ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Domain Model / Class Diagram
Custom Exceptions
Account App

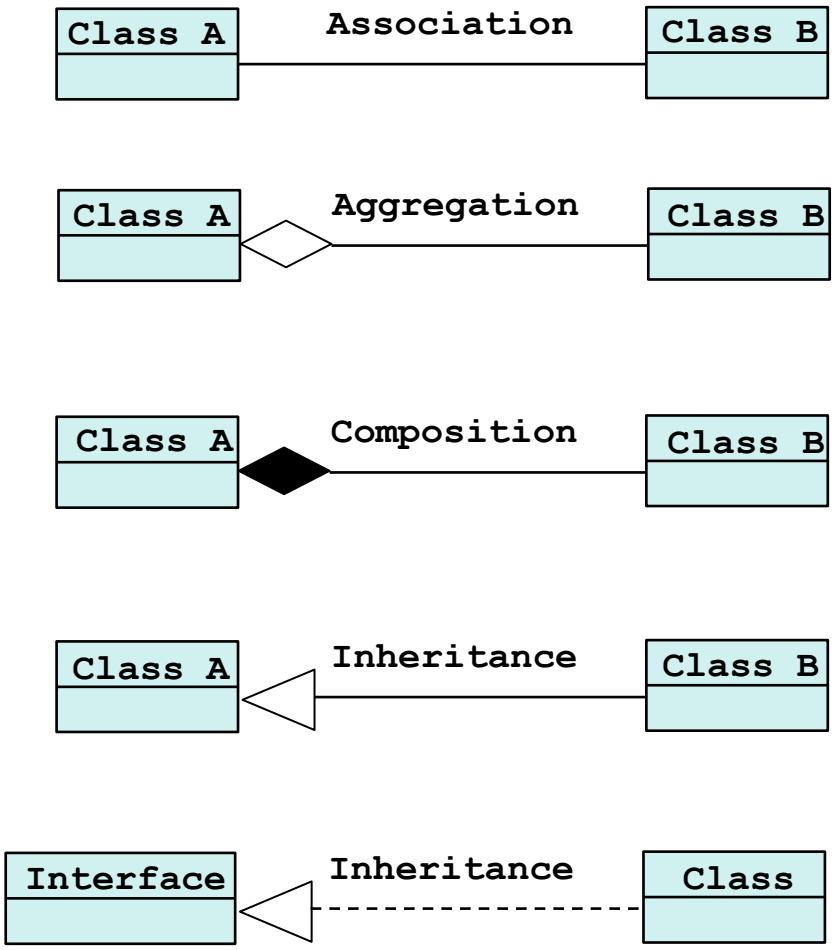
Αθ. Ανδρούτσος

Ανάλυση και Σχεδιασμός

- Κατά τη φάση του σχεδιασμού του μοντέλου δεδομένων μίας εφαρμογής τα εργαλεία που χρησιμοποιούμε στην αντικειμενοστραφή σχεδίαση είναι:

1. **Domain Model**. Όπως στις ΒΔ έχουμε ER Model, έτσι στον αντικειμενοστραφή σχεδιασμό έχουμε το Domain Model. Πρόκειται για **αποτύπωση των οντοτήτων του συστήματος (JavaBeans) και των σχέσεων μεταξύ τους.** Οι κλάσεις του Domain Model **έχουν μόνο τα σημαντικά πεδία και δεν περιέχουν μεθόδους.** Τα πεδία δεν χρειάζεται να έχουν access modifiers
2. **Class Diagram**. Είναι όπως το Domain Model αλλά περιέχει access modifiers, constructors, getters και setters, API

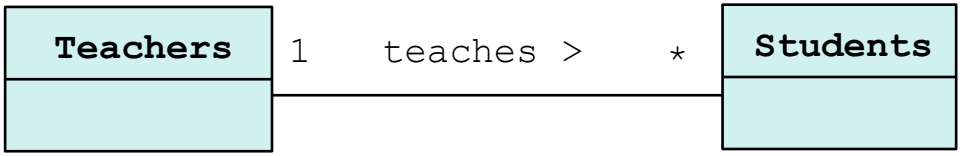
UML – Συσχετίσεις κλάσεων



- Στο **Association** η Class A σχετίζεται με την Class B (π.χ. Καθηγητές – Μαθητές, Πελάτες – Προϊόντα).
- Στο **Aggregation** η Class A περιλαμβάνει την Class B, η οποία είναι ***αυτόνομη*** **και μπορεί και υπάρχει και χωρίς** την Class A
- Στο **Composition** η Class A περιέχει την Class B, **η οποία *δεν μπορεί να υπάρχει χωρίς*** την Class A
- Στο **Inheritance** (Base Class A/Derived Class B) η Class B **κληρονομεί και επεκτείνει** την Class A
- Στο **Inheritance**, η class **υλοποιεί** το Interface

Association / Composition / Aggregation

- Στα association, composition, aggregation μπορεί να υπάρχει και cardinality, δηλαδή ο αριθμός των instances της κλάσης, όπως 1 ή πολλά. Το πολλά συμβολίζεται με *
- Με τα cardinalities μπορούμε να ορίσουμε multiplicity όπως π.χ. 1..* , 1..1, γενικά m..n
- Στο παρακάτω παράδειγμα, ένας Διδάσκων διδάσκει **σε 0 ή περισσότερους (*)** Μαθητές ενώ και ένας Μαθητής διδάσκεται από 1 Διδάσκων. Το αστεράκι (*) είναι ποσοδείκτης και σημαίνει «μηδέν ή περισσότερα instances»



Custom Exceptions

- Πρόκειται για exceptions που ορίζει ο προγραμματιστής (User-define) για να αντιμετωπίσει 'λογικά' σφάλματα της εφαρμογής, όπως για παράδειγμα Insufficient*BalanceException, UserNotFoundException, NegativeAmountException,* και άλλα παρόμοια
- Αντί λοιπόν να χρησιμοποιούμε τη γενική κλάση Exception (η οποία μπορεί να κάνει mask άλλα exceptions) χρησιμοποιούμε ειδικές user-defined κλάσεις για κάθε τύπο exception της εφαρμογής μας

Παράδειγμα

- Ας θεωρήσουμε **το παράδειγμα του τραπεζικού λογαριασμού *Account*** και του public API με τις μεθόδους *deposit*, *withdraw* και *getAccountBalance*
- Θα θέλαμε να χειριζόμαστε περιπτώσεις λογικών σφαλμάτων όπως την αντιμετώπιση ανεπαρκούς υπολοίπου κατά τη διαδικασία ανάληψης, ανεπαρκούς ποσού κατάθεσης και λανθασμένου SSN κατά τη διαδικασία ανάληψης
- Οπότε θα δημιουργήσουμε αντίστοιχα custom exceptions *InsufficientBalanceException*, *InsuffisientAmountException*, *SsnNotValidException*

Ανεπαρκές Υπόλοιπο


```java
package gr.aueb.cf.ch15.accounts.exceptions;

public class InsufficientBalanceException extends Exception {
    private static final long serialVersionUID = 5612658935L;

    public InsufficientBalanceException(double balance, double amount) {
        super("Insufficient balance " + balance + " for amount " + amount);
    }
}
```

- Όλα τα custom exceptions κληρονομούν από την κλάση **Exception**, η οποία είναι serializable (μπορεί να αποθηκευτεί σε αρχείο, θα δούμε στα επόμενα), οπότε χρειάζεται να ορίσουμε ένα *serialVersionUID*
- Ο constructor της custom exception καλεί τον constructor της Exception με super(String) και δημιουργεί ένα exception με κατάλληλη πληροφορία

Ανεπαρκές ποσό κατάθεσης


```java
package gr.aueb.cf.ch15.accounts.exceptions;

public class NegativeAmountException extends Exception {
    private static final long serialVersionUID = 15438972L;

    public NegativeAmountException() {}

    public NegativeAmountException(double amount) {
        super("Amount " + amount + " is insufficient");
    }
}
```

- Στα custom exception πρέπει να δίνουμε ένα ικανοποιητικό ποσό πληροφορίας. Το μήνυμα αυτό δεν είναι το τελικό μήνυμα που δίνουμε στον χρήστη.
- Περισσότερο είναι 'πληροφορία' προς τον client ώστε να δείξει αυτός με τη σειρά του κατάλληλο μήνυμα, το οποίο μπορεί να είναι και localized

Το SSN δεν είναι έγκυρο


```java
package gr.aueb.cf.ch15.accounts.exceptions;

public class SsnNotValidException extends Exception {
    private static final long serialVersionUID = 1234567L;

    public SsnNotValidException(String ssn) {
        super("SSN " + ssn + " is not valid");
    }
}
```

- Με την ίδια λογική δημιουργούμε και ένα άλλο exception SsnNotValidException

Deposit με Custom Exception


```java
/**
 * Deposits a certain amount of money into user-account,
 * thus increasing the {@link #balance}.
 *
 * @param amount
 *      the amount of money to be deposited.
 * @throws NegativeAmountException
 *      if the amount to be deposited is negative.
 */
public void deposit(double amount) throws NegativeAmountException {
    try {
        if (amount < 0) {
            throw new NegativeAmountException(amount);
        }
        balance += amount;
    } catch (NegativeAmountException e) {
        System.err.println("Error: Insufficient amount exception");
        throw e;
    }
}
```

- Στη συνέχεια μπορούμε να ορίσουμε την deposit η οποία κάνει throws ένα *NegativeAmountException*
- Ο τρόπος που το κάνει είναι κάνοντας try-catch για να χειριστεί το exception και μετά rethrow για τον caller

Withdraw με Exceptions


```java
/**
 * Withdraws a certain amount of money from the account.
 * It validates the ssn to be equal to {@link #holder} ssn.
 * @see User#ssn
 *
 * @param amount
 *      the amount of money to be withdrawn.
 * @param ssn
 *      the given ssn.
 * @throws InsufficientBalanceException
 *      if the {@link #balance} is not sufficient.
 * @throws SsnNotValidException
 *      if the {@link User#ssn} is not equal to given ssn.
 */
public void withdraw(double amount, String ssn) throws
        InsufficientBalanceException, SsnNotValidException {
    try {
        if (!isSsnValid(ssn)) {
            throw new SsnNotValidException(ssn);
        }

        if (amount > balance) {
            throw new InsufficientBalanceException(balance, amount);
        }
        balance -= amount;
    } catch (InsufficientBalanceException | SsnNotValidException e) {
        System.err.println("Error in withdraw" + "\n" + e);
        throw e;
    }
}
```

- Όπως πριν, try-catch και rethrow
- Όπως έχουμε ξαναπεί ελέγχουμε στην αρχή του κώδικα για error conditions που δίνουν exceptions

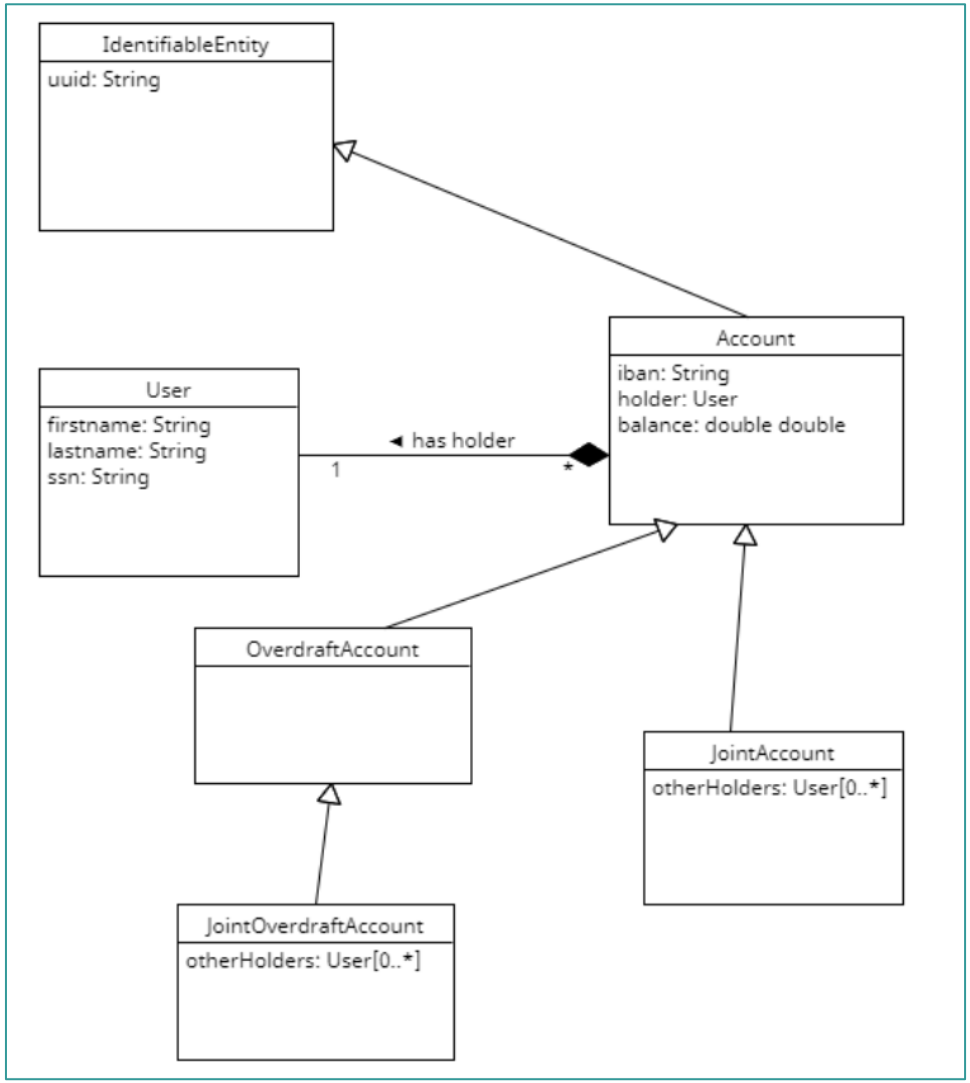
Εφαρμογή Κληρονομικότητας

- Ένα use case όπου μπορούμε να εφαρμόσουμε κληρονομικότητα είναι μέσα σε ένα δικό μας package στο πλαίσιο της δημιουργίας μιας σειράς κλάσεων όπως *Account*, *OverdraftAccount*, *JointAccount* και *OverdraftJointAccount*

Domain Model (1)

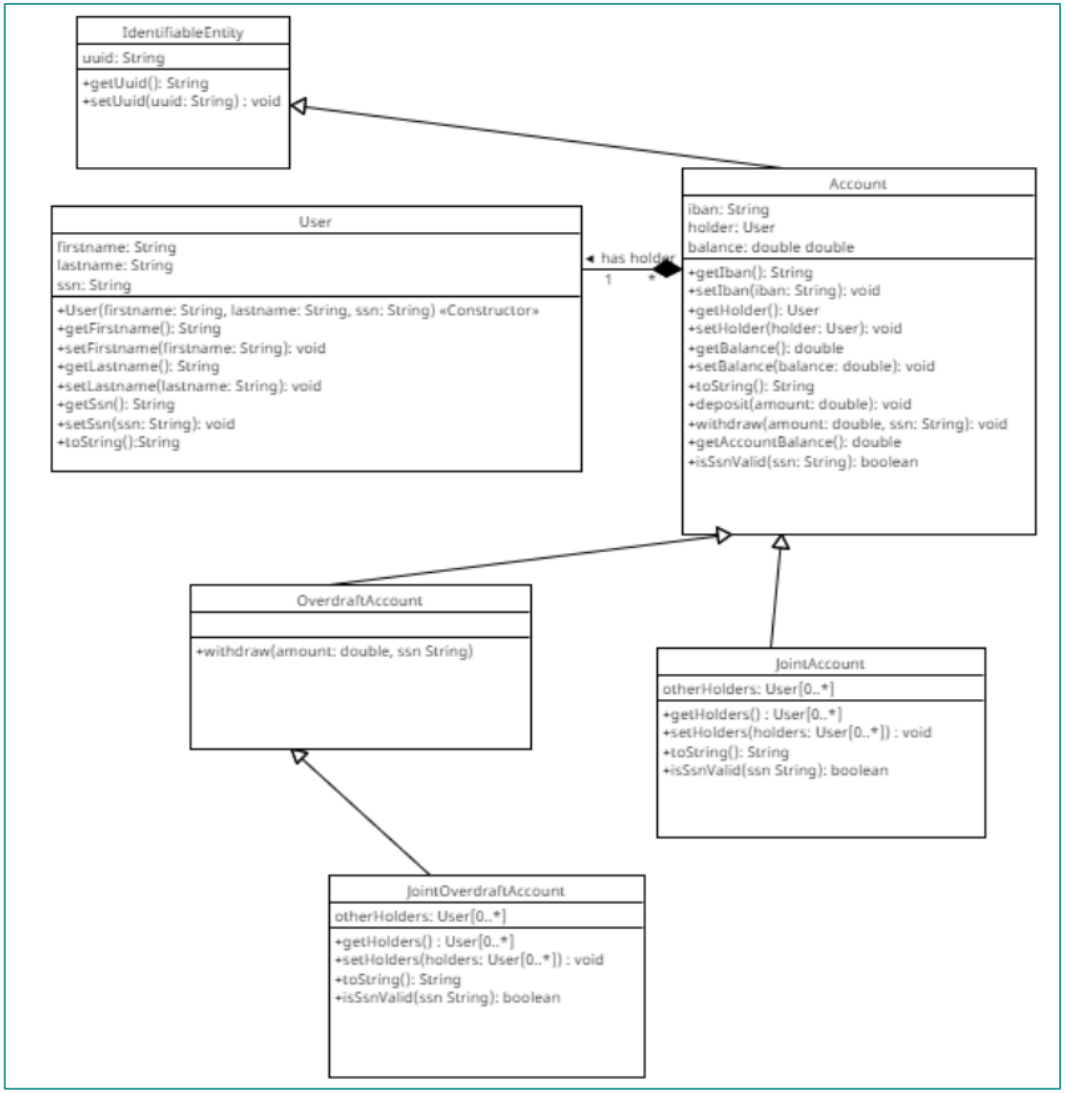
- Θα δημιουργήσουμε ένα Domain Model της εφαρμογής Account
- Οι classes απεικονίζονται στο Domain Model όπως έχουμε αναφέρει, με απλοποιημένο τρόπο
- Περιέχουν μόνο το βασικό state της κλάσης, δηλαδή τα fields χωρίς visibility modifiers (+, -, #, ~)

Domain Model (2)



- Η σχέση με το μαύρο διαμάντι στην UML δηλώνει composition
- Ενώ υπάρχουν και δύο κλάσεις, OverdraftAccount και OverdraftJointAccount που κληρονομούν
- Το JointAccount στο παρόν δεν θεωρείται ότι is-a Account μιας και πρόκειται για ένα Account με δύο κατόχους. Οπότε είναι composition του User.

Class Diagram



- Το class diagram περιέχει data με access modifiers καθώς και μεθόδους

Παράδειγμα (1)

- Όλες οι οντότητες Account, Overdraft Account, Joint Account, Overdraft Joint Account, **έχουν κοινά πεδία όπως id ή uuid, και τα στοιχεία του χρήστη**
- Ειδικά το *id/uuid* είναι το πιο γενικό πεδίο, ενώ τα *στοιχεία του χρήστη* είναι πιο ειδικά, επίσης κοινά πεδία, που αναφέρονται σε Accounts που αντιστοιχούν σε χρήστες (ενώ μπορεί να υπάρχουν και άλλου τύπου λογαριασμοί, όπως εταιρικοί, κλπ.)

Identifiable Entity


```java
package gr.aueb.cf.ch15.accounts.model;

import java.util.UUID;

public class IdentifiableEntity {
    private String uuid;

    public IdentifiableEntity() {
        uuid = UUID.randomUUID().toString();
    }

    public String getUuid() {
        return uuid;
    }

    public void setUuid(String uuid) {
        this.uuid = uuid;
    }
}
```

- Δημιουργούμε μία γενική κλάση IdentifiableEntity που περιέχει μόνο το id μιας και όλες οι κλάσεις περιέχουν id

User


```java
package gr.aueb.cf.ch15.accounts.model;

public class User extends IdentifiableEntity {
    private String firstname;
    private String lastname;
    private String ssn;

    public User() {}

    public User(String firstname, String lastname, String ssn) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.ssn = ssn;
    }

    // Copy constructor
    public User(User user) {
        firstname = user.firstname;
        lastname = user.lastname;
        ssn = user.ssn;
    }
```

- Ο User ***is-a*** IdentifiableEntity και επίσης ορίζει και πεδία που είναι κοινά στα User Accounts: firstname, lastname, και ssn
- Επίσης, ορίζουμε ένα copy constructor για λόγους που θα εξηγήσουμε παρακάτω

Copy Constructor


```
16      // Copy Constructor
17  @   public User(User user) {
18          firstname = user.getFirstname();
19          lastname = user.getLastname();
20          ssn = user.getSsn();
21
```

- Ο Copy Constructor αντιγράφει το state ενός object. Παρατηρήστε ότι κάνουμε inject ένα object ως παράμετρο και στη συνέχεια αντιγράφουμε ένα-ένα τα πεδία του
- Θυμίζουμε ότι ο στόχος είναι να μπορεί η αναφορά σε ένα copy να επηρεάζει τα πεδία του άλλου copy. Εδώ ως πεδία έχουμε Strings που είναι immutable, οπότε το copy παρόλο που αντιγράφουμε τις αναφορές στα Strings δεν μπορεί το ένα copy να αλλάξει τα πεδία του άλλου copy (αφού τα Strings είναι immutable).

Υπόλοιπες μέθοδοι User class


```
24      public String getFirstname() { return firstname; }
27      public void setFirstname(String firstname) { this.firstname = firstname; }
30      public String getLastname() { return lastname; }
33      public void setLastname(String lastname) { this.lastname = lastname; }
36      public String getSsn() { return ssn; }
39      public void setSsn(String ssn) { this.ssn = ssn; }
42
43      @Override
44      public String toString() {
45          return "User{" +
46                  "firstname='" + firstname + '\'' +
47                  ", lastname='" + lastname + '\'' +
48                  ", ssn='" + ssn + '\'' +
49                  '}';
50      }
51  }
```

- Setters / getters / toString

Account class (1)


```java
package gr.aueb.cf.ch15.accounts.model;

import gr.aueb.cf.ch15.accounts.exceptions.InsufficientBalanceException;
import gr.aueb.cf.ch15.accounts.exceptions.NegativeAmountException;
import gr.aueb.cf.ch15.accounts.exceptions.SsnNotValidException;

import java.time.LocalDateTime;

public class Account extends IdentifiableEntity {
    private String iban;
    private User holder;
    private double balance;

    public Account() {

    }

    public Account(String iban, User holder, double balance) {
        this.iban = iban;
        this.holder = holder;
        this.balance = balance;
    }
```

- Η κλάση Account κάνει extends την *IdentifiableEntity* και ορίζει τρία πεδία ένα εκ των οποίων είναι ένα private User instance (composition)

Account class (2)


```
34      public void setIban(String iban) { this.iban = iban; }
37      public User getHolder() { return new User(holder); }
40      public void setHolder(User holder) { this.holder = new User(holder); }
43      public double getBalance() { return balance; }
46      public void setBalance(double balance) { this.balance = balance; }
```

- Παρατηρήστε ότι ο getHolder επιστρέφει ένα copy του holder με τον copy constructor του User, όχι μία απλή αναφορά
- Εδώ είναι απαραίτητο γιατί ο User περιέχει μεν String πεδία, που είναι immutable αλλά δεν είναι immutable συνολικά η κλάση (παρέχει setters)
- Το ίδιο ισχύει και για τον setter

Account class - deposit


```java
50      /**
51       * Deposits a certain amount of money into user-account,
52       * thus increasing the {@link #balance}.
53       *
54       * @param amount
55       *      the amount of money to be deposited.
56       * @throws NegativeAmountException
57       *      if the amount to be deposited is negative.
58      public void deposit(double amount) throws NegativeAmountException {
59          try {
60              if (amount < 0) {
61                  throw new NegativeAmountException(amount);
62              }
63              balance += amount;
64              System.out.println("Balance deposit: " + holder + ", amount: "
65                      + amount + ", " + LocalDateTime.now());
66          } catch (NegativeAmountException e) {
67              System.err.println("Service Error: Negative amount exception");
68              throw e;
69          }
      }
```

- Παρατηρούμε ότι έχουμε user-defined exceptions που είναι ο γενικός τρόπος χειρισμού λογικών λαθών

Account class - withdraw


```java
/**
 * Withdraws a certain amount of money from the account.
 * It validates the ssn to be equal to {@link #holder} ssn.
 * @see User#ssn
 *
 * @param amount
 *      the amount of money to be withdrawn.
 * @param ssn
 *      the given ssn.
 * @throws InsufficientBalanceException
 *      if the {@link #balance} is not sufficient.
 * @throws SsnNotValidException
 *      if the {@link User#ssn} is not equal to given ssn.
 */
public void withdraw(double amount, String ssn)
        throws InsufficientBalanceException, SsnNotValidException {

    try {
        if (!isSsnValid(ssn)) {
            throw new SsnNotValidException(ssn);
        }

        if (amount > balance) {
            throw new InsufficientBalanceException(balance, amount);
        }
        balance -= amount;
        System.out.println("Balance withdraw: " + ssn + ", amount:  " + amount + ", "
                + LocalDateTime.now());
    } catch (InsufficientBalanceException | SsnNotValidException e) {
        System.err.println("Service Error: " + e.getMessage());
        throw e;
    }
}
```

Account – Υπόλοιπες Μέθοδοι


```
99
100     public double getAccountBalance() {
101         return balance;
102     }
103
104     protected boolean isSsnValid(String ssn) {
105         if (ssn == null || getHolder().getSsn() == null) return false;
106
107
108         return this.holder.getSsn().equals(ssn);
109     }
110
111     @Override
112     public String toString() {
115         return "Account{" +
116                 "uuid=" + getUuid() +
117                 ", iban='" + iban + '\'' +
118                 ", holder=" + holder +
119                 ", balance=" + balance +
120                 '}';
121     }
122 }
123
124
```

- Παρατηρήστε ότι η *ssnIsValid* είναι protected μιας και η Account θα κληρονομηθεί από την OverdraftAccount κλπ
- Είμαστε στο ίδιο package και γράφουμε εμείς και τις superclasses και τις subclasses και επομένως το self-use της withdraw και isSsnValid μπορούμε να το χειριστούμε εφόσον το γνωρίζουμε

OverdraftAccount class


```java
package gr.aueb.cf.ch15.accounts.model;

import gr.aueb.cf.ch15.accounts.exceptions.SsnNotValidException;

public class OverdraftAccount extends Account {

    public OverdraftAccount() {
    }

    public OverdraftAccount(String iban, User holder, double balance) {
        super(iban, holder, balance);
    }

    @Override
    public void withdraw(double amount, String ssn) throws SsnNotValidException {
        try {
            if (!isSsnValid(ssn)) {
                throw new SsnNotValidException(ssn);
            }
            setBalance(getBalance() - amount);
        } catch (SsnNotValidException e) {
            System.err.println("Error " + e.getMessage());
            throw e;
        }
    }
```

- Η Overdraft Account κληρονομεί την Account και κάνει override την withdraw
- Η withdraw υλοποιεί την λογική της ανάληψης ποσών μεγαλύτερων από το balance

JointAccount class


```java
package gr.aueb.cf.ch15.accounts.model;

import java.util.Arrays;

public class JointAccount extends Account {
    private User[] otherHolders;

    public JointAccount() {

    }

    public JointAccount(String iban, User holder, double balance, User[] otherHolders) {
        super(iban, holder, balance);
        this.otherHolders = otherHolders;
    }

    public User[] getOtherHolders() { return otherHolders; }

    public void setOtherHolders(User[] otherHolders) { this.otherHolders = otherHolders; }
```

- Κληρονομεί από την Account και περιλαμβάνει και ένα πίνακα από holders

JointAccount - isSsnValid


```
25      @Override
26      protected boolean isSsnValid(String ssn) {
27          super.isSsnValid(ssn);
28          for (User user : otherHolders) {
29              if (user.getSsn().equals(ssn)) {
30                  return true;
31              }
32          }
33          return false;
34      }
35
36      @Override
37      public String toString() {
38          return "JointAccount{" +
39                  "uuid: " + getUuid() +
40                  ", otherHolders=" + Arrays.toString(otherHolders) +
41                  ", balance: " + getBalance() +
42                  '}';
43      }
44  }
```

- Η isSsnValid ελέγχει πλέον και τον πίνακα των holders

OverdraftJointAccount (1)


```java
package gr.aueb.cf.ch15.accounts.model;

import java.util.Arrays;

public class OverdraftJointAccount extends OverdraftAccount {
    private User[] otherHolders;

    public OverdraftJointAccount() {

    }

    public OverdraftJointAccount(String iban, User holder, double balance, User[] otherHolders) {
        super(iban, holder, balance);
        this.otherHolders = otherHolders;
    }

    public User[] getOtherHolders() { return otherHolders; }

    public void setOtherHolders(User[] otherHolders) { this.otherHolders = otherHolders; }
```

- Κληρονομεί από την *OverdraftAccount (δεν μπορεί να κληρονομήσει και από τον Joint Account, στην Java δεν υπάρχει πολλαπλή κληρονομικότητα).* Κάνει override την isSsnNotValid()

OverdraftJointAccount (2)


```
25      @Override
26      protected boolean isSsnValid(String ssn) {
27          for (User user : otherHolders) {
28              if (!user.getSsn().equals(ssn)) {
29                  return false;
30              }
31          }
32          return true;
33      }
34
35      @Override
36      public String toString() {
37          return "OverdraftJointAccount{" +
38                  "otherHolders=" + Arrays.toString(otherHolders) +
39                  '}';
40      }
41  }
42
```