Web Development: FrontController

Created by *Lasse Jenssen*

(Based on material from Atle Geitung, 2021)

Home

Agenda: Back to Web Development

- Backend Web Developement.

- More about MVC.

- More about the design pattern "FrontController".

- Design pattern: Command , Application Controller.

- FrontController with FlowManager.

- Pure J2EE Servlets (later we'll take a look at Frameworks that build on J2EE).

Syllabus for this lecture

Web-tier Archecture (Chapters to - and including - 4.4.2.1.4 Example)

Link: http://fitxers.oriolrius.cat/1797/web-tier5.html

Demo: demo-01-several-controllers

Smal demo applications keeping track of Inventory (Items).

Code: demo-01-several-controllers.zip (see course overview)

Model: Item

```java
1  package no.hvl.dat152.model;
2
3  import java.io.Serializable;
4
5  public class Item implements Serializable {
6
7      private static final long serialVersionUID = 1L;
8
9      private String id;
10     private String name;
11     private Double price;
12     private String description;
13
14     public Item() {
15     }
16
17     public Item(final String id) {
18         this.id = id;
19     }
20
21     public Item(final String id, final String name, final Double price, final S
22         ...
23     }
24 }
```

Repository: interface ItemDAO

```java
1  package no.hvl.dat152.repositories;
2
3  import java.util.List;
4  import no.hvl.dat152.model.Item;
5
6  public interface ItemDAO {
7
8      default void init() {
9          createItem(new Item("9991", "Item01", 1D, "Item01 Description"));
10         createItem(new Item("9992", "Item02", 2D, "Item02 Description"));
11         createItem(new Item("9993", "Item03", 3D, "Item03 Description"));
12     }
13
14     List< Item > findAllItems();
15     Item findItem(String id);
16     void createItem(Item item);
17     void updateItem(String id, Item itemdata);
18     String getNextId();
19 }
```

Repository: ItemDAOMemorySingleton

```java
package no.hvl.dat152.repositories;

import java.util.*;
import no.hvl.dat152.model.Item;

public final class ItemDAOMemorySingleton implements ItemDAO {

    private final List< Item > items = new ArrayList< >();
    private static final Integer FIRST_INDEX = 10000;
    private Integer nextId = FIRST_INDEX;

    // Singleton-things
    private static ItemDAOMemorySingleton instance;

    private ItemDAOMemorySingleton() {}

    public static synchronized ItemDAOMemorySingleton getInstance() {
        if (instance == null) {
            instance = new ItemDAOMemorySingleton();
            instance.init();
        }
        return instance;
    }
    ...
```

Repository: ItemDAOMemorySingleton

```java
 1    ...
 2
 3    @Override
 4    public List< Item > findAllItems() {
 5        return items;
 6    }
 7
 8    @Override
 9    public Item findItem(final String id) {
10        final int index = items.indexOf(new Item(id));
11        return index >= 0 ? items.get(index) : null;
12    }
13
14    @Override
15    public synchronized void createItem(final Item item) {
16        final int index = items.indexOf(item);
17        if (index == -1) {
18            items.add(item);
19        }
20    }
21
22    ...
23 }
```

Repository: ItemDAOMemorySingleton

```java
    ...

    @Override
    public synchronized void updateItem(final String id, final Item itemdata) {
        final int index = items.indexOf(new Item(id));
        if (index >= 0) {
            items.get(index).setName(itemdata.getName());
            items.get(index).setPrice(itemdata.getPrice());
            items.get(index).setDescription(itemdata.getDescription());
        }
    }

    @Override
    public synchronized String getNextId() {
        nextId++;
        return nextId.toString();
    }
}
```

Mapping: src/main/webapp/WEB-INF/web.xml

```xml
1  < ?xml version="1.0" encoding="UTF-8"? >
2  < web-app >
3  < servlet >
4    < servlet-name >ViewItemController< /servlet-name >
5    < servlet-class >no.hvl.dat152.controller.ViewItemController< /servlet-clas
6  < /servlet >
7  < servlet >
8    < servlet-name >ViewShoppinglistController< /servlet-name >
9    < servlet-class >no.hvl.dat152.controller.ViewShoppinglistController< /serv
10 < /servlet >
11 ...
12 < servlet-mapping >
13   < servlet-name >ViewItemController< /servlet-name >
14   < url-pattern >/viewitem< /url-pattern >
15 < /servlet-mapping >
16 < servlet-mapping >
17   < servlet-name >ViewShoppinglistController< /servlet-name >
18   < url-pattern >/viewshoppinglist< /url-pattern >
19 < /servlet-mapping >
20 ...
21 < /web-app >
```

Controller: ViewShoppinglistController

```java
1  package no.hvl.dat152.controller;
2  ...
3  import no.hvl.dat152.model.Item;
4  import no.hvl.dat152.repositories.ItemDAOMemorySingleton;
5
6  public class ViewShoppinglistController extends HttpServlet {
7
8      private static final long serialVersionUID = 1L;
9
10     @Override
11     protected final void doGet(final HttpServletRequest req,
12                                final HttpServletResponse resp)
13         throws ServletException, IOException {
14
15         final List< Item > items =
16             ItemDAOMemorySingleton.getInstance().findAllItems();
17         req.getSession().setAttribute("items", items);
18         req.getRequestDispatcher("shoppinglist.jsp").forward(req, resp);
19     }
20 }
```

View: src/main/webapp/shoppinglist.jsp

```jsp
<%@ page contentType="text/html"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
<body>
  <%@include file="myHeader.html"%>
  <p><a href="createitem">Create item</a></p>

  <table border=1>
  <tr>
    <th>Id</th>
    <th>Name</th>
    <th>Price</th>
    <th>Description</th>
  </tr>
  <c:forEach var="item" items="${items}">
    <tr>
      <td>${item.id}</td>
      <td>${item.name}</td>
      <td>${item.price}</td>
      <td>${item.description}</td>
      <td><a href="viewitem?id=${item.id}">View item</a></td>
    </tr>
  </c:forEach>
  </table>

</body>
</html>
```
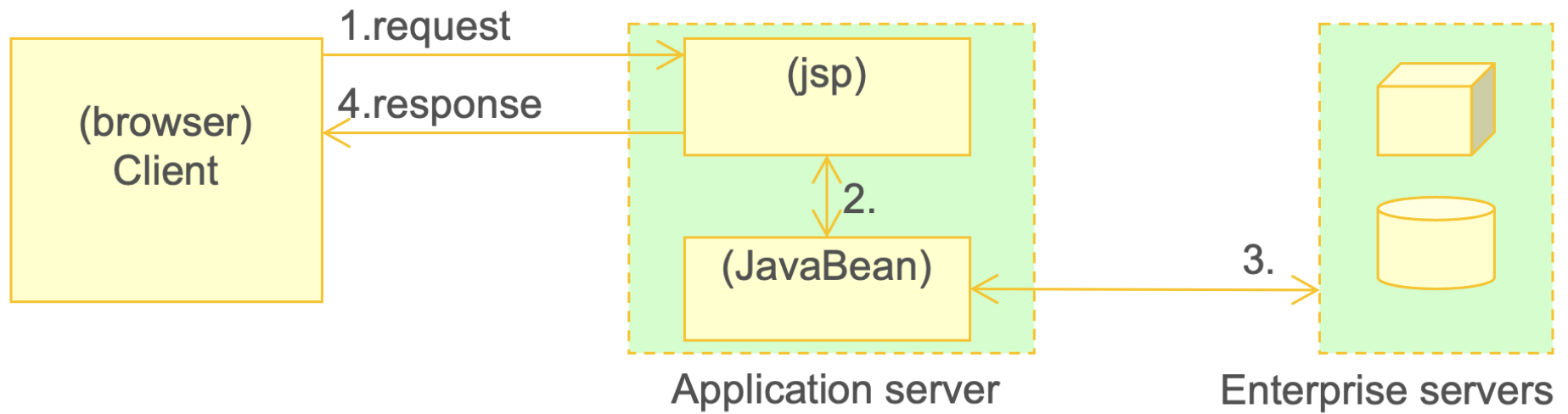
Demo: demo-01-several-controllers

Let's run the code.

MVC: Model 1 vs Model 2

Model 1

- Decentralized and page-centric architecture
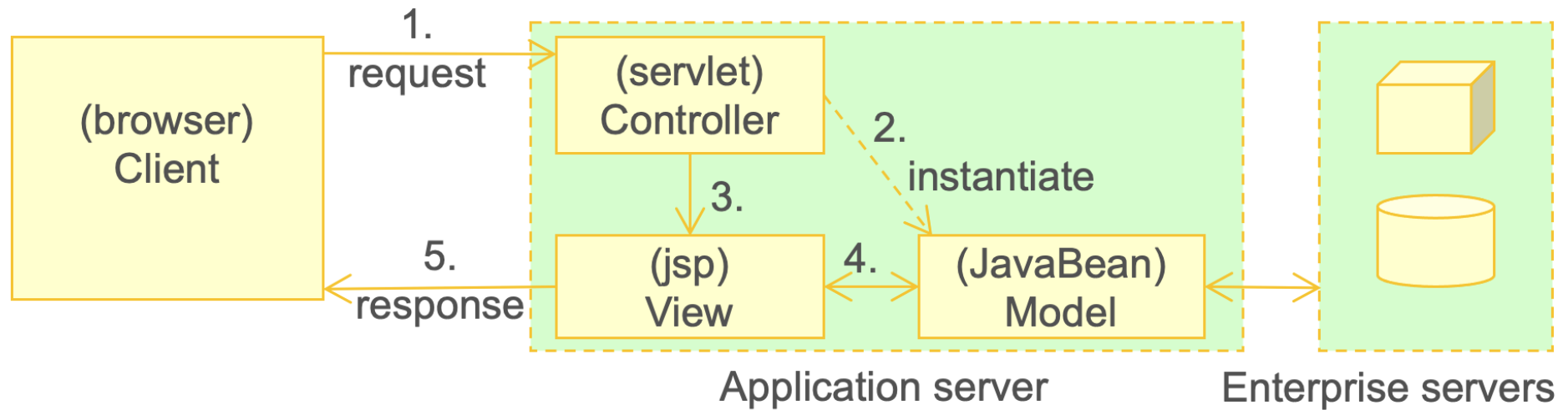- No controller
- Map directly to the next JSP

Model 2

- Centralized architecture
- Requests goes through Controller (Servlet)
- Controller determines next view (JSP)

Web MVC: Model 1

Web MVC: Model 2

Recap: How we have done web development so far

- Used MVC in a special way: One controller per application (or page)

- Used *web.xml* for static dispatching av requests.

- Our earlier architecture is most simular to Model 1: decentralized.

- A little similar to Model 2: used controller Servlets and made the MVC for each use case.

Issues with this method:

- Hard to find a good location for key tasks
  - General controller logic
  - Checking headers and cookies
  - Authentication and authorization
  - Logging
- End up writing a lot of code which is bundled to and dependant on the Servlet API.

Design Pattern: FrontController

- A design pattern dealing with centralization of processing of requests and selections of views in a single component (the frontcontroller).
- The application gets a single access point where all requests go through.
- The wanted command is provided either as part of the URL, or as parameters in the request.
  Examples:
    - http://mittdomene/minapp/front?cmd=visansatte
    - http://mittdomene/minapp/front/visansatte
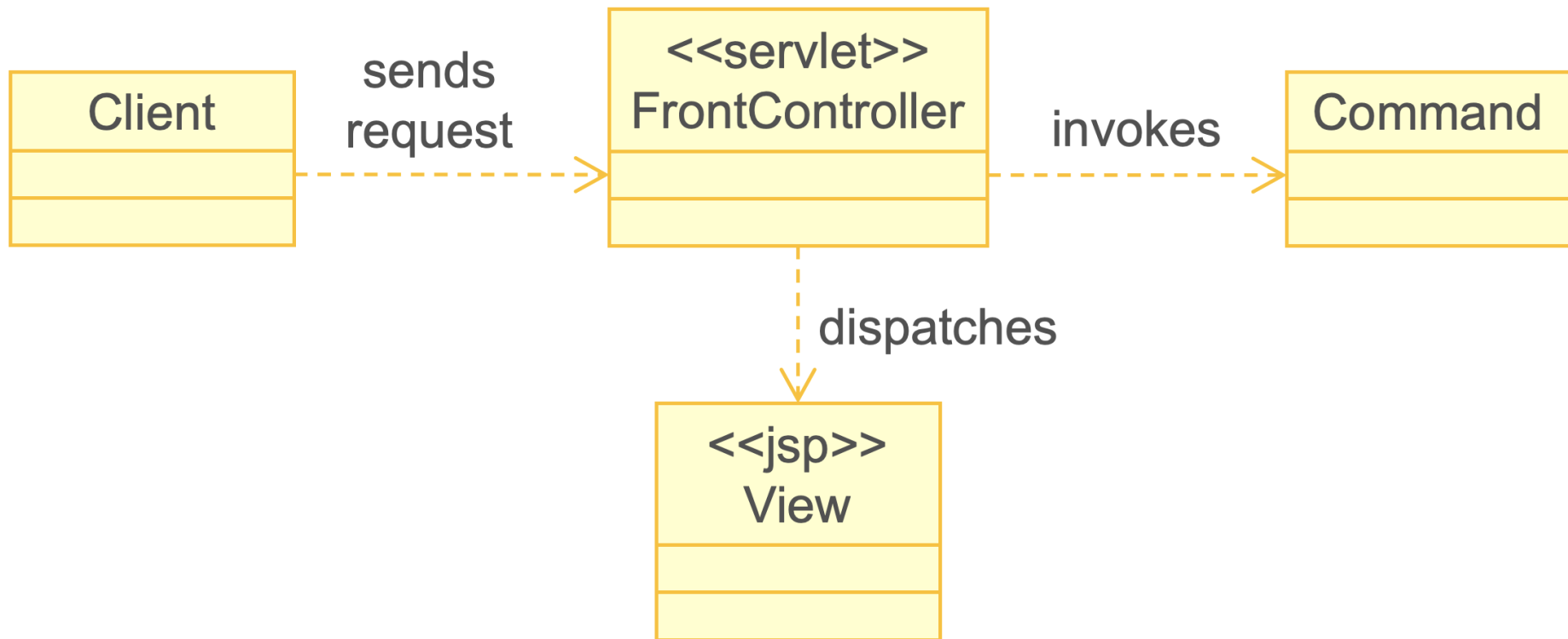
Implementing the FrontController

- The FrontController should be able to process many types of requests.

- First we'll look at a "simple" Front Controller

- It can, for example look like this:

```
1  if (cmd.equals ("/viewshoppinglist")) {
2  ... Doing all the work here
3  } else if (cmd.equals("/viewitem")) {
4  ... Doing all the work here
5  } else if ... etc. ...
```

Demo: demo-02-front-controller

- Same functionallity as "demo-01".

- One Controller: FrontController with if-then-else.

- *Code: demo-02-front-controller.zip (see course overview).*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>front-controller</servlet-name>
        <servlet-class>no.hvl.dat152.controller.FrontController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>front-controller</servlet-name>
        <url-pattern>/do/*</url-pattern>
    </servlet-mapping>
</web-app>
```

```java
@Override
protected final void doGet(final HttpServletRequest req, final HttpServletResp
        throws ServletException, IOException {

    final String cmd = req.getPathInfo();
    //System.out.println("Command: " + cmd);

    if (cmd.equals("/viewshoppinglist")) {
        viewShoppinglist(req, resp);
    } else if (cmd.equals("/viewitem")) {
        viewItem(req, resp);
    } else if (cmd.equals("/updateitem")) {
        updateItemForm(req, resp);
    } else if (cmd.equals("/updateitemsave")) {
        updateItemSave(req, resp);
    } else if (cmd.equals("/createitem")) {
        createItemForm(req, resp);
    } else if (cmd.equals("/createitemsave")) {
        createItemSave(req, resp);
    } else {
        viewShoppinglist(req, resp);
    }
}
```

```java
@Override
protected final void doPost(final HttpServletRequest req,
                               final HttpServletResponse resp)
    throws ServletException, IOException {
  doGet(req, resp);
}
```
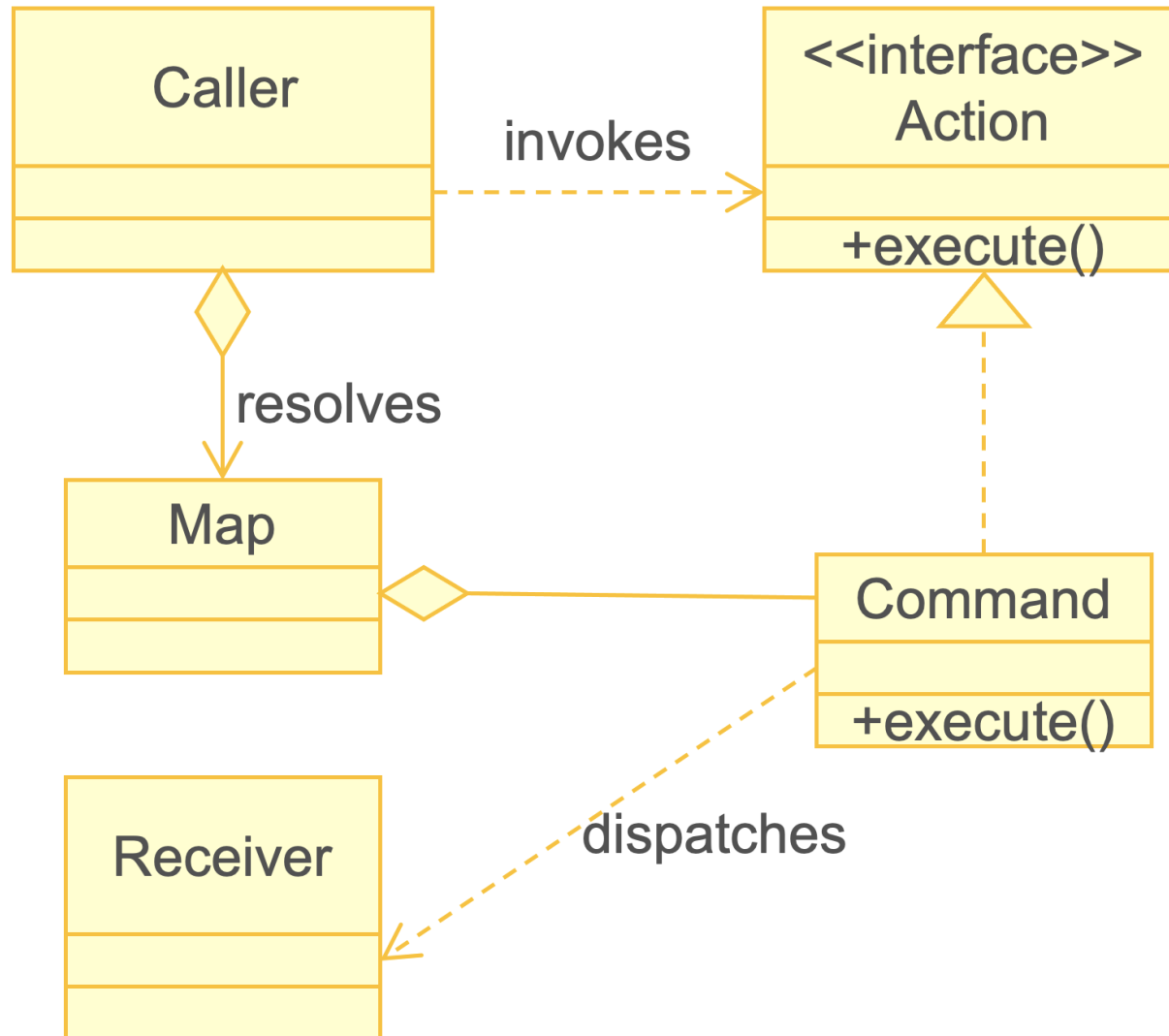
Demo: demo-02-front-controller

- *Let's have a look in Eclipse.*

Design Pattern: Command

- A better solution than if-elseif-else is to use a Command design pattern.
- **Purpose**:
    - Encapsulate a command (for instance "createItem") with associated data and business logic as an object ...
    - .. , and use polymorphism instead of if-elseif-else to perform the right command.
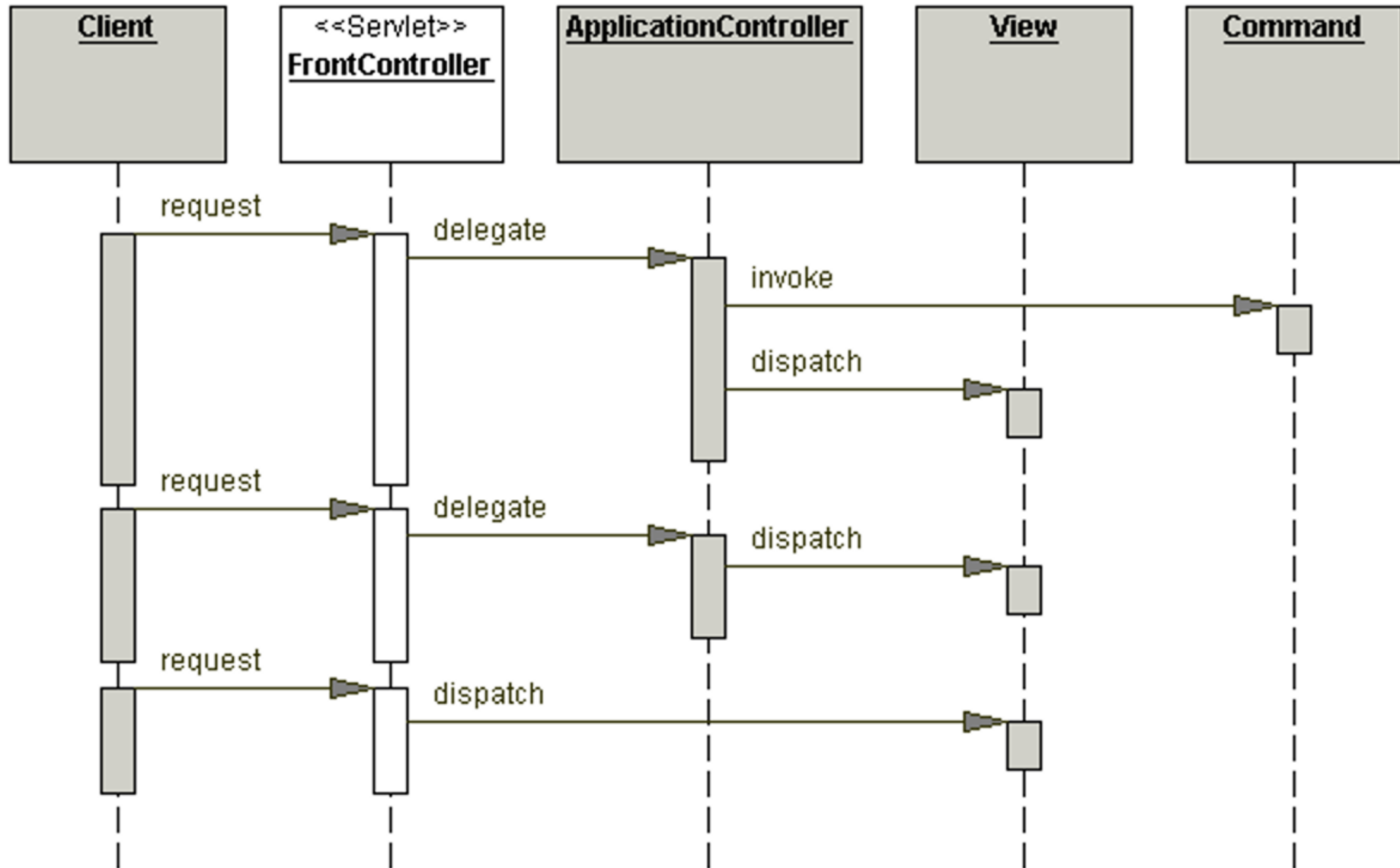- Let us: **Decouple** objects that produce the commands from their consumers

Design Pattern: Command

Design Pattern: Application Controller
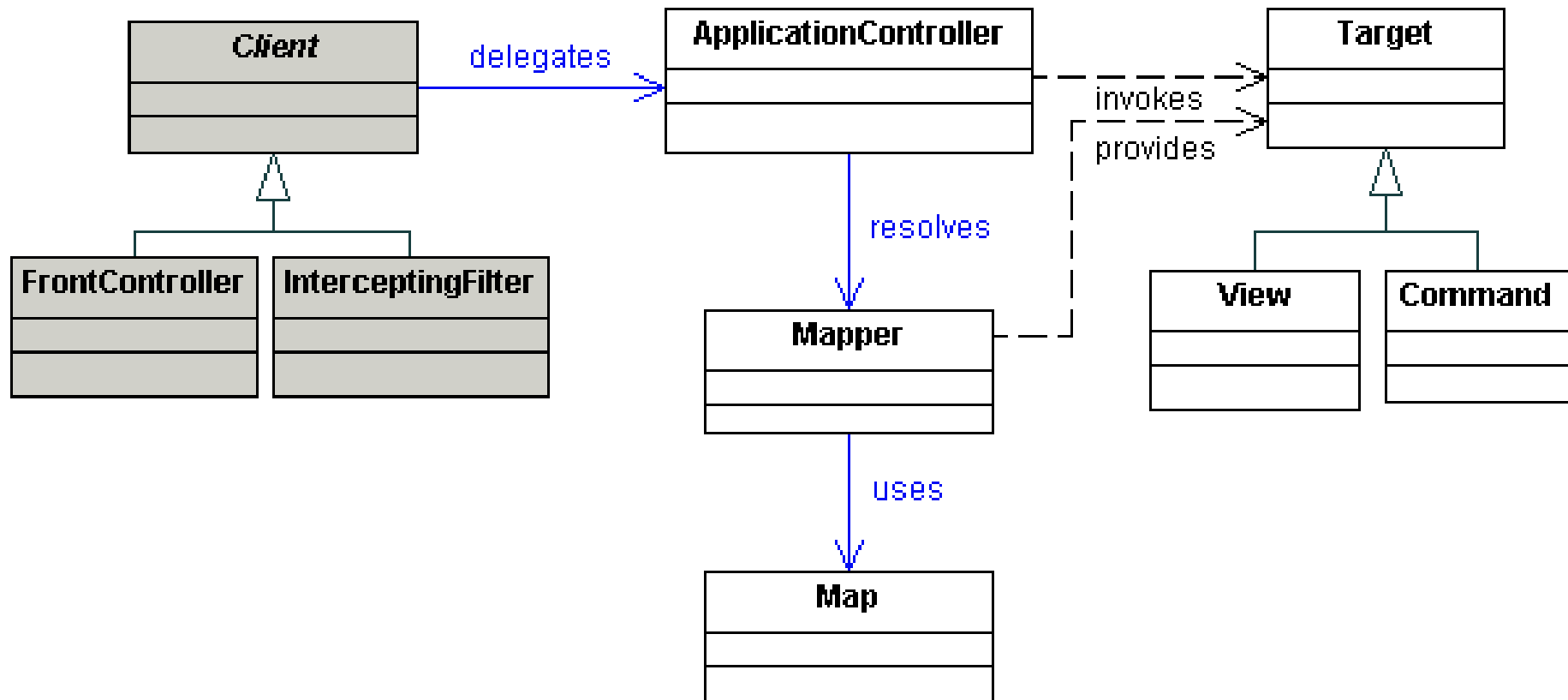
Design Pattern: Application Controller

Demo: demo-03-command-pattern

- Same functionallity as "demo-01".

- Applicaton Logic moved out of FrontController.

- *Code: demo-03-command-pattern.zip (see course overview).*

```java
public class FrontController extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected final void doGet(final HttpServletRequest req,
                              final HttpServletResponse resp)
            throws ServletException, IOException {

        String cmd = req.getPathInfo();
        if (cmd == null || cmd.length() == 1) {
            return;
        }
        cmd = cmd.substring(1);

        final Action action = ActionMapper.mapToAction(cmd);
        action.execute(req, resp);
    }

    @Override
    protected final void doPost(final HttpServletRequest req, final HttpServlet
            throws ServletException, IOException {
        doGet(req, resp);
    }
```

```java
public interface Action {

    void execute(HttpServletRequest req, HttpServletResponse resp)
                                throws ServletException, IOException;

}
```

```java
1  package no.hvl.dat152.action;
2
3  import java.io.IOException;
4  ...
5
6  public class ViewShoppingListAction implements Action {
7
8      @Override
9      public final void execute(final HttpServletRequest req,
10                                    final HttpServletResponse resp)
11          throws ServletException, IOException {
12
13          final List< Item > items =
14                      ItemDAOMemorySingleton.getInstance().findAllItems();
15          req.getSession().setAttribute("items", items);
16
17          req.getRequestDispatcher("/shoppinglist.jsp").forward(req, resp);
18      }
19  }
```

```java
package no.hvl.dat152.action.mapper;

import no.hvl.dat152.action.Action;

public class ActionMapperType {
    private String name;
    private Action action;

    ActionMapperType(String name, Action action) {
        this.name=name;
        this.action=action;
    }

    public String getName() {
        return name;
    }

    public Action getAction() {
        return action;
    }
}
```

```java
public class ActionMapper {

    private static List< ActionMapperType > actionMapperTypeList =
        new ArrayList< >(Arrays.asList(
            new ActionMapperType("viewshoppinglist", new ViewShoppingListAction()
            new ActionMapperType("viewitem", new ViewItemAction()),
            new ActionMapperType("updateitem", new UpdateItemFormAction()),
            new ActionMapperType("updateitemsave", new UpdateItemSaveAction()),
            new ActionMapperType("createitem", new CreateItemFormAction()),
            new ActionMapperType("createitemsave", new CreateItemSaveAction())
        ));

    private static Map< String, ActionMapperType > mapActionType =
            actionMapperTypeList.stream().collect(
                Collectors.toMap(ActionMapperType::getName, Function.identity()));

    public static Action mapToAction(String name) {
        try {
            return (Action) mapActionType.get(name).getAction();
        } catch (Exception e) {
            String x = "";
        }
        return null;
    }
```

Demo: demo-03-command-pattern

- *Let's have a look in Eclipse.*

Demo: demo-03-command-pattern

- So ... now we have centraliced the dispatching of commands.

- But the page flow is still decentralized

  (every action determines what the next page is).

- Next: How to use a FlowManager to keep track of the page flow.

Demo: demo-04-front-controller-flow

- Same functionallity as "demo-01".

- FlowManager: Centralized Page Flow

- *Code: demo-04-front-controller-flow.zip (see course overview).*

```java
public class ViewShoppingListAction implements Action {

    @Override
    public final void execute(final HttpServletRequest req,
                                final HttpServletResponse resp)
            throws ServletException, IOException {

        final List< Item > items = ItemDAOMemorySingleton.getInstance()
                                                    .findAllItems();
        req.getSession().setAttribute("items", items);

        req.getRequestDispatcher("/shoppinglist.jsp").forward(req, resp);
    }
}
```

```java
1  public class ViewShoppingListAction implements Action {
2
3      @Override
4      public final int execute(final HttpServletRequest req,
5                               final HttpServletResponse resp)
6          throws ServletException, IOException {
7
8          final List< Item > items = ItemDAOMemorySingleton.getInstance()
9                                                     .findAllItems();
10         req.getSession().setAttribute("items", items);
11
12         return Action.SUCCESS;
13     }
14 }
```

doGet(): from last FrontController

```java
@Override
protected final void doGet(final HttpServletRequest req,
                          final HttpServletResponse resp)
                throws ServletException, IOException {

    String cmd = req.getPathInfo();

    if (cmd == null || cmd.length() == 1) {
        return;
    }
    cmd = cmd.substring(1);

    final Action action = ActionMapper.mapToAction(cmd);
    action.execute(req, resp);
}
```

doGet(): New FrontController

```java
@Override
protected final void doGet(final HttpServletRequest req,
                          final HttpServletResponse resp)
                throws ServletException, IOException {

    final String cmd = getCommand(req);
    final Action action = ActionMapper.mapToAction(cmd);

    final int result = action.execute(req, resp);

    if (result == Action.SUCCESS) {
        final String nextPage = flowManager.getNextPage(cmd);
        req.getRequestDispatcher(nextPage).forward(req, resp);
    } else {
        // ...
    }
}
```

```java
private String getCommand(final HttpServletRequest req) {

    String cmd = req.getPathInfo();

    if (cmd == null || cmd.length() <= 1) {
        cmd = "";
    } else {
        cmd = cmd.substring(1);
    }

    return cmd;
}
```

```java
public class FlowManager {

    private final Map< String, String > pages;

    public FlowManager() {
        pages = new HashMap< >();
        pages.put("viewshoppinglist", "../shoppinglist.jsp");
        pages.put("viewitem", "../item.jsp");
        pages.put("updateitem", "../updateitemform.jsp");
        pages.put("updateitemsave", "../item.jsp");
        pages.put("createitem", "../createitemform.jsp");
        pages.put("createitemsave", "../shoppinglist.jsp");
    }

    public final String getNextPage(final String cmd) {
        return pages.get(cmd);
    }
}
```

Demo: demo-04-front-controller-flow

- *Let's have a look in Eclipse.*

Summary: Web Development: FrontController

Where are we now?

- Moved towards pure MVC - Model 2.

- FrontController Pattern: Centralized control and common logic.

- Command Pattern: Business logic in regular classes.

- FlowManager: Centralized controll of page flow.

Web Development: Frameworks

Home