Consistent page layout using borders

Markup inheritance is often more convenient to use than Borders; be sure to learn about markup inheritance as well.
This page needs an update

The provided solution doesn't work with either wicket 1.2.x nor 1.3.x, as MarkupContainer#add(Component) is final and can't be overriden.
See this thread on the mailing list.

Creating consistent page layouts using Borders

Borders are one of the most powerful components provided with the Wicket framework. However, this also makes them the most difficult to explain and understand. This short article covers how to use the Border component to create a standard site template that wraps around all of the pages in your site to ensure that they are consistent and to reduce the duplication of common HTML code (such as including menu areas and so on).

To build a site based on a standard border template we need to create three files. The first file is the Java file for the border:

```
    public class MySiteBorder extends Border
    {
        public MySiteBorder(final String id) {
            super(id);
            //TODO: create & add any components that always appear on every page
            //e.g. add(new MenuPanel());
        }
    }
```

We then need to create the HTML markup file:

```
    <?xml version="1.0"?>
    <html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.sourceforge.net/" xml:lang="en"
lang="en">
    <body>
        <wicket:border>
            <table>
                <tr>
                    <td><!--TODO: reference border level components. -->
                        <!-- E.g. <span wicket:id="mainmenu">MAIN MENU</span> --></td>
                    <td><wicket:body/></td>
                </tr>
            </table>
        </wicket:border>
    </body>
    </html>
```

One important thing to note about this is that I have written the HTML markup for the file so that it only represents the BODY part of the common page. It is possible to include the HEAD part of a common page in the markup, but I don't do this for a couple of reasons:

1. I may want a different page title, meta tags, stylesheet and javascript includes on each page. I could do this with some additional clever components and some attribute modifiers that I add to the border, but I have personally found that it is just as easy to do then in a HEAD block on each page
2. By including the HEAD block within each individual page, it keeps the separation of concerns cleaner. That way my HTML team can build each page in their WYSIWYG editor, include whatever Javascript they wish. When I integrate this into the site, all I am doing is wrapping common page components around the body elements they have created. If the HEAD block was inside the border template then I would have to merge any additional Javascript and so on into the common border. (I agree this is not a problem if your site is small, very consistent and you have a team doing both HTML and Java elements).

The next stage is to create a Page base class that all pages must extend. This is worth doing as it then forces use of the border and minimises the amount of code required:

```
    public abstract class MySitePage extends WebPage {
        private Border border;
        public MarkupContainer add(final Component child) {
            // Add children of the page to the page's border component
            if (border == null) {
                // Create border and add it to the page
                border = new MySiteBorder("border");
                super.add(border);
            }
            border.add(child);
            return this;
        }
        public void removeAll() {
            border.removeAll();
        }
        public MarkupContainer replace(final Component child) {
            return border.replace(child);
        }
        public boolean autoAdd(Component component) {
            return border.autoAdd(component);
        }
    }
```

Finally, everything is set up and I am now ready to create real pages for my site. First I need a the Java code for a page:

```
    public class Home extends MySitePage {
        public Home() {
            add(new Label("test", "This is a Test"));
        }
    }
```

Then I just need the HTML to go with the page:

```
    <?xml version="1.0"?>
    <html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.sourceforge.net/" xml:lang="en"
lang="en">
    <head>
        <title>MySite</title>
        <meta name="description" content="MySite"/>
        <meta name="keywords"
              content="wicket,cool,hot,neat"/>
        <link rel="stylesheet" href="styles/mysite.css"/>
    </head>
    <body>
        <span wicket:id="border">
             <span wicket:id="test">TEST LABEL</span>
        </span>
    </body>
    </html>
```

To merge the HEAD part of both HTM files, e.g. to specify a common css for all files but give each page a unique title, modify the above example like this:

The border's markup:

```
    <?xml version="1.0"?>
    <html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.sourceforge.net/" xml:lang="en"
lang="en">
    <head>
        <wicket:head>
            <link rel="stylesheet" href="styles/mysite.css"/>
        </wicket:head>
    </head>
    <body>
        <wicket:border>
            <table>
                <tr>
                    <td><!--TODO: reference border level components. -->
                        <!-- E.g. <span wicket:id="mainmenu">MAIN MENU</span> --></td>
                    <td><wicket:body/></td>
                </tr>
            </table>
        </wicket:border>
    </body>
    </html>
```

and the page markup:

```
    <?xml version="1.0"?>
    <html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.sourceforge.net/" xml:lang="en"
lang="en">
    <head>
        <wicket:head>
            <title>MySite</title>
            <meta name="description" content="MySite"/>
            <meta name="keywords"
                  content="wicket,cool,hot,neat"/>
        </wicket:head>
    </head>
    <body>
        <span wicket:id="border">
             <span wicket:id="test">TEST LABEL</span>
        </span>
    </body>
    </html>
```

Note: If the body element is not an immediate child of border (example see below), than you must use *someContainer.add(getBodyContainer())* to add the body component to the correct container.

```
   <html>
   <body>
     <wicket:border>
       <span wicket:id="someContainer">
         <wicket:body/>
       </span>
     </wicket:border>
   </body>
   </html>
```

Note: contributions to <wicket:head>

Note which tags are contributed to the final page output.

border:

```
<html>
  <head>
    <tag>Will NOT be included</tag>
    <wicket:head>
      <tag>WILL be contributed</tag>
    </wicket:head>
  </head>
</html>
```

page:

```
<html>
  <head>
    <tag>WILL be contributed</tag>
    <wicket:head>
      <tag>WILL be contributed</tag>
    </wicket:head>
  </head>
</html>
```

In a border or in a panel, anything outside <wicket:border>..</wicket:border> and <wicket:panel>...</wicket:panel> tags will be ignored, the exception being the body of the <wicket:head> tag.

At the page level, as pages are "top level" components, they don't need special <wicket:page> tags and the like. Therefore in pages, nothing is ignored unless you explicitly tell Wicket to do so by using <wicket:remove> tags.

Note: the use of <wicket:remove> should be ok within other tags such as <wicket:panel> and <wicket:page>. It should not even be necessary with <wicket:head> (you can simply put the contents outside of the <wicket:head> tag).

The other way around is not true, however. For instance, <wicket:remove><wicket:head></wicket:head></wicket:remove> will throw an exception.