

תוכנה 1

תרגול מספר 12:

GUI

כתיבת מחלקות גנריות

בית הספר למדעי המחשב
אוניברסיטת תל אביב

SWT

- בנויה על העיקרון של publish/subscribe
- אלמנטים בסיסיים (Widgets) מייצרים אירועים (Events) שאליהם נרשמים מאזינים (Listener)
- דוגמא 1: משתמש לוחץ על **כפתור**, שמייצר **אירוע לחיצה**. **מאזין** שנרשם ל**אירוע הלחיצה של הכפתור** יכול לשנות את כותרת החלון
- דוגמא 2: משתמש סוגר את ה**חלון**, שמייצר **אירוע סגירת חלון**. **מאזין** שנרשם ל**אירוע סגירת החלון** פותח חלון ששואל את המשתמש אם הוא רוצה לשמור את השינויים לפני שיצא מהתכנית.
- ה **Widgets** וה- **Events** מוגדרים ע"י כותבי הספרייה
- **מאזינים** נכתבים ע"י כותבי האפליקציה
- מאפשר תגובות שונות לאירועים כתלות באפליקציה

SWT Widgets

- אבני הבניין של ממשקים גרפיים
- מוגדרים ב [org.eclipse.swt.widgets](http://www.eclipse.org/swt/widgets/)
- תת-טיפוסים של המחלקה האבסטרקטית Widget ([קישור לתיעוד](#))
- האתר של SWT מכיל דוגמאות קצרות (snippets) לשימוש בכל Widget <http://www.eclipse.org/swt/widgets/>



Shell

Jack and Jill went up
the hill to fetch a pail
of water, Jack fell
down and broke his
crown and Jill came
tumbling after!

Label

The quick brown fox jum

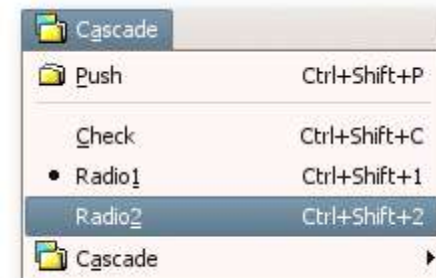
Text

Name	Type	Size
<input type="checkbox"/> Index:0	classes	0
<input checked="" type="checkbox"/> Index:1	databases	2556
<input type="checkbox"/> Index:2	images	9157
<input checked="" type="checkbox"/> Index:3	classes	0
<input type="checkbox"/> Index:4	databases	2556

Table

Apples
Oranges
Bananas
Grapefruit
Peaches
Kiwi
Apricots
Strawberries
The Longest String

List



Menu



Button

עוד על Widgets

- ביצירת Widget נגדיר
 - את ה"הורה" שלו
 - את הסגנון שלו
- ההורה הוא Widget היורש מ-Composite, מה שאומר שניתן להוסיף אליו Widgets אחרים
 - לדוגמא, כפתור שהורה שלו הוא טאב שהורה שלו הוא חלון – יופיע כפתור בתוך הטאב שבחלון
 - ה- Widget מתווסף להורה בזמן הקריאה לבנאי
- עבור סגנונות, קיימים קבועים במחלקה SWT



כפתור

```
public class ShellWithButton1 {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout(SWT.VERTICAL));
        shell.setText("example1");

        Button ok = new Button(shell, SWT.PUSH);
        ok.setText("Push Me!");

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



אז מה היה לנו כאן?

```
Display display = Display.getDefault();
```

- [Display](#) - מקשר בין SWT לתצוגת מערכת ההפעלה (למשל, המסך)

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

- [Shell](#) - חלון. שימו לב שיצירת חלון לא פותחת אותו עדיין.

```
shell.setLayout(new FillLayout(SWT.VERTICAL));
```

- לכל Composite ניתן להוסיף layout שיגדיר כיצד Widgets מסודרים בתוכו
- [FillLayout](#) - ה- Widgets ממלאים את ה-Composite
- [SWT.VERTICAL](#) - ה- Widgets מסודרים בצורה אנכית לפי סדר הוספתם
- ללא layout לא נראה את ה-Widgets כלל



אז מה היה לנו כאן?

```
shell.setText("example1");
```

```
Button ok = new Button(shell, SWT.PUSH);
ok.setText("Push Me!");
```

- `setText` - משתנה בהתאם לטיפוס ה-`Widget`. בחלון - קובע את הכותרת, בכפתור לחיצה (`PUSH`) - קובע מה כתוב על הכפתור.

```
shell.pack();
```

- `pack` - גורם לאובייקט גרפי לחשב ולהתאים את גודלו, למשל בהתאם ל-`layout`, ל-`Widgets` שבתוכו וכו'

```
shell.open();
```

- פתיחת החלון

לולאת האירועים (Event loop)

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

- קוד סטנדרטי שמופיע כמעט בכל תכנית SWT
- כל עוד החלון הראשי של התכנית לא נסגר, טפל באירוע הבא בתור ובדוק האם קיים אירוע נוסף
- אם לא קיים – היכנס למצב שינה עד לקבלת אירוע נוסף
- בסוף משחררים את כל המשאבים שהוקצו ע"י קריאה ל- `dispose`
 - כאשר משחררים אלמנט גרפי, כל הצאצאים שלו גם משתחררים
 - כאשר משחררים את `Display`, כל המשאבים הגרפיים משתחררים

בד"כ נכתוב את מימוש
מחלקת המאזין בעצמנו

הוספת טיפול באירועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול באירוע (Event) "לחיצה"
- ניצור מופע של מחלקה המממשת את המנשק `SelectionListener`
- "נרשום" את המופע להאזין להודעות על אירוע הלחיצה אצל ה-Widget המתאים
- כיצד נדע אילו אירועים מייצר Widget מסוים? איזה מנשק עלינו לממש?
- נסתכל בתיעוד
- התיעוד של `Button`:

Events:
Selection

Method Summary

void [`addSelectionListener`](#) ([`SelectionListener`](#) listener)

Adds the listener to the collection of listeners who will be notified when the of the messages defined in the `SelectionListener` interface.

טיפול במקרים שונים של
היווצרות אירוע לחיצה,
מגדיר את פעולת הכפתור

טיפול באירועים במחלקה נפרדת

```
public class ButtonHandler
    implements SelectionListener {

    public void widgetSelected(SelectionEvent e) {
        if (e.getSource() instanceof Button) {
            Button b = (Button) e.getSource();
            b.setText("Thanks!");
        }
    }

    public void widgetDefaultSelected(SelectionEvent e){
        // TODO Auto-generated method stub
    }
}
```

טיפול באירועים במחלקה נפרדת

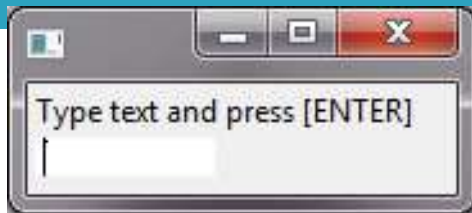
```
Display display = Display.getDefault();
Shell shell = new Shell(display);
shell.setLayout(new FillLayout(SWT.VERTICAL));
shell.setText("example2");

Button ok = new Button(shell, SWT.PUSH);
ok.setText("Push Me!");
ok.addSelectionListener(new ButtonHandler());

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

טיפול בארועים במחלקה פנימית

- לעיתים הטיפול באירוע דורש הכרות אינטימית עם המקור (כדי להימנע מחשיפת המבנה הפנימי של המקור)
- שימוש במחלקה פנימית יוצר את האינטימיות הדרושה
- בדוגמא הבאה נרצה לעדכן תווית על סמך קלט מהמשתמש
- דרושה הכרות לא רק עם יוצר האירוע (Text) אלא גם עם חלקים אחרים במבנה



מחלקה פנימית

```
public class ShellWithLabelAndTextField1 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}

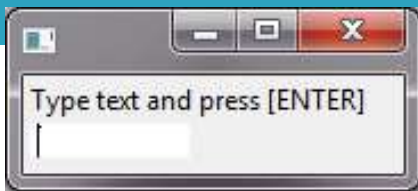
    public void createShell() {...}
```

המחלקה הפנימית ניגשת לשדות הפרטיים של המחלקה העוטפת

```
public class InnerHandler implements KeyListener {
    public void keyPressed(KeyEvent e) {
        if (e.character == SWT.CR) {
            label.setText(text.getText());
            text.setText("");
        }
    }

    public void keyReleased(KeyEvent e) {
        // TODO Auto-generated method stub
    }
}
```

```
}
```



מחלקה פנימית

```
public class ShellWithLabelAndTextField1 {  
    private Label label;  
    private Text text;  
  
    public static void main(String[] args) {  
        ShellWithLabelAndTextField1 shell = new ShellWithLabelAndTextField1();  
        shell.createShell();  
    }  
  
    public void createShell() {  
        Display display = new Display();  
        Shell shell = new Shell(display);  
        shell.setLayout(new RowLayout(SWT.VERTICAL));  
  
        label = new Label(shell, SWT.CENTER);  
        label.setText("Type text and press [ENTER]");  
  
        text = new Text(shell, SWT.LEFT);  
        text.addKeyListener(new InnerHandler());  
        // pack(), open(), while ... Dispose()  
    }  
}
```

שימוש במחלקות אנונימיות

- בדרך כלל נזדקק רק למאזין יחיד לכל אירוע
- נשתמש במחלקה לוקאלית אנונימית

תזכורת: `new ClassName([constructor-arguments]) {classBody}`

- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שיוורשת באופן אוטומטי מ `className`

- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שמממשת באופן אוטומטי את `interfaceName`

`new InterfaceName() {classBody}`

מחלקה אנונימית

```

public class ShellWithLabelAndTextField2 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}

    public void createShell() {
        ...
        text.addListener(new KeyListener() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    label.setText(text.getText());
                    text.setText("");
                }
            }

            public void keyReleased(KeyEvent e) {
                // TODO Auto-generated method stub
            }
        });
        // pack(), open(), while ... Dispose()
    }
}

```

← סוגר סוגריים של המתודה addKeyListener()

שימוש ב Adapter

```

public class ShellWithLabelAndTextField2 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}

    public void createShell() {
        ...
        text.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    label.setText(text.getText());
                    text.setText("");
                }
            }
        });

        // pack(), open(), while ... Dispose()
    }
}

```

```

public class KeyAdapter implements
    KeyListener {
    @Override
    public void keyPressed(KeyEvent arg0) {
        //do nothing
    }

    @Override
    public void keyReleased(KeyEvent arg0) {
        // do nothing
    }
}

```

עוד על טיפוסים מוכללים GENERICIS

שימוש במחלקה גנרית קיימת
כתיבת מחלקה גנרית

תזכורת: Generics

מנגנון תחבירי המאפשר לכתוב מחלקה או מתודה כך שתעבודנה עם אובייקטים מטיפוסים שונים, תוך אכיפה של בטחון-טיפוסים בזמן קומפילציה

Java Generics Tutorial •

<http://docs.oracle.com/javase/tutorial/java/generics/index.html>

כתיבת מחלקה גנרית

מימוש של מחסנית המאחסנת אובייקטים מסוג Integer:

```
class StackOfIntegers {
    private LinkedList<Integer> items = new LinkedList<Integer>();

    public void push(Integer item) { items.addFirst(item); }
    public Integer pop()           { return items.removeFirst(); }
    public boolean isEmpty()       { return items.isEmpty(); } }
}
```

מימוש גנרי של מחסנית המאחסנת אובייקטים מסוג T כלשהו:

```
class Stack<T> {
    private LinkedList<T> items = new LinkedList<T>();

    public void push(T item)      { items.addFirst(item); }
    public T pop()                 { return items.removeFirst(); }
    public boolean isEmpty()      { return items.isEmpty(); }
}
}
```

כתיבת מחלקה גנרית

כשנכתוב מחלקה גנרית, נכריז בראש המחלקה על הפרמטר שיציין את הטיפוס הגנרי (T במקרה זה).

לאחר מכן, נניח שכל הופעה של הפרמטר בקוד המחלקה תוחלף בטיפוס האקטואלי שאתו יאותחל מופע המחלקה.

מימוש גנרי של מחסנית המאחסנת אובייקטים מסוג T כלשהו:

```
class Stack<T> {
    private LinkedList<T> items = new LinkedList<T>();

    public void push(T item)    { items.addFirst(item);    }
    public T pop()              { return items.removeFirst(); }
    public boolean isEmpty()    { return items.isEmpty();   }
}
```

שימושים אפשריים במחלקה הגנרית:

```
Stack<Integer> s1 = new Stack<Integer>();
Stack<String>  s2 = new Stack<String>();
```

דוגמאות לירושה של מחלקות גנריות

- יש לשים לב אילו פרמטרים גנריים צריך לציין בכל מחלקה

