מסמך מלווה למיני-פרוייקט

תכנה לניהול מערך היסעים

מגישים: יוסף מלכה ואהרן קרמר

יש לקרוא את ההוראות בקובץ ה- Readme לפני הרצת התכנית

שכבת PL

הבעיות שעמדו בפתחנו:

- שימוש נרחב ב- Data Binding שעלול לסרבל את הקוד
- פתיחת חלונות רבים המעמיסים על שימוש המשתמש
- קריאה ל- Event-ים עבור איברים בתוך רשימות
- מימוש MVVM בצורה נוחה וידידותית למפתח

פתרונות:
בחרנו להשתמש בספריית Caliburn.Micro כסביבת עבודה נוחה של MVVM, הכלים העומדים לרשותנו בספרייה זו מרובים, נמנה אותם בקצרה:

- שימוש נוח ב- Event-ים של פקדים בארכיטקטורת MVVM: קריאה ל- Event-ים נעשית ב- ViewModel של ה- View המתאים בצורה שונה מהנלמד בהרצאות, לדוגמא: נעיין בשורות הקוד בצד שמאל, הגדרת שם הפקד Button_Name בקוד ה- XAML, כורכת באופן אוטומטי בין הפקד לבין הפונקציה/property, בעלי אותו שם ב- ViewModel התואם את ה- View.

```
<Button x:Name="Button_Name"/>
```

```
public void Button_Name() { }
```

כמו-כן, ניתן להוסיף את המילה Can כתחילית שם פונקציה/property מסוג בוליאני על מנת לשלוט בקלות במצב פעילות הפקד, לדוגמא להגדרת תכונת ה- enable

- שימוש נוח ליישום DataBinding: ניתן לבצע Binding לפקד ע"י הגדרת שמו כשם ה- property שאנו רוצים לכרוך לו. כמו-כן, כל ה- properties באובייקט ה- ViewModel מממשים תבנית עיצוב Observable כתחליף למימוש הממשק INotifyChange. בנוסף, כאשר נרצה ליישם Binding עבור אוסף של אובייקטים, נשתמש בישות BindableCollection היורשת מ- ObservableCollection
- כאשר אנו רוצים לקרוא ל- event עבור איבר של פקד המכיל אוסף של איברים, נוסיף ל- ItemTemplate את המבנה הבא:

```
<i:Interaction.Triggers>
    <i:EventTrigger EventName="Click">
        <cal:ActionMessage MethodName="RemoveStation">
            <cal:Parameter Value="{Binding Path=Code}" />
        </cal:ActionMessage>
    </i:EventTrigger>
</i:Interaction.Triggers>
```

כעת נוכל לכתוב ב- ViewModel את הפונקציה:

```
public void RemoveStation(string code) { }
```

- בעזרת ירושה מאובייקט הנקרא conductor<T>, אנו יכולים להריץ ולהחליף UserControl בתוך חלון מסויים. כך מתאפשר לכתוב את כל התוכנה כך שתוצג בחלון אחד ובמבנה דומה ל- SPA ב- Web.

בונוסים:

- בניית שכבת PL בארכיטקטורת MVVM
- DataTemplate – שימוש ב- DataTemplate לקביעת תצורה עבור הצגת אוספים
- Triger – שימוש ב- Triger על מנת לקרוא לפונקציות בתאים של רשימות בצורה נוחה

שכבת BL

סימולציה:

- בכפוף לאישור המרצה, ד"ר דוד קדרון, העדפנו להשתמש ב- EventHandler כתחליף לשליחת Delegate מסוג Action, מטרת השינוי לאפשר ניטור של התחנה ממספר מערכות שונות

על מנת לדמות את הסימולציה לזמן אמת ככל האפשר, אתחלנו את זמני קוי האוטובוס כאילו יצאו מיעדן קודם מועד הרצת הסימולציה בפועל, כך מיד בעת הרצת הסימולציה תחל פעילות הנסיעה כפי שהיא בזמן אמת

בונוסים:

- ניתן להחליף בין תחנות בזמן הרצת הסימולציה בלוח האלקטרוני
- המעקב אחר התחנה מסתנכרן כל העת ברמת השניה
- חישוב המרחק בין תחנות נעשה על ידי קריאה ל- RESTAPI אשר מחשב מרחק נסיעה **אמיתי** בין שתי תחנות, ולא על ידי חישוב מרחק לפי קוי אורך ורוחב, (ללא התחשבות בפקקים)

שכבת DAL

על מנת לאפשר גמישות מושלמת של נתונים, כתבנו את הקוד של שכבת ה- DAL כך שיעבוד עם אובייקטים מסוג Object בלבד. התנאי היחיד לאובייקט כזה הוא שיכיל את השדות: ID, IsDeleted, IsRunningId. נעזרנו בשימוש רב ב- Reflection לצורך זה.

כתוצאה ממימוש באופן הנ"ל נקרתה בפנינו בעיה:

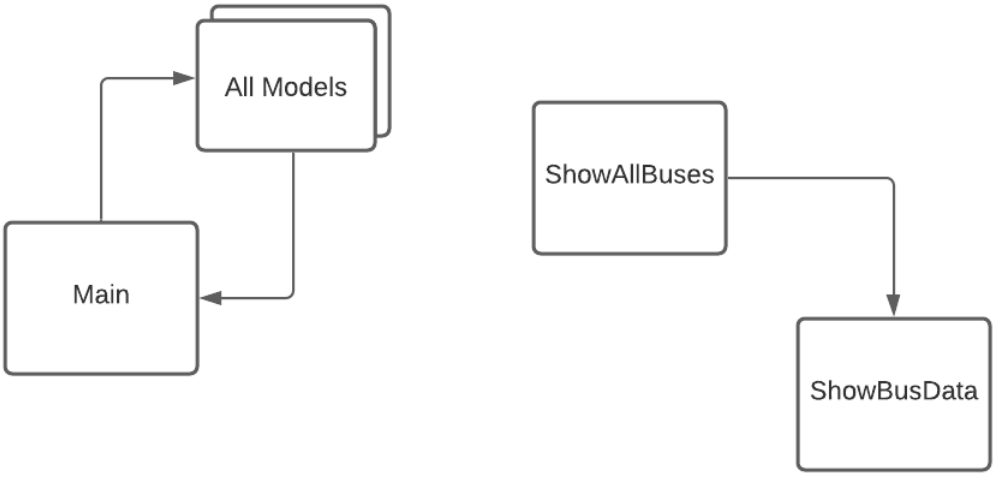
ב-DalFactory הבחנו בבעיה לפיה קבצי ה- Assembly של DALXML ו- DALObject, לא נטענו. לאחר שחקרנו את שורש הבעיה הבנו שהקומפיילר לא טוען את הקבצים משום שלא נעשה בהם שום שימוש בקוד של המחלקה הנ"ל. על מנת לפתור את הבעיה יצרנו פונקציה ששמה הוא DoNothing, שמשתמשת בקבצים מבלי לשנות בהם דבר.

בונוסים:

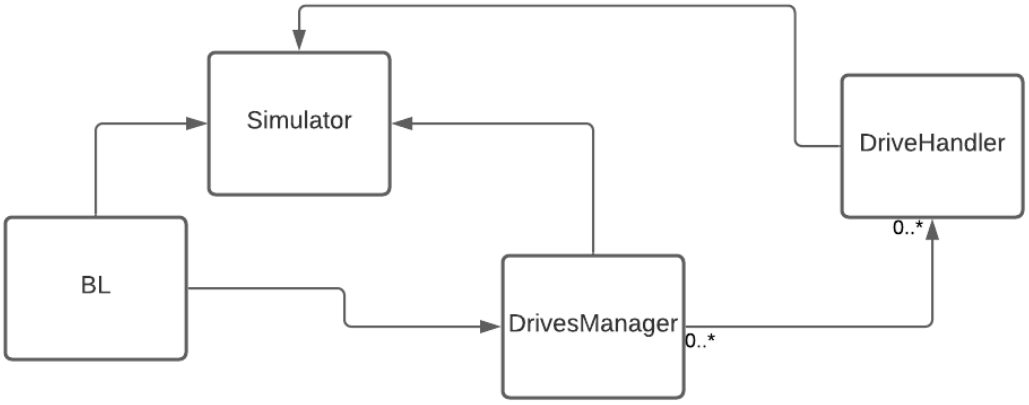
- שימוש ב- IsDeleted
- שימוש בקובץ קונפיגורציה על מנת להחליף בין השימוש ב- DALXML ו- DALObject מבלי שיהיה צורך לקמפל

הערה: אין ספק שניתן לשפר ולייעל את התצוגה עוד ועוד, יכלה הזמן והמה לא יכלו. אנו בחרנו למקד את עיקר המאמץ בכתיבת הקוד ה"תכנותי" של שכבות BL ו- DAL ובלוגיקה של שכבת ה- PL, יחד עם זאת, השתדלנו שאף ממשק המשתמש יהיה ידידותי וקל לתפעול.

מבנה שכבת PL



מבנה הסימולטור



מבנה הפרוייקט