

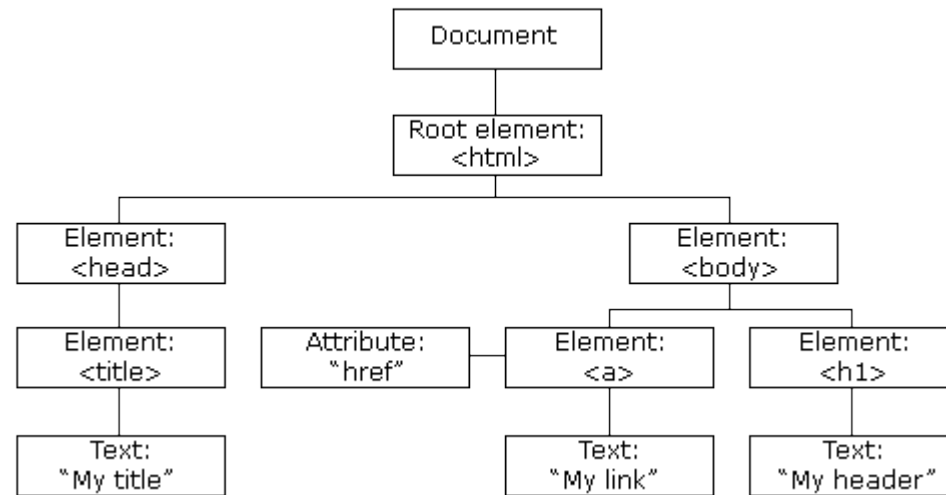
Web Applicaties II

Document Object Model

Wat is DOM?

Wat is DOM?

- Wat is DOM?
 - Via JavaScript willen we graag heel flexibel de HTML-pagina kunnen aanpassen.
 - Elke HTML-pagina kan je zien als een boomstructuur.
Deze boomstructuur kan je manipuleren via JavaScript
 - **DOM is een standaard waarmee we HTML-elementen uit de webpagina kunnen opvragen, wijzigen, toevoegen of verwijderen.**



Wat is DOM?

- Wat is DOM?
 - Tree API ontworpen door W3C: DOM Level 3 : <http://www.w3.org/DOM/>
 - Vanuit een programma kan je een HTML document als volledig object benaderen
 - Bouwt een boomvoorstelling van het HTML document in het geheugen
 - Biedt klassen en methodes (tree-gebaseerde API) aan om via code door de boom te navigeren en bewerkingen uit te voeren.
 - Platform- en taalafhankelijke API
 - Ontworpen voor HTML en XML

Wat is DOM?

- DOM stelt het HTML Document voor **als een boom met nodes**. De nodes stellen de elementen uit het HTML document voor.
- Adhv DOM
 - Toegang tot de inhoud van een document: elementen, attributen,...
 - Wijzigen van de inhoud van een document
 - Creatie van nieuwe documenten in het geheugen
- Nadelen
 - Volledige boom wordt in geheugen opgeslagen, daarom vooral bruikbaar bij kleine documenten die moeten gewijzigd worden
 - Niet bruikbaar voor grote documenten

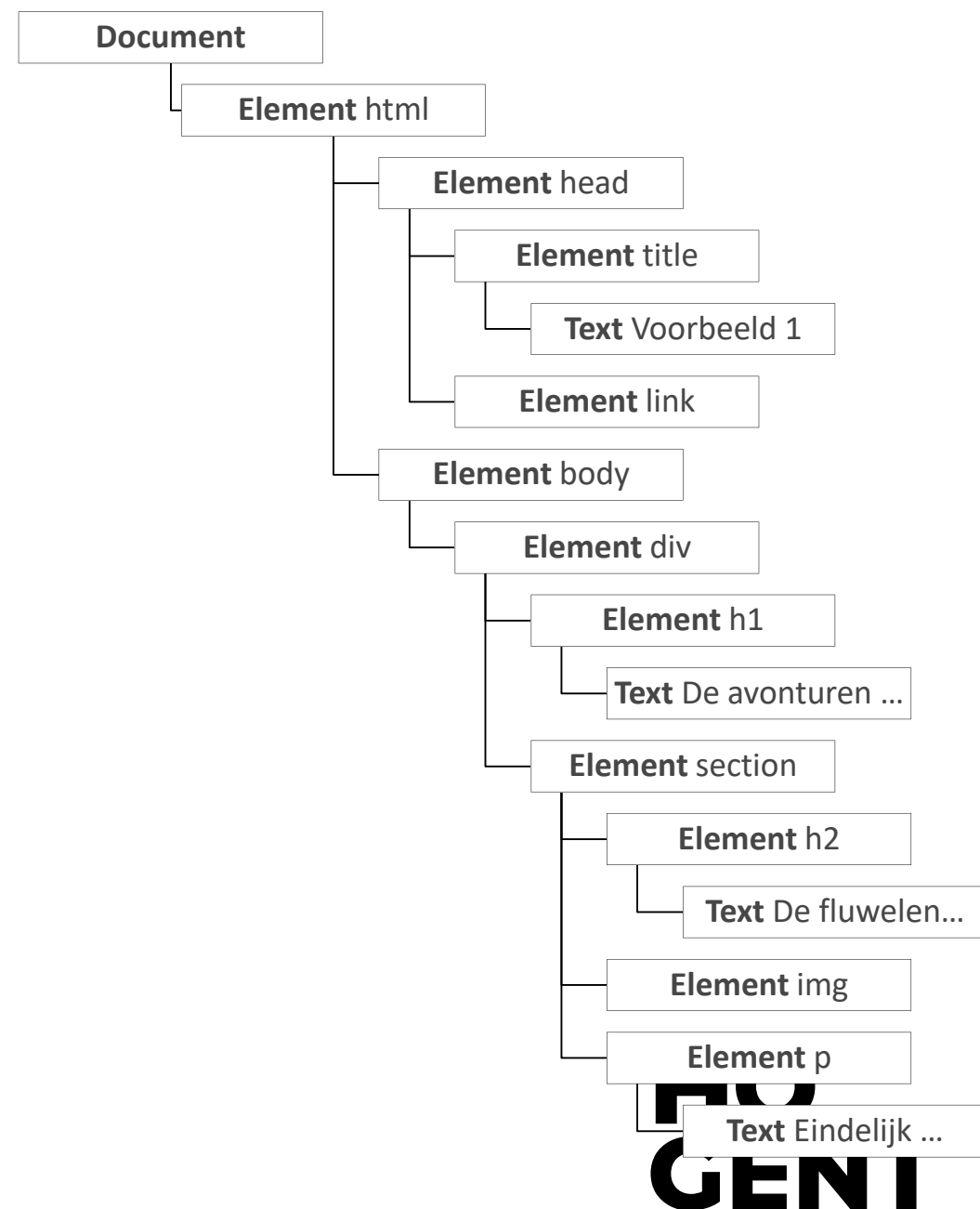
Wat is DOM?

- Elke HTML-pagina is een collectie (=verzameling) van DOM nodes
- Een node kan zijn:
 - **Document**: de top-node in de DOM-boom. Stelt het volledige document voor. Je tekent het net boven het html-element.
 - **Element**: elke html-tag is een element in de boom
 - **Text**: de tekst-inhoud van een bepaald element is een text-node (voorgesteld als kind-element van het welbepaalde element)
Let op: witruimte tussen elementen wordt ook voorgesteld als een text-node.
 - **Attribuut**: een attribuut van een node is bereikbaar via het element, maar wordt niet getekend in de boomstructuur
 - ...

Voorbeeld

.html

```
<html>
  <head>
    <title>Voorbeeld 1</title>
    <link rel="stylesheet"
          href="css/urbanus.css" />
  </head>
  <body>
    <div class="main">
      <h1>De avonturen van Urbanus</h1>
      <section id="urbanus_140">
        <h2>De Fluwelen Grapjas</h2>
        
        <p>Eindelijk erkenning! ...</p>
      </section>
    </div>
  </body>
</html>
```



Dynamisch HTML code genereren

Voorbeeld

- In de volgende slides wordt het onderstaande voorbeeld uitgewerkt
 - De gebruiker selecteert een categorie
 - Het aantal zoekertjes uit die categorie verschijnt
De zoekertjes uit die categorie worden getoond aan de linkerkant
 - Als de gebruiker klikt op een zoekertje aan de linkerkant, verschijnen de details aan de rechterkant.
De tekst van het zoekertje waarop werd geklikt, komt in vet
 - Als de gebruiker klikt op een thumbnail aan de rechterkant, wordt de grote afbeelding aangepast

Voorbeeld



Kies een categorie: **Keuken** ▼

Aantal zoekresultaten: 3



Tajine te koop



Super de luxe grill



Pottenset

Tajine te koop

Tajine - 27 cm - 2 liter van LE CREUSET

Prijs: €100



© 2dehands.be

**HO
GENT**

Voorbeeld – HTML code

- Hieronder staat de HTML code. Let op de id's

```
<body>
  <header>
    <div id="logo"></div>
  </header>
  <aside>
    <div id="zoekpaneel">
      <label for="categorie">Kies een categorie:</label>
      <select name="categorie" id="categorie">
        <option value="Alles">Alles</option>
      </select>
    </div>
  </aside>
  <div id="wrapper">
    <div id="aantal"></div>
    <div id="zoekresultaten"></div>
    <div id="productdetails"></div>
  </div>
  <footer>&copy; 2dehands.be</footer>
  <script type="text/javascript" src="js/2dehands.js"></script>
</body>
```

Voorbeeld

- De klasse Product bevat de properties van één product:

- id
- eigenaar
- postcode
- gemeente
- titel
- omschrijving
- prijs
- categorie
- afbeeldingen

```
class Product {  
    constructor(id, eigenaar, postcode, gemeente,  
        titel, omschrijving, prijs, categorie,  
        afbeeldingen) {  
        this.id = id;  
        this.eigenaar = eigenaar;  
        this.postcode = postcode;  
        this.gemeente = gemeente;  
        this.titel = titel;  
        this.omschrijving = omschrijving;  
        this.prijs = prijs;  
        this.categorie = categorie;  
        this.afbeeldingen = afbeeldingen;  
    }  
    // getters  
    // setters  
}
```

Repository

**HO
GENT**

Repository

- In dit voorbeeld worden alle producten opgeslagen in de ProductenRepository.
- Een repository is een centrale plaats waar data opgeslagen wordt en die een aantal eenvoudige bewerkingen rond deze data implementeert, bijvoorbeeld
 - Één data – item opvragen
 - Alle data opvragen
 - ...
- Voorlopig is de data hard – coded
- In een verdere les zal behandeld worden hoe je data kan toevoegen aan de repository vertrekkend van een JSON bestand of van een webservice.

Repository

- De klasse ProductenRepository bevat de property en de methodes van de repository:

```
class ProductenRepository {  
    constructor() {  
        this._producten = [];  
        this.haalProductenOp();  
    }  
  
    voegProductToe(product) {...}  
    geefAlleCategorieen() {...}  
    geefProduct(id) {...}  
    geefProductenUitCategorie(categorie) {...}  
    haalProductenOp() {...}  
}
```

- Het kan handig zijn om de code van haalProductenOp dicht te vouwen

```
48  | haalProductenOp() { ...  
257  | }
```

Repository

- De functionaliteit van de ProductenRepository moet nog verder uitgebreid worden
 - voegProductToe → om één product toe te voegen aan de array producten. Het product wordt meegegeven als parameter.
 - geefProduct → om één product op te halen op basis van de id
 - geefAlleCategorieën → om de verschillende unieke categorieën op te halen
 - geefProductenUitCategorie → om alle producten van één categorie op te halen
 - haalProductenOp → de data wordt toegevoegd aan de array producten

ProductenRepository

- In de constructor van de klasse ProductenComponent wordt een object van de klasse ProductenRepository gedeclareerd en geïnitieerd.

```
class ProductenComponent {  
    constructor() {  
        this._productenRepository = new ProductenRepository();  
    }  
  
    ...  
}
```

init()

- In de init() – functie wordt een object van de klasse ProductenComponent gedeclareerd en geïnitialiseerd.

```
const init = function () {  
    const productenComponent = new ProductenComponent();  
};
```

productenRepository - voegProductToe

- Voeg code toe aan de functie voegProductToe zodat een nieuw Product wordt toegevoegd aan de array producten

```
voegProductToe(product) {  
    this._producten.push(product);  
}
```

- Test de code uit door de volgende regel uit commentaar te zetten in de init - functie

```
const init = function () {  
    const productenComponent = new ProductenComponent();  
    console.log(productenComponent.productenRepository.producten);  
    ...  
};
```

ProductenRepository - geefProduct

- Voeg code toe aan de functie geefProduct zodat het Product met de opgegeven id wordt geretourneerd.
Maak gebruik van de functie find.

```
geefProduct(id) {  
    return this._producten.find(p => p.id === id);  
}
```

- Test de code uit door de volgende regel uit commentaar te zetten in de init - functie

```
const init = function () {  
    const productenComponent = new ProductenComponent();  
    console.log(productenComponent.productenRepository.producten);  
    console.log(productenComponent.productenRepository.geefProduct('P011'));  
};
```

ProductenRepository - geefAlleCategorieen

- Voeg code toe aan de functie geefAlleCategorieen zodat een lijst met de categorieën geretourneerd wordt. Elke categorie mag maar één keer voorkomen in de lijst. De lijst is alfabetisch gesorteerd.
Maak gebruik van een Set en van de functie map die voor elk product de categorie retourneert. Converteer de Set naar een array om te kunnen sorteren

```
geefAlleCategorieen() {  
  // Mbv map wordt de array van producten overlopen en wordt een array  
  // geretourneerd met per product de categorie  
  // Deze array wordt gebruikt bij de creatie van de set. Als de categorie al bestaat in  
  // de set, zal die niet opnieuw worden toegevoegd. Dat is juist de essentie van een set  
  // Mbv de spread - operator wordt de set omgevormd naar een  
  // array. Daarop kan dan gemakkelijk de sort - bewerking toegepast worden  
  return [...new Set(this._producten.map(product => product.categorie))].sort();  
}
```

- Test de code uit door de volgende regel uit commentaar te zetten in de init - functie

```
const init = function () {  
  ...  
  console.log(productenComponent.productenRepository.geefAlleCategorieen());  
};
```

ProductenRepository - geefProductenUitCategorie

- Voeg code toe aan de functie geefProductenUitCategorie zodat een lijst met de producten uit de opgegeven categorie geretourneerd wordt. Het zoeken moet onafhankelijk zijn van hoofdletters of kleine letters. Maak gebruik van de functie filter.

```
geefProductenUitCategorie(categorie) {  
  if (categorie === 'Alles') return this._producten;  
  // alle producten worden overlopen  
  // de array van de producten met de opgegeven categorie wordt geretourneerd  
  // toUpperCase() --> het zoeken is onafhankelijk van hoofdletters of kleine letters  
  return this._producten.filter(p => p.categorie.toUpperCase() === categorie.toUpperCase());  
});  
}
```

- Test de code uit door de volgende regel uit commentaar te zetten in de init - functie

```
const init = function () {  
  ...  
  console.log(productenComponent.productenRepository.geefProductenUitCategorie('Tuin'));  
};
```

Wijzigingen aanbrengen in een document

Wijzigen van een document

- Voor het wijzigen van een document zijn er verschillende werkwijzes mogelijk
 - Mbv insertAdjacentHTML(..., ...)
 - Mbv innerHTML(...)
 - Mbv createElement(...) + setAttribute(...) + appendChild(...)

Wijzigen van een document - insertAdjacentHTML

- De functie **insertAdjacentHTML()** parset de meegegeven tekst als HTML en voegt deze toe aan de DOM structuur op de meegegeven positie, zoals hieronder geïllustreerd:

```
<!-- beforebegin : voor het element -->
<p>
  <!-- afterbegin : in het element zelf, voor het eerste kind -->
  foo
  <!-- beforeend : in het element zelf, na het laatste kind -->
</p>
<!-- afterend : achter het element -->
```

- Bijvoorbeeld

```
// <div id="one">one</div>
const d1 = document.getElementById('one');
d1.insertAdjacentHTML('afterend', '<div id="two">two</div>');
// <div id="one">one</div><div id="two">two</div>
```

Wijzigen van een document - innerHTML

- Via de element-property **innerHTML** kan je de inhoud van een element instellen .
- Bijvoorbeeld

```
// <div id="one"></div>
const d1 = document.getElementById('one');
d1.innerHTML = '<div id="two">two</div>';
// <div id="one">
//     <div id="two">two</div>
// </div>
```

Wijzigen van een document - createElement

- De methode **createElement(tagName)** creëert een **Element** node met de opgegeven naam
- De methode **createTextNode(text)** creëert een **Text** node met de opgegeven *text*
- De methode **appendChild(newChild)** voegt een node toe na de laatste child node van de opgegeven node. De nieuw toegevoegde node wordt geretourneerd.
- De methode **setAttribute(name, value)** wordt gebruikt om een attribuut van een element in te stellen.
- Bijvoorbeeld

```
const d1 = document.getElementById('one');  
// Het <div> - element wordt gecreëerd  
const divElement = document.createElement("div");  
// Dit <div> - element krijgt een attribuut id => <div id="two"></div>  
divElement.setAttribute("id", "two");  
// Er wordt een TextNode gecreëerd  
const divTekst = document.createTextNode("two");  
// De tekst wordt toegevoegd aan het <div> - element => bijvoorbeeld <div id="two">two</div>  
divElement.appendChild(divTekst);  
// Het <div> - element wordt toegevoegd aan het element met id one  
d1.appendChild(divElement);
```

`<div id="one"></div>`

`<div id="one">
 <div id="two">two</div>
</div>`

Wijzigen van documentstructuur

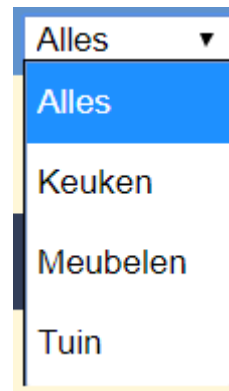
- De methode **removeChild(removechild)** verwijdert een child node van de document tree. De verwijderde node wordt geretourneerd (en kan dus eventueel nog verder gebruikt worden)
- De methode **replaceChild(newchild, oldchild)** vervangt de *oldchild* node door de *newchild* node. De verwijderde oldchild node wordt geretourneerd (en kan dus eventueel nog verder gebruikt worden).
- De methode **cloneNode(deepcopy)** creëert en retourneert een exacte kloon. Als de boolean parameter *deepcopy* gelijk is aan true wordt een kloon gemaakt van de node zelf en de volledige subtree, anders wordt enkel een kloon gemaakt van de node zelf.

Wijzigen van documentstructuur – Voorbeeld 1

- Momenteel heeft de keuzelijst bovenaan enkel de optie Alles
- We willen op basis van de inhoud van productenRepository dynamisch de correcte rubrieken toevoegen
- Daartoe zullen we de functie categorieënToHtml implementeren. In deze functie worden de verschillende categorieën opgehaald uit de productenRepository en wordt dynamisch de volgende HTML code gegenereerd.

Kies een categorie: Alles ▼

```
<select name="categorie" id="categorie">  
  <option value="Alles">Alles</option>  
  <option value="Keuken">Keuken</option>  
  <option value="Meubelen">Meubelen</option>  
  <option value="Tuin">Tuin</option>  
</select>
```



Alles ▼

Alles

Keuken

Meubelen

Tuin

Wijzigen van documentstructuur – Voorbeeld 1

- Daarbij wordt gebruik gemaakt van de JavaScript functie `insertAdjacentHTML`

```
categorieenToHtml() {  
  this._productenRepository.geefAlleCategorieen().forEach(categorie => {  
    // Het nieuwe <option>-element met attribuut value en tekst komt achteraan het element met id categorie  
    document.getElementById('categorie').insertAdjacentHTML('beforeend',  
      `  });  
}
```

- Test de code uit door de volgende regel toe te voegen aan de init - functie

```
const init = function () {  
  ...  
  console.log(productenComponent.productenRepository.geefProductenUitCategorie('Tuin'));  
  productenComponent.categorieenToHtml();  
};
```

Wijzigen van documentstructuur – Voorbeeld 2

- In de linkerkolom moeten de producten van de gekozen categorie verschijnen. Daartoe zullen we de functie zoekresultatenToHtml implementeren. In de volgende slides wordt dit stap voor stap uitgewerkt.
- In deze functie
 - wordt het aantal gevonden zoekresultaten uitgeschreven
 - wordt het element met id zoekresultaten leeg gemaakt
 - worden de verschillende producten van de opgegeven categorie opgehaald uit de productenRepository en wordt per product dynamisch de HTML code gegenereerd.
 - Het gecreëerde <div> - element wordt toegevoegd aan het element met id zoekresultaten

Wijzigen van documentstructuur – Voorbeeld 2

- Opdat we niet altijd eerst een rubriek zouden moeten selecteren vooraleer we zien of de code werkt (dit is voorlopig nog niet geïmplementeerd), wordt de init – functie als volgt aangepast

```
const init = function () {  
    ...  
    console.log(productenComponent.productenRepository.geefProductenUitCategorie('Tuin'));  
    productenComponent.categorieenToHtml();  
    productenComponent.zoekresultatenToHtml('Tuin');  
};
```


Wijzigen van documentstructuur – Voorbeeld 2

- Voor het uitschrijven van het aantal zoekresultaten wordt gebruik gemaakt van de functie `insertAdjacentHTML` en voor het leegmaken van het element met id `zoekresultaten` en van het element met id `zoekresultaten` van de functie `innerHTML`.

```
zoekresultatenToHtml(categorie) {  
    // De producten van de opgegeven categorie worden opgehaald uit de repository  
    const producten = this._productenRepository.geefProductenUitCategorie(categorie);  
    // Het element met id zoekresultaten wordt leeggemaakt  
    document.getElementById('zoekresultaten').innerHTML = '';  
    // Het element met id aantal wordt leeggemaakt  
    document.getElementById('aantal').innerHTML = '';  
    // Het aantal zoekresultaten wordt getoond  
    // Merk op dat het onderstaande een nieuw <h4> - element is.  
    document.getElementById('aantal').insertAdjacentHTML('afterbegin',  
        `<h4>Aantal zoekresultaten: ${producten.length}</h4>`);  
}
```

Aantal zoekresultaten: 7

```
▼<div id="aantal">  
    <h4>Aantal zoekresultaten: 7</h4>  
</div>
```

Wijzigen van documentstructuur – Voorbeeld 3

- De functie zoekresultatenToHtml moet verder uitgewerkt worden. Voor elk opgehaald product moet de volgende HTML – code dynamisch gegenereerd worden:

```
▼<div id="zoekresultaten">  
  ▼<div id="P009">  
      
    <p>Rustieke tuinset</p>  
  </div>
```

- Het nieuw gecreëerde <div> - element moet op het einde toegevoegd worden aan het element met id zoekresultaten
- De opmaak “die verschijnt” (zoals de rand) is afkomstig van het CSS bestand
- Voor het creëren van het <div> - element, wordt de functie createElement gebruikt
- Voor het instellen van het attribuut id, wordt de functie setAttribute gebruikt
- Om het <div> - element achteraan toe te voegen aan het element met id zoekresultaten, wordt de functie appendChild gebruikt



Wijzigen van documentstructuur – Voorbeeld 3

```
zoekresultatenToHtml(categorie) {  
  // De producten van de opgegeven categorie worden opgehaald uit de repository  
  const producten = this._productenRepository.geefProductenUitCategorie(categorie);  
  // Het element met id zoekresultaten wordt leeggemaakt  
  document.getElementById('zoekresultaten').innerHTML = '';  
  // Het element met id aantal wordt leeggemaakt  
  document.getElementById('aantal').innerHTML = '';  
  // Het aantal zoekresultaten wordt getoond  
  // Merk op dat het onderstaande een nieuw <h4> - element is.  
  document.getElementById('aantal').insertAdjacentHTML('afterbegin', `

#### Aantal zoekresultaten: ${producten.length}</h4>`); // De opgehaalde producten worden één voor één overlopen producten.forEach((product) => { // <div></div> wordt gecreëerd const divElement = document.createElement('div'); // het <div> - element krijgt een attribuut id: <div id="..."></div> divElement.setAttribute("id", product.id); divElement.insertAdjacentHTML('afterbegin', `${product.titel} <p>${product.titel}</p>`); // Het <div> - element wordt achteraan het element met id zoekresultaten toegevoegd document.getElementById('zoekresultaten').appendChild(divElement); }) }


```

Werken met attributen

Werken met attributen

- In de voorbeelden werd al gebruik gemaakt van de functie `setAttribute(name, value)`
- Elk HTML-element heeft 0 of meer attributen, die gebruikt worden om extra informatie te voorzien voor een bepaald element. Er bestaan verschillende manieren binnen DOM om deze attributen te verwerken
 - Manier 1: met behulp van de methodes `getAttribute()`, `setAttribute()`, `removeAttribute()`
 - Manier 2: rechtstreeks gescheiden door een punt (bijvoorbeeld `.id`)

Manier1:

- De volgende methodes werken voor alle (ook niet standaard) attributen
 - `getAttribute (attributenam)`
 - `setAttribute (attributenam, attributevalue)`
 - `removeAttribute (attributenam)`
- Dit werd al geïllustreerd in het voorbeeld

Werken met attributen

Manier2:

- Voor standaard attributen (= attributen die herkend worden door de browser) kunnen we gebruik maken van . om de waarde op te vragen of te wijzigen
- Om de waarde van het attribuut class op te vragen => **.className** gebruiken!
- In de functie zoekresultatenToHtml willen we dat de achtergrond van de <div> - elementen alternerend wisselt. Daartoe zullen we gebruik maken van de CSS-class .wit
- Bovendien worden – ter illustratie – de regels code met setAttribute herschreven, met behulp van . – notatie

```
// De even <div> - elementen worden wit  
if (index % 2 === 0) {divElement.className = 'wit'; }
```

- Een alternatief voor het voorgaande:

```
// De even <div> - elementen worden wit  
if (index % 2 === 0) {divElement.style.backgroundColor = 'white';}
```

Wijzigen van documentstructuur – Voorbeeld 3

```
zoekresultatenToHtml(categorie) {  
  // De producten van de opgegeven categorie worden opgehaald uit de repository  
  const producten = this._productenRepository.geefProductenUitCategorie(categorie);  
  // Het element met id zoekresultaten wordt leeggemaakt  
  document.getElementById('zoekresultaten').innerHTML = '';  
  // Het element met id aantal wordt leeggemaakt  
  document.getElementById('aantal').innerHTML = '';  
  // Het aantal zoekresultaten wordt getoond  
  // Merk op dat het onderstaande een nieuw <h4> - element is.  
  document.getElementById('aantal').insertAdjacentHTML('afterbegin', `

#### Aantal zoekresultaten: ${producten.length}</h4>`); // De opgehaalde producten worden één voor één overlopen producten.forEach((product, index) => { // <div></div> wordt gecreëerd const divElement = document.createElement('div'); // Voor de even <div> - elementen wordt de class .wit toegevoegd if (index % 2 === 0) { divElement.className = 'wit'; } // Dit <div> - element krijgt een attribuut id => <div id="..."></div> divElement.id = product.id; divElement.insertAdjacentHTML('afterbegin', `${product.titel} ` ${product.titel}</p>`); // Het <div> - element wordt achteraan het element met id zoekresultaten toegevoegd document.getElementById('zoekresultaten').appendChild(divElement); }) }


```

Events

**HO
GENT**

Events

- Voorlopig gebeurt er nog niets als er de keuzelijst aangepast wordt.
- We willen echter dat de producten getoond worden van de gekozen categorie
- Pas daarom de init – functie aan. Voeg code toe zodat de functie zoekresultatenToHtml aangeroepen wordt als een andere categorie gekozen wordt én dat het element met id productdetails weer onzichtbaar wordt
- Merk op dat er hier gebruik gemaakt wordt van de . –notatie om het attribuut value op te vragen

```
const init = function () {  
    ...  
    productenComponent.categorieenToHtml();  
    // Initieel (= bij het laden van de pagina) laten we alle producten zien  
    productenComponent.zoekresultatenToHtml('Alles');  
    document.getElementById('categorie').onchange = () => {  
        // Met behulp van de . - notatie wordt het attribuut value opgevraagd  
        // De functie zoekresultatenToHtml wordt opgeroepen  
        productenComponent.zoekresultatenToHtml(document.getElementById('categorie').value);  
        // Het element met id productdetails wordt onzichtbaar gemaakt  
        document.getElementById('productdetails').style.display = 'none';  
    }  
};
```

Events

- Op analoge manier moet ervoor gezorgd worden dat de details van het product getoond worden aan de rechterkant wanneer in de linkerkolom op een product geklikt wordt.
- Daartoe is de functie detailsToHtml is reeds geïmplementeerd.
- In zoekresultatenToHtml:

```
zoekresultatenToHtml(categorie) {  
  ...  
  // De opgehaalde producten worden één voor één overlopen  
  producten.forEach((product) => {  
    // <div></div> wordt gecreëerd  
    const divElement = document.createElement('div');  
    // Voor de even <div> - elementen wordt de class .wit toegevoegd  
    if (index % 2 === 0) { divElement.className = 'wit'; }  
    // Dit <div> - element krijgt een attribuut id => <div id="..."></div>  
    divElement.id = product.id;  
    divElement.insertAdjacentHTML('afterbegin',  
    `  
    <p>${product.titel}</p>`);  
    // Het <div> - element wordt achteraan het element met id zoekresultaten toegevoegd  
    document.getElementById('zoekresultaten').appendChild(divElement);  
    // Er wordt geklikt op het <div> - element => de details aan de rechterkant moeten zichtbaar worden  
    divElement.onclick = () => {this.detailsToHtml(product.id);}  
  })  
}
```

Één of meerdere nodes selecteren

Één of meerdere nodes selecteren

- De methode **getElementById (id)** geeft het eerste element met de opgegeven *id*
- De zoekbewerking start meestal vanaf het document element, maar je kan evengoed van elk ander element starten
- De methode **getElementsByTagName (tagname)** retourneert een live HTML collection met alle elementen (0 of meer) met de opgegeven *tagname*
- Een HTML collection is een array-like object van elementen en voorziet properties en functies om elementen te selecteren in de lijst
- De methode **getElementsByClassName (value)** retourneert een HTML collection met alle elementen met de opgegeven *value* als waarde voor het class attribuut.
- De methode **querySelector (CSS selector)** retourneert het eerste element dat voldoet aan de opgegeven CSS selector.
- De methode **querySelectorAll (CSS selector)** retourneert een HTML collection met alle elementen die voldoen aan de opgegeven CSS selector.

Één of meerdere nodes selecteren - Voorbeeld

- Om duidelijk te maken op welk product in de linkerkolom er werd geklikt, willen we de tekst van dat product in vet zetten
- Dit zal gebeuren door er de class tekstVet aan toe te voegen
- In zoekresultatenToHtml:

```
zoekresultatenToHtml(categorie) {  
  ...  
  // De opgehaalde producten worden één voor één overlopen  
  producten.forEach((product) => {  
    ...  
    // Het <div> - element wordt achteraan het element met id zoekresultaten toegevoegd  
    document.getElementById('zoekresultaten').appendChild(divElement);  
    // Er wordt geklikt op het <div> - element  
    divElement.onclick = () => {  
      // de details aan de rechterkant moeten zichtbaar worden  
      this.detailsToHtml(product.id);  
      // de tekst moet in vet komen  
      document.querySelector(`#${product.id} p`).setAttribute('class', 'tekstVet');  
    };  
  })  
}
```

Één of meerdere nodes selecteren - Voorbeeld

- Echter tekst die in vet staat, blijft in vet staan, ook als op een ander product wordt geklikt. De werkwijze moet aangepast worden:
 - Als op een product in de linkerkolom geklikt wordt
 - Van alle producten in de linkerkolom waarbij het <p> - element een class tekstVet heeft, wordt deze class verwijderd. Op die manier staat de tekst niet meer in vet. → Hiervoor wordt querySelector gebruikt
 - Van het product waarop werd geklikt, krijgt het <p> - element de class tekstVet → Hiervoor wordt querySelectorAll gebruikt

```
// Er wordt geklikt op het <div> - element
divElement.onclick = () => {
  // de details aan de rechterkant moeten zichtbaar worden
  this.detailsToHtml(product.id);

  // Alle elementen met class tekstVet binnen het element met id zoekresultaten
  // worden opgevraagd (voor het geval tekstVet nog ergens anders gebruikt zou worden)
  // De spread operator wordt gebruikt om er een array van te maken
  // zodat we vervolgens de forEach - functie kunnen toepassen
  [...document.querySelectorAll(`#zoekresultaten div p.tekstVet`)].forEach((value) => { value.removeAttribute('class'); })

  // de tekst moet in vet komen
  document.querySelector(`#${product.id} p`).setAttribute('class', 'tekstVet');
}
```

Cheat sheet

**HO
GENT**

The most common DOM methods at a glance

Reaching Elements in a Document

`document.getElementById('id')`: Retrieves the element with the given `id` as an object

`document.getElementsByTagName('tagname')`: Retrieves all elements with the tag name `tagname` and stores them in an array-like list

Reading Element Attributes, Node Values and Other Data

`node.getAttribute('attribute')`: Retrieves the value of the attribute with the name `attribute`

`node.setAttribute('attribute', 'value')`: Sets the value of the attribute with the name `attribute` to `value`

`node.nodeType`: Reads the type of the node (1 = element, 3 = text node)

`node.nodeName`: Reads the name of the node (either element name or `#textNode`)

`node.nodeValue`: Reads or sets the value of the node (the text content in the case of text nodes)

Navigating Between Nodes

`node.previousSibling`: Retrieves the previous sibling node and stores it as an object.

`node.nextSibling`: Retrieves the next sibling node and stores it as an object.

`node.childNodes`: Retrieves all child nodes of the object and stores them in an list. here are shortcuts for the first and last child node, named `node.firstChild` and `node.lastChild`.

`node.parentNode`: Retrieves the node containing `node`.

Creating New Nodes

`document.createElement(element)`: Creates a new element node with the name `element`. You provide the name as a string.

`document.createTextNode(string)`: Creates a new text node with the node value of `string`.

`newNode = node.cloneNode(bool)`: Creates `newNode` as a copy (clone) of `node`. If `bool` is `true`, the clone includes clones of all the child nodes of the original.

`node.appendChild(newNode)`: Adds `newNode` as a new (last) child node to `node`.

`node.insertBefore(newNode, oldNode)`: Inserts `newNode` as a new child node of `node` before `oldNode`.

`node.removeChild(oldNode)`: Removes the child `oldNode` from `node`.

`node.replaceChild(newNode, oldNode)`: Replaces the child node `oldNode` of `node` with `newNode`.

`element.innerHTML`: Reads or writes the HTML content of the given element as a string—including all child nodes with their attributes and text content.

Known browser quirks:

`getAttribute` and `setAttribute` are not reliable. Instead, assign the property of the element object directly: `obj.property = value`. Furthermore, some attributes are actually reserved words, so instead of `class` use `className` and instead of `for` use `HTMLfor`.

Real DOM compliant browsers will return linebreaks as text nodes in the `childNodes` collection, make sure to either remove them or test for the `nodeType`.