Lecture 6

Part 1

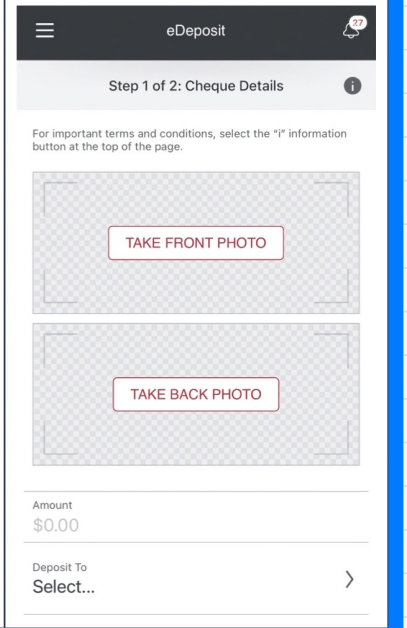
Abstract UI

ETF: Abstract UI vs. Concrete UI

Concrete UI

Concrete UI Interaction

- Insert a bank card
- Validate password
- Select Deposit Transaction
- Select cheque account
- Enter amount
- Select confirm



abstract UI

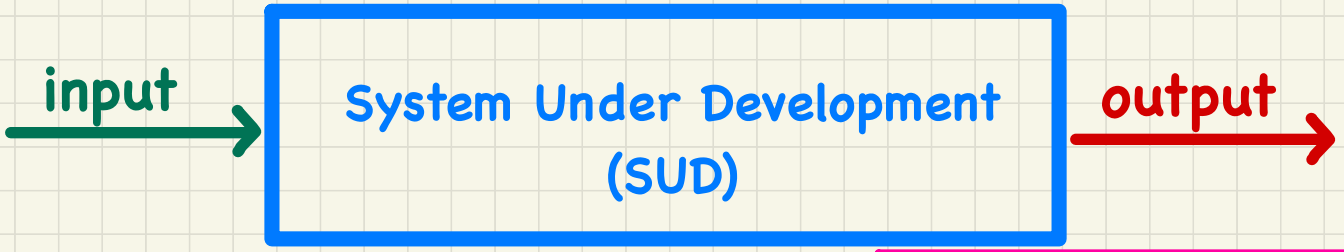
```
system bank

new(id: STRING)
  -- create a new bank account for "id"
deposit(id: STRING; amount: INTEGER)
  -- deposit "amount" into the account of "id"
withdraw(id: STRING; amount: INTEGER)
  -- withdraw "amount" from the account of "id"
transfer(id1: STRING; id2: STRING; amount: INTEGER)
  -- transfer "amount" from "id1" to "id2"
```

Lecture 6

Part 2

Abstract States, Acceptance Tests

Acceptance Testing



$S_0 -> e_1 -> S_1 -> e_2 -> S_2 -> \ldots$

```
new("alan")
new("mark")
deposit("alan", 200)
deposit("mark", 100)
transfer("alan", "mark", 50)
```

```
   {}
-> new("alan")
   {alan: 0}
-> new("mark")
   {alan: 0, mark: 0}
-> deposit("alan", 200)
   {alan: 200, mark: 0}
-> deposit("mark", 100)
    {alan: 200, mark: 100}
-> transfer("alan", "mark", 50)
    {alan: 150, mark: 150}
```

Acceptance Test vs. Unit Test

```
  {}
->new("alan")
  {alan: 0}
->deposit("alan", 200)
  {alan: 200}
```

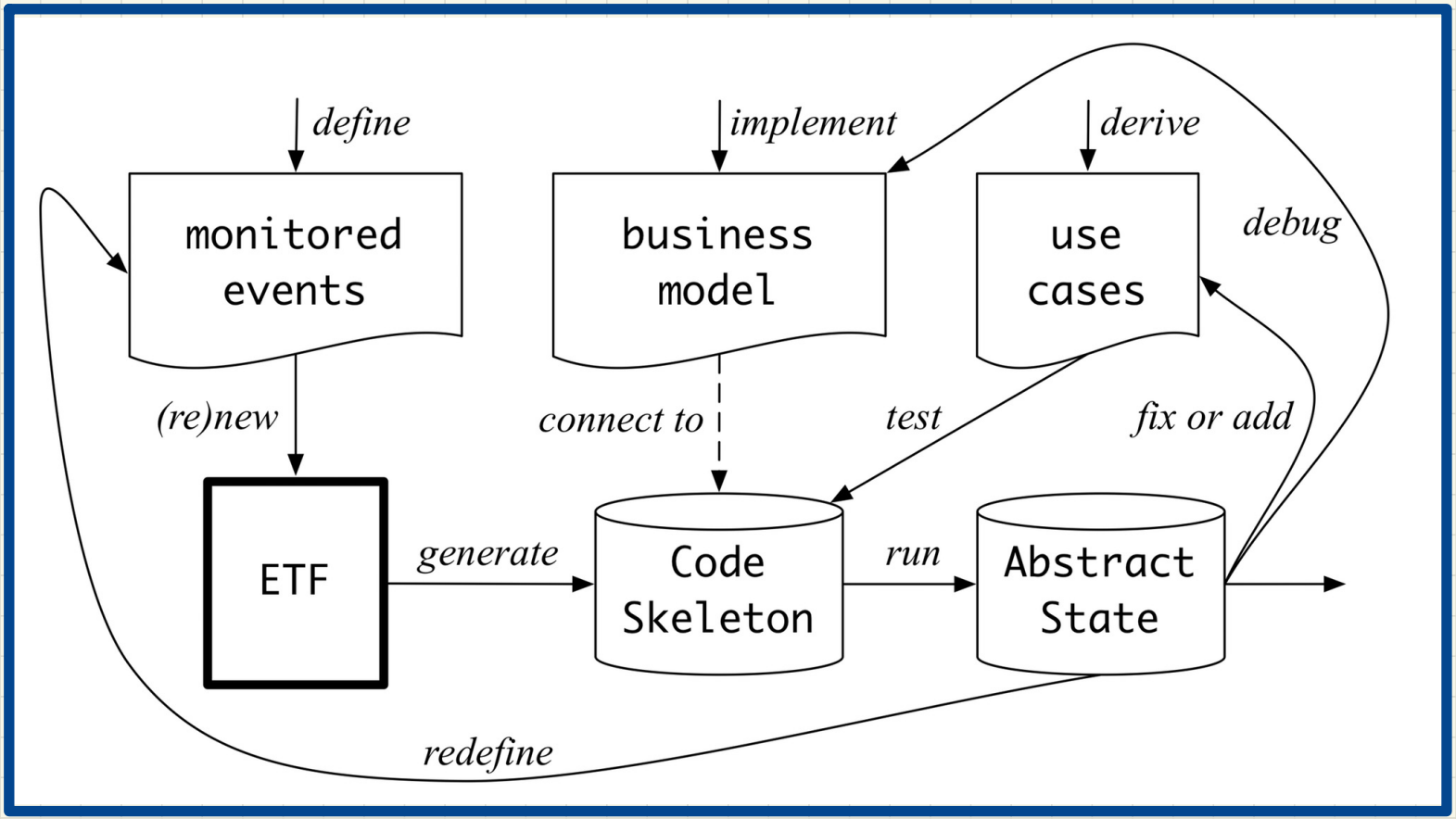
```
test: BOOLEAN
  local acc: ACCOUNT
  do create acc.make("alan")
     acc.add(200)
     Result := acc.balance = 200
  end
```

Lecture 6

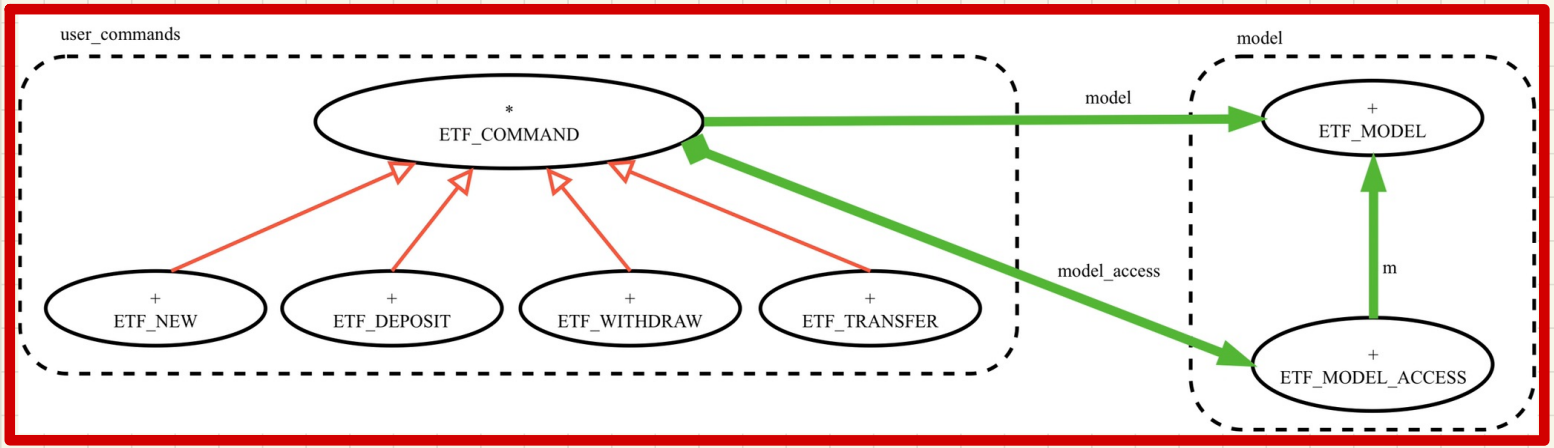
Part 3

ETF: Abstract UI, Model, Regression Tests

ETF: Workflow



ETF: Separating User Interface and Business Model



abstract UI

```
system bank

new(id: STRING)
  -- create a new bank account for "id"
deposit(id: STRING; amount: INTEGER)
  -- deposit "amount" into the account of "id"
withdraw(id: STRING; amount: INTEGER)
  -- withdraw "amount" from the account of "id"
transfer(id1: STRING; id2: STRING; amount: INTEGER)
  -- transfer "amount" from "id1" to "id2"
```

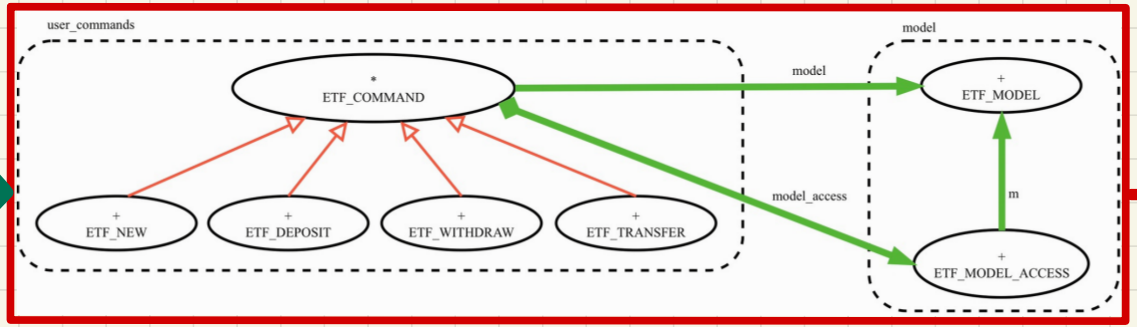
ETF: Singleton Pattern

```
deferred class
  ETF_COMMAND
feature -- Attributes
  model: ETF_MODEL
feature {NONE}
  make(...)
    local
      ma: ETF_MODEL_ACCESS
    do
      ...
      model := ma.m
    end
end
```

```
class
   ETF_DEPOSIT
inherit
   ETF_DEPOSIT_INTERFACE
      redefine deposit end
create
   make
feature -- command
   deposit(id: STRING ; amount: REAL_64)
      do
         if not model.has_user (id) then
            -- Set some error message
         elseif not amount <= model.get_balance (id)  then
            -- Set some other error message
         else
            -- perform some update on the model state
            model.deposit (id, amount)
         end
         -- Publish model update
         etf_cmd_container.on_change.notify ([Current])
      end
end
```

ETF: Input-Output-Based Acceptance Testing

input

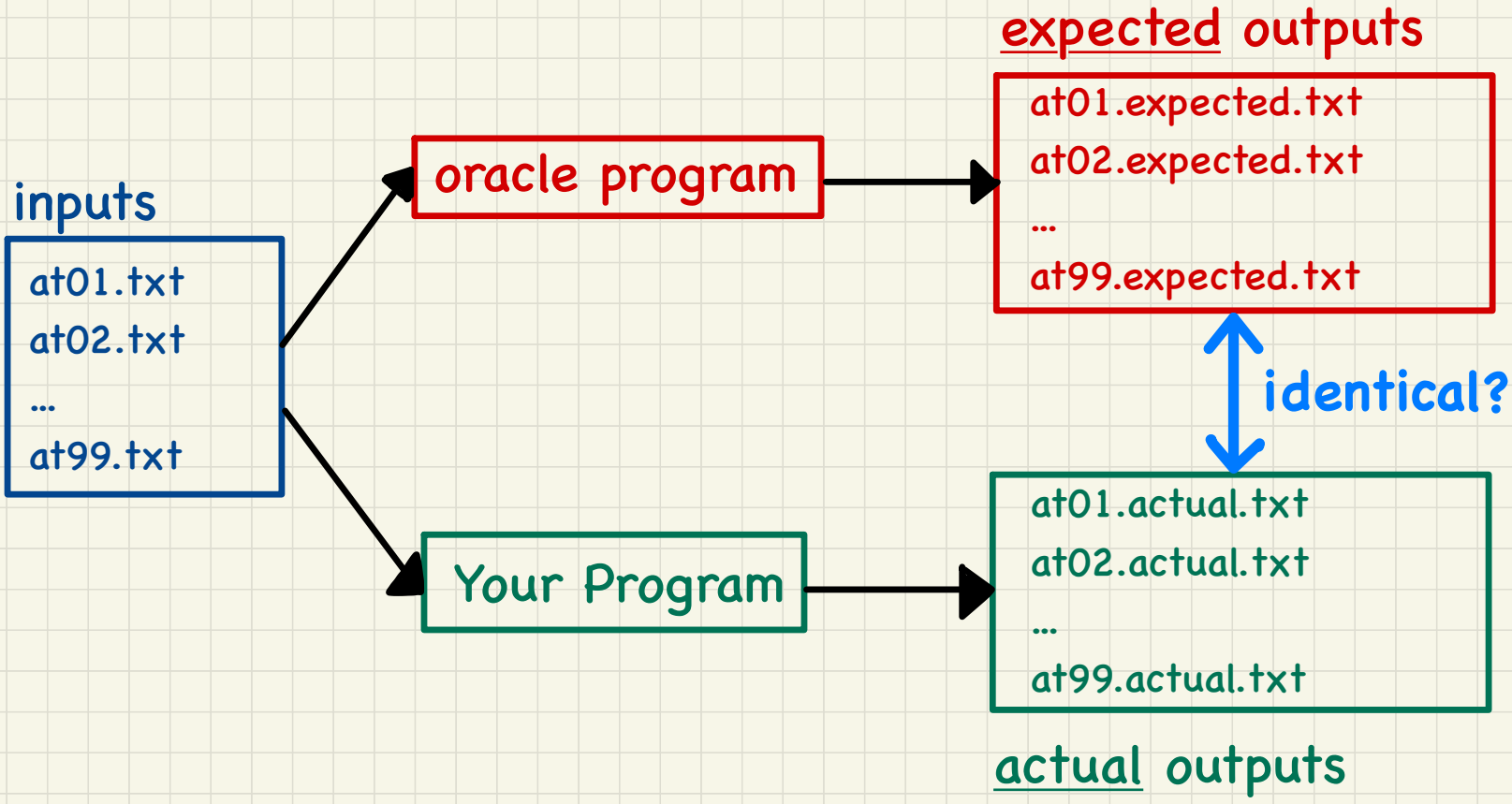


output

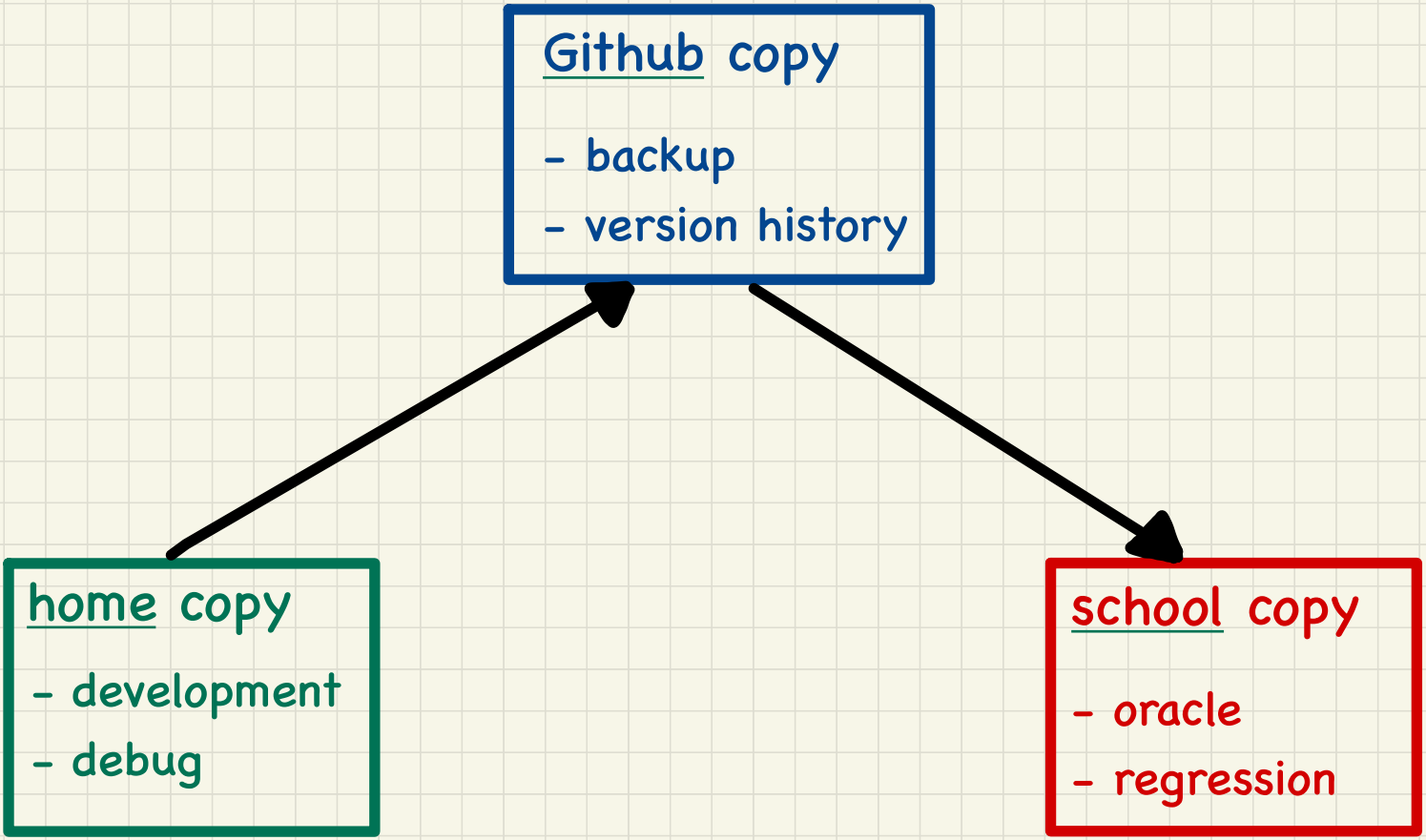
```
new("alan")
new("mark")
deposit("alan", 200)
deposit("mark", 100)
transfer("alan", "mark", 50)
```

```
   {}
-> new("alan")
   {alan: 0}
-> new("mark")
   {alan: 0, mark: 0}
-> deposit("alan", 200)
   {alan: 200, mark: 0}
-> deposit("mark", 100)
    {alan: 200, mark: 100}
-> transfer("alan", "mark", 50)
    {alan: 150, mark: 150}
```

Regression Testing



Automating Regression Testing



Lecture 7

Part 1

A Motivating Problem

Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators

Problem: A *student management system* stores data about students. There are two kinds of university students: *resident* students and *non-resident* students. Both kinds of students have a *name* and a list of *registered courses*. Both kinds of students are restricted to *register* for no more than 10 courses. When *calculating the tuition* for a student, a base amount is first determined from the list of courses they are currently registered (each course has an associated fee). For a non-resident student, there is a *discount rate* applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a *premium rate* applied to the base amount to account for the fee for on-campus accommodation and meals.

Lecture 7

Part 2

1st Design Attempt without Inheritance

1st Design Attempt

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r:  REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r:  REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

1st Design Test

```
test_students: BOOLEAN
 local
   c1, c2: COURSE
   jim: RESIDENT_STUDENT
   jeremy: NON_RESIDENT_STUDENT
 do
   create c1.make ("EECS2030", 500.0)
   create c2.make ("EECS3311", 500.0)
   create jim.make ("J. Davis")
   jim.set_pr (1.25)
   jim.register (c1)
   jim.register (c2)
   Result := jim.tuition = 1250
   check Result end
   create jeremy.make ("J. Gibbons")
   jeremy.set_dr (0.75)
   jeremy.register (c1)
   jeremy.register (c2)
   Result := jeremy.tuition = 750
 end
```

1st Design Attempt

Good design?

Judge by Cohesion

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

1st Design Attempt

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- Change on registration policy?

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r:  REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r:  REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

1st Design Attempt

Good design?

How do you build a

STUDENT_MANGEMENT_SYSTEM

class accordingly?

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r:  REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r:  REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

Without Inheritance (Design 1) Collection of Students

```
class STUDENT_MANAGEMENT_SYSETM
  rs : LINKED_LIST[RESIDENT_STUDENT]
  nrs : LINKED_LIST[NON_RESIDENT_STUDENT]
  add_rs (rs: RESIDENT_STUDENT) do ... end
  add_nrs (nrs: NON_RESIDENT_STUDENT) do ... end
  register_all (Course c) -- Register a common course 'c'.
    do
      across rs as c loop c.item.register (c) end
      across nrs as c loop c.item.register (c) end
    end
end
```

Clinet's Code

```
c: COURSE
rs: RESIDENT_STUDENT
nrs: NON_RESIDENT_STUDENT
sms: SMS
create c.make("3311")
create sms.make
```

```
sms.add_rs(rs)
sms.add_nrs(nrs)
sms.register_all(c)
```

Q: What if **more** kinds of students are to be introduced?

Lecture 7

Part 3

2nd Design Attempt without Inheritance

2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attributes
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attributes
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

Good design?

Judge by Cohesion

```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attribures
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- An existing kind is obeselete?

```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attributes
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

Good design?

How do you build a

STUDENT_MANGEMENT_SYSTEM

class accordingly?

```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

Without Inheritance (Design 2) Collection of Students

```
class
   STUDENT_MANAGEMENT_SYSTEM
feature -- attributes
   students: LINKED_LIST[STUDENT]
feature -- command
   add_student(s: STUDENT)
      do
         students.extend(s)
      end
   register_all (c: COURSE)
      do
        across students is s
           loop
              s.register(c)
           end
      end
end
```

Clinet's Code

```
c: COURSE
rs: STUDENT
nrs: STUDENT
sms: SMS
create c.make("3311")
create sms.make




sms.add_student(rs)
sms.add_student(nrs)
sms.register_all(c)
```

Q: What if more kinds of students are to be introduced?

Design Attempt 2.5

```
class
   STUDENT
create
   make
feature -- kind-specific attributes
   rate: LINKED_LIST[REAL]
   tuition: LINKED_LIST[REAL]
   max: LINKED_LIST[INTEGER]
feature -- attributes
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
       rate :=  << 1.25, 0.75 >>
       max := << 6, 4 >>
       tuition := <<0.0, 0.0 >>
     end
   ...
end
```

```
get_tuition: REAL
   do
      across courses is c loop
         tuition[kind] :=
            tuition[kind] + c.fee
      end
      tuition[kind] :=
         tuition[kind] * rate[kind]
   end
```

```
register (c: COURSE)
   do
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- An existing kind is obeselete?

Lecture 7

Part 4

Using Inheritance for Code Reuse

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

```
class
  RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  premium_rate : REAL
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * premium_rate end
end
```

```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate : REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * discount_rate end
end
```

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

```
class
  RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  premium_rate : REAL
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * premium_rate end
end
```

```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate : REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * discount_rate end
end
```

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

```
class
  RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  premium_rate : REAL
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * premium_rate end
end
```

```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate : REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * discount_rate end
end
```

With Inheritance (Design 3) Collection of Students

```
class
   STUDENT_MANAGEMENT_SYSTEM
feature -- attributes
   students: LINKED_LIST[STUDENT]
feature -- command
   add_student(s: STUDENT)
      do
         students.extend(s)
      end
   register_all (c: COURSE)
     do
       across students is s
          loop
             s.register(c)
          end
     end
end
```

```
c: COURSE
rs: STUDENT
nrs: STUDENT
sms: SMS
create c.make("3311")
create sms.make




sms.add_student(rs)
sms.add_student(nrs)
sms.register_all(c)
```

Q: What if more kinds of students are to be introduced?