Last update: 5 April 2005

Software Architecture

Bertrand Meyer

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Reading assignment for next week

- OOSC2:
  - Chapter 3: Modularity
  - Chapter 6: Abstract Data Types

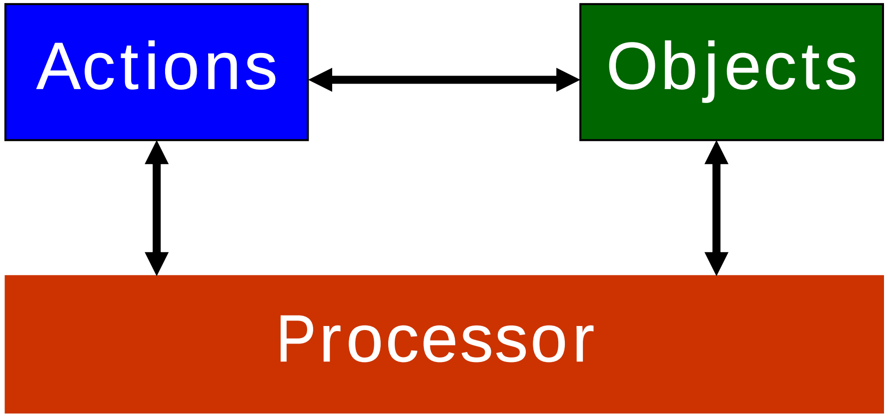
Lecture 3: Abstract Data Types

Abstract Data Types (ADT)

- Why use the objects?
- The need for data abstraction
- Moving away from the physical representation
- Abstract data type specifications
- Applications to software design

The first step

- A system performs certain actions on certain data.
- Basic duality:
  - Functions [or: Operations, Actions]
  - Objects [or: Data]



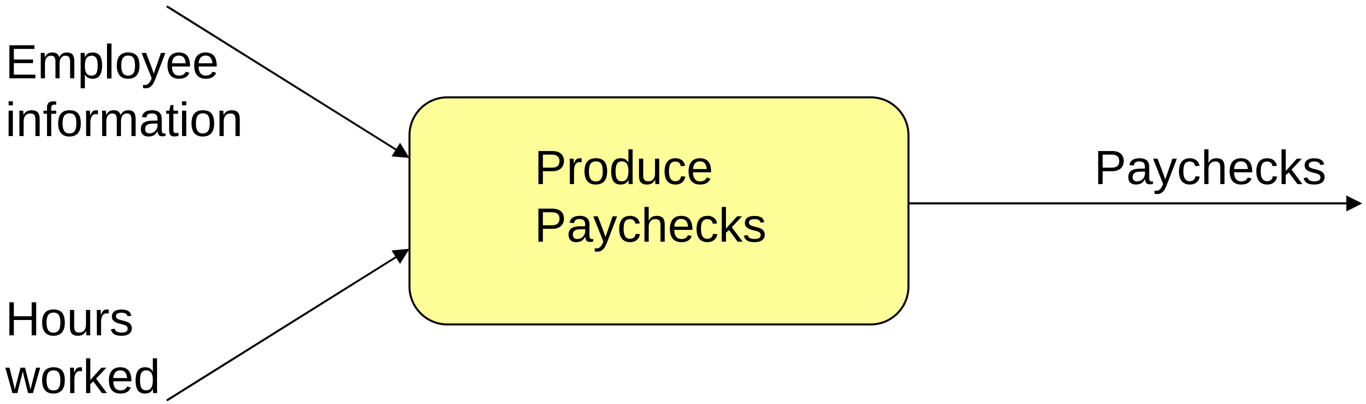
Finding the structure

- The structure of the system may be deduced from an analysis of the functions (1) or the objects (2).

- Resulting analysis and design method:
  - Process-based decomposition: classical (routines)
  - Object-oriented decomposition

Arguments for using objects

- Reusability: Need to reuse whole data structures, not just operations
- Extendibility, Continuity: Objects remain more stable over time.



Object technology: A first definition

- Object-oriented software construction is the approach to system structuring that bases the architecture of software systems on the types of objects they manipulate — not on “the” function they achieve.

The O-O designer’s motto

- Ask NOT first WHAT the system does:

    Ask WHAT it does it TO!

Issues of object-oriented design

- How to find the object types.
- How to describe the object types.
- How to describe the relations and commonalities between object types.
- How to use object types to structure programs.

Description of objects

- Consider not a single object but a type of objects with similar properties.

- Define each type of objects not by the objects' physical representation but by their behavior: the services (FEATURES) they offer to the rest of the world.

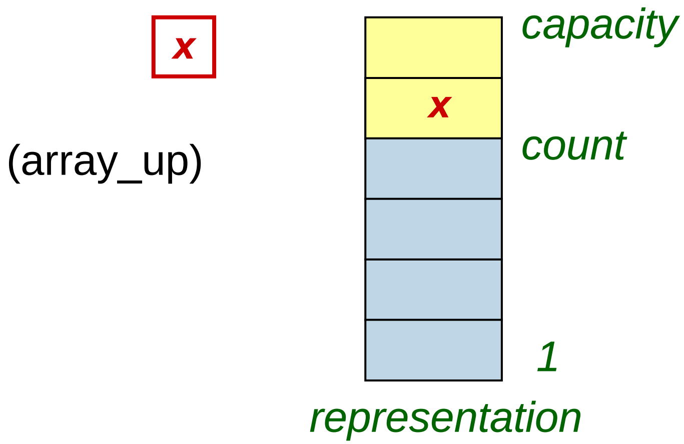
- External, not internal view: ABSTRACT DATA TYPES

The theoretical basis

- The main issue: How to describe program objects (data structures):
  - Completely
  - Unambiguously
  - Without overspecifying?

    (Remember information hiding)

A stack, concrete object

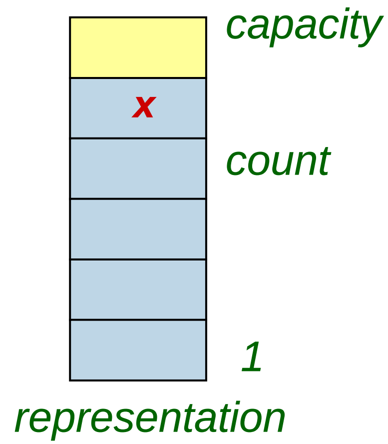


"Push" *x* on stack *representation*:

count := *count* + 1
representation [*count*] := *x*

A stack, concrete object

(array_up)

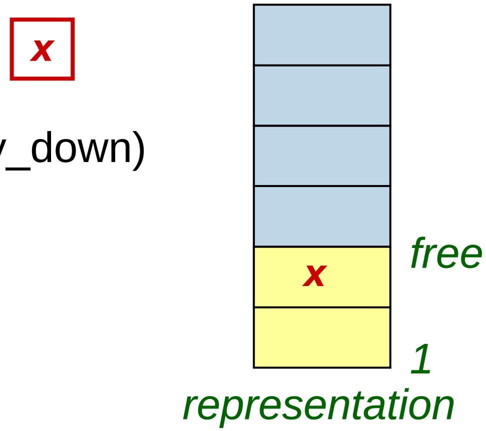


"Push" *x* on stack *representation*:

count := *count* + 1

representation [*count*] := *x*

(array_down)



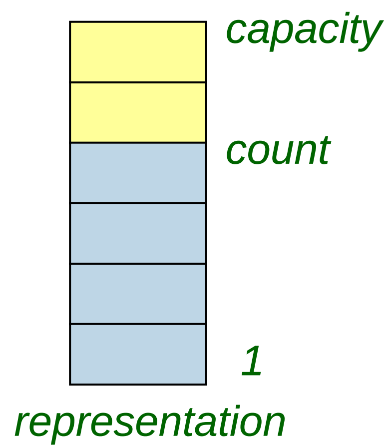
"Push" *x* on stack *representation*:

representation [*free*] := *x*

free := *free* - 1

A stack, concrete object

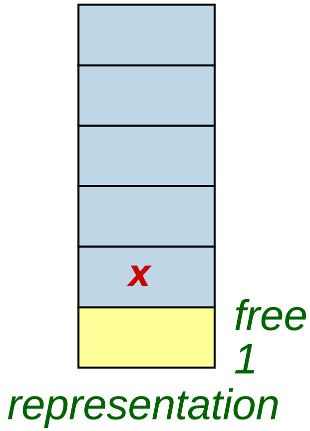
(array_up)



"Push" *x* on stack *representation*:

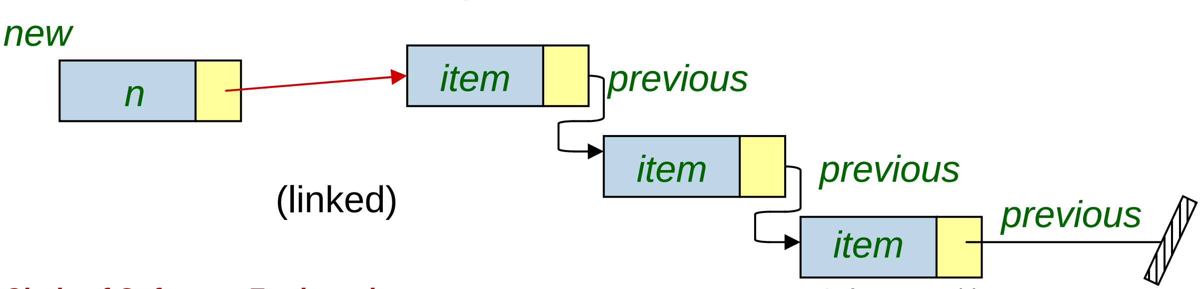
representation [*count*] := *x*
count := *count* + 1

(array_down)



"Push" *x* on stack *representation*:

representation [*free*] := *x*
free := *free* - 1



(linked)

"Push" operation:

new (*n*)
n.*item* := *x*
n.*previous* := *head*
head := *n*

Stack: An Abstract Data Type (ADT)

- Types:

  STACK [*G*]

  -- *G*: Formal generic parameter

- Functions (Operations):

  put: $STACK\ [G] \times G \rightarrow STACK\ [G]$

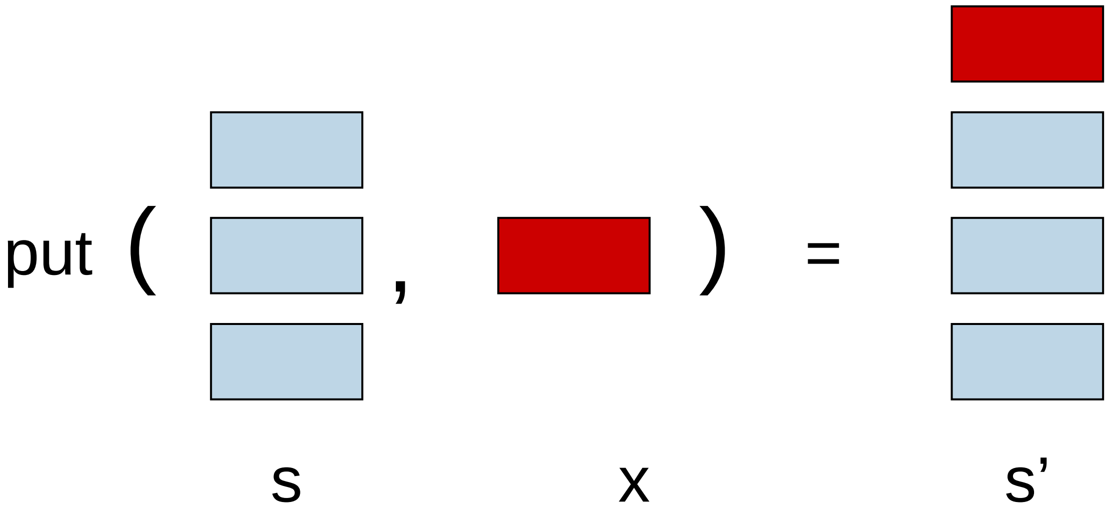
  $remove$: $STACK\ [G] \nrightarrow STACK\ [G]$

  $item$: $STACK\ [G] \nrightarrow G$

  $empty$: $STACK\ [G] \rightarrow BOOLEAN$

  new: $STACK\ [G]$

Using functions to model operations



Reminder: Partial functions

- A partial function, identified here by $\nrightarrow$, is a function that may not be defined for all possible arguments.

- Example from elementary mathematics:
  - inverse: $\Re \nrightarrow \Re$, such that

$$\mathit{inverse}\ (x) = 1 / x$$

The *STACK* ADT (continued)

- Preconditions:

  remove (*s*: *STACK* [*G*]) **require not** *empty* (*s*)

  item (*s*: *STACK* [*G*]) **require not** *empty* (*s*)

- Axioms: For all *x*: *G*, *s*: *STACK* [*G*]

  item (*put* (*s*, *x*)) = *x*

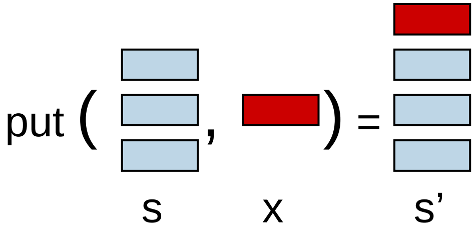
  remove (*put* (*s*, *x*)) = *s*

  empty (*new*)

  (or: *empty* (*new*) = **True**)

  not *empty* (*put* (*s*, *x*))

  (or: *empty* (*put* (*s*, *x*)) = **False**)



Exercises

- Adapt the preceding specification of stacks (LIFO, Last-In First-Out) to describe queues instead (FIFO).

- Adapt the preceding specification of stacks to account for bounded stacks, of maximum size capacity.
  - Hint: *put* becomes a partial function.

Formal stack expressions

value = *item* (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

Expressed differently

value = *item* (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

- *s1* = *new*
- *s2* = *put* (*put* (*put* (*s1*, *x8*), *x7*), *x6*)
- *s3* = *remove* (*s2*)
- *s4* = *new*
- *s5* = *put* (*put* (*s4*, *x5*), *x4*)
- *s6* = *remove* (*s5*)
- *y1* = *item* (*s6*)
- *s7* = *put* (*s3*, *y1*)
- *s8* = *put* (*s7*, *x2*)
- *s9* = *remove* (*s8*)
- *s10* = *put* (*s9*, *x1*)
- *s11* = *remove* (*s10*)
- *value* = *item* (*s11*)

Expression reduction

Stack 1

```
value = item (
            remove (
                 put (
                    remove (
                         put (
                            put (
                               remove (
                                    put (put (put (new, x8), x7), x6)
                                    )
                               , item (
                                    remove (
                                         put (put (new, x5), x4)
                                         )
                                    )
                               )
                            , x2)
                         )
                    , x1)
                 )
            )
```

Expression reduction

```
value = item (
             remove (
                    put (
                        remove (
                               put (
                                   put (
                                       remove (
                                              put (put (put (new, x8), x7), x6)
                                              )
                                       , item (
                                              remove (
                                                     put (put (new, x5), x4)
                                                     )
                                              )
                                       )
                                   , x2)
                               )
                        , x1)
                    )
             )
```

x8

Stack 1

Expression reduction

x7
x8

Stack 1

```
value = item (
              remove (
                      put (
                           remove (
                                   put (
                                        put (
                                             remove (
                                                     put (put (put (new, x8), x7), x6)
                                                     )
                                             , item (
                                                     remove (
                                                             put (put (new, x5), x4)
                                                             )
                                                     )
                                             )
                                        , x2)
                                   )
                           , x1)
                      )
              )
```

Expression reduction

x6
x7
x8

Stack 1

```
value = item (
            remove (
                put (
                    remove (
                        put (
                            put (
                                remove (
                                    put (put (put (new, x8), x7), x6)
                                    )
                                , item (
                                    remove (
                                        put (put (new, x5), x4)
                                        )
                                    )
                                )
                            , x2)
                        )
                    , x1)
                )
            )
```

Expression reduction

x6
x7
x8

Stack 1

```
value = item (
              remove (
                     put (
                         remove (
                                put (
                                    put (
                                        remove (
                                               put (put (put (new, x8), x7), x6)
                                               )
                                        , item (
                                               remove (
                                                      put (put (new, x5), x4)
                                                      )
                                               )
                                        )
                                    , x2)
                                )
                         , x1)
                     )
              )
```

Expression reduction

x7
x8

Stack 1 Stack 2

```
value = item (
            remove (
                  put (
                      remove (
                            put (
                               put (
                                  remove (
                                        put (put (put (new, x8), x7), x6)
                                        )
                                  , item (
                                        remove (
                                              put (put (new, x5), x4)
                                              )
                                        )
                                  )
                               , x2)
                            )
                      , x1)
                  )
            )
```

Expression reduction

Stack 1	Stack 2
x7	
x8	x5

```
value = item (
              remove (
                      put (
                           remove (
                                   put (
                                        put (
                                             remove (
                                                     put (put (put (new, x8), x7), x6)
                                                     )
                                             , item (
                                                     remove (
                                                             put (put (new, x5), x4)
                                                             )
                                                     )
                                             )
                                        , x2)
                                   )
                           , x1)
                      )
              )
```

Expression reduction

Stack 1	Stack 2
x7	x4
x8	x5

```
value = item (
            remove (
                  put (
                      remove (
                            put (
                               put (
                                  remove (
                                        put (put (put (new, x8), x7), x6)
                                        )
                                  , item (
                                        remove (
                                              put (put (new, x5), x4)
                                              )
                                        )
                                  )
                               , x2)
                            )
                      , x1)
                  )
            )
```

Expression reduction

Stack 1
x7
x8

Stack 2
x4
x5

```
value = item (
            remove (
                  put (
                      remove (
                            put (
                               put (
                                  remove (
                                        put (put (put (new, x8), x7), x6)
                                        )
                                  , item (
                                       remove (
                                              put (put (new, x5), x4)
                                             )
                                       )
                                  )
                               , x2)
                            )
                      , x1)
                  )
            )
```

Expression reduction

Stack 1	Stack 2
x7	
x8	x5

```
value = item (
            remove (
                  put (
                      remove (
                            put (
                                put (
                                    remove (
                                          put (put (put (new, x8), x7), x6)
                                          )
                                    , item (
                                          remove (
                                                put (put (new, x5), x4)
                                                )
                                          )
                                    )
                                , x2)
                            )
                      , x1)
                  )
            )
```

Expression reduction

Stack 1	Stack 2
x5	
x7	
x8	x5

```
value = item (
          remove (
               put (
                  remove (
                       put (
                         put (
                            remove (
                                 put (put (put (new, x8), x7), x6)
                                 )
                            , item (
                                remove (
                                     put (put (new, x5), x4)
                                     )
                                )
                            )
                         , x2)
                       )
                  , x1)
               )
          )
```

Expression reduction

Stack 1	Stack 2
x2	
x5	
x7	
x8	

```
value = item (
            remove (
                  put (
                        remove (
                              put (
                                    put (
                                          remove (
                                                put (put (put (new, x8), x7), x6)
                                                )
                                          , item (
                                                remove (
                                                      put (put (new, x5), x4)
                                                      )
                                                )
                                          )
                                    , x2)
                              )
                        , x1)
                  )
            )
```

Expression reduction

Stack 1	Stack 2
x2	
x5	
x7	
x8	

```
value = item (
            remove (
                put (
                    remove (
                        put (
                            put (
                                remove (
                                    put (put (put (new, x8), x7), x6)
                                    )
                                , item (
                                    remove (
                                        put (put (new, x5), x4)
                                        )
                                    )
                                )
                            , x2)
                        )
                , x1)
                )
            )
```

Expression reduction

Stack 1	Stack 2
x1	
x5	
x7	
x8	

```
value = item (
              remove (
                      put (
                          remove (
                                  put (
                                       put (
                                            remove (
                                                    put (put (put (new, x8), x7), x6)
                                                    )
                                            , item (
                                                    remove (
                                                            put (put (new, x5), x4)
                                                            )
                                                    )
                                            )
                                       , x2)
                                  )
                          , x1)
                      )
              )
```

Expression reduction

Stack 1	Stack 2
x1	
x5	
x7	
x8	

```
value = item (
               remove (
                      put (
                          remove (
                                  put (
                                      put (
                                          remove (
                                                  put (put (put (new, x8), x7), x6)
                                                  )
                                          , item (
                                                 remove (
                                                        put (put (new, x5), x4)
                                                        )
                                                 )
                                          )
                                      , x2)
                                  )
                          , x1)
                      )
              )
```

Expression reduction

Stack 1	Stack 2
x5	
x7	
x8	

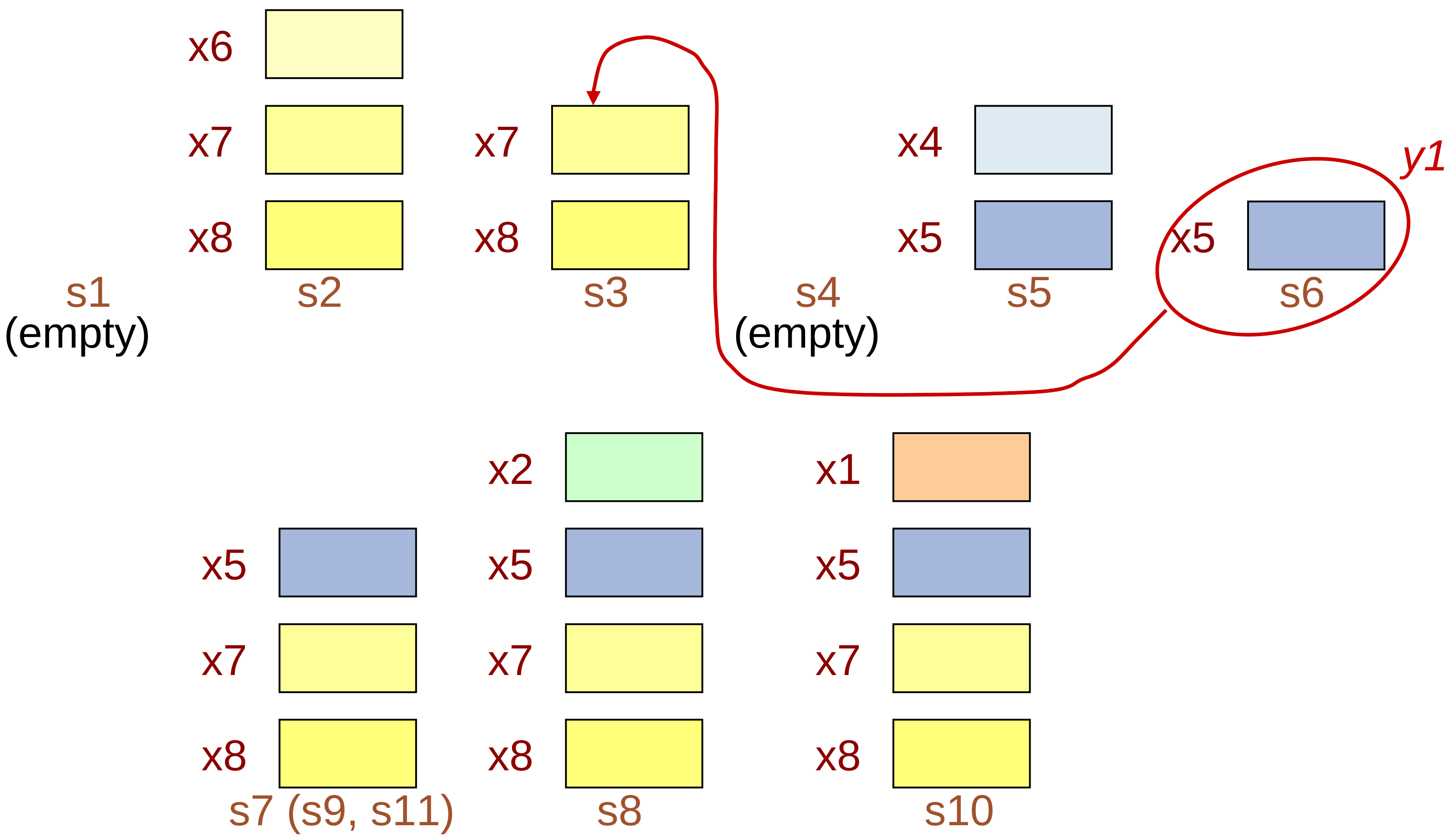
```
value = item (
            remove (
                put (
                    remove (
                        put (
                            put (
                                remove (
                                    put (put (put (new, x8), x7), x6)
                                    )
                                , item (
                                    remove (
                                        put (put (new, x5), x4)
                                        )
                                    )
                                )
                            , x2)
                        )
                    , x1)
                )
            )
```

Expressed differently

value = *item* (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

- *s1* = *new*
- *s2* = *put* (*put* (*put* (*s1*, *x8*), *x7*), *x6*)
- *s3* = *remove* (*s2*)
- *s4* = *new*
- *s5* = *put* (*put* (*s4*, *x5*), *x4*)
- *s6* = *remove* (*s5*)
- *y1* = *item* (*s6*)
- *s7* = *put* (*s3*, *y1*)
- *s8* = *put* (*s7*, *x2*)
- *s9* = *remove* (*s8*)
- *s10* = *put* (*s9*, *x1*)
- *s11* = *remove* (*s10*)
- *value* = *item* (*s11*)

An operational view of the expression



value = *item* (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

Properties of an ADT specification

- **Consistent**

  The axioms do not lead to contradiction

  "Only the truth"

- **Complete**

  The axioms cover all needed properties

  "All the truth"

Completeness

- Cannot be ascertained in general: it would have to be relative to some higher-level specification of the system, for which the same problem arises

- Useful notion in practice: sufficient completeness

Sufficient completess

An ADT specification for a type *T* is **sufficiently complete** if and only if any correct "query Expression" may be reduced, through application of the axioms, to a form not involving *T* .

"Correct" ADT expression

An expression of which we can prove that all arguments to ADT functions satisfy the precondition if any

put (*put* (*new*, *x5*), *x4*) -- Correct

remove (*put* (*put* (*new*, *x5*), *x4*)) -- Correct

item (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

-- Correct

remove (*new*) -- Not correct

Sufficient completeness

Three kinds of functions in specification of an ADT *T*:

- Creators:

  $OTHER \rightarrow T$ e.g. *new*

- Queries:

  $T \times \ldots \rightarrow OTHER$ e.g. *item*, *empty*

- Commands:

  $T \times \ldots \rightarrow T$ e.g. *put*, *remove*

Query expression

An expression in which the outermost function is a query

Examples (correct expressions):

item (*put* (*new*, *x5*))

item (*remove* (*put* (*put* (*new*, *x5*), *x4*))))

item (*remove* (*put* (*remove* (*put* (*put* (*remove* (*put* (*put* (*put* (*new*, *x8*), *x7*), *x6*)), *item* (*remove* (*put* (*put* (*new*, *x5*), *x4*)))), *x2*)), *x1*)))

But not:

new

put (*new*, *x5*)

remove (*put* (*new*, *x5*))

Sufficient completess

An ADT specification for a type *T* is **sufficiently complete** if and only if any correct "query Expression" may be reduced, through application of the axioms, to a form not involving *T* .

Exercise

Prove that the specification of stacks is sufficiently complete.

Stack: An Abstract Data Type

- Types:

  STACK [*G*]

  -- *G*: Formal generic parameter

- Functions (Operations):

  put: $STACK\ [G] \times G \rightarrow STACK\ [G]$

  remove: $STACK\ [G] \nrightarrow STACK\ [G]$

  item: $STACK\ [G] \nrightarrow G$

  empty: $STACK\ [G] \rightarrow BOOLEAN$

  new: $STACK\ [G]$

ADTs and software architecture

Abstract data types provide an ideal basis for modularizing software.

- Build each module as an *implementation* of an ADT:
  - Implements a set of *objects* with same *interface*
  - Interface is defined by a set of operations (the ADT's functions) constrained by abstract properties (its axioms and preconditions).
- The module consists of:
  - A *representation* for the ADT
  - An *implementation* for each of its operations
  - Possibly, auxiliary operations

Implementing an ADT

- Three components:

  (E1) The ADT's specification: functions, axioms, preconditions.

  (Example: stacks.)

  (E2) Some representation choice.

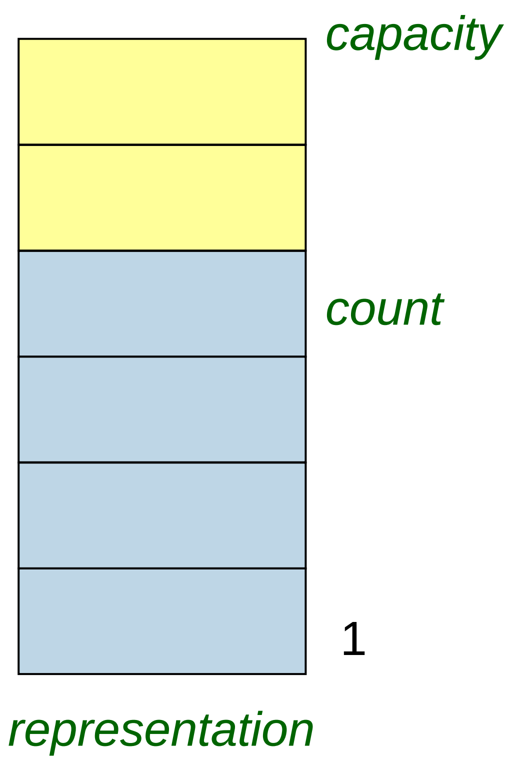
  (Example: <*representation*, *count*>.)

  (E3) A set of subprograms (routines) and attributes, each implementing one of the functions of the ADT specification (E1) in terms of the chosen representation (E2).

  (Example: routines *put*, *remove*, *item*, *empty*, *new*.)

A choice of stack representation

(array_up)



"Push" operation:

count := *count* + 1

representation [*count*] := *x*

Application to information hiding

Public part:
ADT specification (*E1*)

Secret part:

- **Choice of representation (*E2*)**
- **Implementation of functions by features (*E3*)**

Object technology: A first definition

- Object-oriented software construction is the approach to system structuring that bases the architecture of software systems on the types of objects they manipulate — not on “the” function they achieve.

Object technology: More precise definition

- Object-oriented software construction is the construction of software systems as structured collections of (possibly partial) abstract data type implementations.

Classes: The fundamental structure

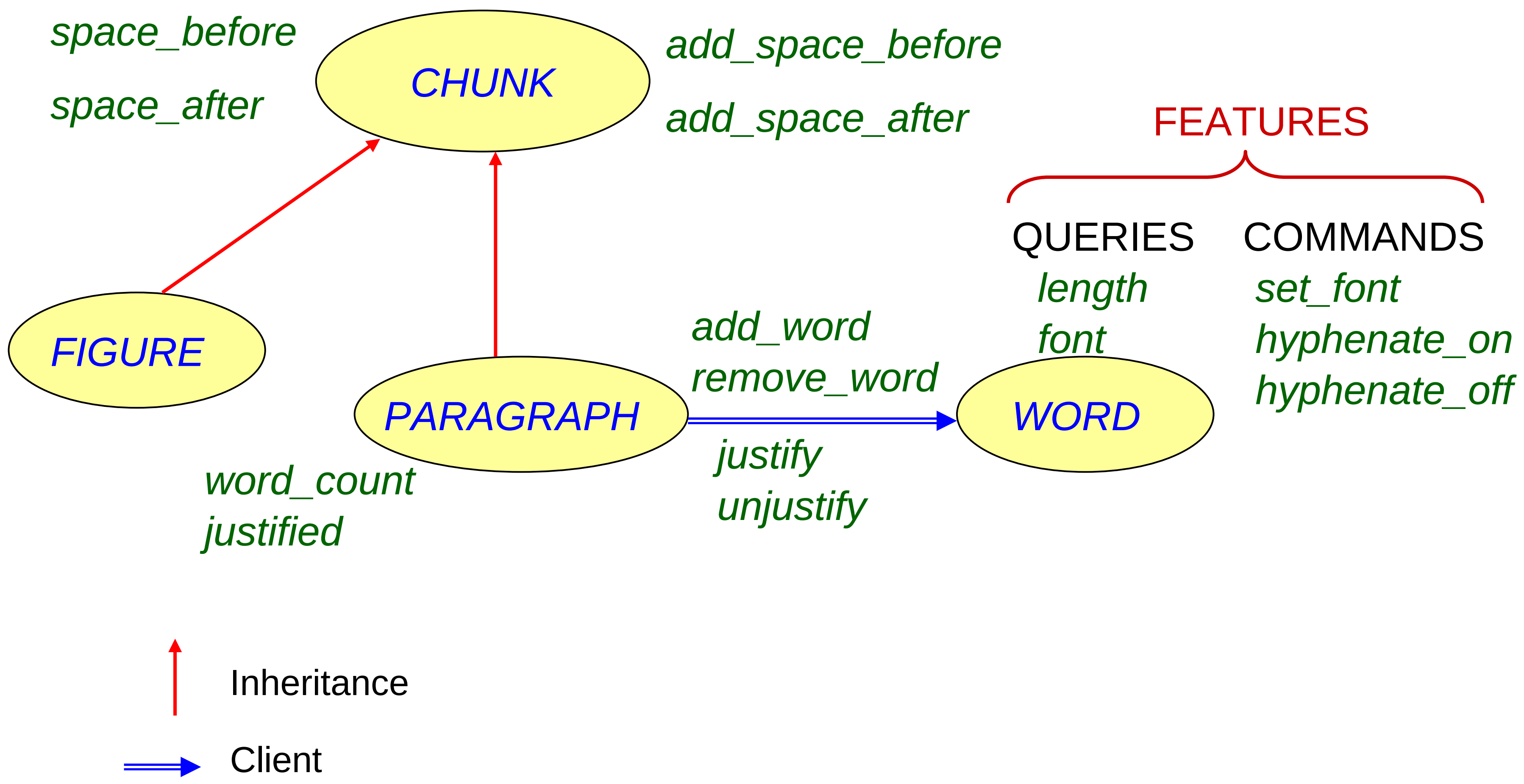
- Merging of the notions of module and type:
  - Module = Unit of decomposition: set of services
  - Type = Description of a set of run-time objects ("instances" of the type)

- The connection:
  - The services offered by the class, viewed as a module, are the operations available on the instances of the class, viewed as a type.

Class relations

- Two relations:
  - Client
  - Heir

Overall system structure



A very deferred class


```
deferred class
    COUNTER

feature
    item: INTEGER is
                    -- Counter value
        deferred
        end

    up is
                    -- Increase item by 1.
        deferred
        ensure
                    item = old item + 1
        end

    down is
                    -- Decrease item by 1.
        deferred
        ensure
                    item = old item – 1
        end

invariant
        item >= 0
end
```

End of lecture 3