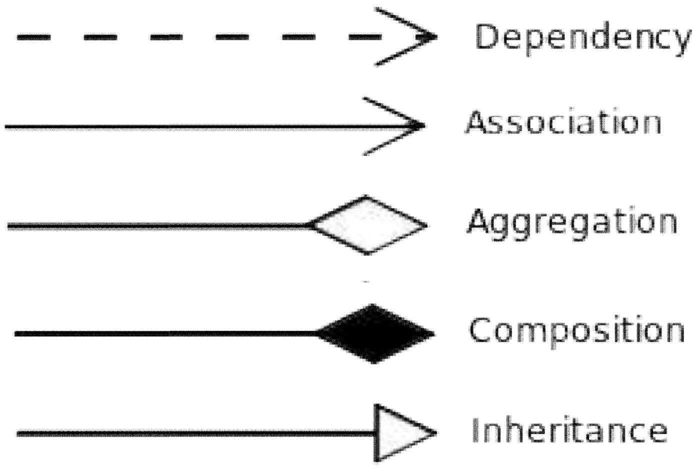
Class/Object Relationships

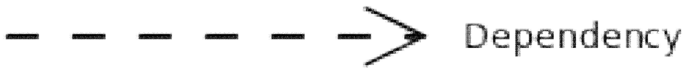
CS(217) Object Oriented Programming

Relationship between classes and Objects

1. Dependency (use-a)
2. Association (use-a)
3. Aggregation (has-a)
4. Composition (whole-part)
5. Inheritance (is-a)

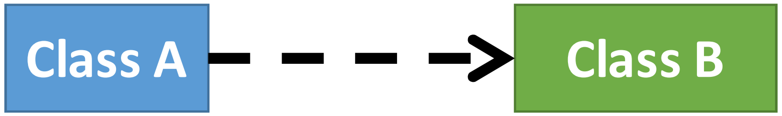


Dependency (use-a)



- Very weak relation
- Object of one class uses objects of other class for a short amount of time (in a function) to perform a specific task

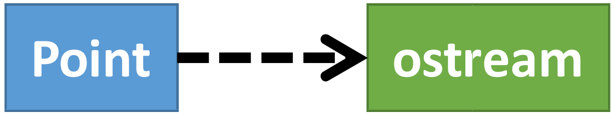
- Life time (creation and destruction) of objects is independent
- Unidirectional relation, used class object is unaware of dependent class



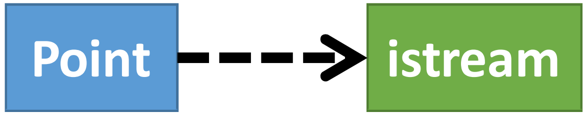
Uses some functions of

Dependency (use-a) Example

- ostream and istream objects are used in operator functions.
  - `friend istream& operator>> (istream& , Point&);`
  - `friend ostream& operator<< (ostream& , const Point&);`
- ostream and istream object are neither created inside class object, nor they are related to the object
- Life time (creation and destruction) of Point, ostream and istream is independent



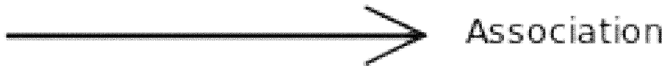
Uses stream functions of ostream



Uses stream functions of istream

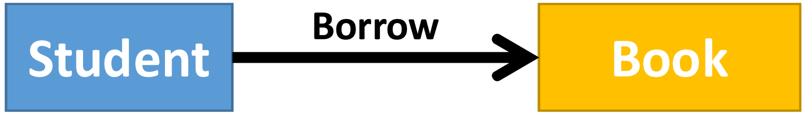
- Unidirectional
  - istream and ostream classes are unaware of existence of Point class and its objects,
  - but Point class is aware of the use in operator functions

Association (use-a)



- Weak relation, no ownership of objects is involved
- Object of one class can be associated with object(s) of other class(s) for performing some tasks
  1. one-to-one,
  2. one-to-many
  3. many-to-many
- Objects have independent life time (creation and destruction)
- Objects are unrelated to one another
- Objects may or may not know about the existence of the object
  - Unidirectional
  - Bidirectional

Association (use-a) Examples

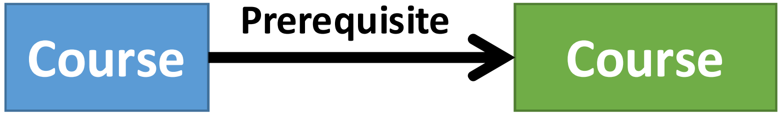

Student
Borrow
Book

- One-to-Many relation, one student can borrow many books from a library.
- No owner ship and lifetime is involved in this relationship.
- A list of ids of borrowed books can be added to student.

```
class student{
private:
    int  sId;
    int *borrowedBooks;
    //Maintain the list of borrowed
    books
public:
    void borrowABook(const int & bid);
    void ReturnABook(const int & bid);

};
```

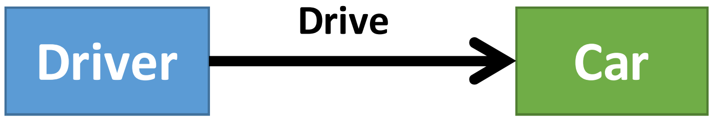
Association (use-a) Examples

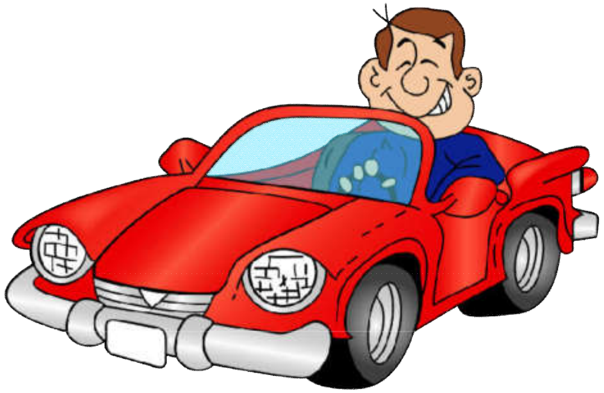


- **Reflexive association**
- One-to-Many relation, one course can have many prerequisite courses.
- No owner ship and lifetime is involved in this relationship.
- How to link courses?
    - A list of ids of prerequisite courses can be added to course.

```
class course{
private:
    int  cId;
    int *prerequisiteCourses;
    //Maintain the list of prerequisites
public:
    void AddPrerequisite(const int & cid);
    void RemovePrerequisite(const int & cid);
};
```

Association (use-a) Examples



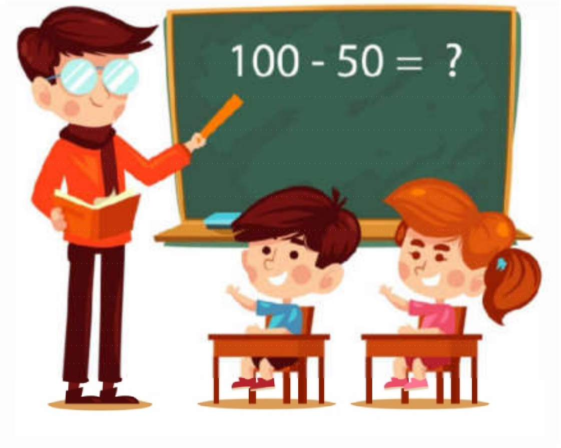


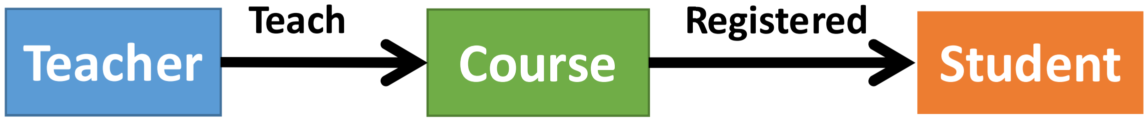
- One-to-Many relation, one driver can drive many cars.
- A list of ids of derived cars can be added to driver class.

```
class driver{
private:
    int  dId;
    int *derivingCars;
    //Maintain the list of derived cars
public:
    void AddCar(const int & cid);
    void RemoveCar(const int & cid);


};
```

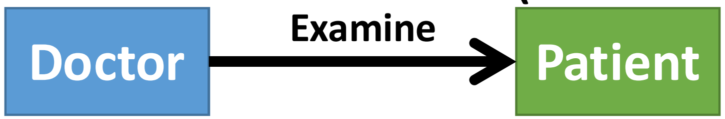
Association (use-a) Examples




Teacher
Teach
Course
Registered
Student

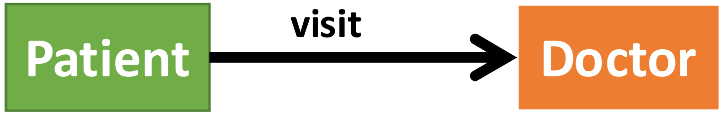
- How to link teacher, course and student in the system?
  - One teacher can teach many courses
  - In one course many students are registered
  - Add courses list in teacher
  - Add student’s ids list in a course

Association (use-a) Examples



- Unidirectional one to many
- A doctor can examine many patients to earn money.

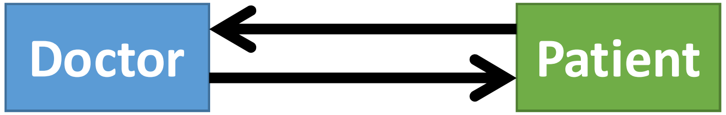
```
class Doctor{
private:
    int  dId;
    int *pateintsExamined;
    //Maintain the history of examined
    patients
public:
    void examinPatient(const int & pid);
};
```



- Unidirectional one to many
- A patient can visit many doctors for different ailments.

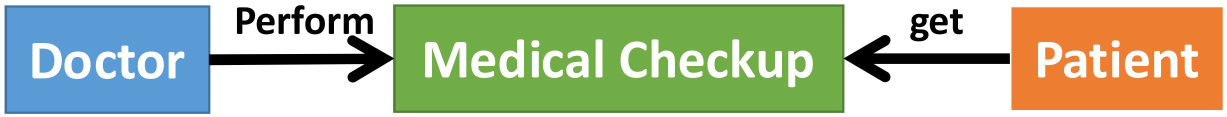
```
class Patient {
private:
    int pId;
    int* visitedDoctors;
    //Maintain the history of patient
    on every visit a docotor
public:
    void visitDoctor(const int & did);
};
```

Association (use-a) Examples



- Many-to-Many relation.
- Bidirectional, doctor knows patient and patient knows doctor.
- A doctor can examine many patients.
- A patient can visit many doctors for different ailments.
- No owner ship and lifetime is involved in this relationship.
- How to link doctor and patient in system?
  - Add ids of examined patients in doctor object?
  - Add ids of visited doctors in patient object?

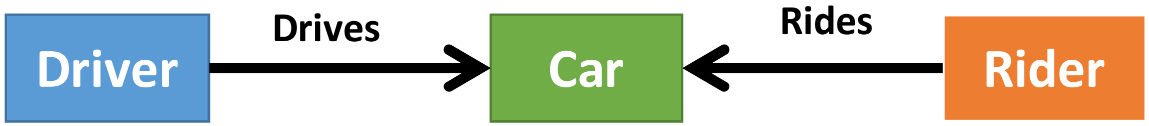
Association (use-a) Examples



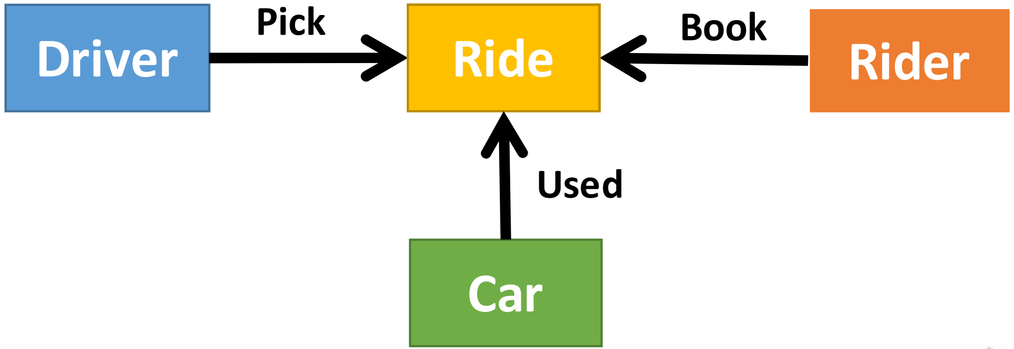
- Bidirectional many to many, breakdown in two one to many relations.
- Add association relation as a new class.
- Store id of doctor and patient for each checkup.
- Add checkups list in both doctor and patient to link them.
  - A doctor can perform many checkups
  - A patient can get many checkups

```
class Checkup{
private:
    int  dId;
    int  pId;
    int checkupId;
    //Maintain the doctor and patient ID
public:
 CheckUp(const int & pid, const int &
did);
};
```

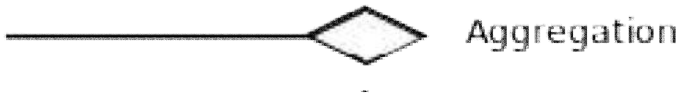
Association (use-a) Examples



- A driver can drive different cars.
- A rider can ride different cars with different drivers.
- A car can be used by different riders and drivers.
- Ternary Relation: How to link driver, car and rider in the system?
  - Add an association class Ride.
  - Add id of driver, car and rider for a ride.
  - Add rides list in driver, rider and car.



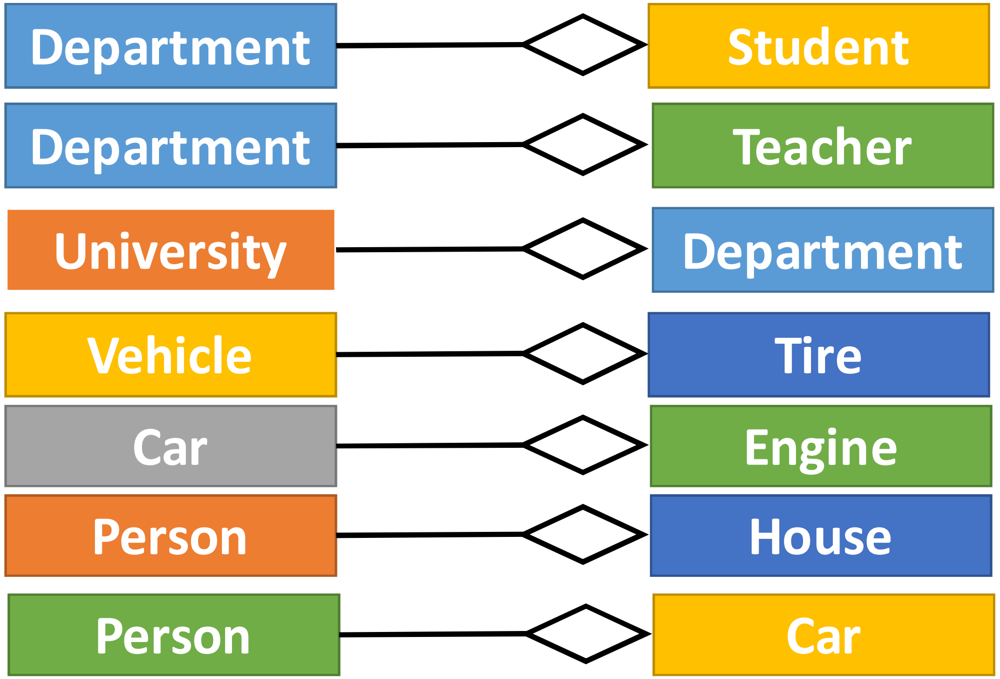
Aggregation (has-a)



- Subset of association relation where ownership is involved
- Weak relation
- Object of one class can contain object(s) of other class(s) for specific amount of time
  1. one-to-one,
  2. one-to-many
- Unidirectional object of container class knows about its parts
- Objects have independent life time (creation and destruction)

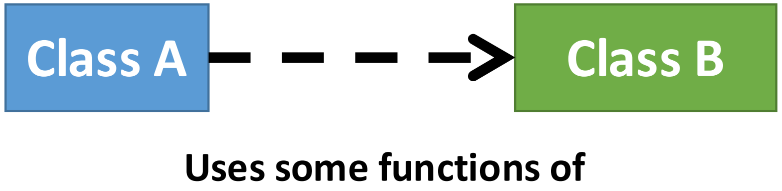
Aggregation (has-a)Examples

- One department has many students.
- A department has many teachers.
- A University has many departments.
- A vehicle has many tires.
- A car has an engine.
- A person owns a house.
- A person owns many cars.



Summary

1. Dependency (use-a)

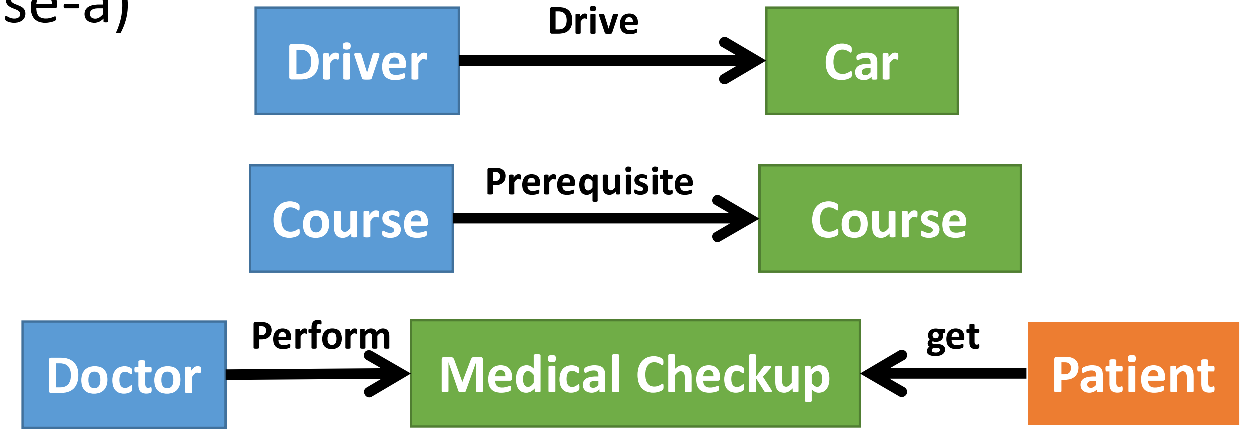


Class A

Class B

Uses some functions of

2. Association (use-a)



Driver

Drive

Car

Course

Prerequisite

Course

Doctor

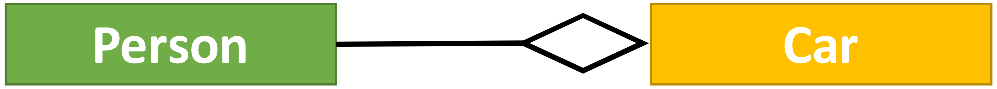
Perform

Medical Checkup

get

Patient

3. Aggregation (has-a)



Person

Car