CODE SPITZ 86

OBJECT ORIENTED JAVASCRIPT

1 2 3 4 5

Strategy

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

Structure & control

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

Strategy

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

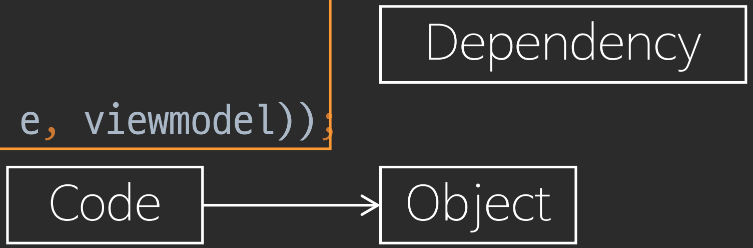
Strategy

Algorithm, Knowledge, Domain

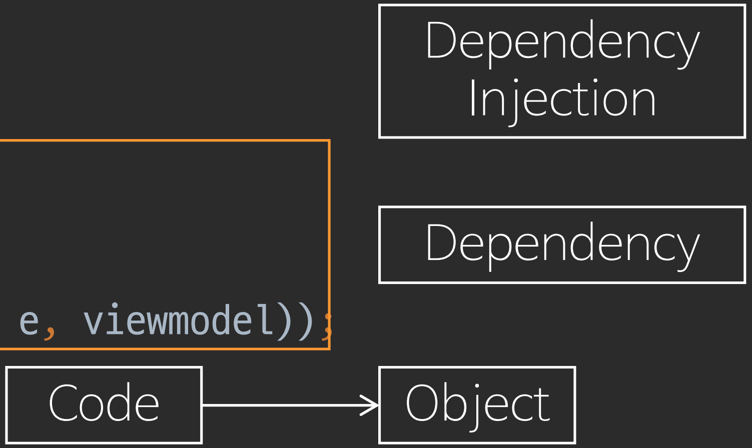
```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

$$\begin{tikzcd}\boxed{\text{Code}} \arrow[r] & \boxed{\text{Object}}\end{tikzcd}$$

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};

const Processor = class{
  process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw "override";}
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};

const Processor = class{
  process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw "override";}
};
```

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};

const Processor = class{
  process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw "override";}
};
```

Template method

```
const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};

const Processor = class{
  process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw "override";}
};
```

```
const Processor = class{
    cat;
    constructor(cat){
      this.cat = cat;
      Object.freeze(this);
    }
    process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
      this._process(vm, el, k, v);
    }
    _process(vm, el, k, v){throw "override";}
};
```

```
const Processor = class{
    cat;
    constructor(cat){
      this.cat = cat;
      Object.freeze(this);
    }
    process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
      this._process(vm, el, k, v);
    }
    _process(vm, el, k, v){throw "override";}
};

new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

```
const Processor = class{
    cat;
    constructor(cat){
      this.cat = cat;
      Object.freeze(this);
    }
    process(vm, el, k, v, _0=type(vm, ViewModel), _1=type(el, HTMLElement), _2=type(k, "string")) {
      this._process(vm, el, k, v);
    }
    _process(vm, el, k, v){throw "override";}
};
```

```
new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

```
new (class extends Processor{
  _process(vm, el, k, v){el[k] = v;}
})("properties")
```

```
new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes")
```

```
new (class extends Processor{
  _process(vm, el, k, v){el["on" + k] =e=>v.call(el, e, vm);}
})("events")
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

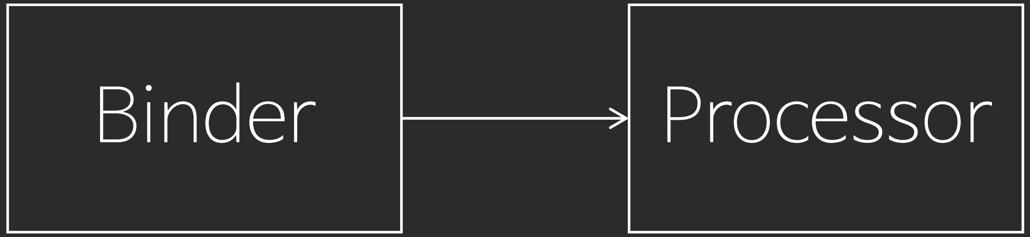
```
const Processor = class{
  cat;
  constructor(cat){
    this.cat = cat;
    Object.freeze(this);
  }
  process(vm, el, k, v, _0=type(vm,…
    this._process(vm, el, k, v);
  }
  _process(vm, el, k, v){throw…
};
```

```
new (class extends Processor{
  process(vm, el, k, v){el.style[k] = v;}
```

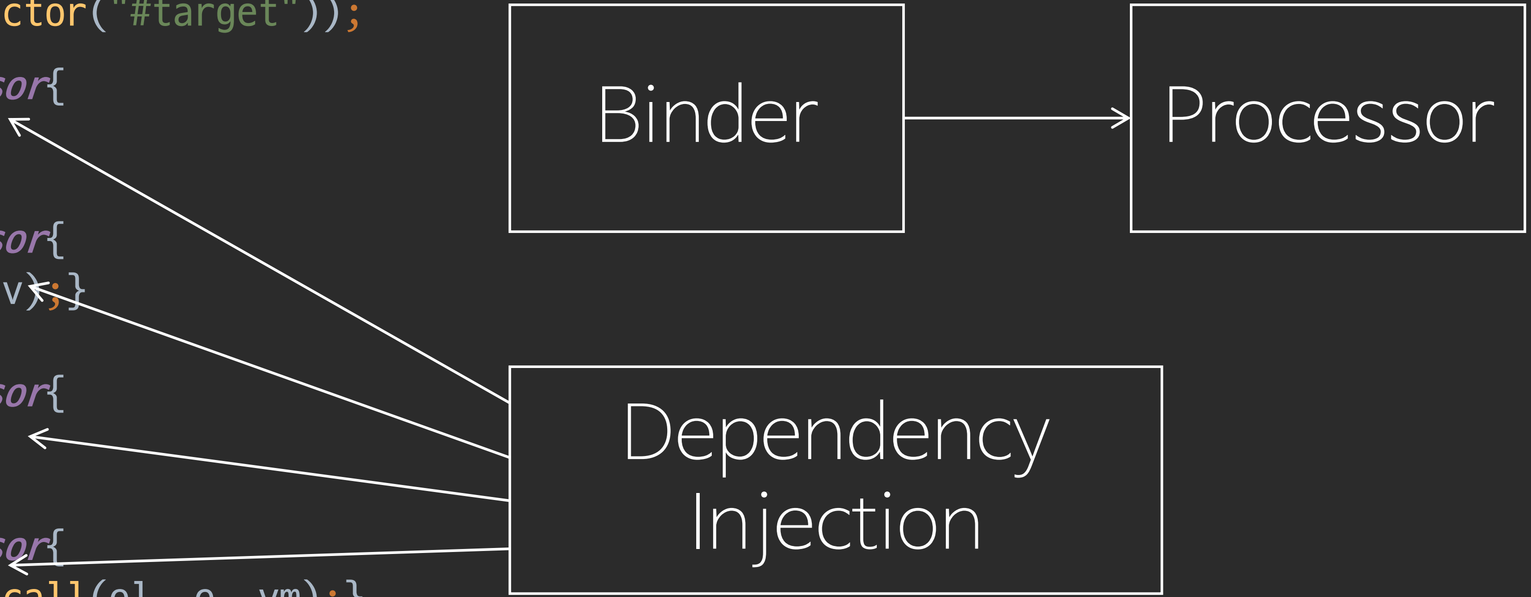
```
render(viewmodel, _ = type(viewmodel, ViewModel)){
  this.#items.forEach(item=>{
    const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
    Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
    Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
    Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
    Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
  });
}
```

```
const binder = scanner.scan(document.querySelector("#target"));
const binder = scanner.scan(document.querySelector("#target"));

binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el["on" + k] =e=>v.call(el, e, vm);}
})("events"));
```

```
const binder = scanner.scan(document.querySelector("#target"));

binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el["on" + k] =e=>v.call(el, e, vm);}
})("events"));
```

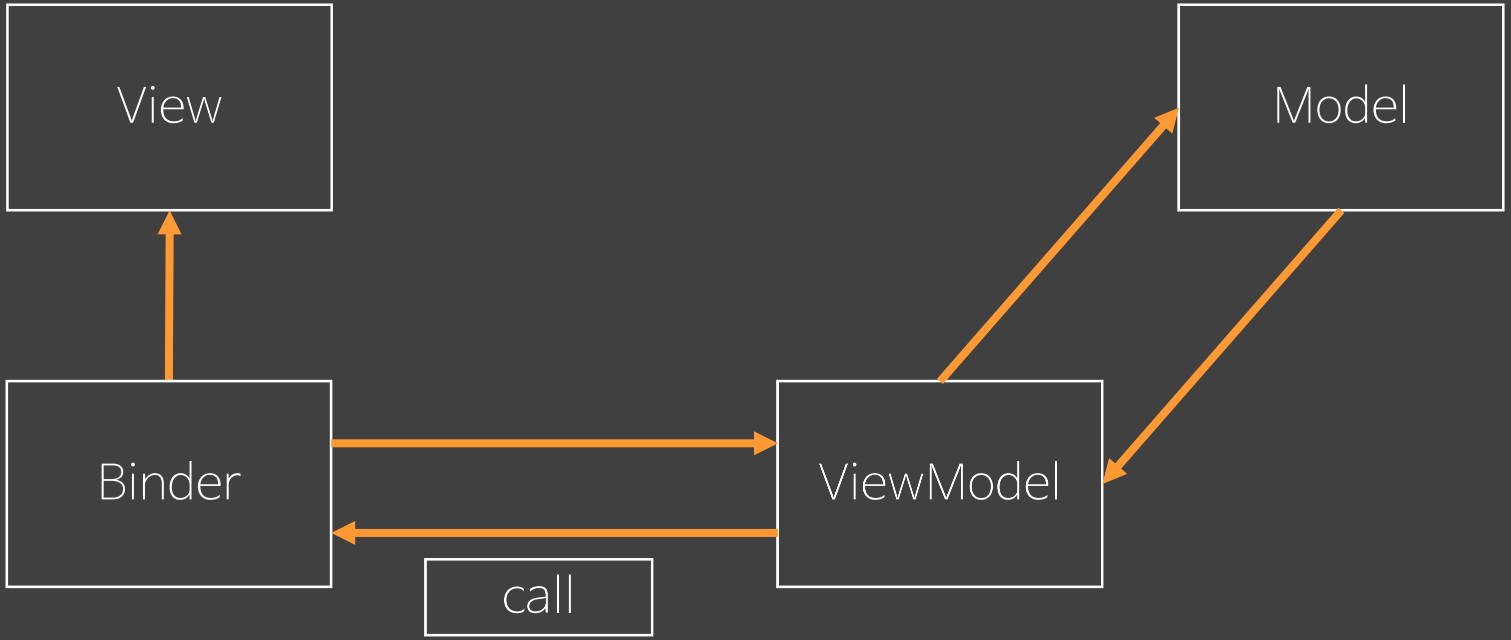


```
const binder = scanner.scan(document.querySelector("#target"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el["on" + k] =e=>v.call(el, e, vm);}
})("events"));
```

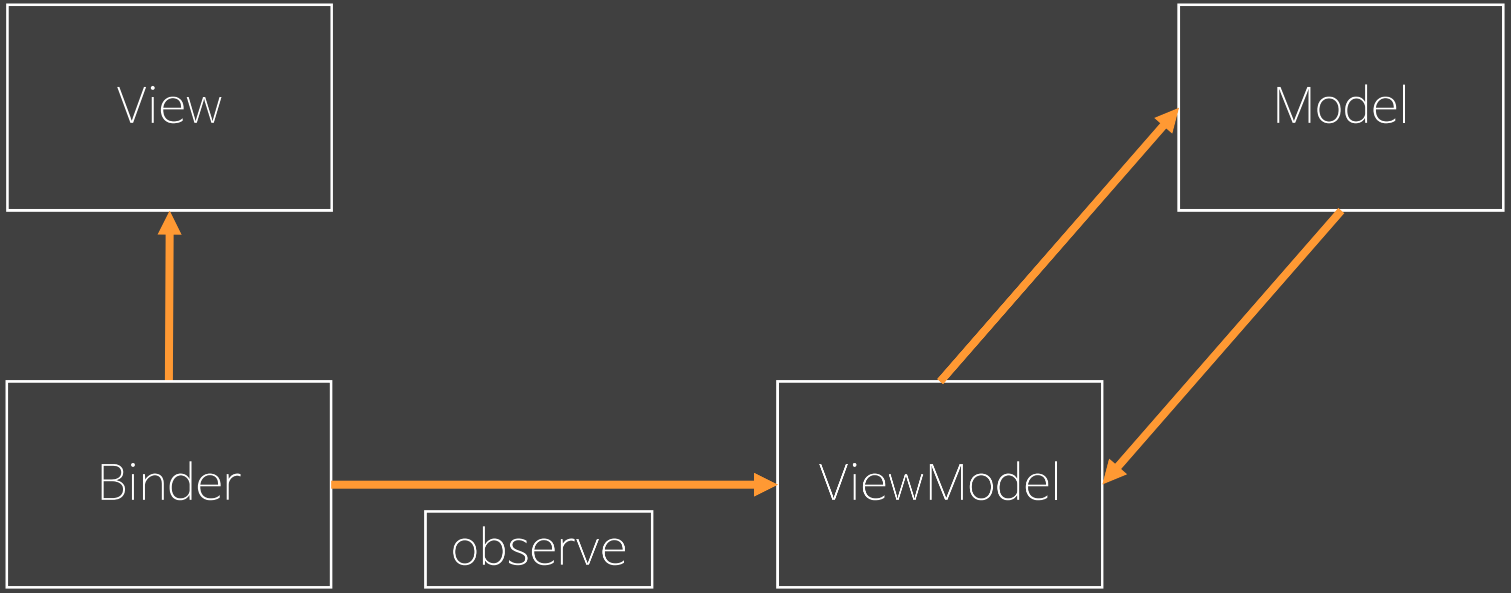


Observer

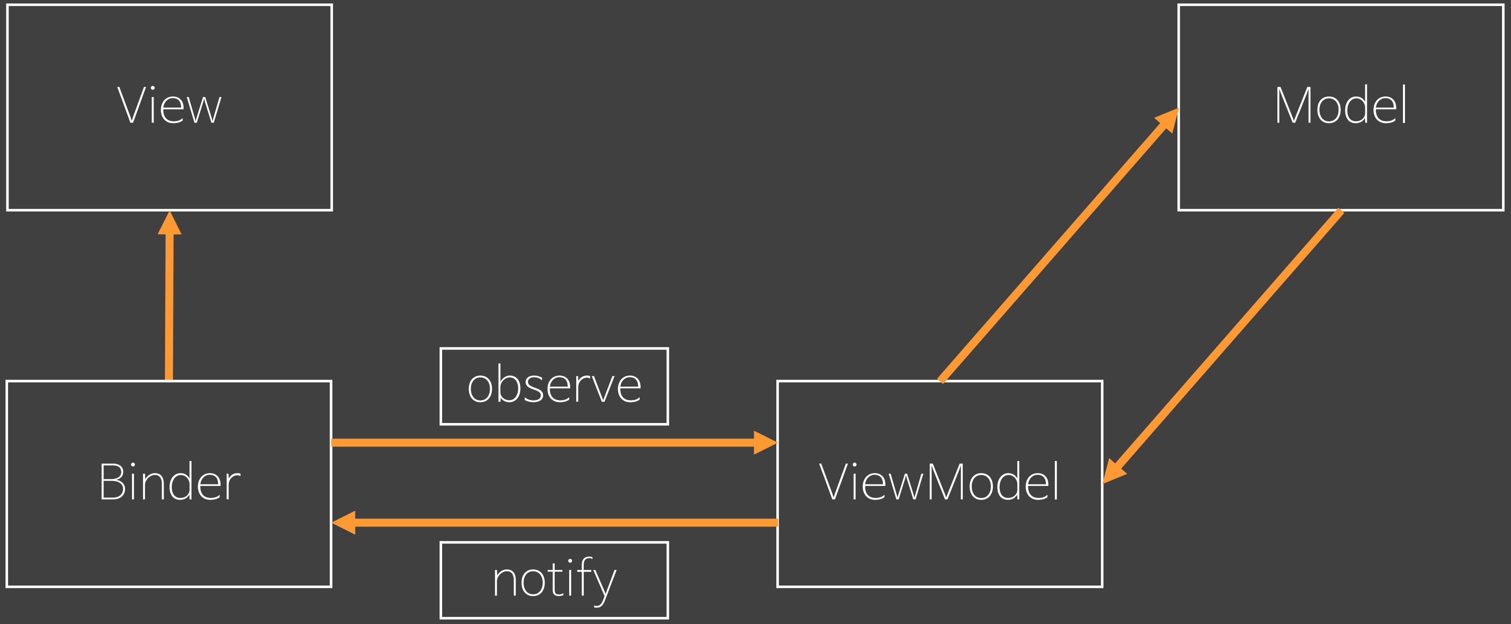
Model View ViewModel



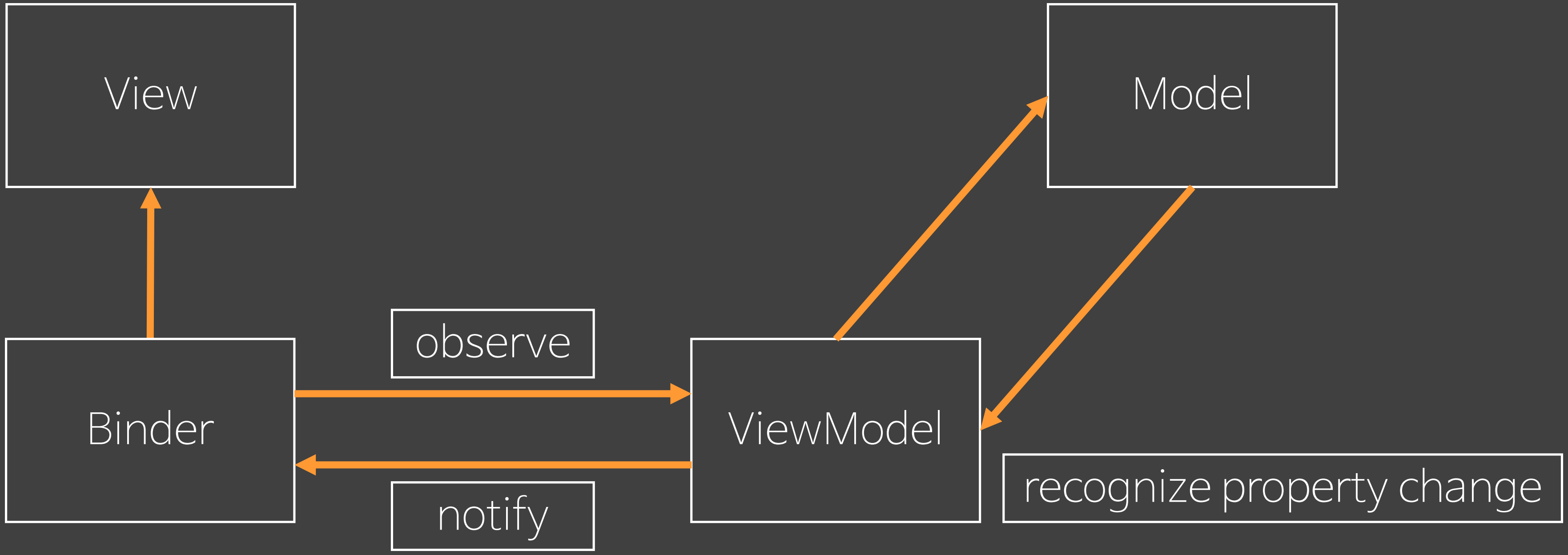
Model View ViewModel



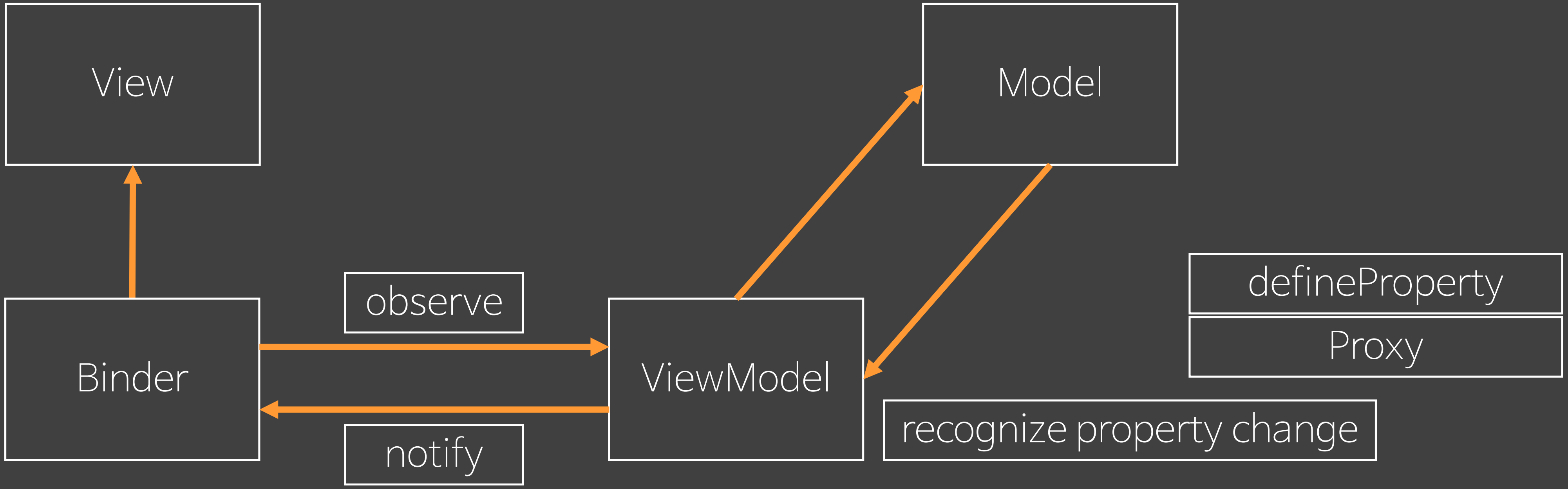
Model View ViewModel



Model View ViewModel



Model View ViewModel



```
const ViewModelListener = class{
    viewmodelUpdated(updated){throw "override";}
};
```

```js
const ViewModelListener = class{
    viewmodelUpdated(updated){throw "override";}
};
```

```js
const ViewModel = class{
  static get(data){return new ViewModel(data);}
  styles={}; attributes={}; properties={}; events={};
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
const ViewModelListener = class{
    viewmodelUpdated(updated){throw "override";}
};
```

```
const ViewModel = class{
  static get(data){return new ViewModel(data);}
  styles={}; attributes={}; properties={}; events={};
  #isUpdated = new Set; #listeners = new Set;
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
const ViewModelListener = class{
    viewmodelUpdated(updated){throw "override";}
};
```

```
const ViewModel = class{
  static get(data){return new ViewModel(data);}
  styles={}; attributes={}; properties={}; events={};
  #isUpdated = new Set; #listeners = new Set;
  addListener(v, _=type(v, ViewModelListener)){
    this.#listeners.add(v);
  }
  removeListener(v, _=type(v, ViewModelListener)){
    this.#listeners.delete(v);
  }
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
const ViewModelListener = class{
    viewmodelUpdated(updated){throw "override";}
};
```

```
const ViewModel = class{
  static get(data){return new ViewModel(data);}
  styles={}; attributes={}; properties={}; events={};
  #isUpdated = new Set; #listeners = new Set;
  addListener(v, _=type(v, ViewModelListener)){
    this.#listeners.add(v);
  }
  removeListener(v, _=type(v, ViewModelListener)){
    this.#listeners.delete(v);
  }
  notify(){
    this.#listeners.forEach(v=>v.viewmodelUpdated(this.#isUpdated));
  }
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, v])=>{
    if("styles,attributes,properties".includes(k)) {
      this[k] = ..
    }else{
      ..
    }
  });
```

```
const ViewModel = class{
  ..
  styles={}; attributes={}; properties={}; events={};
  #isUpdated = new Set; #listeners = new Set;
  ..
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, v])=>{
    if("styles,attributes,properties".includes(k)) {
      this[k] = Object.defineProperties(obj,
        Object.entries(obj).reduce((r, [k, v])=>{
          r[k] = {
            enumerable:true,
            get:_=>v,
            set:newV=>{
              v = newV;
              vm.#isUpdated.add(..);
            }
          };
          return r;
        }, {}));
    }else{
      ..
    }
  });
```

```
const ViewModel = class{
  ..
  styles={}; attributes={}; properties={}; events={};
  #isUpdated = new Set; #listeners = new Set;
  ..
  constructor(checker, data){
    Object.entries(data).forEach(([k, v])=>{
      switch(k){
      case"styles": this.styles = v; break;
      case"attributes": this.attributes = v; break;
      case"properties": this.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, obj])=>{
    if("styles,attributes,properties".includes(k)) {
      this[k] = Object.defineProperties({},
        Object.entries(obj).reduce((r, [k, v])=>{
          r[k] = {
            enumerable:true,
            get:_=>v,
            set:newV=>{
              v = newV;
              vm.#isUpdated.add(..);
            }
          };
          return r;
        }, {}));
    }else{
      ..
    }
  });
```

```
const ViewModel = class{
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
 properties={}; events={};
steners = new Set;

){
rEach(([k, v])=>{

yles = v; break;
s.attributes = v; break;
s.properties = v; break;
        case"events": this.events = v; break;
        default: this[k] = v;
        }
      });
      Object.seal(this);
    }
  };
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, obj])=>{
    if("styles,attributes,properties".includes(k)) {
      this[k] = Object.defineProperties({},
        Object.entries(obj).reduce((r, [k, v])=>{
          r[k] = {
            enumerable:true,
            get:_=>v,
            set:newV=>{
              v = newV;
              vm.#isUpdated.add(
                new ViewModelValue(cat, k, v)
              );
            }
          };
          return r;
        }, {}));
    }else{
      ..
    }
  });
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class{
                     properties={}; events={};
                    steners = new Set;

                    ){
                    rEach(([k, v])=>{

                    yles = v; break;
                    s.attributes = v; break;
                    s.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, v])=>{
    if("styles,attributes,properties".includes(k)) {
      ..
    }else{
      Object.defineProperty(this, k, {
        enumerable:true,
        get:_=>v,
        set:newV=>{
          v = newV;
          this.#isUpdated.add(new ViewModelValue("", k, v));
        }
      });
    }
  });
```

```
const ViewModel = class{
                        properties={}; events={};
                       steners = new Set;

                       ){
                       rEach(([k, v])=>{

                       yles = v; break;
                       s.attributes = v; break;
                       s.properties = v; break;
        case"events": this.events = v; break;
        default: this[k] = v;
        }
      });
      Object.seal(this);
    }
};
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
constructor(checker, data, _=type(data, "object")){
 super();
 Object.entries(data).forEach(([k, v])=>{
  if("styles,attributes,properties".includes(k)) {
   ..
  }else{
   Object.defineProperty(this, k, {
    enumerable:true,
    get:_=>v,
    set:newV=>{
     v = newV;
     this.#isUpdated.add(new ViewModelValue("", k, v));
    }
   });
  }
 });
```

```
const ViewModel = class{
              properties={}; events={};
             steners = new Set;

             ){
             rEach(([k, v])=>{

             yles = v; break;
             s.attributes = v; break;
             s.properties = v; break;
   case"events": this.events = v; break;
   default: this[k] = v;
   }
  });
  Object.seal(this);
 }
};
```

```
const ViewModelValue = class{
 cat; k; v;
 constructor(cat, k, v){
  this.cat = cat;
  this.k = k;
  this.v = v;
  Object.freeze(this);
 }
};
```

Composite

```
constructor(checker, data, _=type(data, "object")){
 super();
 Object.entries(data).forEach(([k, v])=>{
  if("styles,attributes,properties".includes(k)) {
   ..
  }else{
   Object.defineProperty(this, k, {
    enumerable:true,
    get:_=>v,
    set:newV=>{
     v = newV;
     this.#isUpdated.add(new ViewModelValue("", k, v));
    }
   });
  }
 });
```

```
const ViewModel = class{
              properties={}; events={};
             steners = new Set;

             ){
             rEach(([k, v])=>{

             yles = v; break;
             s.attributes = v; break;
             s.properties = v; break;
   case"events": this.events = v; break;
   default: this[k] = v;
   }
  });
  Object.seal(this);
 }
};
```

```
const ViewModelValue = class{
 cat; k; v;
 constructor(cat, k, v){
  this.cat = cat;
  this.k = k;
  this.v = v;
  Object.freeze(this);
 }
};
```

```
constructor(checker, data, _=type(data, "object")){
  super();
  Object.entries(data).forEach(([k, v])=>{
    if("styles,attributes,properties".includes(k)) {
      ..
    }else{
      Object.defineProperty(this, k, {
        enumerable:true,
        get:_=>v,
        set:newV=>{
          v = newV;
          this.#isUpdated.add(new ViewModelValue("", k, v));
        }
      });
      if(v instanceof ViewModel){
        ..
      }
    }
  });
```

```
const ViewModel = class{
                          properties={}; events={};
                         steners = new Set;

                         ){
                         rEach(([k, v])=>{

                         yles = v; break;
                         s.attributes = v; break;
                         s.properties = v; break;
        case"events": this.events = v; break;
        default: this[k] = v;
        }
      });
      Object.seal(this);
    }
};
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class extends ViewModelListener{
  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([k, v])=>{
      if("styles,attributes,properties".includes(k)){..
      }else{
        Object.defineProperty(this, k, {..});
        if(v instanceof ViewModel){
          v.parent = this;
          v.subKey = k;
          v.addListener(this);
        }
      }
    });
```

```
const ViewModel = class{
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
 properties={}; events={};
steners = new Set;

){
rEach(([k, v])=>{

yles = v; break;
s.attributes = v; break;
s.properties = v; break;
      case"events": this.events = v; break;
      default: this[k] = v;
      }
    });
    Object.seal(this);
  }
};
```

```
const ViewModel = class extends ViewModelListener{
  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([k, v])=>{
      if("styles,attributes,properties".includes(k)){..
      }else{
        Object.defineProperty(this, k, {..});
        if(v instanceof ViewModel){
          v.parent = this;
          v.subKey = k;
          v.addListener(this);
        }
      }
    });
  }
  viewmodelUpdated(updated){
    updated.forEach(v=>this.#isUpdated.add(v));
  }
```

```
const ViewModel = class{

                    properties={}; events={};
                   steners = new Set;

                   ){
                   rEach(([k, v])=>{

                   yles = v; break;
                   s.attributes = v; break;
                   s.properties = v; break;
        case"events": this.events = v; break;
        default: this[k] = v;
        }
      });
      Object.seal(this);
    }
  };
```

```
const ViewModelValue = class{
  cat; k; v;
  constructor(cat, k, v){
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class extends ViewModelListener{
  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([k, v])=>{
      if("styles,attributes,properties".includes(k)){..
      }else{
        Object.defineProperty(this, k, {..});
        if(v instanceof ViewModel){
          v.parent = this;
          v.subKey = k;
          v.addListener(this);
        }
      }
    });
  }
  viewmodelUpdated(updated){
    updated.forEach(v=>this.#isUpdated.add(v));
  }
```

```
const ViewModelValue = class{
  subKey; cat; k; v;
  constructor(subKey, cat, k, v){
    this.subKey = subKey;
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class{
  ..
                    properties={}; events={};
                    teners = new Set;

                    {
                    Each(([k, v])=>{

                    les = v; break;
                    .attributes = v; break;
                    .properties = v; break;
                    nts = v; break;
          default: this[k] = v;
          }
        });
        Object.seal(this);
      }
};
```

```
const ViewModel = class extends ViewModelListener{
  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([k, v])=>{
      if("styles,attributes,properties".includes(k)){..
      }else{
        Object.defineProperty(this, k, {
          enumerable:true,
          get:_=>v,
          set:newV=>{
            v = newV;
            this.#isUpdated.add(
              new ViewModelValue(this.subKey, "", k, v)
            );
          }
        });
        if(v instanceof ViewModel){
          v.parent = this;
          v.subKey = k;
          v.addListener(this);
        }
      }
    });
  }
  viewmodelUpdated(updated){
    updated.forEach(v=>this.#isUpdated.add(v));
```

```
const ViewModel = class{
  ..
                     properties={}; events={};
                     teners = new Set;

                     {
                     Each(([k, v])=>{

                     les = v; break;
                     .attributes = v; break;
                     .properties = v; break;
                     nts = v; break;
          default: this[k] = v;
          }
        });
        Object.seal(this);
      }
};
```

```
const ViewModelValue = class{
  subKey; cat; k; v;
  constructor(subKey, cat, k, v){
    this.subKey = subKey;
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class extends ViewModelListener{
  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([k, v])=>{
      if("styles,attributes,properties".includes(k)){..
      }else{
        Object.defineProperty(this, k, {..});
        if(v instanceof ViewModel){
          v.parent = this;
          v.subKey = k;
          v.addListener(this);
        }
      }
    });
    ViewModel.notify(this);
    Object.seal(this);
  }
  viewmodelUpdated(updated){
    updated.forEach(v=>this.#isUpdated.add(v));
  }
```

```
const ViewModel = class{
  ..
                 properties={}; events={};
                 teners = new Set;

                 {
                 Each(([k, v])=>{

                 les = v; break;
                 .attributes = v; break;
                 .properties = v; break;
                 nts = v; break;
          default: this[k] = v;
          }
        });
        Object.seal(this);
      }
};
```

```
const ViewModelValue = class{
  subKey; cat; k; v;
  constructor(subKey, cat, k, v){
    this.subKey = subKey;
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

```
const ViewModel = class extends ViewModelListener{
  static #subjects = new Set;
  static #inited = false;
  static notify(vm){
    this.#subjects.add(vm);
    if(this.#inited) return;
    this.#inited = true;
    const f =_=>{
      this.#subjects.forEach(vm=>{
        if(vm.#isUpdated.size){
          vm.notify();
          vm.#isUpdated.clear();
        }
      });
      requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
  }

  subKey = ""; parent = null;
  constructor(checker, data, _=type(data, "object")){
    ..
    ViewModel.notify(this);
    Object.seal(this);
  }
```

```
const ViewModel = class{
  ..
                       properties={}; events={};
                       teners = new Set;

                       {
                       Each(([k, v])=>{

                       les = v; break;
                       .attributes = v; break;
                       .properties = v; break;
                       nts = v; break;
          default: this[k] = v;
          }
      });
      Object.seal(this);
  }
};
```

```
const ViewModelValue = class{
  subKey; cat; k; v;
  constructor(subKey, cat, k, v){
    this.subKey = subKey;
    this.cat = cat;
    this.k = k;
    this.v = v;
    Object.freeze(this);
  }
};
```

Observer

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){
    this.#processors[v.cat] = v;
  }
  render(viewmodel, _ = type(viewmodel, ViewModel)){
    const processores = Object.entries(this.#processors);
    this.#items.forEach(item=>{
      const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
      processores.forEach(([pk, processor])=>{
        Object.entries(vm[pk]).forEach(([k, v])=>{
          processor.process(vm, el, k, v)
        });
      });
    });
  }
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){..}
  render(viewmodel, _ = type(viewmodel, ViewModel)){..}
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){..}
  render(viewmodel, _ = type(viewmodel, ViewModel)){..}
  watch(viewmodel, _ = type(viewmodel, ViewModel)){
    viewmodel.addListener(this);
    this.render(viewmodel);
  }
  unwatch(viewmodel, _ = type(viewmodel, ViewModel)){
    viewmodel.removeListener(this);
  }
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  viewmodelUpdated(updated){
    const items = {};
    this.#items.forEach(item=>{
      items[item.viewmodel] = [
        type(viewmodel[item.viewmodel], ViewModel),
        item.el
      ];
    });
    updated.forEach(v=>{
      if(!items[v.subKey]) return;
      const [vm, el] = items[v.subKey], processor = this.#processors[v.cat];
      if(!el || !processor) return;
      processor.process(vm, el, v.k, v.v);
    });
  }
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){..}
  render(viewmodel, _ = type(viewmodel, ViewModel)){..}
  watch(viewmodel, _ = type(viewmodel, ViewModel)){..}
  unwatch(viewmodel, _ = type(viewmodel, ViewModel)){..}
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  viewmodelUpdated(updated){
    const items = {};
    this.#items.forEach(item=>{
      items[item.viewmodel] = [
        type(viewmodel[item.viewmodel], ViewModel),
        item.el
      ];
    });
    updated.forEach(v=>{
      if(!items[v.subKey]) return;
      const [vm, el] = items[v.subKey], processor = this.#processors[v.cat];
      if(!el || !processor) return;
      processor.process(vm, el, v.k, v.v);
    });
  }
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){..}
  render(viewmodel, _ = type(viewmodel, ViewModel)){..}
  watch(viewmodel, _ = type(viewmodel, ViewModel)){..}
  unwatch(viewmodel, _ = type(viewmodel, ViewModel)){..}
};
```

```
const Binder = class extends ViewModelListener{
  #items = new Set; #processors = {};
  viewmodelUpdated(updated){
    const items = {};
    this.#items.forEach(item=>{
      items[item.viewmodel] = [
        type(viewmodel[item.viewmodel], ViewModel),
        item.el
      ];
    });
    updated.forEach(v=>{
      if(!items[v.subKey]) return;
      const [vm, el] = items[v.subKey], processor = this.#processors[v.cat];
      if(!el || !processor) return;
      processor.process(vm, el, v.k, v.v);
    });
  }
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  addProcessor(v, _0=type(v, Processor)){..}
  render(viewmodel, _ = type(viewmodel, ViewModel)){..}
  watch(viewmodel, _ = type(viewmodel, ViewModel)){..}
  unwatch(viewmodel, _ = type(viewmodel, ViewModel)){..}
};
```

client

```
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){
        console.log("event", k, v, el)
        el["on" + k] =e=>v.call(el, e, vm);
    }
})("events"));
```

```javascript
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){
        console.log("event", k, v, el)
        el["on" + k] =e=>v.call(el, e, vm);
    }
})("events"));
```

```javascript
const viewmodel = ViewModel.get({
    isStop:false,
    changeContents(){
        this.wrapper.styles.background = `rgb(${..})`;
        this.contents.properties.innerHTML = Math…;
    },
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        },
        events:{
            click(e, vm){
                vm.parent.isStop = true;
                console.log("click", vm)
            }
        }
    }),
    title:..,
    contents:..
});
```

```javascript
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){
        console.log("event", k, v, el)
        el["on" + k] =e=>v.call(el, e, vm);
    }
})("events"));
```

```javascript
const viewmodel = ViewModel.get({
    isStop:false,
    changeContents(){
        this.wrapper.styles.background = `rgb(${..})`;
        this.contents.properties.innerHTML = Math…;
    },
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        },
        events:{
            click(e, vm){
                vm.parent.isStop = true;
                console.log("click", vm)
            }
        }
    }),
    title:..,
    contents:..
});
```

```
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes"));
binder.addProcessor(new (class extends Processor{
    _process(vm, el, k, v){el[k] = v;}
})("properties"));
binder.addProcessor(new (class e
    _process(vm, el, k, v){
        console.log("event", k,
        el["on" + k] =e=>v.call(
    }
})("events"));
```

```
const viewmodel = ViewModel.get({
    isStop:false,
    changeContents(){
        this.wrapper.styles.background = `rgb(${..})`;
        this.contents.properties.innerHTML = Math…;
    },
    wrapper:ViewModel.get({
        styles:{
            width:"50%"
                                              ",



                                          Stop = true;
                                          "click", vm)
            }
        }
    }),
    title:..,
    contents:..
});
```

```
binder.watch(viewmodel);
const f =_=>{
    viewmodel.changeContents();
  if(!viewmodel.isStop) requestAnimationFrame(f);
};
requestAnimationFrame(f);
```