CIS 190: C/C++ Programming

Polymorphism

Outline

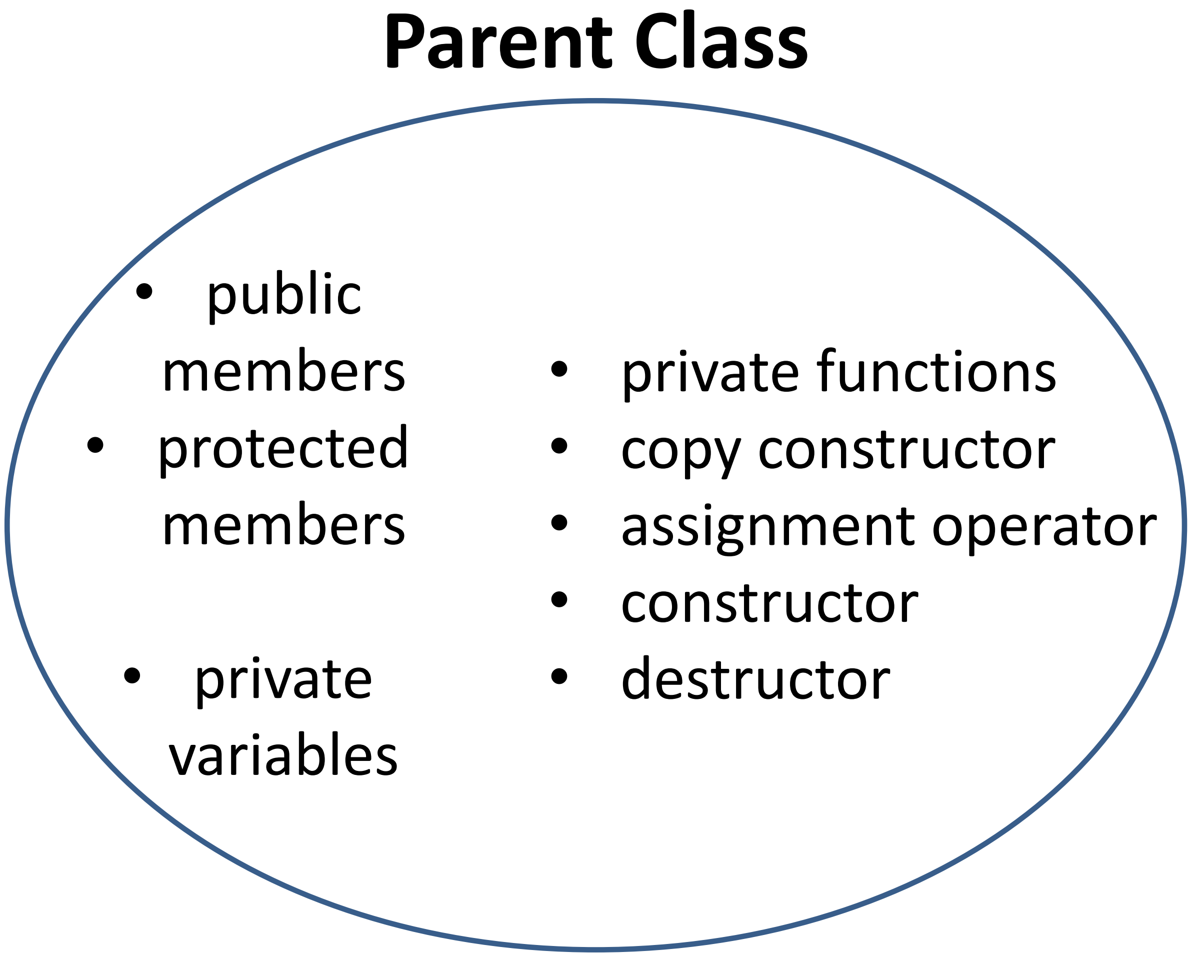
- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Review of Inheritance

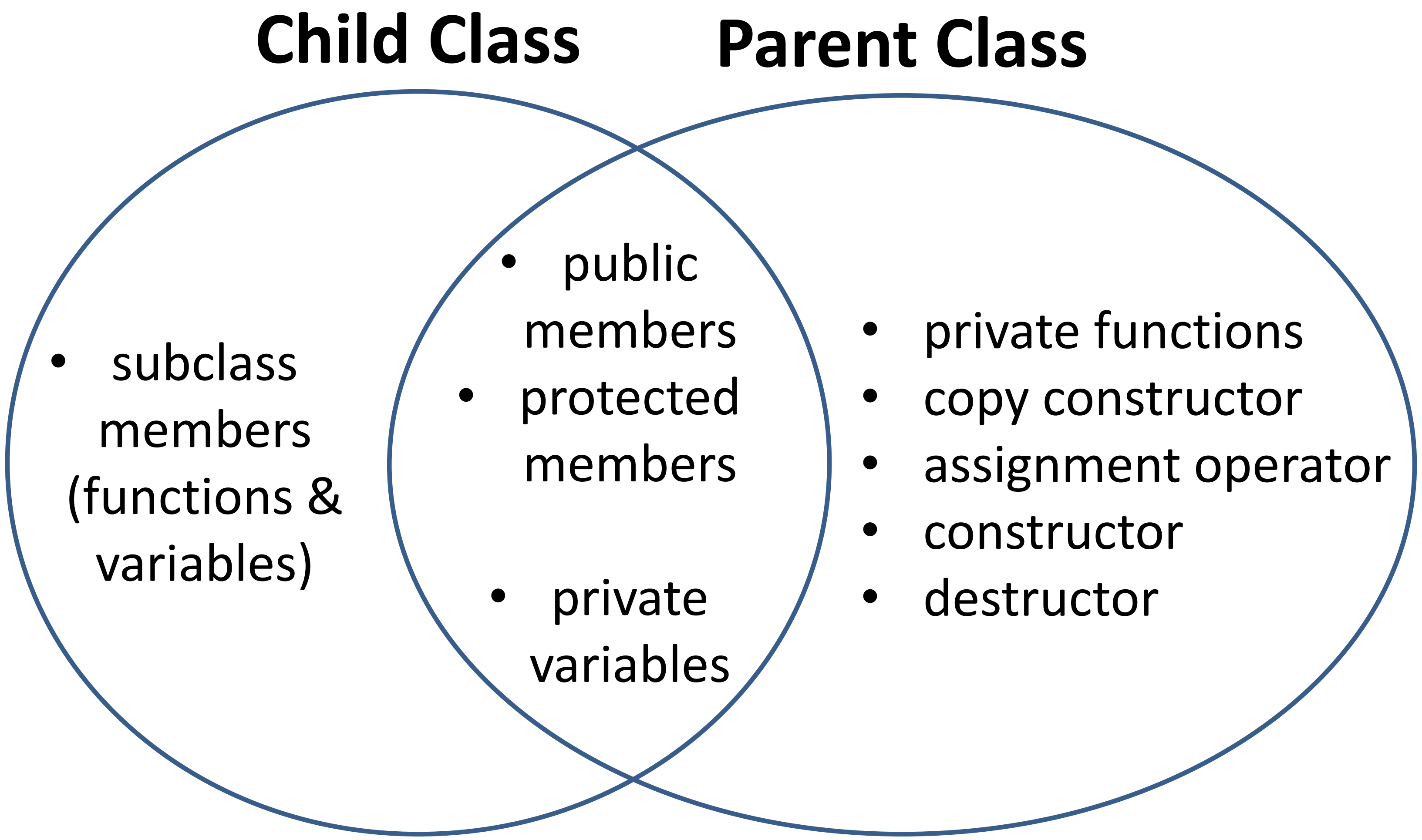
- specialization through sub classes

- child class has direct access to:
  - parent member functions and variables that are:
    - public
    - protected
- parent class has direct access to:
  - **nothing** in the child class

What is Inherited



What is Inherited



Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

What is Polymorphism?

- ability to manipulate objects in a type-independent way

- already done to an extent via overloading

- can take it further using subtyping, AKA *inclusion polymorphism*

Using Polymorphism

- only possible by using pointers to objects

- a pointer of a parent class type can point to an object of any child class type

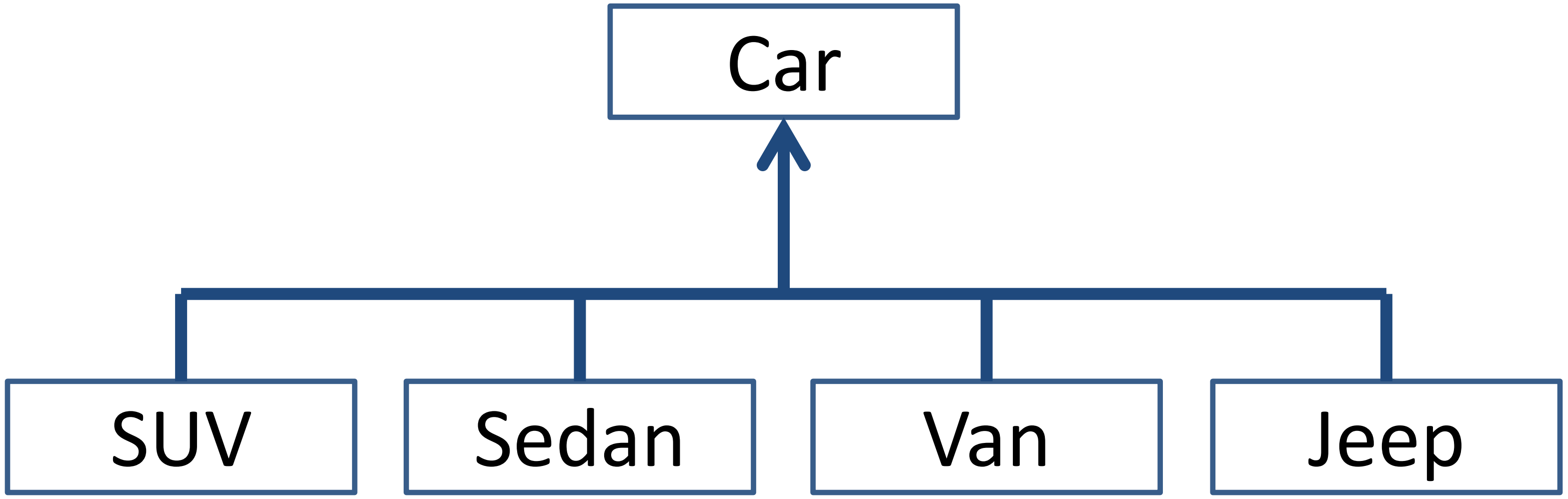
```
Vehicle *vehicPtr = &myCar;
```

- this is valid because **`myCar`** is-a **`Vehicle`**

Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Car Example



```
class SUV:   public Car {/*etc*/};
class Sedan: public Car {/*etc*/};
class Van:   public Car {/*etc*/};
class Jeep:  public Car {/*etc*/};
```

Car Example: Car Rental

- implement a catalog of cars available for rental

- how could we do this (using vectors)?

Car Example: Car Rental

- implement a catalog of cars available for rental

- two options:
  - separate vector for each type of Car (SUV, Van, etc.)
    - have to add a new vector if we add new type
    - must have separate variables for each vector
  - single vector of Car pointers
    - no changes necessary if we add new type

Car Example: Car* vector

```
vector <Car*> rentalList;
```

vector of Car* objects

SUV	SUV	Jeep	Van	Jeep	Sedan	Sedan	SUV

Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Polymorphism Limitations

- parent classes **do not** inherit from child classes
  - **not even** public member variables and functions

```
vehicPtr->PrintSpecs();
```

– will call Vehicle’s PrintSpecs() function, not Car’s

```
vehicPtr->Drive();
```

– will not work; Drive() is a function only of the Car class, and vehicPtr can’t access it

Virtual Functions

- can grant access to child methods by using *virtual functions*

- to do this, declare the function in the parent class with the keyword **`virtual`**
  - can also use virtual keyword in child class, but not required

Virtual Function Example

```
class Vehicle{
   virtual void Drive();
   /* rest of vehicle class */
}
class Car: public Vehicle {
   void Drive();
   /* rest of car class */
}
```

Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Function Types – Pure Virtual

```
virtual void Drive() = 0;
```

- denoted with an “= 0” at end of declaration
- this makes the class an *abstract class*

- child classes **must** have an implementation of the pure virtual function

- **cannot declare objects of abstract class types**

Function Types – Virtual

```
virtual void Drive();
```

- parent class must have an implementation
- child classes may override if they choose to
  - if not overridden, parent class definition used

Function Types – Non-Virtual

```
void Drive();
```

- parent class should have an implementation
- child class **cannot** override function
  - parent class definition always used
- should be used only for functions that won’t be changed by child classes

Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Behind the Scenes

- assume our **`Drive()`** function is pure 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 Tables

- lookup tables of functions
  - employed when we use polymorphism

- virtual tables are created for:
  - classes with virtual functions
  - child classes derived from those classes

- handled by compiler behind the scenes

Virtual Table Pointer

- compiler adds a hidden variable that points to the appropriate virtual table of functions

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

Virtual Table Pointer

- compiler adds a hidden variable that points to the appropriate virtual table of functions

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

Virtual Table Pointer

- compiler adds a hidden variable that points to the appropriate virtual table of functions

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

SUV virtual table

Jeep virtual table

Van virtual table

Sedan virtual table

Virtual Table Pointer

- compiler adds a hidden variable that points to the appropriate virtual table of functions

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

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

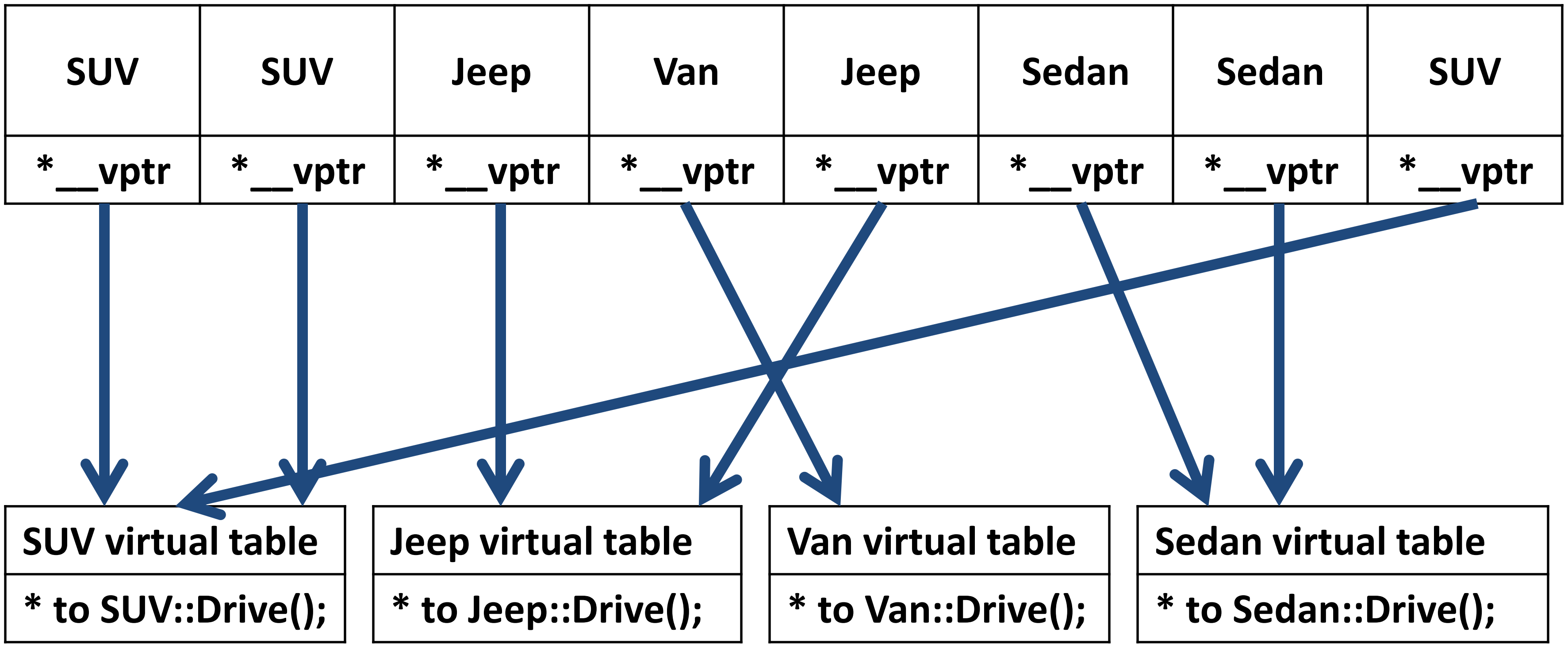
Jeep virtual table
*** to Jeep::Drive();**

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

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

Virtual Table Pointer

- compiler adds a hidden variable that points to the appropriate virtual table of functions



Outline

- Review of Inheritance
- Polymorphism
  - Car Example
  - Virtual Functions
    - Virtual Function Types
  - Virtual Table Pointers
  - Virtual Constructors/Destructors

Virtual Destructors

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

- non-virtual destructors will only invoke the base class's destructor

- for any class with virtual functions, you must declare a virtual destructor as well

Virtual Constructors

- not a thing... why?

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

Project Alphas

- due next Monday (April 14th)

- doesn't:
  - have to be working
  - a complete project

- in a folder named <your_team_name>