ΗΛΕΚΤΡΟΝΙΚΟ ΒΙΒΛΙΟΠΩΛΕΙΟ (ΜΙΚΡΗ ΕΚΔΟΧΗ)

Έστω ότι θέλετε να δημιουργήσετε ένα ηλεκτρονικό βιβλιοπωλείο το οποίο πουλάει δύο διαφορετικά είδη προϊόντων, βιβλία και CD. Κάθε ένα από αυτά περιέχει έναν (μη κενό) τίτλο και μια τιμή (μεγαλύτερη του μηδενός). Κάθε βιβλίο περιέχει και επιπλέον πληροφορίες, όπως τον συγγραφέα του βιβλίου, αν περιέχει μαλακό ή σκληρό εξώφυλλο και το έτος δημοσίευσής του. Όσον αφορά τα CD, προσφέρεται η δυνατότητα της έκπτωσης σε κάποια επιλεγμένα κομμάτια.

Θέλουμε να δημιουργήσουμε μια εφαρμογή που να εμφανίζει στο χρήστη μια λίστα από τα διαθέσιμα βιβλία και CD του ηλεκτρονικού βιβλιοπωλείου, μαζί με τις λεπτομέρειές τους. Επιπλέον, πρέπει να δίνεται η δυνατότητα προσθήκης και διαγραφής των προϊόντων σε ένα καλάθι αγορών. Στη συνέχεια, ο χρήστης της εφαρμογής πρέπει να μπορεί να προβάλει τα προϊόντα τα οποία έχουν προστεθεί στο καλάθι μαζί με τις τιμές τους.

Παρατίθεται το κείμενο προδιαγραφής απαιτήσεων για ένα ΜΙΚΡΟ παράδειγμα, με λίγες use cases και λίγες κλάσεις.

Ζήτημα 0. (α) <u>Χωρίς</u> να δείτε τον κώδικα, βρείτε από την εκφώνηση τα use cases, και παραθέστε για καθένα (α) όνομα, (β) βασικούς actors, (γ) μια περιγραφή της μίας πρότασης

[Αν έχετε διδαχθεί Class Design:]

(β) Επίσης, ξανά <u>χωρίς</u> να δείτε τα παρακάτω, βγάλτε διάγραμμα κλάσεων και χαρακτηρίστε τις κλάσεις αν είναι domain, business logic ή boundary.

(γ) Για κάθε use case, συμπληρώστε από μία γραμμή σε ένα πινακάκι με στήλες: use case / method / class / class type για το ποια είναι η κεντρική μέθοδος με την οποία υλοποιείται η use case στη σχεδίασή σας

Use case	*Method*	*Class*	*Class type*	*Comment*
Add item	*addItem()*	*ItemManager*	*Business Logic*	*… αν χρειάζεται …*
…	…	…	…	…

Ζήτημα 1: Δείτε τις μεθόδους στον κώδικα του συστήματος και ελέγξτε τις use cases που βγάλατε.

Ζήτημα 2: Δοθέντων των use cases βγάλτε test, <u>χωρίς</u> να δείτε τα test που σας δίνονται

Ζήτημα 3: Δείτε τα test που έχει ο κώδικας για κάθε κλάση του συστήματος και απαντήστε:

- **Γιατί αυτά τα test?**
- **Γιατί οι μέθοδοι του συστήματος (όχι των test) έχουν αυτές τις παραμέτρους και αυτό τον τύπο επιστροφής?**

Ζήτημα 4: Καλύπτουν τα test τις προδιαγραφές όπως προκύπτουν από τα <u>use cases?</u>

Ζήτημα 5,6: διορθώστε/αναμορφώστε δοθέντα διαγράμματα κλάσεων (βλ. μέσα)

Διευκρινίζεται ότι παραδοσιακά, στο διαγώνισμα τα ζητούμενα θέματα είναι παρόμοια με τα ζητήματα 0 (κυρίως – αφορά όλα τα υποερωτήματα) και 3. Τα υπόλοιπα ζητήματα, όμως, είναι σημαντικά για να κατανοήσετε το μεθοδολογικό κομμάτι.


```
package bookstoreAcceptable;
```

```
public abstract class Item {
  public Item(){title="";price=-1.0;id=-1;}
  public Item(String aTitle, double aPrice,int id){
     title = aTitle; price=aPrice;this.id =id;
   }

  public abstract String showDetails();

  public abstract String getDetails();

  public abstract double getFinalPrice();

  public String getTitle(){return title;}

  public int getId(){return id;}

  public double getOriginalPrice(){return price;}

  protected String title;
  protected double price;
  protected int id;
}
```

```java
package bookstoreAcceptable;
public class Book extends Item {
private String author;
  private int packaging; // 0 for simple, 1 for hard box
  private int yearPublished;

  public Book(String aTitle, String anAuthor, int aDate, double aPrice,
      int aPackage, int id) {
    super(aTitle, aPrice, id); //constructor of Item!!!
    author = anAuthor; packaging = aPackage; yearPublished = aDate;
  }

  @Override
  public String showDetails() {
    String result = "***Item id: " + id + "***\n" + title + "\t\t Price:" + price +
"\n" + " by " + author + " at " + yearPublished +  "\n";
    String packString;
    if (packaging == 1)
      packString = "hard box";
    else
      packString = "simple";
    result = result + "with " + packString + " packaging.\n\n";
    return result;
  }

  @Override
  public String getDetails(){
    String packString;
    if (packaging == 1)
      packString = "hard box";
    else
      packString = "simple";
    return (title + "\nby " + author + " at " + yearPublished
        + "\nwith " + packString  + " packaging.\n");
  }

  @Override
  public double getFinalPrice() {
    return price;
  }
}//end Book
```

```java
package bookstoreAcceptable;

public class CD extends Item {
  private String artist;
  private double discount;

  public CD(String aTitle, double aPrice, String anArtist, double aDiscount, int id) {
    super(aTitle, aPrice, id);
    artist = anArtist; discount = aDiscount;
  }

  @Override
  public double getFinalPrice() {
    return (price - discount);
  }

  @Override
  public String showDetails() {
    String result = "***Item id: " + id + "*** \n" +
        title + "\t\t Price:" + price + "\n" + "by " + artist + "\n" +
        "final price: " + getFinalPrice()+ "\n\n";
    return result;
  }

  @Override
  public String getDetails(){
    return (title + "\n by " + artist);
  }
}
```



```java
package bookstoreAcceptable;

import java.util.ArrayList;
import java.util.Iterator;

public class ItemManager {
   private ArrayList<Item> allItems;

  public ItemManager(){
    allItems = new ArrayList<Item>();
  }
  public int addItem(Item anItem){
    allItems.add(anItem);
    return allItems.size();
  }
  public int removeItem(int index){
    this.allItems.remove(index);
    return allItems.size();
  }

  public Item getItem(int index){
    return this.allItems.get(index);
  }

  public ArrayList<Item> getAllItems(){
    return this.allItems;
  }

  public String reportAllItems(){
    String result ="";
    Iterator <Item> ii = allItems.iterator();
    while (ii.hasNext()){
      Item i = ii.next();
      result = result + i.showDetails();
    }
    result = result + "Total number of items: " + this.allItems.size()+ "\n";
    System.out.println(result);
    return result;
  }
}
```

```java
package bookstoreAcceptable;

public class ShoppingCart {
  private ArrayList<Item> items;
  private float totalCost;

  public ShoppingCart(){
    this.items = new ArrayList<Item>();
    this.totalCost = 0;
  }

  public int addItem(Item item){
    this.items.add(item);
    return this.items.size();
  }

  public int removeItem(int anId){
    int pos = -1;
    for(int i = 0; i < this.items.size(); i++){
      if (items.get(i).getId() == anId){
        pos = i;
        break;
      }
    }
```

```
    if (pos>=0)
      this.items.remove(pos);
    else{
      System.out.println("The id you have specified is not in the cart");
    }
    return this.items.size();
  }//end removeItem

  private void computeTotalCost(){
    float cost = 0;
    for(Item item: this.items){
      cost += item.getFinalPrice();
    }
    this.totalCost = cost;
  }

  public float getTotalCost(){
    this.computeTotalCost();
    return this.totalCost;
  }

  public ArrayList<Item> getItems(){
    return this.items;
  }

  public String showDetails(){
    String intro = "------------------------\n" +
                   "       CART ITEMS       \n" +
                   "------------------------\n";
    String itemsString = "";
    for(int i = 0; i < this.items.size(); i++){
      //System.out.println("***Item id: " + i + "***");
      itemsString = itemsString + items.get(i).showDetails();
    }
    String costString = "Total cost: " + this.getTotalCost() + "\n";
    String result = intro + itemsString + costString;
    System.out.println(result);
    return result;
  }
}
```

```
package bookstoreAcceptable;

public class SimpleBookstoreApplication {

  private Scanner reader;

  public SimpleBookstoreApplication(){
    reader = new Scanner(System.in);
  }

  public int printMenu(){
    int answerOperation = 0;
    while( answerOperation > 5 || answerOperation <= 0){
      System.out.println("Choose(1-4)\n 1. Show items\n 2. Add item to cart\n 3. Show
cart\n "
          + "4. Remove item from cart\n 5. Exit");
      answerOperation = reader.nextInt();
      if(answerOperation > 5 || answerOperation <= 0)
        System.out.println("Wrong answer! Try again...");
    }

    return answerOperation;
  }

  public Scanner getReader(){
    return reader;
  }
```



```
  public static void main(String args[]){

    SimpleBookstoreApplication app = new SimpleBookstoreApplication();

    ItemManager itemManager = new ItemManager();


    // Book bookRef;

    Item item = new Book("Discours de la methode","Rene Descartes", 1637, 50.00, 0,
0);
    itemManager.addItem(item);
    item = new Book("The Meditations",   "Marcus Aurelius", 180,30.00, 1,1);
    itemManager.addItem(item);
    item = new Book("The Bacchae","Euripides", -405,30.00,  1,2);
    itemManager.addItem(item);
    item = new Book("The Trojan Women","Euripides",-415,40.00, 0,3);
    itemManager.addItem(item);

    //CD cdRef;
    item = new CD("Piece of Mind", 10.0,"Iron Maiden",4.0,4);
    itemManager.addItem(item);
    item = new CD("Matter of Life and Death", 12.0,"Iron Maiden",2.0,5);
    itemManager.addItem(item);
    item = new CD("Perfect Strangers", 12.00, "Deep Purple",1.0,6);
    itemManager.addItem(item);

    ShoppingCart cart = new ShoppingCart();

    while(true){
      int operation = app.printMenu();
      if(operation == 1){
        itemManager.reportAllItems();
      }
      else if(operation == 2){
        System.out.println("Choose the id of the item that you want to add to your
cart");
        int id = app.getReader().nextInt();
        System.out.println(id);
        if(id > itemManager.getAllItems().size())
          System.out.println("Error: there is no product with the specified id");
        else
          cart.addItem(itemManager.getItem(id));
      }
      else if(operation == 3){
        cart.showDetails();
      }
      else if(operation == 4){
        System.out.println("Choose the id of the item that you want to remove from your
cart");
        int id = app.getReader().nextInt();
        System.out.println(id);
        if(id > itemManager.getAllItems().size())
          System.out.println("Error: there is no product with the specified id");
        else
          cart.removeItem(id);
      }
      else{
        break;
      }
    }
  }
}
```

```
package testPackage;
```

```
package test.testPackage;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ BookTest.class, CDTest.class, ItemManagerTest.class,
ShoppingCartTest.class })
public class AllTests {
  //no need to add sth here. The above directives simply run all tests
}
```

```
package test.testPackage;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import bookstoreAcceptable.CD;

public class CDTest {

  private static CD cdToTest; //attn, must be static because we will use it at beforeClass!

  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
    //this runs once, before _all_ tests. Here we create a single object(to save time)
    //and use it in all tests.
    cdToTest = new CD("Piece of Mind", 10.0,"Iron Maiden",4.0, 7);
  }

  @Before
  public void setUp() throws Exception {
    //see BookTest for explanations.
  }

  //Also here, we omit the tests for null / invalid construction values, as we did in
Book

  @Test
  public final void testGetFinalPrice() {
    //CD cdToTest = new CD("Piece of Mind", 10.0,"Iron Maiden",4.0);
    assertEquals("test if getFinalPrice() works OK", 6.00,cdToTest.getFinalPrice(),3);

    //fail("Not yet implemented"); // TODO
  }

  @Test
  public final void testGetPrice() {
    //CD cdToTest = new CD("Piece of Mind", 10.0,"Iron Maiden",4.0);
    assertEquals("test if getPrice() of the ITEM CLASS works OK", 10.00,
cdToTest.getOriginalPrice(), 3);

    //fail("Not yet implemented"); // TODO
  }
}
```



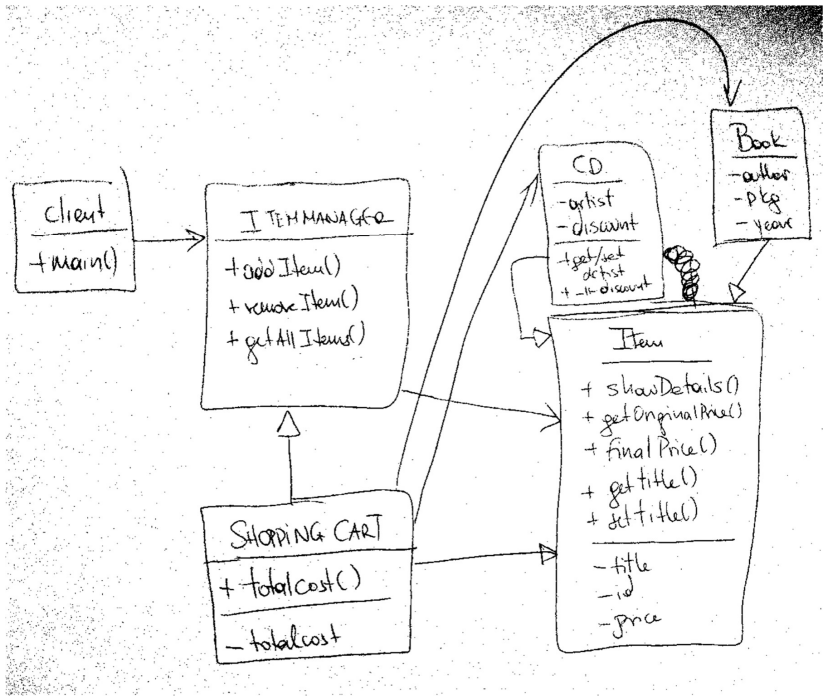
```
package test.testPackage;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import bookstoreAcceptable.Book;

public class BookTest {
  private Book bookToTest;

  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
    //this runs once, before _all_ tests. Nothing todo here.
    //See CDTest for explanations
  }

  @Before
  public void setUp() throws Exception {
    bookToTest = new Book("Discours de la methode", "Rene Descartes",1637, 50.00,0,1);
    //this runs before _each_ test. create a book to test constructor, price and final
price
  }

  @Test
  public final void testBookNull() {
    //remember: the setUp() method has run already: Book should have been initialized!
    assertNotNull("After setup, the book is not null", bookToTest);

    //at the beginning to see that the test works.
    //fail("Not yet implemented");
  }

  @Test
  public final void testBookNoTitle() {

    //See how this test fails. It gets a null title. The constructor should trap this
    //and avoid creating a book without a tile. Unfortunately, it fails...
    bookToTest = new Book(null, "Rene Descartes", 1637, 50.00, 0, 2);
    assertNull("test if constructor prevents creation with null title", bookToTest);

    //at the beginning to see that the test works.
    //fail("Not yet implemented");
  }

  @Test
  public final void testBookNegativePrice() {
  //should do the same with negative price. Again, intentionally put to show failure!
    bookToTest = new Book("Discours de la methode", "Rene Descartes",1637,-12.00,0,3);
    assertNull("test if the constructor prevents creation with negative price",
bookToTest);

    //at the beginning to see that the tes works.
    //fail("Not yet implemented");
  }

  @Test
  public final void testGetFinalPrice() {
    assertEquals("test if getFinalPrice() works OK", 50.00,
                          bookToTest.getFinalPrice(), 3);
    //fail("Not yet implemented");
  }

  @Test
  public final void testGetPrice() {
    assertEquals("test if getOriginalPrice of the ITEM CLASS works OK", 50.00,
                            bookToTest.getOriginalPrice(), 3);
    //fail("Not yet implemented");
  }
}
```

```java
package test.testPackage;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import bookstoreAcceptable.Book;
import bookstoreAcceptable.Item;
import bookstoreAcceptable.ItemManager;

public class ItemManagerTest {

  @Before
  public void setUp() throws Exception {
    //this is supposed to run before _all_ tests
    //nothing to set up here
  }

  @Test
  public void testItemManager() {
    ItemManager amazon = new ItemManager();
    assertNotNull(amazon.getAllItems());
    //before implementing the above, try having just the following fail.
    //Run the test and see that it fails indeed.
    //Then build your test.
    //fail("Intentional failure. Not yet implemented");
  }

  @Test
  public void testAddItem() {
    ItemManager amazon = new ItemManager();
    assertEquals(0,amazon.getAllItems().size());

    Book bookRef;
    bookRef = new Book("Discours de la methode", "Rene Descartes", 1637, 50.00, 0, 9);
    amazon.addItem(bookRef);
    assertNotEquals(0,amazon.getAllItems().size());
    assertEquals(1,amazon.getAllItems().size());

    //fail("Not yet implemented");
  }

  @Test
  public void testReportAllItems() {
    ItemManager itemManager = new ItemManager();

    Item item = new Book("Discours de la methode","Rene Descartes", 1637, 50.00, 0,0);
    itemManager.addItem(item);
    item = new Book("The Meditations",   "Marcus Aurelius", 180,30.00, 1,1);
    itemManager.addItem(item);

    String expectedResult = "***Item id: 0***\n" +
    "Discours de la methode    Price:50.0\n" +
    " by Rene Descartes at 1637\n" +
    "with simple packaging.\n" +
    "\n" +
    "***Item id: 1***\n" +
    "The Meditations    Price:30.0\n" +
    " by Marcus Aurelius at 180\n" +
    "with hard box packaging.\n" +
    "\nTotal number of items: 2\n";
    assertEquals(expectedResult, itemManager.reportAllItems());
  }

}
```



```java
package test.testPackage;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertNotNull;
import org.junit.Test;

import bookstoreAcceptable.Book;
import bookstoreAcceptable.Item;
import bookstoreAcceptable.ShoppingCart;

public class ShoppingCartTest {

  @Test
  public void testItemManager() {
    ShoppingCart cart = new ShoppingCart();
    assertNotNull(cart.getItems());
  }

  @Test
  public void testAddItem() {
    ShoppingCart cart = new ShoppingCart();
    assertEquals(0,cart.getItems().size());

    Book bookRef;
    bookRef = new Book("Discours de la methode", "Rene Descartes", 1637, 50.00, 0, 9);
    cart.addItem(bookRef);
    assertNotEquals(0,cart.getItems().size());
    assertEquals(1,cart.getItems().size());

  }

  @Test
  public void testRemoveItem(){
    ShoppingCart cart = new ShoppingCart();
    assertEquals(0,cart.getItems().size());

    Book bookRef;
    bookRef = new Book("Discours de la methode", "Rene Descartes", 1637, 50.00, 0, 9);
    cart.addItem(bookRef);
    assertNotEquals(0,cart.getItems().size());
    assertEquals(1,cart.getItems().size());

    cart.removeItem(9);
    assertEquals(0,cart.getItems().size());
  }

  @Test
  public void testShowItems(){
    ShoppingCart cart = new ShoppingCart();

    Item item = new Book("Discours de la methode","Rene Descartes", 1637, 50.00, 0,0);
    cart.addItem(item);
    item = new Book("The Meditations",   "Marcus Aurelius", 180,30.00, 1,1);
    cart.addItem(item);

    String expectedResult =
        "------------------------\n" +      "       CART ITEMS       \n" +
        "------------------------\n" +
        "***Item id: 0***\n" +
        "Discours de la methode    Price:50.0\n" +
        " by Rene Descartes at 1637\n" +      "with simple packaging.\n" +
        "\n" +
        "***Item id: 1***\n" +
        "The Meditations    Price:30.0\n" +      " by Marcus Aurelius at 180\n" +
        "with hard box packaging.\n\n" +      "Total cost: 80.0\n";
    assertEquals(expectedResult, cart.showDetails());

  }
}
```

Ζήτημα 5: ΒΡΕΙΤΕ ΤΑ ΛΑΘΗ!

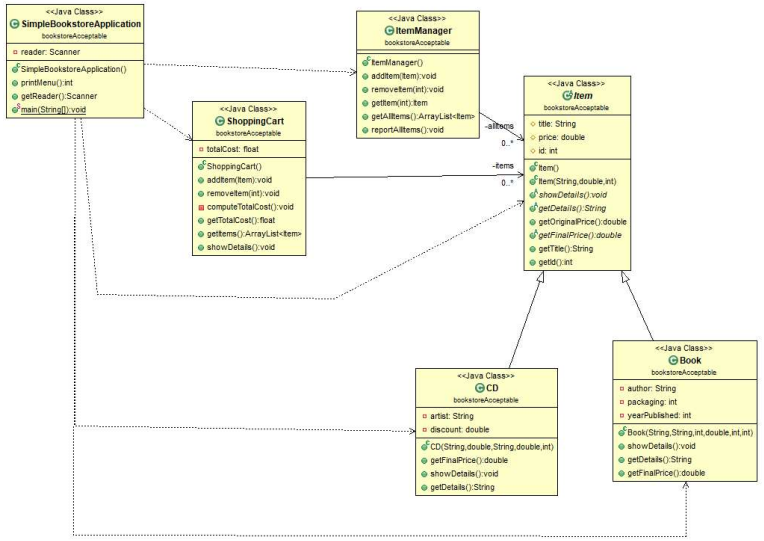


Τι μπορεί να είναι λάθος από πλευράς:

- Μουτζούρας, δυσανάγνωστου κειμένου, κακής διάταξης των βελακίων, …
- Παραβίασης της σωστής «ορθογραφίας» των βελακίων
- Λάθος συσχετίσεων, κληρονομικοτήτων, …
- Κλάσεις που έπρεπε να είναι μέθοδοι και αντίστροφα
- Ξεχασμένων use cases

Δεν είναι απαραίτητο όλα τα λάθη να υπάρχουν εδώ…

ΔΕΙΤΕ ΤΟ ΔΙΑΓΡΑΜΜΑ ΠΟΥ ΑΝΤΙΣΤΟΙΧΕΙ ΣΤΟΝ ΚΩΔΙΚΑ ΠΟΥ ΣΑΣ ΔΟΘΗΚΕ



Ζήτημα 6: Τι θα διορθώνατε / συμπληρώνατε / αλλάζατε στη σχεδίαση του συστήματος?