# DD1339 Introduktion till datalogi 2013/2014

# Uppgift nummer: 5

# Namn: Marcus Larsson

# Grupp nummer: 5

# Övningsledare: Marcus Dicander

# Betyg: ..... Datum: .............. Rättad av: .......................................

# Exercise 4.40-4.42 och 4.54-4.55

import java.util.ArrayList;

/\*\*

\* Store details of club memberships.

\*

\* @author (Marcus Larsson)

\* @version (2013.10.15)

\*/

public class Club

{

// Define any necessary fields here ...

private ArrayList<Membership> members;

/\*\*

\* Constructor for objects of class Club

\*/

public Club()

{

// Initialise any fields here ...

members = new ArrayList<Membership>();

}

/\*\*

\* Add a new member to the club's list of members.

\* @param member The member object to be added.

\*/

public void join(Membership member)

{

members.add(member);

}

/\*\*

\* @return The number of members (Membership objects) in

\* the club.

\*/

public int numberOfMembers()

{

return members.size();

}

/\*\*

\* Determine the number of members who joined in the

\* given month.

\* @param month The month we are interested in.

\* @return The number of members who joined in that month.

\*/

public int joinedInMonth(int month){

int numberOfMembers = 0;

if(month>1&&month<12){

for(Membership current:members){

if(current.getMonth()==month){

numberOfMembers += 1;

}

}

} else {

System.out.println("Not a valid month. Please enter numbers between 1 and 12.");

numberOfMembers = 0;

}

return numberOfMembers;

}

/\*\*

\* Remove from the club's collection all members who

\* joined in the given month, and return them stored

\* in a separate collection object.

\* @param month The month of the membership.

\* @param year The year of the membership.

\* @return The members who joined in the given month and year.

\*/

public ArrayList<Membership> purge(int month, int year){

ArrayList<Membership> temp = new ArrayList<Membership>();

if(month>1&&month<12){

for(Membership current:members){

if(current.getMonth()==month&&current.getYear()==year){

//do nothing if objects with the specified info is found.

} else {

temp.add(current);

}

}

} else {

System.out.println("Not a valid month. Please enter numbers between 1 and 12.");

}

return temp;

}

}

# Exercise 4.48-4.52 (se 4.50 under koden)

import java.util.ArrayList;

/\*\*

\* A simple model of an auction.

\* The auction maintains a list of lots of arbitrary length.

\*

\* @author David J. Barnes and Michael Kölling.

\* @version 2011.07.31

\*/

public class Auction

{

// The list of Lots in this auction.

private ArrayList<Lot> lots;

// The number that will be given to the next lot entered

// into this auction.

private int nextLotNumber;

/\*\*

\* Create a new auction.

\*/

public Auction()

{

lots = new ArrayList<Lot>();

nextLotNumber = 1;

}

/\*\*

\* Enter a new lot into the auction.

\* @param description A description of the lot.

\*/

public void enterLot(String description)

{

lots.add(new Lot(nextLotNumber, description));

nextLotNumber++;

}

/\*\*

\* Show the full list of lots in this auction.

\*/

public void showLots()

{

for(Lot lot : lots) {

System.out.println(lot.toString());

}

}

/\*\*

\* Make a bid for a lot.

\* A message is printed indicating whether the bid is

\* successful or not.

\*

\* @param lotNumber The lot being bid for.

\* @param bidder The person bidding for the lot.

\* @param value The value of the bid.

\*/

public void makeABid(int lotNumber, Person bidder, long value)

{

Lot selectedLot = getLot(lotNumber);

if(selectedLot != null) {

Bid bid = new Bid(bidder, value);

boolean successful = selectedLot.bidFor(bid);

if(successful) {

System.out.println("The bid for lot number " +

lotNumber + " was successful.");

}

else {

// Report which bid is higher.

Bid highestBid = selectedLot.getHighestBid();

System.out.println("Lot number: " + lotNumber +

" already has a bid of: " +

highestBid.getValue());

}

}

}

/\*\*

\* Return the lot with the given number. Return null

\* if a lot with this number does not exist.

\* @param lotNumber The number of the lot to return.

\*/

public Lot getLot(int lotNumber)

{

String errorMessage = "Lot number: " + lotNumber +

" does not exist.";

for(Lot lot : lots){

if(lot.getNumber()== lotNumber) {

return lot;

} else if(lot.getNumber()>lotNumber) {

System.out.println(errorMessage);

return null;

}

}

System.out.println(errorMessage);

return null;

}

/\*\*

\* This method will iterate over all lots and print out what's sold and what's not.

\*/

public void close()

{

for(Lot lot : lots) {

System.out.println(lot.getNumber() + ": " + lot.getDescription());

Bid highestBid = lot.getHighestBid();

if(highestBid != null) {

System.out.println(" Sold!");

System.out.println(" Highest bidder: " +

highestBid.getBidder().getName());

System.out.println(" Bid: " +

highestBid.getValue());

}

else {

System.out.println(" Not sold");

}

}

}

/\*\*

\* Returns a new list with all unsold lots.

\* @return An ArrayList of all unsold lots.

\*/

public ArrayList<Lot> getUnsold(){

ArrayList<Lot> temp = new ArrayList<Lot>();

for(Lot lot : lots){

Bid highestBid = lot.getHighestBid();

if(highestBid == null){

temp.add(lot);

}

}

return temp;

}

/\*\*

\* Remove the lot with the given lot number.

\* @param number The number of the lot to be removed.

\* @return The Lot with the given number, or null if

\* there is no such lot.

\*/

public Lot removeLot(int number){

String errorMessage = "Lot number: " + number +

" does not exist.";

for(Lot lot : lots){

if(lot.getNumber()== number) {

lots.remove(lot);

return lot;

} else if(lot.getNumber()>number) {

System.out.println(errorMessage);

return null;

}

}

System.out.println(errorMessage);

return null;

}

}

# Exercise 4.50

getLot metoden litar på att en Lot är lagrad i listan på platsen getLotNumber()-1. Om man tar bort objekt från en ArrayList så flyttas alla index som var större än det index man tog bort. För att fylla upp tomrummet. Detta gör att alla objekt efter det som togs bort stämmer inte överens med getLotNumber()-1 längre.

# Exercise 4.56 – 4.59

import java.util.ArrayList;

/\*\*

\* Manage the stock in a business.

\* The stock is described by zero or more Products.

\*

\* @author (Marcus)

\* @version (2013-10-09)

\*/

public class StockManager

{

// A list of the products.

private ArrayList<Product> stock;

/\*\*

\* Initialise the stock manager.

\*/

public StockManager()

{

stock = new ArrayList<Product>();

}

/\*\*

\* Add a product to the list.

\* @param item The item to be added.

\*/

public void addProduct(Product item)

{

stock.add(item);

}

/\*\*

\* Receive a delivery of a particular product.

\* Increase the quantity of the product by the given amount.

\* @param id The ID of the product.

\* @param amount The amount to increase the quantity by.

\*/

public void delivery(int id, int amount)

{

Product product = findProduct(id);

if (product!=null){

product.increaseQuantity(amount);

} else{

System.out.println("Product id: "+ id +" does not exist.");

}

}

/\*\*

\* Try to find a product in the stock with the given id.

\* @return The identified product, or null if there is none

\* with a matching ID.

\*/

public Product findProduct(int id)

{

for(Product product:stock){

if(product.getID()==id){

return product;

}

}

return null;

}

/\*\*

\* Locate a product with the given ID, and return how

\* many of this item are in stock. If the ID does not

\* match any product, return zero.

\* @param id The ID of the product.

\* @return The quantity of the given product in stock.

\*/

public int numberInStock(int id)

{

Product product = findProduct(id);

if (product!=null){

return product.getQuantity();

}

return 0;

}

/\*\*

\* Print details of all the products.

\*/

public void printProductDetails()

{

for(Product product:stock){

System.out.println(product);

}

}

}