# B7. Web-Programmierung mit Java

B7.1 Applets 

B7.2 Servlets

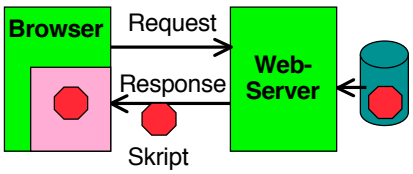
B7.3 Java Server Pages (JSP)

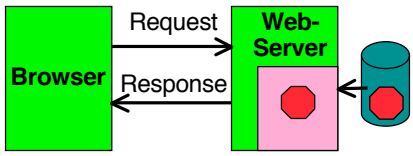
Literatur:

Siehe http://java.sun.com/applets



# Serverseitige vs. clientseitige Dynamik




- Clientseitige Dynamik:
  - Browser enthält Ausführungsmaschine für Programme
  - Programm ist Teil der Antwort vom Server
  - Beispiele: JavaScript, Java Applets
- Serverseitige Dynamik:
  - Web-Server enthält Ausführungsmaschine für Programme
  - Programm wird vor Beantwortung der Anfrage ausgeführt und liefert HTML-Text
  - Beispiele: PHP, Java Servlets, JSP

# Applets

- *Applet:*
  - "application snippet"
  - Java-Programm, das in eine HTML-Seite eingebettet ist
  - Wird in einem Browser ausgeführt
  - Dazu muss der Browser Java unterstützen
    - » direkt oder über *plugin*
  - Enthält keine main-Methode
- *Application:*
  - *Stand-alone* Java-Programm
  - Enthält eine statische main-Methode

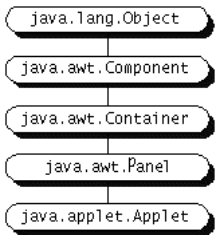


# Beispiel: Hello-World Applet (1)

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
            g.setFont(new Font("SansSerif", Font.PLAIN, 48));
            g.drawString("Hello world!", 50, 50);
    }
}
```

- Eigene Applet-Klasse abgeleitet von **`Applet`**
- **`Applet`** abgeleitet von **`Component`**
  - Deshalb wird **`paint`**-Methode aufgerufen und kann überdefiniert werden



## Beispiel: Hello-World Applet (2)

```
<html>
  <head>
     <title> Hello World </title>
  </head>
  <body>

Das Hello-World Beispiel-Applet wird ausgef&uuml;hrt:
     <br>
     <applet code="HelloWorldApplet.class" width=300>
     </applet>

  </body>
</html>
```



## Parameterübergabe in HTML

Applet:

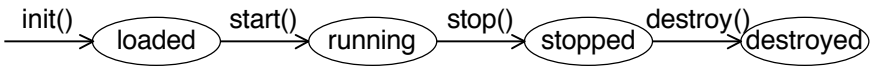
```
public class HelloWorldAppletParam extends Applet {

    public void paint(Graphics g) {
     String zt = getParameter("Zwischentext");
     g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello "+zt+" world!", 50, 50);
    }
}
```

HTML:

```
<html>
  ...
     <br>
     <applet code="HelloWorldAppletParam.class"
          width="800">
          <param name="Zwischentext" value="wonderful">
     </applet>
  ...
</html>
```

## Applet-Lebenszyklus




Callback-Methoden:

```
public class ... extends Applet {
    . . .
    public void init() { . . . }
    public void start() { . . . }
    public void stop() { . . . }
    public void destroy() { . . . }
    . . .
}
```



## Interaktion in Applets

- Applets können auf Benutzereignisse reagieren
  - Ereignishandler definieren
  - In der Applet-Initialisierung registrieren
- Applets haben als lokal ausgeführter Code vollen Zugriff auf die Benutzerinteraktion
  - Bewegungen, Tastendrücke, …
  - Das ist bei serverseitigem Code nicht möglich!
- Applets haben alle Möglichkeiten der Grafikprogrammierung
  - Siehe Java 2D und Java 3D
  - Das ist bei serverseitigem Code nicht möglich!

## Beispiel: Maus-Interaktion in Applets

```
public class ClickMe extends Applet implements MouseListener {
    private Point spot;
    private static final int RADIUS = 7;

    public void init() {
      addMouseListener(this);
    }

    public void paint(Graphics g) {
      . . .
      g.setColor(Color.red);
      if (spot != null) {
            g.fillOval(spot.x - RADIUS, spot.y - RADIUS,'
            RADIUS * 2, RADIUS * 2);
        }
    }

    public void mousePressed(MouseEvent event) {
      if (spot == null)
            spot = new Point();
      spot.x = event.getX();
      spot.y = event.getY();
      repaint();
    }
    . . .
}
```



## Swing-Applets

- Klasse **`javax.swing.JApplet`**
  - Ist von **`Applet`** abgeleitet
  - Ist gleichzeitig ein top-level Swing Container
- Alle Swing-GUI-Komponenten können eingesetzt werden
- Besonderheiten von Swing-Applets:
  - Besitzen verschiedene *Panes*
  - Layout-Manager und Sub-Komponenten immer an die **`ContentPane`** anfügen (wie bei **`JFrame`**)
  - Default-Layout-Manager ist **`BorderLayout`**
  - Direkte Grafikoperationen auf Swing-Applets sind nicht zu empfehlen
  - **`paintComponent`**-Methode überdefinieren
  - Mindestens:

```
public void paintComponent(Graphics g){
   super.paintComponent(g);
   . . .
}
```

# Beispiel: Counter als Swing-Applet (1)

```
public class CounterSwingApplet extends JApplet {

    CounterPanel counterPanel;

    public void init() {
      counterPanel = new CounterPanel();
      getContentPane().add(counterPanel);
  }
}

// The View
class CounterPanel
      extends JPanel implements Observer {

   private Counter ctr;

  JPanel valuePanel = new JPanel();
  JTextField valueDisplay = new JTextField(10);

  JButton countButton = new JButton("Count");
  JButton resetButton = new JButton("Reset");
  JPanel buttonPanel = new JPanel();

  . . .
```




# Beispiel: Counter als Swing-Applet (2)

```
  public CounterPanel () {          class CounterPanel (Forts.)

      ctr = new Counter();
      valuePanel.add(new Label("Counter value"));
      . . .
      add(valuePanel,BorderLayout.NORTH);

      countButton.addActionListener(new ActionListener() {
          public void actionPerformed (ActionEvent event) {
              ctr.count();
          }
      });
      . . .
      ctr.addObserver(this);
   }

   public void update (Observable o, Object arg) {
      valueDisplay.setText(String.valueOf(ctr.getValue()));
   }

   public void paintComponent(Graphics g){
       super.paintComponent(g);
  }
}

class Counter extends Observable { . . . }
```

## Konversion einer Swing-Anwendung in ein Applet

- **`JFrame`** durch **`JApplet`** ersetzen
  - oder ein **`JPanel`** innerhalb des **`JApplet`**
  - Content-Pane-Operationen auf das **`JApplet`** verlagern
  - **`JFrame`**-spezifische Operationen entfernen (z.B. **`setTitle()`, `setVisible()`, `pack()`**)
  - Window-Listener entfernen
  - Exit-Operationen entfernen
- **`paintComponent`**-Methode hinzufügen
- **`init()`**-Methode hinzufügen, ersetzt Hauptprogramm



## Anwendung und Applikation

- Durch Einfügen einer main()-Methode kann ein Applet auch als stand-alone-Anwendung aufrufbar sein:

```
public class Cubic extends JApplet{
    static protected JLabel label;
    CubicPanel cubicPanel;
    public void init(){
      getContentPane().setLayout(new BorderLayout());
      cubicPanel = new CubicPanel();
      getContentPane().add(cubicPanel);
      label = new JLabel("Drag the points to adjust the curve.");
      getContentPane().add("South", label);
    }
    public static void main(String s[]) {
        JFrame f = new JFrame("Cubic");
        CubicPanel cubicPanel = new CubicPanel();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
                {System.exit(0);}
        });
        JApplet applet = new Cubic();
        f.getContentPane().add(applet, BorderLayout.CENTER);
        applet.init();
        f.setSize(new Dimension(350,250));
        f.setVisible(true);
    }
}
```

## Organisation von Bytecode-Dateien

- Beim **<applet>**-Tag sind möglich
  - Angabe eines Verzeichnisses (Codebase)
  - Angabe von (JAR-)Archiven
- Vorteile von Codebase:
  - Java-Bytecode kann an einer Stelle konzentriert werden
  - Java-gerechtere Dateistruktur
- Vorteile von Archiven:
  - Weniger Dateien, weniger HTTP-Verbindungen, bessere Performance
  - Geringere Übertragungsanforderungen wegen (LZW-)Kompression



## Applets und Nebenläufigkeit

- Ein Applet soll schnell laden und schnell sinnvolle Inhalte anzeigen
  - Zeitaufwändige Aktionen in separate Threads (z.B. Laden von Bildern)
  - Idee;

```
//OLD CODE:
public void actionPerformed(ActionEvent e) {
    ...
    //...code that might take a while to execute is here...
    ...
}

//BETTER CODE:
public void actionPerformed(ActionEvent e) {
    ...
    final SwingWorker worker = new SwingWorker() { //separate Thread
        public Object construct() {
            //...code that might take a while to execute is here...
            return someValue;
        }
    };
    worker.start(); ...
}
```

- Swing erwartet alle Swing-relevanten Methoden in *einem* Thread
  - **javax.swing.SwingUtilities.invokeLater(), invokeAndWait()**

## Sicherheit bei Applets

- Dinge, die ein Applet nicht darf („Sandbox security“):
  - Netzverbindungen eröffnen (außer zum Host, von dem es kommt)
  - Programm auf dem Client starten
  - Dateien auf dem Client lesen oder schreiben
  - Bibliotheken laden
  - “Native” methoden (z.B. in C programmiert) aufrufen
- „Trusted“ Applets
  - Lokal auf dem Client installiert oder
  - digital signiert und verifiziert
  - Einschränkungen teilweise aufgehoben, z.B. Dateizugriff



## Vor- und Nachteile von Java Applets

- Vorteile:
  - Interaktion
  - Grafikunterstützung
  - Entlastung des Netzes bei häufiger Interaktion
  - Dezentrale Ausführung (skalierbar auf sehr viele Nutzer)
- Nachteile:
  - Abhängigkeiten von Browser, Java-Plugin, Java-Version
  - Generell relativ störanfällig
  - Unbequemer im Debugging

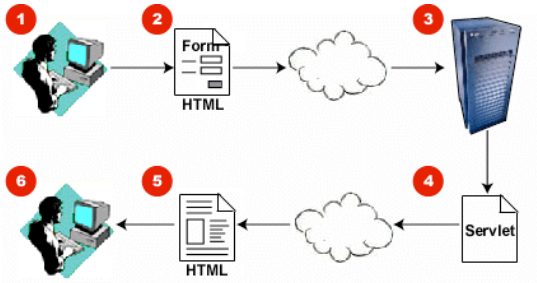
# B7. Web-Programmierung mit Java

B7.1 Applets
B7.2 Servlets 
B7.3 Java Server Pages (JSP)

Literatur:
http://java.sun.com/products/servlet/docs.html
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
http://jakarta.apache.org/tomcat/



# Grundprinzip: Server-seitige Berechnung dynamischer Webseiten




1. Benutzer füllt Formular aus
2. Formular wird als HTTP-Request verschickt
3. Server bestimmt Servlet und führt es aus
4. Servlet berechnet HTML-Text
5. Antwort wird an Browser gesendet
6. Benutzer erhält Servlet-generierte Antwort

## Java-fähige Web Server

- Servlets sind Bestandteil der Java *Enterprise* Edition (J2EE)
  - nicht mehr Standard Edition !
- Grundvoraussetzung:
  - Web-Server muss Java-Servlets einbinden können
  - Erkennen von Servlet-Requests
  - Verwaltung von Servlets
  - Ausführungsumgebung für Servlets *(servlet container)*
- Vor Experimenten mit Servlets:
  - Servlet Container installieren
  - z.B. Apache Tomcat




## Java Servlets

- Erste Version der Servlet API: 1996 (Java: 1995)
- Java Server Pages: 1997-1999

- Wichtige Referenz-Implementierung:
  - "Jakarta"-Projekt der "Apache"-Gruppe
    - » Apache: weitverbreiteter OpenSource-Web-Server
  - "Tomcat":
    - » Unterstützung für Servlet und JSP
    - » Separat oder als Modul für Apache-Server
    - » Entwicklungsumgebungen enthalten gelegentlich eigenen Tomcat-Server

- Grundprinzip der Ausführung:
  - Web-Server ruft Servlet bei Client-Requests auf (Muster Template Method)
  - Servlet bestimmt über Datenstrukturen die Antwort für den Client

## Servlet-API: Grundzüge

- **`abstract class javax.servlet.GenericServlet`**
  - Deklariert Methode **`service()`**
- **`abstract class javax.servlet.http.HttpServlet`**
  - Definiert Standardimplementierung für Methode **`service()`**
  - Gemäß Muster "Template Method" werden aufgerufen:
    - » **`doPost()`, `doGet()`, `doPut()`** etc. je nach Benutzer-Anfrage
  - **`protected void doGet(HttpServletRequest req, HttpServletResponse resp)`**
  - **`protected void doPost(HttpServletRequest req, HttpServletResponse resp)`**
- **`interface javax.servlet.http.HttpServletRequest`**
  - Deklariert Methoden wie **`getAttribute()`, `getParameter()`, `getReader()`**
- **`interface javax.servlet.http.HttpServletResponse`**
  - Deklariert Methoden wie **`setContentType()`, `getWriter()`**



## GET- und POST-Methode in HTTP

- Das Hypertext Transfer Protocol (HTTP) unterstützt zwei Methoden, Parameterwerte an aufgerufene Dokumente zu übergeben
- GET-Methode:
  - Variablenwerte werden als Bestandteil der URL codiert und übergeben:
    **`http://host.dom/pfad/fibonacci2.php?eingabe=12`**
  - Damit können Parameterangaben auch durch Eintippen der URL gemacht werden (ohne Formular)
  - Geeignet für einfache Abfragen
- POST-Methode:
  - Variablenwerte werden nicht in der URL codiert
  - Webserver wartet auf anschließende Übertragung der Variablenwerte (Einlesen vom Standard-Eingabekanal)
  - (Etwas) schwerer von außen zu "manipulieren“
- HTML: Attribut **`method`** beim Formular-Tag **`<form>`**
  - **`method="get"`** (default!) oder **`method="post"`**
- PUT-Methode: Einfacher Datei-Upload

## Beispiel: Hello-World Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
           HttpServletResponse response)
      throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```



## Beispiel: Einfaches dynamisches Servlet

- Aufgabe: HTML-Seite mit aktuellem Datum

```
public class DateServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
              throws ServletException, IOException {
    String title = "Date Servlet Page";
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML><HEAD><TITLE>");
    out.println(title);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + title + "</H1>");
    out.print("<P>Current time is: ");
    out.println(new java.util.GregorianCalendar().getTime());
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

Java HTML

## Kombination von Programmiersprache und HTML

- Servlets (und ebenso CGI, NSAPI, ISAPI):
  - Programme zum Erzeugen von HTML-Code
  - HTML-Elemente als Argumente von "print"-Anweisungen
  - Vorteil:
    - » Flexibilität
  - Nachteil:
    - » Unübersichtlich; HTML-Editoren ("Autorensysteme") nicht verwendbar
- Rollenkonflikt:
  - Softwareentwickler:
    - » bevorzugen Programmiersprache als "Basis" mit eingebettetem HTML
  - Web-Designer:
    - » bevorzugen HTML mit Erweiterungen für dynamische Anzeige
    - » wollen interaktive Werkzeuge mit "WYSIWYG"-Effekt
- Vgl.: Einbettung von SQL in Programmcode



## Beispiel: Java Server Page (JSP)

- Aufgabe: HTML-Seite mit aktuellem Datum

```
<HTML>
<%! String title = "Date JSP"; %>
<HEAD><TITLE> <%=title%> </TITLE></HEAD>
<BODY>
<H1> <%=title%> </H1>
<P>Current time is:
<% java.util.Date now = new GregorianCalendar().getTime(); %>
<%=now%>
</BODY></HTML>
```

- (Naheliegende) Grundidee für Java Server Pages:
  - Definition durch in HTML eingebettete Skripte ("*Scriptlets*")
  - Automatische Übersetzung in Java Servlet

**Java HTML**