תוכנה 1 בשפת Java

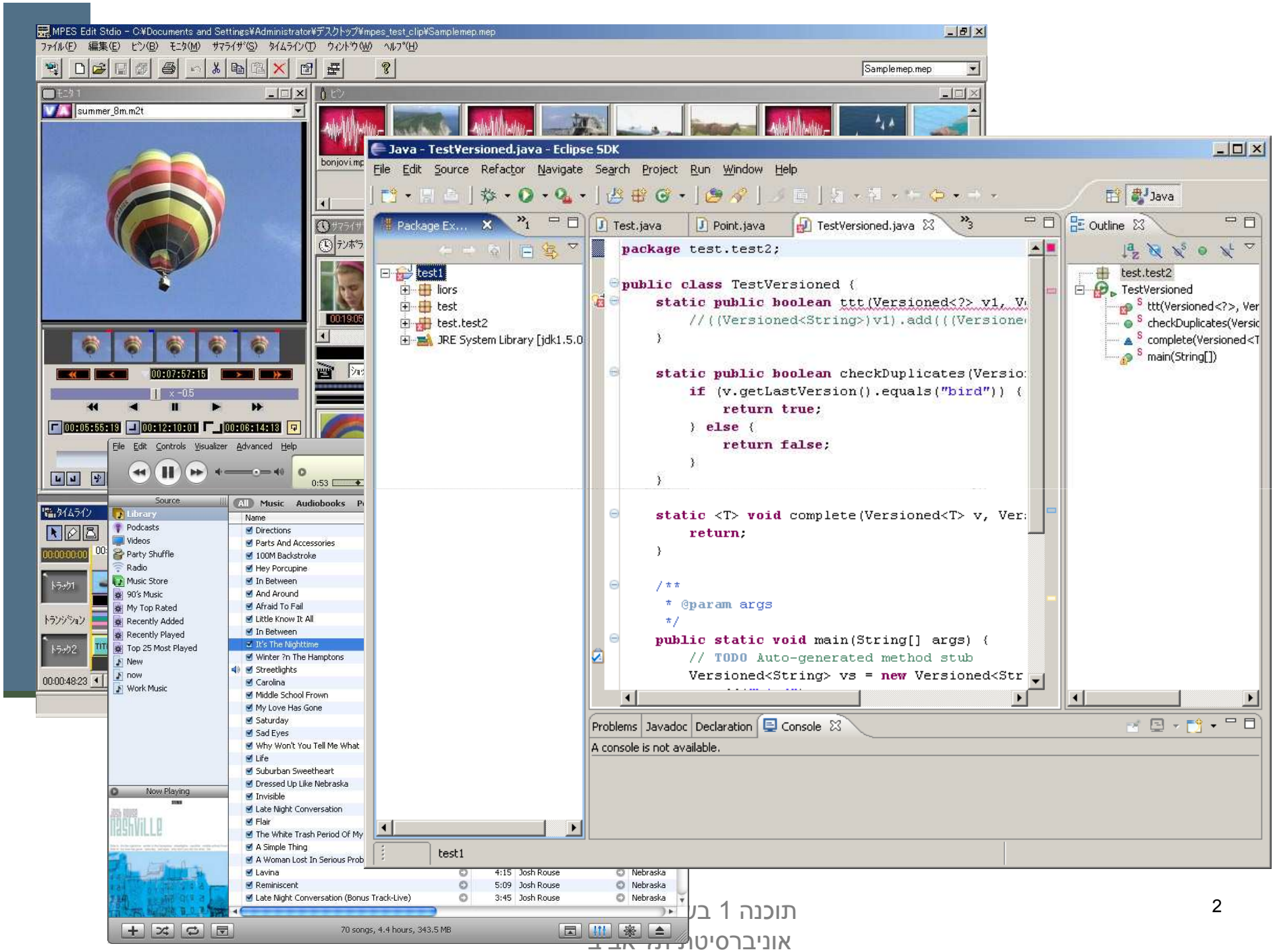
שיעור מספר 10: GUI

ליאור וולף

מתי שומרת

בית הספר למדעי המחשב

אוניברסיטת תל אביב


MPES Edit Stdio - C:¥Documents and Settings¥Administrator¥デスクトップ¥mpes_test_clip¥Samplemep.mep
Java - TestVersioned.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Ex...
test1
liors
test
test.test2
JRE System Library [jdk1.5.0
Test.java
Point.java
TestVersioned.java
package test.test2;
public class TestVersioned {
static public boolean ttt(Versioned<?> v1, V
//((Versioned<String>)v1).add(((Versione
}
static public boolean checkDuplicates(Versio
if (v.getLastVersion().equals("bird")) {
return true;
} else {
return false;
}
}
static <T> void complete(Versioned<T> v, Ver
return;
}
/**
* @param args
*/
public static void main(String[] args) {
// TODO Auto-generated method stub
Versioned<String> vs = new Versioned<Str
Outline
test.test2
TestVersioned
ttt(Versioned<?>, Ver
checkDuplicates(Versic
complete(Versioned<T
main(String[])
Problems Javadoc Declaration Console
A console is not available.
test1
summer_8m.m2t
Samplemep.mep
File Edit Controls Visualizer Advanced Help
Source
Library
Podcasts
Videos
Party Shuffle
Radio
Music Store
90's Music
My Top Rated
Recently Added
Recently Played
Top 25 Most Played
New
now
Work Music
Now Playing
All Music Audiobooks
Name
Directions
Parts And Accessories
100M Backstroke
Hey Porcupine
In Between
And Around
Afraid To Fail
Little Know It All
In Between
It's The Nighttime
Winter ?n The Hamptons
Streetlights
Carolina
Middle School Frown
My Love Has Gone
Saturday
Sad Eyes
Why Won't You Tell Me What
Life
Suburban Sweetheart
Dressed Up Like Nebraska
Invisible
Late Night Conversation
Flair
The White Trash Period Of My
A Simple Thing
A Woman Lost In Serious Prob
Lavina 4:15 Josh Rouse Nebraska
Reminiscent 5:09 Josh Rouse Nebraska
Late Night Conversation (Bonus Track-Live) 3:45 Josh Rouse Nebraska
70 songs, 4.4 hours, 343.5 MB

שלבי פיתוח מנשק גראפי

מהנדס מנשקי אנוש

- מפתח מנשק כך שיהיה מובן, יעיל, נעים
- מודדים את איכות המנשק על קב' משתמשים ומתקנים בהתאם
- **מחליט איך הממשק יתנהג**

מעצב גראפי

לנו אין תפקיד כ"כ גדול בתהליך

מממשים

- מחליט על סוגי האלמנטים שיופיעו על המסך, ועל סידורם
- **מחליט איך המנשק ייראה/יישמע**

תוכניתן

הנדסת מנשקי אנוש

- **אינטואיטיביות**; 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

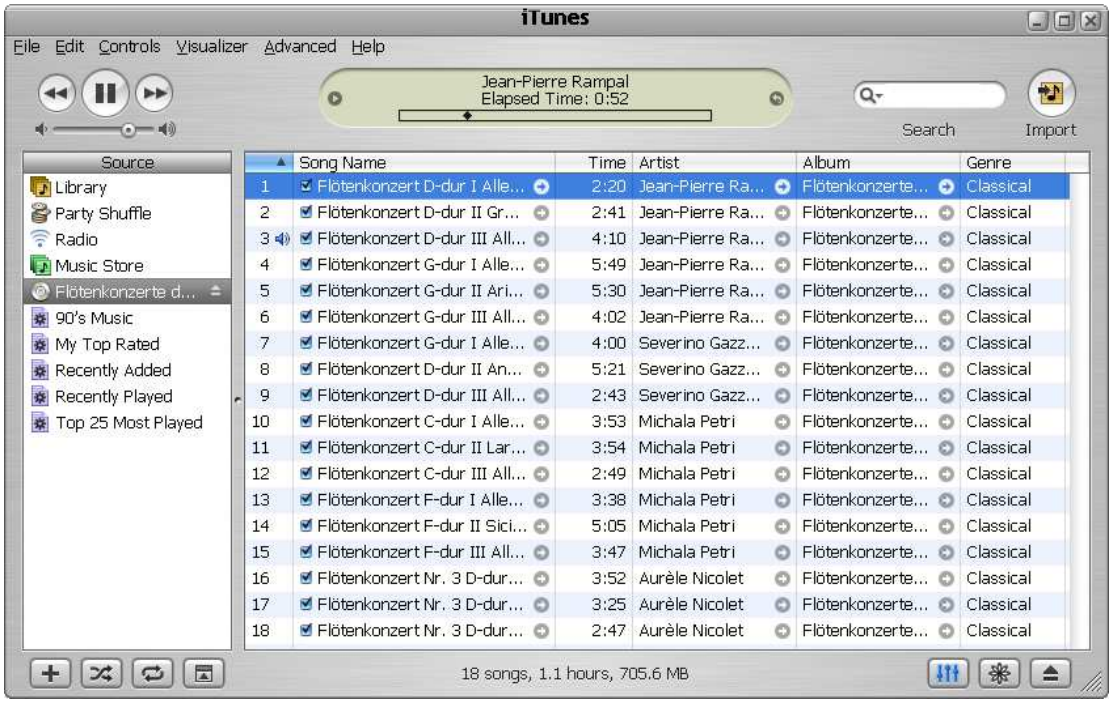
עיצוב מנשקים

- **קונסיסטנטיות**
- **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**

ועכשיו למימוש

שלושת הצירים של תוכנה גרפית

- **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 ("ארועים": הקלדה, הקלקה, גרירה)



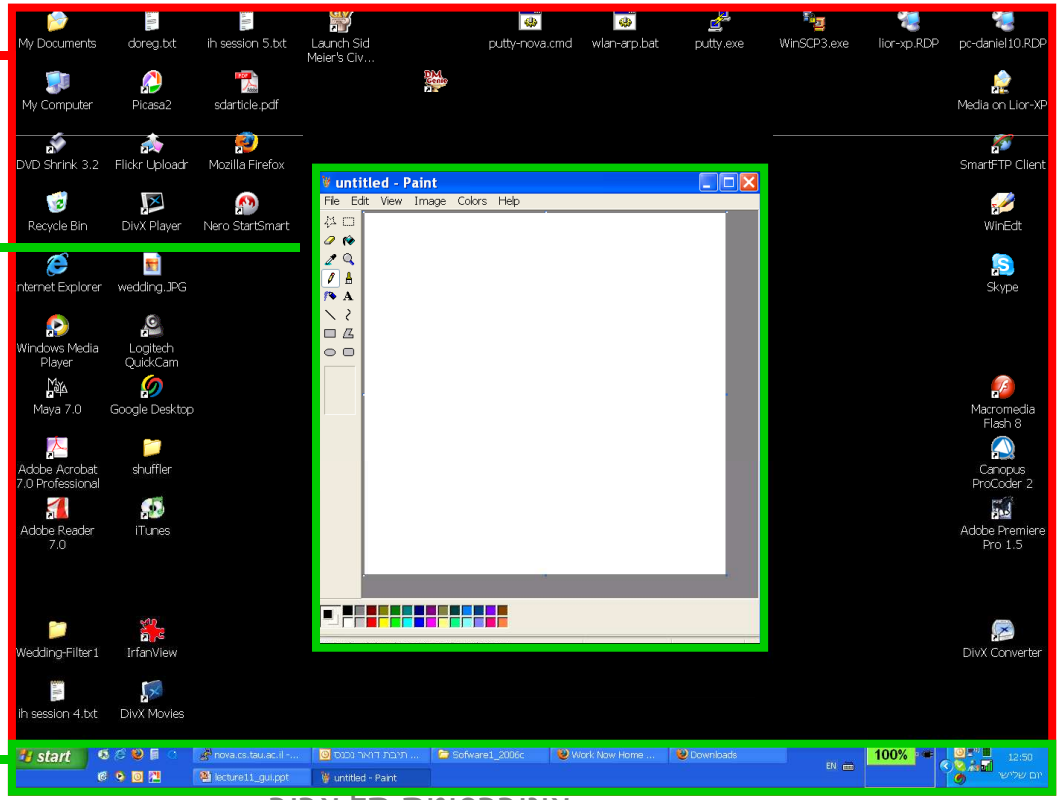
חלונות כמיכלים

- כל דבר הוא widget (חלון, כפתור, תפריט, משטח)
- החלונות מקיימים יחס הכלה (אחד לרבים)

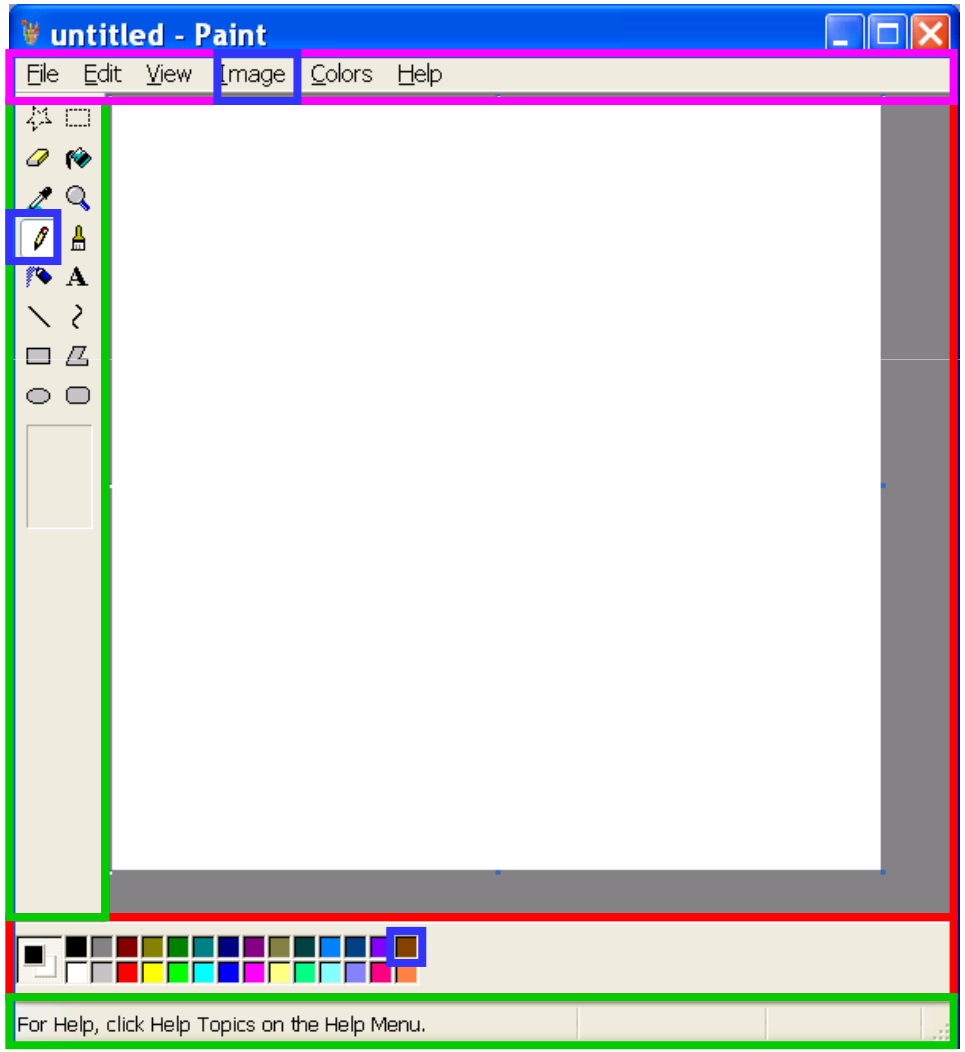
משטח עבודה (אב עליון)

אפליקציה (צייר)

שורת משימות



מודל החלונות (המשך)



"שלום עולם"

```
public class HelloSwt {

   public static void main(String[] args) {

        Display display = Display.getDefault();
        Shell shell = new Shell(display);
        shell.setText("Hello World!");
        shell.setSize(300, 100);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }
}
```

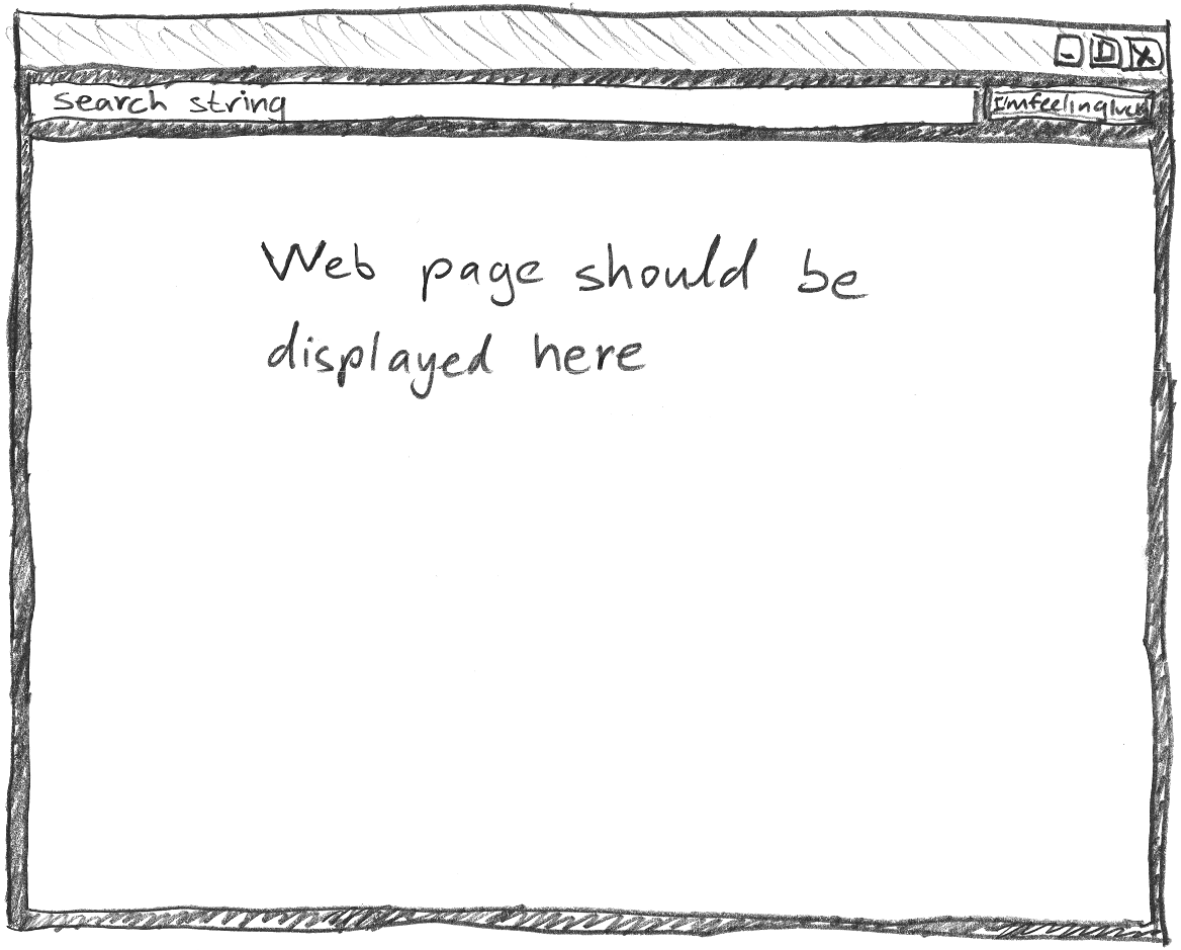
עצם ה Display מייצג את המסך

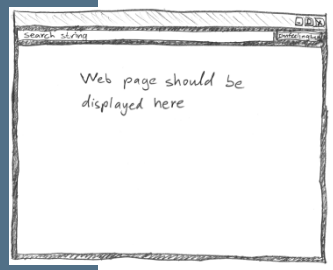
כל חלון מיוצג ע"י עצם מטיפוס Shell
ההורה של החלון הוא המסך

event loop

יש לשחרר משאבים בסיום העבודה בצורה מפורשת

דוגמה יותר מעניינת: דפדפן זעיר מגולגל





מה הדפדפן אמור לעשות

- המשתמשת תקליד מחרוזת חיפוש בשדה בצד שמאל למעלה
- לחיצה על הכפתור I'm feeling lucky מימין לשדה הטקסט תשלח את מחרוזת החיפוש ל-Google
- כאשר תתקבל התשובה, הדפדפן ישלוף מהתשובה של Google את הכתובת (URL) הראשונה ויטען אותה לרכיב הצגת ה-HTML בתחתית המסך, וכן ישנה את כותרת החלון כך שתציג את ה-URL
- נממש את הדפדפן בעזרת ספרייה למימוש מנשקים גראפיים בשם SWT (Standard Widget Toolkit) : http://www.eclipse.org/swt/
- ספריות אחרות למימוש מנשקים גראפיים בג'אווה הן AWT ו-Swing.

SWT לעומת ספריות גרפיות אחרות

- כחלק מהספרייה הסטנדרטית מכילה הפצת Java את החבילה **`java.awt`** המספקת שרותי GUI בסיסיים:
  - **A**bstract **W**indowing **T**oolkit
  - בעיית המכנה המשותף הנמוך ביותר
  - יעיל, יביל, מכוער
- בגרסאות מאוחרות של Java התווספה ספריית **`javax.swing`** המספקת שרותי GUI מתקדמים:
  - JFC/Swing
  - Pluggable Look & Feel
  - עשיר, איטי, כבד, מכוער (שנוי במחלוקת)
- ספריית SWT של IBM מנסה לרקוד על שתי החתונות
  - גם יפה גם אופה
  - המנשק הגרפי של Eclipse מבוסס SWT
  - אינו סטנדרטי (יש להוריד כ zip נפרד)

מבנה המימוש

```
public class GoogleBrowser {

  private Shell   shell   = null;
  private Button  button  = null;
  private Text    text    = null;
  private Browser browser = null;

  /* call createShell and run event loop */
  public static void main(String[] args) {...}

  /* create the GUI */
  private void createShell() {...}

  /* send query to Google and return the first URL */
  private static String search(String q) {...}

}
```

Widgets (אביזרים)

- השדות text, button, browser, ו-shell יתייחסו לרכיבי המנשק 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

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();
}
```

יצירת המנשק הגראפי

```
/* create the GUI */
private void createShell() {

    shell = new Shell();

    shell.setText("Browser Example");

    //layout manager: a grid with 2 unequal columns
    shell.setLayout(new GridLayout(2, false));

    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
```

פריסת רכיבי המנשק במעטפת

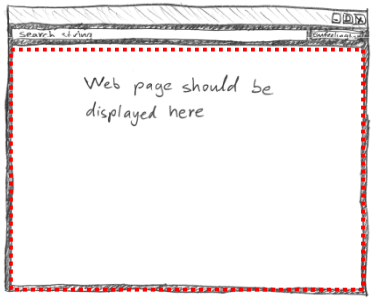
- מעטפות הם רכיבי מנשק שמיועדים להכיל רכיבי מנשק
- את הרכיבים המוכלים צריך למקם; רצוי לא למקם אותם באופן אבסולוטי (ערכי x ו-y בקואורדינטות של הרכיב המכיל)
- מנהלי פריסה (layout mangers) מחשבים את הפריסה על פי הוראות פריסה שמצורפות לכל רכיב מוכל
- GridLayout הוא מנהל פריסה שממקם רכיבים בתאים של טבלה דו-מימדית;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

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 (סיבית 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

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
```

מחלקות פנימיות - תזכורת

- כבר ראינו בקורס מחלקות פנימיות - מחלקות אשר מוגדרות בתוך התחום של מחלקות אחרות
- ראינו כי ניתן להגדיר מחלקות מקומיות אפילו בתוך **שרות** של מחלקה אחרת
- במקרים כאלה, יש למחלקה הפנימית תפקיד מצומצם ומוגבל (אחרת היינו מגדירים אותה מחוץ לשרות)
- פעמים רבות המחלקה הפנימית היא מחלקה עם מופע אחד בלבד
- כדי לטפל במקרים כאלה Java מספקת תחביר מיוחד אשר חוסך את הצורך לתת שם למחלקה – מחלקות אלו נקראות מחלקות חסרות שם (anonymous inner classes):
  - הגדרת המחלקה **אינה כוללת שם** למחלקה
  - ההגדרה מתבצעת **תוך כדי יצירת המופע** של אותה המחלקה

מחלקה אנונימית - דוגמא

```
public class Test {

    public static final double lineComissions = 1.1;

    public static void main(String[] args) {

        BankAccount b = new BankAccount(){
            public double balance(){
                balance -= Test.lineComissions;
                return super.balance();
            }
        };

        b.deposite(100);
        System.out.println(b.balance());
        System.out.println(b.balance());
    }

}
```

הגדרת מופע של מחלקה פנימית אנונימית שיורשת מ- **BankAccount** ודורסת את **balance()**

הפרוצדורה שהכפתור מפעיל

- נוסיף לכפתור הדפדפן שלנו **מאזין** – מחלקה אשר מקשיבה להקלקות על הכפתור
- קיימות מחלקות ברירת מחדל אשר "**מודעות**" להקלקות **אך לא עושות דבר** הנקראות מתאמים (Adapters)
- כדי להגדיר את הפעולה שיש לבצע נירש (אנונימית) מכזה Adapter ונדרוס את השרות **widgetSelected**

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

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

שלוש גישות לטיפול באירועים

- בעזרת טיפוסים ספציפיים לסוג האירוע:
  - למשל, KeyListener הוא מנשק שמגדיר שני שירותים, KeyPressed ו-KeyReleased, שכל אחד מהם מקבל את הדיווח על האירוע בעזרת עצם מטיפוס KeyEvent
- ללא טיפוסים שמתאימים לאירועים ספיציפיים:
  - האירוע מפעיל עצם מטיפוס Listener שמממש שירות בודד, handleEvent, והאירוע מדווח בעזרת טיפוס Event
- יש ספריות של מנשקים גראפיים, למשל AWT, שמשתמשות בירושה:
  - המחלקה שמייצגת את המנשק שלנו מרחיבה את Frame (מקביל ל-Shell) ודורסת את השירות handleEvent, ש-Frame קוראת לו לטיפול באירועים

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);
       }
   });
```

- זהו מקרה פרטי של תבנית העיצוב Observer Design Pattern
- המאזין (בתפקיד ה Observer) **נרשם** אצל הכפתור (בתפקיד ה- Subject)

Listener לעומת Adapter

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

כמעט סיימנו

- נותרו רק שתי שורות שלא ראינו ב-**createShell**:

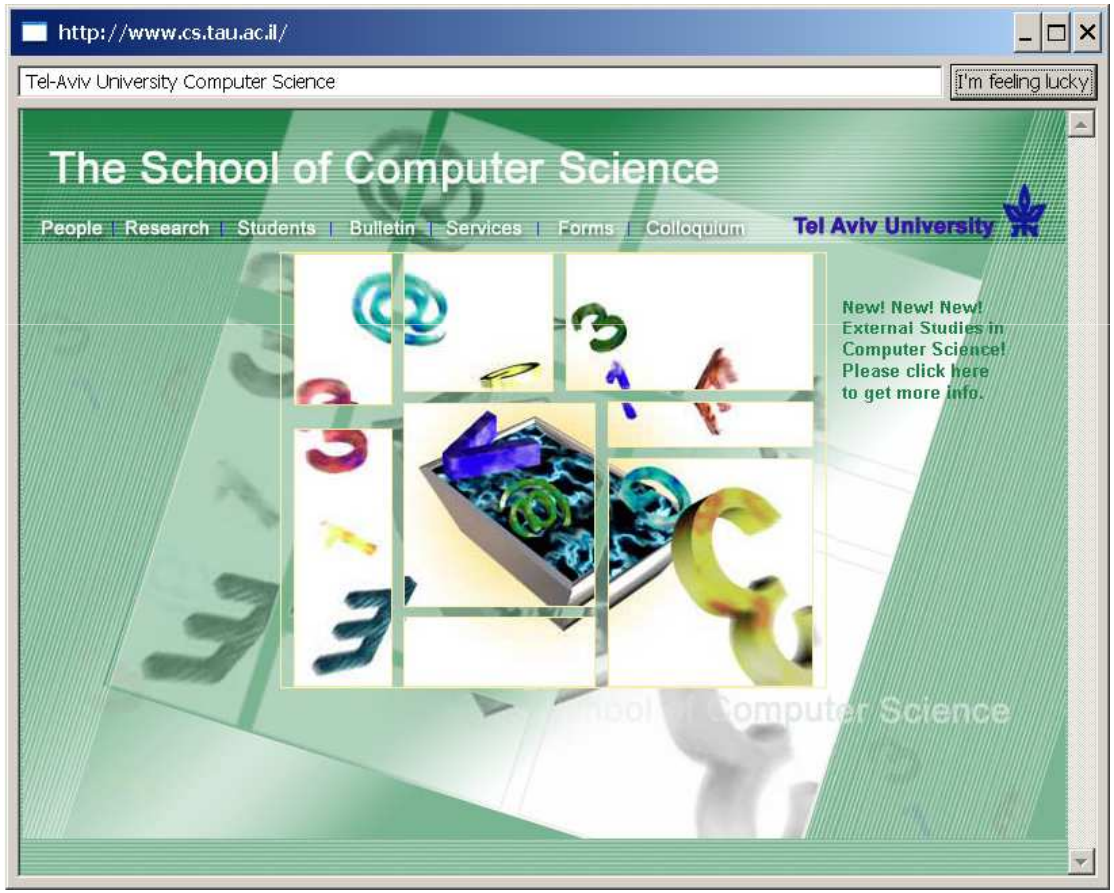
```
private void createShell() {
    ...
    button.addSelectionListener(...);

    //causes the layout manager to lay out the shell
    shell.pack();

    //opens the shell on the screen
    shell.open();
}
```

- והפרוצדורה שמחפשת במנוע החיפוש Google ומחזירה את ה-URL של התשובה הראשונה

חיפוש ב-Google

```
private static String search(String q) {
    GoogleSearch s = new GoogleSearch();
    s.setKey("My Secret Key");
    s.setProxyHost("proxy.tau.ac.il");
    s.setProxyPort(8080);
    s.setQueryString(q);
    s.setStartResult(0);

    GoogleSearchResult r = null;
    try {
       r = s.doSearch();
    } catch (GoogleSearchFault e) {
       e.printStackTrace();
    }
    return (r.getResultElements())[0].getURL();

}
```

והתוצאה


http://www.cs.tau.ac.il/
Tel-Aviv University Computer Science
I'm feeling lucky
The School of Computer Science
People | Research | Students | Bulletin | Services | Forms | Colloquium
Tel Aviv University
New! New! New!
External Studies in
Computer Science!
Please click here
to get more info.

סיכום ביניים

- ראינו את המחלקות שמייצגות רכיבי מנשק גראפי
- ראינו איך נרשמים להגיב על אירוע כגון לחיצה על כפתור
- ראינו כיצד מגדירים את הפריסה של הרכיבים על המסך
- האם המנשק 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 המנשק מורכבות מאוד:
  - צריך ספר או מדריך מקוון (קישורים בסוף המצגת)
  - צריך להתאמן
  - רצוי להשתמש במנגנון עריכה ייעודי למנשקים גראפיים (GUI Builder)

שחרור משאבים

- חלק מהעצמים שמרכיבים את המנשק 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 את המשאבים הללו, 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)

Look and Feel

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

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)

תחושת המנשק בפלטפורמות שונות

- **בחלונות** 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
- בתוכניות רבות **בלינוקס** מספיק **לסמן** קטע בשביל להעתיק אותו, **והכפתור האמצעי** בעכבר משמש להדבקה
- 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, ו-MOD3, כאשר 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

Widget ויורשיו

http://www.eclipse.org/swt/widgets

- Widget
  - Caret
  - Control
    - Button
    - Label
    - Link
    - ProgressBar
    - Sash
    - Scale
    - Scrollable
      - Composite
      - List
      - Text
    - Slider
  - DragSource
  - DropTarget
  - Item
  - Menu
  - ScrollBar
  - Tracker
  - Tray

מחלקת על מופשטת

כפתור המייצג: כפתור לחיץ, כפתור רדיו, checkbox, וכו'

תווית לטקסט או תמונה

איפה נמצאת המחלקה Shell ?

org.eclipse.swt.widgets

Class Shell

```
java.lang.Object
  └ org.eclipse.swt.widgets.Widget
      └ org.eclipse.swt.widgets.Control
          └ org.eclipse.swt.widgets.Scrollable
              └ org.eclipse.swt.widgets.Composite
                  └ org.eclipse.swt.widgets.Canvas
                      └ org.eclipse.swt.widgets.Decorations
                          └ org.eclipse.swt.widgets.Shell
```

פריסה נכונה

- פריסה נכונה של רכיבים היא אחד האתגרים המשמעותיים בפיתוח מנשק גראפי
- התוכנית צריכה להבטיח עד כמה שאפשר שהמנשק 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
  - מיקום רכיבים על המסך בשיעורים מוחלטים אינו רגיש למגוון האפשרויות
  - מיכלי GUI (containers, composites) מבצעים **האצלה** של אסטרטגיית הסידור למחלקה יעודית לכך
- אלגוריתמי פריסה מתוחכמים עבור מיכלים, כגון GridLayout, מסייעים, אבל צריך להבין כיצד מתבצעים חישובי הפריסה וכיצד להשפיע עליהם

דוגמא – שיוך מנהל פריסה למיכל

שיוך מנהל הפריסה (FillLayout) לחלון

```
shell.setLayout(new FillLayout());
Label lbl = new Label(shell, SWT.CENTER);
lbl.setText("Hello world!");
shell.pack();
shell.open();
```

הגדרת הורה – במובן של מיכל מכיל

הצג את החלון על המסך

התאם את גודל הרכיבים לפי אלגוריתם הסידור (אולי התווספו כמה רכיבים)

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?"

פריסה מלמטה למעלה

- כל רכיב צריך לדעת באיזה גודל הוא רוצה להיות (שם השירות ב-SWT הוא computeSize, בספריות אחרות preferredSize)
- יש ספריות שבהן כל רכיב צריך לדעת מה גודלו המינימאלי (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 הויזואליים 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

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 בהינתן רוחב מסוים ולהיפך, ולא רק מה הגודל הרצוי ללא שום אילוץ
- 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

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 לחשבו שוב ושוב

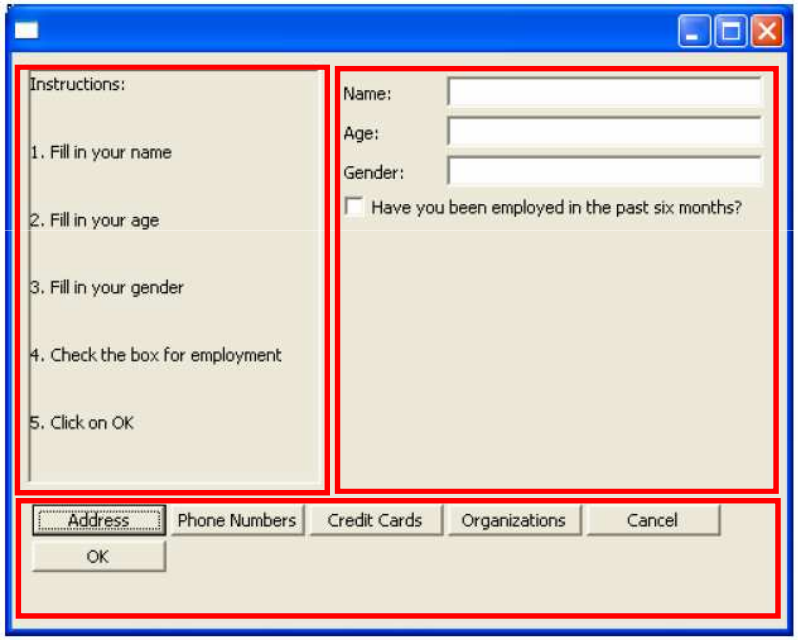
אריזה הדוקה

- השירות pack מחשב את גודלו הרצוי של רכיב או מיכל וקובע את גודלו לגודל זה; המיכל נארז 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

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

הרכבה של Composites



- כדי לבנות בצורה מודולרית מסכים מורכבים (ולפתח כל איזור בנפרד) – רצוי להשתמש במחלקה Composite (מקבילה למחלקה J/Panel ב- Swing/AWT)
- בדוגמא שלפנינו ה Shell מכיל 3 Composites שונים, כל אחד מהם מנוהל ע"י מנהל פריסה משלו

```
shell.setLayout(new FormLayout());

//Fill Layout panel
Composite fillComp = new Composite(shell, SWT.BORDER);
fillComp.setLayout(new FillLayout(SWT.VERTICAL));
Label label0 = new Label(fillComp, SWT.NONE);
label0.setText("Instructions:");
...

//Row Layout panel
Composite rowComp = new Composite(shell, SWT.NONE);
RowLayout rowLayout = new RowLayout();
rowLayout.pack = false;
rowComp.setLayout(rowLayout);
Button b1 = new Button(rowComp, SWT.PUSH);
b1.setText("Address");
...

//Grid Layout panel
Composite gridComp = new Composite(shell, SWT.NONE);
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 2;
gridComp.setLayout(gridLayout);
Label label11 = new Label(gridComp, SWT.NONE);
label11.setText("Name:");
```

משאבים יחודיים

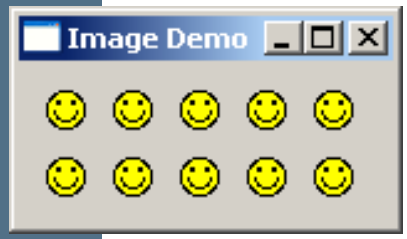
- כמה משאבים שימושיים היורשים מ:
  `org.eclipse.swt.graphics.Resource`
  - צבעים (Color)
  - גופנים (Font)
  - סמנים (Cursor)
  - תמונות (Image)
- יכולים להיות משותפים לכמה רכיבים
- אינם משתחררים אוטומטית ע"י dispose ולכן יש לשחרר אותם מפורשות

דוגמא – צבעים וגופנים

```
Font times16BI = new Font(null, "Times New Roman",
                          16, SWT.BOLD | SWT.ITALIC);
Color deepPurple = new Color(null, 120, 45, 134);

Label lbl = new Label(shell, SWT.CENTER);
lbl.setFont(times16BI);
lbl.setBackground(deepPurple);
lbl.setForeground(
         display.getSystemColor(SWT.COLOR_YELLOW));
lbl.setText("Colors and Fonts");
...

times16BI.dispose();
deepPurple.dispose();
display.dispose();
```

תמונות

- תמונות הן עצמים גדולים מאוד (יחסית לרכיבי GUI אחרים), המקושרים לייצוג התמונה במערכת ההפעלה
- יש לשתף בין תמונות ככל הניתן (לא ליצור שני עצמים לייצוג אותה התמונה)
- המחלקה **`ImageDescription`** היא עצם ב Java המתאר תמונה
  - כמו איזו מחלקה אחרת שראינו?
  - אין צורך לבצע **`dispose`** על **`ImageDescription`**
- השרות **`createreImage`** מייצר תמונה ע"פ **`ImageDescription`**
  - באחריות המתכנתת לבצע **`dispose`** על התמונה הנוצרת



דוגמא

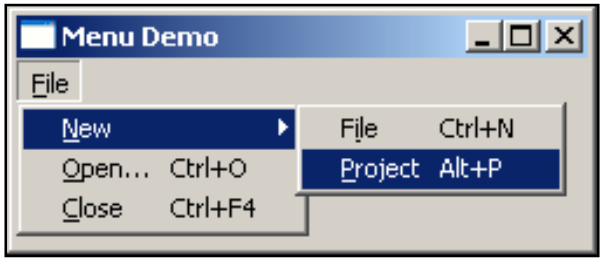
```
ImageDescriptor smileyDesc =
    ImageDescriptor.createFromFile(ImageDemo.class,
                                   "Smiley.gif");
 Image smiley = smileyDesc.createImage();

 for (int i = 0; i < 10; i++) {
     Label lbl = new Label(shell, SWT.CENTER);
     lbl.setImage(smiley);
 }

 ...

 smiley.dispose();
 display.dispose();
```

createFromFile מקבלת שם קובץ
יחסי למיקום המחלקה שהועברה

שימוש בתפריטים

- בעזרת המחלקות **Menu** ו- **MenuItem**
- ניתן בקלות לקונן תפריטים ע"י הוספת תפריט לתפריט
- טיפול בארועים בעזרת הוספת **MenuListener** או **SelectionListeners**
- ניתן להשתמש ב- & כדי לציין את מקש קיצור הדרך לתפריט (יסומן בתפריט כ- _)
- יש להגדיר את מקש קיצור הדרך מפורשות ע"י **setAccelerator**

דוגמא




```
Menu top = new Menu(shell, SWT.BAR);

MenuItem file = new MenuItem(top, SWT.CASCADE);
file.setText("&File");
Menu fileMenu = new Menu(shell, SWT.DROP_DOWN);
file.setMenu(fileMenu);

MenuItem newItem = new MenuItem(fileMenu, SWT.CASCADE);
newItem.setText("&New");

Menu newMenu = new Menu(shell, SWT.DROP_DOWN);
newItem.setMenu(newMenu);

MenuItem new_file = new MenuItem(newMenu, SWT.NULL);
new_file.setText("F&ile\tCtrl+N");
new_file.setAccelerator(SWT.CTRL + 'N');

MenuItem new_project = new MenuItem(newMenu, SWT.NULL);
new_project.setText("&Project\tAlt+P");
new_project.setAccelerator(SWT.ALT + 'P');

MenuItem open = new MenuItem(fileMenu, SWT.NULL);
open.setText("&Open...\tCtrl+O");
open.setAccelerator(SWT.CTRL + 'O');

MenuItem close = new MenuItem(fileMenu, SWT.NULL);
close.setText("&Close\tCtrl+F4");
close.setAccelerator(SWT.CTRL + SWT.F4);

shell.setMenuBar(top);
```

עשה זאת בעצמך

- ניתן לצייר על רכיבי GUI (להבדיל מלהוסיף רכיבים מוכנים)
- כדי שהציור ישמור על עיקביותו גם לאחר ארועי חשיפה (שינוי גודל החלון, הסתרת/מזעור החלון ע"י חלונות אחרים) יש לדאוג לציור מחדש לאחר כל ארוע כזה
- לשם כך נכתוב את פונקצית הציור כשגרת הטיפול **בארועי ציור**
- השגרה מקבלת כארגומנט ארוע ציור **`PaintEvent`** אשר ניתן לחלץ ממנו הפנייה להקשר הגרפי (**`GC – Graphics Context`**)

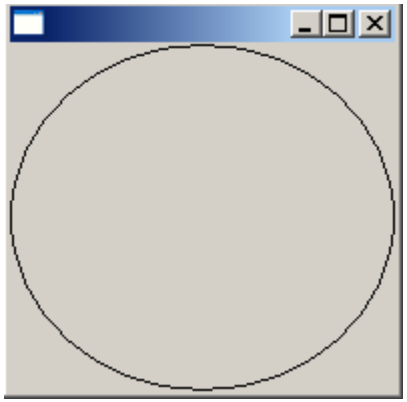
עשה זאת בעצמך

- נצייר על GC ע"י שימוש בשרות drawXXX הכולל את (רשימה חלקית):

```
void drawArc(int x, int y, int width, int height,
                                int startAngle, int arcAngle)
void drawFocus(int x, int y, int width, int height)
void drawImage(Image image, int x, int y)
void drawLine(int x1, int y1, int x2, int y2)
void drawOval(int x, int y, int width, int height)
void drawPath(Path path)
void drawPoint(int x, int y)
void drawPolygon(int[] pointArray)
void drawRectangle(int x, int y, int width, int height)
void drawRoundRectangle(int x, int y, int width,
                    int height, int arcWidth, int arcHeight)
void drawString(String string, int x, int y)
void drawText(String string, int x, int y)
```

"צייר לי עיגול"

```
final Display display = new Display();
final Shell shell = new Shell(display);

shell.addPaintListener(new PaintListener() {
        public void paintControl(PaintEvent event) {
            Rectangle rect = shell.getClientArea();
            event.gc.drawOval(0, 0, rect.width - 1,
              rect.height - 1);
        }
    });

shell.setBounds(10, 10, 200, 200);
shell.open();
```

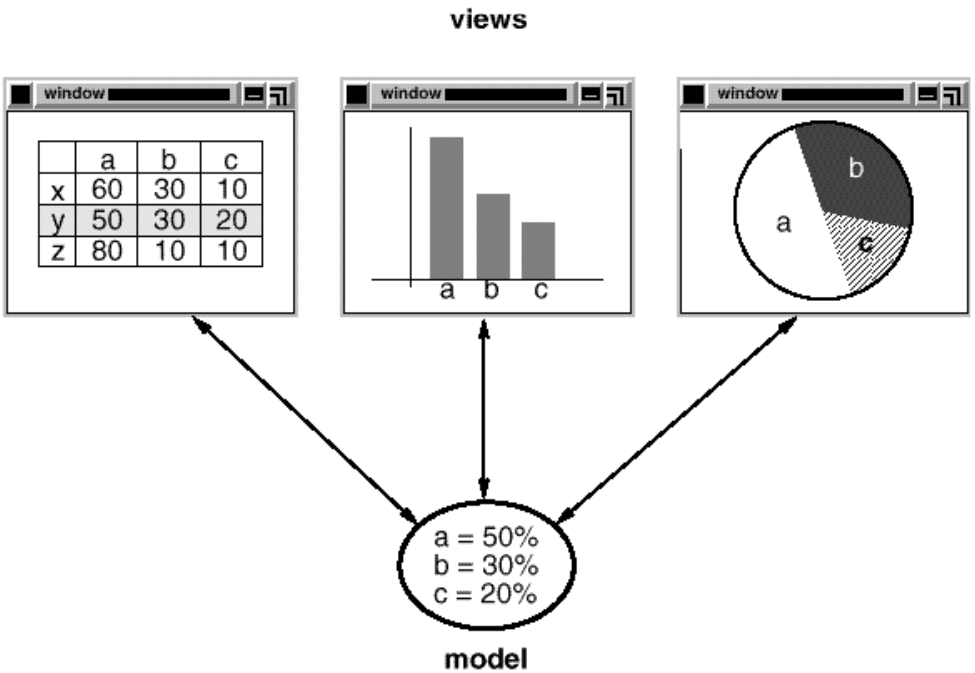


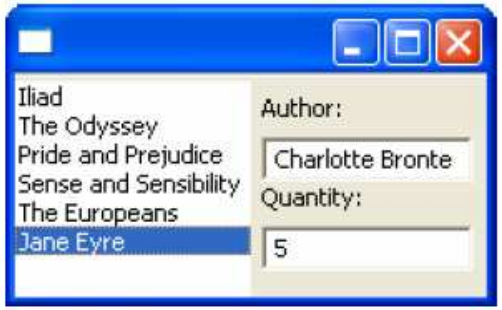
2 חלופות

- איך נתכנן משחק שח גרפי?
  - עשה זאת בעצמך:
    - ציור של משבצות שחור-לבן
    - לכידה של ארועי לחיצה על העכבר
  - שימוש ב widgets:
    - בניית סריג של כפתורים ריבועיים בצבעי שחור ולבן לסרוגין
    - לכידה של ארועי בחירת כפתור
- מה היתרונות והחסרונות של כל אחת מהגישות?

הפרדה בין מודל והצגה

- עקרון מרכזי בבניין יישומים מבוססי גרפיקה הוא ההפרדה בין המודל וההצגה (model/view separation)
- המודל (הנתונים והלוגיקה של התוכנית) אמור להיות אדיש לשינויים בהצגה (ואולי לאפשר ריבוי הצגות במקביל)





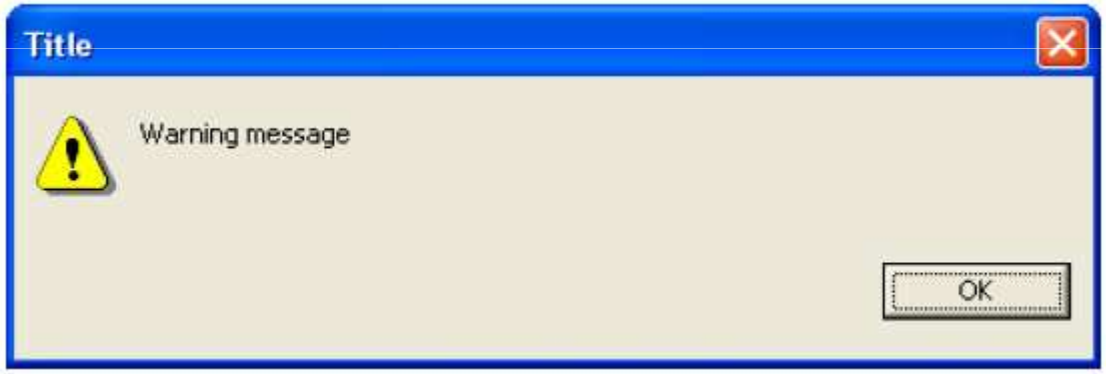
JFace Viewers

- החבילה JFace מציעה מגוון מחלקות המציעות שרותי GUI מתקדמים הכתובים מעל (בעזרת) הספריה SWT
- אחת המשפחות בחבילה מכילה הצגות למבני נתונים שימושיים כגון: **CheckboxTableViewer, CheckboxTreeViewer, ListViewer, TableTreeViewer, TableViewer, TreeViewer**
- למשל, אם ברצוננו להציג למשתמש רשימה של ספרים נרצה לקשור בין רשימת הספרים (עצמים מטיפוס Book) ובין רכיב הרשימה הויזואלית
  - לצורך כך יש להגדיר לרשימת הספרים: **LabelProvider** ו- **StructuredContentProvider** (Renderer בספרית Swing)

JFace Dialogs

- בחבילה JFace ניתן גם למצוא מגוון תיבות דו-שיח לתקשורת עם המשתמש:

```
MessageDialog.openWarning(shell, "Title", "Warning message");
```



JFace Dialogs

```
String[] buttonText =
            new String[] { "Button1", "Button2", "Button3" };

MessageDialog messageBox; =
             new MessageDialog(shell, "Title", null, "Message",
                                MessageDialog.ERROR, buttonText, 1);
messageBox.open();
```

ניתן להגדיר מספר סוגי תיבות דו-שיח:

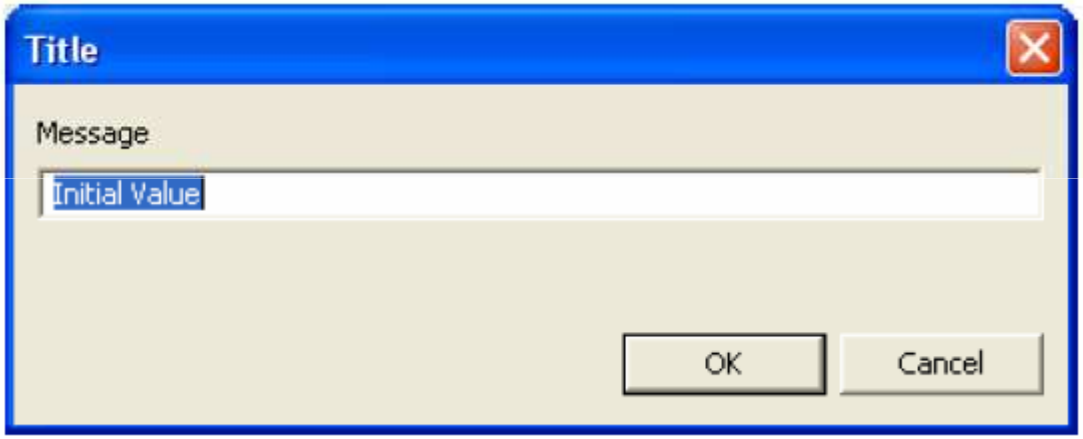
MessageDialog.NONE, MessageDialog.ERROR, MessageDialog.INFORMATION, MessageDialog.QUESTION, MessageDialog.WARNING

קריאת בחירת המשתמש ע"י:

```
messageBox.getReturnCode();
```

JFace Dialogs

```
InputDialog inputBox =
  new InputDialog(shell, "Title", "Message", "Initial Value", null);

inputBox.open();
```



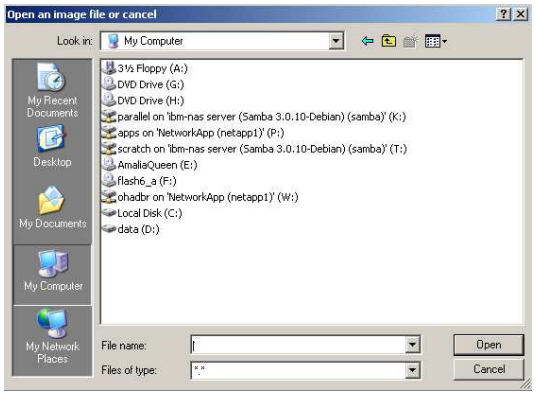
קריאת קלט משתמש ע"י:

```
inputBox.getReturnCode();
inputBox.getValue();
```

JFace Dialogs

```
ColorDialog d = new ColorDialog(shell);
RGB selection = d.open();
```


```
FileDialog dialog = new FileDialog (shell, SWT.OPEN);
dialog.setText ("Open an image file or cancel");
String string = dialog.open ();
```



סיכום מנשקים גרפיים

- דע/י את מקומך
- שלושה מנגנונים כמעט אורתוגונאליים: ירושה, הכלה, אירועים
- פגמים במנשק גראפי נובעים במקרים רבים או מפריסה לא נכונה של רכיבים במיכל, או מחוסר תגובה או תגובה לא מספיקה 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

מקורות מקוונים

- באתר Eclipse: http://www.eclipse.org/swt/
  - מקטעי קוד:

http://www.eclipse.org/swt/widgets

  - דוגמאות לצפייה בתוך eclipse:

http://www.eclipse.org/swt/examples.php

- באתר אוניברסיטת מניטובה (קנדה):

http://www.cs.umanitoba.ca/~eclipse/