

חלק 9

מנשקי אדם־מכונה

(גראפיים)

תוכנה 1 - חלק 9

1

שקף אחד על הנדסת מנשקי אנוש

- קונסיסטינטיות; המנשק צריך להתנהג בהתאם לציפיות המוקדמות של המשתמש/ת; פעולות אוטומטיות (גזור-הדבק, למשל), המראה של פריטים (צלמיות, למשל), המראה וההתנהגות הכללית של התוכנית, של הפלטפורמה
- המשתמש/ת בשליטה, לא המחשב; חזרה אחורה באשף, ידיעה מה המצב הנוכחי של התוכנית ומה היא עושה כרגע
- יעילות של המשתמש, לא של המחשב; חומרה היא זולה, משכורות הן יקרות, ואכזבות הן עוד יותר יקרות
- התאמה לתכיפות השימוש וללימוד התוכנה; האם משתמשים בה באופן חד פעמי (אשף לכתיבת צוואות) או יומיומי (דואל); גם משתמש יומיומי בתוכנה היה פעם מתחיל חסר ניסיון
- פעולה ישירה על ייצוג נראה של עצמים, ללא שיום (בד"כ)

תוכנה 1 - חלק 9

4

ענווה

- לנו כמפתחי תוכנה יש רק חלק קטן בפיתוח מנשקים גרפיים
- פיתוח מנשק גרפי מתחיל במהנדס מנשקי אנוש: איש/אשת מקצוע שיודעים לפתח מנשק שיהיה מובן, יעיל, ונעים
- מהנדס מנשקי האנוש יודע גם למדוד את איכות המנשק על קבוצות משתמשים ולתקן את המנשק בהתאם; גם כאן בדיוק הן מרכיב חשוב, אבל דרך ביצוען שונה לגמרי
- פיתוח המנשק ממשיך במעצב/ת גראפי/ת; עיצוב גרוע או סידור גרוע של האלמנטים על המסך מקשים על הבנת המנשק ועל השימוש בו
- מהנדס המנשקים והמעצב דואגים שגם משתמשים עם מוגבלויות (בעיקר ליקויי ראייה) יוכלו להשתמש בתוכנה
- אם יש מרכיבי קול או מישוש (טעם וריח?), צריך לעצב אותם

תוכנה 1 - חלק 9

2

שקף אחד על עיצוב גראפי (של מנשקים)

- קונסיסטינטיות
- קונטרסט להדגשת מה שבאמת דרוש הדגשה; עומס ויזואלי מפחית את הקונטרסט
- ארגון ברור של המסך (בדרך כלל תוך שימוש בסריג)
- כיוון וסדר ברורים לסריקת המידע (מלמעלה למטה משמאל לימין, או ימין לשמאל)
- העיצוב הגרפי של מנשק של תוכנית בדרך כלל אינו מוחלט; המשתמש ו/או הפלטפורמה עשויים להשפיע על בחירת גופנים ועל הסגנון של פריטים גראפיים (כפתורים, תפריטים); העיצוב צריך להתאים את עצמו לסביבה

תוכנה 1 - חלק 9

5

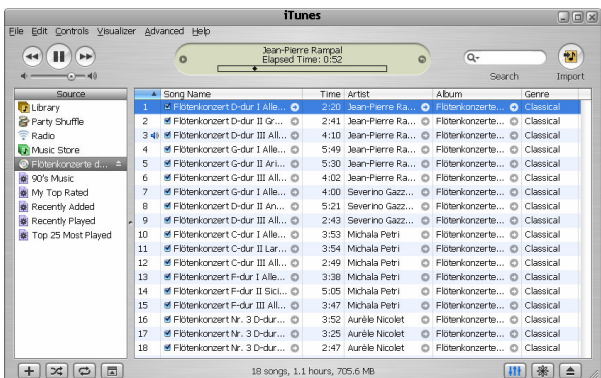
מפתחים רק מבצעים את התיכון והעיצוב

- מהנדס מנשק האנוש מחליט איך המנשק יתנהג, המעצב מחליט איך בדיוק הוא יראה (ישמע, יורגש)
- תוכניתנים מממשים את המנשק הגרפי בהתאם

תוכנה 1 - חלק 9

3

ועכשיו, למימוש



תוכנה 1 - חלק 9

6

מבנה המימוש

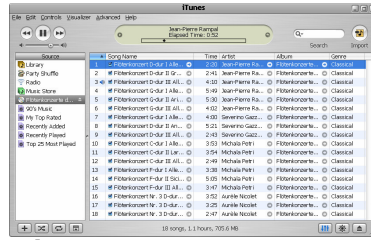
```
public class GoogleBrowser {
    private Shell shell = null;
    private Button button = null;
    private Text text = null;
    private Browser browser = null;
    public static void main(String[] args) {
        call createShell and run event loop }
    private void createShell() { create the GUI }
    private static String search(String q) {
        send query to Google and return the first URL }
}
```

10

תוכנה 1 - חלק 9

שלושת הצירים של תוכנה גראפית

- אלמנטים מסוגים שונים על המסך (היררכיה של טיפוסים)
- הארגון הדו-מימדי של האלמנטים, בדרך כלל בעזרת מיכלים
- ההתנהגות הדינמית של האלמנטים בתגובה לפעולות של המשתמש/ת (הקלדה, הקלקה, גרירה)



7

תוכנה 1 - חלק 9

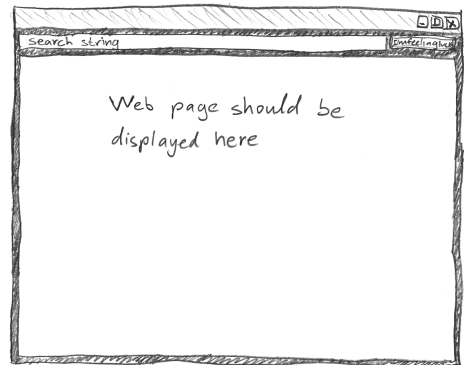
Widgets

- השדות text, button, browser, ו-shell יתייחסו לרכיבי הממשק הגראפי; רכיבים כאלה נקראים widgets
- מעטפת (shell) הוא חלון עצמאי שמערכת ההפעלה מציגה, ושאינו מוכל בתוך חלון אחר; החלון הראשי של תוכנית הוא מעטפת, וגם דיאלוגים (אשף, דיאלוג לבחירת קובץ או גופן, וכדומה) הם מעטפות
- עצם המעטפת בג'אווה מייצג משאב של מערכת ההפעלה
- הרכיבים האחרים הם אלמנטים שמוצגים בתוך מעטפת, כמו כפתורים, תפריטים, וכדומה; חלקם פשוטים וחלקם מורכבים מאוד (כמו Browser, רכיב להצגת HTML)
- לפעמים הם עצמים שממופים לבקרים שמערכת ההפעלה מציגה בעצמה (controls), ולפעמים הם עצמי ג'אווה טהורים

11

תוכנה 1 - חלק 9

דוגמה ראשונה: דפדפן זעיר מגולגל



8

תוכנה 1 - חלק 9

הלולאה הראשית

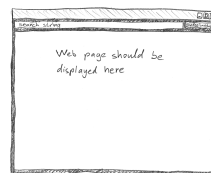
```
public static void main(String[] args) {
    Display display = Display.getDefault();
    GoogleBrowser app
        = new GoogleBrowser();
    app.createShell();
    while (!app.shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}
```

12

תוכנה 1 - חלק 9

מה הדפדפן אמור לעשות

- המשתמש תקליד מחרוזת חיפוש בשדה בצד שמאל למעלה
- לחיצה על הכפתור I'm feeling lucky מימין לשדה הטקסט תשלח את מחרוזת החיפוש ל-Google
- כאשר נתקבל תשובה, הדפדפן ישלף מתשובת Google את הכתובת (URL) הראשונה ויטען אותה לרכיב הצגת ה-HTML בתחתית המסך, ויצג בכותרת החלון שתציג את ה-URL
- נממש את הדפדפן בעזרת ספרייה למימוש מנשקים גראפיים בשם SWT (Standard Widget Toolkit)
- ספריות אחרות למימוש מנשקים גראפיים בג'אווה הן AWT ו-Swing



9

תוכנה 1 - חלק 9

המשך יצירת המנשק

```
button = new Button(shell, SWT.NONE);
button.setText("I'm feeling lucky");
button.setLayoutData(new
    GridData(SWT.RIGHT, SWT.CENTER,
        false, false));
browser = new Browser(shell, SWT.NONE);
browser.setLayoutData(new
    GridData(SWT.FILL, SWT.FILL, fill both ways
        false,
        true, row grabs vertical space
        2, 1)); widget spans 2 columns
```

16

תוכנה 1 - חלק 9

יצירת המנשק הגראפי

```
private void createShell() {
    shell = new Shell();
    shell.setText("Browser Example");
    shell.setLayout(new layout manager: a grid with
        GridLayout(2, false)); 2 unequal columns
    text = new Text(shell, SWT.BORDER);
    text.setLayoutData(new
        GridData(SWT.FILL, horizontal alignment
            SWT.CENTER, vertical alignment
            true, grab horizontal space
            false)); don't grab vertical space
```

13

תוכנה 1 - חלק 9

הפרוצדורה שהכפתור מפעיל

```
button.addSelectionListener(
    new SelectionAdapter() {
        public void
        widgetSelected(SelectionEvent e) {
            String query = text.getText();
            String url = search(query);
            shell.setText(url);
            browser.setUrl(url);
        }
    });
```

17

תוכנה 1 - חלק 9

פריסת הרכיבי המנשק במעטפת

- מעטפות הם רכיבי מנשק שמיועדים להכיל רכיבי מנשק
- את הרכיבים המוכלים צריך למקם; רצוי לא למקם אותם באופן אבסולוטי (ערכי x ו-y בקואורדינטות של הרכיב המכיל)
- מנהלי פריסה (layout managers) מחשבים את הפריסה על פי הוראות פריסה שמצורפות לכל רכיב מוכל
- GridLayout הוא מנהל פריסה שממקם רכיבים בתאים של טבלה דו-מימדית; רכיבים יכולים לתפוס תא אחד או יותר
- רוחב עמודה/שורה נקבע אוטומטית ע"פ הרכיב הגדול ביותר
- GridLayout הוא עצם שמייצג הוראות פריסה עבור GridLayout; כאן ביקשנו מתיחה אופקית של הרכיב עצמו בתוך העמודה ושל העמודה כולה

14

תוכנה 1 - חלק 9

אירועים והטיפול בהם

- מערכת ההפעלה מודיעה לתוכנית על אירועים: הקשות על המקלדת, הזזת עכבר והקלקה, בחירת אלמנטים, ועוד
- ההודעה מתקבלת על ידי עצם יחיד (singleton) מהמחלקה Display, שמייצג את מערכת ההפעלה (מע' החלונות)
- קבלת אירוע מעירה את התוכנית מהשינה ב-sleep
- כאשר קוראים ל-readAndDispatch, ה-display מברר לאיזה רכיב צריך להודיע על האירוע, ומודיע לו
- הרכיב מפעיל את העצמים מהטיפוס המתאים לסוג האירוע שנרשמו להפעלה על ידי קריאה ל-add*Listener

18

תוכנה 1 - חלק 9

בניית רכיבי מנשק

- בנאי שבונה רכיב מנשק מקבל בדרך כלל שני ארגומנטים: ההורה של רכיב המנשק בהיררכיית ההכלה, והסגנון של רכיב המנשק
- כאשר בנינו את שדה הטקסט, העברנו לבנאי את הארגומנטים shell (ההורה) ו-SWT.BORDER (סיביות סגנון)
- למעטפת אין הורה (אבל יכלו להיות לה סיביות סגנון)
- את תכונות ההורות והסגנון אי אפשר לשנות לאחר שהרכיב נבנה
- רכיבים שונים משתמשים בסיביות סגנון שונות; למשל, למעטפת יכולה להיות או לא להיות מסגרת עם כפתורי סגירה ומיזעור (המסגרת נקראת trim), אבל לרכיב פנימי אי אפשר לבחור סגנון שכולל מסגרת כזו

15

תוכנה 1 - חלק 9

שלוש גישות לטיפול באירועים

- בעזרת טיפוסים סטטיים ספציפיים לסוג האירוע; למשל, `KeyListener` הוא ממשק שמגדיר שני שירותים, `KeyPressed` ו-`KeyReleased`, שכל אחד מהם מקבל את הדיווח על האירוע בעזרת עצם מטיפוס `KeyEvent`
- ללא טיפוסים סטטיים שמתאימים לאירועים ספציפיים; האירוע מפעיל עצם מטיפוס `Listener` שמממש שירות `handleEvent`, והאירוע מזווח בעזרת טיפוס `Event`; יותר יעיל, פחות בטוח
- יש ספריות של ממשקים גראפיים, למשל `AWT`, שמשתמשות בירושה: המחלקה שמייצגת את הממשק שלנו מרחיבה את `Frame` (מקביל ל-`Shell`) ודורסת את השירות `handleEvent`, ש-`Frame` קוראת לו לטיפול באירועים

תוכנה 1 - חלק 9

19

כמעט סיימנו

- נותרו רק שתי שורות שלא ראינו ב-`createShell`,
`button.addSelectionListener(...)`;
`shell.pack()`; *causes the layout manager to lay out the shell*
`shell.open()`; *opens the shell on the screen*
}
- והפרוצדורה שמחפשת במנוע החיפוש Google ומחזירה את ה-URL של התשובה הראשונה

תוכנה 1 - חלק 9

22

דוגמה לשימוש במאזין לא ספציפי

```
button.addListener(  
    SWT.Selection, the event we want to handle  
    new Listener() {  
        public void handleEvent(Event e) {  
            String query = text.getText();  
            String url = search(query);  
            shell.setText(url);  
            browser.setUrl(url);  
        }  
    });
```

תוכנה 1 - חלק 9

20

חיפוש ב-Google (לא ממש רלוונטי)

```
private static String search(String q) {  
    GoogleSearch s = new GoogleSearch();  
    s.setKey (" my secrete key");  
    s.setProxyHost ("proxy.tau.ac.il");  
    s.setProxyPort (8080);  
    s.setQueryString(q);  
    s.setStartResult(0);  
    try {  
        GoogleSearchResult r = s.doSearch();  
        return  
            (r.getResultElements())[0].getURL()  
    }  
}
```

תוכנה 1 - חלק 9

23

Adapter לעומת Listener

- לכפתור הוספנו מאזין ספציפי ממחלקה אנונימית שמרחיבה את `SelectionAdapter`
- `SelectionAdapter` היא מחלקה שמממשת את הממשק `KeyListener` שמגדיר שני שירותים
- ב-`SelectionAdapter`, שני השירותים אינם עושים כלום
- הרחבה שלה מאפשרת להגדיר רק את השירות שרוצים, על פי סוג האירוע הספציפי שרוצים לטפל בו; ארועים אחרים יטופלו על ידי שירות שלא עושה כלום
- אם המחלקה האנונימית הייתה מממשת ישירות את `KeyListener`, היא הייתה צריכה להגדיר את שני השירותים, כאשר אחד מהם מוגדר ריק; מסורבל

תוכנה 1 - חלק 9

21

והתוצאה,



תוכנה 1 - חלק 9

24

סיכום ביניים

- ראינו את המחלקות שמייצגות רכיבי ממשק גראפי
- ראינו איך נרשמים להגיב על אירוע כגון לחיצה על כפתור
- ראינו כיצד מגדירים את הפריסה של הרכיבים על המסך
- האם הממשק הגראפי של התוכנית מוצלח? לא, הכפתור מיותר, ובעצם, אפשר היה להשתמש בשדה הטקסט גם עבור חיפוש וגם עבור הקלדת URL באופן ישיר
- המחלקות שמייצגות את רכיבי הממשק מורכבות מאוד: צריך ספר או מדריך מקוון, צריך להתאמן, ורצוי להשתמש במנגנון עריכה ייעודי לממשקים גראפיים (GUI Builder)

תוכנה 1 - חלק 9

25

יתרונות וחסרונות של Pluggable L&F

- מאפשר להגדיר מראות חדשים לרכיבים; שימושי עבור משחקים, עבור תוכניות שרוצים שלא יראו כמו תוכנות מחשב (בעיקר נגני מוסיקה וסרטים), ובשביל מיתוג (branding)
- מאפשר לבנות יישומים עם מראה אחיד על כל פלטפורמה; שימושי ליישומים ארגוניים
- קשה לממש look & feel חדש
- סכנה של מראה מיושן, אם מערכת ההפעלה החליפה את המראה של הרכיבים אבל האמולציה לא עודכנה (למשל מראה של חלונות 2000 על מערכת חלונות XP)
- אי התאמה לקביעת התצורה של המשתמשת (אם היא בחרה למשל להשתמש במראה של חלונות 2000 על חלונות XP)

תוכנה 1 - חלק 9

28

שחרור משאבים

- חלק מהעצמים שמרכיבים את הממשק הגראפי מייצגים למעשה משאבים של מערכת ההפעלה, כמו חלונות, כפתורים, צבעים, גופנים, ותמונות
- כאשר עצם שמייצג משאב נוצר, הוא יוצר את המשאב, ואם לא נשחרר אותו, נדלדל את משאבי מערכת ההפעלה
- למשל, צבעים בתצוגה של 8 או 16 סיביות לכל פיקסל
- ב-SWT, אם יצרנו עצם שמייצג משאב של מערכת ההפעלה, צריך לקרוא לשירות dispose כאשר אין בו צורך יותר
- dispose משחרר גם את כל הרכיבים המוכלים
- על מנת לחסוך במשאבים, יש הפרדה בין מחלקות שמייצגות משאבים (למשל Font) וכאלה שלא (FontData)

תוכנה 1 - חלק 9

26

תחושת הממשק בפלטפורמות שונות

- בחלונות ולינוקס משתמשים בצירופים Control-C, Control-V עבור גזור והדבק
- במחשבי מקינטוש יש מקש Control, אבל יש גם מקש Command, וגזור והדבק מופעלי על ידי Command-C, Command-V, ולא על ידי צירופי Control
- תוכנית שמפעילה גזור והדבק ע"י Control-C/V תחוש לא טבעית במקינטוש
- ב-SWT מוגדרים המקשים Control וכדומה, אבל גם "מקשים מוכללים" MOD1, MOD2, MOD3, וכאשר MOD1 ממופה ל-Control בחלונות אבל ל-Command במקינטוש
- בעיה דומה: הפעלת תפריט הקשר; הקלקה ימנית בחלונות, אבל במקינטוש יש לעכבר רק לחצן אחד; מוגדר אירוע מיוחד

תוכנה 1 - חלק 9

29

Look and Feel

- מערכות הפעלה עם ממשק גראפי מספקות שירותי ממשק (למשל, Windows ו-MacOS; אבל לא לינוקס ויוניקס)
- שימוש בממשקים של מערכת ההפעלה תורם למראה אחיד ולקונסיסטנטיות עם ציפיות המשתמש ועם קביעת התצורה שלו (אם יש דרך לשלוט על מראה הרכיבים, כמו בחלונות)
- ספריות ממשקים משתמשות באחת משתי דרכים על מנת להשיג אחידות עם הממשקים של מערכת ההפעלה
- שימוש ישיר ברכיבי ממשק של מערכת ההפעלה; SWT, AWT
- אמולציה של התנהגות מערכת ההפעלה אבל כמעט ללא שימוש ברכיבי הממשק שלה (פרט לחלונות); למשל Swing, Qt, JFace, זה מאפשר להחליף מראה, pluggable look & feel

תוכנה 1 - חלק 9

27

פריסה נכונה

- פריסה נכונה של רכיבים היא אחד האתגרים המשמעותיים בפיתוח ממשק גראפי
- התוכנית צריכה להבטיח עד כמה שאפשר שהממשק יראה תמיד "נכון", למרות מסכים בגדלים שונים וברזולוציות שונות, כאשר רכיבים כגון טבלאות ושדות טקסט מציגים מעט מידע או הרבה, וכאשר המשתמשת מקטינה או מגדילה את החלון
- אלגוריתמי פריסה מתוחכמים עבור מיכלים, כגון GridLayout, מסייעים, אבל צריך להבין כיצד מתבצעים חישובי הפריסה וכיצד להשפיע עליהם

תוכנה 1 - חלק 9

30

חישובי פריסה

- חישובי פריסה מתבצעים ברקורסיה על עץ ההכלה, אבל בשני כיוונים: מלמטה למעלה (מרכיבים מוכלים למיכלים שלהם עד מעטפות חיצוניות) ומלמעלה למטה
- חישובים מלמטה למעלה (postorder ברקורסיה) עונים על השאלה "באיזה גודל רכיב או מיכל רוצים להיות?"
- חישובים מלמעלה למטה (preorder) עונים על השאלה "בהינתן גודל למיכל, היכן ובאיזה גודל למקם כל רכיב?"

תוכנה 1 - חלק 9

31

חישובים מלמעלה למטה

- השירות layout פורס את הרכיבים המוכלים במיכל לאחר שגודל המיכל נקבע (על ידי setSize או setBounds)
- המיכל פורס בעזרת אלגוריתם הפריסה שנקבע לו
- לפעמים, הפריסה לא תלויה בגודל הרצוי של הרכיבים; למשל, אלגוריתם הפריסה FillLayout מחלקת את המיכל באופן שווה בין הרכיבים המוכלים, לאורך או לרוחב
- בדרך כלל, הפריסה כן תלויה בגודל הרצוי של הרכיבים; ב-GridLayout, למשל, הרוחב של עמודות ושורות לא נמתחות נקבע על פי הרכיב עם הגודל הרצוי המקסימאלי בהן, ושאר העמודות והשורות נמתחות על מנת למלא את שאר המיכל
- רכיבים זוכרים את גודלם הרצוי כדי לא לחשוב שוב ושוב

תוכנה 1 - חלק 9

34

פריסה מלמטה למעלה

- כל רכיב צריך לדעת באיזה גודל הוא רוצה להיות (שם השירות ב-SWT הוא computeSize, בספריות אחרות preferred Size)
- יש ספריות שבהן כל רכיב צריך לדעת מה גודלו המינימלי (minimumSize), אבל לא ב-SWT
- רכיב פשוט מחשב את גודלו הרצוי על פי תוכנו (למשל על פי גודל התווית או הצלמית שהוא מציג) ועל פי החוקים היוזאליים של המנשק (רוחב המסגרת סביב התווית, למשל)
- מיכל מחשב את גודלו הרצוי על ידי חישוב רקורסיבי של הגודל הרצוי של הרכיבים המוכלים בו, והרצת אלגוריתם הפריסה של המיכל על הגדלים הללו
- אבל זה מסתבך

תוכנה 1 - חלק 9

32

אריזה הדוקה

- השירות pack מחשב את גודלו הרצוי של רכיב או מיכל וקובע את גודלו לגודל זה; המיכל נאזר באופן הדוק
- שימושי בעיקר לדיאלוגים לא גדולים
- **סכנת חריגה:** אם המיכל מכיל רכיב עם גודל רצוי ענק (טבלה ארוכה, תווית טקסט ארוכה), החלון עלול לחרוג מהמסך
- עבור חלונות (כולל דיאלוגים), עדיף לחשב את הגודל הרצוי ולקבוע את גודל המעטפת בהתאם רק אם אינו חורג מהמסך, אחרת להגביל את האורך ו/או הרוחב
- **סכנת איטיות:** אם המיכל מכיל המון רכיבים, חישוב גודלו הרצוי יהיה איטי (רוחב עמודה בטבלה ארוכה); כדאי להעריך את הגודל הרצוי בדרך אחרת

תוכנה 1 - חלק 9

35

שני סיבוכים

- יש רכיבים שגובהם תלוי ברוחבם או להיפך; למשל תווית או סרגל כלים שניתן להציג בשורה אחת ארוכה, או לפרוס על פני מספר שורות קצרות
- לכן, computeSize מאפשר לשאול את הרכיב מה גובהו הרצוי בהנתן רוחב מסוים ולהיפך, ולא רק מה הגודל הרצוי ללא שום אילוף
- יש רכיבים שעלולים לרצות גודל עצום, כמו עורכי טקסט, טבלאות, ועצים (ובעצם כל רכיב שעשוי לקבל פס גלילה)
- הגודל הרצוי שהם מדווחים עליו אינו מועיל; צריך לקבוע את גודלם על פי גודל המסך, או על פי מספר שורות ו/או מספר תווים רצוי

תוכנה 1 - חלק 9

33

אלגוריתמי אריזה

- FillLayout: רכיבים בשורה/עמודה, גודל אחיד לכולם
- RowLayout: רכיבים בשורה/עמודה, עם אפשרות שבירה למספר שורות/עמודות, ועם יכולת לקבוע רוחב/גובה לרכיבים
- GridLayout: כפי שראינו, סריג שניתן לקבוע בו איזה שורות ועמודות ימתחו ואיזה לא, ולקבוע רוחב/גובה לרכיבים
- FormLayout: מיקום בעזרת אילוצים על ארבעת הקצוות (או חלקם) של הרכיבים; אילוצים יחסיים או אבסולוטיים ביחס למיכל (למשל, באמצע רוחבו ועוד 4 פיקסלים) או אילוצים אבסולוטיים ביחס לנקודת קצה של רכיב אחר (דבוק לרכיב אחר או דבוק עם הפרדה של מספר פיקסלים נתון)
- StackLayout: ערימה של מיכלים בגודל זהה אבל רק העליון נראה; שימושי להחלפה של תוכן מיכל או חלון

תוכנה 1 - חלק 9

36

סיכום מנשקים גראפיים

- דע/י את מקומך
- שלושה מנגנונים כמעט אורתוגונליים: ירושה, הכלה, אירועים
- חלק מהפגמים במנשק גראפי נובעים מפריסה לא נכונה של רכיבים במיכל, מתגובה לא מספיקה או חסרה לאירועים
- לא קשה, אבל צריך להתאמן בתכנות מנשקים גראפיים
- ספר, GUI Builder, ודוגמאות קטנות מסייעים מאוד
- ממשקים מורכבים בנויים לפעמים תוך שימוש בעצמי תיווך בין רכיבי המנשק ובין החלק הפונקציונאלי של התוכנית (המודל); למשל, jface מעל SWT; קשה יותר ללמוד להשתמש בעצמי התיווך, אבל הם מקטינים את כמות הקוד שצריך לפתח ומשפרים את הקונסיסטנטיות של המנשק