

CMSC202

Computer Science II for Majors

Lecture 10 and 11 – Inheritance

Dr. Katherine Gibson

Last Class We Covered

- Professor Chang substitute taught
- Allocation methods
 - Static, automatic, dynamic
 - **new** and **delete**
 - Dynamically allocating arrays
 - Constructors and destructors

Any Questions from Last Time?

Today's Objectives

- To review the exam results
- To understand the relationships between objects
- To begin learning about inheritance
 - To cover what is being inherited
 - To understand how inheritance and access to member variables interact

Exam 1 Results

Code Reuse

- Important to successful coding
- Efficient
 - No need to reinvent the wheel
- Error free
 - Code has been previously used/tested
 - (Not guaranteed, but more likely)

- What are some ways we reuse code?
 - functions
 - classes

- Any specific examples?
 - calling `Insert()` and a modified `Delete()` for `Move()`
 - calling accessor functions inside a constructor

- What are some ways we reuse code?
 - Functions
 - Classes
 - Inheritance – what we'll be covering today
- Any specific examples?

Object Relationships

- ***Objects*** are what we call an ***instance*** of a ***class***
- For example:
 - **Rectangle** is a class
 - **r1** is a variable of type **Rectangle**
 - **r1** is a **Rectangle** object

- There are two types of object relationships
 - **is-a**
 - inheritance
 - **has-a**
 - composition
 - aggregation
- } both are forms of ***association***

A Car *is-a* Vehicle

- This is called *inheritance*
- The Car class *inherits* from the Vehicle class
- Vehicle is the general class, or the *parent class*
- Car is the specialized class, or *child class*, that inherits from Vehicle

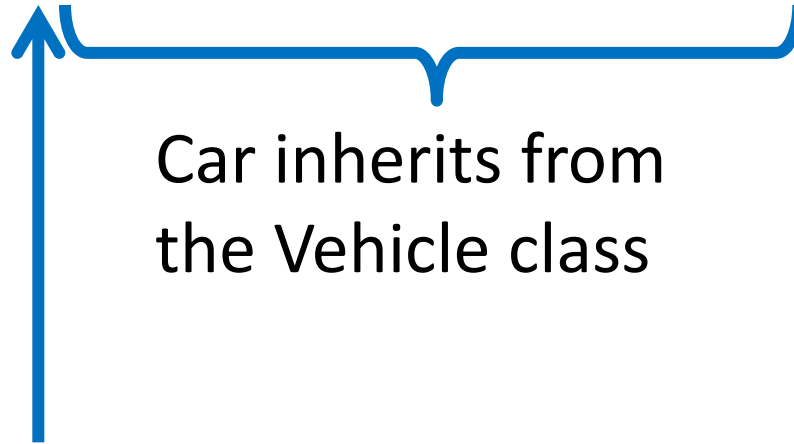
```
class Vehicle {  
    public:  
        // functions  
    private:  
        int     m_numAxles;  
        int     m_numWheels;  
        int     m_maxSpeed;  
        double  m_weight;  
        // etc  
};
```

} all Vehicles have axles, wheels, a max speed, and a weight

```
class Car {
```

```
} ;
```

```
class Car: public Vehicle {
```



Car inherits from
the Vehicle class

don't forget the
colon here!

```
} ;
```



```
class Car: public Vehicle {  
    public:  
        // functions  
    private:  
        int     m_numSeats;  
        double  m_MPG;  
        string  m_color;  
        string  m_fuelType;  
        // etc  
};
```

} all Cars have a number of seats, a MPG value, a color, and a fuel type

```
class Car:
    public Vehicle { /*etc*/ };

class Plane:
    public Vehicle { /*etc*/ };

class SpaceShuttle:
    public Vehicle { /*etc*/ };

class BigRig:
    public Vehicle { /*etc*/ };
```


A Car *has-a* Chassis

- This is called *composition*
- The Car class *contains* an object of type Chassis
- A Chassis object is part of the Car class
- A Chassis cannot “live” out of context of a Car
 - If the Car is destroyed, the Chassis is also destroyed

```
class Chassis {  
    public:  
        // functions  
    private:  
        string m_material;  
        double m_weight;  
        double m_maxLoad;  
        // etc  
};
```

} all Chassis have a material, a weight, and a maxLoad they can hold

```
class Chassis {  
    public:  
        // functions  
    private:  
        string m_material;  
        double m_weight;  
        double m_maxLoad;  
        // etc  
};
```



also, notice
that there is
no inheritance
for the Chassis
class

```
class Car: public Vehicle {
    public:
        // functions
    private:
        // member variables, etc.

        // has-a (composition)
        Chassis m_chassis;
} ;
```

a Car *has-a* Driver

- this is called *aggregation*

A Car *has-a* Driver

- This is called *aggregation*
- The Car class is *linked to* an object of type Driver
- Driver class is not directly related to the Car class
- A Driver **can** live out of context of a Car
- A Driver must be “contained” in the Car object via a pointer to a Driver object


```
class Driver: public Person {  
    public:  
        // functions  
    private:  
        Date    m_licenseExpire;  
        string  m_licenseType;  
        // etc  
};
```

Driver itself is a child class of Person

```
class Driver: public Person {  
    public:  
        // functions  
    private:  
        Date    m_licenseExpire;  
        string  m_licenseType;  
        // etc  
};
```

Driver itself is a child class of Person

Driver inherits all of Person's member variables (Date m_age, string m_name, etc.) so they aren't included in the Driver child class

```
class Car: public Vehicle {
    public:
        // functions
    private:
        // member variables, etc.

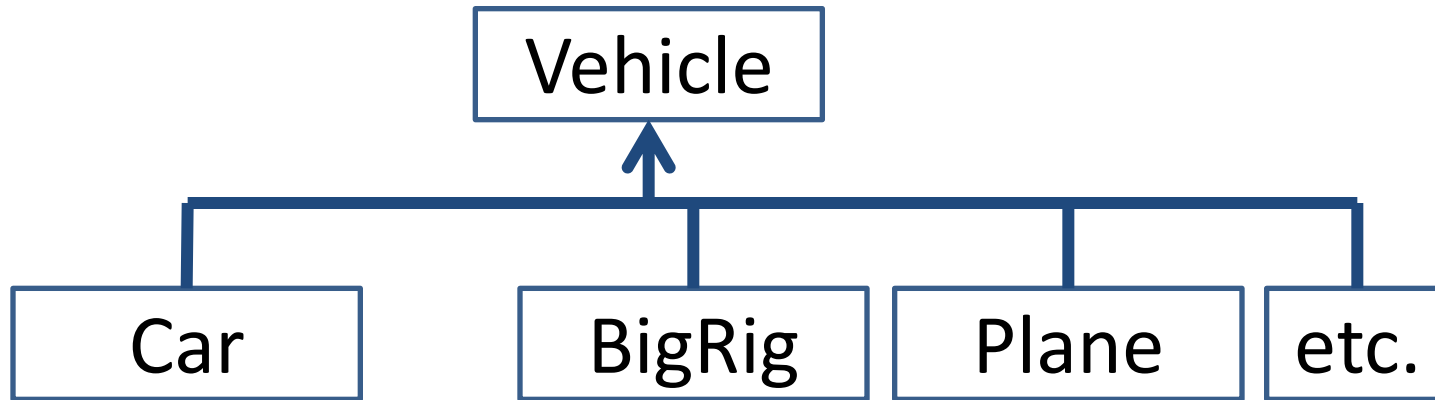
        // has-a (aggregation)
        Person *m_driver;
} ;
```

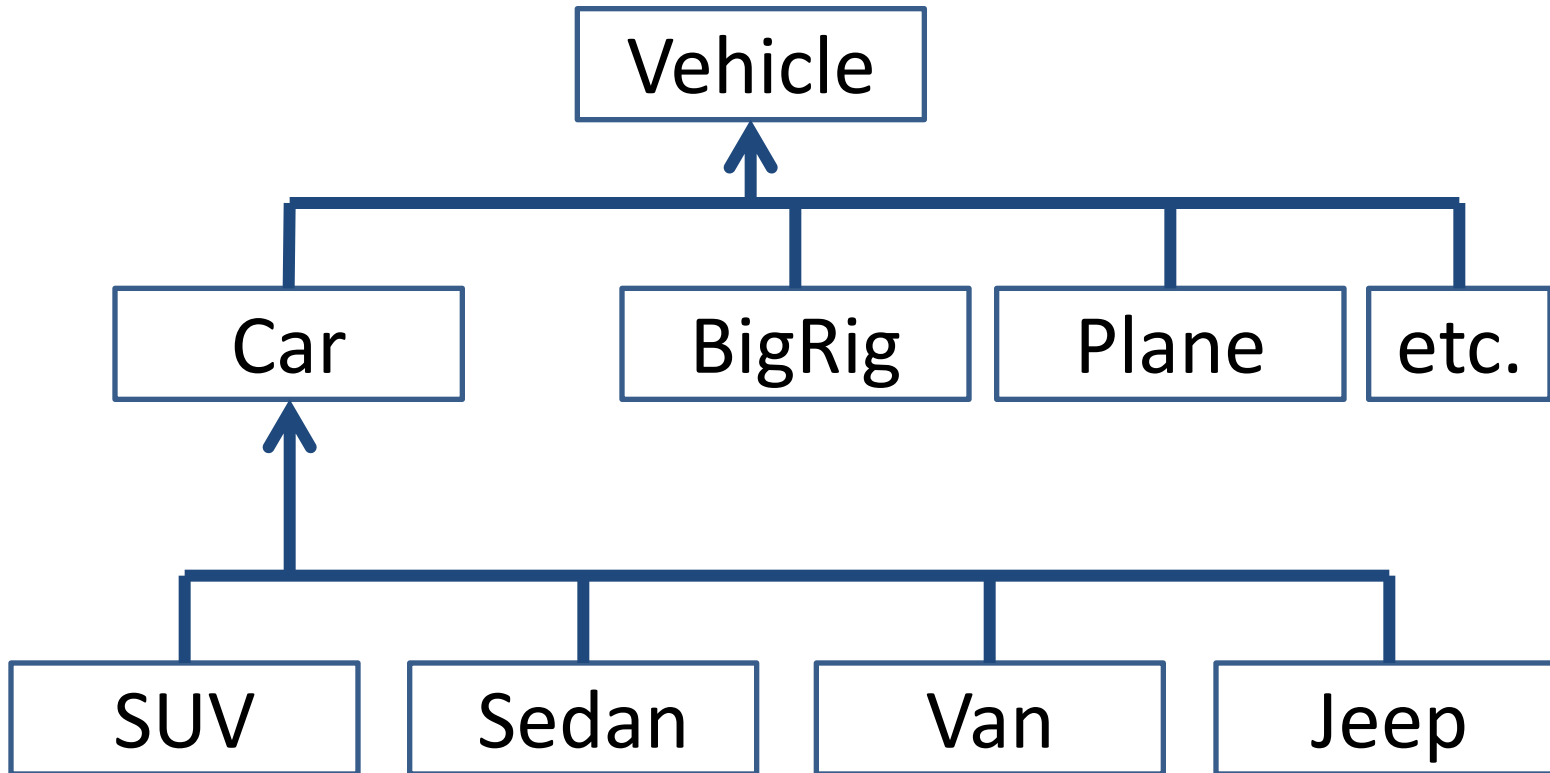
- On paper, draw a representation of how the following objects relate to each other
 - Make sure the type of relationship is clear
-
- Car
 - Vehicle
 - BigRig
 - Rectangle
 - SpaceShuttle
 - Engine
 - Driver
 - Person
 - Owner
 - Chassis

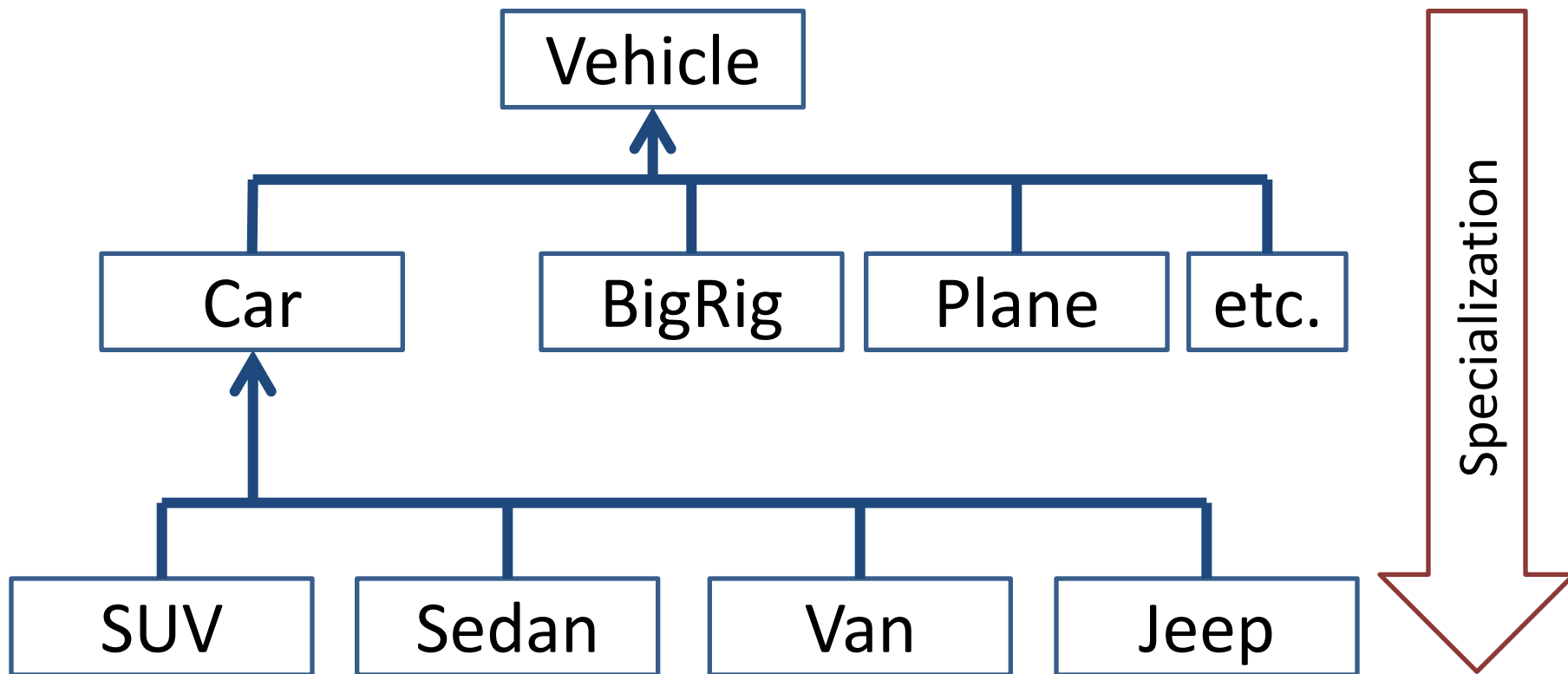
Inheritance

- inheritance can be done via public, private, or protected
- we're going to focus exclusively on public
- you can also have multiple inheritance
 - where a child class has more than one parent
- we won't be covering this

Vehicle







- **more general class** (e.g., Vehicle) can be called:
 - parent class
 - base class
 - superclass
- **more specialized class** (e.g., Car) can be called:
 - child class
 - derived class
 - subclass

- parent class contains all that is common among its child classes (less specialized)
 - Vehicle has a maximum speed, a weight, etc.
because all vehicles have these
- member variables and functions of the parent class are inherited by **all** of its child classes

- child classes can use, extend, or replace the parent class behaviors

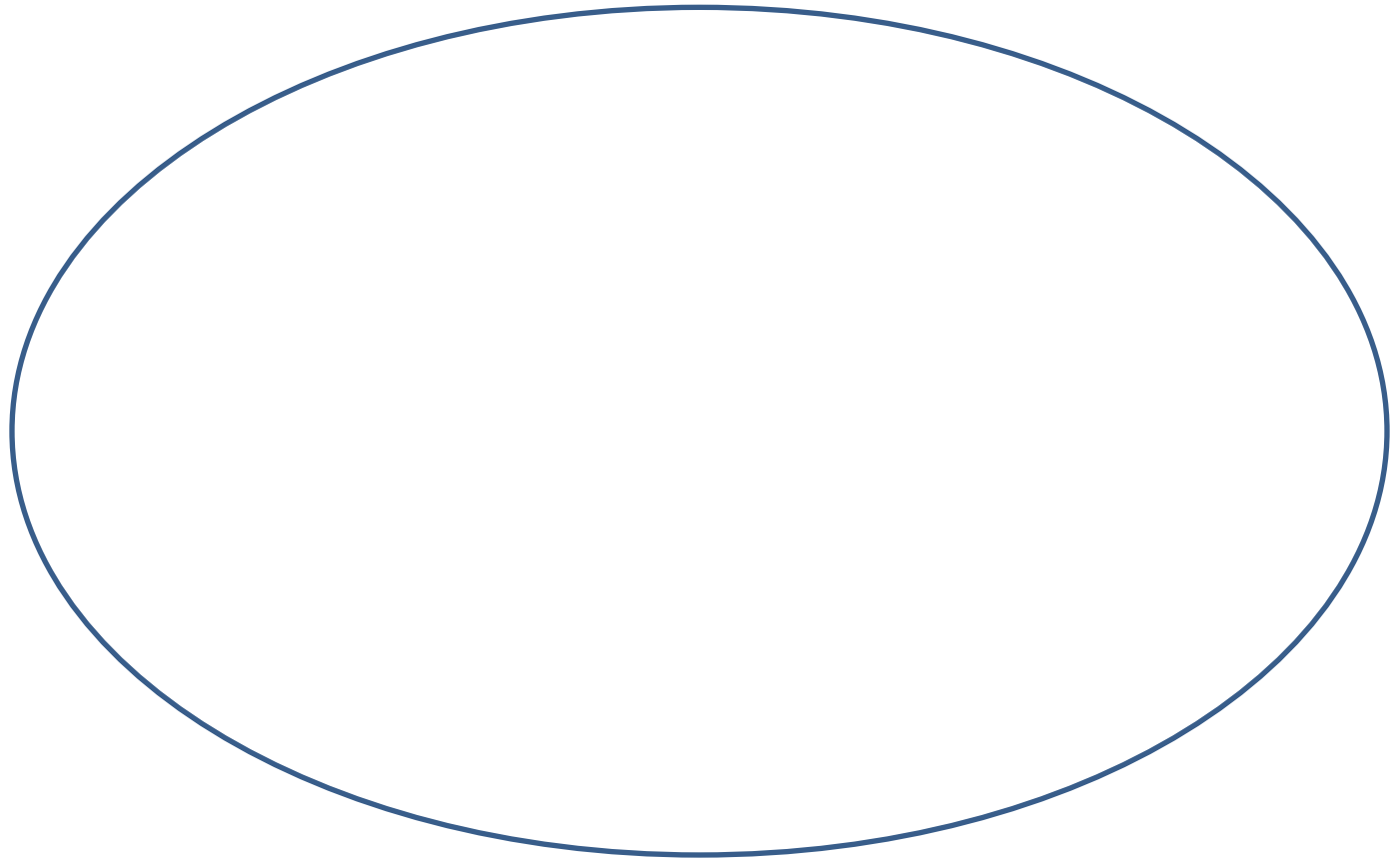
- child classes can **use**, extend, or replace the parent class behaviors
- use
 - the child class takes advantage of the parent class behaviors exactly as they are
 - like the mutators and accessors from the parent class

- child classes can use, **extend**, or replace the parent class behaviors
- **extend**
 - the child class creates entirely new behaviors
 - a **RepaintCar()** function for the Car child class
 - mutators/accessors for new member variables

- child classes can use, extend, or **replace** the parent class behaviors
- replace
 - child class overrides parent class's behaviors
 - (we'll cover this later today)

- Code Reuse
- Object Relationships
- Inheritance
 - What is Inherited
 - Handling Access
- Overriding
- Homework and Project

Vehicle Class



Vehicle Class

- public fxns&vars

Vehicle Class

- public fxns&vars
- protected fxns&vars

Vehicle Class

- public fxns&vars
- protected fxns&vars
- private variables
- private functions

Vehicle Class

- public fxns&vars
- protected fxns&vars
- private variables
- private functions
- copy constructor
- assignment operator
- constructor
- destructor

Car Class

Vehicle Class

- 
- public fxns&vars
 - protected fxns&vars
 - private variables
 - private functions
 - copy constructor
 - assignment operator
 - constructor
 - destructor

Car Class

Vehicle Class

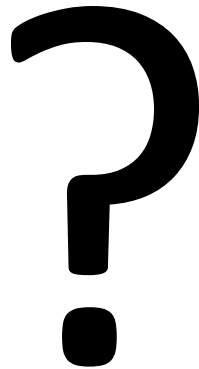
- child class members (functions & variables)

- public fxns&vars
- protected fxns&vars
- private variables
- private functions
- copy constructor
- assignment operator
- constructor
- destructor

Car Class

Vehicle Class

- child class members (functions & variables)



- public fxns&vars
- protected fxns&vars
- private variables
- private functions
- copy constructor
- assignment operator
- constructor
- destructor

Car Class

Vehicle Class

-
- child class members (functions & variables)
 - public fxns&vars
 - protected fxns&vars
 - private variables
 - private functions
 - copy constructor
 - assignment operator
 - constructor
 - destructor

Car Class

Vehicle Class

- child class members (functions & variables)

- public fxns&vars
- protected fxns&vars

- private variables
- private functions
- copy constructor
- assignment operator
- constructor
- destructor

Car Class

Vehicle Class

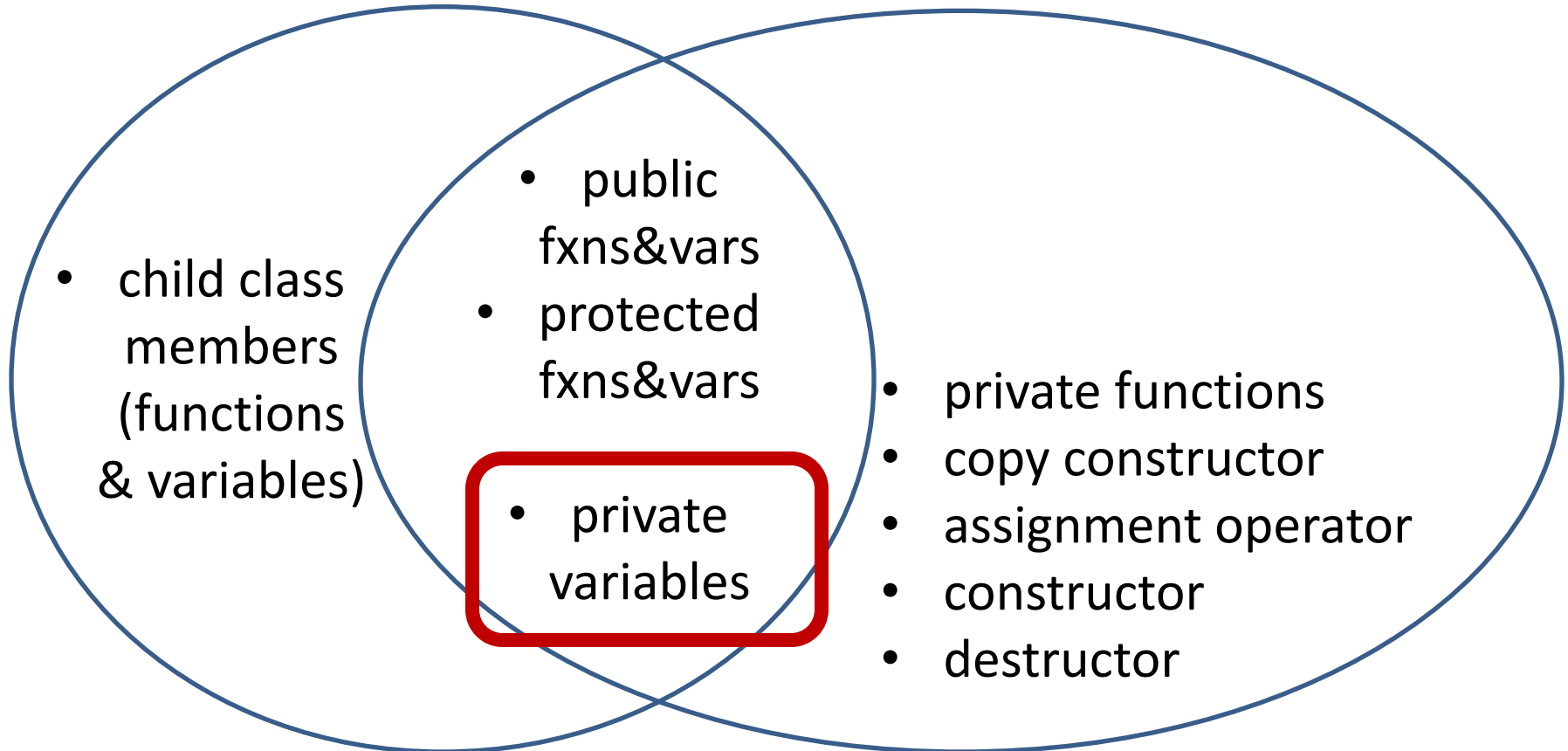
- child class members (functions & variables)

- public fxns&vars
- protected fxns&vars
- private variables

- private functions
- copy constructor
- assignment operator
- constructor
- destructor

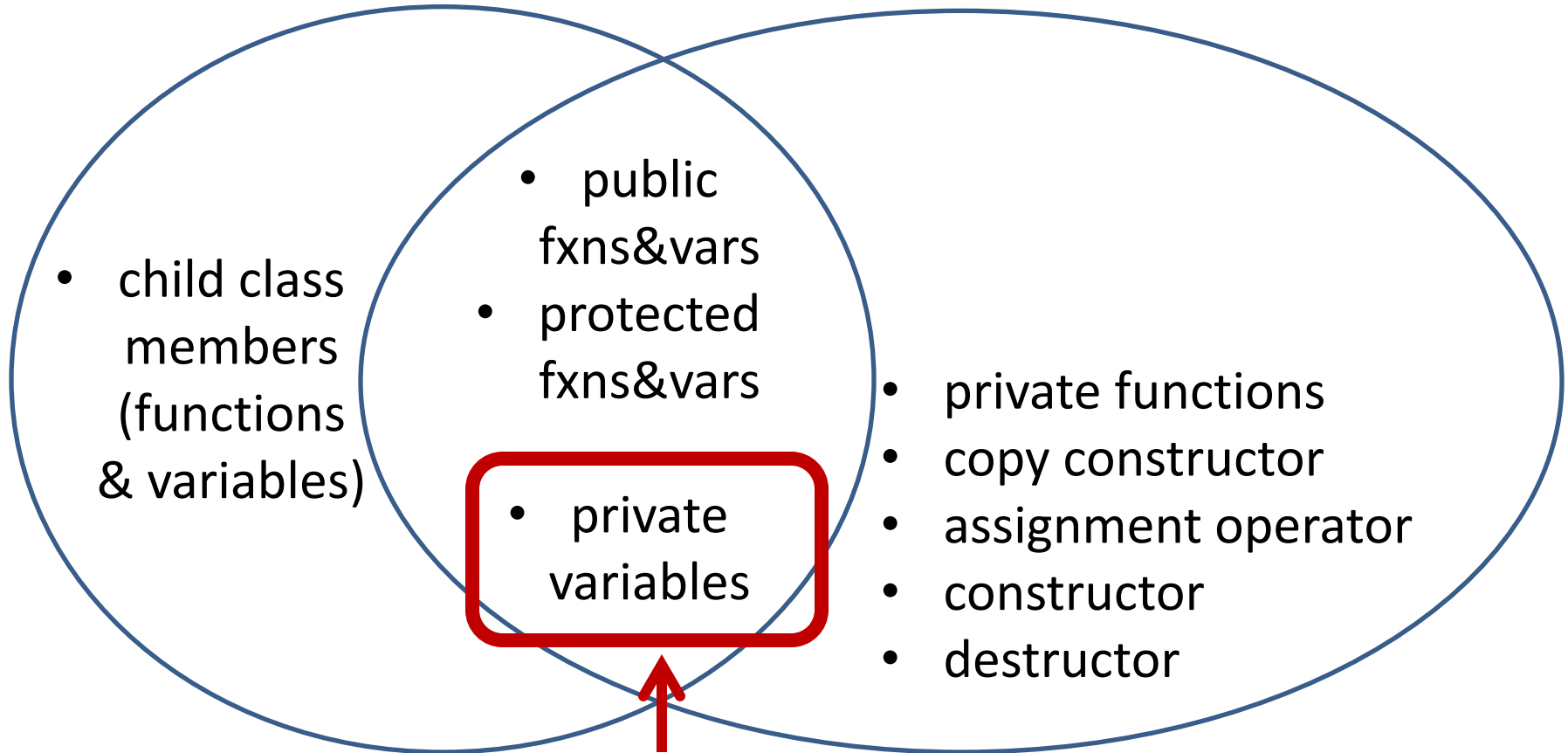
Car Class

Vehicle Class



Car Class

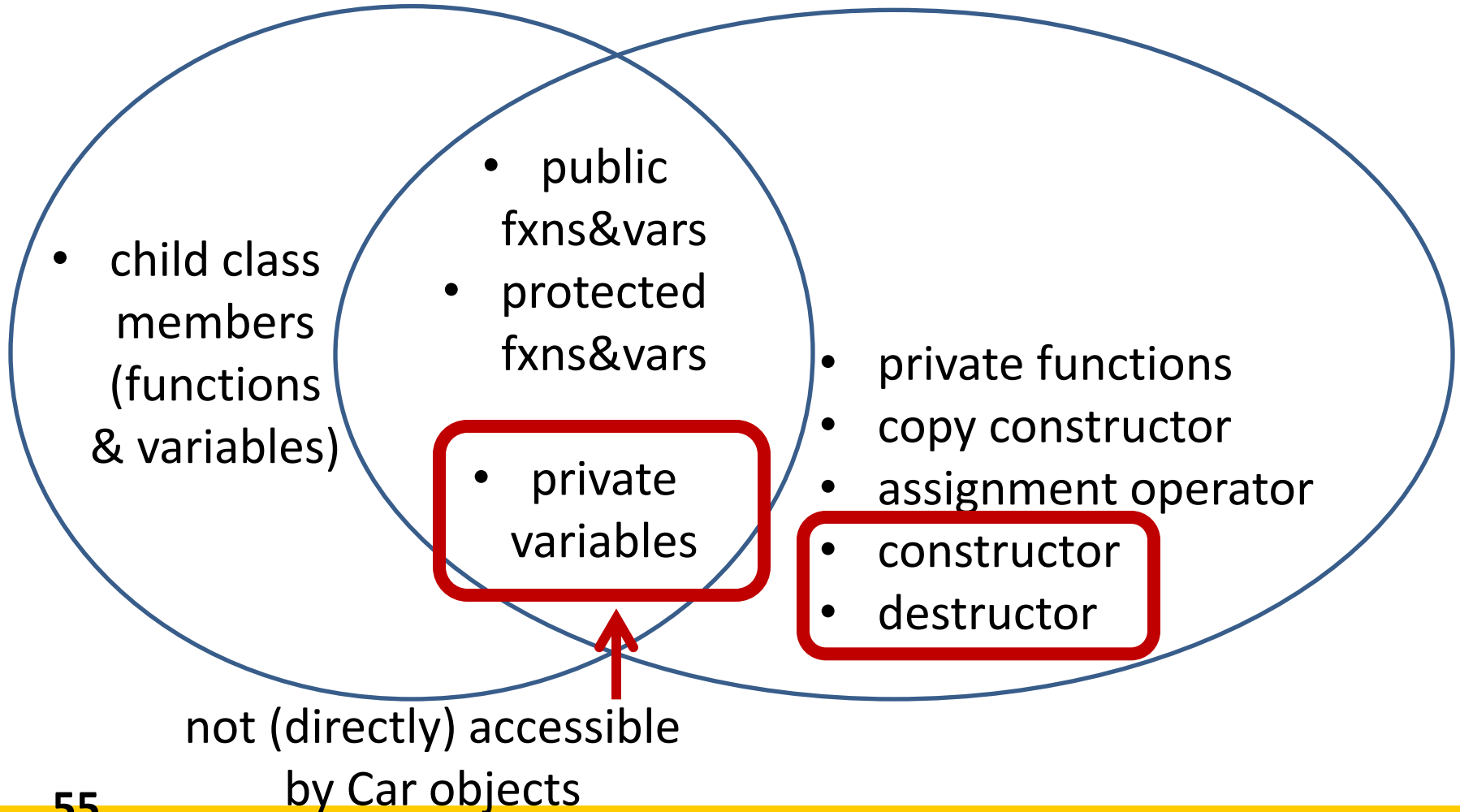
Vehicle Class



not (directly) accessible
by Car objects

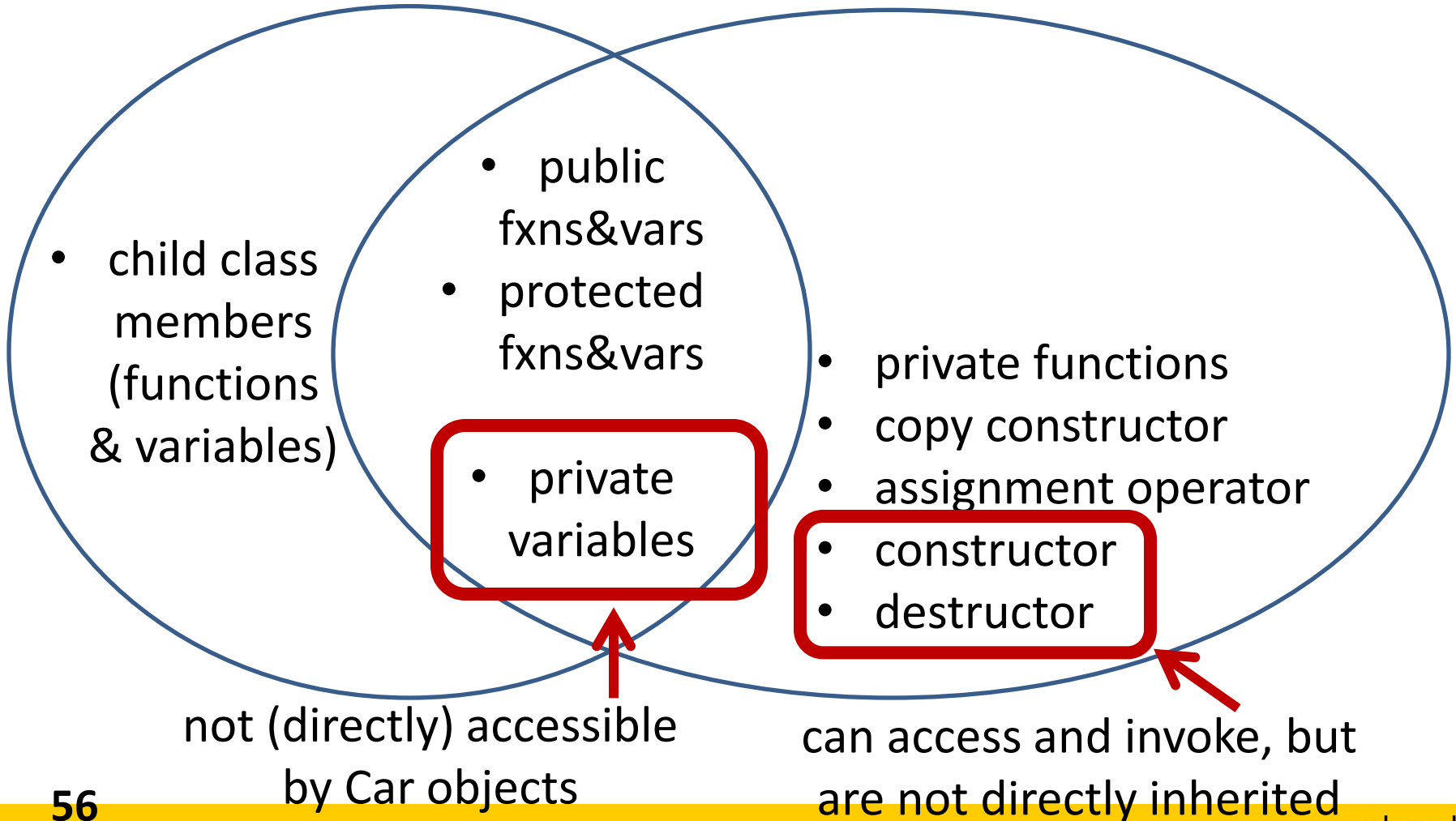
Car Class

Vehicle Class



Car Class

Vehicle Class



- Code Reuse
- Object Relationships
- Inheritance
 - What is Inherited
 - Handling Access
- Overriding
- Homework and Project

- Child class has access to parent class's:
 - **public** member variables/functions
 - **protected** member variables/functions
 - but *not* **private** member variables/functions
- How should we set the access modifier for parent member variables we want the child class to be able to access?

- Do not make these variables protected!
 - Leave them private!
- Instead, child class uses public or protected functions when interacting with parent variables
 - Reason we implement accessors and mutators

- Project 2 is out – you should have started!
 - It is due Thursday, March 10th
- Nothing over Spring Break
 - Enjoy your temporary freedom!