Exercise 1 - Association

- How would you implement in Java these two simple examples of a normal association and a composition (I will go around picking some of your work)

A possible solution for association

```java
public class University {
  private String name;
  private List<Student> students;

  public University(String name) {
    this.name = name;
    this.students = new ArrayList<Student>();
  }

  public void addStudent(Student newStudent){
    if(!students.contains(newStudent)){
      students.add(newStudent);
      newStudent.setUniversity(this);
    }
  }
}
```

```java
public class Student {
  private String id;
  private University university;

  public Student(String id) {
    this.id = id;
  }

  public void setUniversity(University uni){
    this.university = uni;
  }
}
```

A possible solution for composition

```
public class Car {

  private List<Wheel> wheels;


  public Car() {

    this.wheels = new ArrayList<Wheel>(4);

    for (int i = 0; i < wheels.size(); i++) {

      this.wheels.add(i, new Wheel(this));

    }

  }

}
```

```
public class Wheel {

  private Car car;


  Public Wheel(Car car) {

    this.car = car;

  }

}
```

Exercise 2 – Bank System

Design a UML class diagram that could represent this system:

A bank system contains data on customers (identified by name and address) and their accounts.

Each account has a balance and there are 3 type of accounts: checking for daily operations, saving which offers an interest rate and investments used to buy stocks. Stocks (identified by a ref) are bought at a certain quantity for a certain price (ticker).

Basic Design Approach

- What are the smaller elements (classes) in the system?
- What do those elements basically do?
- How are they hierarchically organised (inheritance)
- How are they associated with one another?
- How do they fit into the architecture (maybe each architectural component is a package?)

Naive heuristic for achieving a design. Usually done on requirements (eg. user story)

1. find classes by looking for nouns
2. find methods by looking for verbs
3. find fields by looking at attributes of nouns
4. derive associations and specialisation relationships between classes

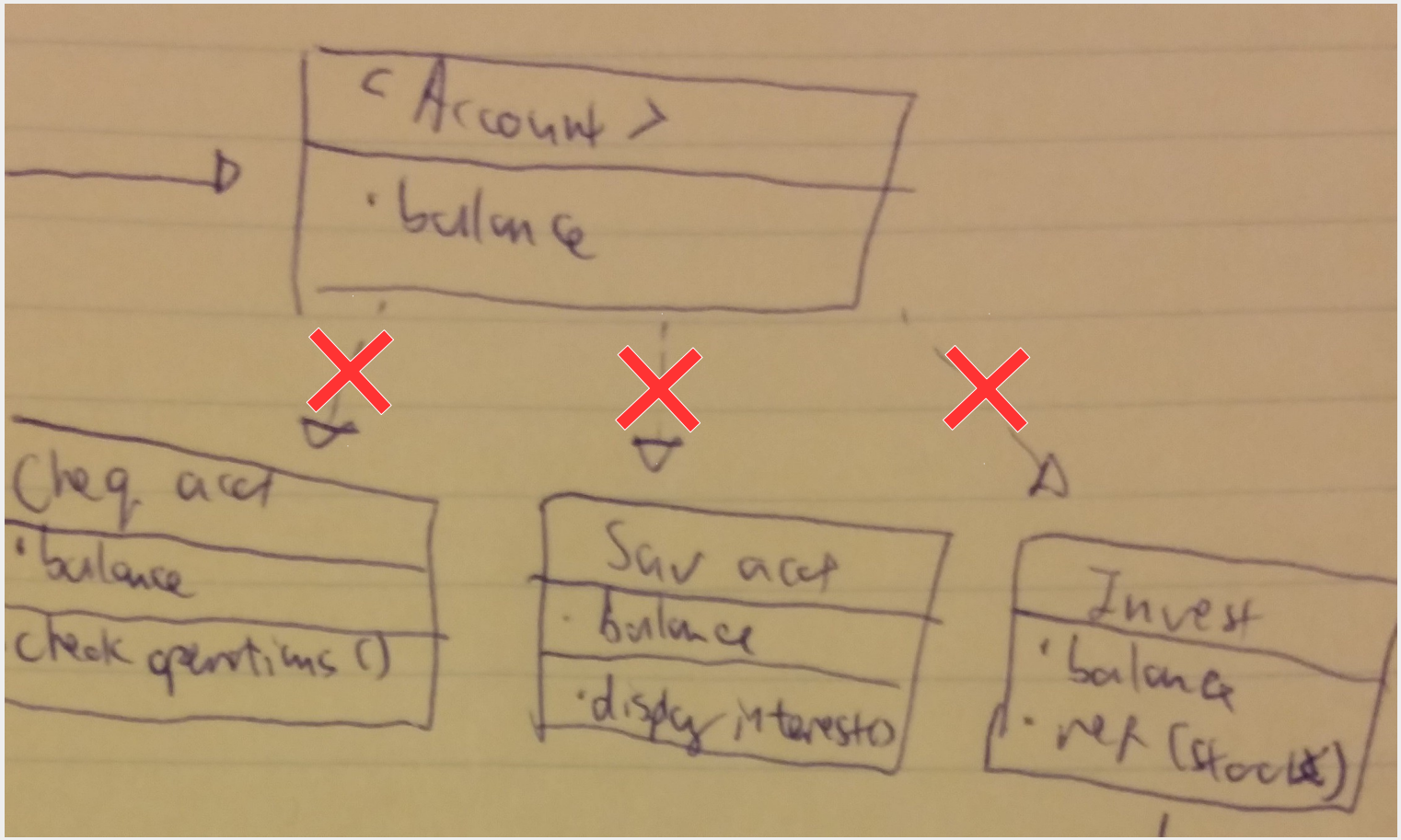
Exercise 2 – Bank System

Design a UML class diagram that could represent this system:

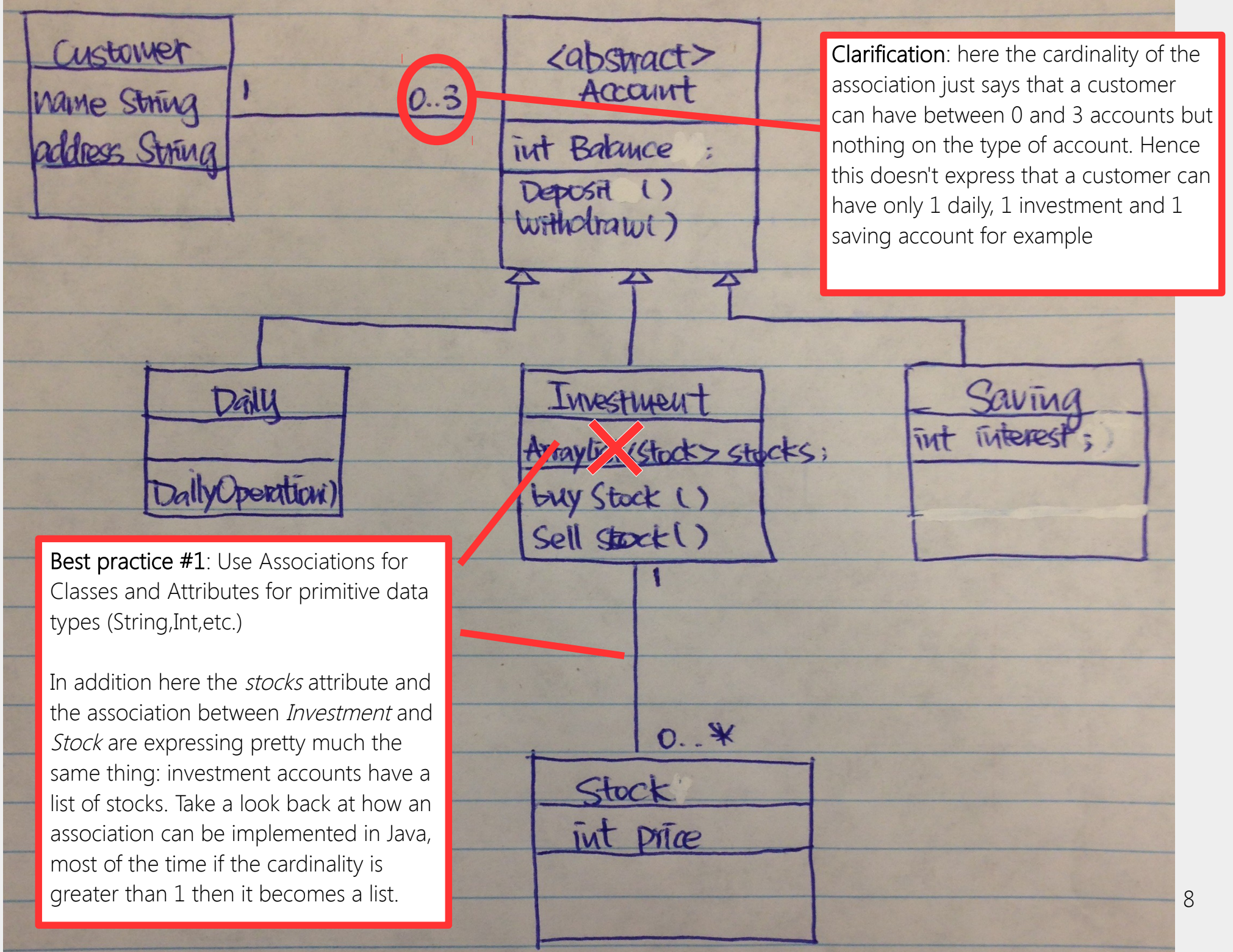
A bank system contains data on customers (identified by name and address) and their accounts.

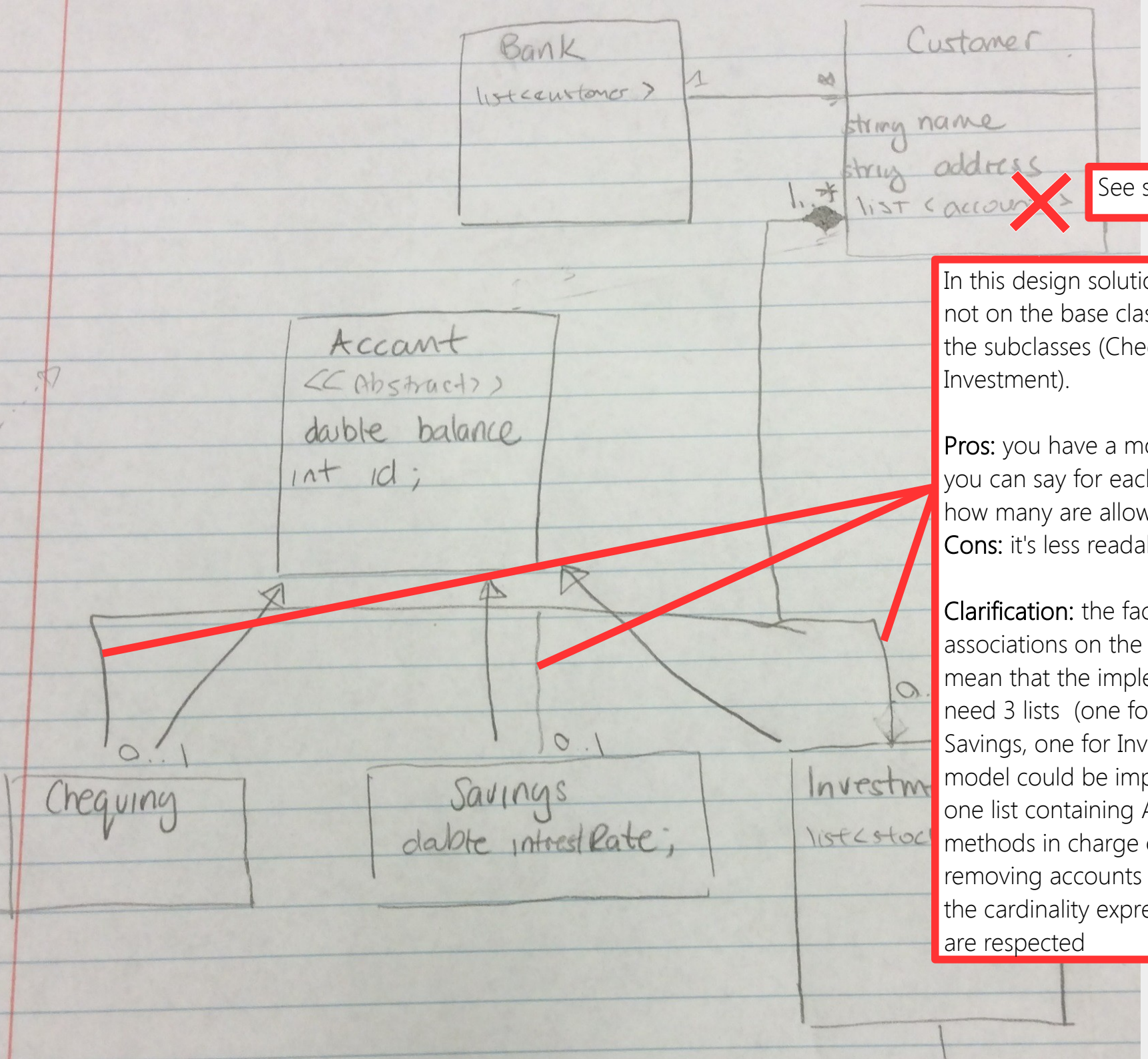
Each account has a balance and there are 3 type of accounts: checking for daily operations, saving which offers an interest rate and investments used to buy stocks. Stocks (identified by a ref) are bought at a certain quantity for a certain price (ticker).

Exercise 2 – Feedback



- Reminder: an inheritance arrow start from the subclass (child) toward the base class (parent)


Customer
name String
address String
1
0..3
<abstract>
Account
int Balance ;
Deposit ()
Withdrawl()
Clarification: here the cardinality of the association just says that a customer can have between 0 and 3 accounts but nothing on the type of account. Hence this doesn't express that a customer can have only 1 daily, 1 investment and 1 saving account for example
Daily
DailyOperation()
Investment
ArrayList<Stock> stocks;
buy Stock ()
Sell Stock()
Saving
int interest;
1
0..*
Stock
int price
Best practice #1: Use Associations for Classes and Attributes for primitive data types (String,Int,etc.)
In addition here the stocks attribute and the association between Investment and Stock are expressing pretty much the same thing: investment accounts have a list of stocks. Take a look back at how an association can be implemented in Java, most of the time if the cardinality is greater than 1 then it becomes a list.



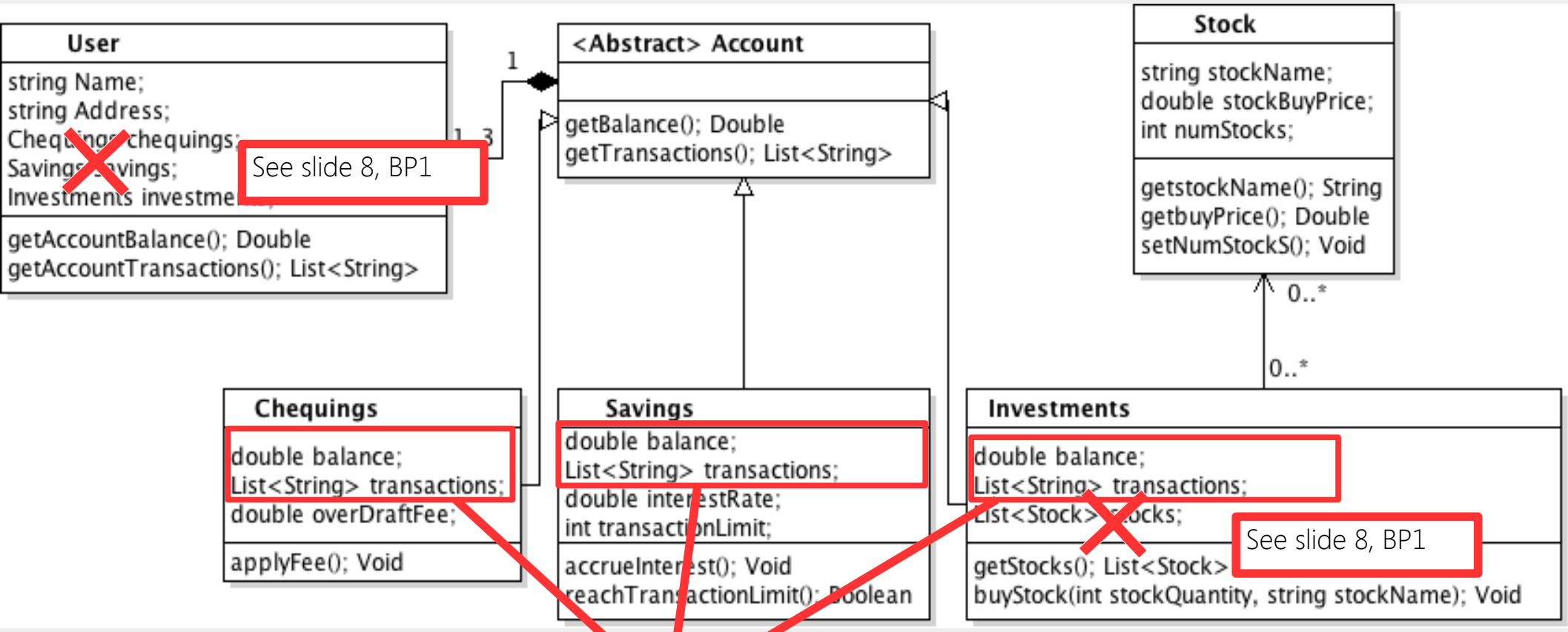
See slide 8, BP1

In this design solution the association is not on the base class (Account) but on the subclasses (Checking, Savings, Investment).

Pros: you have a more precise model, you can say for each type of accounts, how many are allowed

Cons: it's less readable and heavier

Clarification: the fact of using 3 associations on the model doesn't mean that the implementation would need 3 lists (one for Checking, one for Savings, one for Investment). This model could be implement with only one list containing Accounts and the methods in charge of adding and removing accounts would check that the cardinality expressed in the model are respected



In the bank system example, inheritance comes handy since we have common behaviors and attributes for Accounts that we want to reuse in several types of accounts. Here to take advantage of the inheritance mechanism common attributes (balance, transactions) and related methods need to be declared in the parent class Account.