Lecture 17: Multiple inheritance
Combining abstractions

Given the classes

- TRAIN_CAR, RESTAURANT

how would you implement a DINER?
Examples of multiple inheritance

Combining separate abstractions:

- Restaurant, train car
- Calculator, watch
- Plane, asset
- Home, vehicle
- Tram, bus
Composite figures
Multiple inheritance: Composite figures

Simple figures

A composite figure
Defining the notion of composite figure

center
display
hide
rotate
move
...

FIGURE

LIST [FIGURE]

count
put
remove
...

COMPOSITE_FIGURE
In the overall structure

- OPEN _FIGURE
- POLYLINE
- POLYGON
- SEGMENT
- TRIANGLE
- RECTANGLE
- SQUARE
- ELLIPSE
- CIRCLE
- COMPOSITE _FIGURE

- FIGURE
- CLOSED _FIGURE

- LIST [FIGURE ]

- perimeter*
- perimeter*
- perimeter+
- perimeter++
- perimeter++
- perimeter++
- perimeter++
- perimeter++
- perimeter++
A composite figure as a list

Cursor

forth

after
class COMPOSITE_FIGURE inherit FIGURE

LIST [FIGURE]

feature display

-- Display each constituent figure in turn.
do
from start until after loop

item.display

forth

end

end

... Similarly for move, rotate etc. ...

end

Requires dynamic binding
Going one level of abstraction higher

A simpler form of procedures display, move etc. can be obtained through the use of iterators

Use agents for that purpose
Multiple inheritance: Combining abstractions

- **COMPARABLE**
  - `<`, `<=`, `>`, `>=`
  - (total order relation)

- **NUMERIC**
  - `+`, `-`, `*`, `/`
  - (commutative ring)

- **INTEGER**

- **REAL**

- **STRING**

- **COMPLEX**
The Java-C# solution

No multiple inheritance for classes

“Interfaces”: specification only (but no contracts)
  - Similar to completely deferred classes (with no effective feature)

A class may inherit from:
  - At most one class
  - Any number of interfaces
Multiple inheritance: Combining abstractions

- COMPARABLE
  - STRING
  - REAL
- NUMERIC
  - COMPLEX
- INTEGER

- (<, <=, >, >=)
- (+, -, *, /)

(total order relation)
(commutative ring)
How do we write `COMPARABLE`?

deferred class `COMPARABLE [G ]` feature

  less alias "<" (x : COMPARABLE [G ]): BOOLEAN
  deferred
  end

  less_equal alias "<=" (x : COMPARABLE [G ]): BOOLEAN
  do
    Result := (Current < x or (Current = x))
  end

  greater alias ">" (x : COMPARABLE [G ]): BOOLEAN
  do Result := (x < Current) end

  greater_equal alias ">=" (x : COMPARABLE [G ]): BOOLEAN
  do Result := (x <= Current) end
Lessons from this example

Typical example of program with holes

We need the full spectrum from fully abstract (fully deferred) to fully implemented classes

Multiple inheritance is there to help us combine abstractions
Multiple inheritance: Name clashes

f
A

? C

B

f
Resolving name clashes

rename $f$ as $A_f$
Consequences of renaming

a1 : A
b1 : B
c1 : C
...
c1.f
c1.A_f
a1.f
b1.f

Invalid:
- a1.first_f
- b1.first_f
Are all name clashes bad?

A name clash must be removed unless it is:

- Under repeated inheritance (i.e. not a real clash)
- Between features of which at most one is effective (i.e. others are deferred)
Feature merging

\[ f^* \quad A \quad f^* \quad B \quad f^* \quad C \quad f^* \quad D \]

* Deferred
+ Effective
Feature merging: with different names

class D
inhibit
A
  rename
    g as f
  end
B
C
  rename
    h as f
  end
feature ...
end

g *

f *

h +

D

Deferred
Effective
Renaming
Feature merging: effective features

Deferred
Effective
Undefine
deferred class
  T
inherit
  S
  undefine v end
feature
  ...
end
Merging through undefined

```text
class D
  inherit A
  undefine f end
  B
  C
  undefine f end
feature ...
end
```

* Deferred
+ Effective
-- Undefine
Merging effective features with different names

class D
  inherit A
    undefine f end
  B
    rename g as f
    undefine f end
  C
    rename h as f
    end
feature ... end

D

A f +

B g

C h +

D

f --

f

h -- f
Acceptable name clashes

If inherited features have all the same names, there is no harmful name clash if:

- They all have compatible signatures
- At most one of them is effective

Semantics of such a case:

- Merge all features into one
- If there is an effective feature, it imposes its implementation
Feature merging: effective features
A special case of multiple inheritance

Allow a class to have two or more parents.

Examples that come to mind: ASSISTANT inherits from TEACHER and STUDENT.

This is a case of repeated inheritance
Indirect and direct repeated inheritance

A

B

C

D

A

D

A

D
Multiple is also repeated inheritance

A typical case:

```
ANY
  copy
  is_equal

LIST
  copy
  is_equal

D
  copy
  is_equal

C
  C_copy
  C_is_equal
```
Acceptable name clashes

If inherited features have all the same names, there is no harmful name clash if:

- They all have compatible signatures
- At most one of them is effective

Semantics of such a case:

- Merge all features into one
- If there is an effective feature, it imposes its implementation
Features such as $f$, not renamed along any of the inheritance paths, will be shared.

Features such as $g$, inherited under different names, will be replicated.
The need for select

A potential ambiguity arises because of polymorphism and dynamic binding:

\[ a_1 : \text{ANY} \]
\[ d_1 : \text{D} \]

\[ \ldots \]

\[ a_1 := d_1 \]
\[ a.\text{copy} (\ldots) \]
Removing the ambiguity

class D
  inherit LIST [T ]
  select
    copy, is_equal
  end

C
  rename
    copy as C_copy,
    is_equal as C_is_equal,
  ...
end
When is a name clash acceptable?

(Between \( n \) features of a class, all with the same name, immediate or inherited.)

- They must all have compatible signatures.
- If more than one is effective, they must all come from a common ancestor feature under repeated inheritance.
Another application of renaming

Provide locally better adapted terminology.

Example: child \text{(TREE)}; subwindow \text{(WINDOW)}.
What we have seen

A number of games one can play with inheritance:

- Multiple inheritance
- Feature merging
- Repeated inheritance