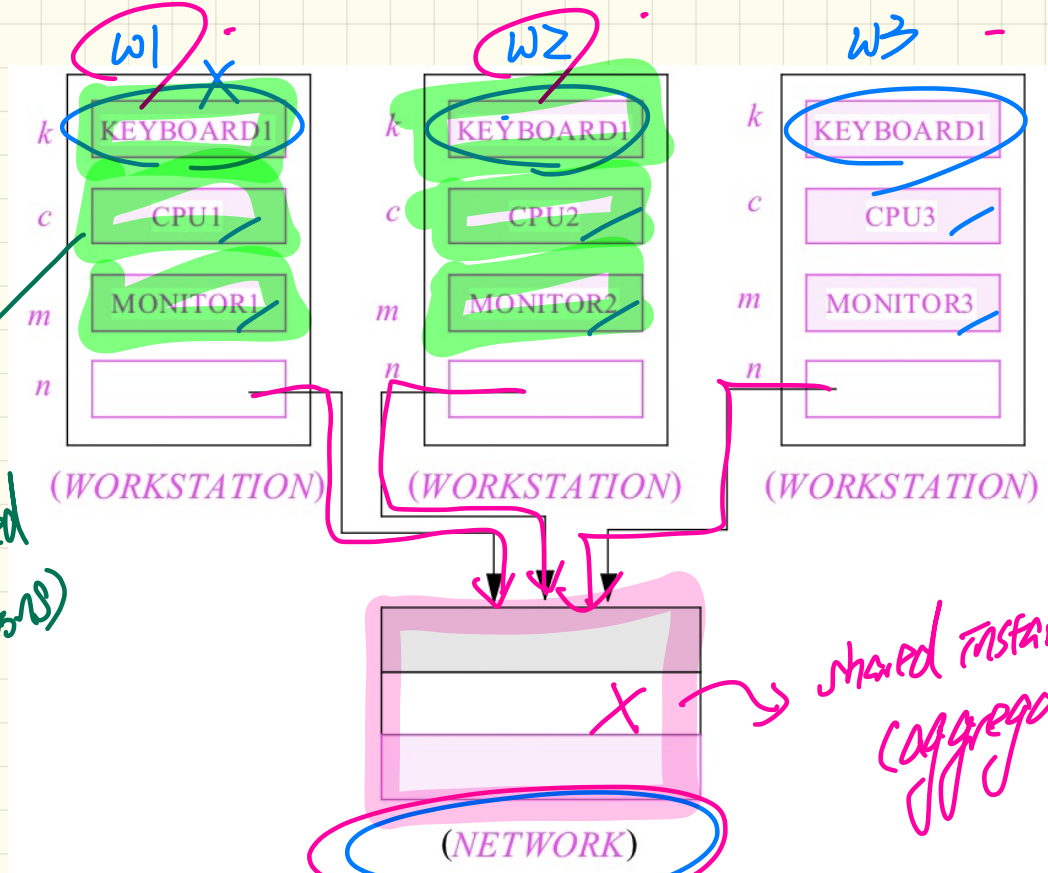


Lecture 5

Part 1

Expanded Types for Compositions

Modelling: Aggregation vs. Composition



Expanded Type for Composition

```
class KEYBOARD ... end class CPU ... end
class MONITOR ... end class NETWORK ... end
class WORKSTATION
  k: expanded KEYBOARD
  c: expanded CPU
  m: expanded MONITOR
  n: NETWORK
end
```

Use this if keyboard should always be an integral part.

Use this if the keyboard class may be either an integral or sharing part.

```
expanded class KEYBOARD ... end
expanded class CPU ... end
expanded class MONITOR ... end
class NETWORK ... end
class WORKSTATION
  k: KEYBOARD
  c: CPU
  m: MONITOR
  n: NETWORK
end
```

not storing address of a keyboard object

Use of Expanded Type

```

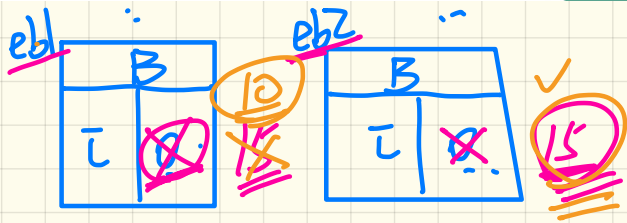
expanded class
  B
feature
  change_i (ni: INTEGER)
  do
    i := ni
  end
feature
  i: INTEGER
end
  
```

```

1 test_expanded
2 local
3   eb1, eb2: B
4 do
5   check eb1.i = 0 and eb2.i = 0 end ✓
6   check eb1 = eb2 end ✓
7   eb2.change_i (15)
8   check eb1.i = 0 and eb2.i = 15 end ✓
9   check eb1 /= eb2 end ✓
10  eb1 := eb2
11  check eb1.i = 15 and eb2.i = 15 end ✓
12  eb1.change_i (10)
13  check eb1.i = 10 and eb2.i = 15 end ✓
14  check eb1 /= eb2 end ✓
15 end
  
```

not comparing addresses
 eb1 ~ eb2

void object call?



does not copy address to result in aliasing
 simply copy over contents

EXERCISE

```
expanded class
  B
  feature
    change_i (ni: INTEGER)
      do
        i := ni
      end
  feature
    i: INTEGER
  end
```

```
1 test_expanded
2   local
3     eb1, eb2: B
4   do
5     check eb1.i = 0 and eb2.i = 0 end
6     check eb1 = eb2 end
7     eb2.change_i (15)
8     check eb1.i = 0 and eb2.i = 15 end
9     check eb1 /= eb2 end
10    eb1 := eb2
11    check eb1.i = 15 and eb2.i = 15 end
12    eb1.change_i (10)
13    check eb1.i = 10 and eb2.i = 15 end
14    check eb1 /= eb2 end
15  end
```

Reference Type or Expanded Type

```

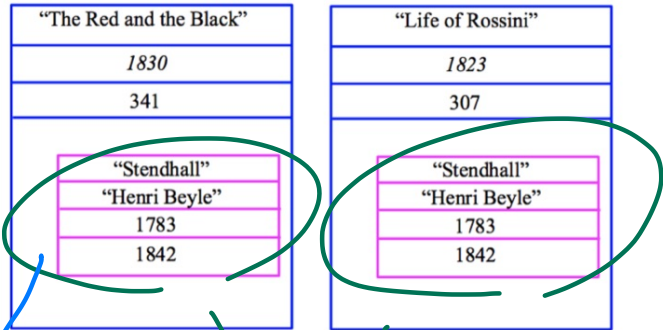
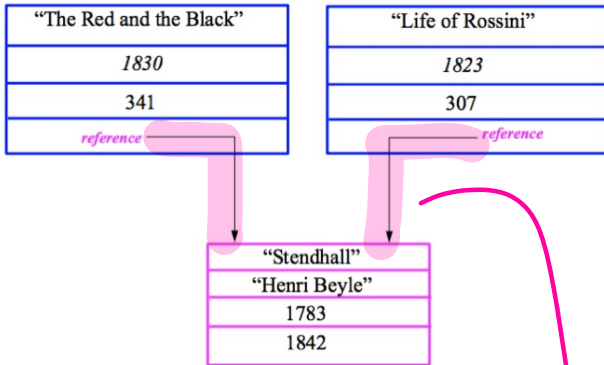
class AUTHOR
  →
end
  
```

```

class Book
  → author: AUTHOR
end
  
```

reference-typed author

expanded-typed author



aggregation (sharing)

```

class Book
  → author: expanded AUTHOR
end
  
```

composition (no sharing)

Lecture 5

Part 2

Sharing via Inheritance

Shared Data via Inheritance

Cohesion → cd ls pwd
Single Choice Principle

A class has a unifying theme of purpose (all features are relevant w.r.t. that purpose).

Descendant:

```
class DEPOSIT inherit SHARED_DATA
  -- 'maximum_balance' relevant
end

class WITHDRAW inherit SHARED_DATA
  -- 'minimum_balance' relevant
end

class INT_TRANSFER inherit SHARED_DATA
  -- 'exchange_rate' relevant
end

class ACCOUNT inherit SHARED_DATA
feature
  -- 'interest_rate' relevant
  deposits: DEPOSIT_LIST
  withdraws: WITHDRAW_LIST
end
```

Ancestor:

```
class SHARED_DATA
feature
  interest_rate: REAL
  exchange_rate: REAL
  minimum_balance: INTEGER
  maximum_balance: INTEGER
  ...
end
```

inherited to DEPOSIT but irrelevant ⇒ violation of cohesion.

Problems?

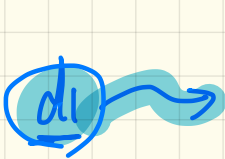
Shared Data via Inheritance

Cohesion

Single Choice Principle

```

class
  SHARED_DATA .
feature
  .interest_rate: REAL
  .exchange_rate: REAL
  .minimum_balance: INTEGER
  .maximum_balance: INTEGER
  ...
end
  
```



DEPOSIT	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M

d2 →

DEPOSIT	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M



WITHDRAW	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M

w2 →

WITHDRAW	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M

d1, d2: DEPOSIT

w1, w2: WITHDRAW

t1, t2: INTERNATIONAL_TRANSFER

d1.set_max_balance (2M)

w2.set_min_balance (500)

t2.set_exchange_rate



TRANSFER	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M

t2 →

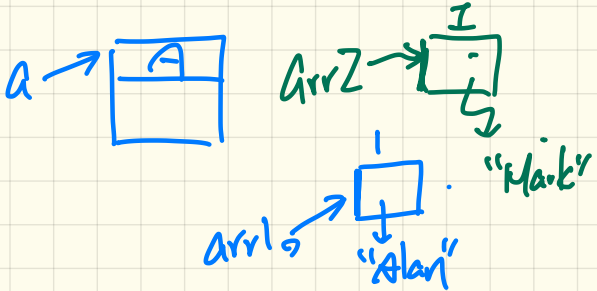
TRANSFER	
ir	0.11
er	2.34
min	1000 500
max	1000000 2M

Lecture 5

Part 3

Once Routines

Once Routine (1)



```

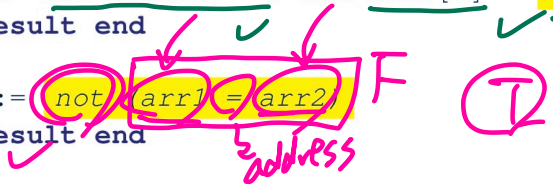
class A
  create make
  feature -- Constructor
    make do end
  feature -- Query
    new_once_array (s: STRING): ARRAY[STRING]
      -- A once query that returns an array.
      once
        create {ARRAY[STRING]} Result.make_empty
        Result.force (s, Result.count + 1)
      end
      new_array (s: STRING): ARRAY[STRING]
        -- An ordinary query that returns an array.
        do
          create {ARRAY[STRING]} Result.make_empty
          Result.force (s, Result.count + 1)
        end
      end
end
  
```

```

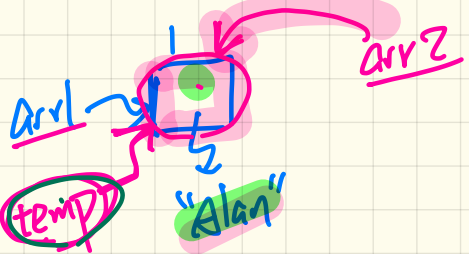
test_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make
    arr1 := a.new_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end
    arr2 := a.new_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Mark"
    check Result end
    Result := not (arr1 = arr2)
    check Result end
  end
  
```

1st call

2nd call



Once Routine (2)



```

class A
  create make
  feature -- Constructor
    make do end
  feature -- Query
    new_once_array (s: STRING): ARRAY[STRING]
      -- A once query that returns an array.
      once
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
      end
    new_array (s: STRING): ARRAY[STRING]
      -- An ordinary query that returns an array.
      do
        create {ARRAY[STRING]} Result.make_empty
        Result.force (s, Result.count + 1)
      end
    end
end
  
```

```

test_once_query: BOOLEAN
local
  a: A
  arr1, arr2: ARRAY[STRING]
do
  create a.make
  arr1 := a.new_once_array ("Alan")
  Result := arr1.count = 1 and arr1[1] ~ "Alan"
  check Result end
  arr2 := a.new_once_array ("Mark")
  Result := arr2.count = 1 and arr2[1] ~ "Alan"
  check Result end

  Result := arr1 = arr2
  check Result end
end
  
```

- Once Routine
1. The very first call
 2. All subsequent calls.

Approximating Once Routines in Java (1)

public

```
class BankData {  
    BankData() { }  
    double interestRate;  
    void setIR(double r);  
    ...  
}
```

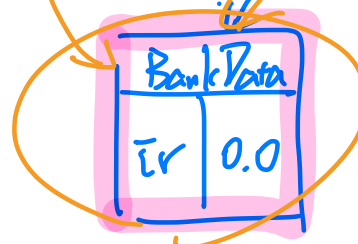
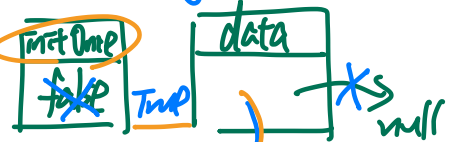
Problem. sharing is not guaranteed.

```
class Account {  
    BankData data;  
    Account() {  
        data = BankDataAccess.getData();  
    }  
    data = BankDataAccess.getData();  
}
```

1st call
2nd call !!

```
class BankDataAccess {  
    static boolean initOnce;  
    static BankData data;  
    static BankData getData() {  
        if (!initOnce) {  
            data = new BankData();  
            initOnce = true;  
        }  
        return data;  
    }  
}
```

Problem?



Approximating Once Routines in Java (2)

data

We may encode Eiffel once routines in Java:

```
class BankData {  
    private BankData() { }  
    double interestRate;  
    void setIR(double r);  
    static boolean initOnce;  
    static BankData data;  
    static BankData getData() {  
        if(!initOnce) {  
            data = new BankData();  
            initOnce = true;  
        }  
        return data;  
    }  
}
```

ACCESS

Problem?

Violating Cohesion

Lecture 5

Part 4

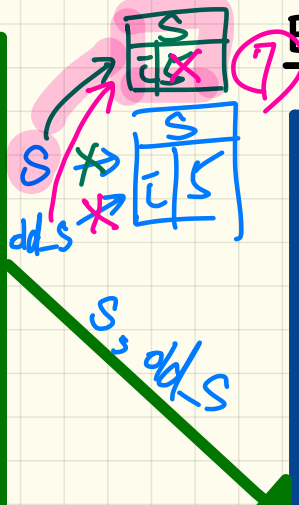
Export Status

Export Status Case 1

```
class CLIENT_1
```

```
...  
test: BOOLEAN  
  local  
    s, old_s: SUPPLIER
```

```
  do  
    → create s.make (5) ✓  
    → old_s := s  
    → create s.make (5) ✓  
    print (old_s = s) F  
    → old_s := s  
    s.make (7) T  
    print (old_s = s) T  
  end  
end
```



```
class CLIENT_2
```

```
...  
test: BOOLEAN  
  local  
    s, old_s: SUPPLIER
```

```
  do  
    create s.make (5)  
    old_s := s  
    create s.make (5)  
    print (old_s = s)  
    old_s := s  
    s.make (7)  
    print (old_s = s)  
  end  
end
```

```
class SUPPLIER
```

```
create { ANY }  
make
```

```
feature { ANY }  
make (init_i: INTEGER)  
  do  
    i := init_i  
  end
```

```
feature  
  i: INTEGER  
end
```

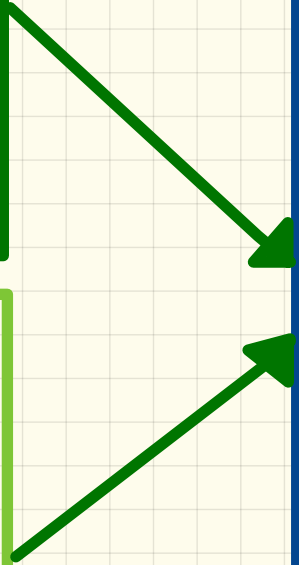
any class can call make as a submethod
any class can call make as a command

Export Status Case 2

```
class CLIENT_1 .  
...  
test: BOOLEAN  
  local  
    s, old_s: SUPPLIER  
  do  
    create s.make (5) ✓  
    old_s := s  
    create s.make (5) ✓  
    print (old_s = s)  
    old_s := s  
    s.make (7) ✗  
    print (old_s = s)  
  end  
end
```

```
class CLIENT_2  
...  
test: BOOLEAN  
  local  
    s, old_s: SUPPLIER  
  do  
    create s.make (5) ✗  
    old_s := s  
    create s.make (5) ✗  
    print (old_s = s)  
    old_s := s  
    s.make (7) ✓  
    print (old_s = s)  
  end  
end
```

```
class SUPPLIER  
  
  create { CLIENT_1 }  
  make  
  
  feature { CLIENT_2 } .  
  make (init_i: INTEGER)  
  do  
    i := init_i  
  end  
  
  feature  
  i: INTEGER  
end
```



Export Status Case 3

```
class CLIENT_1
...
test: BOOLEAN
local
  s, old_s: SUPPLIER
do
  create s.make (5) X
  old_s := s
  create s.make (5) X
  print (old_s = s)
  old_s := s
  s.make (7) X
  print (old_s = s)
end
end
```

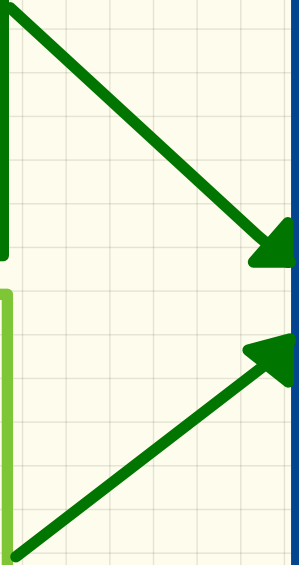
```
class CLIENT_2
...
test: BOOLEAN
local
  s, old_s: SUPPLIER
do
  create s.make (5) X
  old_s := s
  create s.make (5) X
  print (old_s = s)
  old_s := s
  s.make (7) X
  print (old_s = s)
end
end
```

```
class SUPPLIER

create { CLIENT_3 }
  make

feature { CLIENT_3 }
  make (init_i: INTEGER)
  do
    i := init_i
  end

feature
  i: INTEGER
end
```

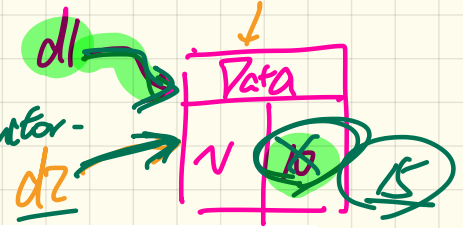


Lecture 5

Part 5

Singleton Pattern

Singleton Design Pattern: Code (1)



only DATA-ACCESS can we make as a constructor-Client:

Supplier:

```
class DATA
create (DATA_ACCESS) make
feature {DATA_ACCESS}
  make do v := 10 end
feature -- Data Attributes
  v: INTEGER
  change_v (nv: INTEGER)
    do v := nv end
end
```

```
expanded class
  DATA_ACCESS .
  feature
    data: DATA
    once create Result.make end
  invariant data = data
```

all separate calls to 'data' retrieve the same 'data' object.
one call to 'make' data
call to 'data' a separate

```
test: BOOLEAN
local
  access: DATA_ACCESS
  → d1, d2: DATA
do
  d1 := access.data 1st call
  d2 := access.data 2nd call
  Result := d1 = d2 (T)
  and d1.v = 10 and d2.v = 10 (T)
check Result end
→ d1.change_v (15)
Result := d1 = d2
  and d1.v = 15 and d2.v = 15
end
end (T)
```

create {DATA} d1. make.

Writing `create d1.make` in test feature does not compile. Why?

Singleton Design Pattern: Code (2.1)

Supplier:

```
class BANK_DATA
create {BANK_DATA_ACCESS} make
feature {BANK_DATA_ACCESS}
  make do ... end
feature -- Data Attributes
  interest_rate: REAL
  set_interest_rate (r: REAL)
  ...
end
```

```
expanded class
  BANK_DATA_ACCESS
feature
  data: BANK_DATA
  -- The one and only access
  once create Result.make end
invariant data = data
```

Client:

```
class
  ACCOUNT ←
feature
  data: BANK_DATA
  make (...)
  -- Init. access to bank data.
  local
    data_access: BANK_DATA_ACCESS
  do
    data := data_access.data
  ...
end
end
```

Writing `create data.make` in client's make feature does not compile. Why?

Singleton Design Pattern: Code (2.2)

```
test_bank_shared_data: BOOLEAN
```

```
-- Test that a single data object is manipulated
```

```
local acc1, acc2: ACCOUNT
```

```
do
```

```
comment("t1: test that a single data object is shared")
```

```
create acc1.make("Bill")
```

```
create acc2.make("Steve")
```

```
Result := acc1.data = acc2.data
```

```
check Result end
```

```
Result := acc1.data ~ acc2.data
```

```
check Result end
```

```
acc1.data.set_interest_rate(3.11)
```

```
Result :=
```

```
acc1.data.interest_rate = acc2.data.interest_rate
```

```
and acc1.data.interest_rate = 3.11
```

```
check Result end
```

```
acc2.data.set_interest_rate(2.98)
```

```
Result :=
```

```
acc1.data.interest_rate = acc2.data.interest_rate
```

```
and acc1.data.interest_rate = 2.98
```

```
end
```

triggers the 1st call to
ID_A3.data

2nd call to
ID_A3.data

