תוכנה 1 בשפת Java

שיעור מספר 10:"צייר לי כבשה" (GUI)

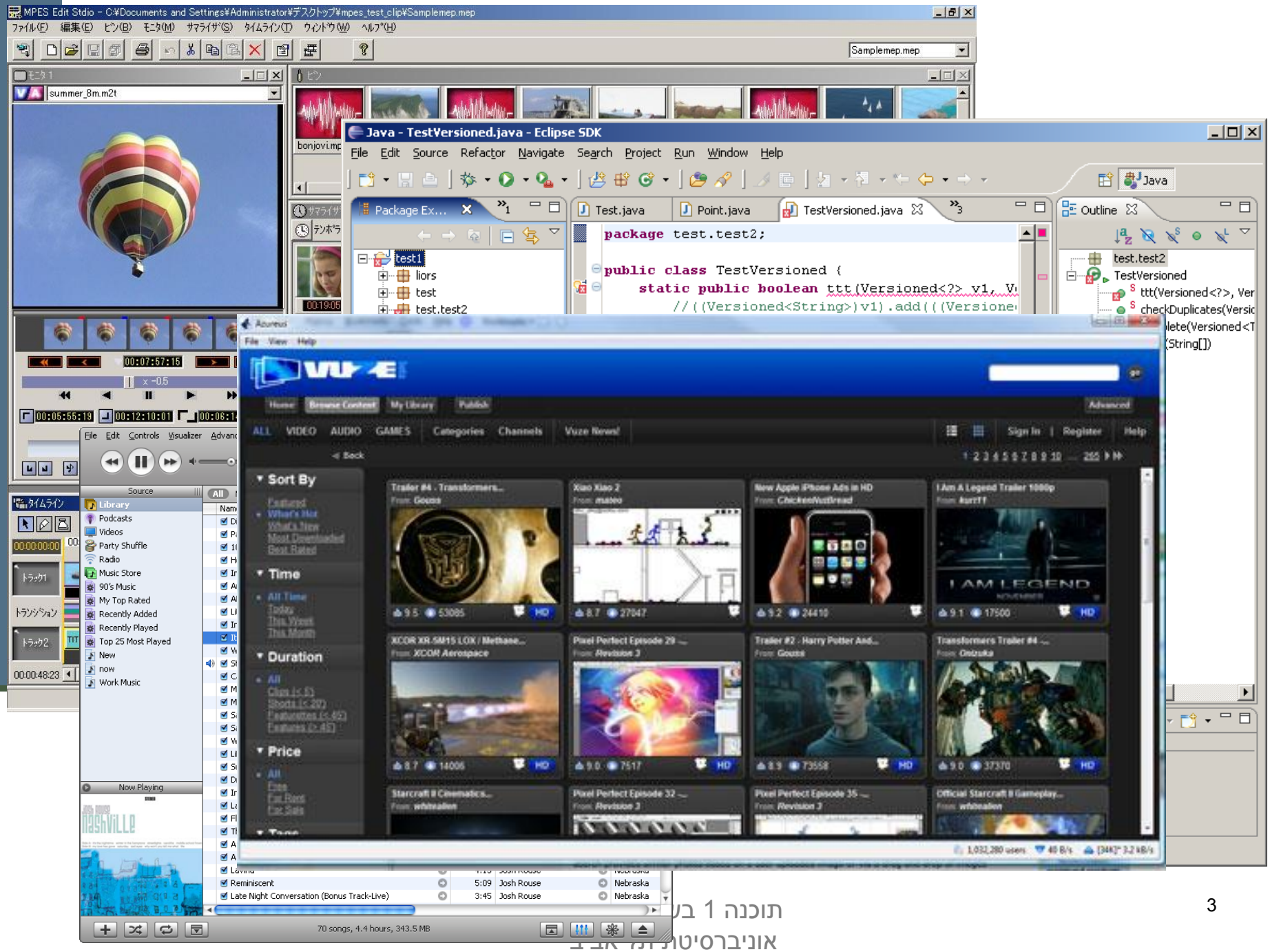
שחר מעוז

בית הספר למדעי המחשב

אוניברסיטת תל אביב

מנשק משתמש גרפי (GUI)

- משפר את האינטראקציה בין המשתמש האנושי ובין המחשב:
  - מחשב => אדם:
    - ניתן להוסיף מימד ויזואלי למידע
    - גופנים, צבעים, תמונות
    - ארגון המידע על המסך בצורה נוחה ואינטואיטיבית
  - אדם => מחשב:
    - המשתמש יכול **לבחור** ולא רק להקליד
    - עכבר ומגע בנוסף למקלדת
- אבסטרקציות חדשות של אינטראקציה:
  - תפריטים, כפתורים, גרירה, הצבעה


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...
Test.java
Point.java
TestVersioned.java
Outline
package test.test2;
public class TestVersioned {
static public boolean ttt(Versioned<?> v1, V
//((Versioned<String>)v1).add(((Versione
test1
liors
test
test.test2
test.test2
TestVersioned
ttt(Versioned<?>, Ver
checkDuplicates(Versic
delete(Versioned<T
(String[])
Sort By
Time
Duration
Price
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
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

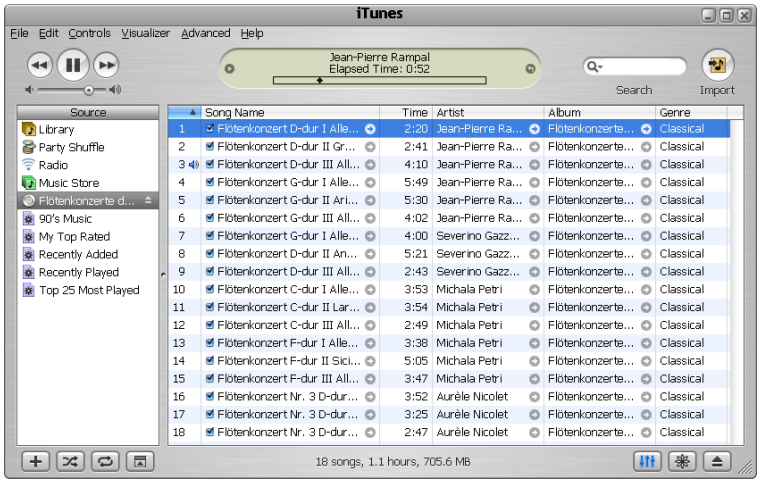
- **התאמה לתכיפות השימוש וללימוד התוכנה**; האם משתמשים בה באופן חד פעמי (אשף לכתיבת צוואות) או יומיומי (דואל); גם משתמש יומיומי בתוכנה היה פעם מתחיל חסר ניסיון

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**

שלושת הצירים של תוכנה גרפית

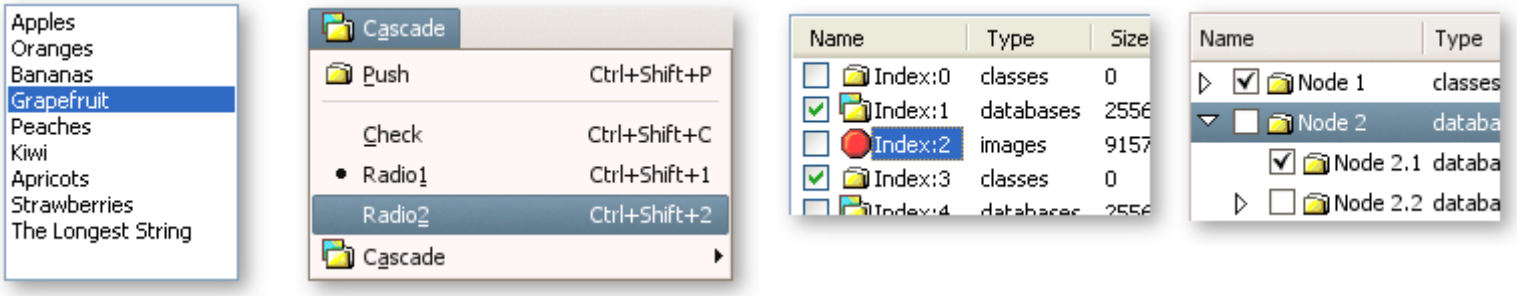
- **אלמנטים** מסוגים שונים על המסך (היררכיה של טיפוסים)
- **הארגון** הדו־מימדי של האלמנטים, בדרך כלל בעזרת מיכלים ומנהלי פריסה
- **ההתנהגות** הדינמית של האלמנטים בתגובה לפעולות של המשתמש/ת ("ארועים": הקלדה, הקלקה, גרירה)



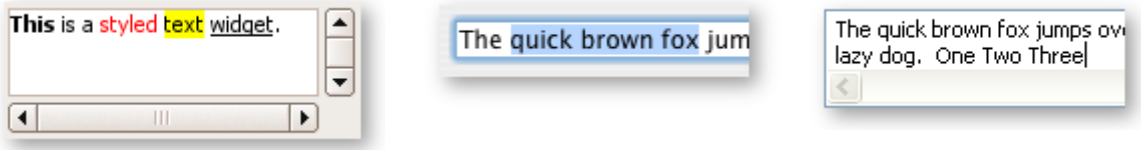
האלמנטים (widgets)

- כפתורים:

- תפריטים, טבלאות, רשימות, עצים:



- תיבות טקסט:



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

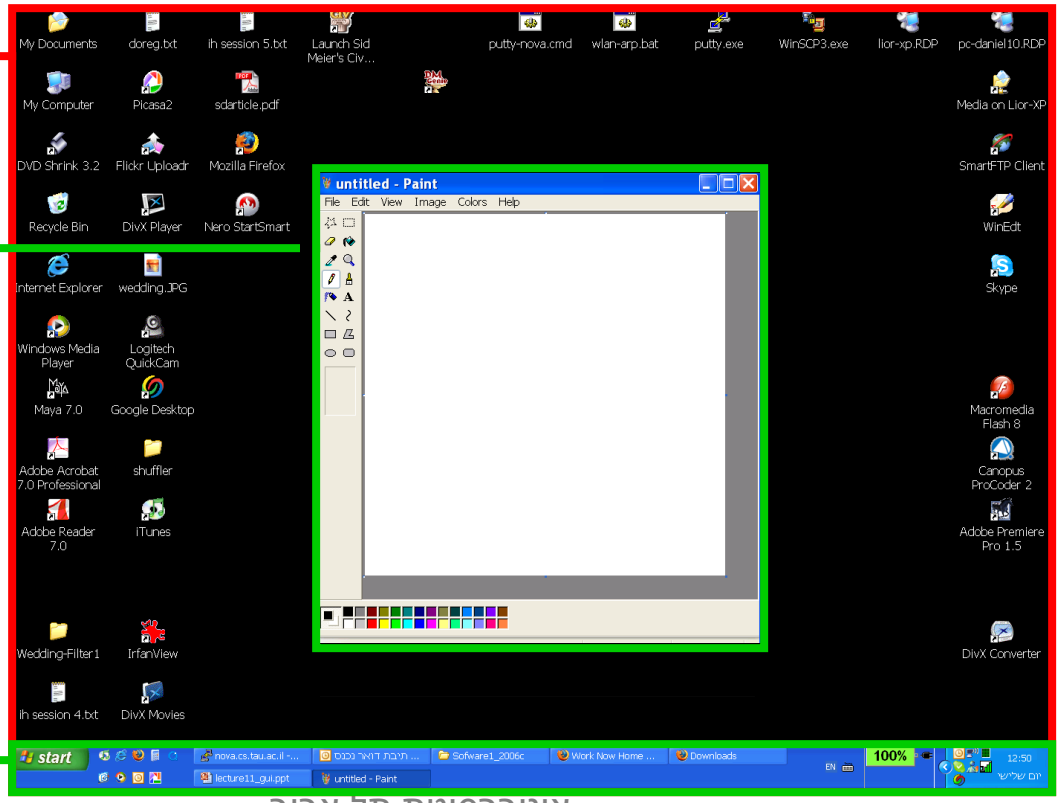
ארגון האלמנטים - חלונות כמיכלים

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

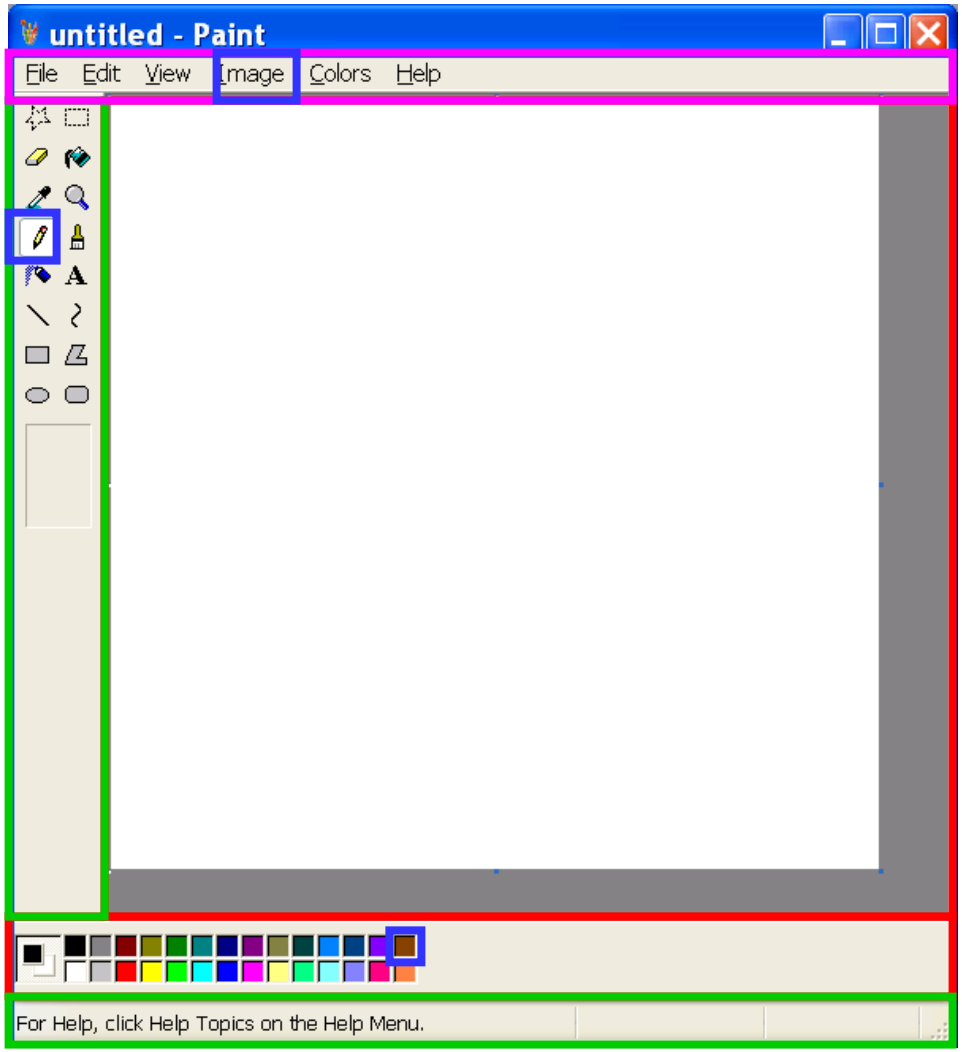
משטח עבודה (אב עליון)

אפליקציה (צייר)

שורת משימות



מודל החלונות (המשך)



ארגון האלמנטים

- כאשר בונים מסך מגדירים עבורו מה יהיה אופן הסידור של איבריו הפנימיים בעזרת יצירת אובייקט Layout מתאים:
  - שורה, טור, טבלה, ערימה
- הבנייה רקורסיבית: חלון (מיכל) עשוי להכיל תת חלונות (מיכלים) אשר לכל אחד מהם פריסה פנימית שונה, כאשר החלון העוטף אחראי רק על הפריסה של תת החלונות
- אובייקט ה Layout גם אחראי להתאמת גודל האיברים כאשר גודל החלון משתנה
  - ניתן לשנות פרמטרים רבים באלגוריתם הסידור
- פרטים בהמשך...

http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html

התנהגות

- כאשר המשתמש עושה משהו ("מחולל ארוע": בחירת כפתור, הקלדת מקלדת, גרירת עכבר, וכו') נקראות פונקציות שונות של האפליקציה
  - למשל: כאשר המשתמש בוחר מהתפריט File => save נרצה שתקרא הפונקציה Game.saveGame()

- בכל ספריות ה GUI המודרניות הדבר מתבצע בעזרת ה Observer Design Pattern (שמכונה בעולם ה GUI דווקא Listener)
  - עבור ארועים שמעניינים אותנו אנו מגדירים **מאזינים** (listeners) שמיודעים בכל פעם שהארוע קרה והם אלה שקוראים לפונקציה המתאימה

- פרטים בהמשך...

"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();
    }
}
```

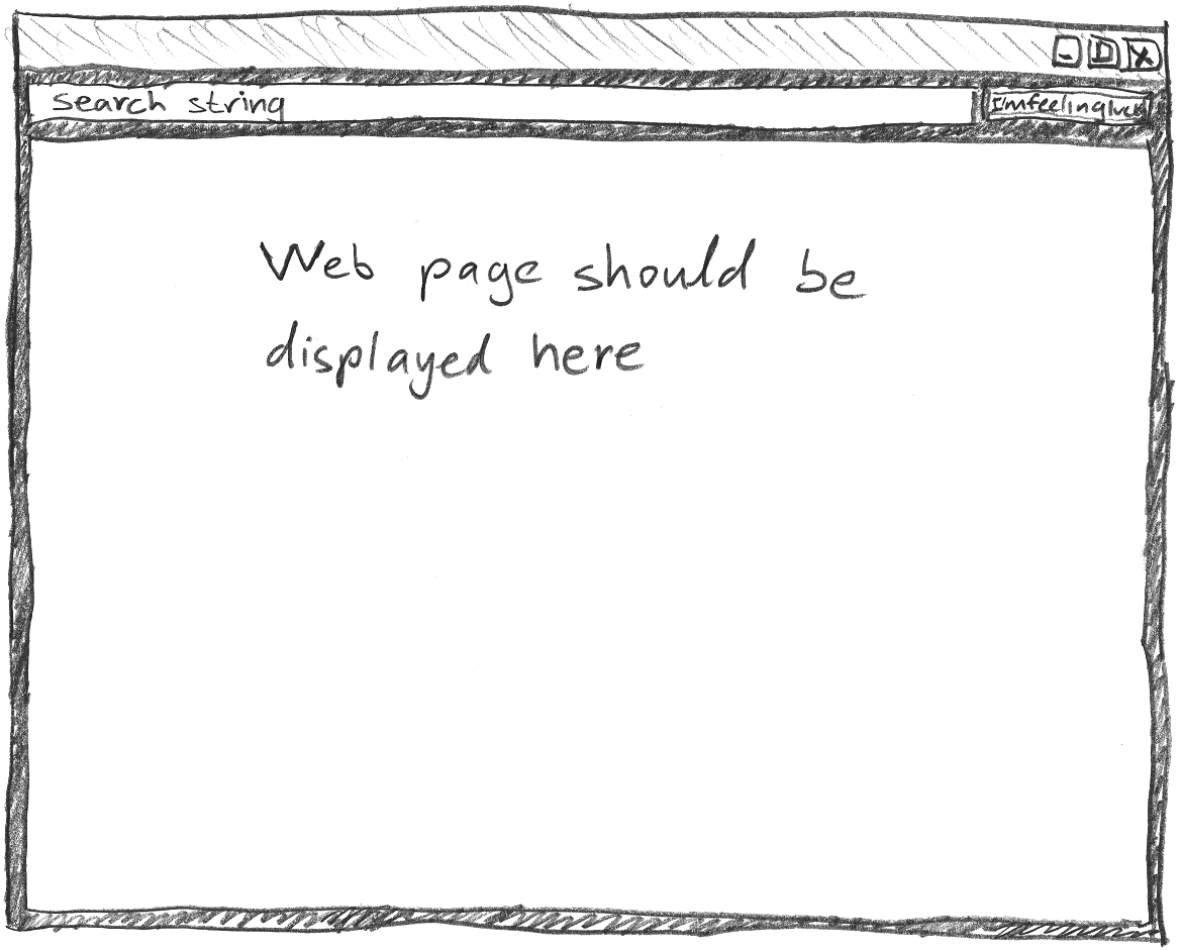
עצם ה Display מייצג את המסך

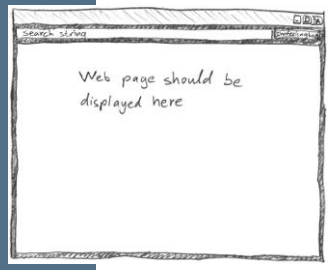
כל חלון מיוצג ע"י עצם מטיפוס Shell
ההורה של החלון הוא המסך

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
  - 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) – מקשה על הפצה כ standalone

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

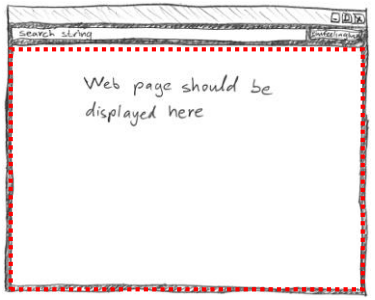
בניית רכיבי מנשק

- 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));
```

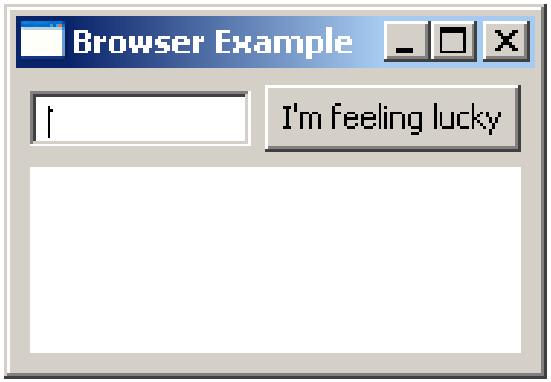
```
browser = new Browser(shell, SWT.NONE);

browser.setLayoutData(new GridData(SWT.FILL,
                SWT.FILL,     //fill both ways
                false, true,  //row grabs vertical space
                2, 1));       //widget spans 2 columns
```



כמעט סיימנו

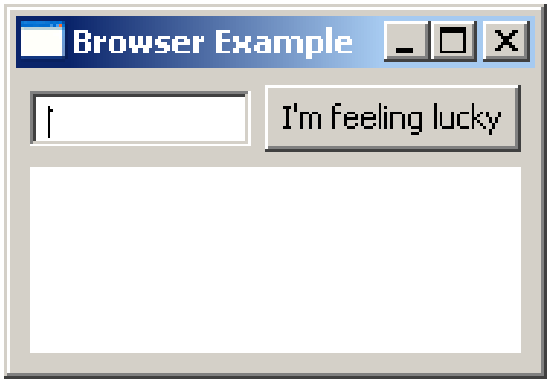
- נותר לבקש ממנהל הפריסה לארגן את הרכיבים ולהציג אותו למסך:

```
private void createShell() {
    ...

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

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



בעיה. היישום לא עושה כלום

- החלון (shell) מוצג כהלכה על המסך, ומתנהג נכון כאשר אנו משנים את גודלו
  - אולם אם מקלידים מילות חיפוש ולוחצים על הכפתור לא קורה דבר
- לשם כך צריך להוסיף ליישום **טיפול בארוע לחיצה** על הכפתור
- בעת לחיצה:
  - יש לקרוא את הטקסט שהוקלד בתיבת החיפוש
  - לבקש מאת גוגל את כתובת האתר המתאים ביותר לטקסט
  - ולהזין את הכתובת לאובייקט ה Browser
- אנו נייצר **מחלקה פנימית אנונימית** שתאזין לארוע ותבצע את הפונקציונליות הדרושה



מחלקות פנימיות - תזכורת

- כבר ראינו בקורס מחלקות פנימיות - מחלקות אשר מוגדרות בתוך התחום של מחלקות אחרות
- בג'אווה 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()**

מחלקה אנונימית - תחביר

כאשר יורשת מהמחלקה *class-name* :

```
new class-name ( [ argument-list ] ) { class-body }
```

כאשר מממשת המנשק *interface-name*:

```
new interface-name () { class-body }
```

הפרוצדורה שהכפתור מפעיל

- נוסיף לכפתור הדפדפן שלנו **מאזין** – מחלקה אשר מקשיבה להקלקות על הכפתור
- המאזין הוא מחלקה אנונימית המממשת את המנשק `SelectionListener`
  - עליו לממש את השרותים `widgetSelected` ו- `widgetDefaultSelected`

```
button.addSelectionListener(
    new SelectionListener() {
        @Override public void widgetSelected(SelectionEvent e) {
            String query = text.getText();
            String url = search(query);  // missing exception handling
            shell.setText(url);
            browser.setUrl(url);
        }
        @Override public void widgetDefaultSelected(SelectionEvent e) {}
    }
);
```

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

ארבע גישות לטיפול באירועים

- בעזרת טיפוסים סטאטיים 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 סטאטיים שמתאימים לאירועים ספיציפיים:
  - האירוע מפעיל עצם מטיפוס Listener שמממש שירות בודד, handleEvent, והאירוע מדווח בעזרת טיפוס Event
- בעזרת מחלקות ברירת מחדל אשר "**מודעות**" לארועים **אך לא עושות דבר** הנקראות מתאמים (Adapters)
  - כדי להגדיר את הפעולה שיש לבצע נירש (אנונימית) מכזה Adapter ונדרוס את השרות המעניין
  - שימושי כאשר במנשק יש שרותים רבים שאינם מעניינים אותנו
- 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
- פרטים בתרגול

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)

דוגמה לשימוש במתאם (adapter)

- 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`**
- אין צורך לדרוס את השרות **`widgetDefaultSelected`**

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

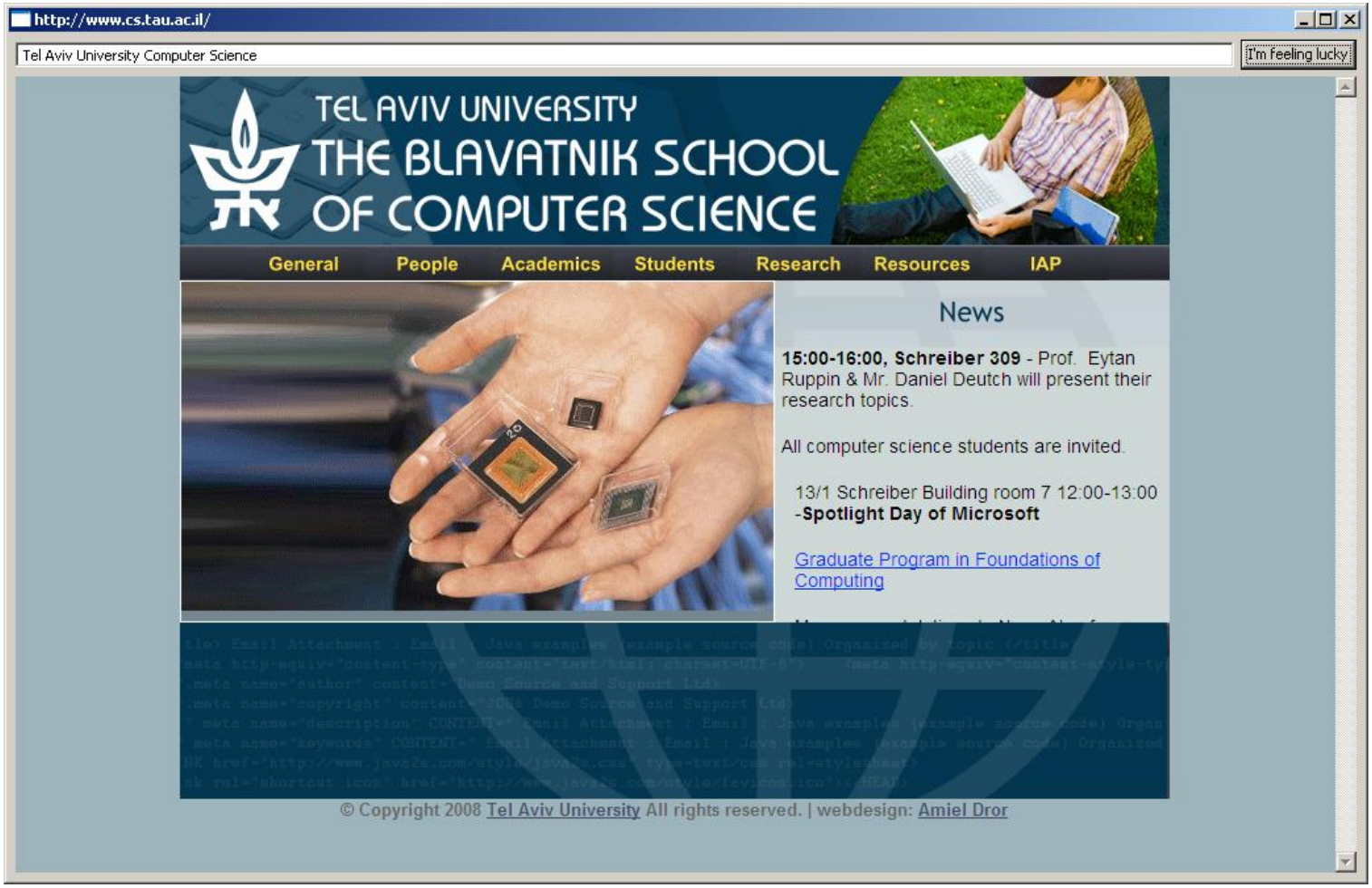
Listener לעומת Adapter

- לכפתור הוספנו מאזין ספציפי ממחלקה אנונימית שמרחיבה את **`SelectionAdapter`**
- **`SelectionAdapter`** היא מחלקה מופשטת שממממשת 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 לכתוב ומסורבל לקרוא
  - כיום בסביבות פיתוח מודרניות ניתן לחולל קוד מימוש ריק אוטומטית

עכשיו באמת כמעט סיימנו.
חיפוש ב-Google

```
private static String search(String q) throws IOException, JSONException {

     URL url = new URL("http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q="
                                                          + URLEncoder.encode(q, "UTF-8"));
     URLConnection connection = url.openConnection();
     StringBuilder builder = new StringBuilder();

     Scanner s = new Scanner(connection.getInputStream());
     while (s.hasNextLine()) {
        builder.append(s.nextLine());
     }

     JSONObject json = new JSONObject(builder.toString()).getJSONObject("responseData");
     JSONArray jsonArr = json.getJSONArray("results");
     json = (JSONObject) jsonArr.get(0);

     return (String) json.get("url");
}
```

והתוצאה


http://www.cs.tau.ac.il/
Tel Aviv University Computer Science
I'm feeling lucky
TEL AVIV UNIVERSITY
THE BLAVATNIK SCHOOL
OF COMPUTER SCIENCE
General
People
Academics
Students
Research
Resources
IAP
News
15:00-16:00, Schreiber 309 - Prof. Eytan Ruppin & Mr. Daniel Deutch will present their research topics.
All computer science students are invited.
13/1 Schreiber Building room 7 12:00-13:00
-Spotlight Day of Microsoft
Graduate Program in Foundations of Computing
© Copyright 2008 Tel Aviv University All rights reserved. | webdesign: Amiel Dror

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
  - אפשר להשתמש במנגנון עריכה ייעודי למנשקים גראפיים (GUI Builder)

GUI on Different Platforms

- בשנים האחרונות יש מגמה של פיתוח מנשקי משתמש לפלטפורמות שונות:
  - מבוססי Web – אשר ניתן להציג אותם בדפדפן
  - לטלפון / לטאבלט
- העקרונות של מנשקים אלו דומים לאלו שראינו אולם הטכנולוגיות המעורבות בכך שונות:
  - HTML / JavaScript
  - Flash, Silverlight
- במקביל, נעשים גם מאמצים לתרגום אוטומטי של קוד Java לקוד HTML / JavaScript למשל GWT

Look and Feel

- מערכות הפעלה עם מנשק גראפי מספקות שירותי מנשק (למשל, Windows MacOS ו-Android 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

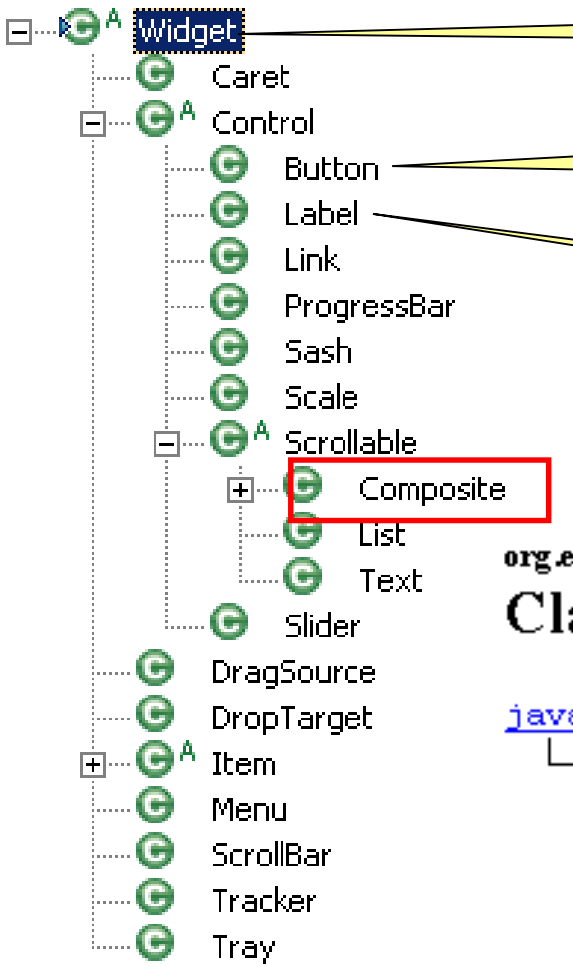
- מאפשר להגדיר מראות חדשים לרכיבים; שימושי עבור משחקים, עבור תוכניות שרוצים שלא יראו כמו תוכנות מחשב (למשל נגני מוסיקה וסרטים), ובשביל מיתוג (branding)
- מאפשר לבנות יישומים עם מראה אחיד על כל פלטפורמה; שימושי ליישומים ארגוניים
- קשה לממש look & feel חדש
- סכנה של מראה מיושן, אם מערכת ההפעלה החליפה את 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 מופעלים 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 – רשימה חלקית



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**

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

פריסה נכונה Correct layout

- 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 – מיכל מכיל

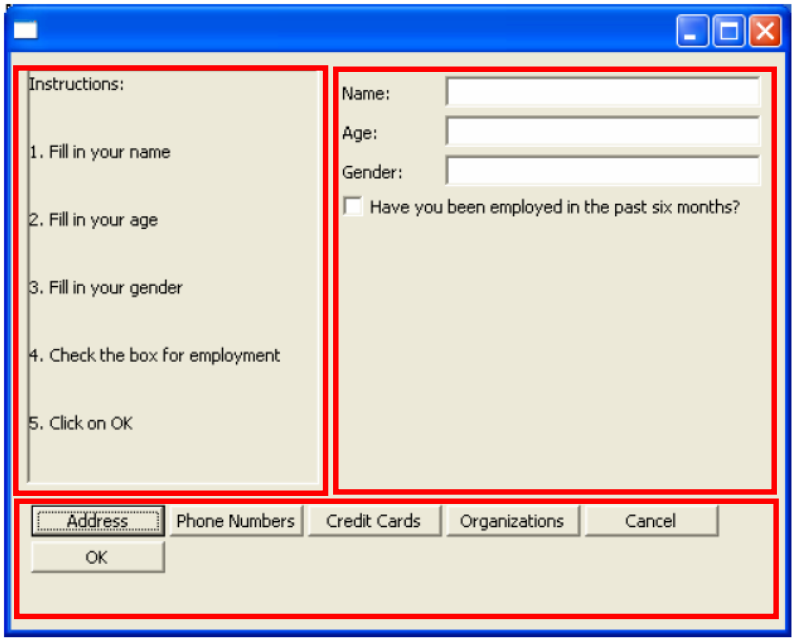
הצג את החלון על המסך

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

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

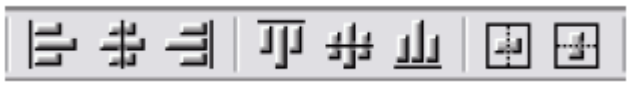


- כדי לבנות בצורה מודולרית מסכים מורכבים (ולפתח כל איזור בנפרד) – רצוי להשתמש במחלקה Composite (מקבילה למחלקה J/Panel ב- Swing/AWT)
- בדוגמא שלפנינו ה Shell מכיל 3 Composites שונים, כל אחד מהם מנוהל ע"י מנהל פריסה משלו

```
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:");
```

GUI Builders

- כאשר ה GUI מורכב, ניתן ומומלץ להשתמש בתוכנות ויזואליות לעיצוב מנשקים
- המתכנת / מעצב בוחר רכיבי GUI מתפריט, הוא יכול עם העכבר לשנות את מיקומם וגודלם, ויכול בכפתור ימני לעדכן את מאפיניהם
- התוכנה מחוללת בצורה אוטומטית את הקוד המתאים – אותו ניתן גם לערוך
- לדוגמא
  - https://eclipse.org/windowbuilder/



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)

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

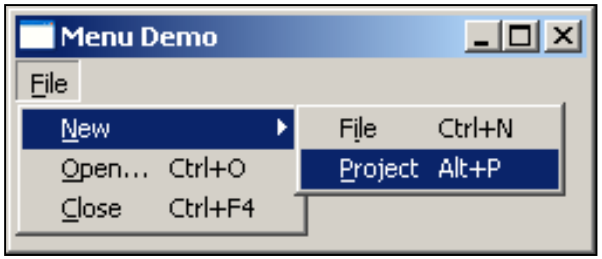
דוגמא – צבעים וגופנים

```
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();
```

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) בעזרת מחלקת ההקשר הגרפי (**GC – Graphics Context**)

- ב **SWT** ניתן לצייר על ה- **GC** של כל רכיב שמממש את המנשק Drawable לרבות: Control ו- Image

- המחלקה Canvas מיועדת לשמש כמשטח ציור עם פונקציות ייעודיות לשם כך

http://www.eclipse.org/articles/Article-SWT-graphics/SWT_graphics.html

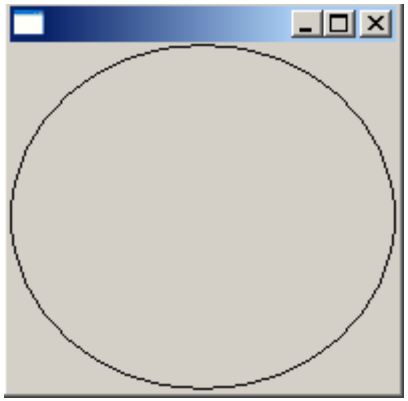
עשה זאת בעצמך

- 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 1"

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

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

GC gc = new GC(shell);
Rectangle rect = shell.getClientArea();
gc.drawOval(0, 0, rect.width - 1, rect.height - 1);

while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
      display.sleep();
}
```

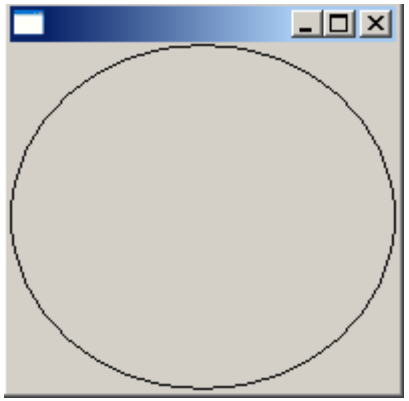


עשה זאת בעצמך

- ואולם, הקוד בשקף הקודם אינו תומך בשינוי גודל החלון
- 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

"צייר לי עיגול 2"

```
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 אירועי לחיצה על העכבר לפי קואורדינטות
  - שימוש ב widgets:
    - בניית סריג של כפתורים ריבועיים בצבעי שחור ולבן לסרוגין
    - לכידה של אירועי בחירת כפתור
- מה היתרונות והחסרונות של כל אחת מהגישות?

JFace

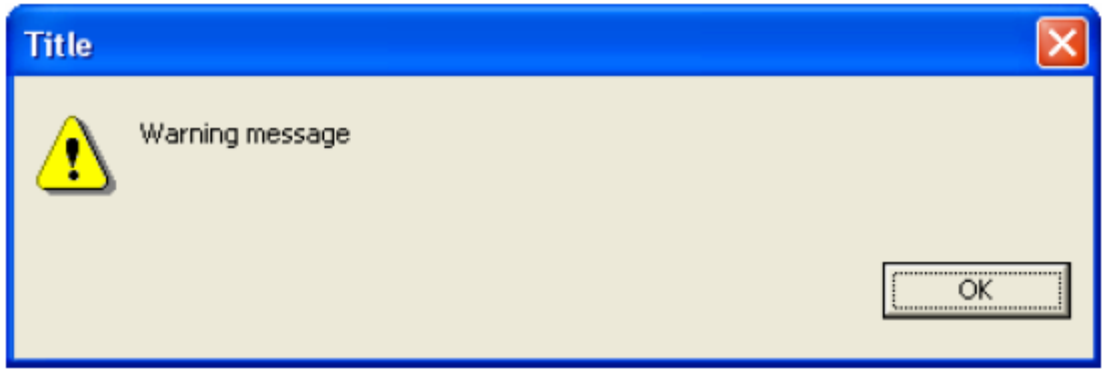
- החבילה JFace מציעה מגוון מחלקות המציעות שרותי GUI מתקדמים (אבל שכיחים) הכתובים מעל (בעזרת) הספריה SWT

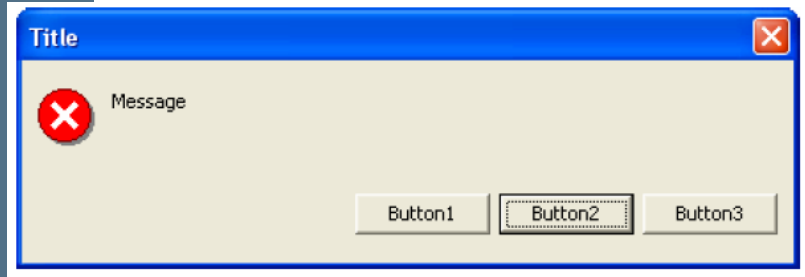
- כך המתכנת אינו צריך לייצר להרכיב את רכיבי ה GUI בעצמו, אלא רק לייצר ולקנפג עצמים שהוכנו מראש

JFace Dialogs

- בחבילה JFace ניתן 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:

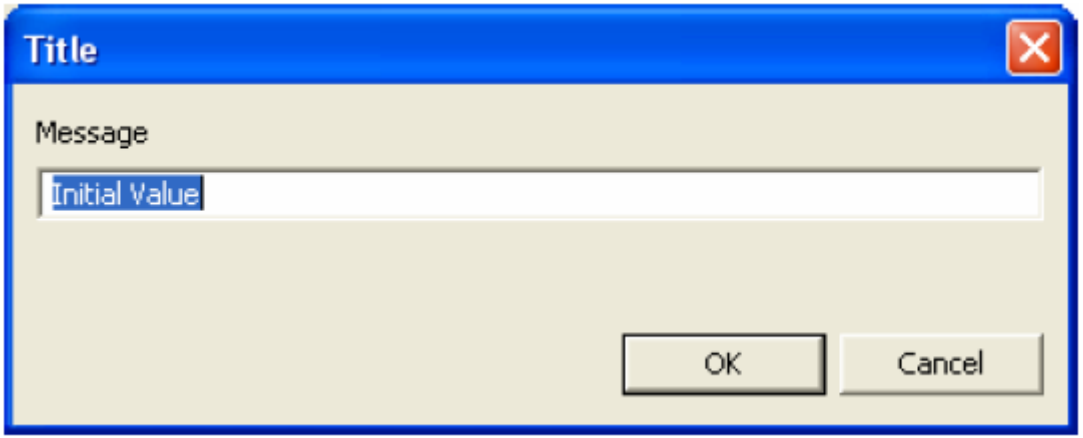
MessageDialog.NONE, MessageDialog.ERROR, MessageDialog.INFORMATION, MessageDialog.QUESTION, MessageDialog.WARNING

קריאת בחירת המשתמש ע"י:

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

JFace Dialogs

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

inputBox.open();
```



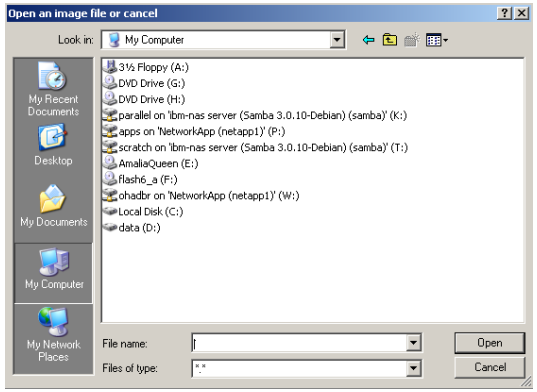
קריאת קלט משתמש ע"י:

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

JFace Dialogs

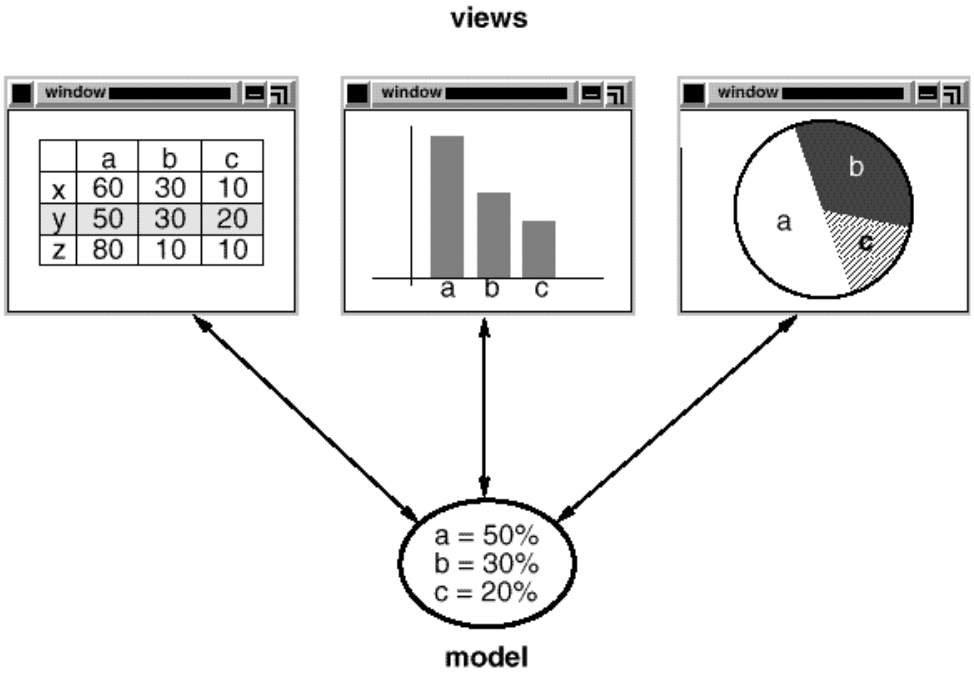
```
ColorDialog d = new ColorDialog(shell);
RGB selection = d.open();
```

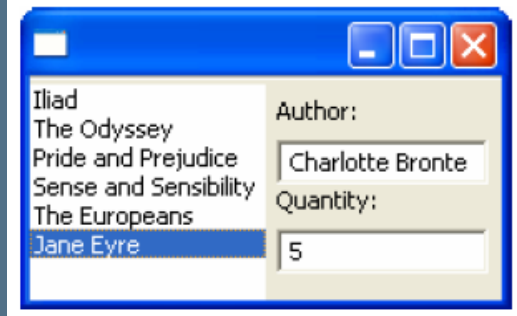
```
FileDialog dialog = new FileDialog (shell, SWT.OPEN);
dialog.setText ("Open an image file or cancel");
String string = dialog.open ();
```



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

- אבל גם הבעיה ההפוכה מעניינת – איך לקשר בין פעולות של המשתמש על רכיבים ויזואלים (רשימות, מחרוזות) ובין העצמים הסמנטיים של המודל
- אחת המשפחות ב JFace 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)

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