INHERITANCE

CS A250 – C++ Programming II

INHERITANCE

- Important concept in **Object-Oriented Programming** (**OOP**)
  - **Inheritance**
    - A mechanism for *enhancing* existing classes
    - Define new classes that are **extensions** of existing classes
    - General form of class is defined
      - *Specialized* versions then inherit properties of general class
      - And add to it/modify its functionality for its appropriate use

INHERITANCE

- New class inherited from another class
- **Base class**
  - "General" class from which other classes derive
- **Derived class**
  - New class
  - Automatically **inherits** from **base class**:
    - Member variables
    - Member functions
 - Can then add additional member functions ("redefine") and variables

DERIVED CLASSES

- Consider example:
  Class of "Employees"

- Composed of:
  - Salaried employees
  - Hourly employees

- Each is "subset" of employees
  - Another subset might be those paid fixed rate each month or week

DERIVED CLASSES (CONT.)

- Don't "need" type of generic "employee"
  - Since no one's just an "employee"

- General concept of employee is helpful!
  - All have names
  - All have social security numbers
  - Associated functions for these "basics" are same among all employees

- So "general" class can contain all these "things" about employees

5

DERIVED CLASSES (CONT.)
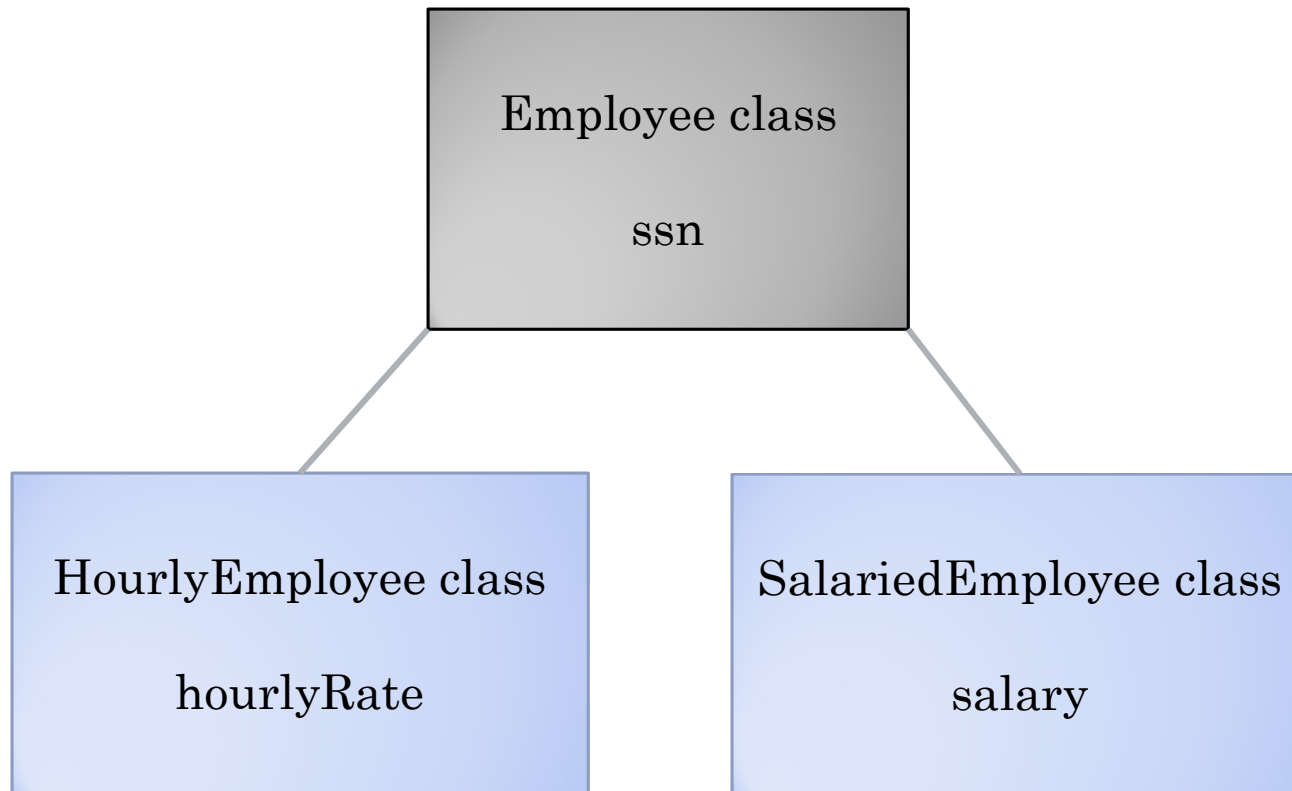
EMPLOYEE CLASS

- Many members of the **Employee** class are used by all types of employees

  - **Accessor** functions

  - **Mutator** functions

  - Other public functions

- We will **<u>not</u>**, however, have "objects" of the class **Employee**.

EXAMPLE 1

- **Project:** Employee Class
  - Employee.h
  - Employee.cpp

DERIVING FROM A CLASS

- The **derived** class automatically "**inherits**" from **base** class:
  - Member **variables**
  - Member **functions**

- The **derived** class can add
  - *New* member **variables**
  - *New* member **functions**

9

WHAT IS NOT INHERITED?

- Functions that are **NOT** inherited:
  - **Constructors**
  - **Private** member functions
  - **Destructors**
  - **Copy constructor**
  - **Overloaded assignment operator =**

  Note: You will study these two later this semester.
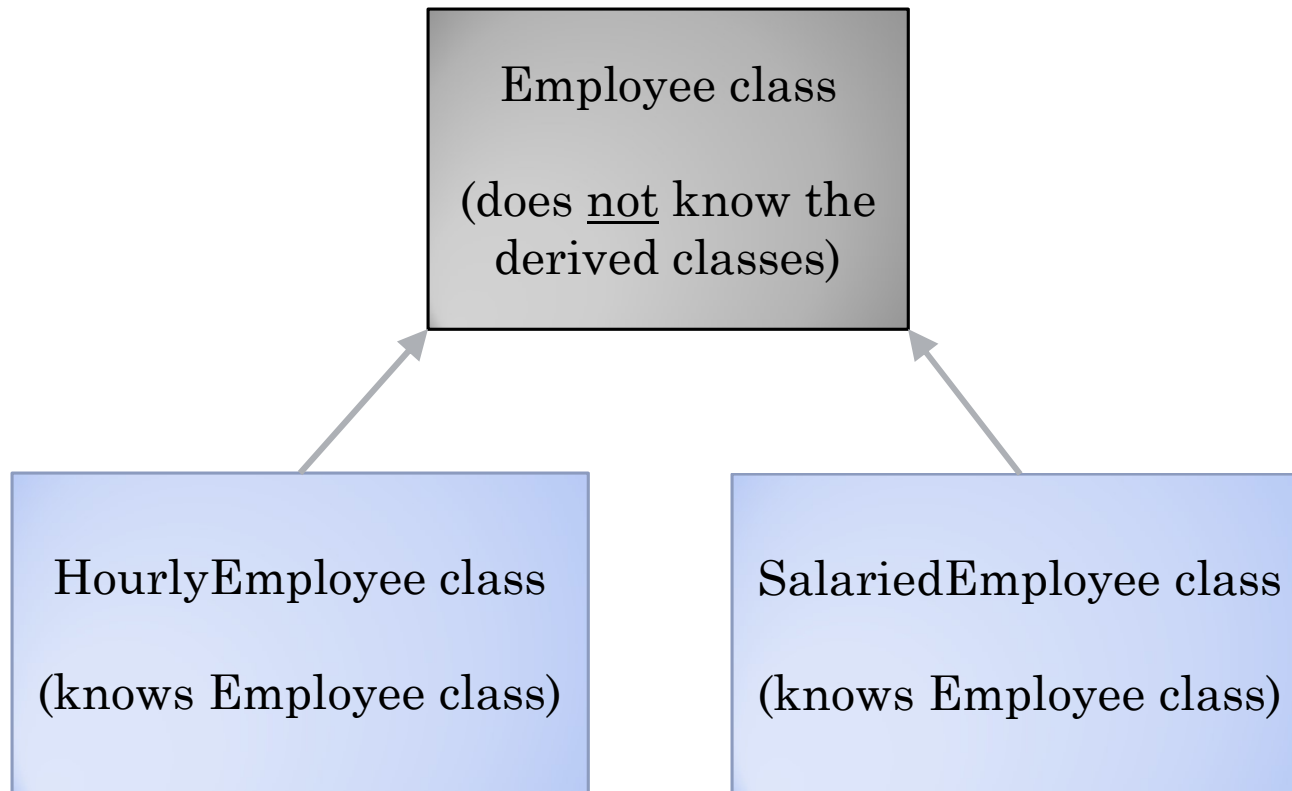
- *Why are they **not** inherited?*
  - Because they all need new information that **only** the **child class** has
    - For example, if a **child class object** uses the **parent constructor**, the **child member variables** will not be initialized, because the **parent constructor** does not recognize those variables.

10

TERMINOLOGY

- **Base class** also called
  - **Parent** class
  - **Ancestor** class
- **Derived class** also called
  - **Child** class
  - **Descendant** class

NOTE: This presentation will use the **parent-child** connotation from this point on.

11

DERIVED CLASSES (CONT.)

Employee class

(does <u>not</u> know the derived classes)

HourlyEmployee class

(knows Employee class)

SalariedEmployee class

(knows Employee class)

HOURLYEMPLOYEE CLASS

- In the **derived** class definition, we declare that the class is derived:

```
class HourlyEmployee : public Employee
```

- The **:** symbol denotes **inheritance**

- The keyword **public** is required to be able to invoke an **Employee** member function on an **HourlyEmployee** object elsewhere
  - If you forget, the compiler will think it is *private*, which will violate the reason for using inheritance

13

HOURLYEMPLOYEE CLASS (CONT.)

- We do **<u>not</u>** have to re-declare the variable `ssn` since we are inheriting it from the parent class

- But we have a new variable

    ```
    double hourlyRate;
    ```

14

```
class Employee
{
public:
    Employee( );
    Employee( const std::string& newSSN );
    std::string getSSN( ) const;
    void setSSN( const std::string& newSSN );

private:
    std::string ssn;
};
```

Parent class definition

```
#include "Employee.h"

class HourlyEmployee : public Employee
{
public:
    HourlyEmployee( );
    HourlyEmployee( const std::string& newSSN,
                    double newRate );
    void setRate( double newRate );
    double getRate( ) const;

private:
    double hourlyRate;
};
```

Child class definition

15

```
class Employee
{
public:
    Employee( );
    Employee( const std::string&
    std::string getSSN( ) const;
    void setSSN( const std::string& newSSN )

private:
    std::string ssn;

};
```

Need to include the parent header file

Need to specify inheritance to class

Need to send new value to parent member variables

```
#include "Employee.h"

class HourlyEmployee : public Employee
{
public:
    HourlyEmployee( );
    HourlyEmployee( const std::string& newSSN,
                    double newRate );
    void setRate( double newRate );
    double getRate( ) const;

private:
    double hourlyRate;
};
```

Child class definition

16

HOURLYEMPLOYEE CLASS (CONT.)

- How do you set the the **ssn** for an hourly employee?
  - We do ***not*** inherit the base constructor, BUT
  - We can ***call*** the base constructor

```
HourlyEmployee::HourlyEmployee (all param types...)
                      : Employee (parent param value)
```

- **Note:** If you *omit* the call to the parent overloaded constructor, then the parent object will be constructed with the default constructor of the parent class.

```
#include "HourlyEmployee.h"

HourlyEmployee::HourlyEmployee()
{
        hourlyRate = 0.0;
}


HourlyEmployee::HourlyEmployee
        ( const string& newSSN, double newRate )

                                        : Employee(newSSN)
{
        hourlyRate = newRate;
}


void HourlyEmployee::setRate(double newRate)
{
        hourlyRate = newRate;
}


double HourlyEmployee::getRate( ) const
{
        return hourlyRate;
}
```

Initialize ONLY own member variables

Call to the parent constructor to send new value to parent member variable

18

HOURLYEMPLOYEE CLASS INTERFACE

- **Note:** Class definition begins **same** as any other:

  - `#ifndef` structure

  - Includes required libraries

  - Also `#include` "`Employee.h`"

HOURLYEMPLOYEE CLASS ADDITIONS

- **Derived** class interface only lists new members
  - Since all others inherited are already defined
  - i.e.: "all" employees have **ssn**

- **HourlyEmployee** class adds:
  - **Constructors**
  - **hourlyRate** variable
  - **setRate()** and **getRate()** member functions

EXAMPLE 2

- **Project:** Employee Class
  - HourlyEmployee.h
  - HourlyEmployee.cpp

THE **protected** QUALIFIER

- **Child class** "inherits" **parent private member variables**

  - **BUT cannot** access them directly
    - Need to use an **accessor function**

  - Use **protected** if you want **parent members** to be accessed by all **child classes**, but *not* by other classes

- **Note:** Many feel this "violates" information hiding

22

REDEFINING FUNCTIONS

- If a **child class** requires a different implementation for an **inherited parent member function**, the function may be "**redefined**" in the **child class** by

  - Listing a **declaration** in the **definition** of the **child class**

    - The declaration will be the same as in the **parent** class

  - **Redefining** → Must have:

    - *same* number and

    - *same* type of parameters

    - (*different* from **overloading** *a function*)

```
class Employee
{
public:
    ...
    void print( ) const;
    ...
private:
    std::string ssn;
};
```

Parent class definition

```
#include "Employee.h"

class HourlyEmployee : public Employee
{
public:
    ...
    void print( ) const;
    ...
private:
    double hourlyRate;
};
```

Child class definition

24

REDEFINING FUNCTIONS (CONT.)

- How can the **derived** print function **print** the member variable (ssn) of the **base** class?

    - **Solution 1:**

        - Call the **parent's print function**
        - Specify that it is the <u>parent's</u> print function and not its own function.

    - **Solution 2:**

        - Call the **parent's accessor function**

```
...
void Employee::print( ) const
{
    cout << "SSN: " << ssn << endl;
}
...
```

Parent class implementation

Child class implementation

Syntax 1

```
...
void HourlyEmployee::print( ) const
{
    Employee::print();
    cout << "Hourly rate: " << hourlyRate << endl;
}
...
```

Two ways to print the parent member variable

```
...
void HourlyEmployee::print( ) const
{
    cout << "SSN: " << getSSN() << endl;
    cout << "Hourly rate: " << hourlyRate << endl;
}
```

Syntax 2

```
...
void Employee::print( ) const
{
    cout << "SSN: " << ssn << endl;
}
...
```

Parent class implementation

Child class definition

Syntax 1

```
...
void HourlyEmployee::print( ) const
{
    Employee::print();
    cout << "Hourly rate: " << hourlyRate << endl;
}
...
```

Call the parent **print ( )** function by using the class name and scope resolution **Employee :: print( )**

```
...
void HourlyEmployee::print( ) const
{
    cout << "SSN: " << getSSN() << endl;
    cout << "Hourly rate: " << hourlyRate << endl;
}
```

Syntax 2

27

```
...
void Employee::print( ) const
{
    cout << "SSN: " << ssn << endl;
}
...
```

Parent class implementation

Child class definition

Syntax 1

```
...
void HourlyEmployee::print( ) const
{
    Employee::print();
    cout << "Hourly rate: " << hourlyRate << endl;
}
...
```

Call the parent accessor function **getSSN ()**
Recall: Cannot directly access the parent private member variables.

```
...
void HourlyEmployee::print( ) const
{
    cout << "SSN: " << getSSN() << endl;
    cout << "Hourly rate: " << hourlyRate << endl;
}
```
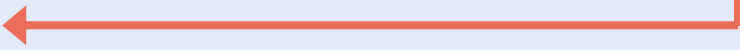
Syntax 2

28

REDEFINING: COMMON ERROR

- If you forget the **parent class qualifier** and the **scope resolution** (**::**), the function will call itself (that would be **recursion**)

Correct implementation:

Employee :: print( );

```
...
void HourlyEmployee::print( ) const
{
    print();
    cout << "Hourly rate: " << hourlyRate<< endl;
}
...
```
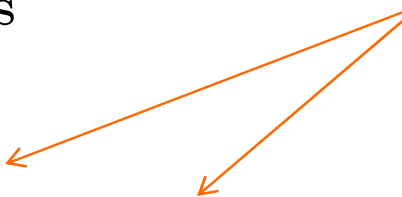
29

EXAMPLE 3

- **Project:** Employee Class
  - SalariedEmployee.h
  - SalariedEmployee.cpp

TO SUM UP...

- Functions that are **NOT** inherited:
  - Constructors
  - Private member functions
  - Destructors
  - Assignment operator =
  - Copy constructor → will be automatically generated if not defined, but does not work correctly everywhere, so it is better to define it

> **Note:** We will cover these two later in the semester.

- **Why** are not these inherited?
  - Because they all need new information that only the child class has
  - For example, new member variables to create the new object

31

COMMON ERRORS

- **Private** inheritance
  - Forget the keyword **public** that must follow the colon after the **child class name**

```
class HourlyEmployee : public Employee
```

- Attempting to access **private** parent member functions and/or variables
  - A **child class** inherits all fields from the **parent class**. If, however, the fields are *private*, the **child class** functions **cannot** access them
    - Need to use the *get* functions

MULTIPLE INHERITANCE

- **Derived** class can have **more than one base** class
  - Syntax just includes all base classes separated by commas:
    class derivedMulti : public base1, base2
    {...}
- Possibilities for ambiguity are endless!
- Dangerous undertaking!
  - Some believe should never be used

INHERITANCE (END)