителя-энтузиаста



Фото: Leopold Kanzler

“Сделаем общий builder для всей модели” © Бобёр-строитель-энтузиаст

Тест бобра-строителя-энтузиаста

```
@Test
public void someTest() {
    SchedulerResult schedule = new SchedulerBuilder()
                                .addStudent()
                                .age(42)
                                .build()
                                .addTeacher()
                                .schedule()

    //куча assertions
}
```


Тест бобра-строителя-энтузиаста

```
@Test
public void someTest() {
    SchedulerResult schedule = new SchedulerBuilder()
                                .addStudent()
                                .age(42)
                                .build()
                                .addTeacher()
                                .schedule()

    //куча assertions
}
```

ненужный код →

Тест бобра-строителя-энтузиаста

```
@Test
```

```
public void someTest() {
```

```
    SchedulerResult schedule = new SchedulerBuilder()
```

на 55% больше букв*



```
        .addStudent()
```

```
        .age(42)
```

```
        .build()
```

```
        .addTeacher()
```

```
        .schedule()
```

```
    //куча assertions
```

```
}
```

Спустя время

- **поддержка** тестов становится **дороже**;
- низкая или средняя читаемость;
- изучение фреймворков;
- повысили уровень входа в поддержку тестов.

Спустя время

- поддержка тестов становится дороже;
- **низкая** или средняя **читаемость**;
- изучение фреймворков;
- повысили уровень входа в поддержку тестов.

Спустя время

- поддержка тестов становится дороже;
- низкая или средняя читаемость;
- изучение фреймворков;
- повысили уровень входа в поддержку тестов.

Спустя время

- поддержка тестов становится дороже;
- низкая или средняя читаемость;
- изучение фреймворков;
- **повысили уровень входа** в поддержку тестов.

Идеальный Тест

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
      student
        name "Ivanov"
        subjects all
      teacher
        availability
          monday 8:00 16:00
          wednesday 8:00 16:00
    assertions
      //проверки на совместимом с Java языке
```



```
test "some test"
```

```
  schedule
```

```
    data
```

```
      start from 8:00
```

```
      subjects "Russian", "Literature"
```

```
      student
```

```
        name "Ivanov"
```

```
        subjects all
```

```
      teacher
```

```
        availability
```

```
          monday 8:00 16:00
```

```
          wednesday 8:00 16:00
```

```
  assertions
```

```
    //проверки на совместимом с Java языке
```

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
      student
        name "Ivanov"
        subjects all
      teacher
        availability
          monday 8:00 16:00
          wednesday 8:00 16:00
```

assertions

//проверки на совместимом с Java языке

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
      student
        name "Ivanov"
        subjects all
      teacher
        availability
          monday 8:00 16:00
          wednesday 8:00 16:00
    assertions
      //проверки на совместимом с Java языке
```

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
    student
      name "Ivanov"
      subjects all
    teacher
      availability
        monday 8:00
        wednesday 8:00 16:00
  assertions
    //проверки на совместимом с Java языке
```

Domain Specific Language

Языки можно разделить на 2 вида

General-purpose languages
Языки общего назначения

Java, Kotlin, C# и т.д.



Domain-specific languages
Проблемно-ориентированные языки

SQL, HTML, TeX и т.д.



Языки можно разделить на 2 вида

General-purpose languages
Языки общего назначения

Java, Kotlin, C# и т.д.



Domain-specific languages
Проблемно-ориентированные языки

SQL, HTML, TeX и т.д.



Embedded DSL

- способ реализации DSL;
- на основе general-purpose языка;
- несколько языковых конструкций образуют базис;
- можем использовать и другие фичи основного языка.

Embedded DSL

- способ реализации DSL;
- на основе general-purpose языка;
- несколько языковых конструкций образуют базис;
- можем использовать и другие фичи основного языка.

Embedded DSL

- способ реализации DSL;
- на основе general-purpose языка;
- несколько языковых конструкций образуют базис;
- можем использовать и другие фичи основного языка.

Embedded DSL

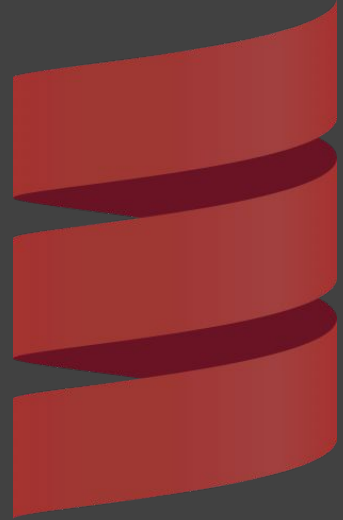
- способ реализации DSL;
- на основе general-purpose языка;
- несколько языковых конструкций образуют базис;
- можем использовать и другие фичи основного языка.

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
    student
      name "Ivanov"
      subjects all
    teacher
      availability
        monday 8:00
        wednesday 8:00 16:00
  assertions
    //проверки на совместимом с Java языке
```

Какой язык?

X3







Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система **вывода типов**;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

Kotlin для реализации DSL

- статически типизированный язык;
- добротная система вывода типов;
- хорошая поддержка IntelliJ IDEA;
- внутри DSL имеем возможности Kotlin;
- поддержка проще поддержки утилитных классов;
- читаемость лучше builder'ов;
- реализация проще изучения фреймворков.

github.com/ivan-osipov/kotlin-dsl-example

Проектирование идеала на Kotlin

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
    student
      name "Ivanov"
      subjects all
    teacher
      availability
        monday 8:00
        wednesday 8:00 16:00
  assertions
    //проверки на совместимом с Java языке
```

```
test "some test"
  schedule
    data
      start from 8:00
      subjects "Russian", "Literature"
      student
        name "Ivanov"
        subjects all
      teacher
        availability
          monday 8:00
          wednesday 8:00 16:00
    assertions
      //проверки на совместимом с Java языке
```

```
@Test fun `some test`() =  
    schedule  
        data  
            start from 8:00  
            subjects "Russian", "Literature"  
            student  
                name "Ivanov"  
                subjects all  
            teacher  
                availability  
                    monday 8:00  
                    wednesday 8:00 16:00  
    assertions  
        //проверки на совместимом с Java языке
```

```
@Test fun `some test`() =  
    schedule {  
        data  
            start from 8:00  
            subjects "Russian", "Literature"  
            student  
                name "Ivanov"  
                subjects all  
            teacher  
                availability  
                    monday 8:00  
                    wednesday 8:00 16:00  
        assertions  
            //проверки на совместимом с Java языке  
    }
```

```
@Test fun `some test`() = schedule {  
    data  
        start from 8:00  
        subjects "Russian", "Literature"  
        student  
            name "Ivanov"  
            subjects all  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    assertions  
        //проверки на совместимом с Java языке  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        start from 8:00  
        subjects "Russian", "Literature"  
        student  
            name "Ivanov"  
            subjects all  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    } assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```



```
@Test fun `some test`() = schedule {  
    data {  
        start from 8:00  
        subjects "Russian", "Literature"  
        student  
            name "Ivanov"  
            subjects all  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    } assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        startFrom("8:00")  
        subjects("Russian", "Literature")  
        student  
            name "Ivanov"  
            subjects all  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    } assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        startFrom("8:00")  
        subjects("Russian", "Literature")  
        student  
            name "Ivanov"  
            subjects all  
            teacher  
                availability  
                    monday 8:00  
                    wednesday 8:00 16:00  
    }  
    assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        startFrom("8:00")  
        subjects("Russian", "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    }  
    assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        startFrom("8:00")  
        subjects("Russian", "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher  
            availability  
                monday 8:00  
                wednesday 8:00 16:00  
    }  
    assertions {  
        //проверки на совместимом с Java языке  
    }  
}
```

```
@Test fun `some test`() = schedule {  
    data {  
        startFrom("8:00")  
        subjects("Russian", "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("8:00")  
                wednesday("8:00", "16:00")  
            }  
        }  
    }  
} assertions {
```

```
data {  
    startFrom("8:00")  
    subjects("Russian", "Literature")  
    student {  
        name = "Ivanov"  
        allSubjects()  
    }  
    teacher {  
        availability {  
            monday("8:00")  
            wednesday("8:00", "16:00")  
        }  
    }  
} assertions {  
    //проверки на совместимом с Java языке
```

```
startFrom("8:00")
subjects("Russian", "Literature")
student {
    name = "Ivanov"
    allSubjects()
}
teacher {
    availability {
        monday("8:00")
        wednesday("8:00", "16:00")
    }
}
} assertions {
    //проверки на совместимом с Java языке
}
```



```
subjects("Russian", "Literature")
```

```
student {
```

```
    name = "Ivanov"
```

```
    allSubjects()
```

```
}
```

```
teacher {
```

```
    availability {
```

```
        monday("8:00")
```

```
        wednesday("8:00", "16:00")
```

```
    }
```

```
}
```

```
} assertions {
```

```
    //проверки на совместимом с Java языке
```

```
}
```

```
}
```

```
student {  
    name = "Ivanov"  
    allSubjects()  
}
```

```
teacher {  
    availability {  
        monday("8:00")  
        wednesday("8:00", "16:00")  
    }  
}
```

```
} assertions {  
    //проверки на совместимом с Java языке  
}
```

```
}
```

```
student {  
    name = "Ivanov"  
    allSubjects()  
}  
teacher {  
    availability {  
        monday("8:00")  
        wednesday("8:00", "16:00")  
    }  
}  
} assertions {  
    //проверки на Kotlin  
}  
}
```

Арсенал разработчика DSL на Kotlin

Название функциональности	DSL синтаксис	Обычный синтаксис
Переопределение операторов	<code>collection += element</code>	<code>collection.add(element)</code>
Псевдонимы типа	<code>typealias Point = Pair<Int, Int></code>	Создание пустых классов-наследников и прочие костыли
Соглашение для get/set методов	<code>map["key"] = "value"</code>	<code>map.put("key", "value")</code>
Мульти-декларации	<code>val (x, y) = Point(0, 0)</code>	<code>val p = Point(0, 0); val x = p.first; val y = p.second</code>
Лямбда за скобками	<code>list.forEach { ... }</code>	<code>list.forEach({...})</code>
Extention функции	<code>mylist.first();</code> // метод first() отсутствует в классе коллекции mylist	Утилитные функции
Infix функции	<code>1 to "one"</code>	<code>1.to("one")</code>
Лямбда с обработчиком	<code>Person().apply { name = «John» }</code>	Нет
Контролирование контекста	<code>@DslMarker</code>	Нет

Название функциональности	DSL синтаксис	Обычный синтаксис
Переопределение операторов	<code>collection += element</code>	<code>collection.add(element)</code>
Псевдонимы типа	<code>typealias Point = Pair<Int, Int></code>	Создание пустых классов-наследников и прочие костыли
Соглашение для get/set методов	<code>map["key"] = "value"</code>	<code>map.put("key", "value")</code>
Мульти-декларации	<code>val (x, y) = Point(0, 0)</code>	<code>val p = Point(0, 0); val x = p.first; val y = p.second</code>
Лямбда за скобками	<code>list.forEach { ... }</code>	<code>list.forEach({...})</code>
Extention функции	<code>mylist.first();</code> // метод first() отсутствует в классе коллекции mylist	Утилитные функции
Infix функции	<code>1 to "one"</code>	<code>1.to("one")</code>
Лямбда с обработчиком	<code>Person().apply { name = «John» }</code>	Нет
Контролирование контекста	<code>@DslMarker</code>	Нет

Лямбды в Kotlin

```
val lambda: () -> Unit = { }
```

Лямбды в Kotlin

```
val lambda: () -> Unit = { }
```


Лямбды в Kotlin

```
val lambda: () -> Unit = { }
```

Лямбды в Kotlin

```
val lambda: () -> Unit = { }
```

```
fun main(args: Array<String>) {  
    lambda()  
}
```

Лямбды в Kotlin

```
val lambda: (String) -> Unit = {  
    println(it)  
}
```

Лямбды в Kotlin

```
val lambda: (String) -> Unit = {  
    println(it)  
}
```

Лямбды в Kotlin

```
val lambda: (String) -> Unit = { param ->
    println(param)
}
```

Лямбды в Kotlin

```
val lambda: (String) -> Unit = { _ ->
    println("Output")
}
```

Лямбды в Kotlin

```
val lambda: (String, String) -> Unit = {  
    println(it) //так нельзя  
}
```

Лямбда — параметр метода

```
fun exec(someCode: () -> Unit){  
    someCode()  
}
```

```
exec({  
    println("Hello")  
})
```


Лямбда — параметр метода

```
fun exec(someCode: () -> Unit){  
    someCode()  
}
```

```
exec({  
    println("Hello")  
})
```

Лямбда вне скобок

```
fun exec(someCode: () -> Unit){  
    someCode()  
}
```

```
exec() {  
    println("Hello")  
}
```

Лямбда вне скобок

```
fun exec(someCode: () -> Unit){  
    someCode()  
}
```

```
exec {  
    println("Hello")  
}
```

Применение

```
schedule {  
    data {  
        startFrom("08:00")  
        subjects("Russian",  
                "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("08:00")  
                wednesday("09:00", "16:00")  
            }  
        }  
    }  
    assertions {  
        // проверки на Kotlin  
    }  
}
```

Лямбда с контекстом

```
val lambda: Context.() -> Unit = { }
```

Лямбда с контекстом

```
val lambda: Context.() -> Unit = { }
```

Лямбда с контекстом

```
val lambda: Context.() -> Unit = {  
    println(this)  
}
```


Лямбда с контекстом

```
val out: String.() -> Unit = {  
    println(this)  
}
```

```
“Hello, world”.out()
```

Лямбда с контекстом

```
val out: String.() -> Unit = {  
    println(this)  
}
```

```
"Hello, world".out()
```

Лямбда с контекстом

```
val out: String.() -> Unit = {  
    println(this)  
}
```

```
out("Hello, world") //так тоже можно
```

Лямбда с контекстом

```
val out: String.() -> Unit = {  
    println(this)  
}
```

```
out() //так нельзя
```

Лямбда с контекстом - параметр

```
fun exec(init: () -> Unit){  
    init()  
}
```

```
exec {  
    //код лямбды  
}
```

Лямбда с контекстом - параметр

```
fun student(init: () -> Unit){  
    init()  
}
```

```
student {  
    //код лямбды  
}
```

```

schedule { //SchedulingContext
    data { //DataContext
        startFrom("08:00")
        subjects("Russian",
                "Literature")
        student { //Student
            name = "Ivanov"
            allSubjects()
        }
        teacher { //Teacher
            availability { //Matrix<Boolean, Boolean>
                monday("08:00")
                wednesday("09:00", "16:00")
            }
        }
    }
} assertions { //AssertionContext
    //...
}
}

```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    init()  
}
```

```
student {  
    //код лямбды  
}
```


Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    init()  
}
```

```
student {  
    //код лямбды  
}
```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    student.init()  
}
```

```
student {  
    //код лямбды  
}
```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    student.name = "Default Name"  
    student.init()  
}
```

```
student {  
    //код лямбды  
}
```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    student.name = "Default Name"  
    student.init()  
}
```

```
student {  
    this.name = "Василий"  
}
```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    student.name = "Default Name"  
    student.init()  
}
```

```
student {  
    name = "Василий"  
}
```

Лямбда с контекстом - параметр

```
fun student(init: Student.() -> Unit){  
    val student = Student()  
    student.name = "Default Name"  
    student.init()  
}
```

```
student {  
    name = "Василий"  
    subjects(...)  
}
```

Применение

```
schedule { //SchedulingContext
  data { //DataContext
    startFrom("08:00")
    subjects("Russian",
             "Literature")
    student { //Student
      name = "Ivanov"
      allSubjects()
    }
    teacher { //Teacher
      availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
      }
    }
  } assertions { //AssertionContext
    //...
  }
}
```


Резюме по лямбдам

```
val lambda: () -> Unit = { }
```

```
val lambda: Context.() -> Unit = {  
    println(this)  
}
```

Резюме по лямбдам

```
val lambda: () -> Unit = { }
```

```
val lambda: Context.() -> Unit = {  
    println(this)  
}
```

Операторы

```
schedule { //SchedulingContext
  data { //DataContext
    startFrom("08:00")
    subjects("Russian",
             "Literature")
    student { //Student
      name = "Ivanov"
      allSubjects()
    }
    teacher {
      availability {
        monday("08:00")
        wednesday("09:00", "16:00")
      }
    }
  } assertions { //AssertionContext
    //...
  }
}
```

```
schedule { //SchedulingContext
  data { //DataContext
    startFrom("08:00")
    subjects("Russian",
             "Literature")
    student { //Student
      name = "Ivanov"
      allSubjects()
    }
    teacher {
      availability {
        monday("08:00") + time("13:00")
        wednesday("09:00", "16:00")
      }
    }
  } assertions { //AssertionContext
    //...
  }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```



```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

DayPointer

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

DayPointer IntRange

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```



```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

DayPointer IntRange

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

DayPointer IntRange

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```

```
availability { //AvailabilityTable
    monday("8:00") + time("13:00")
}
```

DayPointer IntRange

```
class AvailabilityTable {
    fun monday(...): DayPointer { /*mapping*/ }
    fun time(...): IntRange { /*mapping*/ }
}
```

```
class DayPointer(val availability: AvailabilityTable,
                 val dayOfWeek: DayOfWeek) {
    operator fun plus(lessonIndexRange: IntRange): DayPointer {
        /*mapping*/
    }
}
```



```
schedule {  
    data {  
        startFrom("08:00")  
        subjects("Russian",  
                 "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("08:00")  
                wednesday("09:00", "16:00")  
            }  
        }  
    }  
    assertions { //AssertionContext  
        // проверки на Kotlin  
    }  
}
```

СИНГЛТОНЫ

```
object MySingleton {  
    fun printText(s: String) {  
        println(s)  
    }  
}
```

```
MySingleton.printText("Hello!")
```

СИНГЛТОНЫ

```
object MySingleton {  
    fun printText(s: String) {  
        println(s)  
    }  
}
```

```
MySingleton.printText("Hello!")
```

СИНГЛТОНЫ

```
object MySingleton {  
    fun printText(s: String) {  
        println(s)  
    }  
}
```

```
MySingleton.printText("Hello!")
```

СИНГЛТОНЫ

```
public final class MySingleton {  
    public static final MySingleton INSTANCE;  
  
    private MySingleton() {  
        INSTANCE = (MySingleton)this;  
    }  
  
    static {  
        new MySingleton();  
    }  
}
```

Tools -> Kotlin -> Show Kotlin Bytecode -> Decompile

Оператор `invoke` + лямбда с контекстом за скобками

```
val instance = A()
```

```
instance({ })
```

```
class A {  
    operator fun invoke(lambda: Context.() -> Unit) = ...  
}
```

Оператор `invoke` + лямбда с контекстом за скобками

```
val instance = A()
```

```
instance({ })
```

```
class A {  
    operator fun invoke(lambda: Context.() -> Unit) = ...  
}
```

Оператор `invoke` + лямбда с контекстом за скобками

```
val instance = A()
```

```
instance({ })
```

```
class A {  
    operator fun invoke(lambda: Context.() -> Unit) = ...  
}
```


Оператор `invoke` + лямбда с контекстом за скобками

```
val instance = A()
```

```
instance { }
```

```
class A {  
    operator fun invoke(lambda: Context.() -> Unit) = ...  
}
```

Оператор `invoke` + лямбда с контекстом за скобками + синглтон

```
object schedule {  
    operator fun invoke(init: SchedulingContext.() -> Unit) {  
        SchedulingContext().init()  
    }  
}  
  
schedule { }
```

Оператор `invoke` + лямбда с контекстом за скобками + синглтон

```
object schedule {  
    operator fun invoke(init: SchedulingContext.() -> Unit) {  
        SchedulingContext().init()  
    }  
}  
  
schedule { }
```

Оператор `invoke` + лямбда с контекстом за скобками + синглтон

```
object schedule {  
    operator fun invoke(init: SchedulingContext.() -> Unit) {  
        SchedulingContext().init()  
    }  
}  
  
schedule { }
```

```
schedule {  
    data {  
        startFrom("08:00")  
        subjects("Russian",  
                 "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("08:00")  
                wednesday("09:00", "16:00")  
            }  
        }  
    }  
    assertions { //AssertionContext  
        // проверки на Kotlin  
    }  
}
```

```
schedule {  
    data {  
        startFrom("08:00")  
        subjects("Russian",  
                "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("08:00")  
                wednesday("09:00", "16:00")  
            }  
        }  
    }  
} assertions { //AssertionContext  
    // проверки на Kotlin  
}  
}
```

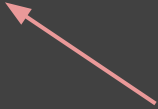
Соглашение для get/set

```
//проверки на Kotlin
for ((day, lesson, student, teacher) in scheduledEvents) {
    val teacherSchedule: Matrix<Event?> = teacher.schedule

    val currentLesson = teacherSchedule[day, lesson]

    currentLesson shouldNotEqual null

    currentLesson!!.student shouldEqual student
}
```



Соглашение для get/set

```
class Matrix(...) {  
    private val content: List<MutableList<T>>  
  
    operator fun get(i: Int, j: Int) = content[i][j]  
  
    operator fun set(i: Int, j: Int, value: T) {  
        content[i][j] = value  
    }  
}
```


Соглашение для get/set

```
val currentLesson = teacherSchedule[day, lesson]
```



```
val currentLesson = teacherSchedule.get(day, lesson)
```

Соглашение для get/set

```
teacherSchedule[day, lesson] = someLesson
```



```
teacherSchedule.set(day, lesson, someLesson)
```

Демо: object, operators

```

schedule {
    data {
        startFrom("08:00")
        subjects("Russian",
                "Literature")
        student {
            name = "Ivanov"
            allSubjects()
        }
        teacher {
            availability {
                monday("08:00")
                wednesday("09:00", "16:00")
            }
        }
    }
} assertions { //AssertionContext
    // проверки на Kotlin
}
}

```

Блок assertions

```
for (event in scheduledEvents) {  
    val teacherSchedule = teacher.schedule  
  
    val currentLesson = teacherSchedule[day, lesson]  
  
    currentLesson shouldNotEqual null  
  
    currentLesson!!.student shouldEqual student  
}
```

Мульти-декларации

```
val event = Event(...)
```

```
val day = event.day
```

```
val lesson = event.lesson
```

```
val student = event.student
```

```
val teacher = event.teacher
```

Мульти-декларации

```
val event = Event(...)
```

```
val (day, lesson, student, teacher) = event
```

Блок assertions

```
for ((day, lesson, student, teacher) in scheduledEvents) {  
    val teacherSchedule = teacher.schedule  
  
    val currentLesson = teacherSchedule[day, lesson]  
  
    currentLesson shouldNotEqual null  
  
    currentLesson!!.student shouldEqual student  
}
```


Мульти-декларации

```
val event = Event(...)
```

```
val day = event.component1()
```

```
val lesson = event.component2()
```

```
val student = event.component3()
```

```
val teacher = event.component4()
```

Мульти-декларации

```
data class Event(val day: Int,  
                 val lesson: Int,  
                 val student: Student,  
                 val teacher: Teacher) {  
  
    //+ equals/hashCode  
    //+ toString  
    //+ componentN  
    //+ copy  
  
}
```

Мульти-декларации

```
data class Event(val day: Int,  
                 val lesson: Int,  
                 val student: Student,  
                 val teacher: Teacher) {  
  
    //+ equals/hashCode  
    //+ toString  
    //+ componentN  
    //+ copy  
  
}
```

Мульти-декларации

```
data class Event(val day: Int,  
                 val lesson: Int,  
                 val student: Student,  
                 val teacher: Teacher) {  
  
    //+ equals/hashCode  
    //+ toString  
    //+ componentN  
    //+ copy  
  
}
```

Мульти-декларации

```
data class Event(val day: Int,  
                 val lesson: Int,  
                 val student: Student,  
                 val teacher: Teacher) {  
  
    //+ equals/hashCode  
    //+ toString  
    //+ componentN  
    //+ copy  
  
}
```

Мульти-декларации

```
class Event(val day: Int,  
            val lesson: Int,  
            val student: Student,  
            val teacher: Teacher) {  
  
    operator fun component1() = day  
    operator fun component2() = lesson  
    operator fun component3() = student  
    operator fun component4() = teacher  
  
}
```

Мульти-декларации

```
val event = Event(...)
```

```
val day = event.component1()
```

```
val lesson = event.component2()
```

```
val student = event.component3()
```

```
val teacher = event.component4()
```

Мульти-декларации

```
val event = Event(...)
```

```
val (day, lesson, student, teacher) = event
```


Псевдонимы типа

```
open class Teacher: Resource() {  
    ...  
    val availability = AvailabilityTable(...)  
}
```

```
typealias AvailabilityTable = Matrix<Boolean>
```

Псевдонимы типа

```
open class Teacher: Resource() {  
    ...  
    val availability = AvailabilityTable(...)  
}
```

```
typealias AvailabilityTable = Matrix<Boolean>
```

Псевдонимы типа

```
open class Teacher: Resource() {  
    ...  
    val availability = Matrix<Boolean>(...)  
}  
  
typealias AvailabilityTable = Matrix<Boolean>
```

```
schedule {  
    data {  
        startFrom("08:00")  
        subjects("Russian",  
                "Literature")  
        student {  
            name = "Ivanov"  
            allSubjects()  
        }  
        teacher {  
            availability {  
                monday("08:00")  
                wednesday("09:00", "16:00")  
            }  
        }  
    }  
    assertions {  
        // проверки на Kotlin  
    }  
}
```

Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
fun Teacher.availability(init: AvailabilityTable.() -> Unit) {
    this.availability.init()
}
```

Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
fun Teacher.availability(init: AvailabilityTable.() -> Unit) {
    this.availability.init()
}
```

Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
fun Teacher.availability(init: AvailabilityTable.() -> Unit) {
    this.availability.init()
}
```

Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
fun Teacher.availability(init: AvailabilityTable.() -> Unit) {
    this.availability.init()
}
```


Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
operator fun AvailabilityTable.invoke(init: AvailabilityTable.() -> Unit) {
    this.init()
}
```

Extension функции

```
teacher { // Teacher
    availability { //AvailabilityTable
        monday("08:00")
        wednesday("09:00", "16:00")
    }
}
```

```
operator fun AvailabilityTable.invoke(init: AvailabilityTable.() -> Unit) {
    this.init()
}
```

Extension функции

```
fun String?.isNullOrEmpty() = ...
```

```
val a: String? = null
```

```
if(a.isNullOrEmpty()) {  
    println("Something is wrong")  
}
```

Extension функции

```
fun String?.isEmpty() = ...
```

```
val a: String? = null
```

```
if(a.isEmpty()) {  
    println("Something is wrong")  
}
```

Extension функции

```
fun String?.isEmpty() = ...
```

```
val a: String? = null
```

```
if(a.isEmpty()) {  
    println("Something is wrong")  
}
```

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом A и расширять класс B;
- заманчивая возможность прикрутить операторы.

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом A и расширять класс B;
- заманчивая возможность прикрутить операторы.

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом A и расширять класс B;
- заманчивая возможность прикрутить операторы.

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом A и расширять класс B;
- заманчивая возможность прикрутить операторы.

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом **A** и **расширять** класс **B**;
- заманчивая возможность прикрутить операторы.

Extension функции. Резюме

- доступ только к публичному API;
- не модифицируют класс;
- определяется по типу переменной, а не по фактическому типу;
- члены класса приоритетнее;
- может быть одновременно частью контекста с классом A и расширять класс B;
- заманчивая возможность прикрутить операторы.

Infix функции

```
fun main(args: Array<String>) {  
    Очередной опасный молоток в руках  
}
```

Infix функции

```
fun main(args: Array<String>) {  
    Очередной опасный молоток в руках  
}
```

Infix функции

```
val a: Int = ...
```

```
a.shouldBeEqual(42)
```

```
infix fun Int.shouldBeEqual(expected: Int){  
    ...  
}
```

Infix функции

```
val a: Int = ...
```

```
a.shouldBeEqual(42)
```

```
infix fun Int.shouldBeEqual(expected: Int){  
    ...  
}
```

Infix функции

```
val a: Int = ...
```

```
a.shouldBeEqual(42)
```

```
infix fun Int.shouldBeEqual(expected: Int){  
    ...  
}
```


Infix функции

```
val a: Int = ...
```

```
a shouldBeEqual 42
```

```
infix fun Int.shouldBeEqual(expected: Int){  
    ...  
}
```

```
schedule { //SchedulingContext
    data {
        startFrom("08:00")
        subjects("Russian",
                "Literature")
        student {
            name = "Ivanov"
            allSubjects()
        }
        teacher {
            availability {
                monday("08:00")
                wednesday("09:00", "16:00")
            }
        }
    }
} assertions {
    // проверки на Kotlin
}
}
```

Infix функции

```
class SchedulingContext {  
    fun data(init: DataContext.() -> Unit): SchedulingResults {  
        ...  
    }  
}  
  
infix fun SchedulingResults.assertions(init: AssertionsContext.() -> Unit){  
    ...  
}  
  
schedule {  
    data {  
        ...  
    } assertions {  
        ...  
    }  
}
```

Infix функции

```
class SchedulingContext {  
    fun data(init: DataContext.() -> Unit): SchedulingResults {  
        ...  
    }  
}  
  
infix fun SchedulingResults.assertions(init: AssertionsContext.() -> Unit){  
    ...  
}  
  
schedule {  
    data {  
        ...  
    } assertions {  
        ...  
    }  
}
```

Infix функции

```
class SchedulingContext {  
    fun data(init: DataContext.() -> Unit): SchedulingResults {  
        ...  
    }  
}  
  
infix fun SchedulingResults.assertions(init: AssertionsContext.() -> Unit){  
    ...  
}  
  
schedule {  
    data {  
        ...  
    } assertions {  
        ...  
    }  
}
```

Infix функции

```
class SchedulingContext {  
    fun data(init: DataContext.() -> Unit): SchedulingResults {  
        ...  
    }  
}  
  
infix fun SchedulingResults.assertions(init: AssertionsContext.() -> Unit){  
    ...  
}  
  
schedule {  
    data {  
        ...  
    } assertions {  
        ...  
    }  
}
```

subject action object

Infix функции

```
data {  
  ...  
} assertions {  
  ...  
}
```


Infix функции

```
data {  
  ...  
} assertions {  
  ...  
}
```

Infix функции

```
data {  
  ...  
} assertions {  
  ...  
}
```

Infix функции. Резюме

- избавляемся от “шумного” синтаксиса;
- должен быть субъект и объект;
- модификатор `infix`;
- точно один параметр;
- можно передавать лямбды.

Infix функции. Резюме

- избавляемся от “шумного” синтаксиса;
- должен быть **субъект** и **объект**;
- модификатор `infix`;
- точно один параметр;
- можно передавать лямбды.

Infix функции. Резюме

- избавляемся от “шумного” синтаксиса;
- должен быть субъект и объект;
- модификатор **infix**;
- точно один параметр;
- можно передавать лямбды.

Infix функции. Резюме

- избавляемся от “шумного” синтаксиса;
- должен быть субъект и объект;
- модификатор `infix`;
- точно **один** параметр;
- можно передавать лямбды.

Infix функции. Резюме

- избавляемся от “шумного” синтаксиса;
- должен быть субъект и объект;
- модификатор `infix`;
- точно один параметр;
- можно передавать лямбды.

Демо

Infix, Extension, Type Aliases, Destructuring Declarations

Контроль контекста

Контроль контекста до Kotlin 1.1

```
schedule { //SchedulingContext
    data { //DataContext + SchedulingContext
        data { } //отсутствие контроля контекста
    }
}
```

```
class SchedulingContext {
    fun data(init: DataContext.() -> Unit) { ... }
}
```

```
class DataContext {
    @Deprecated("Incorrect context", level = DeprecationLevel.ERROR)
    fun data(init: DataContext.() -> Unit) {}
}
```

Контроль контекста до Kotlin 1.1

```
schedule { //SchedulingContext
    data { //DataContext + SchedulingContext
        data { } //отсутствие контроля контекста
    }
}
```

```
class SchedulingContext {
    fun data(init: DataContext.() -> Unit) { ... }
}
```

```
class DataContext {
    @Deprecated("Incorrect context", level = DeprecationLevel.ERROR)
    fun data(init: DataContext.() -> Unit) {}
}
```

Контроль контекста до Kotlin 1.1

```
schedule { //SchedulingContext
    data { //DataContext + SchedulingContext
        data { } //отсутствие контроля контекста
    }
}
```

```
class SchedulingContext {
    fun data(init: DataContext.() -> Unit) { ... }
}
```

```
class DataContext {
    @Deprecated("Incorrect context", level = DeprecationLevel.ERROR)
    fun data(init: DataContext.() -> Unit) {}
}
```

Контроль контекста до Kotlin 1.1

```
schedule { //SchedulingContext
    data { //DataContext + SchedulingContext
        data { } //не компилируется
    }
}
```

```
class SchedulingContext {
    fun data(init: DataContext.() -> Unit) { ... }
}
```

```
class DataContext {
    @Deprecated("Incorrect context", level = DeprecationLevel.ERROR)
    fun data(init: DataContext.() -> Unit) {}
}
```

Контроль контекста после Kotlin 1.1

@DslMarker

annotation class MyCustomDslMarker

Контроль контекста после Kotlin 1.1

```
@MyCustomDslMarker  
class SchedulingContext {  
  
    fun data(init: DataContext.() -> Unit) { ... }  
  
}
```

```
@MyCustomDslMarker  
class DataContext { }
```

Контроль контекста после Kotlin 1.1

```
@MyCustomDslMarker
class SchedulingContext {

    fun data(init: DataContext.() -> Unit) { ... }

}
```

```
@MyCustomDslMarker
class DataContext { }

schedule {
    data {
        data { } //не компилируется
    }
}
```


Особый случай контроля контекста

```
schedule {  
  data {  
    student {  
      name = "Ivanov"  
    }  
    . . .  
  }  
}
```

Особый случай контроля контекста

```
schedule {  
  data {  
    student {  
      student { } //опять та же проблема с контекстом Student  
    }  
    . . .  
  }  
}
```

Решение особого случая контроля контекста

1. Использовать дополнительный контекст `StudentContext`;
2. Создать интерфейсы для всех сущностей (`IStudent`), реализующие их контексты с использованием делегатов

```
@MyCustomDs1Marker
```

```
class StudentContext(val owner: Student = Student()): IStudent by owner
```

3. `@Deprecated`

```
@Deprecated("Incorrect context", level = DeprecationLevel.ERROR)
```

```
fun Identifiable.student(init:() -> Unit) {}
```

Решение особого случая контроля контекста

1. Использовать дополнительный контекст StudentContext;
2. Создать интерфейсы для всех сущностей (IStudent), реализующие их контексты с использованием делегатов

@MyCustomDs1Marker

```
class StudentContext(val owner: Student = Student()): IStudent by owner
```

3. @Deprecated

```
@Deprecated("Incorrect context", level = DeprecationLevel.ERROR)  
fun Identifiable.student(init:() -> Unit) {}
```

Решение особого случая контроля контекста

1. Использовать дополнительный контекст `StudentContext`;
2. Создать интерфейсы для всех сущностей (`IStudent`), реализующие их контексты с использованием делегатов

```
@MyCustomDs1Marker  
class StudentContext(val owner: Student = Student()): IStudent by owner
```

3. @Deprecated

```
@Deprecated("Incorrect context", level = DeprecationLevel.ERROR)  
fun Identifiable.student(init:() -> Unit) {}
```

Особый случай контроля контекста

```
schedule {  
  data {  
    student {  
      student { } //не компилируется  
    }  
    ...  
  }  
}
```

Контроль контекста. Резюме

- защищайте от ошибок пользователей вашего DSL;
- аннотация `@DslMarker` сокращает много рутины;
- `@Deprecated(level = ERROR)` решает нерешаемые проблемы.

Контроль контекста. Резюме

- защищайте от ошибок пользователей вашего DSL;
- аннотация `@DslMarker` сокращает много рутины;
- `@Deprecated(level = ERROR)` решает нерешаемые проблемы.

Контроль контекста. Резюме

- защищайте от ошибок пользователей вашего DSL;
- аннотация `@DslMarker` сокращает много рутины;
- `@Deprecated(level = ERROR)` решает нерешаемые проблемы.

Демо: контроль контекста

Минусы использования DSL

- переиспользование части DSL;
- следите за `this` и `it`;
- вложенность может стать проблемой;
- документация.

Минусы использования DSL

- переиспользование части DSL;
- следите за `this` и `it`;
- вложенность может стать проблемой;
- документация.

Минусы использования DSL

- переиспользование части DSL;
- следите за `this` и `it`;
- вложенность может стать проблемой;
- документация.

Минусы использования DSL

- переиспользование части DSL;
- следите за `this` и `it`;
- вложенность может стать проблемой;
- документация.

Заключение

- отличайте гвозди от шурупов;
- изучайте язык;
- тренируйтесь на “кошках”;
- предварительно проектируйте DSL.

Заключение

- отличайте гвозди от шурупов;
- изучайте язык;
- тренируйтесь на “кошках”;
- предварительно проектируйте DSL.

Заключение

- отличайте гвозди от шурупов;
- изучайте язык;
- тренируйтесь на “кошках”;
- предварительно проектируйте DSL.

Заключение

- отличайте гвозди от шурупов;
- изучайте язык;
- тренируйтесь на “кошках”;
- предварительно проектируйте DSL.

name Ivan Osipov

telegram ivan_osipov

twitter _osipov_

viber

whatsapp /dev/null



i-osipov.ru

github.com/ivan-osipov/kotlin-dsl-example