CMSC202 Computer Science II for Majors

Lecture 15 – Polymorphism (Continued)

Dr. Katherine Gibson

Last Class We Covered

- Review of inheritance
- Overriding (vs overloading)

- Understanding polymorphism
  - Limitations of inheritance
  - Virtual functions
  - Abstract classes & function types

Any Questions from Last Time?

Common Project 3 Error

- Have you seen something like this?

```
g++ -Wall -ansi -g -c test1.cpp

In file included from test1.cpp:4:
TrainCar.h:49: error: expected constructor, destructor,
or type conversion before '&' token
```

- The error is not in TrainCar.h!
  - Where is it?
    - The "error" is in `test1.cpp`
- Do **not** change the TrainCar.h file!

Today's Objectives

- Review of polymorphism
  - Limitations of inheritance
  - Virtual functions
  - Abstract classes & function types
- Finishing polymorphism
  - Virtual function Tables
  - Virtual destructors/constructors
- Livecoding application

Review of
Virtual and Polymorphism

Overview of Polymorphism

- Assume we have `Vehicle *vehiclePtr = &myCar;`
- And this method call: `vehiclePtr->Drive();`

prototype	Vehicle class	Car class
`void Drive()`		
`virtual void Drive()`		
`virtual void Drive() = 0`		

Overview of Polymorphism

- Assume we have `Vehicle *vehiclePtr = &myCar;`
- And this method call: `vehiclePtr->Drive();`

prototype	Vehicle class	Car class
`void Drive()`	• Can implement function • Can create Vehicle	• Can implement function • Can create Car • Calls `Vehicle::Drive`
`virtual void Drive()`	• Can implement function • Can create Vehicle	• Can implement function • Can create Car • Calls `Car::Drive`
`virtual void Drive() = 0`	• **Cannot** implement function • **Cannot** create Vehicle	• **Must** implement function • Can create Car • Calls `Car::Drive`

Overview of Polymorphism

- Assume we have `Vehicle *vehiclePtr = &myCar;`
- And this method call: `vehiclePtr->Drive();`

prototype	Vehicle class	Car class
`void Drive()`	• Can implement function • Can create Vehicle	• Can implement function rive
vi	lement functio te Vehicle	tion • Calls **Car::Drive**
`virtual void Drive() = 0`	• <u>**Cannot**</u> implement function • <u>**Cannot**</u> create Vehicle	• <u>**Must**</u> implement function • Can create Car • Calls **Car::Drive**

If no **Car::Drive** implementation, calls **Vehicle::Drive**

This is a ***pure virtual*** function, and Vehicle is now an ***abstract*** class

Virtual Function Tables

Behind the Scenes

- If our `Drive()` function is virtual, how does the compiler know which child class's version of the function to call?

vector of Car* objects

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	SUV

Virtual Function Tables

- The compiler uses ***virtual function tables*** whenever we use polymorphism

- Virtual function tables are created for:
  - Classes with virtual functions
  - Child classes of those classes

Virtual Table Pointer

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	Van

Virtual Table Pointer

- The compiler adds a hidden variable

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	Van
*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr

Virtual Table Pointer

- The compiler also adds a virtual table of functions for each class

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	Van
*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr

Virtual Table Pointer

- Each virtual table has pointers to each of the virtual functions of that class

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	Van
*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr	*__vptr

SUV virtual table
* to SUV::Drive();

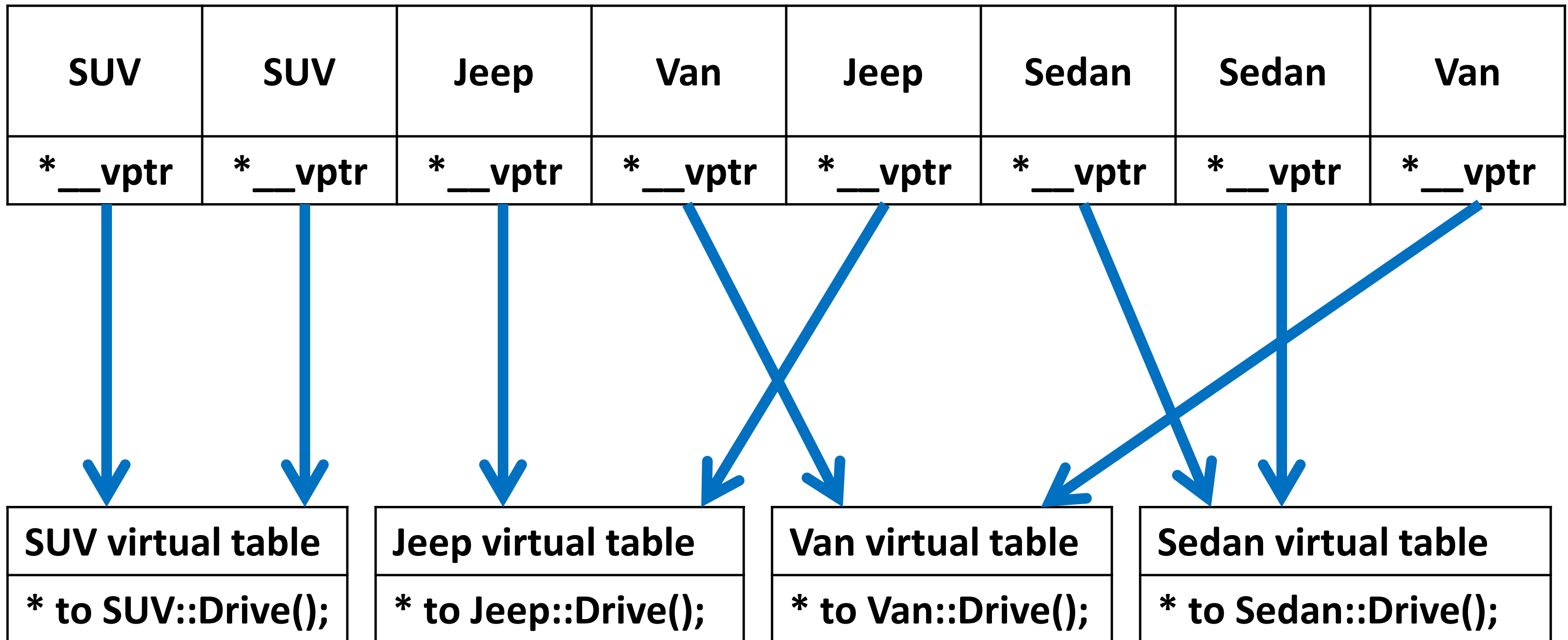
Jeep virtual table
* to Jeep::Drive();

Van virtual table
* to Van::Drive();

Sedan virtual table
* to Sedan::Drive();

Virtual Table Pointer

- The hidden variable points to the appropriate virtual table of functions



Virtual Destructors/Constructors

Virtual Destructors

```
Vehicle *vehicPtr = new Car;
delete vehicPtr;
```

- For any class with virtual functions, you must declare a virtual destructor as well
- Why?
  - Non-virtual destructors will only invoke the base class's destructor

Virtual Constructors

- Not a thing... why?

- We use polymorphism and virtual functions to manipulate objects **without** knowing type or having complete information about the object

- When we construct an object,
we **have** complete information
  - There's no reason to have a virtual constructor

Livecoding

- Animals (Bird, Cat, and Dog)
  - All Animals can: Eat(), Speak(), and Perform()
- Vector of Animal pointers – what happens?

LIVECODING!!!

Announcements

- Project 3 is due tonight!
- Exam 2 is in 1 week
  - Will focus heavily on:
    - Classes
    - Inheritance
    - Linked Lists
    - Dynamic Memory
    - Some Polymorphism