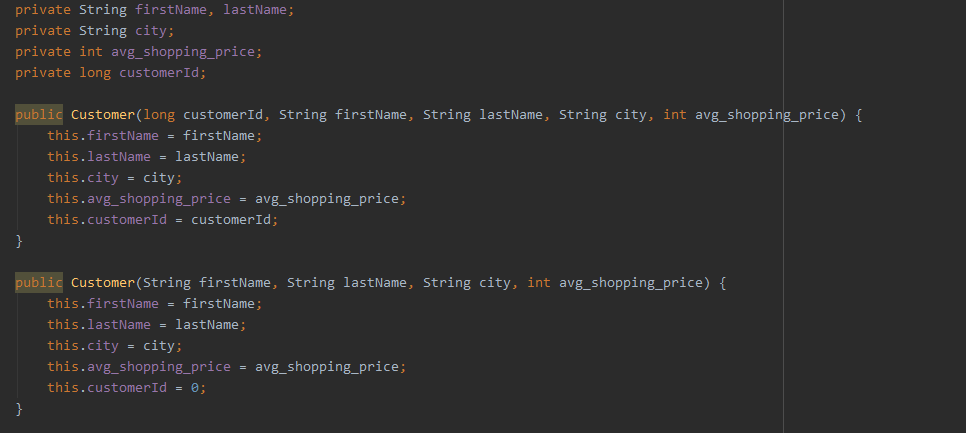
**פיתוח מערכות ניידות – עבודת הגשה 2**

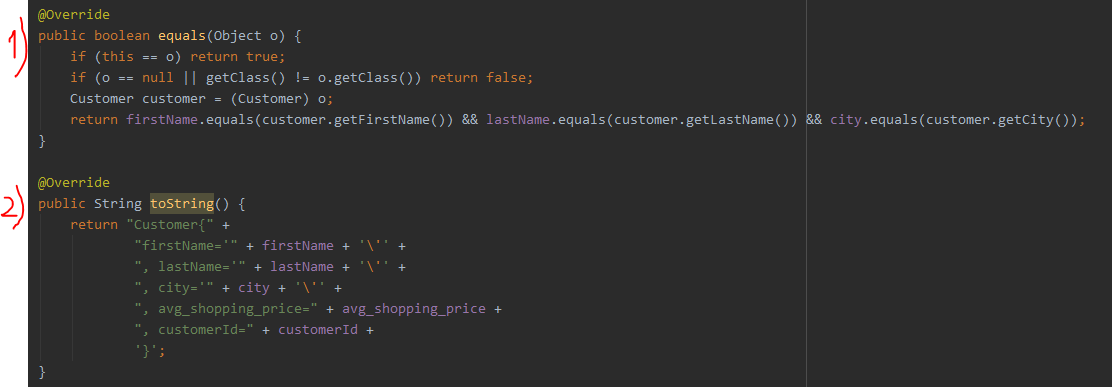
**מגישים:**ליאור וקנין – 208574046  
מורדי דבח – 203507107  
ישראל חדד – 305112005

**מחלקת Customer.java:**

יצרנו מחלקה בשם Customer שמכילה שני בנאים ואת השדות הבאים:  
מספר (מפתח ראשי מתקדם) שמציין לקוח (customer\_id), שם פרטי, שם משפחה, עיר מגורים, וממוצע מחיר קניות כמספר שלם.



יצרנו שני בנאים. הבנאי הראשון מקבל את מספר הלקוח וקובע את מספר הלקוח הנוכחי (this) להיות מספר הלקוח שהוא קיבל. הבנאי השני קובע את מספר הלקוח להיות 0.  
מספר הלקוח יהיה מפתח ראשי מתקדם (Auto increment) בSQL.



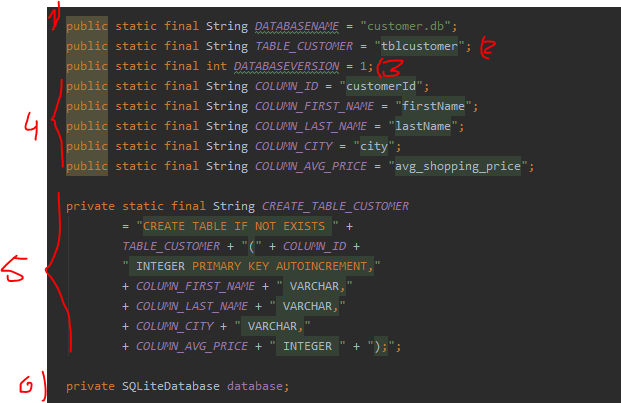
לאחר מכן ביצענו override למתודות toString() וequals.

1. קבענו שמאפיין הזיהוי של לקוח יהיה שם פרטי, שם משפחה ועיר בלבד. כלומר, ללא ממוצע מחיר קניות.
2. מתודת הtoString().

לבסוף יצרנו getters וsetters לכל משתני המחלקה של Customer.

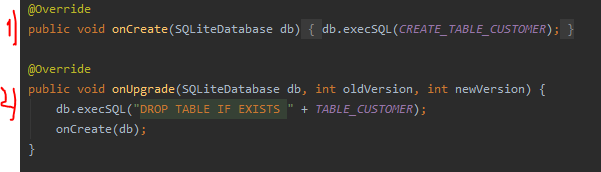
**מחלקת CustomerHelper.java:**

מחלקה זו מטפלת בכל הנוגע לSQL, בנייה ומחיקת הטבלאות, שאילתות select, update, delete וכדומה.  
מחלקה זו יורשת (extends) ממחלקת SQLiteOpenHelper.

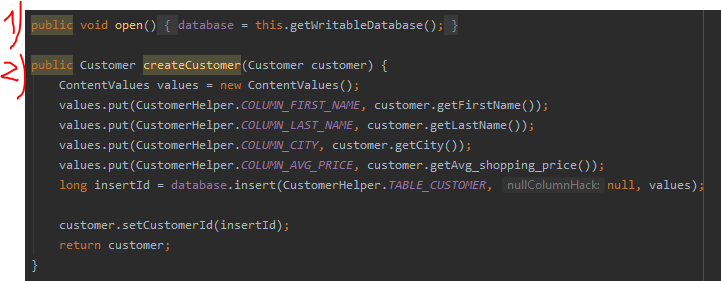


1. שם הdatabase.
2. שם הטבלה, טבלת הלקוחות.
3. גרסת הdatabase.
4. עמודות הטבלה, שכמו שציינו אלה הם משתני המחלקה של מחלקת Customer.
5. שאילתת יצירת הטבלה:  
   השאילתה אומרת ליצור את טבלת הלקוחות (אם היא לא קיימת), קביעת הcustomer\_id להיות המפתח הראשי של הטבלה מטיפוס Integer. ולקבוע את כל שאר עמודות הטבלה. כולן בתור משתנה מחרוזת (VARCHAR), חוץ מהממוצע שהוא Integer.
6. משתנה שמחזיק את הdatabase, בינתיים הוא שווה לnull, והdatabase עוד לא נוצר ועוד לא פתוח לקריאה או לכתיבה.

לאחר מכן יצרנו שני בנאים (הבנאים קוראים לsuper בלבד).  
ובנוסף, שיכתבנו את מתודות onCreate וonUpdate.



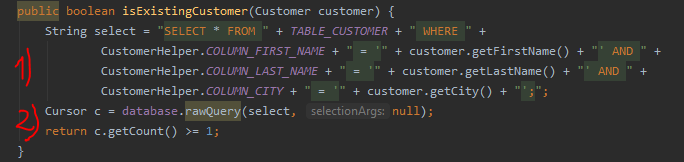
1. מקבלת את הdatabase ופותחת אותו, על ידי שאילתת יצירת הטבלה שהסברנו.
2. מתודה זו אחראית לעדכון הdatabase. ברגע שיש עדכון, מוחקים את הטבלה הישנה (כדי שלא יהיה המון עותקים של הטבלה בזיכרון וסתם יתפוס מקום), ויוצרים את הטבלה על ידי קריאה לonCreate.



1. מאתחלים את הdatabase ופותחים אותו לכתיבה (ולקריאה).
2. שאילתה ליצירת רשומה חדשה של לקוח בdatabase. עושה שימוש בinsert.  
   **עבור סעיף 2ז'.**

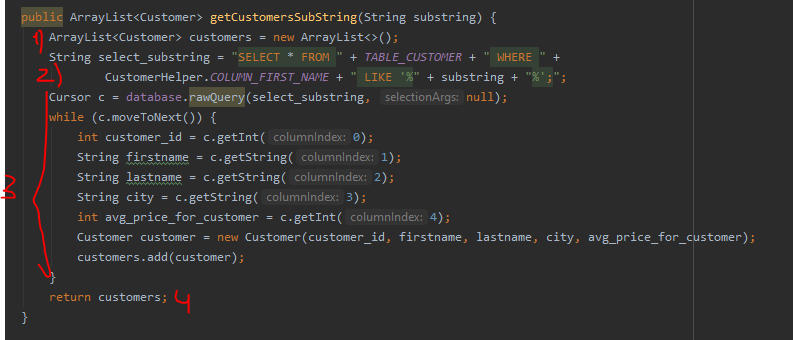
הפונקציה מקבלת אובייקט מטיפוס Customer, קוראת את השדות שלו, מכניסה את השדות שלו לאובייקט מטיפוס ContentValues, ומכניסה אותו לdatabase.  
נשים לב, שלא מכניסים customer\_id, כי ערך זה נקבע אוטומטית להיות primary key שגדל אוטומטית עם כל הכנסה.

נראה מי קורא לפונקציה הזאת כשנגיע לactivities.  
אנחנו צריכים לוודא שאין עוד customer כזה בdatabase לפני שמכניסים אותו לכן כתבנו פונקציה (שאילתה) חדשה.



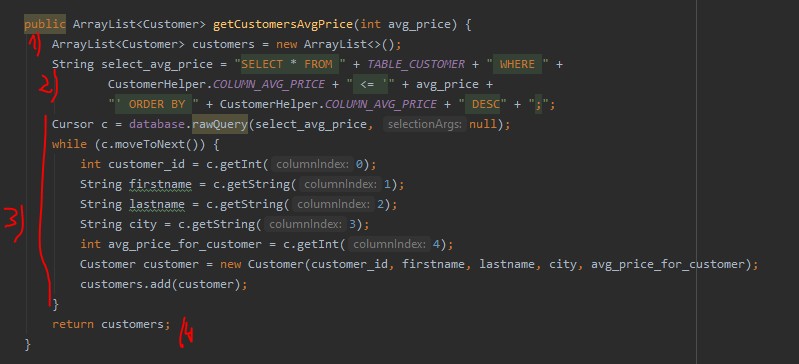
1. כתבנו פונקציה שמקבלת customer.  
   **עבור סעיף 2ז'.**  
   כתבנו שאילתה שבוחרת את כל הרשומות (ואת כל העמודות \*) מטבלת הלקוחות שבהן השדות (העמודות) שוות לפרטי הcustomer שהפונקציה קיבלה. (בודקים רק שם פרטי, משפחה ועיר).
2. לאחר מכן, יוצרים cursor עם השאילתה הזאת. ובודקים את גודל (getCount) של הcursor.  
   אם הוא גדול מ1 מחזירים true, כלומר כן קיים לקוח כזה ולכן לא נכניס לdatabase את הלקוח וניתן למשתמש הודעת שגיאה (נציג את זה בactivity המתאים). אחרת, מחזירים false וכן מכניסים את הלקוח לdatabase.

כעת נציג את שאר הסעיפים בעבודה.  
בחרנו להציג את כל הלקוחות על ידי ListView (לא למדנו את זה עדיין אבל אמרת שיש לנו יד חופשית בעיצוב). כדי להכניס איברים לListView צריך מערך (או ArrayList). בחרנו לבנות את הListView על ידי ArrayList של Customers. לכן השאילתות מעכשיו יחזירו ArrayList<Customer>.



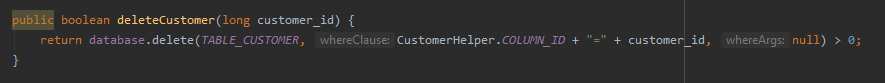
כתבנו פוקנציה שמטרתה למצוא את כל הלקוחות בעלי שם (פרטי) זהה או חלקי (substring) למחרוזת שהפונקציה קיבלה.  
**עבור סעיף 2ח'.**

1. יצרנו רשימה מקושרת של לקוחות ArrayList<Customer>.
2. כתבנו שאילתה, ששואלת תן לי את כל הלקוחות מטבלת הלקוחות בעלי שם פרטי שהוא דומה (על ידי operator LIKE) למחרוזת שהפונקציה קיבלה. LIKE ‘%substring%’ אומר שזה מתחיל או מסתיים בsubstring.
3. רצנו על המידע שהתקבל (על ידי cursor) יצרנו אובייקט customer, הכנסנו אותו לרשימה המקושרת.
4. החזרנו את הרשימה המקושרת.  
   בהמשך נציג כיצד הצגנו את המידע.



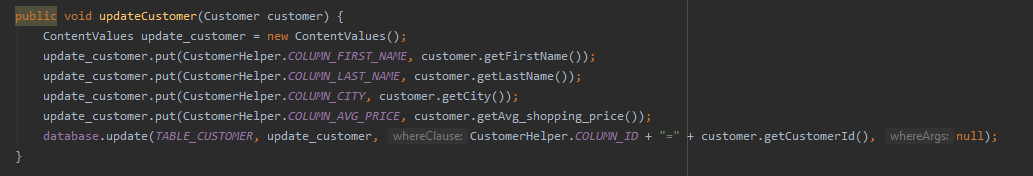
כתבנו פוקנציה שמטרתה למצוא את כל הלקוחות בעלי ממוצע מחיר קניות ששווה או קטן מהממוצע שהתקבל בפונקציה, בסדר יורד.  
**עבור סעיף 2ט'.**

1. יצרנו רשימה מקושרת של לקוחות ArrayList<Customer>.
2. כתבנו שאילתה, ששואלת תן לי את כל הלקוחות מטבלת הלקוחות שעמודת מחיר הקניות שלהן קטן או שווה לממוצע מחיר הקניות שהתקבל בפונקציה, בסדר יורד.
3. רצנו על המידע שהתקבל (על ידי cursor) יצרנו אובייקט customer, הכנסנו אותו לרשימה המקושרת.
4. החזרנו את הרשימה המקושרת.  
   בהמשך נציג כיצד הצגנו את המידע.



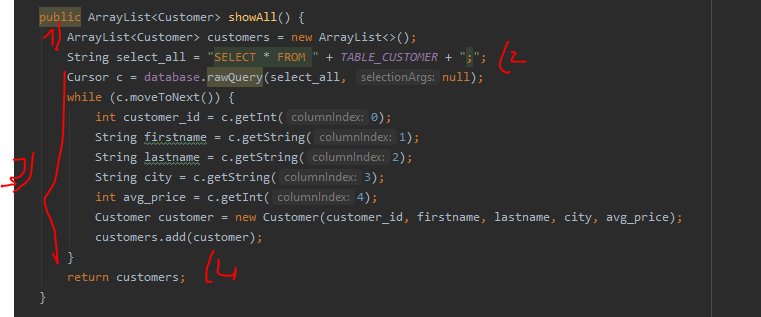
בשלב הבא יצרנו פונקציה שמקבלת מפתח של לקוח ומוחקת אותו מהטבלה.  
**עבור סעיף 2י'.**

הפונקציה השתמשה במתודה delete של הdatabase. מחיקת הלקוח שהid שלו שווה לid שהתקבל. לבסוף הפונקציה המחזירה true/false אם המחיקה הצליחה או לא (בודקים האם פונקציית הdelete של הdatabase החזירה תוצאה שהיא גדולה מ0).



בשלב הבא יצרנו פונקציה שמקבלת לקוח customer ומעדכנת אותו בטבלה.  
**עבור סעיף 2יא'.**

הפונקציה השתמשה במתודה update של הdatabase. עדכון הלקוח שהid שלו שווה לid של הלקוח שהתקבל. יוצרים אובייקט מטיפוס ContentValues, מכניסים לתוכו את השדות של הלקוח, ומעדכנים.

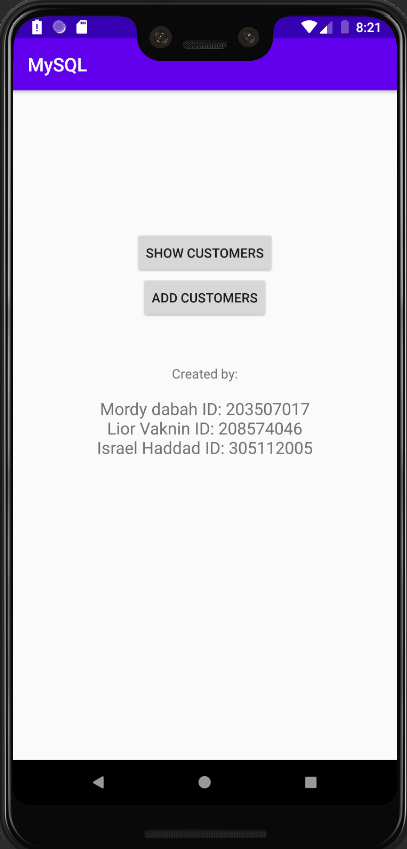


בנוסף, כתבנו שאילתה שמחזירה את כל הלקוחות.  
**עבור סעיף 4א'.**

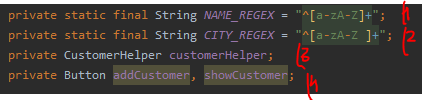
1. יצרנו רשימה מקושרת של לקוחות ArrayList<Customer>.
2. כתבנו שאילתה, ששואלת תן לי את כל הלקוחות מטבלת הלקוחות.
3. רצנו על המידע שהתקבל (על ידי cursor) יצרנו אובייקט customer, הכנסנו אותו לרשימה המקושרת.
4. החזרנו את הרשימה המקושרת.  
   בהמשך נציג כיצד הצגנו את המידע.

**מחלקת MainActivity.java:**

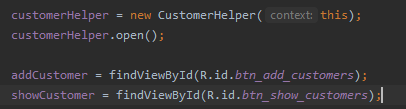
מחלקה זו היא activity (יורשת מAppCompactActivty) והיא אחראית לפרוס את activity\_main.xml מסך הבית של האפליקציה (המסך הראשון שעולה).



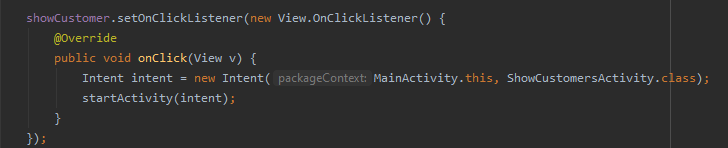
MainActivity פורסת layout של שני כפתורים. הראשון פותח activity להצגת הלקוחות וכל שאר הפונקציונליות של סעיפי 2ח' – 2יא'. השני פותח dialog להכנסת לקוח חדש.



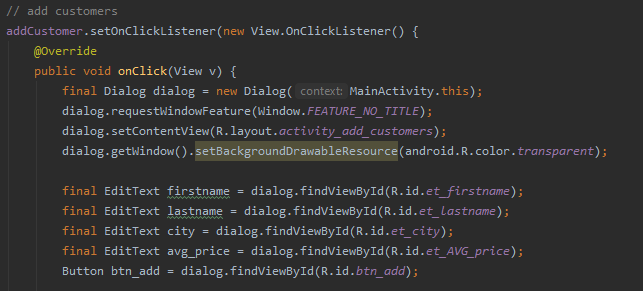
1. ביטוי רגולרי של שם פרטי ושם משפחה. רק אותיות a-z או A-Z ללא רווחים.
2. ביטוי רגולרי של עיר. רק אותיות a-z או A-Z עם רווחים.
3. משתנה של CustomerHelper.
4. משתנים של הכפתורים.



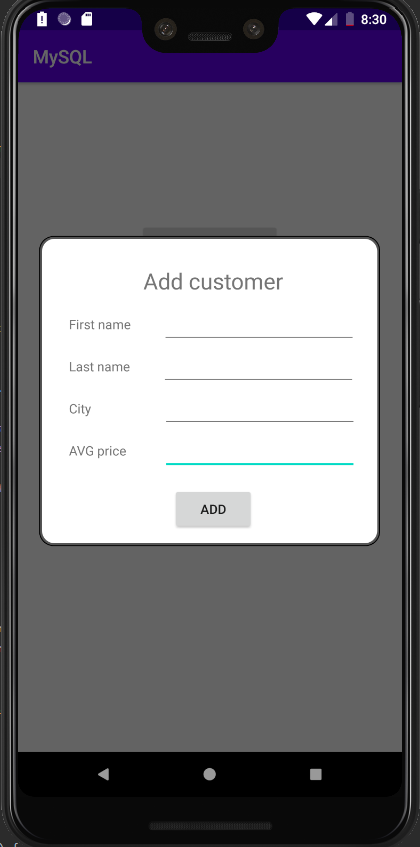
בפונקציית הonCreate אנחנו מאתחלים את אובייקט הcustomerHelper, ומבצעים open (מאתחלים את הdatabase ופותחים אותו לקריאה ולכתיבה). ומקשרים את האיברים (הכפתורים) מהxml.



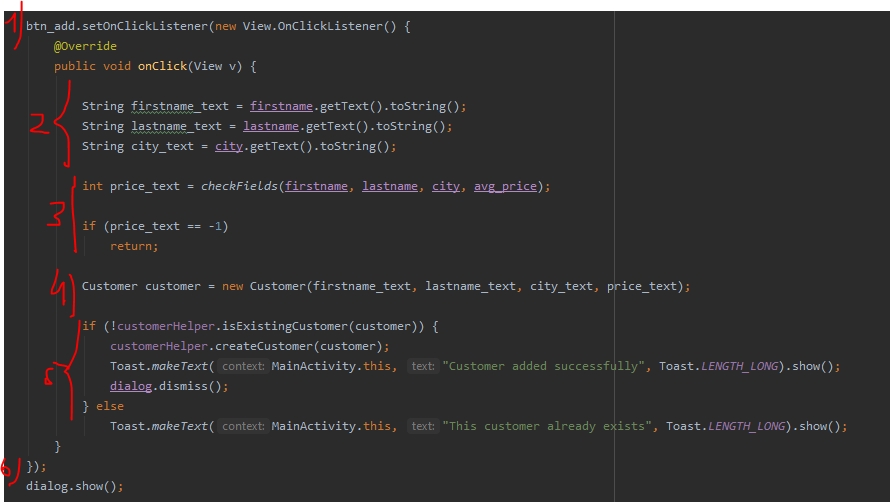
יוצרים listener לכפתור של הצגת הלקוחות. הוא פותח activity חדש שנציג בהמשך.



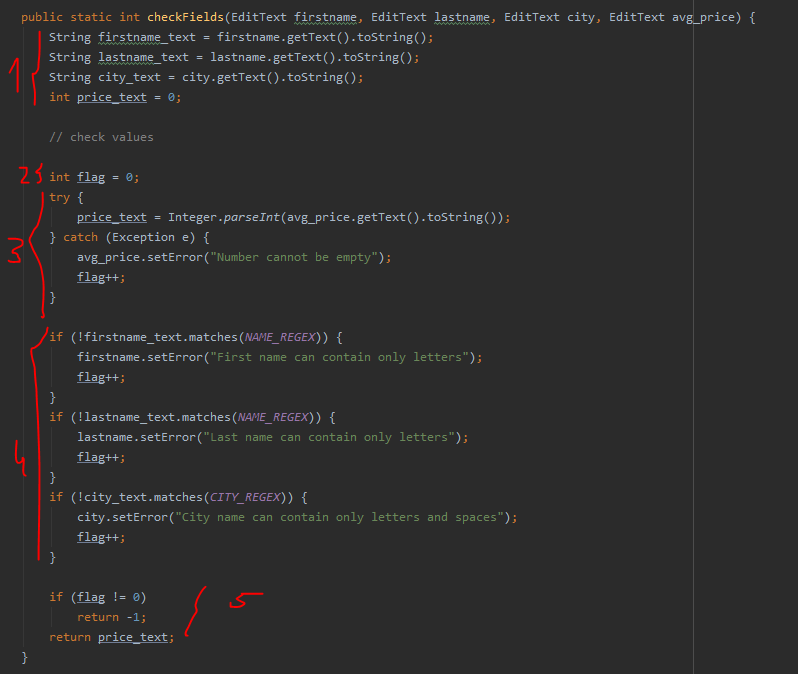
יוצרים listener לכפתור הוספת לקוח חדש. כפתור זה יותר דיאלוג חדש עם layout משלנו (custom layout). ומקשר את השדות של הlayout שמכיל ארבעה EditText וכפתור אישור אחד.



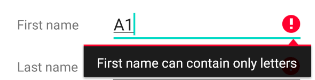
בחלון זה המשתמש מכניס את השדות הרלוונטיים ולוחץ על add להוספת לקוח חדש. (לחיצה על מחוץ לdialog יחזיר את המשתמש לMainActivity). יש לנו כפתור לכן צריכים להגדיר listener.



1. Listener לכפתור add של הdialog.
2. קריאת השדות (הנתונים מרכיבי הEditText).
3. מתבצעת בדיקה של השדות (פונקציית עזר), ואם השדות תקנים מקבלים את ערך מחיר הקניות הממוצע (כי צריכה להתבצע המרה מString לint). אם השדות לא תקינים, מוחזר -1 , עוצרים את הonClick על ידי return, ומציגים למשתמש הודעות שגיאה (זה קורה בפונקציה).
4. יצירת לקוח חדש מהנתונים שהתקבלו.
5. בדיקה שהלקוח הזה לא קיים כבר בdatabase על ידי שאילתה שכבר הסברנו.  
   אם לא קיים, תכניס (על ידי שאילתה שכבר הסברנו). אם קיים, תציג toast שלקוח זה כבר קיים.
6. הצגת הdialog.

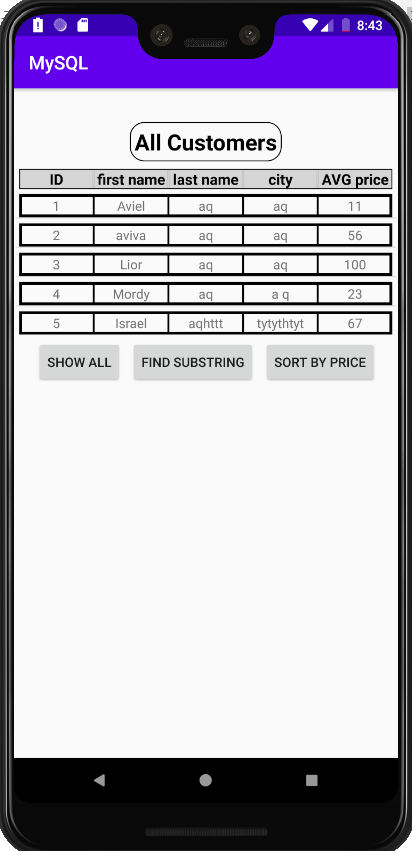


1. פונקציית העזר מקבלת שדות EditText. קוראת את המידע (String) מהם.
2. דגל לביקורת.
3. ניסיון ההמרה של שדה מחיר ממוצע ממחזרות למספר שלם. אם הצליח תקבע שהערך של price יהיה שווה לערך ההמרה. אחרת תגדיל את הדגל באחד ותן הודעת שגיאה על שדה הEditText.
4. בדיקה האם שם פרטי, שם משפחה, והעיר כולם מתאימים לביטויים הרגולריים שהצגנו בהתחלה. אם לא, תגדיל את הדגל באחד ותן הודעת שגיאה על שדה הEditText.
5. אם הדגל שונה מ0, תחזיר -1, אחרת, תחזיר את שדה הprice.



**מחלקת ShowCustomersActivity.java:**

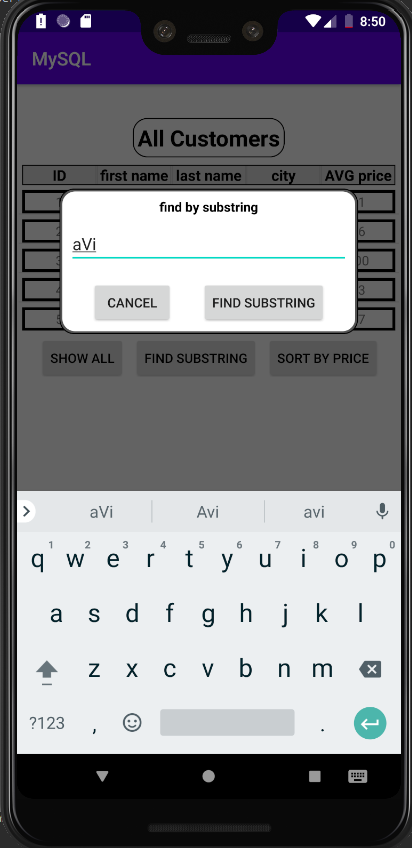
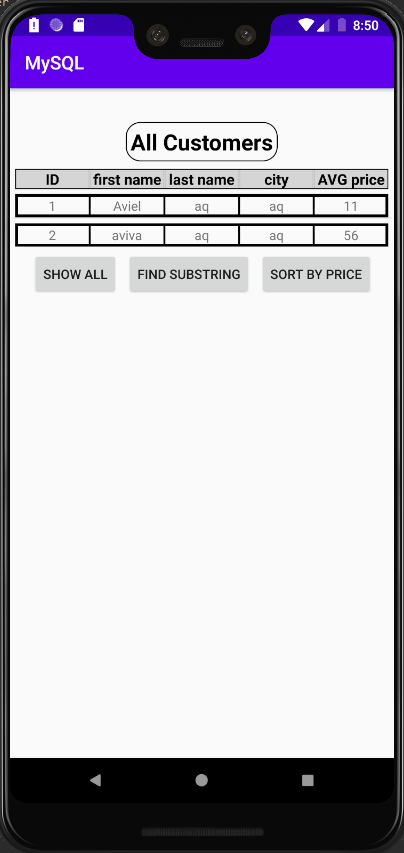
מחלקה זו היא activity (יורשת מAppCompactActivty) והיא אחראית לפרוס את activity\_show\_customer.xml.



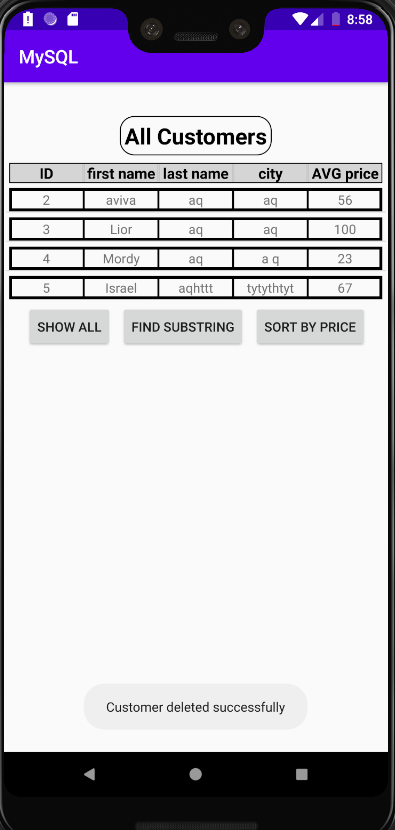
מסך זה מציג ListView של כל הלקוחות שנמצאים בdatabase על ידי שאילתת showAll שכבר הצגנו. הListView הוא עם custom layout, כלומר, כתבנו לו adapter משלו שנציג בהמשך.  
בנוסף, המסך מציג את כל הפונקציונליות של סעיפים 2ח' – 2יא' (כלומר, עדכון, מחיקה, הצגה של substring והצגה לפי price).

המסך מכיל שלושה כפתורים. כפתור show all, שמציג את כל הלקוחות שנמצאים בdatabase על ידי שאילתת showAll שכבר הצגנו (זה האופציה הדיפולטי, בהתחלה מוצג הכול).

כפתור find substring, שפותח dialog ששם המשתמש נדרש להכניס substring ואז הListView משתנה להכיל את כל הלקוחות עם שם פרטי שהוא שווה או חלקי (substring) למה שהמשתמש הכניס. על ידי שאילתה getCustomersSubString בCustomerHelper שכבר הצגנו.

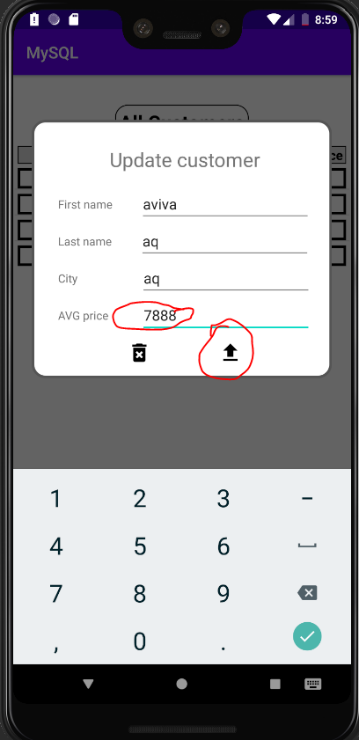


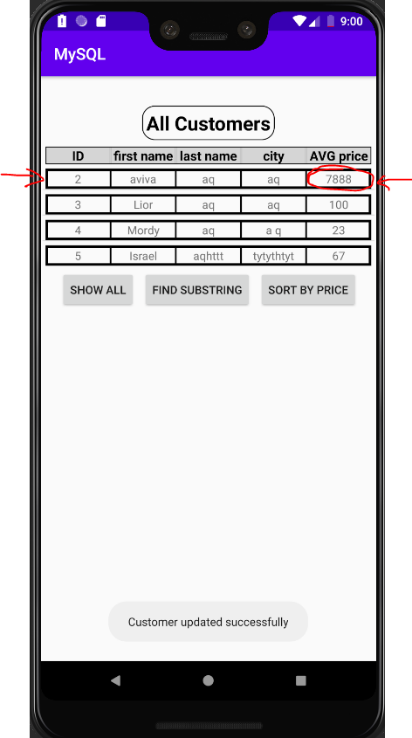
הכפתור השלישי הוא sort by price, שפותח dialog ששם המשתמש נדרש להכניס ממוצע מחיר קניות (מספר שלם, הגבלנו את המשתמש בEditText להכניס רק מספרים שלמים) ואז הListView משתנה להכיל את כל הלקוחות עם ממוצע מחיר קניות שהוא קטן או שווה למה שהמשתמש הכניס בצורה ממוינת מהגדול לקטן. על ידי שאילתה getCustomersAvgPrice בCustomerHelper שכבר הצגנו.

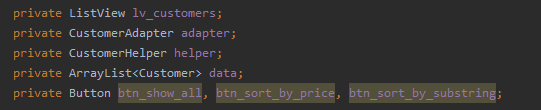
בנוסף לכך, הגדרנו listener על כל שורה בListView. זה כתוב בadapter של הListView שנציג בהמשך.  
בעת לחיצה על שורה בListView נפתח dialog, שנותן אפשרות למשתמש למחוק את הלקוח הספציפי שהוא לחת עליו, או לעדכן אותו (כדי לעדכן צריך להכניס שדות, ואנחנו בודקים את התקינות שלהן באמצעות אותה פונקציית עזר שהייתה בMainActivity, היא הייתה פומבית וסטטית, פונקציית checkFields).



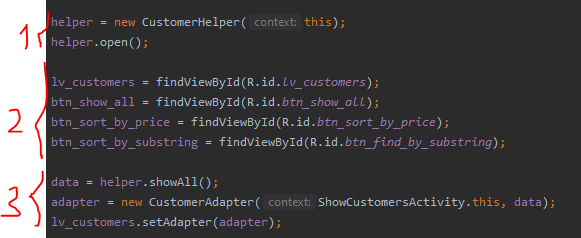
מחיקה:



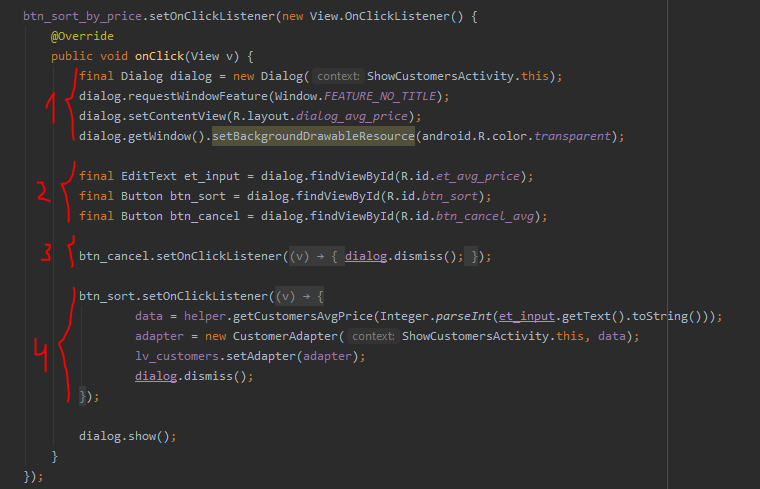
עדכון:



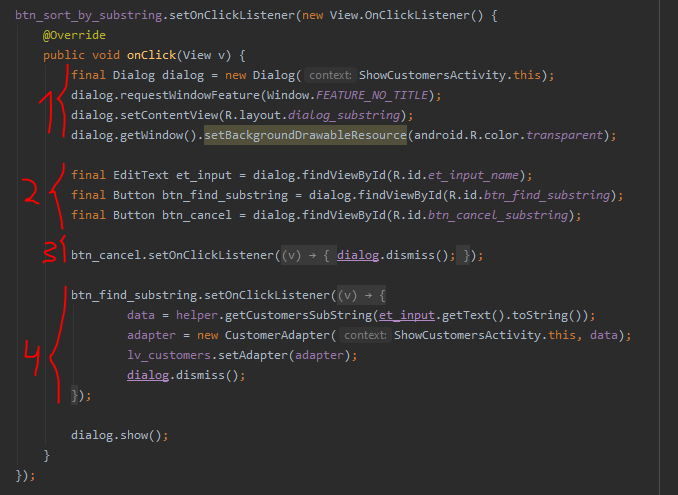
הגדנו בתור משתני מחלקה, את הListView את הadapter המתאים לו, את הCustomerHelper (שישקר אותנו לdatabase), ArrayList שיכיל את הdata (מידע זה ייכנס לadapter של הlist View, ובכל לחיצה על כפתור נשנה את הdata ונעדכן את הadapter שיציג את המידע הרלוונטי). ושלושה כפתורים.



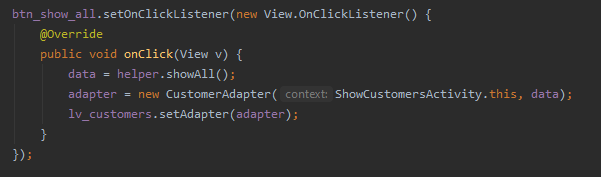
1. יצירת הcustomer Helper, ויצירה, אתחול ופתיחה לכתיבה של הdatabase.
2. קישור (take over) על הרכיבים המתאימים בlayout.
3. בעזרת השאילתה שמציגה את כל הלקוחות (שכבר הצגנו) מעדכנים את הdata של הlistview, יוצרים את הadapter וקובעים את הadapter של הlistview להיות הadapter שיצרנו.



1. Listener לכפתור של מיון לפי ממוצע מחיר קניות (הכפתור השלישי משמאל).  
   יוצרים את הdialog.  
   **עבור סעיף 2ט'.**
2. קוראים את השדות ומקשרים את האיברים (EditText אחד ושני כפתורים, אישור וביטול).
3. קובעים את הlistener של כפתור הcancel. מבצעים dismiss לdialog.
4. קובעים את הlistener לכפתור האישור (המיון). מעדכנים את הdata להיות הArrayList<Customer> שהוחזר מהפונקציה getCustomersAvgPrice שכבר הסברנו (מהhelper). מעדכנים את הadapter ואת הlistview, וסוגרים את הdialog.



1. Listener לכפתור של מיון לפי substring (הכפתור האמצעי).  
   יוצרים את הdialog.  
   **עבור סעיף 2ח'.**
2. קוראים את השדות ומקשרים את האיברים (EditText אחד ושני כפתורים, אישור וביטול).
3. קובעים את הlistener של כפתור הcancel. מבצעים dismiss לdialog.
4. קובעים את הlistener לכפתור האישור (המיון). מעדכנים את הdata להיות הArrayList<Customer> שהוחזר מהפונקציה getCustomersSubString שכבר הסברנו (מהhelper). מעדכנים את הadapter ואת הlistview, וסוגרים את הdialog.



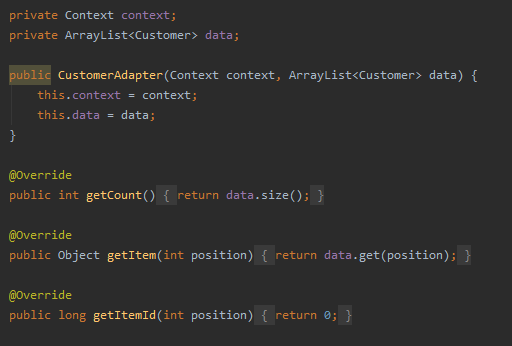
1. Listener לכפתור של show all (הכפתור הראשון משמאל).  
   **עבור סעיף 4א'.**מעדכנים את הdata להיות הArrayList<Customer> שהוחזר מהפונקציה showAll שכבר הסברנו (מהhelper). מעדכנים את הadapter ואת הlistview.



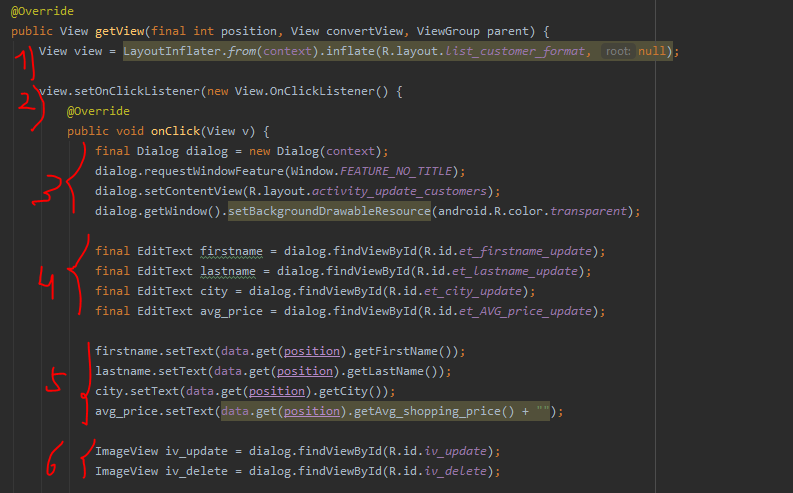
זה הlayout שפורסת מחלקת ShowCustomersActivity. בין היתר, יש כאן listview.

**מחלקת CustomerAdapter:**

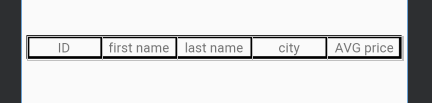
מחלקה פנימית בתוך מחלקת ShowCustomersActivity. מחלקה זו יורשת (extends) ממחלקת BaseAdapter. מחלקה זו נועדה כדי ליצור listview עם custom layout שיציג את כל הלקוחות בצורה של טבלה.



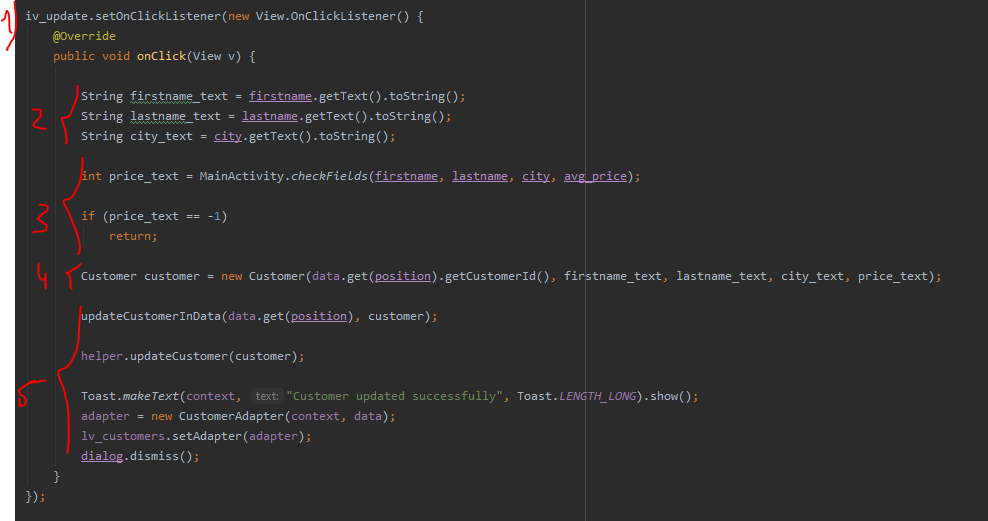
למחלקה זו יש שני משתני מחלקה, context ורשימה מקושרת של לקוחות ArrayList<Customer>.  
בבנאי אנחנו מאתחלים את משתני המחלקה.  
במתודת getCount מחזירים את גודל הרשימה המקושרת.  
במתודת getItem מחזירים את הcustomer שבמיקום המתאים ברשימה המקושרת.  
במתודת getItemId נחזיר 0.



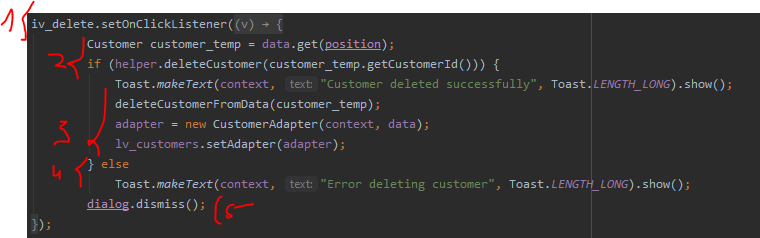
המתודה הרביעית היא getView. היא המתודה שמגדירה כל שורה בlistview.

1. מגדירים view להיות לפי הפריסה של list\_customer\_format.xml.  
     
   בהמשך נראה כיצד ניקח כל TextView ונשנה לו את הערך להיות הערך המתאים של כל לקוח (לכן לא צריך BufferString).
2. יוצרים listener על view. דבר זה יטפל לנו שבעת לחיצה על שרה בlistview ייפתח dialog ששן המשתמש יוכל לעדכן את הלקוח או למחוק אותו.
3. יוצרים את הdialog.
4. מקשרים את השדות (take over) יש ארבעה EditText ושני ImageView (סעיף 6)
5. מבצעים שינוי טקסט של השדות להכיל את הפרטים של הלקוח (שעליו לחצנו). ניתן לדעת איזה שורה נלחצה לפי משתנה position, ויודעים באיזה customer מדובר, כי יש את הdata, לכן ניתן לבצע data.get(position) לקבלת הcustomer.
6. קישור שני הImageView של עדכון ומחיקה.  
   

עדיין בתוך הonClickListener של הview, מגדירים listeners לכל ImageView.



1. הגדרת listener לעדכון לקוח.
2. קוראים את השדות המתאימים מהEditText.
3. בודקים את תקינות הערכים שהוכנסו (כמו בmain Activity) על ידי פונקציית העזר checkFields.
4. יצירת לקוח חדש מעודכן, עם id מתאים ללקוח שהיה.
5. עדכון הלקוח החדש בdatabase על ידי שאילתת העדכון שכבר הסברנו.  
   עדכון הadapter והlistview וסגירת הdialog.

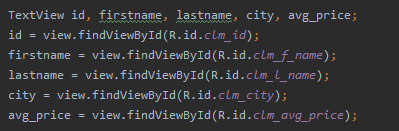


1. הגדרת listener למחיקת לקוח.
2. מוחקים את הלקוח שעליו לחצנו (יודעים מי לפי data.get(position)) על ידי שאילתת העדכון שכבר הסברנו.
3. אם זה הצליח, מעדכנים את הadapter ואת הlistview ומציגים toast הצלחה ללקוח.
4. אחרת, ומציגים toast ללקוח שהמחיקה לא הצליחה.
5. סגירת הdialog.

מחוץ לlistener על הview, מגדירים את התצוגה של כל שורה בlistview.

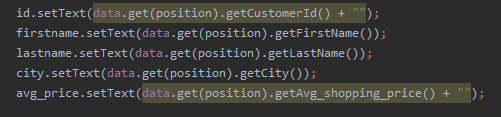






יוצרים TextView ומקשרים את השדות המתאימים.





לכל TextView שמים טקסט מתאים לפי הלקוח.  
קודם כל, הלקוח הוא customer = data.get(position). ככה ניתן לדעת כל רכיב בכל לקוח. כלומר, את שדה הid ואת השם וכדומה.

ככה כשהמסך של הlistview עולה, כל שורה מתאימה ללקוח מסוים.



לבסוף מחזירים את הview.

**יש להסביר בקובץ DOCX המצורף מתי עבדת עם BufferString ומדוע.**

בפרויקט שלנו לא עבדנו עם String Buffer.

בדוגמה שעשינו במהלך ההרצאה השתמשת בString Buffer להצגת כל הלקוחות. יצרת dialog וקבעת שהטקסט של הdialog (set Message) יהיה הbuffer.toString().

כדי להציג את הלקוחות, לא עבדנו עם טקסט לכן לא היינו צריכים String Buffer. במקום זאת, עבדנו עם listview עם custom layout. קישרנו כל רכיב בlayout בקוד, וקבענו שהטקסט שלו יהיה בהתאם לכל שדה של לקוח.

דבר זה קרה במחלקה הפנימית CustomerAdapter (עמוד למעלה).

לדוגמה, קישרנו את רכיב הTextView שנקרא (הid שלו) clm\_id. וקבענו שהטקט שלו יהי (עשינו setText) להיות הid של הלקוח הספציפי עליו אנחנו עומדים בlistview.

לכן, לא השתמשנו בString Buffer בעבודה הזאת.  
(אמרת שיש יד חופשית בעיצוב, ובחרנו להציג את הלקוחות בlistview, לכן לא היה כל שימוש בString Buffer).