Elements of Unified Modeling Language

Shahram Rahatlou

SAPIENZA
UNIVERSITÀ DI ROMA

http://www.roma1.infn.it/people/rahatlou/programmazione++/

Corso di Programmazione++

Roma, 22 June 2009

Today's Lecture

- Introduction to UML
  - object modeling language
- Class diagrams
- Relations between classes
- Suggestions for further reading

Object Oriented Analysis (OOA)

- Build a system composed of objects
- Behavior of system defined by collaboration between objects through sending messages to each other
- State of system defined by states of individual collaborating objects
- Does not take into account implementations constraints such as distribution and persistency
- Result of OOA is a conceptual model focused on ideas and concepts to be implemented

Object Oriented Design (OOD)

- Start from conceptual model provided by OOA and add implementation constraints, e.g. specific programming language
- Treat the objects as instances of collection of classes within a class hierarchy
- Typically four stages in Design:
  - Identify classes and objects
  - Identify their responsibilities
  - Identify their relationship
  - Provide class interface and implementation

Object Modeling Language

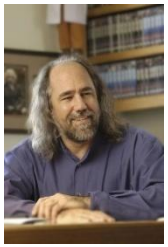
- Standardized set of symbols and relations between them to model object oriented design

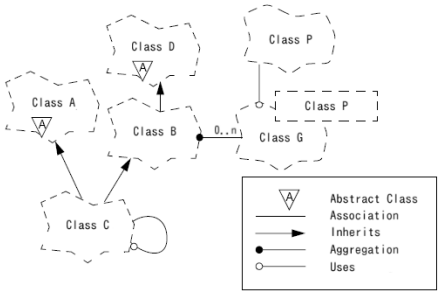
- Visual and graphical representation provides higher level of abstraction important in early analysis and design stage
  - Focus on interaction and relation between objects
  - Define interface rather than internal structure

- Software modeling tools can be used to implement code from visual modeling diagrams

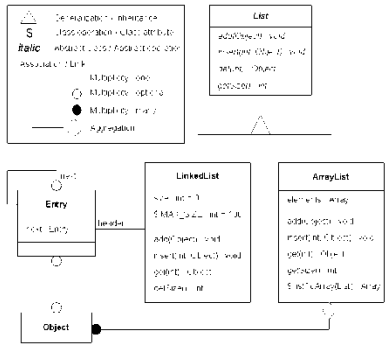
Brief History of Object Modeling Language

- Booch Method
  - Developed by Grady Booch
  - Better for design





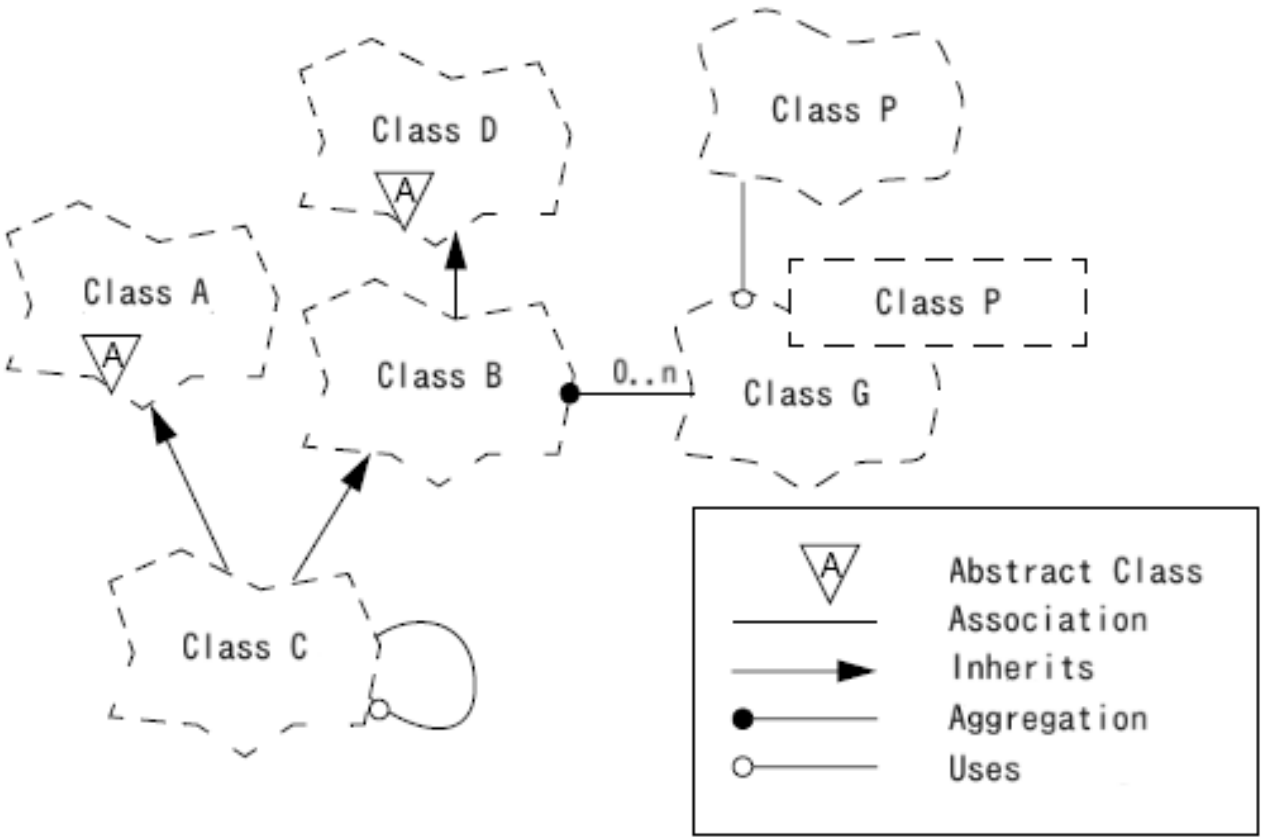
- Object modeling technique (OMT)
  - Developed by Jim Rumbaugh
  - Better for analysis



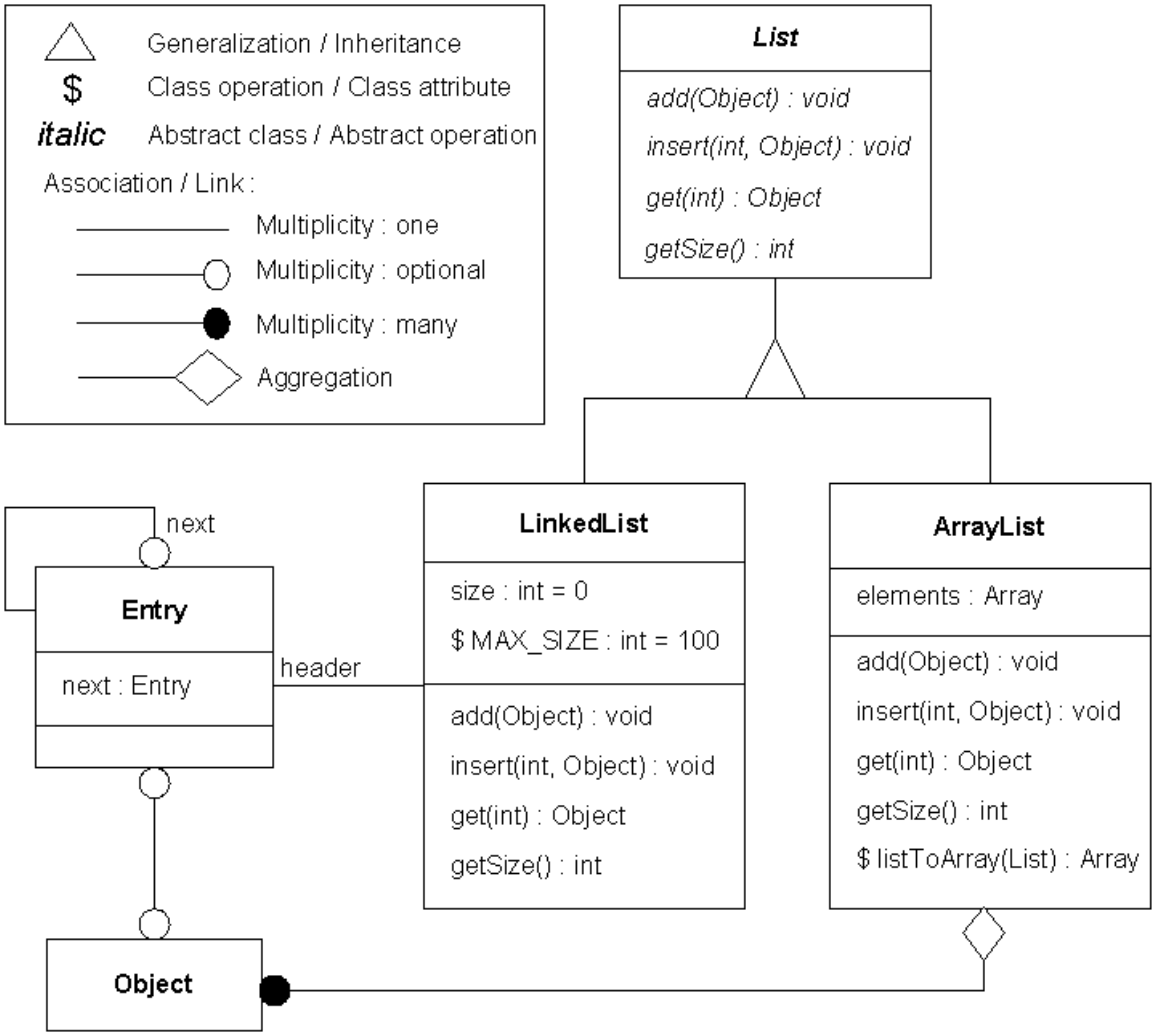
- Objectory
  - Developed by Ivar Jacobsen
  - Treat Use Cases
    - Use case: interaction between system and end user to achieve a specific goal

- Class-Responsibility-Collaboration Cards
  - Proposed by Ward Cunningham

Booch Method

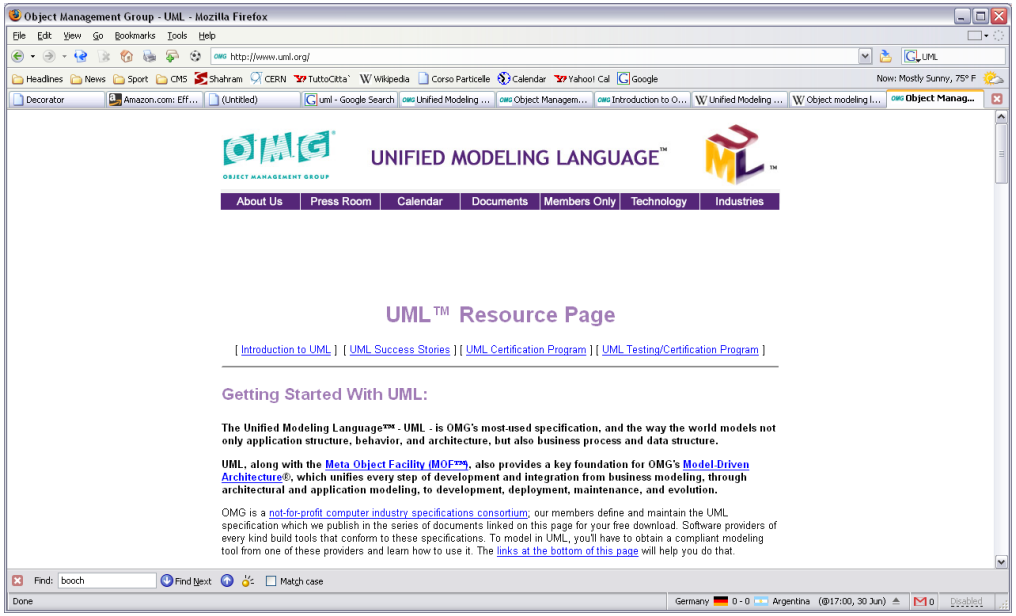


Object Modeling Technique



Unified Modeling Language (UML)

- Many approaches on the market by mid 1990s
- Object Management Group (OMG) called for development of a unified approach
- Consortium including Booch, Jacobsen, and Rumbaugh has developed what today is called Unified Modeling Language

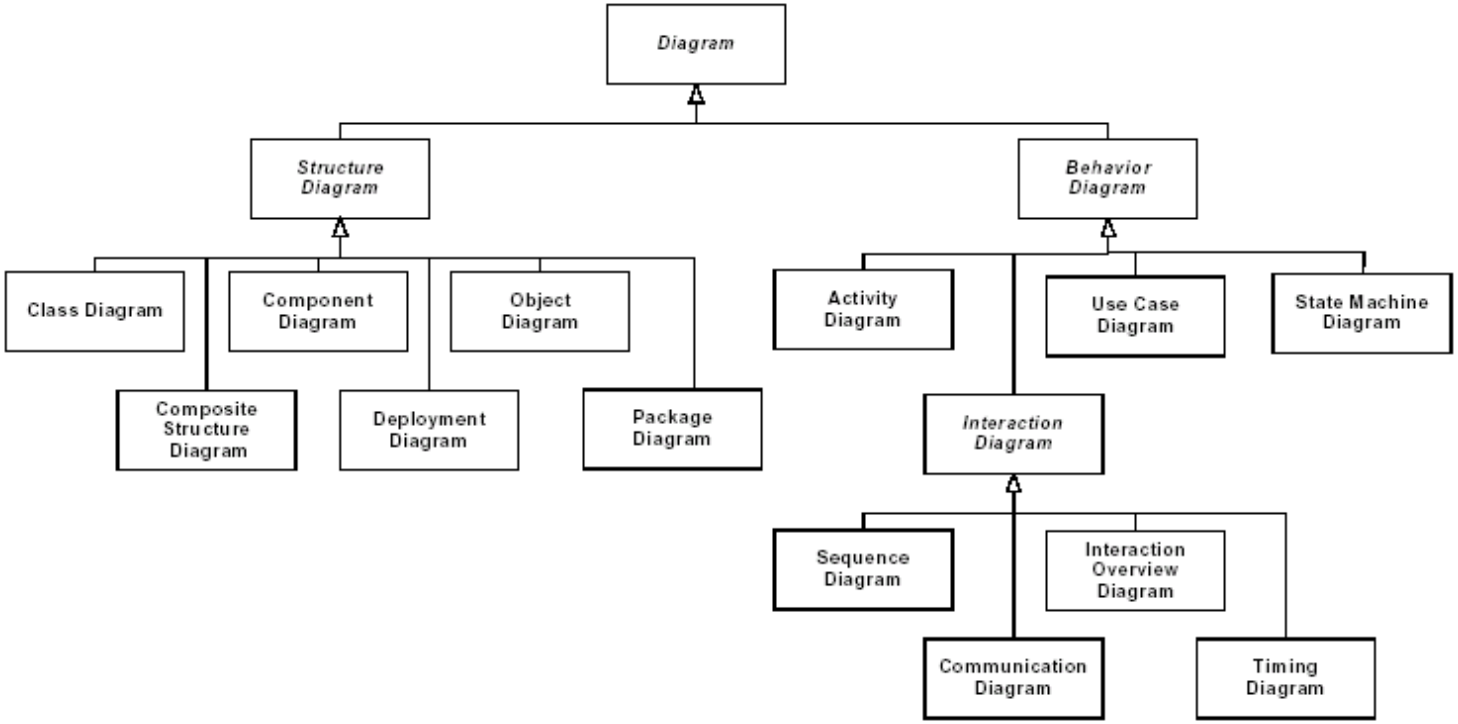


http://www.uml.org/

Unified Modeling Language (UML)

UML Diagrams

- Thirteen diagrams in UML 2.0 organized



Categories of Diagrams

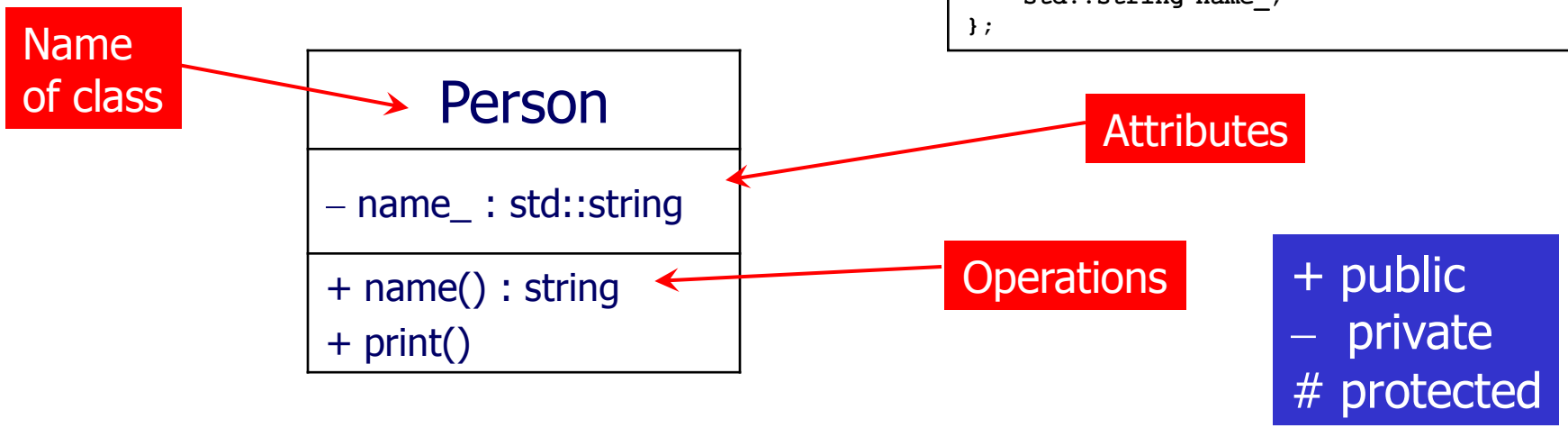
- Structure diagrams: emphasize what things must be in the system
  - Class diagram
  - Component diagram
  - Object diagram
  - Composite structure diagram
  - Deployment diagram
  - Package diagram
- Behavior diagrams: emphasize what must happen in the system
  - Activity diagram
  - Use case diagram
  - State Machine diagram
- Interaction Diagrams: subset of behavior diagrams, emphasize flow of control and data among the things in the system
  - Sequence diagram
  - Collaboration (UML 1.x)/Communication diagram (UML 2.0)
  - Interaction overview diagram (UML 2.0)
  - Timing diagram (UML 2.0)

Class Diagram

- Type of static structure diagram describing structure of a system by showing
  - system's classes
  - relationships between classes

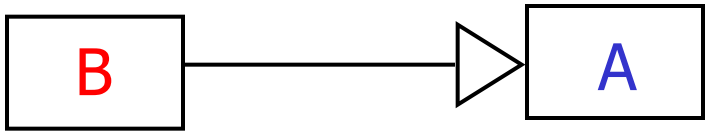
- Graphical representation: box with 3 compartments for
  - Name of class
  - attributes or data members
  - operations or methods

```
class Person {
  public:
    Person(const std::string& name);
    ~Person();
    std::string name() const { return name_; }
    void print() const;

  private:
    std::string name_;
};
```

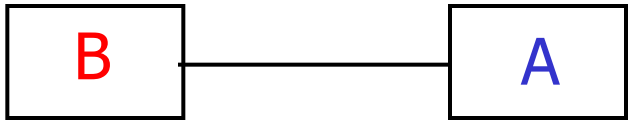


Relations between Classes

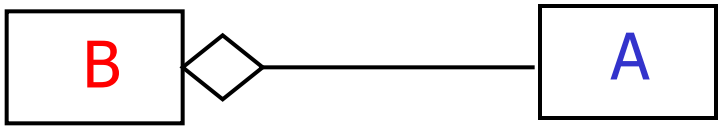
- Generalization or Inheritance
  - an is-a relationship



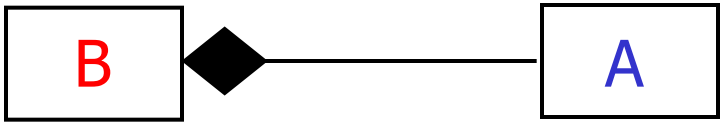
- Association
  - can be mutual or uni-directional



- Aggregation
  - Whole/part relationship. no lifetime control



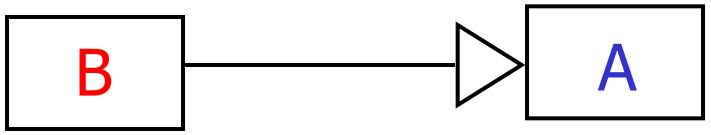
- Composition
  - Aggregation with lifetime control

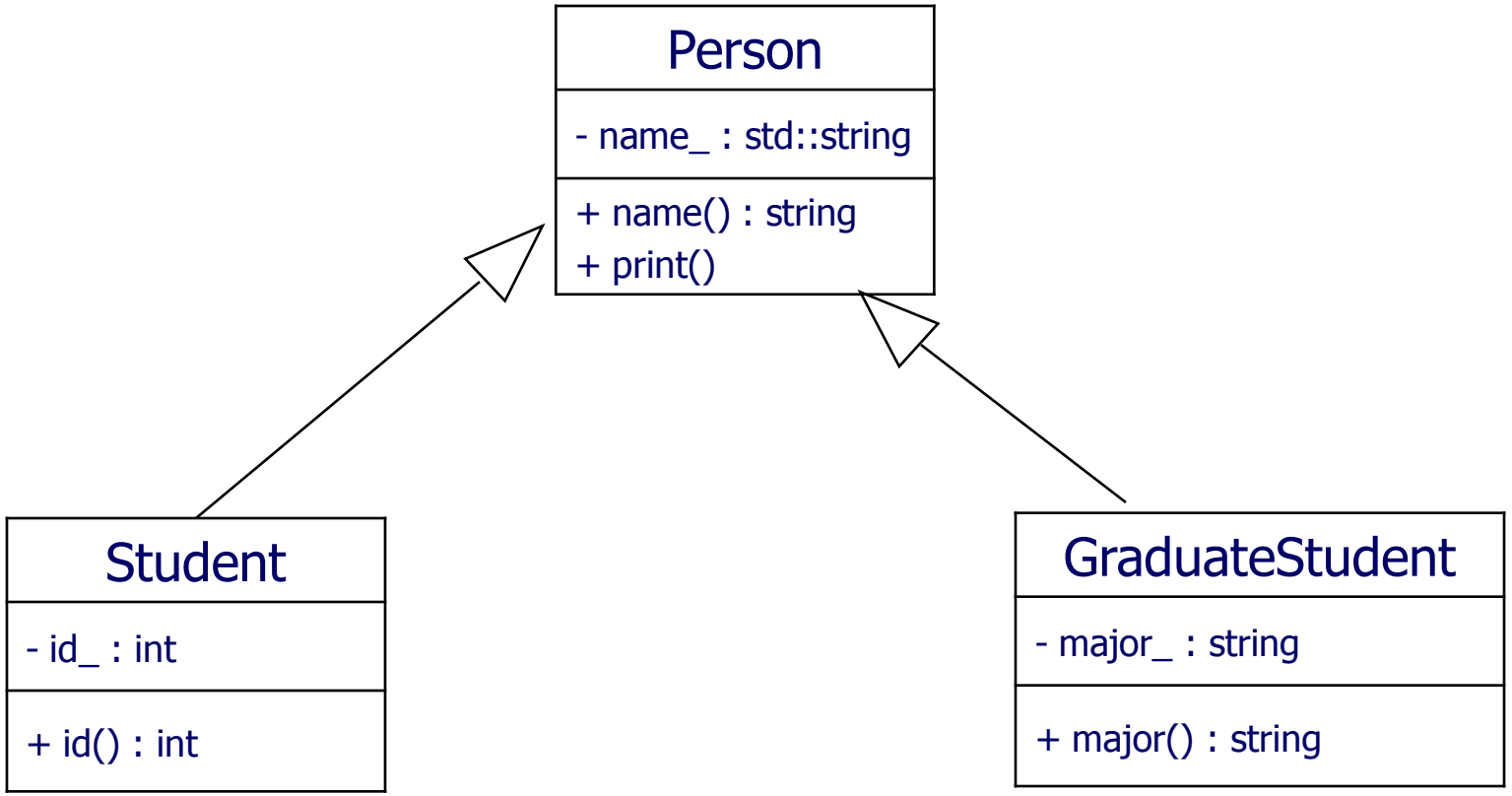


- Dependence
  - uni-directional association
  - only B knows about A

Generalization or Inheritance

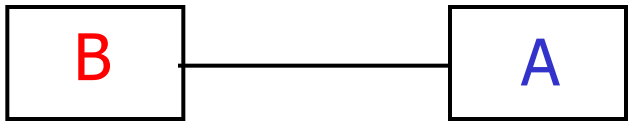
- Is-A relationship between A and B: B is also an A
  - relationship between a base class (super-type, parent) and a derived class (sub-type, child)





Association

- A and B exchange messages
  - Call methods of each other

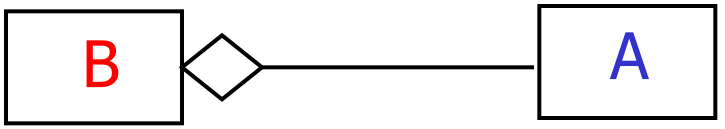


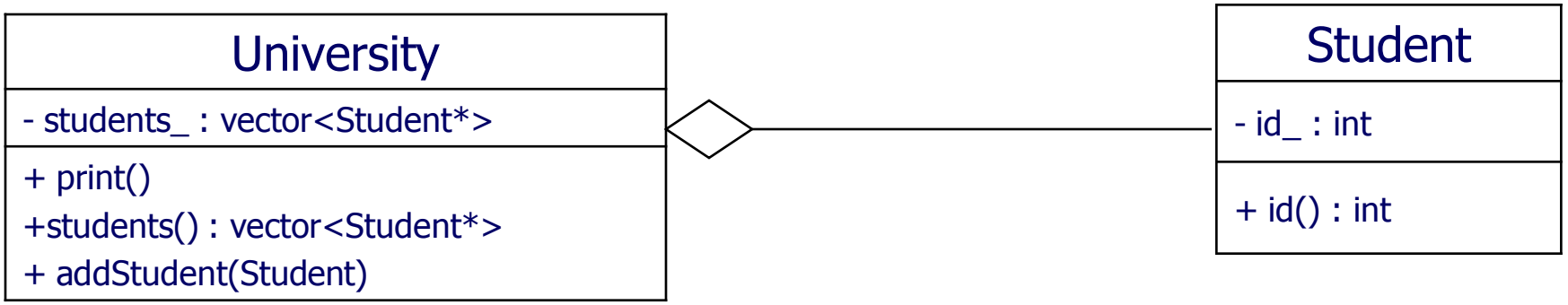

```
class Department {

  private:
    University* myUniversity_;

  public:
    void print() {
      cout << "My University is: : <<
           << myUniv_->name()
           << endl;
    }

}
```

```
class University {

  private:
     Department* myDep_;

  public:
     string department() {
      return myDep_->name();
     }

}
```

Aggregation

- Whole/part association with no lifetime control
  - B contains a pointer to A
  - B does not control lifetime of A
    - A exists regardless of B





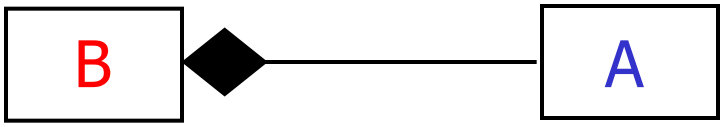

```
class University {

  private:
     vector<Student*> students_;

  public:
     vector<Student*> students() {
      return students_;
     }
     void addStudent(Student* s) {
      students_->push_back(s);
     }

}
```

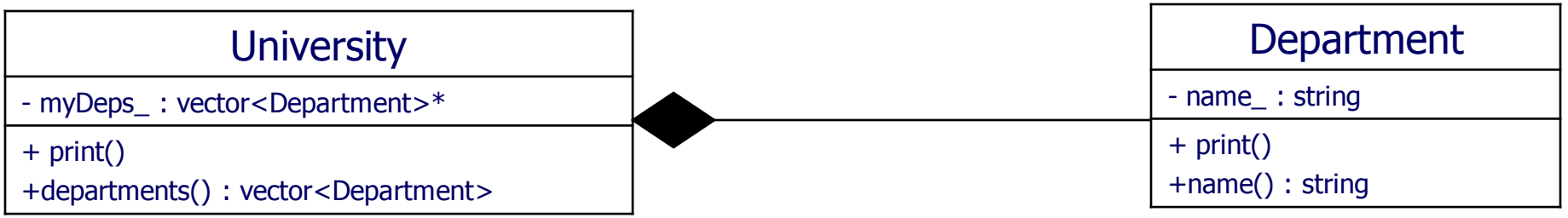
All instances of Student exist regardless of the instance of University

Only keeps pointers but does not control lifetime of objects pointed to

Composition

- Whole/part association with lifetime control
  - B contains instance of A
  - B is responsible for creation of its copies of A and their destruction
  - B can transfer ownership of it's a to others



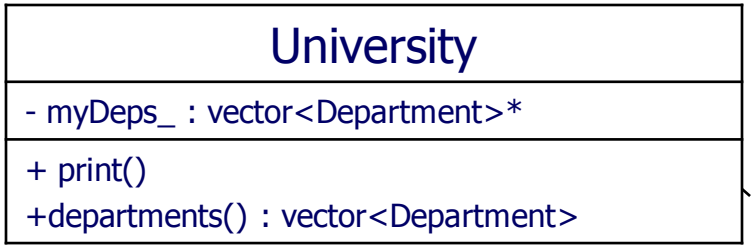


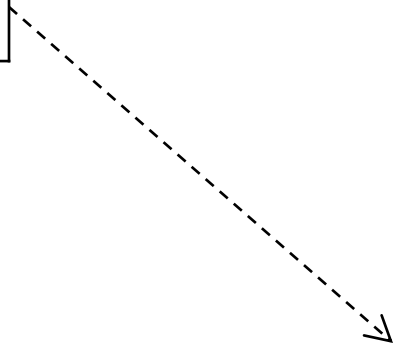

```
class University {

  private:
     vector<Department>* deps_;

  public:
     University() {
       deps_ = new vector<Department>;
       deps_->push_back("physics");
     }
     ~University() { delete deps_; }

     vector<Departments> departments() {
      return *deps_;
     }

}
```

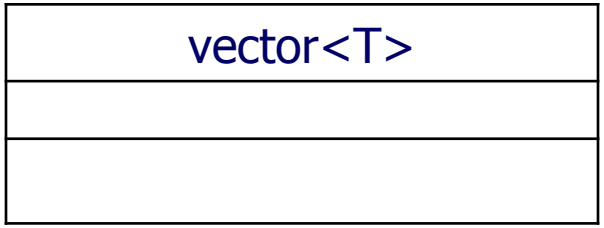
```
class Department {

  private:
    string name_;

  public:
    string name() { return name_; }

}
```

Dependence

- B knows about A but A has no knowledge of B
  - Mostly when A is used in definition of A





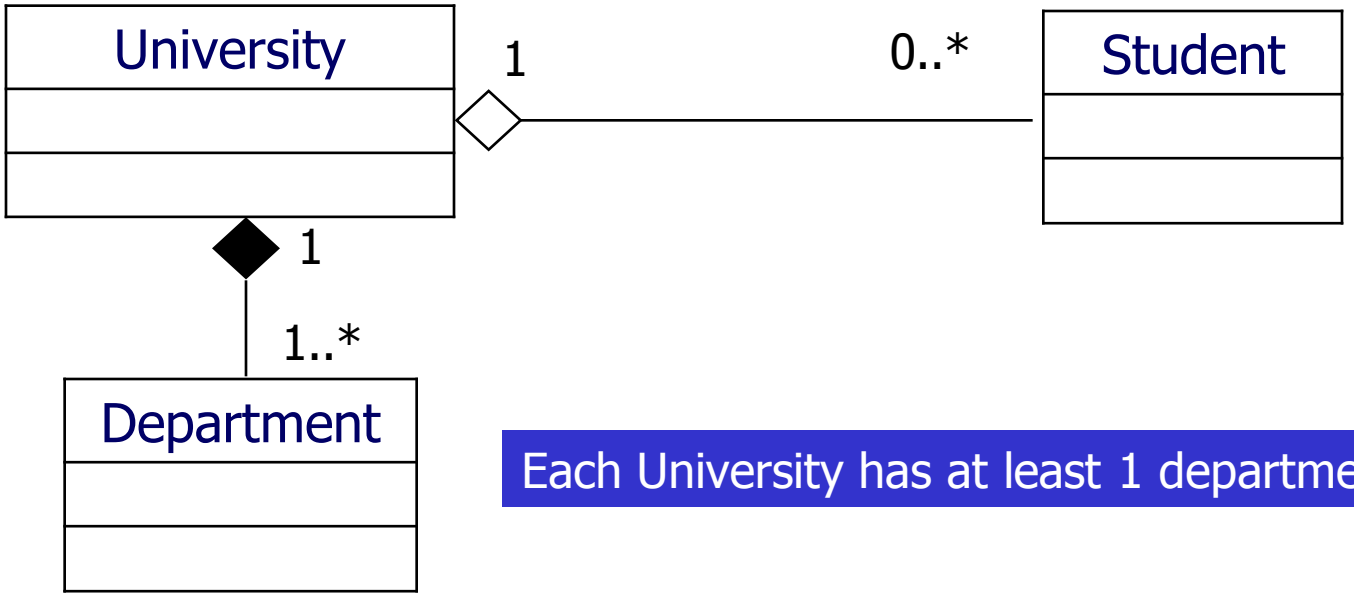



```
class University {

  private:
     vector<Department>* deps_;

  public:
     University() {
       deps_ = new vector<Department>;
       deps_->push_back("physics");
     }
     ~University() { delete deps_; }

     vector<Departments>* departments() {
      return *deps_;
     }

}
```

Multiplicity (a.k.a Cardinality)

- Multiplicity of a role describes number of instances participating in the association
  - * or 0..* : zero to many
  - 1..* : one to many
  - 0..1 : zero or one
  - 1 : one and only one
  - n..m : n or m

University might have no student

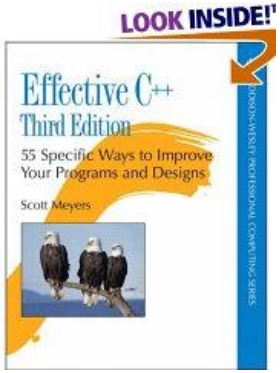


Each University has at least 1 department

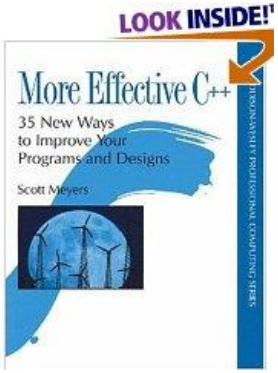
Additional Readings

- Few very good books to improve your skills and learn more about object oriented programming techniques

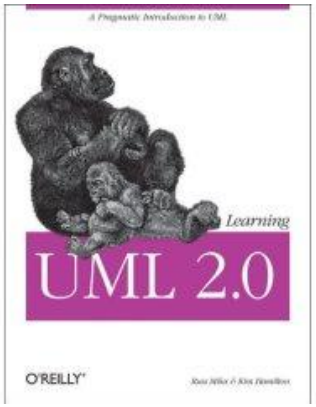
- Effective C++ : 55 Specific Ways to Improve Your Programs and Designs, Scott Meyers

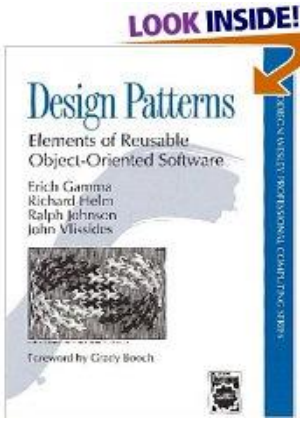


- More Effective C++: 35 New Ways to Improve Your Programs and Designs, Scott Meyers



- Design Patterns: Elements of Reusable Object-Oriented Software, E. Gamma et al.





- Learning UML 2.0, K. Hamilton, R. Miles