**ÔN TẬP PHẦN LÝ THUYẾT**

**MÔN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

1. Phân biệt cách tiếp cận lập trình truyền thống và lập trình hướng đối tượng.
2. Các khái niệm trong lập trình hướng đối tượng

* Object
* Class (phân biệt structures và classes)
* Encapsulation
* Thừa kế
* Đa hình

1. Ngôn ngữ lập trình hướng đối tượng Java

* Kiểu dữ liệu (primary data types)
* Khai báo hằng số, biến (biến static)
* Biểu thức và các loại toán tử trong ngôn ngữ lập trình Java
* Các cấu trúc điều khiển
* Các cấu trúc lặp
* Khai báo phương thức (cú pháp: kiểu dữ liệu trả về, tham số, ….). Phân biệt phương thức *static* và phương thức thông thường. Tại sao phương thức *main* lại khai báo *static*?
* Package

1. Các khái niệm lập trình hướng đối tượng áp dụng vào ngôn ngữ Java

* Class
  + Khai báo lớp
  + Phân biệt constructors/default constructor và các hàm xử lý thông thường
  + Khai báo biến của lớp (instance variables)
  + *this* và cách sử dụng *this*
* Object
  + Khai báo đối tượng
  + Cấp phát vùng nhớ cho đối tượng bằng toán tử *new*
* Encapsulation (lưu ý encapsulation với khai báo public, protected private)
  + Phương thức get/set
* Thừa kế (đơn thừa kế, đa thừa kế).
  + Lớp Object
  + Định nghĩa lớp thừa kế
  + Định nghĩa Interface
  + Định nghĩa abstract class
  + Phân biệt giữa *interface, abstract class và class*
* Đa hình:
  + Phân biệt overloading, overriding
  + Khái niệm dynamic binding
* Exception – Garbage collection
  + Quản lý lỗi (syntax errors, run-time errors)
  + Bắt lỗi try … catch. Ném ra lỗi (throw/throws)
  + Cơ chế hoạt động của Garbage collection
* Collection và mảng
* Generics
  + Khái niệm Generics
  + Generics ở mức lớp/phương thức
  + Dùng Generics với các Collection trong Java
* Thao tác nhập xuất với Stream
  + InputStream và các lớp hỗ trợ nhập (byte, character, filter, data type, object, …)
  + OutputStream và các lớp hỗ trợ xuất (byte, character, filter, data type, object, …)
  + Standard Stream: System.in, System.out, System.err

1. Bài tập tổng hợp: Dựa trên mô tả → xác định thuộc tính, phương thức, sau đó định nghĩa lớp, lớp thừa kế, … theo mô hình lớp (class diagram) của UML

Xây dựng chương trình quản lý danh sách các tài khoản ngân hàng. Hệ thống quản lý 2 loại tài khoản:

*Thông tin về Tài khoản thẻ:* mã số tài khoản bao gồm 8 số (duy nhất), tên khách hàng, địa chỉ liên lạc, ngày tạo tài khoản, số dư tài khoản và ngày giao dịch cuối cùng.

* Ngày tạo tài khoản sẽ là ngày hiện hành.
* Trong trường hợp tài khoản mới chưa giao dịch lần nào thì *ngày tạo tài khoản = ngày giao dịch cuối cùng.*

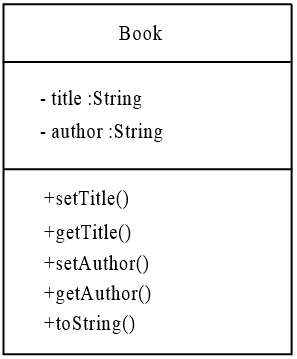
*Thông tin về Tài khoản tiết kiệm:* mã số tài khoản bao gồm 8 số (duy nhất), tên khách hàng, địa chỉ liên lạc, ngày tạo tài khoản, loại tiền gửi, thời hạn gửi, số dư tài khoản và ngày đáo hạn.

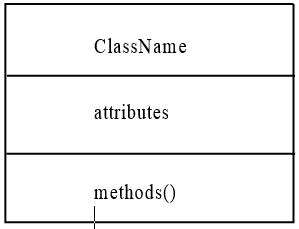
* Ngày tạo tài khoản sẽ là ngày hiện hành.
* Loại tiền gửi chỉ bao gồm tiền VNĐ và tiền USD. Tỷ giá 1 USD = 21000 VNĐ.
* Thời hạn gửi bao gồm 1 tháng, 3 tháng và 6 tháng (1 tháng tương ứng với 30 ngày).
* N*gày đáo hạn > ngày tạo tài khoản + thời hạn gửi ngày giao dịch cuối cùng.*

Thực hiện các yêu cầu sau:

1. Xác định các lớp (entity classes).
2. Xác định các mối quan hệ giữa các lớp (association).
3. Vẽ mô hình lớp (class diagram) với các quan hệ.
4. Định nghĩa các lớp bằng ngôn ngữ lập trình Java
5. Viết chương trình xử lý

**PHẦN THAM KHẢO MÔ HÌNH LỚP (CLASS DIAGRAM)**

Các loại lớp

* Boundary Classes, (View)  
  Bao gồm cả phương thức, thuộc tính. Đa số các Boundary  
  classes là giao diện xử lý giữa người dùng và hệ thống.
* Control Classes (Controller)  
  Chỉ bao gồm các phương thức xử lý. Tối thiểu phải có 1   
  phương thức.
* Entity Classes (Model)  
  Chỉ bao gồm attributes (+get/set/find/toString/equals  
  methods)

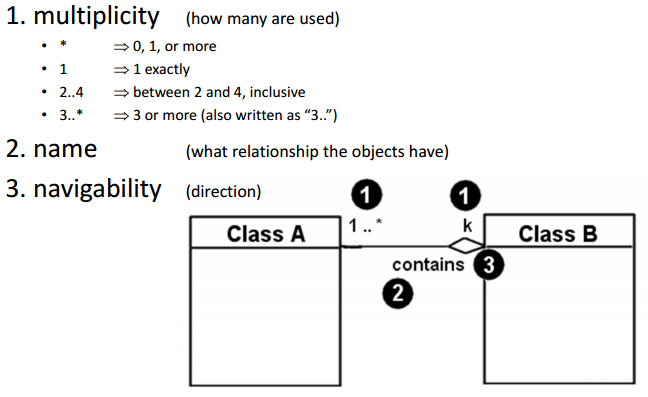
Quan hệ giữa các loại lớp

* Tác nhân hay người sử dụng hệ thống chỉ tương tác với các boundary classess
* Các Entity classes chỉ được truy xuất bởi Control classes.

*Mô hình lớp bao gồm*

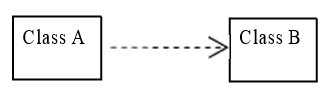
|  |  |  |
| --- | --- | --- |
| Classes  (Các lớp) | Relationships  (Các quan hệ) | Generalization/ Specialization  (Tổng quát hóa, chuyên biệt hóa) |
| attributes  operations  visibility | navigability  multiplicity  dependency  aggregation  composition | inheritance  interfaces |

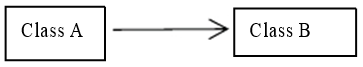
*Ví dụ:* Quan hệ kết hợp giữa các lớp



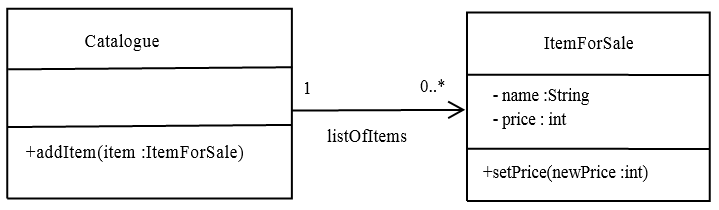
Cú pháp các quan hệ kết hợp các lớp trong UML

1. **Dependency (**Quan hệ “use-a”: lớp A tham chiếu đến một đối tượng thuộc lớp B (không tạo trong lớp A))

* Class A in some way uses facilities defined by Class B
* Class A has a method which is passed a parameter object of Class B, or uses a local variable of that class, or calls ‘static’ methods in Class B

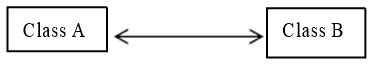
1. **Simple Association (**Quan hệ “Has a”: đối tượng lớp B được tạo trong lớp A nhưng có thể sử dụng ngoại lớp A (có phương thức getter trong lớp A trả về tham chiếu của đối tượng lớp B))

Class A has an attribute of Class B



1. **Bidirectional Association** (Quan hệ 2 chiều)

* Classes A and B have a two-way association
* A Class A object can access the Class B object(s) also, a Class B object can access the Class A object(s)



1. **Aggregation** (Tập hợp - “is part of”)

* Aggregation denotes a situation where Object(s) of Class B ‘belong to’ Class A
* Implies reference from A to B
* Class B retain an existence independent of Class A
* Some designers believe there is no real distinction between aggregation and simple association

1. **Composition** (Thành phần - “is entirely made of”)

* Similar to aggregation but implies *a much stronger belonging relationship* (objects of Class B never exist other than as part of Class A)
* Implies reference from A to B
* Class B retain an existence independent of Class A

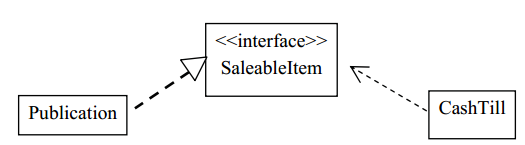
1. **Generalization/Specialization**: **Inheritance**

Class A ‘inherits’ both the interface and implementation of Class B, though it may override implementation details and supplement both

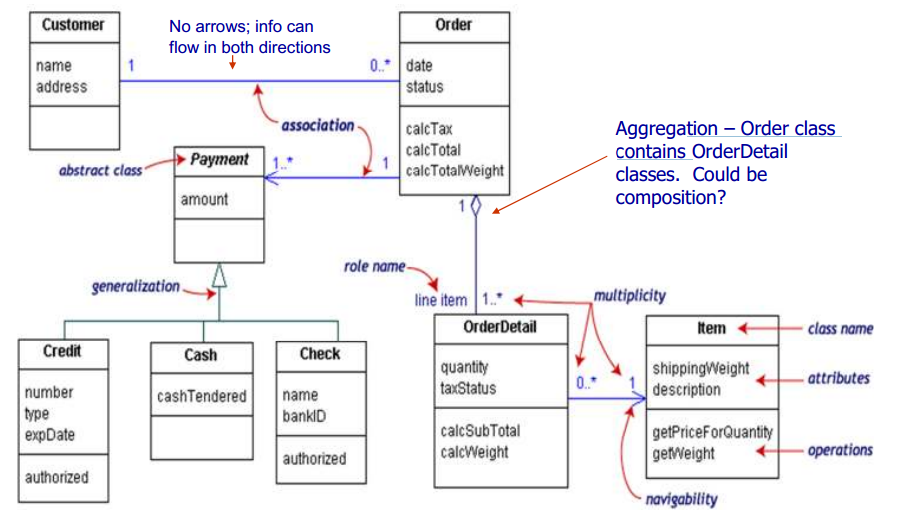


1. **Generalization/Specialization**: **Interfaces**

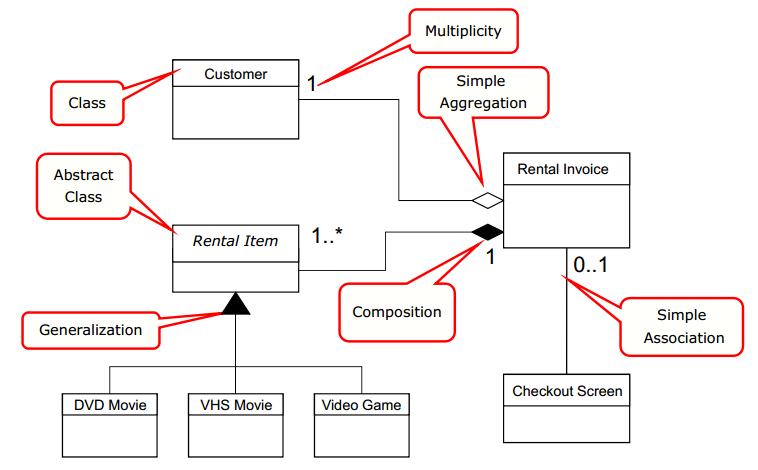
* Similar to inheritance however with interfaces only the interface is inherited
* The methods defined by the interface must be implemented in every class that implements the interface.



*Ví dụ 1:*



*Ví dụ 2:*



*Ví dụ 3:*So sánh cách thể hiện quan hệ kết hợp Aggregation và Composition.

Đoạn code ở trên (Client) là cách thể hiện quan hệ kết hợp Composition, vì acc sẽ bị hủy khi Client bị hủy.

Đoạn code ở dưới (Client1) là cách thể hiện Aggregation, acc ko bị hủy khi Client bị hủy mà có thể tồn tại độc lập:

Class Client  
{  
    BankAccount acc = new BankAccount();

public void addMoneyToBankAccount(decimal amount)  
{   
          acc.AddMoney(amount);  
}   
public decimal CheckBalance()  
{  
         return acc.CheckAccountBalance();  
}

}

Class Client1  
{

BankAccount acc;  
public Client(BankAccount pacc) // constructor  
{  
          acc = pacc;  
}   
public void addMoneyToBankAccount(decimal amount)  
{  
        acc.AddMoney(amount);  
}   
public decimal CheckBalance()  
{

return acc.CheckAccountBalance();

}

}