C++

INHERITANCE

Containment

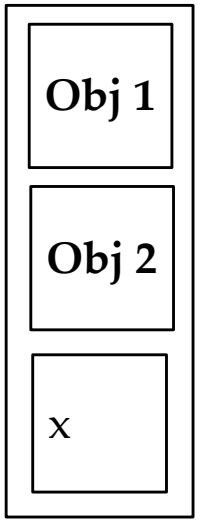
- Containment represents “has a” relationship
- Containment Relationship means the use of an object of a class as a member of another class.

  Ex. Birth_Date or joining date as a part of Employee class
- The container relationship brings reusability of code.

  Ex. Already written Date class can be used in Class Employee.

“HAS-A” Type of Relationship

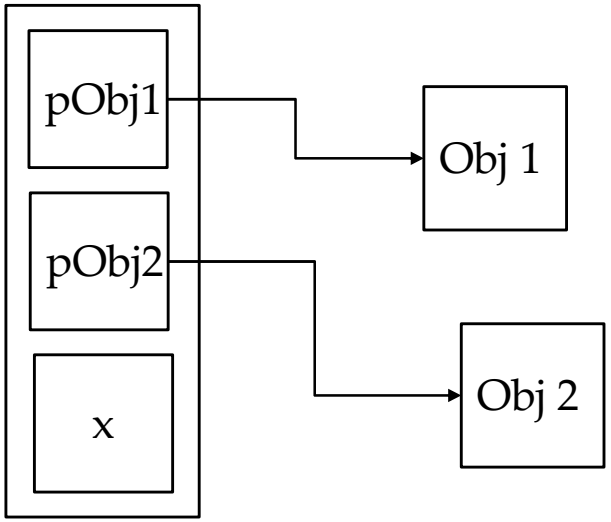
Physical containment



- Also called as tight coupling
- Example

  Car-Engine

Logical containment



- Also called as lose coupling
- Example

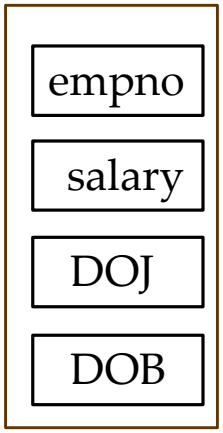
  Car-Documents

Has-a relationship

- Facilitates code reuse
- Already written classes can be used as members of another class
- Example:

  Already written class Date can be used as date members of class Person, Employee or Student as DOB, joining_date etc.

```
class employee{
    int empno;
    float salary;
    Date DOJ, DOB;
};
```



Constructor and Destructor

- Contained objects created first
- Order of creation decided by declarations in container class
- Default constructor gets called
- Object destruction takes place in reverse order

Member initialization list

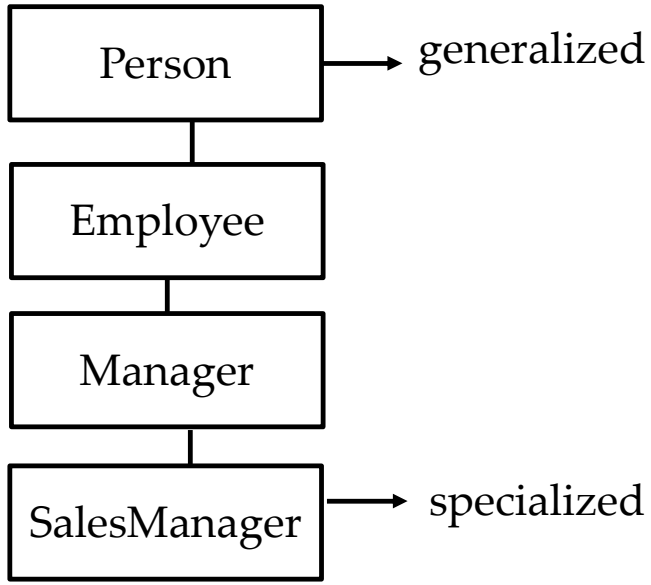
- Container object gets all the data from user and distributes it to appropriate contained object and to itself as well
- If not specified, all contained objects get created using default constructor. User input is reassigned later using member functions
- Call appropriate constructor right at the time of creation. This improves performance
- Even built-in data members can be initialized in this list
- Reference members can be initialized only in this list

Inheritance

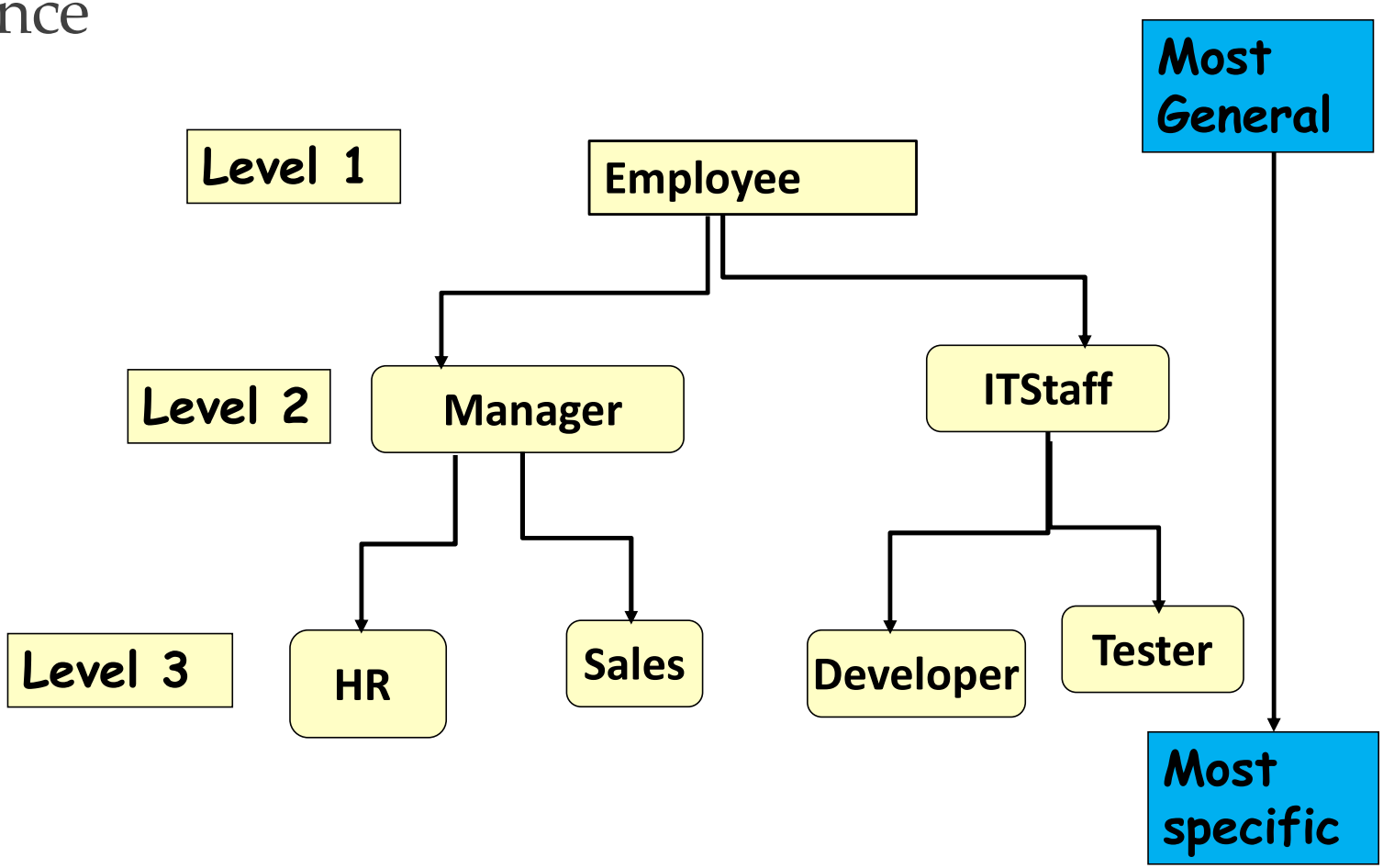
- Inheritance is one of the most powerful feature of object-oriented programming.
- Inheritance is the process of creating new classes, called derived classes, from existing or base classes.
- The derived class inherits all the functionalities of the base class and can add functionalities of its own.
- The base class is unchanged in this process.
- An important result of inheritance is reusability and the ease of distributing class libraries. A programmer can use a class created by another person or company.

Inheritance

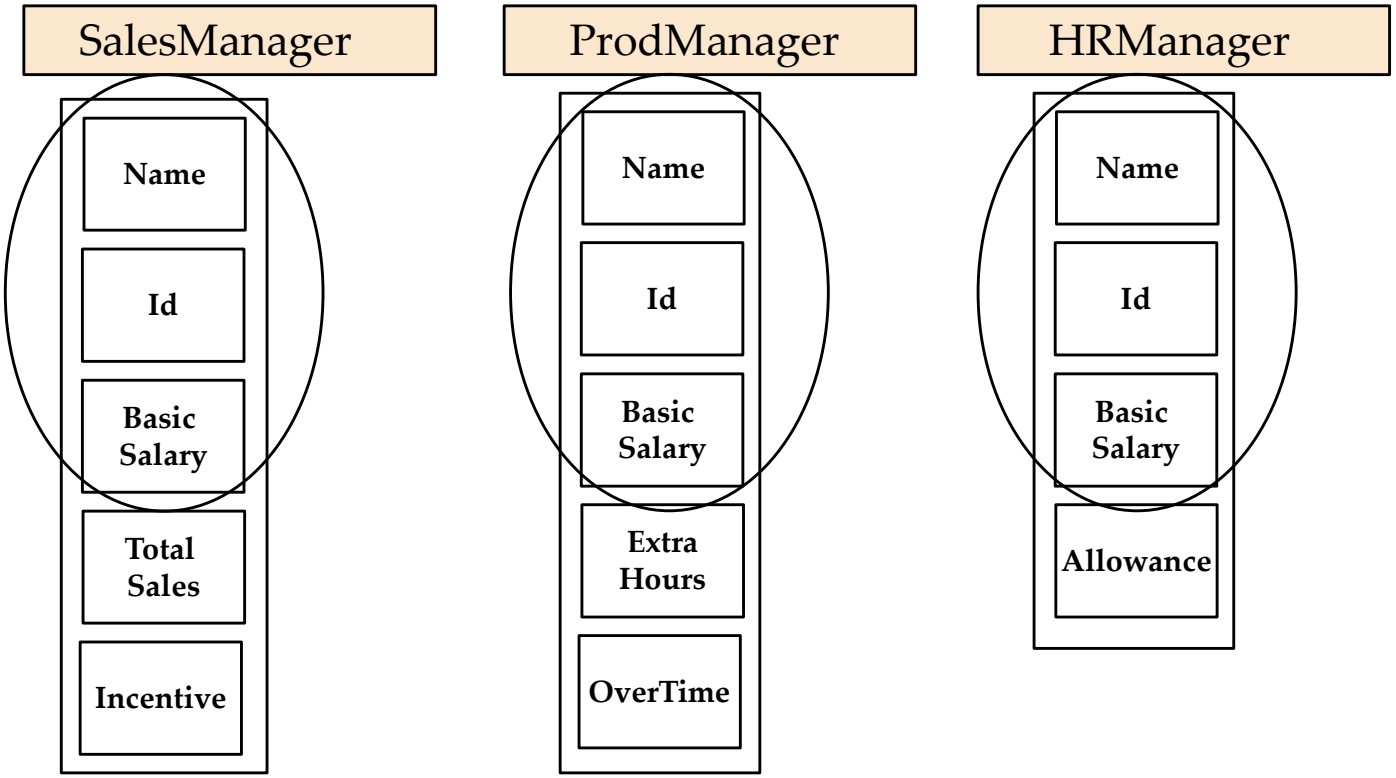
- Establishes 'is-a' kind of relationship
- Moves down from generalization to specialization.
- Most general at top to most specific at the bottom of inheritance chain



Inheritance



“IS-A” Type of Relationship



Syntax For Inheritance

Syntax:

class *DerivedClassName* : **access-level** *BaseClassName*

where

- **access-level** specifies the type of derivation
  - private by default, or
  - public
  - protected

Any class can serve as a base class

- Thus a derived class can also be a base class

Access Specifier & 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 any non-static member functions, friend functions, and non-member functions.	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Private in derived class. Can be accessed directly by any non-static member functions, friend functions.
Protected	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Private in derived class. Can be accessed directly by any non-static member functions, friend functions.
private	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.