Table of Contents

- Inheritance - derived class
- Features of inherited classes
- Access - specifier control
- Access - from derived class view
- Call access - calling methods
- Multiple inheritance

Base class

```cpp
class Employee {
    public:
        Employee();  // default constructor
        Employee(char* theName); // other methods and public data
    private:
        char name[30]; // other general employee information
    }; // end of class Employee
```

Inheritance - derived class

Syntax:
```cpp
class tag-name : access-specifier base-class {
    member list }; // default access specifier is private
```

Example:
```cpp
class Salaried : public Employee {
    private:
        float monthlySalaray;
        int hoursAvail;
    public:
        Salaried(); // default constructor
        Salaried(char* name, float monSal, int vTme); // note: name
        // rest of member list
};
```
Features of inherited classes

- Base class must be defined before derived class
- Derived class creates new base class for each instantiation
- It must initialize its base class
- Initialization - alternative ways (later slide)
- Must supply base initialization information
- Order - base class, data members of derived class, constructor code; reverse for destruction.
- Instantiation (create or declaration) of a variable identical to single class.

```cpp
// add jth employee to an array
Salaried emp[j] = {Salaried("John Jones", 650.00, 120)};
// Note: base constructor information supplied
```

Reasons for Using Inheritance

- Supports top-down programming - base class before derived.
- Can save considerable design and programming by using inherited features.
- Logical interface

Constructor(s) - initialization

- Derived constructor receives base class arguments
- Initialization features - in code or alt. initializer constructor

```cpp
Example:
class Salaried : public Employee {
private:
float monthlySalary;
int vacH;
public:
Salaried(); // default constructor
Salaried(char *name, float, int vacHrs = 120); // other constructor
// other members;
Salaried::Salaried(char *name; float sal) : Employee(name),
vacH(vacHrs) { monthlySalary = sal; }; // illustrates both init. methods

Note: Various way for initializing & the effect of initialization sequence
```
Access - specifier control

- Public - normal member base class access
- Private - restricts access, i.e., all base members of derived class are private
- Derived class can restrict base access but cannot expand it, e.g., can't make a private base members public

When creating base class directly, inheritance access-specifier does not affect since base class knows nothing about derived class.

Access - from derived class view with base public

- Members of base class
  - Public - can be seen anywhere
  - Private - cannot be seen in derived class scope
  - Protected - can be seen in derived class scope
- Members of derived class - standard access
- Base class - doesn't know derived class exists
- Friend of base class - derived unknown
- Friend of derived class - normal derived class access
- Not friend of base class

Call access - calling methods

```cpp
class Parent {
public:    void Print() { cout << age; };
private:   int age;   }; // end class
class Child : public Parent{
public:    void Print() { // overloaded derived class method
                Parent::Print(); // calls parent member; call not automatic
                cout << school; };
private:   char school[30]; }; // end class
Child myChild;    Parent dad; // create variables
myChild.Print();  // calls child member
myChild.Parent::Print(); // direct call to parent member
dad.Print();    // calls different parent
```
Multiple Inheritance

Relationship from multiple bases
⇒ Same base rules as previous apply to each

Mother
Father
Child

Note: The difference between mother and father should not be trivial. Otherwise multiple inheritance shouldn't be used.

Declaration:

```cpp
class Child : public Mother, public Father {
    member-list;
};
```

Virtual base class

Eliminate multiple base copies - virtual

God
Mother
Father
Child

Note: When creating Cain, a separate god would have been created for Eve and for Adam. To prevent this the base calls are declared virtual.

```cpp
class Mother : public virtual God {
    member-list;
};
class Father : public virtual God {
    member-list;
};
```

Note: Not applicable to general parenting!!