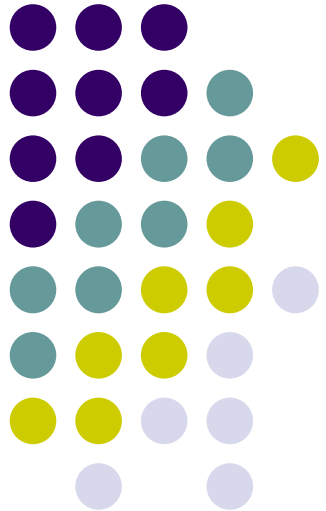
ארועים ב AWT

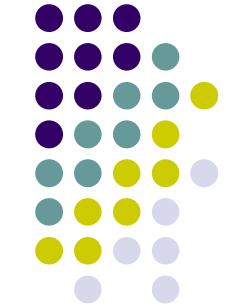
(AWT כמיקרוקוסמוס של תכנות מונחה עצמים)



תכנות מתקדם בשפת Java

אוהד ברזילי

אוניברסיטת תל אביב

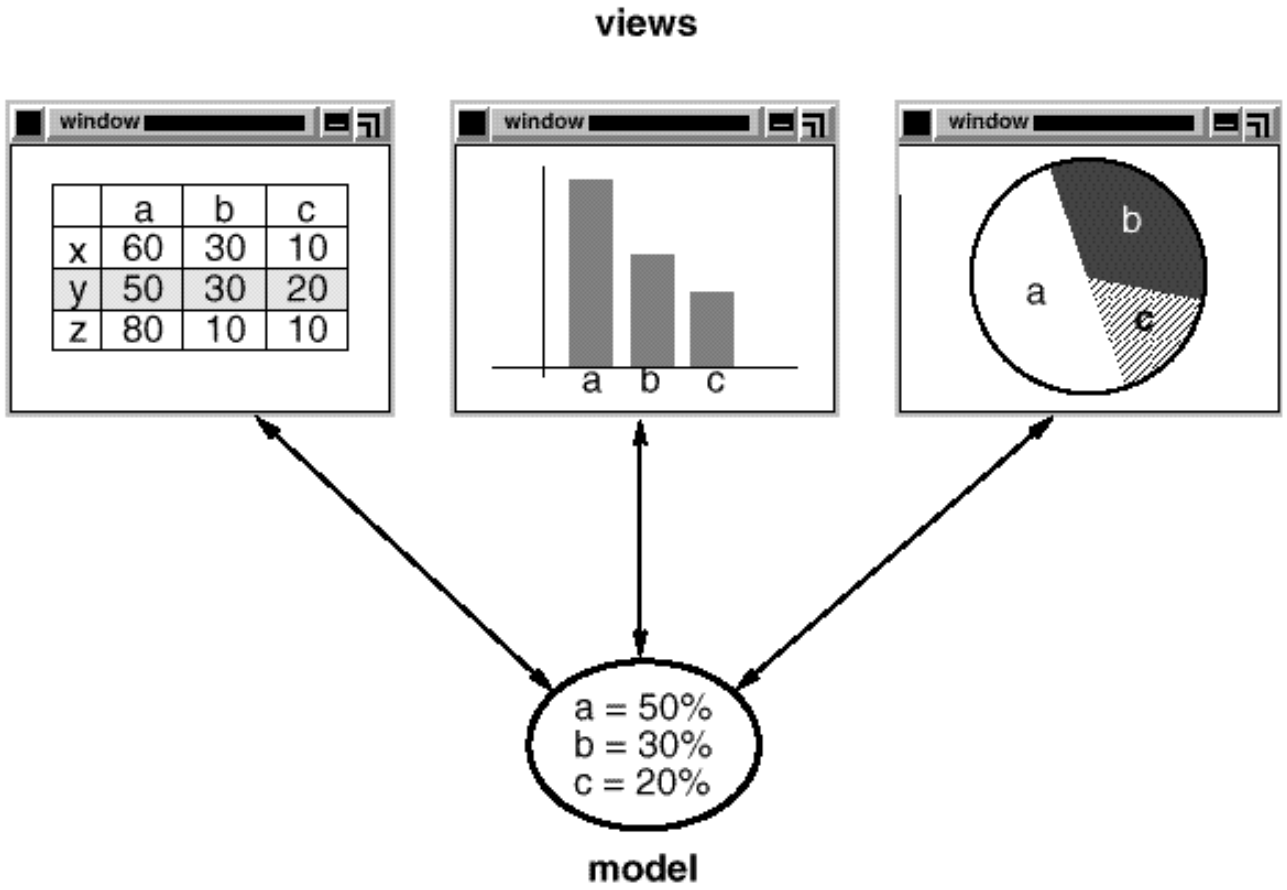


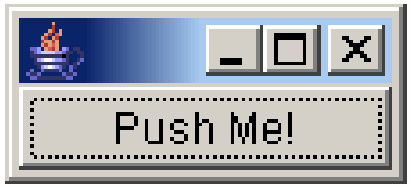
GUI ו- OO

- מערכות ה- GUI המודרניות נחשבות ל killer application של הגישה מונחית העצמים

- מאוד טבעי ואינטואיטיבי לדבר על יסודות OO כגון ירושה, הכלה, האצלה, הפרדת ההצגה והמודל, הסתרת מידע ואחרים בהקשר של GUI

Model View Separation





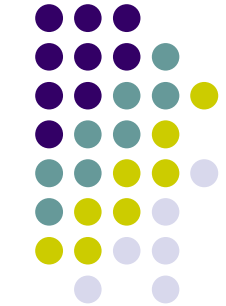
כפתור

```
public class FrameWithButton {

   public static void main(String[] args) {
       Frame f = new Frame();
       Button ok = new Button ("Push Me!");
       f.add(ok);
       f.pack();
       f.setVisible(true);
   }
}
```

הוספת טיפול בארועים

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

Observer Design Pattern



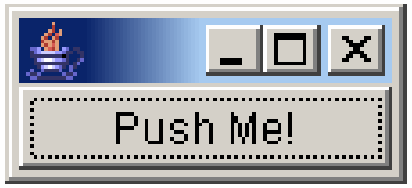
- דרך הטיפול בארועי GUI היא מקרה פרטי של תבנית עיצוב יסודית בגישת התכנות מונחה העצמים
- הבעיה הכללית מאפיינת Subject אשר מחולל ארועים לוגים (לא בהכרח גרפיים) וישויות אחרות במערכת, Observers, אשר מעוניינות לקבל חיווי על כך
- לצורך כך ה Observers נרשמים כמנויים (subscribers) על הארוע הלוגי אצל ה Subject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

ירושה מכפתור

- גישה מקובלת ב- AWT וב- Swing היא הגדרת מחלקה שתירש מכפתור ותממש את המנשק הדרוש

- למרות שהגישה נפוצה מאוד, יש לה מתנגדים מכמה סיבות:
  - הירושה נותנת כוח רב מדי למתכנתת שאינה בקיאה בפרטי הרכיבים השונים
  - אינפלציה של מחלקות

- ב SWT למשל (ספריית GUI שבשימוש Eclipse), החליטו שלא לאפשר ירושה מרכיבי GUI סטנדרטים



ירושה מכפתור

```
public class TestExtendedButton {
   public static void main(String[] args) {
        Frame f = new Frame();
        ExtendedButton ok =
                          new ExtendedButton ("Push Me!");
        ok.addActionListener(ok);
        f.add(ok);
        f.pack();
        f.setVisible(true);
   }
}
```

ירושה מכפתור

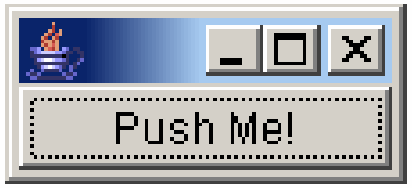
```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ExtendedButton extends Button
                          implements ActionListener{

   public ExtendedButton(String caption) {
       super(caption);
   }

   public void actionPerformed(ActionEvent e) {
       setLabel("Thanks!");
   }
}
```

כפתור מוכל

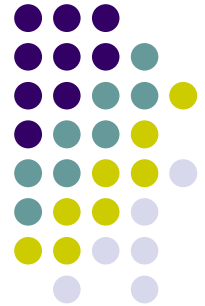
- הגדרת מחלקה אשר תכיל את הכפתור כשדה וגם תממש את לוגיקת הטיפול בארועים פותרת את הצורך לרשת מהמחלקה `Button`

- הכלה (Aggregation) מחייבת את המחלקה החדשה:
  - לאפשר האצלה (delegation) של המתודות של הכפתור
  - לחשוף את הכפתור כלפי חוץ ע"י שאילתה מתאימה

- הערה: חשיפת הכפתור לבדה מספיקה, ואולם האצלה לבדה אינה מספיקה, מכיוון ש Frame יכול להוסיף אליו רכיבים מטיפוס Component

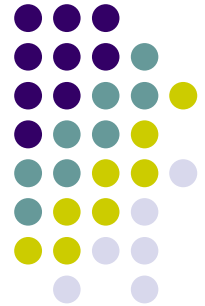


כפתור מוכל האצלה

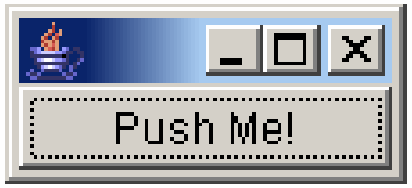


```
public class TestAggregatedButton {
   public static void main(String[] args) {
       Frame f = new Frame();
       AggregatedButton ok =
                    new AggregatedButton ("Push Me!");
       ok.addActionListener(ok);
       f.add(ok);   // Compilation error
                    // can add only Component type

       f.pack();
       f.setVisible(true);
   }
}
```

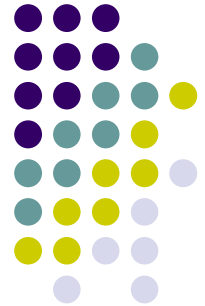
כפתור מוכל האצלה



```
public class AggregatedButton implements ActionListener{

   private Button b;

   public AggregatedButton(String caption) {
       b = new Button(caption);
   }

   public void actionPerformed(ActionEvent e) {
       b.setLabel("Thanks!");
   }

   public void addActionListener(AggregatedButton ok) {
       b.addActionListener(ok);
   }
}
```

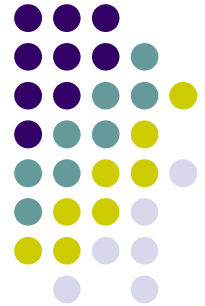


כפתור מוכל
חשיפת הכפתור

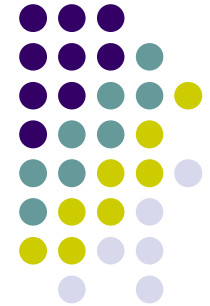


```
public class TestAggregatedButton {
   public static void main(String[] args) {
       Frame f = new Frame();
       AggregatedButton ok =
                   new AggregatedButton ("Push Me!");
       ok.getButton().addActionListener(ok);
       f.add(ok.getButton());
       f.pack();
       f.setVisible(true);
   }
}
```

כפתור מוכל
חשיפת הכפתור



```
public class AggregatedButton
                    implements ActionListener {

   private Button b;

   public AggregatedButton(String caption) {
       b = new Button(caption);
   }

   public void actionPerformed(ActionEvent e) {
       b.setLabel("Thanks!");
   }

   public Button getButton() {
       return b;
   }
}
```



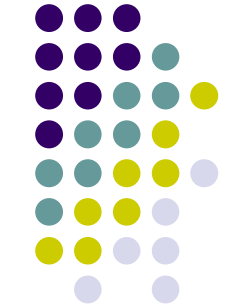
טיפול בארועים במחלקה נפרדת

- 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)

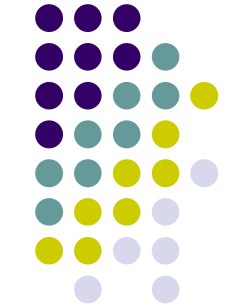
טיפול בארועים במחלקה נפרדת

```
public class TestButtonHandler {
   public static void main(String[] args) {
       Frame f = new Frame();
       Button ok = new Button ("Push Me!");
       ok.addActionListener(new ButtonHandler());
       f.add(ok);
       f.pack();
       f.setVisible(true);
   }
}
```

טיפול בארועים במחלקה נפרדת

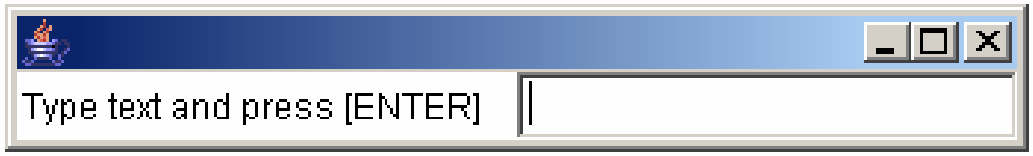


```
public class ButtonHandler implements ActionListener {

   public void actionPerformed(ActionEvent e) {
       if (e.getSource() instanceof Button) {
            Button b = (Button) e.getSource();
            b.setLabel("Thanks!");
       }
   }
}
```

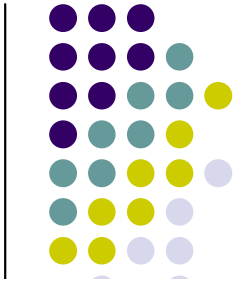


טיפול בארועים במחלקה נפרדת

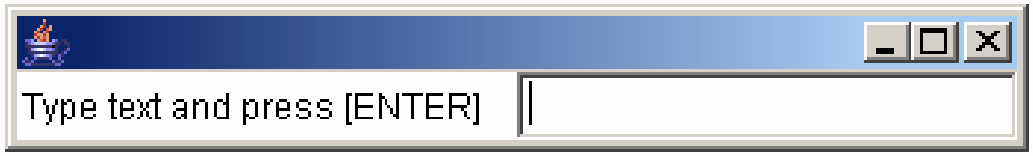
- חסרונות:
  - אם מחלקה נפרדת מטפלת במגוון גדול של ארועים המגיעים ממקורות (רכיבים) שונים בטיפוסם אנו נגררים לבדיקות טיפוס (instanceof) במקום להשתמש בפולימורפיזם
  - לעיתים הטיפול בארוע דורש הכרות אינטימית עם המקור שיצר את הארוע (כדי להימנע מחשיפת המבנה הפנימי של המקור)
- שימוש במחלקה פנימית יוצר את האינטימיות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



מחלקה פנימית



```
public class FrameWithLabelAndTextField {

   private Label l;
   private TextField t;

   public static void main(String[] args) {
        FrameWithLabelAndTextField frame =
                                  new FrameWithLabelAndTextField();
        frame.createFrame();
   }

   public void createFrame() {
        Frame f = new Frame();
        f.setLayout(new GridLayout(1,2););

        l = new Label ("Type text and press [ENTER]");
        f.add(l);

        t = new TextField();
        t.addKeyListener(new InnerHnadler());
        f.add(t);
   }
```

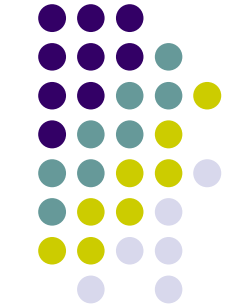


מחלקה פנימית

```
public class FrameWithLabelAndTextField {
    ...
    public class InnerHnadler implements KeyListener {

        public void keyPressed(KeyEvent e) {
                if(e.getKeyChar() == KeyEvent.VK_ENTER){
                        l.setText(t.getText());
                        t.setText("");
                }
        }

        public void keyReleased(KeyEvent arg0) {
                // TODO Auto-generated method stub
        }

        public void keyTyped(KeyEvent arg0) {
                // TODO Auto-generated method stub
        }
    }
}
```

המחלקה הפנימית ניגשת לשדות הפרטיים של המחלקה העוטפת

מחלקה פנימית אנונימית



```
public class FrameWithLabelAndTextField {
   // same as before…
   public void createFrame() {
        ...
        t.addKeyListener(new KeyListener() {

                public void keyPressed(KeyEvent e) {
                        if(e.getKeyChar() == KeyEvent.VK_ENTER){
                                l.setText(t.getText());
                                t.setText("");
                        }
                }

                public void keyReleased(KeyEvent arg0) {}
                public void keyTyped(KeyEvent arg0) {}
                } );
        f.add(t);
        f.pack ();
        f.setVisible(true);
   }
}
```

סוגר סוגריים של המתודה addKeyListener()

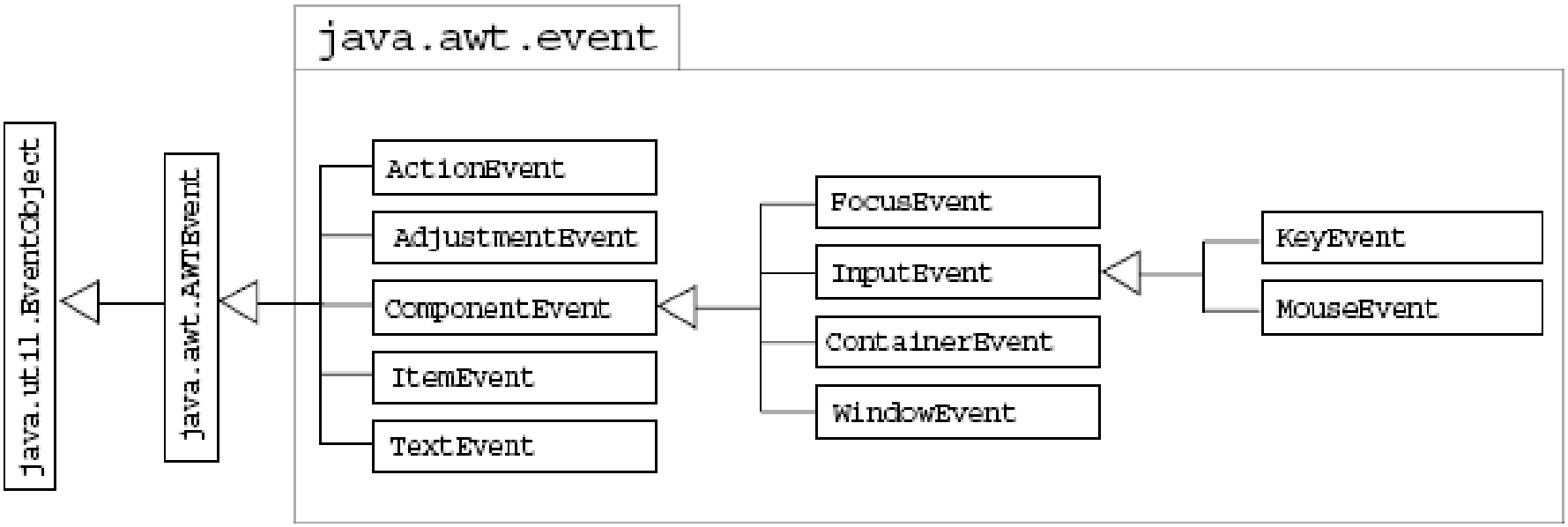
מחלקות פנימיות - דיון

- 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

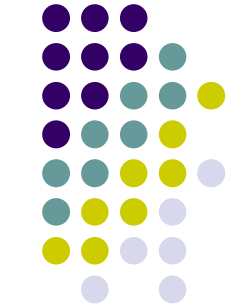
ארועים ומאזינים ב AWT

- AWT מכיל מגוון רחב 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 AWT

ארועים ב AWT


java.awt.event
java.util.EventObject
java.awt.AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
FocusEvent
InputEvent
ContainerEvent
WindowEvent
KeyEvent
MouseEvent

ארועים ב AWT



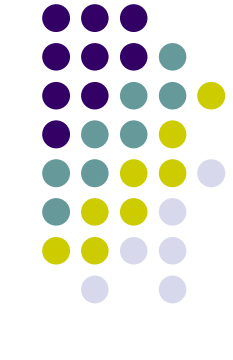
Category	Interface Name	Methods
Action	`ActionListener`	`actionPerformed(ActionEvent)`
Item	`ItemListener`	`itemStateChanged(ItemEvent)`

Mouse	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseClicked(MouseEvent)`

Mouse motion	`MouseMotionListener`	`mouseDragged(MouseEvent)` `mouseMoved(MouseEvent)`

Key	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

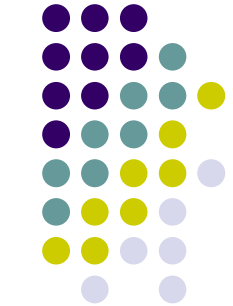
ארועים ב AWT



Category	Interface Name	Methods
Focus	`FocusListener`	`focusGained(FocusEvent)` `focusLost(FocusEvent)`
Adjustment	`AdjustmentListener`	`adjustmentValueChanged` `(AdjustmentEvent)`

Component	`ComponentListener`	`componentMoved(ComponentEvent)`
		`componentHidden(ComponentEvent)`
		`componentResized(ComponentEvent)`
		`componentShown(ComponentEvent)`

ארועים ב AWT



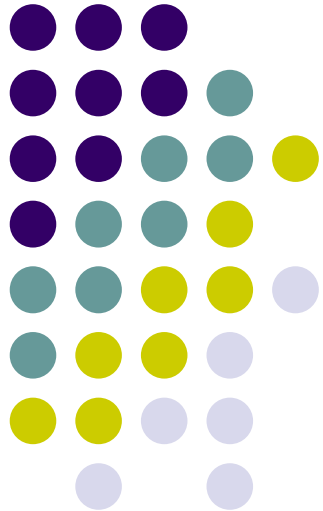
Category	Interface Name	Methods

Window	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

Container	`ContainerListener`	`componentAdded(ContainerEvent)` `componentRemoved(ContainerEvent)`
Text	`TextListener`	`textValueChanged(TextEvent)`

דוגמאות

רכיבים וארועים ב AWT



Checkbox

```
f = new Frame("Sample Checkbox");
Checkbox one = new Checkbox("One", true);
Checkbox two = new Checkbox("Two", false);
Checkbox three = new Checkbox("Three", false);

 one.addItemListener(this);
 two.addItemListener(this);
 three.addItemListener(this);

 f.setLayout(new FlowLayout());
 f.add(one);
 f.add(two);
 f.add(three);
```

Checkbox

```
public void itemStateChanged(ItemEvent ev) {
   String state = "deselected";
   if (ev.getStateChange() == ItemEvent.SELECTED) {
       state = "selected";
   }
   System.out.println(ev.getItem() + " " + state);
}
```

CheckboxGroup

```
Frame f = new Frame("CheckBoxGroup");
CheckboxGroup cbg = new CheckboxGroup();
Checkbox one = new Checkbox("One", cbg, false);
Checkbox two = new Checkbox("Two", cbg, false);
Checkbox three = new Checkbox("Three", cbg, true);

f.setLayout(new FlowLayout());

one.addItemListener(this);
two.addItemListener(this);
three.addItemListener(this);

f.add(one);
f.add(two);
f.add(three);
```

Choice

```
f = new Frame("Sample Choice");
choice = new Choice();
choice.addItem("First");
choice.addItem("Second");
choice.addItem("Third");
choice.addItemListener(this);
f.add(choice, BorderLayout.CENTER);
```

List

```
List lst = new List(4, true);
lst.add("Hello");
lst.add("there");
lst.add("how");
```

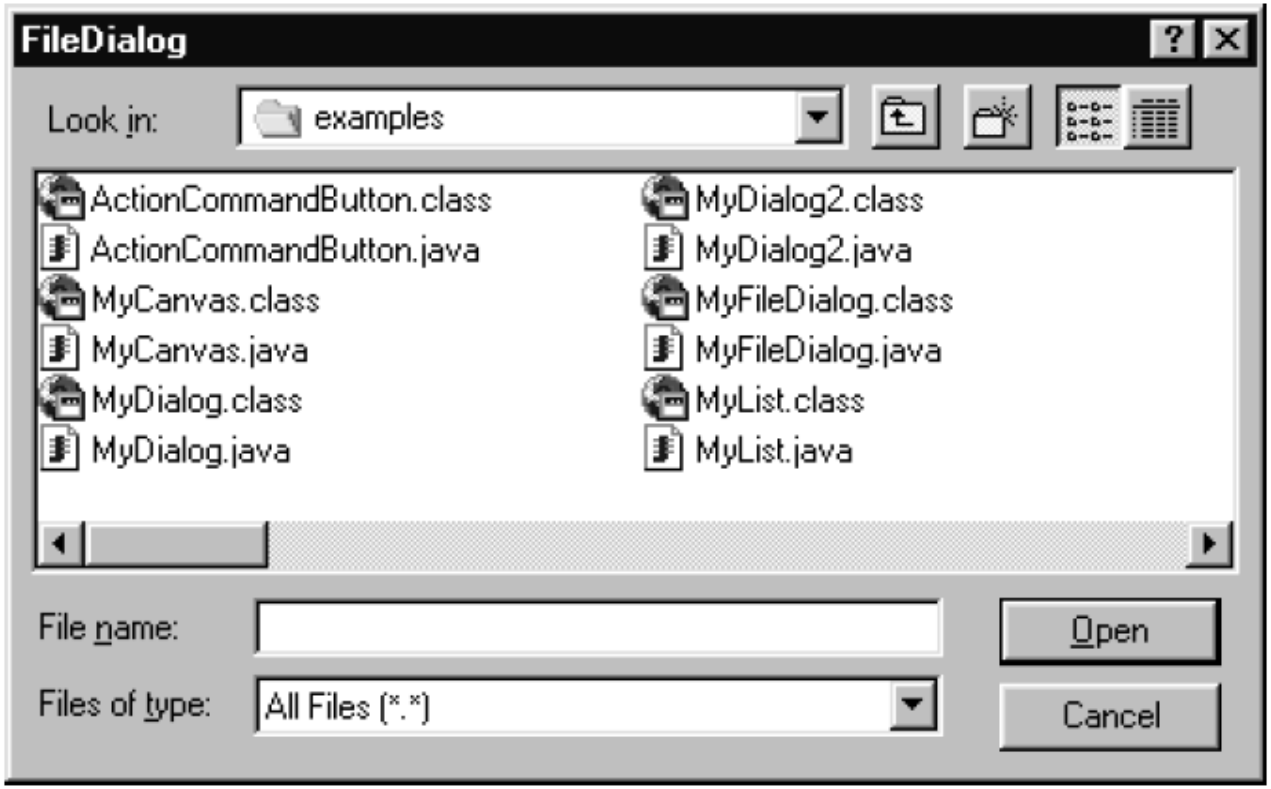
Dialog

```
d = new Dialog(f, "Dialog", true);
d.setLayout(new GridLayout(2,1));
dl = new Label("Hello, I'm a Dialog");
db1 = new Button("OK");
d.add(dl);
d.add(db1);
d.pack();
```


```
public void actionPerformed(ActionEvent ev) {
   d.setVisible(true);
}
```

FileDialog

```
FileDialog d = new FileDialog(parentFrame, "FileDialog");
d.setVisible(true); // block here until OK selected
String fname = d.getDirectory() + d.getFile();
```

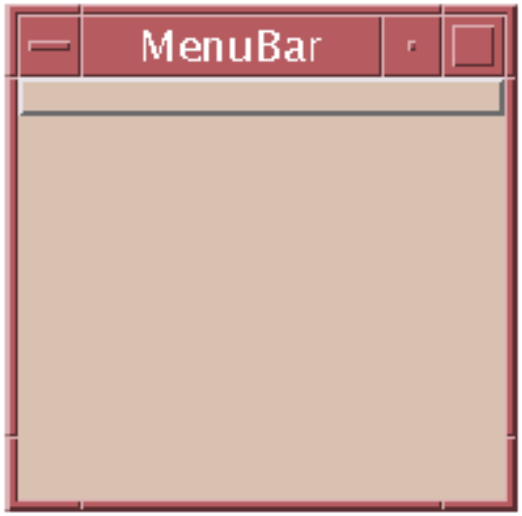


ScrollPane

```
Frame f = new Frame("ScrollPane");
Panel p = new Panel();
ScrollPane sp = new ScrollPane();
p.setLayout(new GridLayout(3, 4));
// ...
sp.add(p);
f.add(sp, BorderLayout.CENTER);
f.setSize(100, 100);
f.setVisible(true);
```

MenuBar

```
Frame f = new Frame("MenuBar");
MenuBar mb = new MenuBar();
f.setMenuBar(mb);
```

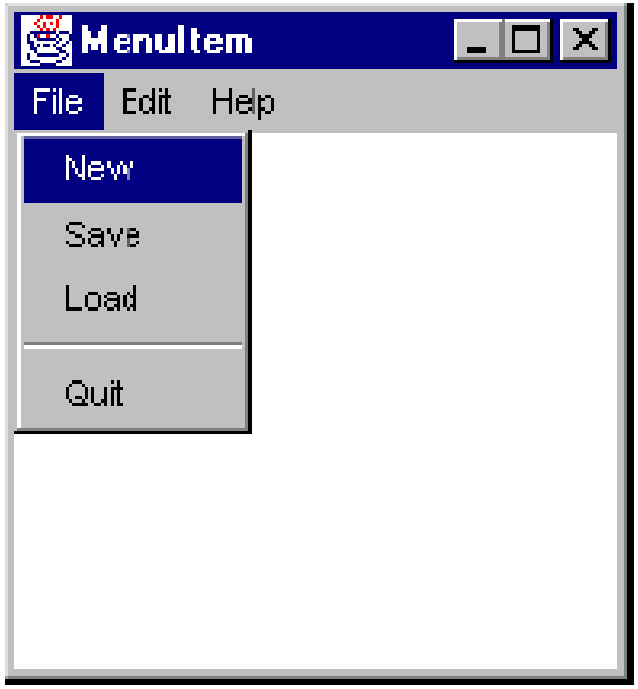


Menu

```
Frame f = new Frame("Menu");
MenuBar mb = new MenuBar();
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);
```

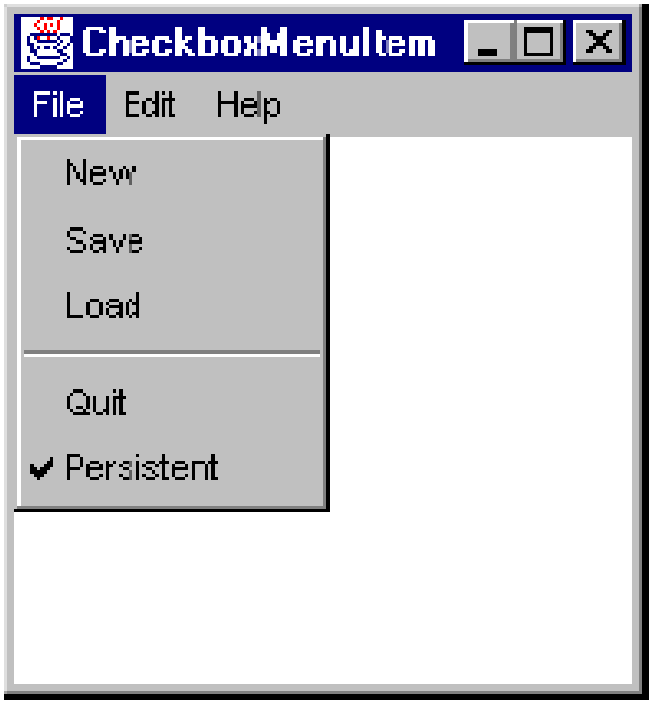
MenuItem

```
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Save");
MenuItem mi3 = new MenuItem("Load");
MenuItem mi4 = new MenuItem("Quit");
mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi4.addActionListener(this);
m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.addSeparator();
m1.add(mi4);
```



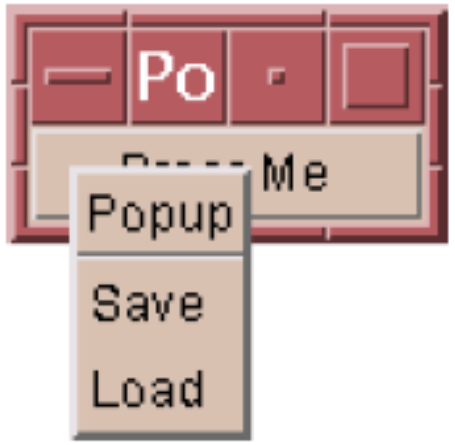
CheckBoxMenuItem

```
MenuBar mb = new MenuBar();
Menu m1 = new Menu("File");
Menu m2 = new Menu("Edit");
Menu m3 = new Menu("Help");
mb.add(m1);
mb.add(m2);
mb.setHelpMenu(m3);
f.setMenuBar(mb);
.....
MenuItem mi2 = new MenuItem("Save");
mi2.addActionListener(this);
m1.add(mi2);
......
CheckboxMenuItem mi5 = new CheckboxMenuItem("Persistent");
mi5.addItemListener(this);
m1.add(mi5);
```



PopupMenu

```
Frame f = new Frame("PopupMenu");
Button b = new Button("Press Me");
PopupMenu p = new PopupMenu("Popup");
MenuItem s = new MenuItem("Save");
MenuItem ld = new MenuItem("Load");
b.addActionListener(this);
f.add(b,BorderLayout.CENTER);
p.add(s);
p.add(ld);
f.add(p);
```




```
public void actionPerformed(ActionEvent ev) {
   // display popup at (10,10) relative to b
   p.show(b, 10, 10);
}
```