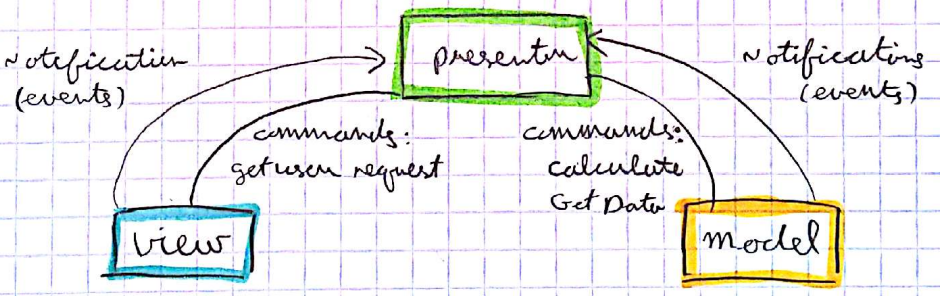
הרצאה 4 ביטיה

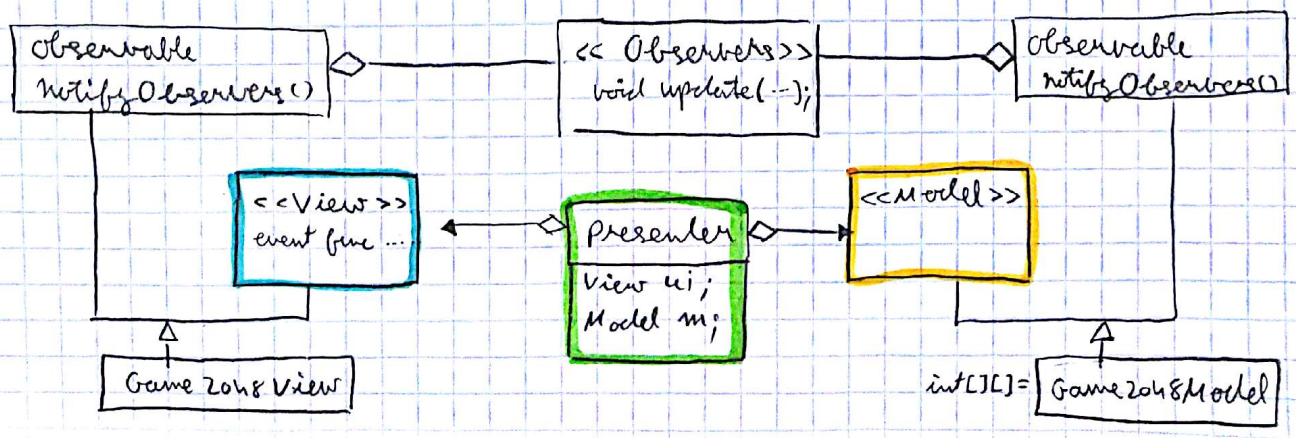
MVP - ה view וה model לא מדברים אל ה קונטרולר. יש פחות תלות מאשר ב MVC.
הקונטרולר חלוי במודל וה view ובמקום שמדבר אותם.

במודל - הלוגיקה והיכולת בתוכנה. כל מה שקורה מאחורי הקלעים באפליקציה שלנו.
ה presenter - מדבר אל המודל וה view. יש לו רפרנס שהוא - m Model.
m מדבר על כל סוג של מודל בהוא בר תרגומה. עדיף עכשיו שתהיה במודל
A, ואחר נרצה להחליף אותו במודל B - רק אכפתי כי m מדבר על כל סוג של מודל
כי הוא מסוג אינטרפייס Model).
בנוסף למודל, יש ל presenter - ע view, והוא יכול לשלוח לה פקודה.
הרבה היתרון שה presenter לא יודע מה מתי להפעיל את הפקודה.

למשל, עדיף ויש לנו עכשיו את הפקודה - getData. הוא ה presenter יודע
מתי ה Data מוכנה? הוא לא יודע, אפשר לקרוא למודל כדי להבין את השינוי
ב data, אבל ה presenter לא יודע מתי לעשות זאת. אלא מי, ה מודל צריך
להודיע ל - presenter מתי יש שינוי. ב - MVC המודל הכיר את הקונטרולר
אבל הוא היה מפעיל פונ' בקונטרולר שאומרת שהמודל מוכן. אבל עכשיו ה מודל
לא מכיר את ה presenter. אז איך נגדע את הדרכים? בעזרת event.




נראה דיאגרמה מפורטת של זה:




Scanned by CamScanner

ב main:

```
Game2048View ui = new Game2048View();
Game2048Model m = new Game2048Model();
Presenter p = new Presenter(ui, m);
```

הפרזנטר הוא מין אובסרבר - הוא יודע לדבר ב view ובמודל.
שם יפעילו אותו את ה - update, הוא יעדכן לדבר את ה ui הפרט את
ה update או ה - m.

ה view וה מודל הם observable. כל אחד מהם מחזיק מערך של
אובסרברס שצופים בהם. בהגדרה שלו נשים בתוכן רק את הפרזנטר.
לכן בהתחלה נכתוב:

```
ui.addObserver(p);
m.addObserver(p);
```

בכל פעם ש view יפעיל את notifyObservers מה יקרה?
ב - p שם ה אובסרבר, יפעל אותו ה update. נניח לדוגמא ה מודל.
מכיוון שזו אותה מתודה של notify אז בתוך המתודה update
יפעל לנו מי פה שלח את ה update.

בתוך ה Program: public delegate void func();
הגדרנו טיפוס func, ושתיה מהסוג הזה יכול להצביע על פונ' שהן מכסות של func.
כלומר: בדיוק אותו הגדרה בפונ' שלו מקבלות פרמטרים ומחזירות void.

בתוך ה אינטרפייס view הגדרנו: event func viewChanged
נכתוב אותו בשם שהוא מתחיל בתוך ה view, והוא מסוג של event func.
כלומר, בתוך ה - viewChanged נכתוב לרשום: אם שם לו פלוס שווה ולהוסיף
לו פונ' מהסוג של func. וכל פעם שקורה את ה viewChanged שם שיקרה
זה שכל הפונ' יופעלו

את אותו הדבר נעשה עבור מודל.

Scanned by CamScanner

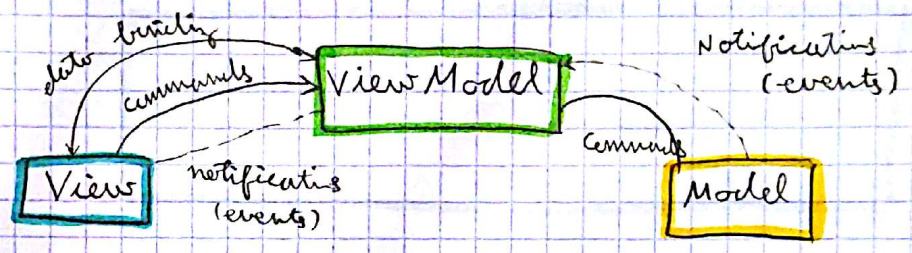
נזכרים מה שבאמת נדרש לעשות זה ליצור void func delegate ואז אבל שיש יתרון ערכים: אי אי שהפעל אותו - object sender, והפי event args (משהו על המידע שקרה).

---

# MVVM

יש view ויש model, ובאמצע המשרת ביניהם ניקרא - ViewModel. למה? כי הוא VM המוח של ה view (ארכיטקטי). ואז אני גם view ארכיטקטי עבור המודל.

ה view מכיל את ה VM, וה-VM שולח נוטיפיקציה ל view.




היתרון המרכזי ב-MVVM זה ה-data binding:

במקום שנצטרך לכתוב קוד שאומר אלו שינוי שדה של טקסט אז תיקש לתפוס ההוא, לסטרינג מסוים ולשנות אותו בהתאמה, או שהסטרינג משתנה, נכתוב קוד שיקשר לשדה של הטקסט ונושה לו set text לאותו הסטרינג: במקום שאנחנו נצטרך לעשות את הדברים האלו כל פעם קרה בצורה אוטומטית, ולא נצטרך לכתוב את ה-קוד-ים האלו וה events הולה. זה חוסך המון עבודה.

מה שבאמת קורה ב MVVM:

מה שהו השתנה במודל - המודל יצריך נוטיפיקציה ל VM (איך זה קורה? יש למודל שמתה מסוג event שאומר המודל מבצע. ה-VM יכול לרשום אליהו delegate אז פה יכו לדע שקרה משהו ובהתאם יכול לשנות מידע אז אומר למודל מה לעשות).

ה V מכיל VM, וה VM שלח נוטיפיקציה ל V. מה זה? אם המודל שלח ל VM נוטיפיקציה ונראה שה-V יכול אז נצריך נוטיפיקציה מה VM ל-V. V שואל מה VM מידע ובהתאם יכול אותו.

Scanned by CamScanner

אירוע זה אובייקט. מבחינתנו כמה אירועים יכולים להתרחש בו־זמנית, וכדי לעשות סדר נקיים אותם לתוך תור. יהיה תהליך מעין שרץ בלולאה וכל פעם שקרה אירוע ושולח אותו לפונ' (event handler) שיודעת איך להתמודד עם האירוע.

כדי שבלולאה הזאת אינו פונ' להפעלה, יש לה event handler. צריך להגדיר לאירוע המסוים. צריך לבדוק קישור בין האירוע לבין ה־event handler המתאים לו.

**בעיה**: נדרש לכתוב קוד פעם אחת שיהיה על פני מס' הפעלה שונות כך שה gui יהיה תואם למס' ההפעלה.

**בפלטפורמות שונות כל מיני פתרונות**:

**AWT**: נשתמש בספרייה המשותפת בינתם מבטן כל מס' נציגי הגרפי של המס' השונות. פותחים אפליקציה אחת כל לוגו ומינימלי.

**swing**: נכתוב ספריות קלות שנותנות את הכלי. הכלים הן נכתבות תיקוני לגבי שהוכרה במס' ההפעלה הרגילות.

היתרון הוא שמתבים את הקוד פעם אחת, כולל הקוד מתבר (כי לא מסתכלים על מס' ההפעלה).

swing לא מספק פתרון טוב לבעיה שלנו כי זה נעלם. ציבור של gui משלו זה תינתן לא בשימוש מס' ההפעלה.

**SWT**: הכל מקום שניתן לתת בו קוד native - ניתן (מיצ נקבל מבוא שמש אינשה כמו מס' ההפעלה). כל מקום שחסר - נשלים עם הקוד.

מה שעושים זה מנוהלים jack מתאים עבור כל מס' הפעלה. ה swt שמשתמש נשאר ללא שינוי והדבר היחיד שמשתנה זה ה jar.

---

**3. מודרניות תכנות**

1. לא סדר מגדולים מחלקה. בדרך המחלקות שנמצאות אינם צריכים לכתוב מחלקה קיימת ובה כל הקוד שאותו נתכתבים הוא להוסיף בתוך תשתית קיימת. אם הגבנו מחלקה "A", לפי נניח איתור ל xaml ונתחיל לקודד הכל מהמחלקה אלו ניתן מחלקה קיימת.

Scanned by CamScanner

2. אם כל הקוד הזה נמצא בתוך מחלקה כ- זה טיפוס. נוצרה שיהיה לנו לינאר אובייקט מוגדר בשורה הלא שיקח לינאר כפרמטר. נוצרה גם שנוכל להסיר את האובייקט מה gui בשורה הלא שיקח להסיר כפרמטר. וגם זה לא יהיה בתוך מחלקה, אלא זה יהיה אוסף של events handlers בלי מיקומם הקוד לא יהיה כל עדכון את הטיפוס ולהשתמש בו לשלב, בפרוייקט אחר.

3. בפיקדונאלים של הטיפוס הזה שמקדמים אומר דרוש להיות הפרדה בין דברים שנכנים בכל פרוייקט לבין דברים ששתנים בין פרוייקט לפרוייקט.
למשל, אם יש לנו את המבוך ויש לנו את הדמות ל 4 הכיוונים אז זה משהו שיהיה נכון לכל מבוך, אבל הדיר של הדמות זה משהו משתנה - אולי בכל מבוך נרצה דמות אחרת? כלומר - את הדמות אפשר להציג בעזרת מבוך הבא אבל הדיר משתנה. אז זה - נשתמש ב strategy pattern, או ב delegate.
(את הדברים שמשתנים בפרוייקט נקרא מבחוץ).

---

## user control

בשביל main window נוצר לצורך את ה user control שיצרנו. איך נעשה את זה?
נייצר אובייקט מהסוג של ה user control ב main window

למשל, אם יצרנו - user control בשם TypePuzzle, אז בתוך ה main window
נייצר namespace בעזרת הבאה: אם TypePuzzle נקרא את Controls
נכתוב: xmlns:Controls="clr-namespace:" (-> השלמה אוטומטית)
ואז בקוד שנרשום "<Controls:" זה יש לנו ל TypePuzzle
כלומר נקבל את כל מה שקבוע תחת אותו namespace - זה כמו לעשות include
ולא הנו צריכים לעשות את כל זה?

בתוך ה xaml הנשי של main window נוצר להוסיף אובייקט חסון TypePuzzle, אבל הוא לא מכיר אותו כי הוא לא יודע בדיוק namespace הוא נמצא.

Scanned by CamScanner