CSCI 200: Foundational Programming Concepts & Design Lecture 27

Object-Oriented Programming:

Inheritance

Previously in CSCI 200

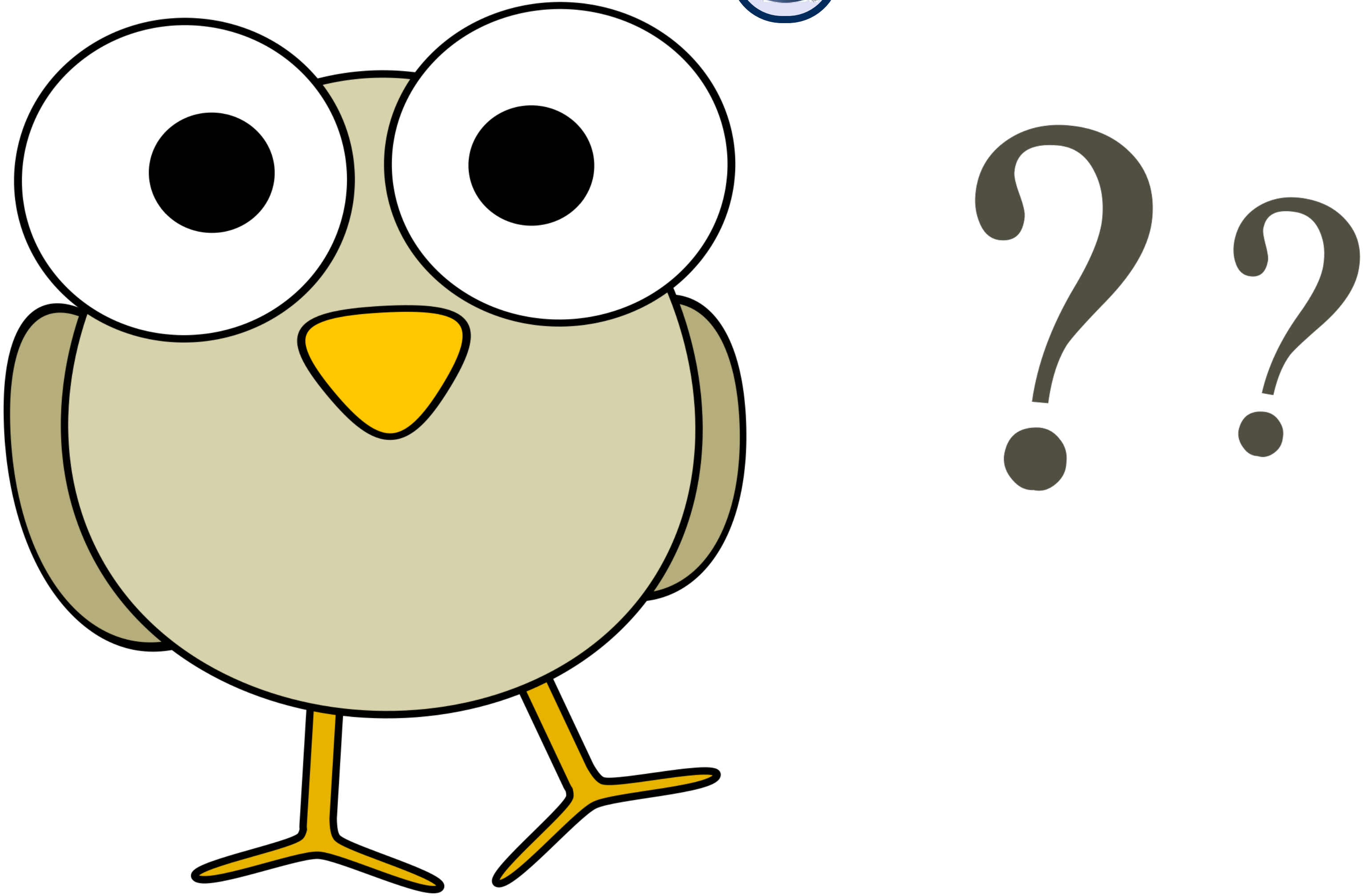
- Libraries
  - Tell compiler where library headers are located

    ```
    g++ -o main.o -c main.cpp -I/folders/include
    ```

  - Tell linker where library archives are located
  - Tell linker which libraries to link against

    ```
    g++ -o P.exe main.o -L/folders/lib -llibName
    ```

Questions?



Learning Outcomes For Today

- Discuss the concept of encapsulation
- Discuss what inheritance is and situations it should be used
- Draw a class diagram using UML to describe the structure of a class, its members, and its parents
- Create a child/derived class that inherits data members and member functions from a parent/base class

On Tap For Today

- Object-Oriented Programming
  - Classes & Objects
  - Inheritance
- Practice

On Tap For Today

- Object-Oriented Programming
  - Classes & Objects
  - Inheritance

- Practice

Object-Oriented Programming

- Imperative Programming where program state is encapsulated in a series of objects
  - Only objects can manipulate their own state

```
int main() {
  SumMachine summer;
  summer.reset( 0 );
  summer.setRange( 1, 10 );
  summer.sum();
  cout << "The sum is: ” << summer.getSum() << endl;
  return 0;
}
```

Object-Oriented Programming (OOP)

- Programming paradigm that groups like attributes and behaviors into a class

- Program to the domain
  - Use terminology and objects that are present in the field

On Tap For Today

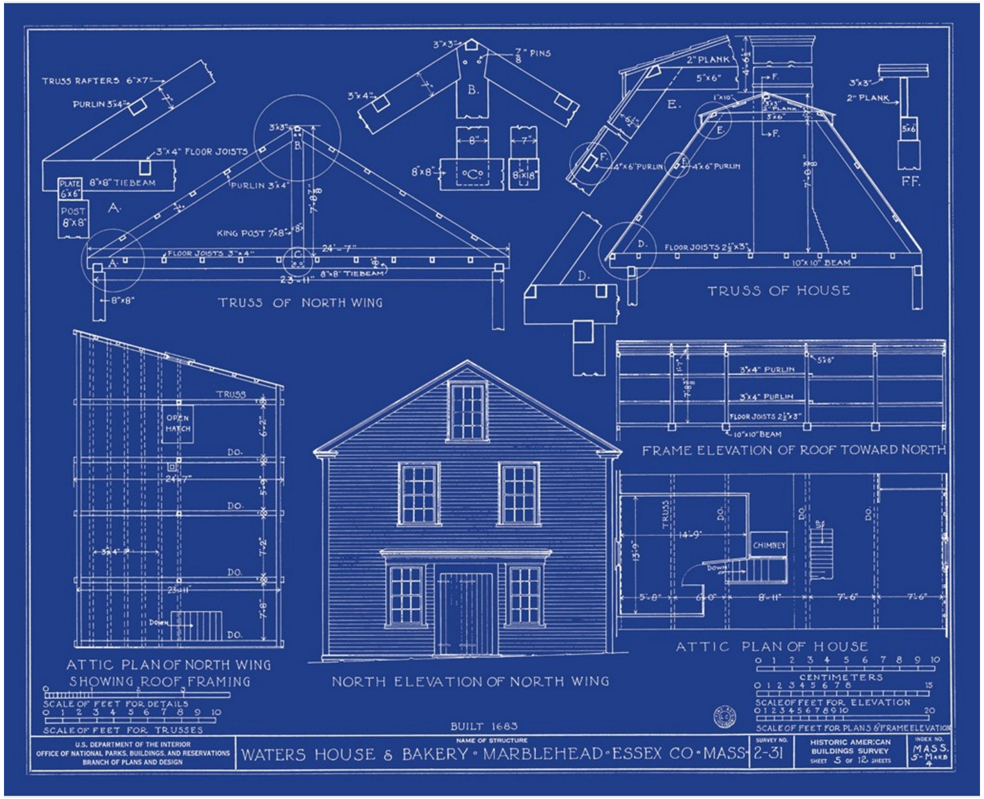
- Object-Oriented Programming
  - Classes & Objects
  - Inheritance
- Practice

Representing other things

- Create a **class** to represent a complex thing
  - A class **encapsulates** attributes (variables) and behavior (functions) of real world things
  - Attributes
  - Behaviors
  - Abstraction!

Classes & Objects

- A `class` is a blueprint

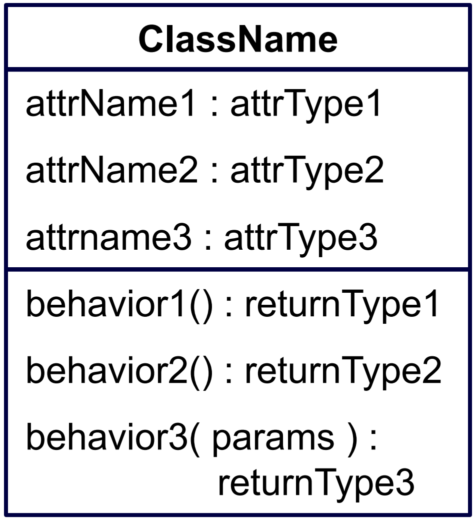


Classes & Objects

- An `object` is an instance of the `class`
- An `object` provides values for the data members of a `class`

Creating a Class Diagram

- Uses Unified Modeling Language (UML) to show structure of a class
- List attributes and behaviors of a class



ClassName

attrName1 : attrType1

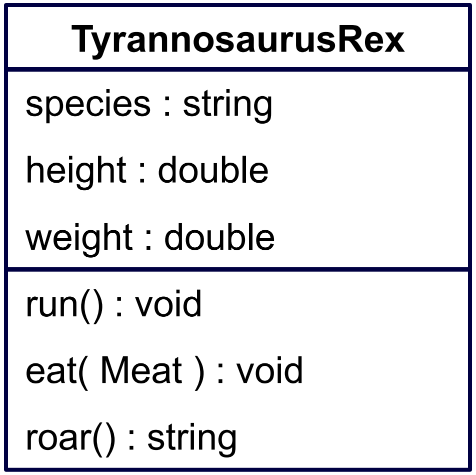
attrName2 : attrType2

attrname3 : attrType3

behavior1() : returnType1

behavior2() : returnType2

behavior3(params) : returnType3



TyrannosaurusRex

species : string

height : double

weight : double

run() : void

eat(Meat) : void

roar() : string

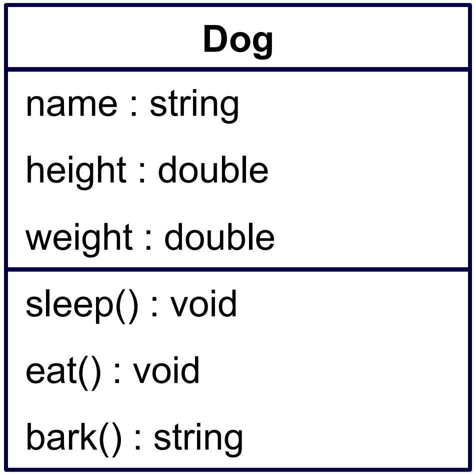
Object-Oriented Programming

- Classes exhibit
  - “Has-A” relationships with its own attributes & state
  - “Is-A” relationships with common ancestors that share attributes & state

On Tap For Today

- Object-Oriented Programming
  - Classes & Objects
  - Inheritance
- Practice

Consider These Classes



Dog

name : string

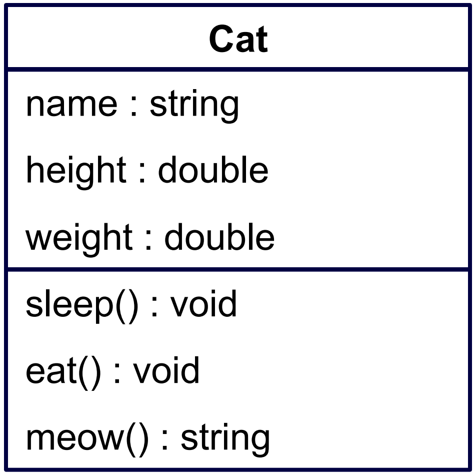
height : double

weight : double

sleep() : void

eat() : void

bark() : string



Cat

name : string

height : double

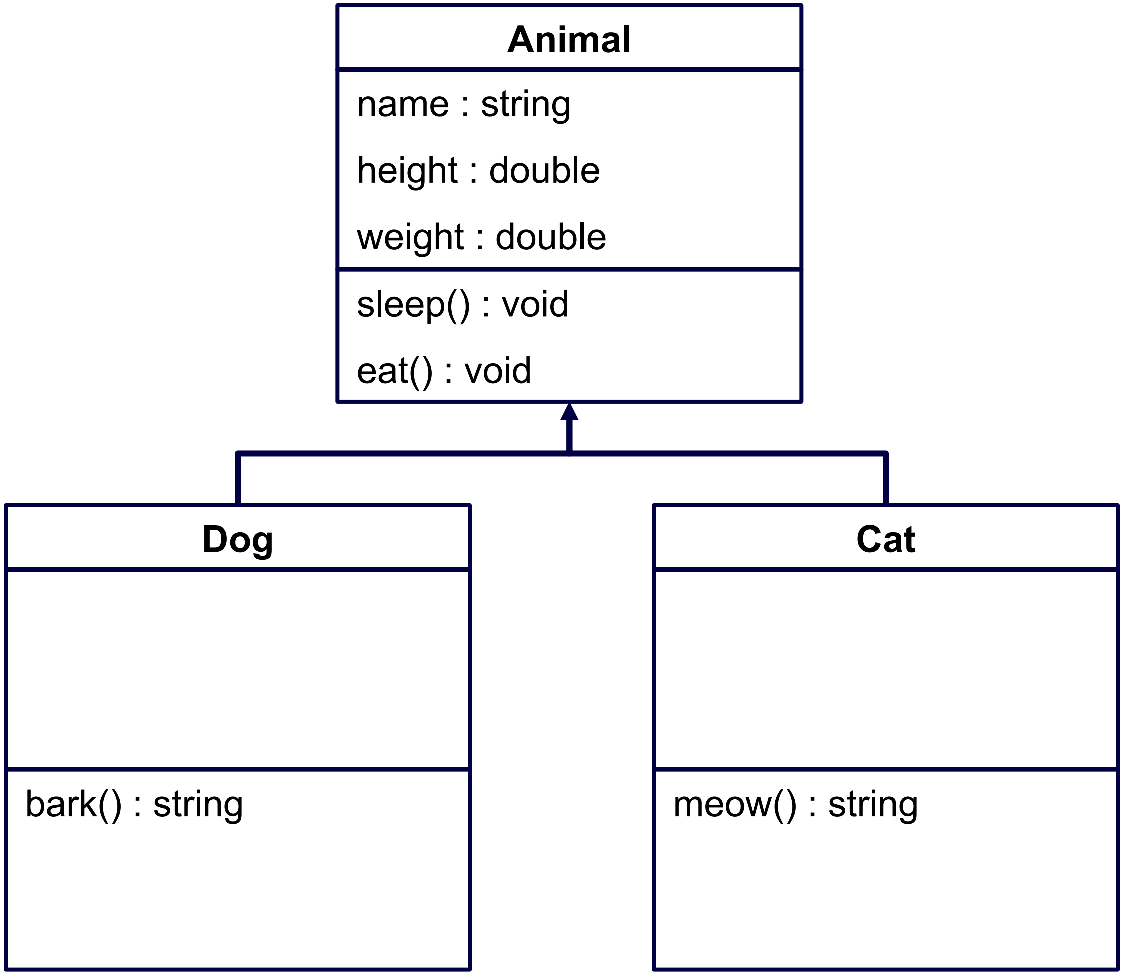
weight : double

sleep() : void

eat() : void

meow() : string

Consider These Classes



Animal

name : string

height : double

weight : double

sleep() : void

eat() : void

Dog

bark() : string

Cat

meow() : string

Creating The Base Class

```
class Animal {
public:
  Animal() { cout << "Creating an animal" << endl; }
  ~Animal() { cout << "Destroying an animal" << endl; }
  string getName() const { return _name; }
  void setName(const string NEW_NAME) { _name = NEW_NAME; }
  // other getters/setters
private:
  string _name;
  double _height;
  double _weight;
};
```

Creating The Derived Classes

```
class Dog : public Animal {
public:
  Dog() { cout << "Creating a dog" << endl; }
  ~Dog() { cout << "Destroying a dog" << endl; }
  void bark() { cout << "Woof" << endl; }
private:
};
class Cat : public Animal {
public:
  Cat() { cout << "Creating a cat" << endl; }
  ~Cat() { cout << "Destroying a cat" << endl; }
  void meow() { cout << "Meow" << endl; }
private:
};
```

Using The Classes

```
int main() {
  Animal anAnimal;  anAnimal.setName( "John" );
  Dog odie;         odie.setName( "Odie" );
  Cat garfield;     garfield.setName( "Garfield" );

  cout << "Animal " << anAnimal.getName() << " can’t speak" << endl;

  cout << "Dog " << odie.getName() << " says ";
  dog.bark();

  cout << "Cat " << garfield.getName() << " says ";
  garfield.meow();

  return 0;
}
```

Class Access Modifiers

- `public`
  - Access inside & outside base class
- `protected`
  - Access inside base & inside derived class
- `private`
  - Access inside base class

Types of Inheritance

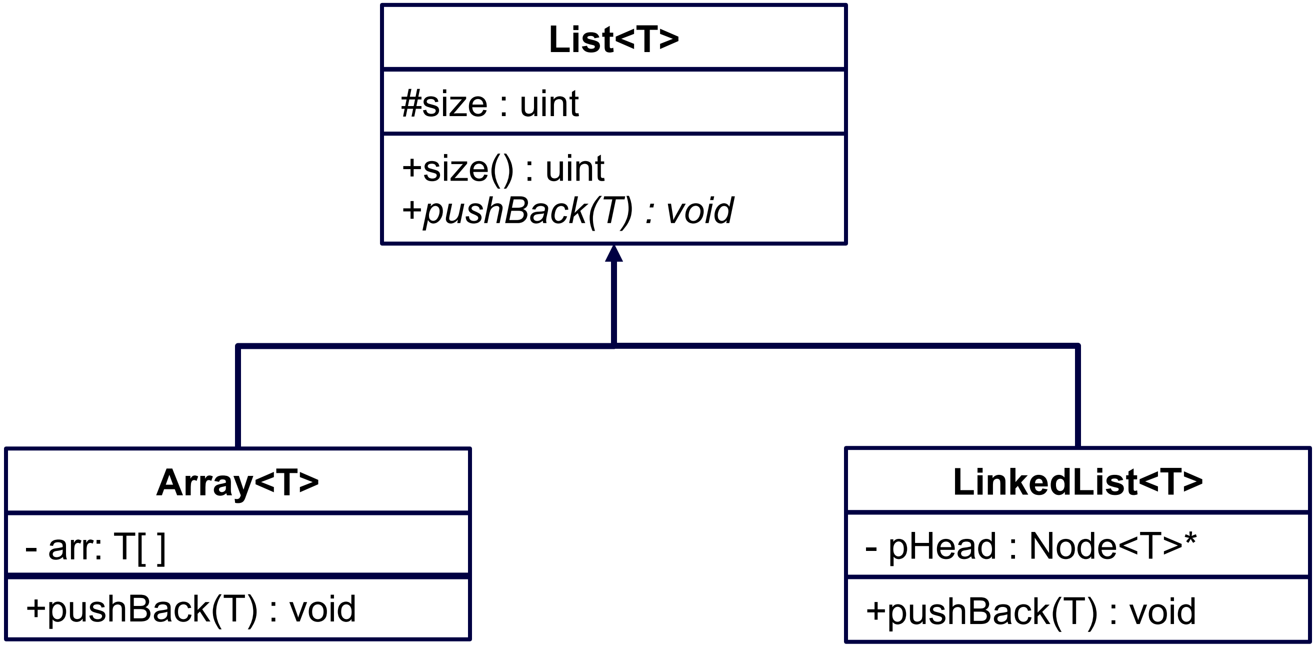
- `public` inheritance
- `protected` inheritance
- `private` inheritance

Inheritance Access

		Base Class Access Modifier		
		public	protected	private
Derived Class Inheritance Modifier	public	public	protected	
	protected	protected	protected	
	private	private	private	

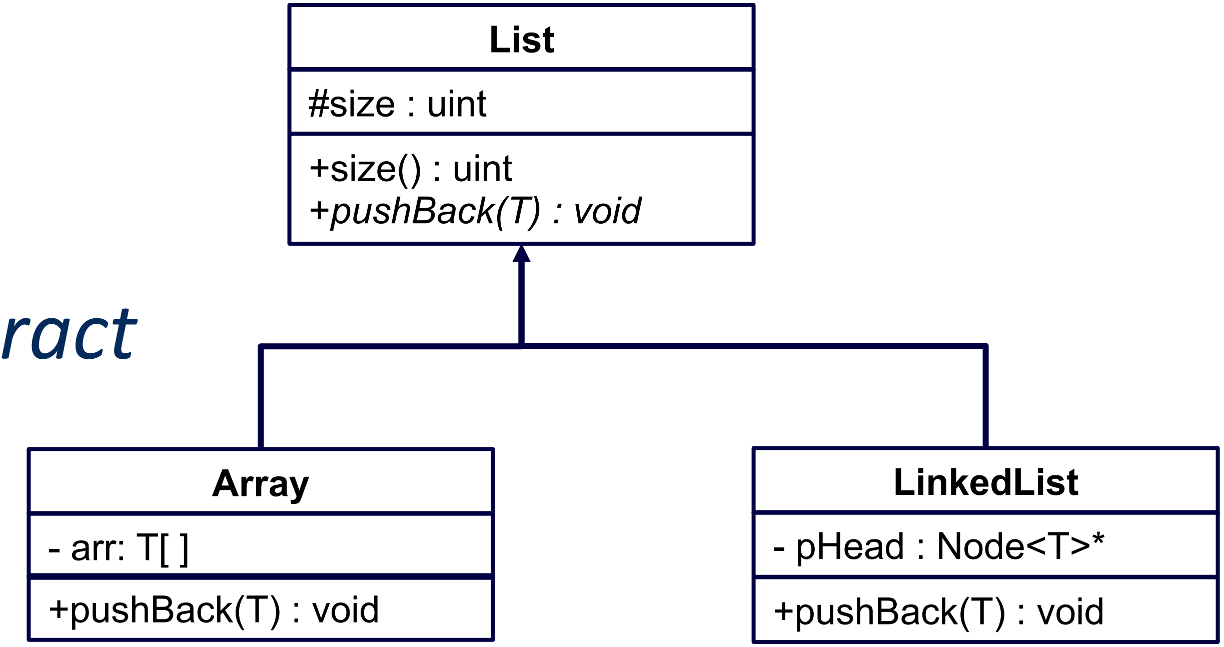
When To Use Which?

- Favor using only **`public`**/**`private`** access modifiers with **`public`** inheritance
  - Have good reason for using **`protected`**
  - Better solutions exist that we'll soon cover



UML Notation

- **ClassName**
- + public
- # protected
- - private
- ↑ extends
- *virtual / abstract*
  - Coming soon!



List

#size : uint

+size() : uint
+pushBack(T) : void

Array

- arr: T[ ]

+pushBack(T) : void

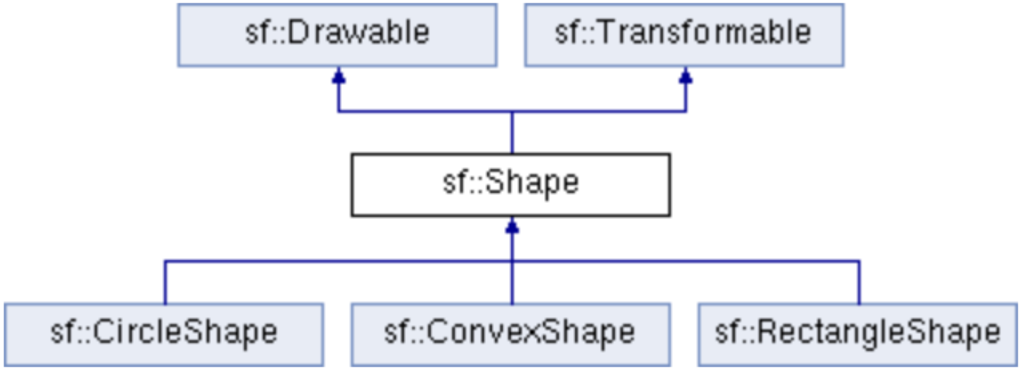
LinkedList

- pHead : Node<T>*

+pushBack(T) : void

SFML Classes

- CircleShape is a Shape
- Shape is both Drawable & Transformable
  - “Multiple Inheritance” – coming soon!



On Tap For Today

- Object-Oriented Programming
  - Classes & Objects
  - Inheritance
- Practice

To Do For Next Time

- Set4 due Tuesday