University at Buffalo
The State University of New York

cse@buffalo

CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ANNOUNCEMENTS

(a very early) EXAM 1 NOTICE

DATE: Tuesday October 4

TIME: 8:45 PM – 9:45 PM

LOCATION: various rooms in NSC

specific room/seat assignments to come

COVERAGE:

lecture material up to and including 9/23 (this week)

lab material up to and including lab 3 (next week)

readings: all assigned up to and including 3.2

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.

IF YOU HAVE A CONFLICT

send me e-mail:
alphonce@buffalo.edu

use this subject line:
[CSE115] Exam 1 conflict

attach documentation of conflict
(e.g. screenshot of class schedule that has your name and the conflict)

no later than:
9:00 PM on Wednesday Sept 28

EXTRA SUPPORT

Extra office hours have been added Th/Fr this week

See PEOPLE page of course website

We are arranging for exam review sessions on the weekend – stay tuned for room/date/time details

ELECTRONICS:

off & away

ROADMAP

Last time

- method definitions
- invocation records
- ‘this’

Today

- Relationships (composition/association)

Coming up

- Relationships (continued)

REVIEW

Scope & Lifetime

	SCOPE	LIFETIME
LOCAL VARIABLE	From point of declaration to end of brace-delimited block containing the declaration For now think roughly: method body	From method invocation to method exit: the duration of a method call. For now, think roughly: short/fleeting
INSTANCE VARIABLE	class body	From object creation to object reclamation. For now, think roughly: long/persistent

Memory organization

All memory allocated by 'new' comes from the heap.

Objects are allocated space by 'new', and their representations (which contain their **instance variables**) therefore exist on the heap.

Local variables are stored on the runtime stack. Each method invocation (call) results in an invocation record (stack frame) being added to the top of the stack. When a method exits, its invocation record is removed from the top of the stack.

MOVING ON

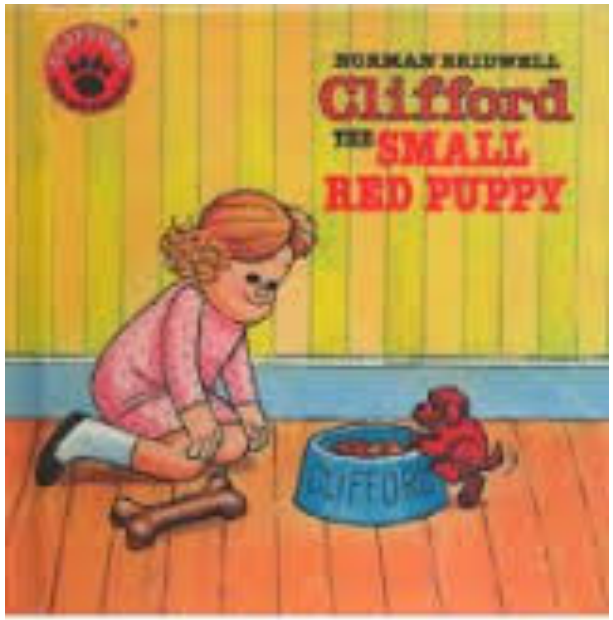
RELATIONSHIPS

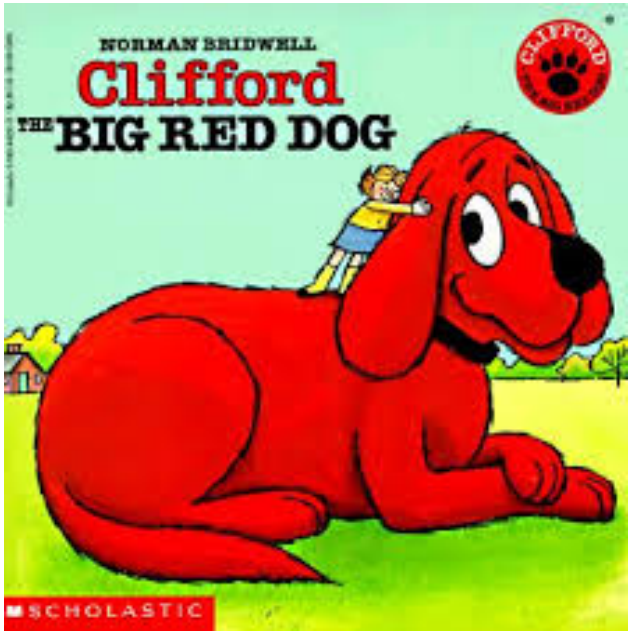
Relationships in model and code

relationships exist between objects in problem domains

want to capture those relationships in our models and express them in our code

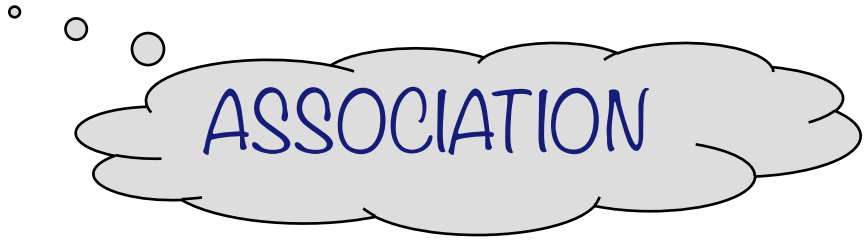
Computing and Clifford





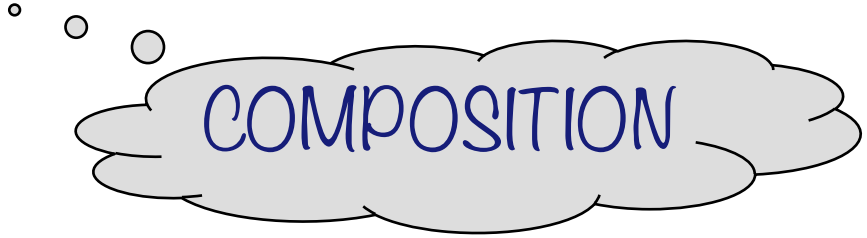
Clifford's relationship to his collar

- Clifford is associated with different collars throughout his life



Clifford's relationship to his tail

- Clifford has the same tail throughout his life



COMPOSITION

(whole-part)

Composition

A whole-part relationship (e.g. Dog-Tail)

Whole and part objects have same lifetime

- when whole is created, it has its parts
- when whole is destroyed, parts go away too

In code, this involves 3 changes to whole class:

Declaration of *instance* variable of part type

Instantiation of part class in whole class constructor

Assignment of new part instance to *instance* variable

Important points about composition

Whole has responsibility for creating its parts (which is why instantiation of parts happens in constructor of whole).

Whole can communicate with parts. This is why an instance variable is declared: to establish a name for the newly created object.

Dog – Tail example in Java

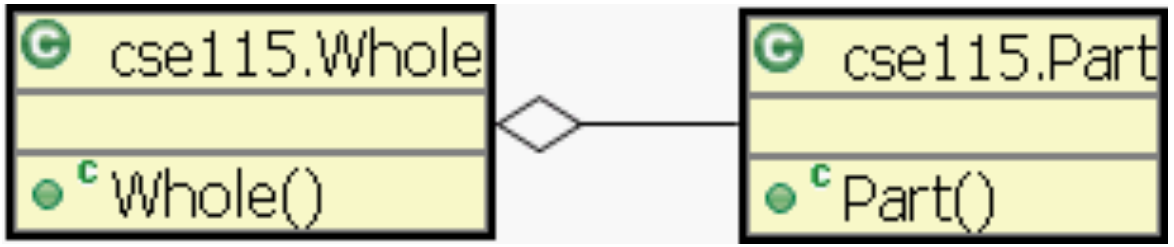
```
public class Dog {
    private Tail _tail;
    public Dog() {
        _tail = new Tail();
    }
}
```

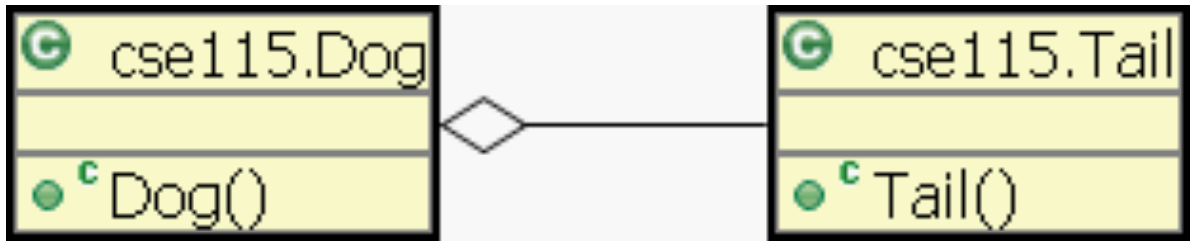
UML CLASS DIAGRAMS

UML = Unified Modeling Language

express design without reference to an implementation language

Examples:





Binary Class Relationships: directional

binary → two classes are involved

- **source class** has code modification
- **target class** does not

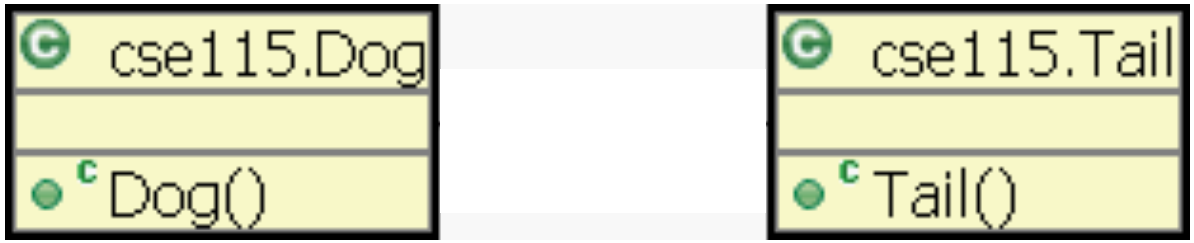
composition

- source: WHOLE
- target: PART

in diagram:

- line decoration is on source/WHOLE
- show only detail that's needed/desired

Without composition

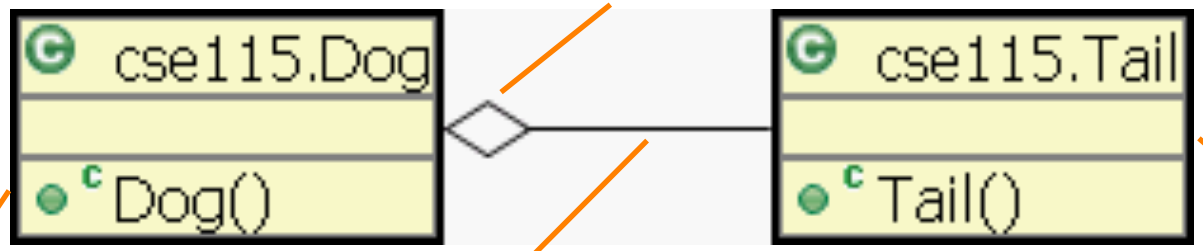



```
package cse115;

public class Dog {

   public Dog() {

   }

}
```

```
package cse115;

public class Tail {

   public Tail() {

   }

}
```

With composition

Decoration (a diamond) is on source side of line for composition



Source class

Relationship line for composition

Target class

```
package cse115;

public class Dog {
   private Tail _tail;
   public Dog() {
      _tail = new Tail();
   }
}
```

```
package cse115;

public class Tail {
   public Tail() {
   }
}
```

In the source class' constructor a reference to a new instance of the target class is assigned to the instance variable.

Source class instantiates the target class in its constructor

Source class declares an instance variable of the target class type

Naming rules

names can consist of

- UPPER CASE LETTERS
- lower case letters
- digits (0 – 9)
- the underscore, '_'
- first character cannot be a digit

this is a slight simplification of the actual rules

- they are the truth
- they are not the whole truth

Naming conventions

packages:

- no upper case letters used (this_is_an_example)

classes (and constructor names):

- first character: upper case letter
- camel case afterwards (ThisIsAnExample)

instance variables:

- first character: underscore '_'
- second character: lower case
- camel case afterwards (_thisIsAnExample)

local variables (and method names):

- first character: lower case
- camel case afterwards (thisIsAnExample)

ASSOCIATION

No necessary lifetime link between the two objects involved

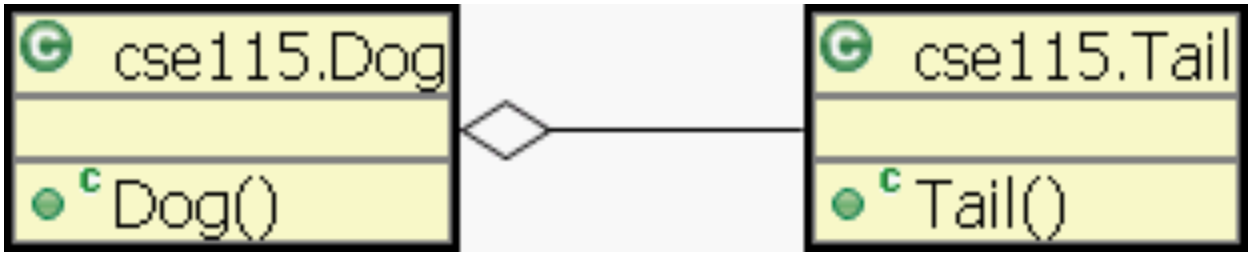
Two implementations:

The first is very similar to composition, but differs in one crucial respect: where the target class is instantiated.

The second, which decouples lifetimes completely, is a bit more complex but also more flexible.

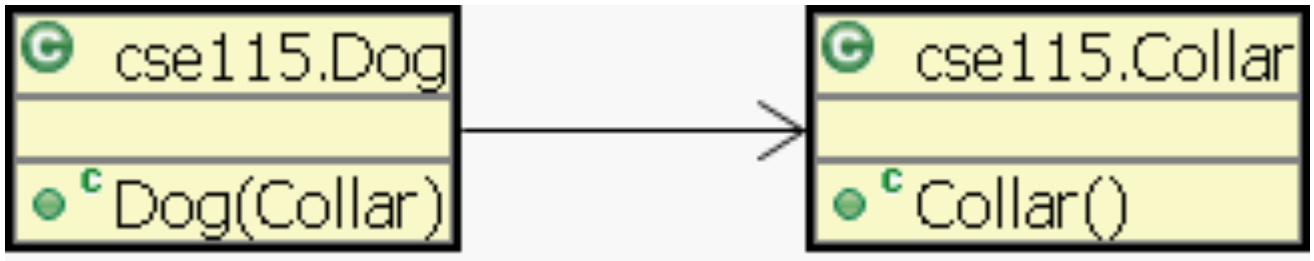
Dog-Tail relationship is COMPOSITION

Dog takes responsibility for creating a Tail



Dog-Collar relationship is ASSOCIATION

Dog takes NO responsibility for creating Collar



3 changes to source class:

1. Declaration of instance variable
2. Assignment of ***existing*** instance to the instance variable
3. Parameter of constructor is of same type as instance variable

Dog – Collar example in Java

```
public class Dog {
    private Collar _myCollar;
    public Dog(Collar c){
        _myCollar = c;
    }
}
```

1
2
3

Parameters vs arguments

A parameter is a variable

declared in the parameter list of a method definition

An argument is a value

provided in the argument list of a method call

When a method is called each parameter is assigned the corresponding argument.

The parameters are local variables, and appear in the method's invocation record.

mathematical example on board

The area of a rectangle whose height is h and width is w can be computed using this function:

f(h,w) = h * w

To compute the area of a specific rectangle we supply concrete values that substitute for the unknowns h and w:

f(3,5) = h * w where h=3 and w=5, or 3*5 = 15

h and w are parameters

3 and 5 are arguments