SHARE
Technology • Connections • Results

8359
More OO Design and UML

Charlote Sutton
Choice Hotels International
charlote_sutton@choicehotels.com

Objectives

- Basics of UML formal notation – Class Diagram basics
- See how an OO design is described through UML diagrams
- Understand OO concepts through UML diagrams
  - Interfaces
  - Inheritance

UML – Formal Notation

- Unified Modeling Language
  - Standard for modeling established by the Object Management Group (OMG)
  - Helps you specify, visualize, and document models of software systems.
  - Twelve standard diagram types, in three categories
    - Four types represent static application structure
    - Five types represent dynamic behavior
    - Three types represent organization of application modules

UML – Formal Notation

- **Structural Diagrams** include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
- **Behavior Diagrams** include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.
- **Model Management Diagrams** include Packages, Subsystems, and Models.

www.omg.org/gettingstarted/what_is_uml.htm

Class Diagram

- *By definition, a class diagram is a diagram showing a collection of classes and interfaces, along with the collaborations and relationships among classes and interfaces.*
  - A class diagram is a pictorial representation of the detailed system design.
  - A class diagram is a static view of a system.
  - The structure of a system is represented using class diagrams.

Class Diagram

- A class diagram is similar to a family tree. To build one, you would:

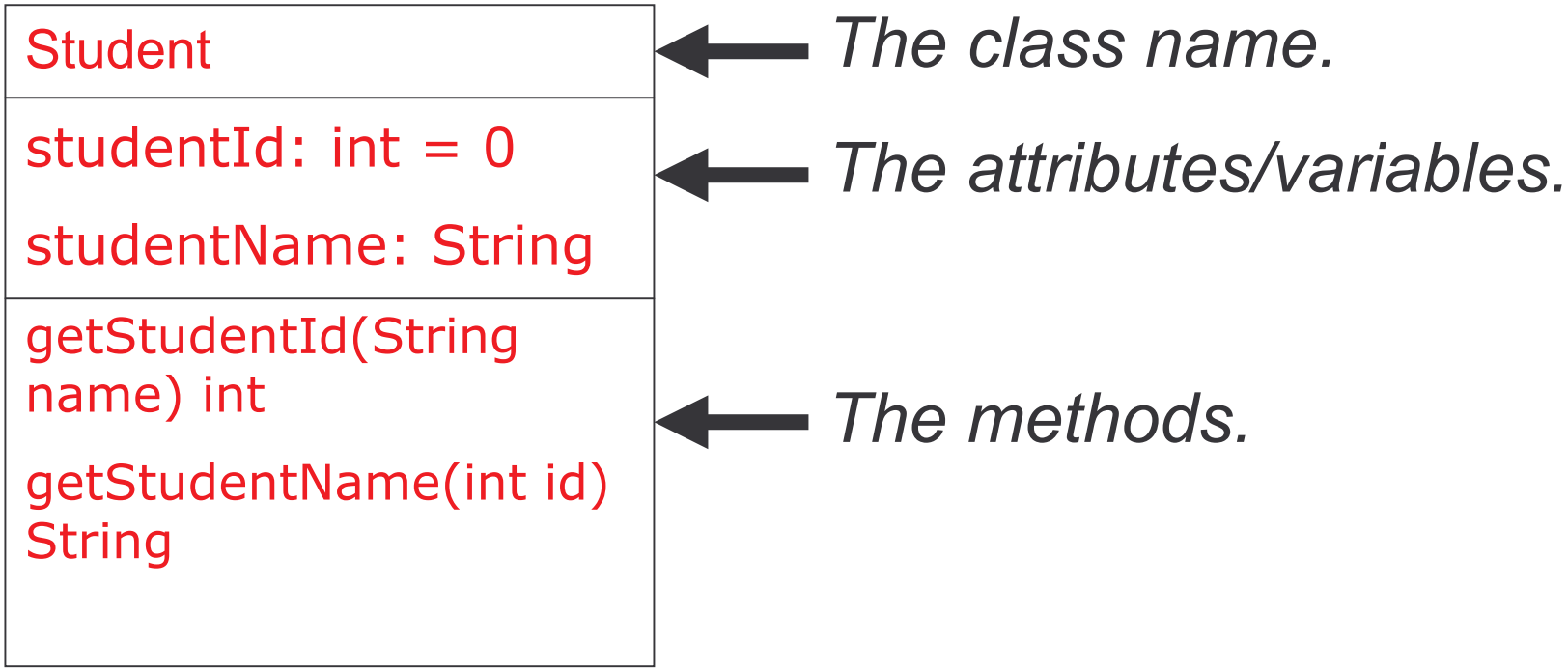
Step 1 - Identify the main members of the family and identify each member's characteristics.

Step 2 - Find relations among family members.

Step 3 - Decide the inheritance of personal traits.

Class Diagram

Step 1 - Identify the main members of the family and identify each member's characteristics.



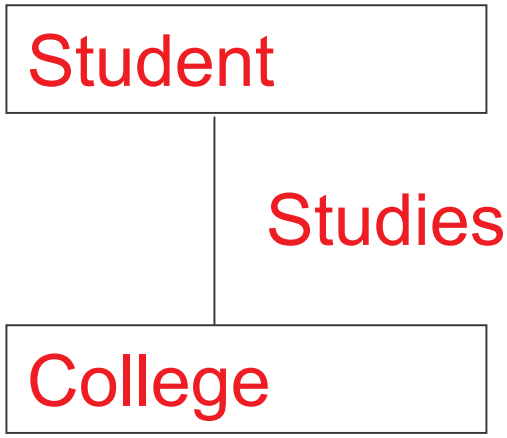
Class Diagram

Step 2 - Find relations among family members.

- Different types of relationships.
  - Association
  - Aggregation
  - Realization

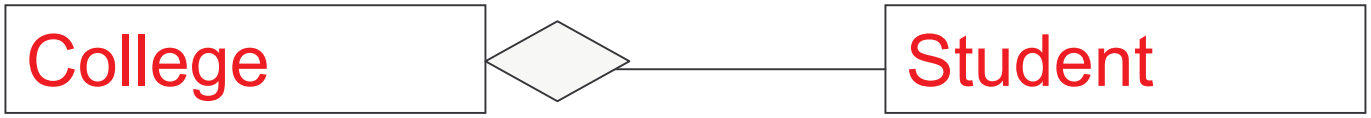
Class Diagram

- Association - When two classes are connected to each other in any way, an association relation is established. For example: A "student studies in a college."



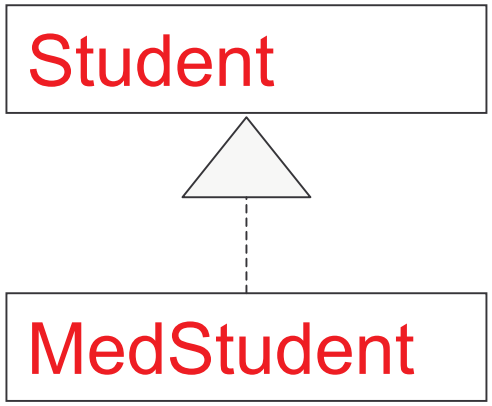
Class Diagram

- Aggregation - When a class is formed as a collection of other classes, it is called an aggregation relationship between these classes. It is also called a "**has a**" relationship. For example: A “college has students."



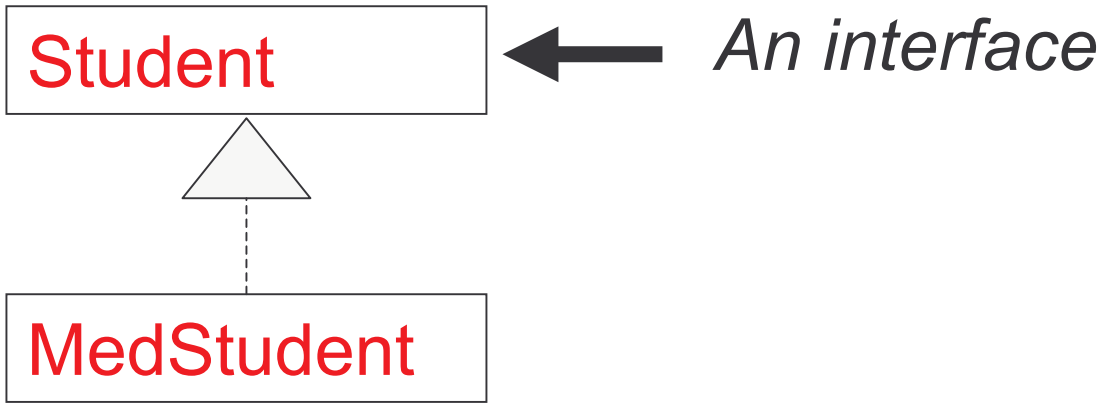
Class Diagram

- Realization - In a realization relationship, one entity defines a set of functionalities as a contract and the other entity "realizes" the contract by implementing the functionality defined in the contract. For example: A “medical student must register for classes.”



Interfaces – A Closer Look

- An interface is a variation of a class.
- An interface provides only a definition of functionality.
- A separate class actually implements that functionality.
- Use an interface when you want programmers to be able to use only the specific class and not the general one.



Interfaces – A Closer Look

- Declaring an interface:

```
public interface Student {
    public abstract void checkCredits();
}
```

- Compiler won't let you instantiate an interface.

```
Student s;
S = new Student;
```

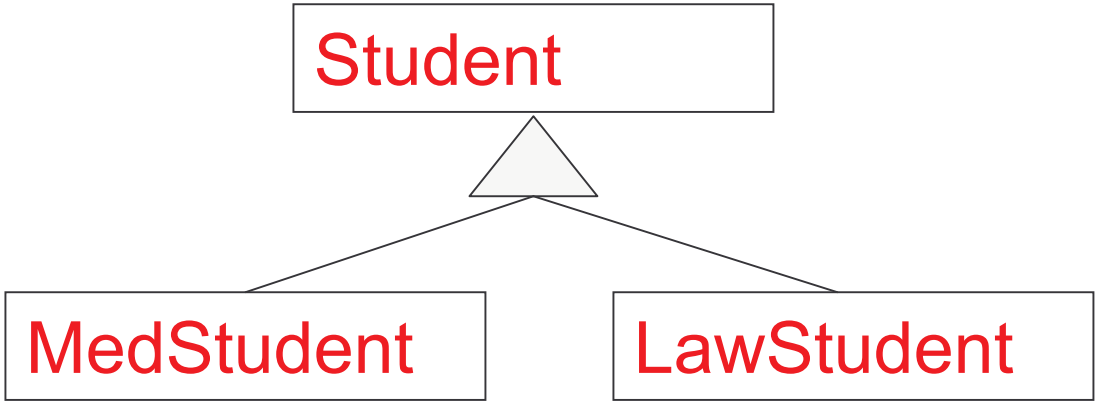
- Using an interface.

```
Class MedStudent extends Student {
    public void checkCredits();
}
```

Class Diagram

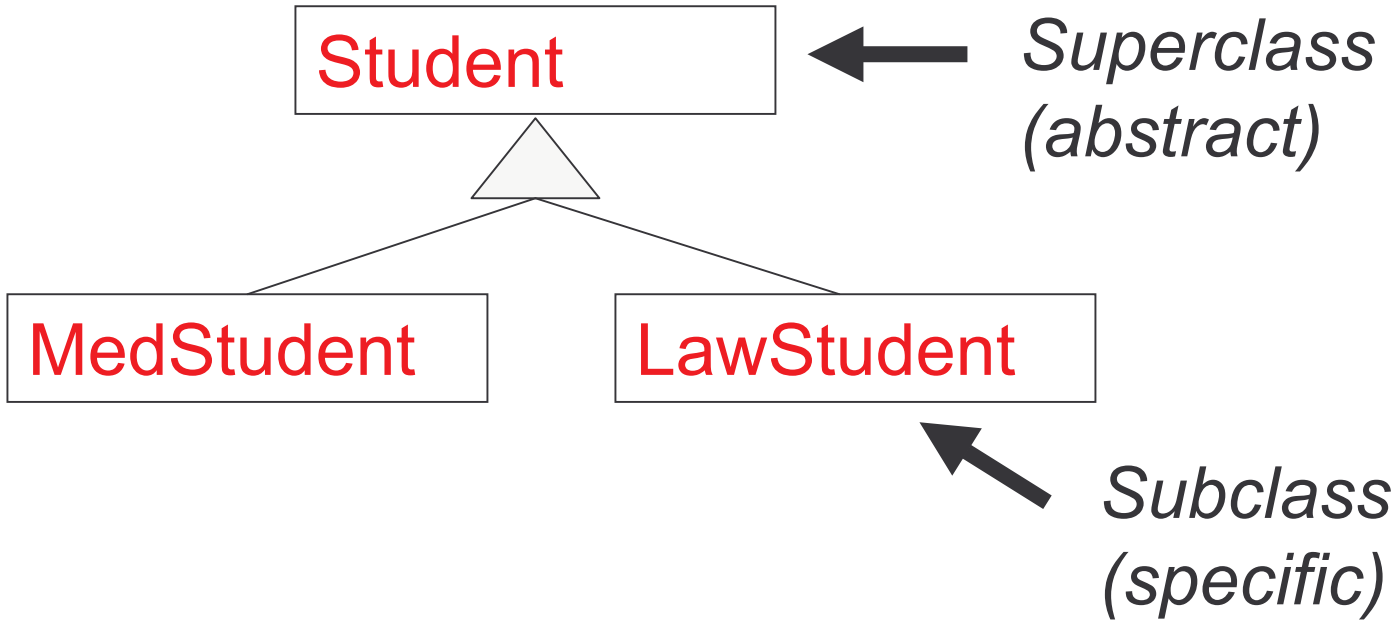
Step 3 - Decide the inheritance of personal traits.

- Inheritance/Generalization -Also called an "**is a**" relationship, because the child class **is a** type of the parent class. For example: A “medical student is a student.”



Inheritance – A Closer Look

- Put common code in a class called a superclass. This is the more abstract class.
- The more specific class is called a subclass.
- A subclass extends the superclass.



Inheritance – A Closer Look

- When abstraction and inheritance is used correctly, other programmers will be able to add new 'kinds' of classes to the program at any time. Some rules to keep in mind:
  - DO - when one class is a more specific type of a superclass.
  - DO – when you have behavior that should be shared among multiple classes of the same general type.
  - DON'T – just so you can reuse code from another class if the relationship violates either of the two rules above.
  - DON'T – if the subclass and superclass do not pass the IS-A test

Inheritance – A Closer Look

Five steps to inheritance:

1. Look for objects that have common attributes and behaviors.

2. Design a class to represent common behaviors.

3. Decide if a subclass needs specific behaviors.

4. Look for more opportunities to use abstraction by finding two or more subclasses that might have common behavior.

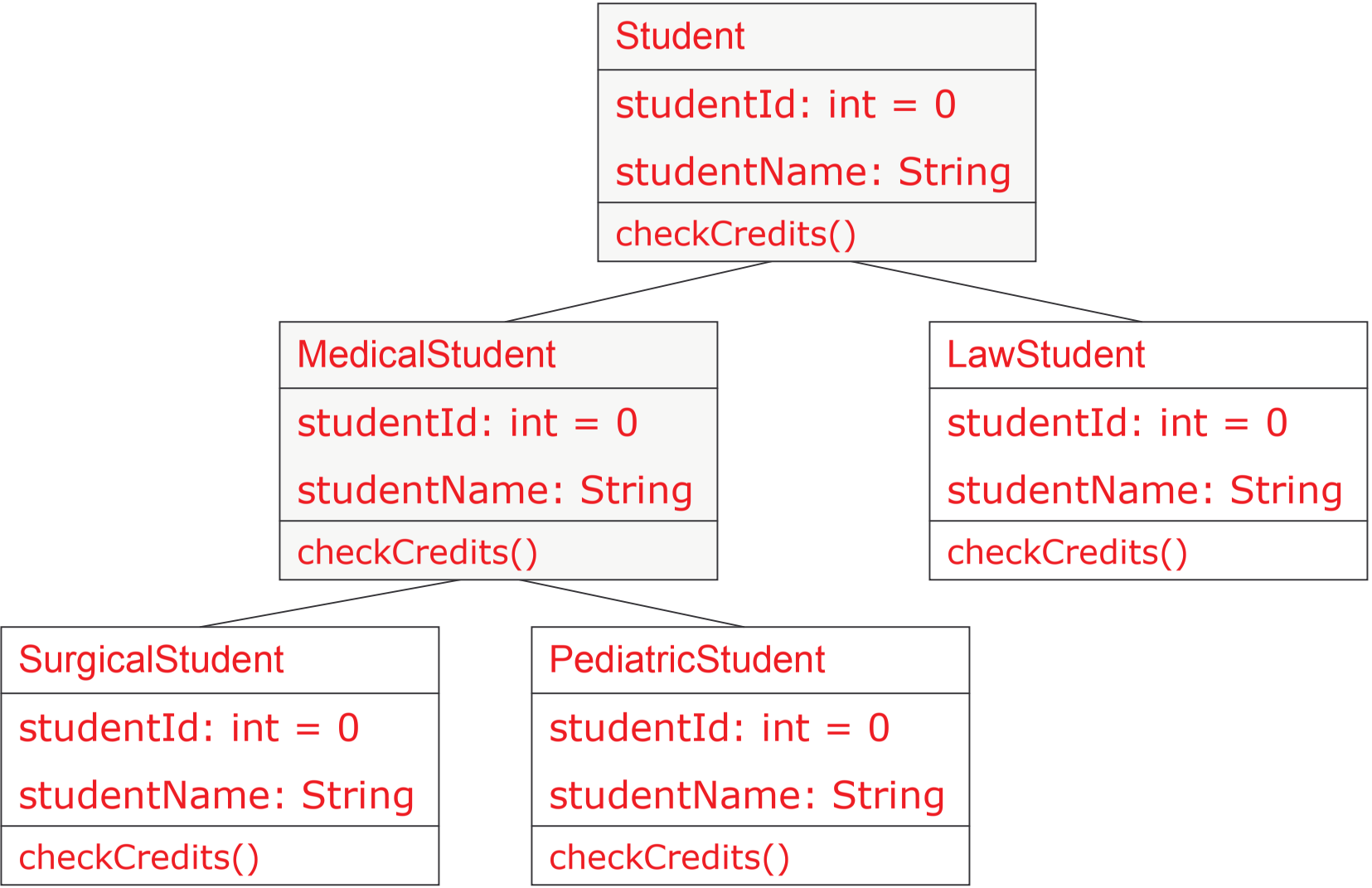
5. Finish the class hierarchy.

Inheritance – A Closer Look

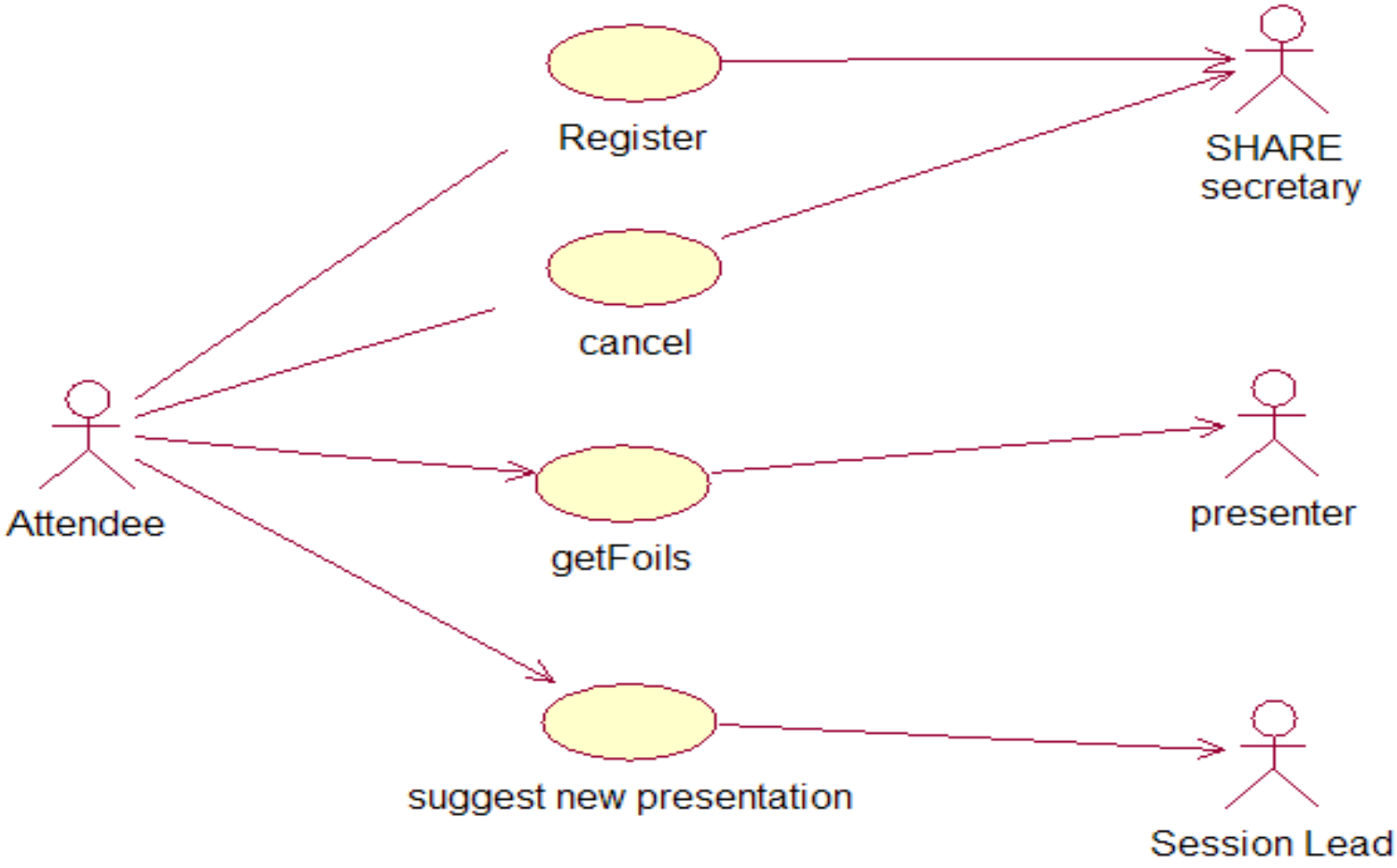
Overriding methods –

- When you design superclasses and subclasses, there may be times that the subclass will have the same method as the superclass. This is called overriding.
- In this situation, the method for the lowest class (the most specific class) gets called.

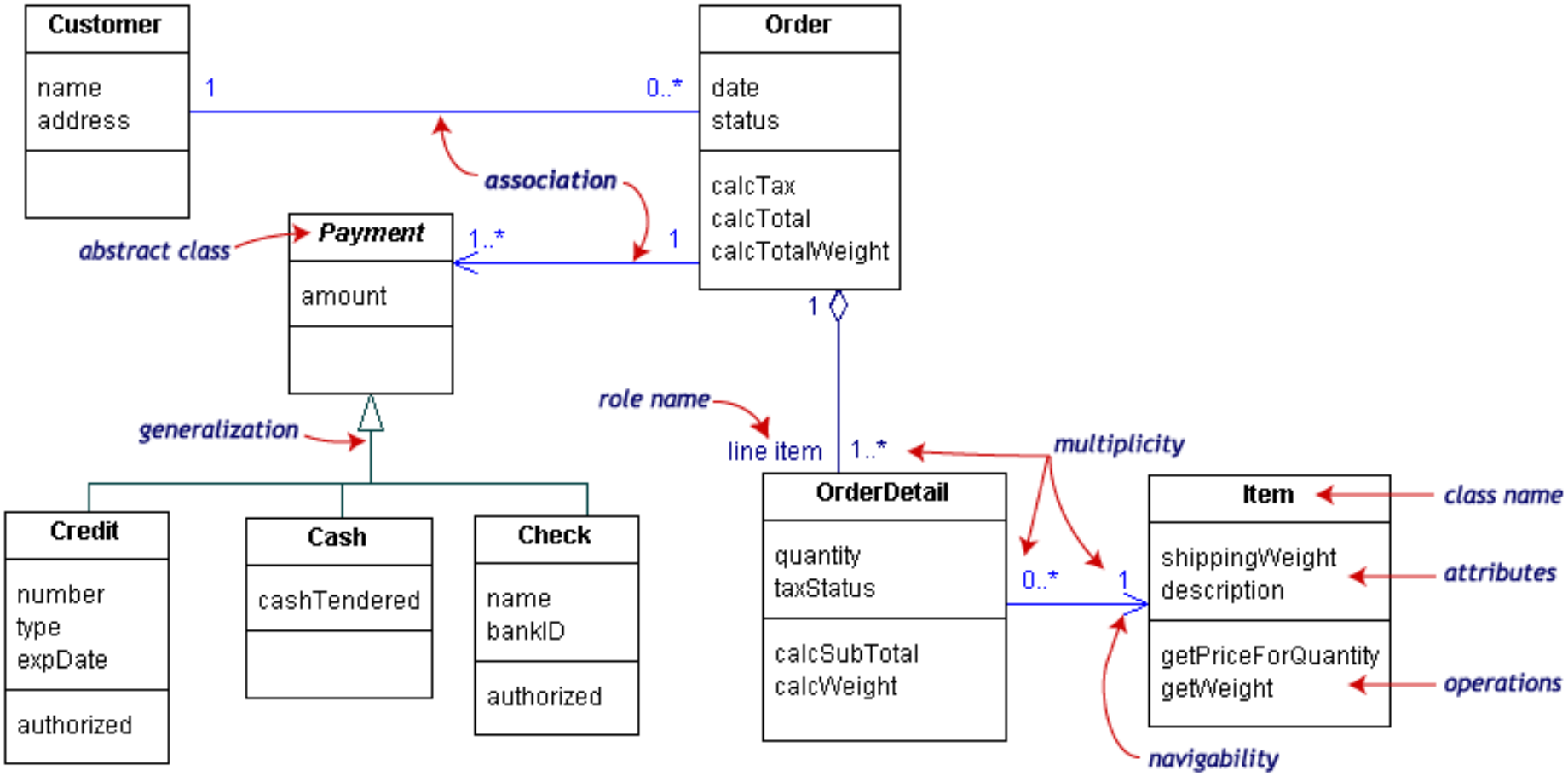
Inheritance – A Closer Look



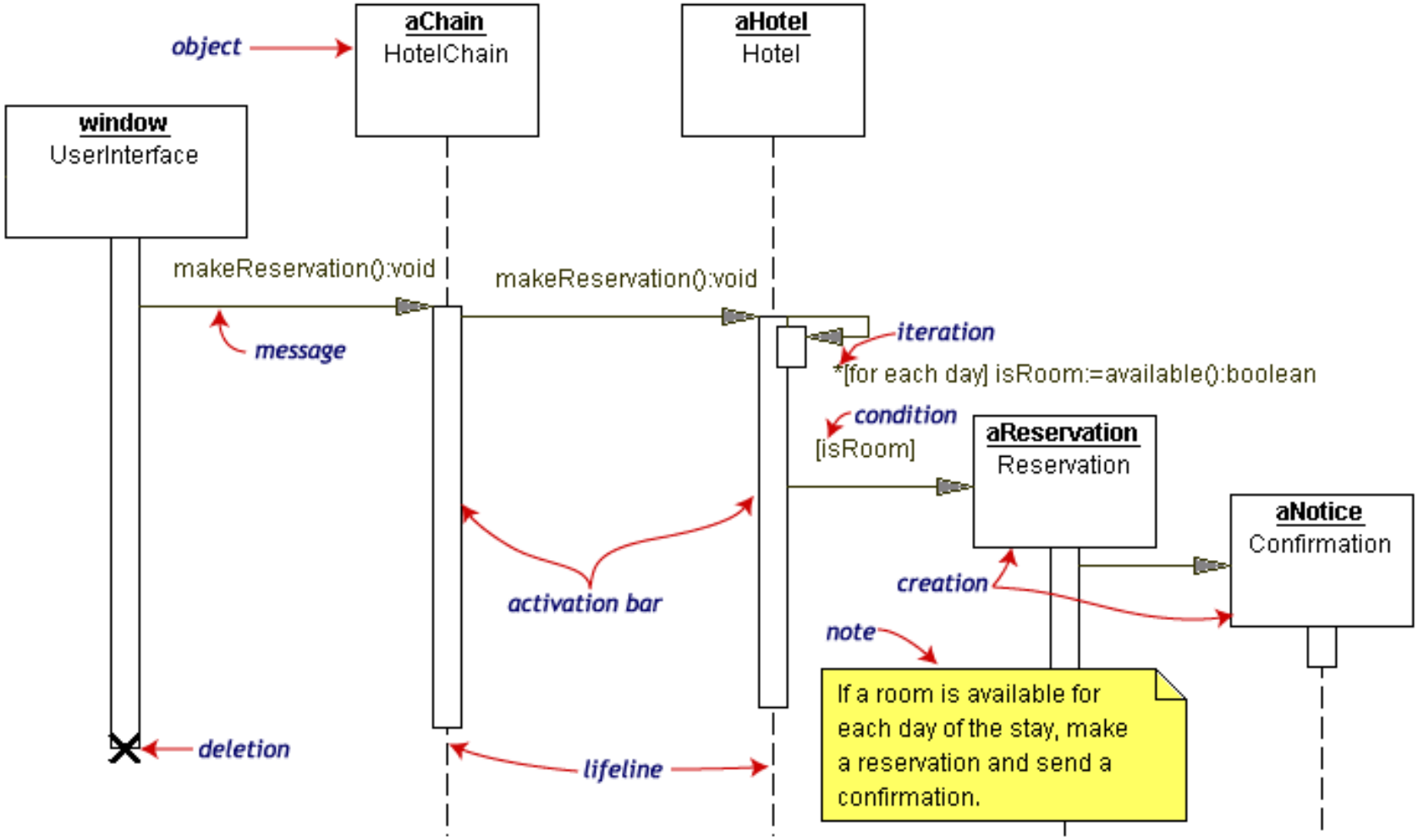
Use Case Diagram



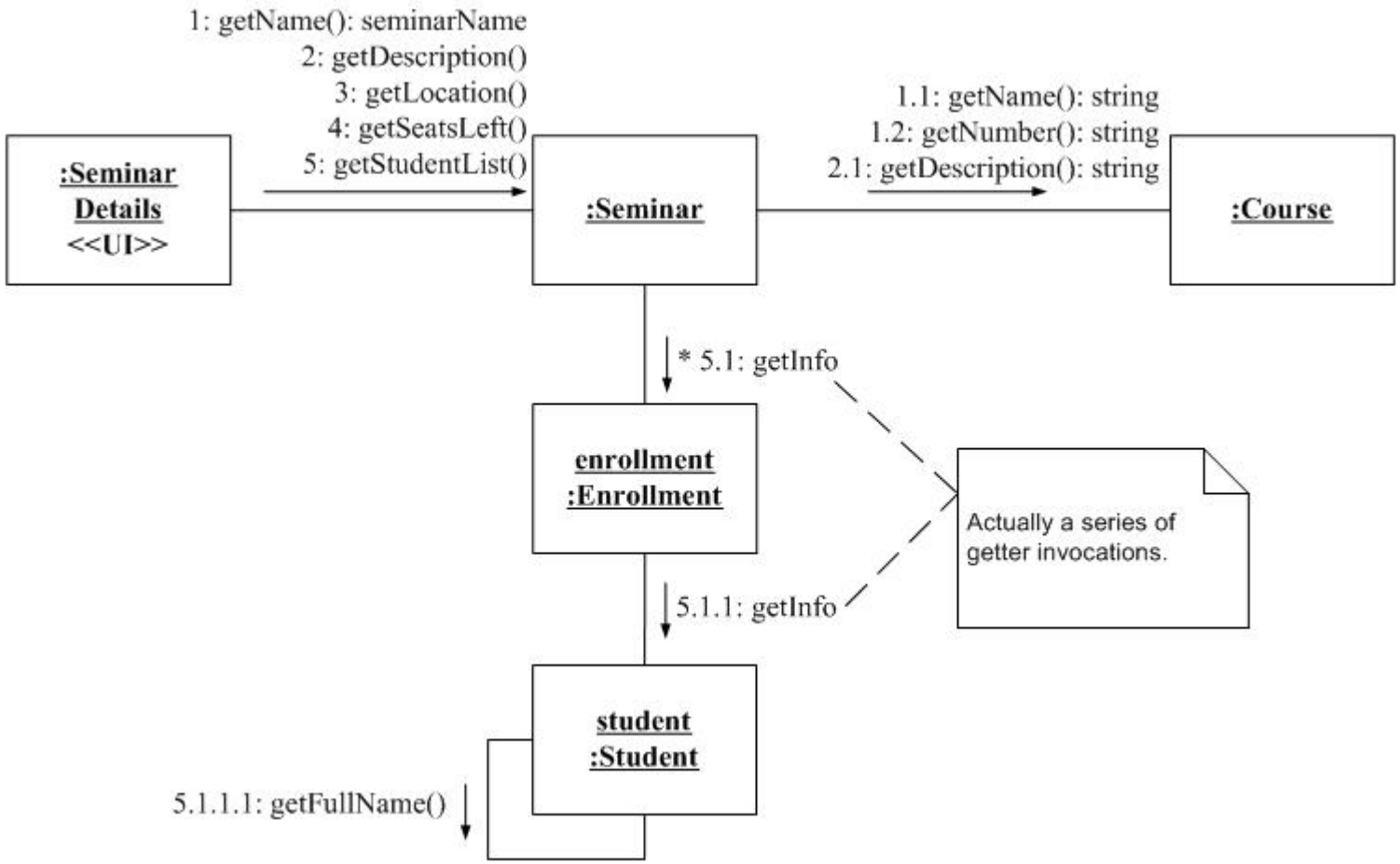
Class Diagram



Sequence Diagram



Collaboration Diagram



Tools and tool functionality

- UML Development Environments
  - Rational Rose http://www.rational.com
  - Jude http://objectclub.esm.co.jp/Jude
  - Borland Together ControlCenter http://www.pugh.co.uk/Products/borland/Together%20ControlCentre.htm
  - Poseiden for UML http://www.gentleware.com
  - ArgoUML http://argouml.tigris.org/

- Tool functionality
  - Model checking
  - Code generation
  - Reverse engineering