מעבדה מספר 3

היכרות עם JavaFX, תוכנית הכוללת Graphical user interface (GUI)

1. **JavaFX**

מטרת מעבדה זו היא להכיר את JavaFX – קבוצה של חבילות גרפיות ומדיה המאפשרת למפתח תוכנה לתכנן, לכתוב ולבדוק אפליקציות עם ממשק משתמש נוח וידידותי שיכול לרוץ על פלטפורמות שונות. ב־Java יש חבילות נוספות ישנות יותר לכתיבת GUI: AWT ו־Swing.

הלימוד כולל הסברים כללים על עקרונות של מרכיבי תוכנה גרפית, מבנה המחלקות ב־JavaFX וכן הסברים מעשיים כיצד לכתוב ממשק גרפי. החבילות הגרפיות הן מקיפות וכוללות מחלקות רבות עם מגוון אפשרויות לתכנון ומימוש ממשקים עם אלמנטים רבים ושונים. במעבדה נדגים חלק קטן בלבד. תוכלו ללמוד ולהעמיק יותר ב־JavaFX בעזרת מדריכים שונים, שניתן להגיע אליהם ע"י חיפוש קצר באינטרנט.

2. **Building a GUI with JavaFX Panes**

הערה חשובה – כאשר משתמשים בצורה נרחבת עם ספריות (כמו JavaFX) תהליך ה-debugging עשוי להיות מורכב. כאשר מתעוררת שגיאה של זמן ריצה המעקב על המחסנית עשוי להיות ארוך כי הקוד כולל קריאות רבות ומקוננות למתודות של מחלקות הספרייה. ניתן לקרוא יותר על תהליך ה־debugging בקישור הבא:

http://cs.brown.edu/courses/cs015/docs/JavaFXGuide.pdf

3. **הוראות התקנה**

מדריך וידאו:

https://www.youtube.com/watch?v=ZfaPMLdgJxQ&ab_channel=BoostMyTool

מדריך ליצירת פרויקט ראשון עם JavaFX בעזרת Maven ב-IntelliJ:

https://www.youtube.com/watch?v=d4FMEgjSdEw&ab_channel=Randomcode

כיוון שאנחנו משתמשים ב־maven, נוכל להתחיל לעבוד במהירות ע"י יצירת פרויקט חדש עם ה־archetype הבא:

- Group Id: org.openjfx
- Artifact Id: javafx-archetype-fxml (recommended) or javafx-archetype-simple
- Version: 0.0.6

ויש לדאוג לשנות את גרסאות ה־dependencies של JavaFX ל־21.0.1 (או בהתאם למש שיש לכם).

בנוסף יש להתקין את התוכנה **JavaFX Scene Builder** - ניתן להורידה מהקישור הבא:

/https://gluonhq.com/products/scene-builder

מרכיבי תוכנה גרפית

כתיבת קוד ליצירת ממשק גרפי כוללת שלושה מרכיבים:

- מרכיב 1 - בחירת אלמנטים מסוגים שונים (כפתורים, רשימות, שדות טקסט, כותרות) שיופיעו על המסך/מסכי התוכנית.
- מרכיב 2 - הארגון הדו־ממדי של האלמנטים ופריסתם על המסך (בתחילת התוכנית ולאחר שינוי גודל החלון ע"י המשתמש/סגירתו/פתיחתו מחדש).
- מרכיב 3 - ההתנהגות הדינמית של האלמנטים בתגובה לפעולות של המשתמש/ת ("אירועים": הקלדה, הקלקה, גרירה).

4. Event Driven Programing

ההתנהגות הדינמית של האלמנטים בתגובה לפעולות של המשתמש ממומשת בעזרת Event Driven Programing. הרעיון המרכזי הוא שהתוכנית ממתינה לאירועים והיא מגיבה להם כאשר הם מתרחשים.

נניח שמשתמש מבצע פעולה כמו לחיצה על מקש במהלך עבודתו עם תוכנית בעלת ממשק גרפי. הלחיצה יוצרת אות הנשלח מלוח המקשים למחשב. מערכת ההפעלה מזהה את האות ומודיעה ל־JVM שהתרחש אירוע של לחיצה על מקש מסוים. כתוצאה מכך, ה־JVM יוצר **אירוע** - אובייקט המכיל את כל המידע על האירוע הפיזי (הלחיצה עצמה) שהתרחש. אירוע זה יועבר לתוכנית והיא צריכה לטפל בו. כדי שזה יקרה יש לכתוב handler עבור האירוע. כלומר אם מעוניינים שהתוכנית תדפיס למשל הודעה לאחר לחיצה על מקש מסוים, יש צורך לכתוב handler למקש שידפיס את ההודעה ובנוסף לבצע רישום (register) כדי לציין ש־handler הוא של המקש. ה־handler יתבצע רק לאחר שהאירוע הפיזי יתרחש ואז יועבר לו כפרמטר אובייקט האירוע (שנוצר על ידי ה־JVM). ה־handler נמצא במצב המתנה - מעין "הקשבה" לאירוע המתאים, ולכן הוא נקרא מקשיבון (listener). במידה ולא כותבים handler מתאים או לחילופין לא רושמים אותו, התוכנית לא תגיב ללחיצה על המקש.

5. מחלקות JavaFX

JavaFX כוללת מחלקות שונות המאפשרות למתכנת לפתח ממשק גרפי מבוסס חלונות.

בחבילות גרפיות מבחינים בין שני סוגים של מחלקות:

- **Container** - מחלקה שהמופע שלה הוא רכיב UI שיכול להכיל בתוכו רכיבים אחרים, סוג של לוח ציור.
- **Leaf Nodes** (או **Component**) – מחלקה שהמופע שלה הוא רכיב בסיסי (כמו כפתור/מקש, רשימה או צורה גאומטרית) שאותו ניתן להציג על container.

<u>מחלקות Container ב-JavaFX</u>

<u>Stage</u> - מחלקה היוצרת את מסגרת החלון החיצוני ומספקת פונקציות בסיסיות של חלון כמו גבול, כפתור הקטנה, כפתור סגירה, יכולת של שינוי גודל החלון וכדומה. השם נגזר מהמושג

stage - בימת התיאטרון עליה מוצגת ההצגה. כך גם תפקיד ה־Stage באפליקציה: החלון הראשי בו יוצגו כל האלמנטים הגרפיים.

Scene - לרוב ה־Scene יכיל מספר קומפוננטות. במקרה הכללי Scene מורכבת מעץ הרכבה הנקרא scene graph. הדבר הכללי ביותר ש־Scene יכול להכיל הוא Node. כל מופע מטיפוס Node יכול לאחזר את ה־scene שלו על ידי קריאה למתודה ()getScene.

Group – מיכל שאינו מגדיר layout ספציפי לקומפוננטות שבתוכו (כולן ממוקמות בו באותה נקודה). לרוב הוא משמש כדי להפעיל אפקט או טרנספורמציה על קבוצה של קומפוננטות.

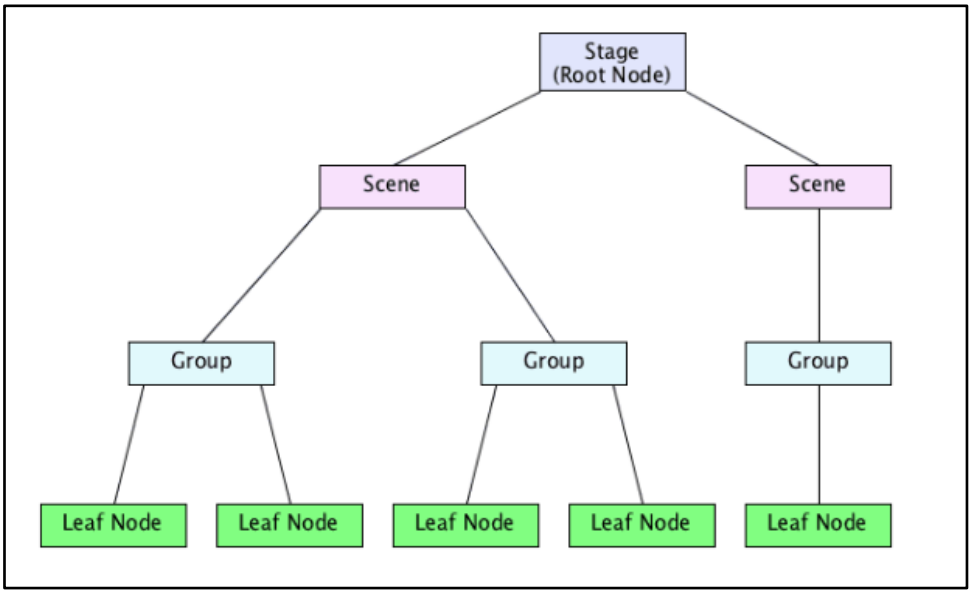
מחלקות Leaf Nodes

הם כפתורים, labels, שדות טקסט, צורות גרפיות או כל אלמנט גרפי שאינו מיכל בעצמו.

ארגון אלמנטי ה־User Interface (UI)

תוכנית JavaFX מארגנת את אלמנטי ה־UI בעץ שצומת השורש שלו הוא אובייקט מטיפוס **Stage** (החלון הראשי של התוכנית). הבנים של Stage הם צמתי Scene. ניתן להחליף Scenes ע"י זימון המתודה ()setScene על אובייקט מטיפוס Stage.

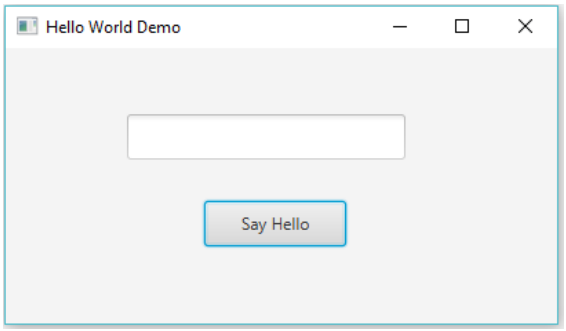
הגרף המובא בעמוד הבא מתאר ארגון האלמנטים השונים בתוכנית.



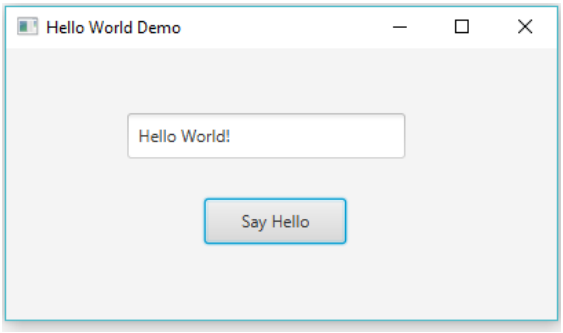
סכימה של ארגון האלמנטים של JavaFX - Stages, Scenes, Groups ו־Leaf Nodes.

6. תוכנית ראשונה – Hello World

בחלק הראשון של המעבדה נלמד לכתוב תוכנית פשוטה עם ממשק משתמש גרפי המציגה למשתמש את החלון הבא:



בלחיצה על כפתור 'Say Hello' תדפיס התוכנית את ההודעה 'Hello World' בשדה הטקסט שמעל לכפתור, כמוצג בחלון הבא:



למרות שהתוכנית פשוטה היא משלבת את שלושת המרכיבים של מודולי התוכנה המטפלים בחלק הגרפי של מערכת תוכנה:

- בחירת האלמנטים: חלון עם כפתור ותיבת טקסט.
- הארגון הדו־ממדי של האלמנטים ופריסתם על המסך
- התנהגות דינמית: לחיצה על הכפתור יוצרת אירוע שיש להגיב עליו.

את התוכנית נכתוב בשתי דרכים:

1. בעזרת JavaFX בלבד. במקרה זה יש לכתוב את כל הקוד בעזרת המחלקות של JavaFX, בדומה לדרך בה כותבים תוכנה עם ממשק גרפי בעזרת חבילות גרפיות ישנות יותר של Java (AWT ו־Swing).
2. בעזרת JavaFX ו־JavaFX Scene Builder המאפשר עריכה של המסכים בצורה נוחה ומייצר חלק מהקוד בצורה אוטומטית.

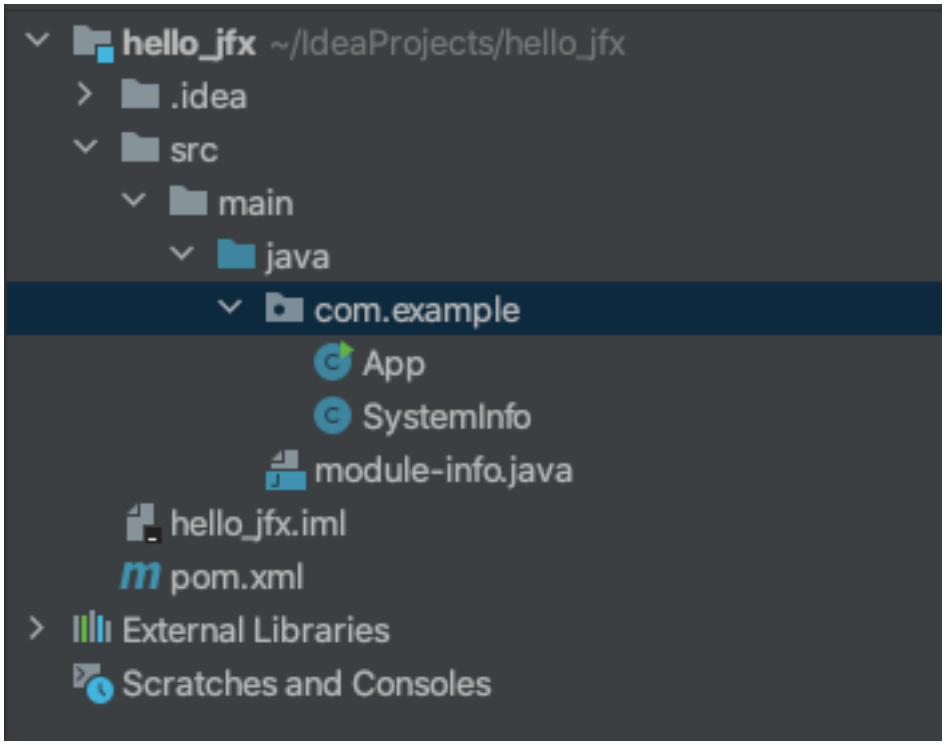
7. כתיבת התוכנית Hello World בעזרת JavaFX

לפני שנתחיל, נוודא שה־plugin של JavaFX מופעל (אפשר לעשות זאת בחלון הראשי).

כדי ליצור תוכנית עם ממשק גרפי בעזרת JavaFX יש ליצור פרויקט JavaFX. ניתן לעשות זאת בעזרת archetype שהוצג מקודם. בדוגמה הזו, נשתמש ב־javafx-archetype-simple.

ניצור פרויקט חדש עם דומיין com.example ושם hello_jfx.

ניתן לראות ב־Project את הפרויקט החדש שנוצר ואת מרכיביו:



קובץ ה־main שנוצר (App.java) מכיל את המחלקה App המרחיבה את המחלקה **javafx.application.Application** ובה מתודת start() אותה יש לדרוס.

הקוד המובא להלן מכיל את קוד המחלקה App הדרוש למימוש התוכנית שהוגדרה לעיל.

כמו כן מובא הסבר על הקוד. ניתן כמובן לשנות את שם המחלקה App לשם אחר.

```
package org.example.hello_jfx;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {

    @Override
    public void start(Stage stage) {
        stage.setTitle("Hello World!");
        TextField helloTF = new TextField("");
        Button btn = new Button();
        btn.setText("Say 'Hello'");
        btn.setOnAction((event) -> {
            helloTF.setText("Hello World!");
        });
        StackPane root = new StackPane();
```

```
        StackPane.setAlignment(helloTF, Pos.TOP_CENTER);
        root.getChildren().addAll(helloTF, btn);
        stage.setScene(new Scene(root, 300, 300));
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

הסבר על התוכנית:

כפי שהוסבר קודם, המחלקה App מרחיבה את המחלקה javafx.application.Application. המתודה start() (שורה 14) היא נקודת הכניסה לכל האפליקציות של JavaFX. המחלקה javafx.application.Application מספקת את כל השירותים הנדרשים של האפליקציה, לאורך מחזור החיים שלה, כמו אתחול, שיגור (launching), התחלה וסיום.

מתודת ה־main משגרת את האפליקציה ע"י זימון המתודה launch(). לאחריה האפליקציה תהיה במצב ready. ה־JavaFX framework (ולא קוד התוכנית של האפליקציה) קוראת למתודה start() כדי שהאפליקציה תתחיל להתבצע.

לכן כל תוכנית הכוללת ממשק גרפי המבוסס על JavaFX תהיה במבנה הנ"ל.

בתוכנית פשוטה זו משתמשים ב־Stage (החלון הראשי) ועליו מציגים את הכפתור ותיבת הטקסט.

במתודה start מגדירים את הפרמטרים של החלון הראשי (שורה 15 – קביעת כותרת החלון), יוצרים את ה-leaf nodes (שורה 16 ליצירת שדה הטקסט ושורות 17-18 ליצירת הכפתור).

בשורות 19-21 נעשה טיפול באירוע. בשורה 19 יוצרים מופע אנונימי מטיפוס מקשיבון ורושמים אותו כמקשיבון של הכפתור btn באמצעות המתודה `setOnAction` של המחלקה Button.

המקשיבון הוא מחלקה המממשת את הממשק הבא

Interface EventHandler<T extends Event>

הכולל מתודה אחת handle המקבלת כפרמטר את אובייקט האירוע.

במימוש המתודה יש לרשום את הקוד אותו על האפליקציה לבצע כתגובה לאירוע. במקרה שלפנינו יש לכתוב את ההודעה "Hello World!" בשדה הטקסט (המופע helloTF של המחלקה TextField). זה נעשה בשורה 20 באמצעות המתודה setText של המחלקה TextField.

שורות 22-25 מטפלות בפריסת האלמנטים בחלונות האפליקציה (המרכיב השני בתוכנה גרפית). JavaFX מגדירה מספר סוגים של מנהלי פריסה הנקראים layout panes וכולם יורשים מהמחלקה javafx.scene.layout.Pane. בקישור הבא ניתן לקרוא על מנהלי הפריסה

http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

בתוכנית זו נבחר מנהל הפריסה StackPane (הממקם את את ה־nodes במחסנית וכל צומת נוסף לראש המחסנית). בשורה 22 יוצרים מופע מטיפוס StackPane בשם root. בשורה 24 מוסיפים את

שני האלמנטים (הכפתור ותיבת הטקס) ל"רשימת הילדים" של ה־root (באמצעות המתודה getChildren של המחלקה Pane).

בשורה 25 יוצרים Scene (אובייקט שמהווה את חלון האפליקציה שלנו שיכלול את root וגודלו הוא 300 על 300 פיקסלים) והוא מועבר לאובייקט ה־Stage של התוכנית. רק בשורה 26 יוצג החלון על המסך (ע"י המתודה show של המחלקה Stage).

על מנת להריץ את התכנית, עלינו להשתמש ב־maven goal בשם javafx:run.

8. **כתיבת התוכנית Hello World בעזרת JavaFX ו-JavaFX Scene Builder**

<u>שלב 1:</u> כמעט זהה למקודם, אך יש להשתמש ב־archetype הבא: javafx-archetype-fxml.

<u>שלב 2:</u> הרצה ראשונית והתבוננות בפרויקט שנוצר.

<u>הסבר:</u> JavaFX מעודדת שימוש בתבניות התכן MVC או MVP. אנו נלמד על תבניות תכן ובין השאר על תבנית התכן MVC בהמשך הקורס (בהרצאה נפרדת בנושא זה). המעוניינים להרחיב יכולים לקרוא בקישור הבא:

https://he.wikipedia.org/wiki/Model_View_Controller

תבנית **Model-View-Controller** מתארת טכניקה לחלוקת היישום לשלושה חלקים, **מודל**, **מבט** ו**בקר**.

למטרות מימוש הממשק הגרפי ה־controller (הבקר) משמש לעיבוד ולתגובה לאירועים המתרחשים ב־view (המבט). אירועים אלו מתרחשים על פי רוב כתגובה לפעולה של המשתמש. לכן בעזרתו נממש את "מרכיב 3 - ההתנהגות הדינמית של האלמנטים בתגובה לפעולות של המשתמש/ת".

נתבונן ב־Java class בשם **PrimaryController.java** בחבילה שבה נמצאת המחלקה App.

```
public class PrimaryController {

    @FXML
    private void switchToSecondary() throws IOException {
        App.setRoot("secondary");
    }
}
```

הריצו את התכנית. יופיע מסך עם כפתור שיאפשר לעבור למסך שני, וחזרה. הפונקציונליות של המעבר מהחלון הראשון שנפתח לחלון השני ממומשת במתודה switchToSecondary שלעיל.

שלב 3:

בשלב זה, נערוך את הקובץ primary.fxml שנמצא ב־ src/main/resources/com/example/hello_jfx_fxml (או נתיב אחר בהתאם לדומיין, כלומר לשילוב של groupId, נקודה, ולבסוף artifactId). זהו קובץ המבוסס על שפת XML המאפשר בנית ממשק משתמש בנפרד מהחלק הלוגי של מערכת התוכנה (שוב על יתרונות גישה זו נלמד בשלבים מאוחרים יותר בקורס). המעוניינים לקרוא יותר על FXML יוכלו לעשות זאת בקישור הבא:

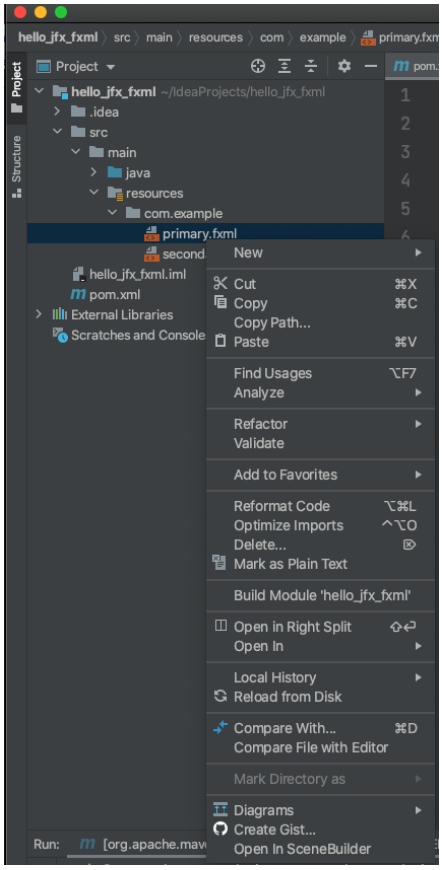
http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDCHIBE

כרגע יש לנו רק label וכפתור.

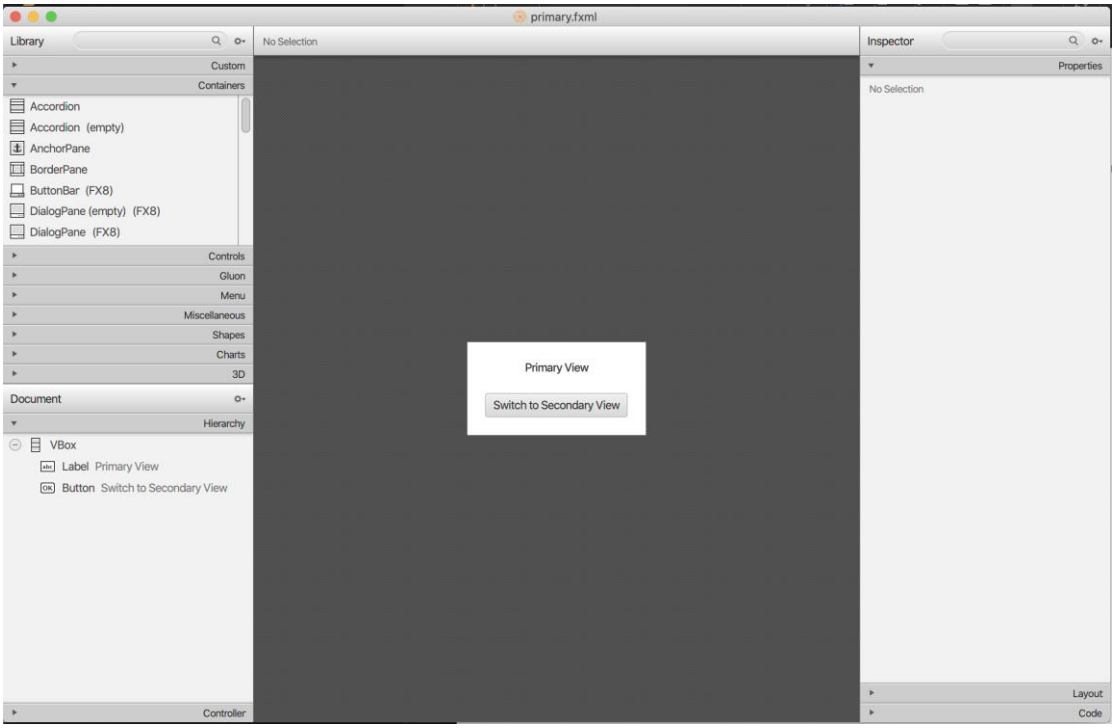
שלב 4: שינוי קובץ FXML בעזרת SceneBuilder

היתרון של השימוש ב־SceneBuilder ביצירת ממשק המשתמש הוא רב: הוא מטפל בפריסת האלמנטים (מרכיב 2 בחלק הגרפי, כפי שצוין לעיל), קובע את האירועים ויוצר skeleton code עבור מחלקת ה־controller (PrimaryController.java).

ב־Project יש לבחור את הקובץ primary.fxml וללחוץ על הכפתור הימני בעכבר. עתה יש לבחור באפשרות של Open with SceneBuilder (בסוף):

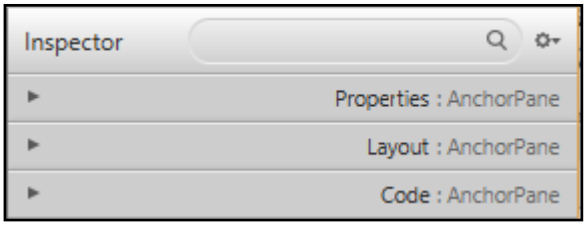


כתוצאה מכך יפתח חלון גרפי לעריכת הקובץ:

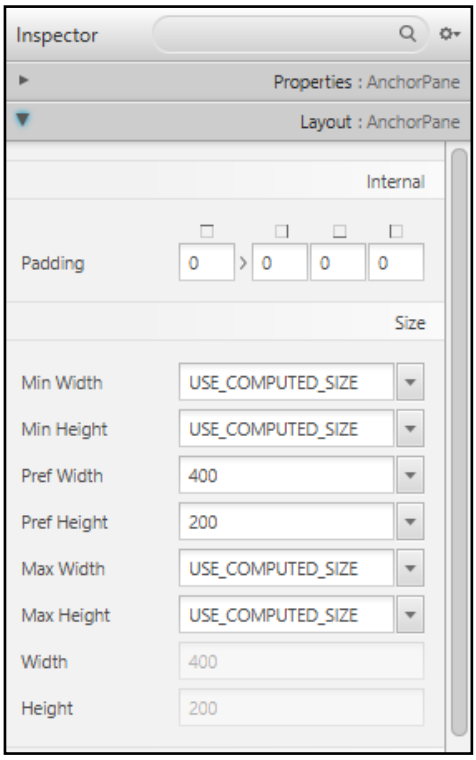


נמחק את כל תוכן המסמך ונחליף אותו באלמנט אחד של AnchorPane.

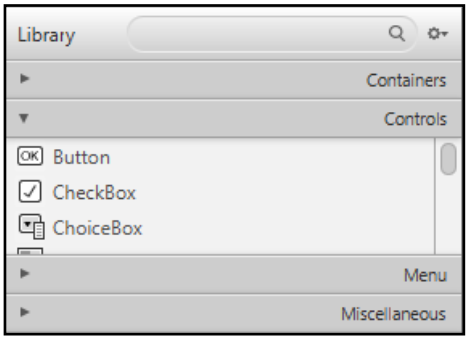
כדי לשנות את גודל AnhcorPane יש לבחור אותו (ב־Hierarchy - החלק הימני התחתון של המסך). כתוצאה מכך יוצגו בחלק השמאלי העליון של המסך מרכיביו השונים:



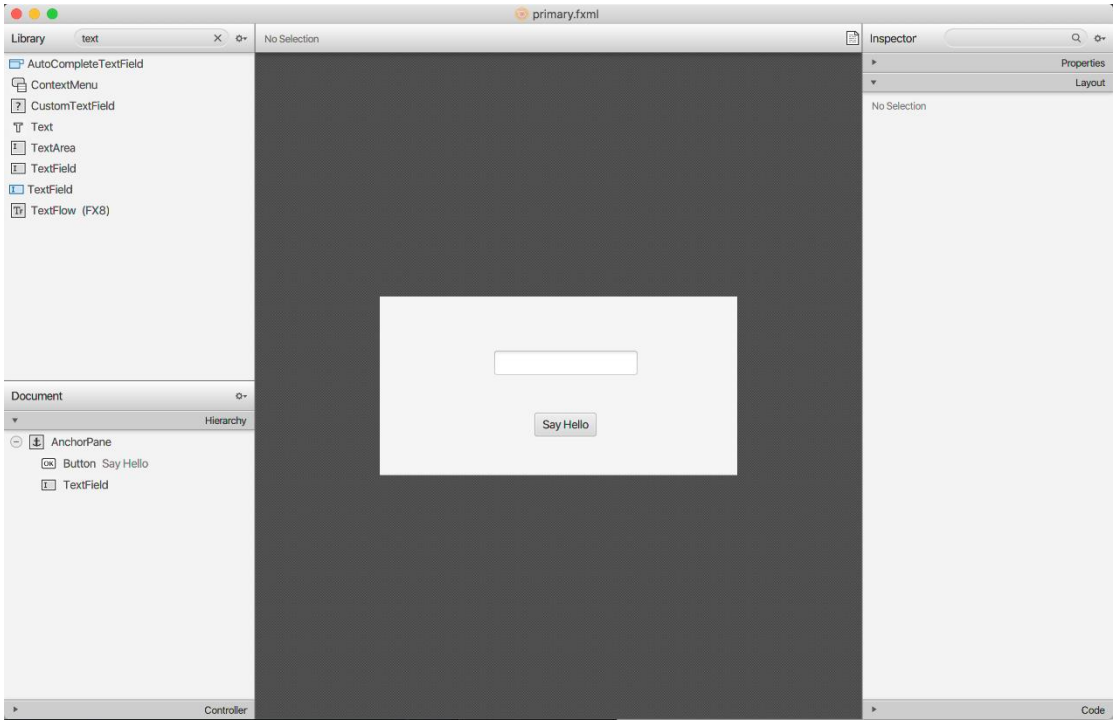
יש לפתוח אל ה־Layout ולשנות את הגודל (Pref Width ו־Pref Height):



כדי להוסיף את שני האלמנטים הנדרשים – כפתור ושדה טקסט – יש לפתוח את ה-Controls בחלק השמאלי העליון של ה־SceneBuilder ולגרור את שניהם למסך התצוגה ולמקמם במקום הרצוי.



כאשר ממקמים את אלמנטי ה־UI ניתן להבחין בקווים אדומים המופיעים ונעלמים. קווים אלה מאפשרים למקם בקלות אלמנטים ביחס לאלמנטים אחרים או למרכז/ליישר את האלמנטים על המסך.

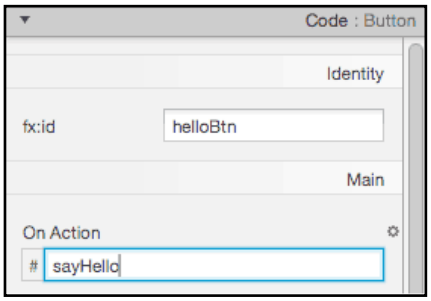


כדי לשנות את הכיתוב על הכפתור לאחר שמוקם בתוך ה־AnchorPane, יש ללחוץ לחיצה כפולה על הכפתור ולערוך את הטקסט.

עכשיו, לאחר ששני האלמנטים מוקמו על ה־AnchorPane, ניתן לראות אותם בחלון ה־Hierarchy בצד שמאל למטה במסך ה־SceneBuilder . הכפתור ושדה הטקסט הם ה־ leaf. nodes. בתוכנית זו אין שימוש ב־groups.

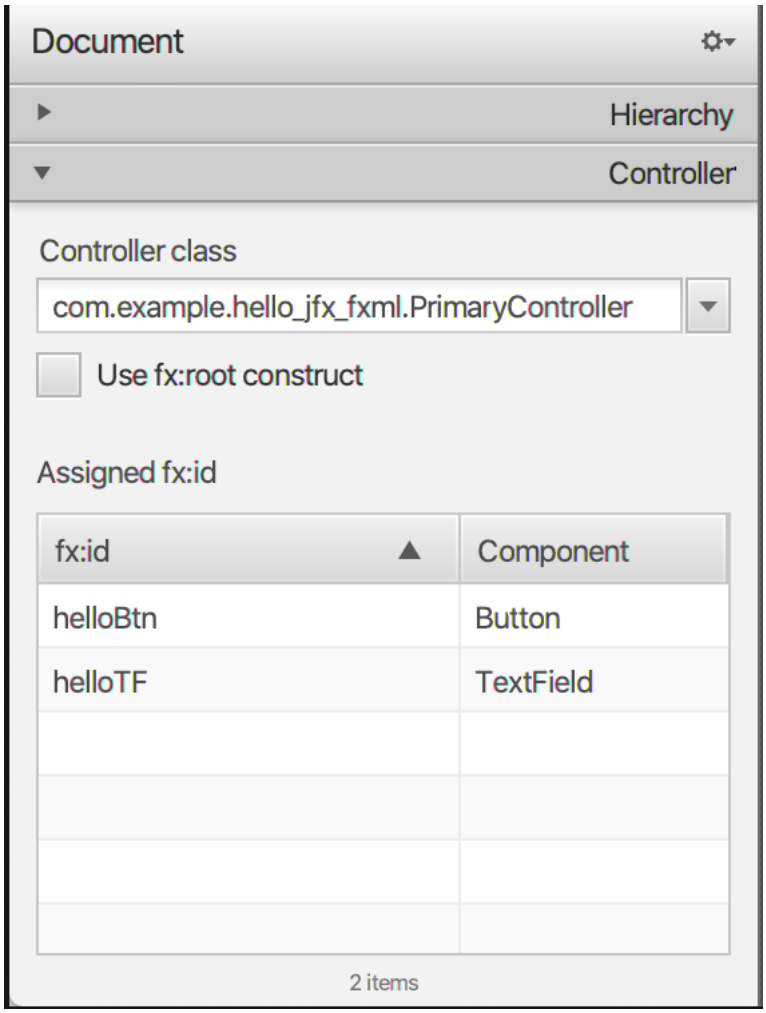
עד עתה טיפלנו בשני המרכיבים הראשונים של הקוד הגרפי: בחירת האלמנטים והפריסה שלהם על המסך. עתה יש לטפל במרכיב השלישי - ההתנהגות הדינמית של האלמנטים בתגובה לפעולות של המשתמש/ת. לשם כך יש לתת לכל אלמנט שם מזהה ולשייך לכפתור ActionListener (מקשיבון). לשם כך יש לפתוח את ה־Code section (בצד ימין של ה-SceneBuilder).

יש לבחור בכפתור תחת ה־Hierarchy section בצד ימין של ה־SceneBuilder ולפתוח את החלק שנקרא Code, המציג את תכונות הקוד המתייחסות לכפתור. יש לקבוע את fx:id ל־"helloBtn" ואת On Action ל-"sayHello".



באופן דומה יש לבחור את שדה הטקסט ב-Hierarchy ולשנות את ה־fx:id (תחת ה-Code) ל־"helloTF".

לבסוף יש לציין שה־Controller שהגדרנו הוא ה־Controller של קובץ ה-FXML. ניתן לעשות זאת ע"י כתיבת קוד או באמצעות ה־SceneBuilder. לשם כך יש לפתוח את חלק ה־Controller (בצד שמאל למטה). בחלק זה כבר מופיעים ה- fx:id של הכפתור ושדה הטקסט. יש להוסיף ולציין את השם המלא של ה- controller ("com.example.hello_jfx_fxml.PrimaryController").



יש לזכור לבצע save לקובץ.

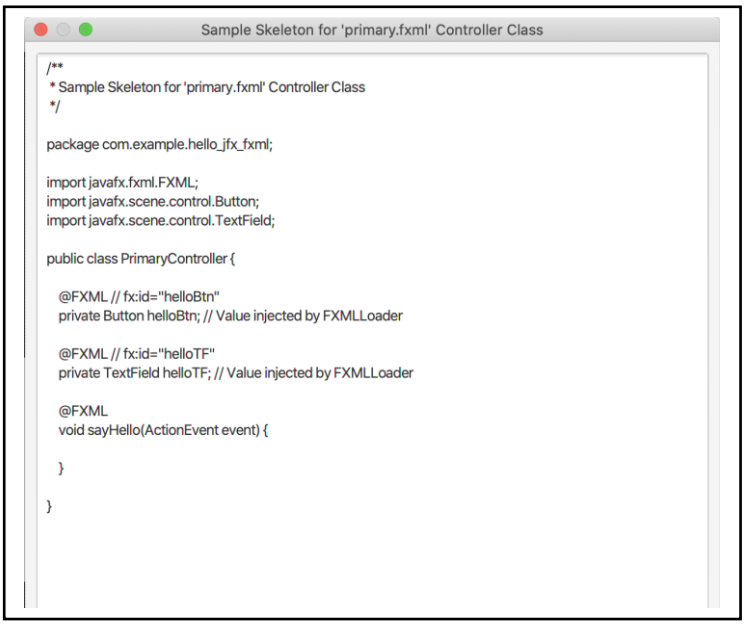
עכשיו ניתן לבצע **preview** ולראות כיצד יראה חלון האפליקציה בזמן ריצה על ידי ביצוע **Preview->Show Preview in Window** בתפריט הראשי של ה־SceneBuilder.

בנוסף ניתן לצפות בקוד הקובץ ע"י פתיחתו ב־IntelliJ.

<u>שלב 5:</u>

התוכנית אותה אנו כותבים אמורה להגיב לאירוע אחד – לחיצה על הכפתור. כתוצאה מכך יש להציג משפט בשדה הטקסט. בשלב זה לא כתבנו כל קוד לביצוע דרישה זו.

לשם כך יש לבחור באופציה **View->Show Sample Controller Skeleton** בתפריט הראשי של ה־SceneBuilder. כתוצאה מכך יוצג הקוד הראשוני של מחלקת ה־controller. בתחתית החלון בחרו "Comments" ואל תבחרו "Full". אחר כך העתיקו את הקוד ע"י לחיצה על "Copy" והעבירו אותו למחלקה PrimaryController ב־IntelliJ.



להלן קוד המחלקה:

```
/**
 * Sample Skeleton for 'primary.fxml' Controller Class
 */

package org.example.hello_jfx_fxml;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class PrimaryController {
```

```
    @FXML // fx:id="helloBtn"
    private Button helloBtn; // Value injected by FXMLLoader

    @FXML // fx:id="helloTF"
    private TextField helloTF; // Value injected by FXMLLoader

    @FXML
    void sayHello(ActionEvent event) {

    }

}
```

הקוד הנ"ל נוצר על ידי ה־SceneBuilder על פי מה שהוגדר על ידינו. למשל בשורה 4 מוגדרת חבילה כי בזמן הגדרת מחלקת ה־controller ציינו את ה־path המלא של קובץ המחלקה. שמות הכפתור ותיבת הטקסט נלקחו מערכי ה־fx:id שהגדרנו בחלק של ה־Code ב־SceneBuilder. שם ה־action listener הוא על פי השם שנקבע ב־On Action property. שימו לב שכל ה־imports הם מתוך החבילה JavaFX. הסימון @ בקוד מציין אנוטציה. הסימון @FXML הוא הוראה לקומפיילר וגם ל־JVM לחבר (hook up) את כל אלמנטי ה-FXML. הערכים של helloBtn ו-helloTF מסופקים בזמן ריצה על ידי קריאה ל-FXMLLoader במתודה ()start שבמחלקה Main.

```
/**
 * Sample Skeleton for 'primary.fxml' Controller Class
 */

package org.example.hello_jfx_fxml;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class PrimaryController {

    @FXML // fx:id="helloBtn"
    private Button helloBtn; // Value injected by FXMLLoader

    @FXML // fx:id="helloTF"
    private TextField helloTF; // Value injected by FXMLLoader

    @FXML
    void sayHello(ActionEvent event) {
        helloTF.setText("Hello World!");
    }
}
```

תוספת הקוד הנדרשת היא בשורות 21-23: הגדרנו מתודה בשם sayHello, המקבלת פרמטר מסוג ActionEvent (שנוצר בזמן האירוע ומכמס מידע עליו). מתודה זו תיקרא עם קבלת האירוע ומה שיתבצע בה היא שורה 22 (הצגת ההודעה המתאימה בשדה הטקסט).

<u>שלב 6:</u> עתה ניתן להריץ את התוכנית באמצעות run של מחלקת App (דרך maven goal, כפי שתואר מקודם).

שימו לב – בגרסאות קודמות של המעבדה נדרש לשנות את הקוד, אך כיוון שהתחלנו מתבנית כלשהי, אין צורך. עם זאת: שימו לב למספר דברים חשובים:

```
package org.example.hello_jfx_fxml;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML("primary"), 640, 480);
        stage.setScene(scene);
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml +
".fxml"));
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }

}
```

מופעלת בהתחלה. נוצר Scene חדש, משויך ל־stage ולתוכו "נשפך" התוכן של primary.fxml. לאחר מכן, ה־stage מוצג.

משמשת להחלפת התוכן של ה־Scene.

פונקציית עזר – טוענת FXML. שימו לב לפרמטר המוחזר – הוא מטיפוס Parent, כלומר ברגע שטענו את הקובץ, ניתן להתייחס אליו כאל אלמנט רגיל של JavaFX.

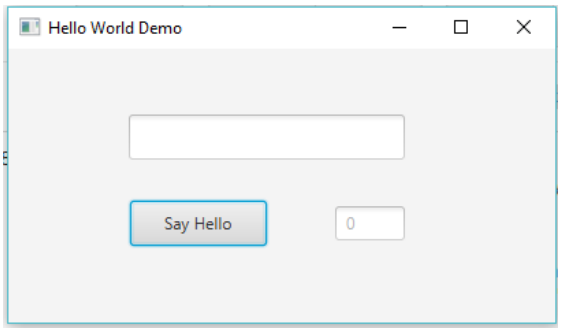
מטלה 1

הוסיפו לתוכנית את הדרישות הבאות:

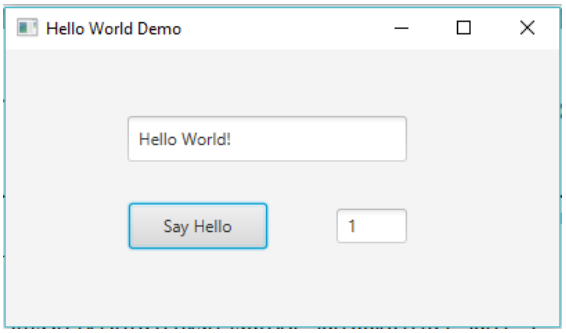
1. לחיצה ראשונה תציג בשדה הטקסט את ההודעה "Hello World!" . הלחיצה הבאה תנקה את שדה הטקסט וזו שלאחריה תציג את ההודעה שוב.
2. בנוסף, בשדה טקסט נוסף יוצג מספר הפעמים שלחצו על הכפתור.
3. כותרת החלון צריכה לכלול את מספרי תעודת הזהות שלכם (בדוגמאות שלהלן דרישה זו לא ממומשת. הכותרת צריכה להיות לדוגמה:

"Hello World Demo – 051231244;067296734".

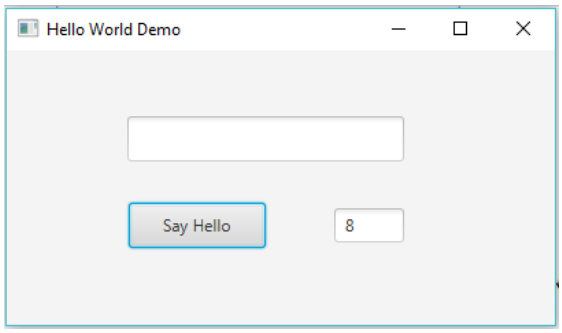
להלן מסך התוכנית לאחר הרצתה המוצג לראשונה למשתמש:



לאחר הלחיצה הראשונה יוצג החלון הבא:



לאחר 8 לחיצות יוצג החלון הבא:

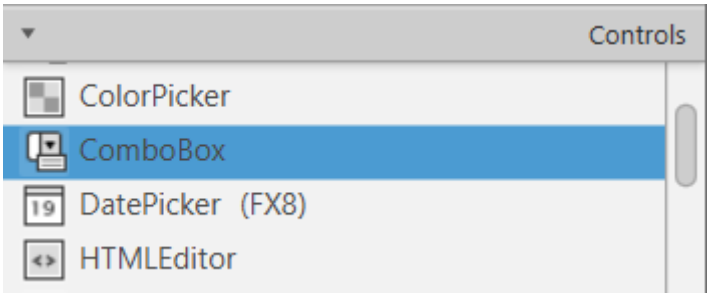


הוראות הגשה: יש לכלול בקובץ ההגשה תמונות מסך של מסכי התוכנית עם הרצתה, לאחר הלחיצה השלישית ולאחר הלחיצה העשירית (סה"כ 3 תמונות).

עד כה למדנו איך להשתמש בשני אלמנטים חשובים – הכפתור ותיבת הטקסט. בשביל המטלה הבאה נלמד על אלמנט נוסף שימושי מאוד – תיבת בחירה (ComboBox). תיבת הבחירה מאפשרת למשתמש לבחור אפשרות אחת מתוך רשימה קיימת.

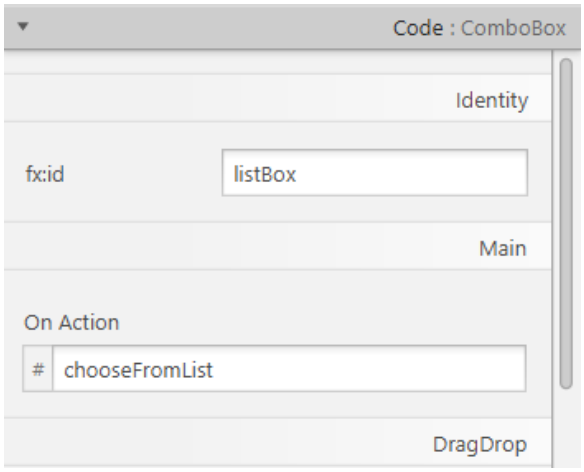
איך טוענים רשימה?

שלב 1: יש לפתוח את ה- Controls בחלק השמאלי העליון של ה־SceneBuilder ולגרור את תיבת הבחירה למסך התצוגה ולמקמם במקום הרצוי.

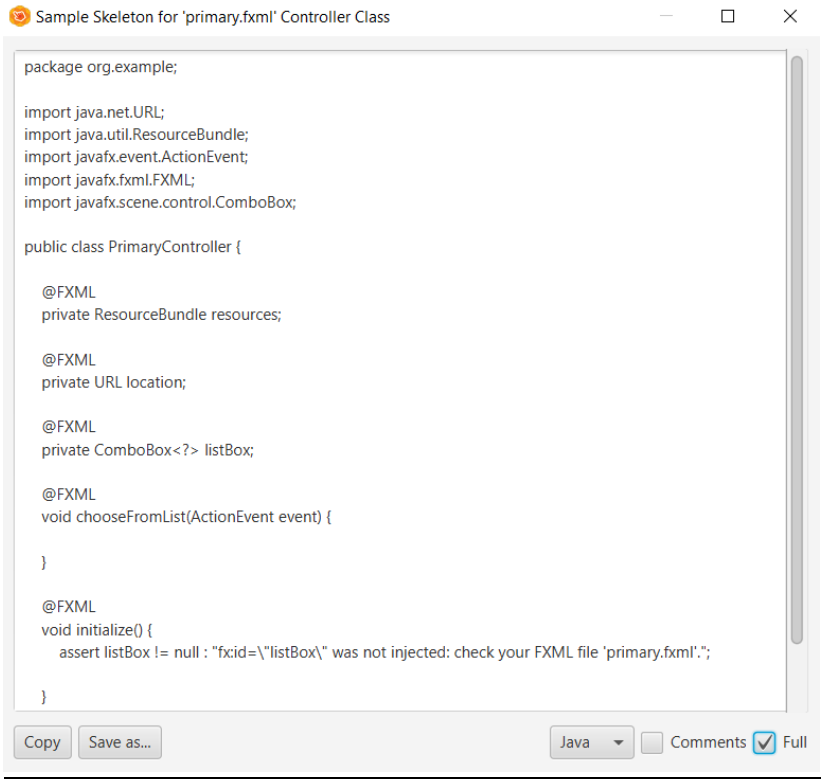


כאשר ממקמים את אלמנטי ה־UI ניתן להבחין בקווים אדומים המופיעים ונעלמים. קווים אלה מאפשרים למקם בקלות אלמנטים ביחס לאלמנטים אחרים או למרכז/ליישר את האלמנטים על המסך.

שלב 2: יש לבחור בכפתור תחת ה־Hierarchy section בצד ימין של ה־SceneBuilder ולפתוח את החלק שנקרא Code, המציג את תכונות הקוד המתייחסות לכפתור. יש לקבוע את fx:id ל־"listBox" ואת On Action ל-"chooseFromList".



שלב 3: יש לבחור באופציה **View->Show Sample Controller Skeleton** בתפריט הראשי של ה־SceneBuilder. כתוצאה מכך יוצג הקוד הראשוני של מחלקת ה־controller. בתחתית החלון בחרו "Comments" ואל תבחרו "Full". אחר כך העתיקו את הקוד ע"י לחיצה על "Copy" והעבירו אותו למחלקה PrimaryController ב־IntelliJ.



להלן קוד המחלקה:

```
/**
 * Sample Skeleton for 'primary.fxml' Controller Class
 */
package org.example;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.ComboBox;

public class PrimaryController {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private ComboBox<?> listBox;

    @FXML
    void chooseFromList(ActionEvent event) {

    }

    @FXML
    void initialize() {
        assert listBox != null : "fx:id=\"listBox\" was not injected:
check your FXML file 'primary.fxml'.";
```

```
    }

}
```

שלב 4: תוספת הקוד הנדרשת תהיה בinitialize וב chooseFromList:

ראשית בהגדרה של הComboBox נגדיר איזה סוג אובייקט הרשימה מקבלת (הרשימה אינה יכולה לקבל טיפוסים פרימיטיבים).

()Initialize – המתודה שמאתחלת את המסך PrimaryController. נוסיף את השורות הבאות למתודה כדי לאתחל את הרשימה של הComboBox:

```
listBox.getItems().add(first);
listBox.getItems().add(second);
```

וכו' בכדי להוסיף את האובייקטים לרשימה.

(chooseFromList(ActionEvent event – מתודה זו תיקרא עם קבלת האירוע ותטפל בבחירה מתוך הרשימה. נוסיף את השורה הבאה למתודה:

```
String chosen = listBox.getSelectionModel().getSelectedItem();
```

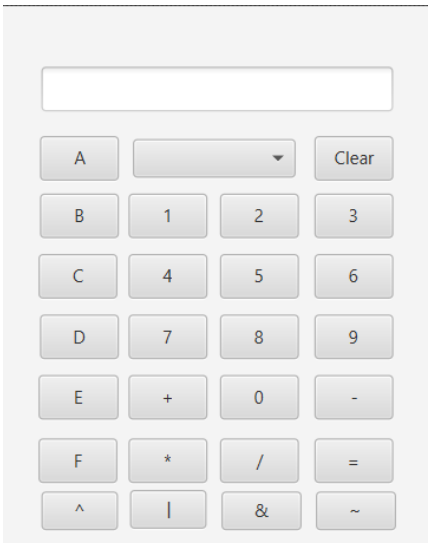
פונקציות שימושיות נוספות לעבודה עם רשימות של ComboBox:

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ComboBox.html

שלב 5: עתה ניתן להריץ את התוכנית באמצעות run של מחלקת App (דרך maven goal, כפי שתואר מקודם).

תרגיל מעבדה 3

יש לכתוב אפליקציה קטנה למימוש מחשבון בסיסי עם פעולות שהתבקשתם לעשות במעבדה 1.
למחשבון 4 מצבים: מצב hex בו החישובים הם של מספרים בבסיס 16,מצב dec בו החישובים הם בבסיס 10, מצב oct בו החישובים הם בבסיס 8 ומצב bin בו החישובים הם בבסיס 2.



הדרישות הן:

1. רשימת ComboBox לבחירת מצב המחשבון: HEX, DEC, OCT, BIN. במצב HEX החישובים הם בבסיס 16 המקשים | ,^ ,& ,~ אינם פעילים (במצב disabled). במצב DEC הטור השמאלי של המקשים (F-A וגם | ,^ ,& ,~) אינו פעיל (במצב disabled) והחישובים הם בבסיס 10. במצב OCT הטור השמאלי אינו פעיל וגם הספרות 8-9 וגם | ,^ ,& ,~ והחישובים הם בבסיס 8. במצב BIN רק הספרות 0-1 פעילות והחישובים הם בבסיס בינארי ורק במצב הזה המקשים | ,^ ,& ,~ פעילים עם שאר האופרטורים.
2. המספרים הם מספרים טבעיים.
3. מקש Clear מאפס את התצוגה.
4. בעזרת מקשי המספרים ומקשי הפעולות ניתן להזין ביטוי אריתמטי.
5. לחיצה על מקש = גורמת לחישוב ערך הביטוי שהתקבל ולהצגתו בתיבת הטקסט.
6. במעבר לבסיס אחר – אם יש ערך בתיבת הטקסט הוא יתורגם לביטוי המתאים בבסיס החדש. אם יש ביטוי אריתמטי בתיבת הטקסט הוא יחושב והתוצאה תתורגם לבסיס החדש.

הוראות הגשה לדוח מעבדה 3:

1. ההגשה בזוגות בלבד באמצעות הגשה אלקטרונית. ניתן להגיש מספר פעמים.
2. יש להגיש קובץ zip אחד. שם קובץ ה־zip הוא מספרי תעודות הזהות של הסטודנטים מופרדים בגרש תחתון. הקובץ כולל שני קבצים:
   א. קובץ word ובו מסכי מטלה 1 כמפורט לעיל.
   ב. קובץ JAR (**calculator.jar**) ע"פ דרישות תרגיל מעבדה 3. שימו לב – חובה לכלול גם את קובצי המקור.
   ג. כדי שהJAR יעבוד אתם צריכים לעשות fake main כמו שהדגמתי במעבדה.
3. אנא הקפידו על הוראות ההגשה.

ע ב ו ד ה נ ע י מ ה!