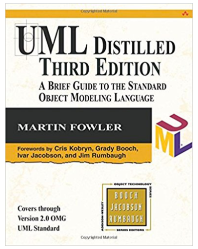
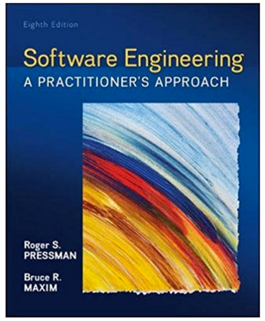
Modeling

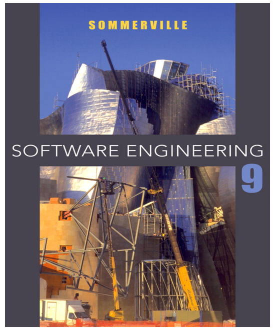

UML Distilled
Third Edition
A Brief Guide to the Standard Object Modeling Language
Martin Fowler
Forewords by Cris Kobryn, Grady Booch, Ivar Jacobson, and Jim Rumbaugh
Covers through Version 2.0 OMG UML Standard

Chapter 3: Class diagrams: the essential

Chapter 4: Sequence diagrams



Appendix 1: an introduction to UML



Chapter 7.1.3 object class identification

Chapter 7.1.4 design models

Design

OO
Class Diagram
Sequence Diagram

Object-Oriented Programming, Classes

- Class
    - Data + Operation

- Encapsulation
- Polymorphism
- Inheritance

- Enhance modularity!

Encapsulation

- “the packing of data and functions into a single component. The features of **encapsulation** are supported using classes. It allows selective hiding of properties and methods in a class by building an impenetrable wall to protect the code from accidental corruption.”

Encapsulation

- “the packing of data and functions into a single component. The features of **encapsulation** are supported using classes. It allows selective hiding of properties and methods in a class by building an impenetrable wall to protect the code from accidental corruption.”

- Implication to design?

Polymorphism

- “to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes”
- “the provision of a single interface to entities of different types.”
- Examples

Polymorphism

- “to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes”
- “the provision of a single interface to entities of different types.”

- Implication to design?
- Benefits?
- Problems?

Inheritance

- “a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces.”

- Examples

Inheritance

- “a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces.”

- Implication to design?
- Benefits?
- Problems?

Class diagram

- Describes the types of objects in the system
- Describes the static relationships among them

http://en.wikipedia.org/wiki/Class_diagram

How to decide/design classes?

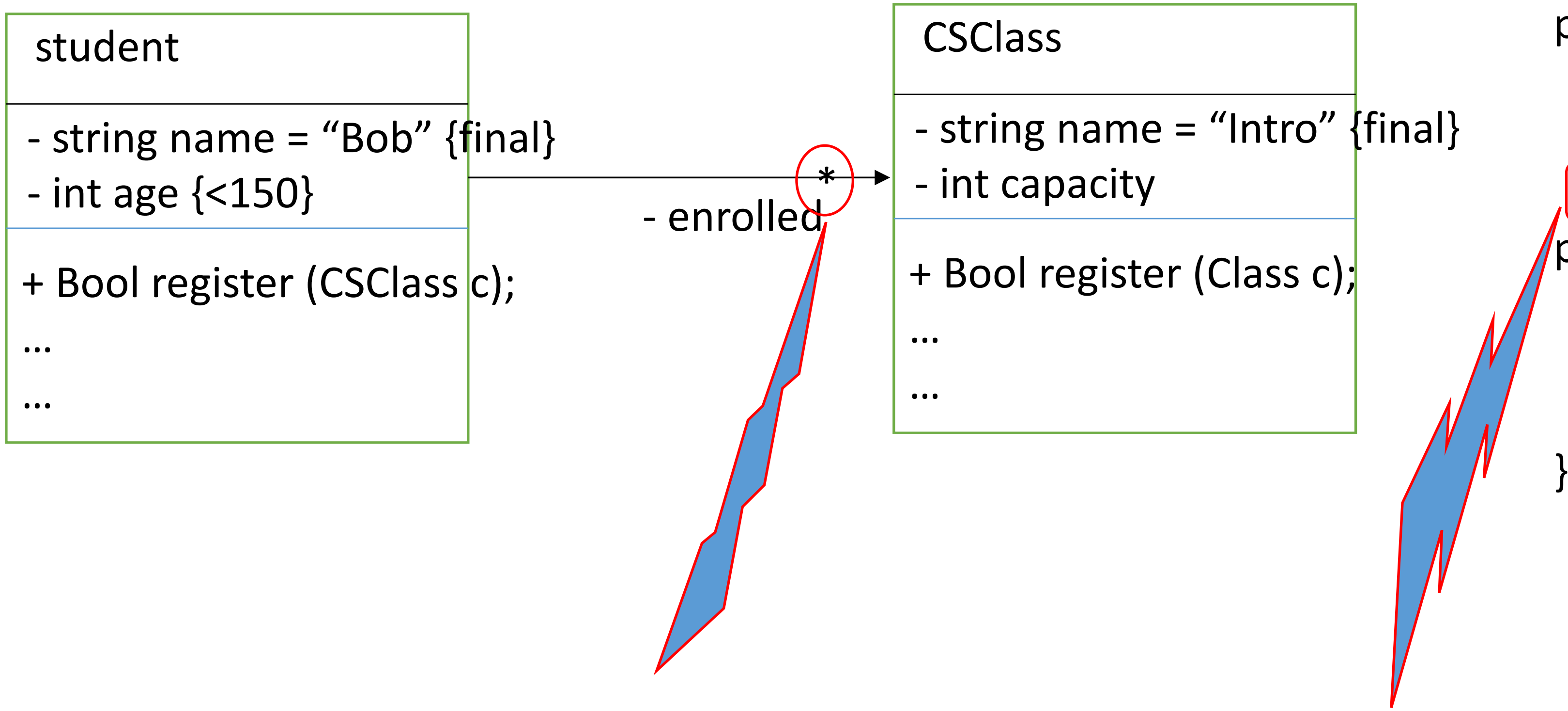
- Data+operation

Components of class diagrams

- Class name
- Class properties
  - Attributes
  - Associations (could be bi-directional) ⟶

  visibility name : type [multiplicity] = default {property-string}
- Class operations

  Visibility name (parameter list) : return-type {property-string}
- Generalization ⟶▷
  - Inheritance (subclass, super class, interface, …)
- Dependency - - - - ->
- Constraints {}



student

- string name = “Bob” {final}
- int age {<150}

+ Bool register (CSClass c);
…
…

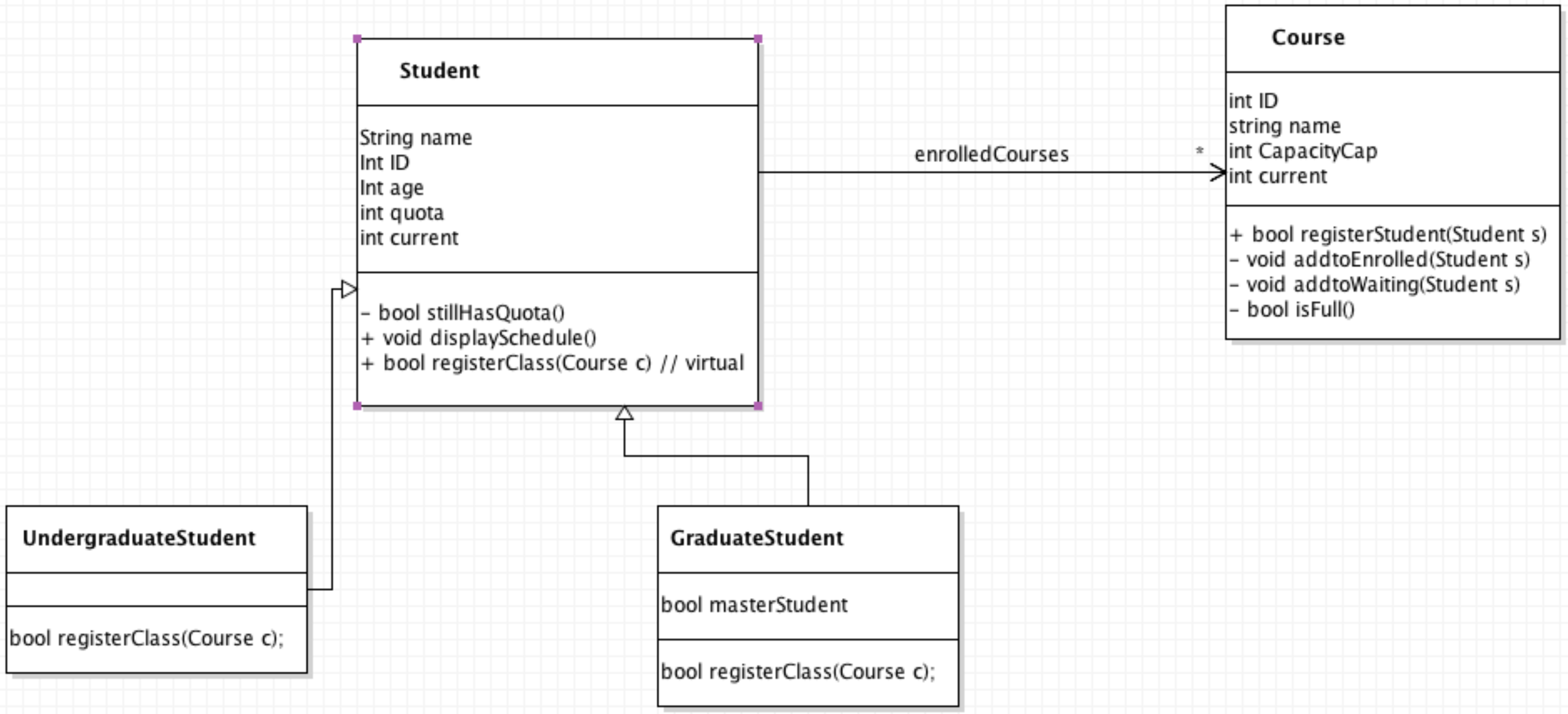
CSClass

- string name = “Intro” {final}
- int capacity

+ Bool register (Class c);
…
…

```
Class student{
private:
   final string name;
   int age;
   Set enrolledSet<CSClass>;
public:
   student (string n, int a);
   bool register (CSClass c);
   …
}
```

- * represents unknown number of CSClass property objects of a student object
- If we put a constant number, like 4, here, we should replace the “Set” data structure into Array



Student

String name
Int ID
Int age
int quota
int current

- bool stillHasQuota()
+ void displaySchedule()
+ bool registerClass(Course c) // virtual

Course

int ID
string name
int CapacityCap
int current

+ bool registerStudent(Student s)
- void addtoEnrolled(Student s)
- void addtoWaiting(Student s)
- bool isFull()

UndergraduateStudent

bool registerClass(Course c);

GraduateStudent

bool masterStudent

bool registerClass(Course c);

UndergraduateStudent and GraduateStudent are subclasses of Student, and inherit all the attributes and methods of Student.
They both re-implement the registerClass function (polymorphism), and both inherit the super-class' implementation of displaySchedule.

How to turn class diagram to code

- A private attribute → ??
- A * attribute/association → ??
- Class declaration
  - Some attributes may not map to fields

Sequence diagram

- Describes how objects collaborate/interact with each other in one scenario

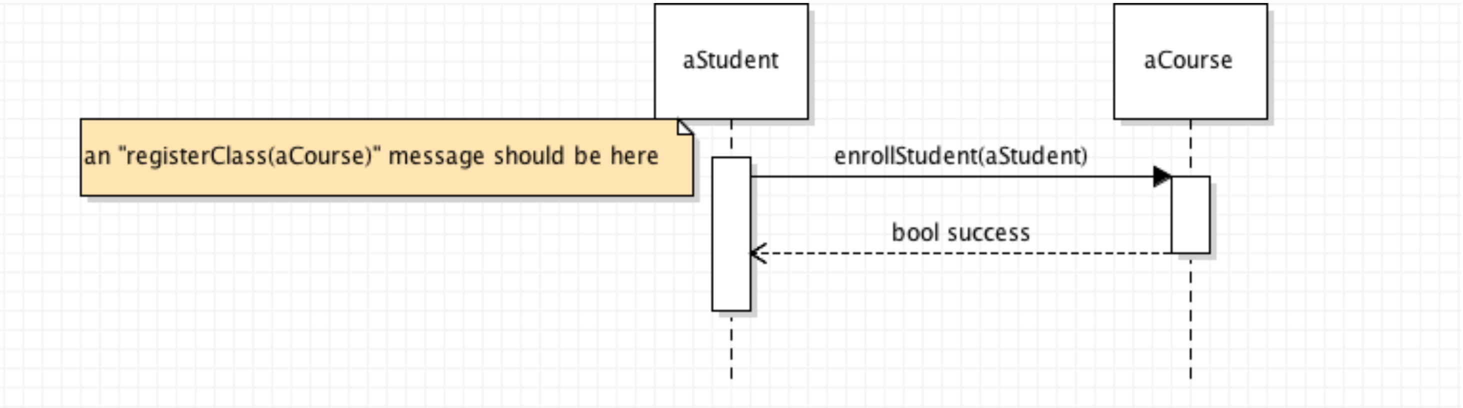
Components of sequence diagram

- Participants
- Life-line
- Activation bar
- Message
  - Regular calls, self calls

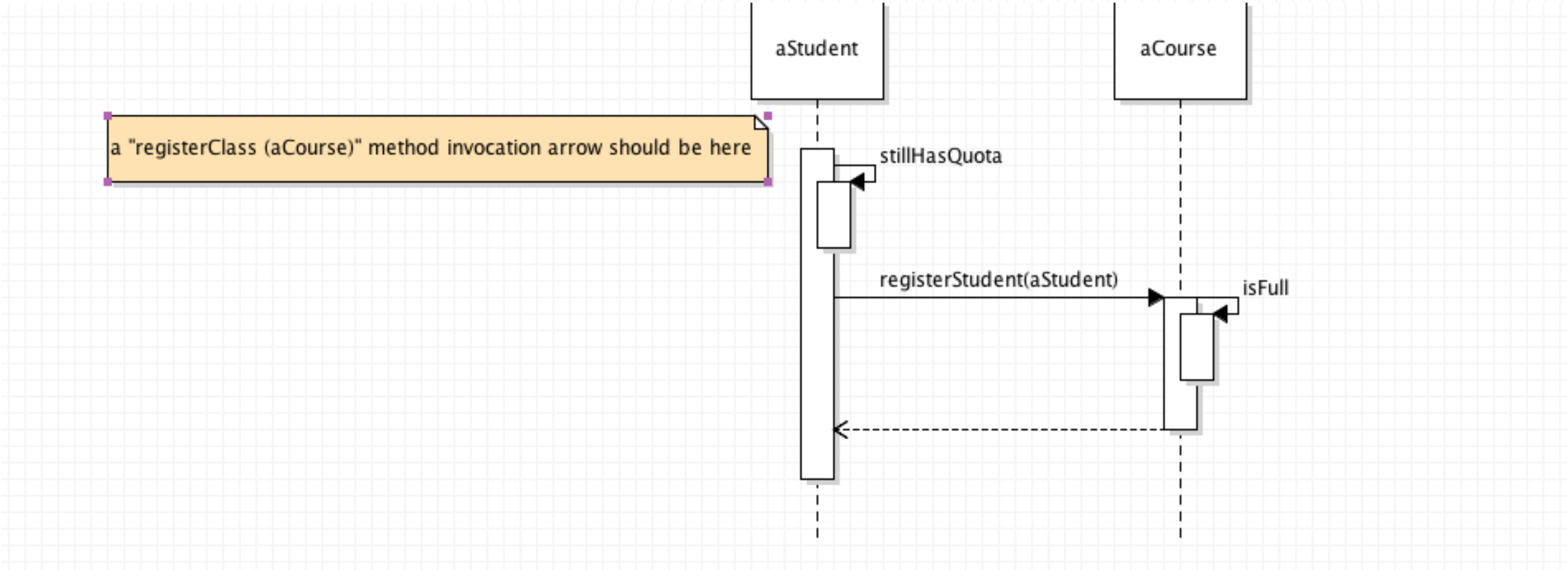
- Creating and deleting object
- Loops and conditionals
  - loop, alt, opt

http://en.wikipedia.org/wiki/Sequence_diagram

Sequence diagram example 1

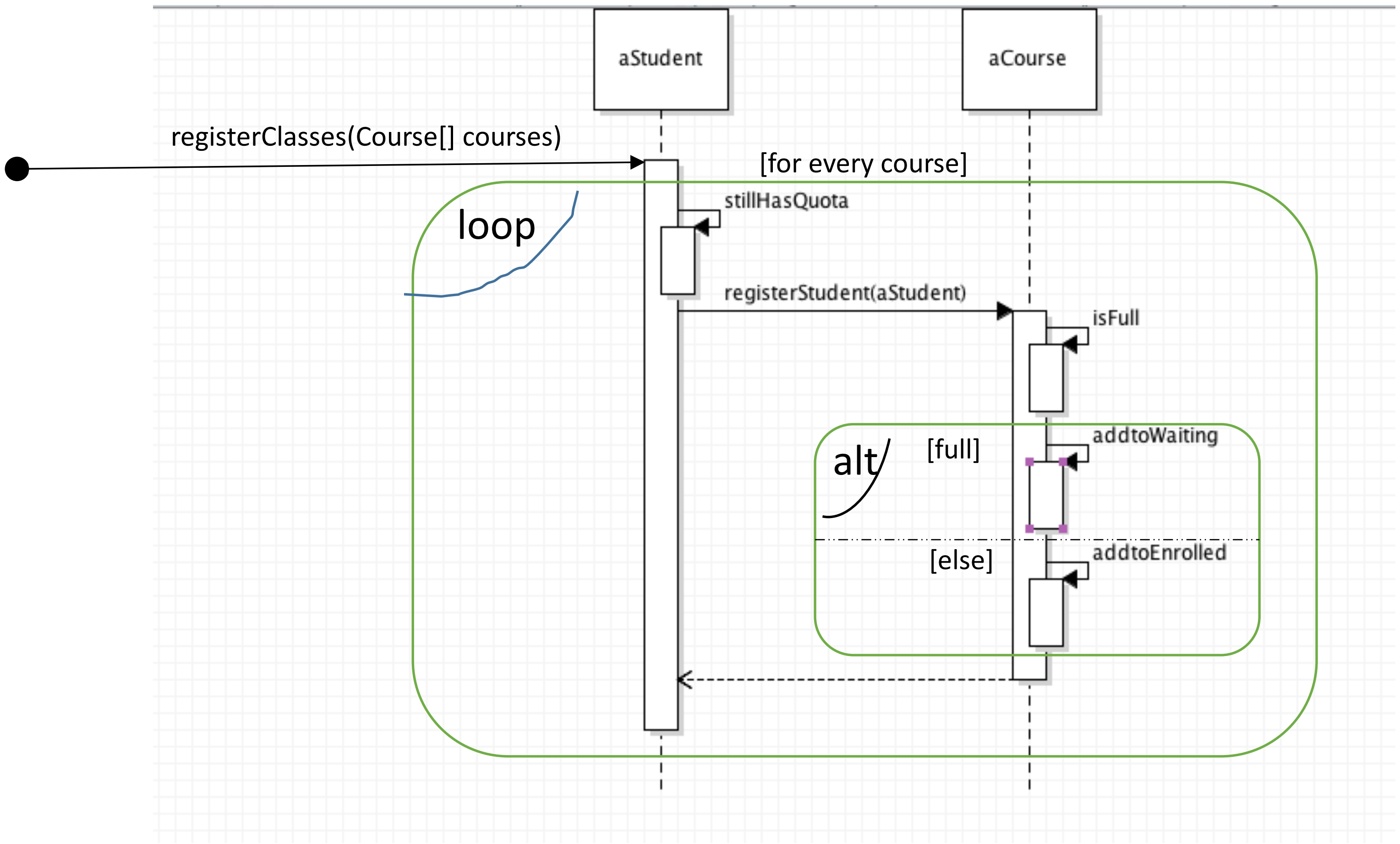


an "registerClass(aCourse)" message should be here



a "registerClass (aCourse)" method invocation arrow should be here

Sequence diagram example 2



Summary

- Class diagram
- Sequence diagram