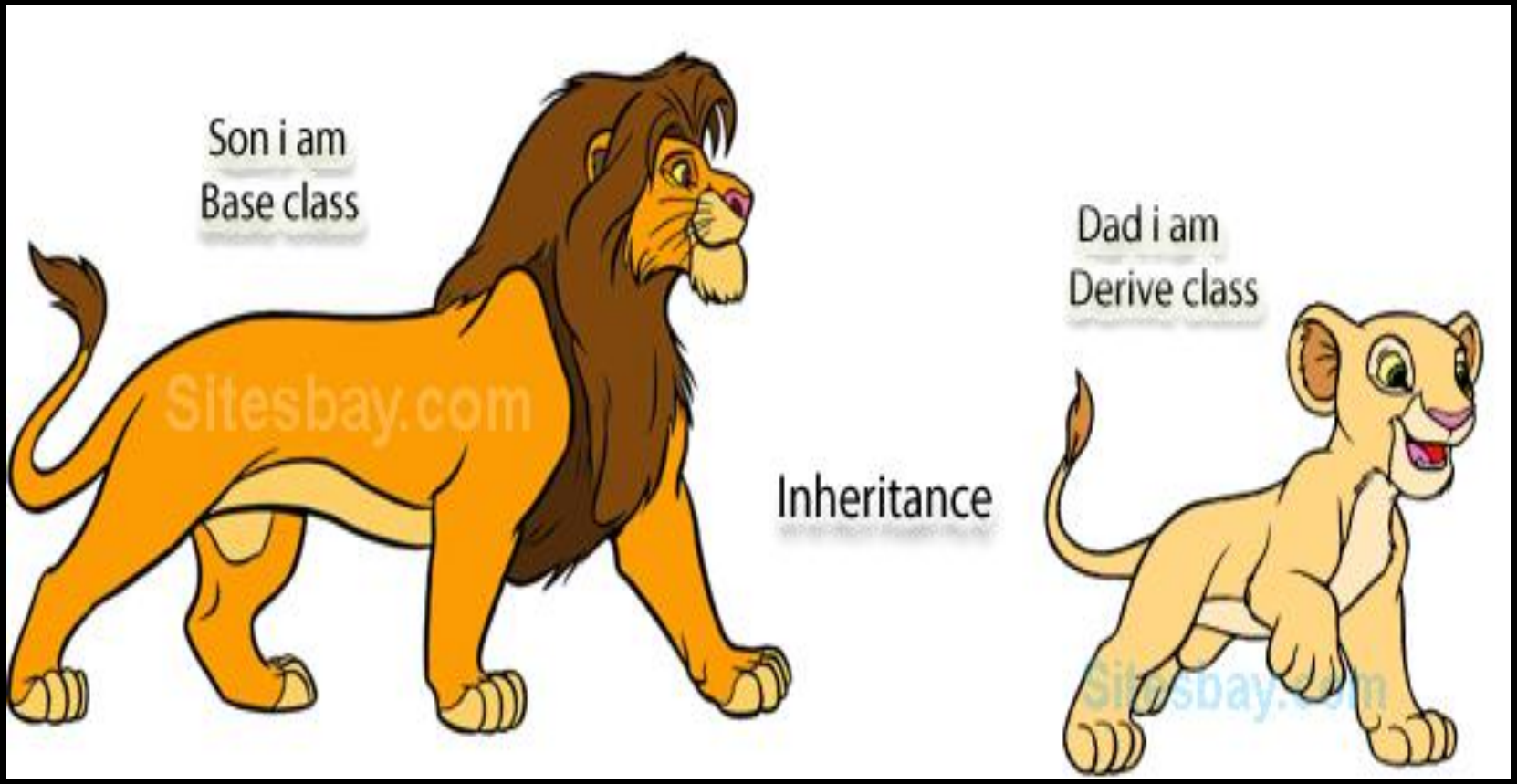
Week 07
Inheritance

March 5, 2018


Son i am
Base class
Inheritance
Dad i am
Derive class

What Is Inheritance?

- Provides a way to create a new class from an existing class
- The new class is a specialized version of the existing class

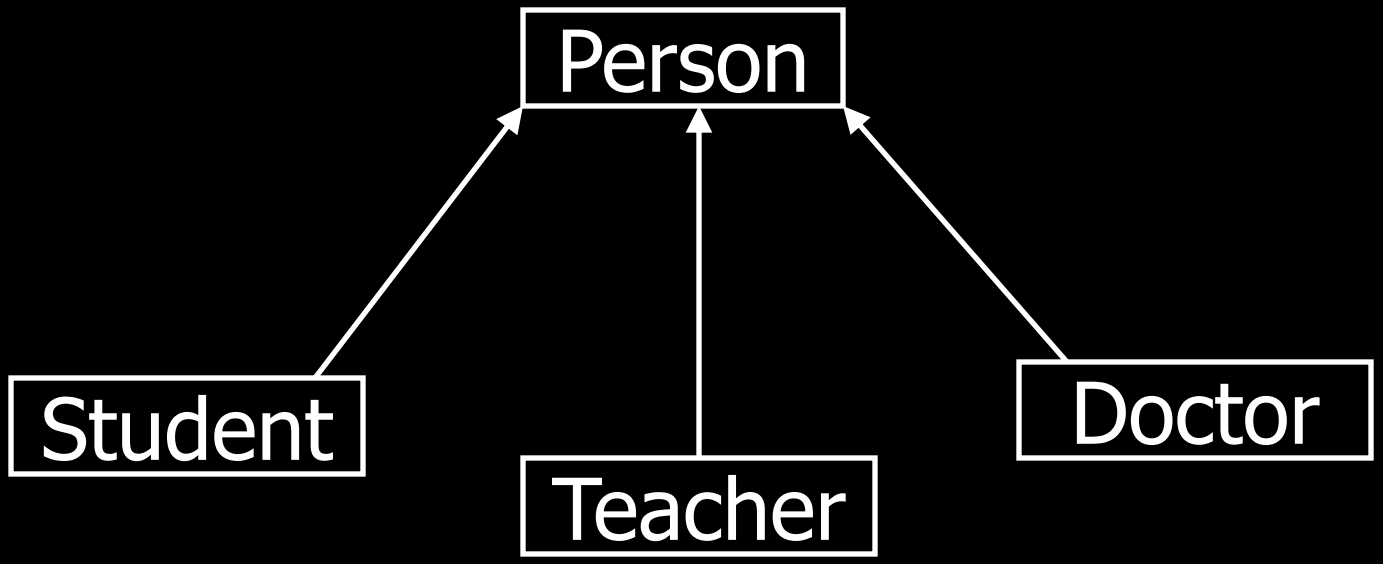
Inheritance

- A child inherits characteristics of its parents
- Besides inherited characteristics, a child may have its own unique characteristics

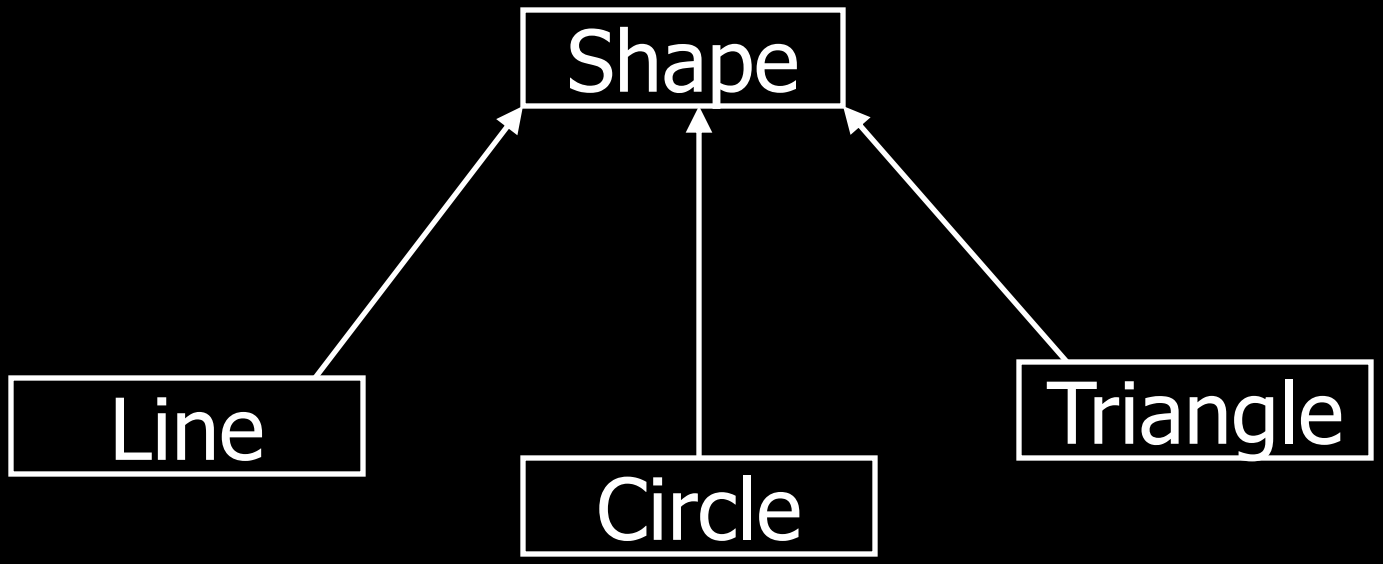
Inheritance in Classes

- If a class B inherits from class A then it contains all the characteristics (information structure and behaviour) of class A
- The parent class is called *base* class and the child class is called *derived* class
- Besides inherited characteristics, derived class may have its own unique characteristics

Example – Inheritance



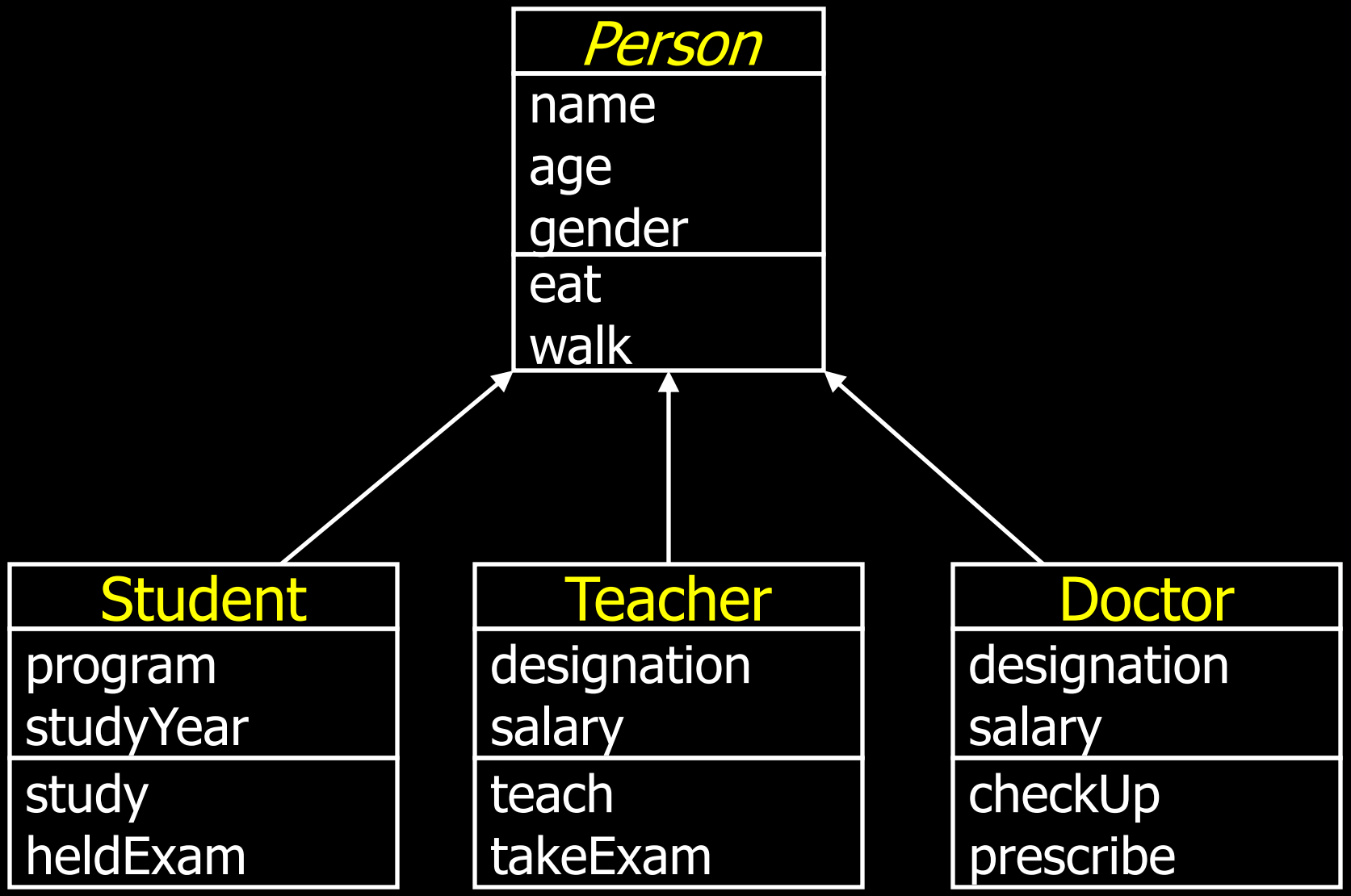
Example – Inheritance



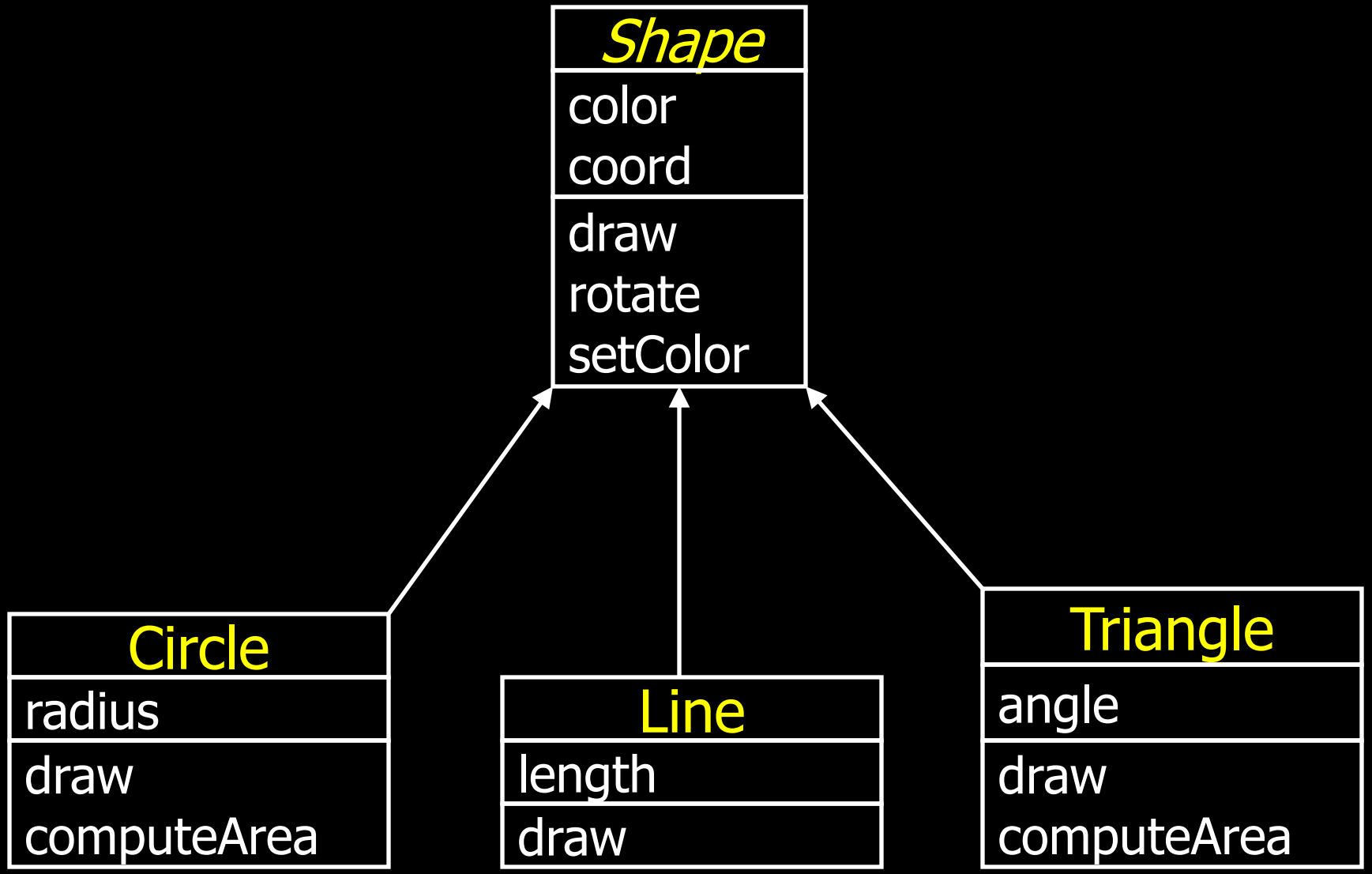
Inheritance – "IS A" or "IS A KIND OF" Relationship

- Each derived class is a special kind of its base class

Example – "IS A" Relationship



Example – "IS A" Relationship



Inheritance – Terminology and Notation in C++

- Base class (or parent) – inherited from
- Derived class (or child) – inherits from the base class
- Notation:

```
class Student                  // base class
{
  . . .
};
class UnderGrad : public student
{                              // derived class
  . . .
};
```

Back to the ‘is a’ Relationship

- An object of a derived class 'is a(n)' object of the base class
- Example:
  - an `UnderGrad` is a `Student`
  - a `Mammal` is an `Animal`
- A derived object has **all** of the characteristics of the base class

Inheritance – Advantages

- Reuse
- Less redundancy
- Increased maintainability

Reuse with Inheritance

- Main purpose of inheritance is reuse
- We can easily add new classes by inheriting from existing classes
  - Select an existing class closer to the desired functionality
  - Create a new class and inherit it from the selected class
  - Add to and/or modify the inherited functionality

Reuse with Inheritance

- C++ strongly supports the concept of Reusability.
- The C++ classes can be reused in several ways. Once a class has been written and tested, it can be adapted by another programmer to suit their requirements.
- This is basically done by creating new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from an old one is called inheritance

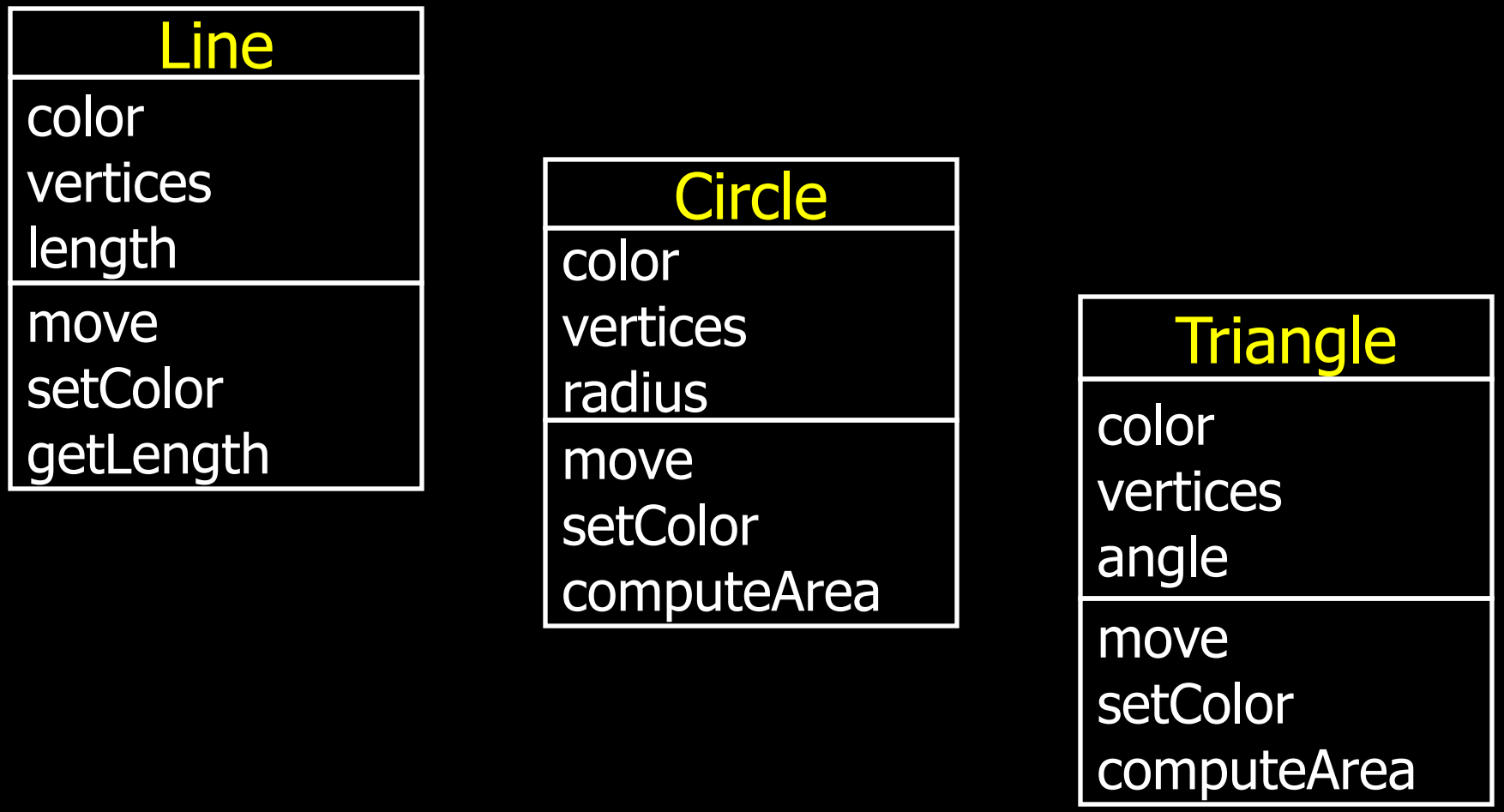
Concepts Related with Inheritance

- Generalization
- Specialization

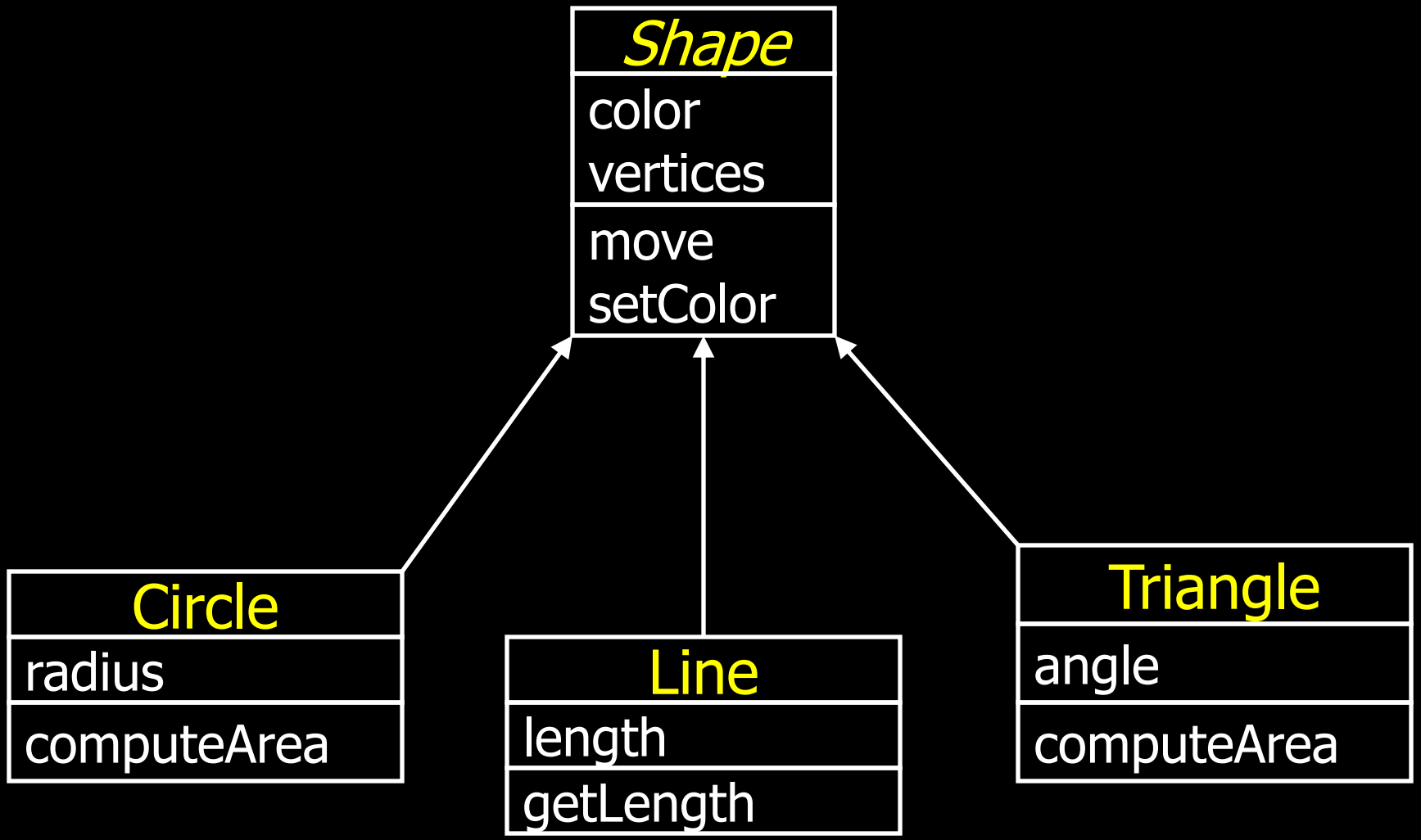
Generalization

- In OO models, some classes may have common characteristics
- We extract these features into a new class and inherit original classes from this new class
- This concept is known as Generalization

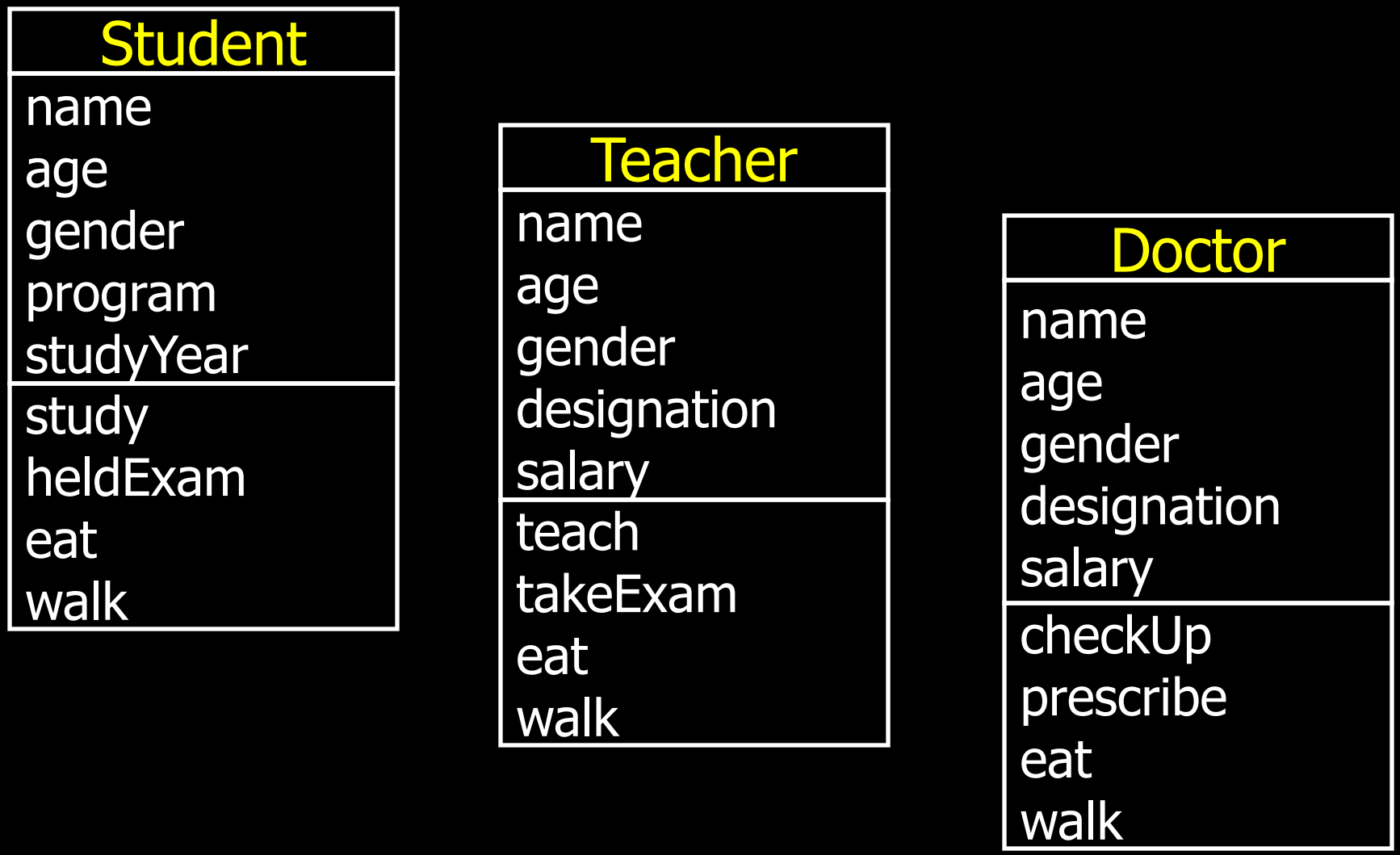
Example – Generalization



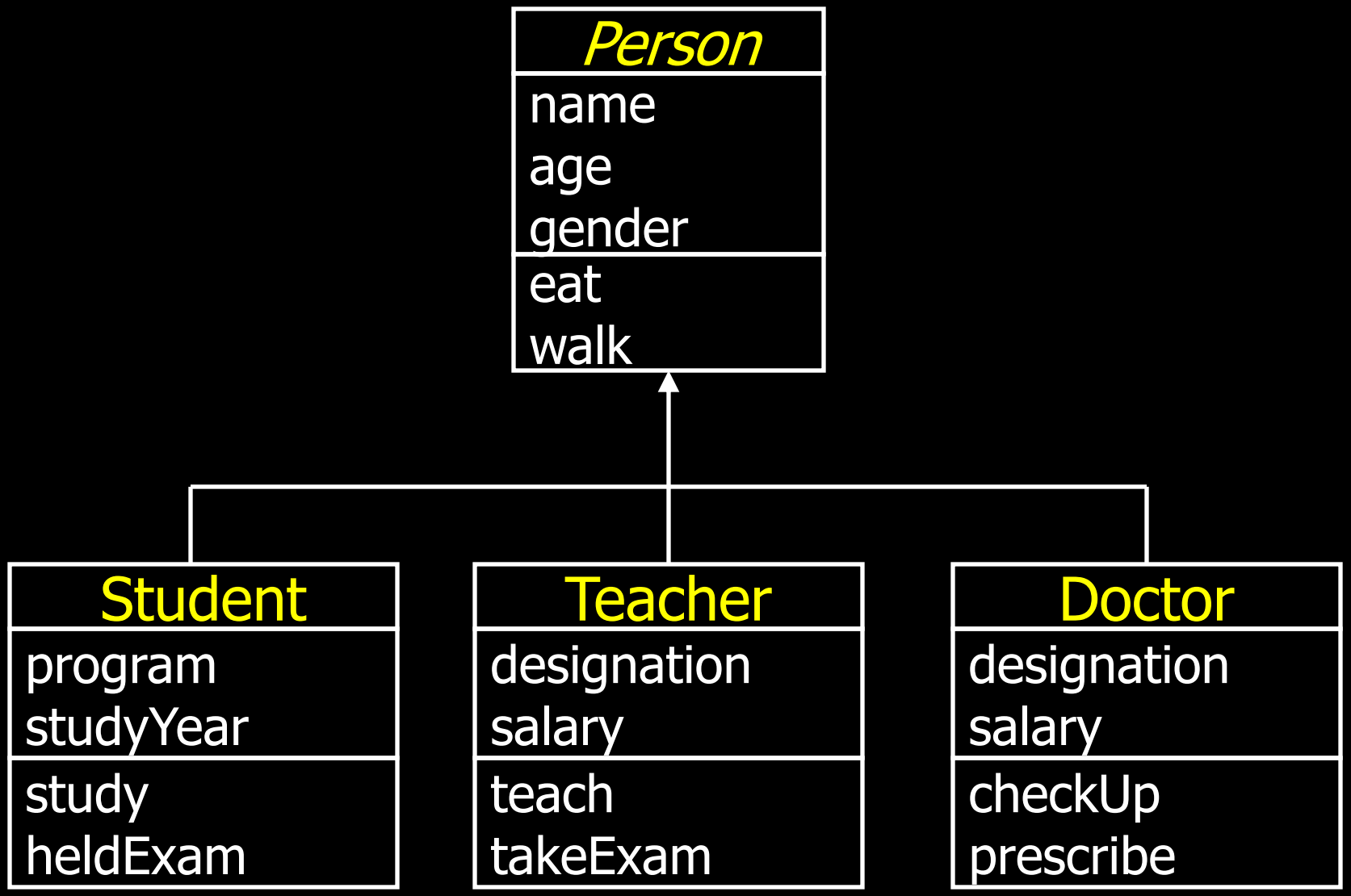
Example – Generalization



Example – Generalization



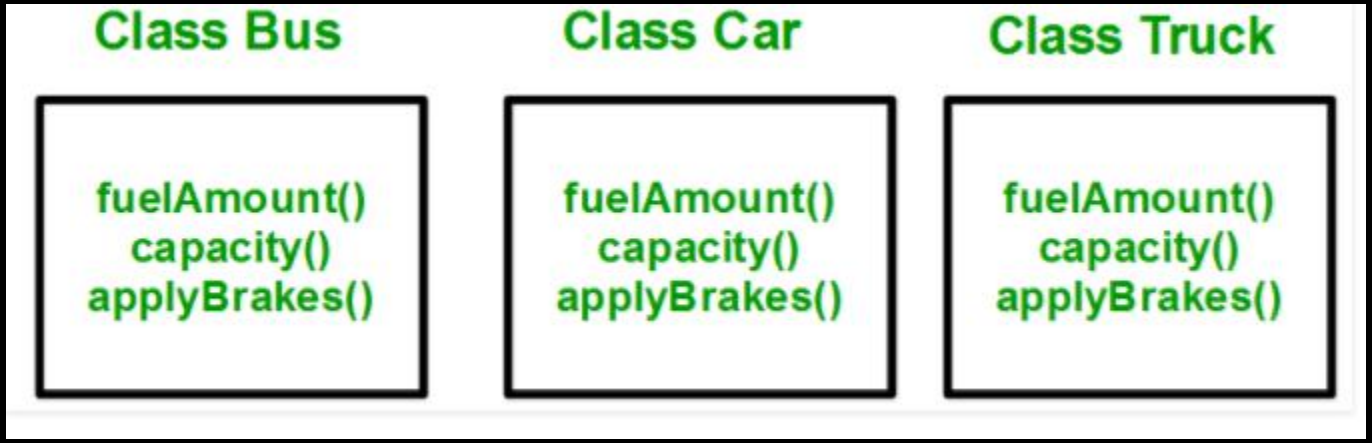
Example – Generalization



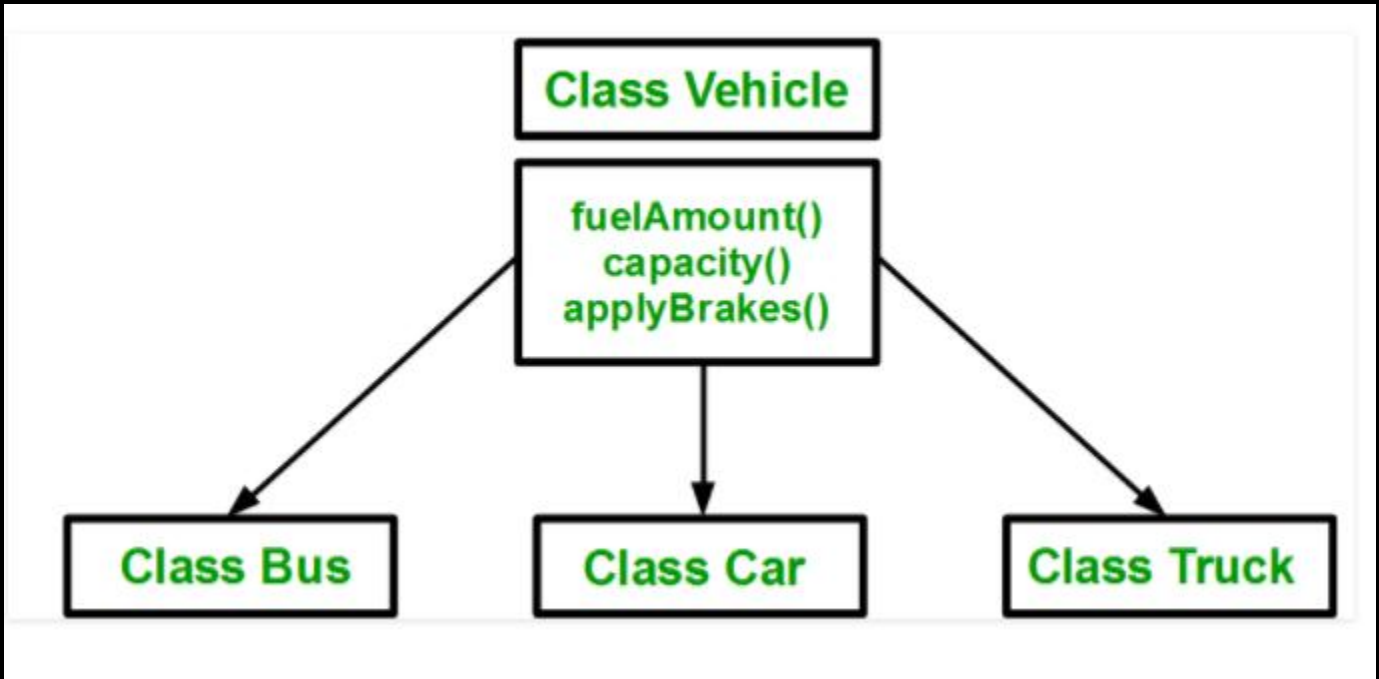
Specialization

- Adding new functionality to the existing class.
  - Extending a class TA (teaching assistant) from the Person (Student, Doctor, Teacher) hierarchy.
  - Extending a class Rectangle from Shape hierarchy.

Generalization/Specialization??



Generalization



Overriding

- A class may need to override the default behaviour provided by its base class

- Reasons for overriding
  - Provide behaviour specific to a derived class
  - Extend the default behaviour
  - Restrict the default behaviour
  - Improve performance

Review

Week 7
Lecture 17

Today's Outline

- Types of Inheritance
- Memory Allocation of Objects
- Base Class Initializer

Types of Inheritance in C++

- There are three types of inheritance in C++
  - Public
  - Private
  - Protected

Types of Inheritance Hierarchy

- Single Inheritance
- Hierarchal Inheritance
- Multilevel Inheritance
- Multiple inheritance
- Hybrid inheritance

“IS A” Relationship

- IS A relationship is modeled with the help of public inheritance

- Syntax

```
class ChildClass
        : public BaseClass{
...
};
```

Example

```
class Person{
  ...
};
class Student: public Person{
...
};
```

Accessing Members

- Public members of base class become public member of derived class
- Private members of base class are not accessible from outside of base class, even in the derived class (Information Hiding)

Example

```
class Person{
  char *name;
  int age;
  ...
public:
  const char* GetName() const;
  int GetAge() const;
  ...
};
```

Example

```
class Student: public Person{
  int semester;
  int rollNo;
  ...
public:
  int GetSemester() const;
  int GetRollNo() const;
  void Print() const;
  ...
};
```

Example

```
void Student::Print()
{
  cout << name << " is in" << "
  semester " << semester;
}
```

ERROR

Example

```
void Student::Print()
{
  cout << GetName()
       << " is in semester "
    << semester;
}
```

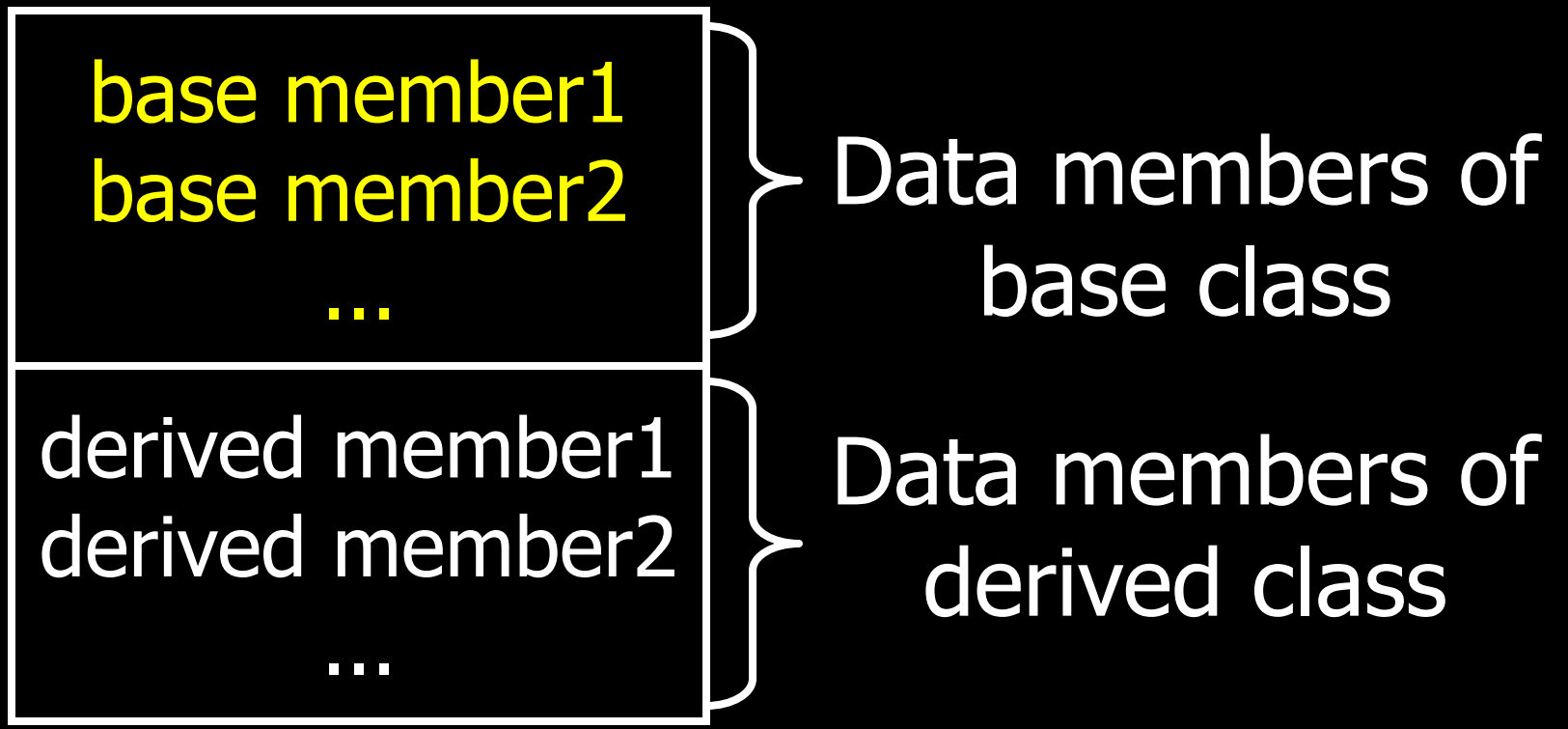
Example

```
int main(){
  Student stdt;

  stdt.semester = 0;//error
  stdt.name = NULL; //error
  cout << stdt.GetSemester();
  cout << stdt.GetName();
  return 0;
}
```

Allocation in Memory

- The object of derived class is represented in memory as follows



Allocation in Memory

- Every object of derived class has an anonymous object of base class

Example

```
class Parent{
public:
  Parent(int i){…};
};
class Child : public Parent{
public:
  Child(int i): Parent(i)
  {…}
};
```

Example

```
class Parent{
public:
   Parent(){cout <<
       "Parent Constructor...";}
   ...
};
class Child : public Parent{
public:
   Child():Parent()
   {cout << "Child Constructor...";}
   ...
};
```

Base Class Initializer

- User can provide base class initializer and member initializer simultaneously

Example

```
class Parent{
public:
  Parent(){…}
};
class Child : public Parent{
  int member;
public:
  Child():member(0), Parent()
  {…}
};
```

Base Class Initializer

- The base class initializer can be written after member initializer for derived class
- The base class constructor is executed before the initialization of data members of derived class.

Initializing Members

- Derived class can only initialize members of base class using overloaded constructors
  - Derived class can not initialize the public data member of base class using member initialization list

Example

```
class Person{
public:
  int age;
  char *name;
  ...
public:
  Person();
};
```

Example

```
class Student: public Person{
private:
  int semester;
...
public:
  Student(int a):age(a)
  {                          //error
  }
};
```

Reason

- It will be an assignment not an initialization

Week 07
Lecture 18

- Destructors in inheritance
- Constructors/Destructors Code Examples
- Function Overriding Code Examples
- Types of Inheritance

Destructors

- Destructors are called in reverse order of constructor called
- Derived class destructor is called before the base class destructor is called

Example

```
class Parent{
public:
   Parent(){cout <<"Parent Constructor";}
   ~Parent(){cout<<"Parent Destructor";}
};

class Child : public Parent{
public:
   Child(){cout << "Child Constructor";}
   ~Child(){cout << "Child Destructo";}
};
```

Example

```
Output:


Parent Constructor
Child Constructor
Child Destructor
Parent Destructor
```

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class

Types of Inheritance

Base-class member-access specifier	Type of inheritance		
	public inheritance	**protected** inheritance	**private** inheritance
public	**public** in derived class. Can be accessed directly by member functions, **friend** functions and nonmember functions.	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**private** in derived class. Can be accessed directly by member functions and **friend** functions.
protected	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**private** in derived class. Can be accessed directly by member functions and **friend** functions.
private	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.

What Does a Child Have?

An object of the derived class has:

- all members defined in child class
- all members declared in parent class

An object of the derived class can use:

- all `public` members defined in child class
- all `public` members defined in parent class

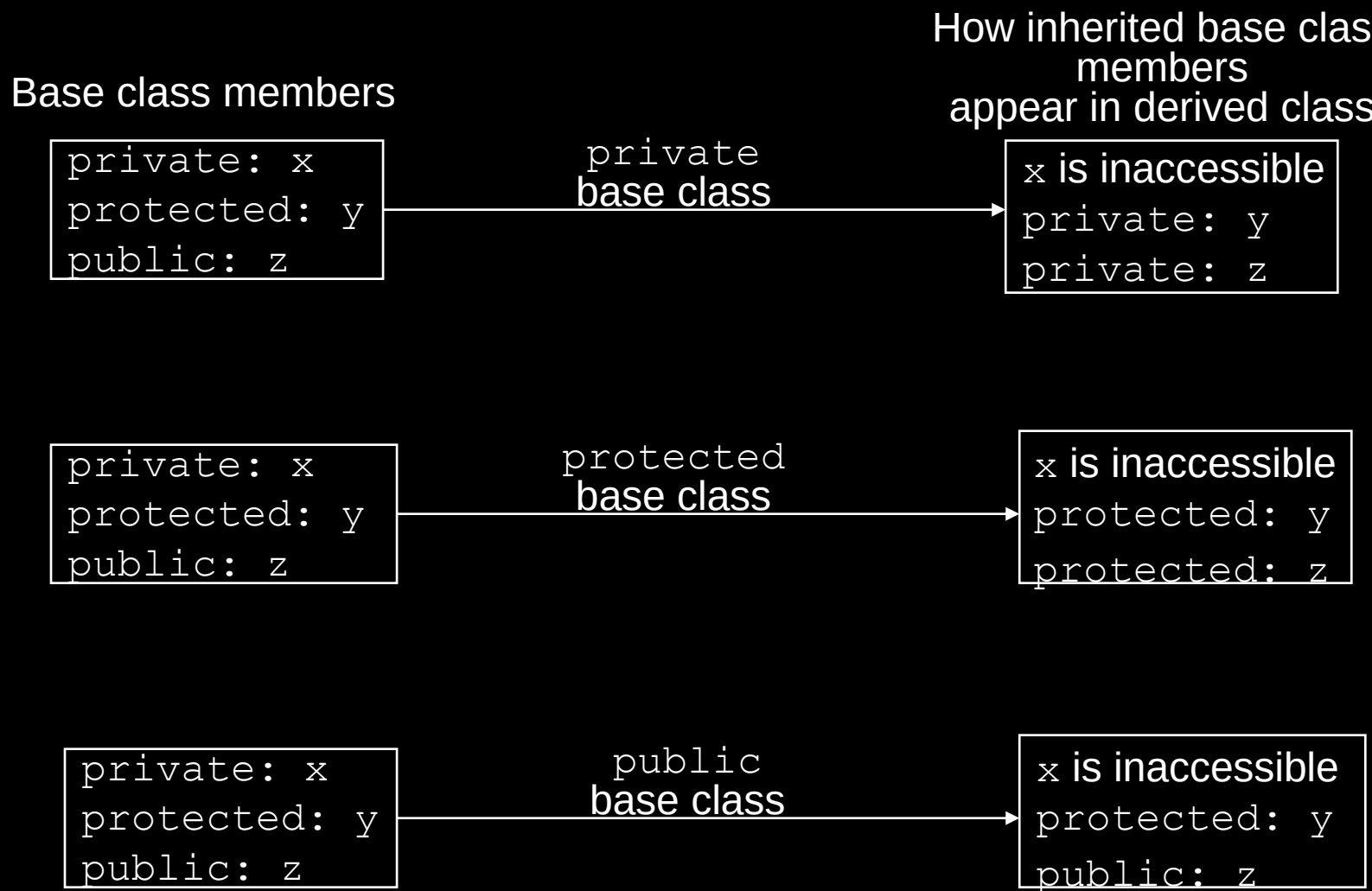
Protected Members and Class Access

- `protected` member access specification: like `private`, but accessible by objects of derived class
- Class access specification: determines how `private`, `protected`, and `public` members of base class are inherited by the derived class

Class Access Specifiers

1) **`public`** – object of derived class can be treated as object of base class (not vice-versa)
2) **`protected`** – more restrictive than `public`, but allows derived classes to know details of parents
3) **`private`** – prevents objects of derived class from being treated as objects of base class.

Inheritance vs. Access



Inheritance vs. Access

class Grade

private members:
```
  char letter;
  float score;
  void calcGrade();
```
public members:
```
  void setScore(float);
  float getScore();
  char getLetter();
```

class Test : public Grade

private members:
```
  int numQuestions;
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```

When `Test` class inherits from `Grade` class using `public` class access, it looks like this: →

private members:
```
  int numQuestions:
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
  void setScore(float);
  float getScore();
  char getLetter();
```

Inheritance vs. Access

class Grade

private members:
```
char letter;
float score;
void calcGrade();
```
public members:
```
void setScore(float);
float getScore();
char getLetter();
```

class Test : protected Grade

private members:
```
int numQuestions;
float pointsEach;
int numMissed;
```
public members:
```
Test(int, int);
```

When `Test` class inherits from `Grade` class using `protected` class access, it looks like this: →

private members:
```
int numQuestions:
float pointsEach;
int numMissed;
```
public members:
```
Test(int, int);
```
protected members:
```
void setScore(float);
float getScore();
float getLetter();
```

Inheritance vs. Access

class Grade

private members:
```
  char letter;
  float score;
  void calcGrade();
```
public members:
```
  void setScore(float);
  float getScore();
  char getLetter();
```

class Test : private Grade

private members:
```
  int numQuestions;
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```

When `Test` class inherits from `Grade` class using `private` class access, it looks like this: →

private members:
```
  int numQuestions:
  float pointsEach;
  int numMissed;
  void setScore(float);
  float getScore();
  float getLetter();
```
public members:
```
  Test(int, int);
```

Your Turn (home work)

Implement public inheritance for the classes mentioned below.

- Employee
  - DailyWagesEmployee
  - MonthlyPaidEmployee
- BankAccount
  - SavingAccount
  - CurrentAccount

Can we inherit Structs?

- **Yes, struct can inherit from class in C++.**
- In C++, *classes and struct are the same except for their default behaviour* with regards to inheritance and access levels of members.
- C++ class
  - Default Inheritance = **private**
  - Default Access Level for Member Variables and Functions = **private**
- C++ struct
  - Default Inheritance = **public**
  - Default Access Level for Member Variables and Functions = **public**

```
struct A{
protected:
int x, y;
};

struct B :public A{
private:
int w, z;
public:
void setattributes(int X, int Y, int Z, int W){
x = X;
y = Y;
z = Z;
w = W;
}
};
```

Review

Code Examples

Base Class Initializer Example

Base Class Initializer Example

```
class Mother {
public:
    Mother()
    {
        cout << "Mother: no parameters\n";
    }
    Mother(int a)
    {
        cout << "Mother: int parameter\n";
    }
};
int main() {
    Daughter kelly(0);
    Son bud(0);
    return 0;
}
```

```
class Daughter : public Mother {
public:
    Daughter(int a)
    {
        cout << "Daughter: int parameter\n\n";
    }
};
class Son : public Mother {
public:
    Son(int a) : Mother(a)
    {
        cout << "Son: int parameter\n\n";
    }
};
```

Function Overriding &
Multi level Inheritance Example

```
class parent{
public:
    parent(){
cout << "parent constructor "<<endl;
    }
    void print(){
cout << " printing print function from
    parent class"<<endl;
    }
~parent(){
cout<< "Parent Destructor "<<endl;
    }
};
```

```
class child:public parent{
public:
child(){
cout<<" Child Constructor " << endl;
}
~child (){
cout<<"Child Destructor " <<endl;
    }
    void print(){
parent::print();
cout << "printing overridden form of
    print function from child class"
    <<endl;
    }
};
```

```
class grandchild:public child{
    public:
        grandchild(){
                cout<< "Grand Child
    Constructor "<<endl;
        }
    ~grandchild(){
        cout<<"Grand Child
    Destructor "<<endl;
    }
    void print(){
        child::print();
        cout << "printing overridden
    form of print function from grand
    child class" <<endl;
    }
};
```

```
int main(){
    grandchild a ;
    a.print();
    return 0;
}
```

Review