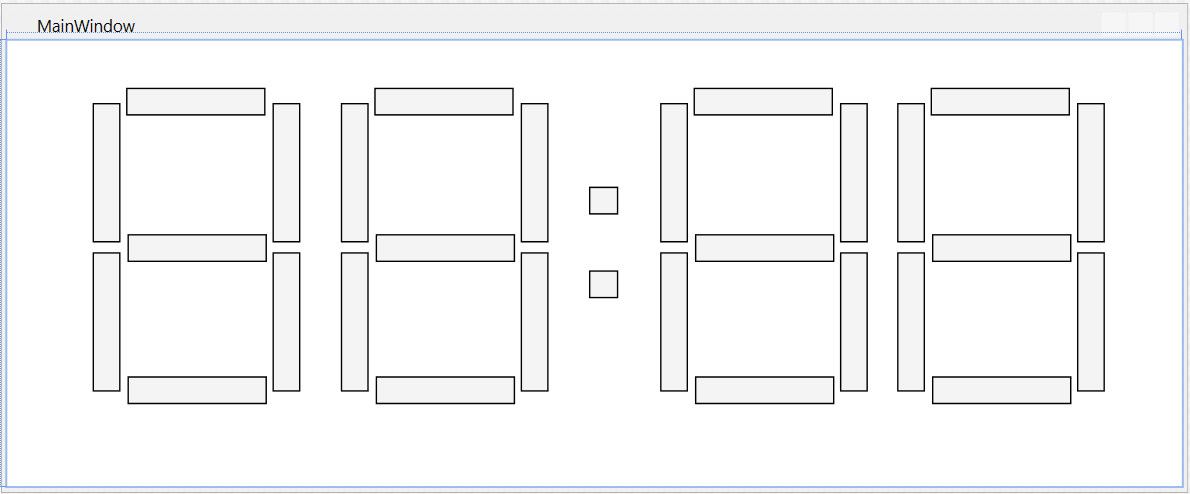
# Aufgabe

Programmierung einer Digitaluhr mit 7-Segment-Anzeige, mit WPF-GUI, mit TDD (Test driven design), getrennter UI- und Business-Logik.

# User Interface

Das User Interface soll in etwa so aussehen:



# Architektur

Das Programm soll in 3 C#-Projekten realisiert werden:

* UI  
  Hauptprogramm, enthält die Controls.   
  Die Controls sollen nur aus Rectangle’s bestehen.  
  Enthält einen Timer, der im Sekundentakt zählt.
* Business Logik  
  Diese Klassenbibliothek soll eine Klasse NumberToLcdConverter mit einer public-Methode „Convert enthalten. Die Methode bekommt einen int von 0 bis 9 und berechnet daraus die 7 Segmente (bool)
* Unit tests  
  In dieser Klassenbibliothek sollen alle Unit Tests implementiert werden.

# Vorgehensweise

TDD (test driven design)

* Zunächst sollen nur Unit Tests für die Methode Convert erstellt werden!
* Dann die Klasse NumberToLcdConverter erstellen!
* Dann soll erst das UI implementiert werden.

# Teilaufgabe 1

Die Klasse NumberToLcdConverter hat folgende public-Methode:   
public string[] Convert(int digit) { . . . }

Implementation aller 10 Tests für Convert.

Dann Implementation der Methode Convert

Dann Implementation des UI mit **EINER** Ziffer. Es soll im Sekundentakt von 0 bis 9 und wieder von vorn zählen.

# Teilaufgabe 2

Implementation eines UserControls mit einer Ziffer. Einbettung des UserControls im WPF-Programm.

# Teilaufgabe 3

Implementation des UI mit allen 4 Ziffern und richtiger Uhrzeit.

# Teilaufgabe 4

Wie kann man den NumberToLcdConverter erweitern, damit man beliebig große Ziffern darstellen kann? Es geht nicht darum, die Rectangles im UI zu vergrößern (was optisch besser wäre), sondern die Ziffern mit mehr Strichen:

Digit Size = 1:

" - "

"| |"

" - "

"| |"

" - "

Digit Size = 2:

" -- "

"| |"

"| |"

" -- "

"| |"

"| |"

" -- "

# WPF Hilfe für Rectangles

Um die Anzeige mit Rectangles in WPF zu realisieren, machen wir ein Layout mit allen benötigten Rechtecken und lassen alle immer Visible. Wir schalten die Striche „an“ und „aus“, indem wir die Füllfarbe ändern. Die Rectangles bekommen Height="20" Width="100". (oder umgekehrt)

Wir machen uns zwei Pinsel für das leere und das ausgefüllte Rechteck:

SolidColorBrush \_FillColor = new SolidColorBrush(Color.FromRgb(0xA6, 0x04, 0x21));  
SolidColorBrush \_EmptyColor = new SolidColorBrush(Color.FromRgb(0xEE, 0xEE, 0xEE));

Das Rechteck in XAML sieht in etwa so aus, wenn man es aus der Toolbar auf die Form gezogen hat:

<Rectangle Fill="#FFF4F4F5" HorizontalAlignment="Left" Height="20" Width="100" Margin="85,34,0,0" Stroke="Black" VerticalAlignment="Top"/>

Wir geben ihm einen Namen und lassen den Rest:

<Rectangle Name="a" . . ./>

Und schalten das Rechteck in der Code behind-Datei mit folgenden Anweisungen:

this.a.Fill = \_FillColor;   
oder:  
this.a.Fill = \_EmptyColor;