

## תוכן העניינים

2	כללי
4	שלב 1
21	שלב 2
22	שלב 3

## כללי

במסגרת הקורס משולב פרויקט מלווה המפורט במסמך.

הפרויקט מחולק לשלושה שלבים:

**שלב 1:** בניית ליבת המערכת. בשלב זה יוקם 'המוח' של המערכת. ליבת המערכת תהיה אחראית על קליטת נתונים, אכסונם במסד נתונים וניהולם. בשלב זה תיושמה מיומנויות התכנות והשימוש בתחביר בסיסי על מנת לייצר מערכת עובדת ולוגיקה משמעותית.

**שלב 2:** שכתוב מערכת הליבה לטכנולוגיות עדכניות בתעשייה. בשלב זה יתבצע מעבר לפלטפורמות מתקדמות ועדכניות בתעשייה כדוגמת Spring ו-Hibernate ושדרוג של הקוד.

**שלב 3:** שילוב יכולות Web וממשקי משתמש. שלב זה יתווסף לליבת המערכת ויאפשר ללקוחות להשתמש בה באמצעות דפדפנים. שלב זה כולל חשיפת הליבה ל-Web ובניית אתר אינטרנט בטכנולוגיות העדכניות ביותר בתעשייה על מנת להפוך אותה לזמינה עבור משתמשי קצה.

חלה חובת הגשה על כל שלושת השלבים.

### תיאור המערכת

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

למערכת יש גם לקוחות רשומים (Customers). הלקוחות יכולים לרכוש קופונים. 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 הלקוחות ומיצירה ועדכון קופונים חדשים ע"י החברות.

הגישה למערכת מתחלקת לשלושה סוגי Clients:

1. Administrator – ניהול רשימת החברות ורשימת הלקוחות.

2. Company – ניהול רשימת קופונים המשוכים לחברה.

3. Customer – רכישת קופונים.

## הנחיות כלליות

- העבודה 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 הצגת הפרויקט מופיעים 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 להציג את הפרויקט עובד, ללא שגיאות קומפילציה וללא קריסות.
- 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-APIs 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

Threads ,JDBC ,DAO ,Java Beans ,OOP

### תיאור:

בשלב זה יוגדר מסד הנתונים לאחסון ושליפת מידע אודות לקוחות, חברות וקופונים. מעל מסד הנתונים תוקם שכבת בידוד בשם DAO (Data Access Objects) שתאפשר עבודה נוחה מ-Java אל מסד הנתונים. 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 ה-Clients של המערכת – אדמיניסטרטור, חברה ולקוח, אשר יתחברו ע"י ביצוע Login.

### להלן שלבי הביצוע 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 שימוש בלוגיקה 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 (Don't Repeat Yourself – DRY).
- יש להשתמש ב-Custom Exceptions (לדוגמה: CouponAlreadyExistsException) עבור חריגות ספציפיות של המערכת.
- יש להגדיר try-catch כללי במחלקת ה-Test המסוגל לתפוס את כל ה-Exceptions.
- מומלץ לעבוד עם מסד הנתונים MySQL.
- חובה לחלק את המחלקות ל-Packages מתאימים, בעלי שמות משמעותיים.

## חלק א' – הגדרת מסד הנתונים ובניית הטבלאות

להלן רשימת הטבלאות הדרושות:

### 1. COMPANIES – טבלת חברות:

- ID (int – מספור אוטומטי – מפתח ראשי)
- NAME (string – שם החברה)
- EMAIL (string – אימייל החברה)
- PASSWORD (string – סיסמת כניסה)

### 2. CUSTOMERS – טבלת לקוחות:

- ID (int – מספור אוטומטי – מפתח ראשי)
- FIRST\_NAME (string – שם פרטי)
- LAST\_NAME (string – שם משפחה)
- EMAIL (string – אימייל הלקוח)
- PASSWORD (string – סיסמת כניסה)

### 3. CATEGORIES – טבלת קטגוריות של קופונים:

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

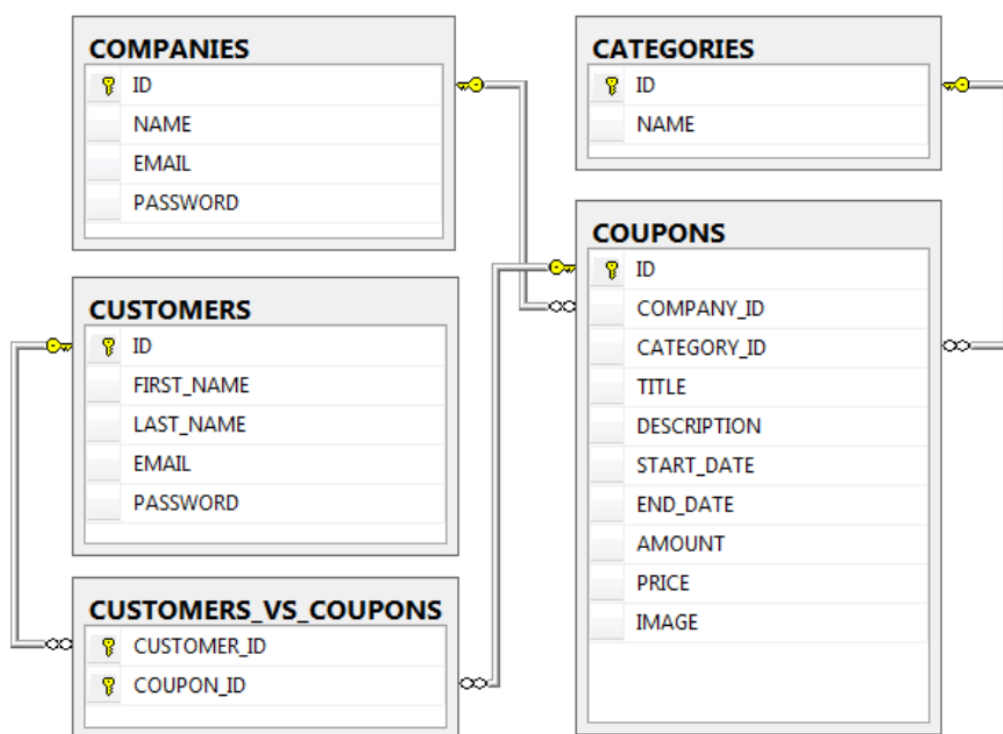
### 4. COUPONS – טבלת קופונים:

- ID (int – מספור אוטומטי – מפתח ראשי)
- COMPANY\_ID (int – מפתח זר לקוד החברה בטבלת החברות)
- CATEGORY\_ID (int – מפתח זר לקוד הקטגוריה בטבלת הקטגוריות)
- TITLE (string – כותרת 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)

5. CUSTOMERS\_VS\_COUPONS – טבלת לקוחות מול קופונים – מאפשרת לדעת אלו קופונים רכש כל לקוח ואלו לקוחות רכשו כל קופון:

- CUSTOMER\_ID (int – לקוחות בטבלת הלקוחות – מפתח זר לקוד הלקוח בטבלת הלקוחות)
- COUPON\_ID (int – קופונים בטבלת הקופונים – מפתח זר לקוד הקופון בטבלת הקופונים)

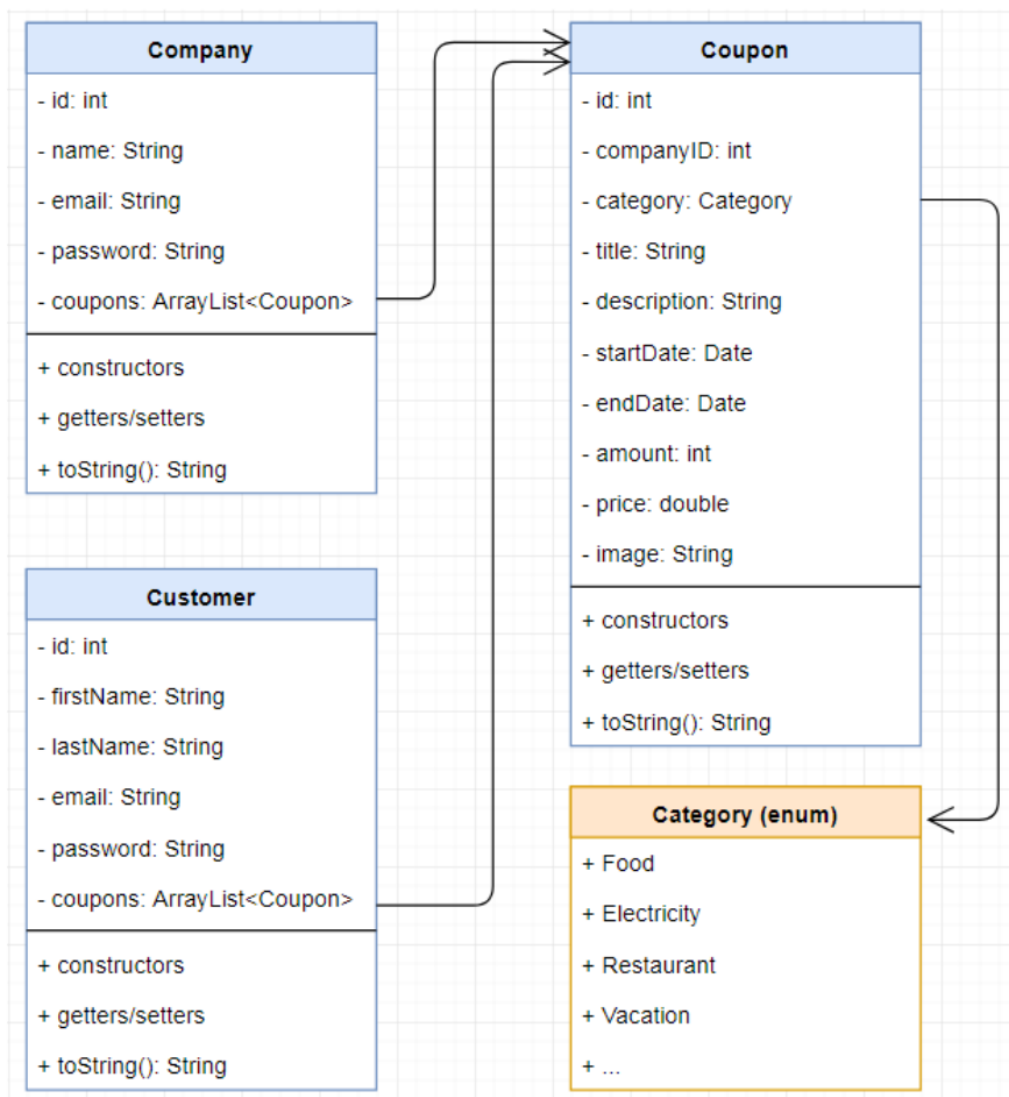
להלן סכמת הטבלאות והקשרים ביניהן:



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

Java Beans הינן מחלקות מידע טהור המייצגות את המידע המנוהל ע"י האפליקציה. מחלקות ה-DAO לדוגמה (Data Access Objects), מקבלות אובייקטי Java Beans ומתרגמות אותם לשאילתות SQL שנשלחות למסד הנתונים. כל טבלה עיקרית מיוצגת ע"י מחלקה משלה. הקשרים בין הטבלאות מיוצגים ע"י Collections מתאימים.

להלן סכמת מחלקות ה-Java Beans:





## 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 מסוג Set) 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 כלשהו שממתין (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:

ConnectionPool
- connections: Set<Connection>
- instance: ConnectionPool
- constructor
+ getInstance(): ConnectionPool
+ getConnection(): Connection
+ restoreConnection(Connection connection)
+ closeAllConnections()

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

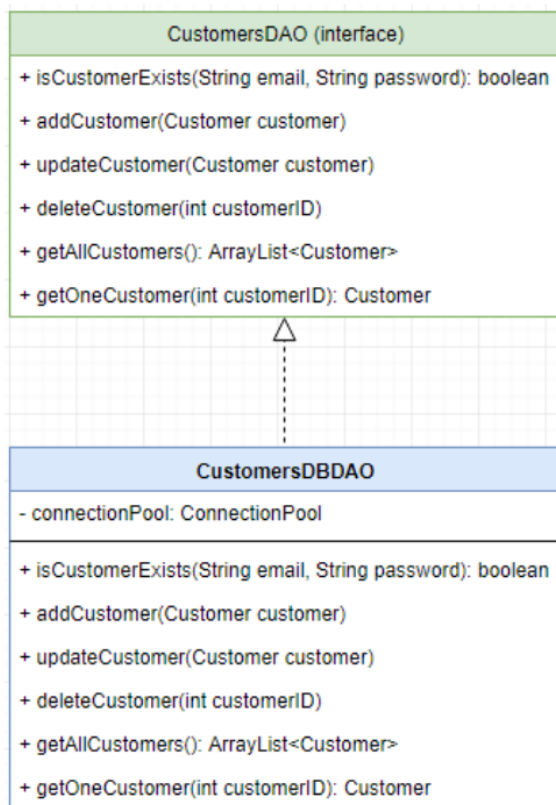
מחלקות DAO (Data Access Objects) אלו מחלקות המאפשרות לבצע פעולות CRUD (Create/Read/Update/Delete) כלליות על הטבלאות במסד הנתונים. מחלקות אלו לא מבצעות את הלוגיקה הקשורה לאפליקציה אלא אך ורק פעולות CRUD כלליות. מחלקות אלו מקבלות כארגומנטים אובייקטי Java Beans או ערכים פרימיטיביים פשוטים (string, int וכדומה), מייצרות מהן שאילתות SQL ומוציאות את השאילתות לפועל במסד הנתונים. כמו כן הן יכולות להחזיר בחזרה אובייקטי Java Beans בודדים, Collections של Java Beans או ערכים פרימיטיביים פשוטים. מחלקות אלו משתמשות 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 interface-ים המכילים את הפונקציונליות הדרושה עבור שכבת ה-DAO ונממש את הפונקציונליות הזו במחלקות ה-DAO השונות.

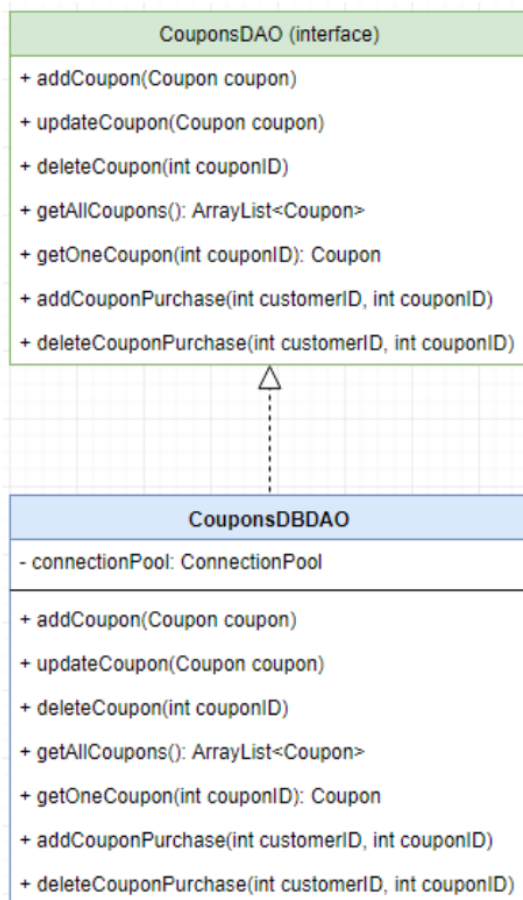
להלן סכמת ה-DAO של טבלת החברות:



להלן סכמת ה-DAO של טבלת הלקוחות:



להלן סכמת ה-DAO של טבלת הקופונים:



- ניתן להוסיף פונקציות CRUD כלליות נוספות ל-interface-ים ולמחלקות הנ"ל אם לדעתכם זה דרוש.
- במקרה של מידע שגוי הנשלח למחלקות אלו, חשוב לזרוק חריגות מתאימות (Custom Exceptions) המתארות את השגיאות בצורה ברורה.

## 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:

1. Administrator – המנהל הראשי של כל המערכת.
2. Company – כל אחת מהחברות שבמערכת.
3. Customer – כל אחד מהלקוחות שבמערכת.

לכל 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 בעזרת מחלקות ה-DAO.

לדוגמה, לצורך רכישת קופון ע"י לקוח (פעולה לוגית אחת שצריכה להיות במחלקת הלוגיקה העסקית של ה-Customers) עלינו לבצע את סדרת הפעולות הבאות ע"י שימוש במחלקות ה-DAO:

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

1. AdminFacade – מכילה את הלוגיקה העסקית של Administrator.
2. CompanyFacade – מכילה את הלוגיקה העסקית של Company.
3. 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 השונות יירשו את מחלקת הבסיס הזו וייצרו את רכיבי ה-DAO המתאימים עבורן.

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 כ-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.
- החזרת כל החברות.
- החזרת 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.
- החזרת כל הלקוחות.
- החזרת לקוח ספציפי לפי קוד לקוח.

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

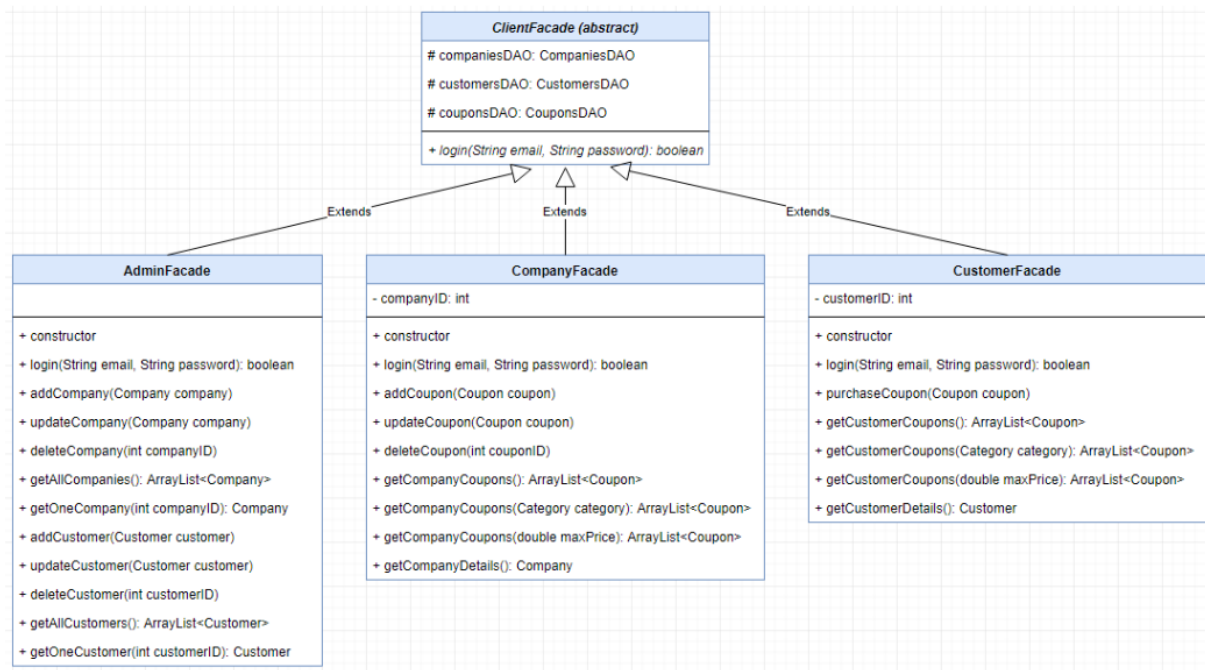
- כניסה למערכת.
  - יש לבדוק את פרטי ה-Login (אימייל וסיסמה) מול מסד הנתונים.
- הוספת קופון חדש.
  - אין להוסיף קופון בעל כותרת זהה לקופון קיים של אותה החברה. מותר להוסיף קופון בעל כותרת זהה לקופון של חברה אחרת.
- עדכון קופון קיים.
  - לא ניתן לעדכן את קוד הקופון.
  - לא ניתן לעדכן את קוד החברה.
- מחיקת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



## להלן סכמת מחלקות ה-Facade:

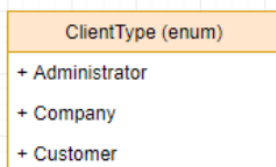


- המשתנה companyId במחלקת ה-CompanyFacade אמור להכיל את קוד החברה שביצעה Login.
- המשתנה customerID במחלקת ה-CustomerFacade אמור להכיל את קוד הלקוח שביצע Login.
- ניתן להוסיף פונקציות עזר נוספות למחלקות הנ"ל אם לדעתכם זה דרוש.
- במקרה של מידע שגוי הנשלח למחלקות אלו, חשוב לזרוק חריגות מתאימות (Custom Exceptions) המתארות את השגיאות בצורה ברורה.

## חלק ו' – בניית מחלקה המאפשרת כניסת Clients ולפיכך החזרת מחלקת ה-Facade המתאימה

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

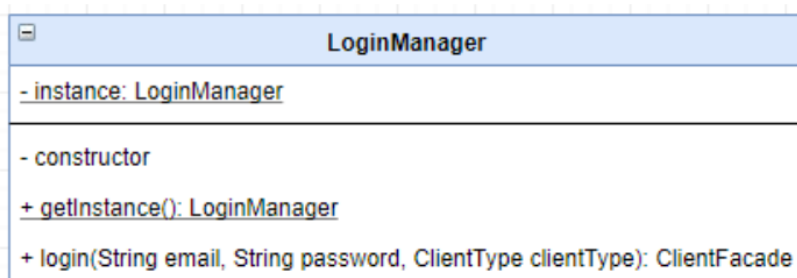
ראשית, יש לבנות Enum בשם ClientType המתאר את כל אחד מסוגי ה-Clients:



שנית, יש לבנות במחלקה LoginManager פונקציית Login שתקבל 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/דקה/שעה וכדומה – האם הזמן הדרוש כבר הגיע, ואם כן – לבצע את הפעולה הדרושה.

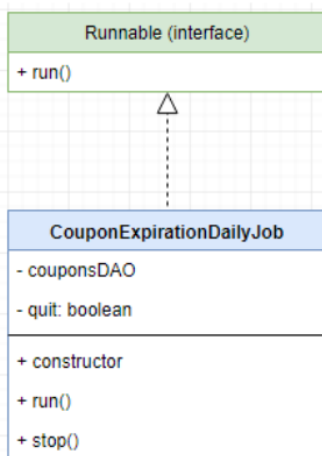
מימוש ה-Job חייב להיות ע"י Thread נפרד, שכן הוא צריך לרוץ ללא הפסקה במקביל לשאר פעילויות המערכת.

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:



- יש להפעיל את ה-Job עם עליית המערכת (תחילת התוכנית).
- יש להפסיק את ה-Job עם סגירת המערכת (סיום התוכנית).

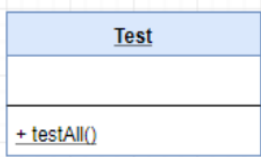
## חלק ח' – בניית מחלקת Test להדגמת יכולות המערכת והפעלתה מה-main

מחלקה זו נקראת Test ומכילה פונקציה סטטית בשם testAll שתפקידה לבדוק את כל המערכת ע"י קריאה לכל אחת מפונקציות הלוגיקה העסקית שבניתם. אין צורך לקבל דבר מהמשתמש. כל הבדיקות צריכות להיות Hard-Coded. בהמשך – המידע יועבר למערכת ע"י אתר האינטרנט.

על הפונקציה testAll לבצע את הפעולות הבאות:

- הפעלת ה-Job היומי.
- התחברות ע"י ה-LoginManager כ-Administrator, קבלת אובייקט AdminFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- התחברות ע"י ה-LoginManager כ-Company, קבלת אובייקט CompanyFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- התחברות ע"י ה-LoginManager כ-Customer, קבלת אובייקט CustomerFacade וקריאה לכל פונקציית לוגיקה עסקית שלו.
- הפסקת ה-Job היומי.
- סגירת כל ה-Connections (קריאה לפונקציה closeAllConnections של ה-ConnectionPool).

את כל הפעולות הללו יש להגדיר בתוך try-catch אחד ולהציג את הודעת השגיאה במקרה של חריגה. אין להציג הודעות של Stack Trace על המסך, אלא רק הודעות שגיאה ברורות המגיעות מה-Custom Exceptions. להלן סכמת מחלקת ה-Test:



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

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

להלן סכמת מחלקת ה-Program:

