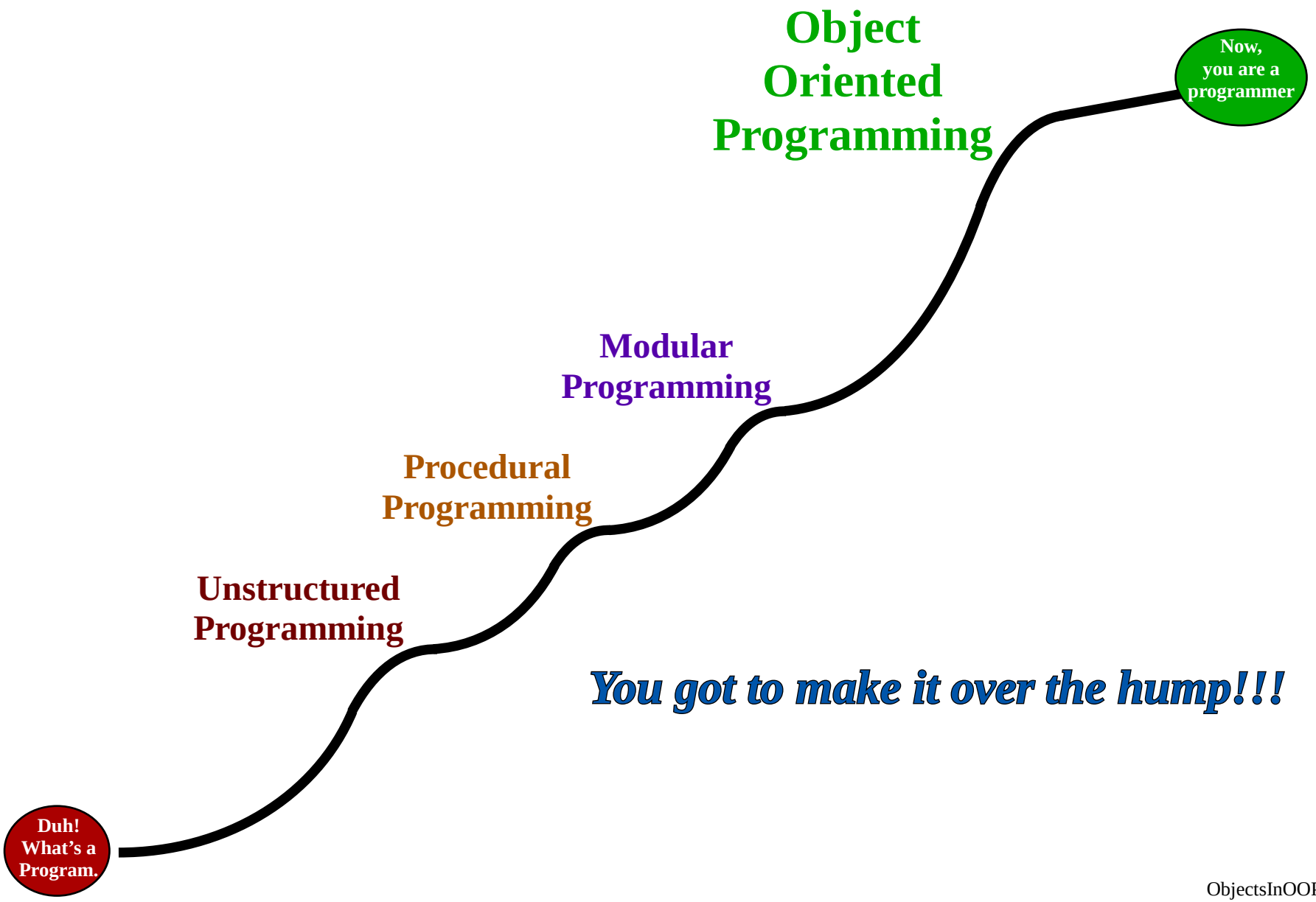
Programming Learning Curve



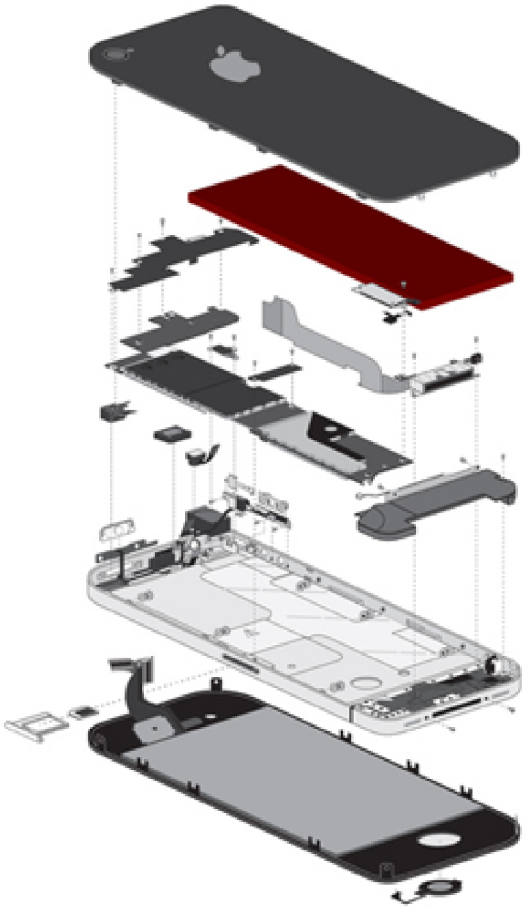
You got to make it over the hump!!!


Objects?

Objects?

1. What states can this object be in?
2. What behaviors can this object perform?
3. What other objects might this object contain?
4. How do you interface with this object?

An iPod is an object.



What is an object as it relates to Software Engineering?

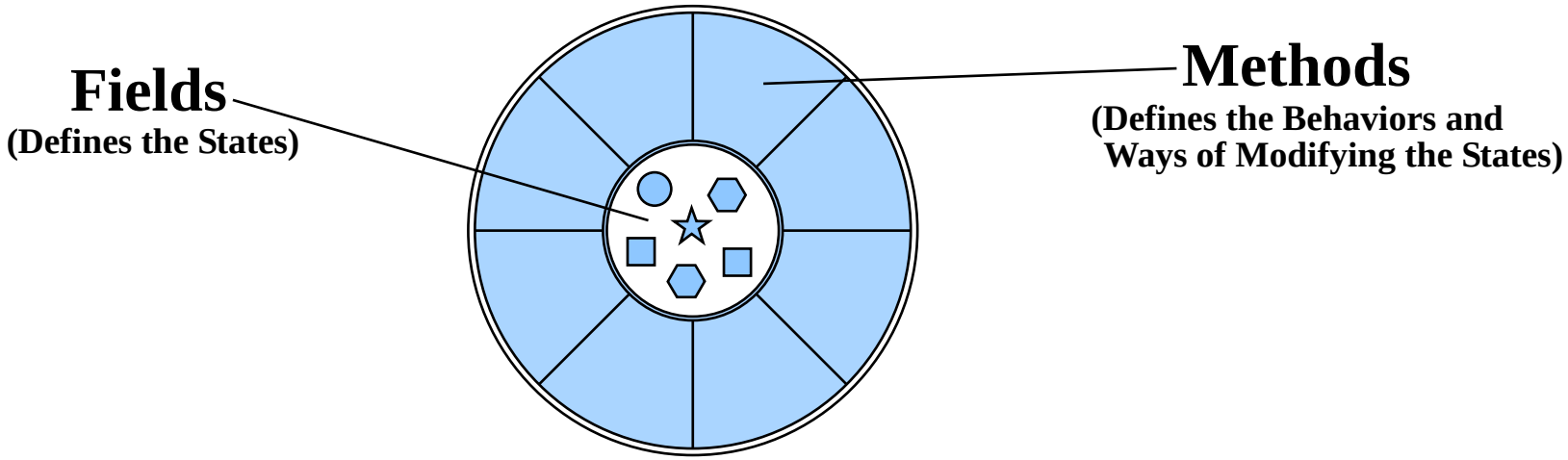
Object - **An object is a software "bundle" consisting of:**

1. **A set of variables which define the states the object can exist in.**
2. **A set of functions that define the behavior of that object.**

Software "objects" are often used to model the real-world objects that you find in everyday life.

Characteristics of an Object in a Program

- State
- Defined Behaviors
- Defined ways of modifying the State

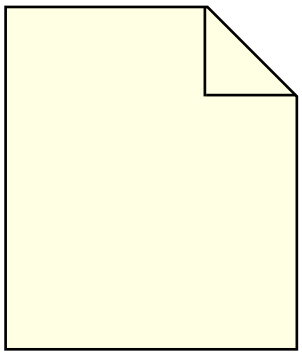


In C++ Object Oriented Programming we use structs and classes to represent objects in a program.

Structures

The most basic of objects - A "can" of variables

```
struct Player
{
        string playerName;
        PlayerClass playerClass;
        PlayerRace playerRace;
        PlayerAlignment playerAlignment;
        int hitPoints;
        int playerTraits[6];        // Strength (0), Dexterity (1), Constitution (2),
                                    // Intelligence (3), Wisdom (4), Charisma (5)
        double weightCarried;
        double maxWeightToCarry;
        int itemsCarried;           // Flags for items carried
};
```



PlayerActions.cpp

```
Player p1;
Player *p2 = new Player();
BuildPlayer(&p1);
BuildPlayer(p2);
cout << p1.playerName << " and " <<
            p2->playerName << " are read to play."
```

BTW: enum data types PlayerClass, PlayerRace, PlayerAlignment have already been defined.

Structures

Nested Structures

```
struct Item
{
        string itemName;
        string description;
        double value;
        double weight;
};
```

```
struct Item
{
        string itemName;
        string description;
        double value;
        double weight;
};
```

```
struct Player
{
        string playerName;
        PlayerClass playerClass;
        PlayerRace playerRace;
        PlayerAlignment playerAlignment;
        int hitPoints;
        int playerTraits[6];   // Strength (0), Dexterity (1), Constitution (2),
                               // Intelligence (3), Wisdom (4), Charisma (5)
        double weightCarried;
        double maxWeightToCarry;
        Item itemsCarried[24];       // Array of items carried
        Location curLocation;  // What room is this player in
};
```

Stupid Structures

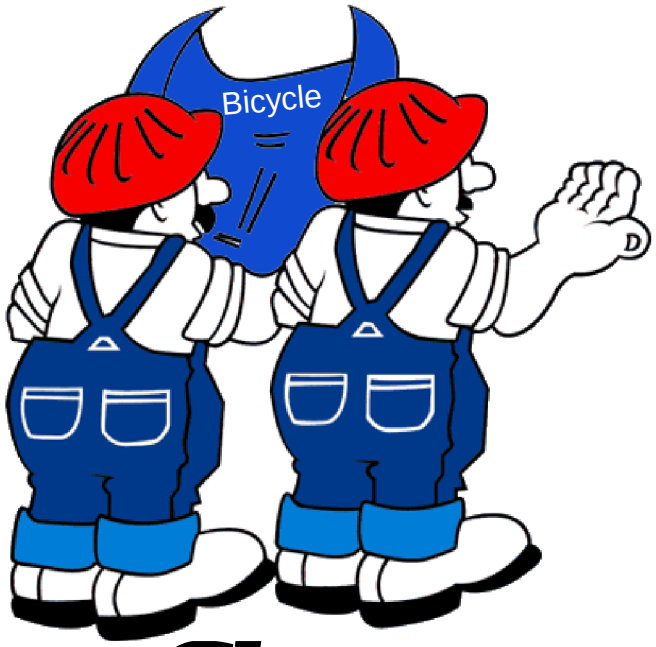
```
struct stupid
{
    int x;
    char ch;
    void init(int X, char c);
    int getX();
    void setX(int x);
    char getChar();
    void setChar(char c);
};
void stupid::init(int X, char c)
{
   x = X;
   ch = c;
};
```

...and so on for all the other functions some idiot put in this structure.

Classes

Instances

The objects created while the program is running.

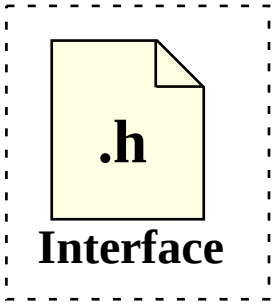


Class

The "Blueprint" for building objects.

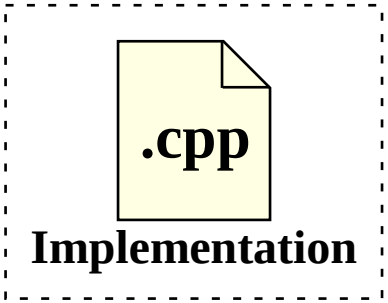
```
Bicycle b1(YELLOW, RACING);
Bicycle b2(ORANGE_FIRE, KIDS | TRAINING_WHEELS);
Bicycle *bpt1 = new Bicycle(MAGENTA, LADIES | BASKET);
Bicycle *bpt2 = new Bicycle(PINK, GIRLS | BELL);
```

Classes - Interface




```
#pragma once        // Tell the preprocessor to execute this header only once
#include “GameInfo.h”
class Player
{
        private:
                string playerName;
                PlayerClass playerClass;
                PlayerRace playerRace;
                PlayerAlignment playerAlignment;
                int hitPoints;
                int playerTraits[6];    // Strength (0), Dexterity (1), Constitution (2),
                                        // Intelligence (3), Wisdom (4), Charisma (5)
                double weightCarried;
                double maxWeightToCarry;
                vector<Item> itemsCarried;          // Items carried
                Location curLocation;               // What room is this player in
        public:
                Player();       // Constructor
                ~Player();      // Destructor
        // get/set functions for each member variable are defined here
                bool movePlayer(int row, int column, int level); // Handle moving player
                int fight(Player *opponent);        // Handle combat events
                bool takeItem(Item *item);          // Pick up an item and carry it
                Item *dropItem(int itemID);         // Drop an item
};
```

Classes - Implementation



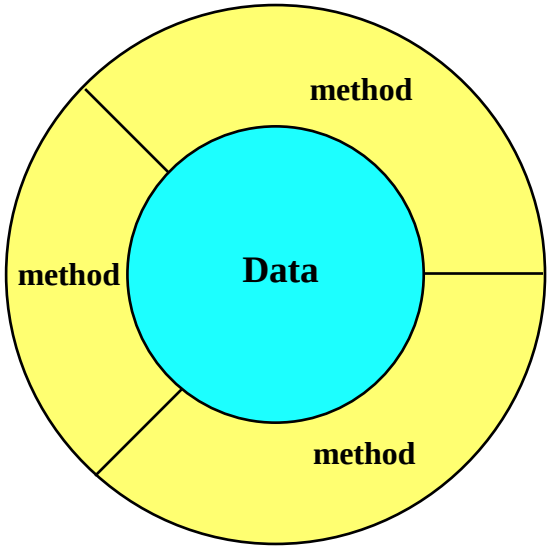

```
// Implementation: Player.cpp
#include “Player.h”

Player::Player()
{
    // Set default values for all member vars.
}

Player::~Player()
{
   // Clean up before destroying instance
}

bool Player::movePlayer(int row, int column,
                        int level)
{
    // Move player to new location
}

int Player::fight(Player *opponent)
{
     // Handle combat events
}
```

```
bool Player::takeItem(Item *item)
{
    // Pick up an item and carry it
}

Item * Player::dropItem(int itemID)
{
    // Drop an item
}


// get/set functions for each member
//  variable are defined here
```

Important OO Terms

Encapsulation

A Class encapsulates data and methods into a single entity.

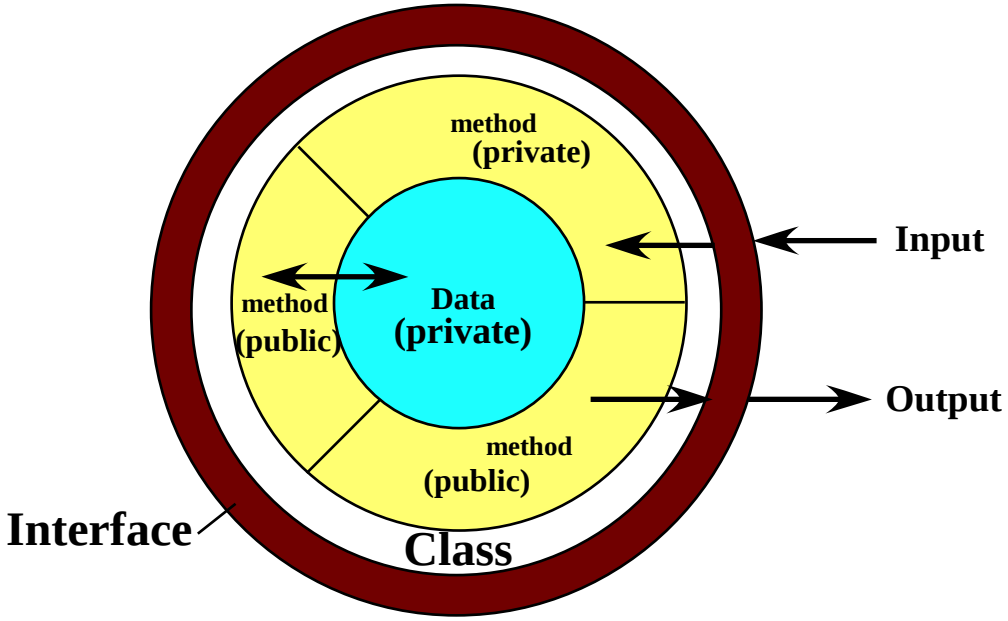


Class

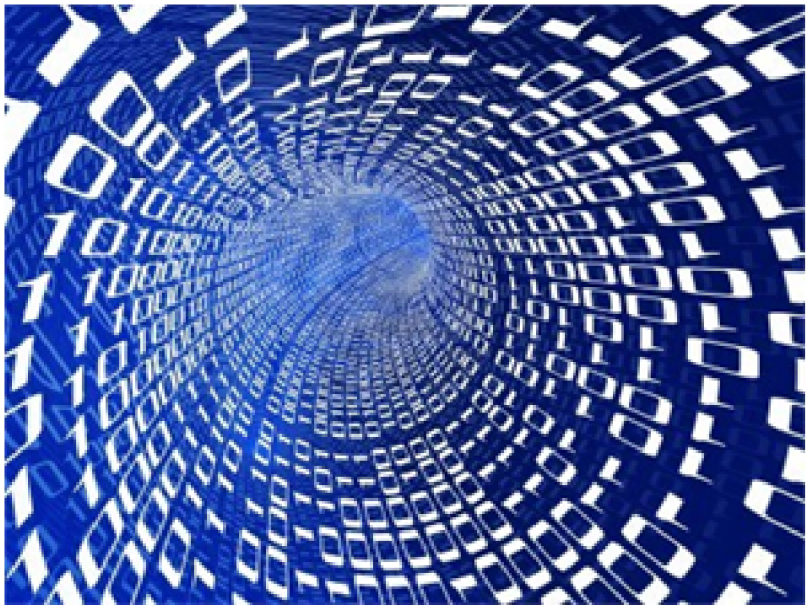
Important OO Terms

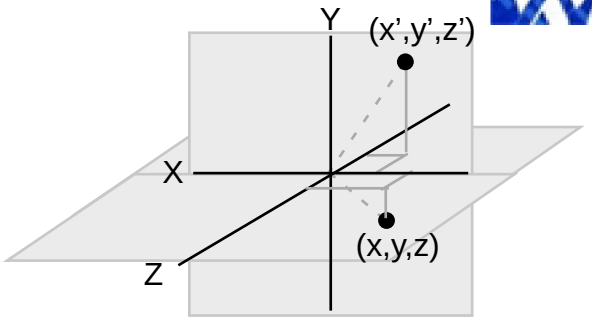
Data Hiding

A Class interface allows the internal data and methods to be hidden. It controls outside access.



Important OO Terms
Data Abstraction





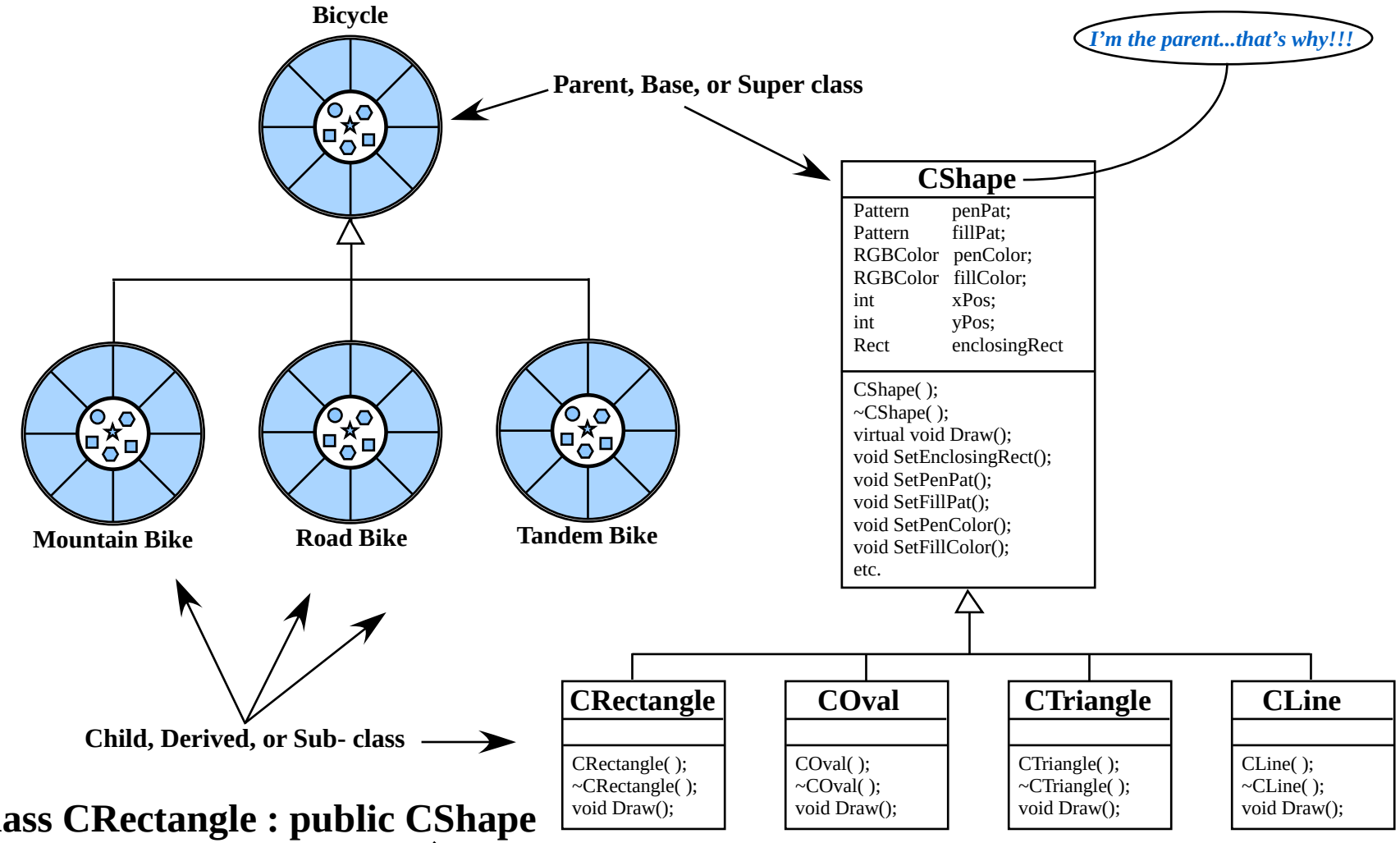
Given point (x,y,z) rotate and translate it a given amount about X, Y, and Z axes.

RotatePoint3D()

point(x,y,z) → **?** → **point(x',y',z')**

How does this happen?

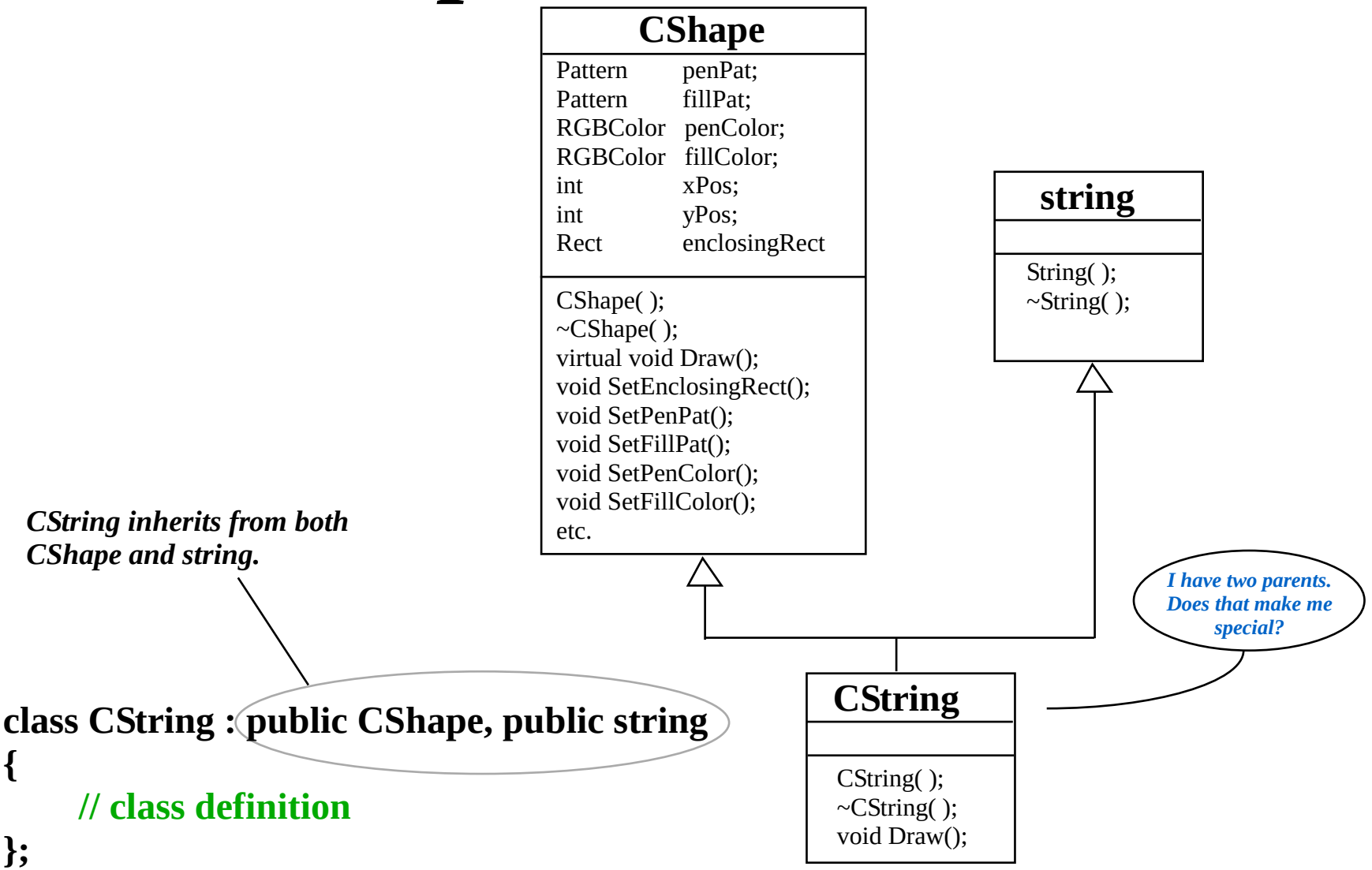
Important OO Terms
Inheritance




```
class CRectangle : public CShape
{
      // class definition
};
```

CRectangle inherits all public and protected items from CShape.

Important OO Terms
Multiple Inheritance

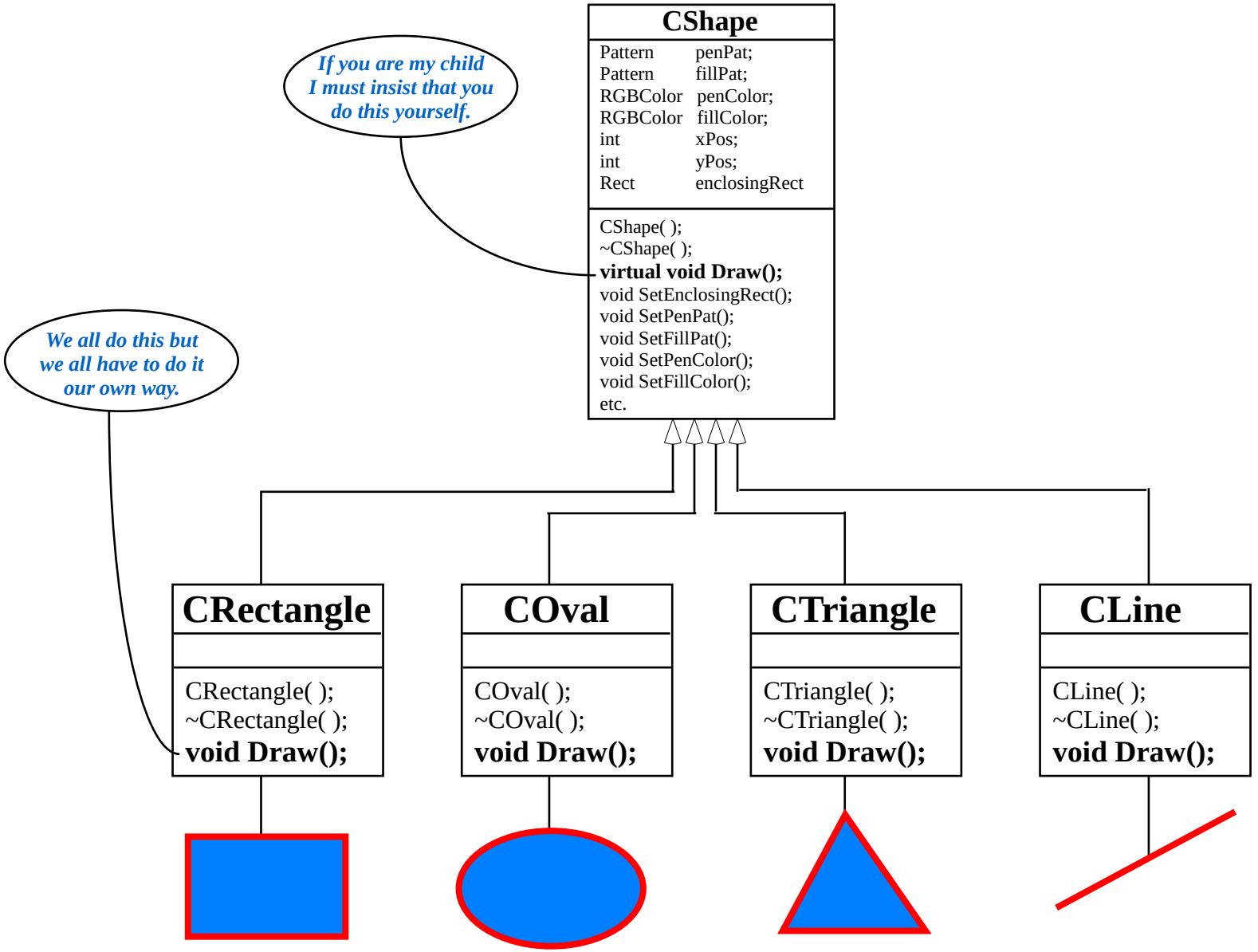


Important OO Terms
Multiple Inheritance

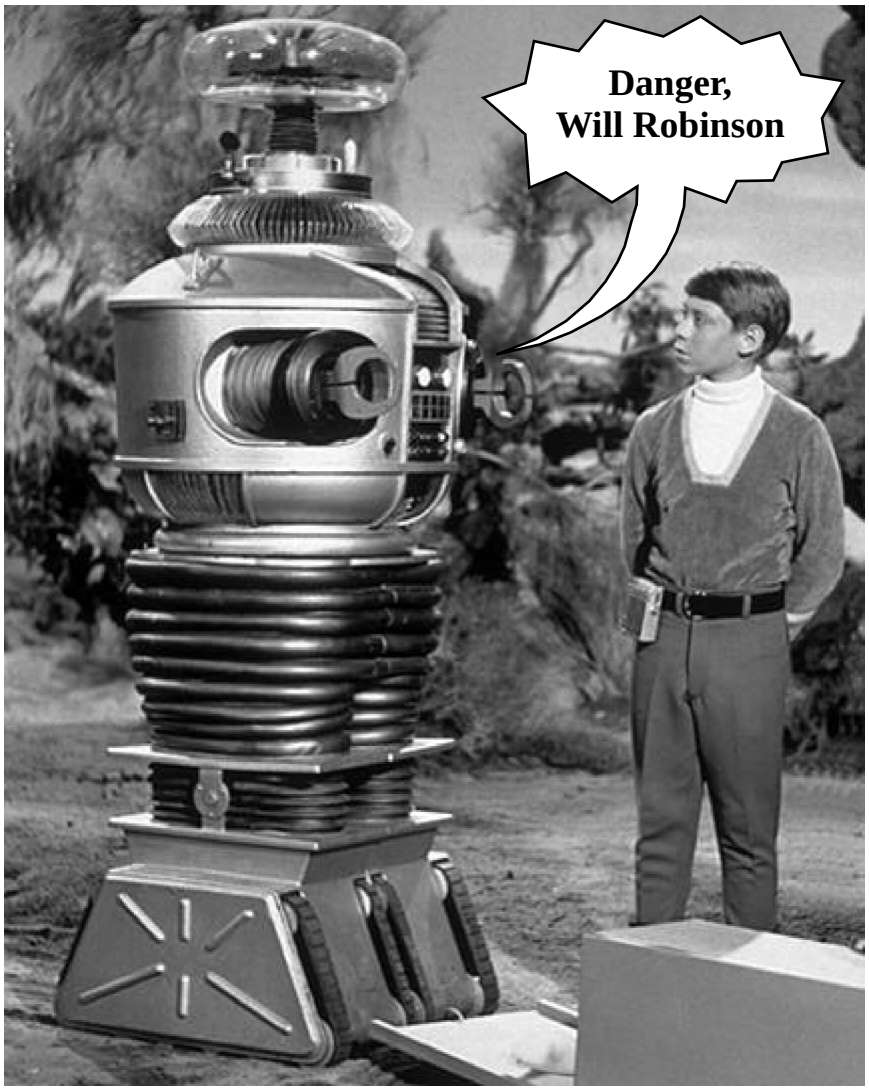
Multiple Inheritance does NOT mean

- ✗ **Multiple classes share the same superclass.**
- ✗ **A class inherits from a class that, itself is a subclass of another class higher up in the hierarchy.**

Important OO Terms
Polymorphism



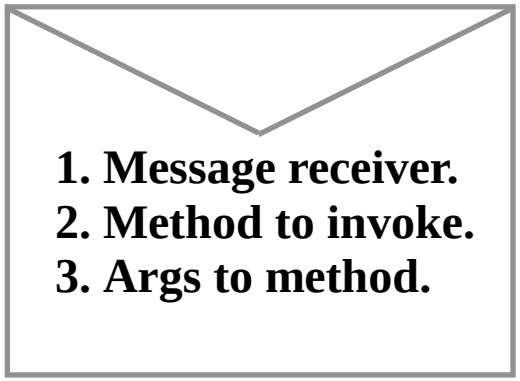
Important OO Terms
Polymorphism



- **If you fail to declare a function in the parent class as virtual.**
- **If there is a subclass specific action in the destructor.**

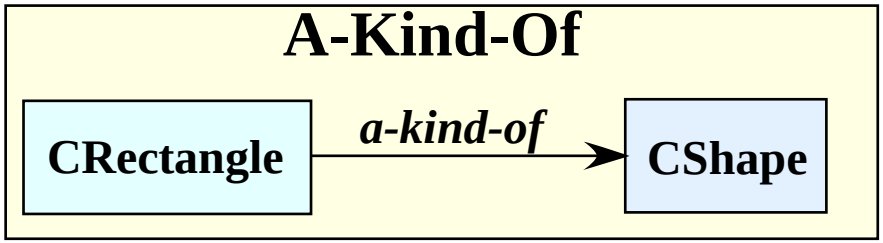
Important OO Concepts
Messages

Important OO Concepts
Messages

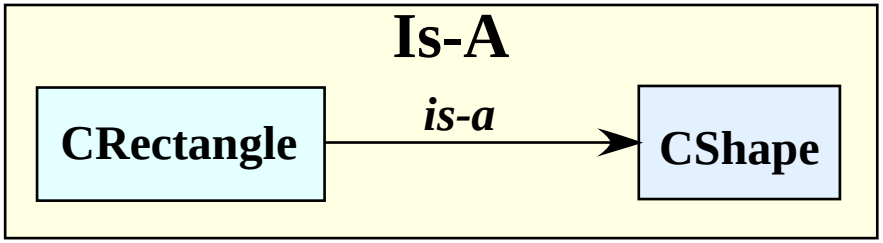


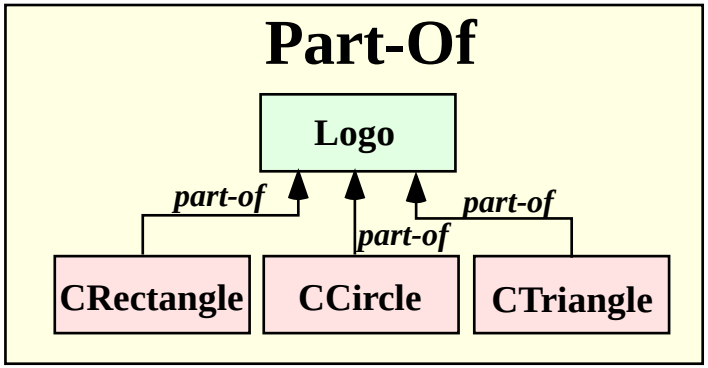
What goes in a message object?

Important OO Concepts
Object Relationships

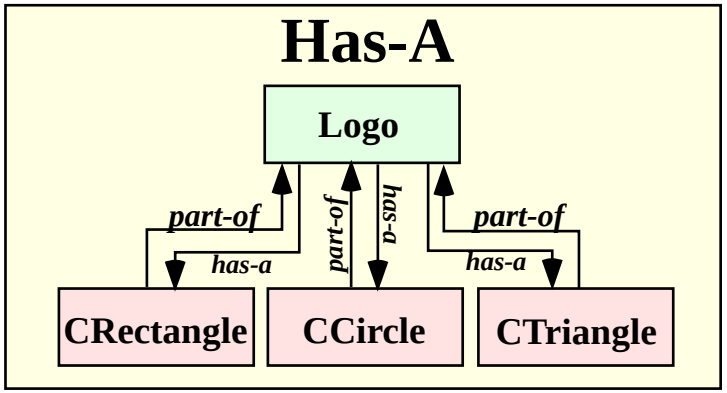


These can be thought of as two ways to describe the same relationship.





These can be thought of as describing different relationships between the same objects.

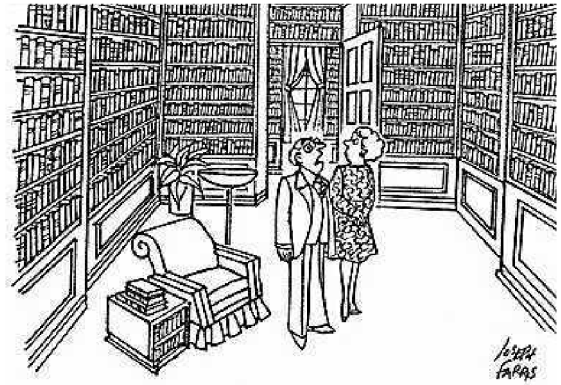


Value of Bundling Programs into Objects

Modularity

What all can you build from these?

Information-hiding



"I hid our Swiss account in one of these books and I can't remember which!"

Code re-use

Pluggability and debugging ease

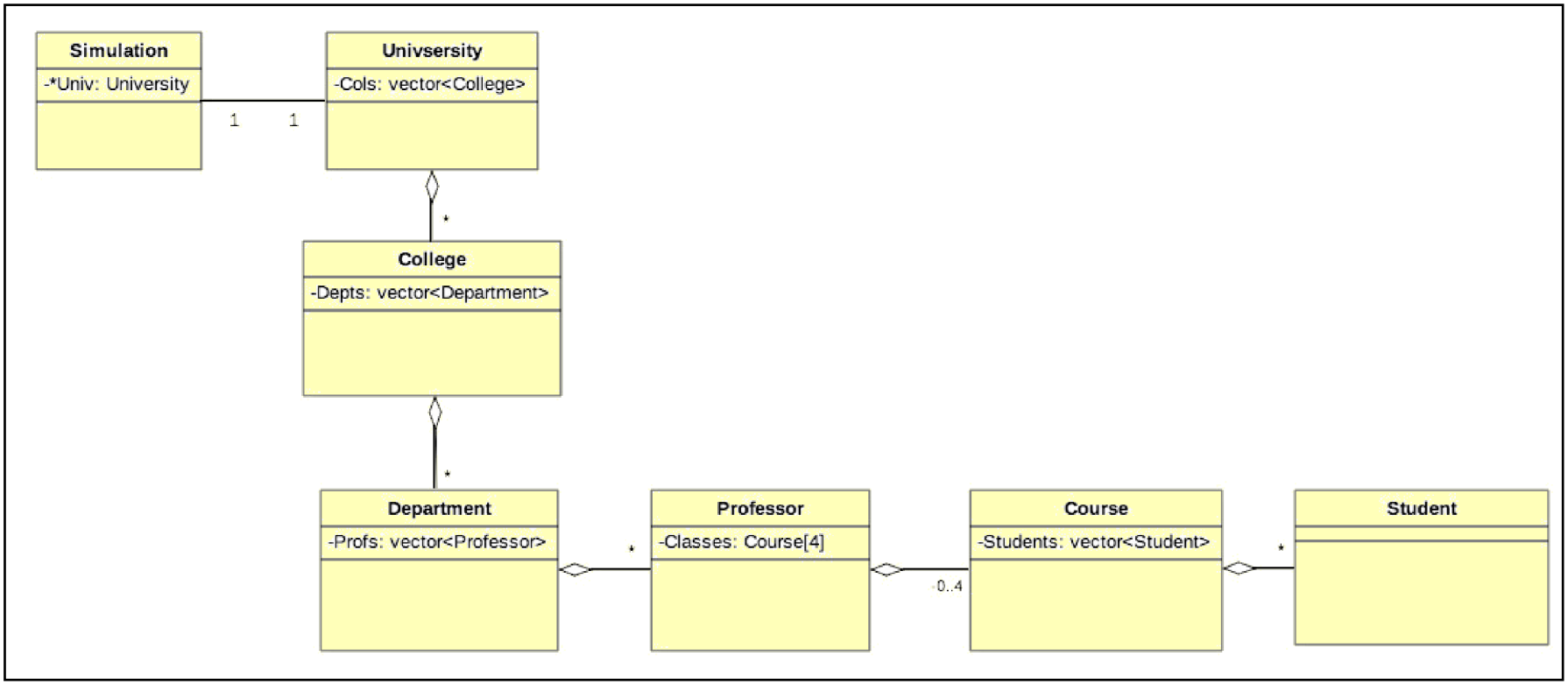
University Simulation Program

Suppose you got the following programming assignment:

> **Write an Object Oriented program that will simulate the organization of a University. The University may contain any number of Colleges. Each College may contain any number of Departments. Each Department may contain any number of Professors. Each Professor may teach up to four Courses. Each Course may contain any number of Students.**

This is NOT your first programming assignment!!!

UML Diagram of this Simulation



Classes in the simulation:

Simulation - In an MVC organization this is the Controller. It has a one-to-one relationship to University.
University - Has a one-to-many Aggregation relationship to College
College - Has a one-to-many Aggregation relationship to Department
Department - Has a one-to-many Aggregation relationship to Professor
Professor - Has a one-to-four Aggregation relationship to Course
Course - Has a one-to-many Aggregation relationship to Student
Student - Represents one student

Simulation

```
void main()
{
        Simulation *theSim = new Simulation();
        theSim -> runSimulation();
}
```

The only action taken in main() is to create an instance of Simulation then pass control to it.

One instance of class Simulation

Responsibilities:

Ask the user for the name of the data file defining the university.
Create an instance of University to represent the university passing it the name of the data file.
Perform any other initializations that are not the responsibility of other classes.

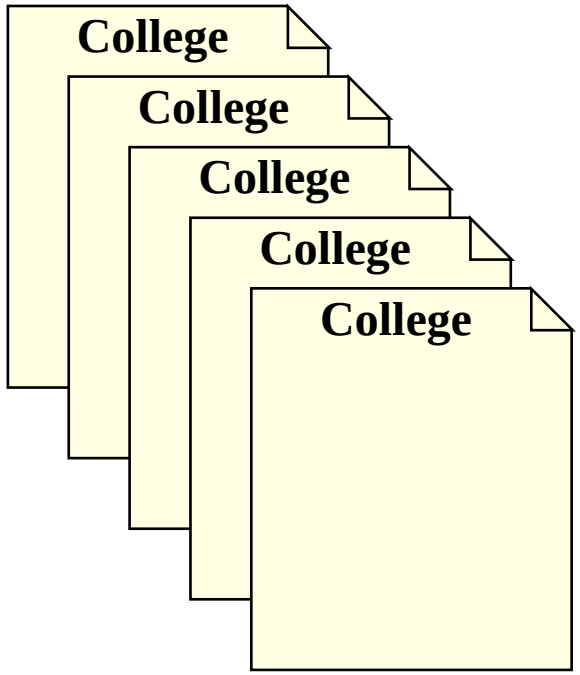
```
void Simulation::runSimulation()
{
        char fileName[64];
        // Get name of the data file
        University *theU = new University(fileName);

        // Code goes here to run the simulation

}
```

University

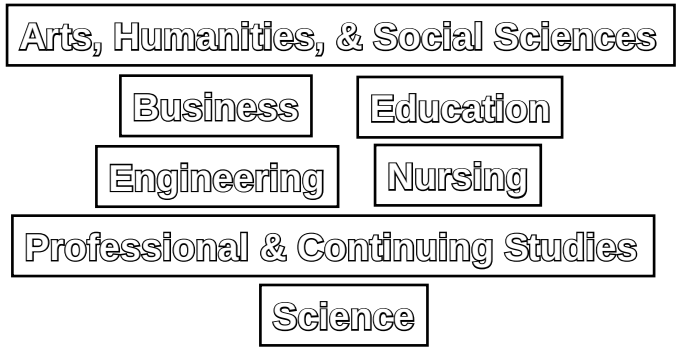
One instance of class University

```
University::University(char *dataFile)
{
        // Open the data file
        fstream *inFile = new fstream();
        inFile.open(dataFile, ios::in);
        if(!inFile.good())
        {
                exit();  // failed to open file, terminate.
        }
        // Read all data defining the university,
        // i.e. name, location, number of colleges, etc.
        // Create all instances of College as defined
        //      in the data file
        for(int i=0; i< numColleges; i++)
        {
                College *col = new College(inFile);
                // Fill in the data on this College
                m_vColleges.push_back(*col);
        }
        // Other initializations as needed here
}
```

This class, when instantiated represents one university as defined in the data file.



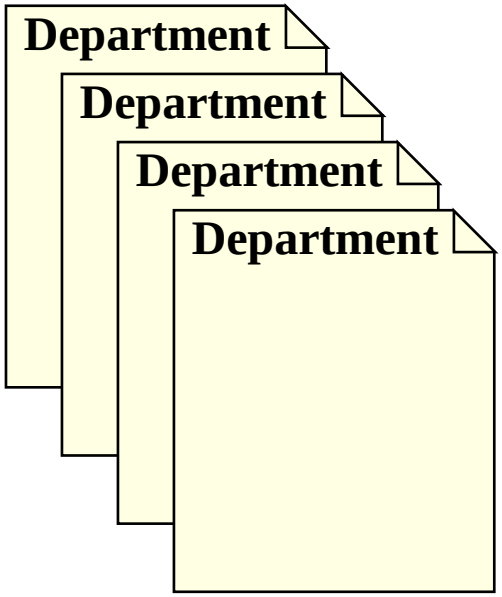
A separate instance of College is created for each college in the University.



College

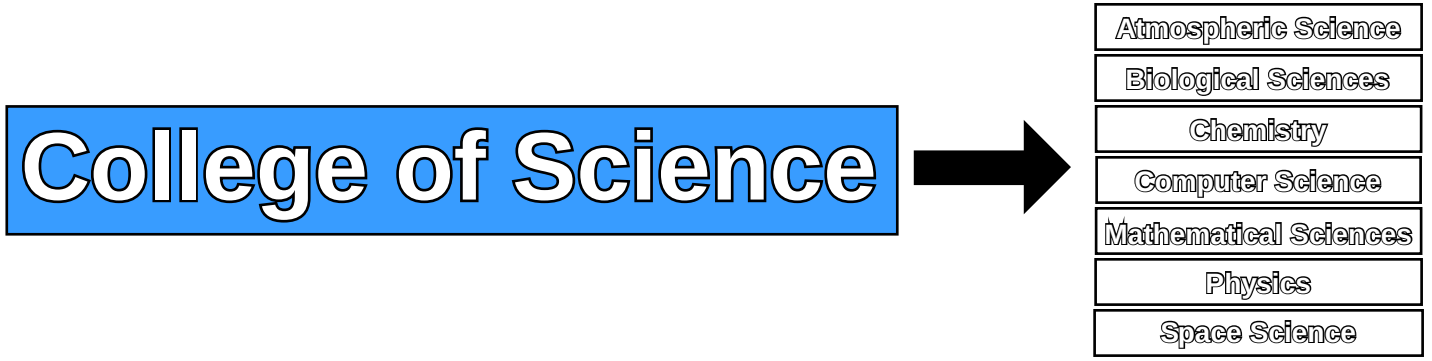
One instance of class College

```
College::College(fstream *inputFile)
{
        // Read all data defining the College, i.e. name,
        //   building location, number of departments,  etc.
        // Create all instances of Department
        for(int i=0; i< numDepartments; i++)
        {
                Department *dept = new Department(inputFile);
                // Fill in the data on this Department
                m_vDepartments.push_back(*dept);
        }
}
```



This class, when instantiated represents ONE college of a university as defined in the data file.

A separate instance of Department is created for each department in the College

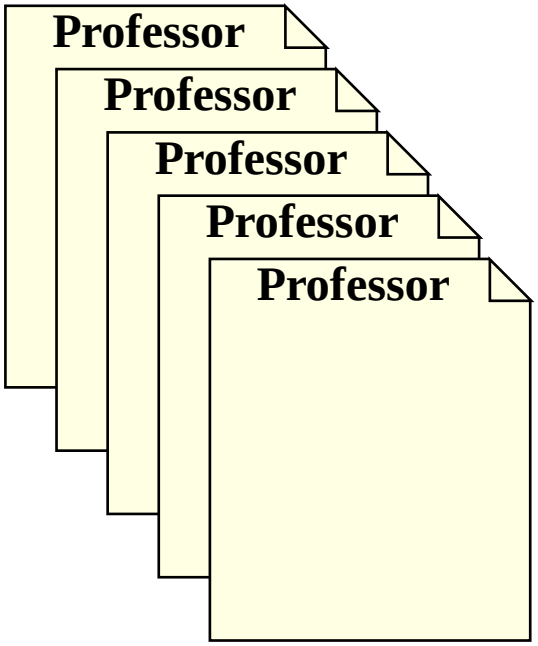


The College class does NOT represent all Colleges in the University.
The College class does NOT contain a collection of instances of College.

Department

One instance of class Department

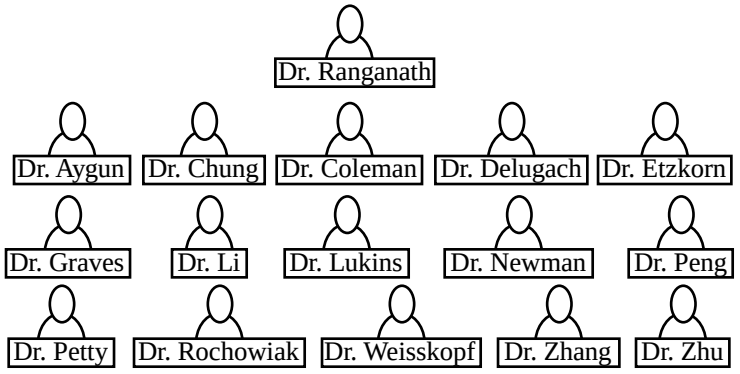
```
Department:: Department (fstream *inputFile)
{
        // Read all data defining the Department,
        //     i.e. name, building location, number of Professors,  etc.
        // Create all instances of Professor as defined in the data file
        for(int i=0; i< numProfessors; i++)
        {
                Professor *prof = new Professor(inputFile);
                // Fill in the data on this professor
                m_vProfessors.push_back(*prof);
        }
}
```



This class, when instantiated, represents ONE Department of a College as defined in the data file.

A separate instance of Professor is created for each professor in the Department

Department of Computer Science

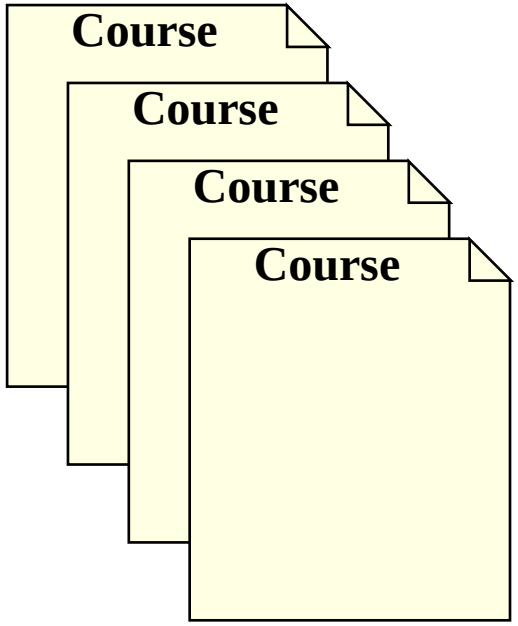


The Department class does NOT represent all Departments in the College
The Department class does NOT contain a collection of instances of Department.

Professor

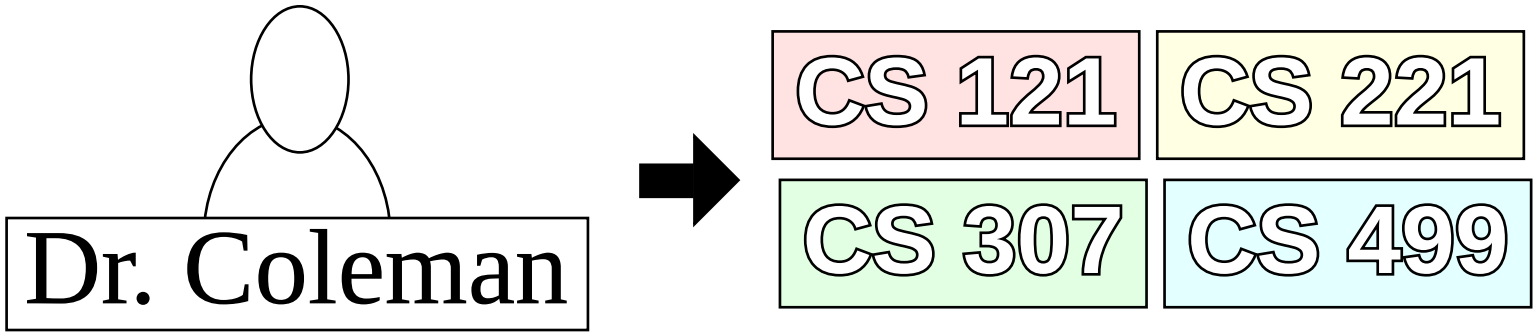
One instance of class Professor

```
Professor:: Professor (fstream *inputFile)
{
        // Read all data defining the Professor, i.e. name, degree,
        //    academic rank, office location, number of courses
        //    taught,  etc.
        // Create all instances of Course as defined in the data file
        for(int i=0; i< numCourses; i++)
        {
                Course *cors = new Course(inputFile);
                // Fill in the data on this course
                m_vCourses.push_back(*cors);
        }
}
```



This class, when instantiated, represents ONE Professor of a Department as defined in the data file.

A separate instance of Course is created for each course for this professor.

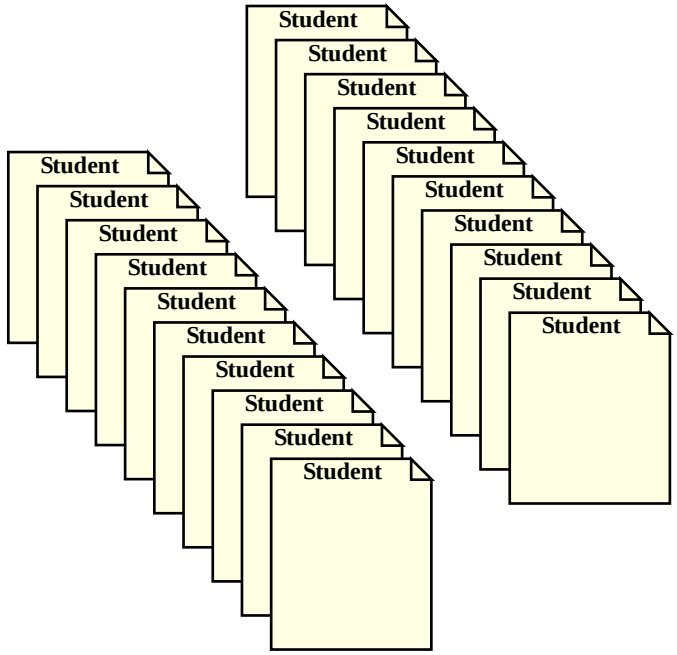


The Professor class does NOT represent all Professors in the Department.
The Professor class does NOT contain a collection of instances of Professor.

Course

One instance of class Course

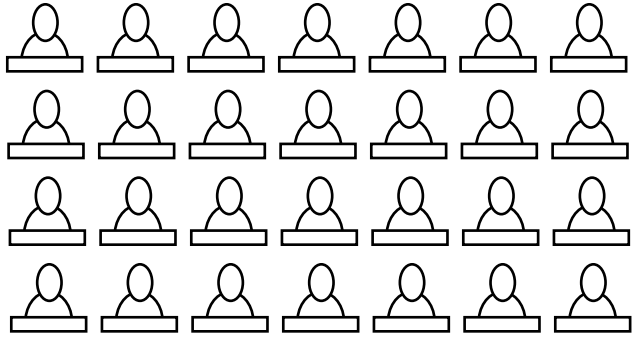
```
Course:: Course (fstream *inputFile)
{
        // Read all data defining the Course, i.e. Department
        //    abbreviation, course number, course name,
        //    hours of credit, number of students, etc.
        // Create all instances of Student as defined in the data file
        for(int i=0; i< numStudents; i++)
        {
                Student *stu = new Student(inputFile);
                // Fill in the data on this student
                m_vStudents.push_back(*stu);
        }
}
```



This class, when instantiated, represents ONE Course of a Professor as defined in the data file.

A separate instance of Student is created for each student in this course.

CS 307



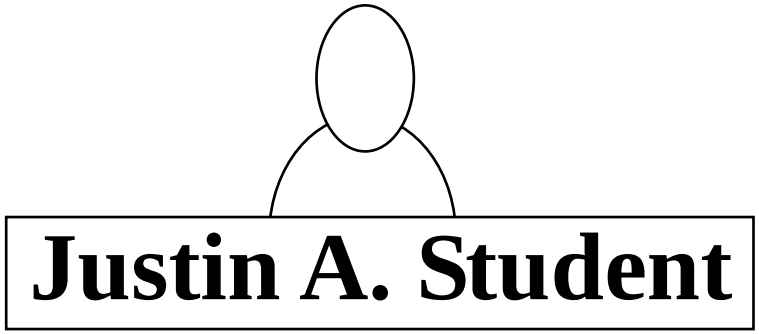
The Course class does NOT represent all Courses for this Professor.
The Course class does NOT contain a collection of instances of Course.

Student

One instance of class Student

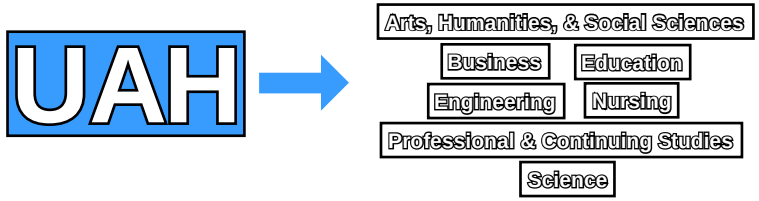
```
Student:: Student (fstream *inputFile)
{
        // Read all data defining the Student,
        //     i.e. Number, student number, etc.
}
```

This class, when instantiated, represents ONE Student in a Course as defined in the data file.

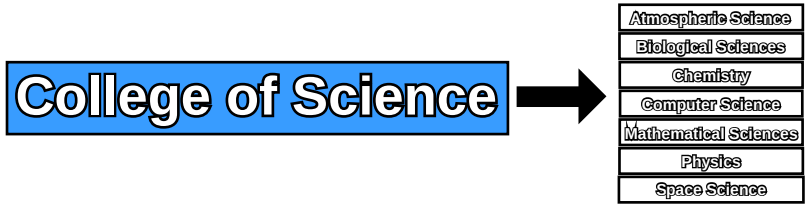


The Student class does NOT represent all Students for this Course.
The Student class does NOT contain a collection of instances of Student.

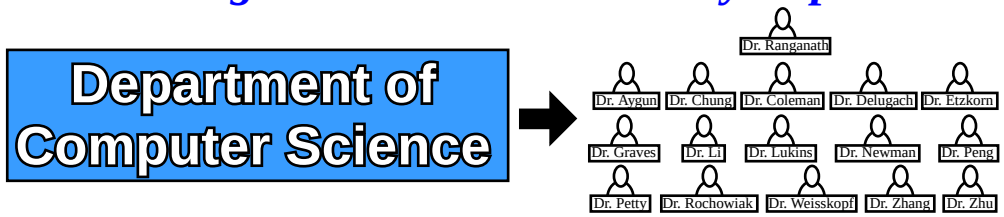
University Simulation Summary



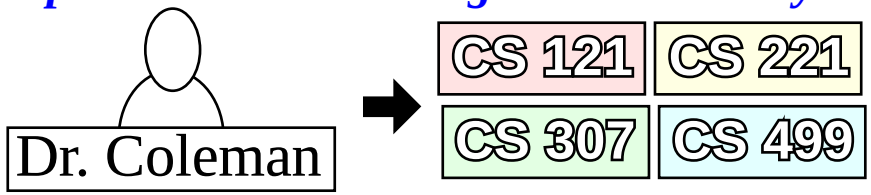
UAH is a University and it contains many Colleges.



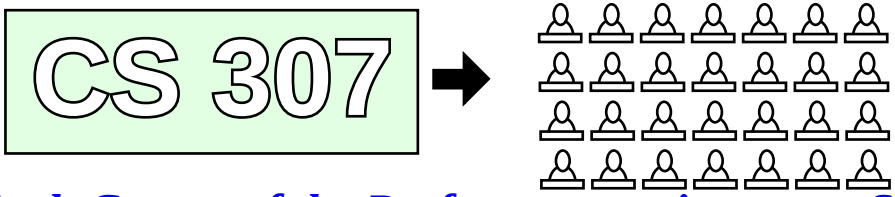
Each College at UAH contains many Departments.



Each Department in the College contains many Professors.



Each Professor teaches up to four Courses.



Each Course of the Professor contains many Students.

Each of these objects, the University, each College, each Department, each Professor, each Course, and each Student is represented in the software by a separate class instance.

Object Oriented Simulations

- **1. A Class (.h and .cpp) defines what one of the objects it represents will "look" like.**
- **2. A Class does NOT represent all instances of a given type of object.**
- **3. There must be a separate instance of each class to represent each object.**

AND

- **4. You should read the "Basic Requirements for All Programming Assignments". Follow the links:**

Find the CS307 web page —*In index on left Click*→ **Programming Assignments** —*Click on*→

Basic requirements for all programming assignments and some common problems encountered

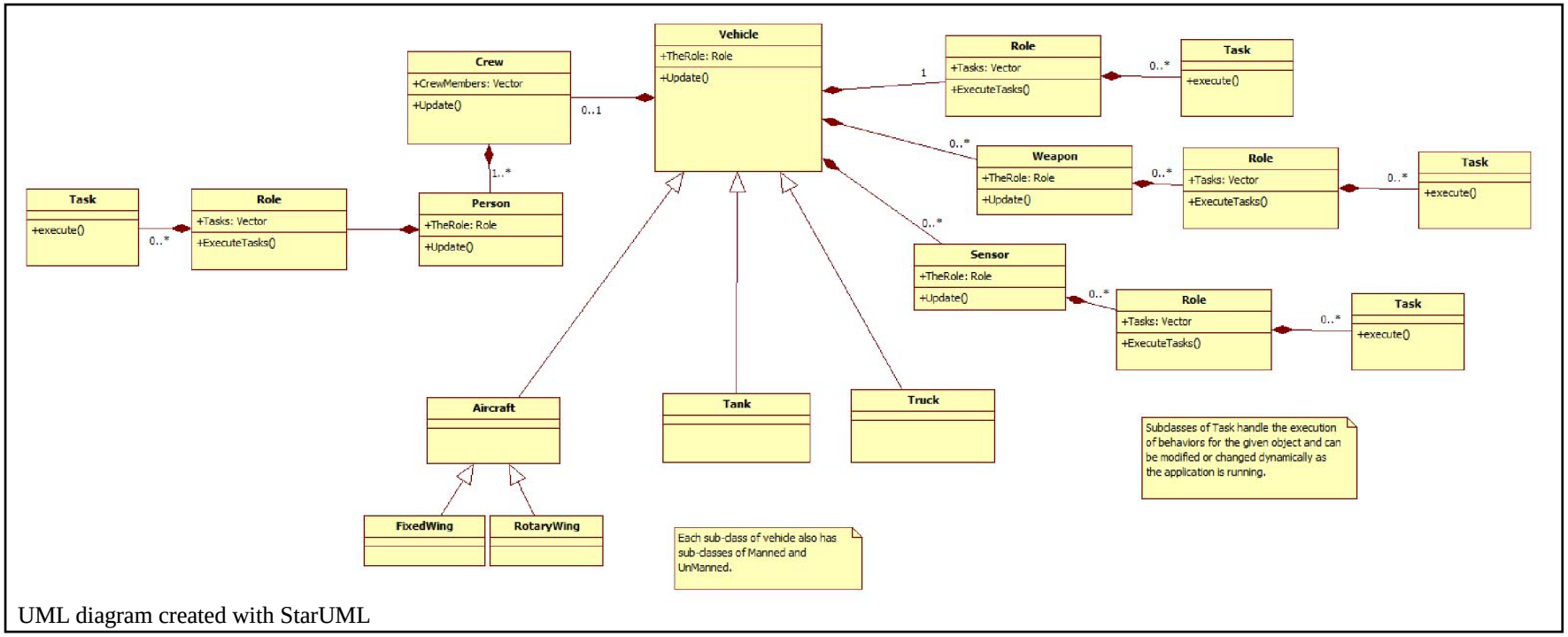
Excerpt from SOW for Vehicles in the Battlefield Simulation Application

Vehicles:

The simulation shall implement all types of vehicles participating in military operations on a battlefield. This shall include, but not be limited to: tanks, artillery, aircraft (fixed and rotary wing), transport vehicles, unmanned vehicles, etc.

The simulation shall be capable of producing simulated battlefield actions based on defined parameters of actual vehicles

UML diagram for Vehicles in the Battlefield Simulation Application



UML diagram created with StarUML

Excerpt from SOW for Terrain in the Battlefield Simulation Application

Terrain:

The simulation shall display the battlefield terrain as a relief map providing information on terrain type, and elevation.

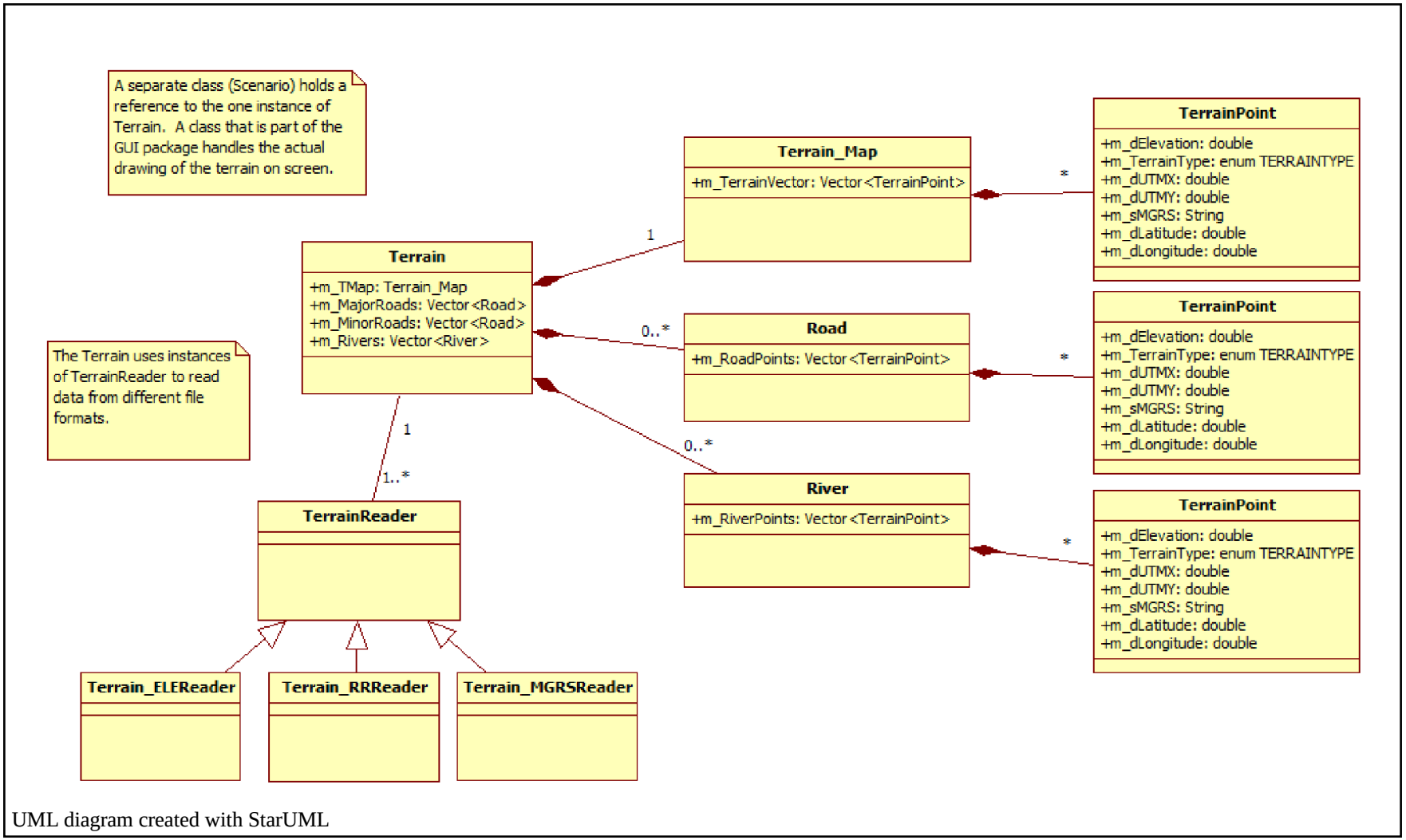
The terrain shall display urban, wooded, and cultivated areas.

The terrain shall display major roads (4+ lanes divided, 4+ lanes not divided, 2 lane paved) and minor roads (1 lane paved, 2 lane dirt, 1 lane dirt).

The terrain shall display water areas including oceans, lakes, rivers, and streams.

The simulation shall be capable reading data from a variety of data formats including, but not limited to, ELE data files, DTED data files, and military MGRS data files.

UML diagram for Terrain in the Battlefield Simulation Application



UML diagram created with StarUML

Excerpt from SOW for a Simulation of the Apple II Microcomputer

Apple II:

Because you are such a Geek and think this would be a fun way to spend your Summer vacation you shall create a simulation of the original Apple II microcomputer created by Steve Wozniak and Steve Jobs in 1977.

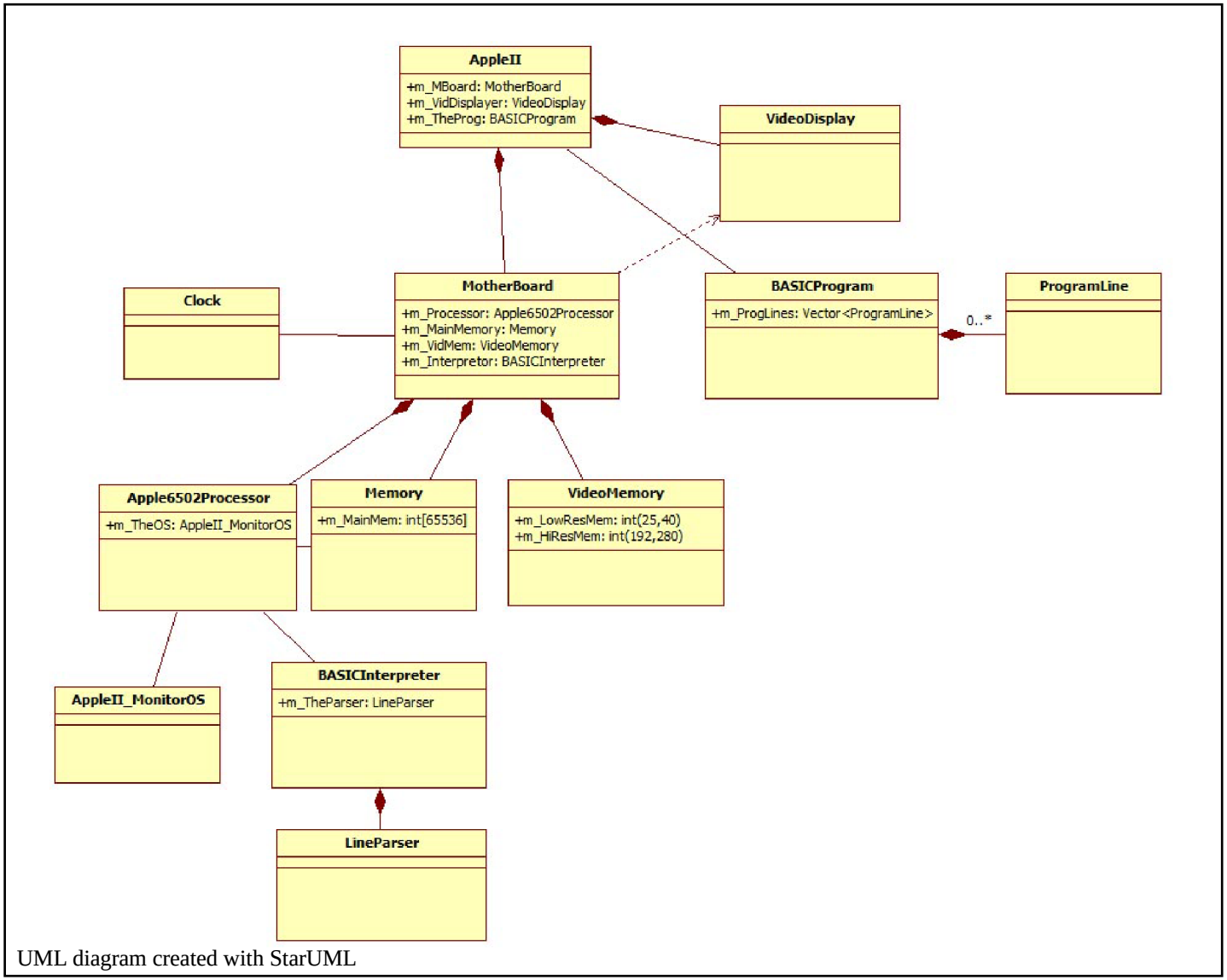
The simulation shall implement Applesoft BASIC and the Apple II Monitor ROM operating system using a simulated 6502 Microprocessor.

The simulation shall allow the user to enter Applesoft BASIC programs via the keyboard.

The simulation shall have an on-screen display that duplicates that of the original Apple II.

All processing, display, input, output, and memory management capabilities of the original Apple II shall be simulated in the application.

UML diagram for the Apple II Simulation



UML diagram created with StarUML

Object Oriented Thinking
Summary

- **Analyze the objects you are working with.**
  - **What are their attributes?**
  - **What tasks must be performed on or by these objects?**
- **How will you pass messages from one object to another?**
- **Use inheritance wisely to reduce the time it takes to design and create new objects.**
- **Use information hiding to make objects secure entities.**

In Class Activity

The registrar at UAH has come to you and asked if you can design a program that will allow them to keep records on all students attending the university. The records must include: (1) Personal information (name, address, phone number, nationality, etc.), (2) University information (major, minor, list of courses completed, current schedule, degree goal, courses remaining for the degree, etc.)

Take out a blank sheet of paper...

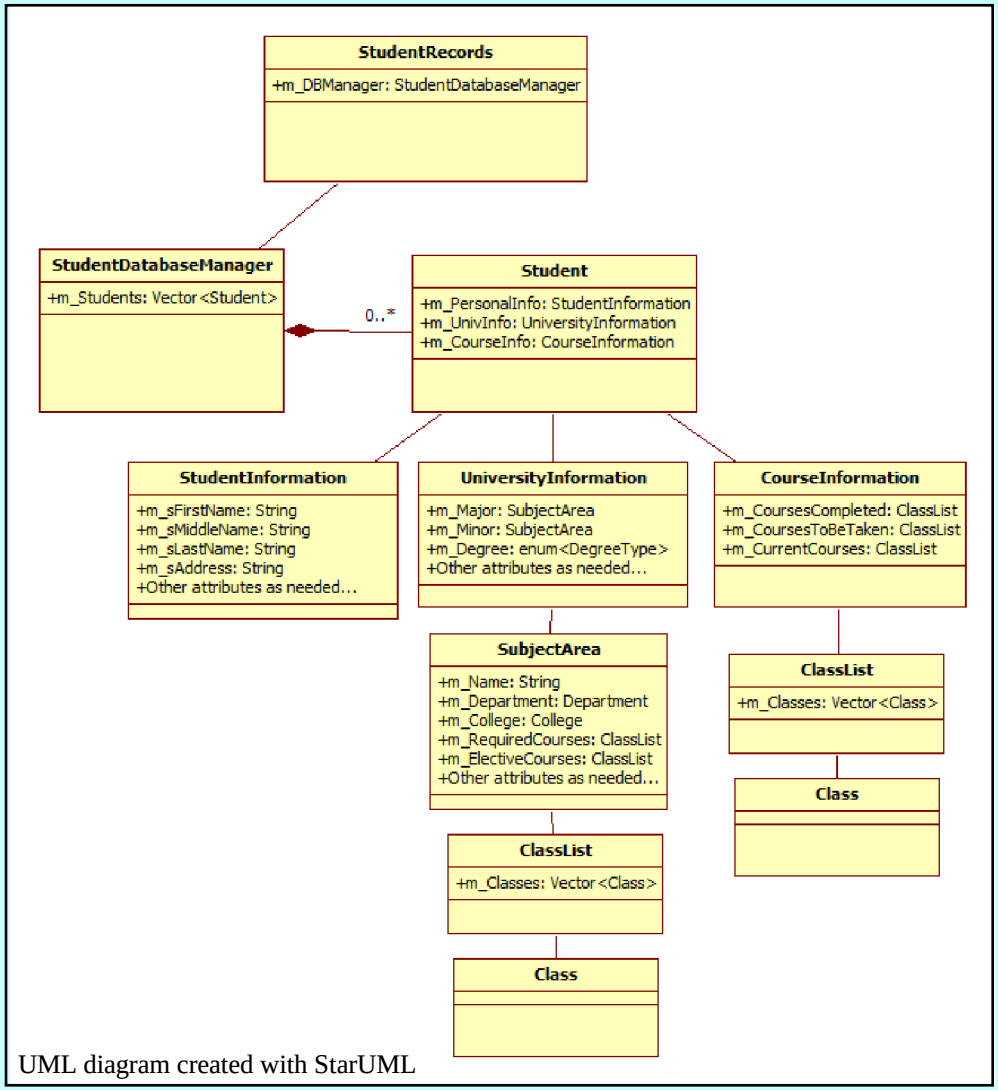
"Yes, he really means take out a blank sheet of paper, because you are about to do something in class. Note: Thinking is a required part of this activity."

List all the objects that might be required for this program.

Take about 5 to 10 minutes to do this, then we will talk about it.

In Class Activity

One Possible Approach



UML diagram created with StarUML

This is only one possible approach. There are many others that are just as good and probably even better.