System.out.println("INSTRUCTIONS:");

System.out.println("Please select what you want to do...\n(1)customer management(2)inventory management(3)checkout customer(4)quit");

/\* System.out.println("Welome to the sample store interface. Using this system you can work with the provided customers and \n"+

"carts to help assist with Books, Music and DVDs purchases. There is a capacity of shopping carts and \n" +

"customers that is set to 75 to accomodate the uses of a small outlet for these items. Should you need \n" +

"increased system capacity to accomodate more carts or customers that must be done by a programmer.");

// createSampleStore(MAX\_CAPACITY); \*/

//A loop for the main menu

switch(input.next()) {

case("1"):

}

while(true) {

System.out.println("\nSTORE MENU");

System.out.print("(1) Assist Customers (e.g. add/remove items to cart)\n(2) View customer cart \n(3) Add more customers (is rarely needed)\n(4) Use checkout counter\n(5) Show all customers \n(6) Show active shopping carts \n(7) Shopping cart interface (preview...not functional)");

//The switch statement provides navigation each case relying on object and instant methods

switch(input.next()) {

case("7"):

//Menu option for cart interface (TBD for later)

System.out.println("\nCART TRACKER:");

System.out.println("\n\nThis is just preview sample of an interface for carts (to add to later).");

showCartArray(carts);

//break statement is left out in order to proceed to the list of active shopping carts

case("6"):

//Menu option for the viewing only of filled shopping carts

try {

showCustArray(customers, "shopping", "cart"); }

catch (Exception e ) {break;}

break;

case("5"):

//

showCustArray(customers);

break;

case("4"):

showCustArray(customers, "line");

try {

System.out.println("Enter custID to checkout.");

custID = input.next();

Customer cu = assignToCustomer(customers, custID);

cu.removeCart();

customers[getCustomerIndex(customers, custID)] = new Customer(CustomerID.generate());

System.out.println("A customer has exited the store following checkout.");

showCustArray(customers);

}

catch (Exception e)

{

break;}

break;

case("3"):

System.out.println("STORE MENU: CUSTOMER ADD");

int cust = getCustomerCount(customers);

System.out.println("There are currently "+ cust + " shoppers in the store, and " + (carts.length - cust) + " carts available.");

System.out.println("Would you like to add 1 additional customer? (y/n)");

if(input.next().toUpperCase().matches("Y"))

{

int i = getLastCustomerIndex(customers);

System.out.println("Press yes again to auto assign a unique id");

customers[i]= new Customer(CustomerID.generate());

//customers[i].assignCart();

}

break;

case ("2"):

try {

System.out.println("Customer listing");

showCustArray(customers,"with","carts");

System.out.println("STORE MENU: PURCHASE TRACKER");

System.out.println("Enter a Customer ID: ");

custID = input.next();

showCustArray(customers, custID);

// showCustArray(customers, getCustomerIndex(customers, custID));

}

catch (Exception e)

{

break;}

break;

case("1"):

System.out.println("\nSTORE MENU: CUSTOMER TRACKER");

showCustArray(customers,"with","CutsomerID's","only");

System.out.println("Enter a Customer ID from above (must enter ALL 5 digits): ");

System.out.println("(You may recall 75 customers have already been generated automatically.)");

custID = input.next();

//user input is used to search the array of objects

try {

Customer obj = assignToCustomer(customers, custID);

if(obj.getCart()==null) obj.assignCart();

//ensures the selected customer has been assigned a cart

System.out.println("This customer is using " + obj.getCustomerCartID());

System.out.println("\n\nSTORE MENU: CUSTOMER CART " + obj.getCustomerCartID().substring(4));

ShoppingCart active = obj.getCart();

System.out.println("(1) add/remove items\n(2) view items in the cart\n(3) return to last menu\nPlease make your selection?");

switch(input.nextInt()) {

case(1):

boolean shopping = true;

do {

System.out.println("Which kind of item would you like to add? (b) Book, (m) Movie, or (c) CD?\n You can add prefix - to remove these items so, -b, -m, -c are valid as well");

System.out.println("Additionally since items are avaialable in discounted prices (books and cds only, no discounted movies for sale), use bD or cD are also valid");

String type = input.next();

if(type.equals("b")) obj.getCart().addItem(new Book());

if(type.equals("m")) obj.getCart().addItem(new Movie());

if(type.equals("c")) obj.getCart().addItem(new CD());

if(type.equals("cD")) obj.getCart().addItem(new CD("dis"));

if(type.equals("bD")) obj.getCart().addItem(new Book("dis"));

//Use the override of toString method to show which items are available for removal by removeItem

if(type.equals("-b")) obj.getCart().removeItem();

if(type.equals("-m")) obj.getCart().removeItem();

if(type.equals("-c")) obj.getCart().removeItem();

showCustArray(customers, getCustomerIndex(customers,obj.getCustomerID()));

System.out.print("Would you like to (1) checkout, or (2) continue shopping");

if(input.nextInt()==1)

shopping=false;

} while (shopping);

System.out.println("Customer " + obj.getCustomerID() + ", " + obj.getName() + " is ready for check out, his total is " + obj.getCart().getTotalPurchaseAmount());

System.out.println("This means that the customer and cart should be made free afterwards.");

obj.removeCart();

customers[getCustomerIndex(customers,obj.getCustomerID())] = new Customer(CustomerID.generate());

break;

case(2):

showCustArray(customers, getCustomerIndex(customers,obj.getCustomerID()));

break;

default:

break;

}

}

catch (Exception e)

{

break;

}

default:

StoreGUI.help();

}

}

//System.exit(0);

}

More garbage code

public static boolean isShopping(Customer[] a) {

for(int i=0; i< a.length; i++) {

// if (a[i].getShoppingcart()!=null)

return true;

}

return false;

}

//Another overvloading of showCust to show the specific cart ID with the customer

private static void showCustArray(Customer[] a, String m, String n) {

int count = 0;

for (int i =0; i<a.length; i++) {

count++;

if(isShopping(a)==true)

System.out.println("The customer " + a[i].getName() + " is shopping");

else

count++;

continue;

}

if(count == a.length-1) {

System.out.println("There are no customers assigned shopping carts at this time.");

}

}

class Customerold {

public String storeInfo;

private String customerID;

private String fName;

private String lName;

private ShoppingCart cartObjectRef; //a pointer to the index within array of shopping cart objects

public Customerold() {

}

public Customerold(String s) {

this.customerID = s;

this.fName = "Joe";

this.lName = "Smith";

}

public String getName() {

return this.fName + " " + this.lName;

}

public String getCustomerID() {

return this.customerID;

}

public String getCustomerCartID() {

return this.cartObjectRef.toString();

}

public ShoppingCart getCart() {

return this.cartObjectRef;

}

/\* //a mutator method in the customer class to reference one of the shopping carts in store

public void assignCart() {

for (ShoppingCart cart : Store.carts) {

if (cart.isCartAvailable() == false) {

this.cartObjectRef = cart;

cart.isInUse();

break;

} else {

System.out.println("There are no more shopping carts.");

}

}

}

\*/

@Override

public String toString() {

return fName + " " + lName + ", ID#" + customerID + ", " + cartObjectRef + ", ";

}

}

class ShoppingCartold {

//A shopping cart is what assists people in their shopping

private String cartID;

private static int cartCount;

private int itemCount;

private boolean inUse;

java.util.ArrayList<Item> items = new java.util.ArrayList<>();

public ShoppingCartold() {

}

public ShoppingCartold(int cartCount) {

this.cartID = "Cart" + cartCount;

this.inUse = false;

cartCount++;

}

public boolean isCartAvailable() {

return this.inUse;

}

public void isNotInUse() {

this.inUse=false;

}

public void isInUse() {

this.inUse=true;

//return this.inUse;

}

public String getCartID() {

return this.cartID;

}

public void addItem(Item i) {

items.add(i);

}

public void removeItem() {

// items.remove((int)Math.random()\*this.itemCount);

items.remove(0);

}

public int getItemCount() {

return items.size();

}

public int getTotalPurchaseAmount() {

int total = 0;

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

total+=items.get(i).getPrice();

}

return total;

}

@Override

public String toString(){

return (cartID);

}

}