COMP3610/6361
Principles of Programming Languages

Peter Höfner

Aug 22, 2023

Section 11

(Imperative) Objects

Case Study

Motivation

- use our language with subtyping, records and references to model key keatures of OO programming
- encode/approximate concepts into our language
- OO concepts
  - *multiple representations* (object carry their methods)
    (in contrast to abstract data types (ADTs)
  - *encapsulation*
  - *subtyping*
    interface is the set of names and types of its operations
  - *inheritance* share common parts (class and subclasses)
    some languages use *delegations* (e.g. C♯), which combine classes and objects
  - *open recursion* (`self` or `this`)

(Simple) Objects

- data structure encapsulating some internal state
- access via methods
- internal state typically a number of mutable instance variables (or fields)
- attention lies on *building*, rather than usage

Reminder

Scope Restriction

$$E ::= \ldots \mid \textbf{let val } x : T = E_1 \textbf{ in } E_2 \textbf{ end}$$

- x is a binder for E_2
- can be seen as syntactic sugar:

$$\textbf{let val } x : T = E_1 \textbf{ in } E_2 \textbf{ end} \equiv (\textbf{fn } x : T \Rightarrow E_2)\ E_1$$

Objects – Example

A Counter Object

let val $c : \{\texttt{get} : \text{unit} \to \text{int}, \texttt{inc} : \text{unit} \to \text{unit}\} =$

  let val $x : \text{int ref} = \text{ref } 0$ **in**

    $\{\texttt{get} = (\textbf{fn}\ y : \text{unit} \Rightarrow !x),$

     $\texttt{inc} = (\textbf{fn}\ y : \text{unit} \Rightarrow x := !x + 1)\}$

  end

in

  $(\#\texttt{inc}\ c)() \ ; \ (\#\texttt{get}\ c)()$

end

$\texttt{Counter} \ = \ \{\texttt{get} : \text{unit} \to \text{int}, \texttt{inc} : \text{unit} \to \text{unit}\}$

Objects – Example

Subtyping I

let val $c : \{\texttt{get} : \text{unit} \to \text{int},\ \texttt{inc} : \text{unit} \to \text{unit}\,,\texttt{reset} : \text{unit} \to \text{unit}\} =$

let val $x : \text{int ref} = \text{ref}\ 0$ **in**

$\{\texttt{get} = (\textbf{fn}\ y : \text{unit} \Rightarrow !x),$

$\texttt{inc} = (\textbf{fn}\ y : \text{unit} \Rightarrow x := !x + 1)\}$

$\texttt{reset} = (\textbf{fn}\ y : \text{unit} \Rightarrow x := 0)\}$

end

in

$(\#\texttt{inc}\ c)()\ ;\ (\#\texttt{get}\ c)()$

end

$\texttt{ResCounter} \ = \ \{\texttt{get} : \text{unit} \to \text{int},\ \texttt{inc} : \text{unit} \to \text{unit}, \texttt{reset} : \text{unit} \to \text{unit}\}$

Objects – Example

Subtyping II

$$\texttt{ResCounter} <: \texttt{Counter}$$

Objects – Example

Object Generators

let val $\texttt{newCounter}$: unit $\rightarrow$ $\texttt{Counter}$ $=$

$(\textbf{fn}\ y$: unit $\Rightarrow$

let val x : int ref $=$ ref 0 **in**

$\{\texttt{get} = (\textbf{fn}\ y$: unit $\Rightarrow\ !x),$

$\texttt{inc} = (\textbf{fn}\ y$: unit $\Rightarrow\ x := !x + 1)\}$

end$)$

in

$(\#\texttt{inc}\ (\texttt{newCounter}()))()$

end

$\texttt{newRCounter}$ defined in similar fashion

Simple Classes

- pull out common features
- ignore complex features
  such as *visibility annotations*, *static fields and methods*, *friend classes* . . .

- most primitive form, a class is a data structure that can
  – be instantiated to yields a fresh object, or – extended to yield another class

Reusing Method Code

$$\texttt{Counter} = \{\texttt{get} : \mathsf{unit} \to \mathsf{int},\ \texttt{inc} : \mathsf{unit} \to \mathsf{unit}\}$$
$$\texttt{CounterRep} = \{p : \mathsf{int\ ref}\}$$

(Simple) Classes

$$\textbf{let val } \texttt{CounterClass} : \texttt{CounterRep} \to \texttt{Counter} =$$
$$(\textbf{fn } x : \texttt{CounterRep} \Rightarrow$$
$$\{\texttt{get} = (\textbf{fn } y : \text{unit} \Rightarrow !(\#p\ x)),$$
$$\texttt{inc} = (\textbf{fn } y : \text{unit} \Rightarrow (\#p\ x) := !(\#p\ x) + 1)\})$$

$$\textbf{let val } \texttt{newCounter} : \text{unit} \to \texttt{Counter} =$$
$$(\textbf{fn } y : \text{unit} \Rightarrow$$
$$\textbf{let val } x : \texttt{CounterRep} = \{p = \text{ref } 0\} \textbf{ in}$$
$$\texttt{CounterClass}\ x$$
$$\textbf{end})$$

IMP vs. Java

```
class Counter
  {  protected int p;
     Counter() { this.p=0; }
     int get () { return this.p; }
     void inc () { this.p++ ; }
  };
```

(Simple) Classes

$$
\begin{array}{l}
(\textbf{fn}\ \texttt{ResCounterClass} : \texttt{CounterRep} \to \texttt{ResCounter} \Rightarrow \\
\quad (\textbf{fn}\ x : \texttt{CounterRep} \Rightarrow \\
\quad\quad \textbf{let val}\ \texttt{super} : \texttt{Counter} = \texttt{CounterClass}\ x\ \textbf{in} \\
\quad\quad\quad \{\texttt{get} = \#\texttt{get}\ \texttt{super}, \\
\quad\quad\quad\ \texttt{inc} = \#\texttt{inc}\ \texttt{super}, \\
\quad\quad\quad\ \texttt{reset} = (\textbf{fn}\ y : \mathsf{unit} \Rightarrow (\#p\ x) := 0)\} \\
\quad\quad \textbf{end}))
\end{array}
$$

$$
\begin{array}{rl}
\texttt{CounterRep} & = \{p : \mathsf{int\ ref}\} \\
\texttt{Counter} & = \{\texttt{get} : \mathsf{unit} \to \mathsf{int},\ \texttt{inc} : \mathsf{unit} \to \mathsf{unit}\} \\
\texttt{ResCounter} & = \{\texttt{get} : \mathsf{unit} \to \mathsf{int},\ \texttt{inc} : \mathsf{unit} \to \mathsf{unit},\ \texttt{reset} : \mathsf{unit} \to \mathsf{unit}\}
\end{array}
$$

IMP vs. Java

```
class ResetCounter
  extends Counter
  {  void reset () {this.p=0;}
  };
```

(Simple) Classes

$$\texttt{BuCounter} = \{\texttt{get} : \mathsf{unit} \to \mathsf{int},\ \texttt{inc} : \mathsf{unit} \to \mathsf{unit},\ \texttt{reset} : \mathsf{unit} \to \mathsf{unit},\ \texttt{backup} : \mathsf{unit} \to \mathsf{unit}\}$$

$$\texttt{BuCounterRep} = \{p : \mathsf{int\ ref},\ b : \mathsf{int\ ref}\}$$

$$\begin{array}{l}
\textbf{let val}\ \texttt{BuCounterClass} : \texttt{BuCounterRep} \to \texttt{BuCounter} = \\
\quad (\textbf{fn}\ x : \texttt{BuCounterRep} \Rightarrow \\
\quad\quad \textbf{let val}\ \texttt{super} : \texttt{ResCounter} = \texttt{ResCounterClass}\ x\ \textbf{in} \\
\quad\quad\quad \{\texttt{get} = \#\texttt{get super},\ \texttt{inc} = \#\texttt{inc super}, \\
\quad\quad\quad\ \texttt{reset} = (\textbf{fn}\ y : \mathsf{unit} \Rightarrow (\#p\ x) := !(\#b\ x))\} \\
\quad\quad\quad\ \texttt{backup} = (\textbf{fn}\ y : \mathsf{unit} \Rightarrow (\#b\ x) := !(\#p\ x))\} \\
\quad\quad \textbf{end})
\end{array}$$