

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

2

אירועים ב SWT כמיקרוקוסמוס של תכנות מונחה עצמים

תוכנה 1 בשפת Java

1

הוספת טיפול באירועים

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

4



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 ();  
    }  
}
```

3

ירוושה 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()`

6

Observer Design Pattern

- דרך הטיפול באירועי GUI היא מקרה פרטי של תבנית עיצוב יסודית בגישת התכנות מונחה העצמים
- הבעיה הכללית מאפיינת Subject אשר מחולל אירועים לוגים (לא בהכרח גראפיים) 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 מיידע את כל מנויו (notify) כל אימת שמתרחש ארוע שיש לו מנויים

5



ירווה 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() {}
}
```

8



ירווה 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 ();
    }
}
```

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 ();
    }
}
```

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 שאילתה מתאימה

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 ();
    }
}
```

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); }
}
```

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)

14



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;  
    }  
}
```

13

טיפול בארועים במחלקה נפרדת

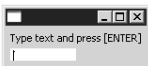
```
public class ButtonHandler  
    implements SelectionListener {  
  
    public void widgetSelected(SelectionEvent e) {  
        if (e.getSource() instanceof Button) {  
            Button b = (Button) e.getSource();  
            b.setText("Thanks!");  
        }  
    }  
  
    public void widgetDefaultSelected(SelectionEvent e){  
        // TODO Auto-generated method stub  
    }  
}
```

16

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 ();  
    }  
}
```

15



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.addListener(new InnerHandler());  
  
        // pack(), open(), while ... Dispose()  
    }  
}
```

18

טיפול בארועים במחלקה נפרדת

• חסרונות:

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

17

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()

20

מחלקה פנימית

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

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

    public class InnerHnadler 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 נגשת לשדות הפרטיים של המחלקה העוטפת

19

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 פרוט של שמות המחלקות השונות שבחבילת SWT

22

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

21

"מאזין לכל ארוע"

org.eclipse.swt.internal
Interface SWTEventListener

All Superinterfaces:
java.util.EventListener

All Known Subinterfaces:

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

All Known Implementing Classes:

[AccessibleAdapter](#), [AccessibleControlAdapter](#), [AccessibleTextAdapter](#), [ControlAdapter](#), [CTabFolder2Adapter](#), [CTabFolderAdapter](#), [DragSourceAdapter](#), [DropTargetAdapter](#), [FocusAdapter](#), [KeyAdapter](#), [LocationAdapter](#), [MenuAdapter](#), [MouseAdapter](#), [MouseTrackAdapter](#), [ProgressAdapter](#), [SelectionAdapter](#), [ShellAdapter](#), [TextAdapter](#), [TitleAdapter](#), [VisibilityWindowAdapter](#)

24

"SWT - הפקת ארועים"

EventObject
↳ AccessibleControlEvent
↳ AccessibleEvent
↳ AccessibleTextEvent
↳ ImageLoadEvent
↳ TypedEvent
↳ ArmEvent
↳ BidirectionalEvent
↳ ControlEvent
↳ CTabFolderEvent
↳ DisposeEvent
↳ DragSourceEvent
↳ DropTargetEvent
↳ ExtendedModifyEvent
↳ FocusEvent
↳ HelpEvent
↳ KeyEvent
↳ TraverseEvent
↳ VerifyEvent
↳ LineBackgroundEvent
↳ LineStyleEvent
↳ LocationEvent
↳ MenuEvent
↳ ModifyEvent
↳ MouseEvent
↳ PaintEvent
↳ ProgressEvent
↳ SelectionEvent
↳ TreeEvent
↳ ShellEvent
↳ StatusTextEvent
↳ TextChangeEvent
↳ TextChangeListener
↳ TitleEvent
↳ WindowEvent

23