

OO - GUI

- מערכות ה-GUI המודרניות נחשבות ל killer application של הגישה מונחת העצמים
- מאוד טבעי ואינטואיטיבי לדבר על יסודות OO כגון ירושה, הכלה, האצלה, הפרדת ההצגה והמודל, הסתרת מידע ואחרים בהקשר של GUI

ארועים ב SWT

(SWT כמיקרוקוסמוס של תכנות מונחה עצמים)

תוכנה 1 בשפת Java
אורנית דרור ואוהד ברזילי

הוספת טיפול בארועים

- הכפתור לא מגיב לחיצות. יש להוסיף טיפול בארוע "לחיצה"
- על המחלקה המטפלת לממש את הממשק `SelectionListener`
- על הכפתור עצמו להגדיר מי העצם (או העצמים) שיטפלו בארוע
- כמה גישות אפשריות:
 - הגדרת מחלקה שתירש מכפתור
 - מחלקה שתכליל כפתור כאחד משדותיה
 - יצירת מחלקה עצמאית שתטפל בארועי הלחיצה
- לכל אחת מהאפשרויות יתרונות וחסרונות שידונו בהמשך

כפתור



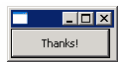
```
public class ShellWithButton {  
  
    public static void main(String[] args) {  
        Display display = new Display ();  
        Shell shell = new Shell (display);  
        Button ok = new Button (shell, SWT.PUSH);  
        ok.setText ("Push Me!");  
        ok.setLocation(0,0);  
        ok.setSize(100,30);  
        shell.pack ();  
        shell.open ();  
        while (!shell.isDisposed ()) {  
            if (!display.readAndDispatch ()) display.sleep ();  
        }  
        display.dispose ();  
    }  
}
```

ירושה מכפתור

- גישה מקובלת ב-AWT וב-Swing היא הגדרת מחלקה שתירש מכפתור ותממש את הממשק הדרוש
- גישה זו אינה מומלצת ב-SWT – המתודה `checkSubclass()` המוגדרת ב `widget` תזרוק בתגובה `SWTException`
- הדבר נועד למנוע ירושה ממי שאינו בקיא בפרטי הרכיבים השונים
- ניתן לעקוף זאת ע"י דריסת המתודה `checkSubclass()`

Observer Design Pattern

- דרך הטיפול בארועי GUI היא מקרה פרטי של תבנית עיצוב יסודית בגישת התכנות מונחה העצמים
- הבעיה הכללית מאפיינת Subject אשר מחולל ארועים לוגיים (לא בהכרח גרפיים) וישויות אחרות במערכת, `Observers`, אשר מעוניינות לקבל חייוו על כך
- לצורך כך ה `Observers` נרשמים כמנויים (`subscribers`) על הארוע הלוגי אצל ה `Subject`
- ה `Subject` מיידיע את כל מנוייו (`notify`) כל אימת שמתרחש ארוע שיש לו מנויים



ירושה מכפתור

```
public class ExtendedButton extends Button
implements SelectionListener{

public ExtendedButton(Composite parent, int style) {
super(parent, style);
}

public void widgetSelected(SelectionEvent e) {
setText("Thanks!");
}

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

@Override
protected void checkSubclass() {}
}
```

אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

8



ירושה מכפתור

```
public class TestExtendedButton {
public static void main(String[] args) {
Display display = new Display ();
Shell shell = new Shell (display);
ExtendedButton ok = new ExtendedButton(shell, SWT.PUSH);
ok.addSelectionListener(ok);
ok.setText ("Push Me!");
ok.setLocation(0,0);
ok.setSize(100,30);
shell.pack ();
shell.open ();
while (!shell.isDisposed ()) {
if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
}
```

אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

7



כפתור מוכל האצלה

```
public class TestAggregatedButton {
public static void main(String[] args) {
Display display = new Display ();
Shell shell = new Shell (display);
AggregatedButton ok = new AggregatedButton(shell, SWT.PUSH);
ok.addSelectionListener(ok);
ok.setText ("Push Me!");
ok.setLocation(0,0);
ok.setSize(100,30);
shell.pack ();
shell.open ();
while (!shell.isDisposed ()) {
if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
}
```

אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

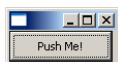
10

כפתור מוכל

- הגדרת מחלקה אשר תכיל את הכפתור כשדה וגם תממש את לוגיקת הטיפול בארועים פותרת את הצורך לרשת מהמחלקה Button
- הכלה (Aggregation) מחייבת את המחלקה החדשה לנקוט אחת מהשתיים:
 - לאפשר האצלה (delegation) של המתודות של הכפתור
 - או
 - לחשוף את הכפתור כלפי חוץ ע"י שאלתה מתאימה

אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

9

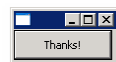


כפתור מוכל חשיפת הכפתור

```
public class TestAggregatedButton {
public static void main(String[] args) {
Display display = new Display ();
Shell shell = new Shell (display);
AggregatedButton ok = new AggregatedButton(shell, SWT.PUSH);
ok.getButton().addSelectionListener(ok);
ok.getButton().setText ("Push Me!");
ok.getButton().setLocation(0,0);
ok.getButton().setSize(100,30);
shell.pack ();
shell.open ();
while (!shell.isDisposed ()) {
if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
}
```

אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

12



כפתור מוכל האצלה

```
public class AggregatedButton implements SelectionListener{
private Button b;

public AggregatedButton(Composite parent, int style) {
b = new Button(parent, style);
}

public void widgetSelected(SelectionEvent e) {
b.setText("Thanks!");
}

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

public void addSelectionListener(SelectionListener listener) {
b.addSelectionListener(listener);
}

public void setSize(int width, int height) { b.setSize(width, height); }
public void setLocation(int x, int y) { b.setLocation(x, y); }
public void setText(String string) { b.setText(string); }
}
```

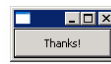
אוניברסיטת תל אביב
אוניברסיטת דוד וואהר ברוי

11

טיפול בארועים במחלקה נפרדת

• יתרונות:

- הלקוח עובד עם כפתור סטנדרטי ולכן אין צורך לחשוף מבנה פנימי ללקוח
- הלקוח עובד עם כפתור סטנדרטי ולכן אין צורך לבצע האצלה לשרותי המחלקה
- מודולריות – הלוגיקה (טיפול בארועים) מופרדת מהצורניות (מיקום, גודל, סגנון)



כפתור מוכל חשיפת הכפתור

```
public class AggregatedButton implements SelectionListener {  
    private Button b;  
  
    public AggregatedButton(Composite parent, int style) {  
        b = new Button(parent, style);  
    }  
  
    public void widgetSelected(SelectionEvent e) {  
        b.setText("Thanks!");  
    }  
  
    public void widgetDefaultSelected(SelectionEvent e) {  
        // TODO Auto-generated method stub  
    }  
  
    public Button getButton() {  
        return b;  
    }  
}
```

טיפול בארועים 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 בארועים במחלקה נפרדת

```
public class TestButtonHandler {  
    public static void main(String[] args) {  
        Display display = new Display ();  
        Shell shell = new Shell (display);  
        Button ok = new Button(shell, SWT.PUSH);  
        ok.addSelectionListener(new ButtonHandler());  
        ok.setText ("Push Me!");  
        ok.setLocation(0,0);  
        ok.setSize(100,30);  
        shell.pack ();  
        shell.open ();  
        while (!shell.isDisposed ()) {  
            if (!display.readAndDispatch ()) display.sleep ();  
        }  
        display.dispose ();  
    }  
}
```

מחלקה פנימית

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

טיפול בארועים במחלקה נפרדת

• חסרונות:

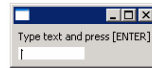
- אם מחלקה נפרדת מטפלת במגוון גדול של ארועים המגיעים ממקורות (רביבים) שונים בטיפוסם או נו גרררים לבדיקות טיפוס (instanceof) במקום להשתמש בפולימורפיזם
- לעיתים הטיפול בארוע דורש הכרות אינטימית עם המקור שיצר את הארוע (כדי להימנע מחשיפת המבנה הפנימי של המקור)
- שימוש במחלקה פנימית יוצר את האינטימיות הדרושה
- בדוגמא הבאה מחלקה המכילה שדה טקסט ותוויית תעדכן את התוויית לפי הנכתב בשדה הטקסט ע"י שימוש במחלקה פנימית

מחלקה פנימית אנונימית

```
public class ShellWithLabelAndTextField {
    ...
    public void createShell() {
        ...
        t.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent e) {
                if (e.character == NEW_LINE_CHAR) {
                    l.setText(t.getText());
                    t.setText("");
                }
            }

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

סוגר סוגריים של המתודה addKeyListener()



מחלקה פנימית

```
public class ShellWithLabelAndTextField {
    private Label l;
    private Text t;

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

    public class InnerHandler implements KeyListener
    {
        public void keyPressed(KeyEvent e) {
            if (e.character == NEW_LINE_CHAR) {
                l.setText(t.getText());
                t.setText("");
            }
        }

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

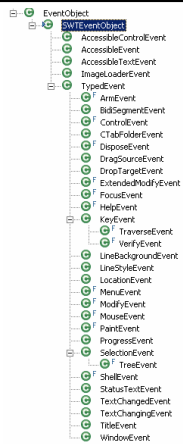
המחלקה הפנימית ניגשת לשדות הפרטיים של המחלקה העוטפת

התרגול הבא

- בתרגול הבא נקיים פתרון מודרך של מבחר שאלות בתכנות JAVA כהכנה לבחינה

מחלקות פנימיות - דיון

- הסתרת מידע
- האם המחלקה הפנימית רלוונטית רק בהקשר של המחלקה העוטפת?
- אינה מעודדת שימוש חוזר – מחלקות פנימיות ובפרט מחלקות פנימיות אנונימיות עשויות לשכפל קוד קריאות קוד
- שימוש במחלקות Adapter משפר את קריאות הקוד אך מגביל את יכולות הירושה



"SWT - הפקת ארועים"

ארועים ומאזינים ב SWT

- SWT מכיל מגוון רחב מאוד של ארועים ושל מאזינים המטפלים בהם
- צורות העבודה עם המאזינים השונים והארועים השונים דומות לצורות שהדגמנו
- בשקפים הבאים תמצאו פירוט של שמות המחלקות השונות שבחבילת SWT

"מאזין לכל ארוע"



org.eclipse.swt.internal
Interface SWTEventListener

All Superinterfaces:
java.util.EventListener

All Known Subinterfaces:
[AccessibleControlListener](#), [AccessibleListener](#), [AccessibleTextListener](#), [ArmListener](#),
[BidSegmentListener](#), [CloseWindowListener](#), [ControlListener](#), [CTabFolder2Listener](#),
[CTabFolderListener](#), [DisposeListener](#), [DragSourceListener](#), [DropTargetListener](#),
[ExtendedModifyListener](#), [FocusListener](#), [HelpListener](#), [ImageLoaderListener](#), [KeyListener](#),
[LineBackgroundListener](#), [LineStyleListener](#), [LocationListener](#), [MenuItemListener](#), [ModifyListener](#),
[MouseListener](#), [MouseMoveListener](#), [MouseTrackListener](#), [OpenWindowListener](#), [PaintListener](#),
[ProgressListener](#), [SelectionListener](#), [ShellListener](#), [StatusTextListener](#), [TextChangedListener](#),
[TitleListener](#), [TraverseListener](#), [TreeListener](#), [VerifyKeyListener](#), [VerifyListener](#),
[VisibilityWindowListener](#)

All Known Implementing Classes:
[AccessibleAdapter](#), [AccessibleControlAdapter](#), [AccessibleTextAdapter](#), [ControlAdapter](#),
[CTabFolder2Adapter](#), [CTabFolderAdapter](#), [DragSourceAdapter](#), [DropTargetAdapter](#),
[FocusAdapter](#), [KeyAdapter](#), [LocationAdapter](#), [MenuItemAdapter](#), [MouseAdapter](#),
[MouseTrackAdapter](#), [ProgressAdapter](#), [SelectionAdapter](#), [ShellAdapter](#), [TreeAdapter](#),
[VisibilityWindowAdapter](#)