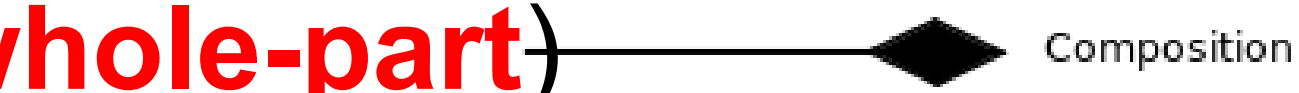
Class/Object Relationships

Composition

CS(217) Object Oriented Programming

Abeeda Akram

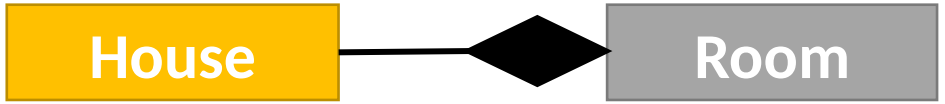
Composition (whole-part)



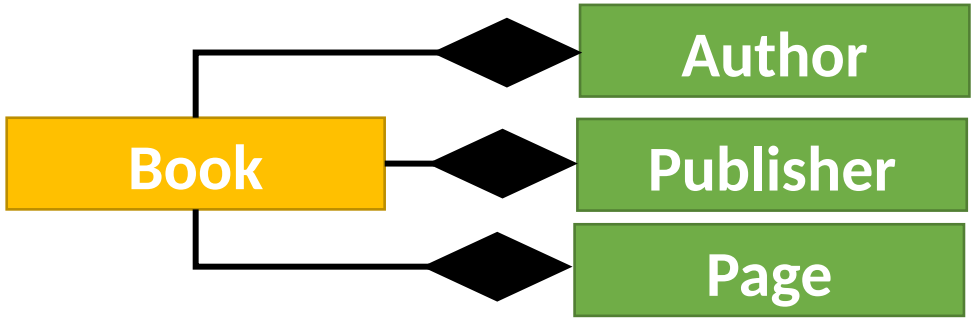
- Subset of aggregation relation where ownership is involved
- Strong relation
- Object of one class can contain object(s) of other class(s) for lifetime
  1. one-to-one,
  2. one-to-many
- Unidirectional object of container class knows about its parts
- Objects have dependent life time (creation and destruction)
  - When whole destroy part is also destroyed
  - Creation and destruction of part is controlled by whole
  - Part object can belong only to one whole class

Composition (whole-part) Examples

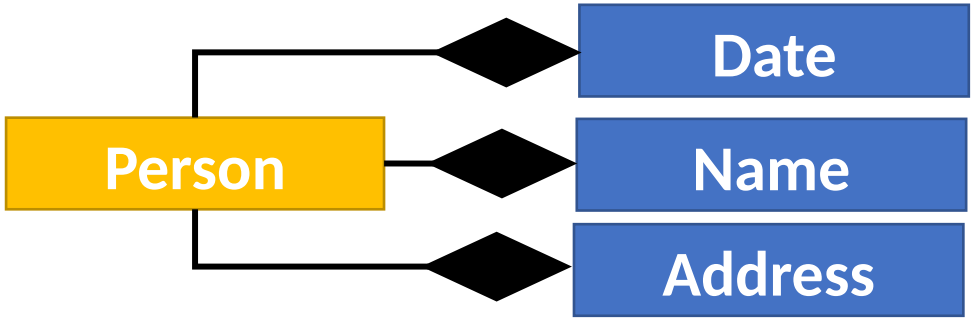
House cannot exist without rooms.



- Book cannot exist without author(s), ISBN, publisher, pages.



- Person cannot exist without name, date of birth, ID, address.



Composition (whole-part) Implementation

- Add object variable as **data member** of class.

```
void main(){
    A a, a2(3, 4);
}
```

- When object of A is created object of B is created inside A too.
- When object of A is destroyed part object B is also destroyed.

```
class B{
    int b;
public:
    B(int b=0){ this->b=b;}
};
class A{
    int  a;
    B  objB; //variable
public:
    A(int a=0, int b=0):objB(b){
        this->a=a;
    }
    //call parametrized constructor of part
    ~A(){}
    //nothing to do with part destroyed
    automatically
};
```

Composition (whole-part) Implementation

- Add **Pointer** to part object as member of class.

```
void main(){
    A a, a2(3, 4);
}
```

- When object of A is created object of B is created inside A too.
- When object of A is destroyed part object B is also destroyed.

```
class B{
    int b;
public:
    B(int b=0){ this->b=b;}
};
class A{
    int  a;
    B * objB; //pointer
public:
    A(int a=0, int b=0){
    this->a=a;
    objB = new B(b);
      //create part with whole
    }
    ~A(){ delete objB;}
    //destroy part with whole
};
```

Composition (whole-part) Exan

- Single class person controls every thing

```
class Person{
    int  pid;
// Name
    char * fname;
    char * lname;
//Date of Birth
    int day;
    int mon;
    int year;
//Address
    char * city;
    char *country;
    int streetNo;
    int houseNo;
};
```

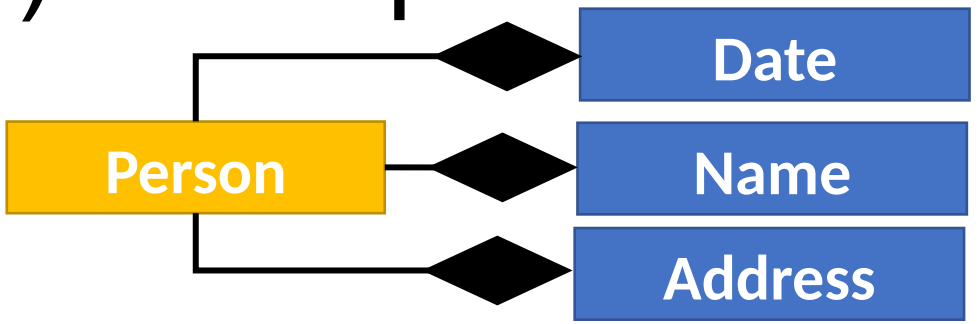
- Not scalable
- Error prone
- Not reusable in other class
- Redefine all attributes and functions separately for other classes
- For example student, doctor teacher, and patient

Composition (whole-part) Example

- Design separate classes

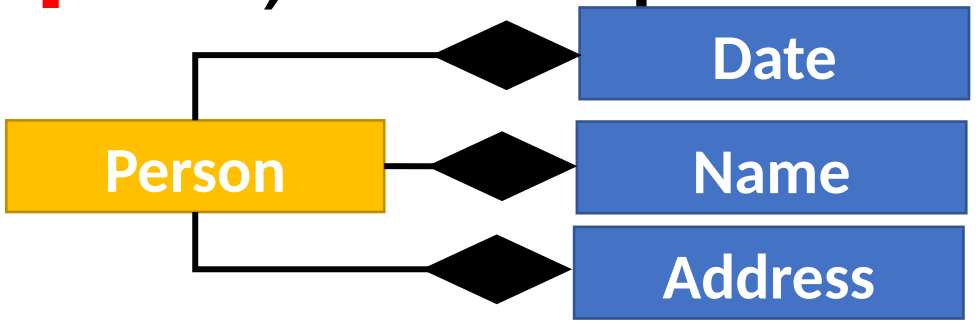
```
class name{
    char * fname;
    char * lname;
};
class date{
    int day;
    int mon;
    int year;
};
class address
    char * city;
    char *country;
    int streetNo;
    int houseNo;
};
```

```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
};
```



- Add objects as variables in class
- Scalable
- Less Error prone
- Reusable in other classes such as student, doctor and teacher, patient
- No need to redefine all attributes and functions separately for other classes

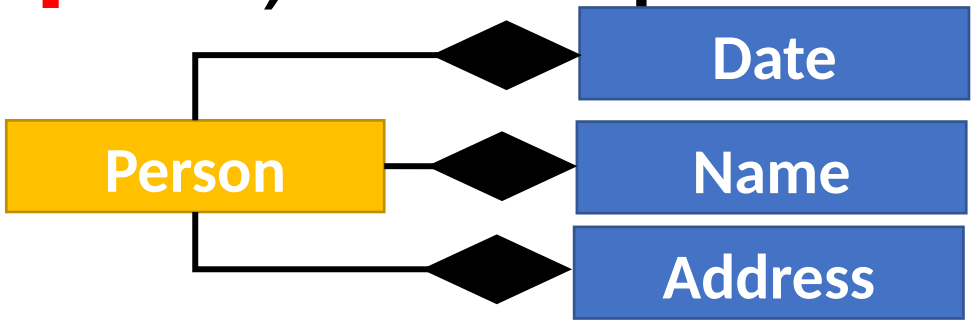
Composition (whole-part) Example



- Call functions of composed classes

```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
public:
    person(int pid, char*fn, char*ln, int d, int m, int y, char*city,
    char*country, int street, int house)
    :pname(fn,ln), dateofBirth(d,m,y), paddress(city, country, street, house)
    {
        this->pid=pid;
    }
//call parameterized constructors for object separately
};
```

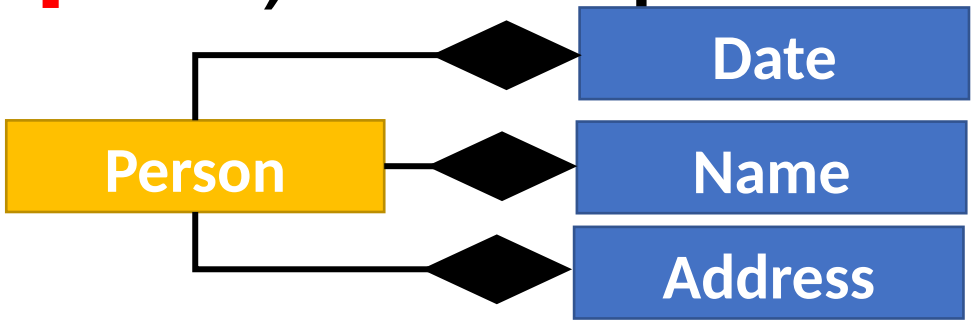
Composition (whole-part) Example



- Call functions of composed classes

```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
public:
    void setName(char*fn, char*ln){
    pname.setname(fn, ln);
    }
    void setDateofBirth(int d, int m, int y){
    dateofBirth.setDate(d,m,y);
    }
    void setAddress(char*city, char*country, int street, int house){
        paddress.setaddress(city, country, street, house);
    }
//Reuse functions of defined objects in person class
};
```

Composition (whole-part) Example

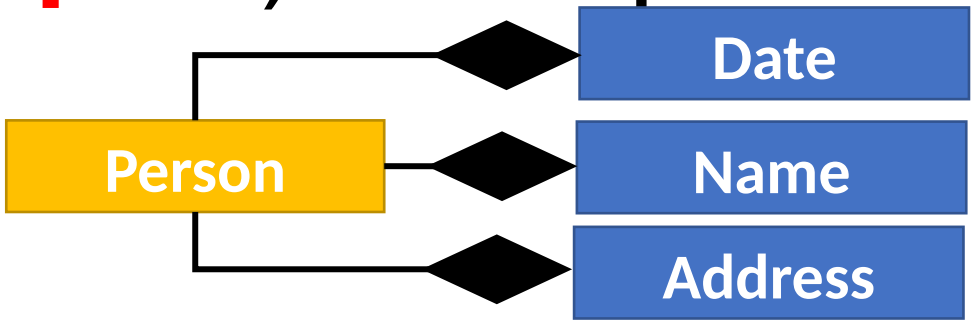


- Call functions of composed classes

```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
public:
    char * getfirstName(){
    return pname.getfirstname();
    }
    char * getlastName(){
    return pname.getlastname();
    }
    Date getDateofBirth(int d, int m, int y){
    return dateofBirth.getDate();
    }
//Reuse functions of defined objects in person class
};
```

Composition (whole-part) Example

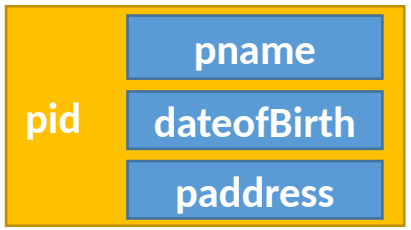
- Call functions of composed classes

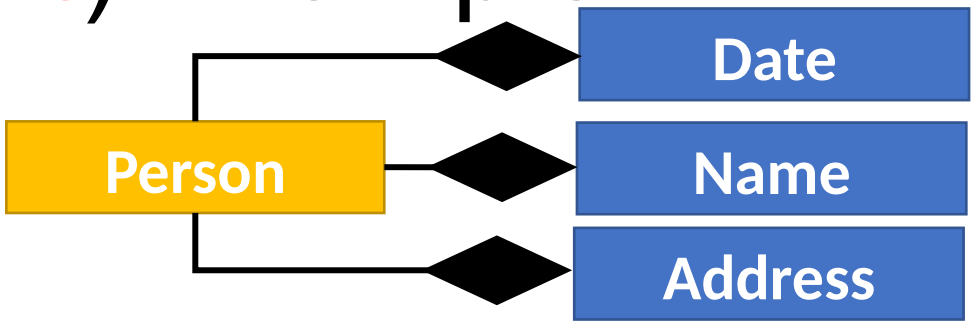



```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
public:
    friend ostream& operator<< (ostream& , const Person&);
  //Reuse functions of defined objects in person class
};
friend ostream& operator<< (ostream& out , const Person& p){
out<< “Person id:” << pid;
out<< “Name:” << pname;
out<< “Date of Birth:”   << dateofBirth;
out<< “Address:” << paddress;
} //Call ostream operator functions of name, date and address class
```

Composition (whole-part) Example

```
class Person{
int  pid;
name pname;
date dateofBirth;
address paddress;
};
void main(){ Person p; }
```





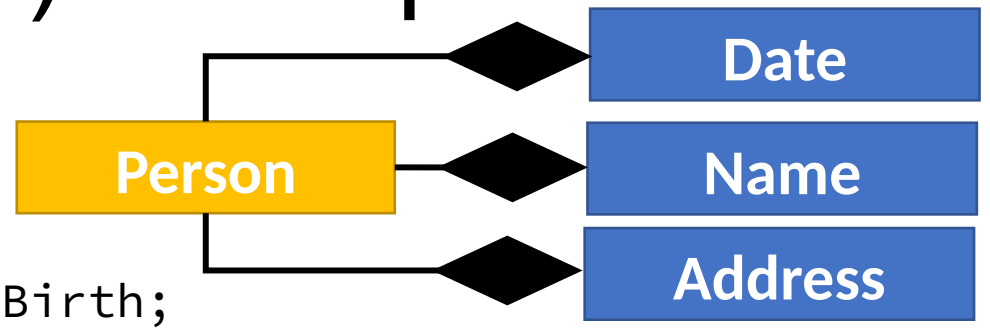
- Calling sequence
  - **Default constructor:** in same order as defined objects in class
    1)name 2)date 3)address 4)person
  - **Destructor:** in reverse order as defined objects in class
    1)person 2)address 3)date 4)name
  - **Parametrized constructor:** called in order of member initializer syntax
    : dateofBirth(d,m,y), pname(fn,ln), paddress(city, country, street, house)
    1)date 2)name 3)address 4)person

Composition (whole-part) Example

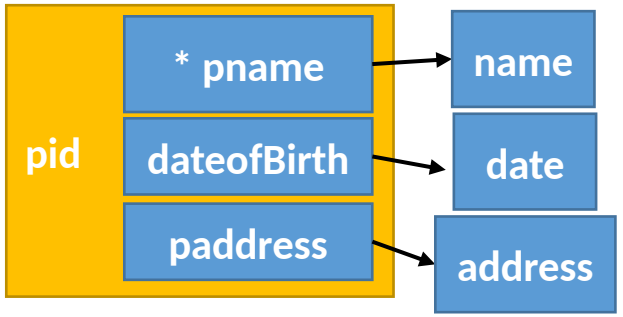
- Design separate classes

```
class name{
    char * fname;
    char * lname;
};
class date{
    int day;
    int mon;
    int year;
};
class address
    char * city;
    char *country;
    int streetNo;
    int houseNo;
};
```

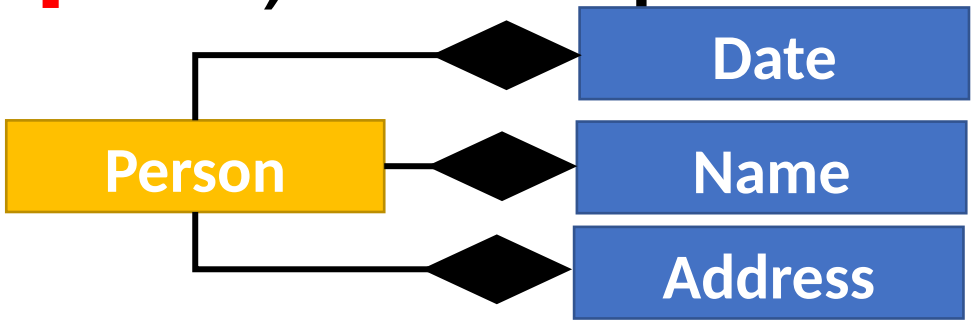
```
class Person{
    int  pid;
    name * pname;
    date * dateofBirth;
    address * paddress;
};
```



- Add objects as pointer to variables in class
- Scalable Less Error prone
- Reusable in other classes such as student, doctor and teacher, patient
- No need to redefine all attributes and functions separately for other classes



Composition (whole-part) Example

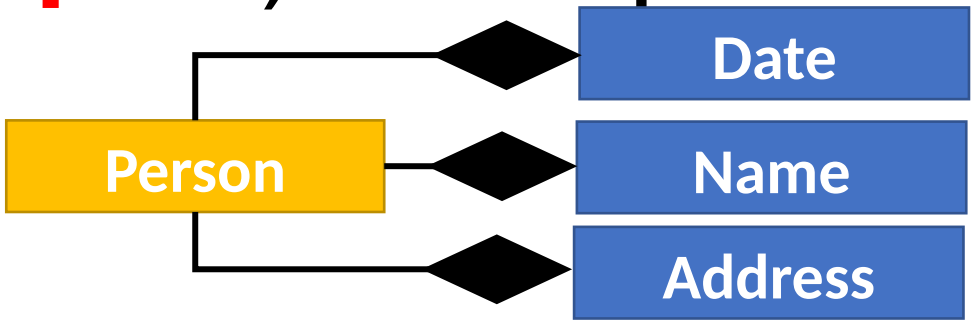


- Call functions of composed classes

```
class Person{
int  pid;
name * pname;
date * dateofBirth;
address * paddress;
public:
    person(int pid, char*fn, char*ln, int d, int m, int y, char*city, char*country,
    int street, int house)
    {   pname = new name(fn, ln);
        dateofBirth = new date(d, m, y);
        paddress = new address(city, country, street, house);
       this->pid=pid;
    }
//call parameterized constructors for dynamic objects
};
```

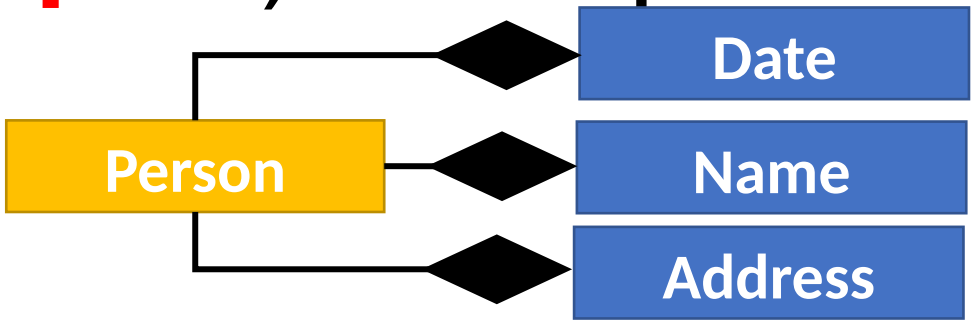
Composition (whole-part) Example

- Call functions of composed classes




```
class Person{
int  pid;
name * pname;
date * dateofBirth;
address * paddress;
public:
    void setName(char*fn, char*ln){
    pname->setname(fn, ln);
    }
    void setDateofBirth(int d, int m, int y){
    dateofBirth->setDate(d,m,y);
    }
    void setAddress(char*city, char*country, int street, int house){
        paddress->setaddress(city, country, street, house);
    }
//Reuse functions of defined objects in person class
};
```

Composition (whole-part) Example

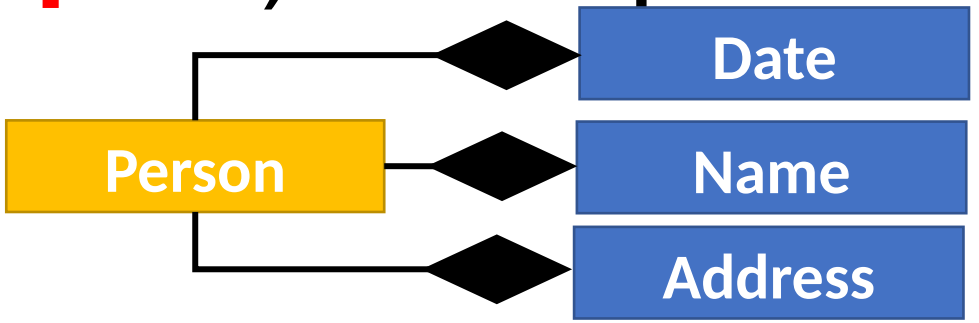


- Call functions of composed classes

```
class Person{
int  pid;
name * pname;
date * dateofBirth;
address * paddress;
public:
    ~person()
    {   delete pname;
        delete dateofBirth ;
        delete paddress;
    }
//call destructors of composed objects within destructor of person, sequence is
controlled by programmer
};
```

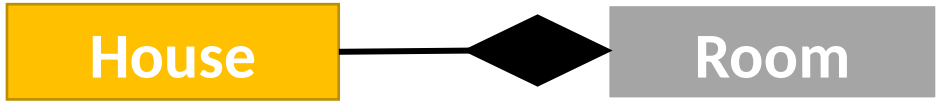
Composition (whole-part) Example

```
class Person{
    int  pid;
    name * pname;
    date * dateofBirth;
    address * paddress;
};
void main(){Person p;}
```



- Calling sequence
  - System will only call constructors and destructor of Person
  - Programmer controls the ordering and call of constructors and destructor, for composed objects.

Composition (whole-part) Example



- **One to many**

```
class Room{
    int rid;
    float size; //in sqfeet
public:
    Room(int t=0, float s=0.0f){
        tid=t;
        Size = s;
        }
};
void main(){ House h1(2, 4); }
```

```
class House{
    int  hid;
    int noofrooms;
    Room ** rList; //pointers list
public:
    House(int id=0, int rooms=2){
        hid = id;
        noofrooms = rooms;
        rList = new Room*[noofrooms];
        for(int i=0; i<noofrooms; i++)
            rList[i] = new Room(i+1,
    i+10*i+10);
    } //create part with whole
    ~House(){
        for(int i=0; i<noofrooms; i++)
            delete rList[i];
        delete [] rList;
    } //destroy part with whole
};
```