Programming Abstractions

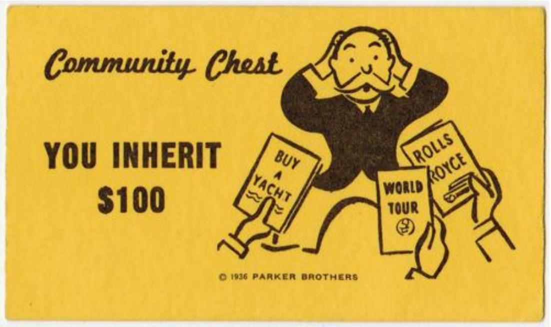
CS106X

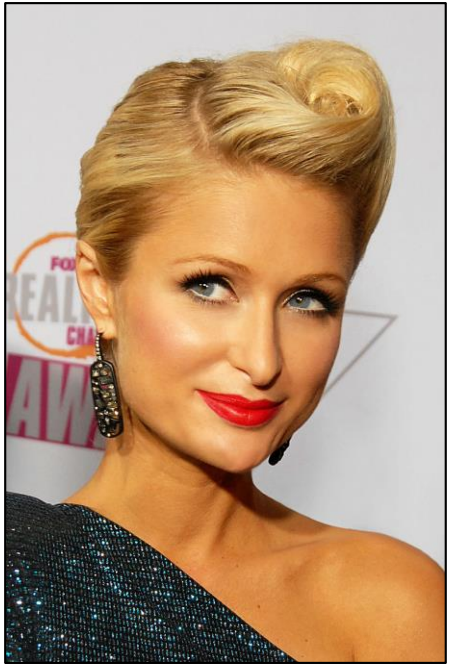
Cynthia Lee

Inheritance Topics

Inheritance

- The basics
  - Example: Stanford GObject class
- Polymorphism
  - Example: Expression trees (final project)





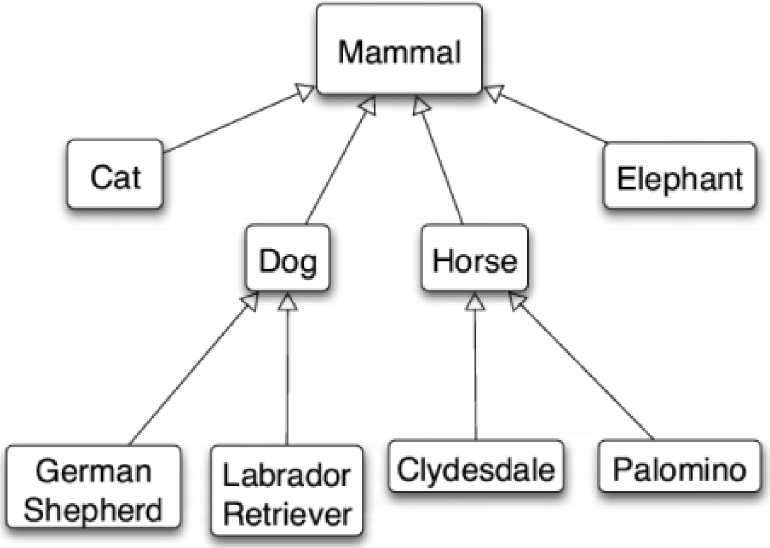
Inheritance

What? Why? How?

Inheritance: what?

is-a relationship: A hierarchical connection where one category can be treated as a specialized version of another.

- every rectangle *is a* shape
- every lion *is an* animal
- every lawyer *is an* employee



type hierarchy: A set of data types connected by is-a relationships that **can share common code**.

- Re-use!

Inheritance: why?

- Remember the #1 rule of computer scientists:
  - Computer scientists are super lazy
  - …in a good way!

- We want to reuse code and work as much as possible
- You've already seen this going back to the very start of your CS education:
  - Loops and Functions *(instead of copy&paste to repeat code)*
  - Arrays *(instead of copy&paste to make 100 named variables)*
  - Data structures *(same idea as arrays but more expressive)*

- Inheritance is another way of organizing smart reuse of code

Inheritance: how?

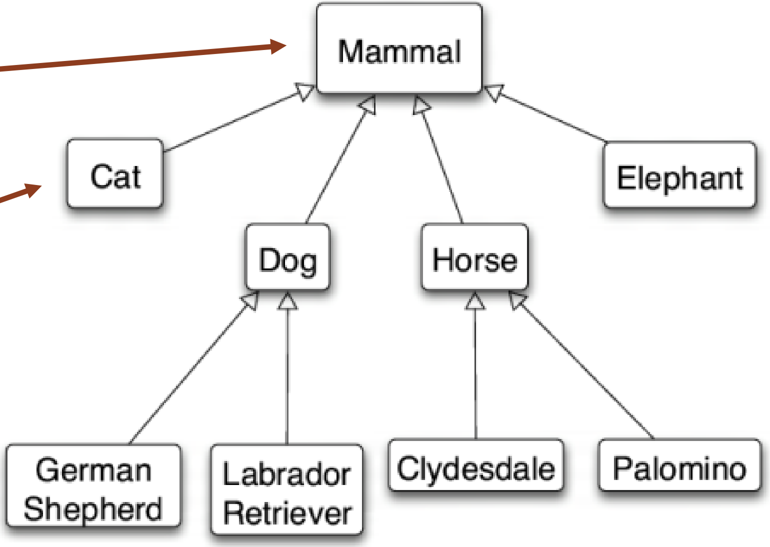
inheritance: A way to form new classes based on existing classes, taking on their attributes/behavior.

- a way to group related classes
- a way to share code between two or more classes

One class can *extend* another, absorbing its data/behavior.

Inheritance vocab

- **superclass** (base class): Parent class that is being extended.
- **subclass** (derived class): Child class that inherits from the superclass.
  - › Subclass gets a copy of every field and method from superclass.
  - › Subclass can add its own behavior, and/or change inherited behavior.

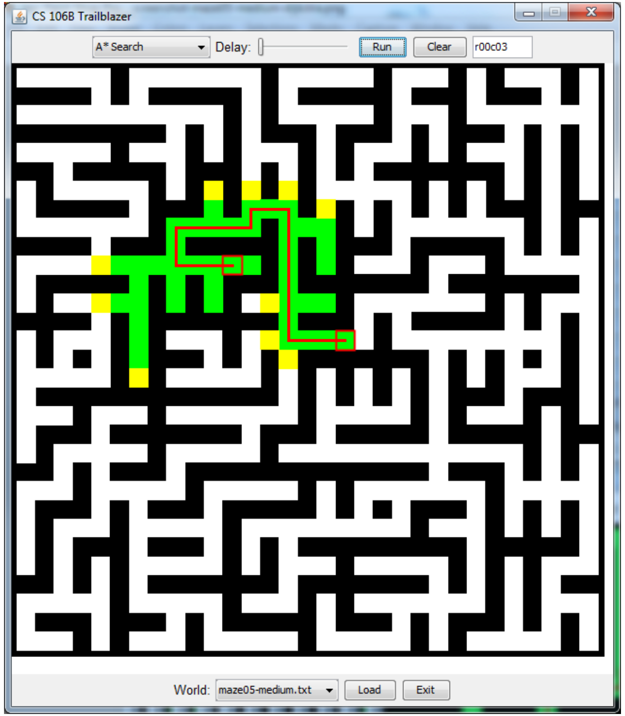


Inheritance Example

Stanford Library GObject family of classes

Behind the scenes…

- We’ve always told you not to worry about the graphics parts of your assignments.
- “Just call this BoggleGUI function…”
- Now you can go ahead and take a look!

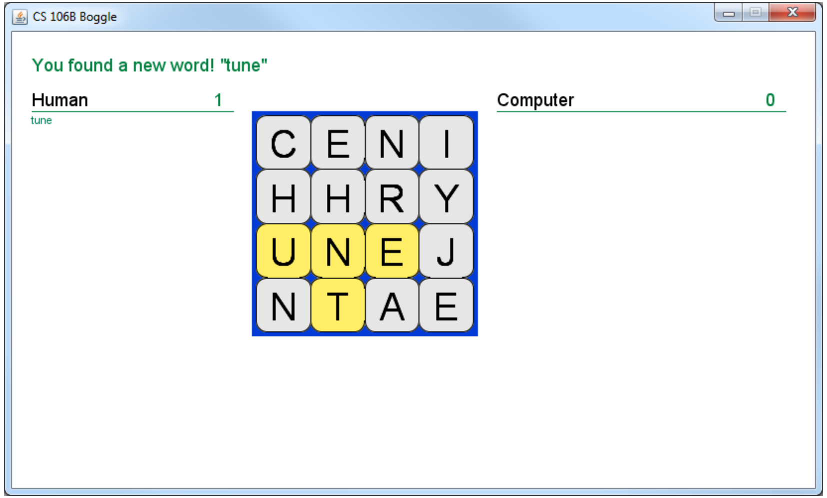

CS 106B Trailblazer

A* Search

Delay:

r00c03

World: maze05-medium.txt


CS 106B Boggle

You found a new word! "tune"

Human 1

tune

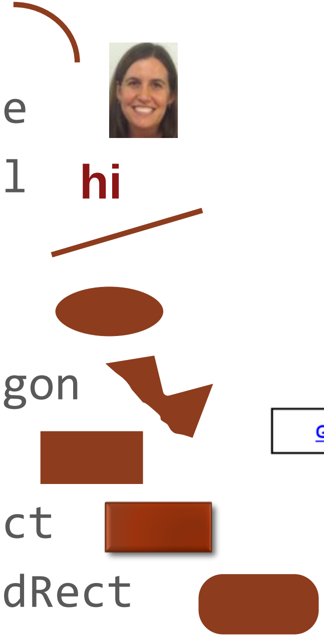
Computer 0

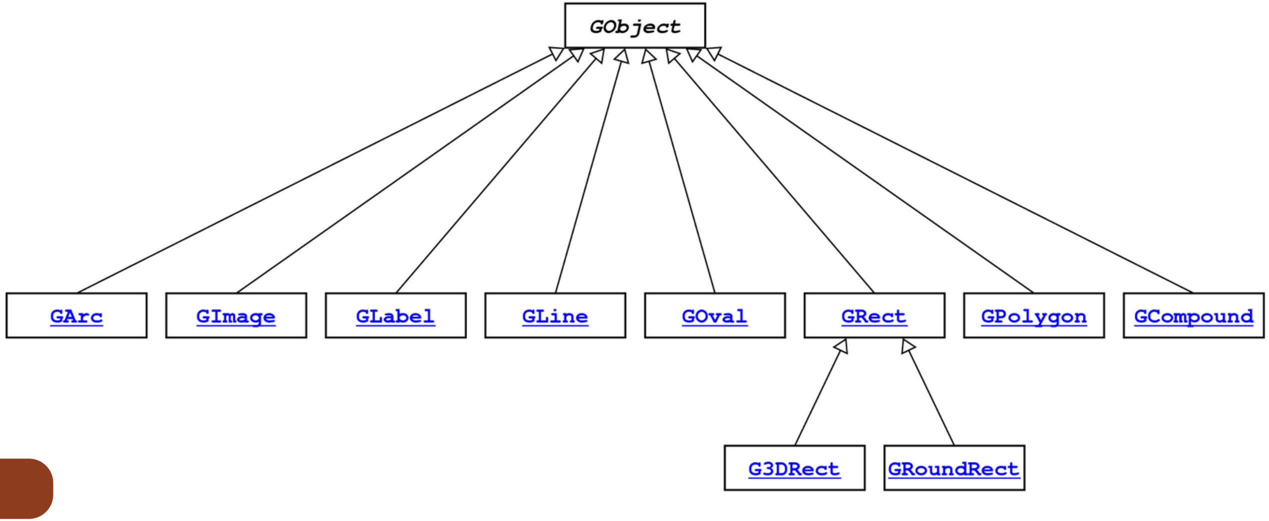
C	E	N	I
H	H	R	Y
U	N	E	J
N	T	A	E

GObject hierarchy

The Stanford C++ library contains a hierarchy of graphical objects based on a common base class named `GObject`.

- `GArc`
- `GImage`
- `GLabel`
- `GLine`
- `GOval`
- `GPolygon`
- `GRect`
- `G3DRect`
- `GRoundRect`





Stanford University

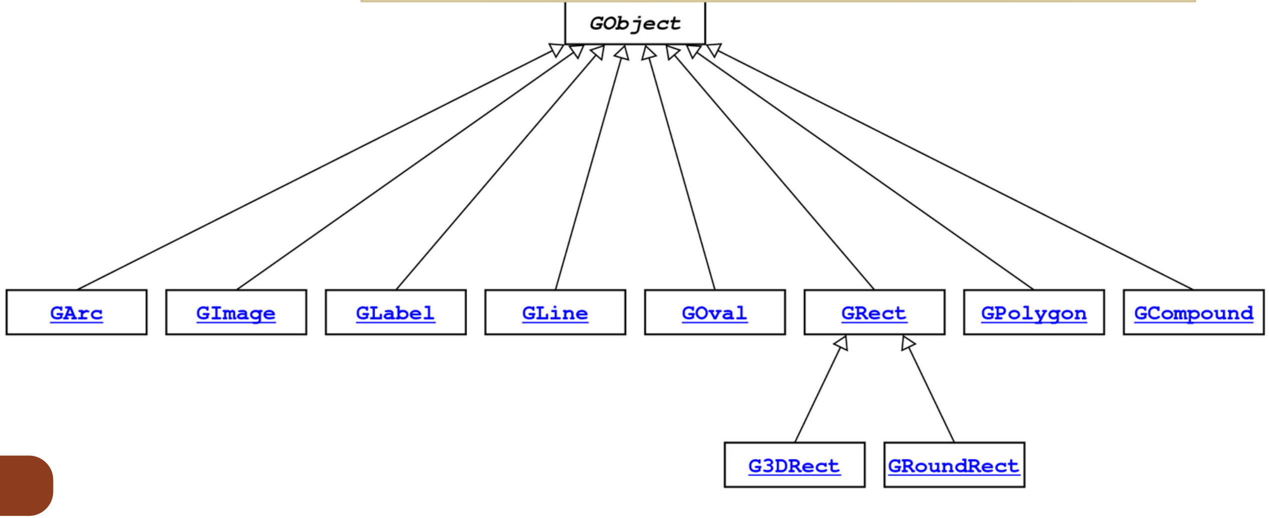
GObject hierarchy

The Stanford C++ library contains a h
objects based on a common base

Q: Rectangle is-a Polygon, right?
Why doesn’t it inherit from Polygon?

Think about it as we go through some details, and we’ll revisit the question later.

- GArc
- GImage
- GLabel **hi**
- GLine
- GOval
- GPolygon
- GRect
- G3DRect
- GRoundRect



Your turn: GObject design

How many of the following would you put in the base class (GObject), as opposed to a derived class?

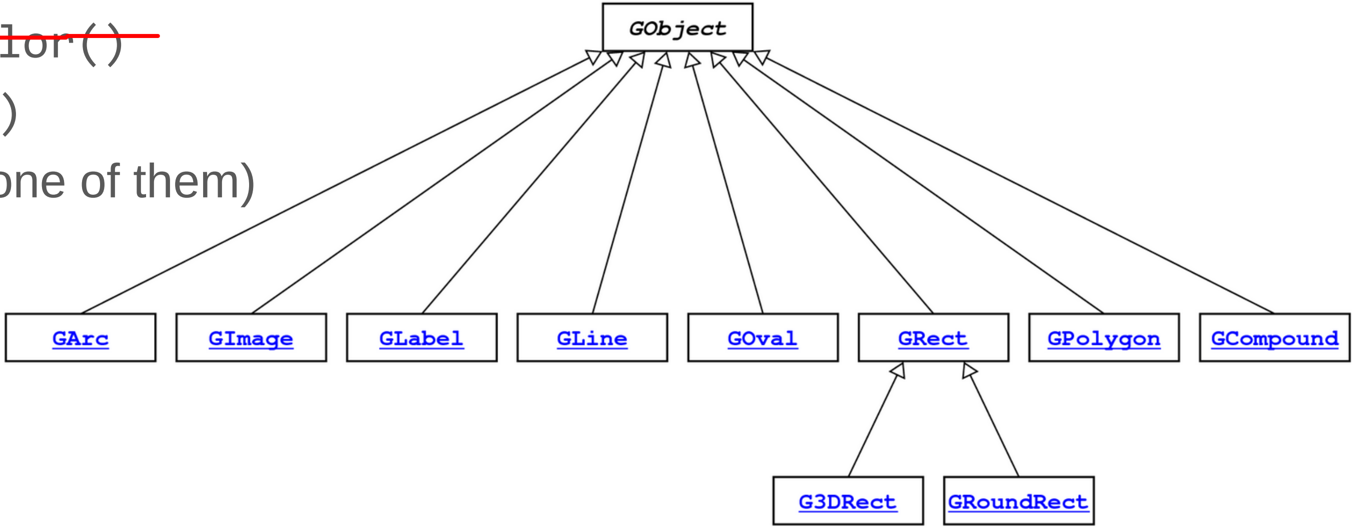
- `contains(`***x, y***`)` – returns true if (x,y) lands on the item
- ~~`setFont(`***f***`)` – sets the font for writing~~
- `setColor(`***c***`)`
- ~~`getFillColor()`~~
- `getWidth()`

A. 0-1 (none or one of them)

B. 2

[✓ C. 3]

D. 4-5 of them



GObject

GArc

GImage

GLabel

GLine

GOval

GRect

GPolygon

GCompound

G3DRect

GRoundRect

GObject members

`GObject` defines the state and behavior common to all shapes:

- contains(***x***, ***y***)
- get/setColor()
- getHeight(), getWidth()
- get/setLocation(), get/setX(), get/setY()
- move(***dx***, ***dy***)
- setVisible(***visible***)

```
double x;
double y;
double lineWidth;
std::string color;
bool visible;
```

The subclasses add state and behavior unique to them:

```
Glabel
    get/setFont
    get/setLabel

    ...

GLine
    get/setStartPoint
    get/setEndPoint

    ...

GPolygon
    addEdge
    addVertex
    get/setFillColor
    ...

GOval
    getSize
    get/setFillColor

    ...
```

GObject members

`GObject` defines the state and behavior common to all shapes:

- `contains(`***`x, y`***`)`
- `get/setColor()`
- `getHeight(), getWidth()`
- `get/setLocation(), get/setX(), get/setY()`
- `move(`***`dx, dy`***`)`
- `setVisible(`***`visible`***`)`

```
double x;
double y;
double lineWidth;
std::string color;
bool visible;
```

The subclasses add state and behavior unique to them:

```
Glabel
    get/setFont
    get/setLabel

    ...
```

```
GLine
    get/setStartPoint
    get/setEndPoint

    ...
```

```
GPolygon
    addEdge
    addVertex
    get/setFillColor
    ...
```

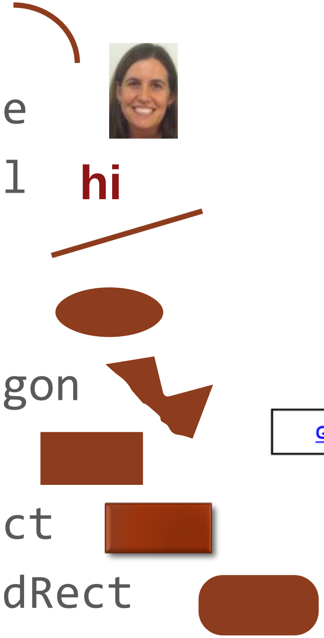
```
GOval
    getSize
    get/setFillColor

    ...
```

GObject hierarchy

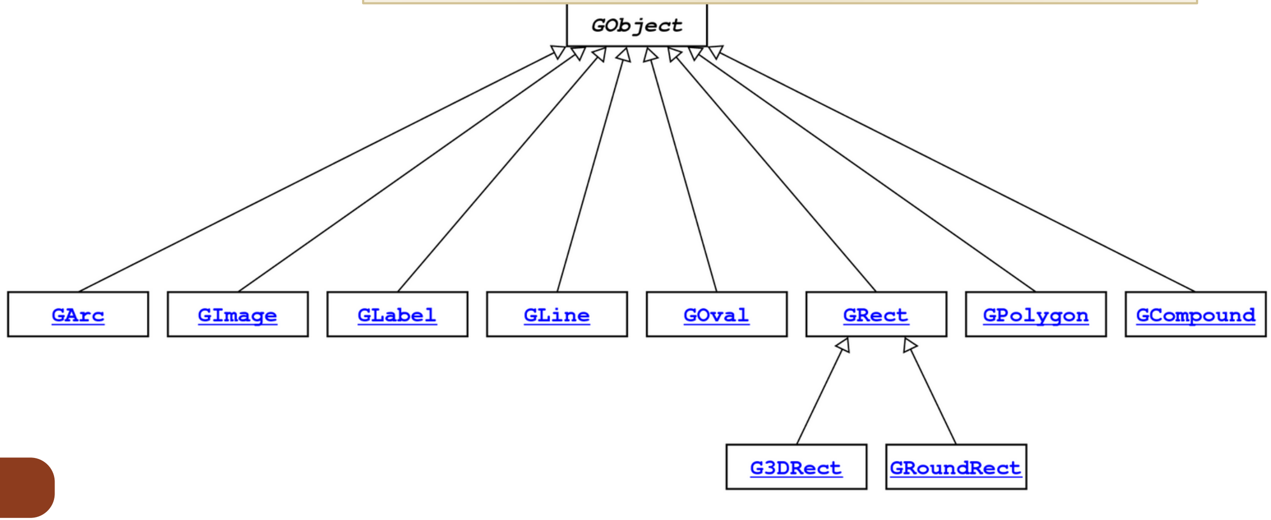
The Stanford C++ library contains a h
objects based on a common base

- GArc
- GImage
- GLabel
- GLine
- GOval
- GPolygon
- GRect
- G3DRect
- GRoundRect



Q: Rectangle is-a Polygon, right?
Why doesn't it inherit from Polygon??

Although true in geometry, they don't share many fields and methods in this case.



Inheritance Example

Your turn: let’s write an Employee family of classes

Example: Employees

Imagine a company with the following **employee regulations**:

- All employees work 40 hours / week
- Employees make $40,000 per year plus $500 for each year worked
  - Except for lawyers who get twice the usual pay, and programmers who get the same $40k base but $2000 for each year worked
- Employees have 2 weeks of paid vacation days per year
  - Except for programmers who get an extra week (a total of 3)

Each type of employee has some unique behavior:

- **Lawyers** know how to sue
- **Programmers** know how to write code
- **IT** person knows how to fix PCs
- **Network IT** person knows how to fix PCs and how fix the network

Employee class

```
// Employee.h
class Employee {
public:
    Employee(string name,
             int years);
    virtual int hours();
    virtual string name();
    virtual double salary();
    virtual int vacationDays();
    virtual int years();

private:
    string m_name;
    int m_years;
};
```

```
// Employee.cpp
Employee::Employee(string name, int years) {
    m_name = name;
    m_years = years;
}

int Employee::hours() {
    return 40;
}

string Employee::name() {
    return m_name;
}

double Employee::salary() {
    return 40000.0 + (500 * m_years);
}

int Employee::vacationDays() {
    return 10;
}

int Employee::years() {
    return m_years;
}
```

Exercise: Employees

Exercise: Implement classes `Lawyer` and `Programmer`.

- A Lawyer remembers what **law school** he/she went to.
- Lawyers make twice as much **salary** as normal employees.
- Lawyers know how to **sue** people (unique behavior).
- Lawyers put “, Esq.” at the end of their name.

- Programmers make the same base salary as normal employees, but they earn a **bonus of $2k/year** instead of $500/year.
- Programmers know how to write **code** (unique behavior).

Inheritance syntax

```
class Name : public SuperclassName {
```

- Example:

```
class Lawyer : public Employee {
    ...
};
```

By extending `Employee`, each `Lawyer` object now:

- receives a `hours`, `name`, `salary`, `vacationDays`, and `years` method automatically
- can be treated as an `Employee` by client code (see this next class!)

Call superclass c'tor

```
SubclassName::SubclassName(params)
         : SuperclassName(params) {
    statements;
}
```

To call a superclass constructor from subclass constructor, use an *initialization list*, with a colon after the constructor declaration.

- Example:

```
Lawyer::Lawyer(string name, string lawSchool, int years)
       : Employee(name, years) {
    // calls Employee constructor first
    m_lawSchool = lawSchool;
}
```

Your turn: inheritance

```
string Lawyer::name() {
    ???
}
```

For adding “, Esq.” to the name, which of the following could work?

A. `return m_name + “, Esq.”;`

B. `return name() + “, Esq.”;`

C. `return Employee::name() + “, Esq.”;`

D. None of the above

E. More than one of the above

```
// Employee.h
class Employee {
public:
    Employee(string name,
             int years);
    int hours();
    string name();
    double salary();
    int vacationDays();
    string vacationForm();
    int years();

private:
    string m_name;
    int m_years;
};
```

Call superclass member

```
SuperclassName::memberName(params)
```

To call a superclass overridden member from subclass member.

- Example:

```
double Lawyer::salary() {          // paid twice as much
    return Employee::salary() * 2;
}
```

- **Note: Subclass *cannot* access private members of the superclass.**
- Note: You only need to use this syntax when the superclass's member has been overridden.
  - If you just want to call one member from another, even if that member came from the superclass, you don't need to write `Superclass::` .