Lab 9: Object-oriented programming

Agenda

- **Lecture Topic Review**
- Practice Questions
- Lab 9 assignment
- Weekly Reminders
- Q&A

Lecture Review

- Object Oriented Programming
  - Structs
  - Classes

Keywords: constructors, mutators, accessors, access modifiers

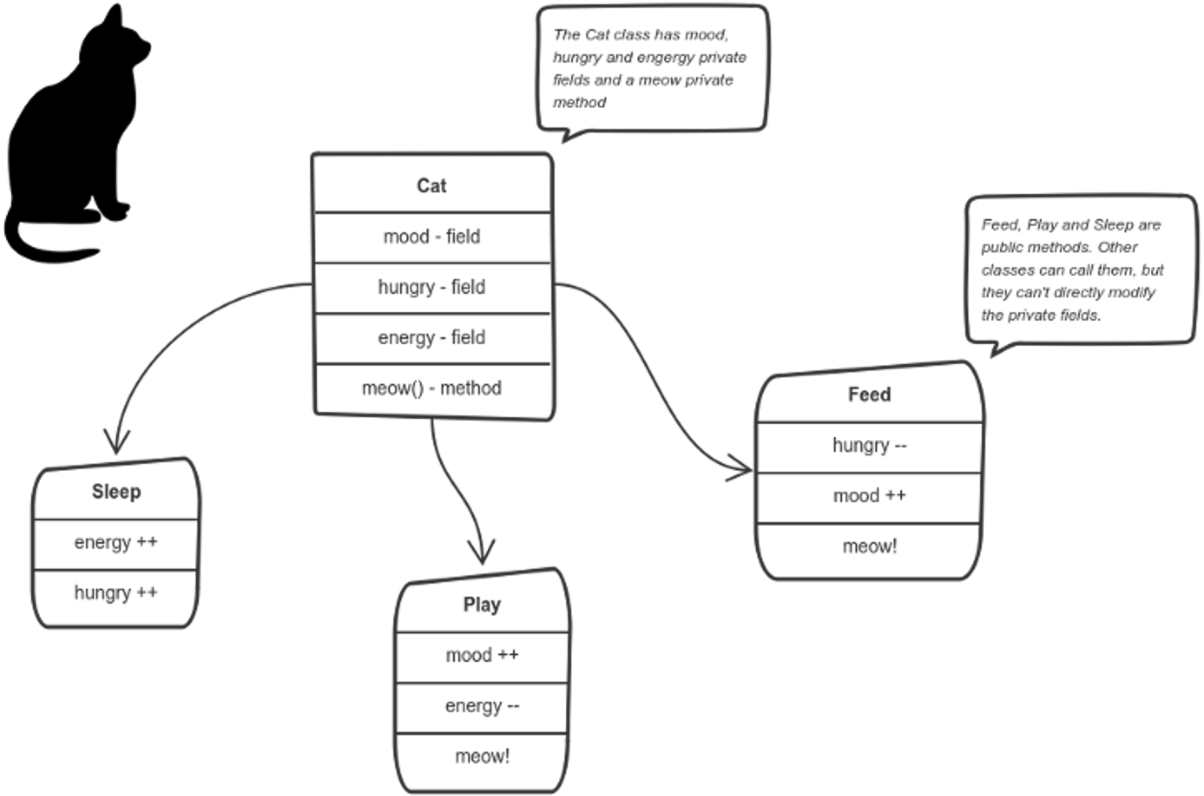
Object-oriented Programming

What is OOP?

Fundamental concepts:

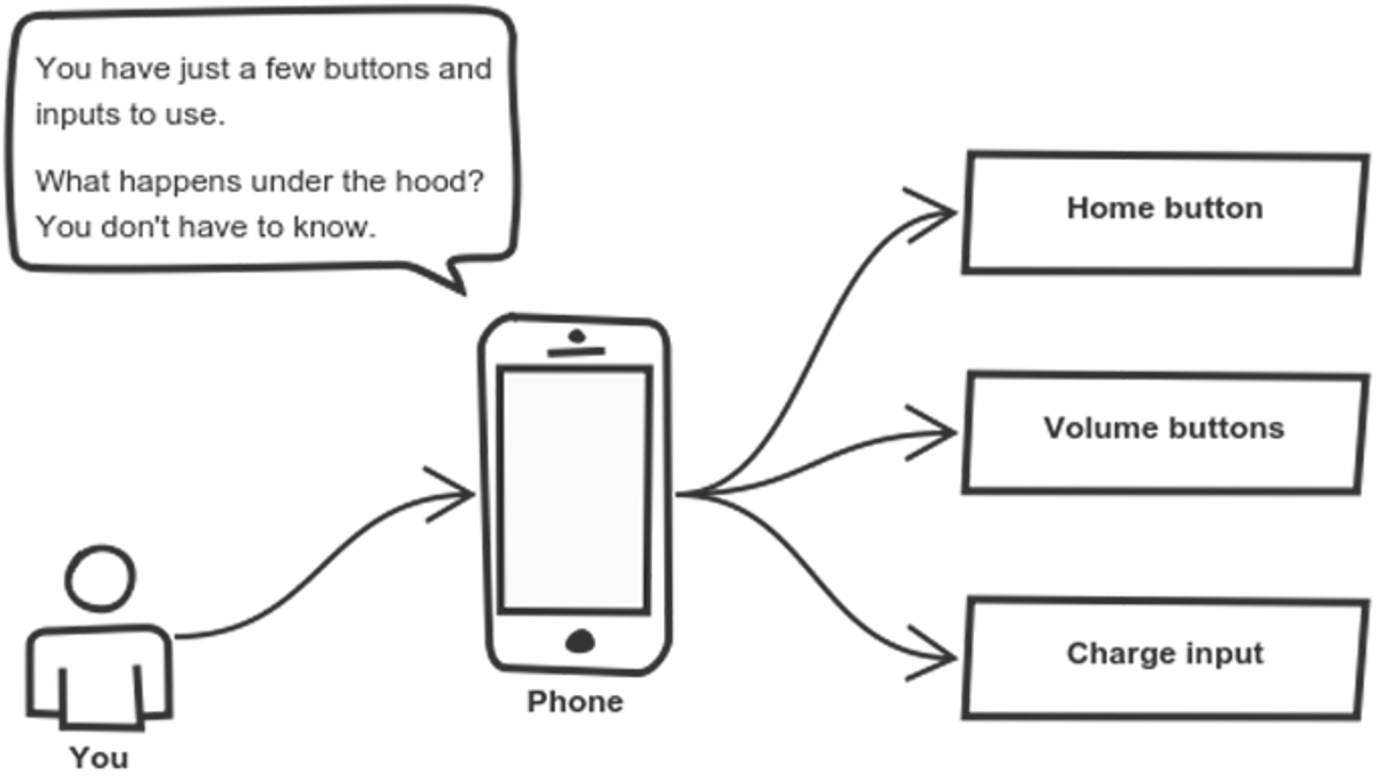
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

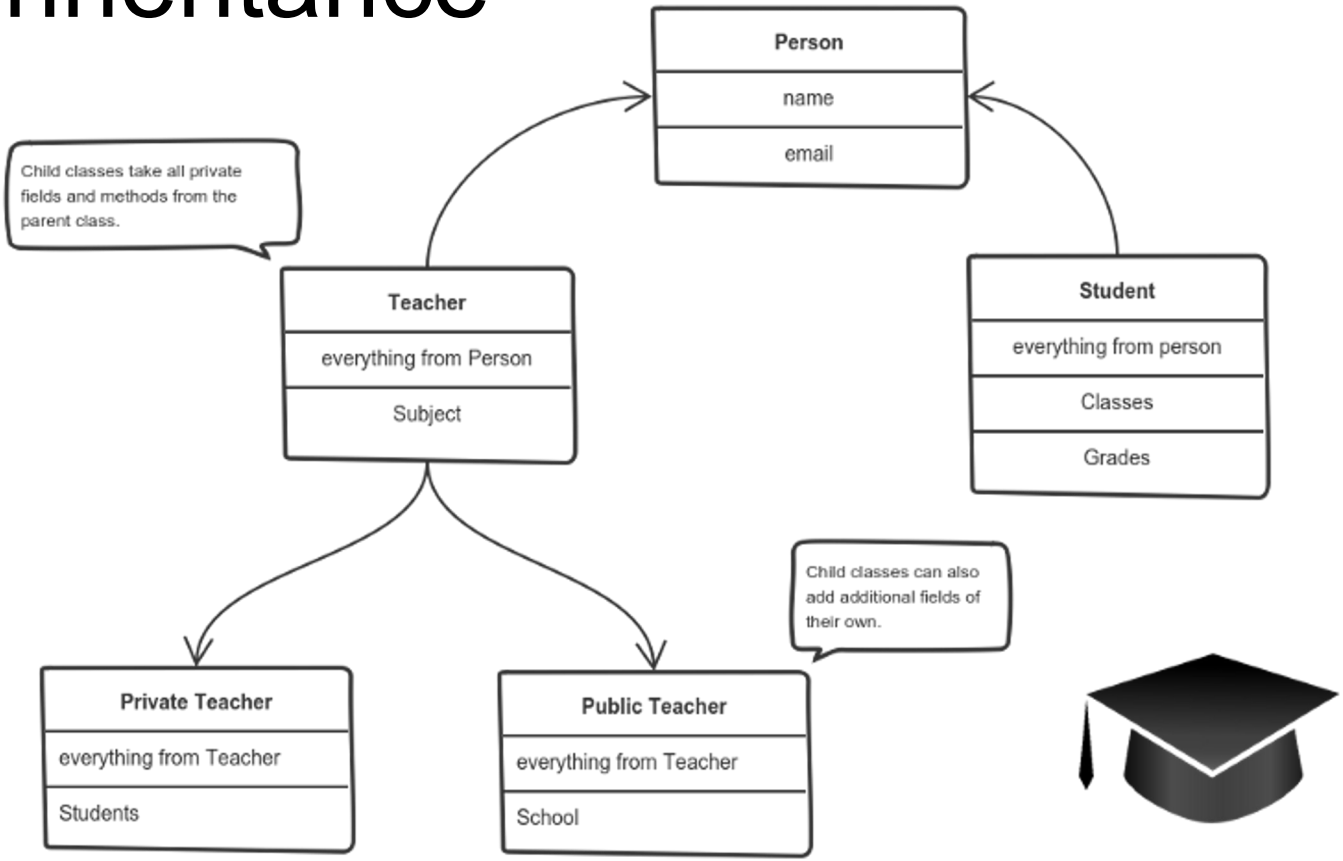


- Keeps related data and functionality together

Abstraction

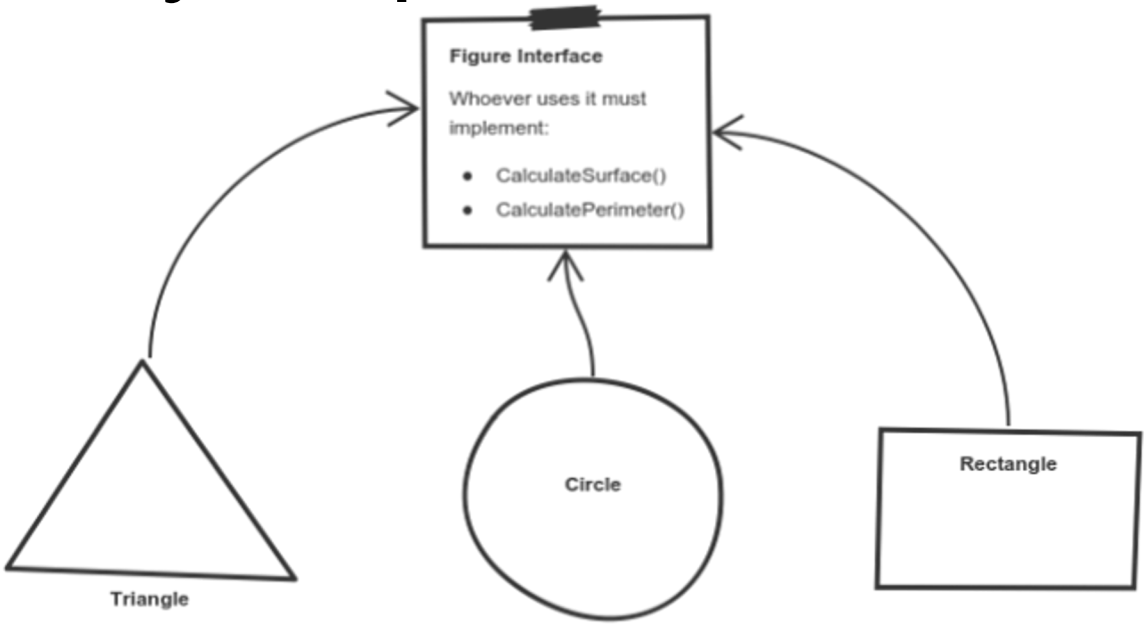


Inheritance



- A new class inherits all properties of existing class
- Prevents code duplication

Polymorphism



Triangle, **Circle** and **Rectangle** inherit the Figure interface or abstract class.

They implement their own versions of **CalculateSurface()** and **CalculatePerimeter()**.

They can be used in a mixed collection of Figures.

- Allows new class to decide how a function is implemented for its specific use
- The class's implementation will override the base class's implementation of a function of the same name

Check out:
https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/

Structs in C++

Keeping track of heterogeneous data!

```
 7  struct Fish {
 8      string species;
 9      double length; //inches
10      string name;
11  };
```

...

```
41      Fish fish1;
42      fish1.name = "Buford";
43      Fish fish2 = {"Leaf fish", 5, "Leif"};
44      Fish fish3 = fish2;
```

Notice:

- `struct` keyword
- Struct name
- Member variables
- End definition with a semicolon
- Default public members

Classes in C++

Creating our own objects to use complete with data and functions!

Advantage of Classes

- Members are default private
  - Unlike structs (default public)
  - Protects the code
- Heterogeneous data
  - Can have functions and variables of different types
  - Can’t do this with arrays or vectors!

General Syntax

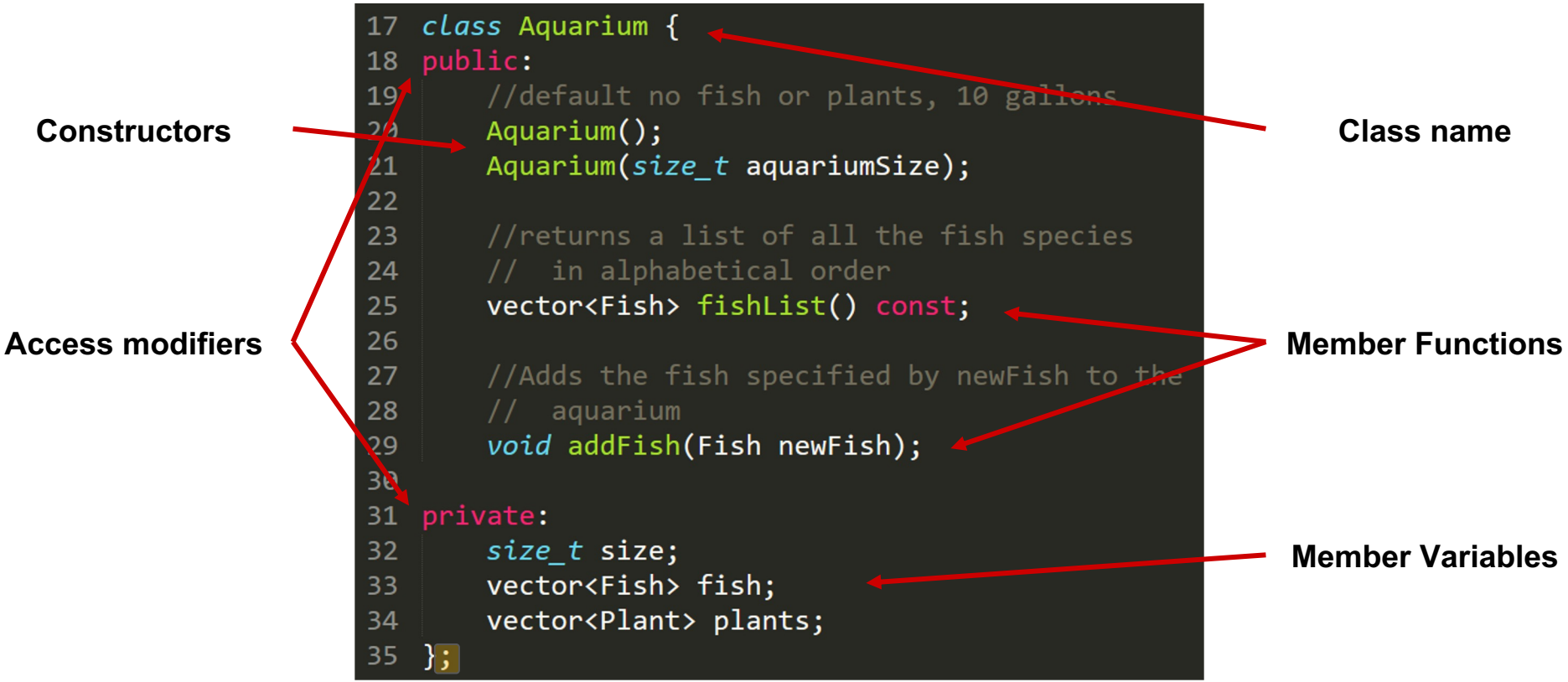
```
class my_class
{
    private:
    // private members here

    public:
    // public members here


};
```

Create object

```
27    T identifier(arg_list);
```

type (class name)

comma-separated list of zero or more arguments (depending on T)


Constructors
Access modifiers
Class name
Member Functions
Member Variables
17 class Aquarium {
18 public:
19 //default no fish or plants, 10 gallons
20 Aquarium();
21 Aquarium(size_t aquariumSize);
22
23 //returns a list of all the fish species
24 // in alphabetical order
25 vector<Fish> fishList() const;
26
27 //Adds the fish specified by newFish to the
28 // aquarium
29 void addFish(Fish newFish);
30
31 private:
32 size_t size;
33 vector<Fish> fish;
34 vector<Plant> plants;
35 };

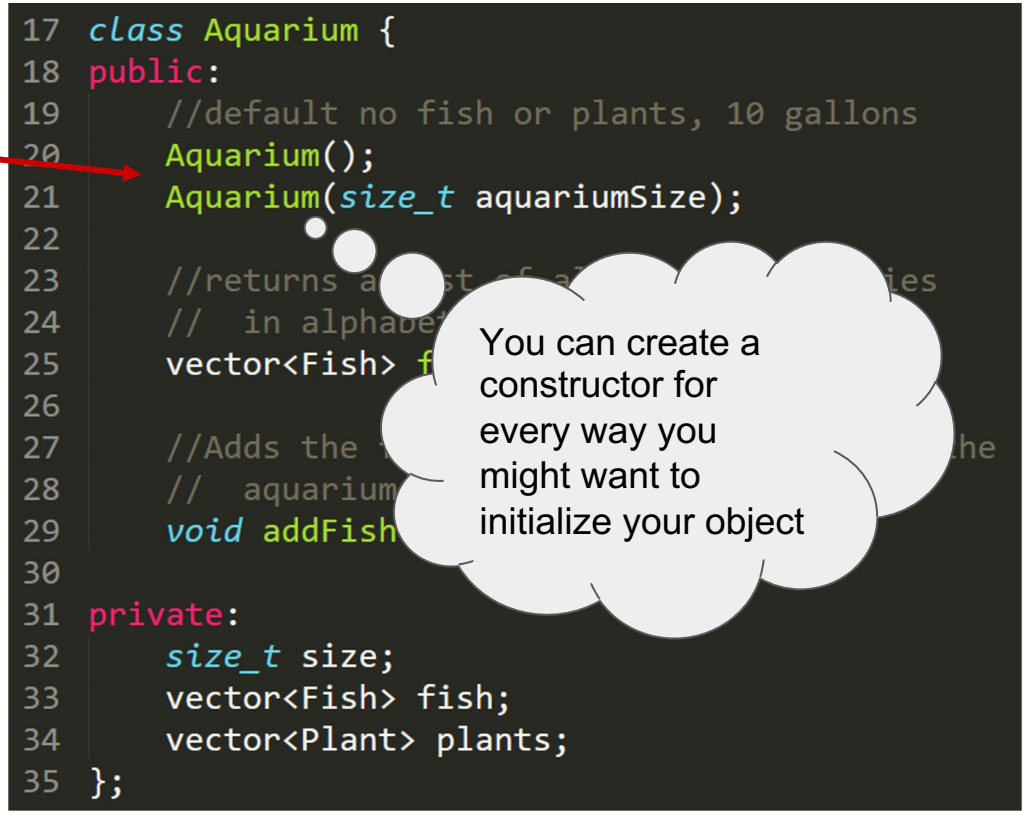
Constructors

Access modifiers

Class name

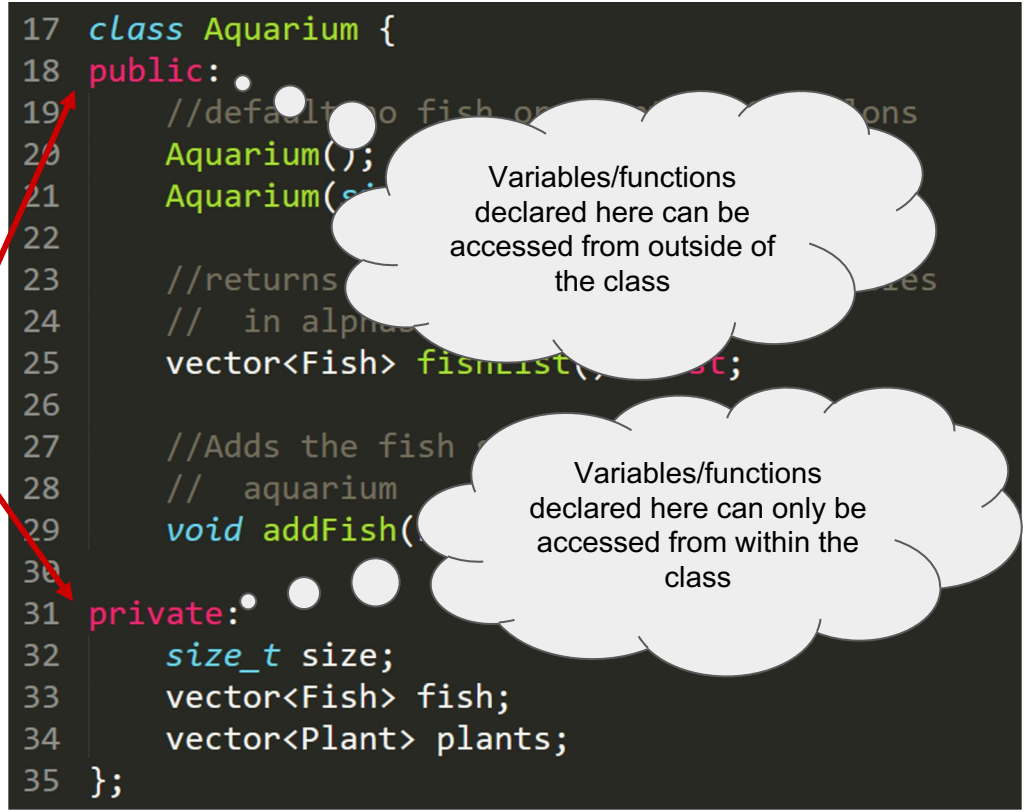
Member Functions

Member Variables



Constructors

Access modifiers



Class name

Member Functions

Member Variables

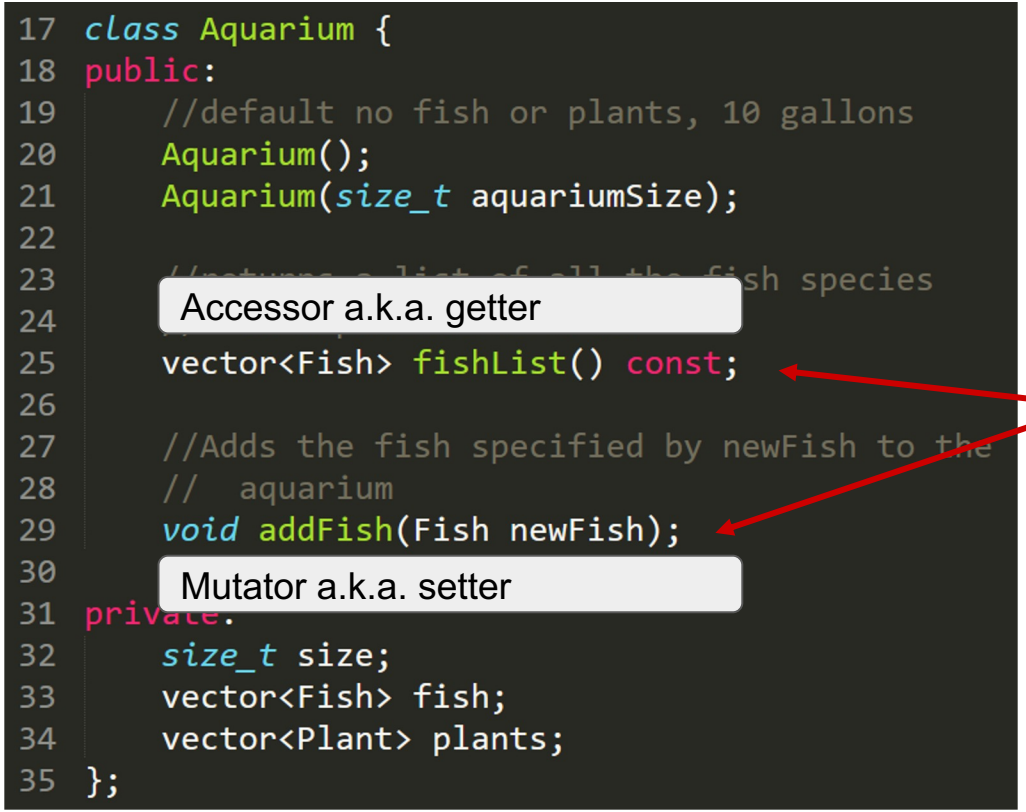
Constructors

Access modifiers

Class name

Member Functions

Member Variables



Constructors

Access modifiers

```
17 class Aquarium {
18 public:
19     //default no fish or plants, 10 gallons
20     Aquarium();
21     Aquarium(size_t aquariumSize);
22
23     //returns a list of all the fish species
24     //  in alphabetical order
25     vector<Fish> fishList() const;
26
27     //Adds the fish specified by newFish to the
28     //  aquarium
29     void addFish(Fish newFish);
30
31 private:
32     size_t size;
33     vector<Fish> fish;
34     vector<Plant> plants;
35 };
```

Class name

Member Functions

Member Variables

Defining class member functions

In many cases, it's much cleaner to declare your functions separately from the implementation:

In the class definition, write:

```
//returns a list of all the fish species
//  in alphabetical order
vector<Fish> fishList() const;
```

Outside of the class definition, write:

```
vector<Fish> Aquarium::fishList() const {
    return fish;
}
```

Defining class member functions

Constructors

In the class definition, write:

```
//default no fish or plants, 10 gallons
Aquarium();
Aquarium(size_t size);
```

Outside of the class definition, write:

```
Aquarium::Aquarium() {
    size = 10;
}

Aquarium::Aquarium(size_t size_in) {
    size = size_in;
}
```

Reiterating Some Important Things!!!

- Scope Resolution Operator
  - Indicates a function is a member function of a certain class
- const after member function names
  - Prevents unintended changes to data members (so be careful of where you put const keyword)!
- Don’t forget the semicolon after class declaration/s closing curly bracket!

Agenda

- Lecture Topic Review
- **Practice Questions**
- Lab 9 assignment
- Weekly Reminders
- Q&A

Practice Problem #1

Given the following code already implemented

```
class person {
     public:
          person() {fullName = "";} // Default ctor
          person(string fullNameIn);
          string get_name();
          void set_name(string fullNameIn);
     private:
          string fullName;
};

person::person(string fullNamein) {
fullName = fullNamein; }

string person::get_name() {
return fullName; }

void person::set_name(string fullNameIn) {
fullName = fullNameIn; }
```

Do we need `get_name()` and `set_name()`? (i.e. if you remove the `get_name()` and `set_name()` functions, keeping everything else the same, can you still have the same ability to **set** variable `fullName` to a **new value** and can you still have the same ability to **obtain** the value stored in `fullName`?

A. Yes, we need the two functions
B. No, we do not need the 2 functions
C. Doesn't matter

Practice Problem #1: Solution

Given the following code already implemented

```
class person {
    public:
        person() {fullName = "";} // Default ctor
        person(string fullNameIn);
        string get_name();
        void set_name(string fullNameIn);
    private:
        string fullName;
};

person::person(string fullNamein) {
fullName = fullNamein; }

string person::get_name() {
return fullName; }

void person::set_name(string fullNameIn) {
fullName = fullNameIn; }
```

Do we need `get_name()` and `set_name()`? (i.e. if you remove the `get_name()` and `set_name()` functions, can you still have the same ability to **set** variable `fullName` to a **new value** and can you still have the same ability to **obtain** the value stored in `fullName`?

A. Yes, we need the two functions
B. No, we do not need the 2 functions
C. Doesn't matter

Practice Problem #2

Given the following code already implemented

```
class person {
     public:
          person() {fullName = "";} // Default ctor
          person(string fullNameIn);
          string get_name();
          void set_name(string fullNameIn);
     private:
          string fullName;
};

person::person(string fullNamein) {
fullName = fullNamein; }

string person::get_name() {
return fullName; }

void person::set_name(string fullNameIn) {
fullName = fullNameIn; }
```

How do we create an object `person1` of class `person` with name `Jane Doe`?

```
person person1("Jane Doe"); // A

person person1; // B

person person1; // C
person1.set_name("Jane Doe");
```

A. Option A
B. Option B
C. Option B & C
D. Option A & C

Practice Problem #2: Solution

Given the following code already implemented

```
class person {
     public:
          person() {fullName = "";} // Default ctor
          person(string fullNameIn);
          string get_name();
          void set_name(string fullNameIn);
     private:
          string fullName;
};

person::person(string fullNamein) {
fullName = fullNamein; }

string person::get_name() {
return fullName; }

void person::set_name(string fullNameIn) {
fullName = fullNameIn; }
```

How do we create an object `person1` of class `person` with name `Jane Doe`?

```
person person1("Jane Doe"); // A

person person1; // B

person person1; // C
person1.set_name("Jane Doe");
```

A. Option A
B. Option B
C. Option B & C
D. Option A & C

Agenda

- Lecture Topic Review
- Practice Questions
- **Lab 9 assignment**
- Weekly Reminders
- Q&A

Lab Assignment

Part 1: Struct practice - define two structs and analyze the code.

Part 2: Classes practice - define a class and finish implementing a program using that class - Note that the source file is a bit larger than what we've worked with so far, so follow the instructions carefully and make sure you understand why the steps are being taken to save time debugging.

Weekly Reminders

- Lab 9 is due before the next lab.
- Project 3 is due November 10th.

Questions?

Bonus Challenge Questions

Also try to practice implementing your own classes. Especially try practicing inheritance!

Practice Problem #3

We want to create a class `student` that *inherits* from the `person` class, and also has its own additional function to set a variable called `finalGrade`.

```
class person {
      // Can be accessed outside of class
      public:
            person() {fullName = "";}
            person(string fullNameIn);
            string get_name();
            void set_name(string fullNameIn);

      // Can't be accessed outside of class member functions
      private:
            string fullName;
};
person::person(string fullNamein) { fullName = fullNamein; }
string person::get_name() { return fullName; }
void person::set_name(string fullNameIn) { fullName = fullNameIn; }
```

Say we have this in main.

```
student student1("Bob");
student1.set_grade(79.2);
cout << student1.get_grade() << endl;
cout << student1.get_name() << endl;
```

And the output is:

```
79.2
Bob
```

Which implementation on the right is correct?

A.

```
class student: public person {
      public:
            student(string name_in) : person(name_in) {}
            student(string name_in, float gradeIn)
                  : person(name_in) {finalGrade = gradeIn;}
            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
};
```

B.

```
class student: public person {
      public:
            student(string name_in) {}
            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
};
```

C.

```
class student: public person {
      public:
            student(string name_in){ fullName = name_in;}

            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
```

Practice Problem #3: Solution

We want to create a class `student` that *inherits* from the `person` class, and also has its own additional function to set a variable called `finalGrade`.

```
class person {
      // Can be accessed outside of class
      public:
            person() {fullName = "";}
            person(string fullNameIn);
            string get_name();
            void set_name(string fullNameIn);

      // Can't be accessed outside of class member functions
      private:
            string fullName;
};
person::person(string fullNamein) { fullName = fullNamein; }
string person::get_name() { return fullName; }
void person::set_name(string fullNameIn) { fullName = fullNameIn; }
```

Say we have this in main.

```
student student1("Bob");
student1.set_grade(79.2);
cout << student1.get_grade() << endl;
cout << student1.get_name() << endl;
```

And the output is:

```
79.2
Bob
```

Which implementation on the right is correct?

A.

```
class student: public person {
      public:
            student(string name_in) : person(name_in) {}
            student(string name_in, float gradeIn)
                  : person(name_in) {finalGrade = gradeIn;}
            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
};
```

B.

```
class student: public person {
      public:
            student(string name_in) {}
            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
};
```

C.

```
class student: public person {
      public:
            student(string name_in){ fullName = name_in;}

            void set_grade(float gradeIn) { finalGrade = gradeIn;}
            float get_grade() {return finalGrade;}
      private:
            float finalGrade;
```

Practice Problem #4

Given the following code:

```
class Student {
public:
      Student(string name_in, int id_in)
            : name(name_in), studentID(id_in) {}
      string get_name() { return name;}
      int get_ID() { return studentID;}

private:
      string name;
      int studentID;
};

class classMajor {
public:
      classMajor(int class_size) {
            size = class_size;}
      void add_student(Student student_in) {
            studentList.push_back(student_in); }
      int get_size() { return size;}
      Student get_student(int location) {
            return studentList.at(location); }
private:
      vector<Student> studentList;
      int size;
};
```

If we have the following in main:

```
classMajor freshman(10);
for (int i = 0; i < (freshman.get_size()); i++) {
      string student = "student";
      student.append(to_string(i));
      Student studentNew(student, (i + 2000));
      freshman.add_student(studentNew);
}
```

If we want to print out all of the student IDs in the vector `studentList`, which of the following implementations is correct?

A.

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_student(i).get_ID() << endl;
}
```

B. Adsf

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_student(i).get_ID(i) << endl;
}
```

C. Adsf

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_ID().get_student(i) << endl;
}
```

Practice Problem #4: Solution

Given the following code:

```
class Student {
public:
      Student(string name_in, int id_in)
            : name(name_in), studentID(id_in) {}
      string get_name() { return name;}
      int get_ID() { return studentID;}

private:
      string name;
      int studentID;
};

class classMajor {
public:
      classMajor(int class_size) {
            size = class_size;}
      void add_student(Student student_in) {
            studentList.push_back(student_in); }
      int get_size() { return size;}
      Student get_student(int location) {
            return studentList.at(location); }
private:
      vector<Student> studentList;
      int size;
};
```

If we have the following in main:

```
classMajor freshman(10);
for (int i = 0; i < (freshman.get_size()); i++) {
      string student = "student";
      student.append(to_string(i));
      Student studentNew(student, (i + 2000));
      freshman.add_student(studentNew);
}
```

If we want to print out all of the student IDs in the vector `studentList`, which of the following implementations is correct?

A.

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_student(i).get_ID() << endl;
}
```

B. Adsf

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_student(i).get_ID(i) << endl;
}
```

C. Adsf

```
for (int i = 0; i < (freshman.get_size()); i++) {
      cout << freshman.get_ID().get_student(i) << endl;
}
```