

# Web 2

Les 4: MVC

# AGENDA

- ❑ Herhaling
- ❑ MVC
- ❑ Packages



# GET



```

@WebServlet("/Guess")
public class GuessServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private int number = new Random().nextInt(10) + 1;

    protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        String guessFromParameter = request.getParameter("guess");
        int guessedNumber = Integer.parseInt(guessFromParameter);

        String resultMessage = null;
        if(guessedNumber == number) {
            resultMessage = "Well done! That was correct!";
            number = new Random().nextInt(10) + 1;
        } else if (guessedNumber > number) {
            resultMessage = "Lower...";
        } else {
            resultMessage = "Higher...";
        }

        request.setAttribute("result", resultMessage);
        RequestDispatcher view = request.getRequestDispatcher("result.jsp");
        view.forward(request, response);
    }
}

```

resultaat meegeven aan request

forward naar JSP

GuessServlet.java

De JSP zal dan de HTML opbouwen.

Hij krijgt de request mee, dus ook de parameter 'guess', maar hij heeft ook de gegenereerde boodschap nodig. Wij kunnen zelf géén extra parameters toevoegen aan onze request! We kunnen wel extra **attributen** meegeven met de methode `setAttribute()`. We geven aan deze methode een naam mee voor het attribuut en de inhoud.

```

<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>

<!DOCTYPE html>
<html>
<head>
    <title>Guess</title>
    <link rel="stylesheet" href="css/sample.css">
</head>
<body>
    <h1>Thinking a number</h1>
    <p>You guessed <%=request.getParameter("guess")%>...</p>
    <h2><%= request.getAttribute("result") %></h2>
    
</body>
</html>

```

parameter opvragen

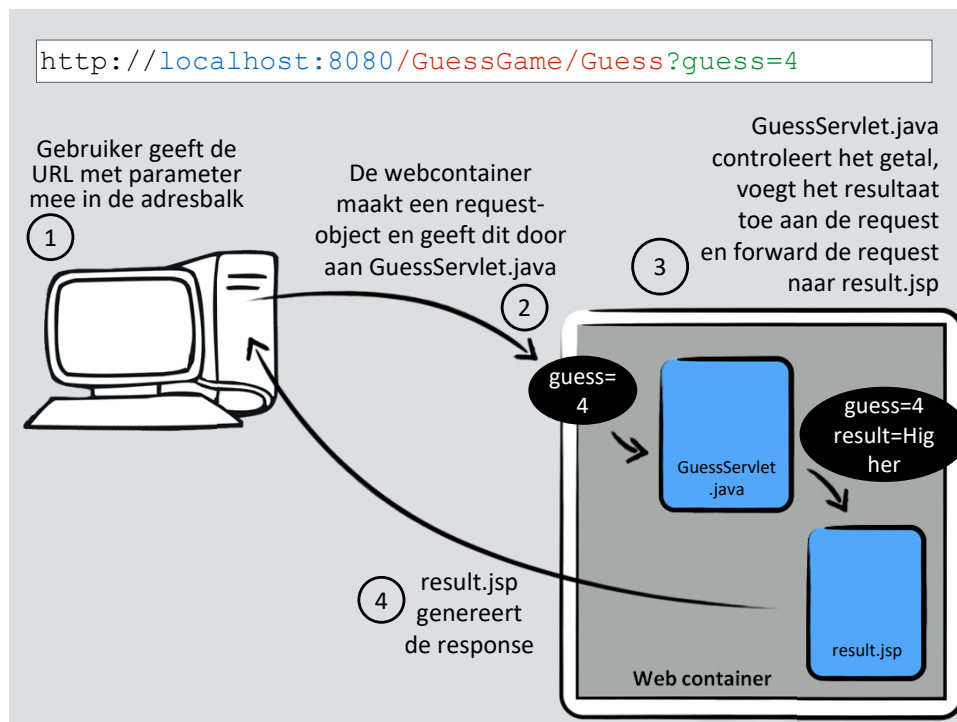
attribuut opvragen

result.jsp

Net zoals we met de methode `getParameter()` een parameter kunnen opvragen, kunnen we de methode `getAttribute()` aan het request-object een attribuut opvragen, waarbij de naam van het attribuut dat men wilt opvragen meegegeven wordt.

De JSP pagina hoeft niets meer te berekenen, hij vraagt gewoon alles wat hij wilt tonen aan het request object. Hierdoor wordt het bestand

overzichtelijker, en beter herbruikbaar.




Beste oplossing:

- alle bestanden zijn herbruikbaar
- servlet bevat geen html
- jsp bevat een minimum aan Java-code.

URL bevat niet meer Guess.jsp, maar Guess: de url die we in de @WebServlet annotatie in GuessServlet hadden gezet.

# AGENDA

- 
- ☒ Herhaling
  - ☐ MVC
  - ☐ Packages



```

@WebServlet("/Guess")
public class GuessServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private int number = new Random().nextInt(10) + 1;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String guessFromParameter = request.getParameter("guess");
        int guessedNumber = Integer.parseInt(guessFromParameter);

        String resultMessage = null;
        if(guessedNumber == number) {
            resultMessage = "Well done! That was correct!";
            number = new Random().nextInt(10) + 1;
        } else if (guessedNumber > number) {
            resultMessage = "Lower...";
        } else {
            resultMessage = "Higher...";
        }

        request.setAttribute("result", resultMessage);
        RequestDispatcher view = request.getRequestDispatcher("/view");
        view.forward(request, response);
    }
}

import java.util.Random;

public class GuessGame {
    private int number = thinkNumber();

    public String guess(int guessedNumber){
        String message = null;
        if(guessedNumber == number) {
            message = "Well done! That was correct!";
            number = thinkNumber();
        } else if (guessedNumber > number) {
            message = "Lower...";
        } else {
            message = "Higher...";
        }
        return message ;
    }

    private int thinkNumber() {
        return new Random().nextInt(10) + 1;
    }
}

```

**DRY !**

Dubbele code —> don't repeat yourself!

We zouden beter de klasse die we al hadden hergebruiken.

```
import java.util.Random;

public class GuessGame {
    private int number = thinkNumber();

    public String guess(int guessedNumber){
        String message = null;
        if(guessedNumber == number) {
            message = "Well done! That was correct!";
            number = thinkNumber();
        } else if (guessedNumber > number) {
            message = "Lower...";
        } else {
            message = "Higher...";
        }
        return message ;
    }

    private int thinkNumber() {
        return new Random().nextInt(10) + 1;
    }
}
```

GuessGame.java

Deze klasse bestond al...

```

@WebServlet("/Guess")
public class GuessServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private GuessGame game = new GuessGame();

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        String guessFromParameter = request.getParameter("guess");
        int guessedNumber = Integer.parseInt(guessFromParameter);

        request.setAttribute("result", game.guess(guessedNumber));
        RequestDispatcher view = request.getRequestDispatcher("result.jsp");
        view.forward(request, response);
    }
}

```

## GuessServlet.java

In plaats van de spel logica te kopiëren naar de servlet, kunnen we beter de bestaande klasse gebruiken.

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>




<!DOCTYPE html>
<html>
<head>
    <title>Guess</title>
    <link rel="stylesheet" href="css/sample.css">
</head>
<body>
    <h1>Thinking a number</h1>
    <p>You guessed <%=request.getParameter("guess")%>...</p>
    <h2><%= request.getAttribute("result") %></h2>
    
</body>
</html>
```

result.jsp

De jsp pagina hadden we al opgekuist in vorige stap, die hoeven we dus niet meer te veranderen...

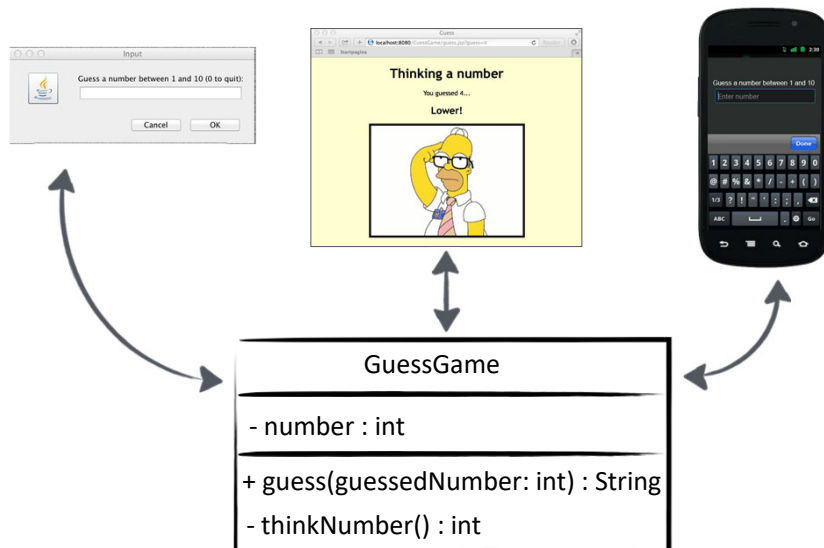
# MODEL VIEW CONTROLLER

“MVC scheidt de *businesslogica* van de *user interface* en laat zo toe om elk onafhankelijk van elkaar te ontwikkelen, testen en onderhouden.”

- **Model:** alle data, toestand en businesslogica  `GuessGame.java`
- **View:** visuele representatie van model  `result.jsp`
- **Controller:** reageert op de input van de gebruikers en stuurt de communicatie tussen view en model  `GuessServlet.java`

Deze werkwijze wordt Model - View - Controller (MVC) genoemd.

# VOORDEEL MVC



Voordeel: de klassen die de logica bevatten (= de kern van de applicatie) kunnen herbruikt worden in een

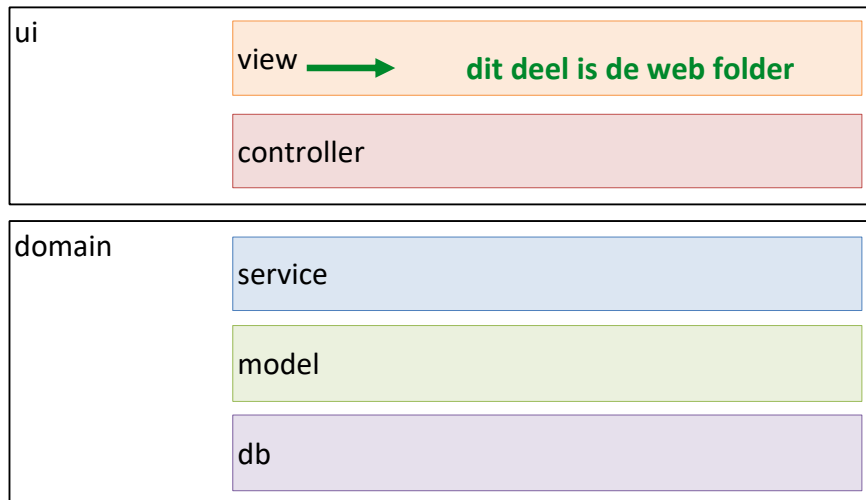
- desktop applicatie
- web applicatie
- mobiele applicatie

# AGENDA

- ☒ Herhaling
- ☒ MVC
- ☐ Packages



# PACKAGES



In Web 2:

- ui.view: wordt niet gebruikt (want jsp-bestanden in web folder)
- ui.controller: servlet
- domain.model: individuele items
- domain.db: geheel van items (bewaard in ArrayList; methodes: voegToe(), getAll(), mostPopular(), ...)



# SELENIUM TESTEN

- In een aparte src folder die we test noemen en daar dezelfde package structuur gebruiken
- Selenium testen komen in het package ui.view
- JUnit testen voor domain klassen komen in het package domain.model

# AGENDA



## INSTRUCTIES

- ☒ Analyseer de **HTTP error code**
- ☒ Ontcijfer de **logfile**
- ☒ **Debug**
- ☒ Bekijk de code in **View Page Source**
- ☒ Bekijk de request/response in **Chrome**

Niet vergeten...

## INSTRUCTIES

- ☒ Geen HTML in Java
- ☒ Zo weinig mogelijk Java in HTML

# DEMO

[https://github.com/UCLLWeb2-2122/Demo MVC POST](https://github.com/UCLLWeb2-2122/Demo_MVC_POST)