OBJECT-ORIENTED LANGUAGE AND THEORY

14. REVIEW

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

1

Content

1. Knowledge
2. Exam
3. Exam Organization

2

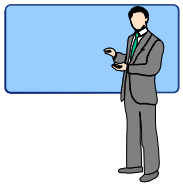
Key OO Concepts/Techniques

- **Abstraction**, Encapsulation, Data Hiding
  - Object vs Class
  - Attribute/Field, Method
  - Method Overloading
- Object Initialization & Usage
  - Constructor
  - Operation vs Method
- **Association**, Aggregation, Composition
- **Generalization**, Inheritance
  - Method Overriding
- Interface vs Abstract Class
- **Polymorphism**

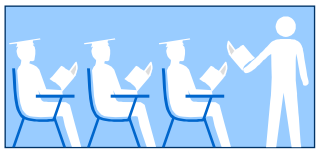
3

Review: Example: Abstraction

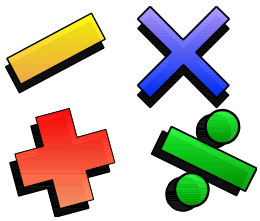
Student



Professor



Course Offering (9:00 AM, Monday-Wednesday-Friday)

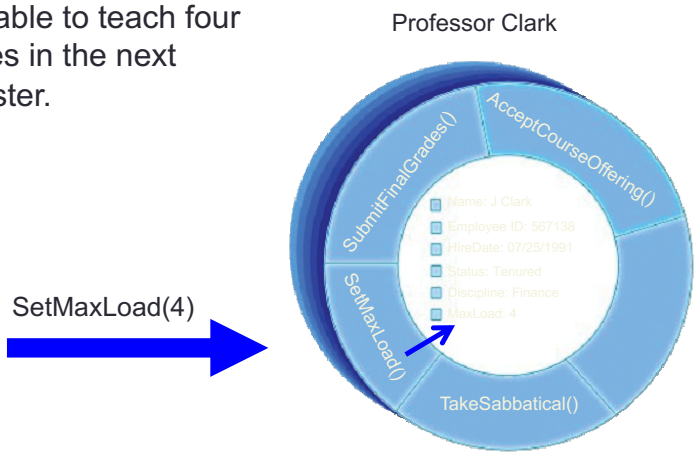


Course (e.g., Algebra)

4

Review: Encapsulation Illustrated

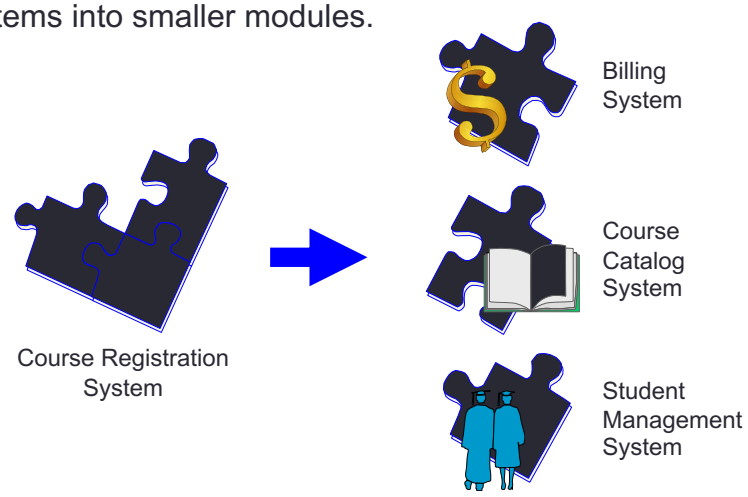
- Professor Clark needs to be able to teach four classes in the next semester.



5

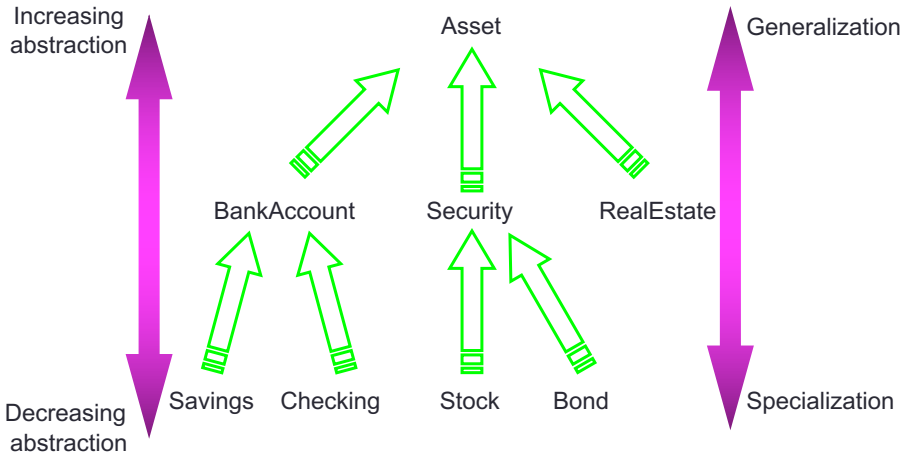
Review: Example: Modularity

- For example, break complex systems into smaller modules.



6

Review: Example: Hierarchy

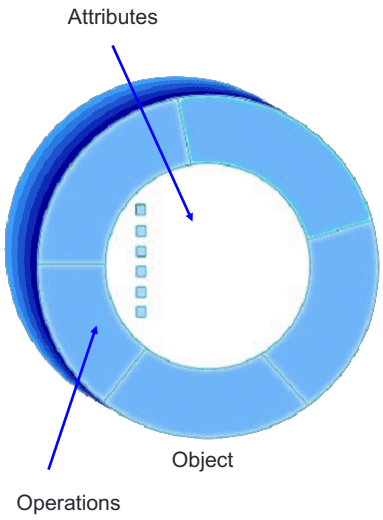


Elements at the same level of the hierarchy should be at the same level of abstraction.

7

Review: What Is an Object?

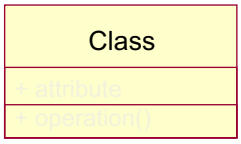
- An object is an entity with a well-defined boundary and *identity* that encapsulates *state* and *behavior.*
  - State is represented by attributes and relationships.
  - Behavior is represented by operations, methods, and state machines.



8

Review: What Is a Class?

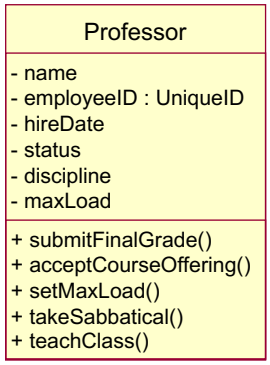
- A class is a description of a set of objects that share the same *attributes*, *operations*, *relationships*, and semantics.
  - An object is an instance of a class.
- A class is an abstraction in that it
  - Emphasizes relevant characteristics.
  - Suppresses other characteristics.



9

Review: Representing Classes in the UML

- A class is represented using a rectangle with compartments.

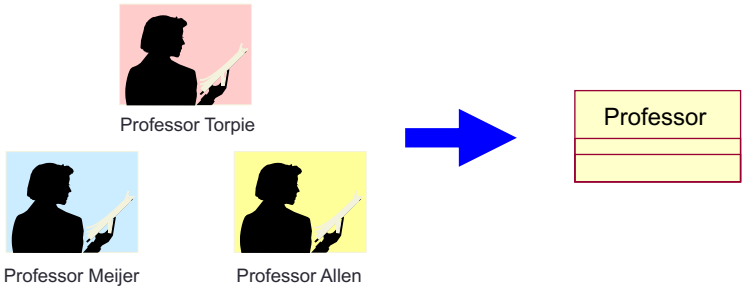


Professor J Clark

10

Review: The Relationship Between Classes and Objects

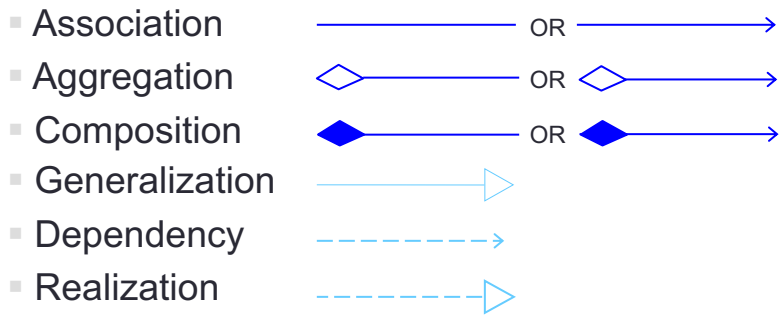
- A class is an abstract definition of an object.
  - It defines the structure and behavior of each object in the class.
  - It serves as a template for creating objects.
- Classes are not collections of objects.



11

Review: Class Relationships

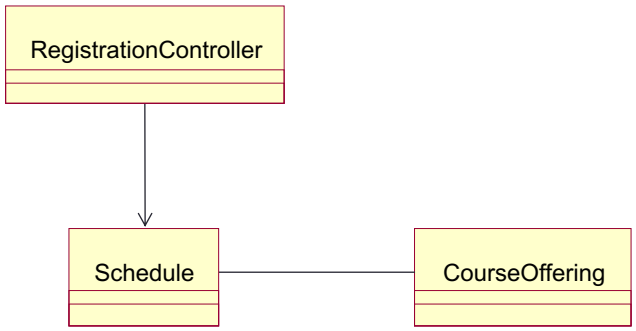
- "The semantic connection between classes" ~ Grady Booch
- Class diagrams may contain the following relationships:



12

What Is Navigability?

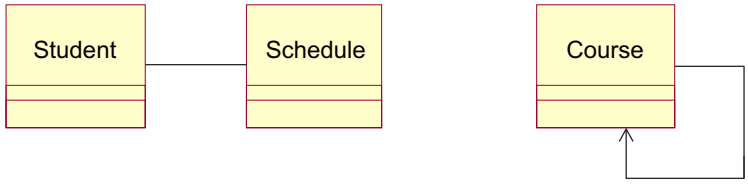
- Indicates that it is possible to navigate from a associating class to the target class using the association



13

Review: What Is an Association?

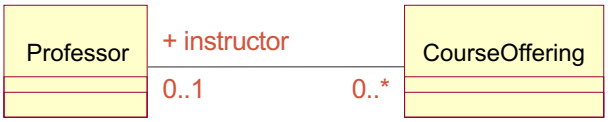
- The semantic relationship between two or more classifiers that specifies connections among their instances
  - A structural relationship, specifying that objects of one thing are connected to objects of another



14

Review: What Is Multiplicity?

- Multiplicity is the number of instances one class relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.



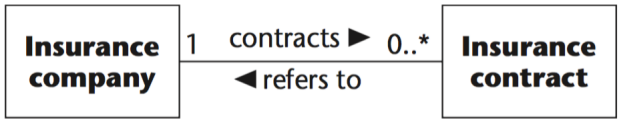
15

Review: Multiplicity Indicators

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional scalar role)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

16

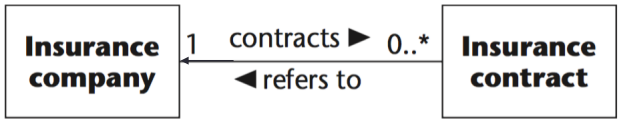
Java implementation for Association




```
//InsuranceCompany.java file
public class InsuranceCompany
{
   // Many multiplicity can be implemented using Collection
   private List<InsuranceContract> contracts;


   /* Methods */
}
// InsuranceContract.java file
public class InsuranceContract
{
   private InsuranceCompany refers_to;

   InsuranceContract(){ }

   /* Methods */
   public void setCompany(InsuranceCompany comp) { refers_to = comp; }
}
```

Field Reference

17




```
//InsuranceCompany.java file
public class InsuranceCompany
{

   /* Methods */
}
// InsuranceContract.java file
public class InsuranceContract
{
   private InsuranceCompany refers_to;

   InsuranceContract(){ }

   /* Methods */
   public void setCompany(InsuranceCompany comp) { refers_to = comp; }
}
```

18

Review: What Is Aggregation?

- An aggregation is a special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - An aggregation is an "Is a part-of" relationship.
- Multiplicity is represented like other associations.

19

Aggregation – Java implementation

```
class Car {
    private List<Door> doors; //Field reference
    Car(String name, List<Door> doors) {
        this.doors = doors;
    }

    public List<Door> getDoors() {
        return doors;
    }
}

class Person {
        Computer computer;
        Person(){ }
        setComputer(Computer computer) { this.computer = computer; }
}
```

20

What Is Composition?

- A composition is a stronger form of association in which the composite has sole responsibility for managing its parts – such as their allocation and deallocation.
- It is shown by a diamond filled adornment on the opposite end.

21

Composition – Java implementation

```
final class Car {
    // For a car to move, it need to have a engine.
    private final Engine engine; // Composition
    //private Engine engine;     // Aggregation

    Car(Engine engine) {
        this.engine = engine;
    }

    // car start moving by starting engine
    public void move() {
        //if(engine != null)
        {
            engine.work();
            System.out.println("Car is moving");
        }
    }
}
```

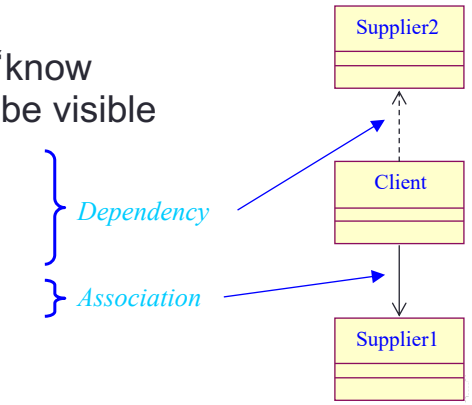
```
class Engine {
    // starting an engine
    public void work() {
        System.out.println("Engine has been started");
    }
}
```

```
class Test {
    // starting an engine
    public void main(String args[]) {
        Engine new_engine = new Engine();
        Car car = new Car(new_engine);
        new_engine = null;
    }
}
```

22

Dependencies vs. Associations

- Associations are structural relationships
- Dependencies are non-structural relationships
- In order for objects to "know each other" they must be visible
  - Local variable reference
  - Parameter reference
  - Global reference
  - Field reference



23

Associations vs. Dependencies in Collaborations

- An instance of an association is a link
  - All links become associations unless they have global, local, or parameter visibility
  - Relationships are context-dependent
- Dependencies are transient links with:
  - A limited duration
  - A context-independent relationship
  - A summary relationship

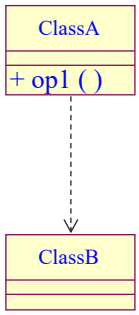
A dependency is a secondary type of relationship in that it doesn't tell you much about the relationship. For details you need to consult the collaborations.

24

Dependecy: Local Variable Visibility

- The op1() operation contains a local variable of type ClassB

```
class ClassA {
       //ClassB b → Association
      op1() {
            ClassB b = new ClassB();
            …
            b.m();
      }
}
```

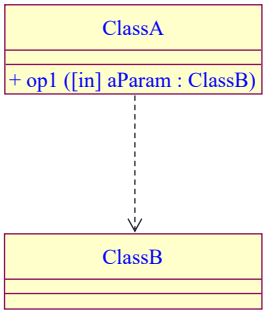


25

Dependecy: Parameter Visibility

- The ClassB instance is passed to the ClassA instance

```
class ClassA {
       //ClassB b → Association
      op1(ClassB b) {

            …
            b.m();
      }
}
```

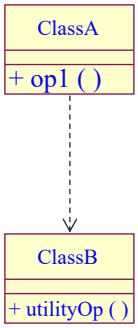


26

Dependecy: Global Visibility

- The ClassUtility instance is visible because it is global

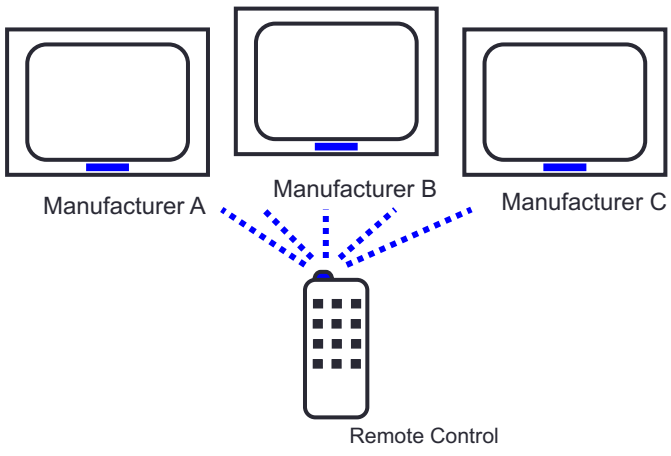
```
class ClassA {
        //ClassB b → Association
       op1() {
              …
              ClassB.utilityOp();
       }
}
```



27

Why Polymorphism?

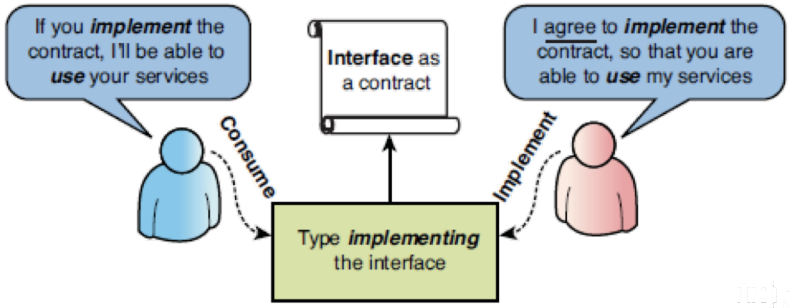
- The ability to hide many different implementations behind a single interface



28

What Is an Interface?

- A declaration of a coherent set of public features and obligations
  - A contract between providers and consumers of services

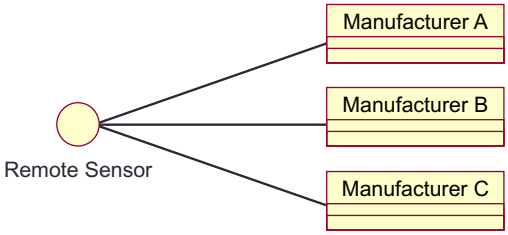


29

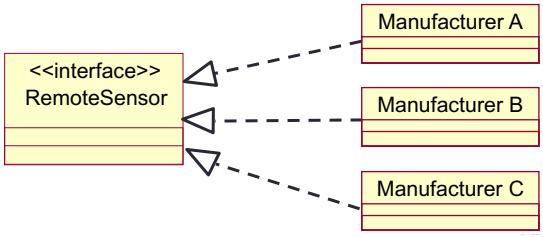
30

Interface Representation in UML

Elided/Iconic Representation ("ball")



Canonical (Class/Stereotype) Representation



30

```
interface TVInterface {
        public void turnOn();
        public void volumnUp(int steps); …
}
class TVA implements TVInterface {
        public void turnOn() { … }
        …
}
class TVB implements TVInterface {…}
class TVC implements TVInterface {…}
class RemoteControl {
        TVInterface tv;
        RemoteControl(TVInterface tv){setTV(tv);}
        void setTV(TVInterface tv){
                this.tv = tv;
        }
        volumnUp(int steps){ tv.volumnUp(steps); }
}
```

```
RemoteControl remote = new RemoteControl(new TVA());
remote.volumnUp(3);
```


Polymorphism

- Polymorphism: multiple ways of performance, of existance
- Polymorphism in OOP
  - Method polymorphism:
    - Methods with the same name, only difference in argument lists => method overloading
  - Object polymorphism
    - **Multiple types:** A single object to represent multiple different types (upcasting and downcasting)
    - **Multiple implementations/behaviors:** A single interface to objects of different types (upcasting+overriding – dynamic binding)

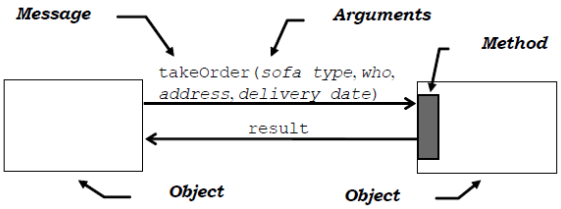
      → **Dynamic Polymorphism**

Comparison

- Object vs Class
- Operation vs Method
- Method vs Message
- Abstract Class vs Interface
- Aggregation vs Inheritance
- Association vs Aggregation
- Aggregation vs Composition
- Association vs Dependency
- Call Function vs Send Message
- Method Overloading vs Method Overriding...

Message vs. Method

- Message
  - Is sent from an object to another object and does not contain any piece of code to be executed
- Method
  - Method/function in structure programming languages
  - Is an execution of service that is requested in the message
  - Is a piece of code to be executed in order to respond to a message sent to an object



Function call vs. Message passing

- Call function
  - Indicate the exact piece of code to be executed.
  - Has only an execution of a function with some specific name.
  - There are no functions with the same name
- Message passing
  - **Request a service from an object and the object will decide what to do**
  - **Different objects will have different re-actions/behaviors for a message.**

35


```
class A {
        private int a1;
        public float m(int i){
                …
                return …
        }
}

class B {
        void b(){
                A a = new A();
                a.m(9);
        }
}
```

- attribute
- behavior/operation: method signature
- **method: how?**

Objects can communicate to each other through **sending messages**

36

Exam Structure Example (90 minutes)

- Part 1
  - Short questions: Analysis or comparison on OO concepts/techniques (more on theory/design)
  - Short exercises: Given a problem and class diagram, implement in Java code (more on code)
- Part 2
  - Given the requirement for a program
  - Draw Use case diagram (optional)
  - Draw Class diagram
  - Write source code for a part of the program