Built-in Directives

Angular has a couple of useful built-in directives.

TODO(Note on directive names, docs and template usage)

`NgClass`

- `import {NgClass} from 'angular2/common';` , `directives: [NgClass]`
- Template Usage:

`<div class="button" [ngClass]="{active: isActive, disabled: !isActive}"`

Note that we are using `ngClass` in the template, but not `NgClass`

`NgIf`

Usage

```
<div *ngIf="isDone">{{ list }}</div>
```

or in long-hand form:

```
<template>
  <div [ngIf]="isDone">{{ list }}</div>
</template>
```

Details

From the docs: "The ngIf directive does not hide the element. Using browser developer tools we can see that, when the condition is true, the top paragraph is in the DOM and the bottom disused paragraph is completely absent from the DOM! In its place are empty `<script>` tags. We could hide the unwanted paragraph by setting its css display style to none. The element would remain in the DOM while invisible. Instead we removed it with ngIf.

The difference matters. When we hide an element, the component's behavior continues. It remains attached to its DOM element. It continues to listen to events. Angular keeps checking for changes that could affect data bindings. Whatever the component was doing it keeps doing.

Although invisible, the component — and all of its descendant components — tie up resources that might be more useful elsewhere. The performance and memory burden can be substantial and the user may not benefit at all.

On the positive side, showing the element again is very quick. The component's previous state is preserved and ready to display. The component doesn't re-initialize — an operation that could be expensive.

ngIf is different. Setting ngIf to false does affect the component's resource consumption. Angular removes the element from DOM, stops change detection for the associated component, detaches it from DOM events (the attachments that it made) and destroys the component. The component can be garbage-collected (we hope) and free up memory.

Components often have child components which themselves have children. All of them are destroyed when ngIf destroys the common ancestor. This cleanup effort is usually a good thing.

Of course it isn't always a good thing. It might be a bad thing if we need that particular component again soon.

The component's state might be expensive to re-construct. When ngIf becomes true again, Angular recreates the component and its subtree. Angular runs every component's initialization logic again. That could be expensive ... as when a component re-fetches data that had been in memory just moments ago."

`NgSwitch`

Usage

```
<div [ngSwitch]="status">
  <template [ngSwitchWhen]="'inProgress'">In Progress</template>
  <template [ngSwitchWhen]="'isDone'">Finished</template>
  <template ngSwitchDefault>Unknown</template>
```

```
</div>
```

TODO

`NgFor`

Usage

```
<ul>
    <li *ngFor="#item of items">{{ item }}</li>
</ul>
```

or in long-hand form:

```
<ul>
  <template ngFor #item [ngForOf]="items">
    <li>{{ item }}</li>
  </template>
</ul>
```

TODO(Details)