CAPÍTULO 9

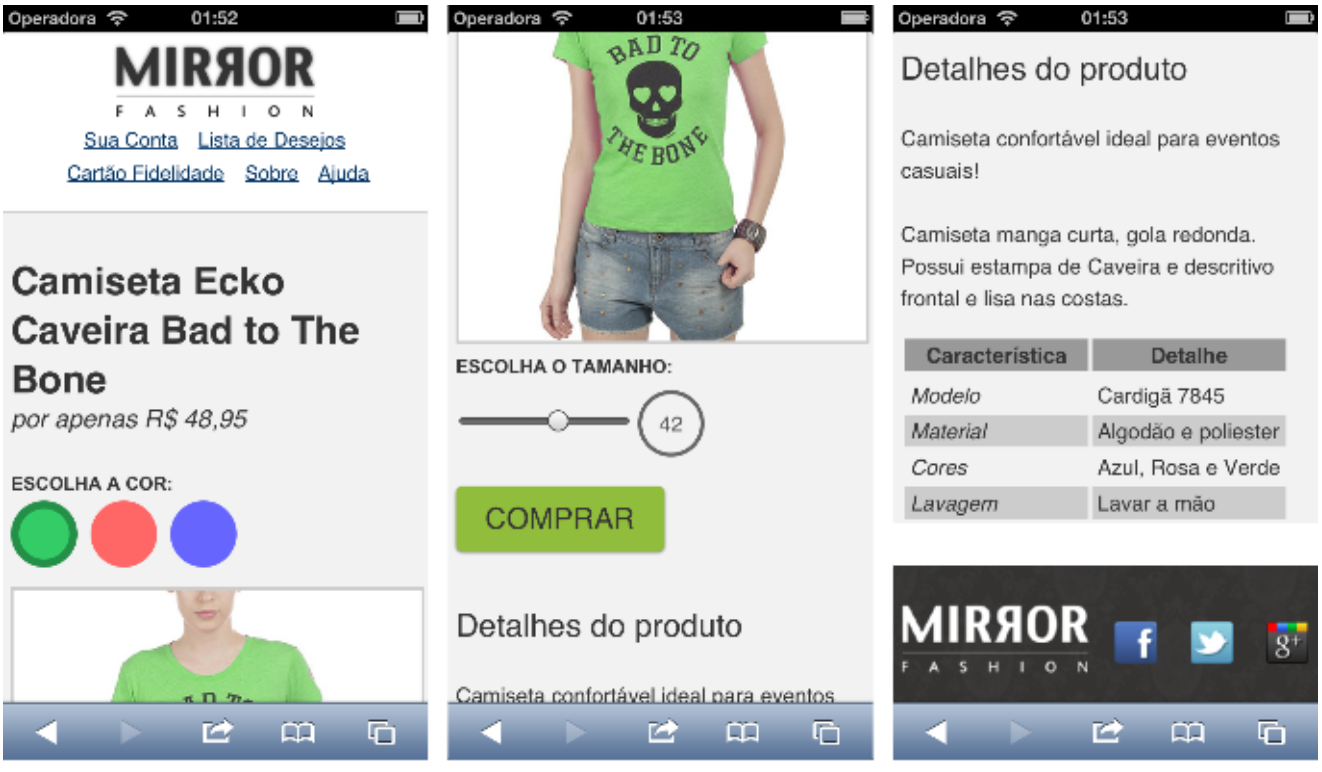
# Progressive enhancement e mobile-first

*"Qualquer tolo consegue escrever código que um computador consiga entender. Bons programadores escrevem código que humanos conseguem entender."*
*— Martin Fowler*

Vamos criar a próxima página da nossa loja, que será uma página para mostrar os detalhes de um produto após o usuário clicar em um produto na lista da home. Essa página vai mostrar uma foto grande, mostrar opções de cores e preço, exibir a descrição do produto e permitir que o usuário faça a compra.

Nosso designer criou um design para essa página, com estilo mais minimalista e focado no conteúdo a ser exibido.

Eis nosso design aplicado na tela do iPhone para visualizarmos:



## 9.1 – Formulário de compra

Um dos aspectos fundamentais dessa página é permitir que o usuário escolha a variação correta do produto para ele. Repare que ele pode escolher a cor e o tamanho e depois comprar o produto específico que escolheu.

Quando clicar nesse botão de comprar, as escolhas do usuário precisam ser enviadas para o servidor processar a compra em si. São, claro, parâmetros que o usuário pode escolher.

Esqueça por um instante o design que vimos antes e pense na **funcionalidade** que estamos tentando implementar. Queremos uma maneira do usuário **escolher entre diversas opções** e **enviar sua escolha** para o servidor. Falando assim, é quase óbvio que estamos descrevendo um `<form>`.

Mais: se queremos escolher, por exemplo, a cor da roupa dentre 3 opções possíveis, temos componentes específicos de formulário para isso. Podemos fazer um combobox com `<select>` ou implementar com **3 radio buttons**. Vamos fazer esse último.

```
<form>
  <input type="radio" name="cor"> Verde
  <input type="radio" name="cor"> Rosa
  <input type="radio" name="cor"> Azul
</form>
```

Muito simples e funciona. Mas tem várias falhas de acessibilidade e HTML semântico. O texto que descreve cada opção, por exemplo, não deve ficar solto na página. Devemos usar o elemento `<label>` para representar isso. E associar com o respectivo input.

```
<form>
  <input type="radio" name="cor" id="verde">
  <label for="verde">Verde</label>

  <input type="radio" name="cor" id="rosa">
  <label for="rosa">Rosa</label>

  <input type="radio" name="cor" id="azul">
  <label for="azul">Azul</label>
</form>
```

Mais ainda, repare que temos que explicar para o usuário o que ele está escolhendo com esses radio buttons. É a frase *"Escolha a cor"* que vemos no design. Como escrevê-la no HTML? Um simples `<p>`? Não.

Semanticamente, esses 3 inputs representam a mesma coisa e devem ser agrupados em um `<fieldset>` que, por sua vez, recebe um `<legend>` com a legenda em texto apropriada.

```
<form>
  <fieldset>
    <legend>Escolha a cor</legend>

    <input type="radio" name="cor" id="verde">
    <label for="verde">Verde</label>

    <input type="radio" name="cor" id="rosa">
    <label for="rosa">Rosa</label>

    <input type="radio" name="cor" id="azul">
    <label for="azul">Azul</label>
  </fieldset>

  <input type="submit" class="comprar" value="Comprar">
</form>
```

Aproveitamos e colocamos o botão de submit para enviar a escolha da compra.

Podemos ainda melhorar mais. Em vez de mostrar só o nome da cor ("Verde") no label, podemos colocar a foto de verdade da roupa naquela cor. Uma imagem vale mais que mil palavras. Mas, claro, isso para quem enxerga. Para leitores de tela e outros browsers não visuais, vamos usar o atributo `alt=` na imagem para manter sua acessibilidade.

```
<form>
  <fieldset>
    <legend>Escolha a cor</legend>

    <input type="radio" name="cor" id="verde">
    <label for="verde">
      <img src="produto-verde.png" alt="Produto na cor verde">
    </label>

    <input type="radio" name="cor" id="rosa">
    <label for="rosa">
      <img src="produto-rosa.png" alt="Produto na cor rosa">
    </label>

    <input type="radio" name="cor" id="azul">
    <label for="azul">
      <img src="produto-azul.png" alt="Produto na cor azul">
    </label>
  </fieldset>

  <input type="submit" class="comprar" value="Comprar">
</form>
```

Se implementarmos esse código, sem visual nenhum, e testarmos no browser, teremos uma funcionalidade completa, funcional e acessível. Isso é fantástico. Resolveremos o visual depois.

## 9.2 – Exercício: formulário da página de produto

Vamos implementar nossa página de produto. O primeiro passo é a construção de um HTML semântico.

1. Edite a página **produto.php** e, entre os `includes` do cabeçalho e do rodapé, adicione um formulário com radios e labels para a escolha da cor. Também usaremos o atributo `alt` para boa acessibilidade:

```
<div class="produto">
  <h1>Fuzzy Cardigan</h1>
  <p>por apenas R$ 129,00</p>

  <form>
      <fieldset class="cores">
          <legend>Escolha a cor:</legend>

          <input type="radio" name="cor" value="verde" id="verde" checked>
          <label for="verde">
            <img src="img/produtos/foto2-verde.png"
              alt="verde">
          </label>

          <input type="radio" name="cor" value="rosa" id="rosa">
          <label for="rosa">
            <img src="img/produtos/foto2-rosa.png"
              alt="rosa">
          </label>

          <input type="radio" name="cor" value="azul" id="azul">
          <label for="azul">
            <img src="img/produtos/foto2-azul.png"
            alt="azul">
          </label>

      </fieldset>

      <input type="submit" class="comprar" value="Comprar">
  </form>
</div>
```

2. Teste o HTML do exercício anterior. Veja seu funcionamento sem interferência do CSS e do JS que faremos depois.

## 9.3 - Design mobile-first

Quando criamos a home page do projeto não sabíamos ainda otimizar nosso site para dispositivos móveis. Vimos o design e codificamos originalmente pensando nos browsers do desktop. Mais tarde, aplicamos os conceitos de media queries e viewport para ajustar o projeto para telas menores.

Esse tipo de fluxo de desenvolvimento é muito comum. Desenvolver para desktop primeiro, que são a maioria dos usuários, e depois ajustar o design para mobile. Mas isso **não é o melhor**, nem o mais fácil.

Muita gente argumenta a favor de uma técnica inversa, **mobile-first**. Isso significa fazer o design mobile primeiro, implementar o código para mobile primeiro e, depois, ajustar para o desktop. O resultado final deve ser um site que funciona tanto no desktop quanto no mobile, como antes, só mudamos a ordem do *fluxo de desenvolvimento*.

Na prática, o que muita gente descobriu é que criar pensando no ambiente mais restritivo (o mobile) e depois adaptar para o ambiente mais poderoso (desktop) é mais fácil que fazer o contrário. desktop-first acaba sendo difícil por colocar o mobile, que é complicado, só no final do projeto.

Um exemplo prático que passamos na nossa home page. Fizemos antes um menu com CSS usando hover para abrir subcategorias. Isso é algo super comum e funciona muito bem no desktop. Mas é um desastre no mobile, onde não existe hover. Podemos agora repensar nossa home para ser compatível com mobile. Mas se tivéssemos, desde o início, pensando em mobile, talvez nem criássemos o menu hover.

Outro exemplo: os links do menu são bastante inacessíveis em mobile. As opções estão muito próximas uma das outras e tocar na opção certa com o dedo

(gordo!) é muito difícil. No desktop não há esse problema pois usamos mouse, por isso não pensamos nisso antes.

Se tivéssemos começado pelo mobile, já teríamos feito os links maiores e mais espaçados pensando nos dedos gordos (costuma-se recomendar um valor médio de 50px para cada item clicável).

E, fazendo tudo maior e mais espaçado, assim como evitando o hover, o site funciona bem no mobile mas, não só isso, funciona muito bem no desktop. Um site bem feito para mobile funciona perfeito no desktop mas o contrário nem sempre é verdade. Por isso o **mobile-first**.

Repare que o designer já mandou a página de produtos para nós pensando em mobile-first: pouca informação, só o essencial, prioritário. Botões grandes e espaçados. Nenhum efeito de hover. Tudo numa coluna só de informações para dar scroll, já que a tela é pequena.

Nem sabemos ainda como será a versão desktop, e não interessa por enquanto.

> **Mais sobre mobile-first**
>
> Não vamos nos estender no assunto aqui no curso mas, se interessar, existe um livro só sobre o tema, chamado *Mobile-first* de *Luke Wroblewski*. Em português, você pode ler mais no livro *A Web Mobile* do instrutor da Caelum *Sérgio Lopes*.

## 9.4 - Progressive enhancement

No exercício vamos ver como usar CSS para estilizar aquele formulário anterior em algo parecido com o design desejado. Mas o importante é perceber como temos uma página funcional e acessível antes de pensarmos em visual.

O papel do HTML é esse. Estruturar o conteúdo da página de maneira semântica e acessível, provendo uma base de funcionalidades para a página sem relação imediata com visual.

O CSS e o visual dele são uma segunda camada, que vem em cima do HTML semântico e bem construído. Depois, vamos ver até que um pouco de JavaScript é necessário para implementar outro recurso da página. Mas ele é opcional. Só o

HTML, sozinho, seria suficiente para uma experiência completa e funcional da página.

Esse tipo de pensamento é o **progressive enhancement** novamente. Construir uma base sólida, simples, portável e acessível e, depois, progressivamente, incrementar a página com recursos mais avançados tanto de estilo com CSS quanto de comportamento com JavaScript.

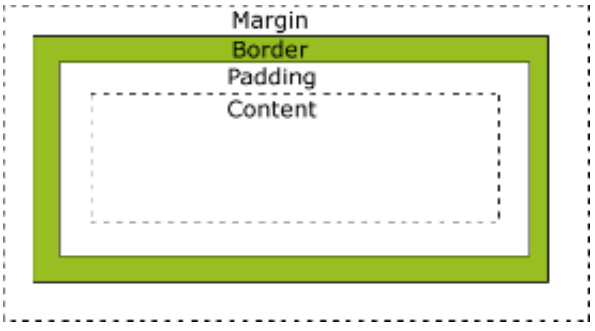
Hoje no mercado há muita falta de profissionais com experiência e completo entendimento das implicações de progressive enhancement. Amadores focam em olhar o design do Photoshop e copiar na página. Profissionais focam em experiências Web acessíveis, semânticas e portáveis, onde o design tem um papel decorativo.

## 9.5 - Box model e box-sizing

### O Box Model padrão do W3C

Quando alteramos as propriedades de elementos dentro de uma página, precisamos estar cientes de como essas alterações se comportarão na presença de outros elementos. Uma forma de entender o impacto causado pela mudança é pensar no **box model** ou **modelo em caixa** em português.

O **box model** é constituído de quatro áreas retangulares: conteúdo (content), espaçamento (padding), bordas (borders) e margens (margin) conforme a figura abaixo:




Essas áreas se desenvolvem de dentro para fora, na ordem listada abaixo:

- **conteúdo (content)**: aquilo que será exibido;
- **espaçamento (padding)**: distância entre a borda e o conteúdo;
- **borda (borders)**: quatro linhas que envolvem a caixa (box);

- **margem (margin)**: distância que separa um box de outro.

Tendo em mente o `box model`, precisamos ter atenção na alteração de propriedades de um elemento visualizando o impacto em sua apresentação ao lidar com as propriedades listadas acima.

### Box-sizing

Os primeiros a perceberem que o box model do CSS era esquisito foi a Microsoft. Já no IE6 em *quirks mode* eles trocaram o box model para que o width significasse o tamanho total até a borda. A ideia era boa mas o problema foi eles atropelarem a especificação da época, o que acabou criando boa parte dos problemas de incompatibilidade entre navegadores. O IE em *standards mode* usa o box model oficial e esse é o padrão a partir do IE8.

Mas como a ideia, no fim, era boa, isso acabou se transformando no `box-sizing` do CSS3, que nos permite trocar o box model que queremos usar.

Por padrão, todos os elementos têm o valor `box-sizing: content-box` o que indica que o tamanho dele é definido pelo seu conteúdo apenas -- em outras palavras, é o tal box model padrão que vimos antes. Mas podemos trocar por `box-sizing: border-box` que indica que o tamanho agora levará em conta até a borda – ou seja, o width será a soma do conteúdo com a borda e o padding.



## 9.6 - Exercícios: página de produto

Vamos estruturar nosso CSS para implementar a funcionalidade de troca de

cores. A cada passo, teste no browser para ir acompanhando o resultado.

1. **Crie** um novo arquivo **produto.css** na pasta **css/**.

2. Em **produto.php**, importe o **produto.css** após todos os outros css's:

```
<link rel="stylesheet" href="css/produto.css">
```

3. Primeiro, vamos desenhar as bolinhas coloridas com pseudo-elementos do CSS3 usando um truque com bordas redondas grandes:

```
.cores label:after {
  content: '';
  display: block;

  border-radius: 50%;
  width: 50px;
  height: 50px;
}

label[for=verde]:after {
  background-color: #33CC66;
}

label[for=rosa]:after {
  background-color: #FF6666;
}

label[for=azul]:after {
  background-color: #6666FF;
}
```

Próximo passo é estilizar a bolinha atualmente selecionada usando pseudo-classe `:checked`:

```
.cores input:checked + label:after {
  border: 6px solid rgba(0,0,0,0.3);
}
```

Repare como a borda da bolinha selecionada aumenta o tamanho total da bolinha por causa do *box model padrão*. Uma solução é trocar o box model com a propriedade `box-sizing`:

```
.cores label:after {
  -moz-box-sizing: border-box;
       box-sizing: border-box;
}
```

Agora que temos as bolinhas coloridas visuais configuradas, a bolinha do input radio é desnecessária. Esconda-a:

```
.cores input[type=radio] {
```

```
  display: none;
}
```

Para fechar a funcionalidade de escolha das cores, falta apenas exibir apenas a imagem atualmente selecionada. Outra forma de falar isso é que *devemos esconder as imagens dos radios não selecionados*. Podemos usar um seletor avançado para isso:

```
.cores input:not(:checked) + label img {
  display: none;
}
```

Reflita sobre esse último seletor. O que ele faz exatamente? Esteja certo de ter entendido cada parte dele antes de prosseguir.

4. Teste a página. A troca de imagens deve estar funcionando, apesar das coisas não estarem ainda posicionadas corretamente.

5. Com a troca de imagens funcionando, vamos implementar o *posicionamento* correto das bolinhas lado a lado. Para isso, use `position:absolute` já que seus tamanhos são conhecidos:

```
.cores label:after {
  position: absolute;
}
```

As bolinhas vão ser posicionadas com relação ao fieldset cores, então ele precisa estar posicionado. O padding superior é para abrir espaço para as bolinhas:

```
.cores {
  position: relative;
  padding-top: 80px;
}
```

Cada bolinha é um `label:after`, basta posicioná-las absolutamente:

```
.cores label:after {
  position: absolute;
  top: 30px;
}
```

As bolinhas ficaram sobrepostas na esquerda. Para corrigir, basta colocar uma coordenada `left` diferente para cada uma:

```
label[for=verde]:after {
  left: 0;
}

label[for=rosa]:after {
  left: 60px;
```

```
}

label[for=azul]:after {
  left: 120px;
}
```

Teste o resultado.

6. Um ponto importante de uma solução responsiva é que as imagens se adaptem ao tamanho da tela. Às vezes, usamos imagens maiores e, quando a tela é pequena, a imagem fica "vazando" para fora do elemento pai.

Uma forma de corrigir esse problema é garantir que ela nunca passe o tamanho do pai com `max-width`:

```
.cores label img {
  display: block;
  max-width: 100%;
}
```

> **Bug no Firefox**
>
> Nosso max-width: 100% não vai funcionar se você usar um Firefox antigo. Neste navegador, o elemento `fieldset` não respeitava o comportamento correto de largura de um elemento e, assim, a largura da imagem também fica incorreta (o bug já está registrado em https://bugzilla.mozilla.org/show_bug.cgi?id=261037).
>
> O problema foi corrigido no Firefox 27.

7. Com toda a parte funcional e de posicionamento pronta, vamos estilizar alguns detalhes visuais da página.

Primeiro, detalhes de tipografia e espaçamento para toda página de produtos:

```
.produto {
  color: #333;
  line-height: 1.3;
  margin-top: 2em;
}
```

O nome do produto e seu preço também ganham estilo:

```
.produto h1 {
  font-size: 1.8em;
  font-weight: bold;
```

```
}

.produto p {
  font-size: 1.2em;
  font-style: italic;
  margin-bottom: 1em;
}
```

O `<legend>` ganha um destaque:

```
.produto legend {
  display: block;
  font: bold 0.9em/2.5 Arial;
  text-transform: uppercase;
}
```

E por fim, o botão de comprar deve ficar em evidência:

```
.comprar {
  background: #91bd3c;
  border: none;
  color: #333;

  font-size: 1.4em;
  text-transform: uppercase;

  box-shadow: 0 1px 3px #777;

  display: block;
  padding: 0.5em 1em;
  margin: 1em 0;
}
```

Teste e observe o estilo simples da página.

8. (opcional) Quando selecionamos a bolinha, uma borda escura aparece. Use `transition` para fazer a borda aparecer suavemente, como um fadein.

Adicione o seletor no início de **produto.css**:

```
.cores label:after {
  border: 6px solid rgba(0,0,0,0);
  transition: 1s;
}
```

E podemos adicionar também um estilo para quando passar o mouse em cima da bolinha, como mostrar uma borda mais leve, também no início de **produto.css**:

```
.cores label:hover:after {
  border: 6px solid rgba(0,0,0,0.1);
}
```

## 9.7 - Evoluindo o design para desktop

Feito o design mobile-first, é hora de expandir o site para as versões maiores. Do ponto de vista de design, significa ajustar os elementos para melhor aproveitar o espaço maior das telas de tablets e desktops. Do ponto de vista de código, é usar media queries para implementar essas mudanças.

Um exemplo: imagine que, em telas maiores que 600px, queremos flutuar uma imagem a esquerda:

```
@media (min-width: 600px) {
  img {
    float: left;
  }
}
```

Ao desenvolver mobile-first, usamos muitas media queries do tipo **min-width** para implementar as mudanças para o tablet/desktop.

## 9.8 - Media queries de conteúdo

Ao escrever medias queries, você precisa escolher algum valor para colocar lá. É o que chamamos de ***breakpoints***, os pontos onde seu layout vai ser ajustado por causa de uma resolução diferente. E escrever bons breakpoints é essencial para um design responsivo de qualidade.

E o que mais aparece de pergunta de quem está começando com design responsivo é: quais os valores padrões de se colocar nas media queries? E logo surge uma lista parecida de tamanhos comuns: 320px, 480px, 600px, 768px, 992px, 1200px. O pessoal chama essa prática de device-driven breakpoints, pois são valores gerados a partir de tamanhos de dispositivos.

Mas evite esse tipo de breakpoint. Essa lista pensa em meia dúzia de tipos de dispositivos, mas obviamente não atende todos (e os 360px de um Galaxy S4?). Usar esses valores de media queries não garante que seu design funcionará em todos os dispositivos, mas apenas nos dispositivos "padrões" -- seja lá o que for isso.

Prefira breakpoints com valores baseados no seu conteúdo. Ou seja, achar suas media queries a partir do seu conteúdo e do seu design. Fica bem mais fácil garantir que sua página funciona em todos os dispositivos.

Na prática, faça seu design mobile-first, abra no navegador pequeno, vá redimensionando a janela até achar um ponto que o design quebra ou fica feio; anote o tamanho da janela e crie um breakpoint lá. Não precisa ser um valor bonitinho como 600px. Às vezes sua página vai quebrar justo em 772px. Não tem problema.



## 9.9 - Exercícios: responsive design

Nosso layout anterior foi feito com mobile em mente, **mobile-first**. A parte boa é que, quando abrimos no desktop, tudo funciona muito bem. Mas o espaço maior não é bem aproveitado.

Se você redimensionar a janela para um tamanho grande, notará que nosso conteúdo não está centralizado na página como o restante. Lembre que criamos uma classe `container` para isso. Podemos usá-la novamente.

1. Na página **produto.php**, **crie** uma nova `<div class="container">` ao redor do conteúdo da página. Isto é, será uma div pai da div com class `produto`:

```
<!-- envolendo a div produto pela nova div -->
<div class="container">
  <div class="produto">
  .........
  </div>
</div>
```

Vamos usar media queries para ajustar o design para um estilo duas colunas.

Todo o CSS dos exercícios seguintes estará dentro de uma media query que só vai disparar em telas maiores.

2. Edite **produto.css** e adicione a media query em seu final:

```
@media (min-width: 630px) {

}
```

Na versão desktop, queremos reposicionar as coisas em duas colunas. Vamos colocar a foto do produtos à esquerda posicionada em relação ao `.produto`. Isso vai afetar o posicionamento das bolinhas então vamos trocar para posicioná-las com `float` simples.

O código é curto mas cheio de detalhes. Acompanhe os comentários para entender o papel de cada item.

3. **Dentro** da media query anterior, acrescente: (não precisa copiar os comentários):

```
.produto {
  /* a foto vai se posicionar absolutamente com relação
     a esse elemento, por isso preciso estar posicionado
   */
  position: relative;

  /* deixo 40% de espaço em branco na esquerda para foto ocupar */
  padding-left: 40%;
}

.cores {
  /* eu era relative antes; reinicio para static para evitar
     que a foto se posicione com relação a mim
  */
  position: static;

  /* reinicio meu padding-top que tinha antes e não preciso mais */
  padding: 0;
}

.cores label img {
  /* imagem se posiciona absolutamente à esquerda com relação ao .produto */
  position: absolute;
  top: 0;
  left: 0;
}

.cores label:after {
  /* as bolinhas coloridas tinham posição absoluta.
     não precisamos mais, basta flutuar uma do lado da outra */
  position: static;
  float: left;
}
```

Teste a página e veja que a imagem deixa a desejar, pois ainda não está posicionada corretamente. Apesar disso, o restante já começa a ficar no lugar.

4. As próximas regras devem ser adicionadas **dentro** da media query anterior. Você pode até escrever apenas as propriedades **dentro dos seletores** existentes na media query.

Primeiro, para evitar que a imagem vaze para fora do espaço que lhe foi determinado, vamos usar as propriedades `max-width` e `max-height`:

```
.cores label img {
  max-width: 35%;
  max-height: 100%;
}
```

Podemos aumentar um pouco o tamanho das fontes usadas no produto. Como usamos `em` antes, basta aumentar a fonte do elemento pai, o `produto` e tudo escala proporcionalmente.

```
.produto {
  font-size: 1.2em;
}
```

Último ajuste é uma pequena margem entre as bolinhas coloridas:

```
.cores label:after {
  margin-right: 10px;
}
```

## 9.10 - HTML5 Input range

No design original, havia a previsão de se criar uma maneira de selecionar também o tamanho da roupa além de sua cor. O tamanho é algo simples em nossa loja. Temos valores possíveis 36, 38, 40, 42, 44 e 46.

E há muitas formas corretas e semânticas de implementar isso no formulário. Pode ser um `<select>` com esses valores, radio buttons ou, como vamos ver, o **novo input range do HTML5**.

O `<input type="range">` é um componente novo do HTML5 com bom suporte já nos navegadores que representa um slider numérico. Ele recebe atributos **min** e **max** com o intervalo numérico possível. Opcionalmente, há o atributo **step** que indica de quanto em quanto o número deve pular (algo bem útil para tamanho de roupa, que só tem números pares).

```
<input type="range" min="36" max="46" value="42" step="2" name="tamanho">
```

O legal de componentes HTML5 é que eles são nativos dos browsers. Isso

significa que não precisamos de trabalho para usá-los ou estilizá-los. Eles já vêm com estilo padrão do navegador em questão o que é bem interessante. A interface padrão é familiar para o usuário.

Veja o range no Safari do iPhone:

E veja o mesmo componente no IE10 do Windows 8:



Visuais totalmente diferentes mas totalmente adaptados à plataforma em questão.

**Suporte ao input range**

Todos os browsers modernos suportam o input range. Você terá problemas porém em versões mais antigas. O IE só suporta a partir do 10, o Android a partir do 4.2, e o Firefox no 23.

http://caniuse.com/input-range

Lembre que aqui no curso estamos estudando novas ideias. Se você precisar suportar os navegadores antigos nesse exercício, sempre poderá substituir por um `<select>` simples ou um conjunto de radio buttons.
Funcionalmente, terão o mesmo resultado.

## 9.11 - Exercícios: seletor de tamanho

1. Implemente a funcionalidade de escolher o tamanho da roupa usando um input range do HTML5, colocando o código a seguir logo abaixo do nosso primeiro fieldset:

```
<fieldset class="tamanhos">
 <legend>Escolha o tamanho:</legend>

 <input type="range" min="36" max="46" value="42" step="2" name="tamanho"
id="tamanho">
</fieldset>
```

Teste seu funcionamento nos browsers modernos.

2. Temos dois fieldsets, um para escolher cor e outro, tamanho. No mobile, eles ficam um em cima do outro. No desktop, podemos posicioná-los lado a lado.

**Dentro** da media query anterior, acrescente:

```
fieldset {
  display: inline-block;
  vertical-align: top;

  margin: 1em 0;
  min-width: 200px;
  width: 45%;
}
```



## 9.12 - Tabelas

O uso de tabelas era muito comum há alguns anos para a definição de áreas. Seu uso para essa finalidade acabou se tornando prejudicial pela complexidade da marcação, o que dificulta bastante a manutenção das páginas. Além disso havia uma implicação direta na definição de relevância do conteúdo das tabelas para os indexadores de conteúdo por mecanismos de busca.

Ainda assim, hoje, quando queremos exibir uma série de dados tabulares, é indicado o uso da tag de **tabela** `<table>`.

```
<table>
  <tr>
    <th>Título da primeira coluna</th>
    <th>Título da segunda coluna</th>
  </tr>
```

```
  <tr>
    <td>Linha 1, coluna 1.</td>
    <td>Linha 1, coluna 2.</td>
  </tr>
  <tr>
    <td>Linha 2, coluna 1.</td>
    <td>Linha 2, coluna 2.</td>
  </tr>
</table>
```

Note que na primeira linha `<tr>` da tabela, as células são indicadas com a tag `<th>`, o que é útil para diferenciar seu conteúdo das células de dados.

Existem diversas maneiras de se alterar uma estrutura de uma tabela, como por exemplo indicamos que uma célula `<td>` ou `<th>` ocupa mais de uma linha de altura por meio do atributo `rowspan`, ou então que ela ocupa mais de uma coluna de largura com o uso do atributo `colspan`.

Podemos adicionar um título à nossa tabela com a tag `<caption>`.

Ainda existem as tags `<thead>`, `<tfoot>` e `<tbody>`, que servem para agrupar linhas de nossa tabela. Vale ressaltar que dentro do grupo `<thead>` devemos ter apenas linhas contendo a tag `<th>` como célula.

Outra tag de agrupamento que temos na tabela é a que permite que sejam definidas as colunas, é a tag `<colgroup>`. Dentro dessa tag definimos uma tag `<col>` para cada coluna, e dessa maneira podemos adicionar alguns atributos que influenciarão todas as células daquela coluna.

A seguir um exemplo completo de como utilizar essas tags dentro de uma tabela.

```
<table>
  <caption>Quantidade e preço de camisetas.</caption>
  <colgroup>
    <col width="10%">
    <col width="40%">
    <col width="30%">
    <col width="20%">
  </colgroup>
  <thead>
    <tr>
      <th rowspan="2">
      <th colspan="2">Quantidade de Camisetas</th>
      <th rowspan="2">Preço</th>
    </tr>
    <tr>
      <th>Amarela</th>
      <th>Vermelha</th>
```

```
    </tr>
  </thead>

<tfoot>
    <tr>
      <td>
      <td>Total de camisetas amarelas: 35</td>
      <td>Total de camisetas vermelhas: 34</td>
      <td>Valor total: $45.00</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Polo</td>
      <td>12</td>
      <td>5</td>
      <td>$30.00</td>
    </tr>
    <tr>
      <td>Regata</td>
      <td>23</td>
      <td>29</td>
      <td>$15.00</td>
    </tr>
  </tbody>
</table>
```

## 9.13 – Exercícios: detalhes

1. Crie a seção de detalhes logo abaixo da `div` com a classe `produto`, mas ainda **dentro** do container:

```
<div class="container">
  <div class="produto">
  ...
  </div>
  <div class="detalhes">
    <h2>Detalhes do produto</h2>

    <p>Esse é o melhor casaco de Cardigã que você já viu. Excelente
    material italiano com estampa desenhada pelos artesãos da
    comunidade de Krotor nas ilhas gregas. Compre já e receba hoje
    mesmo pela nossa entrega a jato.</p>
  </div>
</div>
```

2. O estilo é bastante simples, apenas para deixar o texto mais bonito. Adicione no final, fora da media query que fizemos antes:

```
.detalhes {
  padding: 2em 0;
}
.detalhes h2 {
  font-size: 1.5em;
```

```
  line-height: 2;
}
.detalhes p {
  margin: 1em 0;
  font-size: 1em;
  line-height: 1.5;
  max-width: 36em;
}

@media (min-width: 500px) {
  .detalhes {
    font-size: 1.2em;
  }
}
```

3. Crie uma tabela com características do produto contendo informações técnicas. **Adicione** dentro da div detalhes:

```
<table>
  <thead>
    <tr>
      <th>Característica</th>
      <th>Detalhe</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Modelo</td>
      <td>Cardigã 7845</td>
    </tr>
    <tr>
      <td>Material</td>
      <td>Algodão e poliester</td>
    </tr>
    <tr>
      <td>Cores</td>
      <td>Azul, Rosa e Verde</td>
    </tr>
    <tr>
      <td>Lavagem</td>
      <td>Lavar a mão</td>
    </tr>
  </tbody>
</table>
```

4. Estilize a tabela para deixá-la mais agradável. Use o seletor de filhos múltiplos para um estilo zebrado.

Adicione o estilo fora da media query:

```
table {
  border-spacing: 0.2em;
  border-collapse: separate;
}
thead {
```

```
  background-color: #999;
}
thead th {
  font-weight: bold;
  padding: 0.3em 1em;
  text-align: center;
}
td {
  padding: 0.3em;
}
tr:nth-child(2n) {
  background-color: #ccc;
}
td:first-child {
  font-style: italic;
}
```

## 9.14 – Exercícios opcionais: fundo

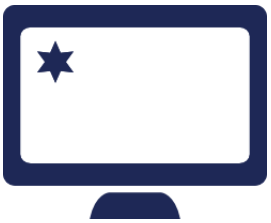
1. Para implementarmos o fundo cinza em tela cheia, vamos precisar de um novo elemento pai para conter todos os elementos da página. Crie um `<div class="produto-back">` ao redor do div container que tínhamos antes.

Apenas para referência, nesse momento, seu HTML deve estar mais ou menos assim:

```
<div class="produto-back">
  <div class="container">
    <div class="produto">
      ...
    </div>
    <div class="detalhes">
      ...
    </div>
  </div>
</div>
```

2. O estilo é bastante simples, apenas usando uma cor e bordas sutis para criar um efeito mais elegante:

```
.produto-back {
  background-color: #F2F2F2;
  margin-top: 1em;
  border-top: 2px solid #ccc;
}
.cores label img {
  border: 2px solid #ccc;
}
```



A **Alura** oferece dezenas de **cursos online** em sua plataforma exclusiva de ensino que favorece o aprendizado com a **qualidade** reconhecida da Caelum. Você pode escolher um curso nas áreas de Java, Ruby, Web, Mobile, .NET e outros, com uma **assinatura** que dá acesso a todos os cursos.

Conheça os cursos online Alura.

### 9.15 – Discussão em aula: importância do Progressive Enhancement na Web atual

CAPÍTULO ANTERIOR:

Introdução a PHP

PRÓXIMO CAPÍTULO:

PHP: parâmetros e bancos de dados

Facebook

Newsletter

Casa do Código

Twitter