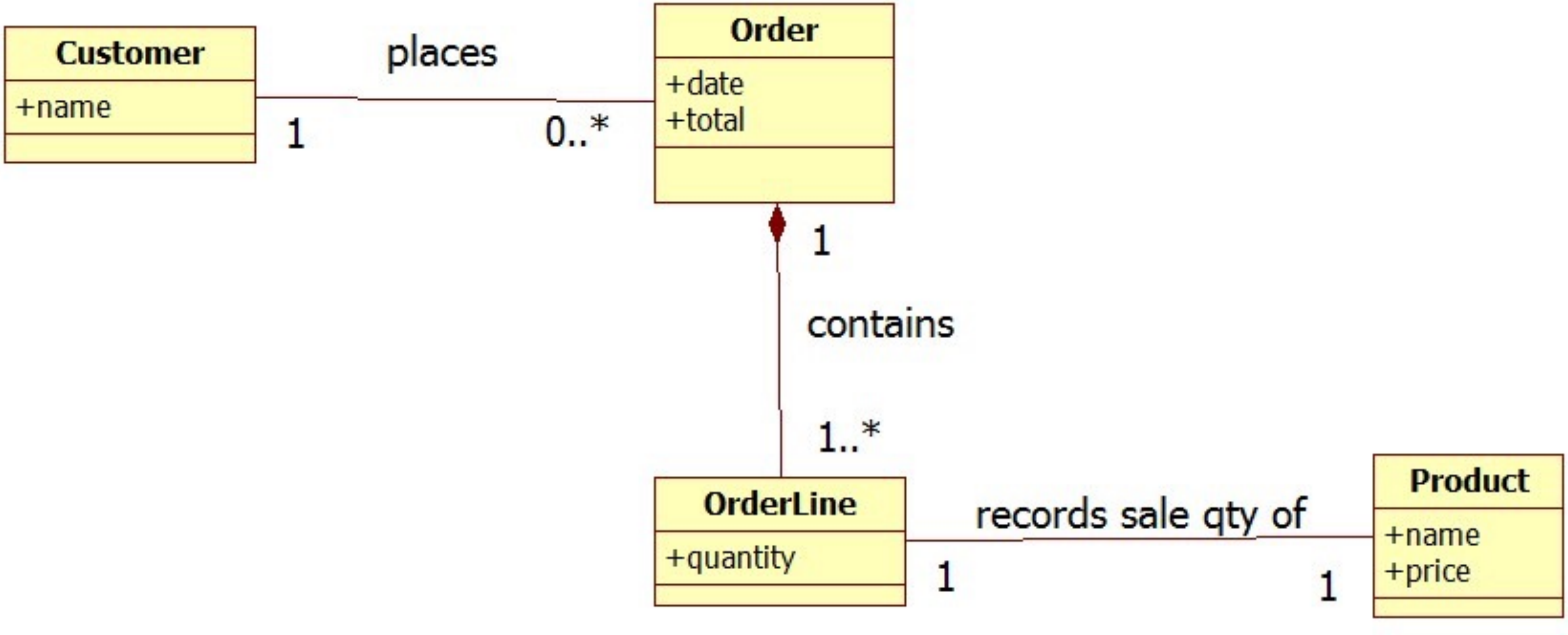
SYS466
Analysis and Design

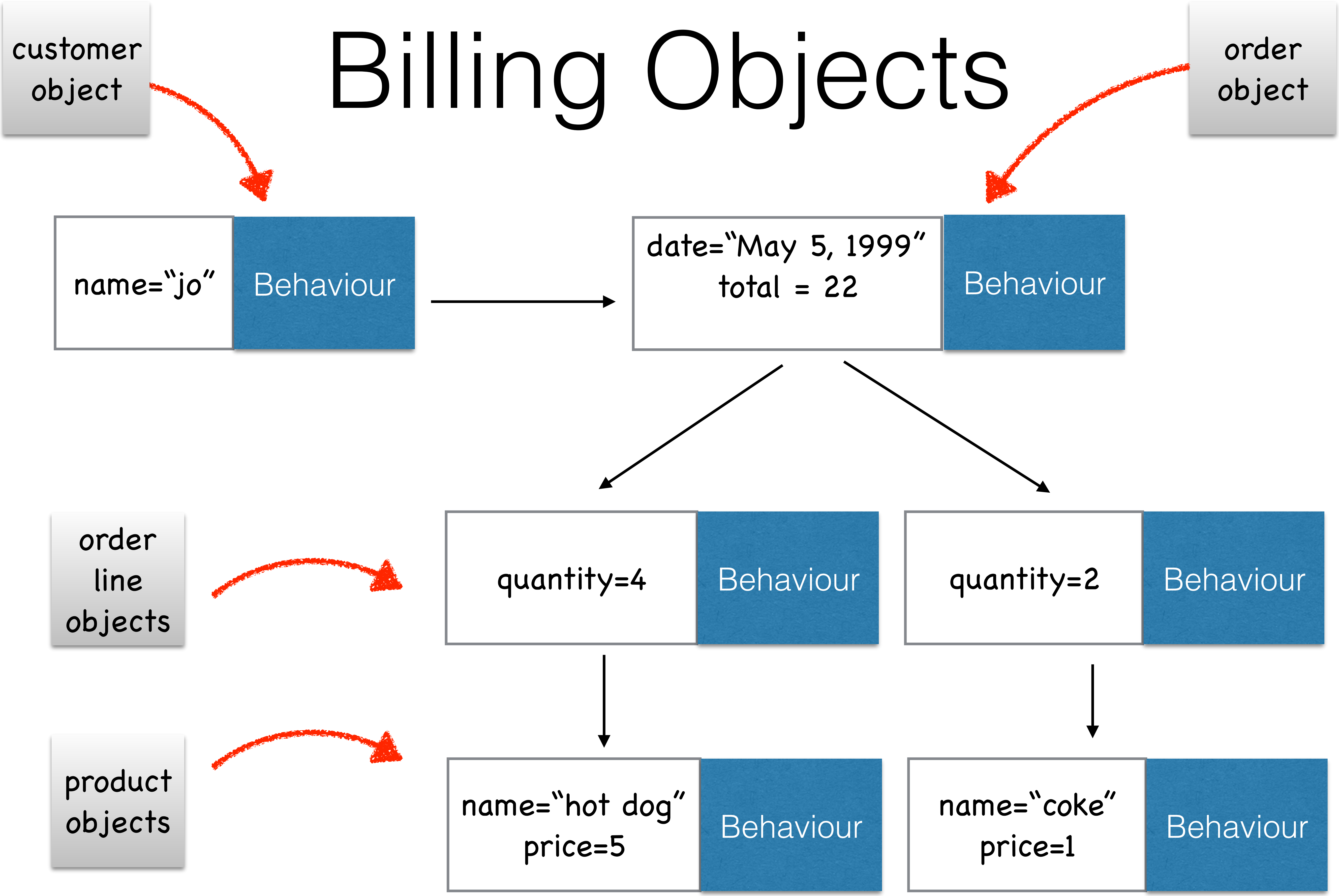
Lecture 4 - Advanced Domain Modelling
School of Information and Communications Technology
Seneca College



Domain Model for Bills

A common group of conceptual classes in a customer billing system

Billing Objects



customer object

order object

name="jo"

Behaviour

date="May 5, 1999"
total = 22

Behaviour

order line objects

quantity=4

Behaviour

quantity=2

Behaviour

product objects

name="hot dog"
price=5

Behaviour

name="coke"
price=1

Behaviour

Notes about Order Example

- observations
  - Order contains many OrderLine objects.
  - OrderLine objects have no “life” outside of order.
  - OrderLine objects have no meaning outside of order.
  - If you delete Order, all OrderLine objects will be gone.
  - Order “encapsulates” OrderLine.

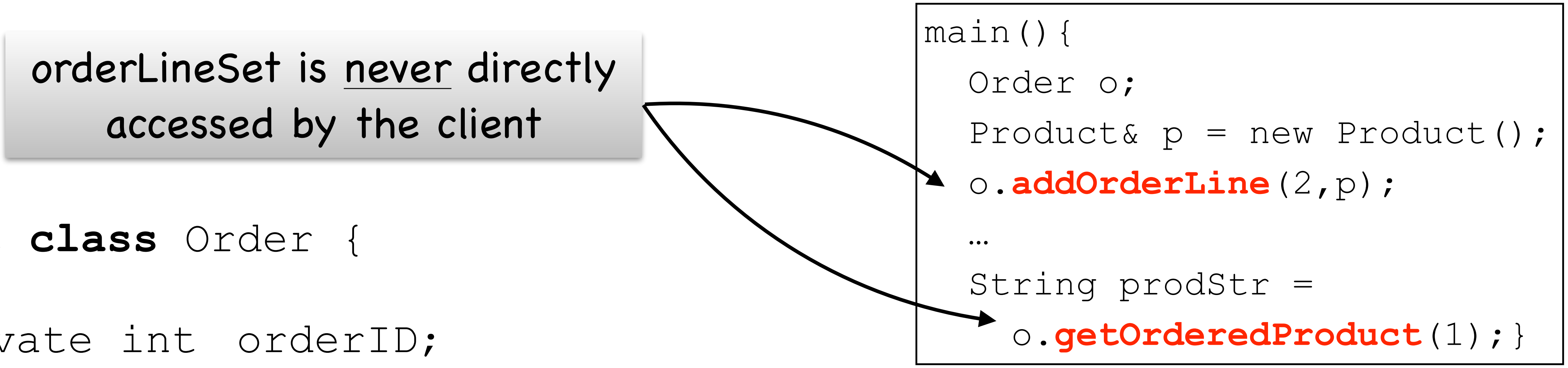
Order-OrderLine Association

- Order becomes the “proxy” for OrderLine:
  - Order takes responsibility for creation of OrderLine objects
  - Order controls all communication to OrderLine objects
  - No other class knows that OrderLine exists
  - But OrderLine knows about other classes (e.g. Product)
  - OrderLine might not even appear in higher level models; only Order.

Potential Implementation

```
class Order {

  private int orderID;
  private Customer customer;
  private List<OrderLine> orderLineSet;

  public Order(int newOrdID) {
    orderID = newOrdID; customer = new Customer();
    orderLineSet = new ArrayList<OrderLine>();}

  public Order() {
    orderID = 0; customer = new Customer();
    orderLineSet = new ArrayList<OrderLine>();}

  public void addOrderLine(int inQty, Product inProd) {
     OrderLine newOrderLine = new OrderLine(inQty,inProd);
     orderLineSet.add(newOrderLine);
  }

….
```

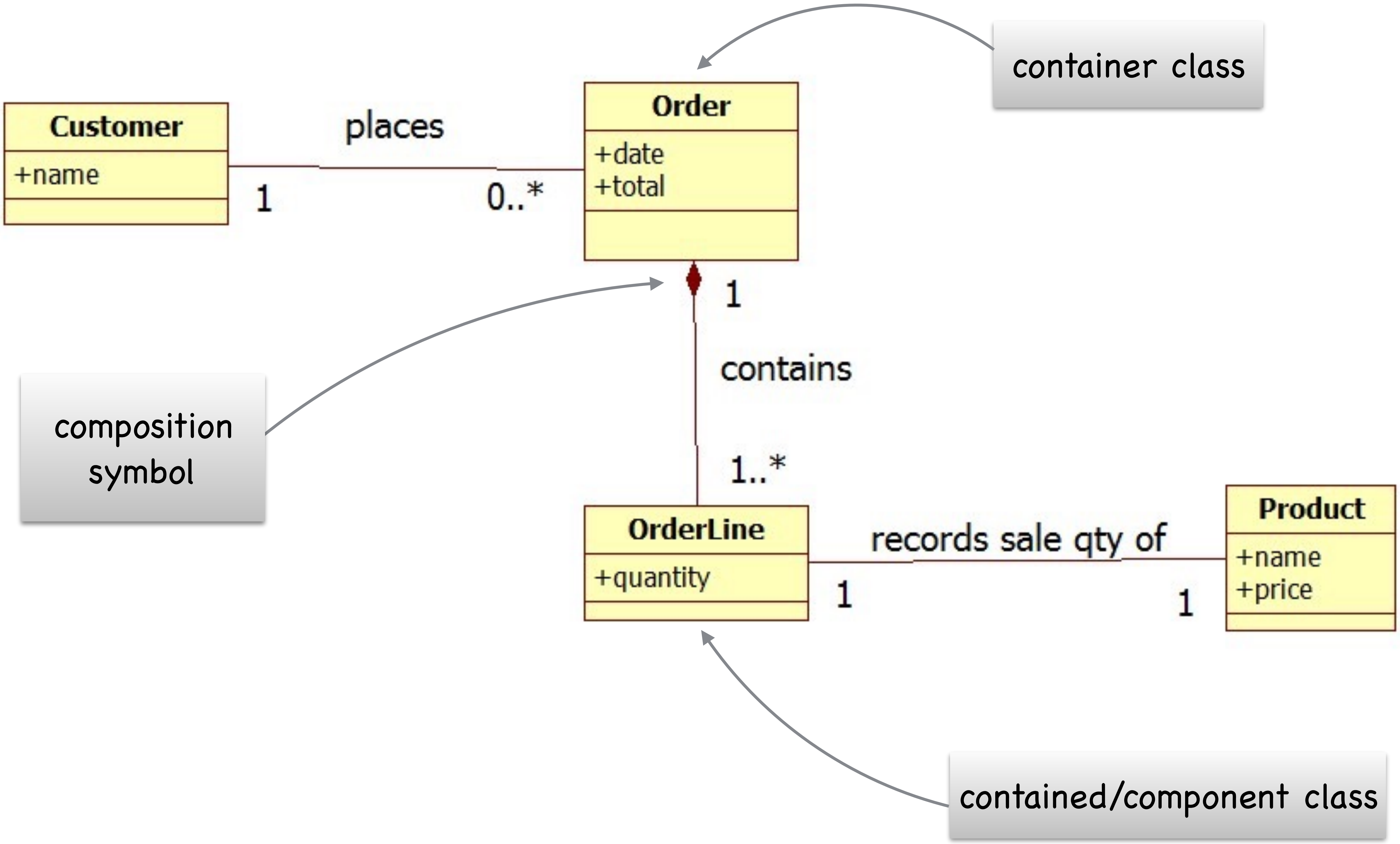
Potential Implementation



orderLineSet is never directly accessed by the client

```
main(){
  Order o;
  Product& p = new Product();
  o.addOrderLine(2,p);
  …
  String prodStr =
    o.getOrderedProduct(1);}
```

```
public class Order {

   private int  orderID;
   private Customer customer;
   private List<OrderLine> orderLineSet;

   public void addOrderLine(int inQty, Product inProd) {
      OrderLine newOrderLine = new OrderLine(inQty,inProd);
      orderLineSet.add(newOrderLine);
   }
   public String getOrderedProduct(int orderLineInd) {
      return orderLineSet.get(orderLineInd).getProductName();
   }
   public int getOrderedQty(int orderLineInd) {
      return orderLineSet.get(orderLineInd).getQty();
   }}
```



Composition in Class Diagrams

When to use Composition?

- key notes
  - To denote a contained relationship that effectively hides a contained class from “outsiders”.
  - Using a container can simplify a model; the container will be a proxy for all attributes and operations of the contained class. The contained class can be hidden.
  - A composite relationship is reflected in code; if it is not specified then the code may not do what it is meant to. (e.g. the contained class may be visible to “outsiders”).

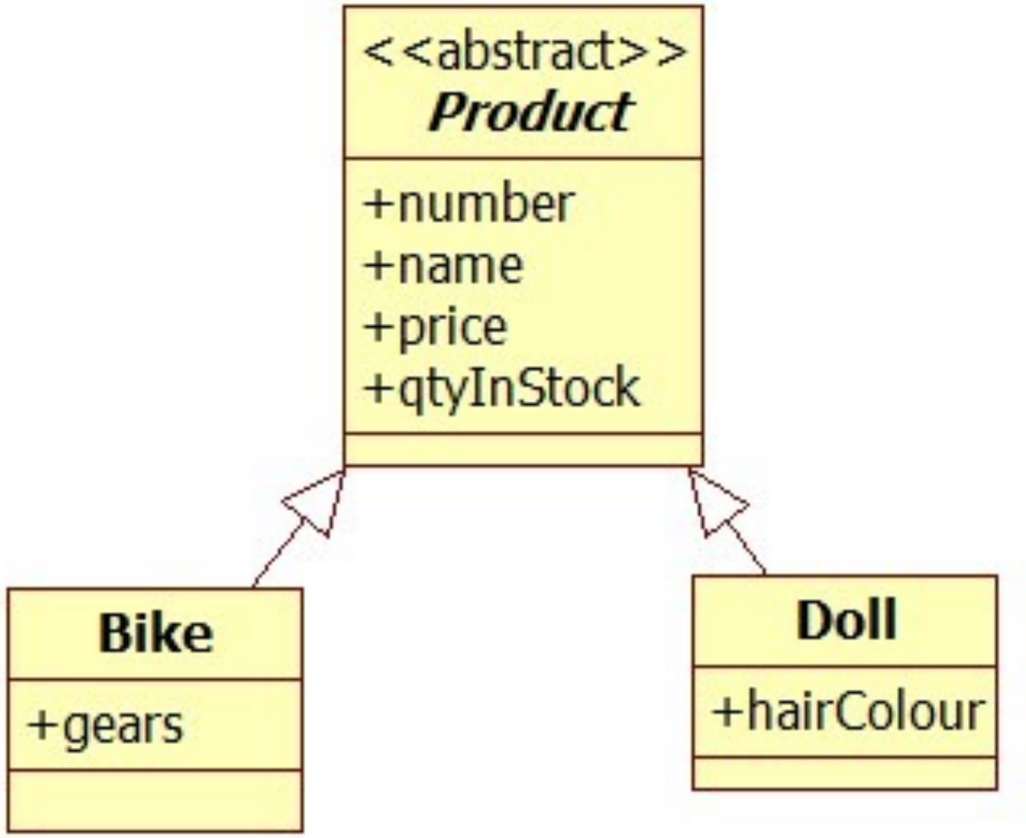
Generalization

- when multiple classes share certain properties, they may be “grouped” into a conceptual hierarchy

- a superclass represents the general concept

- subclasses represent <u>specializations</u> of the superclass

- the superclass is deemed to be <u>abstract</u>, if it can not be created in your conceptual model



Example: Bank Accounts

Interest Chequing Account

No Interest Chequing Account

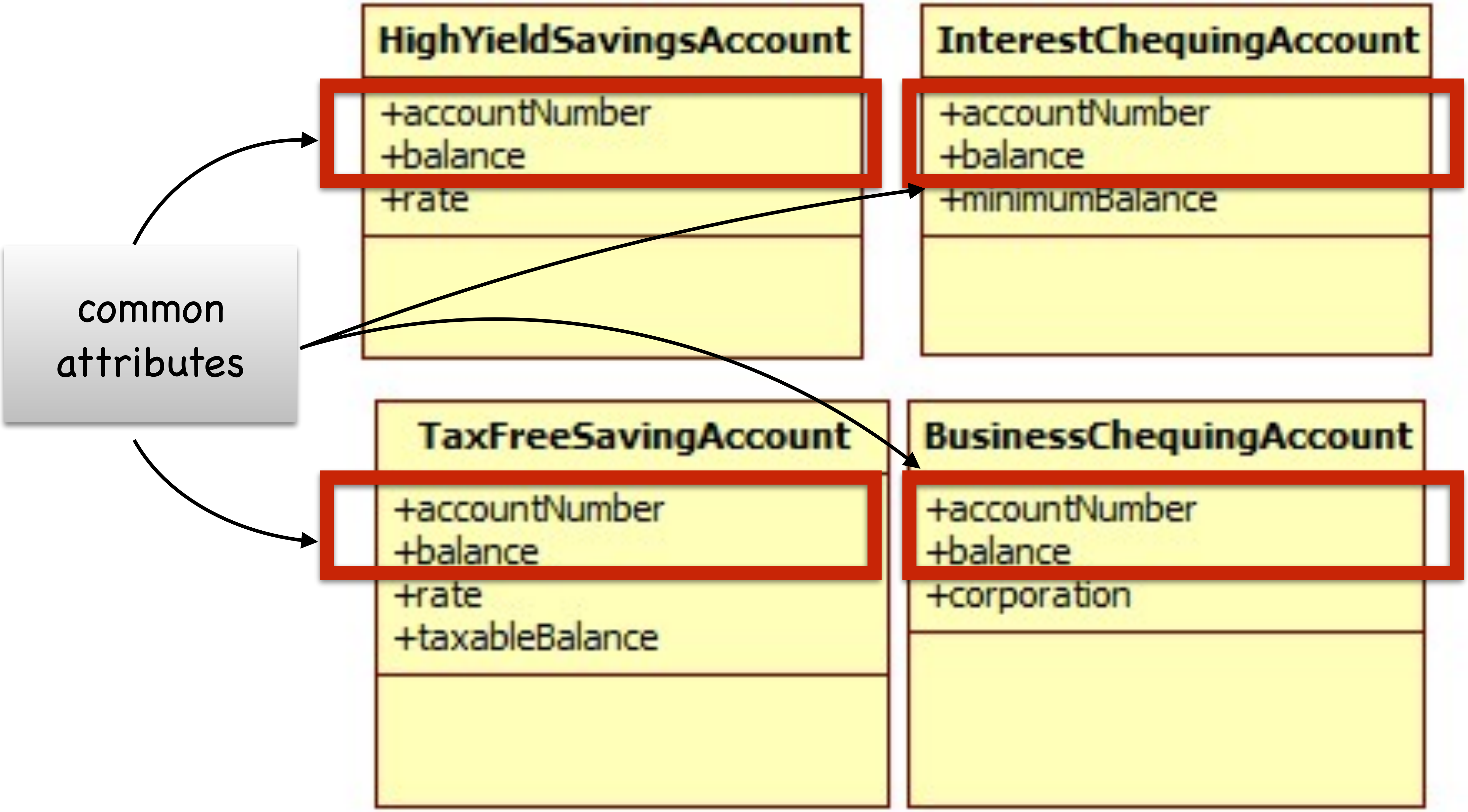
High Yield Savings Account

Business Chequing Account

Tax Free Savings Account

Bonus Savings account

Potential Class Diagram



common attributes

HighYieldSavingsAccount

+accountNumber
+balance
+rate

InterestChequingAccount

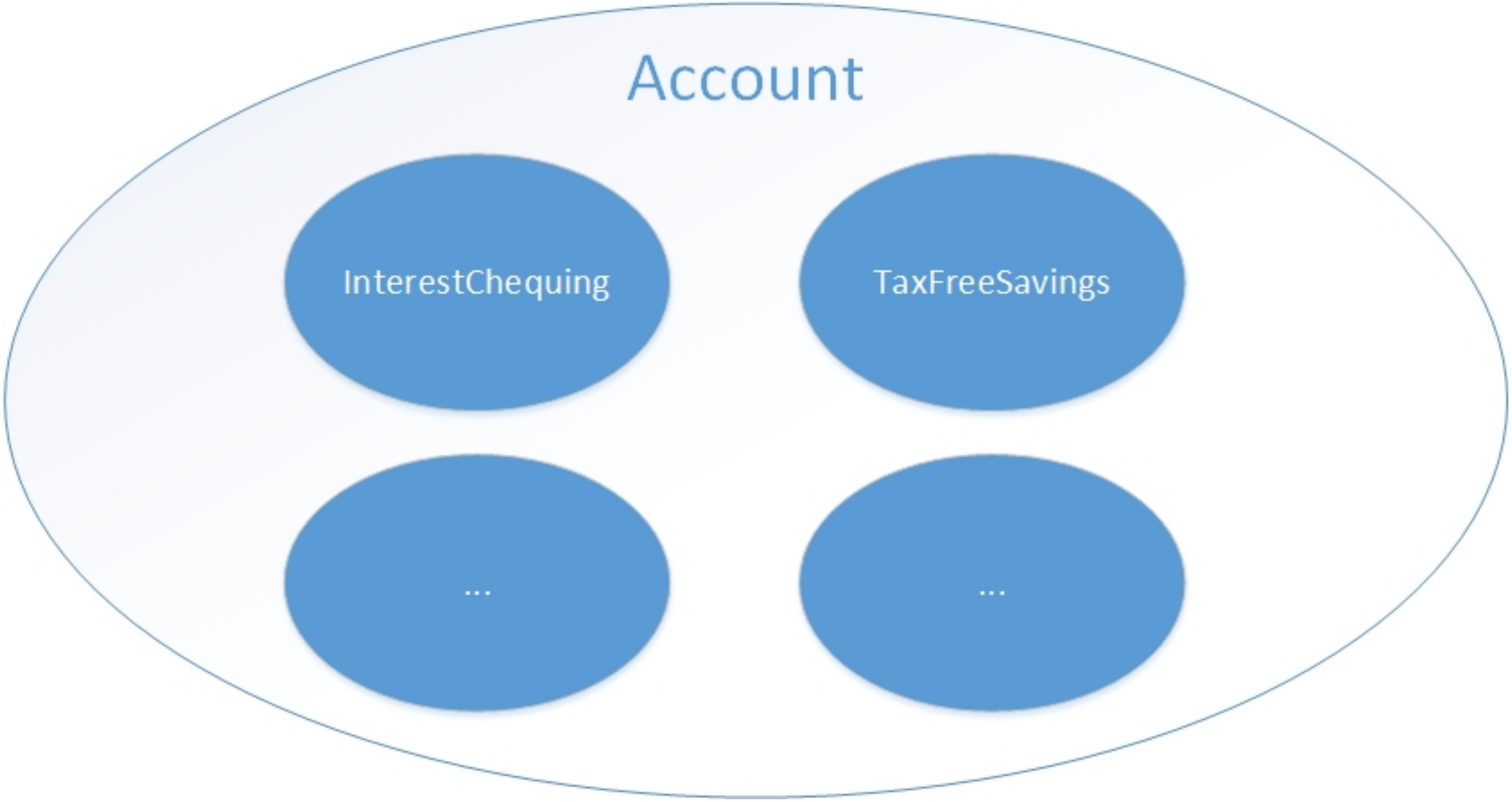
+accountNumber
+balance
+minimumBalance

TaxFreeSavingAccount

+accountNumber
+balance
+rate
+taxableBalance

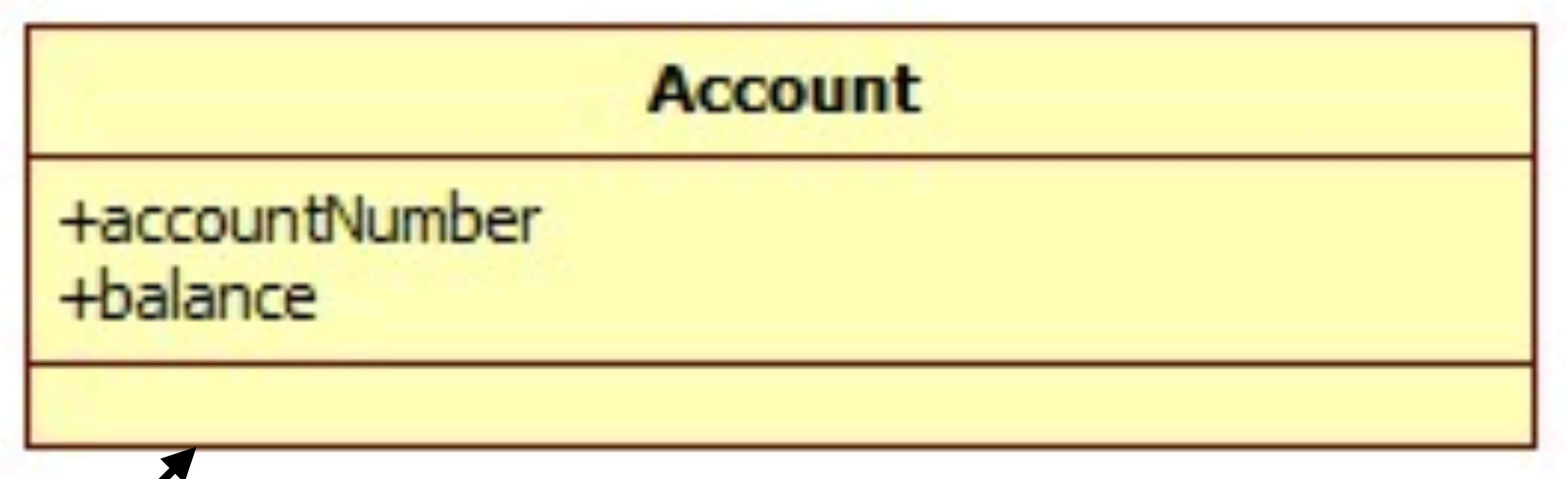
BusinessChequingAccount

+accountNumber
+balance
+corporation



Specialization

Different account types can be seen as "special" accounts

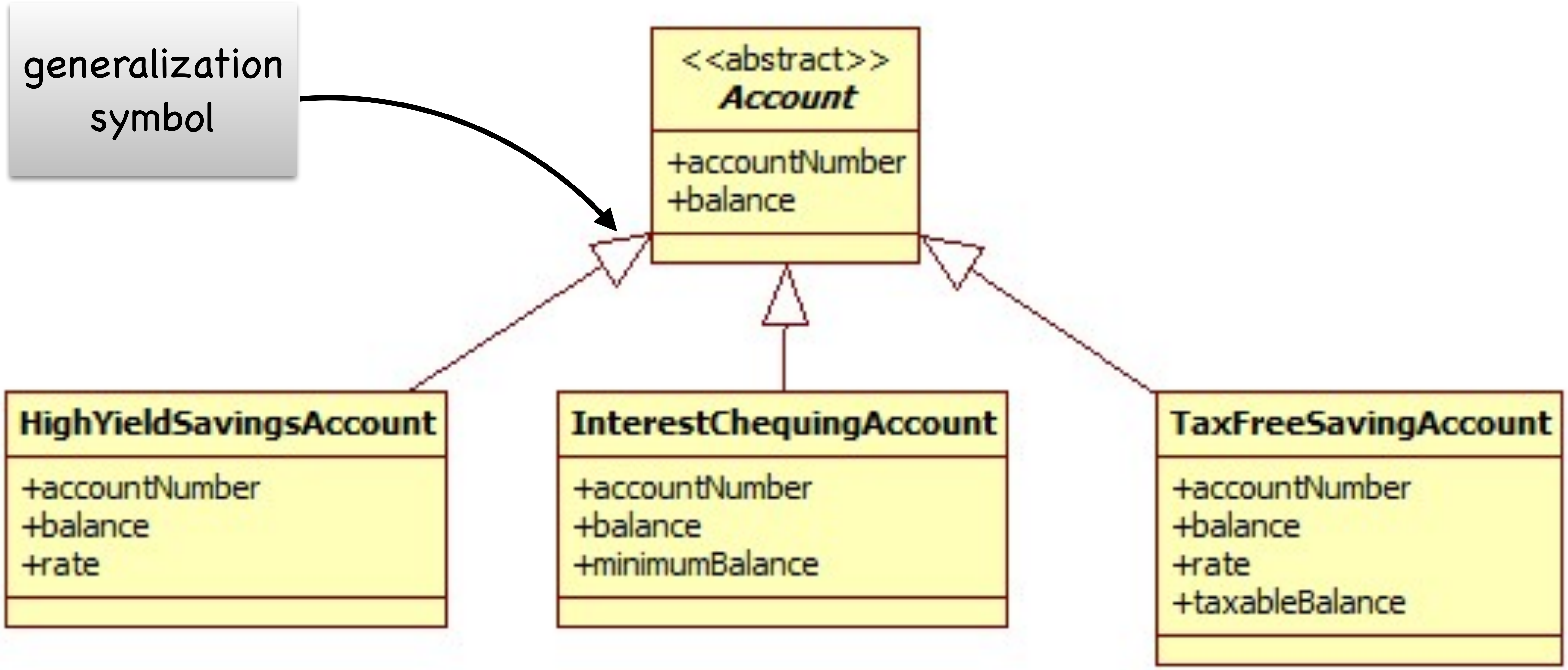


conceptual superclass

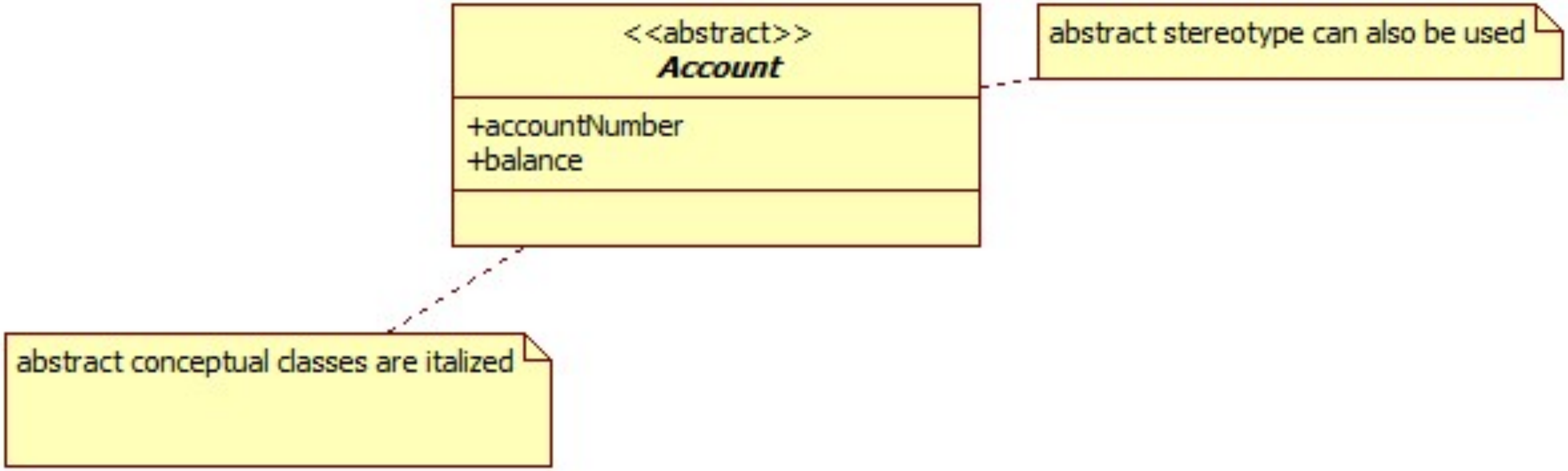
Generalization

… and all these different accounts can be seen as variations of some “general”account…

Generalization in UML



generalization symbol


<<abstract>>
Account
+accountNumber
+balance
abstract stereotype can also be used
abstract conceptual classes are italized

Abstract Conceptual Class

all objects must be a member of a subclass

Generalization Summary

- Each subclass is a specialization of the superclass
- The superclass is a generalization of all of its subclasses
- A conceptual subclass:
  - inherits attributes from its conceptual superclass
  - represents a variation of its conceptual superclass
- model determines whether superclass objects can be created
- if superclass cannot be created, it is abstract