Aplikasi Enterprise-3

Developing Enterprise Application Patterns

Wiratmoko Yuwono

Developing Enterprise Application Patterns (Outline)

- Organize Business Logic
- Mapping to relational database, ’
- Web presentation,
- Concurrency,
- Session
- Putting all together

Organize the Business Logic

Transaction Script

- **prosedur/routine** yang mengambil input dari presentation layer dan memprosesnya dengan berbagai validasi dan kalkulasi, menyimpan data dalam database, dan melakukan beragam operasi dari sistem yang lain.
- Transaction script biasanya berupa **script** yang digunakan untuk menghandle berbagai aksi yang mungkin dilakukan oleh user
  - Tidak harus dijadikan satu, **bisa dipisah-pisah**, dalam bentuk procedure-procedure, atau fungsi-fungsi.
  - **Contoh**: script untuk checkout barang, menambah barang ke shopping cart, untuk menampilkan status delivery, dan lain-lain.

Contoh Transaction Script

```
Public Class OrderProcessing

  Public Shared Function PlaceOrder(ByVal order As OrderInfo) As Long

        ' Start a transaction

        ' Check Inventory

        ' Retrieve customer information, check credit status

        ' Calculate price and tax

        ' Calculate shipping and total order

        ' Save Order to the database and commit transaction

        ' Send an email confirming the order

        ' Return the new order ID

    End Function

End Class
```

Contoh Transaction Script

```
Public Structure OrderInfo
    Public ProductID As Long
    Public Quantity As Integer
    Public CustomerId As Long
    Public ShipVia As ShipType
End Structure

Public Enum ShipType
   FedEx
   UPS
   Postal
End Enum
```

Contoh Transaction Script

```
  Public Shared Function SaveCustomer(ByVal customer As CustomerInfo) As Long

        ' Validate Address

        ' Start a transaction

        ' Look for duplicate customers based on email address

        ' Save customer to database and commit transaction

        ' Return the new customer ID

  End Function
End Class


Public Structure CustomerInfo

  Public Name As String

  Public Address As String

  Public City As String

  Public State As String

  Public PostalCode As String

  Public Email As String
End Structure
```

Transaction Script

(+)

- Berupa **prosedur** yang simpel dan mudah dimengerti oleh para developer.
- Mudah digunakan dan digabungkan dengan **data source layer** menggunakan **database gateway**.
- Mudah ditentukan batas transaksinya:
  - Dimulai dari opening transaction dan diakhiri dengan closing transaction

(-)

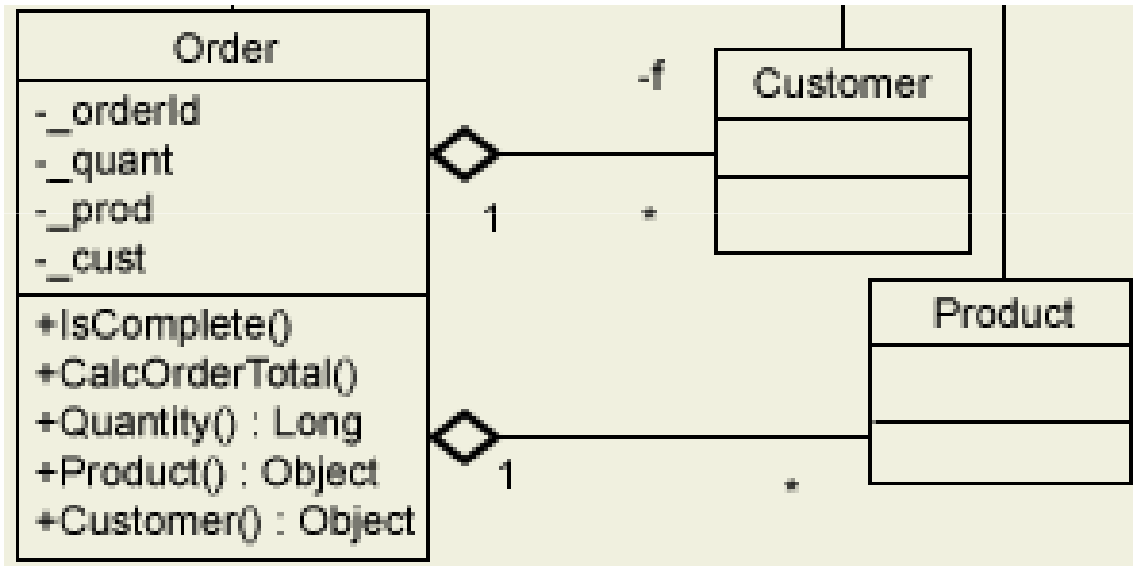
- Ketergantungan antar transaksi masih tinggi
- Tidak cocok untuk domain logic yang **kompleks**
- Bisa terjadi **duplikasi** kode karena beberapa transaksi perlu melakukan hal yang sama.

Organize the Business Logic

Domain Model

- Suatu sistem pengorganisasian bisnis logik dengan pendekatan sistem **berorientasi obyek**, sehingga kita diharuskan membangun sebuah model dari **domain permasalahan** yang kita hadapi.
  - Semua logika tersebut dimasukkan dalam **method/routine** obyek tersebut.

Domain Model



Perbedaan

- Pada *pendekatan (orientasi) nya*.
- Transaction Script beorientasi **method/fungsi/rutin** di mana seluruh logic user dijadikan satu dalam **fungsi-fungsi tanpa** memperhatikan domain permasalahan,
- Domain Model: logic user dibuat dalam **obyek-obyek** yang berkaitan dengan domain permasalahan.

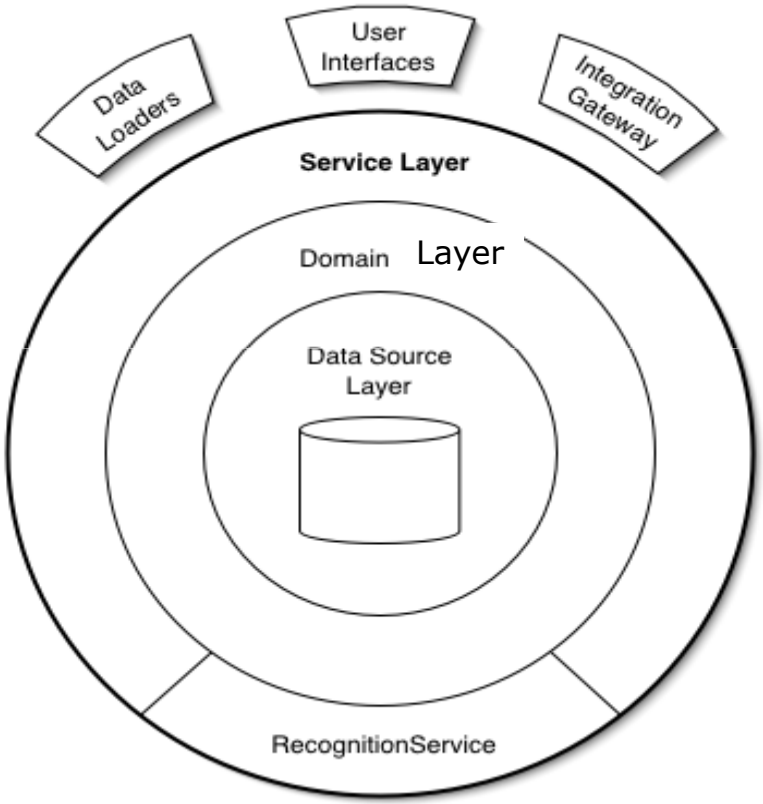
Organize the Business Logic

Table Module

- Table Module menggunakan pengorganisasian class dan obyek seperti layaknya Domain Model,
- **Domain Model** memiliki 1 instance untuk tiap-tiap class-nya di dalam database.
  - 1 tabel -> 1 class
- **Table Module** hanya memiliki 1 instance saja untuk **seluruh kelas** dan didesain untuk bekerja dengan Record Set / Result Set / Data Set
  - Semua query yang dilakukan harus melalui Record Set
  - Kembalian datanya biasanya banyak
  - 1 database -> 1 class

Service Layer & Domain Layer

- Pendekatan umum untuk menghandle **bisnis logic** adalah membagi bisnis logic ke dalam 2 bagian.
  - **Service Layer**: digunakan jika menggunakan **Domain Model atau Table Module**.
    - Service Layer beraksi seperti **API** Service.
  - **Domain Layer**: digunakan jika menggunakan **Transaction Script**
    - Lebih dekat dengan data layer.

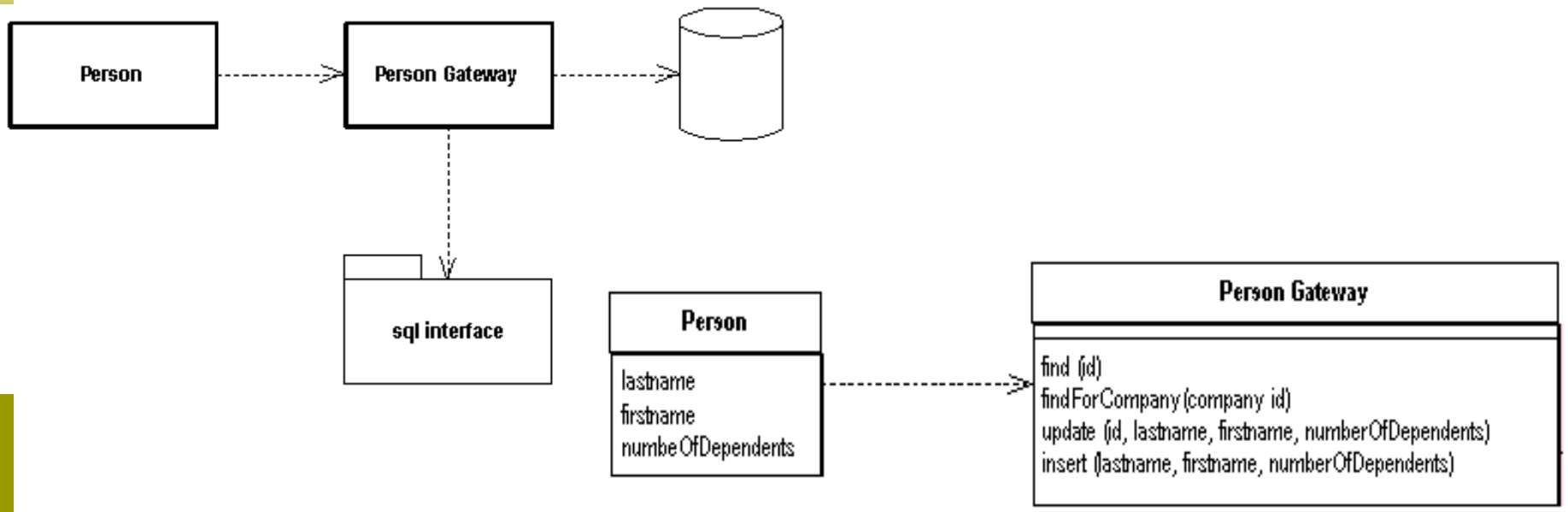


Mapping to Relational Databases

- Dari segi **arsitektur**: bagaimana bussines logic berhubungan dengan database
- Karena SQL tidak terlalu seragam, maka jalan terbaik adalah memisahkan akses SQL dari bussines logic dan menempatkannya dalam kelas yang terpisah.
  - Misalnya satu class mewakili satu tabel.
- Kelas-kelas ini akan membentuk **Database Gateway** ke dalam tabel-tabel sehingga programmet tidak harus tahu tentang SQL.
- Model Pengaksesan data:
  - **per 1 record** dan
  - **sekaligus seluruh/beberapa record** dalam 1 tabel oleh 1 query
- Bisa juga menggunakan **Object Relational Model**: LINQ, Subsonic, Hibernate

Model Pengaksesan data

- *A* **Row Data Gateway** *has one instance per row returned by a query*
- *A* ***Table Data Gateway*** *has one instance per table, returned a recordset*



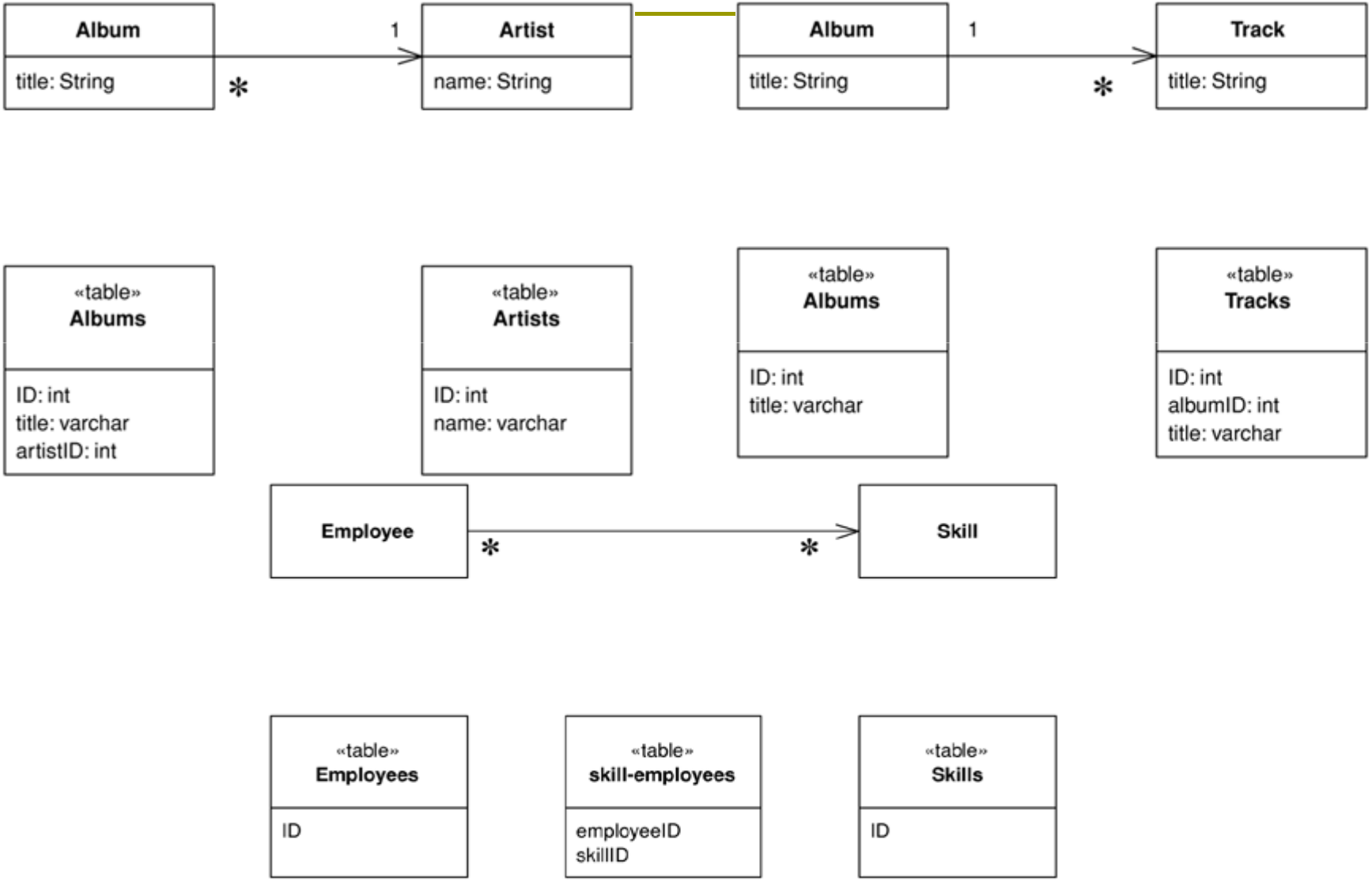
Contoh Model Akses Data

```
create table people (ID int primary key, lastname varchar, firstname varchar, number_of_dependents int)
INSERT INTO people VALUES(1,'Fowler', 'Martin', 1)
INSERT INTO people VALUES(2,'Rice', 'Dave', 0)
INSERT INTO people VALUES(3,'Foemmel', 'Matt', 0)

class Person...
        private String lastName;
        private String firstName;
        private int numberOfDependents;

        public String getLastName() {
                return lastName;
        }
        public void setLastName(String lastName) {
                this.lastName = lastName;
        }
        public String getFirstName() {
                return firstName;
        }
        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }
        public int getNumberOfDependents() {
                return numberOfDependents;
        }
        public void setNumberOfDependents(int numberOfDependents) {
                this.numberOfDependents = numberOfDependents;
        }
```

Contoh Model Akses Data (2)

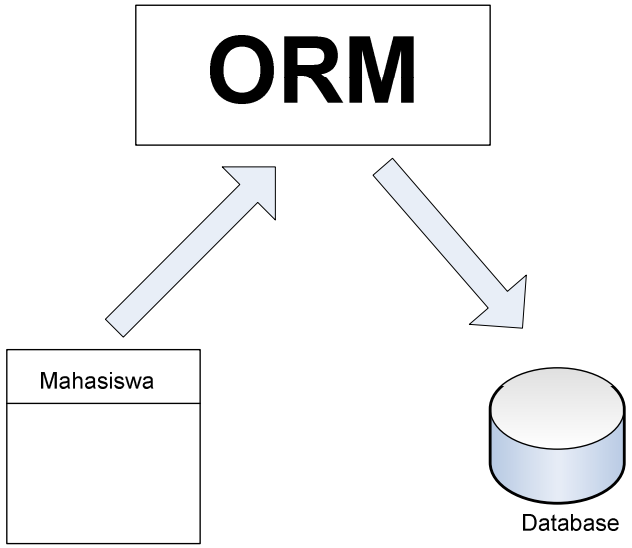
```
public Person find(Long id) {
        Person result = (Person) Registry.getPerson(id);
        if (result != null) return result;
        try {
                findStatement.setLong(1, id.longValue());
                ResultSet rs = findStatement.executeQuery();
                rs.next();
                result = Person.load(rs);
                return result;
        } catch(SQLException e) {throw new ApplicationException(e);}
}


class Person...
        public static Person load(ResultSet rs) {
                try {
                        Long id = new Long(rs.getLong(1));
                        Person result = (Person) Registry.getPerson(id);
                        if (result != null) return result;
                        String lastNameArg = rs.getString(2);
                        String firstNameArg = rs.getString(3);
                        int numDependentsArg = rs.getInt(4);
                        result = new Person(id, lastNameArg, firstNameArg, numDependentsArg);
                        Registry.addPerson(result);
                        return result;
                } catch(SQLException e) {throw new ApplicationException(e);}
        }
```

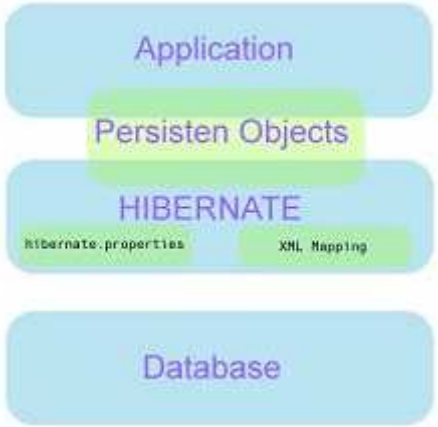
Structural Mapping



ORM – Object Relational Mapping

- Teknik memetakan database menggunakan **object**
- Jembatan antara OOP dengan RDBMS
- Pertama populer di Java
- Kemudian diikuti bahasa lainnya (ex: .NET)
- Disadvantages of O/R mapping tools are in areas where **proprietary**, or **database-specific techniques** have been highly optimized





Kenapa ORM ?

- Menghindarkan programmer dari kode SQL yang bermacam-macam
- Memisahkan kode SQL dari logika aplikasi
  - Kode SQL dibungkus class dan method
- Mempermudah maintenance program
- Menghindari ketergantungan aplikasi terhadap vendor database
  - Aplikasi dapat menggenerate kode SQL CRUD untuk berbagai jenis database

Non ORM

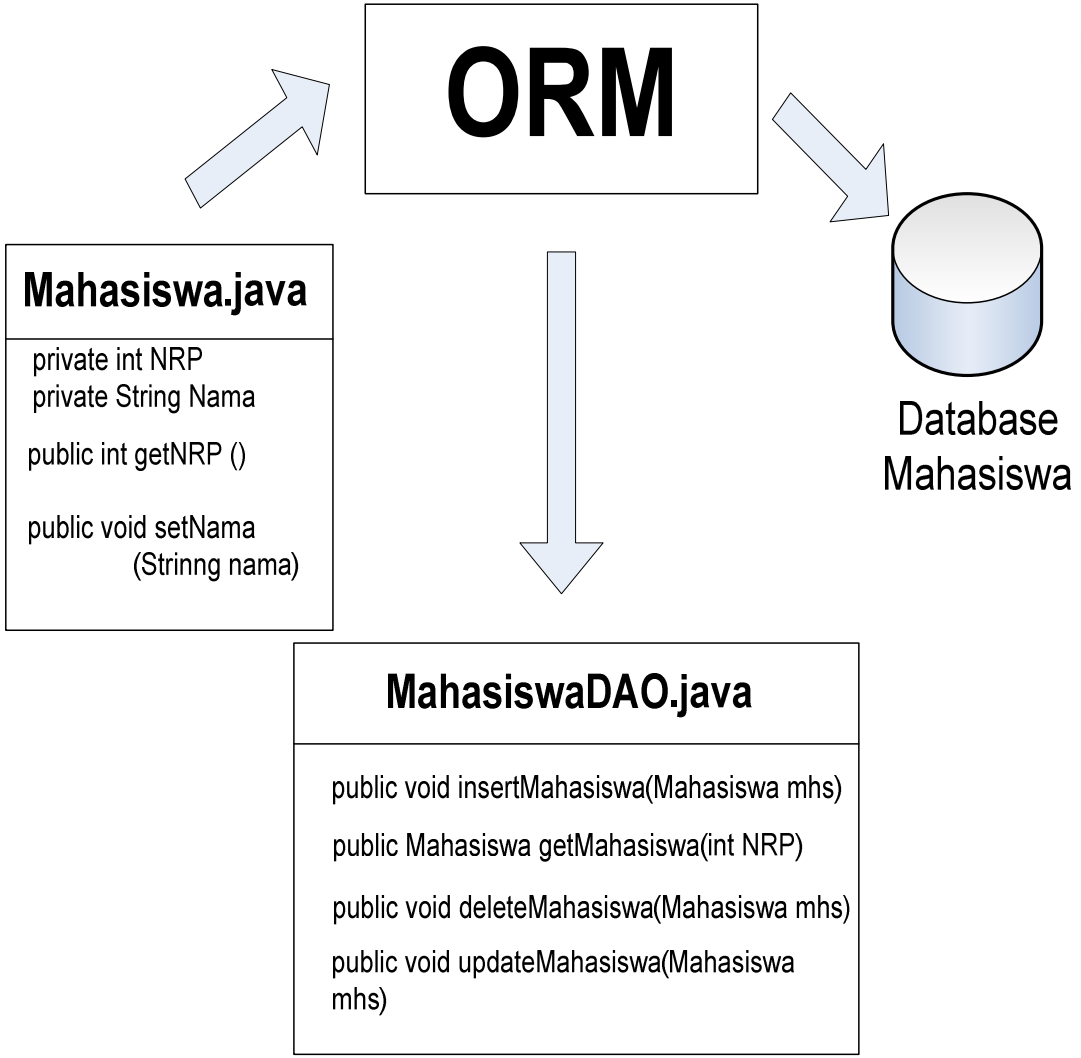
```
String host = "localhost";
String db = "kiki";
String user = "root";
String pass = "";
String driver = "com.mysql.jdbc.Driver";
String koneksi = "";


koneksi = "jdbc:mysql://"  + host + ":3306/" + db + "?user=" + user +"&password=" + pass;

Class.forName(driver).newInstance();
Connection con = DriverManager.getConnection(koneksi);
Statement stmt = con.createStatement();

String ReqUser =  request.getParameter("username");
String ReqPass = request.getParameter("password");

String QUERY_INSERT = "INSERT INTO Mahasiswa VALUES (“ +ReqUser+ “,” +ReqPass+” )””
```

ORM

```
public void tambahMhs (){

        Mahasiswa ophex = new Mahasiswa();
        ophex.setNRP(1);
        ophex.setNama("ophex");
        ophex.setPassword("mbuh");
        HibernateUtil.getSessionFactory().getCurrentSession().save(ophex);


                }
```

- Tidak ada kode SQL pada source code
- Tidak ada ketergantungan terhadap DB server
- Baris kode lebih sedikit

Cara Kerja ORM

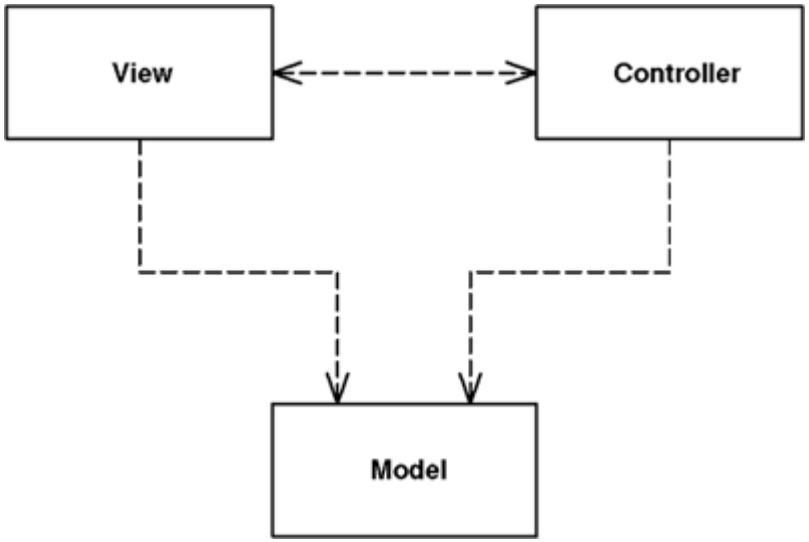


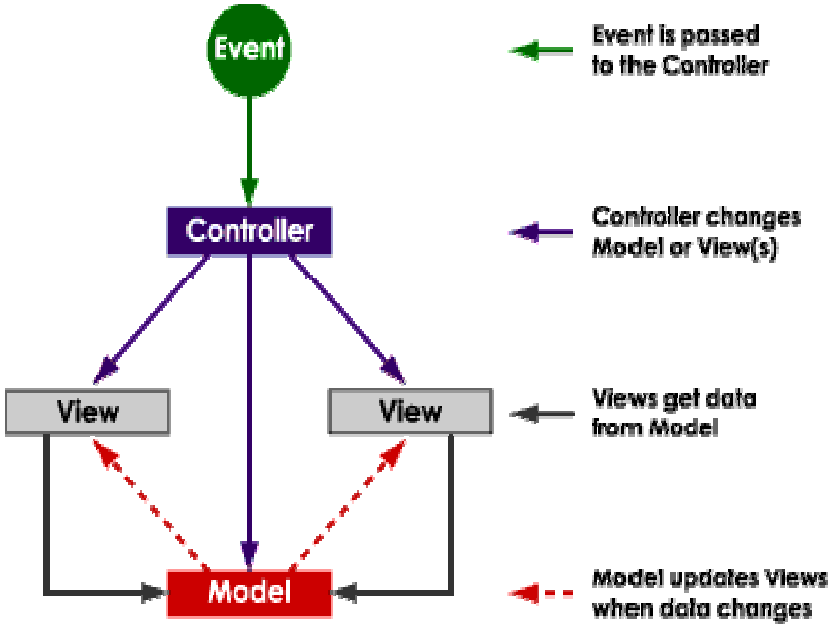
- Database dipetakan menggunakan **POJO (Plain Object Java Object)**
- Proses **CRUD** dilakukan menggunakan **DAO (Data Access Object)**

Presentation Model

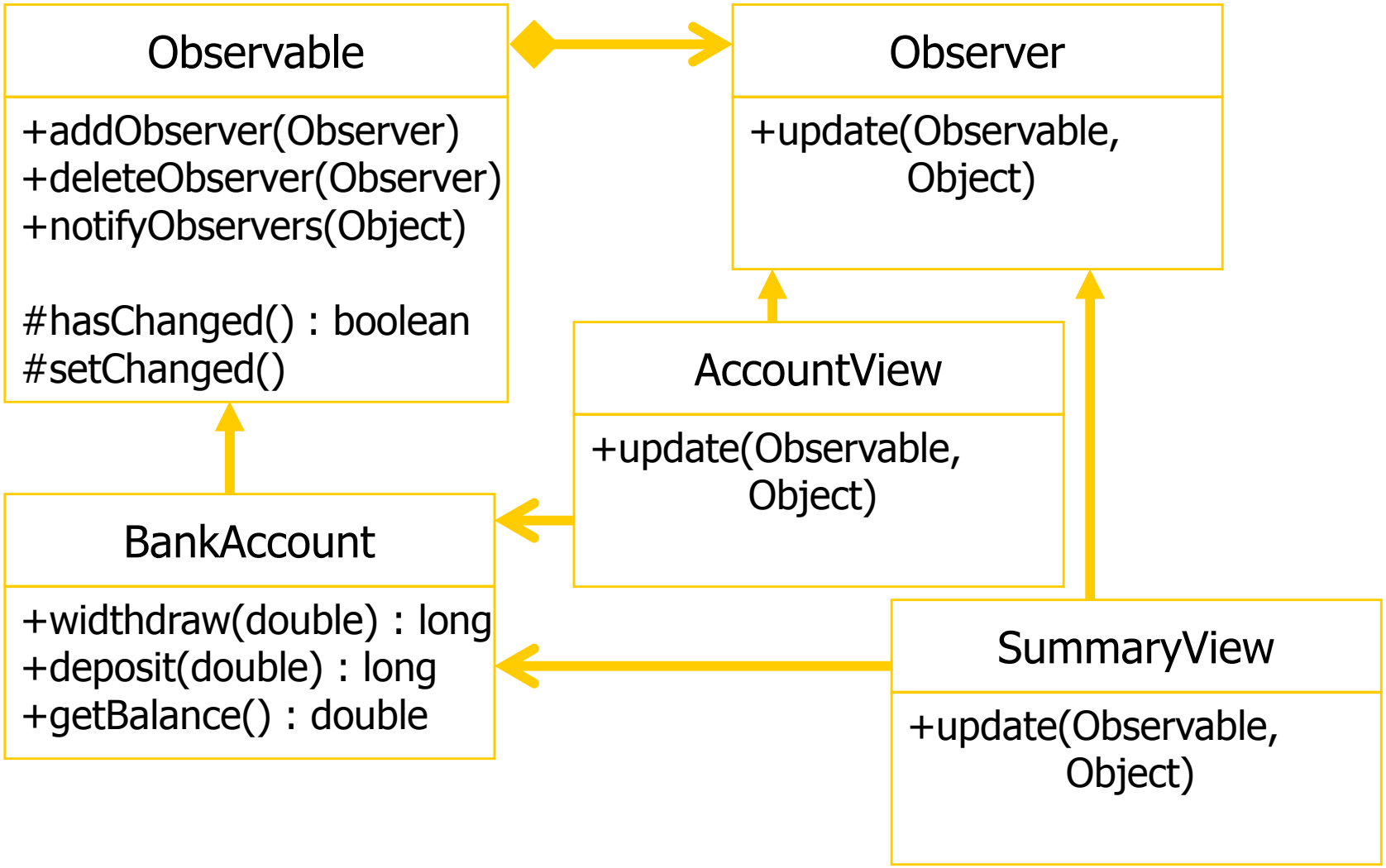
Model View Controller (MVC)

- Membagi interaksi user interface ke dalam 3 bagian: View, Controller, dan Model
- Dibuat oleh Trygve Reenskaug untuk Smalltalk-80 di Xerox PARC.

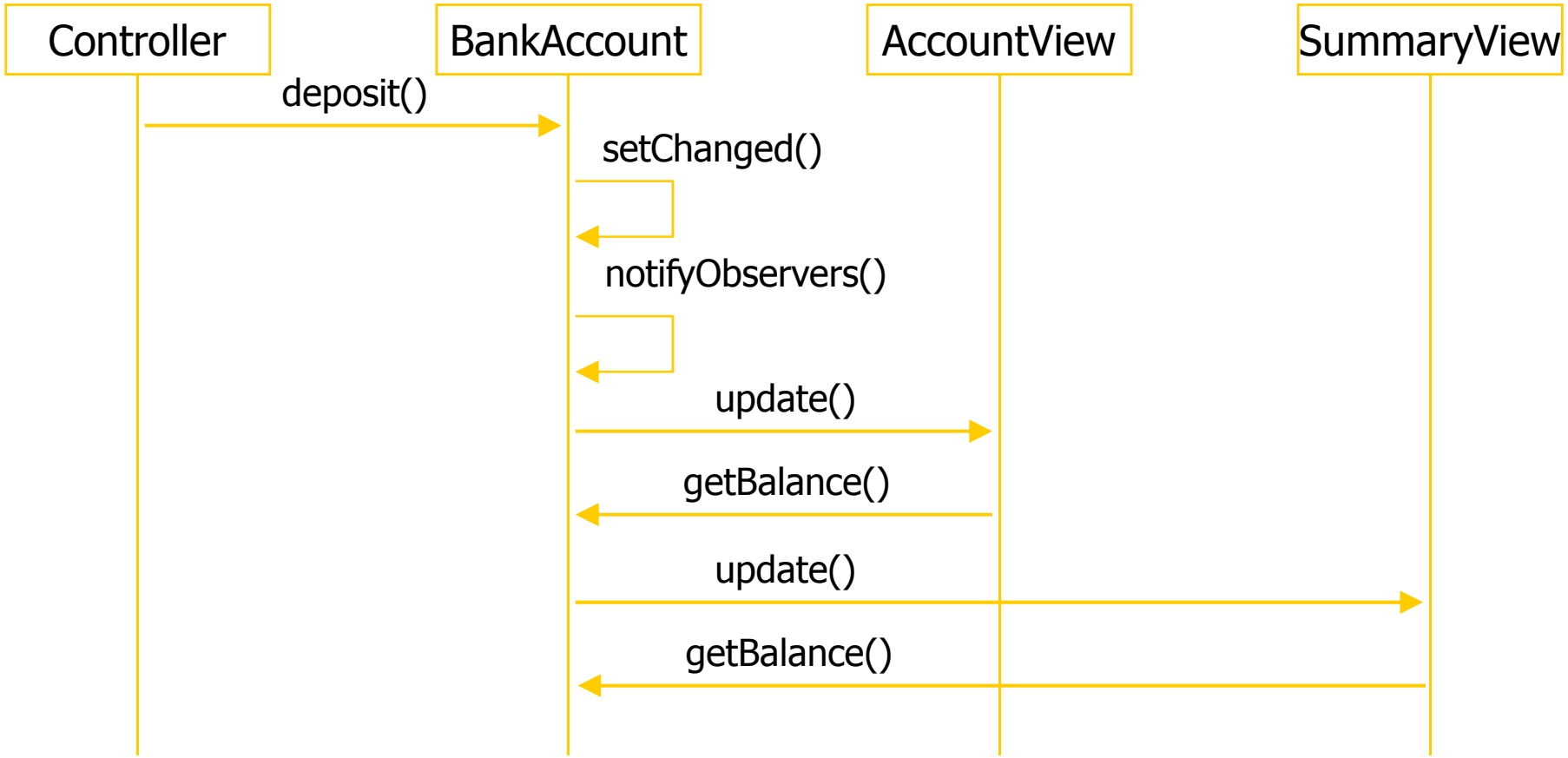




Berbasis pada Observer Pattern
Cth: pada Bank



When transactions happen!

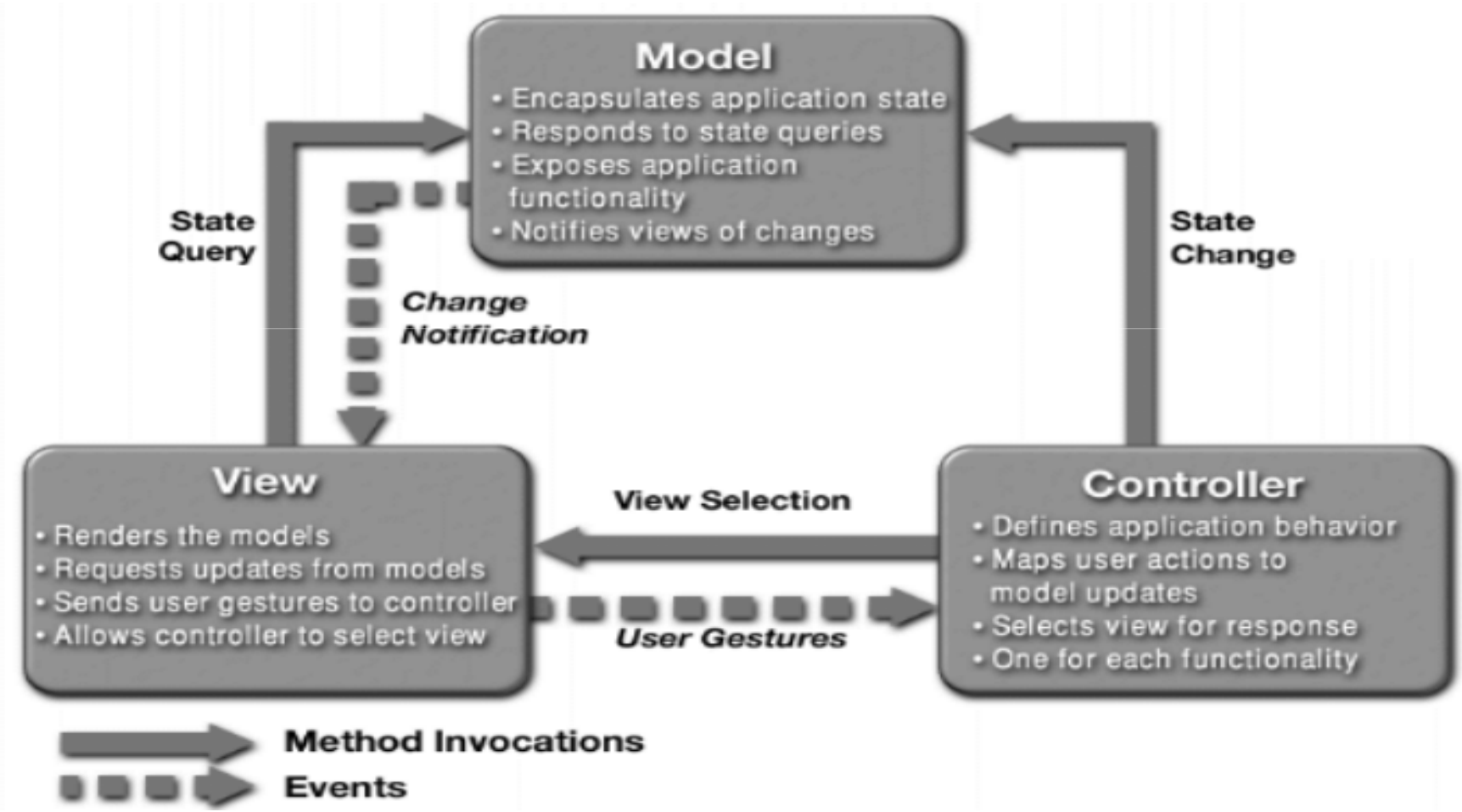


Presentation Model

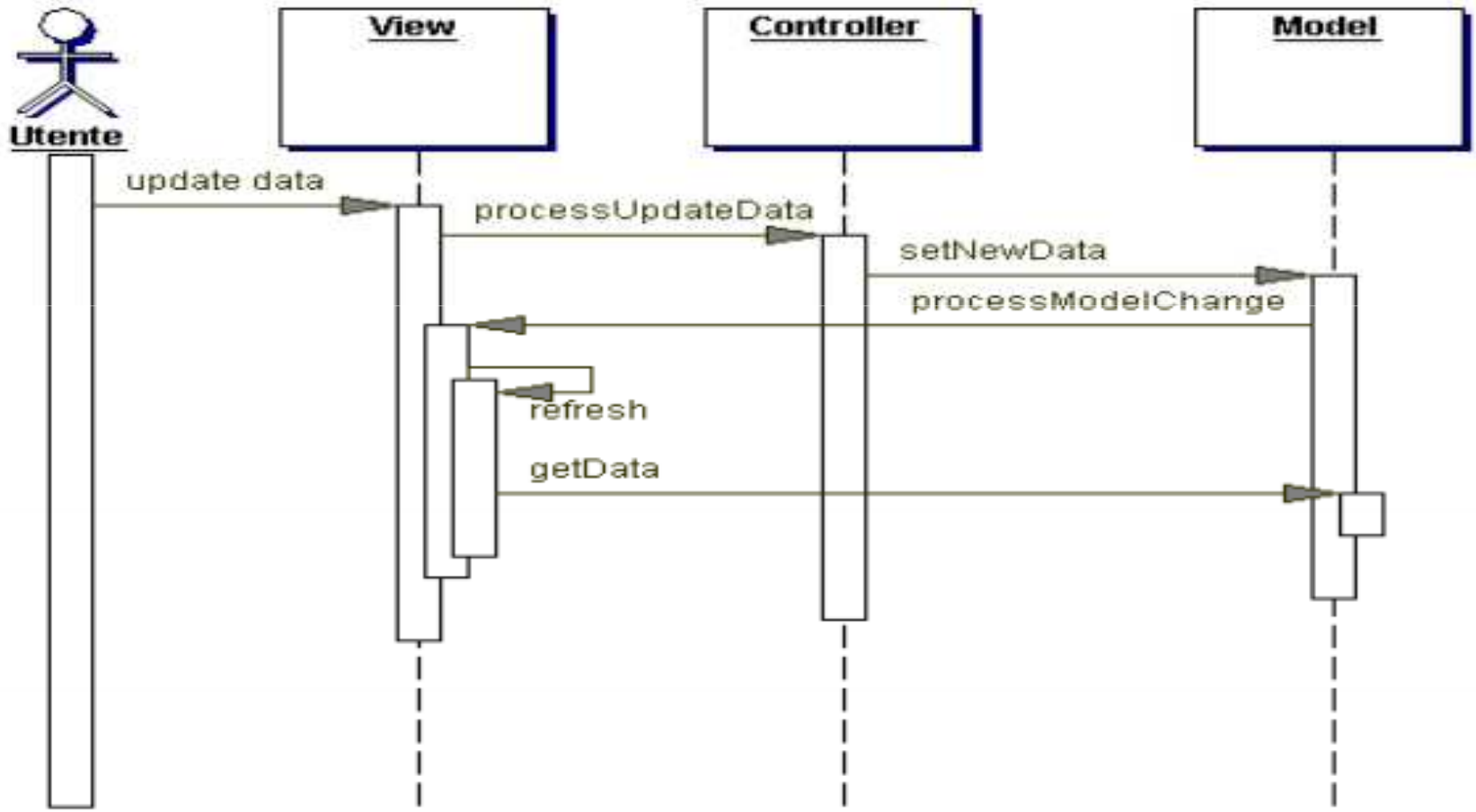
MVC

- **View**: merpresentasikan layout dari sebuah model dalam UI yang juga merepresentasikan data dalam tampilan tertentu.
- **Controller** (program): menghandle semua perubahan terhadap informasi/data.
  - Controller mengambil inputan user, memanipulasi data, dan meminta view untuk mengupdatenya.
- **Model:** menampilkan informasi mengenai domain data, mengatur business policies, dan database.
  - Berupa non-visual object yang berisi semua data dan tingkah lakunya

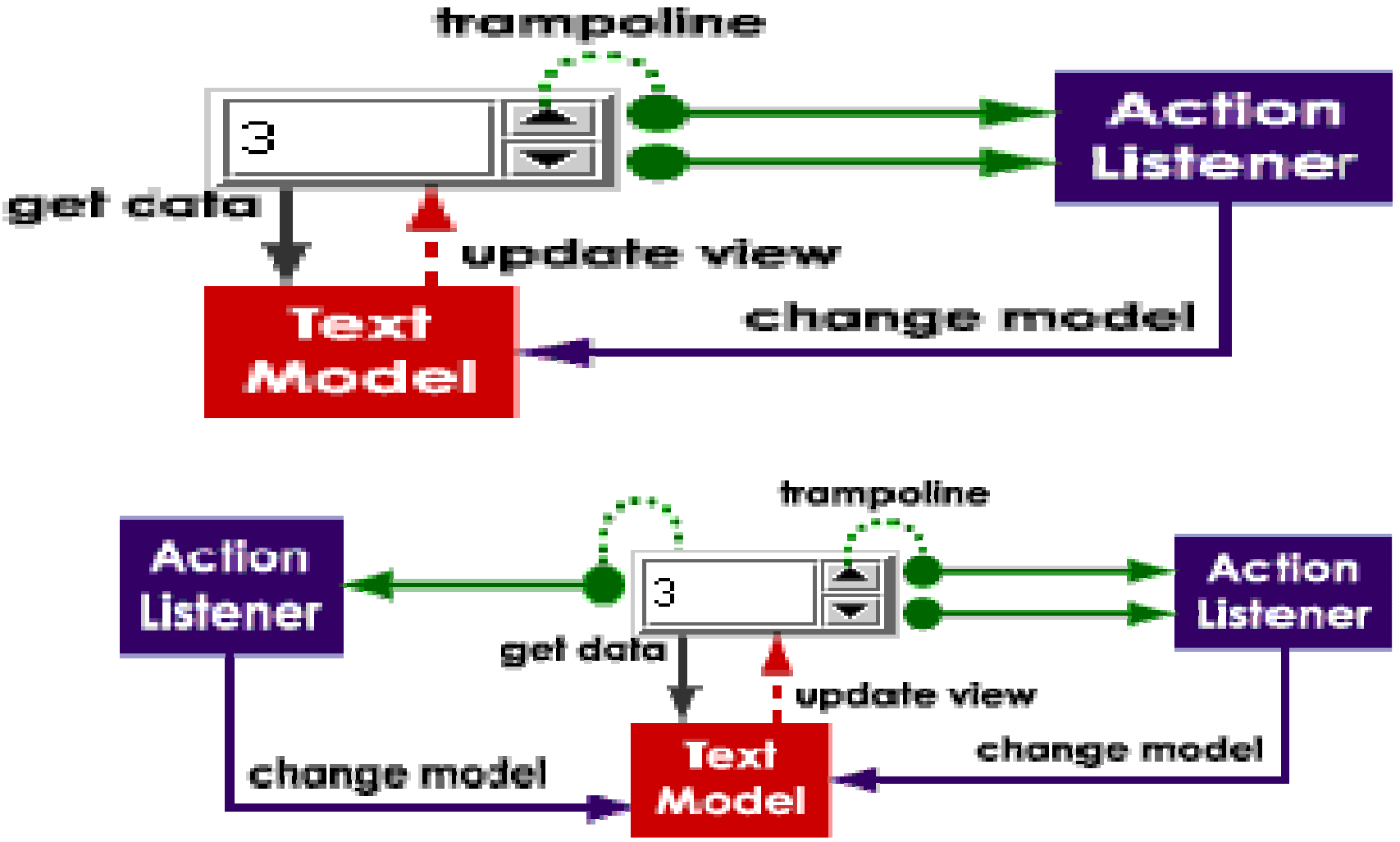
MVC


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

MVC Secara Umum



Contoh MVC pada Desktop



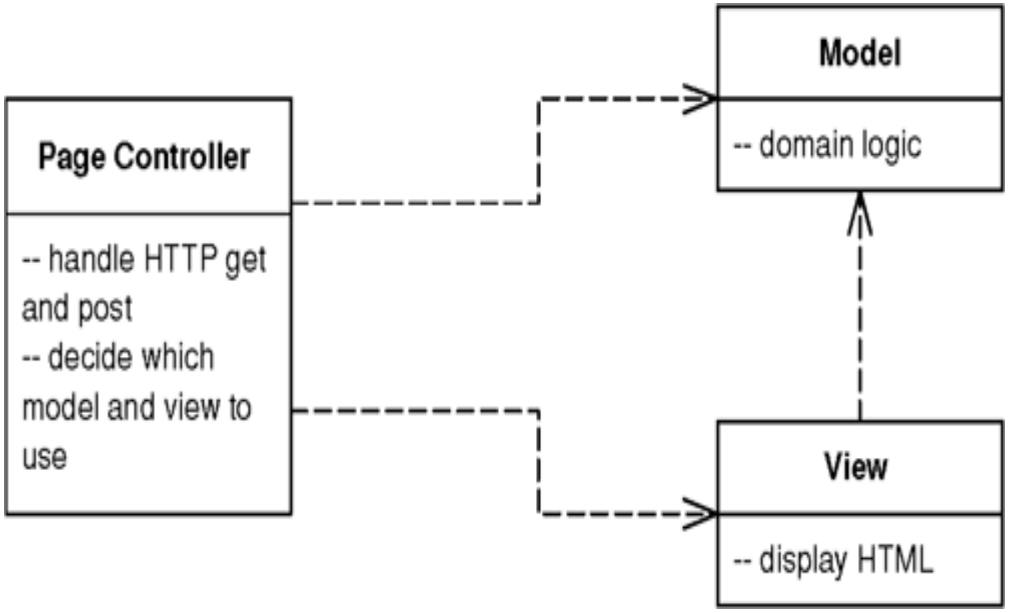
MVC Separation

- Separating the **presentation** from the **model**
  - Presentasi dan model memiliki fokus yang berbeda:
    - presentasi ke tampilan, model ke data.
  - Perbedaan ketergantungan:
    - presentasi bergantung pada model tapi tidak sebaliknya.
  - Sehingga dapat mengembangkan multiple presentation dengan model yang sama
- Separating the controller from the view
  - Contoh: .NET Web application
  - File: *.aspx dan *.aspx.vb
- Contoh: PHP Zend Framework, Code Igniter, CakePHP

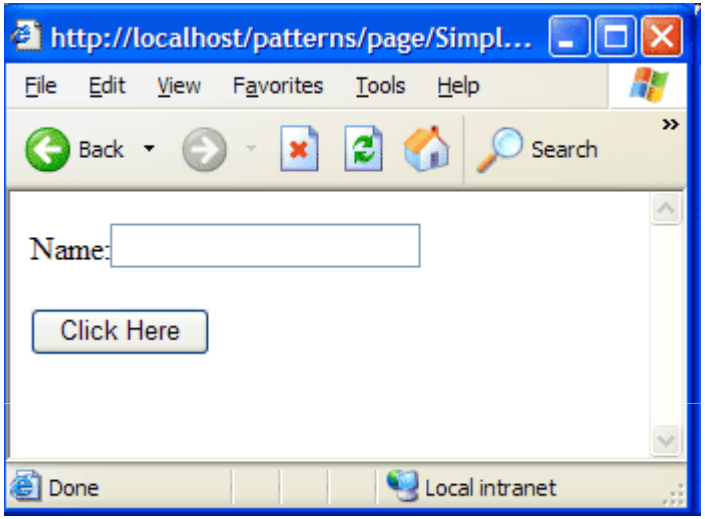
Presentation Model

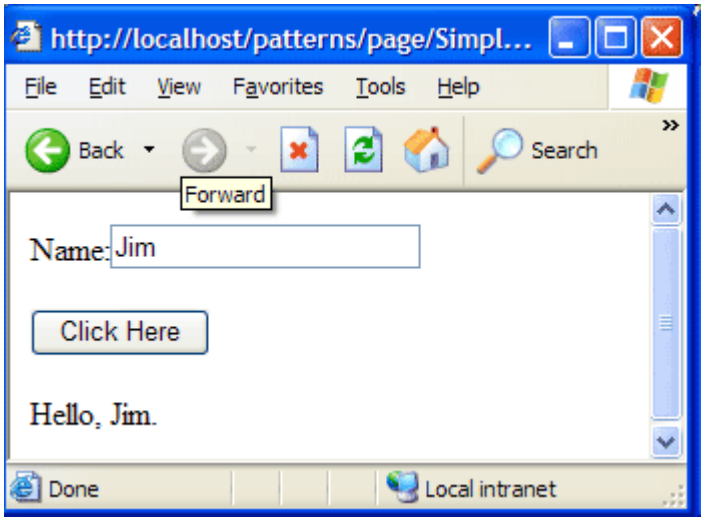
Page/Action Controller

- An **object / controller** that handles a **request** for a **specific** page or action on a web site.



Page Controller Example





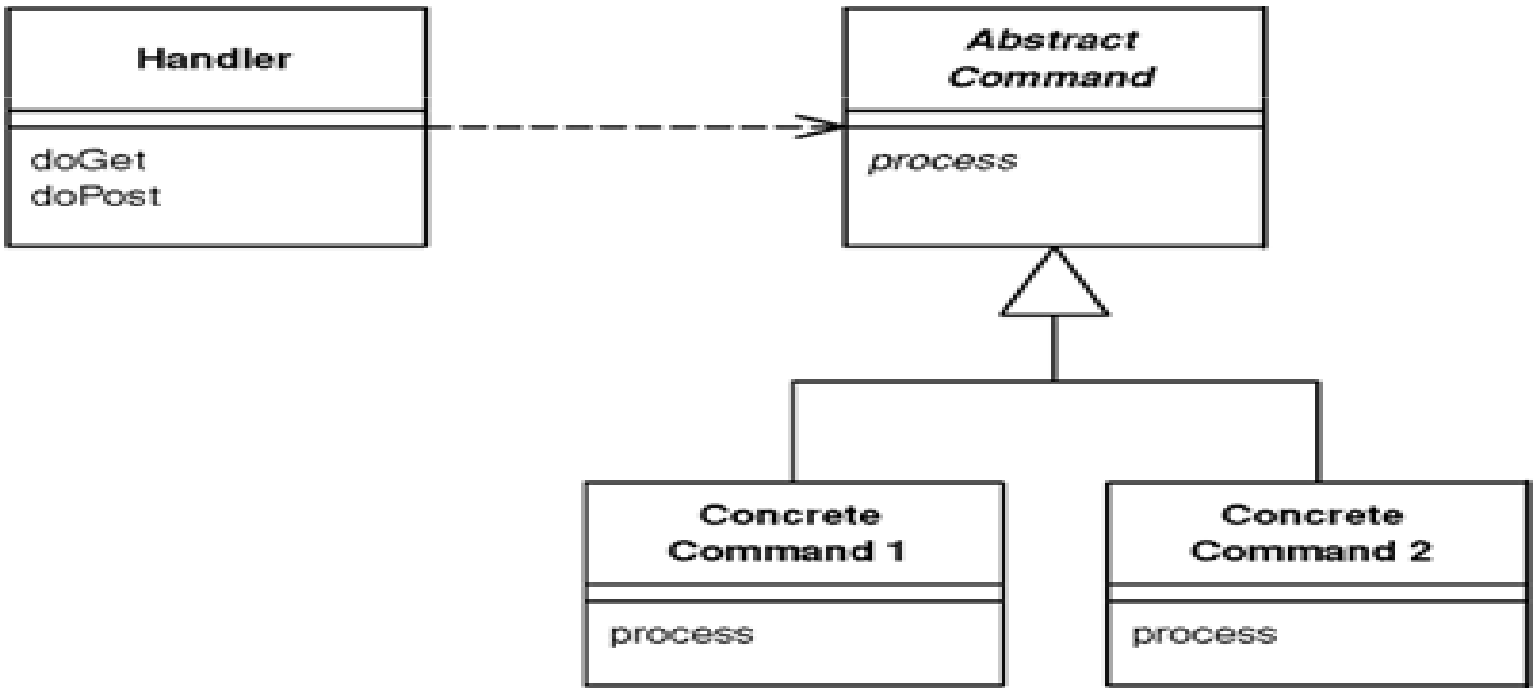

```
<%@ Page language="c#" Codebehind="SimplePage.aspx.cs" AutoEventWireup="false" Inherits="SimplePage" %>
<HTML>
  <body>
    <form id="Form1" runat="server">
      Name:<asp:textbox id="name" runat="server" />
      <p />
      <asp:button id="MyButton" text="Click Here" OnClick="SubmitBtn_Click" runat="server" />
      <p />
      <span id="mySpan" runat="server"></span>
    </form>
  </body>
</HTML>
```

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public class SimplePage : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.TextBox name;
  protected System.Web.UI.WebControls.Button MyButton;
  protected System.Web.UI.HtmlControls.HtmlGenericControl mySpan;

  public void SubmitBtn_Click(Object sender, EventArgs e)
  {
    mySpan.InnerHtml = "Hello, " + name.Text + ".";
  }
}
```

Presentation Model

Front Controller

- Sebuah **controller** yang menangani semua request untuk sebuah web site.



Front Controller Example

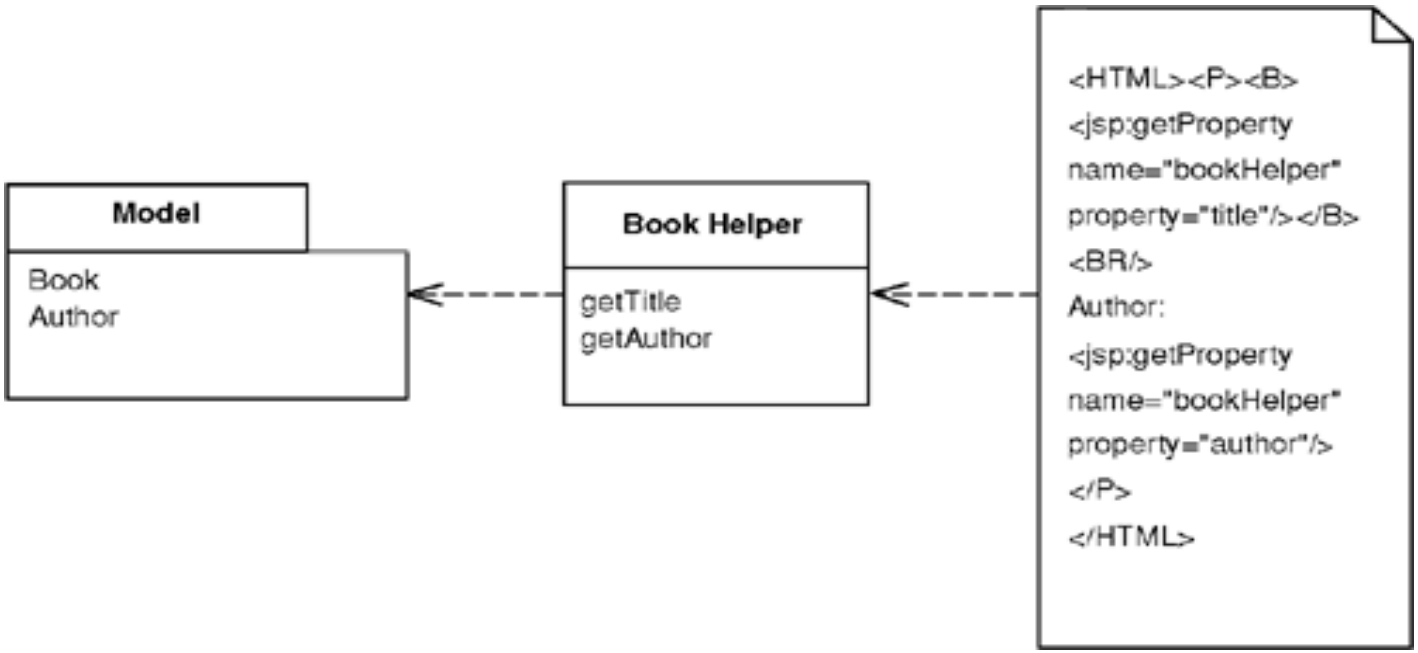
http://mysite.com/foo/bar

```
< ?
//A file with all the common configuration, like web roots, security,
//database, language, etc.
require_once('lib/config.php')

//Your header, the same regardless of the page
require_once('parts/header.php')

//This is the content area that will change based on the URL.
//My div is called 'textarea'. Yours might be called something else.
echo '
';

//The switch statement
$url = substr($_SERVER['REQUEST_URI'], strlen(URLROOT));
switch($url){
        case (''):
                require_once('pages/root.php');
        break;
        case ('foo'):
                require_once('pages/foo.php');
        break;
        case ('bar'):
                require_once('pages/bar.php');
        break;
        default:
                //the default is an error!
                require_once('pages/error.php');
        break;
}
echo '';
//Your footer, the same regardless of the page
require_once('parts/footer.php')
?>
```

Presentation Model

Template View

- Merender informasi ke dalam sebuah halaman web berdasarkan tanda-tanda (**markup**) ada didalamnya.
- Ketika halaman melayani sebuah request, maka **markup** tersebut akan diganti dengan hasil dari komputasi seperti misalnya dari query database.



Simple Template View

```
// script 1.1
   <html>

   <head><title><?php echo $title; ?></title></head>

   <body>

   <?php echo $heading; ?>

   <?php echo $content; ?>

   </body>

   </html>
```

Advanced Template View

```php
// View Helper
class NewsViewHelper
{
    public function __construct($connection)
    {
        $this->_model = new NewsModel($connection);
        // let's just make believe that we are good and have separated our DB logi
    }
        public function getTitle()
        {
            if ($this->_model->getTitle()) {
                return $this->_model->getTitle();
            } else {
                return 'No Page Title';
            }

        }

        public function listSomeTable()
        {
            $HTML = '<h2>No records to display</h2>';
            while ($row = $this->_model->getList())  {
                $HTML .= '<h2>'.$row->heading. '</h2>';
                $HTML .= '<p>'.$row->summary.'</p>';
            }
            return $HTML;
        }

    }
```

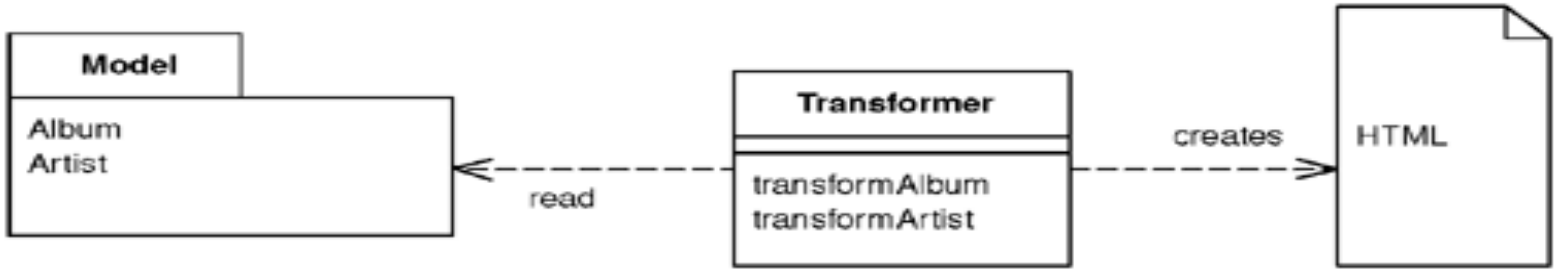
Advanced Template View

```
// Template View (script 1.3)
    $helper = $this->getHelper('NewsViewHelper'); // I am using a temp variable as


    <html>
    <head>
    <title><?php echo $helper->getTitle(); ?></title>
    </head>
    <body>
    <?php echo $helper->listSomeTable(); ?>
    <hr />
    <?php echo $helper->someOtherCrap();?>
    </body>
    </html>
```

Presentation Model

Transform View

- Sebuah view yang memproses domain data, elemen demi elemen dan mengubahnya ke dalam HTML.
- Proses **transformasi** hanya pada elemen atau bagian yang diperlukan saja.
- Contoh: data ditransformasikan ke dalam XML



Presentation Model (Transform View)

```
class AlbumCommand...

    public void process() {
        try {
            Album album = Album.findNamed(request.getParameter("name"));
            Assert.notNull(album);
            PrintWriter out = response.getWriter();
            XsltProcessor processor = new SingleStepXsltProcessor("album.xsl");
            out.print(processor.getTransformation(album.toXmlDocument()));
        } catch (Exception e) {
            throw new ApplicationException(e);
        }
    }
```

The XML document may look something like this:

```
<album>
    <title>Stormcock</title>
    <artist>Roy Harper</artist>
    <trackList>
        <track><title>Hors d'Oeuvres</title><time>8:37</time></track>
        <track><title>The Same Old Rock</title><time>12:24</time></track>
        <track><title>One Man Rock and Roll Band</title><time>7:23</time></track>
        <track><title>Me and My Woman</title><time>13:01</time></track>
    </trackList>
</album>
```

Presentation Model (Transform View)

- XSLT

```
<xsl:template match="album">
   <HTML><BODY bgcolor="white">
   <xsl:apply-templates/>
   </BODY></HTML>
</xsl:template>
<xsl:template match="album/title">
   <h1><xsl:apply-templates/></h1>
</xsl:template>
<xsl:template match="artist">
      <P><B>Artist: </B><xsl:apply-templates/></P>
</xsl:template>
```

```
<xsl:template match="trackList">
   <table><xsl:apply-templates/></table>
</xsl:template>
<xsl:template match="track">
   <xsl:variable name="bgcolor">
      <xsl:choose>
         <xsl:when test="(position() mod 2) = 1">linen</xsl:when>
         <xsl:otherwise>white</xsl:otherwise>
      </xsl:choose>
   </xsl:variable>
   <tr bgcolor="{$bgcolor}"><xsl:apply-templates/></tr>
</xsl:template>
<xsl:template match="track/title">
   <td><xsl:apply-templates/></td>
</xsl:template>
<xsl:template match="track/time">
   <td><xsl:apply-templates/></td>
</xsl:template>
```

Concurrency Problems

- Jika ada dua atau lebih proses/thread yang mengakses data yang sama
  - **Lost updates**: yang terakhir update yang akan mengupdate.
  - **Inconsistent read**: jika ada 2 atau lebih pembacaan, tapi salah satu ada yang menyimpan, maka data menjadi tidak konsisten
- Becomes **Race Condition**

Isolation

- **Optimistic locking**: semua dapat copy dari data, jika salah satu proses sudah mengupdate data, maka proses lain masih dibiarkan mengakes data sampai proses tersebut hendak mengupdate data, dan pada saat itu, akan dilakukan blocking. (conflict detection).
  - Deteksi dilakukan dengan version checker.
- **Pessimistic locking**: semua dapat copy dari data, jika sudah ada proses yang mengupdate data, maka semua proses lain langsung diblok, semua proses lain harus menunggu (conflict prevention) sampai disave dan dapat menyebabkan deadlock
  - Bagaimana cara mencegah dan mengatasi deadlock?

Transaction

- **Atomicity**: setiap langkah dalam suatu sekuens harus bergantian dan bersifat uninterrupable, jadi suatu kejadian harus selesai terlebih dahulu baru kejadian berikutnya.
- **Consistency**: sumber daya sistem harus konsisten, tidak boleh corrupt sampai selesai transaksi.
- **Isolation**: hasil dari individual transaction tidak boleh diketahui oleh proses lain, sampai berhasil di commit.
- **Durability**: hasil apapun yang sudah dicommit harus bersifat permanen.
  - Harus bisa mengatasi jika terjadi crash => recovery

Stateless & Stateful Server

- **Stateless server** berarti server tersebut tidak pernah menyimpan suatu data dari sebuah request yang diajukan kepadanya setelah proses response berakhir.
- **Stateful server** butuh untuk menjaga semua statenya sementara sedang menunggu seorang user menyelesaikan requestnya.

Session

- Kita membutuhkan server object yang berguna untuk menyimpan suatu data dari user dalam suatu waktu tertentu, yaitu **session**.
- Seorang user memiliki session yang berbeda-beda.
- Session berbeda dengan record data, session yang sudah dicommit baru disebut dengan record data.

Ways to Store Session State

- **Client session state** = menyimpan data di client.
  - Ex: hidden field, URL encoding, dan cookies.
  - Kelemahan: butuh transfer data dari client ke server, keamanan, dan integritas.
  - Hanya mungkin digunakan untuk data-data kurang penting dan kecil ukurannya.
- **Server session state** = menyimpan data di memori server pada saat request terjadi.
  - Ex: session object dengan session_id.
  - Harus ada mekanisme session_timeout.
  - Sulit menangani sistem crash.
- **Database session state**: menyimpan data di database.
  - Kompleks tapi mampu menangani sistem crash.

Putting all together

- **Domain logic**: Transaction Script, Table Module, and Domain Model
- **Data Source Layer**: Row data gateway, dan Table data gateway
- **Presentation**: MVC, Page Controller, Front Controller, Template View, dan Transform View
- **Technology specific**: J2EE or .NET dan Web Service.

NEXT

- Enterprise Application Integration & SOA