Object-Z

is a specification language extending Z so as to facilitate the specification of systems in an object-oriented style.

The view is taken that systems are composed of communicating objects.

When specifying a system in Object-Z,

- identify and specify the underlying objects;
- specify the system in terms of the communication between the underlying objects.

An object may itself be a system of communicating objects.

The Class Construct: Encapsulation

Class Name

visibility list
inherited classes
local types
state
initial state
operations

- the class construct encapsulates all relevant features; it is like a template from which objects of the class can be stamped
- the visibility list specifies the interface between object of the class and their environment
- a class incorporates all the features of its inherited classes
- local types have the syntax of Z global types
- the state, initial state and operations have a syntax based on that for Z schemas
- variables declared in the state are called **attributes**
- an instance of a class is an assignment of values to attributes consistent with the state; at any time an object of a class will have an associated value which is some instance of the class

Object-Z Case Study: A Buffer of Messages

$Buffer$

$\upharpoonright(max, \textsc{Init}, Join, Leave)$ [visibility list]

$max : \mathbb{N}$ [constant]

$items : \mathrm{seq}\ MSG$ [state schema]

$\#items \leqslant max$

$\textsc{Init}$

$items = \langle\ \rangle$ [initial state]

$Join$

$\Delta(items)$

$msg? : MSG$

$\#items < max$

$items' = items \frown \langle msg? \rangle$

$Leave$

$\Delta(items)$

$msg! : MSG$ [operation]

$items \neq \langle\ \rangle$

$items = \langle msg! \rangle \frown items'$

- the state is an unnamed schema
- the state schema is implicitly merged into the initial schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textsc{Init}} \\
\hline
max : \mathbb{N}; \quad items : \mathrm{seq}\, MSG \\
\hline
\# items \leqslant max \land items = \langle\ \rangle \\
\hline
\end{array}
$$

- the state schema in both primed and unprimed form is implicity merged into each operation schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{Join} \\
\hline
max, max' : \mathbb{N}; \quad items, items' : \mathrm{seq}\, MSG \\
msg? : MSG \\
\hline
\# items \leqslant max \land \# items' \leqslant max \land \# items < max \\
items' = items \frown \langle msg? \rangle \land max' = max \\
\hline
\end{array}
$$

- the Δ convention is modified: only attributes that may change are listed—attributes not listed do not change

Inheritance

$LossyBuffer$

$Buffer$

$Lose$

$\Delta(items)$

$items \neq \langle\,\rangle$

$items' = \operatorname{tail} items$

$LossyBuffer$

$max : \mathbb{N}; \quad items : \operatorname{seq} MSG$

$\# items \leqslant max$

$\textsc{Init}$

$items = \langle\,\rangle$

$Join$

$\ldots$

$Leave$

$\ldots$

$Lose$

$\Delta(items)$

$items \neq \langle\,\rangle \land items' = \operatorname{tail} items$

Instantiation and Communication

$$
\begin{array}{l}
\underline{\mathit{Channel}} \\
\hline
b_1, b_2 : \mathit{Buffer} \\
\hline
b_1 \neq b_2 \\
\hline
\underline{\textsc{Init}} \\
b_1.\textsc{Init} \\
b_2.\textsc{Init} \\
\hline
\mathit{Join} \mathrel{\widehat{=}} b_1.\mathit{Join} \\
\mathit{Leave} \mathrel{\widehat{=}} b_2.\mathit{Leave} \\
\mathit{Transfer} \mathrel{\widehat{=}} b_1.\mathit{Leave} \parallel b_2.\mathit{Join} \\
\hline
\end{array}
$$

- an object is a variable of class type — objects are instantiations of classes
- objects have **integrity** — change state via class operations only
- objects have **persistence** — exist from creation to deallocation
- objects communicate by message passing — engage in cooperative operations

The initial schema of *Channel* is equivalent to:

$$\begin{array}{|l}
\textsc{Init} \\
\hline
b_1.items = \langle\ \rangle \land b_2.items = \langle\ \rangle \\
\hline
\end{array}$$

The operations of *Channel* are equivalent to:

$$\begin{array}{|l}
Join \\
\hline
msg? : MSG \\
\hline
open(b_1) \\
\# b_1.items < b_1.max \\
b_1.items' = b_1.items \frown \langle msg? \rangle \\
b_1.max' = b_1.max \\
\hline
\end{array}
\qquad
\begin{array}{|l}
Leave \\
\hline
msg! : MSG \\
\hline
open(b_2) \\
b_2.items \neq \langle\ \rangle \\
b_2.items = \langle msg! \rangle \frown b_2.items' \\
b_2.max' = b_2.max \\
\hline
\end{array}$$

Transfer

$$\exists\, msg : MSG$$
$$\qquad b_1.Leave[msg/msg!]$$
$$\qquad b_2.Join[msg/msg?]$$

or

Transfer

$$open(b_1, b_2)$$
$$b_1.items \neq \langle\ \rangle$$
$$\# b_2.items < b_2.max$$
$$\exists\, msg : MSG$$
$$\qquad b_1.items = \langle msg \rangle ^\frown b_1.items'$$
$$\qquad b_2.items' = b_2.items ^\frown \langle msg \rangle$$
$$b_1.max' = b_1.max$$
$$b_2.max' = b_2.max$$

In general

- $a.op$
  denotes the operation op performed upon object a; the operation op must be one of the operations specified in the class of a

- $a.op_1 \parallel b.op_2$
  denotes the operation op_1 performed upon object a, in parallel with the operation op_2 performed upon object b; inputs and outputs having the same base name (i.e. apart from the '?' and '!') are identified (equated) and hidden

Aggregation and Identity

BufferSystem

$$\begin{array}{|l}
\hline
buffers : \mathbb{P}\, Buffer \\
\hline
\end{array}$$

INIT

$$buffers = \varnothing$$

AddBuffer

$$\begin{array}{|l}
\Delta(buffers) \\
b? : Buffer \\
\hline
b? \notin buffers \wedge b?.\mathrm{INIT} \\
buffers' = buffers \cup \{b?\}
\end{array}$$

RemoveBuffer

$$\begin{array}{|l}
\Delta(buffers) \\
b? : Buffer \\
\hline
b? \in buffers \\
buffers' = buffers - \{b?\}
\end{array}$$

SelectBuffer

$$\begin{array}{|l}
b? : Buffer \\
\hline
b? \in buffers
\end{array}$$

Select2Buffers

$$\begin{array}{|l}
b_1?, b_2? : Buffer \\
\hline
\{b_1?, b_2?\} \subseteq buffers \wedge b_1? \neq b_2?
\end{array}$$

$$Join \mathrel{\widehat=} SelectBuffer \bullet b?.Join$$

$$Leave \mathrel{\widehat=} SelectBuffer \bullet b?.Leave$$

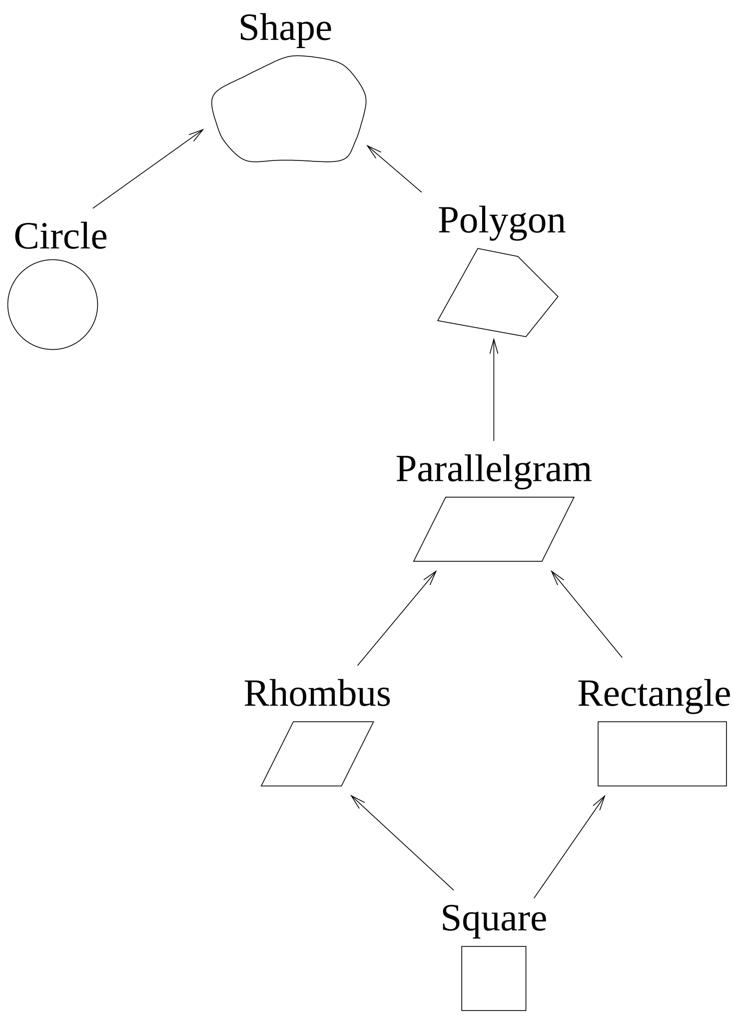
$$Transfer \mathrel{\widehat=} Select2Buffers \bullet b_1?.Leave \parallel b_2?.Join$$

The operations *Join* and *Transfer* are equivalent to:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Join}} \\
b? : \mathit{Buffer} \\
msg? : \mathit{MSG} \\
\hline
open(b?) \\
b? \in buffers \\
\# b?.items < b?.max \\
b?.items' = b?.items \frown \langle msg? \rangle \\
b?.max' = b?.max \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Transfer}} \\
b_1?, b_2? : \mathit{Buffer} \\
\hline
open(b_1?, b_2?) \\
\{b_1?, b_2?\} \subseteq buffers \\
b_1? \neq b_2? \\
b_1?.items \neq \langle\ \rangle \\
\# b_2?.items < b_2?.max \\
\exists\, msg : \mathit{MSG} \bullet \\
\qquad b_1?.items = \langle msg \rangle \frown b_1?.items' \\
\qquad b_2?.items' = b_2?.items \frown \langle msg \rangle \\
b_1?.max' = b_1?.max \\
b_2?.max' = b_2?.max \\
\hline
\end{array}
$$

Object-Z Case Study: A Shapes Hierarchy



$Vector == \mathbb{R} \times \mathbb{R}$

$\_ + \_ : Vector \times Vector \rightarrow Vector$

$|\_|: Vector \rightarrow \mathbb{R}$

$\_ \perp \_ : Vector \leftrightarrow Vector$

$$
\begin{array}{|l}
\textit{Shape} \\
\hline
\quad \begin{array}{|l}
\hline
refpoint : Vector \\
perim : \mathbb{R} \\
\hline
perim > 0 \\
\hline
\end{array} \\
\\
\quad \begin{array}{|l}
\textit{Translate} \\
\hline
\Delta(refpoint) \\
v? : Vector \\
\hline
refpoint' = refpoint + v? \\
\hline
\end{array} \\
\hline
\end{array}
$$

Circle

$$\mathit{Shape}[\mathit{centre}/\mathit{refpoint}, \mathit{circum}/\mathit{perim}]$$

$$\begin{array}{|l}
\mathit{radius} : \mathbb{R} \\
\hline
\mathit{circum} = 2\,\pi\,\mathit{radius}
\end{array}$$

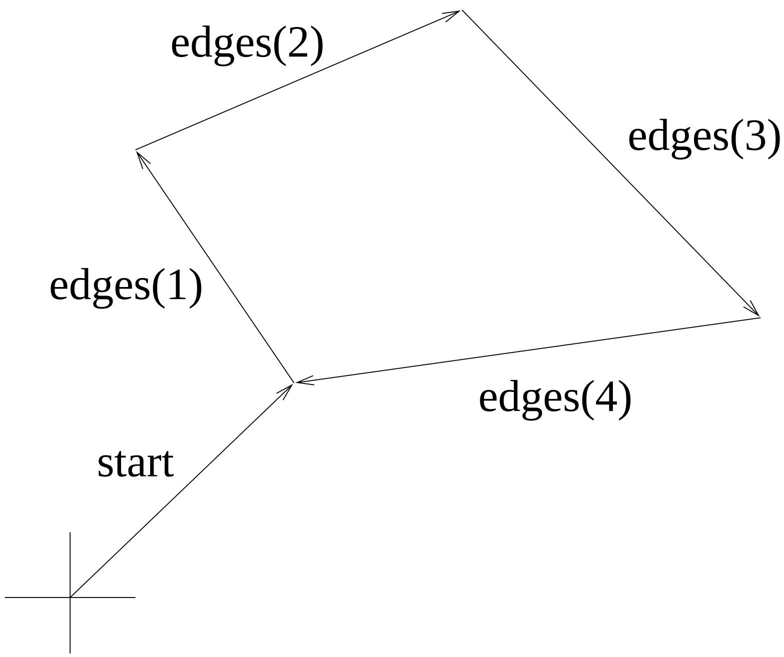
i.e.

Circle

$$\begin{array}{|l}
\mathit{centre} : \mathit{Vector} \\
\mathit{circum} : \mathbb{R} \\
\mathit{radius} : \mathbb{R} \\
\hline
\mathit{circum} > 0 \\
\mathit{circum} = 2\,\pi\,\mathit{radius}
\end{array}$$

Translate

$$\begin{array}{|l}
\Delta(\mathit{centre}) \\
v? : \mathit{Vector} \\
\hline
\mathit{centre}' = \mathit{centre} + v?
\end{array}$$



$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Polygon}} \\
\hline
Shape[start/refpoint] \\
\quad \begin{array}{|l}
\hline
edges : \mathrm{seq}\ Vector \\
\hline
\#edges \geqslant 3 \\
\mathbf{O} \notin \mathrm{ran}\ edges \\
(\sum i : \mathrm{dom}\ edges \bullet edges(i)) = \mathbf{O} \\
perim = \sum i : \mathrm{dom}\ edges \bullet | edges(i) | \\
\ldots \qquad\qquad\qquad\qquad [\textsf{connectivity} \ldots] \\
\hline
\end{array} \\
\hline
\end{array}
$$

Parallelogram

Polygon

$\# edges = 4$

$edges(1) + edges(3) = \mathbf{O}$

Rhombus

Parallelogram

$\mid edges(1) \mid = \mid edges(2) \mid$

Rectangle

Parallelogram

$edges(1) \perp edges(2)$

Square

Rhombus

Rectangle

Expanding *Square* gives

Square

$$
\begin{array}{l}
\quad \textit{start} : \textit{Vector} \\
\quad \textit{perim} : \mathbb{R} \\
\quad \textit{edges} : \mathrm{seq}\ \textit{Vector} \\
\hline
\quad \#\textit{edges} = 4 \land \mathbf{O} \notin \mathrm{ran}\ \textit{edges} \\
\quad (\sum i : \mathrm{dom}\ \textit{edges} \bullet \textit{edges}(i)) = \mathbf{O} \\
\quad \textit{perim} = \sum i : \mathrm{dom}\ \textit{edges} \bullet \mid \textit{edges}(i) \mid \\
\quad \textit{edges}(1) + \textit{edges}(3) = \mathbf{O} \\
\quad \mid \textit{edges}(1) \mid = \mid \textit{edges}(2) \mid \land\ \textit{edges}(1) \perp \textit{edges}(2)
\end{array}
$$

Translate

$$
\begin{array}{l}
\quad \Delta(\textit{start}) \\
\quad v? : \textit{Vector} \\
\hline
\quad \textit{start}' = \textit{start} + v?
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Figure} \\
\quad \begin{array}{|l}
\hline
\textit{shapes} : \mathbb{P} \downarrow \textit{Shape} \\
\textit{totalperim} : \mathbb{R} \\
\hline
\textit{totalperim} = \sum s : \textit{shapes} \bullet s.\textit{perim} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{SelectShape} \\
s? : \downarrow \textit{Shape} \\
\hline
s? \in \textit{shapes} \\
\hline
\end{array} \\
\quad \textit{ShapeTranslate} \mathrel{\widehat{=}} \textit{SelectShape} \bullet s?.\textit{Translate} \\
\quad \textit{FigureTranslate} \mathrel{\widehat{=}} \; \| \; s : \textit{shapes} \bullet s.\textit{Translate} \\
\hline
\end{array}
$$

i.e.

$$
\begin{array}{|l}
\hline
\textit{FigureTranslate} \\
v? : \textit{Vector} \\
\hline
\forall s : \textit{shapes} \bullet s.\textit{Translate} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{FigureTranslate} \\
v? : \textit{Vector} \\
\hline
\forall s : \textit{shapes} \bullet \\
\qquad \textit{open}(s) \\
\qquad s.\textit{refpoint}' = s.\textit{refpoint} + v? \\
\qquad s.\textit{perim}' = s.\textit{perim} \\
\hline
\end{array}
$$

Object-Z Case Study: An Electronic Key System

Informal Description

- there is a fixed set of magnetic keys
- there is a fixed set of rooms
- each key has access to a subset of these rooms
- a room may be added to the set accessed by a key
- a room may be removed from the set accessed by a key

Key

$\upharpoonright(Insert)$

Insert

$key! : Key$

$key! = self$

Keys

$\upharpoonright(keys, \textsc{Init}, Insert)$

$keys : \mathbb{P}\, Key$

$\textsc{Init}$

$keys \neq \varnothing$

SelectKey

$k? : Key$

$k? \in keys$

$Insert \mathrel{\widehat{=}} SelectKey \bullet k?.Insert$

Room

$$\upharpoonright(\textsc{Init}, \mathit{InsertedAndUnlock}, \mathit{Lock})$$

$$\mathit{Status} ::= \mathit{locked} \mid \mathit{unlocked}$$

$$\mathit{status} : \mathit{Status}$$

$\textsc{Init}$

$$\mathit{status} = \mathit{locked}$$

Inserted

$$\mathit{room}! : \mathit{Room}$$

$$\mathit{room}! = \mathit{self}$$

Unlock

$$\Delta(\mathit{status})$$

$$\mathit{status} = \mathit{locked} \land \mathit{status}' = \mathit{unlocked}$$

$$\mathit{InsertedAndUnlock} \mathrel{\widehat{=}} \mathit{Inserted} \land \mathit{Unlock}$$

Lock

$$\Delta(\mathit{status})$$

$$\mathit{status} = \mathit{unlocked} \land \mathit{status}' = \mathit{locked}$$

$$
\begin{array}{l}
\textit{Rooms} \\
\hline
\upharpoonright(\mathit{rooms}, \textsc{Init}, \mathit{Unlock}, \mathit{Lock}) \\[1ex]
\quad \mathit{rooms} : \mathbb{P}\, \mathit{Room} \\[1ex]
\quad \textsc{Init} \\
\quad \hline
\quad \mathit{rooms} \neq \emptyset \\
\quad \forall\, r : \mathit{rooms} \bullet r.\textsc{Init} \\[1ex]
\quad \mathit{SelectRoom} \\
\quad \hline
\quad r? : \mathit{Room} \\
\quad \hline
\quad r? \in \mathit{rooms} \\[1ex]
\mathit{Unlock} \mathrel{\widehat=} \mathit{SelectRoom} \bullet r?.\mathit{InsertedAndUnlock} \\
\mathit{Lock} \mathrel{\widehat=} \mathit{SelectRoom} \bullet r?.\mathit{Lock}
\end{array}
$$

DataBase

$$\upharpoonright(access, \textsc{Init}, AuthorizeAccess, RescindAccess, CheckAccess)$$

$$access : Key \leftrightarrow Room$$

$\textsc{Init}$

$$access = \varnothing$$

AuthorizeAccess

$$\Delta(access)$$
$$key? : Key$$
$$room? : Room$$

$$\neg\ (key?\ \underline{access}\ room?)$$
$$access' = access \cup \{(key?, room?)\}$$

RescindAccess

$$\Delta(access)$$
$$key? : Key$$
$$room? : Room$$

$$key?\ \underline{access}\ room?$$
$$access' = access - \{(key?, room?)\}$$

CheckAccess

$$key? : Key$$
$$room? : Room$$

$$key?\ \underline{access}\ room?$$

$KeySystem$

$$keys : Keys$$
$$rooms : Rooms$$
$$database : DataBase$$

$$database.access \subseteq keys.keys \times rooms.rooms$$

$\textsc{Init}$

$$keys.\textsc{Init} \land rooms.\textsc{Init} \land database.\textsc{Init}$$

$$AuthorizeAccess \mathrel{\widehat{=}} database.AuthorizeAccess$$
$$RescindAccess \mathrel{\widehat{=}} database.RescindAccess$$
$$\begin{aligned} Unlock \mathrel{\widehat{=}} {} & (keys.Insert \land rooms.Unlock) \\ & \parallel \\ & database.CheckAccess \end{aligned}$$
$$Lock \mathrel{\widehat{=}} rooms.Lock$$

Case Study: The Game of Tic Tac Toe

An Informal View:

- there are two players and a board
- the board consists of 9 positions in a 3×3 array
- initially all positions are unoccupied
- the players take it in turns to occupy unoccupied positions
- the first player to occupy three positions in a horizontal, vertical or diagonal row is the winner

A Formal Description – Initial Abstractions

A Data Structure for the Board

$$Posn == 1 \mathinner{.\,.} 3 \times 1 \mathinner{.\,.} 3$$

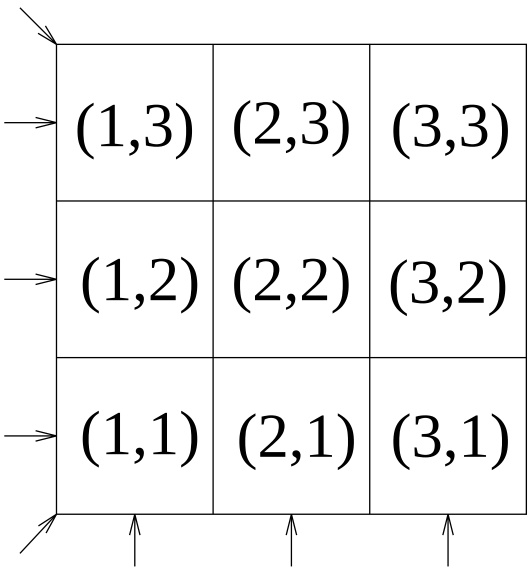
the abstraction:

(1,3)	(2,3)	(3,3)
(1,2)	(2,2)	(3,2)
(1,1)	(2,1)	(3,1)

Three in a Row (1st Abstraction)

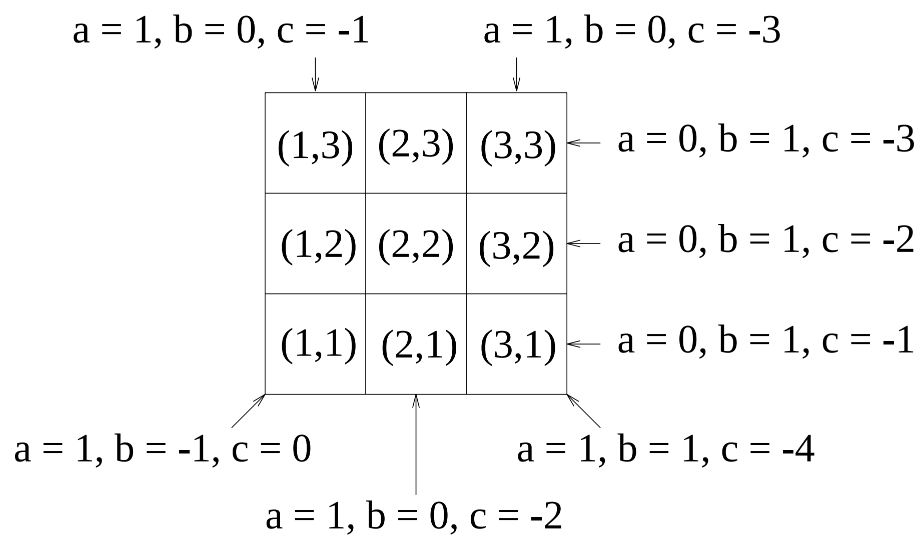
$$3InRow : \mathbb{P}\,Posn \rightarrow \mathbb{B}$$

$$
\begin{array}{l}
\forall\, ps : \mathbb{P}\,Posn \bullet \\
\quad 3InRow(ps) \Leftrightarrow \\
\qquad\qquad \{(1,1),(1,2),(1,3)\} \subseteq ps \\
\qquad\quad \vee\ \{(2,1),(2,2),(2,3)\} \subseteq ps \\
\qquad\quad \vee\ \{(3,1),(3,2),(3,3)\} \subseteq ps \\
\qquad\quad \vee\ \{(1,1),(2,1),(3,1)\} \subseteq ps \\
\qquad\quad \vee\ \{(1,2),(2,2),(3,2)\} \subseteq ps \\
\qquad\quad \vee\ \{(1,3),(2,3),(3,3)\} \subseteq ps \\
\qquad\quad \vee\ \{(1,1),(2,2),(3,3)\} \subseteq ps \\
\qquad\quad \vee\ \{(1,3),(2,2),(3,1)\} \subseteq ps
\end{array}
$$



Three in a Row (2nd Abstraction)

$$
\begin{array}{|l}
3InRow : \mathbb{P}\, Posn \rightarrow \mathbb{B} \\
\hline
\forall\, ps : \mathbb{P}\, Posn \bullet \\
\quad 3InRow(ps) \;\Leftrightarrow\; \exists\, a, b, c : \mathbb{Z} \bullet \\
\qquad \{a, b, c\} \neq \{0\} \\
\qquad \#\{(x, y) : ps \mid ax + by + c = 0\} = 3
\end{array}
$$



$$Colour \;::=\; black \mid white$$

Board

$$\begin{array}{l} bposn, wposn : \mathbb{P}\, Posn \\ turn : Colour \\ \hline bposn \cap wposn = \varnothing \end{array}$$

INIT

$$\begin{array}{l} bposn = \varnothing \\ wposn = \varnothing \\ turn = black \end{array}$$

BlackMove

$$\begin{array}{l} \Delta(bposn, turn); \quad p? : Posn \\ \hline \neg\, 3InRow(wposn) \\ p? \notin bposn \cup wposn \\ bposn' = bposn \cup \{p?\} \\ turn = black \land turn' = white \end{array}$$

WhiteMove

$$\begin{array}{l} \Delta(wposn, turn); \quad p? : Posn \\ \hline \neg\, 3InRow(bposn) \\ p? \notin bposn \cup wposn \\ wposn' = wposn \cup \{p?\} \\ turn = white \land turn' = black \end{array}$$

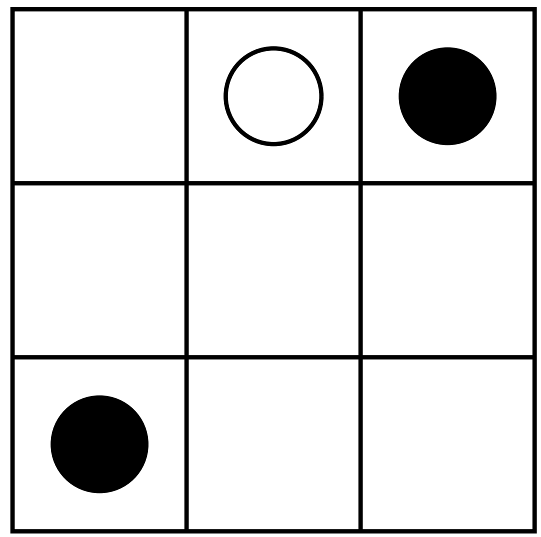
TheWinner

$$\begin{array}{l} winner! : Colour \\ \hline 3InRow(bposn) \ \lor \ 3InRow(wposn) \\ 3InRow(bposn) \ \Rightarrow \ winner! = black \land 3InRow(wposn) \ \Rightarrow \ winner! = white \end{array}$$

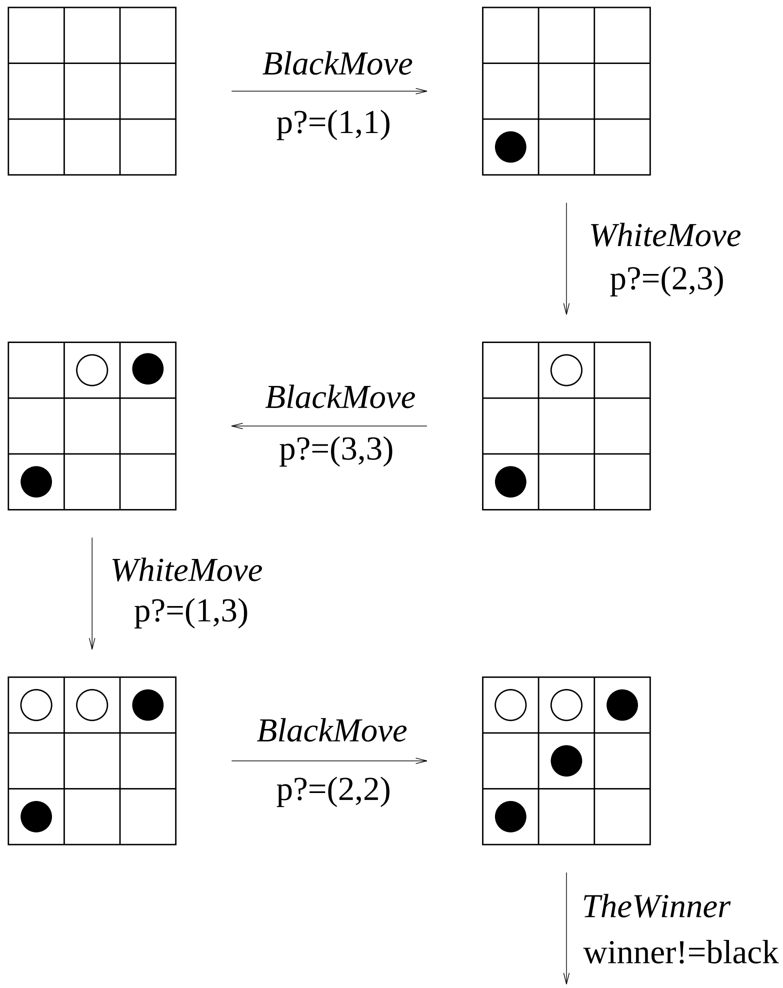
Board Example: A Static View

$bposn = \{(1,1),(3,3)\}$
$wposn = \{(2,3)\}$
$turn = white$

$3InRow(bposn) = false$
$3InRow(wposn) = false$
$bposn \cap wposn = \varnothing$
$bposn \cup wposn \neq Posn$



Board Example: A Dynamic View



The Player Class

Player

Move

$p! : Posn$

The Game

$TicTacToe$

$bplayer, wplayer : Player$
$board : Board$

$\textsc{Init}$
$board.\textsc{Init}$

$BlackMove \mathrel{\widehat=} bplayer.Move \parallel board.BlackMove$
$WhiteMove \mathrel{\widehat=} wplayer.Move \parallel board.WhiteMove$
$TheWinner \mathrel{\widehat=} board.TheWinner$

The Game Communication

	$bplayer$	$board$	$wplayer$
$BlackMove$	$p! : Posn$	$p? : Posn$	
$WhiteMove$		$p? : Posn$	$p! : Posn$
$TheWinner$		$winner! : Colour$	

Operation Expressions

promotion

$$object.operation$$

indicates that the named object undergoes the named operation. This operation must be specified in the class of the object.

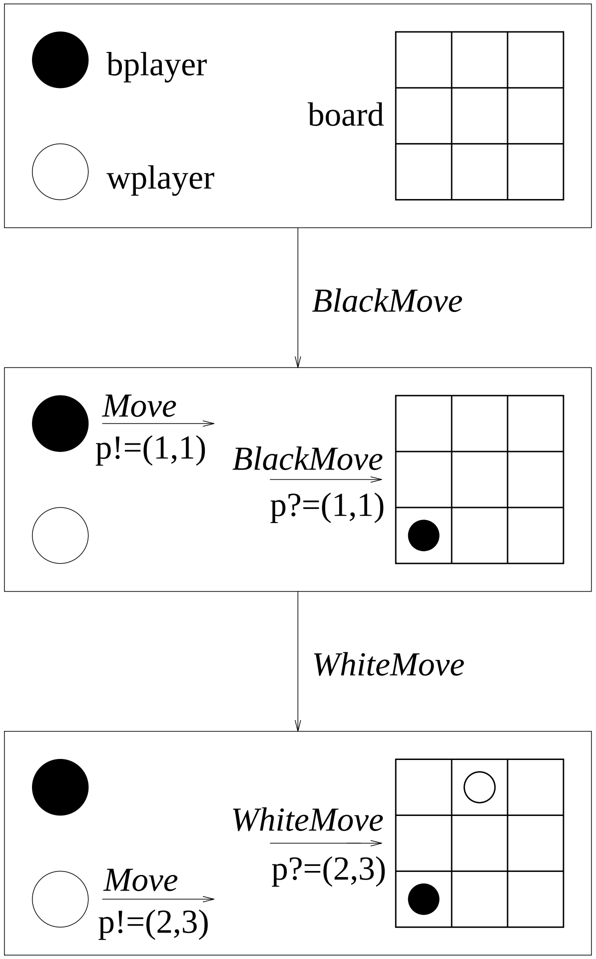
parallel operator

$$object_1.operation_1 \parallel object_2.operation_2$$

indicates that the objects synchronise, with output from the first component being identified with the same named input (apart from the ? and ! decorations) to the second component; this communication is hidden (internal).

All other input/output is with the environment.

Game Example: A Dynamic View



Extending Player by Inheritance

$Player_+$

$Player$

$board : Board$

$Move$

$p! \notin board.bposn \cup board.wposn$

which expands to

$Player_+$

$board : Board$

$Move$

$p! : Posn$

$p! \notin board.bposn \cup board.wposn$

The Game Revisited

TicTacToe

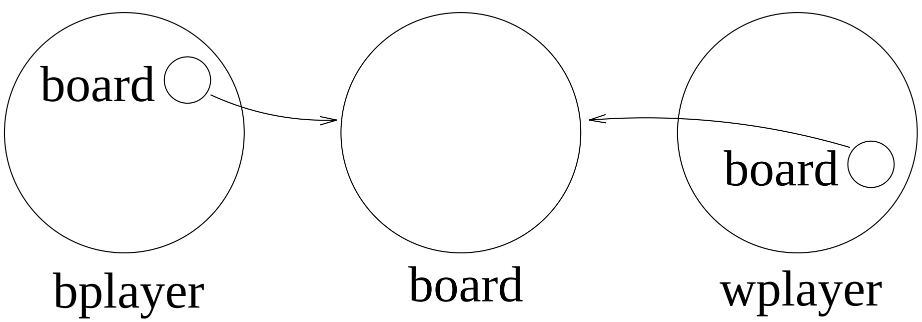
$$
\begin{array}{l}
bplayer, wplayer : Player_{+} \\
board : Board \\
\hline
bplayer.board = board = wplayer.board
\end{array}
$$

INIT

$board.\textsc{Init}$

$BlackMove \mathrel{\widehat=} bplayer.Move \parallel board.BlackMove$
$WhiteMove \mathrel{\widehat=} wplayer.Move \parallel board.WhiteMove$
$TheWinner \mathrel{\widehat=} board.TheWinner$

The Object Structure



Specializing Player by Inheritance

Bplayer

$$\mathit{Player}_{+}$$

Move

$$\neg\ 3\mathit{InRow}(\mathit{board}.\mathit{wposn})$$
$$\mathit{board}.\mathit{turn} = \mathit{black}$$

Wplayer

$$\mathit{Player}_{+}$$

Move

$$\neg\ 3\mathit{InRow}(\mathit{board}.\mathit{bposn})$$
$$\mathit{board}.\mathit{turn} = \mathit{white}$$

e.g. *Wplayer* expands to

Wplayer

$$\mathit{board} : \mathit{Board}$$

Move

$$p! : \mathit{Posn}$$

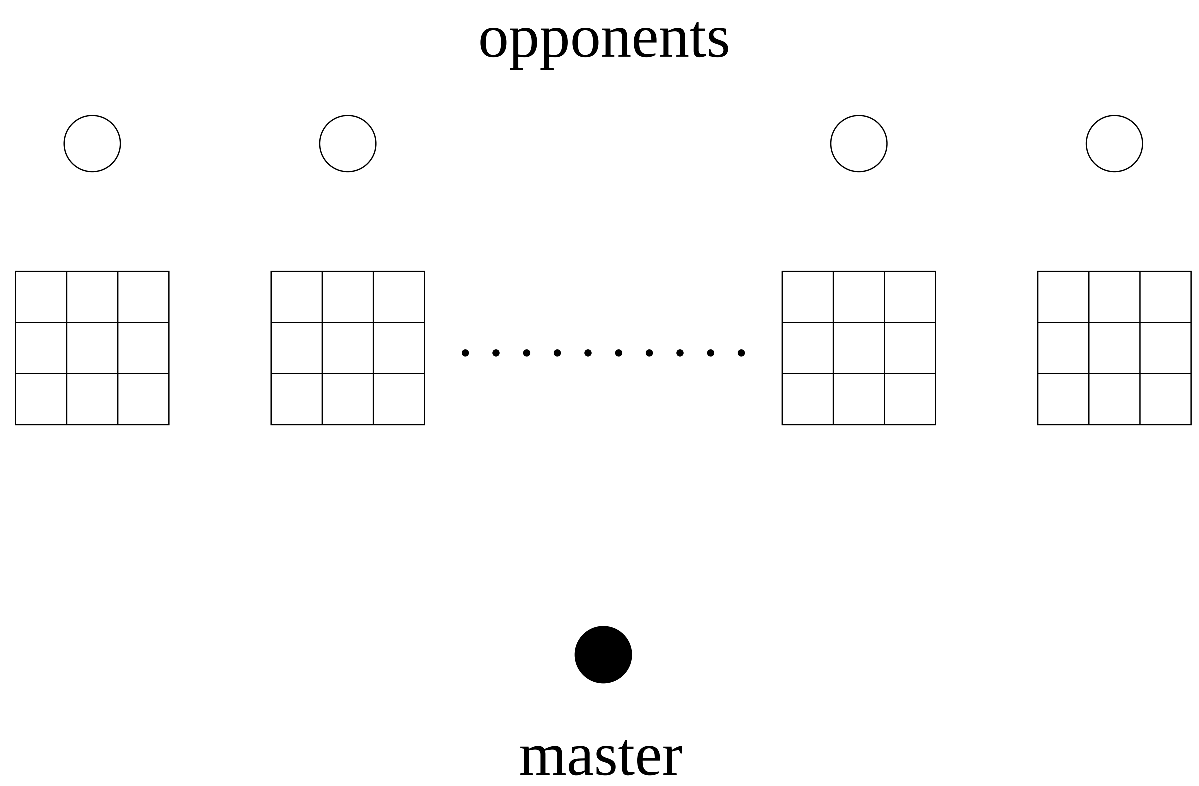
$$\neg\ 3\mathit{InRow}(\mathit{board}.\mathit{bposn})$$
$$p! \notin \mathit{board}.\mathit{bposn} \cup \mathit{board}.\mathit{wposn}$$
$$\mathit{board}.\mathit{turn} = \mathit{white}$$

The Game Again

$$
\begin{array}{l}
\textit{TicTacToe} \\
\hline
\begin{array}{|l}
bplayer : Bplayer \\
wplayer : Wplayer \\
board : Board \\
\hline
bplayer.board = board \\
wplayer.board = board
\end{array}
\qquad
\begin{array}{|l}
\textsc{Init} \\
\hline
board.\textsc{Init}
\end{array} \\
BlackMove \mathrel{\widehat{=}} bplayer.Move \parallel board.BlackMove \\
WhiteMove \mathrel{\widehat{=}} wplayer.Move \parallel board.WhiteMove \\
TheWinner \mathrel{\widehat{=}} board.TheWinner
\end{array}
$$

A Tic Tac Toe Tournament

A tic tac toe master is prepared to play any number of opponents concurrently.



An Opponent

$$
\begin{array}{l}
\textbf{Opponent} \\
\hline
Wplayer \\
\quad \begin{array}{|l}
\textit{Move} \\
\hline
b! : Board \\
\hline
b! = board
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\textbf{Opponent} \\
\hline
\quad \begin{array}{|l}
\hline
board : Board \\
\hline
\end{array} \\
\quad \begin{array}{|l}
\textit{Move} \\
\hline
p! : Posn \\
b! : Board \\
\hline
\neg\ 3InRow(board.bposn) \\
p! \notin board.bposn \cup board.wposn \\
board.turn = white \\
b! = board
\end{array}
\end{array}
$$

The Master

$$
\begin{array}{l}
\textbf{schema } \mathit{Master} \\
\quad \begin{array}{|l}
\hline
\mathit{boards} : \mathbb{P}\, \mathit{Board} \\
\hline
\end{array} \\[1em]
\quad \textbf{schema } \mathit{Move} \\
\quad \begin{array}{|l}
p! : \mathit{Posn} \\
b! : \mathit{Board} \\
\hline
\neg\, 3\mathit{InRow}(b!.\mathit{wposn}) \\
p! \notin b!.\mathit{bposn} \cup b!.\mathit{wposn} \\
b!.\mathit{turn} = \mathit{black} \\
b! \in \mathit{boards} \\
\hline
\end{array}
\end{array}
$$

The Tournament

Tournament

$$
\begin{array}{l}
master : Master \\
opponents : \mathbb{P}\, Opponent \\
boards : \mathbb{P}\, Board \\
\hline
master.boards = boards \\
\{op : opponents \bullet op.board\} = boards
\end{array}
$$

INIT

$$\forall\, b : boards \bullet b.\mathrm{INIT}$$

$$
\begin{array}{l}
MasterMove \mathrel{\widehat=} master.move \parallel ([b? : boards] \bullet b?.BlackMove) \\
OpponentMove \mathrel{\widehat=} [op : opponents] \bullet op.move \\
\qquad\qquad\qquad\qquad \parallel \\
\qquad\qquad\qquad\qquad [b? : boards] \bullet b?.WhiteMove \\
Awinner \mathrel{\widehat=} [b! : boards] \bullet b!.TheWinner
\end{array}
$$

	$master$	$boards$	$opponents$
$MasterMove$	$p! : Posn$ $b! : Board$	$p? : Posn$ $b? : Board$	
$OpponentMove$		$p? : Posn$ $b? : Board$	$p! : Posn$ $b! : Board$
$Awinner$		$b! : Board$ $winner! : Colour$	

Notation

Set Abstraction

$$\{a : A \bullet f(a)\}$$

is the set of all elements $f(a)$ where $a \in A$. e.g.

$$\{n : \mathbb{N} \bullet 2n\}$$

is the set of even natural numbers.

$$\{op : opponents \bullet op.board\}$$

is the set of boards associated with the set of opponents.

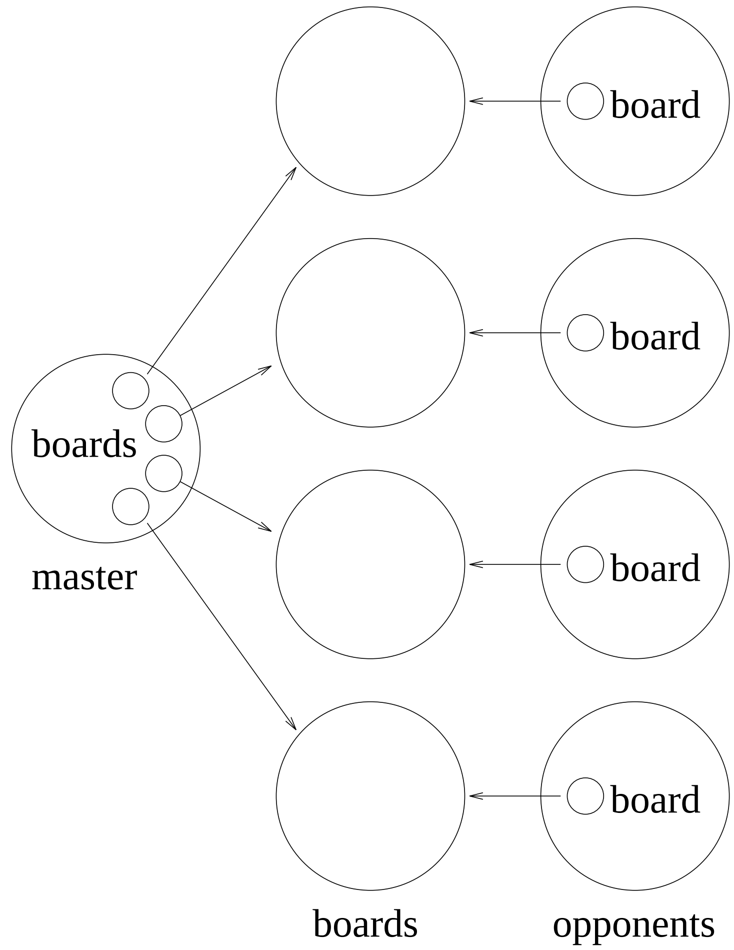
Object Selection

$$[ob : \text{set of objects}] \bullet ob.operation$$

indicates that object ob is selected from the given set of objects and performs $operation$.

Object selection may be via input/output communication.

The Object Structure



An Alternative Informal View

- there are two players and nine marbles
- each marble is uniquely labelled with a number between 1 and 9
- initially the players have no marbles
- the players take it in turns to select a marble from those not already selected
- the first player to have three marbles whose labels add to 15 is the winner

Initial Abstractions

$$Posn == 1 \mathrel{..} 9$$

The Abstraction

6	1	8
7	5	3
2	9	4

Three in a Row

$$\begin{array}{|l}
3InRow : \mathbb{P}\, Posn \rightarrow \mathbb{B} \\
\hline
\forall\, ps : \mathbb{P}\, Posn \bullet \\
\quad 3InRow(ps) \Leftrightarrow \\
\qquad \exists\, p, q, r : ps \bullet \\
\qquad\quad \#\{p, q, r\} = 3 \\
\qquad\quad p + q + r = 15
\end{array}$$