

---

**פיתוח מערכות תוכנה**  
**הפרדה בין מודל ותצוגה**  
**מקרה מבחן לעקרונות רב מימדיים**

אוהד ברזילי

# MVC

## Model View Controller Design Pattern (MVC) ■

רעיון ההפרדה בין מודל לתצוגה אינו ייחודי רק ל GUI ■  
לדוגמא: ■

מה תדפיס השורה: `System.out.println(new Date());` ;  
איך מיוצגת המחלקה Date בזיכרון?

נדון בנושא בכמה הקשרים: ■

עיצוב ספריות GUI ■

רמות הפשטה שונות של רכיבי GUI ■

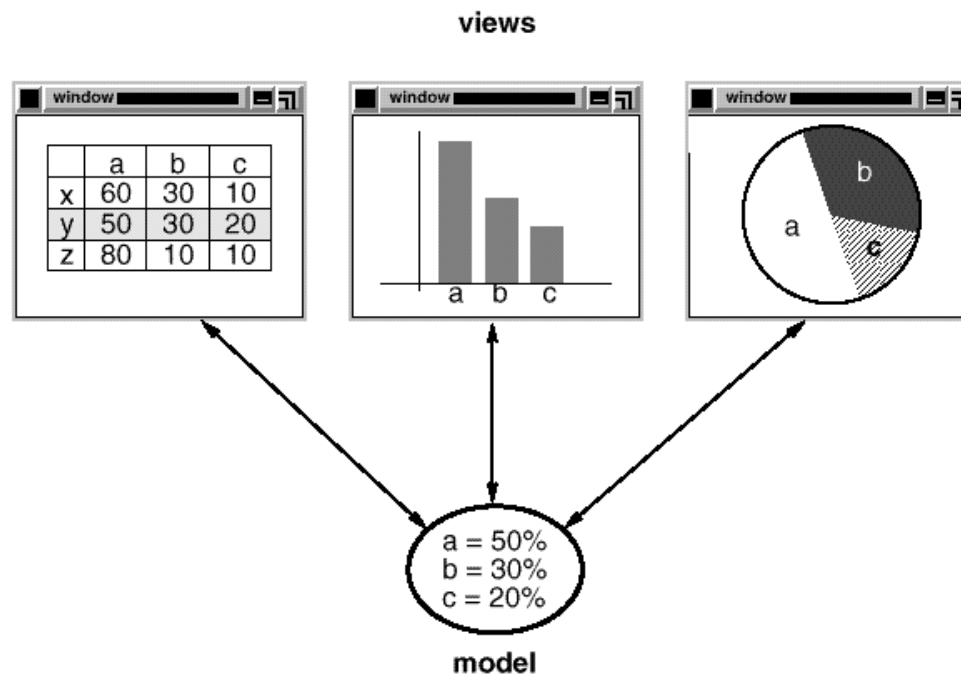
Builder Design Pattern ■

CSS / HTML ■

שפות וסביבות מבוססות MVC ■

# הפרדה בין מודל והצגה

- המודל (הנתונים והלוגיקה של התוכנית) אמור להיות אדיש לשינויים בהצגה (ואולי לאפשר ריבוי הצגות במקביל)
- ספריות GUI רבות נכתבו מתוך הבנת חשיבות ההפרדה



# Abstraction Level

■ איך נתכנן משחק שח גרפי?

■ עשה זאת בעצמך:

■ ציור של משבצות שחור-לבן

■ לכידה של ארועי לחיצה על העכבר

■ שימוש ב widgets:

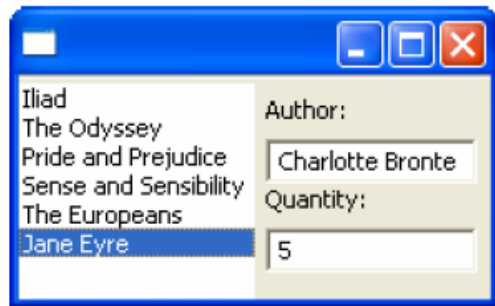
■ בניית סריג של כפתורים ריבועיים בצבעי שחור ולבן  
לסרוגין

■ לכידה של ארועי בחירת כפתור

■ מה היתרונות והחסרונות של כל אחת מהגישות?

# רכיבים המפרידים בין מודל ותצוגה

- חלק מספריות ה GUI מספקות הפרדה בין מודל ותצוגה גם ע"י רכיבים ייעודיים
- דוגמא: Java Beans הן מחלקות Java רגילות אשר יודעות לדווח ל"עולם" על ארועים שונים (fire event)
- דוגמא נוספת:
  - טבלה המצוירת על המסך מכילה מחרוזות ואולם בעצם היא מתארת עצם סמנטי שזהו רק הייצוג שלו
  - היה נחמד לו ניתן היה להציג בצורה אוטומטית Map בתור טבלה ויזואלית



# JFace Viewers

■ החבילה JFace מציעה מגוון מחלקות המציעות שרותי GUI מתקדמים הכתובים מעל הספרייה SWT

■ אחת המשפחות בחבילה מכילה הצגות למבני נתונים שימושיים כגון: `CheckboxTableViewer`, `CheckboxTreeViewer`, `ListViewer`, `TableTreeViewer`, `TableViewer`, `TreeViewer`

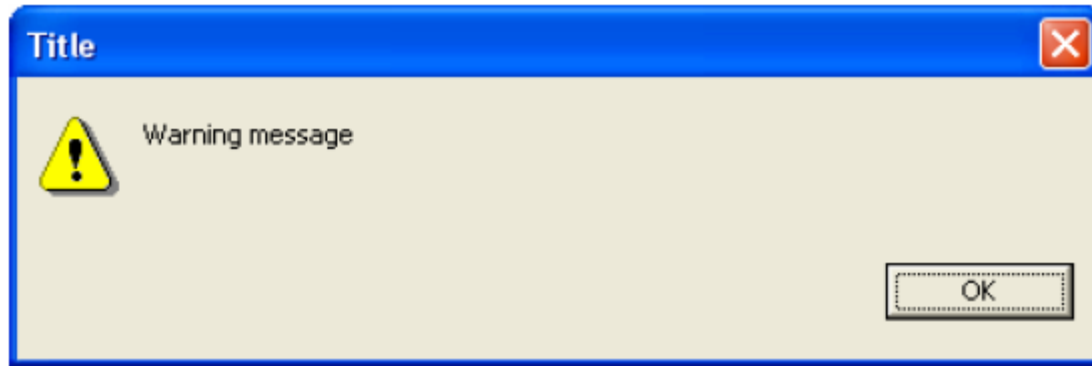
■ למשל, אם ברצוננו להציג למשתמש רשימה של ספרים נרצה לקשור בין רשימת הספרים (עצמים מטיפוס `Book`) ובין רכיב הרשימה היוזואלית

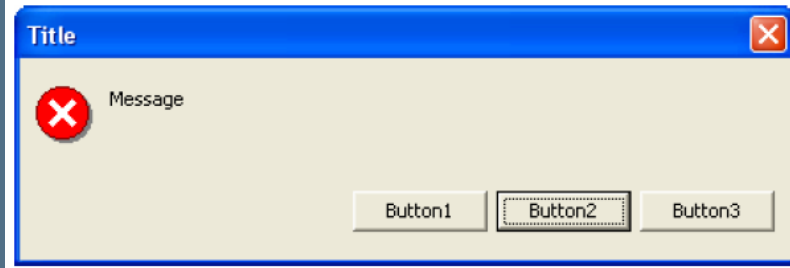
■ לצורך כך יש להגדיר לרשימת הספרים: `LabelProvider` ו- `StructuredContentProvider` (בספריית `Swing Renderer`)

# JFace Dialogs - GUI ללא GUI

- בחבילה JFace ניתן גם למצוא מגוון תיבות דו-שיח לתקשורת עם המשתמש:

```
MessageDialog.openWarning(shell, "Title", "Warning message");
```





# JFace Dialogs

```
String[] buttonText =  
    new String[] { "Button1", "Button2", "Button3" };  
  
MessageDialog messageBox; =  
    new MessageDialog(shell, "Title", null, "Message",  
        MessageDialog.ERROR, buttonText, 1);  
messageBox.open();
```

ניתן להגדיר מספר סוגי תיבות דו-שיח:

MessageDialog.NONE, MessageDialog.ERROR, MessageDialog.INFORMATION,  
MessageDialog.QUESTION, MessageDialog.WARNING

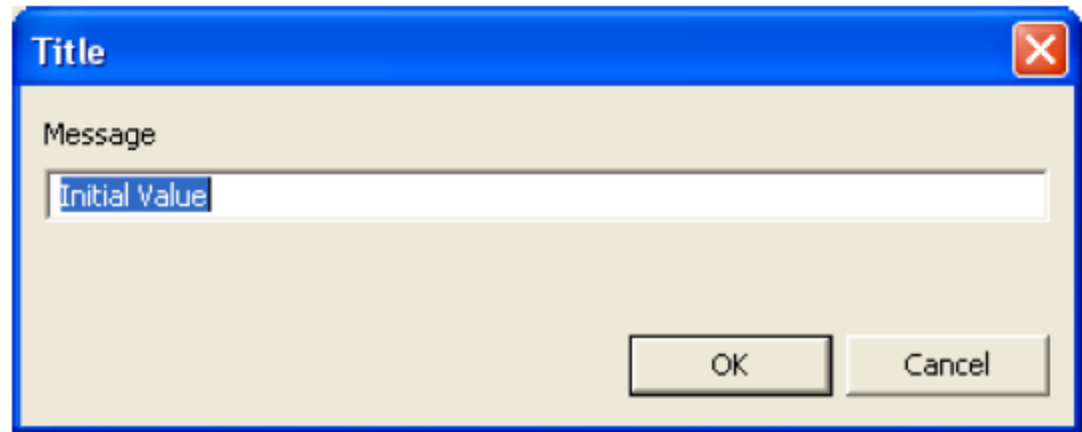
קריאת בחירת המשתמש ע"י:

```
messageBox.getReturnCode();
```



# JFace Dialogs

```
InputDialog inputBox =  
    new InputDialog(shell, "Title", "Message", "Initial Value", null);  
  
inputBox.open();
```

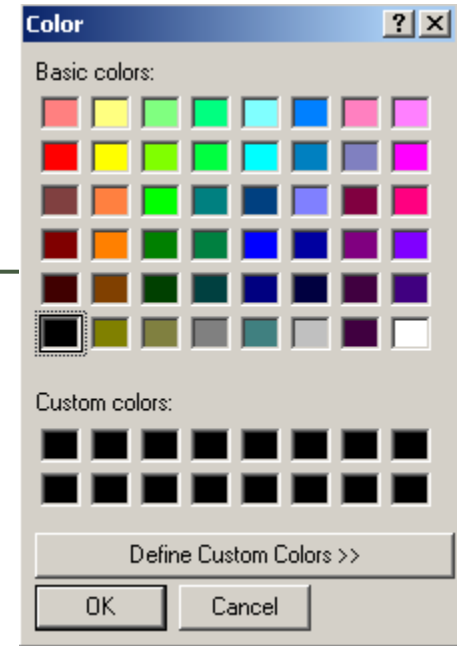


קריאת קלט משתמש ע"י:

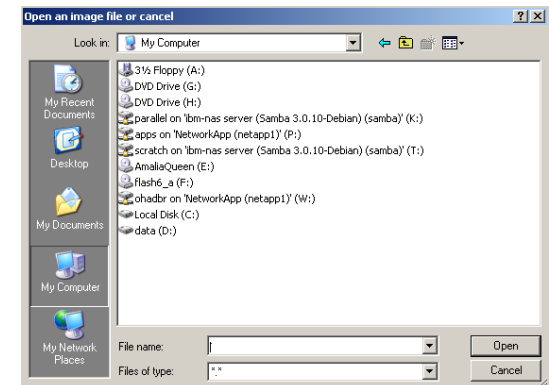
```
inputBox.getReturnCode();  
inputBox.getValue();
```

# JFace Dialogs

```
ColorDialog d = new ColorDialog(shell);  
RGB selection = d.open();
```



```
FileDialog dialog = new FileDialog (shell, SWT.OPEN);  
dialog.setText ("Open an image file or cancel");  
String string = dialog.open ();
```



פיתוח מערכות תוכנה  
אוהד ברזילי

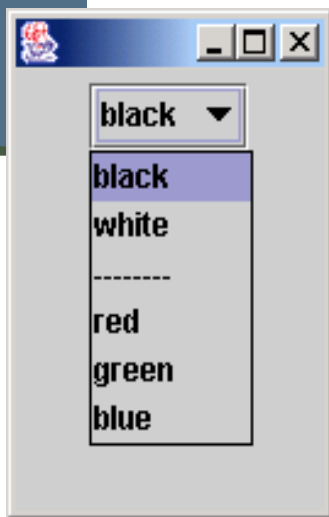
# המודל של ה View

לפעמים (בדר"כ) גם לתצוגה יש מודל פנימי משל עצמה

כאשר אנו מחליפים טכנולוגיה אנו משכפלים מודל זה שלא לצורך

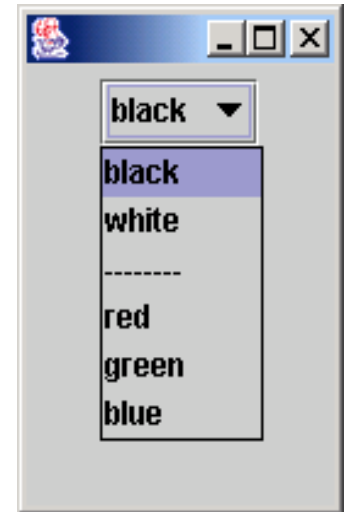
תבנית העיצוב Builder נותנת אפשרות להבעה מפורשת של המודל הפנימי

לדוגמא: איך נתכנן את תפריטי הבחירה הבאים:



```
abstract class Builder {  
    public abstract void addPart(String choiceStr);  
    public abstract void addSeparator();  
    public abstract Component getResult();  
}
```

```
class ComboBuilder extends Builder {  
  
    private JComboBox combo = new JComboBox();  
  
    public void addPart(String choiceStr) {  
        combo.addItem(choiceStr);  
    }  
  
    public void addSeparator() {  
        combo.addItem("-----");  
    }  
  
    public Component getResult() { // Return the Complex object  
        return combo;  
    }  
}
```



```

class RadioButtonBuilder extends Builder {

    private Box panel = Box.createVerticalBox();
    private ButtonGroup group = new ButtonGroup();

    public void addPart(String choiceStr) {
        JRadioButton bt = new JRadioButton(choiceStr);
        group.add(bt);
        panel.add(bt);
    }

    public void addSeparator() {
        panel.add(new JSeparator());
    }

    public Component getResult() { // Return the Complex object
        // select first radio button
        ((JRadioButton)group.getElements().nextElement())
            .setSelected(true);
        return panel;
    }
}

```



```

class ListDirector {

    public Component create(Builder builder, String[] choiceStrings){
        for(int i=0; i<choiceStrings.length; i++){
            String s = choiceStrings[i];
            if (s==null)
                builder.addSeparator();
            else
                builder.addPart(s);
        }
        return builder.getResult();
    }
}

```

### Usage:

```
ListDirector director = new ListDirector();
```

```
String[] choiceStrings = {
    "black", "white", null, "red", "green", "blue"
};
```

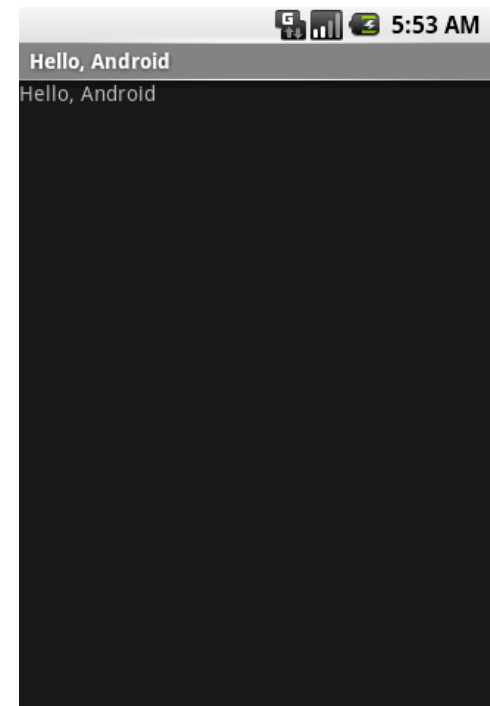
```
Component comp = director.create( new RadioButtonBuilder(), choiceStrings);
Component comp2 = director.create( new ComboBuilder(), choiceStrings);
```

# Extracting Structure and Data in Android

```
package com.android.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```



# Extracting Structure and Data in Android

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
```

/res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am a string resource!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

/res/values/strings.xml

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

HelloAndroid.java



# MVC וטכנולוגיות Web

- ניתן לבחון את טכנולוגיות צד הלקוח בפרספקטיבה של MVC
  - HTML
  - CSS
  - JavaScript

# Example: Simple rollover

- The following code will make the text **Hello** **red** when the mouse moves over it, and **blue** when the mouse moves away

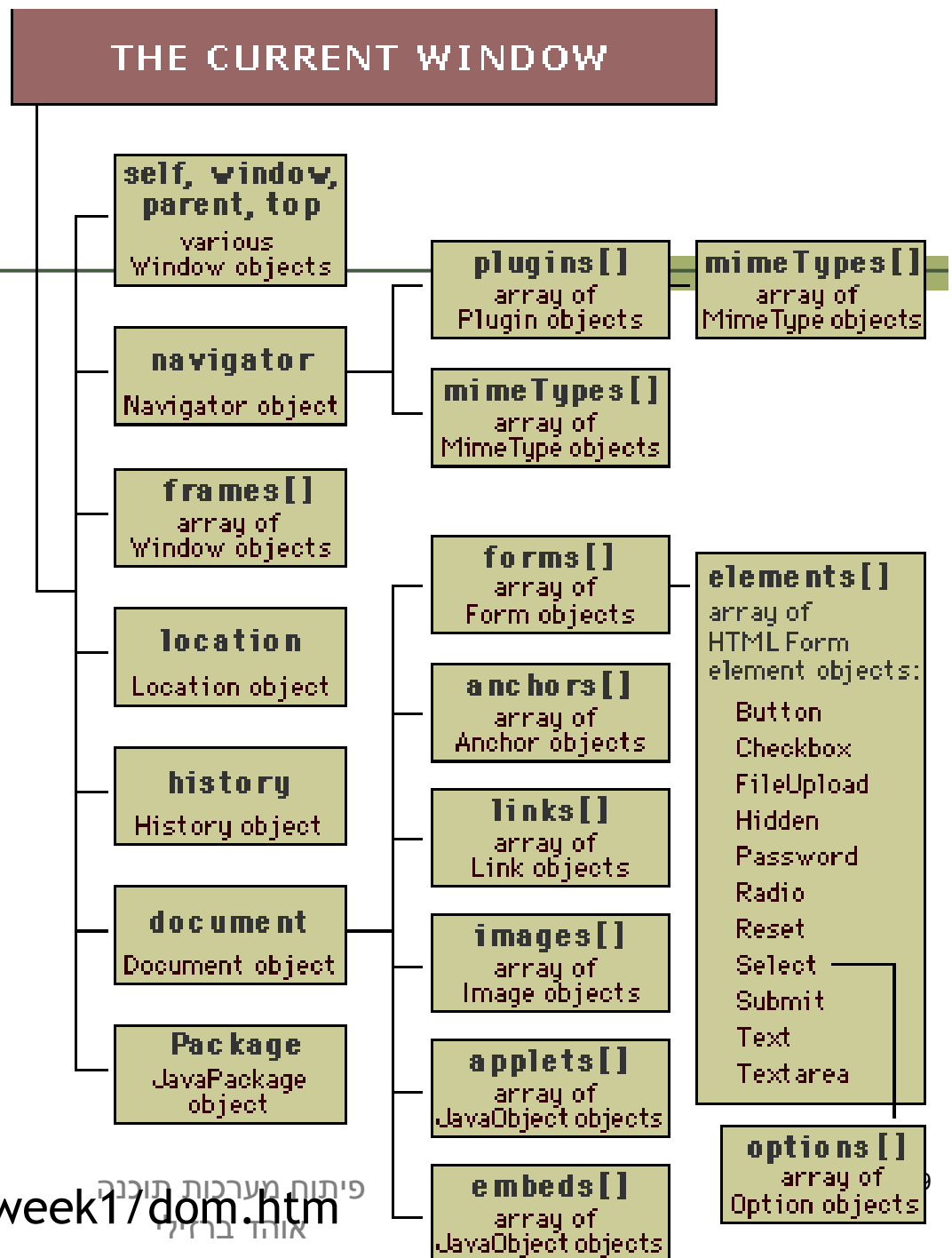
```
<h1 onMouseOver="style.color='red';"  
  onMouseOut="style.color='blue';">Hello </h1>
```

- Image rollovers are just as easy:

```

```

# The DOM hierarchy

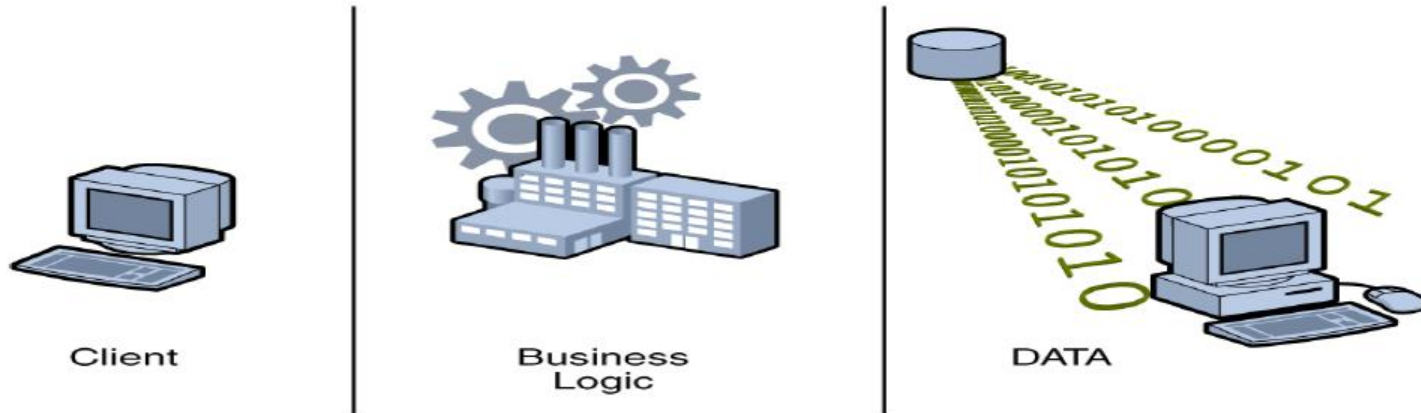


Source:

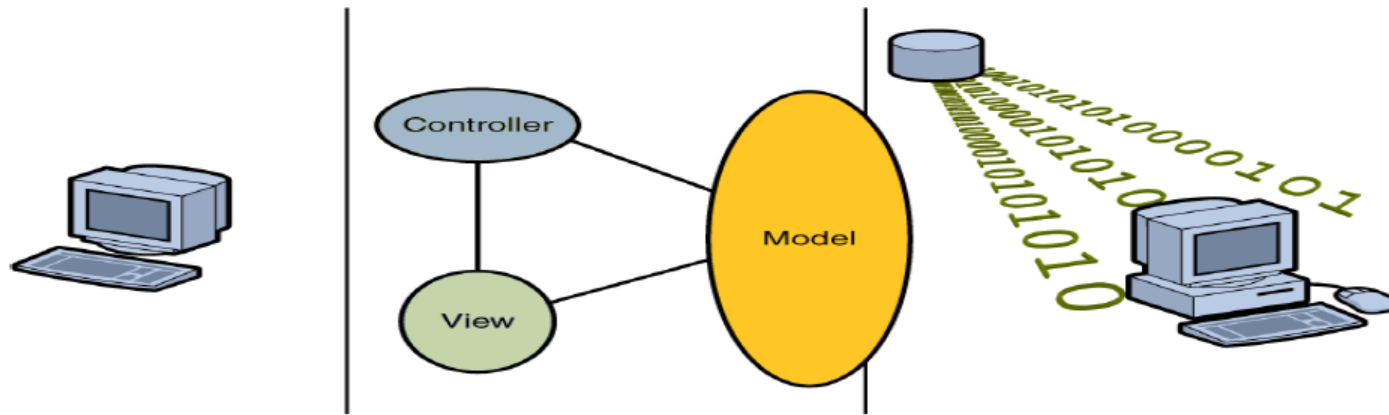
<http://sislands.com/coin70/week1/dom.htm>

פיתוח מערכות חינוך  
אוניברסיטת בר-אילן

# Web Application – Three-Tier Architecture

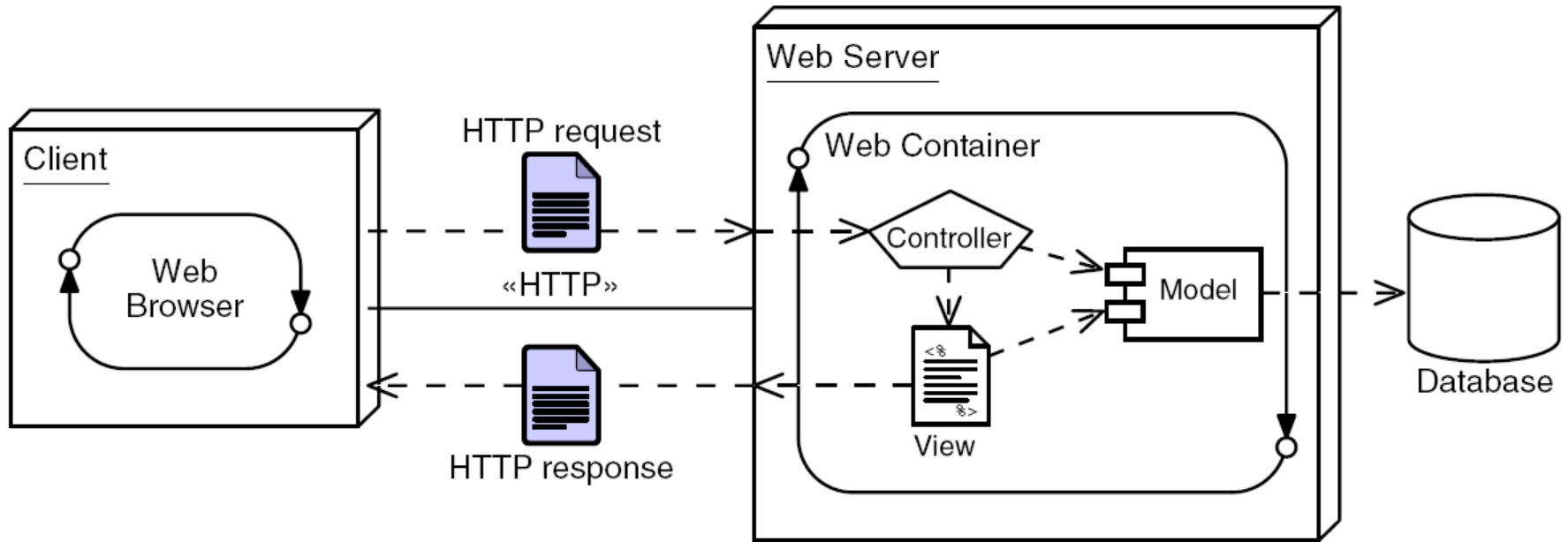


# Model-View-Controller (MVC) Architecture in a Web Application



# Model 2 Architecture

Deployment diagram of a web container using Model 2 architecture:



# Model 2 Frameworks

- Frameworks are partial implementations on which you can build your components.
- There are several Model 2 frameworks available:
  - Struts from the Jakarta group
  - JavaServer Faces technology from Sun
  - Velocity from Apache

# MVC Frameworks

קיימות שפות תכנות וסביבות פיתוח אשר MVC

מהווה מרכיב מרכזי בהן:

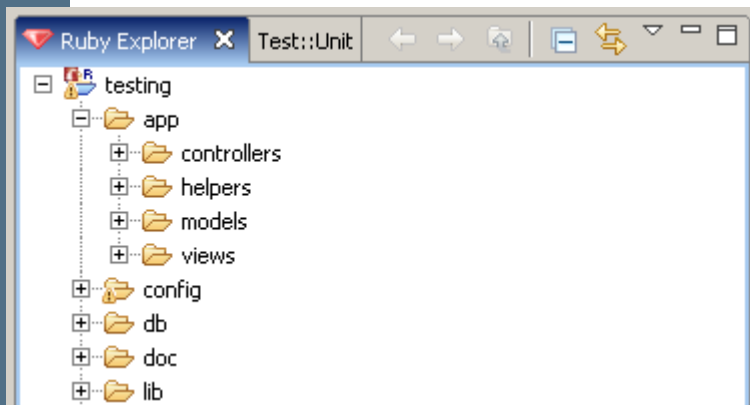
Ruby on Rails

MFC

Struts

לדוגמא:

פרוייקטים ב Ruby on Rails נוצרים אוטומטית עם תיקיות נפרדות ל Views , models ו- controllers

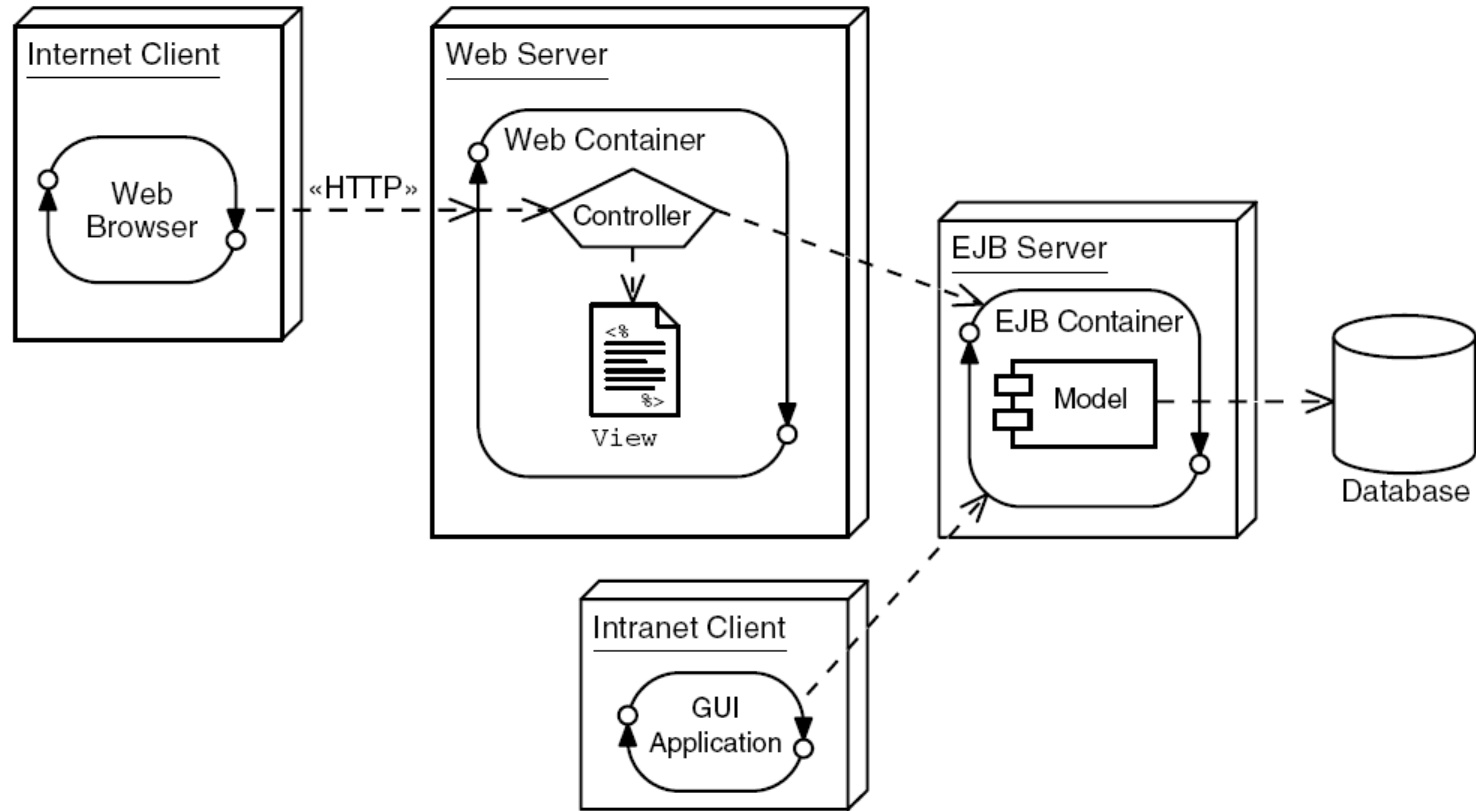




# Java EE Containers

- Modular design allows for easier modification of the business logic.
- Enterprise components can use container-provided services such as presentation, security, transaction, persistence, and life cycle management.

# Java EE Architecture Example



# Job Roles

The modularity of Java EE architecture clearly distinguishes several job roles:

- Web Designer – Creates View elements
- Web Component Developer – Creates Controller elements
- Business Component Developer – Creates Model elements
- Data Access Developer – Creates database access elements

# לסיכום - אז למה דיברנו על MVC?

- כי כדאי להשתמש ברעיונות טובים בטכנולוגיות חדשות
  - רעיונות כאלה לא תמיד קיימים ב"גרסה 1.0" ומשתלבים בה מאוחר יותר כחלק מהתהליך ההבשלה של טכנולוגיה חדשה
- כי כדאי להשתמש ברעיונות טובים בהקשרים חדשים
  - תבנית עצוב, שפת תכנות, framework, ארכיטקטורה, כח עבודה