Object Composition

Destructuring, Aggregation, Concatenation, Delegation

Table of Contents

Objects Holding Other Objects

Object Composition

What is Object Composition?

- **Combining** simple objects or data types into more **complex ones**

```
let student = {
  firstName: 'Maria',
  lastName: 'Green',
  age: 22,
  location: { lat: 42.698, lng: 23.322 }
}
console.log(student);
console.log(student.location.lat);
```

Composing Objects

```
let name = "Sofia";
let population = 1325744;
let country = "Bulgaria";
let town = { name, population, country };
console.log(town);
// Object {name: "Sofia", population: 1325744,
country: "Bulgaria"}
```

Combine variables into object

```
town.location = { lat: 42.698, lng: 23.322 };
console.log(town); // Object {…, location: Object}
```

Combining Data with Functions

```
let rect = {
  width: 10,
  height: 4,
  grow: function(w, h) {
    this.width += w; this.height += h;
  },
  print: function() {
    console.log(`[${this.width} x ${this.height}]`);
  }
};
rect.grow(2, 3);
rect.print(); // [12 x 7]
```

Printing Objects: ToString() Function

```
let rect = {
  width: 10,
  height: 4,
  toString: function() {
    return `rect[${this.width} x ${this.height}]`;
  }
};
console.log(rect); // Object {width: 10, height: 4}
// This will invoke toString() to convert the object to String
console.log('' + rect); // rect[12 x 7]
```

Destructuring

Destructuring

- The ability to "**dive into**" an **array** or **object** and **reference** something inside of it more **directly**

```
const department = {
 name: "Engineering",
 data: {}
}
const { data } = department //now data references the data object directly
const objectList = [ { key: 'value' }, { key: 'value' }, { key: 'value' } ]
const [ obj, obj1, obj2 ] = objectList
// now each object can be referenced directly
```

Nested Destructuring

- Destructuring can work **beyond** the **top level**

```
const department = {
  name: "Engineering",
  data: {
    director: {
      name: 'John',
      position: 'Engineering Director'
    },
    employees: [],
    company: 'Quick Build'
  }
}
const {data: {director}} = department
// director is { name: 'John', position: 'Engineering Director'}
```

Destructuring Nested Arrays

- You need to know the **position** of what you're looking for
- Provide a **reference** variable (or comma placeholder) for each element up and until the one you're looking for

```
const departments = [['Engineering', ['secretary', 'director', 'worker']], ['
Accounting', ['director', 'accountant']]];


const [[name, positions]] = departments
// name is 'Engineering'
// positions is [ 'secretary', 'director', 'worker' ]
```

Objects and Arrays Destructuring

```
const employees = [{name: 'John', position: 'worker'},
{name: 'Jane', position: 'secretary'}]

const [{name}] = employees // name is 'John'
```

```
const company = {
  employees: ['John', 'Jane', 'Sam', 'Suzanne'],
  name: 'Quick Build',
}

const {employees:[employee]} = company // employee is 'John'
```

Forms of Object Composition

Aggregation

- Object is formed from an **enumerable collection** of **subobjects**
- An **aggregate** is an object which **contains** other objects
- When to use
  - Collections of objects which need to **share common operations**
  - When you want a single item to **share** the same interface as many items

Aggregation Example

```
let dataArray = [ { id: "a", score: 1 }, { id: "b", score: 2 },
{ id: "c", score: 5 }, { id: "a", score: 3 }, { id: "c", score: 2 }, ];
let res1 = dataArray.reduce((acc, curr, index, array) => {
  let same = acc.find(e => e.id === curr.id);
  if (!same) {
    acc.push(curr);
  } else {
    same.score += curr.score;
  }
  return acc;
}, []);
console.log(res1);
//[ { id: 'a', score: 4 }, { id: 'b', score: 2 }, { id: 'c', score: 7 } ]
```

Concatenation

- Concatenation is when an object is formed by adding **new properties** to an **existing object**
- When to use
  - merging JSON objects
  - Creating updates to immutable state
  - Etc…

Concatenation Example

```
const objs = [
                {name: 'Peter',age:35 },
            { age: 22 },
            {name: "Steven"},
            {height:180}
];
const concatenate = (a, o) => ({...a, ...o});
const c = objs.reduce(concatenate, {});
console.log(c);// { name: 'Steven', age: 22, height: 180 }
```

Delegation

- Delegation is commonly used to **imitate class inheritance** in JavaScript
- Composes objects by **linking** together an object delegation chain
  - An object forwards **property lookups** to another object
    - `[].map()` delegates to `Array.prototype.map()`

Delegation Example

```
const objs = [
        {name: 'Peter',age:35},
        {age: 22},
        {name: "Steven"},
        {height:180}
];
const delegate = (a, b) => Object.assign(Object.create(a), b);
const d = objs.reduceRight(delegate, {});
console.log(d); // { name: 'Peter', age: 35 }
console.log(d.height); // 180
```

Summary

- Object composition combines **data** and **functions** into JS objects

```
let r = {w:5, h:3, grow:function() { … }}
```

- Three main types of object composition
  - **Aggregation** - forming an object from an **enumerable** collection
  - **Concatenation** - adding new properties
  - **Delegation** - imitates class inheritance

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com


KINGSLAND
UNIVERSITY
THANK YOU