פרויקט מלווה

מערכת ניהול קופונים

קורס 822

הסבה לתכנות Java לסביבות

Big Data-ו Client, Enterprise

- **כללי**

במסגרת הקורס משולב פרויקט מלווה המפורט במסמך.

הפרויקט מחולק ל-3 שלבים:

שלב 1: בניית ליבת המערכת. בשלב זה יוקם 'המוח' של המערכת. ליבת המערכת תהיה אחראית על קליטת נתונים, אכסונם במסד נתונים וניהולם.

שלב 2: שלב ה-Web. בשלב זה תיחשף המערכת לאינטרנט. אופן בניית מודול זה יתמוך במגוון של לקוחות. במסגרת שלב זה יבנה אתר אינטרנט אך תהיה אפשרות לבנות גם לקוחות מובייל ו-PC.

שלב 3: שילוב יכולות Enterprise. שלב זה יתווסף לליבת המערכת אך יעשה שימוש בתשתיות שרת מתקדמות. התוספת לליבת המערכת תתרכז בתיעוד הכנסות המערכת. תיעוד זה יעשה ע"י בניית שירותים הנקראים Microservice תוך שימוש בטכנולוגיות Java עדכניות לפיתוח צד-שרת.

חלה חובת הגשה על כל השלבים.

תיאור המערכת

מערכת ניהול קופונים מאפשרת לחברות לייצר קופונים כחלק מקמפיינים פרסומיים ושיווקיים שהן מקיימות.

למערכת יש גם לקוחות רשומים. הלקוחות יכולים לרכוש קופונים, כאשר תיעוד ההכנסות מהקופונים מתווסף בשלב השלישי של הפרויקט. קופונים מוגבלים בכמות ובתוקף. לקוח מוגבל לקופון אחד מכל סוג.

המערכת מתעדת את הקופונים שנרכשו ע"י כל לקוח.

הכנסות המערכת הן מרכישת קופונים ע"י לקוחות רשומים ומיצירה ועדכון קופונים חדשים ע"י החברות.

גישה למערכת מתחלקת לשלושה סוגי לקוחות:

1. Administrator – ניהול רשימת החברות ורשימת הלקוחות.
2. Company – ניהול רשימת קופונים המשויכים לחברה.
3. Customer – רכישת קופונים.

הנחיות כלליות

- עבודה על הפרויקט יכולה להתבצע באופן עצמאי או בקבוצות של עד שלושה מתכנתים.
- מועדי הנחיה והגשה – כל שלב בפרויקט מוגש עד לתאריך הנחיית השלב הבא. מועדי הנחיית הפרויקט מופיעות בתוכנית הקורס. לוח זמנים זה קשיח ולשינויו נדרש אישור מרכזת הקורס או מהגורמים המקצועיים. בכל מקרה של בקשה לשינוי בלוח הזמנים, יש לקחת בחשבון הורדת ציון.
- יש להגיש את הפרויקט עובד ומותקן.
- יש להגיש את קבצי המקור ע"י שליחת קבצים באינטרנט.
- יש להשתמש בתיעוד. רצוי לייצר Java Docs.
- יש להגיש בכל שלב מסמך טקסט מפורט לגבי התקנה, שמות משתמשים וסיסמאות, נתוני התחברות למסד הנתונים והנחיות למשתמש.

- במידה והתווספו יכולות מעבר לדרישות ו/או נעשה שימוש ב-API מעבר לנלמד בכיתה – הדבר מבורך ויש לציין רשימת תוספות אלו עם ההגשה על מנת שיתייחסו לכך בבדיקה.
- אין להגיש פרויקט אשר מדפיס Stack Trace במקרים של Exceptions. יש לייצר Exceptions מותאמים למערכת ולהשתמש בהודעות ברורות וקריאות למשתמש בכל מקרה של שגיאה.

• שלב 1

בניית ליבת המערכת

~ OOP, Java Beans, DAO, JDBC, Threads ~

<u>תיאור:</u>

בשלב זה יוגדר מסד הנתונים לאכסון ושליפת מידע אודות לקוחות, חברות וקופונים. מעל מסד הנתונים תוקם שכבת בידוד שתאפשר עבודה נוחה מ-Java אל מסד הנתונים (DAO – Data Access Objects).

כמו כן, יוקמו שירותי תשתית בסיסיים כגון מאגר קישורים למסד הנתונים (ConnectionPool) ו-Job יומי המתחזק את המערכת ומנקה אותה מקופונים שפג תוקפם.

יוגדרו שלושה Entry Points למערכת עבור כל אחד מסוגי לקוחותיה – אדמיניסטרטור, חברה או לקוח אשר יתחברו בביצוע Login.

חלקי הפיתוח עבור שלב 1:

חלק א' – הגדרת מסד הנתונים ובניית הטבלאות.

חלק ב' – בניית מחלקות Java Beans המייצגות את המידע שבמסד הנתונים.

חלק ג' – בניית ConnectionPool המאפשר ניהול מאגר ה-Connections למסד הנתונים.

חלק ד' – בניית מחלקות ה-DAO המאפשרות ביצוע פעולות CRUD כלליות על מסד הנתונים.

חלק ה' – בניית הלוגיקה העסקית הדרושה ע"י שלושת סוגי ה-Clients של המערכת.

חלק ו' – בניית מחלקה המאפשרת כניסת Clients ולפיכך החזרת הלוגיקה העסקית המתאימה.

חלק ז' – בניית Job יומי למחיקת קופונים שפג תוקפם מהמערכת.

חלק ח' – בניית מחלקת Test להדגמת יכולות המערכת והפעלתה מה-main.

דגשים:

- חשוב לתעד את הקוד.
- חשוב להשתמש בשמות משמעותיים עבור משתנים/פונקציות/מחלקות/Packages וכדומה.
- חובה לכתוב באופן יעיל, ללא העתקת קוד (DRY – Don’t Repeat Yourself).
- יש להשתמש ב-Custom Exceptions (לדוגמה: CouponAlreadyExistsException) עבור חריגות ספציפיות של המערכת.
- יש להגדיר try-catch כללי ב-main המסוגל לתפוס את כל ה-Exceptions.

- מומלץ לעבוד עם מסד הנתונים Derby. בפרויקט זה אין תועלת בעבודה עם מסדי נתונים המצריכים התקנות גדולות וניהול מורכב. Derby הינו מסד נתונים פשוט ביותר להתקנה ולהפעלה.
- חובה לחלק את המחלקות ל-Packages מתאימים, בעלי שמות משמעותיים.

חלק א' – הגדרת מסד הנתונים ובניית הטבלאות

להלן רשימת הטבלאות הדרושות:

- COMPANIES – טבלת חברות:

- ID (int – מפתח ראשי, מספור אוטומטי)
- NAME (string – שם החברה)

- EMAIL (string – אימייל החברה)
- PASSWORD (string – סיסמת כניסה)

- CUSTOMERS – טבלת לקוחות:

- ID (int – מפתח ראשי, מספור אוטומטי)
- FIRST_NAME (string – שם פרטי)
- LAST_NAME (string – שם משפחה)
- EMAIL (string – אימייל הלקוח)
- PASSWORD (string – סיסמת כניסה)

- CATEGORIES – טבלת קטגוריות של קופונים:

- ID (int – מפתח ראשי, מספור אוטומטי)
- NAME (string – שם קטגוריית הקופון – מזון, מוצרי חשמל, מסעדות, נופש וכו')

- COUPONS – טבלת קופונים:

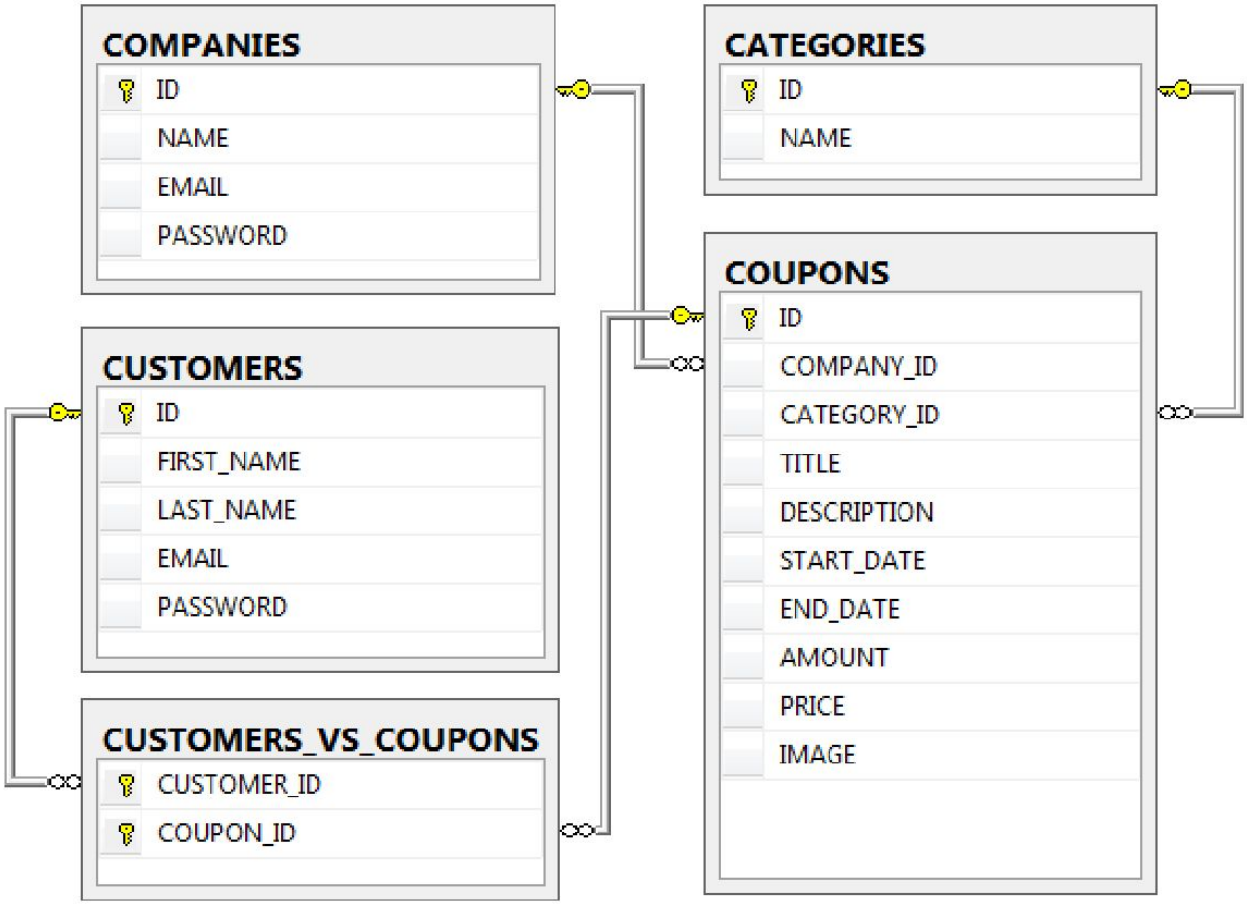
- ID (int – מפתח ראשי, מספור אוטומטי)
- COMPANY_ID (int – מפתח זר, קוד החברה שיצרה את הקופון)
- CATEGORY_ID (int – מפתח זר, קוד קטגוריית הקופון)
- TITLE (string – תיאור קצר של הקופון)
- DESCRIPTION (string – תיאור מפורט של הקופון)
- START_DATE (date – תאריך יצירת הקופון)
- END_DATE (date – תאריך תפוגת הקופון)
- AMOUNT (integer – כמות קופונים במלאי)
- PRICE (double – מחיר הקופון ללקוח)

- IMAGE (string – שם קובץ תמונה עבור הקופון)

- CUSTOMERS_VS_COUPONS – טבלת לקוחות מול קופונים – מאפשרת לדעת אלו קופונים רכש כל לקוח ואלו לקוחות רכשו כל קופון:

- CUSTOMER_ID (int – מפתח ראשי בטבלה זו ומפתח זר לקוד הלקוח)
- COUPON_ID (int – מפתח ראשי בטבלה זו ומפתח זר לקוד הקופון)

להלן סכמת הטבלאות והיחסים ביניהן:

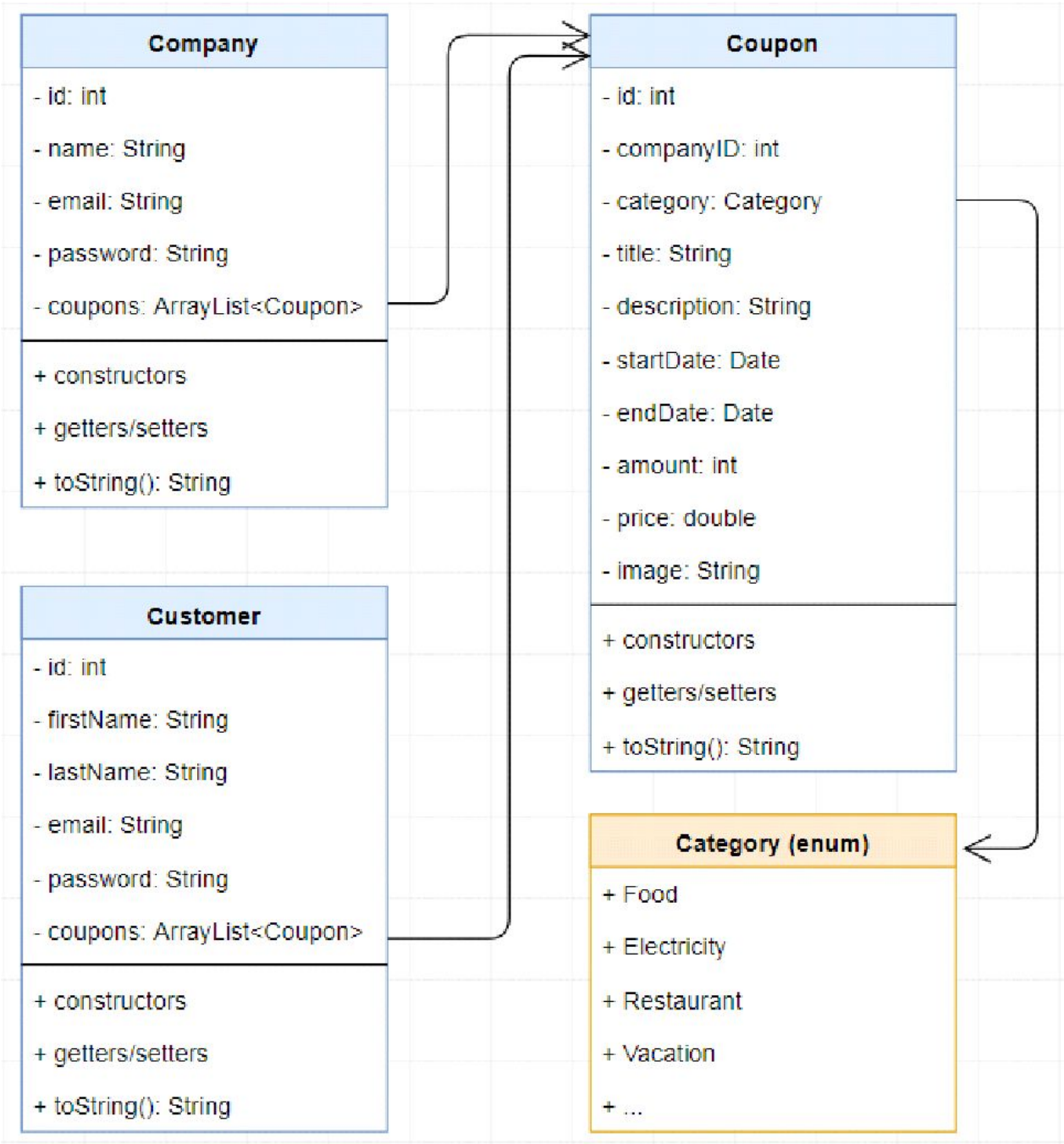

COMPANIES
ID
NAME
EMAIL
PASSWORD
CATEGORIES
ID
NAME
COUPONS
ID
COMPANY_ID
CATEGORY_ID
TITLE
DESCRIPTION
START_DATE
END_DATE
AMOUNT
PRICE
IMAGE
CUSTOMERS
ID
FIRST_NAME
LAST_NAME
EMAIL
PASSWORD
CUSTOMERS_VS_COUPONS
CUSTOMER_ID
COUPON_ID

חלק ב' – בניית מחלקות Java Beans המייצגות את המידע שבמסד הנתונים

Java Beans **הינן מחלקות מידע טהור המייצגות את המידע המועבר בין האפליקציה לבין מחלקות ה-**DAO (Data Access Objects)**. מחלקות ה-**DAO **מתרגמות את אובייקטי ה-**Java Beans **שנשלחים אליהן**

לשאילתות SQL**. כל טבלה עיקרית מיוצגת ע"י מחלקה משלה. הקשרים בין הטבלאות מיוצגים ע"י** Collections **מתאימים.**

להלן סכמת מחלקות ה-Java Beans:


Company
- id: int
- name: String
- email: String
- password: String
- coupons: ArrayList<Coupon>
+ constructors
+ getters/setters
+ toString(): String
Coupon
- id: int
- companyID: int
- category: Category
- title: String
- description: String
- startDate: Date
- endDate: Date
- amount: int
- price: double
- image: String
+ constructors
+ getters/setters
+ toString(): String
Customer
- id: int
- firstName: String
- lastName: String
- email: String
- password: String
- coupons: ArrayList<Coupon>
+ constructors
+ getters/setters
+ toString(): String
Category (enum)
+ Food
+ Electricity
+ Restaurant
+ Vacation
+ ...

חלק ג' – בניית ConnectionPool המאפשר ניהול מאגר ה-Connections למסד הנתונים

ConnectionPool הינה מחלקת Singleton (מחלקה ממנה קיים אובייקט אחד ויחיד) המאפשרת לנהל מספר קבוע של Connections למסד הנתונים. ה-Connections שמורים במאגר (אוסף) ברמת המחלקה.

להלן המתודות הדרושות במחלקה:

Connection getConnection()

מוציאה מהמאגר אובייקט Connection אחד ומחזירה אותו ע"י return כך שניתן יהיה להשתמש בו לצורך ביצוע פעולות על מסד הנתונים. במידה והמאגר ריק כי כל ה-Connections כרגע בשימוש – מבצעת wait בכדי להמתין עד ש-Connection כלשהו יתפנה ויוחזר למאגר.

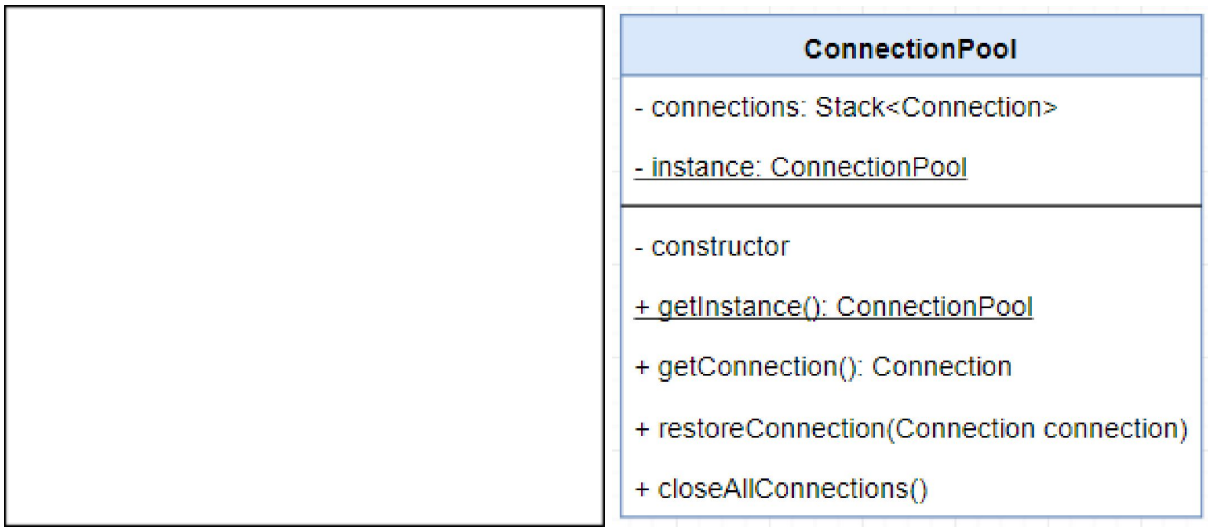
void restoreConnection(Connection connection)

מקבלת כארגומנט אובייקט Connection אחד שהתפנה ומחזירה אותו למאגר. בגלל שכעת התווסף למאגר Connection נוסף – מבצעת notify בכדי להודיע ל-Thread אחד שהמתין (כלומר ביצע wait) שכעת Connection אחד התפנה וכך ניתן לנסות לקבל אותו.

void closeAllConnections()

סוגרת את כל ה-Connections מבחינת מסד הנתונים.

להלן סכמת מחלקת ה-ConnectionPool:

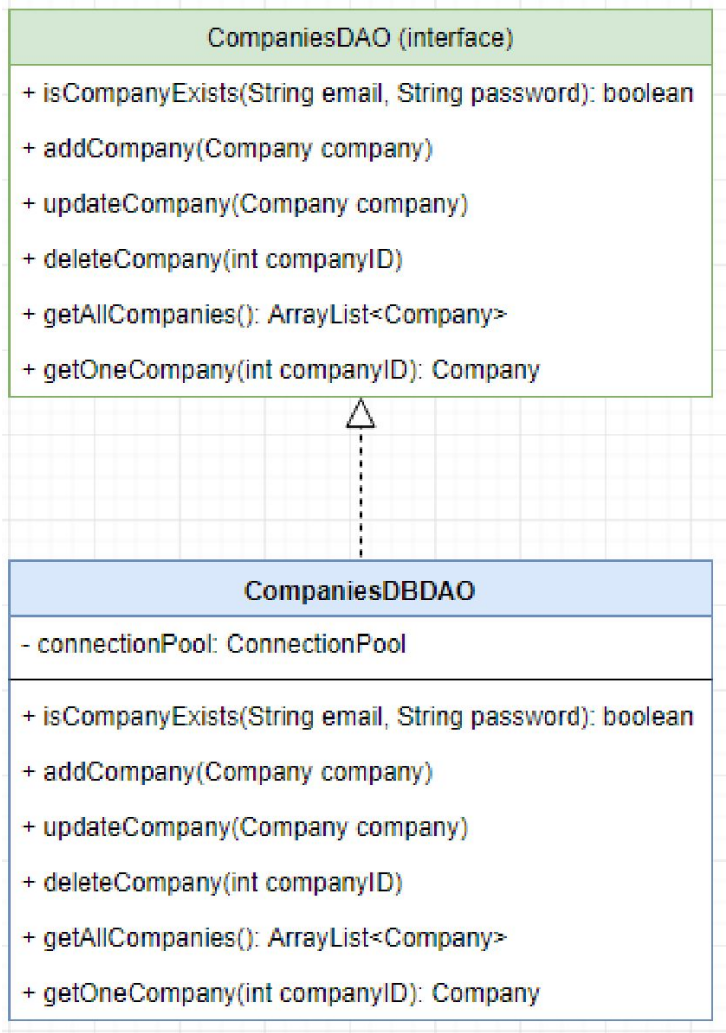


חלק ד' – בניית מחלקות ה-DAO המאפשרות ביצוע פעולות CRUD כלליות על מסד הנתונים

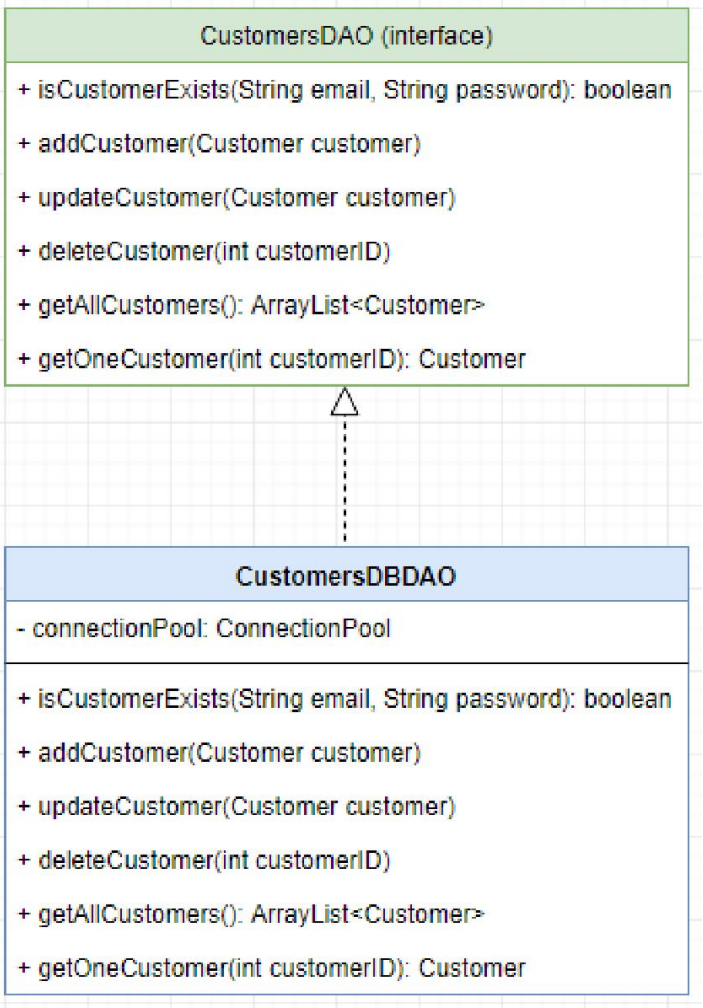
מחלקות DAO (Data Access Objects) אלו מחלקות המאפשרות לבצע פעולות CRUD כלליות (Create/Read/Update/Delete) על הטבלאות במסד הנתונים. מחלקות אלו לא מבצעות את הלוגיקה הקשורה לאפליקציה אלא אך ורק פעולות CRUD כלליות. מחלקות אלו מקבלות כארגומנטים אובייקטי Java Beans או ערכים פרימיטיביים פשוטים (int, string וכדומה), מייצרות מהן שאילתות SQL ומוציאות את השאילתות לפועל במסד הנתונים. ויכולות להחזיר בחזרה אובייקטי Java Beans בודדים, Collections של Java Beans או ערכים פרימיטיביים פשוטים. כמו כן הן מבצעות שימוש ב-ConnectionPool לצורך השגת Connection למסד הנתונים עבור ביצוע הפעולות השונות.

עקב ההפרדה של שכבה זו, אם דרוש יהיה בעתיד מסד נתונים אחר מהקיים, ניתן יהיה להחליף את מסד הנתונים ללא שום נגיעה בשאר חלקי המערכת אלא אך ורק ע"י החלפת שכבת ה-DAO. לצורך הפרדה מושלמת בין שכבה זו לבין שאר חלקי המערכת, נגדיר interfaces המכילים את הפונקציונליות הדרושה עבור שכבת ה-DAO ונממש את הפונקציונליות הזו במחלקות ה-DAO השונות.

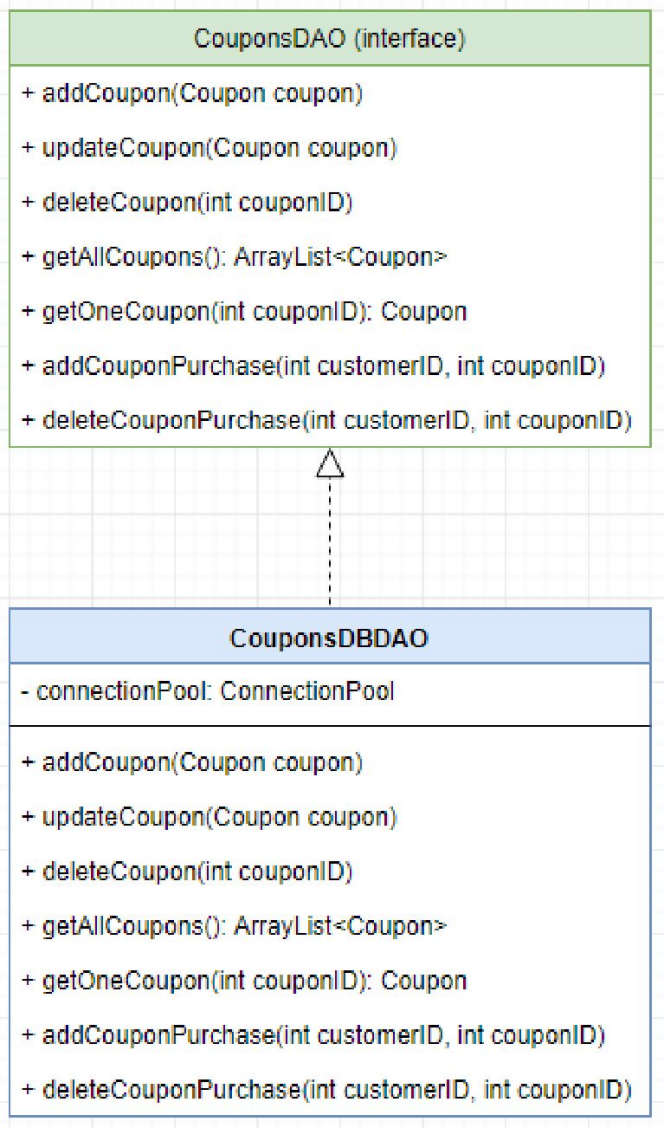
להלן סכמת ה-DAO של טבלת החברות:


CompaniesDAO (interface)
+ isCompanyExists(String email, String password): boolean
+ addCompany(Company company)
+ updateCompany(Company company)
+ deleteCompany(int companyID)
+ getAllCompanies(): ArrayList<Company>
+ getOneCompany(int companyID): Company
CompaniesDBDAO
- connectionPool: ConnectionPool
+ isCompanyExists(String email, String password): boolean
+ addCompany(Company company)
+ updateCompany(Company company)
+ deleteCompany(int companyID)
+ getAllCompanies(): ArrayList<Company>
+ getOneCompany(int companyID): Company

להלן סכמת ה-DAO של טבלת הלקוחות:


CustomersDAO (interface)
+ isCustomerExists(String email, String password): boolean
+ addCustomer(Customer customer)
+ updateCustomer(Customer customer)
+ deleteCustomer(int customerID)
+ getAllCustomers(): ArrayList<Customer>
+ getOneCustomer(int customerID): Customer
CustomersDBDAO
- connectionPool: ConnectionPool
+ isCustomerExists(String email, String password): boolean
+ addCustomer(Customer customer)
+ updateCustomer(Customer customer)
+ deleteCustomer(int customerID)
+ getAllCustomers(): ArrayList<Customer>
+ getOneCustomer(int customerID): Customer

להלן סכמת ה-DAO של טבלת הקופונים:


CouponsDAO (interface)
+ addCoupon(Coupon coupon)
+ updateCoupon(Coupon coupon)
+ deleteCoupon(int couponID)
+ getAllCoupons(): ArrayList<Coupon>
+ getOneCoupon(int couponID): Coupon
+ addCouponPurchase(int customerID, int couponID)
+ deleteCouponPurchase(int customerID, int couponID)
CouponsDBDAO
- connectionPool: ConnectionPool
+ addCoupon(Coupon coupon)
+ updateCoupon(Coupon coupon)
+ deleteCoupon(int couponID)
+ getAllCoupons(): ArrayList<Coupon>
+ getOneCoupon(int couponID): Coupon
+ addCouponPurchase(int customerID, int couponID)
+ deleteCouponPurchase(int customerID, int couponID)

חלק ה' – בניית הלוגיקה העסקית הדרושה ע"י שלושת סוגי ה-Clients של המערכת.

Client נחשב כל מי שיכול להשתמש במערכת. שלושת סוגי ה-Clients של המערכת הינם:

- Administrator – המנהל הראשי של כל המערכת.
- Company – כל אחת מהחברות שבמערכת.
- Customer – כל אחד מהלקוחות שבמערכת.

לכל סוג Client כזה ישנן פעולות לוגיות עסקיות הדרושות לביצוע על ידו. לדוגמה, Administrator יכול להוסיף חברה חדשה, אולם Company או Customer לא יכולים (ולא אמורים) להוסיף חברה חדשה. לדוגמה, Customer יכול לרכוש לעצמו קופון, אולם Administrator או Company לא יכולים (ולא אמורים) לרכוש לעצמם קופון. לכן, כל הפעולות הדרושות לביצוע ע"י כל Client יהיו במחלקה ייעודית המכילה את כל הלוגיקה העסקית הדרושה עבור אותו ה-Client.

כל פעילות לוגית עסקית כזו תתבצע ע"י פונקציה ייעודית ותשתמש במחלקות ה-DAO בכדי לבצע את הפעילות הלוגית שלה. פונקציה כזו, יכולה לצורך ביצוע הפעילות שלה לבצע סדרת פעולות עם מחלקות ה-DAO. לדוגמה, לצורך רכישת קופון ע"י לקוח (פעולה לוגית אחת שתהיה במחלקת הלוגיקה העסקית של ה-Customer) עלינו לבצע סדרת פעולות ע"י שימוש במחלקות ה-DAO:

- לוודא שהלקוח לא רכש כבר בעבר קופון כזה.
- לוודא שהקופון הדרוש עדיין קיים במלאי (הכמות שלו גדולה מ-0).
- לוודא שתאריך התפוגה של הקופון עדיין לא הגיע.
- לבצע את רכישת הקופון ע"י הלקוח.
- להוריד את הכמות במלאי של הקופון ב-1.

Design כזה, המאפשר להחצין פעולות לוגיות פשוטות, שמאחורי הקלעים משתמשות בסדרת פעולות בסיסיות יותר, נקרא Facade (חזית) שכן לאחר מכן אנו נצטרך רק להשתמש במחלקות ה-Facade הללו לצורך ביצוע הלוגיקה העסקית הדרושה במערכת, ללא שום שימוש נוסף במחלקות ה-DAO הבסיסיות יותר.

שלושת מחלקות הלוגיקה העסקית הינן:

- AdminFacade – מכילה את הלוגיקה העסקית של Administrator.
- CompanyFacade – מכילה את הלוגיקה העסקית של Company.

- CustomerFacade – מכילה את הלוגיקה העסקית של Customer.

בגלל ששלושת המחלקות הללו צריכות להשתמש ברכיבי DAO, יהיה נכון להגדיר מחלקת בסיס (אבסטרקטית כמובן) שמכילה את רכיבי ה-DAO, או לפחות רק References שלהם, ומחלקות ה-Facade יורשות את מחלקת הבסיס הזו.

להלן הלוגיקה הדרושה לביצוע ע"י Client מסוג Administrator, שיש לבנות במחלקת ה-AdminFacade:

- כניסה למערכת.
  - במקרה זה (רק עבור Administrator) אין צורך לבדוק את האימייל והסיסמה מול מסד הנתונים, אלא יש לבדוק אותם Hard-Coded. האימייל תמיד יהיה admin@admin.com והסיסמה תמיד תהיה admin.
- הוספת חברה חדשה.
  - לא ניתן להוסיף חברה בעלת שם זהה לחברה קיימת.
  - לא ניתן להוסיף חברה בעלת אימייל זהה לחברה קיימת.
- עדכון חברה קיימת.
  - לא ניתן לעדכן את קוד החברה.
  - לא ניתן לעדכן את שם החברה.
- מחיקת חברה קיימת.
  - יש למחוק בנוסף גם את הקופונים שיצרה החברה.
  - יש למחוק בנוסף גם את היסטוריית רכישת הקופונים של החברה ע"י לקוחות.
- קבלת כל החברות.
- קבלת חברה ספציפית לפי קוד חברה.
- הוספת לקוח חדש.
  - לא ניתן להוסיף לקוח בעל אימייל זהה ללקוח קיים.
- עדכון לקוח קיים.
  - לא ניתן לעדכן את קוד הלקוח.
- מחיקת לקוח קיים.

- יש למחוק בנוסף גם את היסטוריית רכישת הקופונים של הלקוח.
- קבלת כל הלקוחות.
- קבלת לקוח ספציפי לפי קוד לקוח.

להלן הלוגיקה הדרושה לביצוע ע"י Client מסוג Company, שיש לבנות במחלקת ה-CompanyFacade:

- כניסה למערכת.
  - יש לבדוק את פרטי ה-Login (אימייל וסיסמה) מול מסד הנתונים.
- הוספת קופון חדש.
  - אין להוסיף קופון בעל כותרת זהה לקופון קיים של אותה החברה. מותר להוסיף קופון בעל כותרת זהה לקופון של חברה אחרת.
- עדכון קופון קיים.
  - לא ניתן לעדכן את קוד הקופון.
  - לא ניתן לעדכן את קוד החברה.
- מחיקת קופון קיים.
  - יש למחוק בנוסף גם את היסטוריית רכישת הקופון ע"י לקוחות.
- קבלת כל הקופונים של החברה.
  - כלומר יש להחזיר את כל הקופונים של החברה שביצעה Login.
- קבלת כל הקופונים מקטגוריה ספציפית של החברה.

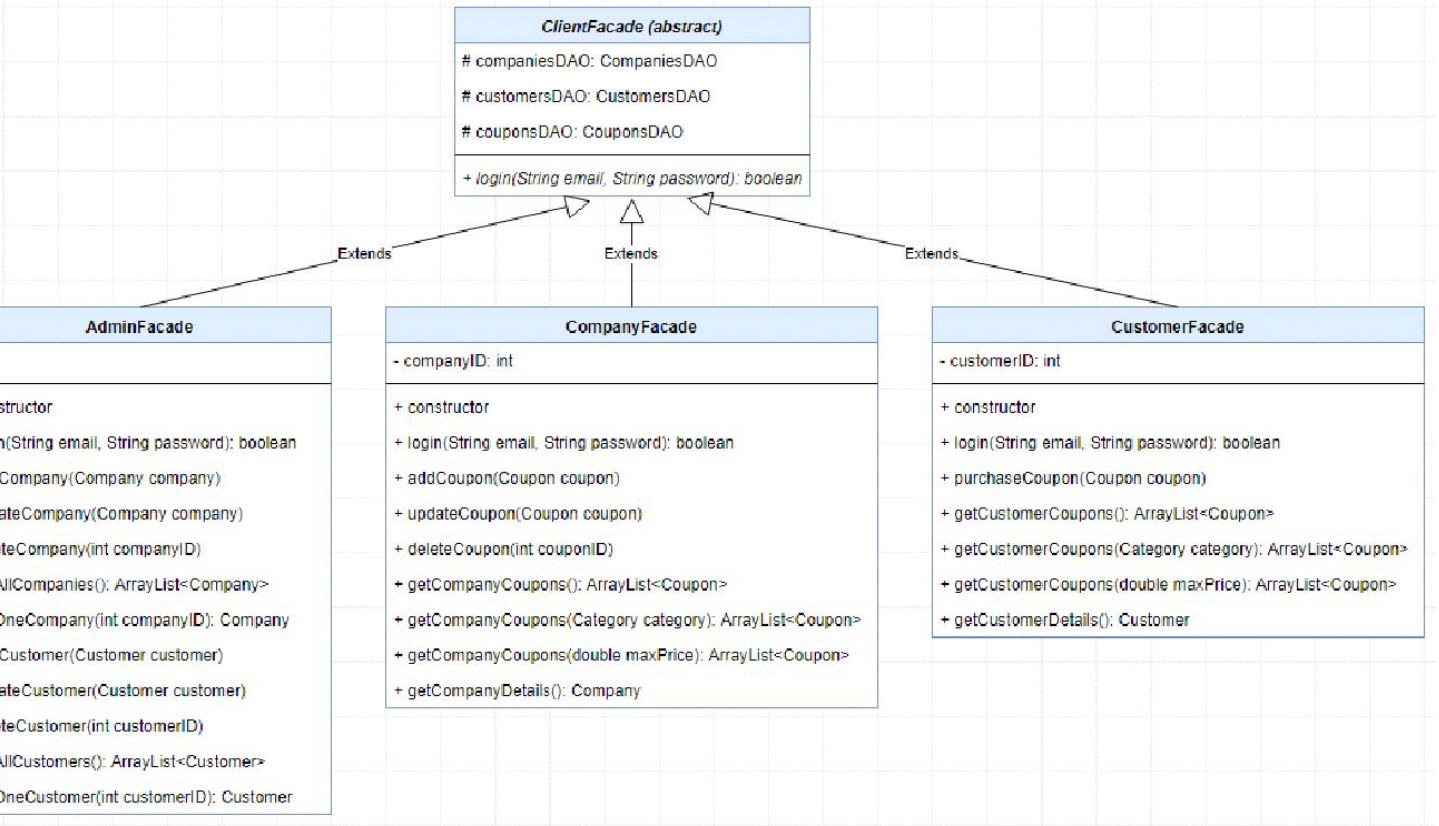
- כלומר יש להחזיר רק קופונים מקטגוריה ספציפית של החברה שביצעה Login.
- קבלת כל הקופונים עד מחיר מקסימלי של החברה.
  - כלומר יש להחזיר רק קופונים עד מחיר מקסימלי של החברה שביצעה Login.
- קבלת פרטי החברה.
  - כלומר יש להחזיר את פרטי החברה שביצעה Login.

להלן הלוגיקה הדרושה לביצוע ע"י Client מסוג Customer, שיש לבנות במחלקת ה-CustomerFacade:

- כניסה למערכת.
  - יש לבדוק את פרטי ה-Login (אימייל וסיסמה) מול מסד הנתונים.
- רכישת קופון.
  - לא ניתן לרכוש את אותו הקופון יותר מפעם אחת.
  - לא ניתן לרכוש את הקופון אם הכמות שלו היא 0.
  - לא ניתן לרכוש את הקופון אם תאריך התפוגה שלו כבר הגיע.

- לאחר הרכישה, יש להוריד את הכמות במלאי של הקופון ב-1.
- קבלת כל הקופונים שהלקוח רכש.
  - כלומר יש להחזיר את כל הקופונים שרכש הלקוח שביצע Login.
- קבלת כל הקופונים מקטגוריה ספציפית שהלקוח רכש.
  - כלומר יש להחזיר רק קופונים מקטגוריה ספציפית של הלקוח שביצע Login.
- קבלת כל הקופונים עד מחיר מקסימלי שהלקוח רכש.
  - כלומר יש להחזיר רק קופונים עד מחיר מקסימלי של הלקוח שביצע Login.
- קבלת פרטי הלקוח.
  - כלומר יש להחזיר את פרטי הלקוח שביצע Login.

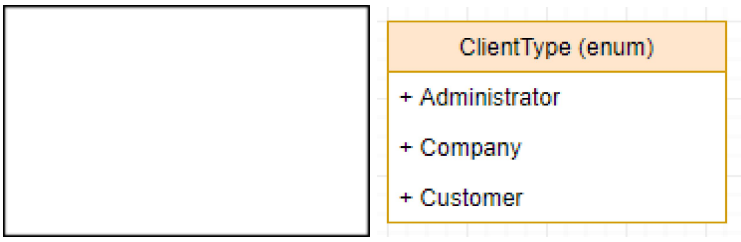
להלן סכמת מחלקות ה-Facade:


ClientFacade (abstract)
companiesDAO: CompaniesDAO
customersDAO: CustomersDAO
couponsDAO: CouponsDAO
+ login(String email, String password): boolean
Extends
Extends
Extends
AdminFacade
structor
n(String email, String password): boolean
Company(Company company)
ateCompany(Company company)
teCompany(int companyID)
AllCompanies(): ArrayList<Company>
OneCompany(int companyID): Company
Customer(Customer customer)
eteCustomer(Customer customer)
teCustomer(int customerID)
AllCustomers(): ArrayList<Customer>
OneCustomer(int customerID): Customer
CompanyFacade
- companyID: int
+ constructor
+ login(String email, String password): boolean
+ addCoupon(Coupon coupon)
+ updateCoupon(Coupon coupon)
+ deleteCoupon(int couponID)
+ getCompanyCoupons(): ArrayList<Coupon>
+ getCompanyCoupons(Category category): ArrayList<Coupon>
+ getCompanyCoupons(double maxPrice): ArrayList<Coupon>
+ getCompanyDetails(): Company
CustomerFacade
- customerID: int
+ constructor
+ login(String email, String password): boolean
+ purchaseCoupon(Coupon coupon)
+ getCustomerCoupons(): ArrayList<Coupon>
+ getCustomerCoupons(Category category): ArrayList<Coupon>
+ getCustomerCoupons(double maxPrice): ArrayList<Coupon>
+ getCustomerDetails(): Customer

חלק ו' – בניית מחלקה המאפשרת כניסת Clients ולפיכך החזרת הלוגיקה העסקית המתאימה

מחלקה זו הינה מחלקת Singleton בשם LoginManager המכילה פונקציית Login שמאפשרת לכל אחד משלושת סוגי ה-Clients להתחבר למערכת.

ראשית, יש לבנות Enum בשם ClientType המתאר את כל אחד מסוגי ה-Clients:

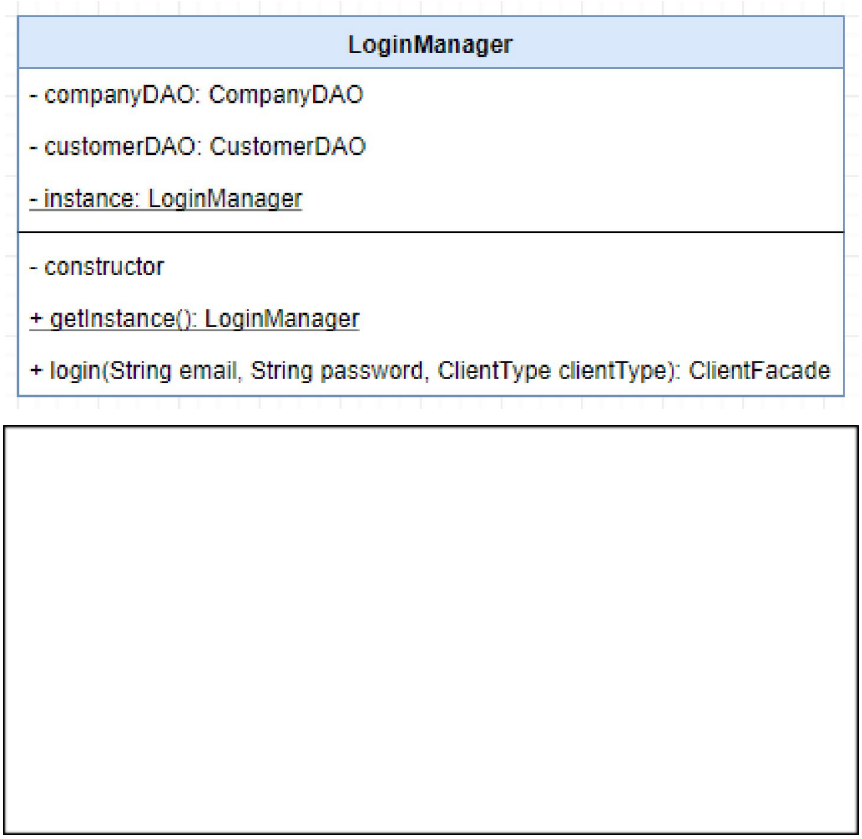


פונקציית ה-Login תקבל אימייל, סיסמה ומשתנה מסוג ClientType. על הפונקציה לבדוק האם פרטי ה-Login נכונים בהתאם ל-ClientType. אם פרטי ה-Login שגויים – הפונקציה תחזיר null. אם פרטי ה-Login נכונים – הפונקציה תחזיר את מחלקת ה-Facade המתאימה:

- עבור Administrator שביצע כניסה נכונה – יוחזר אובייקט AdminFacade.

- עבור Company שביצע כניסה נכונה – יוחזר אובייקט CompanyFacade.
- עבור Customer שביצע כניסה נכונה – יוחזר אובייקט CustomerFacade.
- עבור כל כניסה שגויה – יוחזר null.

להלן סכמת מחלקת ה-LoginManager:



חלק ז' – בניית Job יומי למחיקת קופונים שפג תוקפם מהמערכת

Job הינו תהליך שרץ ברקע באופן קבוע ומבצע פעולה כלשהי. Job יכול לבצע את הפעולה שלו ללא הפסקה, או שהוא יכול לבצע אותה בזמנים ספציפיים, לדוגמה פעם בשעה או פעם ביום או פעם בשבוע וכו' – תלוי בפעילות שהוא אמור לבצע. לצורך ביצוע פעולה בזמנים ספציפיים אפשרי לבדוק בכל שנייה או דקה או שעה וכו', האם הזמן הדרוש כבר הגיע – ואם כן – לבצע את הפעולה הדרושה.

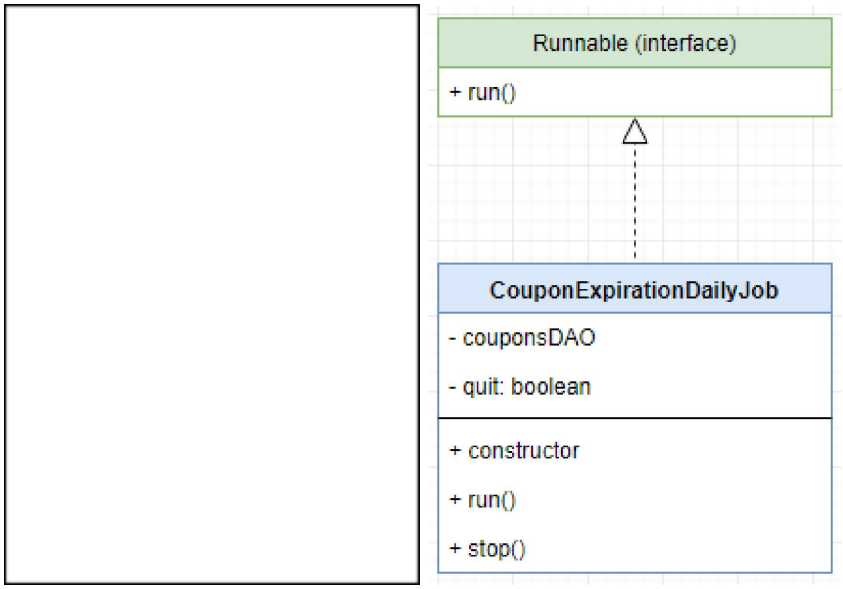
מימוש ה-Job חייב להיות ע"י Thread נפרד, שכן הוא צריך לרוץ במקביל לפעילות המערכת.

בחלק זה דרוש בניית Job יומי, כלומר שיתבצע פעם אחת בכל יום, ושימחק קופונים שפג תוקפם. לצורך מחיקת קופון שפג תוקפו – יש למחוק את הקופון מטבלת הקופונים ויש למחוק בנוסף גם את היסטוריית הקנייה של הקופון.

במחלקת ה-Job יש לבנות פונקציה המתחילה את ה-Job ופונקציה הגורמת ל-Job להסתיים.

על ה-Job להתחיל לעבוד בתחילת התוכנית ולהסתיים בסוף התוכנית.

להלן סכמת ה-Job למחיקת קופונים שפג תוקפם מהמערכת:



חלק ח' – בניית מחלקת Test להדגמת יכולות המערכת והפעלתה מה-main

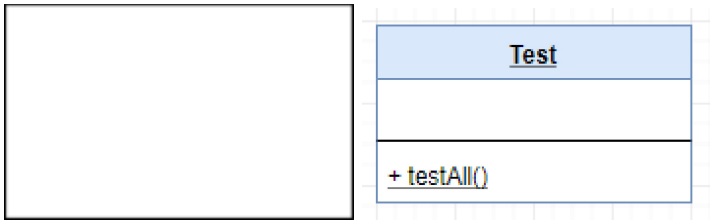
מחלקה זו נקראת Test ומכילה פונקציה סטטית אחת בשם testAll שתפקידה לבדוק את כל המערכת ע"י קריאה לכל לוגיקה עסקית שבנינו. אין צורך לקבל דבר מהמשתמש. כל הבדיקות צריכות להיות Hard-Coded. בהמשך – המידע יועבר למערכת ע"י אתר האינטרנט שיבנה בשלב הבא.

על הפונקציה testAll לבצע את הפעולות הבאות:

- הפעלת ה-Job היומי.
- התחברות ע"י ה-LoginManager כ-Administrator, קבלת AdminFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- התחברות ע"י ה-LoginManager כ-Company, קבלת CompanyFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- התחברות ע"י ה-LoginManager כ-Customer, קבלת CustomerFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- הפסקת ה-Job היומי.

את כל הפעולות הללו יש להגדיר בתוך try-catch אחד ולהציג את הודעת השגיאה במקרה של חריגה.

להלן סכמת מחלקת ה-Test:



המחלקה הראשית של המערכת הינה מחלקת ה-Program, המכילה את הפונקציית ה-main.

על פונקציית ה-main לבצע קריאה אחת לפונקציה testAll של מחלקת ה-Test.

להלן סכמת מחלקת ה-Program: