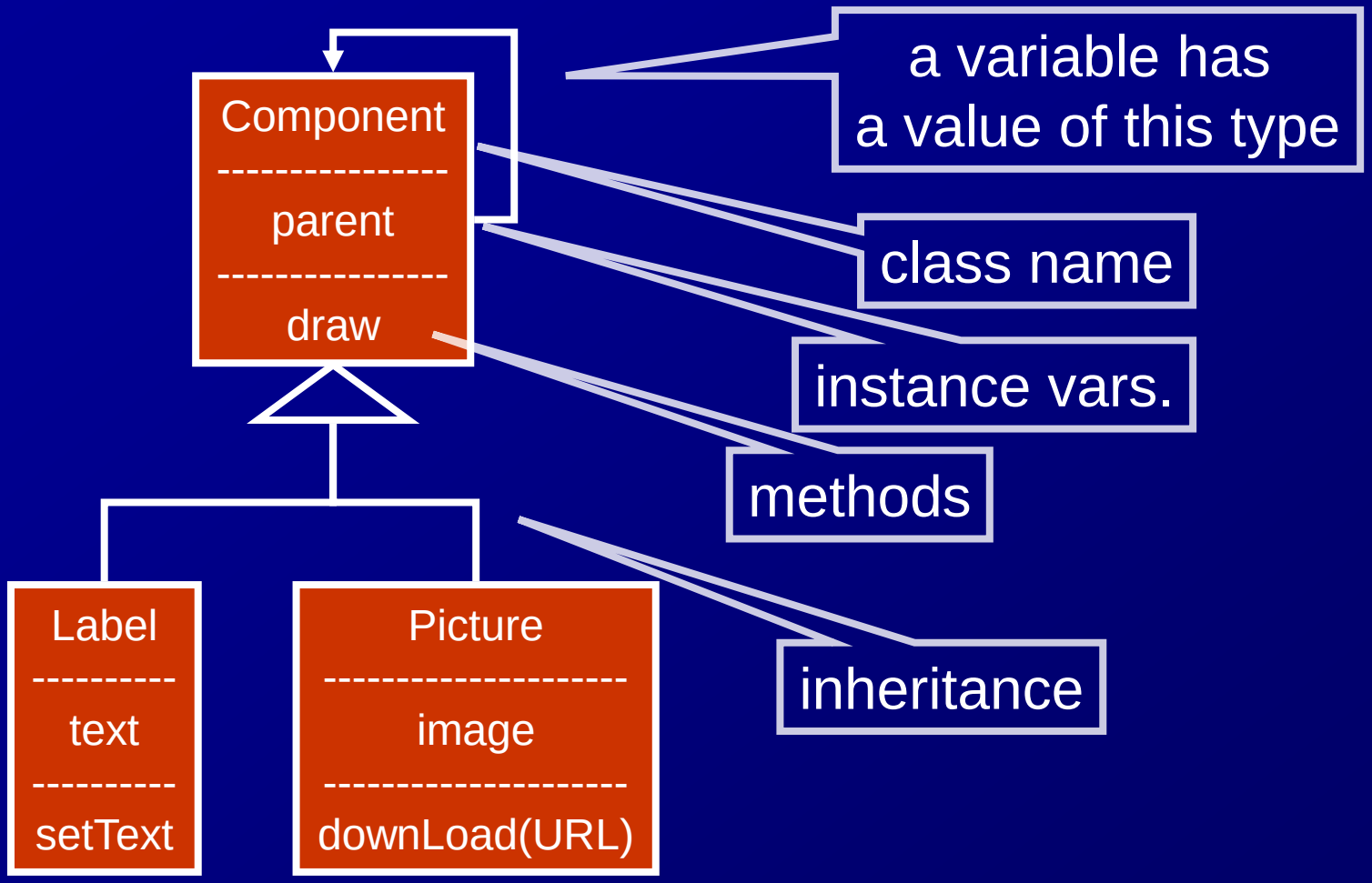
Programming Language Design

2015

Week #4: Object-oriented programming (OOP) (2)

Instructor: Hidehiko Masuhara

Class diagram



(anecdote) Origin of class diagrams

- Class diagram is a pictorial language to express classes and their relationships
- Current standard: UML (unified modeling language)
  - UML is defined in UML (metamodeling)
  - unification of 3 major languages: OMT (Rumbaugh), OOSE (Jacobson), and Booch method

Quiz(1/2): design a school management system (10min.)

(design at the class diagram level; can assume any language you like)

- Casts: Teacher, Student, TA
  - Each has a name and an ID
  - A teaching staff (teacher or TA) has a list of teaching classes
  - A student (student or TA) has a list of registered classes
- Requirements
  - Calculate a total amount of annual salary of all teaching staff
    - Teacher: no teaching class → 4M Yen: at least one → 6M Yen
    - TA: the number of teaching classes x 50K Yen
  - Calculate the average number of registered classes of all students (including TAs)
  - Calculate the average "working" hours of all members
    - for teaching staff: # of teaching classes x 3 hours
    - for students: # of registered classes x 2 hours
    - for TAs: sum of above

Quiz (2/2): design a matrix library (10min.)

- Classes and methods
  - Classes: Matrix, IdMatrix
  - Methods: add(other), mul(other), print(), at(i,j)
- Matrix represents general matrices
- IdMatrix represents identity matrices E
  - faster mult. by using: A x E = E x A = A
  - (other operations are the same)

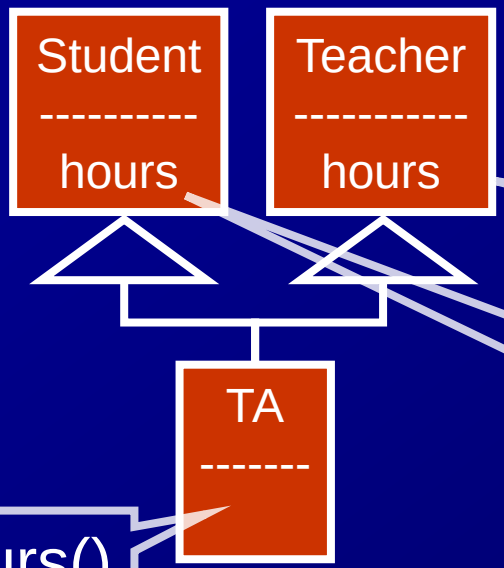
Multiple inheritance

- Defining a class by inheriting definitions from multiple classes
  - eg: C++, CLOS, Eiffel
- Example: TA inherits Teacher and Student
  - class TA extends Teacher, Student { ... }
  - for total amount salary, it behaves as a Teacher
  - for average registered classes, it behaves as a student

Problem of multiple inheritance: ambiguity [Singh94]

- If two superclasess have the method definitions of the same name,
  - which one will be inherited? (Eg: if both Teacher and Student have hours(), which one will be used for TA?)
  - how can we "supercall" both? (Eg: hours() for TA is [hours() for Teacher] + [hours() for Student])
- Solution in C++:
  - Compile error when ambiguous
  - Specify a superclass name upon a super call
- Solution in CLOS:
  - Chooses the "closest" ancestor in some order
  - (no direct solution for supercalls)

Multiple inheritance and ambiguity: C++ vs CLOS



C++

- specify a super class

using Student::hours()

CASE#1
for hours(), TAs are considered as students

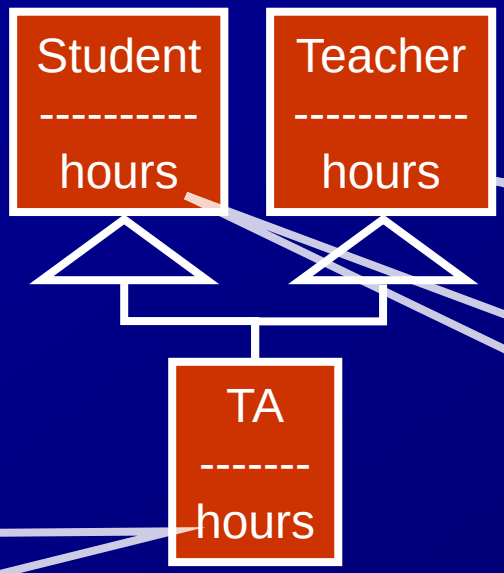
teaching.size()*3 + super.hours()

registered.size()*2 + super.hours()

CLOS

- linearization: Student takes over Teacher

Multiple inheritance and ambiguity: C++ vs CLOS



C++

- specify a super class

Student::hours() + Teacher::hours()

teaching.size()*3 + super.hours()

registered.size()*2 + super.hours()

CASE#2
for hours(), TAs are combination of both

CLOS

- no direct solution

Problem of multiple inheritance: diamond inheritance

- When parents have the same ancestor, and the ancestor defines an instance var., how many instance vars. should the class have?
- eg: name&id for TA (defined in Person)
  - 1 inst. var. (its unique to a person)
- Should inst. vars. always unique?
  - or, different IDs as a student and as a teaching staff
  - another example (next)

Example of diamond inheritance: GUI library

- Drawable: anything on screen (eg: buttons)
- ColorDrawable: drawable with colors
- Scriptable: anything that can be represented as a string
- Rectangle
- Button: string in a rectangle, reacting to mouse clicks
  - when Button inherits from Scriptable ad Button, how color information is managed?

Example of diamond inheritance: solution in C++

- specify "uniqueness" upon inheritance
  - virtual parent: inst. vars. in common ancestors are unique
  - non-virtual parent: ancestors' inst. vars. are distinct
- most other languages supports only virtual parents
  - other mechanisms for distinct inst. vars.

Quiz (2/2): design a matrix library (10min.)

- Classes and methods
  - Classes: Matrix, IdMatrix
  - Methods: add(other), mul(other), print(), at(i,j)
- Matrix represents general matrices
- IdMatrix represents identity matrices E
  - faster mult. by using: A x E = E x A = A
  - (other operations are the same)

Double-dispatching for selecting methods with two or more arguments

```
class Matrix {
  mul(right) {
   return right.mulMatrix(this);
  }
  mulMatrix(left) {
    for(i=...) for(j=...) for(k=...) ...
  }
  mulIdMatrix(left) { return this; }
}
class IdMatrix {
  mul(right) {
    return right.mulIdMatrix(this);
  }
  mulMatrix(right) { return right; }
  mulIdMatrix(right) { return this; }
}
```

- define dispatching method in C:

```
class C {
  m(arg) {
    arg.mC(this);
  }
}
```

- define body method for a pair of C & D:

```
class D {
 mC(arg) { ...body... }
}
```

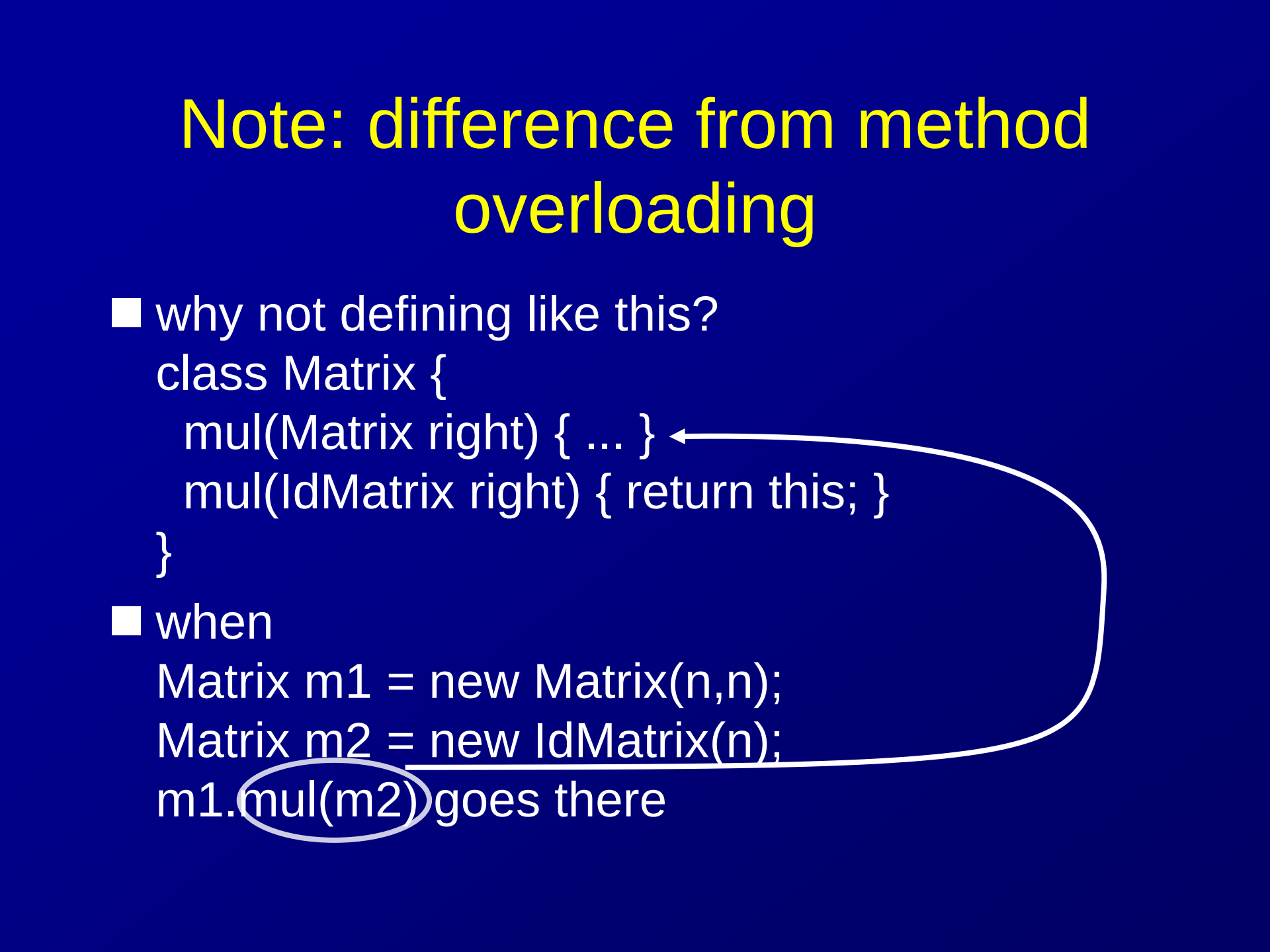
Note: difference from method overloading

- why not defining like this?

```
class Matrix {
  mul(Matrix right) { ... }
  mul(IdMatrix right) { return this; }
}
```

- when

```
Matrix m1 = new Matrix(n,n);
Matrix m2 = new IdMatrix(n);
m1.mul(m2) goes there
```



Generic function (aka multi-method)

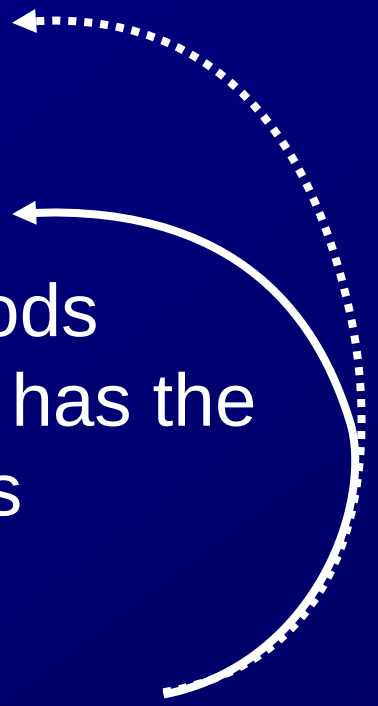
- a function that can specify expected classes of all arguments
  - can define more than one with the same name
- dispatched on the runtime classes of all arguments
- Best known: CLOS

A matrix library with generic functions

- mul(Matrix left, Matrix right) {
  for(i=...) for(j=...) for (k=...)
  }
  mul(IdMatrix left, Matrix right) {
  return right;
  }
  mul(Matrix left, IdMatrix right) {
  return left;
  }

Ambiguity in generic functions

- mul(Matrix left, Matrix right) { ... }
  mul(IdMatrix left, Matrix right) { ... }
  mul(Matrix left, IdMatrix right) { ... }
- When there are more than one methods applicable, dispatch to a method that has the most specific classes in its arguments
- e.g., where these exps go?
  mul(new Matrix(n), new IdMatrix(n))
  mul(new IdMatrix(n),
  new IdMatrix(n))?



Resolving ambiguity in CLOS

- Assume there are methods of: $m(C_1,D_1,E_1)$, $m(C_2,D_2,E_2)$, ...
- and $m(C_0,D_0,E_0)$ is performed:

1. collect methods of type (C_i, D_i, E_i) where $C_0 <: C_i$, $D_0 <: D_i$, $E_0 <: E_i$
2. dispatch to i when $C_i <: C_j$, $D_i <: D_j$, $E_i <: E_j$ for all j
3. dispatch to i when $C_i <: C_j$, $D_i :> D_j$, $E_i = E_j$ for all j (left arguments precedes)

Predicate dispatch

- Assume
  - class: Integer, Float
  - methods: add(left,right), mul(left,right)
    generic function of 4 combinations of methods
- Optimize in the following cases
  0 + x = x + 0 = x
  0 * x = x * 0 = 0
  1 * x = x * 1 = 0
- (Note: in the matrix class example, an identity array is in a different class)

Predicate dispatch [EKC98]

- method definition can have conditions
  mul(x@Integer, y@Integer) { return x*y }
  mul(x@Integer, y@Integer) when x==0 {
    return y; }
  mul(x@Integer, y@Integer) when y==0 {
    return x; }
- conditions on object's states
  class File { int fd... }
  read(f@File{fd=d}) when d>=0 { ... }
  read(f@File{fd=d}) when d<0 { error }
  (enables a deep pattern matching)

References

- [Singh94] Ghan Bir Singh. 1994. Single versus multiple inheritance in object oriented programming. SIGPLAN OOPS Mess. 5, 1 (January 1994), 34-43.
- [EKC98] Michael Ernst, Craig Kaplan, and Craig Chambers. "Predicate dispatching: A unified theory of dispatch." In *Proceedings of ECOOP*, 1998. pp.186-211.