

מערכי שיעור – WPF

❖ הערת פתיחה

החומרים לחלופה זו נלקחו מהשתלמות "תכנות אסינכרוני" שהעביר ארז קלר. לאחר שקיבלתי את רשותו, אני משתפת פה את המצגות שארז הכין ושממשות אותי בכיתה תוך כדי הוראת החלופה. החלוקה לשיעורים נעשתה ע"פ נסיוני בכיתה. כאשר בהגדרת שיעור הכוונה היא לשנים (או שלושה) שיעורי מעבדה רצופים (של 45 דק' כל אחד) תלוי ברמת התלמידים עצמם. ישנם פרקים בהם הוספתי נושאים בסיסיים, שאולי אין צורך בהם, או מתקדמים, שאולי יתאימו רק מאוחר יותר. הכל בהתאם למורה ולתלמידים בכיתה. צרפתי בסוף מספר "שיעורי הרחבה והעשרה" שהם לא בדיוק באורך של שיעורים רגילים, אבל הם חומרים שאני מלמדת ומעבירה לתלמידים שלי, תוך כדי שהם מתרגלים את הדברים הבסיסיים.

מדריך WPF שמצאתי : <http://webmaster.org.il/articles/wpf>

שיעור 1 – היכרות עם הכלים והסביבה

❖ מה זה WPF ולמה זה טוב ?

WPF (Windows Presentation Foundation) הינה טכנולוגיה לפיתוח אפליקציות Desktop עם דגש על עיצוב גרפי מתקדם וחוויית משתמש מודרנית. מתבססת על DirectX ולא על המנוע הגרפי הוותיק GDI\GDI+ בו השתמשנו באפליקציות WinForms הישנות. טכנולוגיה זו מהווה התחלה של מימוש חזון של הפיתוח המשותף למספר סביבות : PC, Web, Mobile. בסביבת WPF קיימת הפרדה מלאה בין הקוד של עיצוב המסך, הוויז'ואל, (נכתב בשפת תגיות חדשה הנקראת XAML, 'זאמל') לבין ה-Code Behind שנכתב בשפת C# והמטפל בהתנהגות ובאירועי הדף, ממש כמו בעולם ה-Asp.net.

ניתן בקלות להפריד את בניית ממשק המשתמש מהקוד, ולבנות את XAML בכלים גרפיים מקצועיים לעריכתו (כגון BLEND) ולהגדיר את ההתנהגות באמצעות כלים תכנותיים אחרים (Visual Studio).

ה-XAML ממש מזכיר את סביבת ה-HTML והוא "מתעלה עליו" בגמישות העיצוב, ביכולות הגרפיות, באפשרויות האנימציה ועוד ועוד. כל מאפיין של תגית ניתן להרחבה ולשינוי, כך שאנו מרוויחים גמישות ואדפטיביות בעיצוב. השימוש ב-Binding מייתר, בהרבה מקרים, את בדיקות התקינות כמו גם מקצר את תהליך הפיתוח.

מצגת: <http://www.corner.co.il/Articles/ArticleDetials/19/Introduction-to-WPF>

❖ היכרות עם XAML בסיסי

XAML (Extensive Application Markup Language) הינה שפת תגיות XML אשר פותחה על ידי מיקרוסופט כשפת ייצוג והגדרה של ממשק המשתמש של WPF. XAML הינה שפה פשוטה וקלה ללמידה בעלת יכולות גראפיות עשירות (אפקטים, אנימציה...) המאפשרת למפתחים לייצר בקלות יחסית ליצור ממשק אדפטיבי שיודע להתאים עצמו לגודל המסך וכן אפקטים חזותיים מרשימים. כמו כל קבצי ה-XML, גם XAML היא היררכית וכך כמובן גם האובייקטים המוגדרים בה. כל אלמנט ב-XAML מתורגם לאובייקט ב-.NET Core. אובייקט המכיל מאפיינים, מתודות ואירועים, כלומר עצם ממחלקה. שם המחלקה ושמות המאפיינים בה זהים בשםם לאלמנטים המקבילים להם בקובץ ה-XAML. לדוגמה, במקום להגדיר:

```
<TextBlock Text="Hello World" FontSize="30" ></TextBlock>
```

ניתן להגדיר בקובץ ה-CodeBehind :

```
TextBlock txt = new TextBlock();  
txt.Text = "Hello World";  
txt.FontSize = 30;
```

מצגת: <http://www.corner.co.il/Articles/ArticleDetials/20/XAML>
דף הסבר ודוגמאות קוד: <http://www.corner.co.il/Articles/ArticleDetials/188/XAML>
[חזרת המצגת](#) [דוגמאות קוד](#)

Layout and Panels <

Layout – מנגנון או מכניזם של WPF באמצעותם מגדירים כיצד הפקדים יוצגו או יסודרו על המסך. לכל Layout יש לוגיקה משל עצמו, וכל אחד מסדר את הפקדים בצורה שונה. דף מכיל Layout בודד וכל Layout מכיל מגוון רכיבים כאשר חלקן יכולים להיות Layout המכילים רכיבים.... (וחוזר חלילה)

קיימים מספר רכיבי Layout פופולאריים:

- 1. Stack Panel** - מסדר את הרכיבים לאורך או לרוחב. גודל ה-Layout נקבע ע"פ הרכיבים המרכיבים אותו.
- 2. Wrap Panel** - מסדר את הרכיבים לאורך או לרוחב. במידה ול-Layout אין מקום בשורה / בטור לכל הרכיבים אז נעשה שימוש, אוטומטית, בשורה / טור נוספים. ה-Wrap Panel משנה את מיקום הרכיבים, עצמאית, תוך כדי שינוי גודל החלון.
- 3. Dock Panel** - סידור הרכיבים ב-Layout נעשה בצורה יחסית לחלון ולרכיבים אחרים. כלומר: ניתן למקם רכיב בחלק העליון/תחתון/ימני/שמאלי של החלון, מעל/מתחת/מימין/משמאל לרכיב אחר או באיזור שנותר אחרי שכל שאר הרכיבים "תפסו את מקומם".
- 4. Canvas Panel** - סידור הרכיבים ב-Layout נעשה ע"פ המיקום (x, y) בחלון. מיקום זה נשאר קבוע גם כשהחלון משנה את גודלו. סידור זה מזכיר את סידור הרכיבים ב-WinForms הישן.

5. Grid Panel – מגדיר טבלה בעלת שורות ועמודות. כל רכיב מקבל מיקום בטבלה, ממש כמו ב-HTML. גודל הרכיב נגזר מהגודל של התא בו הוא נמצא. ניתן להגדיר את גודל השורות/עמודות, כרצוננו, בצורה קבועה/אוטומטית ע"פ התכולה / יחסית ע"פ השאר. Layout זה הוא הנוח ביותר, ברב המקרים, ולכן הוא גם הפופולרי ביותר.

6. UniformGrid – מגדיר טבלה בעלת שורות ועמודות בגודל אחיד.

מצגת (ודוגמאות קוד): <http://www.corner.co.il/Articles/ArticleDetials/21/Layout-and-Panels>

<http://www.corner.co.il/Articles/ArticleDetials/192/Layout-9קדד-Layout>

[הורדת מצגת](#) - [דוגמאות קוד](#)

פיקד בסיסי - כפתור :

קיימים מספר רכיבים ממשק, כאשר הבסיסי שביניהם הוא הכפתור. הגדרת הכפתור, בקובץ ה-Xaml, תעשה תוך שימוש בתגיות. כל תגית Xaml יכולה להכתב כתגית מקומית או כתגית תחום, כרצוננו. ממש כמו ב-HTML לתגית יש מספר מאפיינים. בדוגמא לעיל השתמשתי במאפיינים המגדירים את גודל הפונט, את צבע הרקע את התוכן שיוצג על הכפתור כמו-גם את מיקום הכפתור ב-Grid המכיל אותו והגדרת הפעולה שתופעל, ב-Code Behind, בעת לחיצה עליו.

```
<Button Grid.Column="1" Content="Red" Background="Red" FontSize="25" Click="Button_Click_1" />
<Button Grid.Column="1" Content="Red" Background="Red" FontSize="25" Click="Button_Click_1">
</Button>
```

כאמור, כל מאפיין ב-Xaml ניתן ל"פתיחה" כך שאפשר להגדיר אותו בצורה מותאמת. דבר המגדיל את הגמישות בעיצוב.

בדוגמא לעיל תוכן הכפתור (Button.Content) נפתח ע"י תגיות משלו ובו הגדרנו StackPanel המסודר לאורך המכיל בתוכו 3 רכיבים כאשר הראשון מהם הוא StackPanel המסודר, הפעם, לרוחב והמכיל טקסט + תמונה. כך שקיבלנו כפתור מורכב ו...בקלות יחסית.



דוגמא:

```
<Button Height="150" >
  <Button.Content>
    <StackPanel >
      <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
        FlowDirection="RightToLeft">
        <TextBlock Text=" רקפת " FontSize="35" FontWeight="Bold"
          Foreground="Blue" TextAlignment="Center" />
        <Image Source="Assets/Cyclamen.png" Stretch="None" ></Image>
      </StackPanel>
      <TextBlock Text="... שנתו רב גאופיט סוג היא רקפת" TextWrapping="Wrap"
        TextAlignment="Right" FontSize="15" Foreground="Black">
      </TextBlock>
      <TextBlock Text=" המלא למאמר לקריאת להק" FontSize="15" Foreground="Red"
        VerticalAlignment="Center" TextAlignment="Center" />
    </StackPanel>
  </Button.Content>
</Button>
```

משימת תרגול <

יש לבנות אפליקציית WPF פשוטה המכילה 3 כפתורים Blue / Red / Green. ב-CodeBehind נגדיר 3 משתנים (מסוג byte) עבור הערך של מרכיבי ה: אדום / כחול / ירוק של צבע הרקע. בעת לחיצה על כל אחד מהכפתורים נוסיף לערך מרכיב הצבע 10 ונציג את הצבע החדש שנוצר. נשתמש ב-Layout מסוג Grid כדי לארגן את הרכיבים (כפתורים) בחלון.



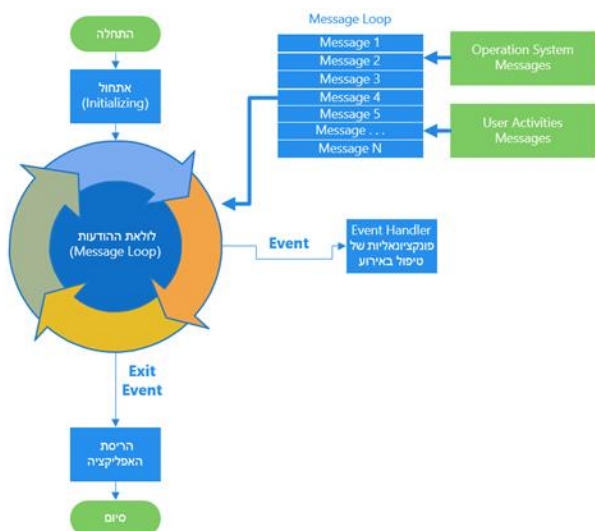
מצגת (הכוללת גם קוד חלקי הנדרש לפתרון) ודוגמאות קוד :

<http://www.corner.co.il/Articles/ArticleDetials/22/Controls-Part-1>

הרחבה: תוכנות מונחה אירועים (Event Driven)

בתוכניות הפשוטות מסוג Console Application הקוד מתבצע Top Down, מתחילים מהפעולה הראשית, Main, והפעולות מתבצעות אחת אחרי השניה, ע"פ הסדר.

אפליקציות Desktop Application (כגון אפליקציית WinForm או אפליקציית WPF) אשר מכילות חלון גרפי, חייבות מודל שונה. מודל בו מערכת ההפעלה היא זו שיוזמת את האינטראקציה עם התוכנית באמצעות שליחת הודעות (Messages) הנוצרות כתוצאה מפעילות משתמש (עכבר, מקלדת, סגירת חלון) או מיוזמת מערכת ההפעלה (טעינת חלון, הצגת חלון, שינוי פוקוס של פקד, שינויים בסטטוס של האפליקציה....).



מודל זה הינו "מונחה אירועים" (Event Driven).

- ע"פ מודל זה בעת הרצת האפליקציה נבצע מספר פעולות איתחול (Initialize) המגדירות וטוענות את החלון הראשי.
- עם סיומן האפליקציה נכנסת למצב המתנה שנקרא Idle בה היא ממתינה לאירוע (Event).
- האירוע מעורר את האפליקציה אשר מפעילה את הפעולה המטפלת בו, אם הוגדרה פעולה כזו. בסיום הטיפול האפליקציה חוזרת חזרה למצב Idle, ולהמתנה לאירוע הבא.
- האירועים נכנסים לתור הודעות ומטופלים אחד-אחד ע"פ סדר כניסתם לתור.
- תהליך זה חוזר על עצמו עד לקבלת אירוע של "סגירת האפליקציה".

מושגי יסוד:

אירועים – (Events) פעילויות המתרחשות ביוזמת המשתמש או מערכת ההפעלה.
תור ההודעות – (Message Queue) מערכת ההפעלה יוצרת לכל אפליקציה מבנה נתונים תור אשר מקבל את כל האירועים המיועדים לתוכנית.
לולאת ההודעות (Message Loop) - לולאה אשר ותפקידה למשוך את ההודעות מתור ההודעות ולהעבירן לטיפול האפליקציה, לולאה זו רצה ברקע ומתקיימת לאורך כל ריצת האפליקציה.
מצב ההמתנה (Idle) נוצר כאשר תור ההודעות ריק והאפליקציה ממתינה עד אשר תתקבל הודעה חדשה.

דף הסבר ומצגת :

<http://www.corner.co.il/Articles/ArticleDetails/190> /תכנות-מונחה-אירועים-(Event-Driven)

[מצגת להורדה](#)

שעור 3 – כריכת נתונים DataBinding

כריכת נתונים DataBinding מקשרת בין מספר רכיבים. כך שסביבת ה-WPF עצמה אחראית על קבלת הנתונים והעדכון שלהם, כמעט ללא שורות קוד של המתכנת. כריכת הנתונים נעשית ע"י שימוש בסוגריים המסולסלים – {} ובתוכם יש שימוש במחלקה Binding. באמצעות המחלקה Binding נגדיר את הפרמטרים הנדרושים לכריכת הנתונים.

כריכת הנתונים נחלקת ל-2 סוגים:

א. ניתן לכרוך 2 רכיבי תצוגה. כך ששינוי ברכיב אחד יגרור, אוטומטית, שינוי ברכיב השני. בדוגמא הנ"ל, כל שינוי במאפיין Text של ה-TextBox יגרור אוטומטית שינוי במאפיין Text של ה-TextBlock:

```
<StackPanel Margin="10" >
  <StackPanel Orientation="Horizontal" >
    <TextBlock Text="Source: " FontSize="20"></TextBlock>
    <TextBox Name="txtValue" FontSize="20" Width="300"/>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <TextBlock Text="Target: " FontSize="20"></TextBlock>
    <TextBlock Text="{Binding ElementName=txtValue, Path=Text}" FontSize="20" />
  </StackPanel>
</StackPanel>
```

ב. שימוש ב-DataContext מאפשר לכרוך עצם שהוגדר ב-CodeBehind עם רכיב התצוגה המיועד לו. שיטה זו היא השכיחה הפופולרית ביותר לשימוש והיא תואמת לטכניקת MVC (Model View Controller) בה ניצרך עצמים בקוד (ב-C#) ונגדיר רכיבי תצוגה ב-XAML המציגים עצמים אלו.

לדוגמא:

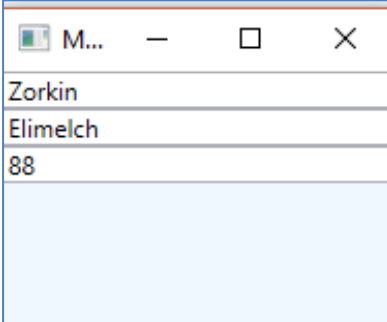
```
class Person // הגדרת מחלקה
{
  // Get & Set במחלקה חייב שיהיו לו את המאפיינים
  public string LastName { get; set; }
  public string FirstName { get; set; }
  public int Age { get; set; }
}

public MainWindow()
{
  InitializeComponent();

  // יצירת עצם מסוג Person עם ערכים, כרצוננו
  Person person = new Person { FirstName = "Elimelech", LastName = "Zorkin", Age = 88 };

  // כריכת העצם עם החלון עצמו תוך שימוש במאפיין DataContext
  this.DataContext = person;
}

<StackPanel>
  <TextBox Text="{Binding LastName}"></TextBox>
  <TextBox Text="{Binding FirstName}"></TextBox>
  <TextBox Text="{Binding Age}"></TextBox>
</StackPanel>
```



Zorkin
Elimelech
88

עדכון הערכים הוא דו-כיווני. אם נשנה את הערכים ב-TextBox נוכל ללחוץ על כפתור ולצפות בתוכן המשתנה, תוך שימוש ב-debugger – נגלה כי המשתנה לו עשינו DataContext עבר שינוי. וזאת, אפילו מבלי שכתבנו שורת קוד אחת מעבר להגדרת ה-Binding.
מצגת (ודוגמאות קוד): <http://www.corner.co.il/Articles/ArticleDetails/23/Data-Binding>

לתשומת ♥:

מכיון שהערכים בתצוגה כרוכים ישירות למאפיין בעצם. אזי, לא ניתן להכניס בהם ערך לא תקין, למשל לא ניתן להכניס ערך לא מספרי (לא int) למאפיין גיל. כלומר, אין צורך בדיקת תקינות לשדה.

משימת תרגול: <

יש לבנות אפליקציית WPF המציגה ומאפשרת עדכון תלמיד בודד וכן יצירת תלמיד חדש, וזאת תוך כדי שימוש בערכים המתקבלים מהמשתמש (להוספה + לעדכון).



קובץ תרגילים fdp.gnidniBataD FPW

קובץ תרגילים WPF

Data Binding

הצגה ועדכון של רשימת תלמידים

שלב 1 – הצגת ועדכון פרטי תלמיד בודד

כתבו אפליקציה באמצעותה נוכל להציג ולעדכן פרטי תלמיד בודד, תוך שימוש ב-DataBinding. כמו-כן נשתמש בכפתור המייצר עצם חדש מסוג "תלמיד".

```
public class Student
{
    private string firstName;
    private string lastName;
    private int age;
    private string email;
}
```

נוסיף כפתור שבלחיצה עליו נוכל לצפות בערך של העצם "תלמיד" ולראות שקיבל את השינויים מהחלון.

כריכת ותווים DataBinding – נושאים מתקדמים

נושאים מתקדמים אלו מיועדים לתלמידים טובים מאד או, לחליפין, מתאימים להוראה ושימוש רק בשלב מתקדם יותר, וגם זאת במסגרת הצרכים והאילוצים בפרויקט הפרטני של התלמיד.

INotifyPropertyChanged <

ניתן לראות שאם נשנה, בקוד, את ערכי העצם המוצג ע"י DataContext – הערכים לא יוצגו כראוי. כלומר רכיבי התצוגה כלל לא "מודעים" לשינוי שחל בעצם אליו הם כרוכים. כדי לפתור בעיה זו נשתמש בממשק INotifyPropertyChanged המוגדר במרחב השמות: System.ComponentModel המגדיר אירוע PropertyChanged – עדכון על שינוי במאפיין.

לשם כך נגדיר את מחלקת Person כך:

```
class Person : INotifyPropertyChanged // הגדרת מחלקה
{
    // Get & Set מאפיין במחלקה חייב שיהיו לו את המאפיינים
    public string LastName { get; set; }
    public string FirstName { get; set; }

    // הוספת דיווח שינויים במצב של פעולת Set, לדוגמא עבור המאפיין Age
    private int age;
    public int Age
    {
        get { return age; }
        set
        {
            if (age != value)
            {
                age = value;
                OnPropertyChanged("Age");
            }
        }
    }

    // מימוש הממשק
    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

בדוגמא זו טיפלנו, לשם הדוגמא, בשינויים של המאפיין Age בלבד.

בדף עצמו ניתן לקבל את האירוע ולהגיב, כרצוננו:

```
public MainWindow()
{
    Person person = new Person { FirstName = "Elimelech", LastName = "Zorkin", Age = 88 };

    person.PropertyChanged += person_PropertyChanged;
    this.DataContext = person;
}

void person_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    // קוד כרצוננו שיופעל ברגע וזוהה שינוי במאפיין של person
}
```

טריגר לשינוי בתצוגה יהיה ע"י הגדרת UpdateSourceTrigger ב-Binding.

לדוגמא:

```
<TextBox Text="{Binding LastName, UpdateSourceTrigger=PropertyChanged}" TextChanged="TextBox_TextChanged" />
```

אם נשתמש, עבור רשימת התלמידים, במשתנה מסוג ObservableCollection<T> ולא סתם במשתנה מסוג List<T> - אזי הוא כבר כולל בתוכו "רגישות" לשינויים. כלומר הוא יתעדכן אוטומטית, גם בהוספה ומחיקה של עצמים מהרשימה, וזאת מבלי צורך לרשום שורת קוד אחת אפילו.

ValueConverter <

ניתן לכתוב קוד, כרצוננו, המקבל עצם, או מאפיין, מה-DataContext ומבצע מה שרוצים. למשל: מקבל את העצם של person ומציג "שם פרטי, שם משפחה" או "מספר טלפון-קידומת" כערך בודד. דבר שהוא שימושי, למשל, כאשר מדובר ברשימה ComboBox של אנשים. הסבר: <https://www.dotnetforall.com/wpf-using-ivalueconverter-code-example>

א. נגדיר מחלקה היורשת מ- IValueConverter.
הממשק מוגדר במרחב השמות System.Globalization.
לדוגמא:

```
public class PeopleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        Person person = value as Person; //type checked
        if (person == null)
            return null;
        else
            return person.FirstName + ", " + person.LastName;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return parameter;
    }
}
```

```

public class TelephoneConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        Person person = value as Person; //type checked
        if (person == null)
            return null;
        else
            return person.Kidomet.KidometNum + "-" + person.Telephone;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return parameter;
    }
}

```

ב. בקובץ התצוגה, XAML, נגדיר בתגית Page, את מרחב השמות של הפרוייקט כך שייקרא local.xmlns:local="clr-namespace:WpfApp8"

בתגית Window.Resources נגדיר את ה-ValueConverter בהם נרצה להשתמש בדף.

```

<Window.Resources>
    <local:PeopleConverter x:Key="peopleConverter" />
    <local:TelephoneConverter x:Key="telephoneConverter" />
</Window.Resources>

```

בסוגריים של ה-Binding נגדיר את ה-ValueConverter הרלוונטי לכריכה (Binding) ה"ל".

```

<TextBox Text="{Binding Converter={StaticResource peopleConverter}}" />

```

בדוגמא זו העצם הכרוך אותו אנו מעבירים ל-Converter הוא כל העצם של ה-DataContext, אין ציון של מאפיין ספציפי ב-Binding עצמו.

◀ : DataValidation

ניתן לכתוב קוד, כרצוננו, המקבל עצם, או מאפיין, מה-DataContext ובודק את תקינותו. למשל, בודק האם אימייל או ת.ז. הם חוקיים או האם גיל הוא הגיוני.. הבדיקה יכולה להתבצע, למשל, תוך כדי שימוש ב-RegularExpression (Regex הנמצא במרחב השמות System.Text.RegularExpressions). הסבר: [/https://www.dotnetforall.com/wpf-validation-examples-exception-rules](https://www.dotnetforall.com/wpf-validation-examples-exception-rules)

נייצר מחלקה היורשת מ-ValidationRule. בה נגדיר את הפעולה המרכזית Validate הבודקת את תקינות העצם Value אליו כרוכים במסגרת ה-Binding. הפעולה תחזיר משתנה מסוג ValidationResult. המכיל, במאפיין הראשון, True או False בהתאם לתקינות הערך ובמאפיין השני, הודעת שגיאה.

```

public override ValidationResult Validate(object value, CultureInfo cultureInfo)
{
    Regex regex = new Regex(@"^\d?(\d{3})\d?\d?(\d{3})\d?(\d{4})$");
    Match match = regex.Match(value.ToString());
    if (match == null || match == Match.Empty)
        return new ValidationResult(false, "Invalid input format");
    else
        return ValidationResult.ValidResult;
}

```

בקובץ ה-XAML נגדיר בתגית Window.Resources מה יקרה באירוע של "שגיאה":

```
<Window.Resources>

  <!--The tool tip for the TextBox to display the validation error message.-->
  <Style x:Key="textBoxInError" TargetType="TextBox">
    <Style.Triggers>
      <Trigger Property="Validation.HasError" Value="true">
        <Setter Property="ToolTip"
          Value="{Binding RelativeSource={x:Static RelativeSource.Self},
            Path=(Validation.Errors)[0].ErrorContent}"/>
      </Trigger>
    </Style.Triggers>
  </Style>
</Window.Resources>
```

וכן נוסיף את ה-ValidationRule ל-Binding, אחרי ש"הרחבנו" אותו.

```
<TextBox Name="num2" Width="200" Style="{StaticResource textBoxInError}">
  <Binding Path="Num" >
    <Binding.ValidationRules>
      <local:AgeRangeRule Min="21" Max="130"/>
      <!--validation Rule with parameters-->
    </Binding.ValidationRules>
  </Binding>
</TextBox>
```

שעור 4 – פקדים המיועדים להצגת אוספים של מידע

ListView הוא הפקד הנפוץ והנוח להצגת אוספים של מידע. המאפיין ItemsSource מקשר לרכיב ה-ListView את הנתונים אותם יציג. כמו-כן ניתן, בצורה קלה, להגדיר את המידע כך שיוצג בצורה המותאמת עבורנו. למשל ניתן בצורה טבלאית פשוטה:

```
<ListView Name="lstView2" Margin="5">
  <ListView.View>
    <GridView>
      <GridViewColumn Header="Last Name" Width="120" DisplayMemberBinding="{Binding LastName}" />
      <GridViewColumn Header="First Name" Width="120" DisplayMemberBinding="{Binding FirstName}" />
      <GridViewColumn Header="Age" Width="50" DisplayMemberBinding="{Binding Age}" />
    </GridView>
  </ListView.View>
</ListView>
```

או בכל צורה, כרצוננו. כולל קיבוץ של הנתונים. (דוגמאות ← במצגת)

רכיב נוסף, פחות שימושי הוא ה-TreeView.

מצגת ודוגמאות קוד : <http://www.corner.co.il/Articles/ArticleDetials/24/Controls-Part-2>

← **משימת תרגול, בהמשך למשימה מהשלב הקודם :**
יש לבנות אפליקציית WPF המציגה רשימת תלמידים ומאפשרת עדכון תלמיד בודד וכן הוספה של תלמיד חדש לרשימה, תוך כדי שימוש בערכים המתקבלים מהמשתמש (להוספה + לעדכון).



קובץ תרגילים fdp.gnidniBataD FPW

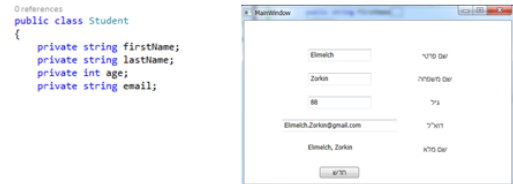
קובץ תרגילים WPF

Data Binding

הצגה ועדכון של רשימת תלמידים

שלב 1 – הצגת ועדכון פרטי תלמיד בודד

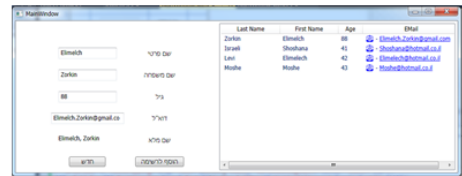
כתבו אפליקציה באמצעותה נוכל להציג ולעדכן פרטי תלמיד בודד, תוך שימוש ב-DataBinding. כמו-כן נשתמש בכפתור המייצר עצם חדש מסוג "תלמיד".



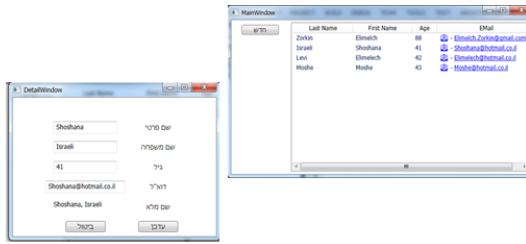
נוסיף כפתור שבלחיצה עליו נוכל לצפות בערך של העצם "תלמיד" ולראות שקיבל את השינויים מהחלון.

שלב 2 – הצגת רשימת תלמידים והוספת תלמיד לרשימה.

הוסיפו לאפליקציה הצגה של רשימת תלמידים תוך שימוש ב-DataBinding. בעת בחירת תלמיד מהרשימה – יוצגו פרטי התלמיד וניתן יהיה לעדכן אותם. כמו-כן נשתמש בכפתור המוסיף את התלמיד המוצג לרשימת התלמידים.



שלב 3 – שימוש ב-2 חלונות, העשרה תוך לימוד עצמי. נפרד את הצגת רשימת התלמידים מהצגת פרטי תלמיד לשני חלונות שונים.



שימו לב:

יצירת חלון חדש: Window win = new Window1();
הצגת החלון: win.ShowDialog();
או win.Show();

יצירת עותק מעצם: Student obj = student.Clone();
זהו בתנאי שמחלקת Student מסוגלת לייצר עותק

```
public class Student: ICloneable
{
    ...
    public object Clone()
    {
        return this.MemberwiseClone();
    }
}
```

משימה למתקדמים – הצגת Xml\Rss

יש לבנות אפליקציית WPF המציגה את שערי המטבעות היומיים, כפי שהורדו מהאתר של בנק ישראל, ומאפשרת לבצע חישוב של המרות מטבע שונות.

תרגיל – הצגת XML\RSS

בנק ישראל מפרסם את שערי המטבעות היומיים בקובץ XML.

הקובץ מפורסם בכתובת: <http://www.boi.org.il/currency.xml>

א - כתוב תכנית שמציגה את שערי המטבעות היומיים ומאפשרת לבצע חישוב של המרות מטבע שונות.

ב - ניתן לקבל גם את שערי המטבעות של יום הסחר הקודם על פי הפורמט הבא:

<http://www.boi.org.il/currency.xml?rdate=yyyymmdd>

לדוגמה:

ה- 05 בינואר 2014:

<http://www.boi.org.il/currency.xml?rdate=20140105>

שימו לב שימי המסחר של שערי המטבעות הם א-ה.

במידה ושער המטבע עלה יש לסמן אותו ב- ↑

במידה ושער המטבע ירד יש לסמן אותו ב- ↓

יש להציג את המידע בפקדים באמצעות Data Binding



תרגיל הצגת LMX_SSR fdp

שעור 5 – נושאי עיצוב מתקדמים

Style & Triggers <

כמו בעולם ה-web נתן להגדיר Style שיתאים לסוג מסויים של תגיות, למשל ל-TextBlock וזאת כדי לא לחזור שוב ושוב על הגדרות העיצוב.

כמו-כן, ניתן להגדיר Triggers, כלומר Style המותנה בתנאי כלשהו. למשל במצב של שגיאה. יכולות אלו משפרות את ההפרדה ואת האי-התלות המתחייבת בין הגרפיקה לאלגוריתמיקה.

מצגת ודוגמאות קוד : <http://www.corner.co.il/Articles/ArticleDetails/25/Style-And-Triggers>

Graphics and Animation <

ההתייחסות לרכיבים הגרפיים (עיגול ריבוע וכיוצ"ב) הינה זהה להתייחסות של כל שאר הרכיבים: טקסט, רשימות וכדומה. בדרך-כלל נשתמש ברכיבים גרפיים על-גבי Layout מסוג Canvas.

ניתן להכניס מספר רכיבים גרפיים לתוך ViewBox כך שהם נשארים קבועים, מבחינת היחס ביניהם, גם אם גודל של ה-ViewBox משתנה.

רכיבים גרפיים נוספים הינם קו, Polyline, ופוליגון.

ב-WPF קיים שימוש נרחב במברשות כך שעיצוב האפליקציה הופך ליפה בקלות יחסית.

קיימים מספר סוגי מברשות :

- **SolidColorBrush** – מיועד לצביעת צורה באמצעות צבע יחיד.
- **LinearGradientBrush** - מיועד לצביעה באמצעות צבע מדורג, מילוי שמשתנה בהדרגה בצורה ליניארית מצבע אחד לצבע שני, או בין מספר גדול יותר של צבעים.
- **RadialGradientBrush** - מיועד לצביעה באמצעות צבע מדורג, מילוי שמשתנה בהדרגה בצורה מעגלית מצבע אחד לצבע שני, או בין מספר גדול יותר של צבעים.
- **ImageBrush** - מתבסס על תמונה כצורת המברשת.

ניתן להגדיר רכיב כך שיכיל גם הסטה (Transformations), סיבוב (RotateTransform), קביעת גודל (ScaleTransform), עיקום (SkewTransform) מהמצב המקורי שלו .

Storyboard

מאפשר לרכז מספר פעולות אנימציה. כמו-גם מאפשר לשלוט על ה- Playback של האנימציה. בנוסף, מאפשר להגדיר אנימציה ב-XAML על ידי Trigger.

מצגת ודוגמאות קוד : <http://www.corner.co.il/Articles/ArticleDetails/42/Graphics-and-Animation>

שעור 6 – דפדוף ומעבר בין דפים / חלונות

חלון הוא אובייקט חשוב ומרכזי ב-WPF, אובייקט החלון הינו העצם הגרפי המייצג את החלון הראשי של האפליקציה. הוא נוצר בעת הרצת האפליקציה ובעת הסגירה שלו – נסגרת גם האפליקציה. מודל ה-Event Driven דורש חלון. החלון מכיל מנגנון שנקרא "משאבת הודעות (Message Pump)" אשר מושך את ההודעות מתור ההודעות לאובייקט ה"חלון". אי לכך, חלון הינו אובייקט גדול וכבד המכיל הרבה משאבים. בחלון ניתן למקם Content אחד בלבד מסוג Layout כלשהו או Frame בתוכו נמקם בכל פעם Page אחר המכיל את התוכן המוצג למשתמש. (תפיסת עבודה המזכירה את ה-MasterPage ו-ContentPages בעולם ה-Asp.net). מטעמי נוחות למשתמש וקלילות הקוד, נשתמש ב-Window אחד וראשי לאפליקציה בו נגדיר Frame (אחד או יותר). ובמהלך ריצת הקוד ננווט (Navigate) בכל פעם דף (Page) אחר ל-Frame. הניווט נעשה תוך שימוש במחסנית של דפים, ולכן קיימת גם אפשרות קלה לדפדוף קדימה או אחורנית.

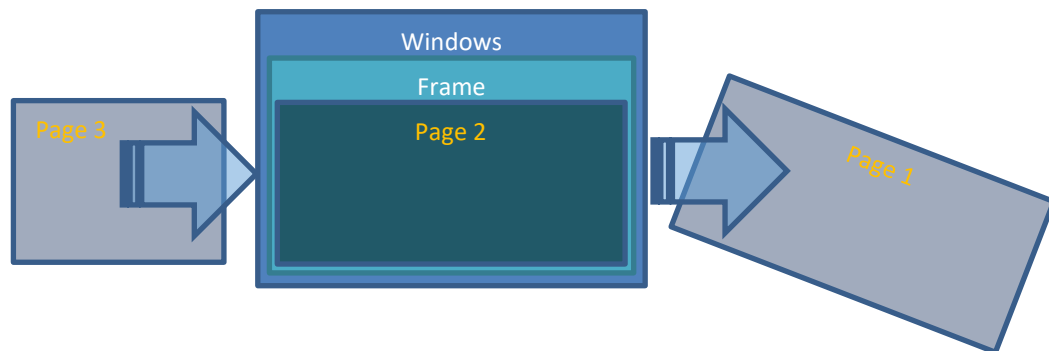
דפדוף יתבצע ע"י הקוד הבא:

```
this.myFrame.Navigate(new Page1()); // מתוך הדף המכיל את ה-frame
```

```
// Navigate ע"י הגענו קודם ע"י Navigate  
NavigationService nav = NavigationService.GetNavigationService(this);  
nav.Navigate(new Page2());
```

דפדוף אחורנית, למשל, יתבצע ע"י:

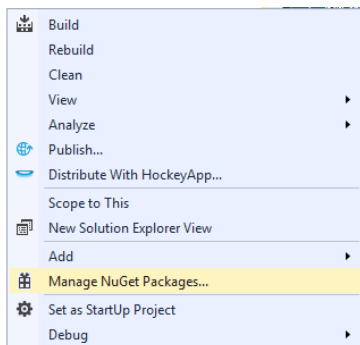
```
if (this.myFrame.CanGoBack)  
{  
    this.myFrame.GoBack();  
}
```



דף הסבר : <http://www.corner.co.il/Articles/ArticleDetails/191/Window,-Frame-and-Page>
[הורדת המצגת](#)

הורדה ושימוש בחבילות תוכנה מוכנות -Nuggets-

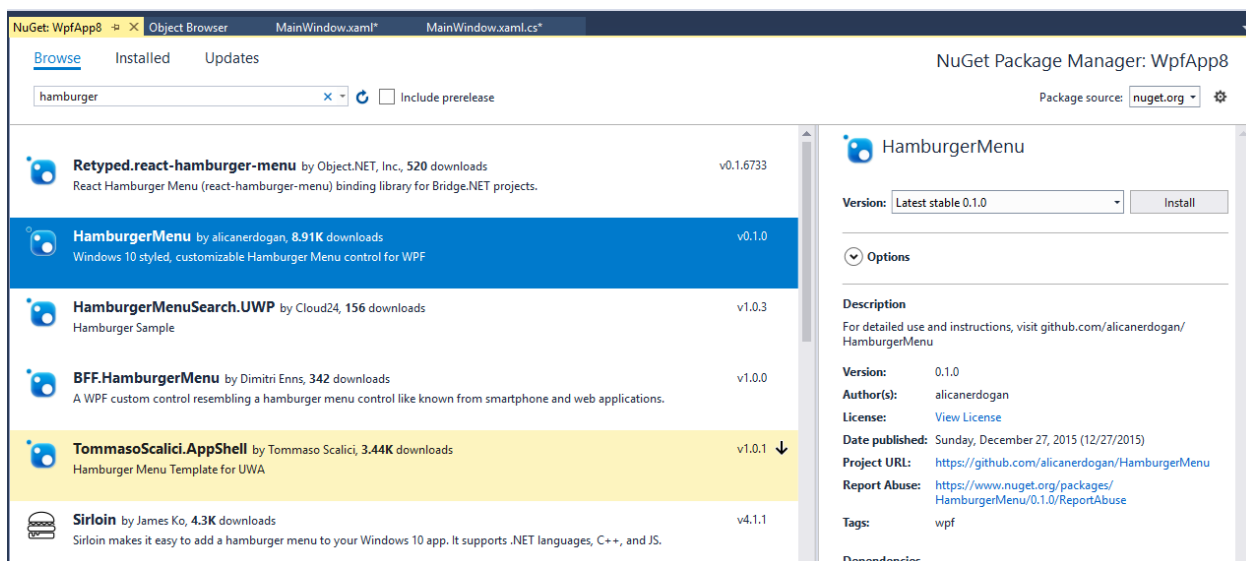
לאחרונה סביבת ה-VisualStudio הפכה יותר לסביבה המשתפת קודים של משתמשים. כדי לחפש ולטעון חבילת קוד נשתמש במנגנון ה-Nuggets.



ימני על הפרוייקט נותן לנו את האפשרות "manage Nuget Packages...". בחלון שנפתח נוכל לחפש ואז להתקין / לחפש / להסיר / לעדכן חבילות כרצוננו. למשל:

אצלי בכיתה התלמידים אוהבים ומורידים את החבילה הנקראת HamburgerMenu הניתנת להורדה בקלות והמכילה תפריט קל ויפה שגם נפתח. בחלון, מצד ימין, ניתן לקבל אינפורמציה מלאה על החבילה, כולל קישור לאתר בו יש הרחבות והסברים על החליבלה הנבחרת.

ניתן לשדרג את התפריט ע"י הוספת תמונת רקע, אייקונים, כרצוננו, כך שהגמישות והיצירתיות רבה.



מי שיחפש ימצא הרבה חבילות בנושאים שונים. ברוב המקרים ניתן גם לקבל את הקוד בצורה חופשית, אם מישהו מתעניין או רוצה לשנות אותו. בדרך-כלל הכי פשוט לחבר את החבילה לאפליקציה שלנו ולעבוד איתה כ"קופסא שחורה".

שעור 7 – עבודה אסינכרונית בחלון WPF

אם ברצוננו לעבוד בצורה א-סינכרונית ובחלון WPF ניתקל בבעיה. סביבת ה-WPF (או WinForms) מחייב אותנו לבצע את כל עבודת ה-UI רק בתהליך שיצר את ה-UI (כלומר בתהליך המוגדר (STA - Single Thread Apartment) וזאת עקב תכתיב של מערכת ההפעלה.

הפתרון: שימוש ברכיב הנקרא BackgroundWorker. העובד ב-Thread אחר תוך כדי איפסור גישה לרכיבי תצוגה (UI Elements).

1. הוספת אובייקט מהמחלקה בתכונה של מחלקת החלון (או ב-XAML אם רוצים):

```
BackgroundWorker background_worker = new BackgroundWorker();
```

2. הגדרת התכונות הבאות בבנאי של החלון: מאפשרים ביטול ומאפשרים קבלת דו"ח התקדמות:

```
public MainWindow()
{
    InitializeComponent();
    background_worker.WorkerSupportsCancellation = true;
    background_worker.WorkerReportsProgress = true;
}
```

3. הגדרת האירועים הבאים:

- ☛ הגדרת האירוע DoWork אשר מועלה כאשר מופעלת המתודה RunWorkerAsync, הטיפול באירוע מממש את הפעולה האסינכרונית שרוצים לבצע ברקע.
- ☛ האירוע ReportProgress יופעל בכל פעם שנרצה לעדכן את הממשק הן בהתקדמות התהליך והן על מנת לעדכן תוצאות ביניים.
- ☛ האירוע השלישי, RunWorkerCompleted מועלה כאשר התהליך מסתיים או מבוטל.

```
public MainWindow()
{
    InitializeComponent();
    background_worker.WorkerSupportsCancellation = true;
    background_worker.WorkerReportsProgress = true;

    background_worker.DoWork += background_worker_DoWork;
    background_worker.ProgressChanged += background_worker_ProgressChanged;
    background_worker.RunWorkerCompleted += background_worker_RunWorkerCompleted;
}
```

4. הרצת התהליך:

```
background_worker.RunWorkerAsync(1000);
```

5. מימוש האירוע DoWork, המממש את התהליך אשר ירוץ ברקע.
ניתן להעביר פרמטר, הפרמטר הוא מטיפוס Object.
יש לבדוק במהלך הביצוע האם התהליך בוטל על מנת לעצור את ריצת התהליך

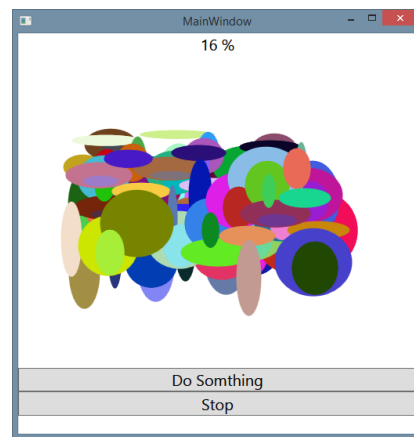
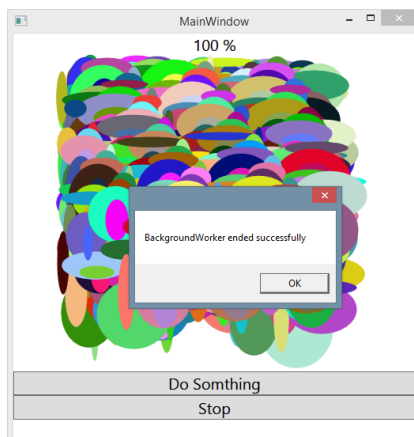
```
void background_worker_DoWork(object sender, DoWorkEventArgs e)
{
    for (int i = 0; i < (int)e.Argument; i++)
    {
        if (background_worker.CancellationPending)
        {
            e.Cancel = true;
            break;
        }
        Parameters parameter = Paint();
        background_worker.ReportProgress(i / 100, parameter);
        Thread.Sleep(10);
    }
}
```

6. הטיפול באירוע ProgressChanged – עדכון ממשק המשתמש:

```
void background_worker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    Parameters parameter = e.UserState as Parameters;
}
```

7. סיום התהליך:

```
void background_worker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Cancelled)
    {
        MessageBox.Show("BackgroundWorker canceled");
    }
    else
    {
        MessageBox.Show("BackgroundWorker ended successfully");
    }
}
```



הסבר מצגת ודוגמאות קוד (תיכנות א-סינכרוני) : <http://www.corner.co.il/Articles/ArticleDetials/51/Threads>

הסבר מצגת ודוגמאות קוד (BackgroundWorker) : www.corner.co.il/Articles/ArticleDetials/52/Threads-in-wpf

[הורדת מצגת - דוגמאות קוד](#)

פתרון נוסף (ופשוט): שימוש ברכיב DispatcherTimer (שייך למרחב השמות System.Windows.Threading). הפעולה מורצת ה-DispatcherTimer, הוא Timer המפעיל כל Interval של זמן פעולה (DoWork). אך בניגוד ל-Timer רגיל, הפעולה מורצת ב-Thread נפרד, אך מאפשרת גישה לרכיבי התצוגה (UI Elements).
מה שהופך את הפתרון הנ"ל לפשוט וקל.

```
public DispatcherTimerSample()
{
    InitializeComponent();

    DispatcherTimer timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromSeconds(1);
    timer.Tick += timer_Tick;
    timer.Start();
}

void timer_Tick(object sender, EventArgs e)
{
    lblTime.Content = DateTime.Now.ToLongTimeString();
}
```

קישור להסבר ולדוגמא: <https://www.wpf-tutorial.com/misc/dispatchertimer/>

ה ה ח ט ז ה א

