Introduction to UML

C++ Object Oriented Programming
Pei-yih Ting
NTOU CS

Contents

Introduction to Modeling

- The models we choose have a profound influence on the solution we provide
- Every model may be expressed at different levels of abstraction
- The best models are connected to reality
- No single model is sufficient, a set of models is needed to solve any nontrivial system

Importance of Modeling

- Why do we model?
- A model is a simplification at some level of abstraction
- We build models to better understand the systems we are developing
  - To help us visualize
  - To specify structure or behavior
  - To provide template for building system
  - To document decisions we have made

Software Modeling

- Traditionally two approaches to modeling a software system
  - Algorithmically – becomes hard to focus on as the requirements change
  - Object-oriented – models more closely real world entities

UML is a visual modeling language

- "A picture is worth a thousand words." - old saying

- **United Modeling Language:**

"A language provides a vocabulary and the rules for combining words […] for the purpose of communication. A *modeling* language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. A modeling language such as the UML is thus a standard language for software blueprints."

- *from "UML user guide"*

Software Invisibility

- Brooks in his famous article 'No Silver Bullet-Essence and Accidents of Software Engineering':

"*invisibility is an inherent, not accidental, property of software*"

- The multi-dimensional nature of software does not easily lend itself to a single 2D or 3D diagrammatic form and thereby deprives us one of our most powerful conceptual tools: Our visual and spatial perception.

UML History

- UML: Unified Modeling Language
  - Grady Booch: Booch notation 1994
    - language design, focus on structural aspects esp. inheritance
  - James Rumbaugh et al.: OMT 1991
    - background in database and Entity Relation modeling
  - Evar Jacobson: OOSE 1992
    - use cases / requirements

- The Three Amigos joined in 1997
  - unified means "joint effort instead of wars"

Usages of UML

- UML is used in the course to
  - document designs
    - design patterns / frameworks
  - represent different views/aspects of design – visualize and construct designs
    - static / dynamic / deployment / modular aspects
  - provide a *next-to-precise*, common, language – specify visually
    - for the benefit of analysis, discussion, comprehension…
  - **abstraction takes precedence over precision!**
    - aim is overview and comprehension; **not** execution

Building Blocks of UML

- Things
- Relationships
- Diagrams

Things

- Structural things
  - *classes, interfaces, collaborations, use cases, active classes, components, nodes.*
- Behavioral things
  - *interactions, finite state machines.*
- Grouping things
  - *packages.*
- Annotational things
  - *notes.*

Relationships

- Dependency
- Association
- Generalization
- Realization

Diagrams

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

Structural Things

- Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical.

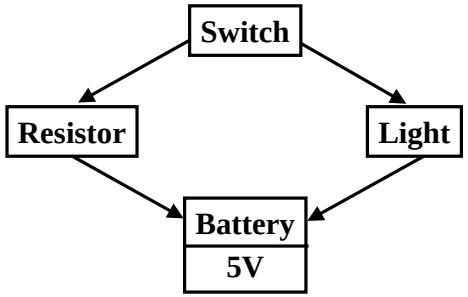
Structural Things (cont'd)

- Class

  A description of a set of objects that share the same attributes, operations, relationships, and semantics

  - Attribute
    - An attribute is a named property of a class that describes a range of values that instances of the property may hold.
  - Operation
    - An operation is the implementation of a service that can be requested from any object to affect behavior.

Class Diagram



Structure of system (objects, attributes, associations, operations)

Structural Things (cont'd)

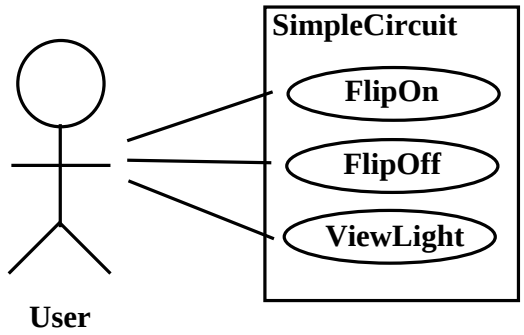
- Use case

  specifies the behavior of a system or a part of a system and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor

  - Actor

    An actor represents a coherent set of roles that users of use cases play when interacting with these use cases.

Use Case Diagram



Functionality from user's point of view

Structural Things (cont'd)

- Interface

  a collection of operations that specify a service of a class or component

- Collaboration

  A collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements.

Structural Things (cont'd)

- Active class

  An active class is a class whose objects own one or more processes or threads and therefore can initiate control activity.

- Component

  A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.

- Node

  A node is a physical element that exists at run time and represents a computational resource.

Behavioral Things

Behavioral things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space.

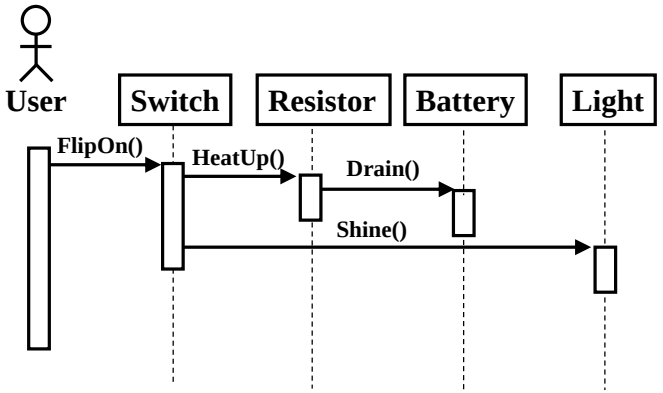
- Interaction

  An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose.

- State machine

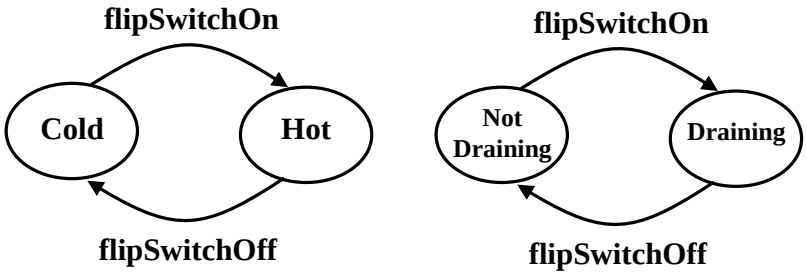
  A state machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its response to those events.

Interaction Diagram: Sequence Diagram



Messages between objects

Statechart Diagram (different objects)



(Resistor) **(Battery)**

Grouping and Annotational Things

Grouping things are the organizational parts of UML models.

- Package

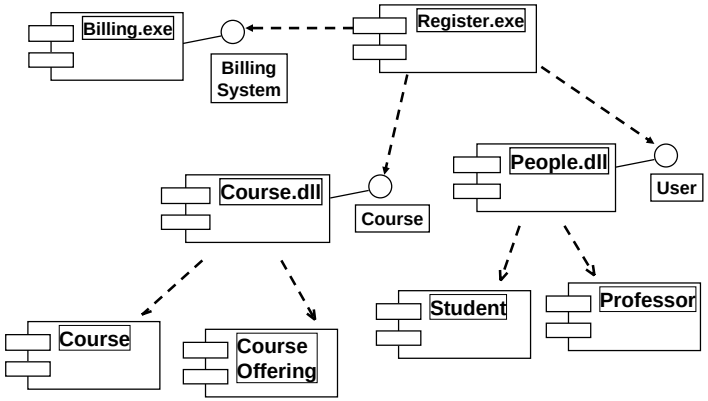
  A package is a general purpose mechanism for organizing elements into groups.

Annotational things are the explanatory parts of UML models.

- Note

  A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

Component Diagram



class packaging and dependencies

Relationships

- Dependency

  A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse. (Usually a class depends on some interfaces or abstract classes instead of another class.)

- Association

  An association is a structural relationship that specifies that objects of one thing are connected to objects of another.

Relationships (cont'd)

- Aggregation

  An aggregation is a special form of association that specifies a whole-part relationship between the aggregate (the whole) and a component (the part).

- Generalization

  A generalization is a relationship between a general thing and a more specific kind of that thing. Sometimes it is called an "is-a-kind-of" relationship.

- Realization

  A realization is a semantic relationship between classifiers, wherein, one classifier specifies a contract (interface) that another classifier promises to carry out.

Diagrams

- Class diagram

  A class diagram shows a set of classes, interfaces, and collaborations and their relationships.

- Object diagram

  An object diagram shows a set of objects and their relationships.

- Use case diagram

  A use case diagram shows a set of use cases and actors and their relationships. A Use case is a literary form of describing user goals, as a set of scenarios. A *scenario* is a sequence of steps describing interaction between a user and a system.

Diagrams (cont'd)

- Sequence diagram

  A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages.

- Collaboration diagram

  A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

- Statechart diagram

  A statechart diagram shows a state machine, consisting of states, transitions, events, and activities.

Diagrams (cont'd)

- Activity diagram

  An activity diagram is a special kind of a statechart diagram that shows the flow from activity to activity within a system.

- Component diagram

  A component diagram shows the organization and dependencies among a set of components.

- Deployment diagram

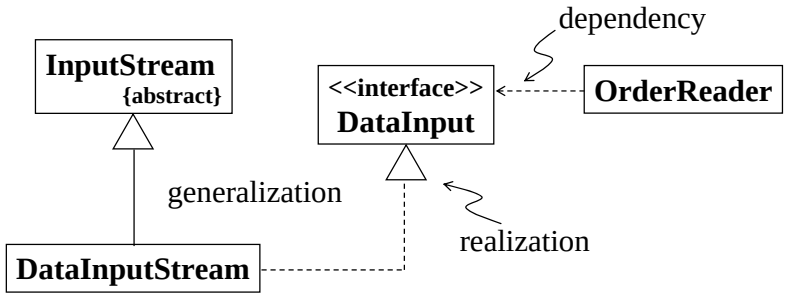
  A deployment diagram shows the configuration of runtime processing nodes and the components that live on them.

Class Diagrams

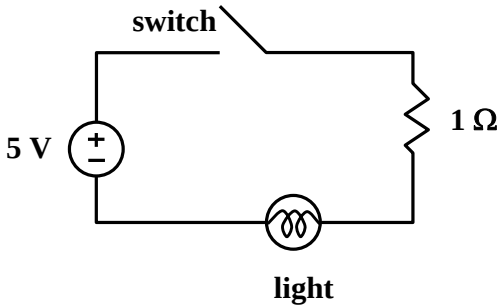
- Same diagram – different perspectives
  - Conceptual
    - focus: domain modeling
    - "software independent" – no software specific parts
  - Specification
    - focus: responsibilities and contracts/interfaces
    - we are talking **software** i.e. we include software related aspects: design patterns, frameworks, etc.
  - Implementation
    - close mapping to actual source code

Contracts and Responsibility

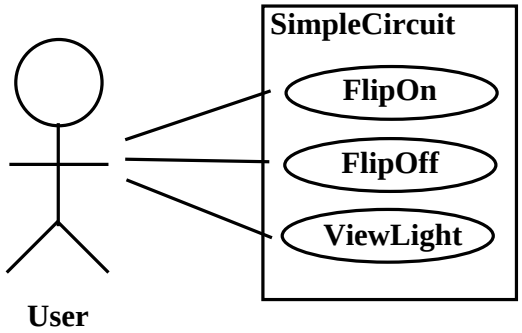
- **Classes are too close to implementation**.
- Instead think in terms of **contracts** and **responsibility!**
- UML (and java) approximation is *interfaces*



A Simple Problem

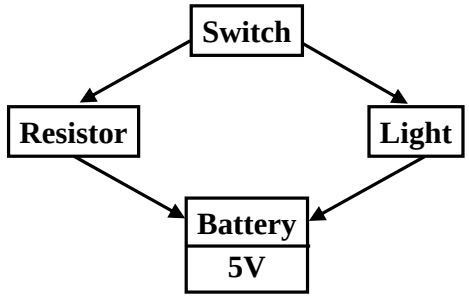


Use Case Diagram



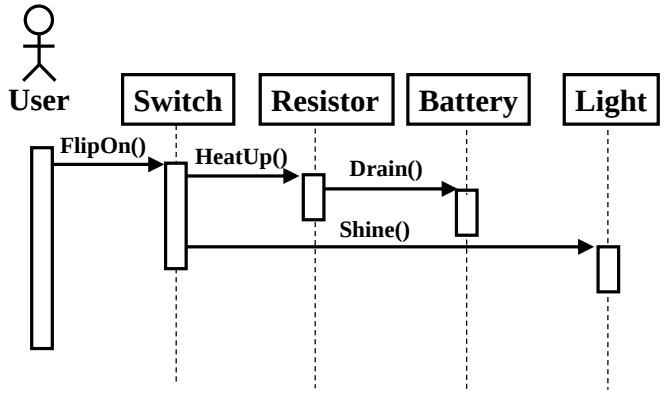
Functionality from user's point of view

Class Diagram



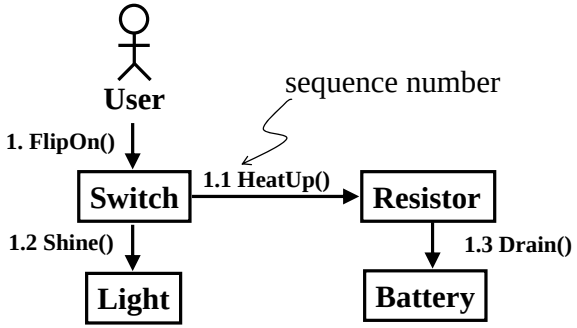
Structure of system (objects, attributes, associations, operations)

Interaction Diagram: Sequence Diagram



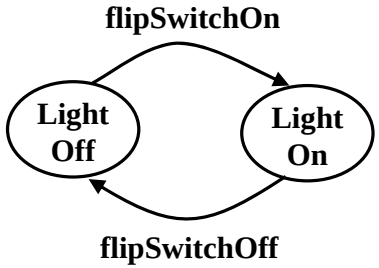
Messages between objects

Interaction Diagram: Collaboration Diagram



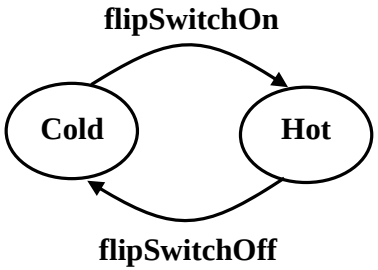
Alternative to sequence diagram,
More compact, but harder to interpret

Statechart Diagram

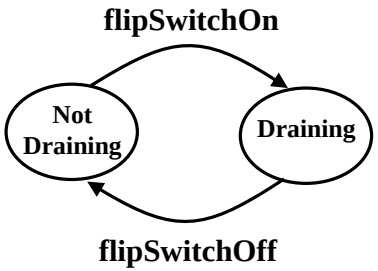


Transitions between states of an object
(Extension of Finite State Machine (FSM) model)

Statechart Diagram (different objects)

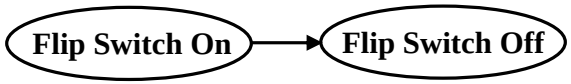


(Resistor)



(Battery)

Activity Diagram



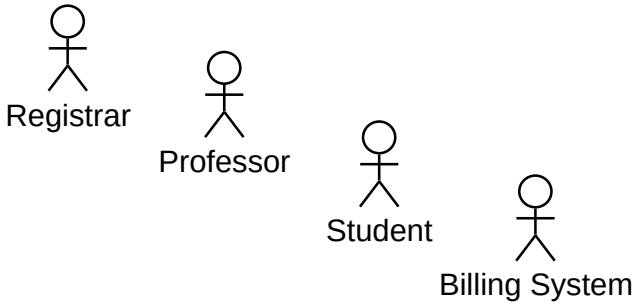
- Actions are states
- shows the flow from activity to activity within a system

More Elaborated Example

- The ESU University wants to computerize their registration system
  - The Registrar sets up the curriculum for a semester
    - One course may have multiple course offerings
  - Students select 4 primary courses and 2 alternate courses
  - Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
  - Students may use the system to add/drop courses for a period of time after registration
  - Professors use the system to receive their course offering rosters
  - Users of the registration system are assigned passwords which are used at logon validation

Actors

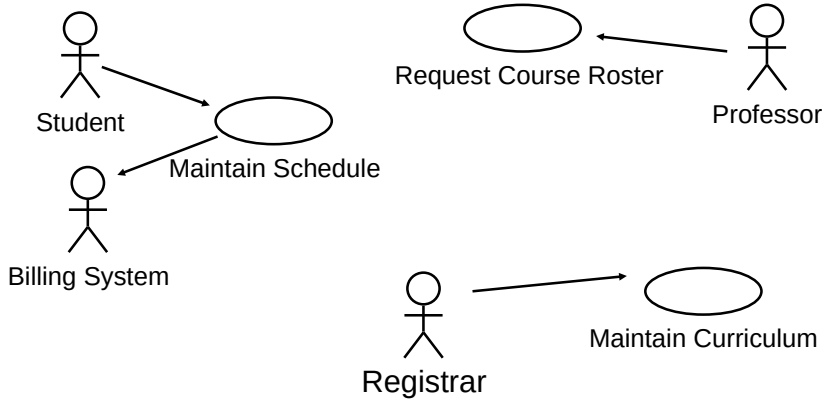
- An actor is someone or some thing that must interact with the system under development



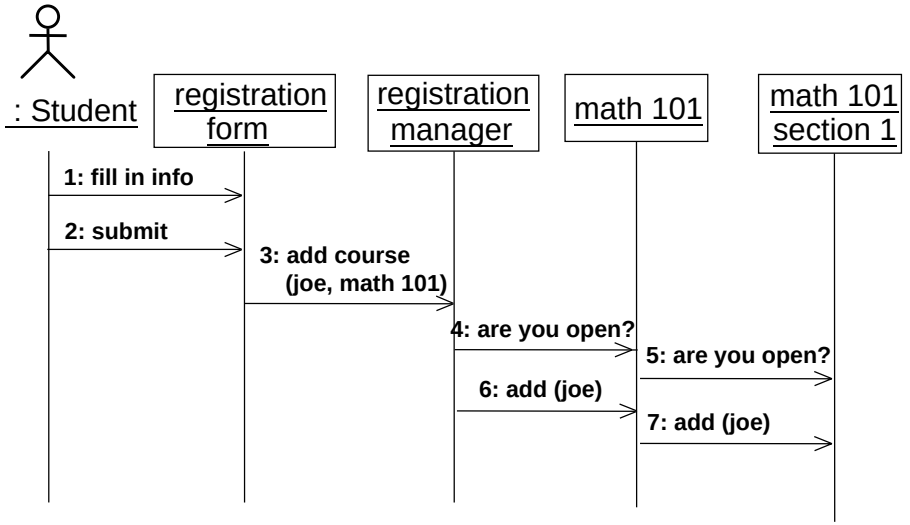
Use Cases

- use case is a pattern of behavior the system exhibits
  - Each use case is a sequence of related transactions performed by an actor and the system in a dialogue
- Actors are examined to determine their needs
  - Registrar -- maintain the curriculum
  - Professor -- request roster
  - Student -- maintain schedule
  - Billing System -- receive billing information from registration

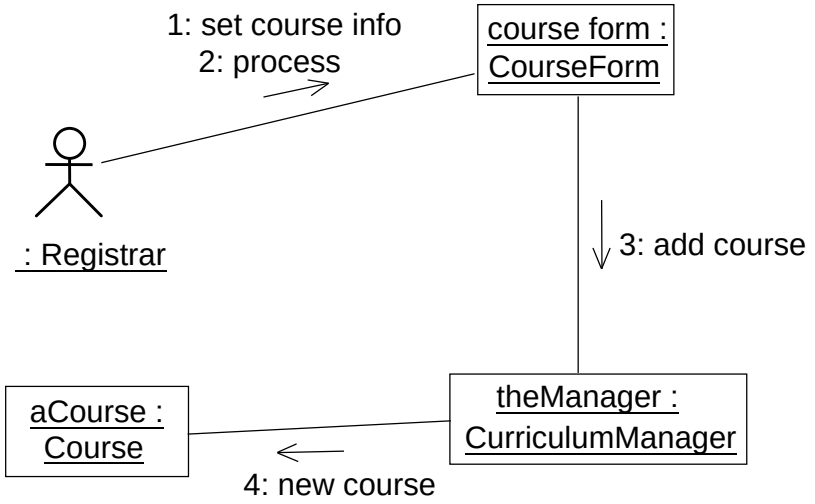
Use Case Diagram



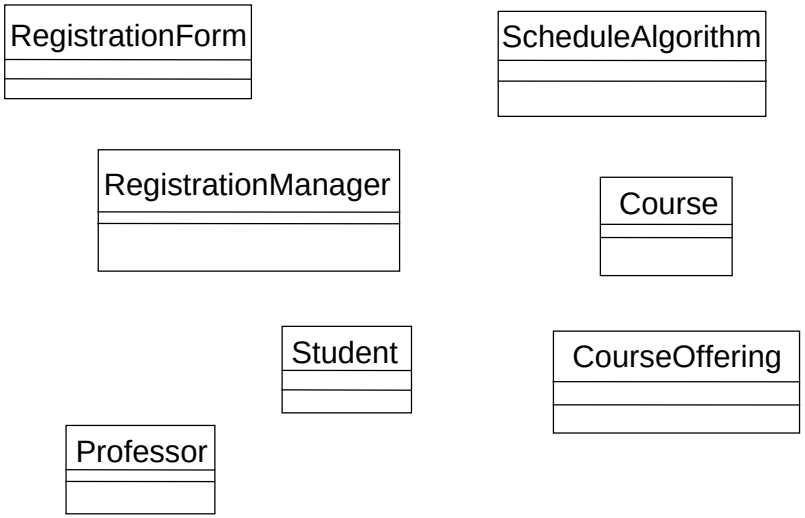
Sequence Diagram



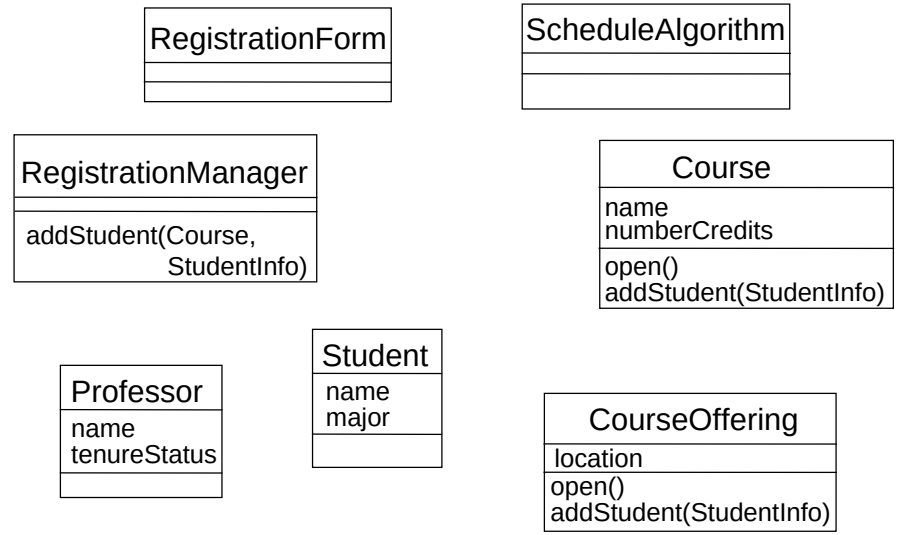
Collaboration Diagram



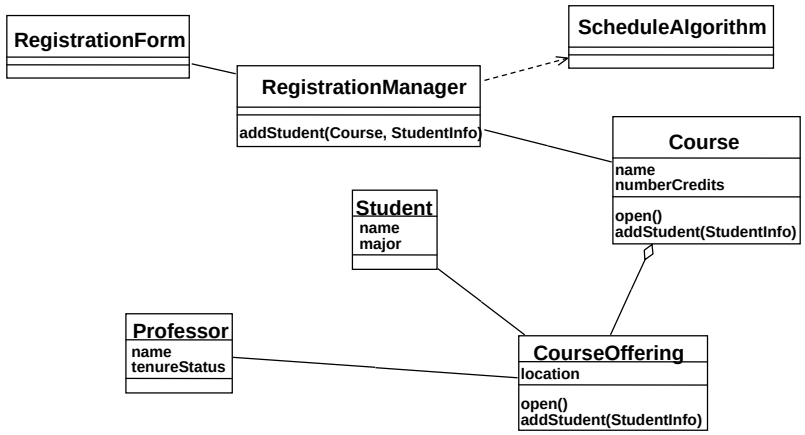
Classes



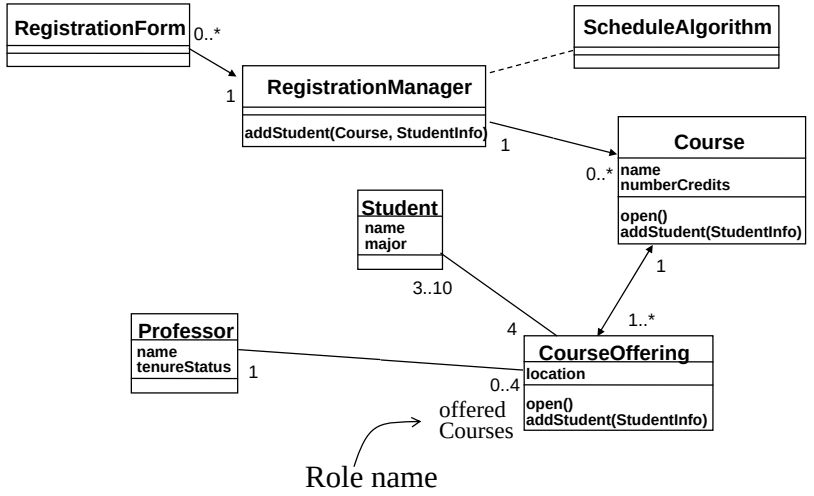
Classes: Attributes and Operations



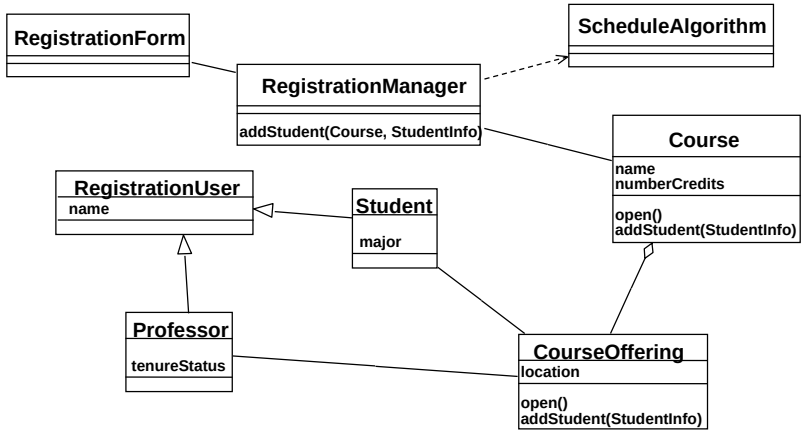
Relationships



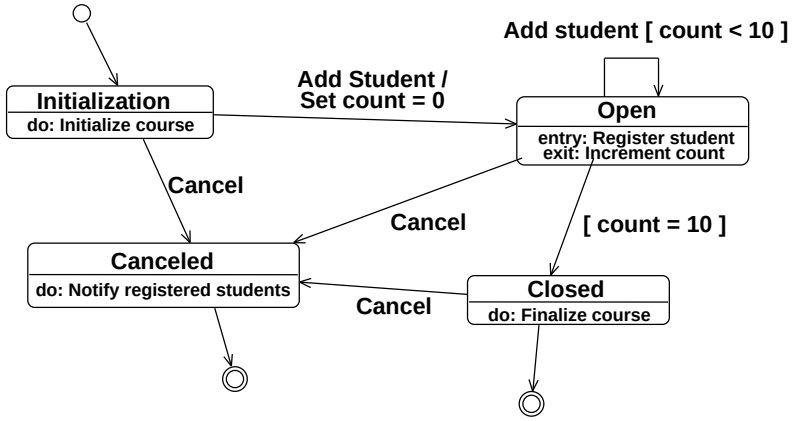
Multiplicity and Navigation



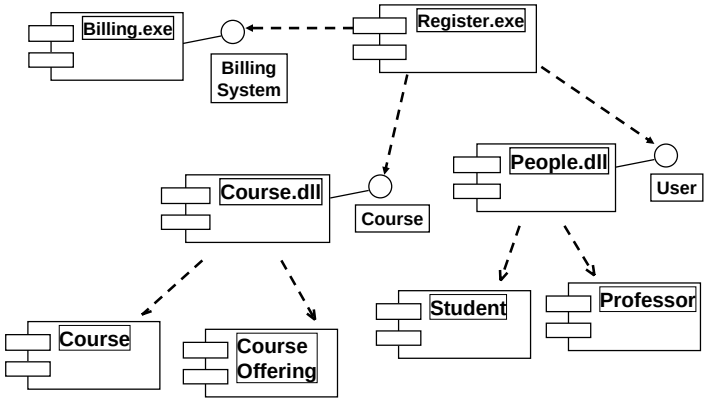
Inheritance



State Transition Diagram

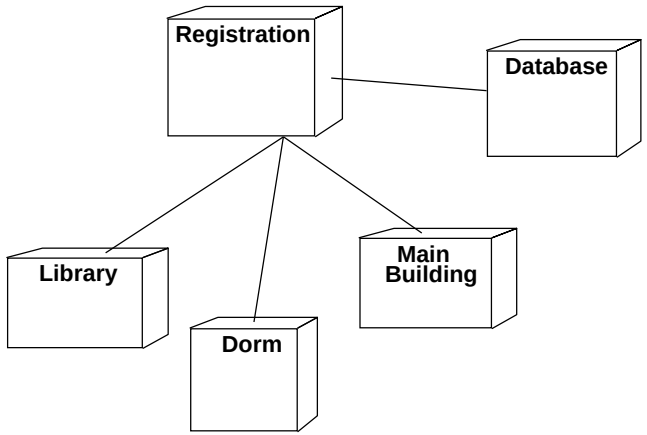


Component Diagram



class packaging and dependencies

Deployment Diagram

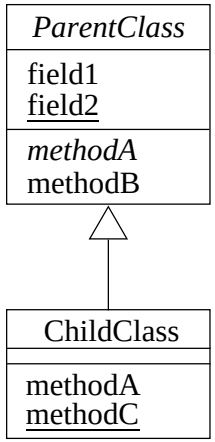


physical setup

More Graphical Notations

- Class Diagram: abstract, static

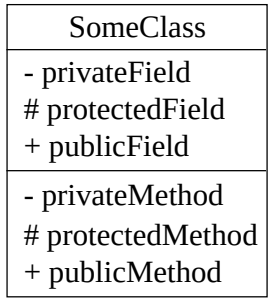
```
abstract class ParentClass {
   int field;
   static char field2;
   abstract void methodA();
   double methodB() {
      …
   }
}

class ChildClass extends ParentClass {
   void methodA() {
      …
   }
   static void methodC() {
      …
   }
}
```



More Graphical Notations

- Access Control

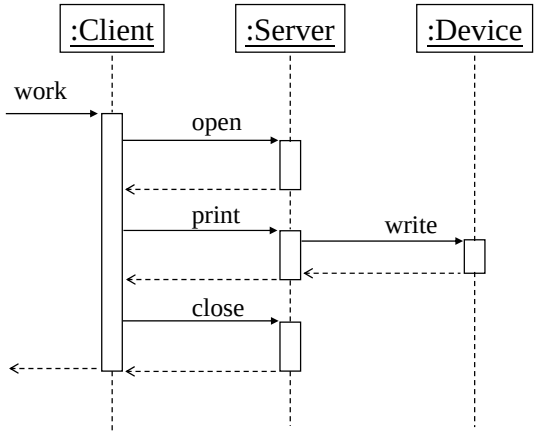
```
class SomeClass {
   private int privateField;
   protected int protectedField;
   public int publicField;
   private void privateMethod() {
   }
   protected void protectedMethod() {
   }
   public void publicMethod() {
   }
}
```



More Graphical Notations

- Sequence diagram: message, return, lifeline, activation

```
class Server {
   Device device;
   void open() {
      …
   }
   void print(String s) {
      device.write(s);
      …
   }
   void close() {
      …
   }
   …
}
class Client {
   Server server;
   void work() {
      server.open();
      server.print("Hello");
      server.close();
   }
   …
}
```




```
class Device {
   void write(String s) {
      …
   }
}
```


References

a. UML Distilled, Applying the Standard Object Modeling Language, Martin Fowler, (UML 精華:應用標準物件模式語言, 許銀雄譯, AW/松崗)

b. 物件導向系統分析與設計, 使用 UML 與 C++, 周斯畏編著, 全華, 92/05, 九十年度非同步遠距教學

c. UML 理論與實作 --- 個案討論與經驗分享, 張裕益著, 博碩, 91/02

d. UML 使用手冊, 張裕益譯, 博碩, 90/10