# Lab session 3: Implementing inheritance
# 24292-Object Oriented Programming

## 1 Introduction

The aim of this lab session is to implement the design of Seminar 3 that partially models an online store. The session is mandatory and you have to deliver the source code of the Java project and a document describing the implementation decision and details. And remember to print information at each step like sell, buy, make bid and so on.

## 2 OnlineStore

Start a new Netbeans Java project called OnlineStore. Leave the option "Create main class" selected, as this is not a GUI project.

In the main method of the online store, you will have to declare a list of users, items and packages. Create instances for each one of them, buy items, sell items, expel users, and calculate total price and benefit of the application.

The figure on the next page shows the complete class diagram and a possible solution to Seminar 3.

Declare two variables *totalPrice* and *totalProfit*, whose values are initially zero. Each time an item is bought, remove the item from the list of items being sold, and update *totalPrice* and *totalProfit* to reflect the earnings of the store.

## 3 Items

First implement the hierarchy of items from the seminar. A sample solution appears on the next page (optionally, components in italics can be declared as abstract):

In the main method, declare a list whose elements are of type Item. This list will store all items being sold by the store. Create instances of items and add those items to the list.

Add two static final variables in AuctionItem class, one for the fixed fee of 5 euros the online store charges to these items and a percentage of 5 percent from the total price in case its sold.

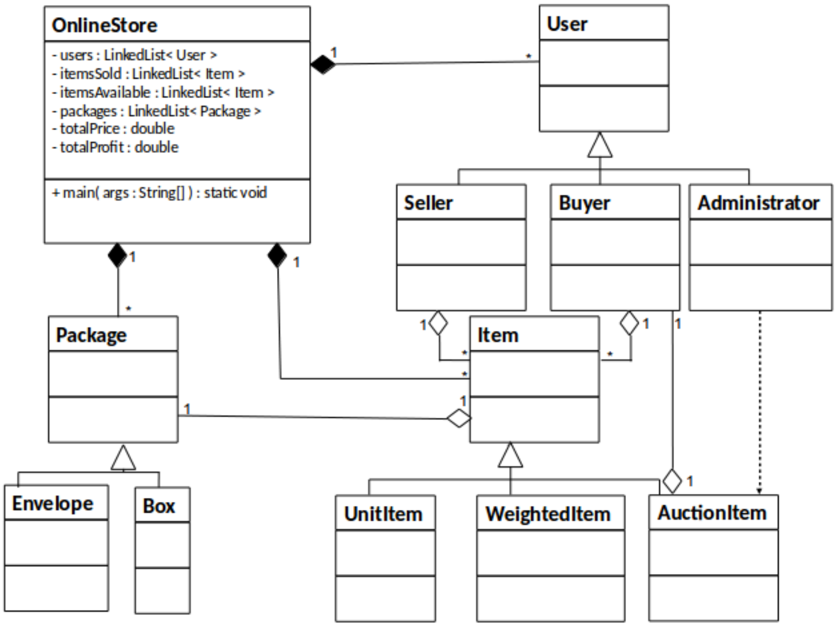
OnlineStore
- users : LinkedList< User >
- itemsSold : LinkedList< Item >
- itemsAvailable : LinkedList< Item >
- packages : LinkedList< Package >
- totalPrice : double
- totalProfit : double
+ main( args : String[] ) : static void
User
Seller
Buyer
Administrator
Package
Item
Envelope
Box
UnitItem
WeightedItem
AuctionItem
1
*
1
*
1
*
1
*
1
*
1
1
1
1

Figure 1: Class diagram for the store

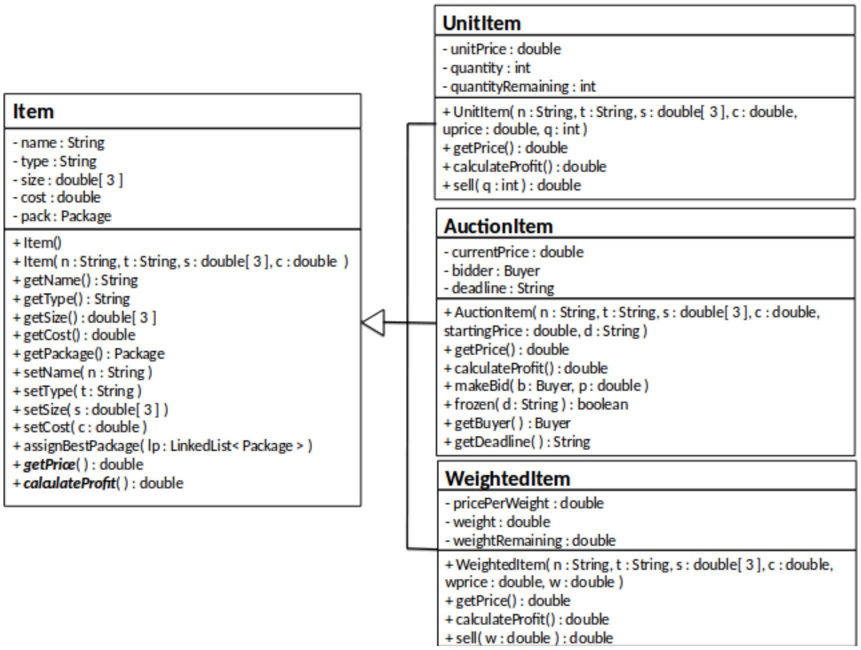
Item
- name : String
- type : String
- size : double[ 3 ]
- cost : double
- pack : Package
+ Item()
+ Item( n : String, t : String, s : double[ 3 ], c : double )
+ getName() : String
+ getType() : String
+ getSize() : double[ 3 ]
+ getCost() : double
+ getPackage() : Package
+ setName( n : String )
+ setType( t : String )
+ setSize( s : double[ 3 ] )
+ setCost( c : double )
+ assignBestPackage( lp : LinkedList< Package > )
+ getPrice( ) : double
+ calculateProfit( ) : double
UnitItem
- unitPrice : double
- quantity : int
- quantityRemaining : int
+ UnitItem( n : String, t : String, s : double[ 3 ], c : double, uprice : double, q : int )
+ getPrice() : double
+ calculateProfit() : double
+ sell( q : int ) : double
AuctionItem
- currentPrice : double
- bidder : Buyer
- deadline : String
+ AuctionItem( n : String, t : String, s : double[ 3 ], c : double, startingPrice : double, d : String )
+ getPrice() : double
+ calculateProfit() : double
+ makeBid( b : Buyer, p : double )
+ frozen( d : String ) : boolean
+ getBuyer( ) : Buyer
+ getDeadline( ) : String
WeightedItem
- pricePerWeight : double
- weight : double
- weightRemaining : double
+ WeightedItem( n : String, t : String, s : double[ 3 ], c : double, wprice : double, w : double )
+ getPrice() : double
+ calculateProfit() : double
+ sell( w : double ) : double

Figure 2: Class diagram for items

# 4 Packaging

In order to deliver items from our online store to our buyers we need packages. Item class requires a method to select the best available package based on two conditions. The first one is if the depth of the item is less than 3cm then use an envelope, or a box otherwise. The second one is to select the best envelope or box based on the item size (the one where the item fits best the volum).

Below is a possible hierarchy design for packages. Create the classes and implement them, and do not forget to use this hierarchy from the Item class.

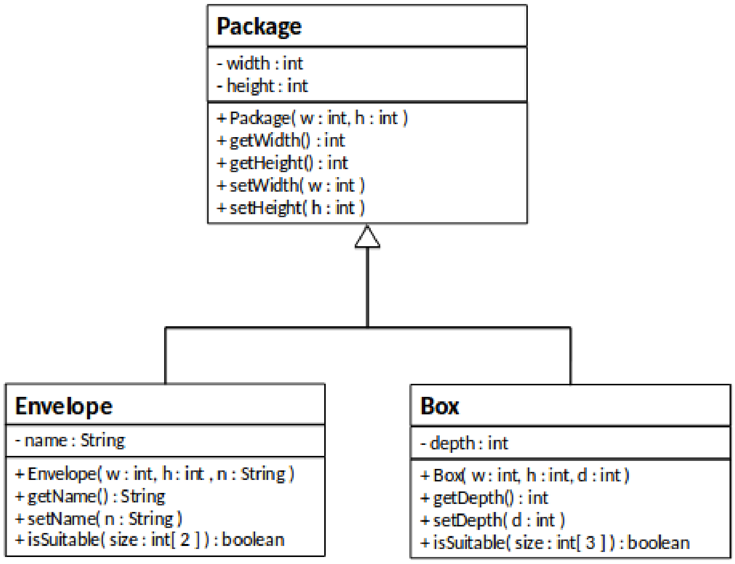

Figure 3: Class diagram for packages

At this point test for multiple items you are selecting the correct packages.

# 5 Users

As a next step, implement the hierarchy of users from the seminar. A possible design of this hierarchy is shown on the next page.

You should now be able to simulate items being bought (and sold, in the case of AuctionItems). Test the methods for buying and selling items, and the method makeBid of an AuctionItem for raising the current bid.

# 6 Submission

The deadline for this lab session 3 is before next lab session 4. Do not forget to write a report explaining your code, ideas, problems and how you solved them.

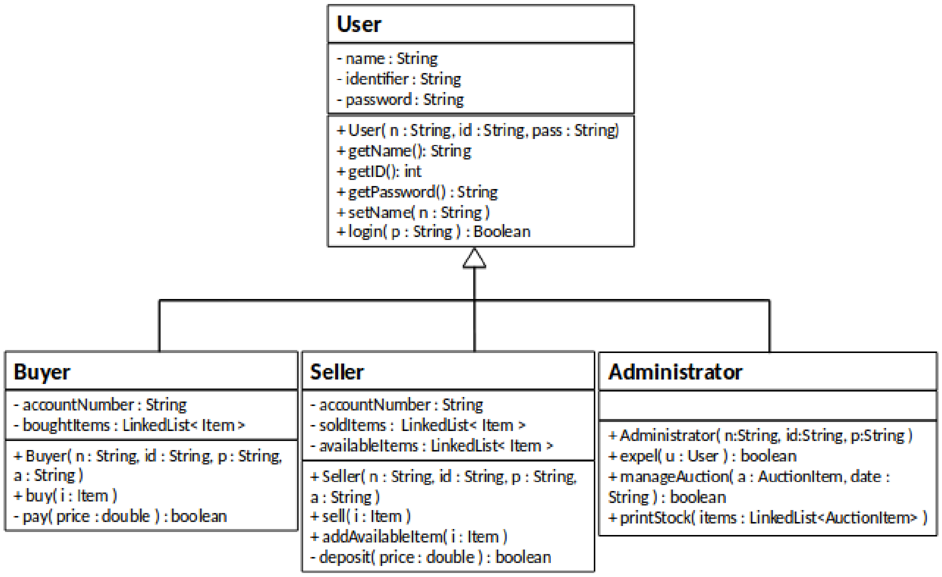
User
- name : String
- identifier : String
- password : String
+ User( n : String, id : String, pass : String)
+ getName(): String
+ getID(): int
+ getPassword() : String
+ setName( n : String )
+ login( p : String ) : Boolean
Buyer
- accountNumber : String
- boughtItems : LinkedList< Item >
+ Buyer( n : String, id : String, p : String, a : String )
+ buy( i : Item )
- pay( price : double ) : boolean
Seller
- accountNumber : String
- soldItems : LinkedList< Item >
- availableItems : LinkedList< Item >
+ Seller( n : String, id : String, p : String, a : String )
+ sell( i : Item )
+ addAvailableItem( i : Item )
- deposit( price : double ) : boolean
Administrator
+ Administrator( n:String, id:String, p:String )
+ expel( u : User ) : boolean
+ manageAuction( a : AuctionItem, date : String ) : boolean
+ printStock( items : LinkedList<AuctionItem> )

Figure 4: Class diagram for users