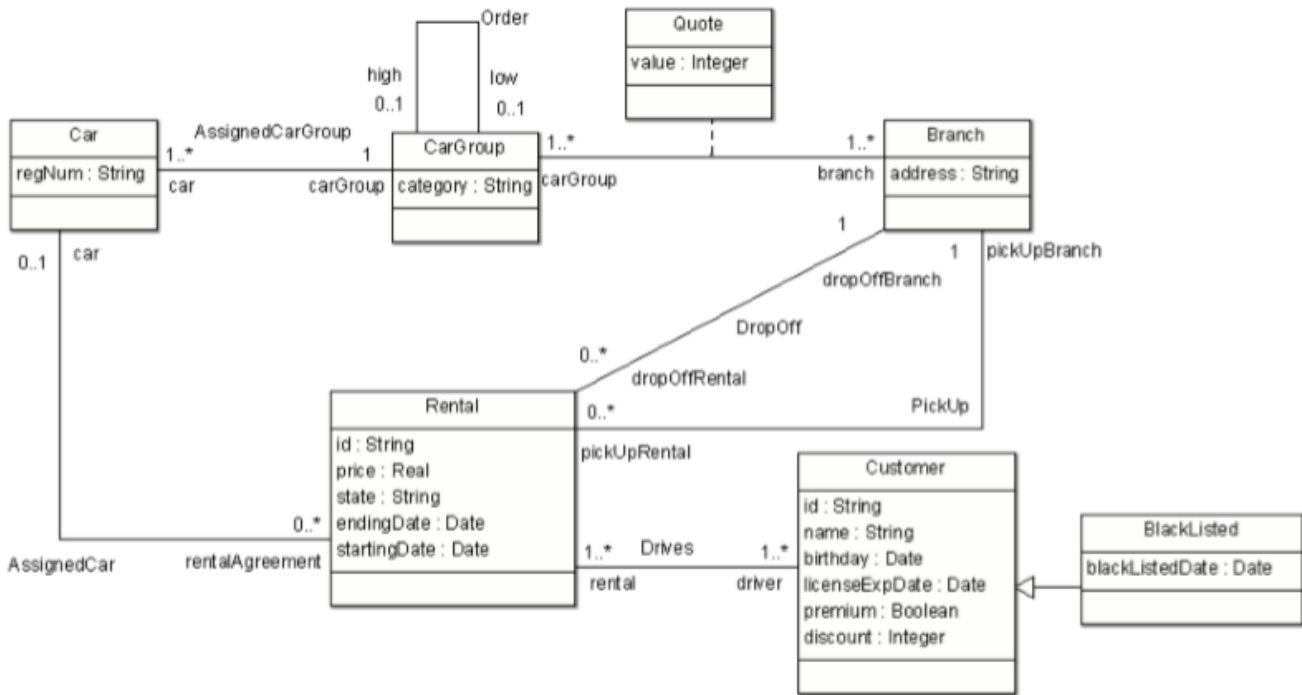


OCL简要教程

一. 基本语法



作为示例的UML图，之后的所有例子都在此基础上进行。

1. 不变量 (inv)

指定对象的整个生命周期都需要满足的约束

最基础的不变量声明，context声明一条新语句的开始，Quote为类名，inv为不变量关键词，冒号后为一个boolean值的表达式

```
context Quote inv: self.value > 0
```

在上述基础上，可以为不变量语句进行命名QuoteOverZero

```
context Quote inv QuoteOverZero: self.value > 0
```

可以依据自己的想法换行以使得语句更加清晰

```
context Quote inv QuoteOverZero:
self.value > 0
```

一个更为复杂的例子，定义了一个约束NoRentalsBlackListed，要求订单中所有订单不在用户进入黑名单后被设立。其中forall相当于数学公式中的 \forall 任意，返回一个boolean值，具体介绍在之后进行。

```
context BlackListed inv NoRentalsBlackListed: self.rental->forAll(r | r.startDate <
self.blackListedDate)
```

2. 初始化 (init)

指定对象的属性在创建对象时必须采用的初始值

```
context Customer::premium: Boolean init: false
```

premium为类Customer的一个所属变量，Boolean为要求的变量类型，init为初始化的关键词。

与inv相似，可以换行，这一点之后不再赘述。

```
context Customer::premium: Boolean
init: false
```

3. 派生 (derive)

与init类似，使用一样的句式结构，使用derive关键词，表示从其它模型元素的值或总体中推断出来的变量

此处使用一个较复杂的例子，展示了换行，if then else在内的各种语法。

该例子中，高级会员会获得最高的30折扣，其它人若至少租了5次高级汽车，也将获得15折扣，否则没有折扣。

select操作会从某集合中按要求产生新的集合，具体介绍在之后进行。

```
context Customer::discount: integer
derive:
  if not self.premium then
    if self.rental.car.carGroup->
      select(c|c.category='high')->size()>=5
    then 15
    else 0 endif
  else 30 endif
```

4. 查询 (Query)

与上述语句使用类似的句式结构，使用body关键词，表示查询系统数据

该例子中，如果执行该操作的汽车在租赁系统中最受欢迎，则以下查询操作将返回 true

```
context Car::mostPopular(): boolean
body: Car::allInstances()->forAll(c1|c1<>self implies c1.rentalAgreement->size()
<=self.rentalAgreement->size())
```

5. 前置 (Precondition) 与 后置 (Postcondition)

在声明式规范中，可以为操作提供合同，合同包括前置与后置两种。

前置条件定义了一组关于操作输入和系统状态的条件，这些条件在操作发出时必须保持而后置条件则定义了在执行结束时系统状态必须满足的一组条件

以下例子介绍了前置pre和后置post的使用方法，我们无需完全读懂该例子，只需关注其中的重点。

首先是主体，主题为操作函数newRental，建立一个新订单。

pre检查并确保用户在租赁期间拥有有效的许可证，该检查在执行开始前进行。

post检查并确保新建操作结束后，新建的对象为全新的。其中one操作返回一个boolean值，当满足一系列条件的元素有且仅有一个时，one操作返回true

```
context Rental::newRental(id:Integer, price:Real, startingDate:Date, endingDate:Date,
customer:Customer, carRegNum:String, pickupBranch: Branch, dropOffBranch: Branch)
pre: customer.licenseExpDate>endingDate
post: Rental.allInstances->one(r | r.oclIsNew() and r.oclIsTypeOf(Rental) and
r.endingDate=endingDate and r.startingDate=startingDate and r.driver=customer and
r.pickupBranch=pickupBranch and r.dropOffBranch=dropOffBranch and
r.car=Car.allInstances()->any(c | c.regNum=carRegNum))
```

二. OCL中的类型

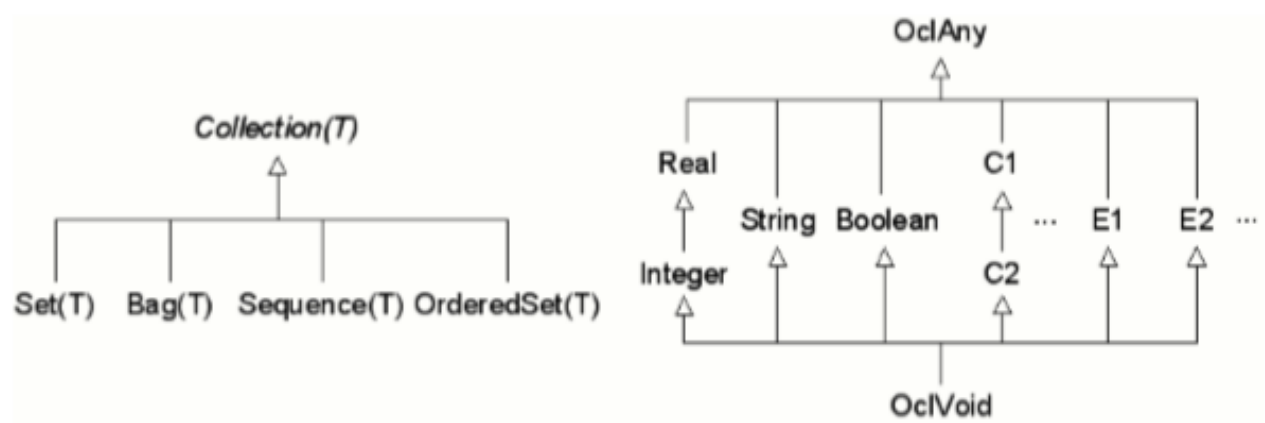


Fig. 4. OCL Type Inheritance Hierarchy

1. 四种集合类型

- Set: 无序, 不可重复
- Bag: 无序, 可重复
- Sequence: 有序, 可重复
- OrderedSet: 有序, 不可重复

2. 数据类型

- Real 浮点数, 小数
- Integer 整数
- String 字符串
- Boolean 布尔型

三. 常用各类函数与操作

- `=` 等于
- `<>` 不等于
- `if e1 then e2 else e3 endif` 条件语句

以下操作都返回Boolean参数

```
e1->forAll(r | e2)
对e1中所有的元素, 如果都满足e2, 则返回true

e1->one(r | e2)
对e1中所有的元素, 如果有且仅有一个元素满足e2, 则返回true

e1->exists(r | e2)
对e1中所有的元素, 如果存在一个元素满足e2, 则返回true
```

集合相关操作

将当前集合转化为某个特定的集合

```
asBag(), asSet(), asOrderedSet(), asSequence()
```

including操作, 即向某集合插入元素的操作

```
Sequence{7,8,7}=Sequence{7,8}->including(7)
```

excluding操作, 即从某集合中删除元素的操作, 删除所有指定元素

```
Set{7,8}->excluding(8)=Set{7}
Bag{7,8,7}->excluding(7)=Bag{8}
```

includes用于判断集合是否包含某个元素

```
Set{7,8}->includes(9) = true
```

excludes与includes相对

```
Bag{7,8}->excludes(9) = true
```

includesAll 和 excludesAll则是进行批量的判断

```
Sequence{7,9,8,7}->includesAll(Sequence{7,8,8}) = true
```

```
OrderedSet{7,9,8,7}->excludesAll(OrderedSet{3,2,4,2}) = true
```

isEmpty 和 notEmpty判断集合是否为空

```
Set{7}->excluding(7)->isEmpty() = true
```

```
Bag{7}->excluding(8)->notEmpty() = true
```

size将会返回集合的大小

```
Set{7,8,7,8,9}->size() = 3
```

```
Bag{7,8,7,8,9}->size() = 5
```

```
Sequence{7,8,7,9}->size() = 4
```

```
OrderedSet{7,8,7,9}->size() = 3
```

select, reject, collect操作都能从原集合中生成符合条件的新集合，其中select表示需满足列出条件，reject表示需不满足相关条件，collect表示对所有元素进行列出的操作

```
Set{7,8,9}->select(i|i.mod(2)=1) = Set{7,9}
```

```
Bag{7,8,7,9}->reject(i|i.mod(2)=1) = Bag{8}
```

```
Sequence{7,8,9}->collect(i|i*i) = Sequence{49,64,91}
```

```
Set{-1,0,+1}->collect(i|i*i) = Bag{0,1,1}
```

any操作将会把集合中所有满足条件的元素返回

```
Set{7,8,9}->any(true) = 7,8,9
```

```
Set{7,8,9}->any(i|i.mod(2)=0) = 8
```

union操作将合并两个集合

```
Set{7,8}->union(Set{9,8}) = Set{7,8,9}
```

```
Bag{7,8}->union(Bag{9,8}) = Bag{7,8,8,9}
```

```
Sequence{7,8}->union(Sequence{9,8}) = Sequence{7,8,9,8}
```

四. 部分实验要求与提示

- 每条语句都以context开始
- ocl语句主要用于表述那些在模型中很难用图表述的约束条件，以我们最开始的Uml图为例，以下问题就很难回答。有些问题可能按照常识来看无需解答，但是事实上确实没有在模型图中体现。我们的ocl语句可以参考这些问题的思路，对绘制的ifml模型进行更加准确的约束。

1. 黑名单上的人可以租车吗？
2. 如何计算租金？
3. 延长现有租赁协议的条件是什么？
4. 是否有最低驾驶资历要求？ 同一个司机可以有两个活跃的租车吗？
5. 我可以选择已经分配给其他租车公司的汽车吗？

- 建议与第三部分实验ifml模型绘制结合完成实验
- 将app中不同种类的界面作为不同的类，这些类有同一父类
- 将界面上的组件作为有同一父类的类，并依附于某界面
- 将界面上的动作作为有同一父类类，并依附于某个组件
- 一些思路，还有很多可写的，此处随意列出一些

进行点击文件，或长按选中文件操作时，需确保存在可选中的文件

新建文件时，文件名为空，为重复文件名，为特殊文件名是否有影响

进行文本框清除操作时(常见于搜索操作)，文本框中是否需要已有文本输入

新建文件后，文件的排列顺序是否有特殊规律（尝试借助Sequence或OrderedSet表述）

某个button上的文本是否是固定的，图标是否是固定的（给图标命名）

- 本教程未列出所有可用操作，可以自行阅读附带的英文文档进行学习。
- 对实验二的评价将会从ocl语句数，ocl语句语法语义正确性等方面进行，请大家在保证有尽量多的ocl语句的同时，确保语句的正确性，并尽量覆盖ifml模型中无法准确表述的部分。