How to load an external image

Sometimes it is desirable to serve images from a separate server rather than the one that the Wicket application is running on for better performance. This can be achieved by a few different techniques...

1. Add an AttributeModifier for the "src" attribute. This technique will allow for dynamic updates to the image source, including Ajax target. addComponent(dynamicImage).

```
// NOTE: overriding the "onComponentTag" method will not work when adding the image to an Ajax target
(target.addComponent(dynamicImage);). This is why an AttributeModifier is used instead
dynamicImage = new Image("img-status");
dynamicImage.add(new AttributeModifier("src", true, new AbstractReadOnlyModel() {
        private static final long serialVersionUID = 1L;

        /**
         * {@inheritDoc}
         */
        @Override
        public final Object getObject() {
                // based on some condition return the image source
                if (myCondition) {
                        return "http://wicket.apache.org/style/wicket.png";
                } else {
                        return "http://wicket.apache.org/style/apache.png";
                }
        }
}));
dynamicImage.setOutputMarkupId(true);
```

2. Create your own image class:

```
    public class StaticImage extends WebComponent {

        public StaticImage(String id, IModel model) {
            super(id, model);
        }

        protected void onComponentTag(ComponentTag tag) {
            super.onComponentTag(tag);
            checkComponentTag(tag, "img");
            tag.put("src", getModelObjectAsString());
            // since Wicket 1.4 you need to use getDefaultModelObjectAsString() instead of
getModelObjectAsString()
        }

    }

    add(new StaticImage("img", new Model("http://foo.com/bar.gif"));
```

The class can be updated to allow a URL to be passed and have it display a correctly written img tag

```
public class ExternalImageUrl extends WebComponent {

        public ExternalImageUrl(String id, String imageUrl) {
            super(id);
            add(new AttributeModifier("src", true, new Model(imageUrl)));
            setVisible(!(imageUrl==null || imageUrl.equals("")));
        }

        protected void onComponentTag(ComponentTag tag) {
            super.onComponentTag(tag);
            checkComponentTag(tag, "img");
        }

}
```

Even with the code above it wasn't obvious for us how to create clickable icons (serving as external links), which we needed to have as part of the Panel, used to populate a grid with. Thanks to Igor Vaynberg's hint, we were able to connect those pieces together. Here is a sample code and related HTML markup for our Panel component, in case someone needs to do something similar.

```
public class GridThumbnailPanel extends Panel {
    public GridThumbnailPanel( String id, String label, String iconURL, String linkURL) {
        super( id);
        Label label = new Label( "label", label);
        add( label);
        StaticImage img=new StaticImage( "icon", new Model(iconURL));
        ExternalLink link=new ExternalLink( "link", linkURL);
        link.add(img);
        add( link);
    }
}
```

```
<wicket:panel>
    <div wicket:id="label">[test]</div>
    <a wicket:id="link" target="_blank"> <img wicket:id="icon"/> </a>
</wicket:panel>
```

Please, notice that our panel consists of a label and an icon. To construct the panel, one needs to supply its wicket id, text for the label and 2 URL-s (one for the icon itself, another - for the external link). The external link URL is used to populate the new browser page opened when the icon is clicked upon, and _blank target is what causes the new browser page to open after the click.