Object Oriented Programming

C++ Lecture 5

Nick Matthews

Adam Kohl

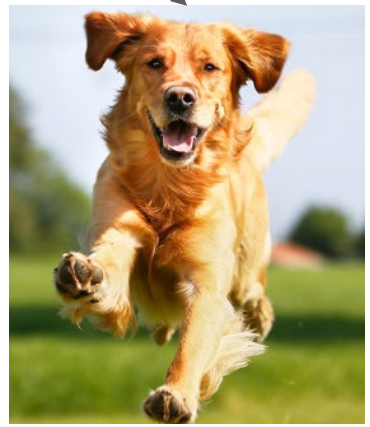
Motivation

- Many times we will have collections of variables and functionality we want to use again and again
- Don't want to recreate them from scratch
- Soooo we use objects which are created using classes
- These classes can have relationships which we call inheritance

What Are Classes and Objects?

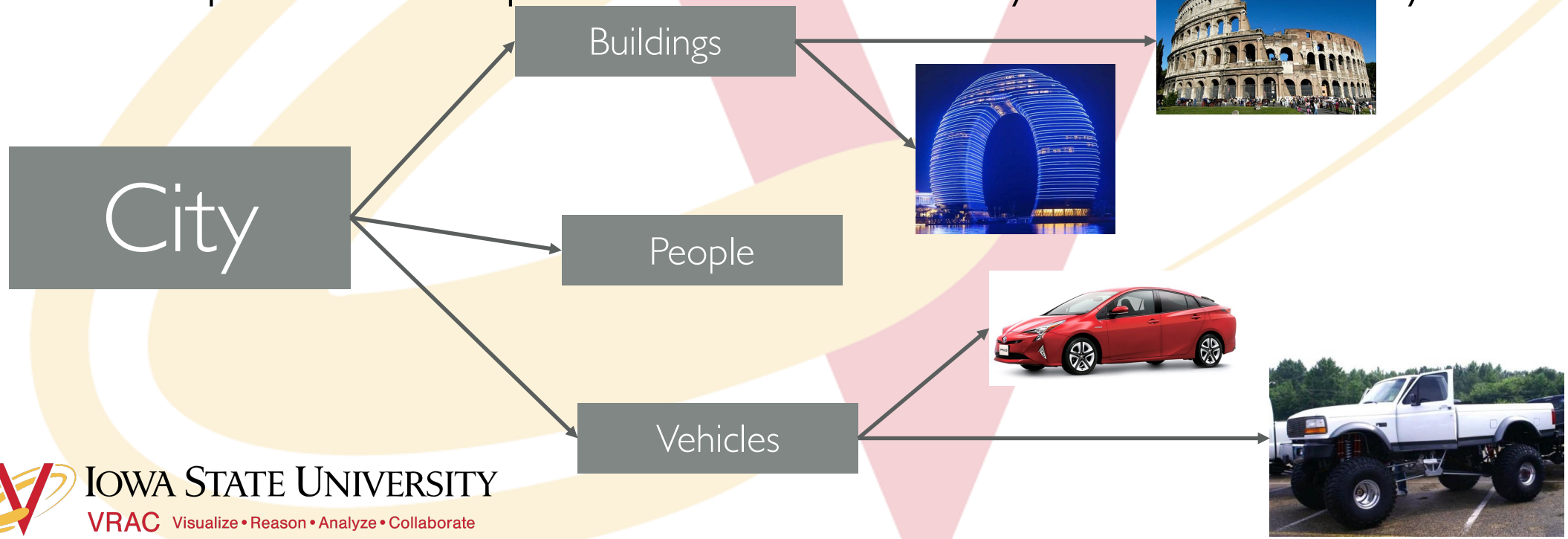
- A class is a general blueprint for an object
  - Mammals are a class
  - Dogs and Cats are classes
  - Dogs and Cats are each a type or subclass of Mammals

- Objects are specific instantiations of classes
  - A specific dog can be named Spot and have brown f



Why Do We Use Objects?

- Allows us to breakup and organize code into functional areas
- Makes solving problems easier and cleaner
- Helps us decouple code functionality for extensibility



Defining and Creating Classes

- Behavior of classes are defined using a collection of functions
- Terminology
  - A class defines the behavior of some object
  - An object is an instance of a class that can be created and assigned variables
  - There can be multiple instances (objects) of a class

Class Syntax

```
class Square    // Class keyword tells compiler to expect a class definition
{
public:
    Square(float w){    // Class constructor
        width = w;
    };

    ~Square(){  // Class destructor
    };

    float area(){   // Area function definition
        return width*width;
    };

protected:
    float width;    // Width variable used in the constructor and the area calculation

};
```

```
Square s = Square(5.0); // Instantiating a square object
std::cout << s.area() << std::endl; // Prints out 25.0
```

Instantiation

- When we create a new object of some class and assign the object to a variable, we are instantiating it or creating a unique instance of that class
- When a new object is created the constructor is called
- The constructor is responsible for setting up the object
- The constructor always has the same name as the class

```
Square s1 = Square(5.0);    // Instantiating a square object
std::cout << s1.area() << std::endl; // Prints out 25.0

Square s2 = Square(2.0);    // Instantiating a square object
std::cout << s2.area() << std::endl; // Prints out 4.0

Square s3 = Square(10.0);   // Instantiating a square object
std::cout << s3.area() << std::endl; // Prints out 100.0
```

Class Members

- Classes also have members
- A member is a function or variable included in the class
- Members are accessed using the "dot" syntax

```
class Square    // Class keyword tells compiler to expect a class definition
{
public:
    Square(float w){    // Class constructor
        width = w;
    };

    ~Square(){  // Class destructor
    };

    float area(){   // Area member function definition
        return width*width;
    };

protected:
    float width;    // Width is a memeber variable

};
```

```
Square s2 = Square(2.0);    // Instantiating a square object
std::cout << s2.area() << std::endl; // Prints out 4.0
```

Access Specifiers

- Access to class members can be controlled using the public, protected, and private keywords
- Public members can be accessed outside of the class
- Protected and private can only be accessed inside of the class

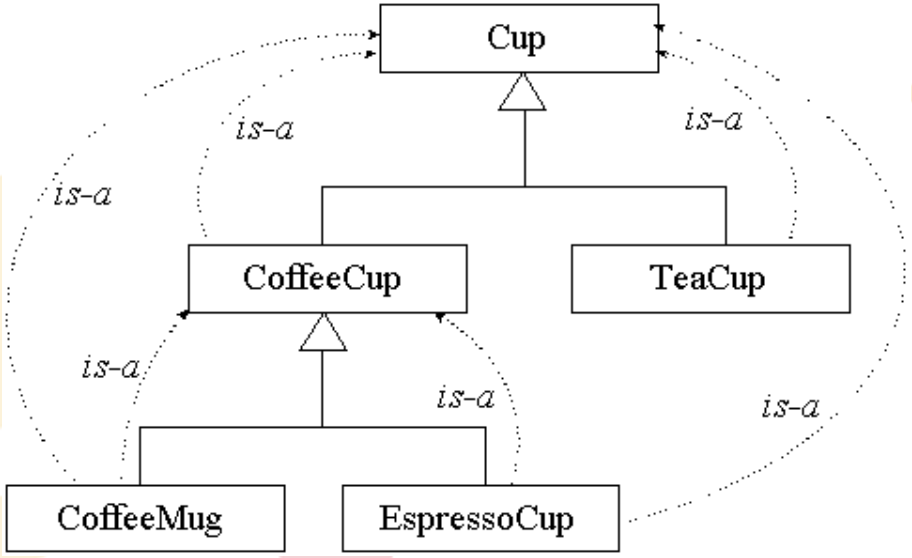
```
Square s3 = Square(10.0);   // Instantiating a square object
std::cout << s3.area() << std::endl; // Prints out 100.0
s3.width = 100; // Error! member variable is protected!
```

Checkpoint

- Create a rectangle class
  - Constructor should take in the length and width
  - Add a member functions to get and set the length/width
  - Add a function to compute the area of the rectangle
  - Print out area, length, and width to command line

Challenge: Inheritance

- Using the shapes analogy
  - Shape is a superclass or parent class of squares, circles, and triangles
  - Meaning squares, circles, and triangles are subclasses or child classes of shapes
- Using inheritance a child can inherit the members of its parent



Challenge: Inheritance

```
class Shape
{
public:
    Shape() {
    };

    ~Shape() {
    };

    void setColor(std::string aColor)
    {
        color = aColor;
    };

    std::string returnColor()
    {
        return color;
    };

protected:
    std::string color;
};
```

```
class Square:public Shape
{
public:
    Square(float w){    // Class constructor
        width = w;
    };

    ~Square(){  // Class destructor
    };

    float area(){   // Area member function definition
        return width*width;
    };

protected:
    float width;    // Width is a memeber variable

};
```

```
Square s3 = Square(10.0);   // Instantiating a square object
std::cout << s3.area() << std::endl; // Prints out 100.0
s3.setColor("blue");    // Inhereted from shape
std::cout << s3.returnColor() << std::endl; // Returns blue
```

Challenge: Inheritance

- A subclass must declare what access specifier it inherits from
- For most cases you will use public
- Private members cannot be inherited

Challenge: Overriding

- Subclasses can override parent functions in the subclass
- Subclass functions will be called instead of the parent class

```
class Square:public Shape
{
public:
    Square(float w){    // Class constructor
        width = w;
    };

    ~Square(){  // Class destructor
    };

    float area(){   // Area member function definition
        return width*width;
    };

    void returnColor()
    {
        std::cout << "Override" << std::endl;
    };

protected:
    float width;    // Width is a memeber variable

};
```

```
class Shape
{
public:
    Shape() {
    };

    ~Shape() {
    };

    void setColor(std::string aColor)
    {
        color = aColor;
    };

    std::string returnColor()
    {
        return color;
    };

protected:
    std::string color;
};
```

```
Square s3 = Square(10.0);   // Instantiating a square object
std::cout << s3.area() << std::endl; // Prints out 100.0
s3.setColor("blue");    // Inhereted from shape
s3.returnColor();   // Prints out Override
```

Challenge: Header and Source Files

- Let us split up our code into multiple files

```cpp
#include "Rectangle.h"

Rectangle::Rectangle()
{
    // This is the constructor it is called every time
}

Rectangle::~Rectangle()
{
    // This is the destructor
}

void Rectangle::setArea(float area)
{
    m_area = area;
}

float Rectangle::returnArea()
{
    return m_area;
}
```

```cpp
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle {

public:
    Rectangle();    // The constructor prototype
    ~Rectangle();   // The destructor prototype

    void setArea(float area);   // Setter
    float returnArea(); // Getter

protected:
    float m_area;   // Member variable
};

#endif RECTANGLE_H
```

```cpp
// main.cpp : Defines the entry point for the console application.
//

#include "Rectangle.h"  // Coppies all the rectangle code into this file
#include <iostream>

int main()
{
    Rectangle aRec;
    aRec.setArea(45.0);
    std::cout << "Area: " << aRec.returnArea() << std::endl;

    return 0;
}
```

Questions?

Assignment

- Make classes for rectangle, circle, triangle that inherit from shape
- Use the functions you have been working on in your classes
- Prompt the user to select a shape and to input values to calculate the area of the shape
- Challenge: Look up model, view, controller and structure your code that way