UML for .NET Developers

Model Constraints & The Object Constraint Language

Jason Gorman

parlez|uml

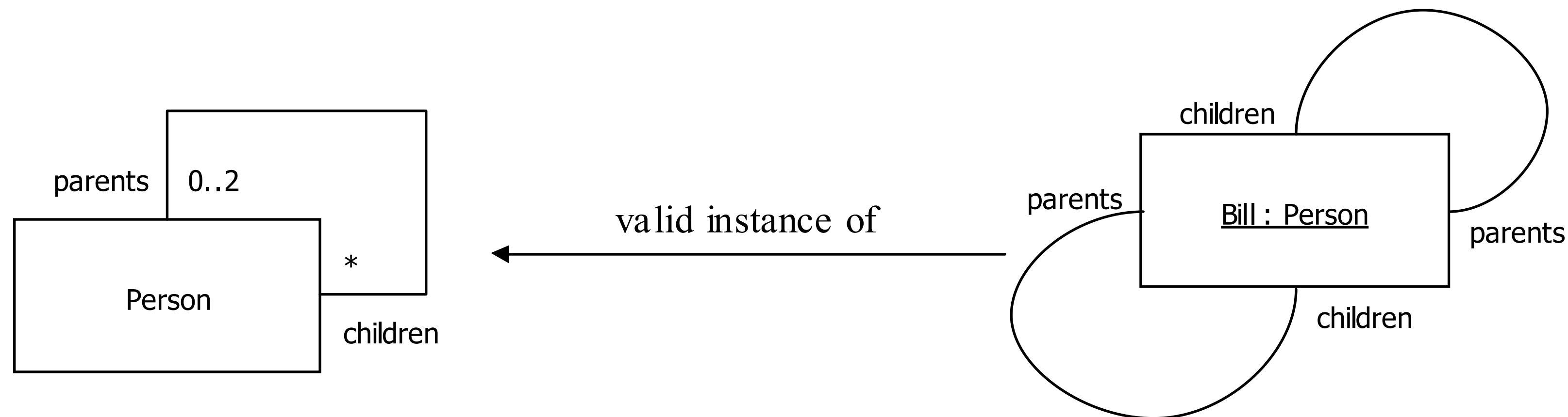
© Jason Gorman 2005. All rights reserved.

Accelerated learning for C# developers

Get the UML for .NET Premium Package at
http://www.parlezuml.com/tutorials/umlfordotnet.htm

© Jason Gorman 2005. All rights reserved.

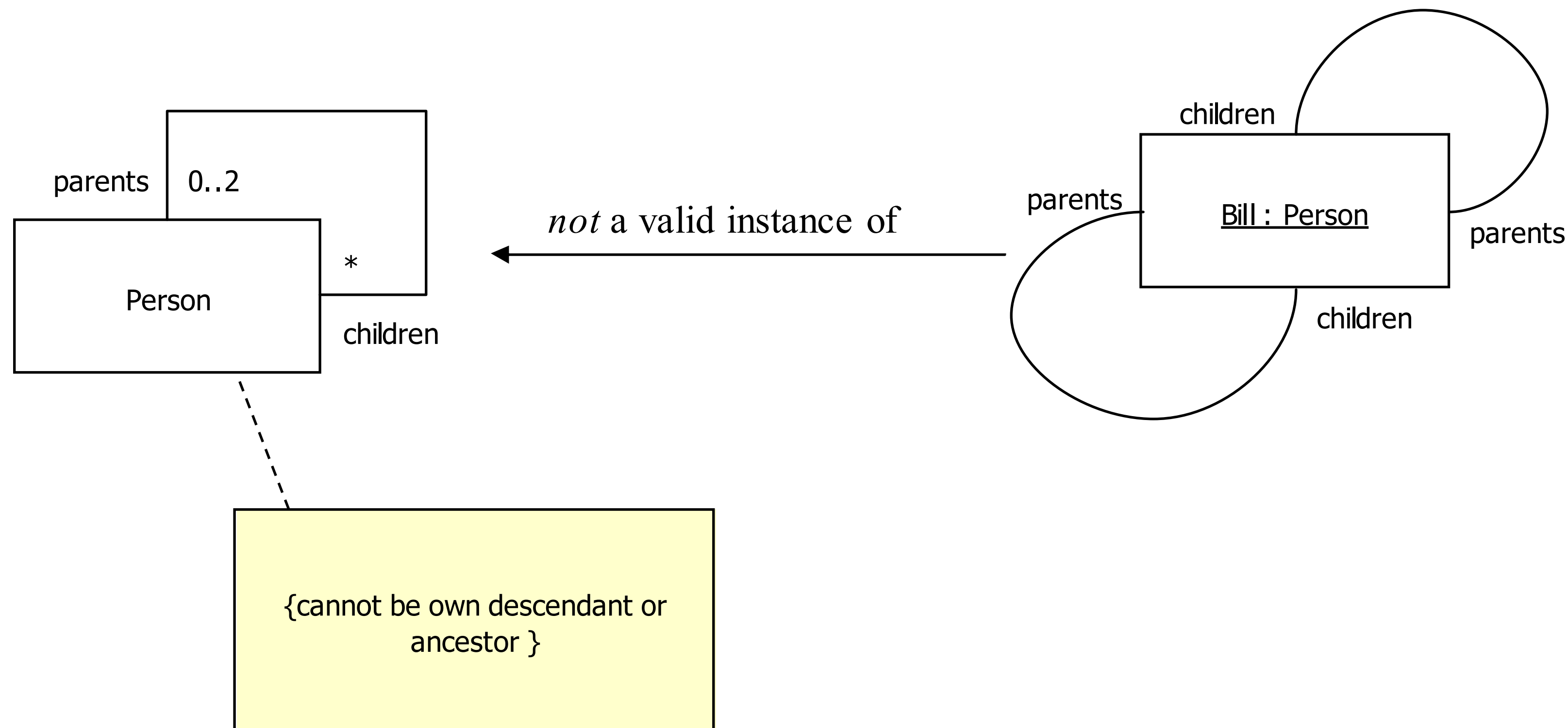
UML Diagrams Don’t Tell Us Everything



parlez|uml

© Jason Gorman 2005. All rights reserved.

Constraints Make Models More Precise



parlez|uml

© Jason Gorman 2005. All rights reserved.

What is the Object Constraint Language?

- A language for expressing necessary extra information about a model
- A precise and unambiguous language that can be read and understood by developers and customers
- A language that is purely declarative – ie, it has *no side-effects* (in other words it describes *what* rather than *how*)

parlez|uml

© Jason Gorman 2005. All rights reserved.

What is an OCL Constraint?

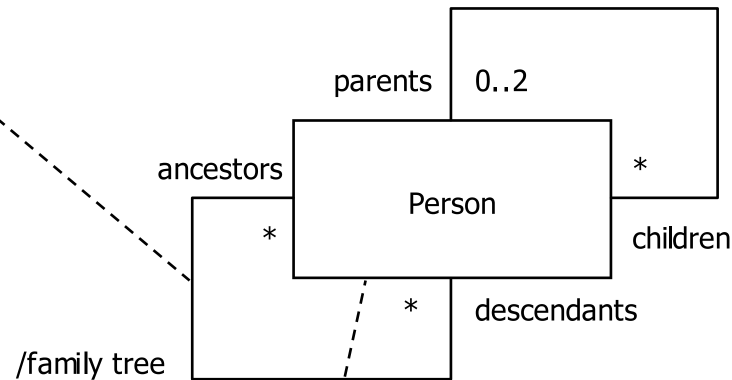
- An OCL constraint is an OCL expression that evaluates to true or false (a Boolean OCL expression, in other words)

parlez|uml

© Jason Gorman 2005. All rights reserved.

OCL Makes Constraints Unambiguous

{ancestors = parents->union(parents.ancestors->asSet())}
{descendants = children->union(children.descendants->asSet())



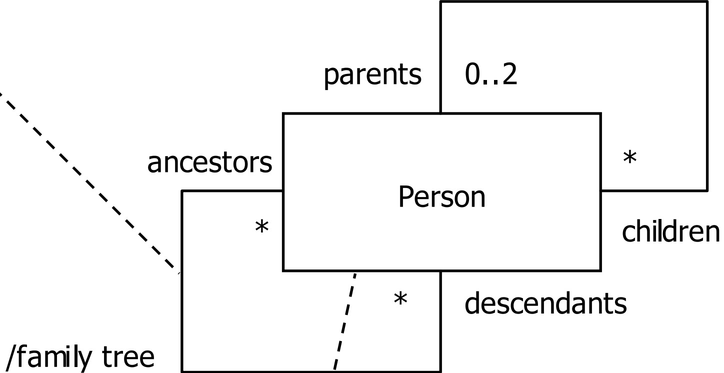
{ancestors->excludes(self) and descendants->excludes(self) }

parlez|uml

© Jason Gorman 2005. All rights reserved.

Introducing OCL – Constraints & Contexts

{ancestors = parents->union(parents.ancestors->asSet())}
{descendants = children->union(children.descendants->asSet())



Q: To what which type this constraint apply?
A: Person

```
context Person
inv: ancestors->excludes(self) and descendants->excludes(self)
```

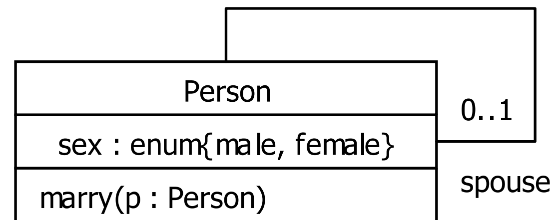
Q: When does this constraint apply?
A: inv = invariant = always

{ancestors->excludes(self) and descendants->excludes(self) }

parlez|uml

© Jason Gorman 2005. All rights reserved.

Operations, Pre & Post-conditions



optional constraint name

applies to the marry() operation of the type Person

comments start with --

```
context Person::marry(p : Person)
pre cannot_marry_self:  not (p = self)
pre not_same_sex: not (p.sex = self.sex)
-- neither person can be married already
pre  not_already_married: self.spouse->size() = 0 and p.spouse->size() = 0
post : self.spouse = p and p.spouse = self
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Design By Contract :Debug.Assert()

```
public enum Sex
{
        male,
        female
}

public class Person
{
        public Sex sex;
        public Person spouse;

        public void marry(Person p)
        {
                Debug.Assert(p != this);
                Debug.Assert(p.sex != this.sex);
                Debug.Assert(this.spouse = null && p.spouse = null);

                this.spouse = p;
                p.spouse = this;
        }
}
```

self

self.spouse->size = 0

```
context Person::marry(p : Person)
pre cannot_marry_self: not (p = self)
pre not_same_sex: not (p.sex = self.sex)
-- neither person can be married already
pre not_already_married: self.spouse->size() = 0 and p.spouse->size() = 0
post : self.spouse = p and p.spouse = self
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Accelerated learning for C# developers

Get the UML for .NET Premium Package at
http://www.parlezuml.com/tutorials/umlfordotnet.htm

parlez|uml

© Jason Gorman 2005. All rights reserved.

Defensive Programming : Throwing Exceptions

```
public class Person
{
        public Sex sex;
        public Person spouse;

        public void marry(Person p)
        {
                if(p == this)
                {
                        throw new ArgumentException("cannot marry self");
                }
                if(p.sex == this.sex)
                {
                        throw new ArgumentException("spouse is same sex");
                }
                if((p.spouse != null || this.spouse != null)
                {
                        throw new ArgumentException("already married");
                }

                this.spouse = p;
                p.spouse = this;
        }
}
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Referring to previous values and operation return values

Account

balance : Real = 0

deposit(amount : Real)
withdraw(amount : Real)
getBalance() : Real

balance before execution of operation

```
context Account::withdraw (amount : Real)
pre: amount <= balance
post: balance = balance@pre - amount

context Account::getBalance() : Real
post : result = balance
```

return value of operation

parlez|uml

© Jason Gorman 2005. All rights reserved.

@pre and result in C#

```
context Account::withdraw(amount : Real)
pre: amount <= balance
post: balance = balance@pre - amount

context Account::getBalance() : Real
post : result = balance
```

```
public class Account
{
        private double balance = 0;

        public void withdraw(double amount)
        {
                Debug.Assert(amount <= balance);

                balance = balance - amount;
        }

        public void deposit(double amount)
        {
                balance = balance + amount;
        }

        public double getBalance()
        {
                return balance;
        }
}
```

result = balance

```
[Test]
public void withdrawTest()
{
        Account account = new Account();

        account.deposit(500);

        double balanceAtPre = account.getBalance();

        double amount = 250;

        account.withdraw(amount);

        Assertion.Assert(account.getBalance() == balanceAtPre - amount);
}
```

balance = balance@pre - amount

parlez|uml

© Jason Gorman 2005. All rights reserved.

OCL Basic Value Types

Account
balance : Real = 0 name : String id : Integer isActive : Boolean
deposit(amount : Real) withdraw(amount : Real)

- **Integer** : A whole number of any size
- **Real** : A decimal number of any size
- **String** : A string of characters
- **Boolean** : True/False

id : Integer	balance : Real = 0	name : String	isActive : Boolean
int id;	double balance = 0;	string name;	bool isActive;
long id;	float balance = 0;	char name =;	
byte id;	decimal balance = 0;	char[] name;	
sbyte id;			
short id;			
ushort id;			
uint id;			
ulong id;			

parlez|uml

© Jason Gorman 2005. All rights reserved.

Operations on Real and Integer Types

Operation	Notation	Result type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiply	a * b	Integer or Real
divide	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs	Integer or Real
maximum	a.max(b)	Integer or Real
minimum	a.min(b)	Integer or Real
round	a.round	Integer
floor	a.floor	Integer

Eg, 6.7.floor() = 6

parlez|uml

© Jason Gorman 2005. All rights reserved.

Operations on String Type

Operation	Expression	Result type
concatenation	s.concat(string)	String
size	s.size	Integer
to lower case	s.toLower	String
to upper case	s.toUpper	String
substring	s.substring(int, int)	String
equals	s1 = s2	Boolean
not equals	s1 <> s2	Boolean

Eg, ‘jason’.concat(‘ gorman’) = ‘jason gorman’

Eg, ‘jason’.substring(1, 2) = ‘ja’

parlez|uml

© Jason Gorman 2005. All rights reserved.

Operations on Boolean Type

Operation	Notation	Result type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implication	a implies b	Boolean
if then else	if a then b1 else b2 endif	type of b

Eg, true or false = true

Eg, true and false = false

parlez|uml

© Jason Gorman 2005. All rights reserved.

Navigating in OCL Expressions

In OCL:

account.holder

Evaluates to a customer object who is in the role holder for that association

And:

customer.accounts

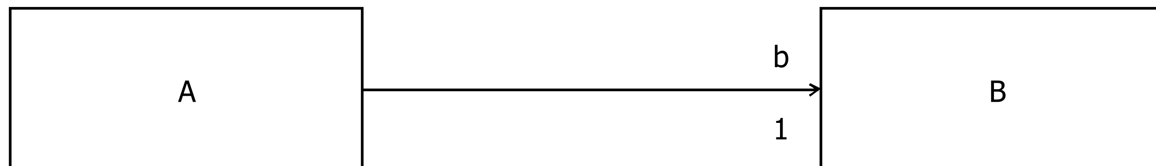
Evaluates to a *collection* of Account objects in the role accounts for that association

```
Account account = new Account();
Customer customer = new Customer();

customer.accounts = new Account[] {account};
account.holder = customer;
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Navigability in OCL Expressions



a.b is allowed

b.a is *not* allowed – it is not navigable

```
public class A
{
            public B b;
}

public class B
{
}
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Accelerated learning for C# developers

Get the UML for .NET Premium Package at
http://www.parlezuml.com/tutorials/umlfordotnet.htm

parlez|uml

© Jason Gorman 2005. All rights reserved.

Calling class features

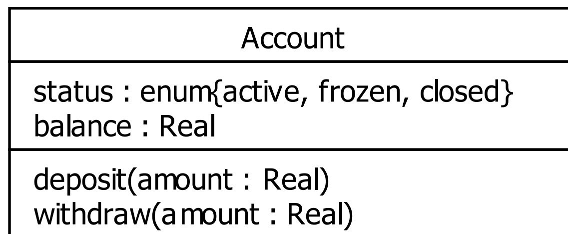
Account
id : Integer status : enum{active, frozen, closed} balance : Real nextId : Integer
deposit(amount : Real) withdraw(amount : Real) fetch(id : Integer) : Account

```
context Account::createNew() : Account
post: result.oclIsNew() and
      result.id = Account.nextId@pre and
      Account.nextId = result.id + 1
```

parlez|uml

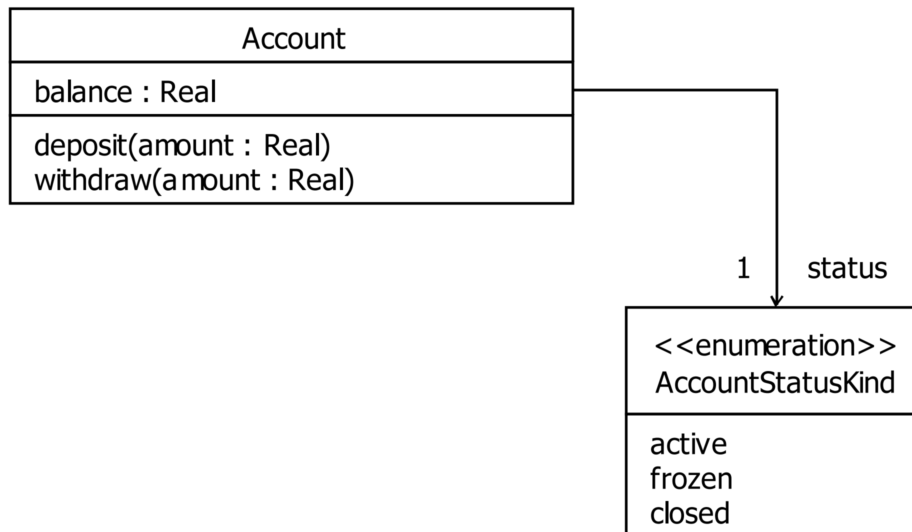
© Jason Gorman 2005. All rights reserved.

Enumerations in OCL




```
context Account::withdraw(amount : Real)
pre: amount <= balance
pre: status = #active
post: balance = balance@pre - amount
```

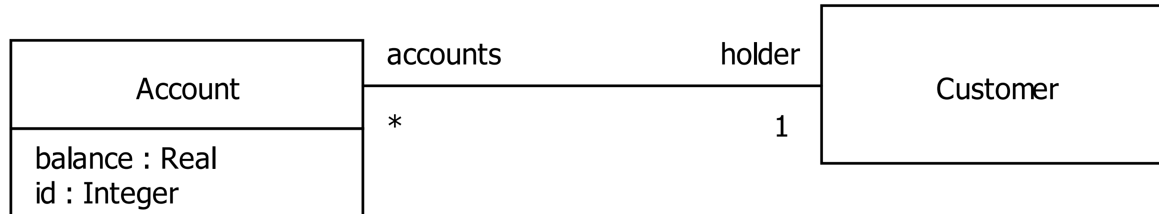
or



parlez|uml

© Jason Gorman 2005. All rights reserved.

Collections in OCL



customer.accounts.balance = 0 is *not* allowed

customer.accounts->select(id = 2324).balance = 0 is allowed

parlez|uml

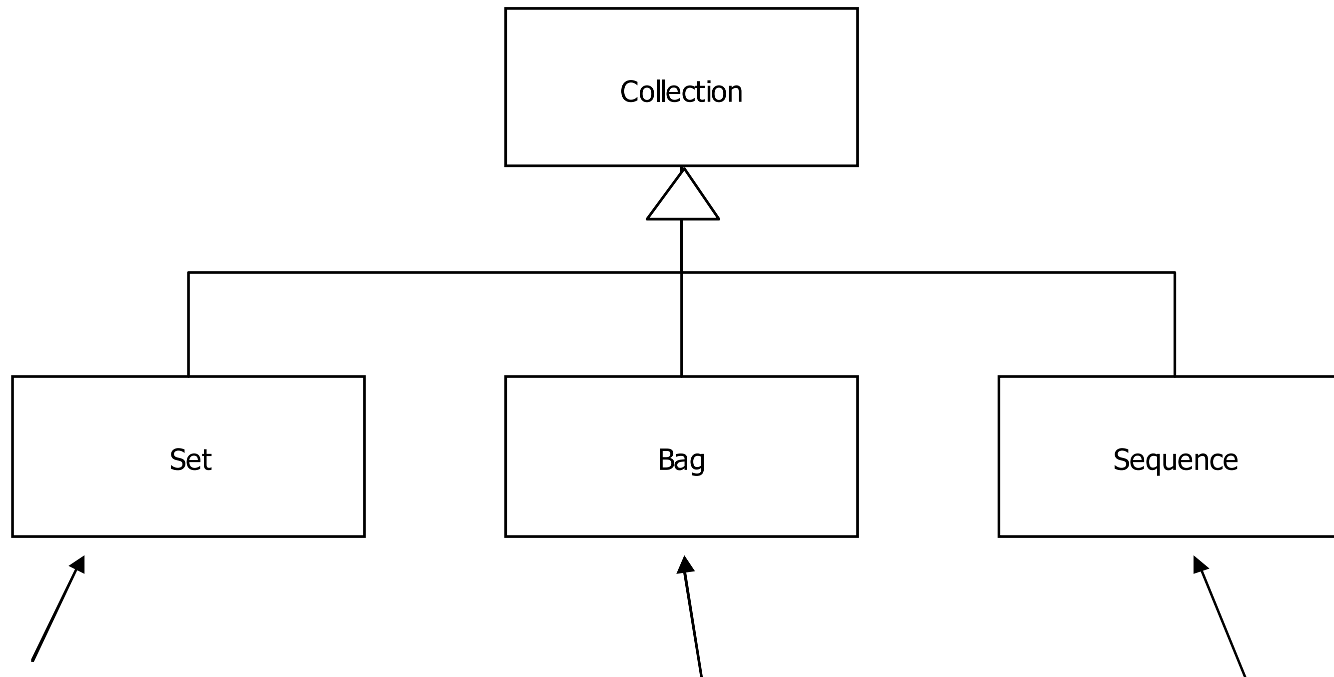
© Jason Gorman 2005. All rights reserved.

Collections in C#

```
public class Account
{
        public double balance;
        public int id;
}

public class Customer
{
        Account[] accounts;

        public Account SelectAccount(int id)
        {
                Account selected = null;

                foreach(Account account in accounts)
                {
                        if(account.id = id)
                        {
                                selected = account;
                                break;
                        }
                }

                return selected;
        }
}
```

© Jason Gorman 2005. All rights reserved.

The OCL Collection Hierarchy



Elements can be included only once, and in no specific order

Elements can be included more than once, in no specific order

Elements can be included more than once, but in a specific order

parlez|uml

© Jason Gorman 2005. All rights reserved.

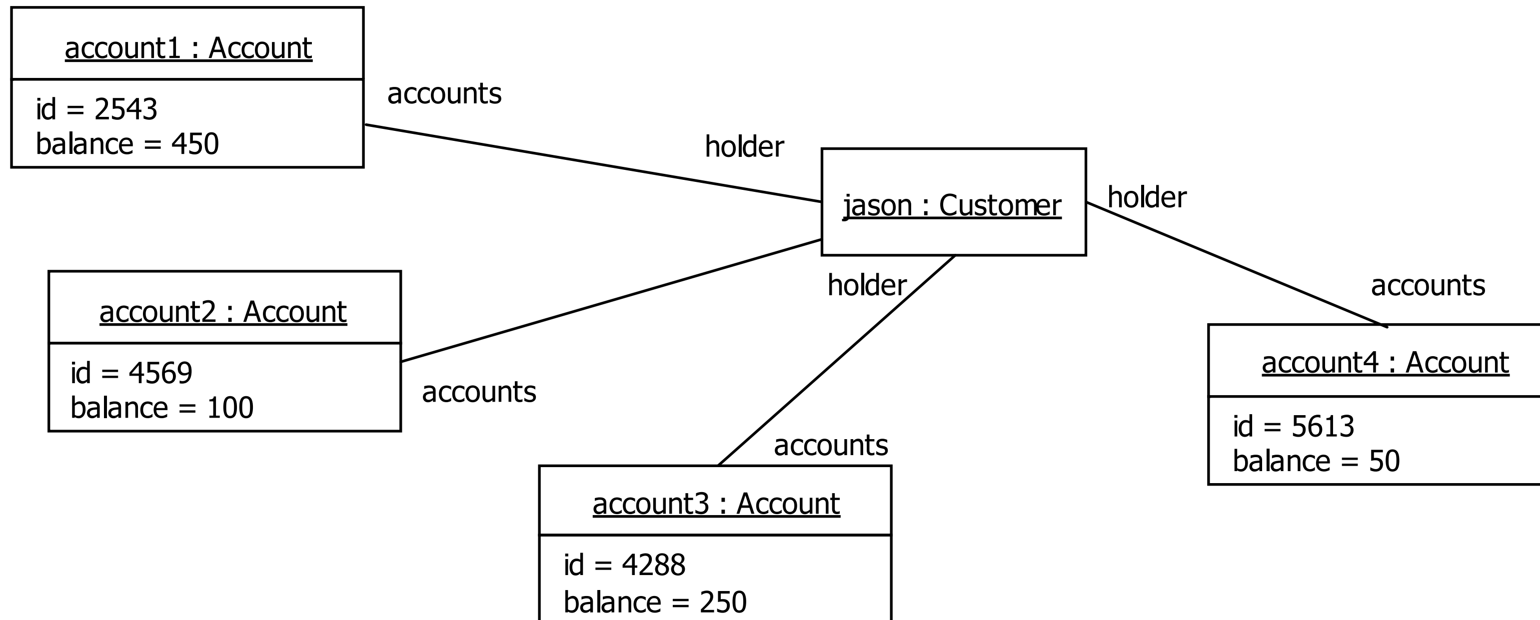
Operations on All Collections

Operation	Description
size	The number of elements in the collection
count(object)	The number of occurences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter collection are present in the current collection.
isEmpty	True if the collection contains no elements.
notEmpty	True if the collection contains one or more elements.
iterate(expression)	Expression is evaluated for every element in the collection.
sum(collection)	The addition of all elements in the collection.
exists(expression)	True if expression is true for at least one element in the collection.
forAll(expression)	True if expression is true for all elements.
select(expression)	Returns the subset of elements that satisfy the expression
reject(expression)	Returns the subset of elements that do not satisfy the expression
collect(expression)	Collects all of the elements given by expression into a new collection
one(expression)	Returns true if exactly one element satisfies the expression
sortedBy(expression)	Returns a Sequence of all the elements in the collection in the order specified (expression must contain the < operator

parlez|uml

© Jason Gorman 2005. All rights reserved.

Examples of Collection Operations



jason.accounts->forAll(a : Account | a.balance > 0) = true

jason.accounts->select(balance > 100) = {account1, account3}

jason.accounts->includes(account4) = true

jason.accounts->exists(a : account | a.id = 333) = false

jason.accounts->includesAll({account1, account2}) = true

jason.accounts.balance->sum() = 850

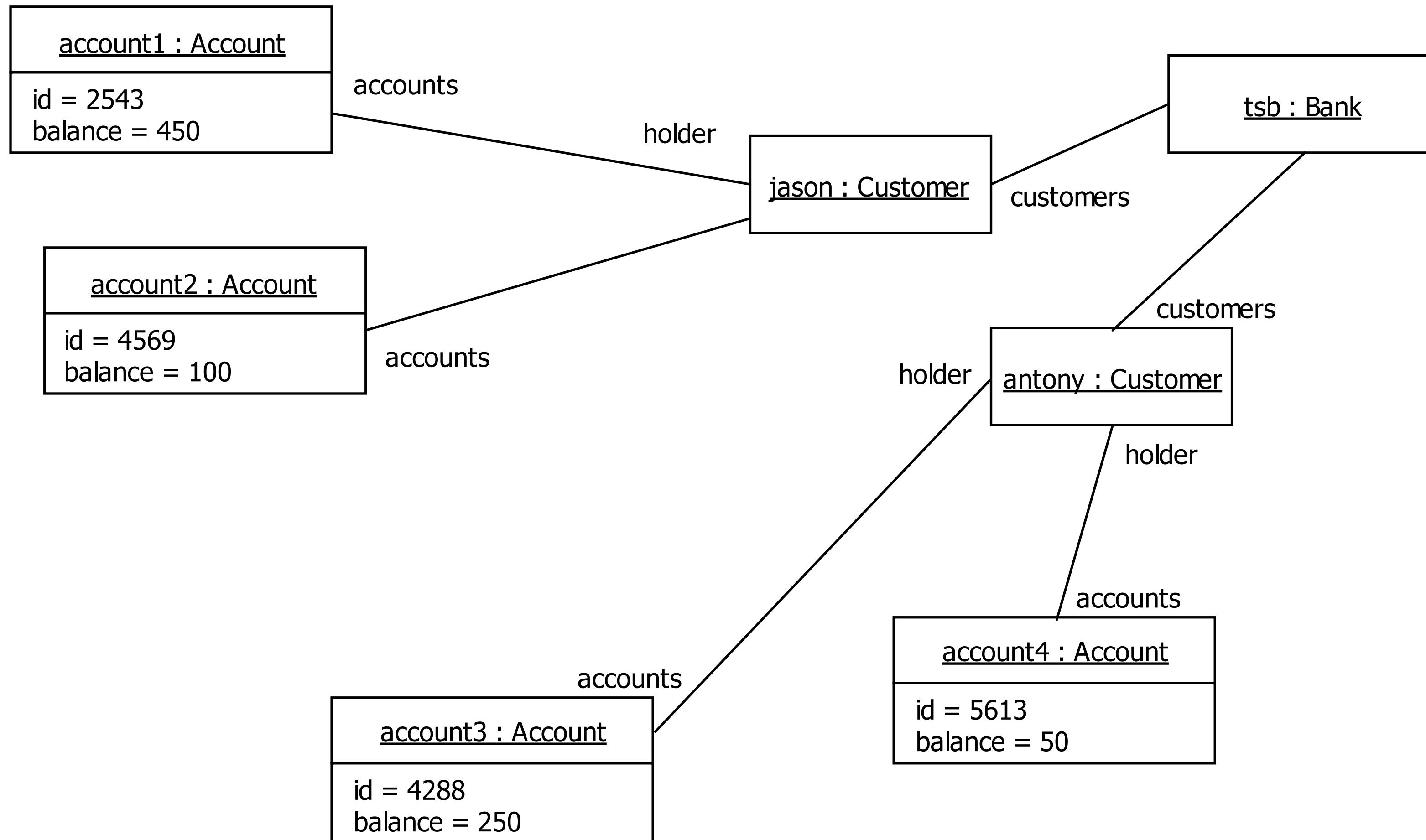
Jason.accounts->collect(balance) = {450, 100, 250, 50}

```
bool forAll = true;
foreach(Account a in accounts)
{
        if(!(a.balance > 0))
        {
                forAll = forAll && (a.balance > 0);
        }
}
```

parlez|uml

© Jason Gorman 2005. All rights reserved.

Navigating Across & Flattening Collections

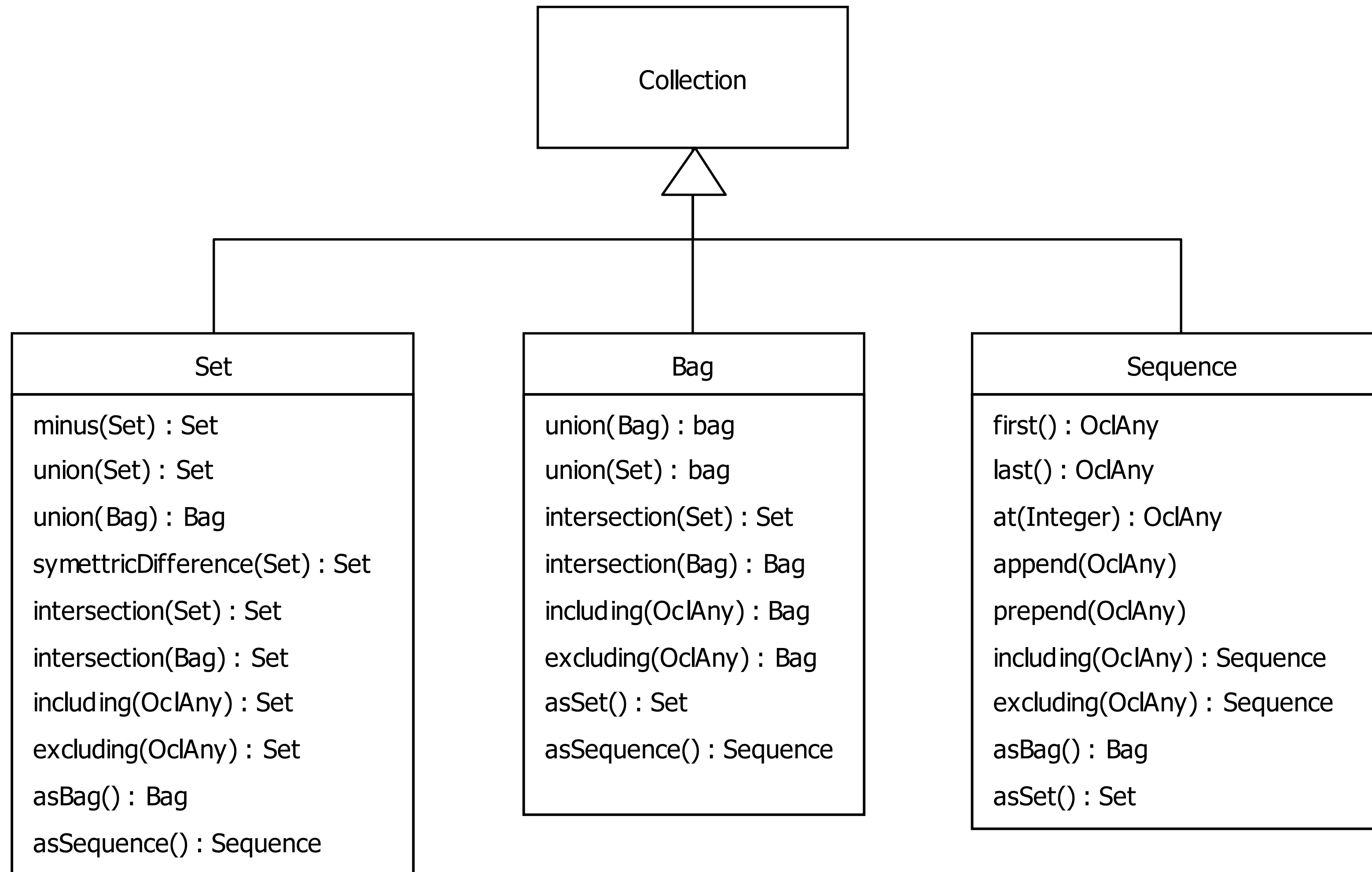


tsb.customers.accounts = {account1, account2, account3, account}
tsb.customers.accounts.balance = {450, 100, 250, 50}

parlez|uml

© Jason Gorman 2005. All rights reserved.

Specialized Collection Operations



Eg, Set{4, 2, 3, 1}.minus(Set{2, 3}) = Set{4, 1}
Eg, Bag{1, 2, 3, 5}.including(6) = Bag{1, 2, 3, 5, 6}
Eg, Sequence{1, 2, 3, 4}.append(5) = Sequence{1, 2, 3, 4, 5}

parlez|uml

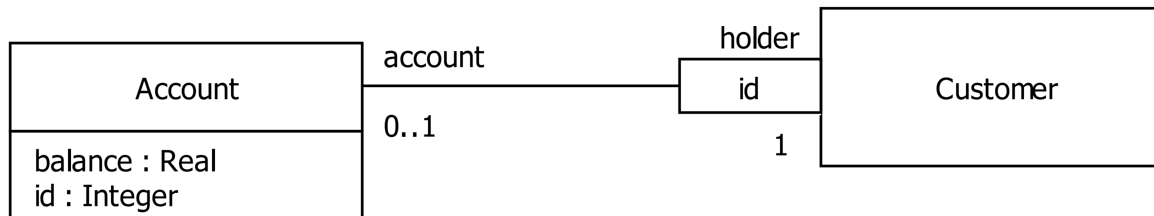
© Jason Gorman 2005. All rights reserved.

Accelerated learning for C# developers

Get the UML for .NET Premium Package at
http://www.parlezuml.com/tutorials/umlfordotnet.htm

© Jason Gorman 2005. All rights reserved.

Navigating across Qualified Associations



customer.account[3435]

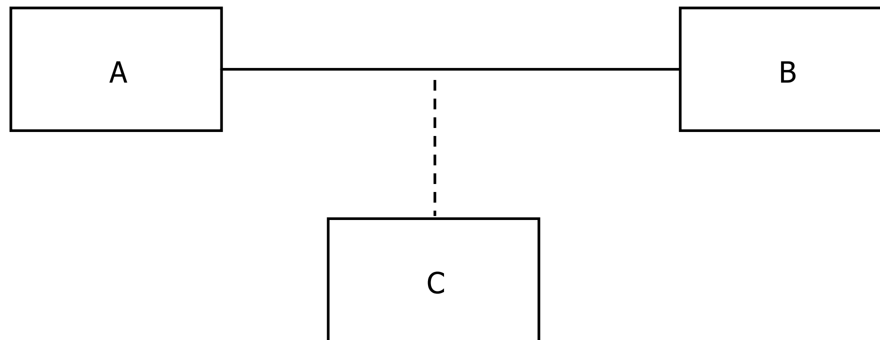
Or

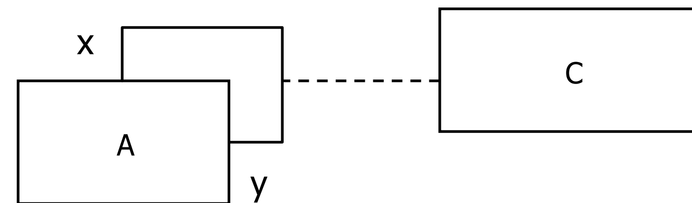
customer.account[id = 3435]

parlez|uml

© Jason Gorman 2005. All rights reserved.

Navigating to Association Classes





context A inv: self.c

context B inv: self.c

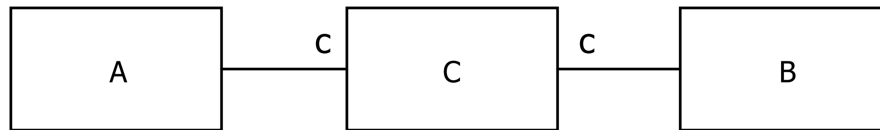
context A inv: self.c[x]

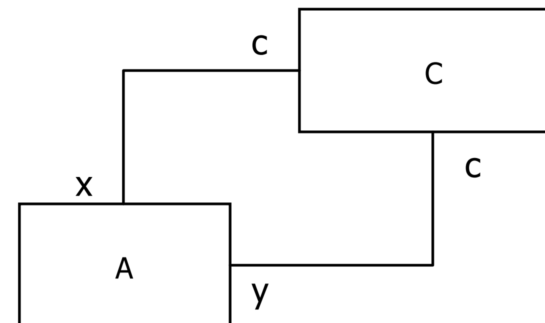
context A inv: self.c[y]

parlez|uml

© Jason Gorman 2005. All rights reserved.

Equivalents to Association Classes

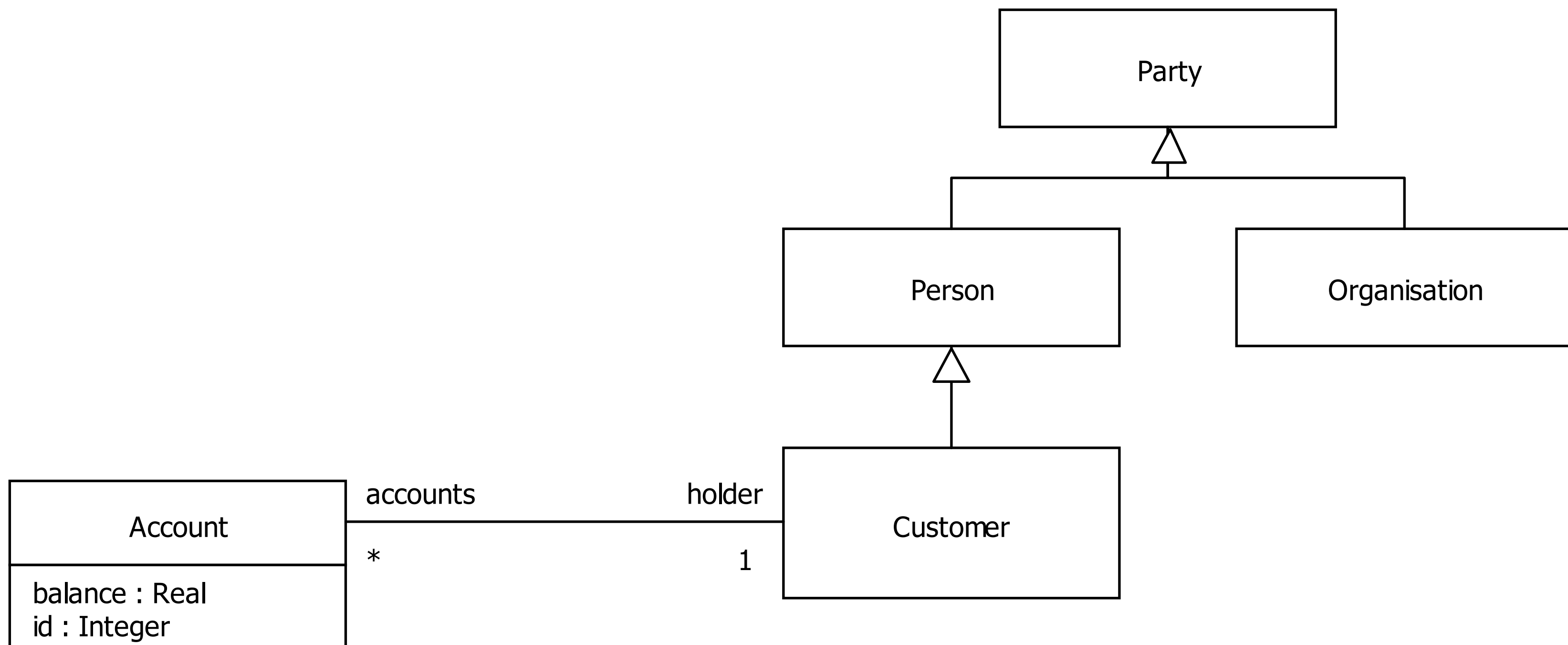


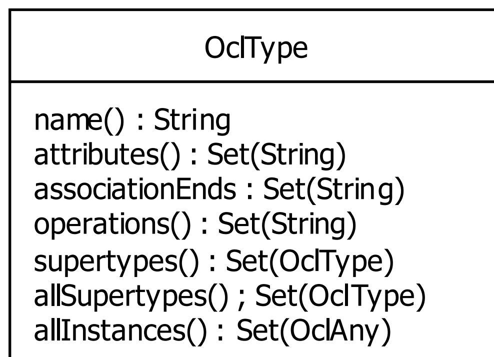


parlez|uml

© Jason Gorman 2005. All rights reserved.

Built-in OCL Types : OclType





Eg, Account.name() = "Account"

Eg, Account.attributes() = Set{"balance", "id"}

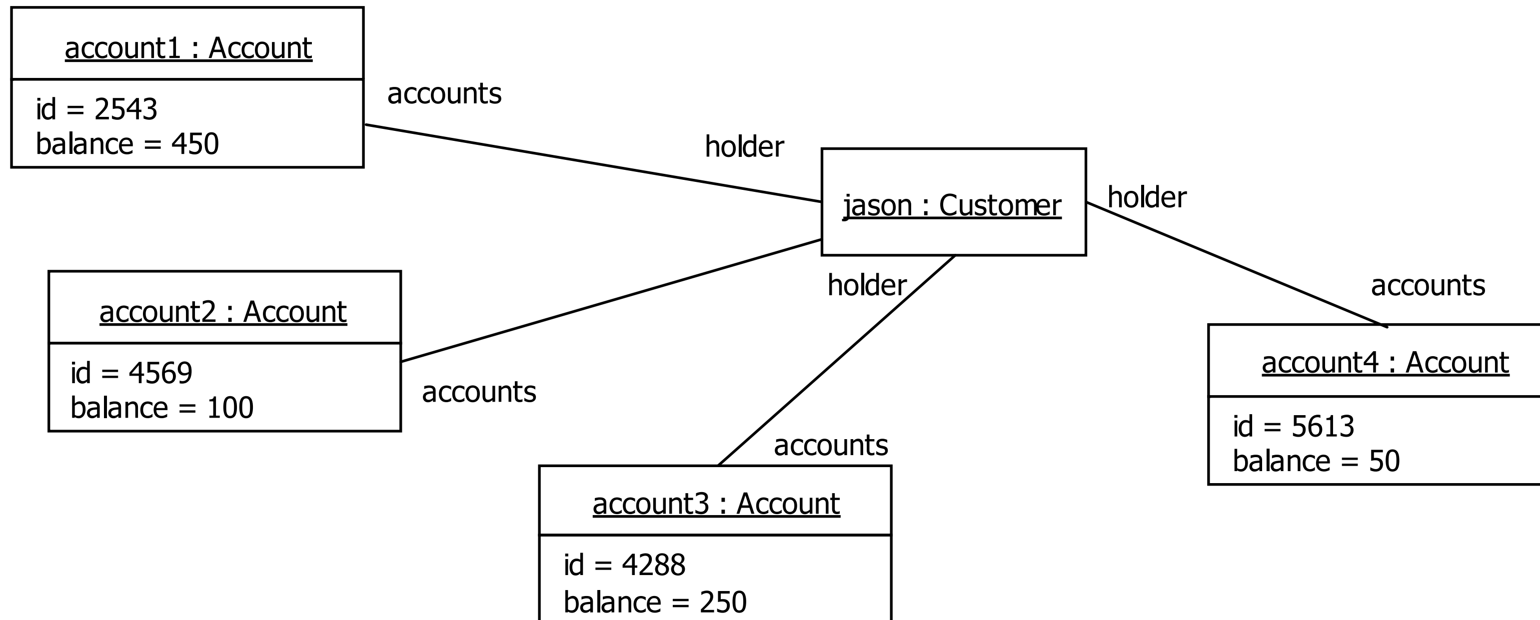
Eg, Customer.supertypes() = Set{Person}

Eg, Customer.allSupertypes() = Set{Person, Party}

parlez|uml

© Jason Gorman 2005. All rights reserved.

Built-in OCL Types : OclAny



OclAny
oclIsKindOf(OclType) : Booelan oclIsTypeOf(OclType) : Boolean oclAsType(OclType) : OclAny oclInState(OclState) : Boolean oclIsNew() : Boolean oclType() : OclType

Eg, jason.oclType() = Customer

Eg, jason.oclIsKindOf(Person) = true

Eg, jason.oclIsTypeOf(Person) = false

Eg, Account.allInstances() = Set{account1, account2, account3, account4}

parlez|uml

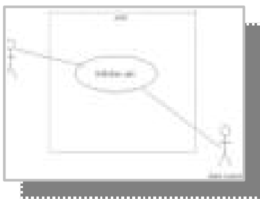
© Jason Gorman 2005. All rights reserved.

More on OCL

- OCL 1.5 Language Specification
- OCL Evaluator – a tool for editing, syntax checking & evaluating OCL
- Octopus OCL 2.0 Plug-in for Eclipse

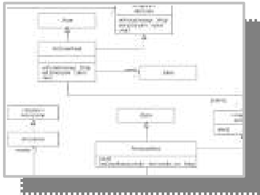
parlez|uml

© Jason Gorman 2005. All rights reserved.



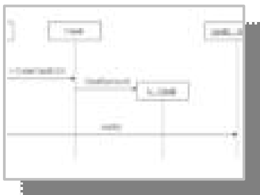
Use Case Diagrams

Model the users of the system and the goals they can achieve by using it



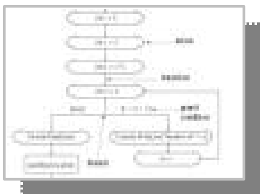
Class Diagrams

Model types of objects and the relationships between them.



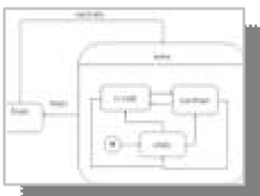
Sequence Diagrams

Model how objects interact to achieve functional goals



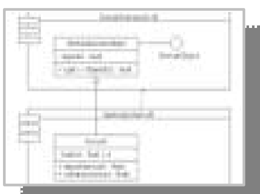
Activity Diagrams

Model the flow of use cases and single and multi-threaded code



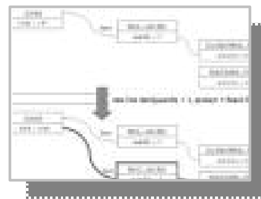
Statechart Diagrams

Model the behaviour of objects and event-driven applications



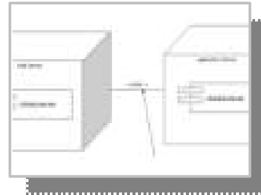
Design Principles

Create well-designed software that's easier to change and reuse



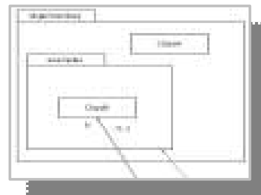
Object Diagrams & Filmstrips

Model snapshots of the running system and show how actions change object state



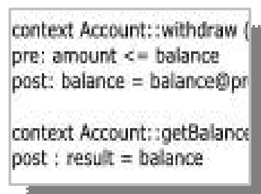
Implementation Diagrams

Model the physical components of a system and their deployment architecture



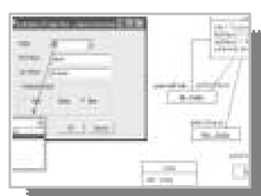
Packages & Model Management

Organise your logical and physical models with packages



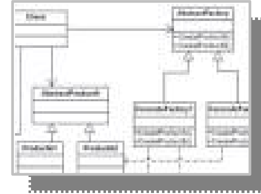
Object Constraint Language

Model business rules and create unambiguous specifications



User Experience Modeling

Design user-centred systems with UML



Design Patterns

Apply proven solutions to common OO design problems

UML for .NET Premium Package

Available exclusively from www.parlezuml.com

© Jason Gorman 2005. All rights reserved.