CSU NUPAC Tutorials
2019

Gabriel Palacino

As usual I’m borrowing a lot of the material from Sam Meehan (University of Washington)
Inheritance - Basic

- Things are related in real life
  - Example 1: Child gets to use parents’ car - the car is a “shared” thing between the parent/child
  - Example 2: Specific types of shapes all have some common/shared attributes (e.g. area)
- Goal: Relate the common/shared information and functionality to simplify life
Inheritance - Basic

- Things are related in real life
  - Example 1: Child gets to use parents’ car - the car is a “shared” thing between the parent/child
  - Example 2: Specific types of shapes all have some common/shared attributes (e.g. area)
- Goal: Relate the common/shared information and functionality to simplify life

Shape

- width
- height

area = width*height

area = ½ * width*height

area = (height*width*widthRatio) + ½ * height*(1-widthRatio)*width

add “widthRatio” attribute

width*widthRatio
Inheritance - Basic

- Achieve this via “inheritance” of classes
  - **Base** class: “Shape” → **Derived** classes: “Rectangle”, “Triangle”, “Trapezoid”
  - The derived classes inherit the members and methods from the base class

```plaintext
Shape
  width
  height
  void setDimensions(a, b)

Rectangle
  int getArea()  

Triangle
  int getArea()  

Trapezoid
  int getArea()
  void setRatio()
```
Inheritance at work

- How does this look like for one of the specific shapes?
- Start with a basic class: **Shape**
  - This will be our base class
Inheritance at work

- How does this look like for one of the specific shapes?
- Start with a basic class: **Shape**
  - This will be our base class
- Can store/retrieve information
Inheritance at work

- Creating a derived class requires
  - Telling it who its “base” class is
  - Defining functions according to their “namespace”
Inheritance at work

- Creating a derived class requires
  - Telling it who its "base" class is
  - Defining functions according to their "namespace"

This is the base class that we inherit from

This is the new functionality of our derived class

Do functions belong to the "Shape" or the "Rectangle" class?
Inheritance at work - almost

- Access specifier interlude: protected
  - Allows members to be accessed/modified by derived class
  - Example: Daughter gets to use the parents’ car ➔ it is protected by the parent but she derives from them
- Good habit: make members of classes “protected” and access them via helper funcs
Inheritance at work

- Must specify the base class or origin
  - The derived class starts with this functionality
- Can additionally define other members and functionality (e.g. `getArea()`)

```cpp
// my_class::source code

#include "Shape.h"

int main()
{
    // shape members
    Shape myShape;
    myShape.setDimensions(3,4);
    myShape.getShapeDescription();

    // rectangle members
    Rectangle myRectangle;
    myRectangle.setDimensions(6,7);
    myRectangle.getRectangleDescription();
    std::cout << "Area: " << myRectangle.getArea() << std::endl;

    // other members
    std::cout << "Shape : width" << width << height;
    std::cout << "Rectangle : width" << width << height;
    std::cout << "Rectangle : width" << width << height;
}
```
Inheritance at work

- Must specify the base class or origin
  - The derived class starts with this functionality
- Can additionally define other members and functionality (e.g. `getArea()`)
Multiple inherited classes

- This process can be expanded multiple times
  - One-to-many correspondence between base → derived classes
Multiple inherited classes

- This process can be expanded multiple times
  - One-to-many correspondence between base → derived classes
  - Think of derived classes as “specific types of” the base class
- If the (your) code is well-documented, the interface is all you (they) need to know
  - Parameterizing THEIR ignorance
Inheritance - Redefining

- It would be nice to not have to rename functions that do the same thing
  - In this example: `getShapeDescription()` vs. `getRectangleDescription()` vs. `getTriangleDescription()` ...
- This is achieved via the concept of **polymorphism** of functions
  - This is achieved by the use of the “**virtual**” function definition modifier

### Unique Function Definitions

#### Headers

```cpp
// Headers with Unique Function Definitions

``` javascript

### With Polymorphism

#### Headers

```cpp
// Headers with Polymorphism

``` javascript

- Allows this function to be redefined in a derived class
- Two functions that will do different things depending on class
Polymorphism

- The context (and therefore the specific definition) is determined by the namespace
  - Is `getDescription()` coming from a “Shape” or a “Rectangle”?  

Unique Function Definitions

With Polymorphism

Comparing: Source code
Polymorphism

- The same function called from two different objects carries out different tasks
  - If you are confused about what function is being called: PRINT STUFF TO THE SCREEN!
Inheritance Trees

- We can graphically represent inheritance via a “tree” structure.
- This is a very good tool (again, with pen and paper) to layout your thought process when beginning a project.

A Simple Inheritance Structure

A Multi-layered Inheritance Structure
Inheritance Trees

- We can graphically represent inheritance via a “tree” structure.
- This is a very good tool (again, with pen and paper) to layout your thought process when beginning a project.

A Very Complex Inheritance Structure
Part 1 - reminder

• Create two classes
  ‣ A 3D vector class (euclidean) with functions to access its members and its norm
  ‣ A Lorentz four-vector class with functions to access its members and its norm using the $\eta(+- - -)$ metric with functions
  ‣ Implement constructors and destructors.

Part 2

• Create a derive class from `Interaction`
  ‣ Use it to describe the interaction between two objects of class `Particle` in `main()`
  ‣ Implement a function to calculate the distance between two points using the 3D vectors and use this in conjunction with your interaction.
    ‣ Can this functionality be implemented as an operation of the class?

Due midnight of Tuesday, February 19