Programming Abstractions
CS 106B
Cynthia Lee
Inheritance Topics

Inheritance

- The basics
  - Example: Stanford GObject class
- Friday: Polymorphism
Inheritance

Why? What? How?
Inheritance: why?

- Remember the #1 rule of software engineers:
  - **Software engineers 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: why?

Let’s say you really like Stanford library Grid, but you really need it to do just one extra thing that it doesn’t currently do

- In your business, you often need to have a Grid and then periodically clear out all the values that are on the diagonal.
- The type of the values stored in the Grids varies, and I want this clearing of the diagonal code to work for any Grid, regardless of what type it is storing.
- What I’d really like to do is add one more function to the Grid class, but I can’t edit the Stanford libraries…
  1. Could just copy & paste all the Grid class code and then make my edits.
  2. Could write my own from scratch.
Inheritance: what is it?

Inheritance is a way of saying that I want to make a new class, which will “inherit” (re-use) everything some existing class does, plus add some additional data and/or functions.

- I would make a MyBusinessGrid that would *inherit* from Stanford library Grid.
- I get everything Stanford library Grid already does for “free”
- I can add additional customizations that I need.
Inheritance: what is it?

**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

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

- Re-use!
Inheritance vocab

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

- **subclass** (derived class): Child class that inherits from the superclass.
Inheritance Example

Stanford Library G-Object 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!
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
**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)`

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`
  - ...

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

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

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

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

Each type of employee has some unique behavior:

- **Lawyers** know how to sue
- **Programmers** know how to write code
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 which law school they 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 when printing 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
Call superclass c'tor

```cpp
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:

```cpp
Lawyer::Lawyer(string name, string lawSchool, int years)
    : Employee(name, years) {
    // calls Employee constructor first
    // then does Lawyer-specific constructor stuff
    m_lawSchool = lawSchool;
}
```
Your turn: inheritance

```cpp
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:

  ```cpp
  string Lawyer::name() {
    // add Esq.
    return Employee::name() + ", Esq."
  }
  ```

- 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::.