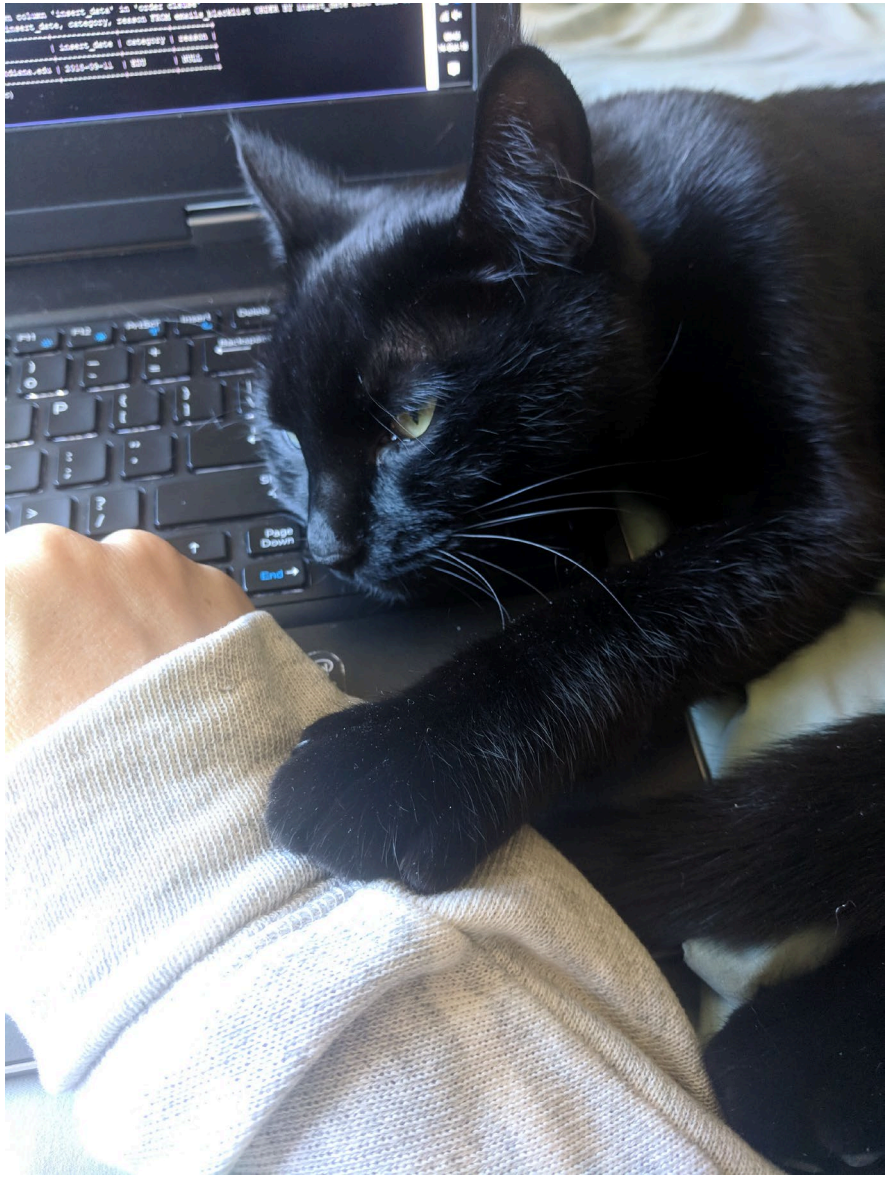


CS 202 Review

ERIN KEITH

Goals

1. OOP
2. Operator Overloading
3. Interfaces
4. Templates
5. makefiles

Homework 1

Project Goals

The goal of this project is to:

1. Review **C++**
2. Review **Templates**
3. Review **Interfaces**
4. Review **Operator Overloading**
5. Review **makefiles**

Goals

1. **OOP**
2. **Operator Overloading**
3. Interfaces
4. Templates
5. makefiles

Quiz on Wednesday 1/31

Homework 1

Project Goals

The goal of this project is to:

1. Review **C++**
2. Review **Templates**
3. Review **Interfaces**
4. Review **Operator Overloading**
5. Review **makefiles**

Department Policy

To obtain a C or better in the class, you must obtain a C (70%) or better in the following class components:

- **Average between the midterms and final exam**
- **Overall class grade**
- **Less than a 70% average in either of the above components will result in C-.**

Course Requirements

CS 202

- Principles of Object-Oriented design
- C++ classes and interfaces
- C++ templates

OOP Review?

QUESTION 1

List the principles of object-oriented programming.

- For each principle include
  - the definition
  - a C++ tool used to implement the principle

OOP Principles

ENCAPSULATION

INHERITANCE

POLYMORPHISM

Abstraction

Definition

- Higher level thinking
  - Allows us to focus on the ideas, not details
  - ***What does it mean to be a thing?***
  - Relies on imagination!
    - What makes an object unique in our system?
      - ***What should the object have?***
      - ***What should the object do?***

Abstraction

Example

- What does it mean to be a **Pet**?
- If it's more than just a *scalar* type, then
  - What does a **Pet** have?
  - What does a **Pet** do?

ENCAPSULATION

Definition

- Packaging ***state and behavior*** together into a single (*composite*)type of object.

C++ tools:

- Structs
- Classes

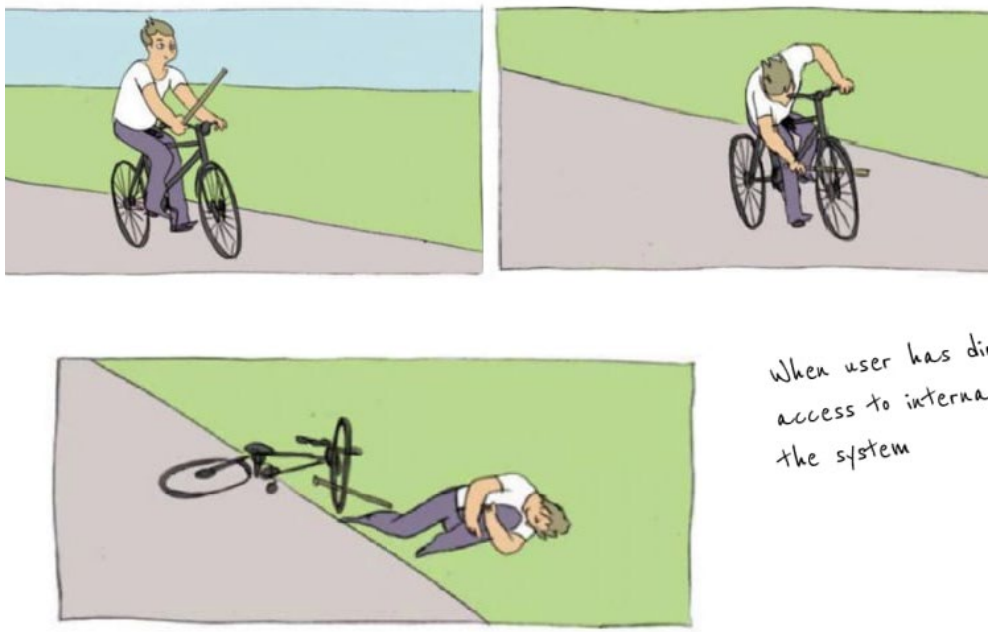
ENCAPSULATION

Definition

- This also enables us to define ***boundaries*** and how we interface with specific objects (data hiding)

C++ tools:

- Access specifiers



OOP Review?

QUESTION 2

Provide a brief example of ENCAPSULATION in C++.

ENCAPSULATION

Example – Pet

What does it mean to be a Pet?

What should a Pet have?

What should a Pet do?

The *class definition* becomes a ***blueprint*** for what an object should be and do.

```
#ifndef PET_H
#define PET_H

#include<string>
using namespace std;

class Pet{
    int age;
    bool tail;
    string name;
  public:
    Pet();
    Pet(int, bool, string);
    Pet(const Pet&);

    int getAge() const;
    bool hasTail() const;
    string getString() const;
    void setAge(int);
    void setTail(bool);
    void setName(string);

    void feed(string);
};
#endif
```

INHERITANCE

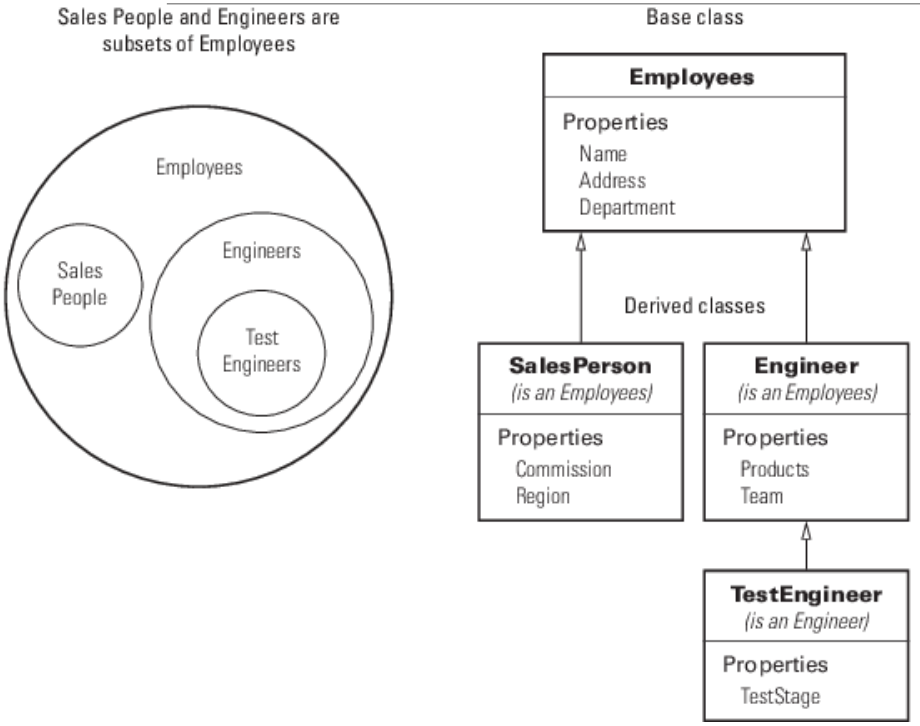
Definition

- A relationship which ***extends*** another class's data and behavior

C++ tools:

- Access specifiers
  - **`protected`**
- The **`this`** pointer

INHERITANCE



- Creates a *hierarchy* which allows us to ***reuse*** commonalities in the base and move unique qualities into a separate class

Tools:

- Base class → Derived class
- "is a"?
- "what do these have in common?"
- "what is different?"

INHERITANCE

C++ Tools

- The Base class has access to the Parent class's **`public`** and **`protected`** member functions and variables.
- While we can use the Base class's **`getters`** and **`setters`**, we will have to build new constructors for our Derived classes.
- If not implemented, **`copy constructors`** and **`assignment operators`** will be automatically created, in part by invoking the Base Class's functions first.

OOP Review?

QUESTION 3

Provide a brief example of INHERITANCE in C++.

INHERITANCE

Example – Pet → Cat

- A Cat ***is a*** Pet
- What does Cat have in common with Pet?
- What about Cat is different than Pet?

```
#ifndef CAT_H
#define CAT_H

#include "Pet.h"

#include<string>
using namespace std;

class Cat: Pet{
    int whiskers;
  public:
    Cat();
    Cat(int, bool, string, int);
    Cat(const Cat&);

    int getWhiskers() const;
    void setWhiskers(int);

    void feed(string);
};
#endif
```

POLYMORPHISM

Definition

- "The condition of occurring in several different forms."

C++ tools:

- There are actually 4 types of polymorphism in C++!

POLYMORPHISM

C++ tools:

- Subtype
  - Runtime polymorphism
- Parametric
  - Compile-time polymorphism
- Ad-hoc
  - overloading
- Coercion
  - casting

Subtype Polymorphism

Definition

- Runtime Polymorphism (what everyone thinks of when they hear "polymorphism")
- replacing base class place holders with objects of the derived type

C++ tools:

- base class pointers and references
- **`virtual`**

Example:

- How can we require that each derived class implement their own feed function?

OOP Review?

QUESTION 4

Provide a brief example of parametric/compile time POLYMORPHISM in C++.

Parametric Polymorphism

Definition

- Compile-time Polymorphism (what we'll use most in this class!)
- Execute the same code for any data type

C++ tools:

- **`templates`**

Example:

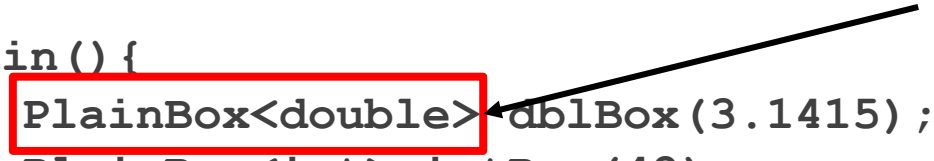
Templated Class

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX

template <typename ItemType>
class PlainBox{
  ItemType item;
public:
  PlainBox();
  PlainBox(const ItemType& theItem);
  void setItem(const ItemType& theItem);
  ItemType getItem() const;
};
#include "PlainBox.cpp"
#endif
```

Driver

```
#include "PlainBox.h"

#include <iostream>
using namespace std;

int main(){
        PlainBox<double> dblBox(3.1415);
        PlainBox<int> intBox(42);

        cout << dblBox.getItem() << endl;
        cout << intBox.getItem() << endl;

        return 0;
}
```

During compile time, the compiler to creates the code for a PlainBox class that holds a double.



Created Class

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX

template <typename double>
class PlainBox{
  double item;
public:
  PlainBox();
  PlainBox(const double& theItem);
  void setItem(const double& theItem);
  double getItem() const;
};
#include "PlainBox.cpp"
#endif
```

Ad-hoc Polymorphism

Definition

- functions with the same name behave differently for each type
- particularly useful in conjunction with runtime polymorphism

C++ tools:

- Overloading
- Overriding

Ad-hoc Polymorphism

Consider the addition operator. It takes two operands and returns the sum. Do you think it works the same way for ints as for floats? Why does it work differently when the operands are of mixed type?

Would we ever want to add objects together? Can we? If so, how?

Operator overloading! Operators act like functions and we can define our own behavior through code.

Next Class

Module:

Week 2: Review

Topic:

Overloading

Interfaces

Templates

makefiles