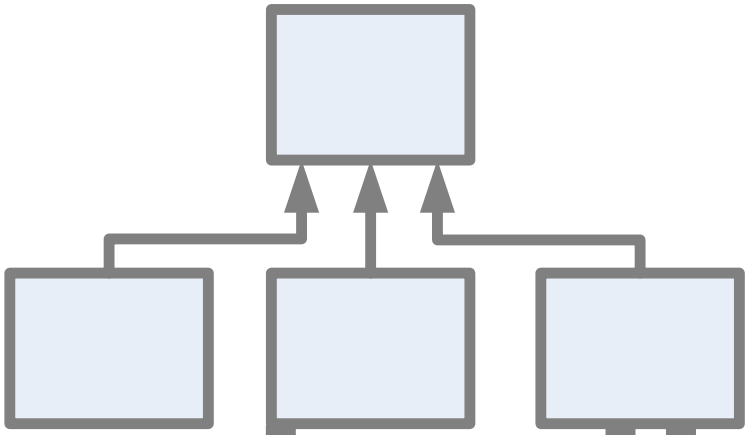


Functions Reference

1. Введение

Данный документ описывает фабричные классы (далее по тексту они называются фабриками функций), которые используются при «сборке» функций.

Фабрики функций находятся в пакете **com.vyhodb.f** (архив **vdb-core-0.9.0.jar**).

Фабрика	Краткое описание
CommonFactory	Общие функции.
PredicateFactory	Предикаты – функции возвращающие Boolean.
NavigationFactory	Обход children, parent записей, результатов поиска по индексу. Обход иерархии записей.
PrintFactory	Печать дерева записей в соответствии с алгоритмом обхода.
RecordFactory	Работа с записью: получение/установка значения полей, получение/установка родительской записи, и т.д.
AggregateFactory	Агрегирующие функции: sum(), min(), max(), count().
CollectionFactory	Функции по работе с коллекцией, размещенной в контексте.
StringFactory	Функции работы со строками.

Для каждого метода фабрики, в данном руководстве, приведены: сигнатура фабричного метода, описание метода, пример использования метода.

Данный документ входит в пакет документации vyhodb, состав которого представлен в следующей таблице:

Документ	Описание
Getting Started	Быстрый старт. Документ даёт представление о vyhodb API на простых примерах без детального описания.
Developer Guide	Руководство разработчика. Описывает различные vyhodb API и их использование.
Functions Reference	Справочник по функциям Functions API
Administrator Guide	Руководство администратора. Описывает архитектуру vyhodb, её конфигурирование и администрирование.

1.1. Примеры кода и их использование

Все примеры в данном руководстве можно разделить на три группы. Обратите внимание, что для использования примеров второй и третьей группы необходимо внести изменения в класс com.vyhodb.freference.Example.

1.1.1. Первая группа примеров

Первая группа примеров – самая простая. Она не нуждается, в каких либо данных, запущенном сервере vyhodb. Данные примеры передают null в качестве текущего объекта для исполнения дерева функций. Подобные примеры используются для иллюстрации фабрик CommonFactory, StringFactory, частично PredicateFactory. Например:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

public class Current {

    public static void main(String[] args) {
        System.out.println( current().eval(null) );
```

```
        System.out.println( current().eval("Hello") );
        System.out.println( current().eval(12) );
        System.out.println( current().eval(false) );
    }
}
```

1.1.2. Вторая группа примеров

Вторая группа примеров нуждается в рабочем vyhodb сервере, запущенном в embedded режиме. Данные примеры унаследованы от класса com.vyhodb.freference.Example:

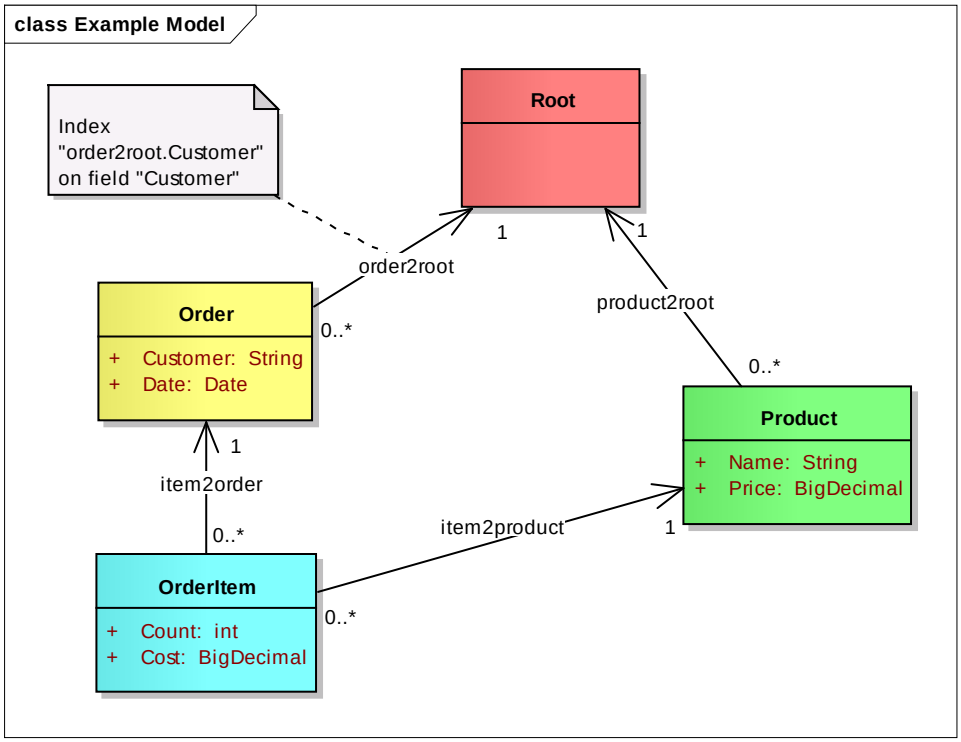
```
package com.vyhodb.freference;

import java.io.IOException;
import java.util.Properties;

import com.vyhodb.f.F;
import com.vyhodb.server.Server;
import com.vyhodb.server.TrxSpace;
import com.vyhodb.space.Record;
import com.vyhodb.utils.DataGenerator;

public abstract class Example {

    public static final String LOG = "C:\\vyhodb-0.9.0\\storage\\vyhodb.log";
    public static final String DATA = "C:\\vyhodb-0.9.0\\storage\\vyhodb.data";

    protected void run() throws IOException {
        Properties props = new Properties();
        props.setProperty("storage.log", LOG);
        props.setProperty("storage.data", DATA);

        try (Server server = Server.start(props)) {
            TrxSpace space = server.startModifyTrx();
            Record root = space.getRecord(0L);
            generateTestData(root);

            getF().eval(root);

            space.rollback();
        }
    }

    protected abstract F getF();

    protected void generateTestData(Record root) {
        DataGenerator.generate(root);
    }
}
```

Классы подобных примеров реализуют метод getF() который возвращает функцию. Данная функция исполняется в классе Example, при этом открывается modify транзакция и функции передаётся корневая запись (id==0). После исполнения функции, транзакция откатывается.

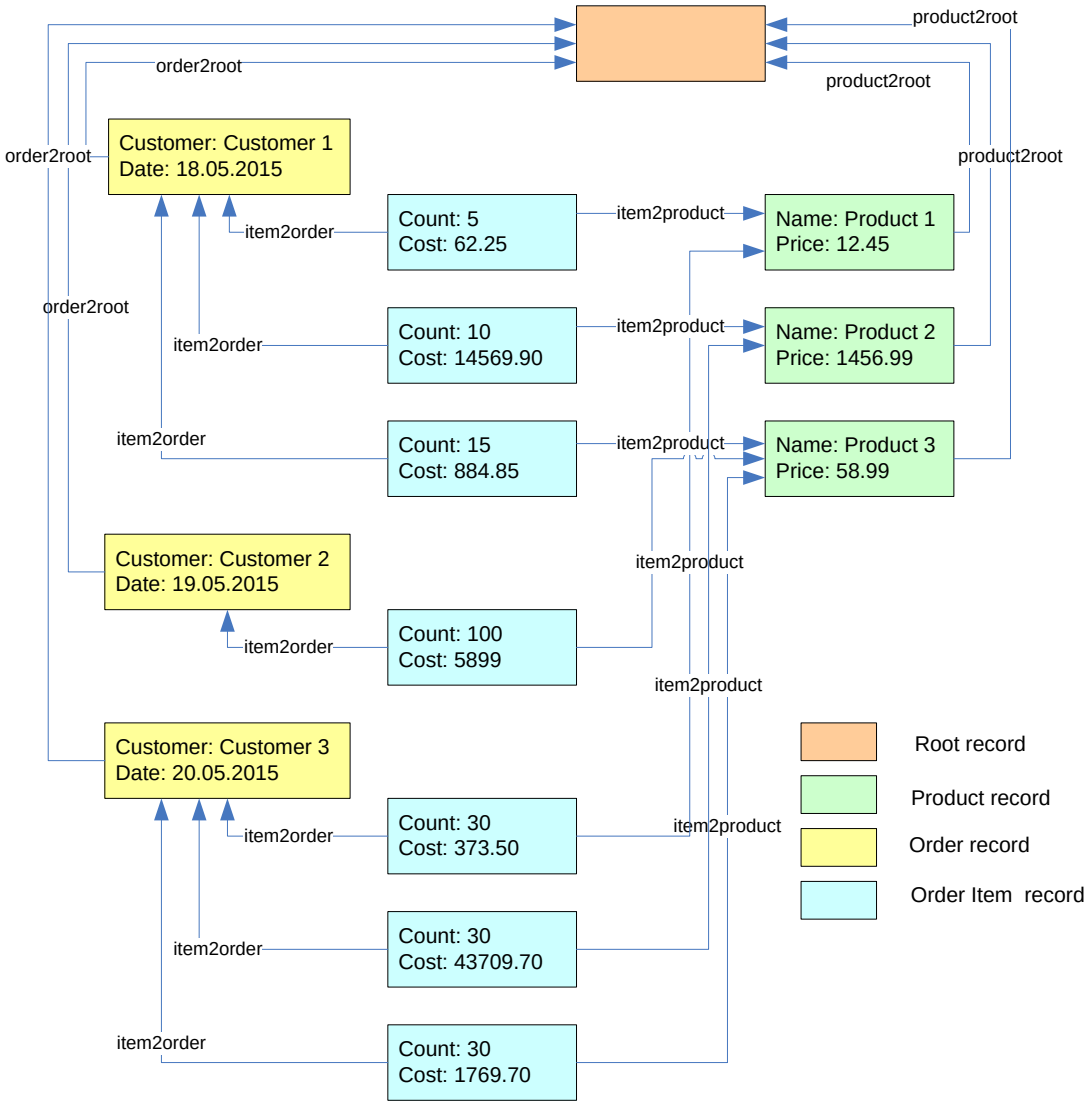
Для корректной работы данных примеров необходимо в классе Example прописать пути к файлам данных и лога базы данных, которую вы собираетесь использовать.

По умолчанию, перед исчислением функции, в рамках транзакции создаётся тестовый набор данных методом **com.vyhodb.utils.DataGenerator#generate()** (методы класса **com.vyhodb.utils.DataGenerator** также используются в примерах кода к документам “Getting Started”, “Developer Guide”).

Модель данных тестового набора:



На диаграмме ниже показаны записи, поля и линки, генерируемые методом **com.vyhodb.DataGenerator#generate()**:



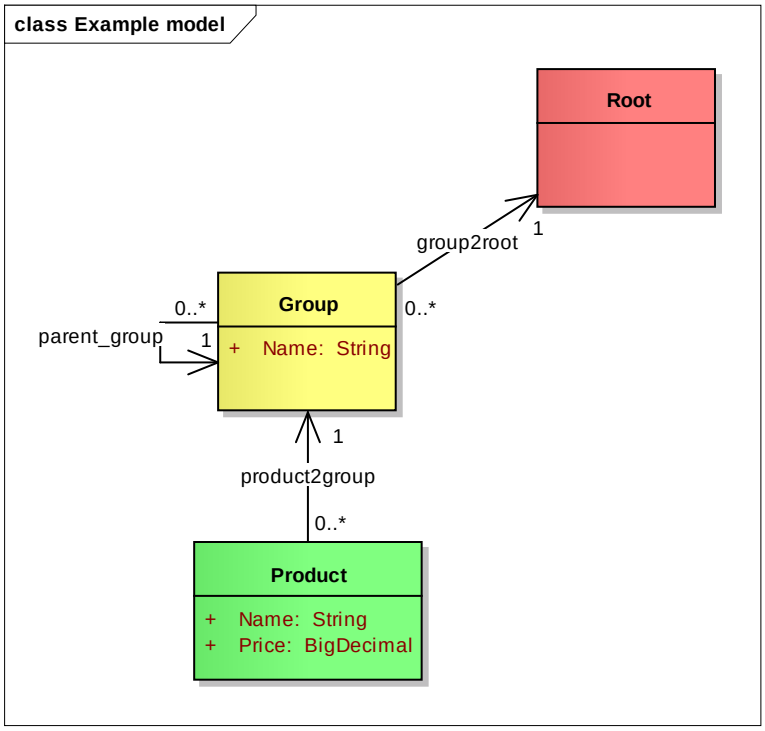
1.2.3. Третья группа примеров

Третья группа примеров используется для иллюстрации обхода иерархий и унаследована от класса com.vyhodb.freference.HierarchyExample:

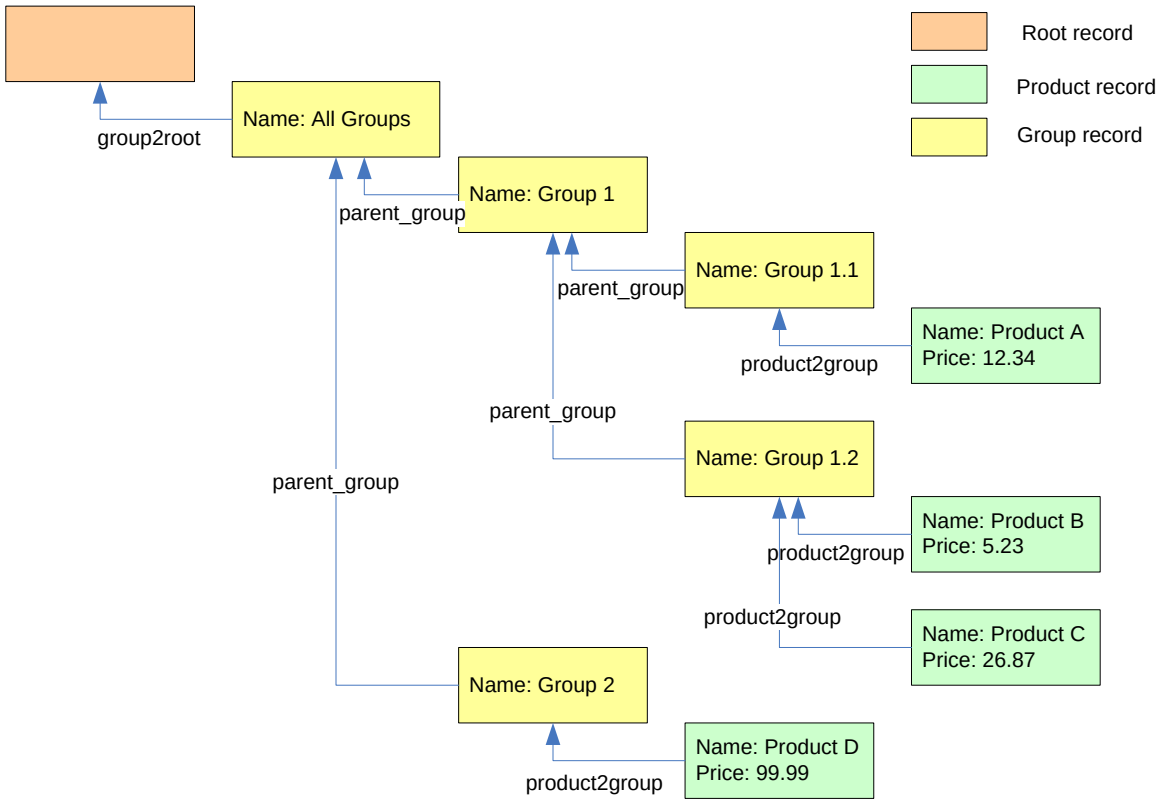
```
package com.vyhodb.freference;

import com.vyhodb.DataGenerator;
import com.vyhodb.space.Record;

public abstract class HierarchyExample extends Example {

    @Override
    protected void generateTestData(Record root) {
        DataGenerator.generateHierarchy(root);
    }
}
```

Главное отличие третьей группы примеров от предыдущей заключается модели и наборе данных, который генерируется после открытия транзакции. В данном случае для создания тестового набора данных используется метод **com.vyhodb.utils.DataGenerator#generateHierarchy()**.

Модель данных:



Данные:



2. CommonFactory

Фабрика содержит конструкторы базовых функций.

1.1. nil()

```
public static final F nil()
```

Функция возвращает null. Имя функции навеяно функцией nil языка LISP.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

public class Nil {

    public static void main(String[] args) {
        System.out.println( nil().startEval(null) );
    }
}
```

Результат:

```
null
```

1.2. c()

```
public static F c(Object value)
```

Функция каждый раз, в результате своего исполнения, возвращает один и тот же объект, переданный ей при создании.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

public class C {

    public static void main(String[] args) {
        System.out.println( c("Hello").startEval(null) );
        System.out.println( c(42).startEval(null) );
        System.out.println( c(true).startEval(null) );
        System.out.println( c(null).startEval(null) );
    }
}
```

Результат:

```
Hello
42
true
null
```

1.3. current()

```
public static F current()
```

Функция возвращает текущий объект, который был ей передан в качестве аргумента метода evalTree().

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

public class Current {

    public static void main(String[] args) {
        System.out.println( current().startEval(null) );
        System.out.println( current().startEval("Hello") );
        System.out.println( current().startEval(12) );
        System.out.println( current().startEval(false) );
    }
}
```

Результат:

```
null
Hello
12
false
```

1.4. composite()

Данный фабричный метод очень важен, так как используется почти во всех фабричных методах.

Он получает на вход массив функций (задаётся в виде F… next) и возвращает одну функцию. Метод composite() имеет свою логику, которая не ограничивается лишь созданием объекта функции:

```
    public final static F composite(F... functions) {
        if (functions == null) return nil();

        switch(functions.length) {
            case 0:
                return nil();

            case 1:
                return functions[0];

            default:
                return new Composite(functions);
        }
    }
```

Сама функция com.vyhodb.f.common.Composite последовательно исполняет каждую функцию из переданного ей массива functions.

Функция возвращает результат исполнения последней функции в массиве functions.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

import com.vyhodb.f.F;

public class Composite {

    public static void main(String[] args) {
        F nilF = composite();
        F oneF = composite(c("One"));
        F twoF = composite(c("One"), c("Two"));
```

```
        System.out.println( nilF.startEval(null) );
        System.out.println( oneF.startEval(null) );
        System.out.println( twoF.startEval(null) );
    }
}
```

Результат:

```
null
One
Two
```

1.5. put(), get(), clear()

Данные функции предназначены для занесения значения в контекст, чтения значения из контекста и очищения значения в контексте.

```
public static F put(String contextKey, F value)

public static F put(String contextKey, Object value)
```

Функция заносит в контекст значение, возвращаемое функцией value или определяемое как константа Object, по ключу contextKey.

Функция возвращает значение value.

```
public static F get(String contextKey)
```

Функция читает из контекста значение по ключу contextKey и возвращает его.

```
public static F clear(String contextKey)
```

Функция очищает значение в контексте по ключу contextKey. Функция возвращает предыдущее значение контекста по ключу contextKey.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

import com.vyhodb.f.F;

public class PutGetClear {

    public static void main(String[] args) {
        F f = composite(
            print( put("Some key", 42) ),
            print( get("Some key") ),
            print( clear("Some key") ),
            print( get("Some key") )
        );

        f.startEval(null);
    }
}
```

Результат:

```
42
42
42
```

```
null
```

1.6. _if_else(), _if()

```
public static F _if_else(Predicate predicate, F trueF, F falseF)
```

В зависимости от результата предиката predicate функция исполняет одну из функций (trueF, falseF) и возвращает её результат.

```
public static F _if(Predicate predicate, F... trueF)
```

В случае положительного результата предиката predicate, функция _if вызывает функцию(-и) trueF и возвращается её результат. В случае отрицательного результата предиката predicate, функция _if возвращает null и не вызывает trueF.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import com.vyhodb.f.F;

public class If {

    public static void main(String[] args) {
        F f1 =
        _if_else(
            falseF(),
            print("True"),
            print("False")
        );

        F f2 =
        _if(
            trueF(),
            print("True")
        );

        f1.startEval(null);
        f2.startEval(null);
    }
}
```

Результат:

```
False
True
```

1.7. _case(), when()

Пара функций _case(), when() используется вместе для замены значения по условию.

```
public static When when(Object when, Object then)
```

When не является функцией, хотя и имеет метод eval(Object value) который используется _case для тестирования и замены значения:

```
public Object eval(Object value)
```

Логика действий метода eval() следующая: если объект value эквивалентен (equals()) объекту when, тогда возвращается объект then; иначе – возвращается объект value.

```
public static F _case(F valueF, When... whens)
```

Функция вызывает valueF. Полученный результат передаётся первому объекту when. Результат первого объекта when передаётся на вход второму объекту when, и т.д. Функция _case возвращает результат последнего when.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

import com.vyhodb.f.F;

public class CaseWhen {

    public static void main(String[] args) {
        F f =
        _case(c("Hello"),
            when("Hello", "Buy"),
            when("Buy", "Question"),
            when("Question", 42)
        );

        System.out.println( f.startEval(null));
    }
}
```

Результат:

```
42
```

1.8. _caseNull()

```
public static F _caseNull(F valueF, Object nullReplaceValue)
```

Функция возвращает nullReplaceValue в случае если valueF == null. Иначе – возвращается valueF.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

public class CaseNull {

    public static void main(String[] args) {
        System.out.println( _caseNull(c(null), "Null value").startEval(null) );
        System.out.println( _caseNull(c(265), "Null value").startEval(null) );
    }
}
```

Результат:

```
Null value
265
```

1.9. omitContext()

```
public static F omitContext(String contextKey, F... next)
```

Функция удаляет из контекста значение по ключу contextKey и исполняет функцию(-и) next. После исполнения next, функция omitContext восстанавливает в контексте удалённое значение с ключом contextKey.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

import com.vyhodb.f.F;

public class OmitContext {

    public static void main(String[] args) {
        F f =
        composite(
            put("Key", 42),

            omitContext("Key",
                print(get("Key")),
                put("Key", "Hello")
            ),

            print(get("Key"))
        );

        f.startEval(null);
    }
}
```

Результат:

```
null
42
```

1.10. loop()

```
public static Loop loop()
```

Несмотря на наличие фабричного метода, обычно функция создаётся её конструктором.

loop() используется для зацикливания функций. Смысл зацикливая заключается в том, что loop исполняет какую-либо функцию(-и) среди которых есть ссылка на сам объект Loop(). Функция, которая «зацикливается» передается не качестве конструкторе Loop() или её фабричном методе loop(), а устанавливается методом:

```
public void setLoop(F... loopF)
```

В нижеприведенном примере объявляется новая функция Decrement, которая уменьшает на 1 значение Long хранящееся в контексте:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;
```

```
import java.util.Map;

import com.vyhodb.f.F;
import com.vyhodb.f.common.Loop;

public class LoopExample {

    public static void main(String[] args) {
        Loop loop = new Loop();

        loop.setLoop(
                _if(less(c(0L), get("Dec")),
                    dec("Dec"),
                    print(get("Dec")),
                    loop
                )
        );

        F f =
        composite(
            put("Dec", 7L),
            loop
        );

        f.startEval(null);
    }

    public static F dec(String contextKey) {
        return new DecrementF(contextKey);
    }

    public static class DecrementF extends F {
        private String _contextKey;

        public DecrementF(String contextKey) {
            _contextKey = contextKey;
        }

        @Override
        public Object eval(Object current, Map<String, Object> context) {
            Number number = (Number) context.get(_contextKey);
            long result = number.longValue() - 1;
            context.put(_contextKey, result);
            return result;
        }
    }
}
```

Результат:

```
6
5
4
3
2
1
0
```

1.11. print()

```
public static F print(F valueF)

public static F print(Object value)
```

Выводит на экран (используя System.out.println()) значение value или результат исполнения функции valueF.

Пример:

```
package com.vyhodb.freference.common;

import static com.vyhodb.f.CommonFactory.*;

import com.vyhodb.f.F;

public class Print {

    public static void main(String[] args) {
        F f =
        composite(
            print("Hello"),
            print(c(42))
        );

        f.startEval(null);
    }
}
```

Результат:

```
Hello
42
```

1.12. printCurrent()

```
public static F printCurrent()
```

Выводит на экран текущий объект. В реальности является композицией следующих функций: print(current());

2. PredicateFactory

Фабрика предоставляет методы по созданию предикатов. Предикат – это функция, методы которой возвращают объект типа Boolean:

```
package com.vyhodb.f;

import java.util.Map;

public abstract class Predicate extends F {

    @Override
    public abstract Boolean evalTree(Object current, Map<String, Object> context);
}
```

Класс Predicate унаследован от F и поэтому является полноценной функцией.

Предикаты используются для проверки каких-либо условий. Так, например, предикаты используются в навигационных функциях для фильтрации посещаемых записей (см. **NavigationFactory**).

2.1. toBoolean()

```
public static Predicate toBoolean(F value)
```

Функция используется для приведения результата исчисления функции к типу Boolean.

Функция использует следующие правила преобразования значения, возвращаемого функцией value:

1) Если value() возвращает Boolean – то оно и возвращается
2) Если value() возвращает значение Number != 0, то возвращается true
3) Во всех остальных случаях возвращается false.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

public class ToBoolean {

    public static void main(String[] args) throws IOException {
        System.out.println( toBoolean(c(0)).startEval(null) );
        System.out.println( toBoolean(nil()).startEval(null));
        System.out.println( toBoolean(c("Hello")).startEval(null) );
        System.out.println( toBoolean(c(-0.1234)).startEval(null) );
        System.out.println( toBoolean(c(true)).startEval(null) );
    }
}
```

Результат:

```
false
false
false
true
true
```

2.2. trueF() and falseF()

Функции возвращают соответственно Boolean.TRUE и Boolean.FALSE.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

public class TrueAndFalse {

    public static void main(String[] args) throws IOException {
        System.out.println( trueF().startEval(null));
        System.out.println( falseF().startEval(null));
    }
}
```

Результат:

```
true
false
```

2.3. isNull and isNotNull()

```
public static Predicate isNull(F value)
```

Возвращает true, если результат value() == null, иначе возвращает false.

```
public static Predicate isNotNull(F value)
```

Возвращает true, если результат value() != null, иначе возвращает false.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

public class IsNull {

    public static void main(String[] args) throws IOException {
        System.out.println( isNull(nil()).startEval(null) );
        System.out.println( isNull(c("Hello")).startEval(null));
        System.out.println( isNotNull(nil()).startEval(null) );
        System.out.println( isNotNull(c("Hello")).startEval(null));
    }
}
```

Результат:

```
true
false
false
true
```

2.4. not(), and(), or()

```
public static Predicate not(Predicate predicate)
```

Функция реализует логическое отрицание (NOT).

```
public static Predicate and(Predicate... predicates)
```

Функция реализует логическое умножение (AND).

```
public static Predicate or(Predicate... predicates)
```

Функция реализует логическое сложение (OR).

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.PredicateFactory.*;

public class NotAndOr {

    public static void main(String[] args) {
        System.out.println( not( trueF()).startEval(null) );
        System.out.println( not( falseF()).startEval(null) );

        System.out.println( and( trueF(), trueF(), falseF()).startEval(null) );
        System.out.println( and( trueF(), trueF(), trueF()).startEval(null) );

        System.out.println( or( falseF(), falseF(), trueF()).startEval(null) );
        System.out.println( or( falseF(), falseF(), falseF()).startEval(null) );
    }
}
```

Результат:

```
false
true
false
true
true
false
```

2.5. equal()

```
public static Predicate equal(F... values)
```

Функция возвращает true в случае, когда все значения, возвращаемые функциями values, эквиваленты между собой. false – в другом случае.

Для тестирования эквивалентности используется метод **equals()**.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.PredicateFactory.*;
import static com.vyhodb.f.CommonFactory.*;

public class Equal {
    public static void main(String[] args) {
        System.out.println( equal( c("Hello"), c("Hello"), c("Hello")).startEval(null) );
```

```
        System.out.println( equal( c("Hello"), c("Hello"), c("")).startEval(null) );
    }
}
```

Результат:

```
true
false
```

2.6. fieldsEqual()

```
public static Predicate fieldsEqual(String[] keyFieldNames, Object... keyValues)

public static Predicate fieldsEqual(String keyFieldName, Object keyValue)
```

Функция преобразовывает текущий объект в запись и возвращает true если запись содержит поля с указанными именами и эквивалентными значениями (Object.equals()), иначе возвращает false.

Пример:

```
package com.vyhodb.freference.predicate;

import java.io.IOException;
import java.util.Date;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class FieldsEqual extends Example {

    public static void main(String[] args) throws IOException {
        new FieldsEqual().run();
    }

    @Override
    protected F getF() {
        String[] fields = {"Customer", "Date"};
        Object[] values = {"Customer 2", new Date(115, 4, 19)};

        return
        childrenIf("order2root", fieldsEqual(fields, values),
            printCurrent()
        );
    }
}
```

Результат:

```
{Customer="Customer 2", Date=[Tue May 19 00:00:00 BRT 2015]} id=45745
```

2.7. less(), lessEqual()

```
public static Predicate less(F... values)
```

Функция возвращает true, в случае когда: values[i]() < values[i+1](). False – иначе.

Значения, возвращаемые функциями values должны реализовывать интерфейс **java.lang.Comparable**.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class Less {
    public static void main(String[] args) {
        System.out.println( less( c(1), c(2), c(10)).startEval(null) );
        System.out.println( less( c(1), c(1), c(-3)).startEval(null) );
    }
}
```

Результат:

```
true
false
```

```
public static Predicate lessEqual(F... values)
```

Функция возвращает true, в случае когда: values[i]() <= values[i+1](). False – иначе.

Значения, возвращаемые функциями values должны реализовывать интерфейс **java.lang.Comparable**.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class LessEqual {
    public static void main(String[] args) {
        System.out.println( lessEqual( c(1), c(1), c(10)).startEval(null) );
        System.out.println( lessEqual( c(1), c(1), c(-3)).startEval(null) );
    }
}
```

Результат:

```
true
false
```

2.8. more(), moreEqual()

```
public static Predicate more(F... values)
```

Функция возвращает true, в случае когда: values[i]() > values[i+1](). False – иначе.

Значения, возвращаемые функциями values должны реализовывать интерфейс **java.lang.Comparable**.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class More {
    public static void main(String[] args) {
        System.out.println( more( c(10), c(5), c(1)).startEval(null) );
```

```
        System.out.println( more( c(10), c(10), c(100)).startEval(null) );
    }
}
```

Результат:

```
true
false
```

```
public static Predicate moreEqual(F... values)
```

Функция возвращает true, в случае когда: values[i]() >= values[i+1](). False – иначе.

Значения, возвращаемые функциями values должны реализовывать интерфейс **java.lang.Comparable**.

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class MoreEqual {
    public static void main(String[] args) {
        System.out.println( moreEqual( c(10), c(10), c(5)).startEval(null) );
        System.out.println( moreEqual( c(10), c(100), c(1000)).startEval(null) );
    }
}
```

Результат:

```
true
false
```

2.9. unique()

```
public static Predicate unique(F... values)
```

Функция возвращает true когда значения, возвращаемые функциями values уникальны между собой. Для проверки уникальности используется внутренний HashMap().

Пример:

```
package com.vyhodb.freference.predicate;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class Unique {
    public static void main(String[] args) {
        System.out.println( unique( c("Hello"), c(42), c(10)).startEval(null) );
        System.out.println( unique( c(42), c(42), c(100)).startEval(null) );
    }
}
```

Результат:

```
true
false
```

2.10. everyChild()

```
public static Predicate everyChild(String childrenLinkName, Predicate predicate)
```

Функция преобразовывает текущий объект к записи, получает дочерние записи с именем линка childrenLinkName, и для каждой дочерней записи вызывает predicate.

Если predicate хотя бы для одной дочерней записи возвращает false, то исчисление функции прерывается и возвращается false. Иначе (после обхода всех дочерних), функция возвращает true.

Пример:

```
package com.vyhodb.freference.predicate;

import java.io.IOException;
import java.util.Date;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.PredicateFactory.*;
import static com.vyhodb.f.RecordFactory.*;
import static com.vyhodb.f.CommonFactory.*;

public class EveryChild extends Example {

    public static void main(String[] args) throws IOException {
        new EveryChild().run();
    }

    @Override
    protected F getF() {
        Date moreDate = new Date(115, 4, 17);

        return
        print(
            everyChild("order2root",
                more(getField("Date"), c(moreDate))
            )
        );
    }
}
```

Результат:

```
true
```

2.11. someChildren()

```
public static Predicate someChildren(String childrenLinkName, Predicate predicate)
```

Функция преобразовывает текущий объект к записи, получает дочерние записи с именем линка childrenLinkName, и для каждой дочерней записи вызывает predicate.

Если predicate хотя бы для одной дочерней записи возвращает true, то выполнение функции прерывается и возвращается true. Иначе (после обхода всех дочерних), функция возвращает false.

Пример:

```
package com.vyhodb.freference.predicate;

import java.io.IOException;
import java.util.Date;
```

```
import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.PredicateFactory.*;
import static com.vyhodb.f.CommonFactory.*;

public class SomeChildren extends Example {

    public static void main(String[] args) throws IOException {
        new SomeChildren().run();
    }

    @Override
    protected F getF() {
        Date searchDate = new Date(115, 4, 18);

        return
        print(
            someChildren("order2root",
                fieldsEqual("Date", searchDate)
            )
        );
    }
}
```

Результат:

```
true
```

2.12. everySearch()

```
public static Predicate everySearch(String indexName, Criterion criterion, Predicate
predicate)
```

Функция преобразовывает текущий объект к записи, осуществляет поиск дочерних записей по индексу с именем indexName и критерием поиска criterion. Для каждой найденной дочерней записи вызывает predicate.

Если predicate хотя бы для одной найденной дочерней записи возвращает false, то выполнение функции прерывается и возвращается false. Иначе (после обхода всех дочерних записей), функция возвращает true.

Пример:

```
package com.vyhodb.freference.predicate;

import java.io.IOException;
import java.util.Date;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;
import com.vyhodb.space.CriterionFactory;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class EverySearch extends Example {

    public static void main(String[] args) throws IOException {
        new EverySearch().run();
    }

    @Override
```

```
    protected F getF() {
        Date moreDate = new Date(115, 4, 17);

        return
        print(
            everySearch("order2root.Customer", CriterionFactory.startsWith("Customer"),
                more(getField("Date"), c(moreDate))
            )
        );
    }
}
```

Результат:

```
true
```

2.13. someSearch()

```
public static Predicate someSearch(String indexName, Criterion criterion, Predicate predicate)
```

Функция преобразовывает текущий объект к записи, осуществляет поиск дочерних записей по индексу с именем indexName и критерием поиска criterion. Для каждой найденной дочерней записи вызывает predicate.

Если predicate хотя бы для одной найденной дочерней записи возвращает true, то выполнение функции прерывается и возвращается true. Иначе (после обхода всех дочерних записей), функция возвращает false.

Пример:

```
package com.vyhodb.freference.predicate;

import java.io.IOException;
import java.util.Date;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;
import com.vyhodb.space.CriterionFactory;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class SomeSearch extends Example {

    public static void main(String[] args) throws IOException {
        new SomeSearch().run();
    }

    @Override
    protected F getF() {
        Date searchDate = new Date(115, 4, 18);

        return
        print(
            someSearch("order2root.Customer", CriterionFactory.startsWith("Customer"),
                fieldsEqual("Date", searchDate)
            )
        );
    }
}
```

Результат:

```
true
```

3. NavigationFactory

Данная фабрика содержит методы, конструирующие функции навигации. Функции навигации получают текущий объект, преобразовывают его в vyhodb запись (Record), а далее получают дочерние/родительские записи, и для каждой из полученных записей вызывают следующую функцию.

Если у функции указан предикат, то он используется для проверки дочерней/родительской записи. В случае если предикат возвращает false, дочерняя/родительская запись не обрабатывается и следующие функции для данной записи не вызываются.

3.1. children()

```
public static F children(String linkName, F... next)
```

Функция преобразовывает текущий объект в запись, получает все дочерние записи по имени линка linkName. Для каждой дочерней записи вызывается функция(-и) next и дочерняя запись передается в качестве текущего объекта.

Функция возвращает результат исполнения next для последней дочерней записи. Функция возвращает null, в случае если не существует дочерних записей или next=nil().

3.2. childrenIf()

```
public static F childrenIf(String linkName, Predicate predicate, F... next)
```

Функция childrenIf() используется для указания предиката, который проверяет каждую дочернюю запись. Функция next при этом вызывается лишь для тех дочерних записей, для которых предикат вернул true.

Функция возвращает результат исполнения next для последней дочерней записи. Функция возвращает null, в случае если не существует дочерних записей или next=nil().

Пример:

```
package com.vyhodb.freference.navigation;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

public class Children extends Example {

    public static void main(String[] args) throws IOException {
        new Children().run();
    }

    @Override
    public F getF() {
        return
        childrenIf("order2root", fieldsEqual("Customer", "Customer 3"),
            children("item2order",
                printCurrent()
            )
```

```
        );
    }
}
```

Результат выполнения (id может отличаться):

```
{Cost=373.50, Count=30} id=2217
{Cost=43709.70, Count=30} id=2228
{Cost=1769.70, Count=30} id=2239
```

3.3. search()

```
public static F search(String indexName, Criterion criterion, F... next)
```

Функция преобразовывает текущий объект в запись, и осуществляет поиск дочерних записей, используя индекс с именем indexName и критерий criterion.

Для каждой найденной дочерней записи вызывается функция(-и) next и дочерняя запись передается в качестве текущего объекта.

Функция возвращает результат исполнения next для последней найденной дочерней записи. Функция возвращает null, в случае если не существует дочерних записей или next=nil().

Пример:

```
package com.vyhodb.freference.navigation;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;
import com.vyhodb.space.CriterionFactory;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

public class Search extends Example {

    public static void main(String[] args) throws IOException {
        new Search().run();
    }

    @Override
    public F getF() {
        return
        search("order2root.Customer", CriterionFactory.equal("Customer 3"),
            printCurrent()
        );
    }
}
```

Результат:

```
{Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195
```

3.4. searchIf()

```
public static F searchIf(String indexName, Criterion criterion, Predicate predicate, F...
next)
```

Функция преобразовывает текущий объект в запись, и осуществляет поиск дочерних записей, используя индекс с именем indexName и критерий criterion.

Каждая найденная дочерняя запись проверяется предикатом predicate, и в случае положительного результата (true) вызывается функция(-и) next а дочерняя запись передаётся в качестве текущего объекта.

Функция возвращает результат исполнения next для последней найденной дочерней записи. Функция возвращает null, в случае если не существует дочерних записей или next=nil().

Пример:

```
package com.vyhodb.freference.navigation;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;
import static com.vyhodb.f.RecordFactory.*;

import java.io.IOException;
import java.util.Date;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;
import com.vyhodb.space.CriterionFactory;

public class SearchIf extends Example {

    public static void main(String[] args) throws IOException {
        new SearchIf().run();
    }

    @Override
    public F getF() {
        Date moreDate = new Date("06/01/2015");

        return
        searchIf("order2root.Customer", CriterionFactory.equal("Customer 3"),
            more(getField("Date"), c(moreDate)),
            printCurrent()
        );
    }
}
```

Результат (пустой, так как найденная по индексу дочерняя запись отфильтровывается предикатом):

```

```

3.5. parent()

```
public static F parent(String linkName, F... next)
```

Функция преобразует текущий объект в запись, получает её родительскую запись но линку с именем linkName. Если родительская запись != null, то вызывает функцию(-и) next и передаёт родительскую запись в качестве текущего объекта.

Функция возвращает результат исчисления next.

Пример (печатает все product, на которые ссылаются order item-ы):

```
package com.vyhodb.freference.navigation;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
```

```
import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class Parent extends Example {

    public static void main(String[] args) throws IOException {
        new Parent().run();
    }

    @Override
    public F getF() {
        return
        children("order2root",
            children("item2order",
                parent("item2product",
                    printCurrent()
                )
            )
        );
    }
}
```

Результат (id могут отличаться):

```
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
{Name="Product 3", Price=58.99} id=2085
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
```

3.6. parentIf()

```
public static F parentIf(String linkName, Predicate predicate, F... next)
```

Функция преобразует текущий объект в запись, получает её родительскую запись по линку с именем linkName.

Если родительская запись != null, и предикат для родительской записи возвращает true, то вызывается функция(-и) next в которую передаётся родительская запись в качестве текущего объекта.

Функция возвращает результат исчисления next.

Пример (печатает все product с именем “Product 2”, на которые ссылаются order item-ы):

```
package com.vyhodb.freference.navigation;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class ParentIf extends Example {

    public static void main(String[] args) throws IOException {
        new ParentIf().run();
    }
```

```
    @Override
    public F getF() {
        return
        children("order2root",
            children("item2order",
                parentIf("item2product", fieldsEqual("Name", "Product 2"),
                    printCurrent()
                )
            )
        );
    }
}
```

Результат (id могут отличаться):

```
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 2", Price=1456.99} id=2074
```

3.7. _break()

```
public static F _break()
```

Функция прекращает работу ближайшей «родительской» функции children(), childrenIf(), search(), searchIf().

Функция используется для прекращения обработки дочерних записей. Это позволяет повысить производительность в случаях, когда нужная дочерняя запись найдена, и нет необходимости обрабатывать остальные дочерние записи.

Технически, функция выбрасывает исключение, которое обрабатывается ближайшей функцией children(), childrenIf(), search(), searchIf().

Увеличение производительности заключается в том, что каждая дочерняя запись читается с внешних носителей лишь при получении её объекта. Таким образом, сокращаются ненужные запросы во внешнюю память по чтению дочерних записей.

Пример:

```
package com.vyhodb.freference.navigation;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class Break extends Example {

    public static void main(String[] args) throws IOException {
        new Break().run();
    }

    @Override
    public F getF() {
        return
        children("order2root",
            children("item2order",
                printCurrent()
            ),
            _break()
```

```
        );
    }
}
```

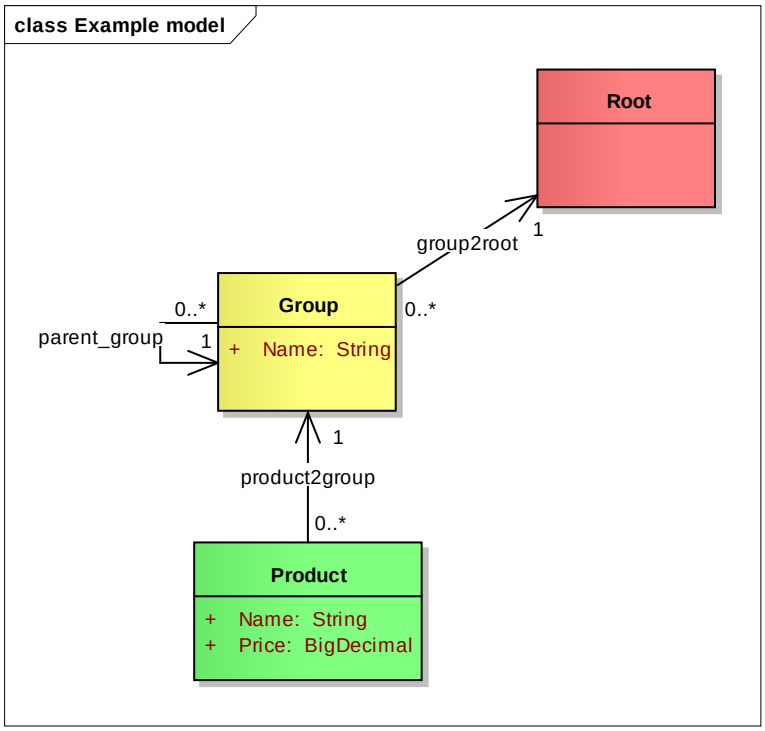
Результат (выводятся order item-ы лишь для первого order-а):

```
{Cost=62.25, Count=5} id=2129
{Cost=14569.90, Count=10} id=2140
{Cost=884.85, Count=15} id=2151
```

3.8. Обход иерархий

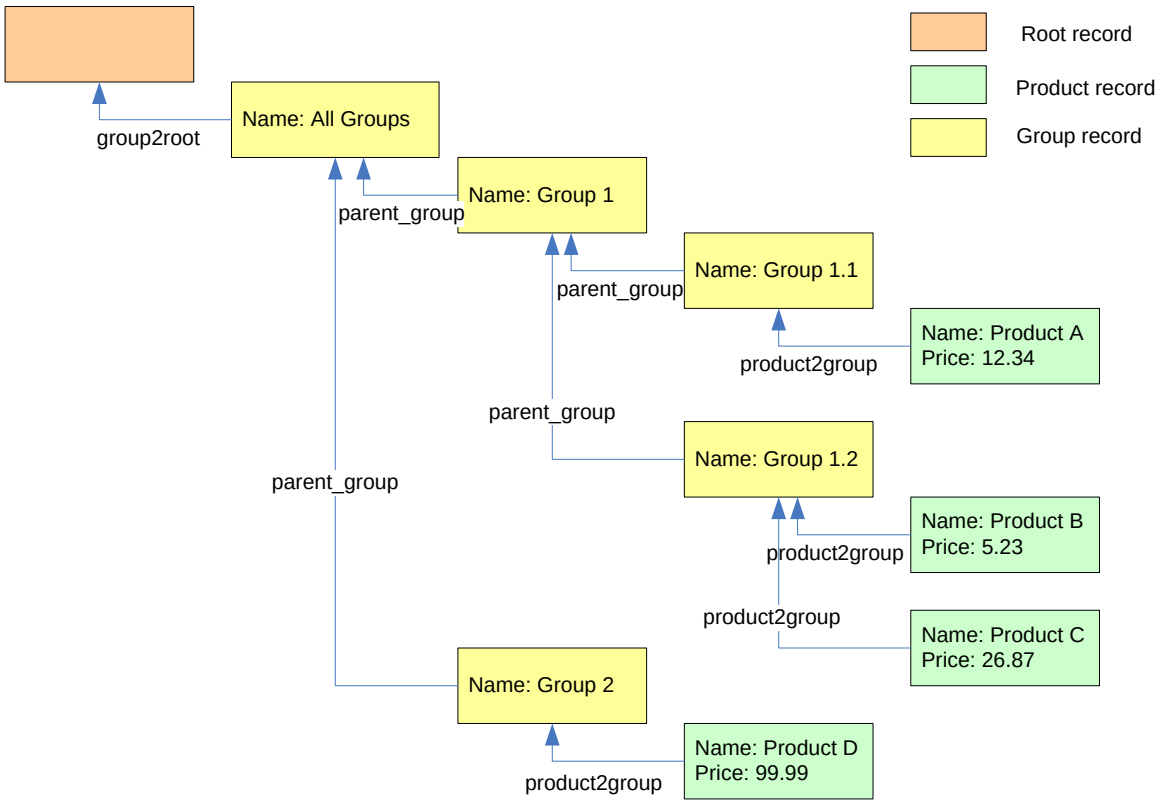
Иерархией называется структура записей, когда у одной записи существуют одновременно дочерние и родительская запись с одним и тем же именем линка.

Чтобы показать более наглядно, обратимся к модели данных, которую мы будем использовать в примерах данного раздела:



Записи **Group** образуют иерархию. Так у любой записи **Group** одновременно может быть родительская запись **Group** с линком “**parent_group**”, и дочерние записи **Group** с линком “**parent_group**”.

В классе **com.vyhodb.utils.DataGenerator** определен метод **generateHierarchy()**, который создаёт следующий тестовый набор данных в соответствии с вышеприведенной моделью:



В данном наборе данных жёлтые записи **Group** образуют иерархию. Именно этот набор данных мы и будем использовать в примерах.

3.8.1. hierarchy()

```
public static F hierarchy(String childLinkName, F... levelF)
```

Функция осуществляет обход иерархии сверху вниз. Функция преобразовывает текущий объект в запись и выполняет рекурсивно следующие шаги:

1) Для текущей записи вызывается функция(-и) levelF.
2) У текущей записи получаются дочерние записи с именем линка childLinkName
3) Для каждой дочерней записи итеративно выполняется данный алгоритм, начиная с пункта 1.

Функция возвращает результат последнего исполнения функции levelF. Так, для приведеднного ниже примера, hierarchy() возвратит запись с полем Name=”Group 2” (функция printCurrent() возвращает запись).

Пример (печать всех групп иерархии):

```
package com.vyhodb.freference.navigation.hierarchy;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.HierarchyExample;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

public class HierarchyGroup extends HierarchyExample {
```

```
    public static void main(String[] args) throws IOException {
        new HierarchyGroup().run();
    }

    @Override
    public F getF() {
        return
        children("group2root",
                hierarchy("parent_group",
                        printCurrent()
                )
        );
    }
}
```

Результат (id могут отличаться):

```
{Name="All Groups"} id=2063
{Name="Group 1"} id=2074
{Name="Group 1.1"} id=2085
{Name="Group 1.2"} id=2096
{Name="Group 2"} id=2107
```

Пример (печать всех продуктов):

```
package com.vyhodb.freference.navigation.hierarchy;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.HierarchyExample;

public class HierarchyProduct extends HierarchyExample {

    public static void main(String[] args) throws IOException {
        new HierarchyProduct().run();
    }

    @Override
    public F getF() {
        return
        children("group2root",
            hierarchy("parent_group",
                children("product2group",
                    printCurrent()
                )
            )
        );
    }
}
```

Результат (id могут отличаться):

```
{Name="Product A", Price=12.34} id=2118
{Name="Product B", Price=5.23} id=2129
{Name="Product C", Price=26.87} id=2140
{Name="Product D", Price=99.99} id=2151
```

3.8.2. hierarchyIf()

```
public static F hierarchyIf(String childLinkName, Predicate levelPredicate, F... levelF)
```

Функция осуществляет обход иерархии сверху вниз с использованием фильтрации записей иерархии.

Функция преобразовывает текущий объект в запись и выполняет следующие шаги итеративно:

1) Для текущей записи вызывается функция(-и) levelF.
2) У текущей записи получаются дочерние записи с именем линка childLinkName
3) Для каждой дочерней записи вызывается предикат levelPredicate.
4) В случае положительного результата levelPredicate, для дочерней записи итеративно выполняется данный алгоритм, начиная с пункта 1. В случае отрицательного результата, дочерняя запись пропускается и не участвует (включая её дочерние записи) в дальнейшем обходе иерархии.

Функция возвращает результат последнего исполнения функции levelF.

Пример:

```
package com.vyhodb.freference.navigation.hierarchy;

import static com.vyhodb.f.CollectionFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.RecordFactory.*;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import com.vyhodb.f.F;
import com.vyhodb.freference.HierarchyExample;

public class HierarchyIfGroup extends HierarchyExample {

    public static void main(String[] args) throws IOException {
        new HierarchyIfGroup().run();
    }

    @Override
    public F getF() {
        List<String> groupNames = Arrays.asList("All Groups", "Group 1.1", "Group 1.2", "Group
2");

        return
        composite(
            put("groups", groupNames),
            children("group2root",
                hierarchyIf("parent_group", collectionContains("groups", getField("Name")),
                    printCurrent()
                )
            )
        );
    }
}
```

Результат:

```
{Name="All Groups"} id=2063
{Name="Group 2"} id=2107
```

Обратите внимание, что, несмотря на то, что имена групп {“Group 1.1”, “Group 1.2”} включены в набор разрешенных, функция hierarchyIf() не обходит их, так как имя их родительской группы ”Group 1” не включено в набор, отфильтровывается и не участвует (включая дочерние записи) в дальнейшем обходе иерархии.

3.8.3. loop()

Так как обход иерархии представляет собой рекурсивный процесс, мы можем использовать функцию loop для конструирования функции по обходу иерархии.

Пример:

```
package com.vyhodb.freference.navigation.hierarchy;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.common.Loop;
import com.vyhodb.freference.HierarchyExample;

public class HierarchyLoop extends HierarchyExample {

    public static void main(String[] args) throws IOException {
        new HierarchyLoop().run();
    }

    @Override
    public F getF() {
        Loop loop = new Loop();

        loop.setLoop(
            printCurrent(),
            children("parent_group", loop)
        );

        return
        children("group2root",
            loop
        );
    }
}
```

Результат (id могут отличаться):

```
{Name="All Groups"} id=2063
{Name="Group 1"} id=2074
{Name="Group 1.1"} id=2085
{Name="Group 1.2"} id=2096
{Name="Group 2"} id=2107
```

3.9. Stack

Часто возникает ситуация когда необходимо для каждой записи, принимающей участие в обходе, выполнить какие-то действия. Один из подходов заключается в том, чтобы создавать новые функции, которые частично содержат логику children, search, parent, hierarchy. Другой подход заключается в использовании объекта Stack.

Объект Stack это объект, реализующий интерфейс **com.vyhodb.f.Stack**, который размещается в контексте, и методы которого вызываются функциями children, search, parent, hierarchy в процессе обхода записей. Другими словами, интерфейс Stack реализует паттерн Visitor.

Данный подход позволяет описывать логику обхода записей в виде стандартных функций children, index, parent, hierarchy, что существенно повышает понимание кода и его читаемость. Логика обработки записей, участвующих в обходе, выносится в класс, реализующий интерфейс Stack:

```
package com.vyhodb.f;

public interface Stack {

    public static final String DEFAULT_CONTEXT_KEY = "Sys$Stack";

    public void pushParent(String linkName, Object parent);

    public void pushChild(String linkName, Object child);

    public void pop();

    public Object peek();
}
```

Имя ключа контекста (по умолчанию) используемого для хранения объекта стека – “Sys$Stack”. Контекст не обязательно должен содержать объект Stack; функции children(), search(), parent(), hierarchy в этом случае выполняют обход без использования объекта Stack.

Перейдем к примерам. Мы создадим простой объект, реализующий интерфейс Stack и выводящий на экран parent и child объекты, а также проиллюстрируем использование данного стека в примерах обхода записей.

PrintStack:

```
package com.vyhodb.freference.navigation.stack;

import com.vyhodb.f.Stack;

public class PrintStack implements Stack {

    @Override
    public void pushParent(String linkName, Object parent) {
        System.out.println("Parent: " + parent);
    }

    @Override
    public void pushChild(String linkName, Object child) {
        System.out.println("Child: " + child);
    }

    @Override
    public void pop() {}

    @Override
    public Object peek() {
        return null;
    }
}
```

Пример: обход order, item, product с использованием PrintStack:

```
package com.vyhodb.freference.navigation.stack;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.Stack;
import com.vyhodb.freference.Example;

public class StackOrders extends Example {

    public static void main(String[] args) throws IOException {
```

```
        new StackOrders().run();
    }

    @Override
    protected F getF() {
        PrintStack stack = new PrintStack();

        return
        composite(
            put(Stack.DEFAULT_CONTEXT_KEY, stack),
            childrenIf("order2root", fieldsEqual("Customer", "Customer 2"),
                children("item2order",
                    parent("item2product")
                )
            )
        );
    }
}
```

Результат:

```
Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162
Child: {Cost=5899.00, Count=100} id=2184
Parent: {Name="Product 3", Price=58.99} id=2085
```

Пример: обход иерархии с использованием PrintStack:

```
package com.vyhodb.freference.navigation.stack;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.Stack;
import com.vyhodb.freference.HierarchyExample;

public class StackHierarchy extends HierarchyExample {

    public static void main(String[] args) throws IOException {
        new StackHierarchy().run();
    }

    @Override
    protected F getF() {
        PrintStack stack = new PrintStack();

        return
        composite(
            put(Stack.DEFAULT_CONTEXT_KEY, stack),
            children("group2root",
                hierarchy("parent_group")
            )
        );
    }
}
```

Результат:

```
Child: {Name="All Groups"} id=2063
Child: {Name="Group 1"} id=2074
Child: {Name="Group 1.1"} id=2085
Child: {Name="Group 1.2"} id=2096
Child: {Name="Group 2"} id=2107
```

3.9.1. Stack, предикаты и _if()

С точки зрения использования Stack существует большая разница между функцией навигации с предикатом (-If()) и функцией _if().

Покажем на примерах и прокомментируем это.

В обоих примерах выполняем обход Order и фильтруем их по имени Customer. В первом случае используем функцию childrenIf(), во втором children() и _if().

Пример 1.

```
package com.vyhodb.freference.navigation.stack;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.Stack;
import com.vyhodb.freference.Example;

public class StackPredicate extends Example {

    public static void main(String[] args) throws IOException {
        new StackPredicate().run();
    }

    @Override
    protected F getF() {
        PrintStack stack = new PrintStack();

        return
        composite(
            put(Stack.DEFAULT_CONTEXT_KEY, stack),
            childrenIf("order2root", fieldsEqual("Customer", "Customer 2"))
        );
    }
}
```

Результат:

```
Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162
```

Пример 2:

```
package com.vyhodb.freference.navigation.stack;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PredicateFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.Stack;
import com.vyhodb.freference.Example;

public class StackIf extends Example {

    public static void main(String[] args) throws IOException {
        new StackIf().run();
    }

    @Override
    protected F getF() {
        PrintStack stack = new PrintStack();

        return
        composite(
            put(Stack.DEFAULT_CONTEXT_KEY, stack),
```

```
            children("order2root",
                _if(fieldsEqual("Customer", "Customer 2"))
            )
        );
    }
}
```

Результат:

```
Child: {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096
Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162
Child: {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195
```

Как мы видим, первый пример выводит на печать лишь одну запись Order тогда как второй все 3 записи, не смотря на то, что мы используем один и тот же предикат фильтрации.

Дело в том, что методы Stack вызываются лишь в случае успешного исполнения предиката функций childrenIf(), parentIf(), searchIf(), hierarchyIf().

В нашем первом примере, при исполнении функции childrenIf(“order2root”, fieldsEqual("Customer", "Customer 2")) лишь запись с полем Customer=”Customer 2” удовлетворяет предикату и соответственно лишь для этой записи будет вызван метод **com.vyhodb.f.Stack#pushChild(**).

Во втором примере, children(“order2root”, …) не имеет предиката, и для каждой дочерней записи “order2root” будет вызван **com.vyhodb.f.Stack#pushChild().**

4. PrintFactory

Фабричные методы класса com.vyhodb.f.PrintFactory используются для вывода графа vyhodb записей в строку. Граф vyhodb записей, который необходимо вывести на печать, задаётся навигационными функциями, которые «оборачиваются» функцией печати из фабрики PrintFactory.

В фабрике определены два типа функций для разных форматов вывода: так называемый “простой” формат и json.

Каждая из функций печати создаёт свой объект Stack, который и реализует вывод в строку посещаемых vyhodb записей.

По умолчанию, функции выводят все поля vyhodb записей. Но существует возможность настройки фильтра, который определяет имена полей, которые будут выведены при формировании результирующей строки. Фильтр задаётся в виде массива String[].

4.1. startPrint()

```
public static F startPrint(F... next)

public static F startPrint(String[] fieldsFilter, F... next)
```

Функция возвращает строку, представляющую граф vyhodb записей. Граф vyhodb записей формируется навигационными функциями.

Пример:

```
package com.vyhodb.freference.print;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PrintFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class PrintSimple extends Example {

    public static void main(String[] args) throws IOException {
        new PrintSimple().run();
    }

    @Override
    protected F getF() {
        return
        print(
            startPrint(
                children("order2root",
                    children("item2order",
                        parent("item2product")
                    )
                )
            )
        );
    }
}
```

Результат

```
    {Current Time="Wed Sep 09 02:52:18 BRT 2015"} id=0
```

```
        "order2root" ←
            {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096
                "item2order" ←
                    {Cost=62.25, Count=5} id=2129
                        "item2product" →
                            {Name="Product 1", Price=12.45} id=2063
                    {Cost=14569.90, Count=10} id=2140
                        "item2product" →
                            {Name="Product 2", Price=1456.99} id=2074
                    {Cost=884.85, Count=15} id=2151
                        "item2product" →
                            {Name="Product 3", Price=58.99} id=2085
            {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162
                "item2order" ←
                    {Cost=5899.00, Count=100} id=2184
                        "item2product" →
                            {Name="Product 3", Price=58.99} id=2085
            {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195
                "item2order" ←
                    {Cost=373.50, Count=30} id=2217
                        "item2product" →
                            {Name="Product 1", Price=12.45} id=2063
                    {Cost=43709.70, Count=30} id=2228
                        "item2product" →
                            {Name="Product 2", Price=1456.99} id=2074
                    {Cost=1769.70, Count=30} id=2239
                        "item2product" →
                            {Name="Product 3", Price=58.99} id=2085
```

Формат вывода

Каждая выводимая строка является либо текстовым представлением vyhodb записи (используется **com.vyhodb.space.Record#toString()**) либо описанием линка.

Описание линка состоит из имени линка и его типа. Тип определяется стрелкой: левое направление – дочерние линки, правое направление – родительские линки:

```
"item2order" ←
"item2product" →
```

Пример печати с фильтрацией полей:

```
package com.vyhodb.freference.print;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PrintFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class PrintSimpleFiltered extends Example {

    public static void main(String[] args) throws IOException {
        new PrintSimpleFiltered().run();
    }

    @Override
    protected F getF() {
        return
        print(
            startPrint(new String[]{"Customer", "Count"},
                children("order2root",
```

```
                    children("item2order",
                        parent("item2product")
                    )
                )
            )
        );
    }
}
```

Результат:

```
    {} id=0
        "order2root" ←
            {Customer="Customer 1"} id=2096
                "item2order" ←
                    {Count=5} id=2129
                        "item2product" →
                            {} id=2063
                    {Count=10} id=2140
                        "item2product" →
                            {} id=2074
                    {Count=15} id=2151
                        "item2product" →
                            {} id=2085
            {Customer="Customer 2"} id=2162
                "item2order" ←
                    {Count=100} id=2184
                        "item2product" →
                            {} id=2085
            {Customer="Customer 3"} id=2195
                "item2order" ←
                    {Count=30} id=2217
                        "item2product" →
                            {} id=2063
                    {Count=30} id=2228
                        "item2product" →
                            {} id=2074
                    {Count=30} id=2239
                        "item2product" →
                            {} id=2085
```

4.2. startPrintJson()

```
public static F startPrintJson(F... next)

public static F startPrintJson(boolean formatted, F... next)

public static F startPrintJson(String[] fieldsFilter, F... next)

public static F startPrintJson(String[] fieldsFilter, boolean formatted, F... next)
```

Функция возвращает строку в формате JSON, представляющую граф vyhodb записей. Граф vyhodb записей формируется навигационными функциями.

Параметр **formatted** определяет, будет ли JSON строка отформатирована или нет (без пробелов и символов переводов строки). По умолчанию, результирующая строка отформатирована.

Пример:

```
package com.vyhodb.freference.print;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.PrintFactory.*;

import java.io.IOException;
```

```
import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class PrintJson extends Example {

    public static void main(String[] args) throws IOException {
        new PrintJson().run();
    }

    @Override
    protected F getF() {
        return
        print(
            startPrintJson(
                children("order2root",
                    children("item2order",
                        parent("item2product")
                    )
                )
            )
        );
    }
}
```

Результат:

```
{
    "id":0,
    "Current Time":"Wed Sep 09 02:52:18 BRT 2015",
    "order2root":[
        {
            "id":2096,
            "Customer":"Customer 1",
            "Date":"Mon May 18 00:00:00 BRT 2015",
            "item2order":[
                {
                    "id":2129,
                    "Cost":62.25,
                    "Count":5,
                    "item2product":
                        {
                            "id":2063,
                            "Name":"Product 1",
                            "Price":12.45
                        }
                },
                {
                    "id":2140,
                    "Cost":14569.90,
                    "Count":10,
                    "item2product":
                        {
                            "id":2074,
                            "Name":"Product 2",
                            "Price":1456.99
                        }
                },
                {
                    "id":2151,
                    "Cost":884.85,
                    "Count":15,
                    "item2product":
                        {
                            "id":2085,
                            "Name":"Product 3",
                            "Price":58.99
                        }
                }
```

```
            ]
        },
        {
            "id":2162,
            "Customer":"Customer 2",
            "Date":"Tue May 19 00:00:00 BRT 2015",
            "item2order":[
                {
                    "id":2184,
                    "Cost":5899.00,
                    "Count":100,
                    "item2product":
                        {
                            "id":2085,
                            "Name":"Product 3",
                            "Price":58.99
                        }
                }
            ]
        },
        {
            "id":2195,
            "Customer":"Customer 3",
            "Date":"Wed May 20 00:00:00 BRT 2015",
            "item2order":[
                {
                    "id":2217,
                    "Cost":373.50,
                    "Count":30,
                    "item2product":
                        {
                            "id":2063,
                            "Name":"Product 1",
                            "Price":12.45
                        }
                },
                {
                    "id":2228,
                    "Cost":43709.70,
                    "Count":30,
                    "item2product":
                        {
                            "id":2074,
                            "Name":"Product 2",
                            "Price":1456.99
                        }
                },
                {
                    "id":2239,
                    "Cost":1769.70,
                    "Count":30,
                    "item2product":
                        {
                            "id":2085,
                            "Name":"Product 3",
                            "Price":58.99
                        }
                }
            ]
        }
    ]
}
```


Пример с фильтрацией:

```
package com.vyhodb.freference.print;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
```

```
import static com.vyhodb.f.PrintFactory.*;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

public class PrintJsonFiltered extends Example {

    public static void main(String[] args) throws IOException {
        new PrintJsonFiltered().run();
    }

    @Override
    protected F getF() {
        return
        print(
            startPrintJson(new String[]{"Customer", "Count"},
                children("order2root",
                    children("item2order",
                        parent("item2product")
                    )
                )
            )
        );
    }
}
```

Результат:

```
{
    "id":0,
    "order2root":[
        {
            "id":2096,
            "Customer":"Customer 1",
            "item2order":[
                {
                    "id":2129,
                    "Count":5,
                    "item2product":
                        {
                            "id":2063
                        }
                },
                {
                    "id":2140,
                    "Count":10,
                    "item2product":
                        {
                            "id":2074
                        }
                },
                {
                    "id":2151,
                    "Count":15,
                    "item2product":
                        {
                            "id":2085
                        }
                }
            ]
        },
        {
            "id":2162,
            "Customer":"Customer 2",
            "item2order":[
                {
                    "id":2184,
```

```
                    "Count":100,
                    "item2product":
                        {
                            "id":2085
                        }
                }
            ]
        },
        {
            "id":2195,
            "Customer":"Customer 3",
            "item2order":[
                {
                    "id":2217,
                    "Count":30,
                    "item2product":
                        {
                            "id":2063
                        }
                },
                {
                    "id":2228,
                    "Count":30,
                    "item2product":
                        {
                            "id":2074
                        }
                },
                {
                    "id":2239,
                    "Count":30,
                    "item2product":
                        {
                            "id":2085
                        }
                }
            ]
        }
    ]
}
```

5. RecordFactory

Фабрика предназначена для конструирования функций, работающих с записями vyhodb. Каждая из функций преобразует текущий объект в запись и оперирует над ней.

5.1. getRecord()

```
public static F getRecord(long recordId)
```

Функция ищет запись с заданным идентификатором. Текущий объект преобразовывается в запись и с него получается объект Space (**Record#getSpace()**) который используется для поиска записи.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class GetRecord extends Example {

    public static void main(String[] args) throws IOException {
        new GetRecord().run();
    }

    @Override
    protected F getF() {
        return
        print(getRecord(0L));
    }
}
```

Результат:

```
{} id=0
```

5.2. getField(), setField()

```
public static F getField(String fieldName)
```

Функция преобразует текущий объект в запись и возвращает значение поля fieldName.

```
public static F setField(String fieldName, F valueF)
```

Функция преобразует текущий объект в запись и изменяет значение поля fieldName на значение, возвращаемое функцией valueF. Функция возвращает результат исчисления valueF.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;
```

```
public class GetSetField extends Example {

    public static void main(String[] args) throws IOException {
        new GetSetField().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            print( getField("Field")),
            setField("Field", c(42)),
            print( getField("Field"))
        );
    }
}
```

Результат:

```
null
42
```

5.3. getParent()

```
public static F getParent(String linkName)
```

Функция преобразует текущий объект в запись и возвращает родительскую запись для линка с именем linkName.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

public class GetParent extends Example {

    public static void main(String[] args) throws IOException {
        new GetParent().run();
    }

    @Override
    protected F getF() {
        return
        children("order2root",
            children("item2order",
                print(
                    getParent("item2product")
                )
            )
        );
    }
}
```

Результат:

```
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
```

```
{Name="Product 3", Price=58.99} id=2085
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
```

5.4. setParent()

```
public static F setParent(String linkName, F parent)
```

Функция преобразует текущий объект в запись и устанавливает родительскую запись для линка с именем linkName. Родительская запись возвращается функцией parent.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

public class SetParent extends Example {

    public static void main(String[] args) throws IOException {
        new SetParent().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                children("item2order",
                    setParent("item2root", getRecord(0L))
                )
            ),
            children("item2root",
                printCurrent()
            )
        );
    }
}
```

Результат:

```
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
{Name="Product 3", Price=58.99} id=2085
{Name="Product 1", Price=12.45} id=2063
{Name="Product 2", Price=1456.99} id=2074
{Name="Product 3", Price=58.99} id=2085
```

5.5. getChildrenCount()

```
public static F getChildrenCount(String childrenLinkName)
```

Функция преобразует текущий объект в запись и возвращает количество её дочерних записей для линка childrenLinkName.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class GetChildrenCount extends Example {

    public static void main(String[] args) throws IOException {
        new GetChildrenCount().run();
    }

    @Override
    protected F getF() {
        return
        print(
            getChildrenCount("order2root")
        );
    }
}
```

Результат:

```
3
```

5.6. getChildFirst(), getChildLast()

```
public static F getChildFirst(String childLinkName)
```

Функция преобразует текущий объект в запись и возвращает первую дочернюю запись с именем линка childLinkName.

```
public static F getChildLast(String childLinkName)
```

Функция преобразует текущий объект в запись и возвращает последнюю дочернюю запись с именем линка childLinkName.

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class GetFirstLast extends Example {

    public static void main(String[] args) throws IOException {
        new GetFirstLast().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            print(getChildFirst("order2root")),
            print(getChildLast("order2root"))
```

```
        );
    }
}
```

Результат:

```
{Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096
{Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195
```

5.7. searchMinChild(), searchMaxChild()

```
public static F searchMinChild(String indexName)
```

Функция преобразует текущий объект в запись и возвращает индексированную дочернюю запись с минимальным значением индексированного поля(-ей).

Для поиска используется метод **com.vyhodb.space.Record#searchMinChild().**

```
public static F searchMaxChild(String indexName)
```

Функция преобразует текущий объект в запись и возвращает индексированную дочернюю запись с максимальным значением индексированного поля(-ей). Для поиска используется метод **com.vyhodb.space.Record#searchMaxChild().**

Пример:

```
package com.vyhodb.freference.record;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class SearchMinMax extends Example {

    public static void main(String[] args) throws IOException {
        new SearchMinMax().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            print(searchMinChild("order2root.Customer")),
            print(searchMaxChild("order2root.Customer"))
        );
    }
}
```

Результат:

```
{Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096
{Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195
```

6. AggregateFactory

Агрегаты, это функции, которые используются для подсчёта суммы, количества, минимальных/максимальных, средних значений.

Агрегаты реализованы как функции, которые принимаю на вход значение от другой функции, выполняют над ним какое-либо действие и сохраняют (агрегируют) результат в контексте.

Рассмотрим пример с подсчётом суммы всех продаж:

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.AggregatesFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class AggregatesExample1 extends Example {

    public static void main(String[] args) throws IOException {
        new AggregatesExample1().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                children("item2order",
                    sum(getField("Cost"))
                )
            ),
            print(getSum())
        );
    }
}
```

Результат:

```
67268.90
```

Функция **sum()** в данном случае получает значение, возвращаемое функцией **getField()** и суммирует его со значением, хранящимся в контексте.

По умолчанию, ключ контекста для функции **sum()** – “Sys$Sum”. Все значения ключей контекста, которые используются для хранения агрегатов, определены в классе **com.vyhodb.f.AggregatesFactory**.

Для получения значения суммы используется функция **getSum()**, которая в реальности является функцией **CommonFactory#get(“Sys$Sum”)**. Точно таким же образом (но со своими ключами) определены функции по получению значений других агрегатов (**getMin()**, **getMax()**, **getCount()**).

Также, в **com.vyhodb.f.AggregatesFactory** определены функции по сбросу значений агрегатов в контексте: **clearSum()**, **clearCount()**, **clearMin()**, **clearMax()**. Каждая clear функция в реальности является **CommonFactory#clear()** с соответствующим именем ключа контекста.

В данном разделе мы остановимся лишь на описании функций sum(), min(), max(), count(), опустив описание функций clear и get.

При вычислении агрегатов можно явно указать имена ключей контекста, в которых будут храниться значения агрегатов. Это особенно полезно при вычислении нескольких агрегатов одного типа. Например, следующий пример подсчитывает сумму продаж и сумму проданных единиц, используя лишь один проход по записям (что увеличивает производительность):

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.AggregatesFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class AggregatesExample2 extends Example {

    public static void main(String[] args) throws IOException {
        new AggregatesExample2().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                children("item2order",
                    sum("Cost", getField("Cost")),
                    sum("Count", getField("Count"))
                )
            ),
            print(getSum("Cost")),
            print(getSum("Count"))
        );
    }
}
```

Результат:

```
67268.90
220.0
```

6.1. sum()

```
public static F sum(F valueF)

public static F sum(String contextKey, F valueF)

public static F sum(SumType sumType, F valueF)

public static F sum(SumType sumType, String contextKey, F valueF)
```

Функция суммирует результат функции valueF и значение, хранящиеся в контексте (ключ контекста по умолчанию «Sys$Sum»).

Результат функции value преобразуется к типу java.lang.Number.

По умолчанию, для хранения суммы используется тип BigDecimal. Тип используемый для хранения суммы можно изменить, задав аргумент sumType (доступные значения: Long, Double, Decimal).

Пример:

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.f.aggregates.SumType;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.AggregatesFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class Sum extends Example {

    public static void main(String[] args) throws IOException {
        new Sum().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                children("item2order",
                    sum(SumType.DOUBLE, getField("Cost"))
                )
            ),
            print(getSum())
        );
    }
}
```

Результат:

```
67268.9
```

6.2. count()

```
public static F count()

public static F count(String contextKey)
```

Функция увеличивает значение агрегата в контексте на 1. Используется для подсчёта количества её вызовов.

Следующий пример подсчитывает количество дочерних записей **order2root**:

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.AggregatesFactory.*;

public class Count extends Example {
```

```
    public static void main(String[] args) throws IOException {
        new Count().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                    count()
            ),
            print(getCount())
        );
    }
}
```

Результат:

```
3
```

6.3. min()

```
public static F min(F valueF)

public static F min(String contextKey, F valueF)

public static F min(Comparator<?> comparator, F valueF)

public static F min(String contextKey, Comparator<?> comparator, F valueF)
```

Функция сравнивает результат функции valueF со значением в контексте и заносит наименьшее из них в контекст. По умолчанию, для хранения наименьшего значения, используется ключ «Sys$Min», который может быть переопределен (параметр contextKey).

В случае, когда задан comparator, он используется для сравнения значений. Если comparator не задан, оба значения приводятся к Comparable и сравниваются между собой.

Значения null не учитываются при сравнении.

Пример поиска item с минимальной стоимостью (поле “Cost”):

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;
import java.util.Comparator;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;
import com.vyhodb.space.Record;
import com.vyhodb.utils.FieldComparator;

import static com.vyhodb.f.AggregatesFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;

public class Min extends Example {

    public static void main(String[] args) throws IOException {
        new Min().run();
    }

    @Override
    protected F getF() {
        Comparator<Record> comparator = new FieldComparator("Cost");
```

```
        return
        composite(
            children("order2root",
                children("item2order",
                    min(comparator, current())
                )
            ),
            print(getMin())
        );
    }
}
```

Результат:

```
{Cost=62.25, Count=5} id=2129
```

6.4. max()

```
public static F max(F valueF)

public static F max(String contextKey, F valueF)

public static F max(Comparator<?> comparator, F valueF)

public static F max(String contextKey, Comparator<?> comparator, F valueF)
```

Функция сравнивает результат функции valueF со значением в контексте и заносит наибольшее из них в контекст. По умолчанию, для хранения наименьшего значения, используется ключ «Sys$Max», который может быть переопределен (contextKey).

В случае, когда задан comparator, он используется для сравнения значений. Если comparator не задан, оба значения приводятся к Comparable и сравниваются между собой.

Значения null не учитываются при сравнении.

Пример поиска максимальной даты заказа:

```
package com.vyhodb.freference.aggregates;

import java.io.IOException;

import com.vyhodb.f.F;
import com.vyhodb.freference.Example;

import static com.vyhodb.f.AggregatesFactory.*;
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.NavigationFactory.*;
import static com.vyhodb.f.RecordFactory.*;

public class Max extends Example {

    public static void main(String[] args) throws IOException {
        new Max().run();
    }

    @Override
    protected F getF() {
        return
        composite(
            children("order2root",
                max(getField("Date"))
            ),
            print(getMax())
        );
```

```
    }
}
```

Результат:

```
Wed May 20 00:00:00 BRT 2015
```

7. CollectionFactory

Функции, конструируемые этой фабрикой, предназначены для работы над объектом java.util.Collection, размещенным в контексте.

7.1. collectionAdd()

```
public static F collectionAdd(String contextKey, F elementF)

public static F collectionAdd(String contextKey, Object element)
```

Функция добавляет элемент в коллекцию, хранящуюся в контексте по ключу contextKey. Элемент задаётся в виде константы element или же возвращается как результат исполнения функции elementF.

Функция collectionAdd возвращает добавленный элемент.

Пример:

```
package com.vyhodb.freference.collection;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.CollectionFactory.*;

import java.util.ArrayList;

import com.vyhodb.f.F;

public class collectionAdd {

    public static void main(String[] args) {
        F f =
        composite(
            put("array", new ArrayList<>()),
            collectionAdd("array", c(42)),
            collectionAdd("array", 265),
            collectionAdd("array", "Hello"),
            get("array")
        );

        System.out.println(f.startEval(null));
    }
}
```

Результат:

```
[42, 265, Hello]
```

7.2. collectionContains()

```
public static Predicate collectionContains(String contextKey, F elementF)

public static Predicate collectionContains(String contextKey, Object element)
```

Предикат возвращает true, если коллекция, хранящаяся в контексте по ключу contextKey, содержит элемент, false – в случае, когда не содержит элемент. Элемент задаётся в виде константы element или же возвращается как результат исполнения функции elementF.

Пример:

```
package com.vyhodb.freference.collection;
```

```
import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.CollectionFactory.*;

import java.util.Arrays;
import java.util.List;

import com.vyhodb.f.F;

public class CollectionContains {

    public static void main(String[] args) {
        List<String> array = Arrays.asList("Hello", "world", "!");

        F f =
        composite(
            put("array", array),
            print( collectionContains("array", "!")),
            print( collectionContains("array", c("WWW")))
        );

        f.startEval(null);
    }
}
```

Результат:

```
true
false
```

7.3. collectionRemove()

```
public static F collectionRemove(String contextKey, F elementF)

public static F collectionRemove(String contextKey, Object element)
```

Функция удаляет элемент из коллекции, хранящейся в контексте по ключу contextKey. Удаляемый элемент задаётся в виде константы element или же возвращается как результат исполнения функции elementF.

Функция возвращает удаленный элемент.

Пример:

```
package com.vyhodb.freference.collection;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.CollectionFactory.*;

import java.util.ArrayList;
import java.util.Arrays;

import com.vyhodb.f.F;

public class CollectionRemove {

    public static void main(String[] args) {
        ArrayList<String> array = new ArrayList<>(Arrays.asList("Hello", "world", "!"));

        F f =
        composite(
            put("array", array),
            collectionRemove("array", "!"),
            collectionRemove("array", c("Hello"))
        );

        f.startEval(null);
```

```
        System.out.println(array);
    }
}
```

Результат:

```
[world]
```

7.4. collectionClear()

```
public static F collectionClear(String contextKey)
```

Функция удаляет все элементы в коллекции, хранящейся в контексте по ключу contextKey.

Функция возвращает коллекцию, хранящуюся в контексте по ключу contextKey.

Пример:

```
package com.vyhodb.freference.collection;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.CollectionFactory.*;

import java.util.ArrayList;
import java.util.Arrays;

import com.vyhodb.f.F;

public class CollectionClear {

    public static void main(String[] args) {
        ArrayList<String> array = new ArrayList<>(Arrays.asList("Hello", "world", "!"));

        F f =
        composite(
            put("array", array),
            collectionClear("array")
        );

        f.startEval(null);

        System.out.println(array);
    }
}
```

Результат:

```
[]
```

8. StringFactory

Фабрика содержит методы по конструированию функций работы со строками. Возвращаемые функции представляют собой «обёртки» вокруг методов класса java.lang.String.

8.1. strLowerCase(), strUpperCase()

```
public static F strLowerCase(F stringValueF)
```

Функция возвращает строку stringValueF приведенную к нижнему регистру.

```
public static F strUpperCase(F stringValueF)
```

Функция возвращает строку stringValueF приведенную к верхнему регистру.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class LowerUpperCase {

    public static void main(String[] args) {
        F f =
        composite(
            print( strUpperCase(c("Hello")) ),
            print( strLowerCase(c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
HELLO
hello
```

8.2. strIndex(), strLastIndex()

```
public static F strIndex(String pattern, F stringValueF)
```

Функция возвращает позицию первого вхождения подстроки pattern в строке stringValueF.

```
public static F strLastIndex(String pattern, F stringValueF)
```

Функция возвращает позицию последнего вхождения подстроки pattern в строке stringValueF.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class Index {
```

```
    public static void main(String[] args) {
        F f =
        composite(
            print( strIndex("l", c("Hello")) ),
            print( strLastIndex("l", c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
2
3
```

8.3. strSub()

```
public static F strSub(F beginF, F endF, F stringValueF)
```

Функция возвращает подстроку из строки stringValueF, начиная с позиции beginF и заканчивая (endF – 1).

Результаты функций beginF, endF преобразуются к типу java.lang.Number. Результат функции stringValueF преобразуется к типу java.lang.String.

```
public static F strSub(F beginF, F stringValueF)
```

Функция возвращает подстроку из строки stringValueF, начиная с позиции beginF и до конца строки stringValueF.

Результаты функции beginF, преобразуются к типу java.lang.Number.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StrSub {

    public static void main(String[] args) {
        F f =
        composite(
            print( strSub(c(2), c(4), c("Hello")) ),
            print( strSub(c(2), c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
ll
llo
```

8.4. strTrim()

```
public static F strTrim(F stringValueF)
```

Функция удаляет пробелы в начале и конце строки stringValueF.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StrTrim {

    public static void main(String[] args) {
        F f =
        composite(
            print( strTrim(c("   Hello   ")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
Hello
```

8.5. strLength()

```
public static F strLength(F stringValueF)
```

Функция возвращает длину строку stringValueF.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StrLength {

    public static void main(String[] args) {
        F f =
        composite(
            print( strLength(c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
5
```

8.6. strContains()

```
public static Predicate strContains(String pattern, F stringValueF)
```

Данный предикат возвращает true, если строка stringValueF, содержит подстроку pattern.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StrContains {

    public static void main(String[] args) {
        F f =
        composite(
            print( strContains("ll", c("Hello")) ),
            print( strContains("lll", c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
true
false
```

8.7. strStartsWith(), strEndsWith()

```
public static Predicate strStartsWith(String prefix, F stringValueF)
```

Данный предикат возвращает true если строка stringValueF начинается с префикса prefix; false – иначе.

```
public static Predicate strEndsWith(String suffix, F stringValueF)
```

Данный предикат возвращает true если строка stringValueF заканчивается суффиксом suffix.

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StartsEndsWith {

    public static void main(String[] args) {
        F f =
        composite(
            print( strStartsWith("He", c("Hello")) ),
            print( strEndsWith("llo", c("Hello")) )
        );

        f.startEval(null);
    }
```

```
}
```

Результат:

```
true
true
```

8.8. strMatches()

```
public static Predicate strMatches(String regExpression, F stringValueF)
```

Предикат возвращает true в случае если строка stringValueF удовлетворяет регулярному выражению pattern. False – иначе.

Предикат использует метод java.lang.String#matches().

Пример:

```
package com.vyhodb.freference.string;

import static com.vyhodb.f.CommonFactory.*;
import static com.vyhodb.f.StringFactory.*;

import com.vyhodb.f.F;

public class StrMatches {

    public static void main(String[] args) {
        F f =
        composite(
            print( strMatches("H.*o", c("Hello")) )
        );

        f.startEval(null);
    }
}
```

Результат:

```
true
```