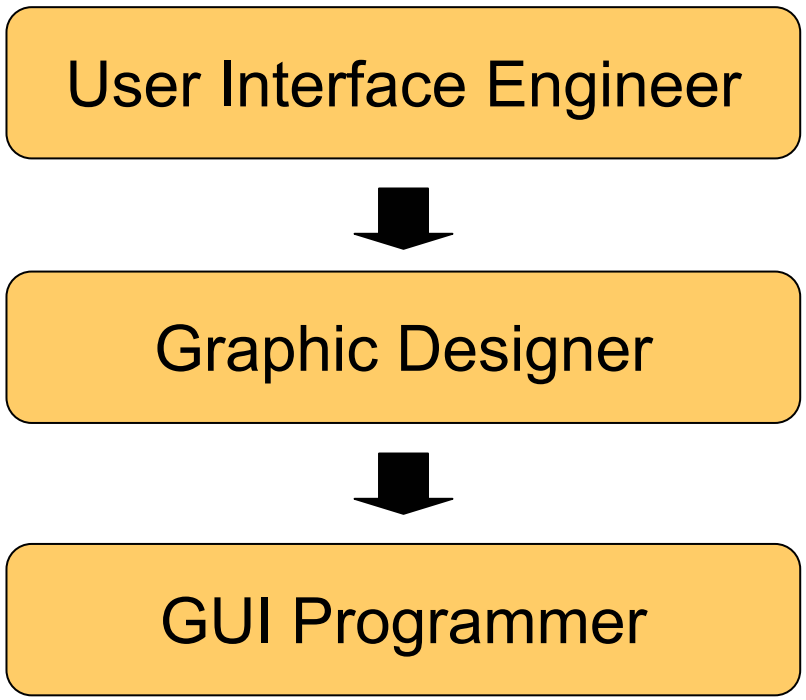
Software 1

Recitations No. 11-12:
SWT GUI Package

The GUI Development Process

- **GUI**: **G**raphical **U**ser **I**nterface



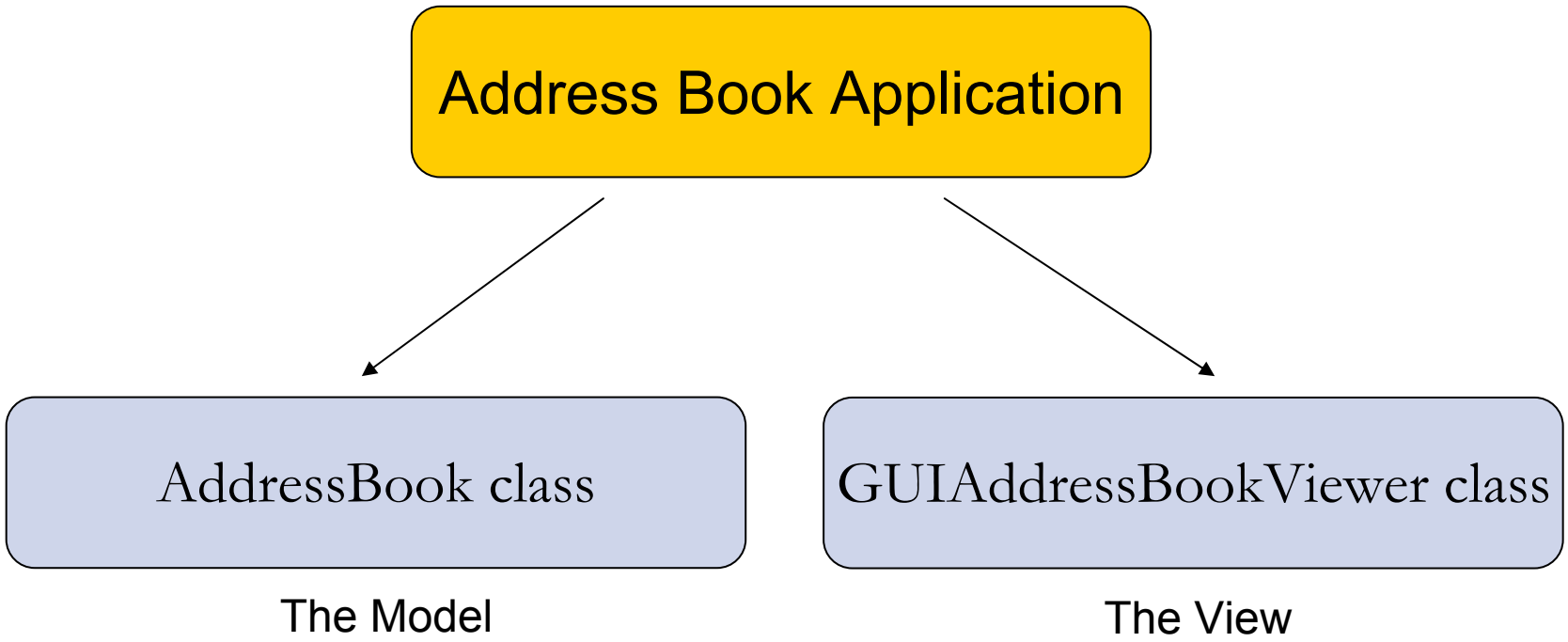
GUI Application

- When implementing a GUI application one should specify:
  - the GUI elements
  - the 2D arrangement of the GUI elements
  - the behavior of the GUI elements
- Java GUI libraries:
  - AWT (**A**bstract **W**indowing **T**oolkit)
  - Swing
  - SWT (**S**tandard **W**idget **T**oolkit)

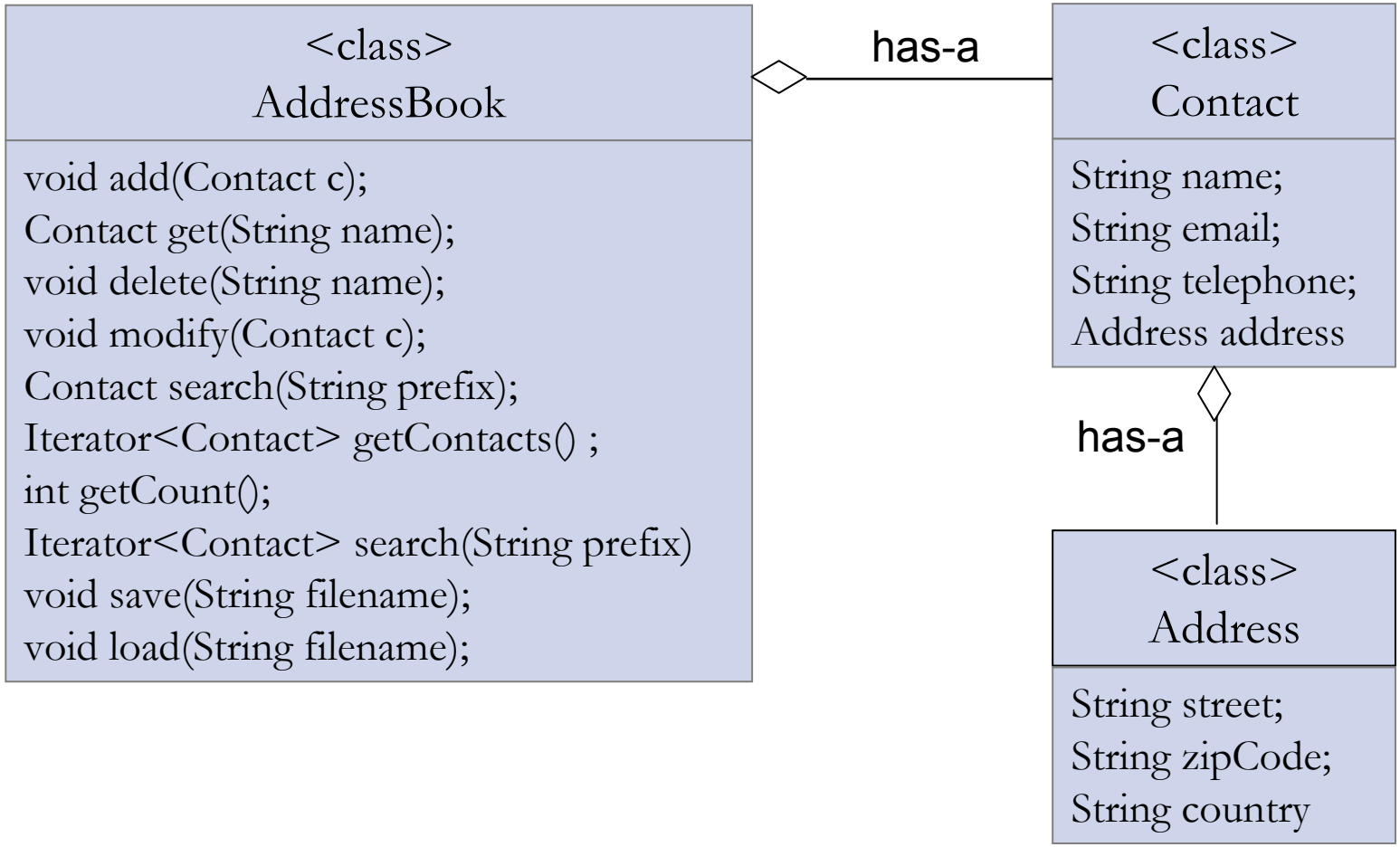
Model-View Separation

- Separate between the application logic (***model***) part and the GUI (***view***) part.
- Ensures that view changes have no effect on the basic model
- Enables us to maintain one model for several different views

Example: Address Book



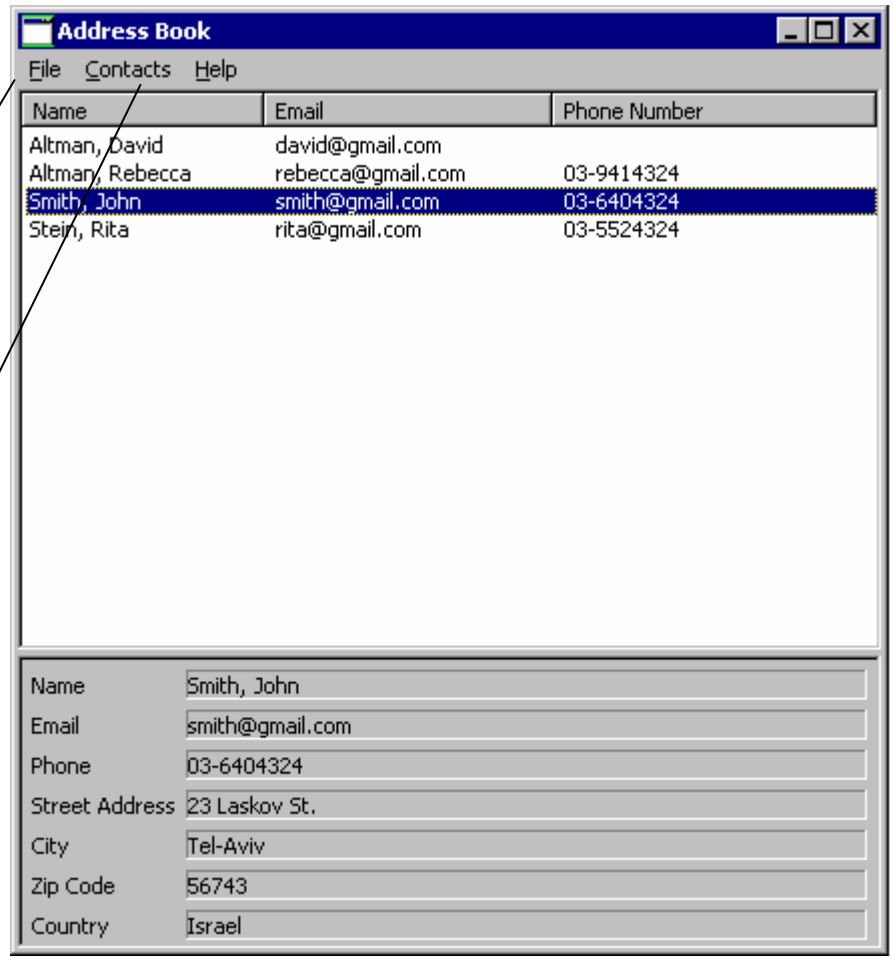
The Model


<class>
AddressBook
void add(Contact c);
Contact get(String name);
void delete(String name);
void modify(Contact c);
Contact search(String prefix);
Iterator<Contact> getContacts() ;
int getCount();
Iterator<Contact> search(String prefix)
void save(String filename);
void load(String filename);
has-a
<class>
Contact
String name;
String email;
String telephone;
Address address
has-a
<class>
Address
String street;
String zipCode;
String country

The View

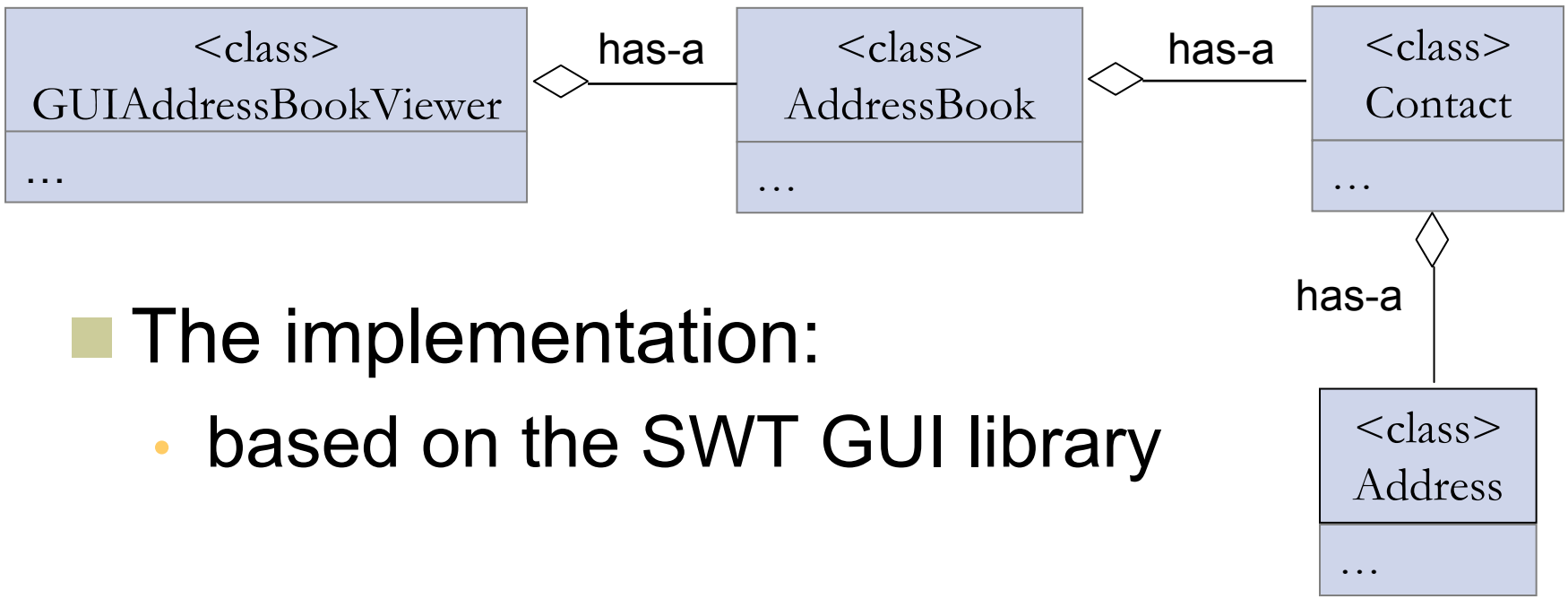
New Address Book Ctrl+N
Open Ctrl+O
Save Ctrl+S

New Contact Ctrl+M
Edit
Delete



The View

- The class diagram:



- The implementation:
  - based on the SWT GUI library

SWT

- Online Documentation:
  - SWT HomePage:

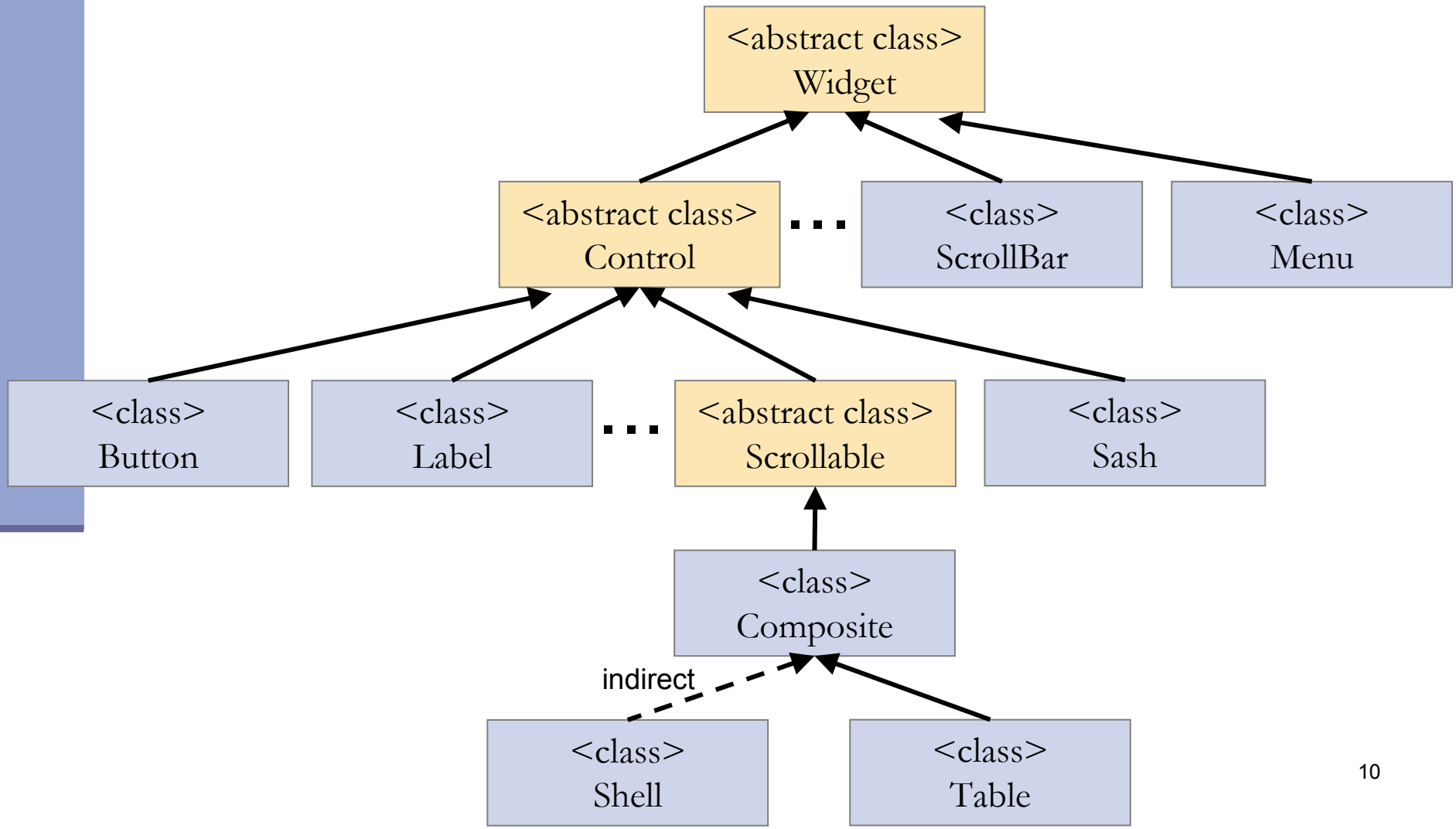
    http://www.eclipse.org/swt/

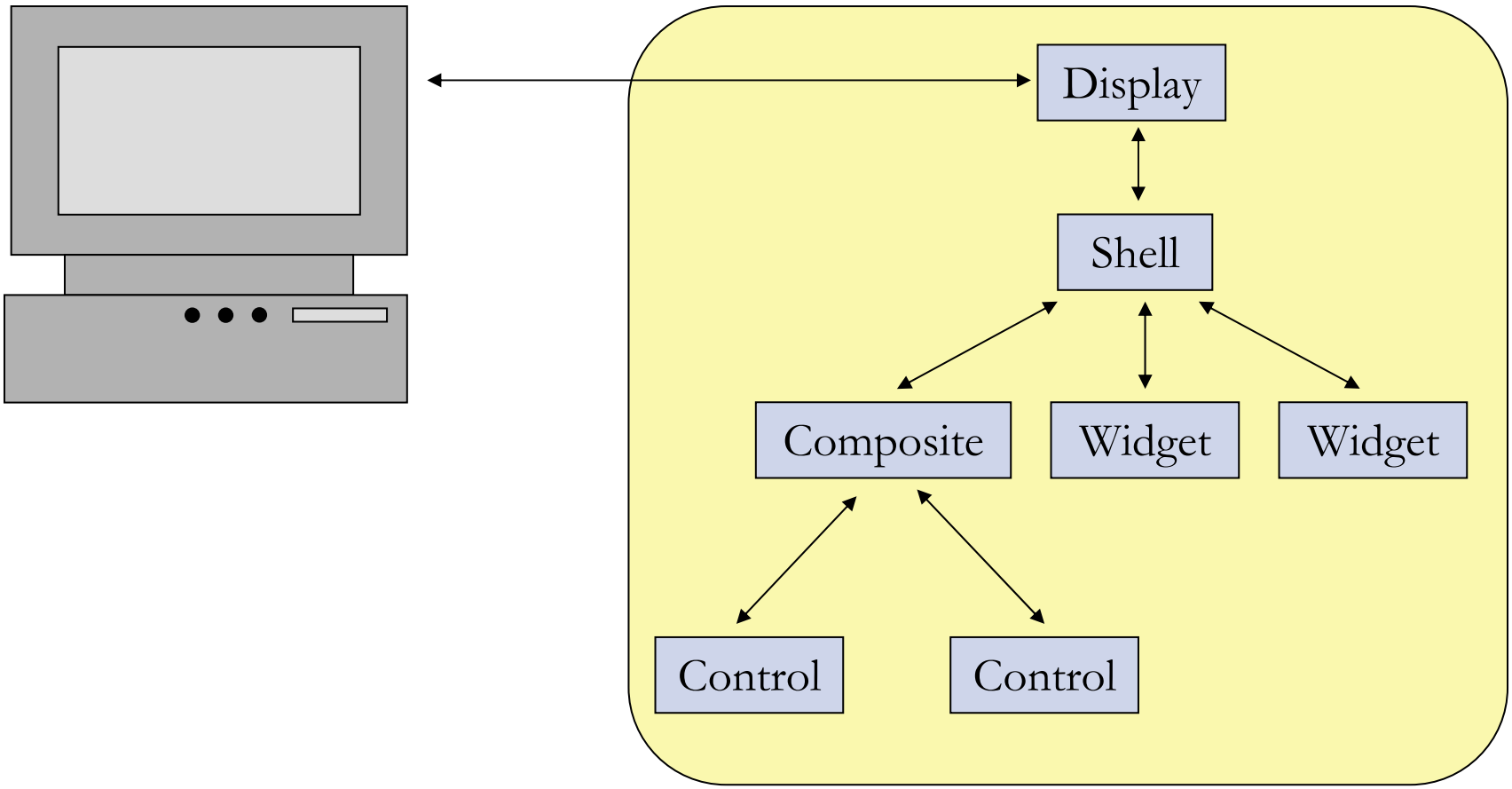
    - JavaDoc

    - Snippets
  - Getting Started with Eclipse and the SWT:

    http://www.cs.umanitoba.ca/~eclipse/

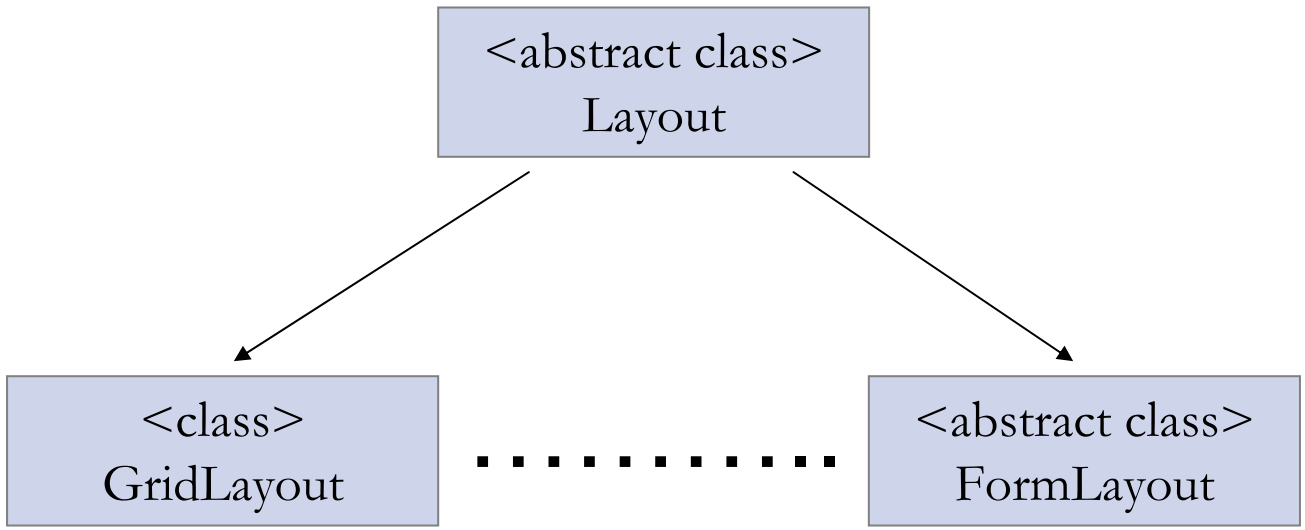
Widgets


<abstract class>
Widget
<abstract class>
Control
...
<class>
ScrollBar
<class>
Menu
<class>
Button
<class>
Label
...
<abstract class>
Scrollable
<class>
Sash
<class>
Composite
indirect
<class>
Shell
<class>
Table


Display
Shell
Composite
Widget
Widget
Control
Control

Layouts

- A Layout controls the position and size of Control widgets in a Composite.



GridLayout

- Lays out the Control widgets in a grid.

Each column is as wide as Wide Button 2

GridLayout

- Lays out the Control widgets in a grid.
- GridLayout Configuration fields:

Field	Default	Description
horizontalSpacing verticalSpacing	5	Horizontal/vertical space between the grid cells
marginHeight marginWidth	5	The size of the horizontal/vertical margins of the layout
numColumns	1	Number of columns
makeColumnsEqualWidth	false	If true, all columns will have the same size

GridLayout (cont.)

GridData:

- Use GridData objects to configure the Control widgets in a GridLayout.
- Use the setLayoutData() to set a GridData object into a Control, e.g.

```
label.setLayoutData(new GridData(…));
```

- Do not reuse GridData objects

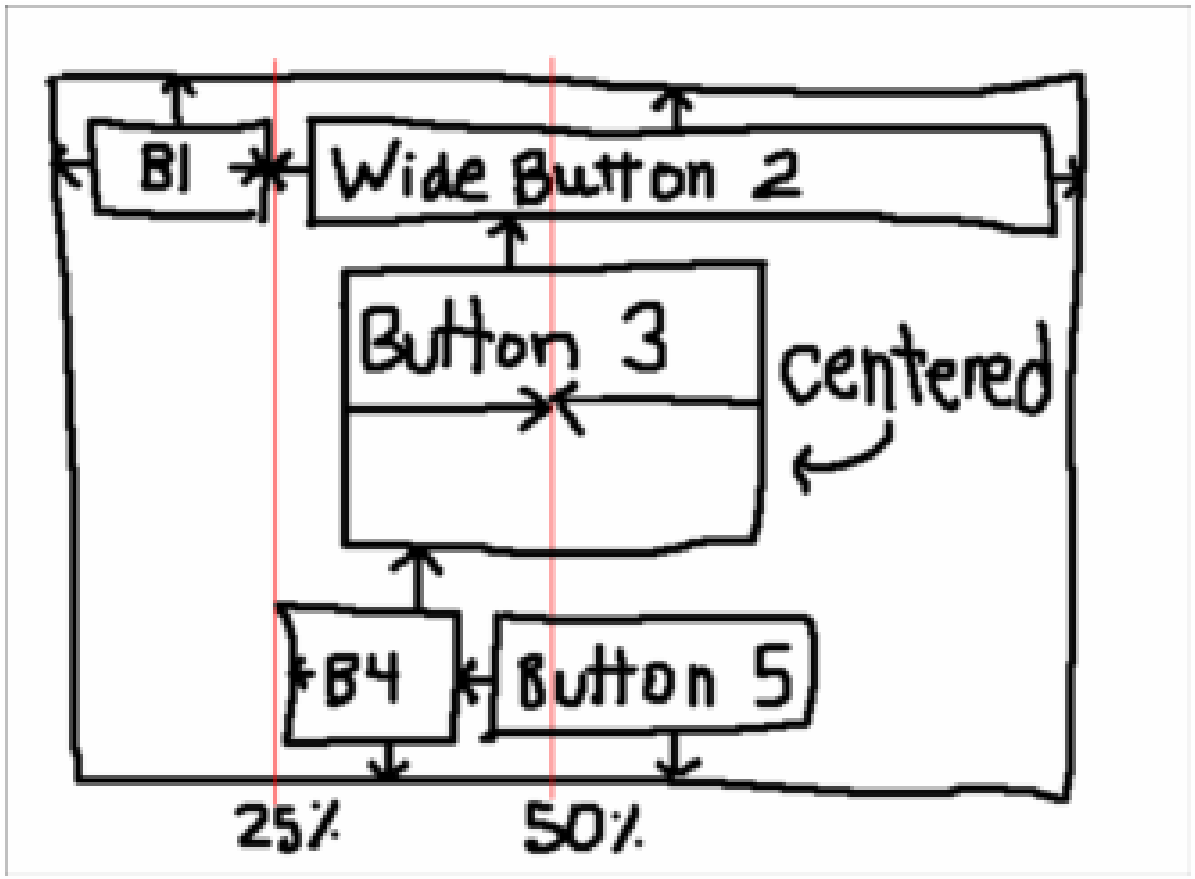
GridLayout (cont.)

- GridData Configuration Fields:

Field	Default	Description
grabExcessHorizontalSpace grabExcessVerticalSpace	false	If true, the width/length of the widget will be as large as possible to fit the remaining space.
heightHint widthHint	SWT.DEFAULT (no minimum)	A minimum width/height for the widget.
horizontalSpan verticalSpan	1	the number of column/row cells that the widget will take up.
horizontalIndent	0	the number of indentation pixels along the left side of the cell.
horizontalAlignment verticalAlignment	GridData.BEGINNING GridData.CENTER	how controls will be positioned horizontally/vertically within a cell.

FormLayout

- A very flexible layout



FormLayouts

- A very flexible layout
- FormLayout Configuration Properties:

Field	Default	Description
marginHeight marginWidth	0	the margin width/height
spacing	0	the number of pixels between the edge of one control and the edge of its neighbouring control.

FormLayouts (cont.)

- Use FormData objects to configure the Control widgets in a FormLayout.
- Use the setLayoutData() to set a FormData object into a Control widget.
- A FormData object has a FormAttachment object for each edge of the Control.

Field	Description
width/height	the desired width/height in pixels.
top/bottom/left/right	Specifies the position of the control attachment.

FormLayouts (cont.)

- A FormAttachment defines where to attach the side of a Control by using the equation: y = ax + b.

an **offset**, in pixels

A fraction defined by:
-**numerator**
-**denominator**

the width/height of a Control to which the control side is attached (**control**).

FormLayouts (cont.)

- Main FormAttachment Constructors:
  - public FormAttachment(Control control)
  - public FormAttachment(Control control, int offset)
  - public FormAttachment(int numerator)
  - public FormAttachment(int numerator, int offset)

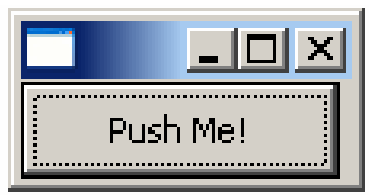
Field	Default
control	Parent Composite
numerator	100
denominator	100
offset	0

$$y = \frac{numerator}{denominnator} \bullet x + offet$$

x – control's width/height

Events

```
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 ();
    }
```



Events

הכפתור לא מגיב ללחיצות.

איך אפשר לטפל בלחיצות:

- הגדרת מחלקה שתירש מכפתור
- מחלקה שתכיל כפתור כאחד משדותיה
- יצירת מחלקה עצמאית שתטפל באירועי הלחיצה (SelectionEvent):
  - על המחלקה המטפלת לממש את המנשק SelectionListener
  - לצורך נוחות: המחלקה SelectionAdopter מספקת מימוש ברירת מחדל

Observer Design Pattern

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

טיפול בארועים במחלקה נפרדת

- יתרונות:
  - ביחס לירושה: הלקוח עובד עם כפתור סטנדרטי ולכן אין צורך לחשוף מבנה פנימי ללקוח
  - ביחס להכלה: 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 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 ();
}
```

טיפול בארועים במחלקה נפרדת

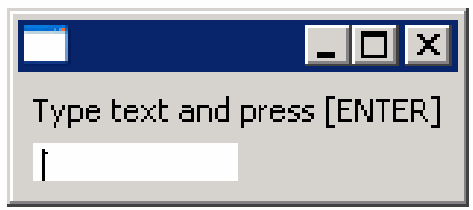
```
public class ButtonHandler implements SelectionListener {
    public void widgetSelected(SelectionEvent e) {
        if (e.getSource() instanceof Button) {
            Button b = (Button) e.getSource();
            b.setText("Thanks!");
        }
    }
}
```

טיפול בארועים במחלקה נפרדת

- חסרון המימוש הקודם:
  - לעיתים הטיפול באירוע דורש הכרות אינטימית עם המקור שיצר את האירוע (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

מחלקה פנימית

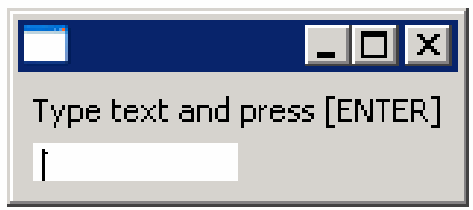
```
public class ShellWithLabelAndTextField {

    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 InnerHnadler());

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



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 ניגשת לשדות הפרטיים של המחלקה העוטפת



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
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