COMPONENTS

Darstellung

Labels

- Dynamische Ausgabe von Zeichenketten
- Convenience-Konstruktor ohne Model

- HTML

```
<p>Anbieter des Trainings: <span wicket:id="trainer"></span></p>
```

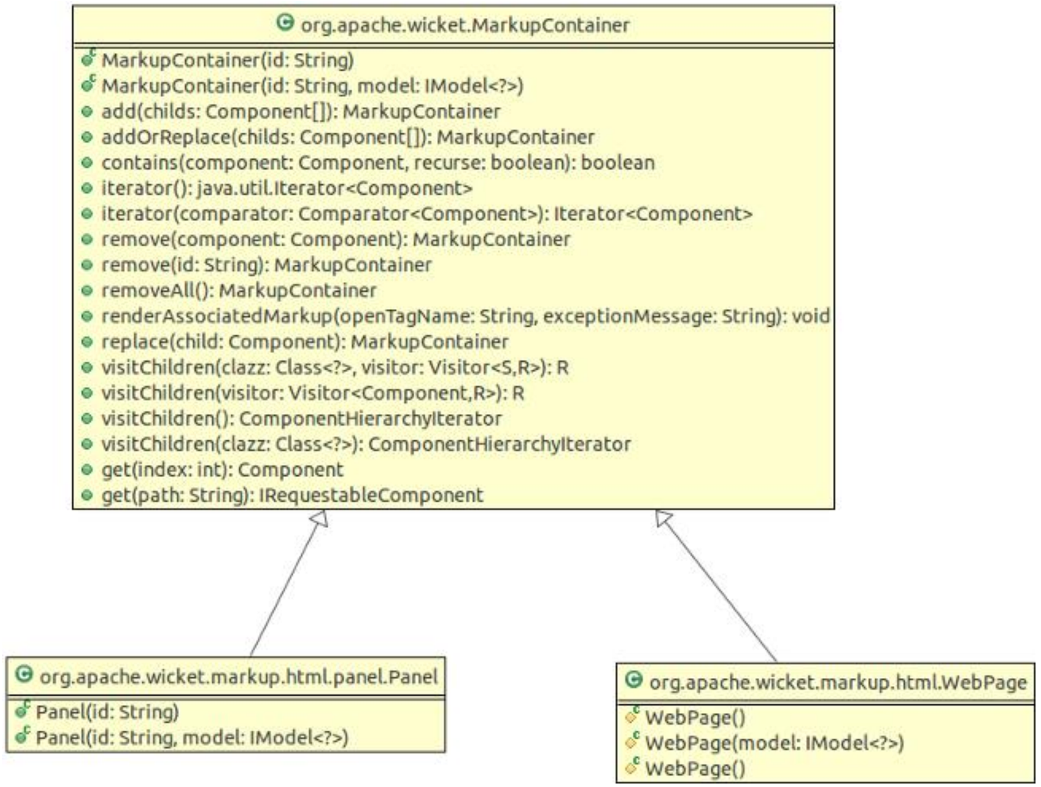
- Java

```
add(new Label("trainer", "GFU"));
```

Darstellung

Panels

- Wiederverwendbare Container für Komponenten
- HTML + Java


org.apache.wicket.MarkupContainer
MarkupContainer(id: String)
MarkupContainer(id: String, model: IModel<?>)
add(childs: Component[]): MarkupContainer
addOrReplace(childs: Component[]): MarkupContainer
contains(component: Component, recurse: boolean): boolean
iterator(): java.util.Iterator<Component>
iterator(comparator: Comparator<Component>): Iterator<Component>
remove(component: Component): MarkupContainer
remove(id: String): MarkupContainer
removeAll(): MarkupContainer
renderAssociatedMarkup(openTagName: String, exceptionMessage: String): void
replace(child: Component): MarkupContainer
visitChildren(clazz: Class<?>, visitor: Visitor<S,R>): R
visitChildren(visitor: Visitor<Component,R>): R
visitChildren(): ComponentHierarchyIterator
visitChildren(clazz: Class<?>): ComponentHierarchyIterator
get(index: int): Component
get(path: String): IRequestableComponent
org.apache.wicket.markup.html.panel.Panel
Panel(id: String)
Panel(id: String, model: IModel<?>)
org.apache.wicket.markup.html.WebPage
WebPage()
WebPage(model: IModel<?>)
WebPage()

Darstellung

Panels

- HTML: Panel

```
<html>

<head></head>

<body>

<wicket:panel>
   <!-- Diverse Labels. Äußeres Gerüst wird ignoriert! -->
</wicket:panel>

</body>

</html>
```

Darstellung

Panels

- Java: Panel

```
public class PersonPanel extends Panel {

    public PersonPanel(String id, IModel<Person> person) {
        super(id);
        setDefaultModel(new CompoundPropertyModel(person));

        add(new Label("name"));
        add(new Label("surname"));
        add(new Label("address"));
        add(new Label("email"));
        add(new Label("spouse.name"))
    }
}
```

Darstellung

Panels

- Java: Page

```java
public class PersonPage extends WebPage {

    public PersonPage() {
        Person john = new Person("John", "Doe");
        IModel<Person> person = new Model<>(john);

        add(new PersonPanel("person", person));
    }
}
```

Darstellung

Templating

- Vorgabe eines gemeinsamen Rahmens
  - Header
  - Navigation
  - Content
  - Footer
- Strategie
  - Panels für gemeinsame Bereiche konstruieren
  - Panels zu Template Page zusammenbauen
  - Subpages: Nutzung von Java und Markup Inheritance
  - Subpages: Spezifischen Content definieren
- Spezielle Tags
  - wicket:child
  - wicket:extend

Darstellung

Templating

- HTML: Template Page

```
<html>

<head></head>

<body>

<div id="header" wicket:id="headerPanel">header</div>
<div id="body">
    <div id="menu" wicket:id="menuPanel">menu</div>
    <wicket:child/>
</div>
<div id="footer" wicket:id="footerPanel">footer</div>

</body>

</html>
```

Darstellung

Templating

- Java: Template Page

```
public class GFUTemplate extends WebPage {

    private Component headerPanel;
    private Component menuPanel;
    private Component footerPanel;

    public GFUTemplate(){
        add(headerPanel = new HeaderPanel("headerPanel"));
        add(menuPanel = new MenuPanel("menuPanel"));
        add(footerPanel = new FooterPanel("footerPanel"));
    }
}
```

Darstellung

Templating

- HTML: Page

```
<html>

<head></head>

<body>

<wicket:extend>
    <!-- Tags für die gewünschten Komponenten einfügen -->
</wicket:extend>

</body>
</html>
```

Darstellung

Templating

- Java: Page

```
public class PersonPage extends GFUTemplate {

    public PersonPage(){
        super();

        // Gewünschte Komponenten einfügen
    }
}
```

Darstellung

Repeater

- Mehrfaches Anzeigen von Elementen
- Beispiel: Listen
- Alternativen in Wicket
  - RepeatingView
  - ListView
  - DataView

Darstellung

RepeatingView

- Wiederholung eines einfachen Markup-Fragments

- HTML

```
<ul>
    <li wicket:id="listItems"></li>
</ul>
```

- Java

```
RepeatingView listItems = new RepeatingView("listItems");

listItems.add(new Label(listItems.newChildId(), "green"));
listItems.add(new Label(listItems.newChildId(), "blue"));
listItems.add(new Label(listItems.newChildId(), "red"));
```

Darstellung

ListView

- Darstellung komplexeren Markups
- Jedes Element wird zu eigenem ListItem

- HTML

```
<div wicket:id="persons">
    <div><b>Full name: </b></div>
    <div wicket:id="fullName"></div>
</div>
```

Darstellung

ListView

- Java

```
public HomePage(final PageParameters parameters) {
    List<Person> persons = Arrays.asList(new Person("John", "Doe"));

    add(new ListView<Person>("persons", persons){

        @Override
        protected void populateItem(ListItem<Person> item) {
            item.add(new Label("fullName",
                         new PropertyModel(item.getModel(), "fullName")));
        }
    });
}
```

Darstellung

DataView

- Effiziente Darstellung großer Datenmengen
- DataProvider
    - Paging, Sorting
    - ListDataProvider
    - SortableDataProvider

- HTML

```
<table>
    <tr>
        <th>Name</th><th>Vorname</th>
    </tr>
    <tr wicket:id="rows">
        <td wicket:id="dataRow"></td>
    </tr>
</table>
```

Darstellung

DataView

- Java

```
//Methode außerhalb definiert
List<Person> persons = loadPersons();

ListDataProvider<Person> listDataProvider = new ListDataProvider<>(persons);

DataView<Person> dataView = new DataView<>("rows", listDataProvider) {

   @Override
   protected void populateItem(Item<Person> item) {
      Person person = item.getModelObject();
      RepeatingView repeatingView = new RepeatingView("dataRow");

      repeatingView.add(new Label(repeatingView.newChildId(), person.getName()));
      repeatingView.add(new Label(repeatingView.newChildId(), person.getSurname()));
      item.add(repeatingView);
   }
};

add(dataView);
```