The Object-Z Specification Language

Dr. Constantinos Constantinides, P.Eng.

Department of Computer Science and
Software Engineering
Concordia University

Template for a class definition

$ClassName$

$< visibility\ list >$

$< parent\ class >$

$< state >$

$< initialization\ of\ state >$

$< list\ of\ operations >$

Example 1: Stack ADT – Visibility list and interface

$$\upharpoonleft (Push, Pop, Top)$$

State schema

$$
\begin{array}{|l}
\hline
elements : seq\ \ T \\
count : \mathbb{N} \\
\hline
count >= 0 \\
\hline
\end{array}
$$

Initialization of state

$$\begin{array}{|l} \hline \textit{INIT} \\ \hline elements = \langle\rangle \\ count = 0 \\ \hline \end{array}$$

Operation Push

$$
\begin{array}{|l}
\hline
\textit{Push} \\
\Delta(elements, count) \\
el? : T \\
\hline
elements' = \langle el? \rangle \frown elements \\
count' = count + 1 \\
\hline
\end{array}
$$

Operation Pop

$$
\begin{array}{|l}
\hline
\textit{Pop} \\
\Delta(elements, count) \\
el! : T \\
\hline
count > 0 \\
el! = head(elements) \\
elements' = tail(elements) \\
count' = count - 1 \\
\hline
\end{array}
$$

Operation Top

$$
\begin{array}{|l}
\hline
\textit{Top} \\
el! : T \\
\hline
count > 0 \\
el! = head(elements) \\
elements' = elements \\
count' = count \\
\hline
\end{array}
$$

$Stack[T]$

$\upharpoonright (Push, Pop, Top)$

$elements : seq\ T$
$count : \mathbb{N}$

$count >= 0$

$INIT$

$elements = \langle\rangle$
$count = 0$

$Push$

$\Delta(elements, count)$
$el? : T$

$elements' = \langle el? \rangle \frown elements$
$count' = count + 1$

Pop

$\Delta(elements, count)$
$el! : T$

$count > 0$
$el! = head(elements)$
$elements' = tail(elements)$
$count' = count - 1$

Top

$el! : T$

$count > 0$
$el! = head(elements)$
$elements' = elements$
$count' = count$

Instantiating a stack of natural numbers

$$
\begin{array}{|l}
\textit{IntStack} \\
\hline
\quad \begin{array}{|l} \textit{items} : \textit{Stack}(\mathbb{N}) \\ \hline \end{array} \\
\textit{Push} \mathrel{\widehat{=}} \textit{items}.\textit{Push} \\
\textit{Pop} \mathrel{\widehat{=}} \textit{items}.\textit{Pop} \\
\textit{Top} \mathrel{\widehat{=}} \textit{items}.\textit{Top} \\
\hline
\end{array}
$$

Inheritance

- A class in Object-Z may be specified as a specialization or extension of another class using inheritance.

- A class S can inherit another class P by including the name of the parent class after the visibility list in S.

Inheritance for *specialization*

- The subclass is a specialized version of the parent class, and thus satisfies the specification (interface) of the parent class in all relevant aspects, adding any particular behavior through overriding.

Inheritance for *extension*

- A subclass merely adds new behavior and does not modify or alter any of the inherited features.

Inheritance /cont.

- The subclass inherits every feature (variables, constants, initial state schema and operations), except the visibility list.

- The subclass must define its own visibility list.

- This implies that a feature that is declared private in the parent class may now be declared as visible, and vice versa: A visible feature from the parent class can now be declared as private by not being included in the visibility list of the subclass.

State and behavior in the presence of inheritance

- State variables in the parent class are merged with those of the subclass.

- The subclass may redefine a state variable, but only in a compatible way, expanding or restricting the type of a variable with the same name, for example restricting an integer variable to one that can hold only positive integers.

State and behavior in the presence of inheritance /cont.

- If an operation is redefined in the subclass, the declaration of an operation in the parent class is merged with that of the same operation in the subclass.

- An operation's predicate part is conjoined with that of the same operation in the subclass.

Subclassifying Stack to define BoundedStack
Inheritance for *specialization*

$BoundedStack[T]$

$\upharpoonright (Push, Pop, Top)$

$Stack[T]$

$capacity : \mathbb{N}$

$count <= capacity$

$\text{I}\textsc{nit}$

$capacity = 10$

$Push$

$count < capacity$

Example 2: Queue ADT

Front of Queue

Rear of Queue

$$\Lambda = < \boxed{el_1}, el_2, \ldots, \boxed{el_n} >$$

Head of Λ

Queue ADT – State schema

$$
\begin{array}{|l}
\hline
elements : seq\ T \\
count = \mathbb{N} \\
\hline
count >= 0 \\
\hline
\end{array}
$$

Initialization of state

$$\textit{I}{\small NIT}$$

$$elements = \langle\rangle$$

$$count = 0$$

Operation Enqueue

$$
\begin{array}{|l}
\hline
\textit{Enqueue} \\
\Delta(elements, count) \\
el? : T \\
\hline
elements' = elements \frown \langle el? \rangle \\
count' = count + 1 \\
\hline
\end{array}
$$

Operation Dequeue

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Dequeue}} \\
\hline
\Delta(elements, count) \\
el! : T \\
\hline
count > 0 \\
el! = head(elements) \\
elements' = tail(elements) \\
count' = count - 1 \\
\hline
\end{array}
$$

Instantiating a queue of natural numbers

IntQueue

$$items : Queue(\mathbb{N})$$

$$Enqueue \mathrel{\widehat=} items.Enqueue$$
$$Dequeue \mathrel{\widehat=} items.Dequeue$$

Subclassifying Queue to define BoundedQueue Inheritance for *specialization*

$BoundedQueue[T]$

$\upharpoonright (Enqueue, Dequeue)$

$Queue[T]$

$capacity : \mathbb{N}$

$count <= capacity$

$\textit{I}\textsc{nit}$

$capacity = 10$

$Enqueue$

$count < capacity$

Subclassifying Queue to define RessetableQueue
Inheritance for *extension*

$$\textit{ResettableQueue}[T]$$

$$\upharpoonright (\textit{Enqueue}, \textit{Dequeue}, \textit{Reset})$$

$$\textit{Queue}[T]$$

$$\textit{Reset}$$

$$\Delta(\textit{elements}, \textit{count})$$

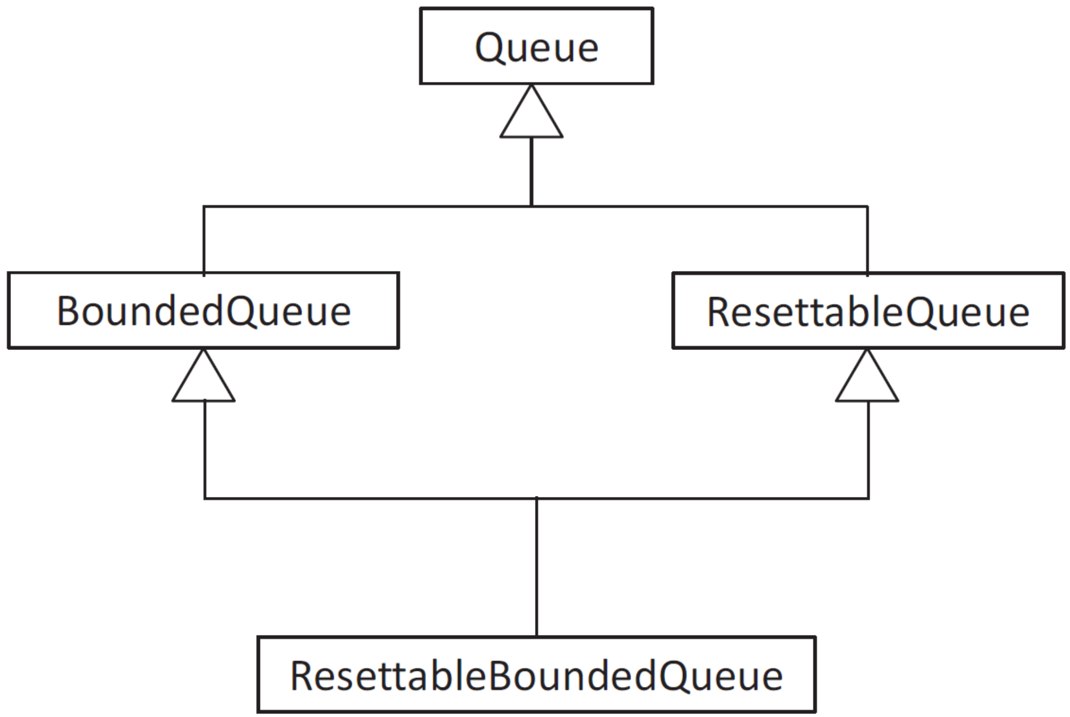
$$\textit{elements}' = \langle\rangle$$

$$\textit{count}' = 0$$

Inheritance for combination

- Object-Z supports multiple inheritance.
- A subclass is formed by combining features from more than one types.

Multiple inheritance



Class RessetableBoundedQueue Inheritance for *combination*

$ResettableBoundedQueue[T]$

$\upharpoonright (Enqueue, Dequeue, Reset)$

$BoundedQueue[T]$

$ResettableQueue[T]$

Handling errors and providing robust specifications

Name

$\upharpoonright (Op_1)$

...

$Op_1 OK$

...

...

Success

$result! : Report$

$result! = ok$

Error

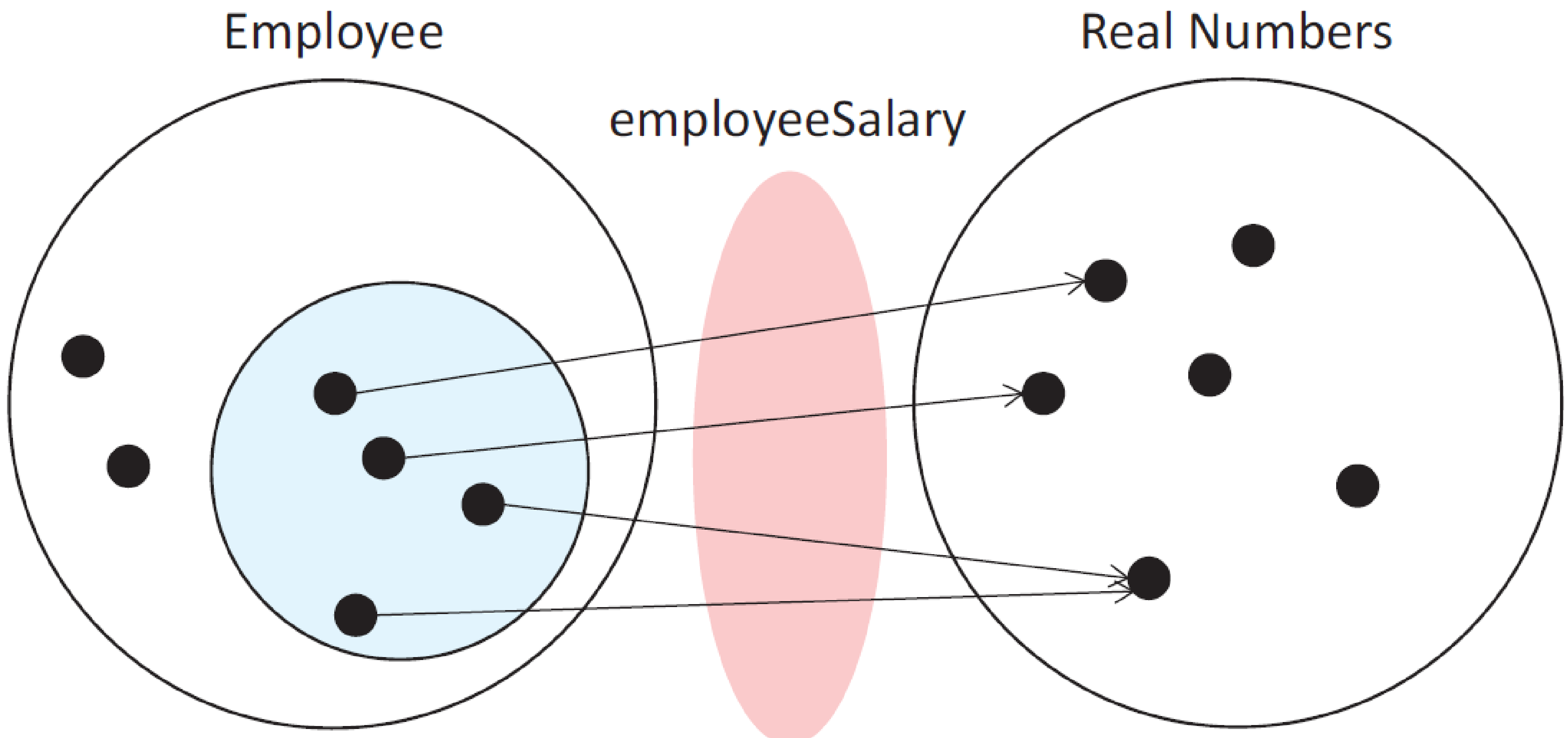
$result! : Report$

...

$result! = error$

$$Op_1 \mathrel{\widehat=} (Op_1 OK \land Success) \oplus Error$$

Example: Managing employees



Interface, state schema and initialization

$$\upharpoonright (AddEmployee, DeleteEmployee, ModifySalary)$$

$$employeeSalary : Employee \pfun \mathbb{R}$$

$$\forall d : \mathrm{dom}\ employeeSalary \bullet employeeSalary(d) > 0.0$$

$INIT$

$$employeeSalary = \varnothing$$

Operation AddEmployee

$$
\begin{array}{|l}
\hline
\textit{AddEmployee} \\
\Delta(\textit{employeeSalary}) \\
\textit{newEmployee}? : \textit{Employee} \\
\textit{salary}? : \mathbb{R} \\
\hline
\textit{salary}? > 0.0 \\
\textit{newEmployee}? \notin \operatorname{dom}\ \textit{employeeSalary} \\
\textit{employeeSalary}' = \textit{employeeSalary} \cup \{\textit{newEmployee}? \mapsto \textit{salary}?\} \\
\hline
\end{array}
$$

Operation DeleteEmployee

$$
\begin{array}{|l}
\hline
\textit{DeleteEmployee} \\
\Delta(employeeSalary) \\
who? : Employee \\
\hline
who? \in \operatorname{dom} \ employeeSalary \\
employeeSalary' = \{who?\} \mathbin{\lhd} employeeSalary \\
\hline
\end{array}
$$

Operation ModifySalary

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ModifySalary}} \\
\hline
\Delta(employeeSalary) \\
employee? : Employee \\
newSalary? : \mathbb{R} \\
\hline
newSalary? > 0.0 \\
employee? \in \mathrm{dom}\ \ employeeSalary \\
employeeSalary' = employeeSalary \oplus \{employee? \mapsto newSalary?\} \\
\hline
\end{array}
$$

Examining the specification: Initial state

employeeSalary	**dom employeeSalary**	**ran employeeSalary**
{ }	{ }	{ }

$$\forall d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{INIT}} \\ \hline
employeeSalary = \varnothing \\ \hline
\end{array}$$

AddEmployee(Syd, 90)

employeeSalary	dom employeeSalary	ran employeeSalary
{ }	{ }	{ }

$$\forall d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$
\begin{array}{|l}
\multicolumn{1}{l}{\text{\_\_ } AddEmployee \text{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}} \\
\Delta(employeeSalary) \\
newEmployee? : Employee \\
salary? : \mathbb{R} \\
\hline
\boxed{\begin{array}{l} salary? > 0.0 \ \checkmark \\ newEmployee? \notin \mathrm{dom}\ \ employeeSalary \ \checkmark \end{array}} \\
employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\} \\
\hline
\end{array}
$$

Precondition

AddEmployee(Syd, 90)

employeeSalary	**dom employeeSalary**	**ran employeeSalary**
{ (Syd, 90) }	{ Syd }	{ 90 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$
\begin{array}{|l}
\hline
\textit{AddEmployee} \\
\Delta(employeeSalary) \\
newEmployee? : Employee \\
salary? : \mathbb{R} \\
\hline
salary? > 0.0 \\
newEmployee? \notin \mathrm{dom}\ \ employeeSalary \\
\boxed{employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\}} \\
\hline
\end{array}
$$

Postcondition

AddEmployee(David, 100)

employeeSalary

{
(Syd, 90)
}

dom employeeSalary

{ Syd }

ran employeeSalary

{ 90 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{AddEmployee} \\
\Delta(employeeSalary) \\
newEmployee? : Employee \\
salary? : \mathbb{R} \\
\hline
\boxed{\begin{array}{l} salary? > 0.0 \ ✓ \\ newEmployee? \notin \mathrm{dom}\ \ employeeSalary \ ✓ \end{array}} \\
employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\} \\
\hline
\end{array}
$$

Precondition

AddEmployee(David, 100)

employeeSalary

{
(Syd, 90) ,
(David, 100)
}

dom employeeSalary

{ Syd, David }

ran employeeSalary

{ 90, 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$AddEmployee$

$\Delta(employeeSalary)$
$newEmployee? : Employee$
$salary? : \mathbb{R}$

$salary? > 0.0$
$newEmployee? \notin \mathrm{dom}\ \ employeeSalary$

Postcondition

$\boxed{employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\}}$

AddEmployee(Roger, 100)

employeeSalary	**dom employeeSalary**	**ran employeeSalary**
{ (Syd, 90) , (David, 100) }	{ Syd, David }	{ 90, 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$\begin{array}{|l}
\hline
\textit{AddEmployee} \\
\Delta(employeeSalary) \\
newEmployee? : Employee \\
salary? : \mathbb{R} \\
\hline
\boxed{\begin{array}{l} salary? > 0.0 \\ newEmployee? \notin \mathrm{dom}\ \ employeeSalary \end{array}} \\
employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\} \\
\hline
\end{array}$$

✓ ✓ **Precondition**

AddEmployee(Roger, 100)

employeeSalary	**dom employeeSalary**	**ran employeeSalary**
{ (Syd, 90) , (David, 100), (Roger, 100) }	{ Syd, David, Roger }	{ 90, 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$AddEmployee$

$\Delta(employeeSalary)$
$newEmployee? : Employee$
$salary? : \mathbb{R}$

$salary? > 0.0$
$newEmployee? \notin \mathrm{dom}\ \ employeeSalary$
$\boxed{employeeSalary' = employeeSalary \cup \{newEmployee? \mapsto salary?\}}$

Postcondition

DeleteEmployee(Syd)

employeeSalary

{
(Syd, 90) ,
(David, 100),
(Roger, 100)
}

dom employeeSalary

{ Syd, David, Roger }

ran employeeSalary

{ 90, 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$\begin{array}{|l}
\multicolumn{1}{l}{DeleteEmployee} \\
\hline
\Delta(employeeSalary) \\
who? : Employee \\
\hline
\boxed{who? \in \mathrm{dom}\ \ employeeSalary \ \checkmark} \\
employeeSalary' = \{who?\} \lhd employeeSalary \\
\hline
\end{array}$$

Precondition

DeleteEmployee(Syd)

employeeSalary

{
~~(Syd, 90)~~ ,
(David, 100),
(Roger, 100)
}

dom employeeSalary

{ ~~Syd,~~ David, Roger }

ran employeeSalary

~~{ 90,~~ 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad DeleteEmployee \qquad\qquad\qquad\qquad\qquad\qquad}} \\
\Delta(employeeSalary) \\
who? : Employee \\
\hline
who? \in \mathrm{dom}\ \ employeeSalary \\
\boxed{employeeSalary' = \{who?\} \lhd employeeSalary} \\
\hline
\end{array}$$

Postcondition

ModifySalary(David, 110)

employeeSalary

{
(David, 100),
(Roger, 100)
}

dom employeeSalary

{ David, Roger }

ran employeeSalary

{ 100 }

$$\forall\, d : \mathrm{dom}\ \ employeeSalary \bullet employeeSalary(d) > 0.0$$ ✓ **Invariant**

ModifySalary

$$\Delta(employeeSalary)$$
$$employee? : Employee$$
$$newSalary? : \mathbb{R}$$

$$\boxed{\begin{array}{l} newSalary? > 0.0 \ ✓ \\ employee? \in \mathrm{dom}\ \ employeeSalary \ ✓ \end{array}}$$ **Precondition**

$$employeeSalary' = employeeSalary \oplus \{employee? \mapsto newSalary?\}$$

ModifySalary(David, 110)

employeeSalary	**dom employeeSalary**	**ran employeeSalary**
{ (David, 110), (Roger, 100) }	{ David, Roger }	{ 100, 110 }

$$\forall\, d : \mathrm{dom}\ \mathit{employeeSalary} \bullet \mathit{employeeSalary}(d) > 0.0$$ ✓ **Invariant**

ModifySalary

$\Delta(\mathit{employeeSalary})$
$\mathit{employee?} : \mathit{Employee}$
$\mathit{newSalary?} : \mathbb{R}$

$\mathit{newSalary?} > 0.0$
$\mathit{employee?} \in \mathrm{dom}\ \mathit{employeeSalary}$

Postcondition

$\boxed{\mathit{employeeSalary}' = \mathit{employeeSalary} \oplus \{\mathit{employee?} \mapsto \mathit{newSalary?}\}}$

Example: CreditCard
Visibility list, constants, state and initialization

$CreditCard$

$\upharpoonright (Withdraw, Deposit, GetAvailableFunds)$

$$number : \mathbb{N}$$
$$limit : \mathbb{R}$$

$$limit \in \{1000, 5000, 10000\}$$

$$balance : \mathbb{R}$$

$$balance + limit \geq 0$$

$\mathit{I}\textsc{nit}$

$$balance = 0$$

Operation Withdraw

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{Withdraw}} \\
\Delta(balance) \\
amount? : \mathbb{R} \\
\hline
amount? > 0 \\
amount? \leq balance + limit \\
balance' = balance - amount? \\
\hline
\end{array}
$$

Operation Deposit

$$
\begin{array}{|l}
\hline
\textit{Deposit} \\
\Delta(balance) \\
amount? : \mathbb{R} \\
\hline
amount? > 0 \\
balance' = balance + amount? \\
\hline
\end{array}
$$

Operation GetAvailableFunds

$$
\begin{array}{|l}
\hline
\textit{GetAvailableFunds} \\
amount! : \mathbb{R} \\
\hline
amount! = balance + limit \\
\hline
\end{array}
$$

Example: CreditCard2
Subclassifying CreditCard

CreditCard2

$\upharpoonright (Withdraw, Deposit, GetAvailableFunds)$

$CreditCard$

$withdrawals : \mathbb{N}$

INIT

$withdrawals = 0$

Withdraw

$\Delta(withdrawals)$

$withdrawals' = withdrawals + 1$

Example: CreditCompany Visibility list, state and initialization

$CreditCompany$

$\upharpoonright (AddAccount, DeleteAccount)$

$accounts : \mathbb{P}\ CreditCard$
$count : \mathbb{N}$

$\forall a_i, a_j : accounts \bullet a_i.number \neq a_j.number$
$count = \#accounts$

$\text{I}{\small NIT}$

$accounts = \{\}$

Operation AddAccount

$$
\begin{array}{|l}
\hline
\textit{AddAccount} \\
\Delta(accounts) \\
account? : CreditCard \\
\hline
account? \notin accounts \\
accounts' = accounts \cup \{account?\} \\
count' = count + 1 \\
\hline
\end{array}
$$

Operation DeleteAccount

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{DeleteAccount} \\
\Delta(accounts) \\
account? : CreditCard \\
\hline
account? \in accounts \\
accounts' = accounts \setminus \{account?\} \\
count' = count - 1 \\
\hline
\end{array}
$$

Inheritance and subtyping

- Each class defines a type and all instances of the class constitute legitimate values of that type.
- Every instance of a subclass is also an instance of a superclass, but not vice-versa.
- The type defined by the subclass is a subset of the type defined by its superclasses as the set of all instances of a subclass is included in the set of all instances of its superclass.

Polymorphism

- In

  $$account :\downarrow Account$$

  variable *account* can hold an instance of *Account* or any of its subclasses.

- The declaration

  $$accounts : \mathbb{P} \downarrow Account$$

  indicates that *account* is a set of elements from *Account* as well as from any of its subclasses.

Example: Bank
[To be covered in tutorials this week]

$Account$

$\upharpoonright (accountNumber, Deposit, Withdraw)$

$SavingsAccount$

$\upharpoonright (accountNumber, balance, Deposit, Withdraw)$
$Account$

$Bank$

$accounts : \mathbb{P} \downarrow Account$

$\forall a_1, a_2 : accounts \bullet a_1.accountNumber = a_2.accountNumber \Leftrightarrow a_1 = a_2$

Cancellation and redefinition of features through renaming

$A[T]$

$x : T$
$y : \mathbb{P}\,T$

$x \in y$

Op

$\Delta(x)$
$x? : T$

$x? \in y$
$x' = x?$

Cancellation and redefinition of features through renaming /cont.

$B[T]$

$\upharpoonleft (Op)$

$A[y1/y, Op1/Op]$

$y : \mathrm{bag}\ T$

$x \in y$

Op

$\Delta(x)$

$x? : T$

$x? \in y$

$x' = x?$

Explicit redefinition and removal of operations

$$
\begin{array}{|l}
\hline
\textit{DoublePushStack}[T] \\
\hline
\upharpoonright (Push, Pop, Top) \\
BoundedStack[T][\mathbf{redef}\ Push] \\
\\
\quad \begin{array}{|l}
\hline
Push \\
\hline
\Delta(elements, count) \\
item? : T \\
\hline
count < capacity - 1 \\
elements' = \langle item?, item? \rangle \frown elements \\
count' = count + 2 \\
\hline
\end{array} \\
\hline
\end{array}
$$

Explicit redefinition and removal of operations /cont.

$OnlyPushStack[T]$

$\upharpoonright (Push)$

$Stack[T][\textbf{remove}\ Pop]$