OBJECT ORIENTED JAVASCRIPT

1 2 3 4 5

ISP

ViewModel

```
const ViewModel = class extends ViewModelListener{
    static get(data){return new ViewModel(data);}
    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);
    }
    styles = {}; attributes = {}; properties = {}; events = {};
    subKey = ""; parent = null;
    #isUpdated = new Set; #listeners = new Set;
    constructor(data, _=type(data, "object")){
        super();
        Object.entries(data).forEach(([cat, obj])=>{
            if("styles,attributes,properties".includes(cat)) {
                if (!obj || typeof obj != "object") throw `invalid object cat:${cat}, obj:${obj}`;
                this[cat] = Object.defineProperties({}, Object.entries(obj).reduce((r, [k, v])=>{
                    r[k] = {
                        enumerable:true,
                        get:_=>v,
                        set:newV=>{
                            v = newV;
                            this.add(new ViewModelValue(this.#subKey, cat, k, v));
                        }
                    };
                    return r;
                }, {}));
            }else{
                Object.defineProperties(this, {
                    [cat]: {
                        enumerable: true,
                        get:_=>obj,
                        set:newV=>{
                            obj = newV;
                            this.add(new ViewModelValue(this.#subKey, "root", cat, obj));
                        }
                    }
                });
                if(obj instanceof ViewModel){
                    obj.parent = this;
                    obj.subKey = cat;
                    obj.addListener(this);
                }
            }
        });
        ViewModel.notify(this);
        Object.seal(this);
    }
    viewmodelUpdated(updated){
        updated.forEach(v=>this.#isUpdated.add(v));
    }
    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));
    }
};
```

. . . .

```
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));
}
```

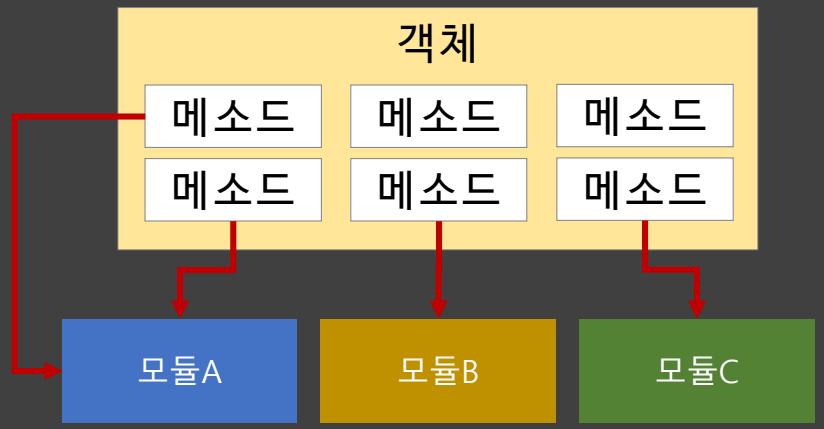
```
#isUpdated = new Set; #listeners = new Set;

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));
}
```

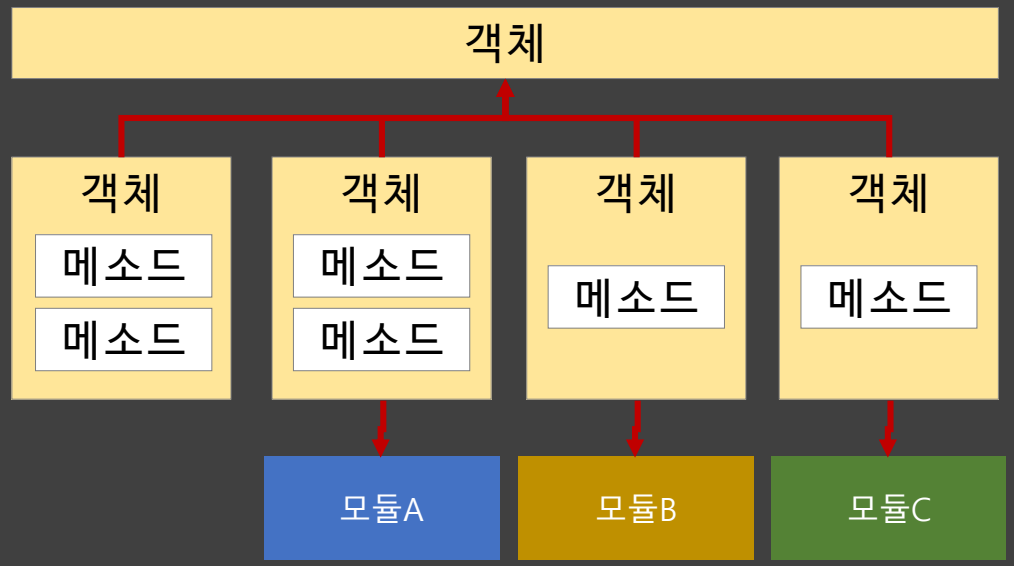
```
#isUpdated = new Set; #listeners = new Set;

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));
}
```

```
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);
}
```

SOLID원칙

SRP Single Responsibility 단일책임
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리





```
const ViewModelSubject = class extends ViewModelListener{
    #info = new Set; #listeners = new Set;
    add(v, _=type(v, ViewModelValue)){this.#info.add(v);}
    clear(){this.#info.clear();}
    addListener(v, _=type(v, ViewModelListener)){
        this.#listeners.add(v);
        ViewModelSubject.watch(this);
    }
    removeListener(v, _=type(v, ViewModelListener)){
        this.#listeners.delete(v);
        if(!this.#listeners.size) ViewModelSubject.unwatch(this);
    }
    notify(){this.#listeners.forEach(v=>v.viewmodelUpdated(this.#info));}
};
```

```
#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)
  );
}
```

```
const ViewModelSubject = class extends ViewModelListener{
    #info = new Set; #listeners = new Set;
    add(v, _=type(v, ViewModelValue)){this.#info.add(v);}
    clear(){this.#info.clear();}
    addListener(v, _=type(v, ViewModelListener)){
        this.#listeners.add(v);
        ViewModelSubject.watch(this);
    }
    removeListener(v, _=type(v, ViewModelListener)){
        this.#listeners.delete(v);
        if(!this.#listeners.size) ViewModelSubject.unwatch(this);
    }
    notify(){this.#listeners.forEach(v=>v.viewmodelUpdated(this.#info));}
};
```

```
#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)
  );
}
```

```
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);
}
```

```
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);
}
```

```
static #subjects = new Set; static #inited = false;
static notify(){
    const f =_=>{
        this.#subjects.forEach(v=>{
            if(v.#info.size){
                v.notify();
                v.clear();
            }
        });
        if(this.#inited) requestAnimationFrame(f);
    };
    requestAnimationFrame(f);
}
```

```
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);
}
```

```
static #subjects = new Set; static #inited = false;
static notify(){…}
static watch(vm, _=type(vm, ViewModelListener)){
    this.#subjects.add(vm);
    if(!this.#inited){
        this.#inited = true;
        this.notify();
    }
}
```

```
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);
}
```

```
static #subjects = new Set; static #inited = false;
static notify(){…}
static watch(vm, _=type(vm, ViewModelListener)){
    this.#subjects.add(vm);
    if(!this.#inited){
        this.#inited = true;
        this.notify();
    }
}
static unwatch(vm, _=type(vm, ViewModelListener)){
    this.#subjects.delete(vm);
    if(!this.#subjects.size) this.#inited = false;
}
```

섬세한 권한 조정

```
const ViewModel = class extends ViewModelListener{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    subKey = ""; parent = null;
```

```
const ViewModel = class extends ViewModelListener{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    subKey = ""; parent = null;
```

```
const ViewModel = class extends ViewModelSubject{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    #subKey = "";
    get subKey(){return this.#subKey;}
    #parent = null;
    get parent(){return this.#parent;}
    setParent(parent, subKey){
        this.#parent = type(parent, ViewModel);
        this.#subKey = subKey;
        this.addListener(parent);
    }
```

```
const ViewModel = class extends ViewModelListener{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    subKey = ""; parent = null;
```

```
const ViewModel = class extends ViewModelSubject{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    #subKey = "";
    get subKey(){return this.#subKey;}
    #parent = null;
    get parent(){return this.#parent;}
    setParent(parent, subKey){
        this.#parent = type(parent, ViewModel);
        this.#subKey = subKey;
        this.addListener(parent);
    }
```

```
constructor(data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([cat, obj])=>{
        if("styles,attributes,properties".includes(cat)) {
            if (!obj || typeof obj != "object") throw `invalid object cat:${cat}, obj:${obj}`;
            this[cat] = Object.defineProperties({}, Object.entries(obj).reduce((r, [k, v])=>{
                r[k] = {
                    enumerable:true,
                    get:_=>v,
                    set:newV=>{
                        v = newV;
                        this.add(new ViewModelValue(this.#subKey, cat, k, v));
                    }
                };
                return r;
            }, {}));
        }else{…}
    });
    Object.seal(this);
}
viewmodelUpdated(updated){updated.forEach(v=>this.add(v));}
```

```
constructor(data, _=type(data, "object")){
    super();
    Object.entries(data).forEach(([cat, obj])=>{
        if("styles,attributes,properties".includes(cat)) {…
        }else{
            Object.defineProperties(this, {
                [cat]: {
                    enumerable: true,
                    get: _ => obj,
                    set: newV => {
                        obj = newV;
                        this.add(new ViewModelValue(this.#subKey, "root", cat, obj));
                    }
                }
            });
            if(obj instanceof ViewModel) obj.setParent(this, cat);
        }
    });
    Object.seal(this);
}
viewmodelUpdated(updated){updated.forEach(v=>this.add(v));}
```

Visitor

```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```

```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```

```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```

```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};
```

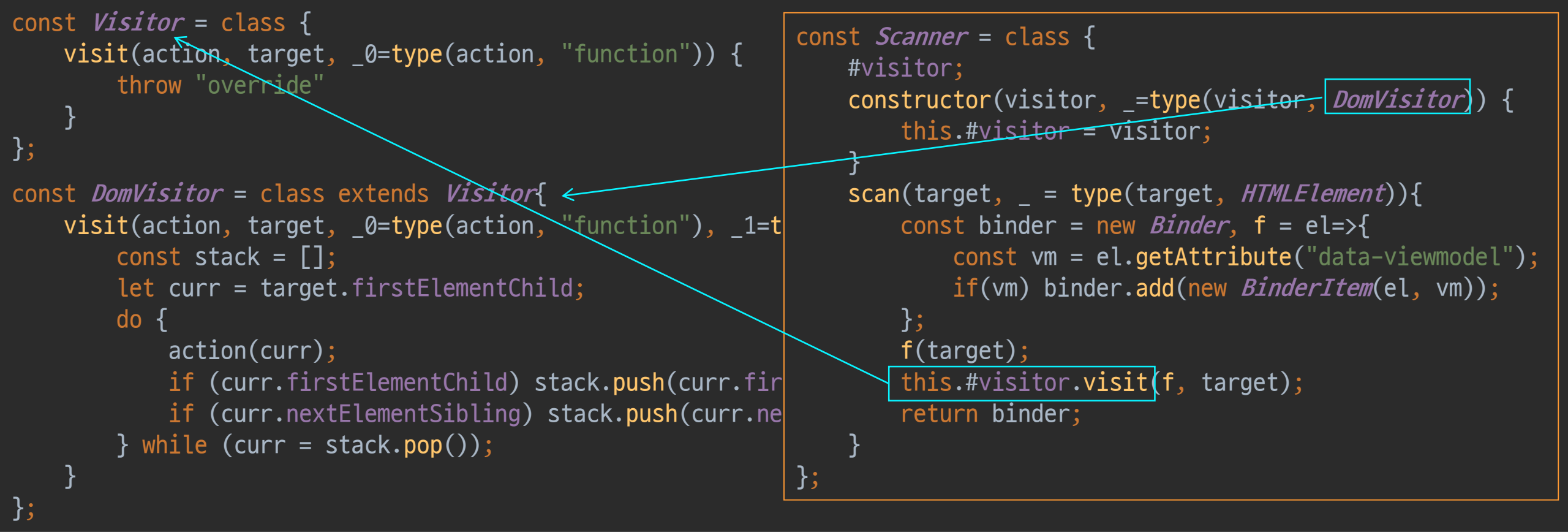
```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};

const DomVisitor = class extends Visitor{
    visit(action, target, _0=type(action, "function"), _1=type(target, HTMLElement)) {
        const stack = [];
        let curr = target.firstElementChild;
        do {
            action(curr);
            if (curr.firstElementChild) stack.push(curr.firstElementChild);
            if (curr.nextElementSibling) stack.push(curr.nextElementSibling);
        } while (curr = stack.pop());
    }
};
```

```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};

const DomVisitor = class extends Visitor{
    visit(action, target, _0=type(action, "function"), _1=type(target, HTMLElement)) {
        const stack = [];
        let curr = target.firstElementChild;
        do {
            action(curr);
            if (curr.firstElementChild) stack.push(curr.firstElementChild);
            if (curr.nextElementSibling) stack.push(curr.nextElementSibling);
        } while (curr = stack.pop());
    }
};
```

```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};
const DomVisitor = class extends Visitor{
    visit(action, target, _0=type(action, "function"), _1=t
        const stack = [];
        let curr = target.firstElementChild;
        do {
            action(curr);
            if (curr.firstElementChild) stack.push(curr.fir
            if (curr.nextElementSibling) stack.push(curr.ne
        } while (curr = stack.pop());
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _=type(visitor, DomVisitor)) {
        this.#visitor = visitor;
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.#visitor.visit(f, target);
        return binder;
    }
};
```

추상계층 불일치


const Visitor = class {
visit(action, target, _0=type(action, "function")) {
throw "override"
}
};
const DomVisitor = class extends Visitor{
visit(action, target, _0=type(action, "function"), _1=t
const stack = [];
let curr = target.firstElementChild;
do {
action(curr);
if (curr.firstElementChild) stack.push(curr.fir
if (curr.nextElementSibling) stack.push(curr.ne
} while (curr = stack.pop());
}
};
const Scanner = class {
#visitor;
constructor(visitor, _=type(visitor, DomVisitor)) {
this.#visitor = visitor;
}
scan(target, _ = type(target, HTMLElement)){
const binder = new Binder, f = el=>{
const vm = el.getAttribute("data-viewmodel");
if(vm) binder.add(new BinderItem(el, vm));
};
f(target);
this.#visitor.visit(f, target);
return binder;
}
};

```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};
const DomVisitor = class extends Visitor{
    visit(action, target, _0=type(action, "function"), _1=t
        const stack = [];
        let curr = target.firstElementChild;
        do {
            action(curr);
            if (curr.firstElementChild) stack.push(curr.fir
            if (curr.nextElementSibling) stack.push(curr.ne
        } while (curr = stack.pop());
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _=type(visitor, DomVisitor)) {
        this.#visitor = visitor;
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.#visitor.visit(f, target);
        return binder;
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _ = type(visitor, Visitor)) {
        this.#visitor = visitor;
    }
    visit(f, target){this.#visitor.visit(f, target);}
    scan(target){throw "override"}
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _=type(visitor, DomVisitor)) {
        this.#visitor = visitor;
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.#visitor.visit(f, target);
        return binder;
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _ = type(visitor, Visitor)) {
        this.#visitor = visitor;
    }
    visit(f, target){this.#visitor.visit(f, target);}
    scan(target){throw "override"}
};
const DomScanner = class extends Scanner{
    constructor(visitor, _=type(visitor, DomVisitor)) {
        super(visitor);
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.visit(f, target);
        return binder;
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _=type(visitor, DomVisitor)) {
        this.#visitor = visitor;
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.#visitor.visit(f, target);
        return binder;
    }
};
```

```
const Scanner = class {
    #visitor;
    constructor(visitor, _ = type(visitor, Visitor)) {
        this.#visitor = visitor;
    }
    visit(f, target){this.#visitor.visit(f, target);}
    scan(target){throw "override"}
};
```

```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};
```

```
const DomScanner = class extends Scanner{
    constructor(visitor, _=type(visitor, DomVisitor)) {
        super(visitor);
    }
    scan(target, _ = type(target, HTMLElement)){
        const binder = new Binder, f = el=>{
            const vm = el.getAttribute("data-viewmodel");
            if(vm) binder.add(new BinderItem(el, vm));
        };
        f(target);
        this.visit(f, target);
        return binder;
    }
};
```

```
const DomVisitor = class extends Visitor{
    visit(action, target, _0=type(action, "function"), _1=
        const stack = [];
        let curr = target.firstElementChild;
        do {
            action(curr);
            if (curr.firstElementChild) stack.push(curr.fi
            if (curr.nextElementSibling) stack.push(curr.n
        } while (curr = stack.pop());
    }
};
```

```
const scanner = new Scanner;
```

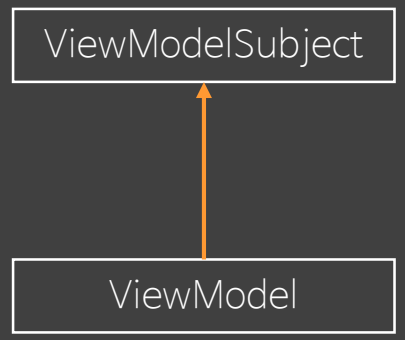
```
const scanner = new Scanner;
```

```
const scanner = new DomScanner(new DomVisitor);
```

설계 종합

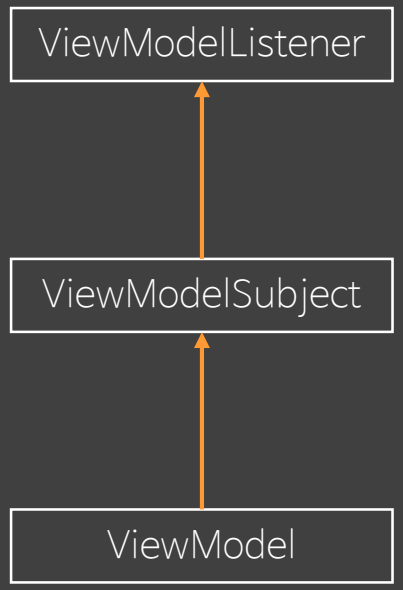
```
const ViewModel = class extends ViewModelSubject{
    static get(data){return new ViewModel(data);}
    styles = {}; attributes = {}; properties = {}; events = {};
    #subKey = "";
    get subKey(){return this.#subKey;}
    #parent = null;
    get parent(){return this.#parent;}
    setParent(parent, subKey){…}
    constructor(data, _=type(data, "object")){…}
    viewmodelUpdated(updated){updated.forEach(v=>this.add(v));}
};
```

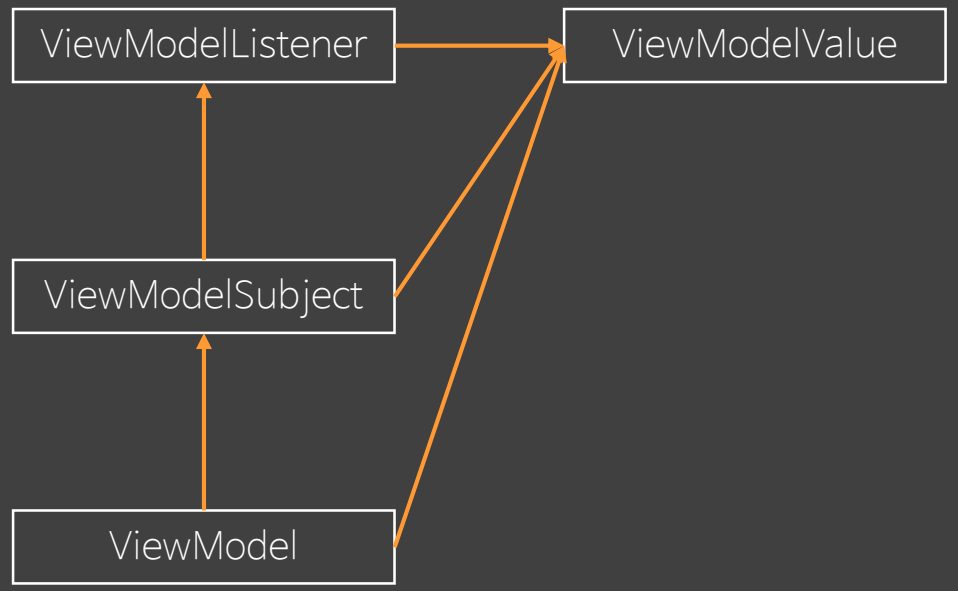
ViewModel



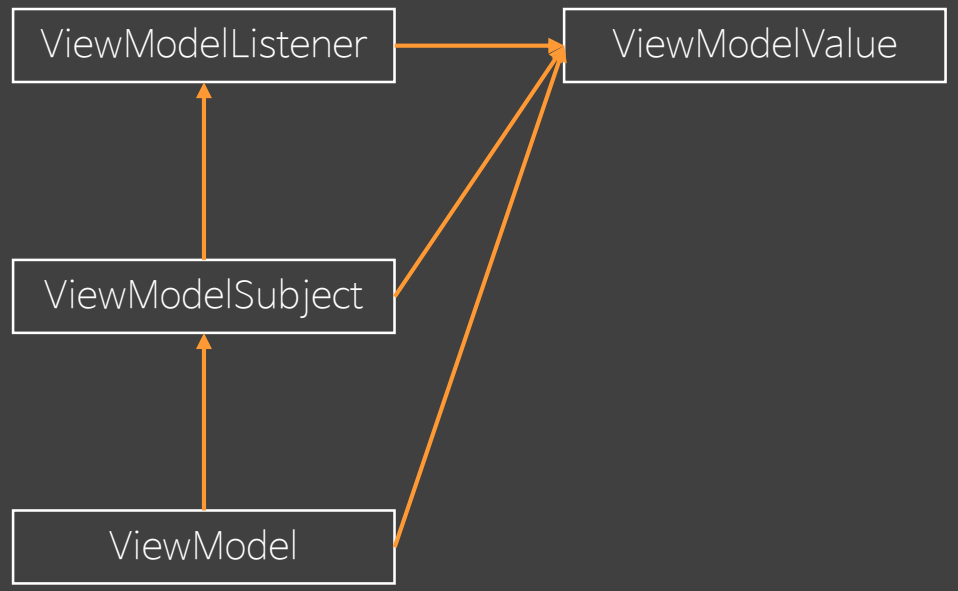

```
const ViewModelSubject = class extends ViewModelListener{
    static #subjects = new Set;
    static #inited = false;
    static notify(){…}
    static watch(vm, _=type(vm, ViewModelListener)){…}
    static unwatch(vm, _=type(vm, ViewModelListener)){…}
    #info = new Set; #listeners = new Set;
    add(v, _=type(v, ViewModelValue)){this.#info.add(v);}
    clear(){this.#info.clear();}
    addListener(v, _=type(v, ViewModelListener)){…}
    removeListener(v, _=type(v, ViewModelListener)){…}
    notify(){…}
};
```

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





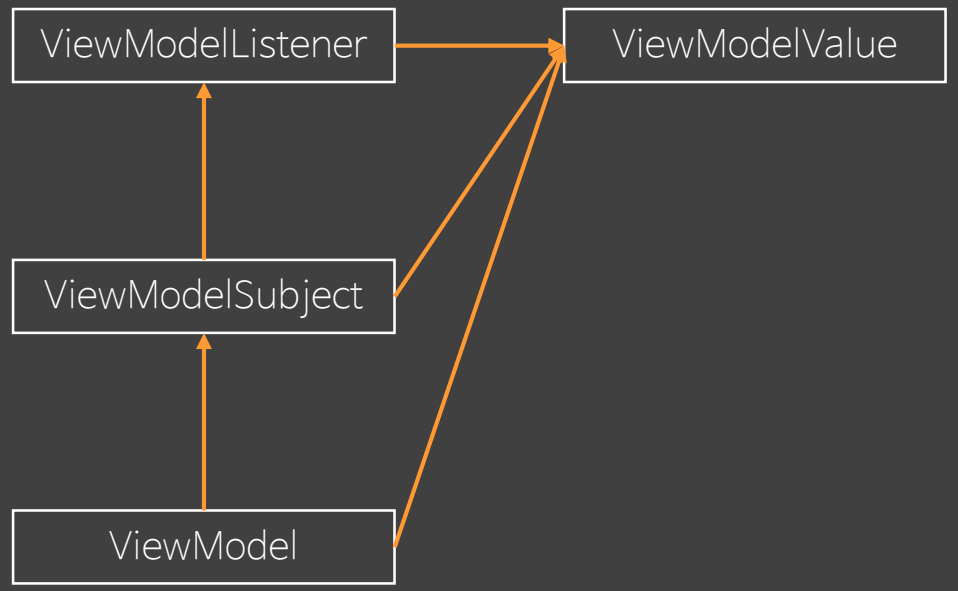

```
const ViewModelValue = class{
    subKey; cat; k; v;
    constructor(subKey, cat, k, v){
        this.subKey = subKey;
        this.cat = cat;
        this.k = k;
        this.v = v;
        Object.freeze(this);
    }
};
```

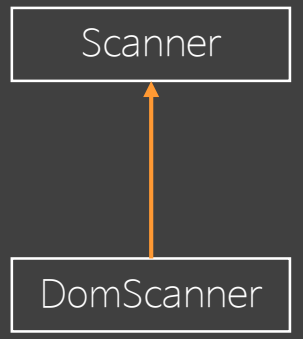



```
const Scanner = class {
    #visitor;
    constructor(visitor, _ = type(visitor, Visitor)){…}
    visit(f, target){…}
    scan(target) {…}
};
```

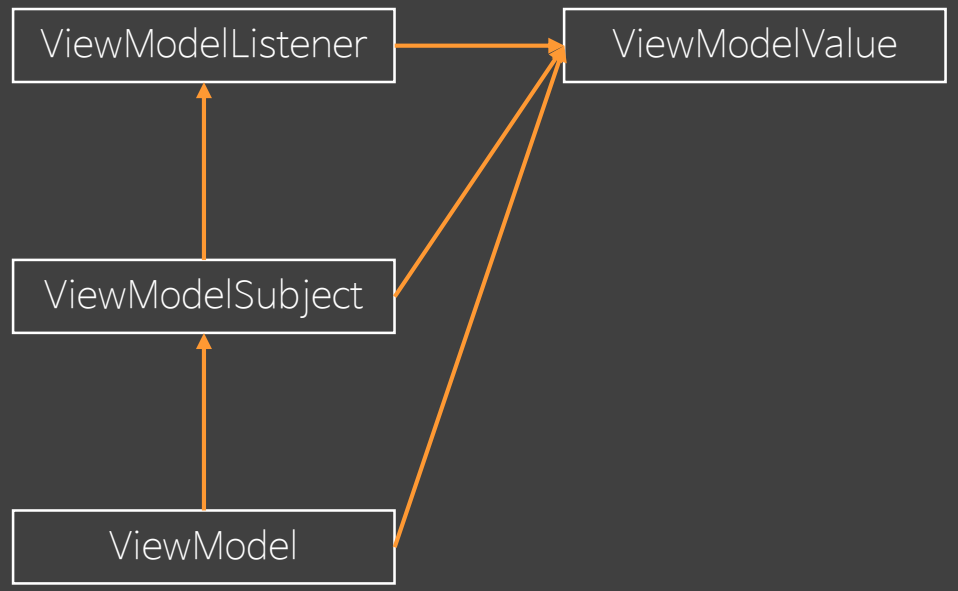
Scanner

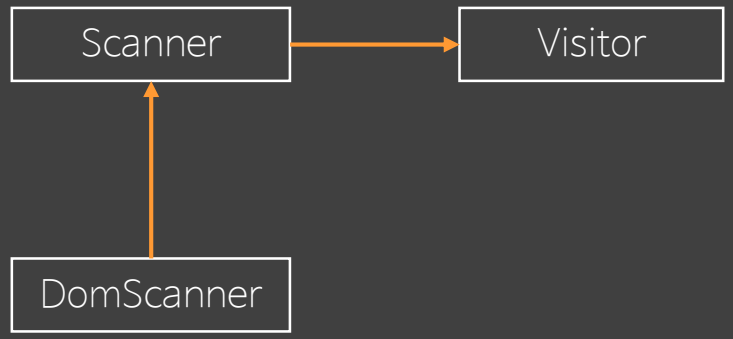
```
const DomVisitor = class extends Visitor{
    visit(action, target,
      _0=type(action, "function"),
      _1=type(target, HTMLElement)) {…}
};
```



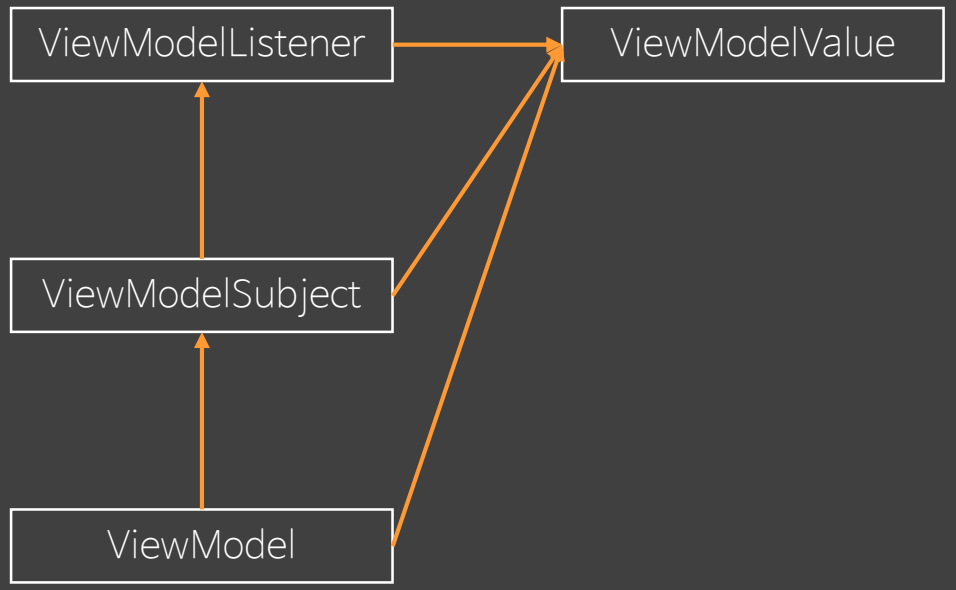


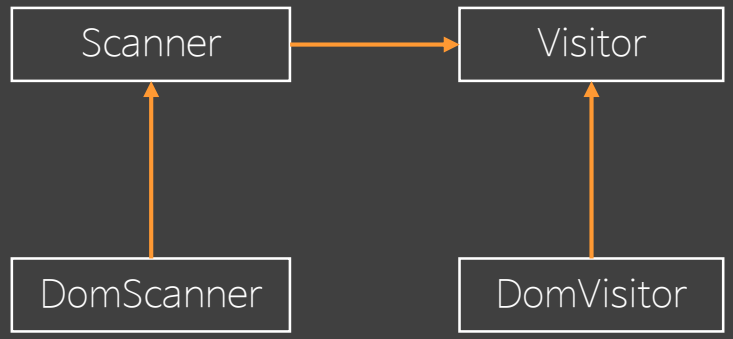
```
const Visitor = class {
    visit(action, target, _0=type(action, "function")) {
        throw "override"
    }
};
```

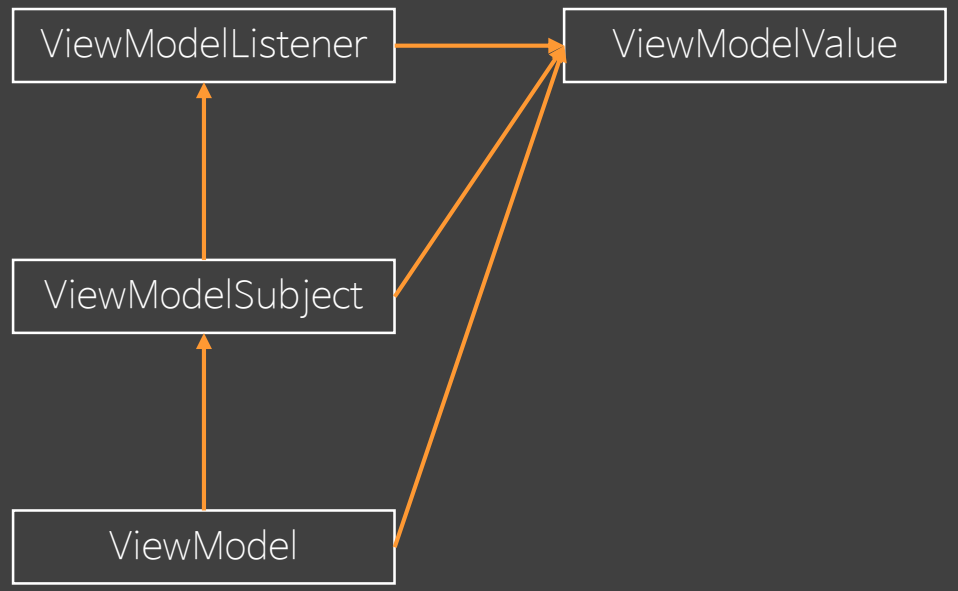




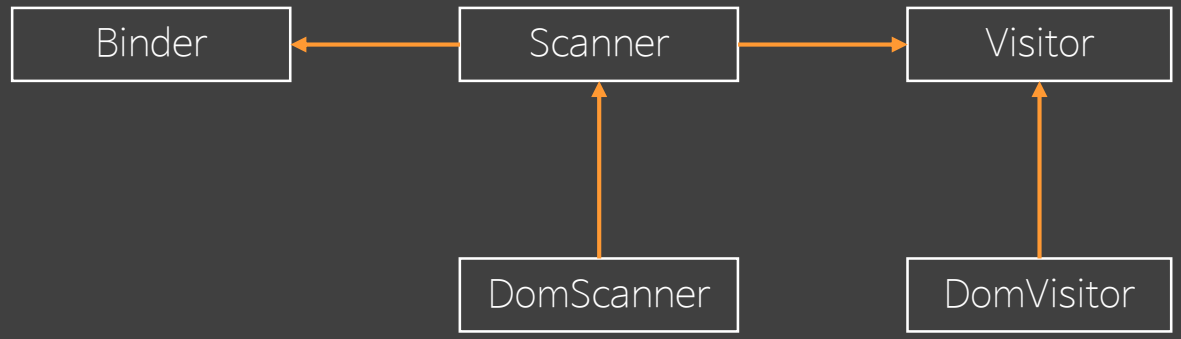
```
const DomVisitor = class extends Visitor{
    visit(action, target,
      _0=type(action, "function"),
      _1=type(target, HTMLElement)) {…}
};
```

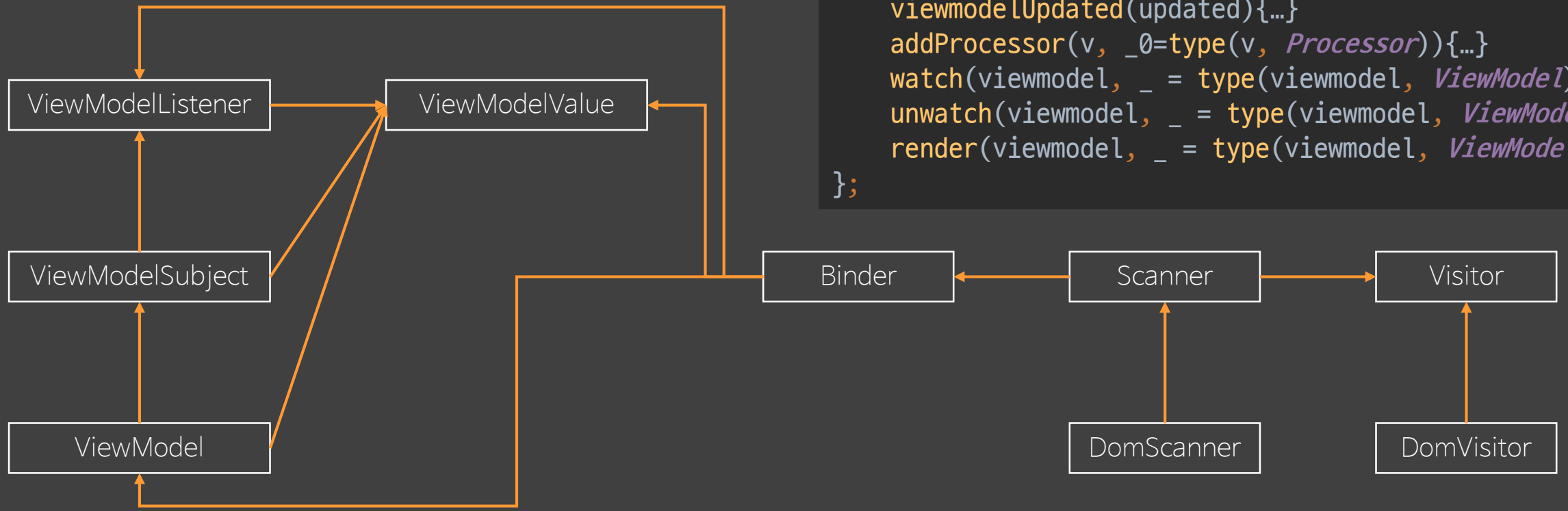






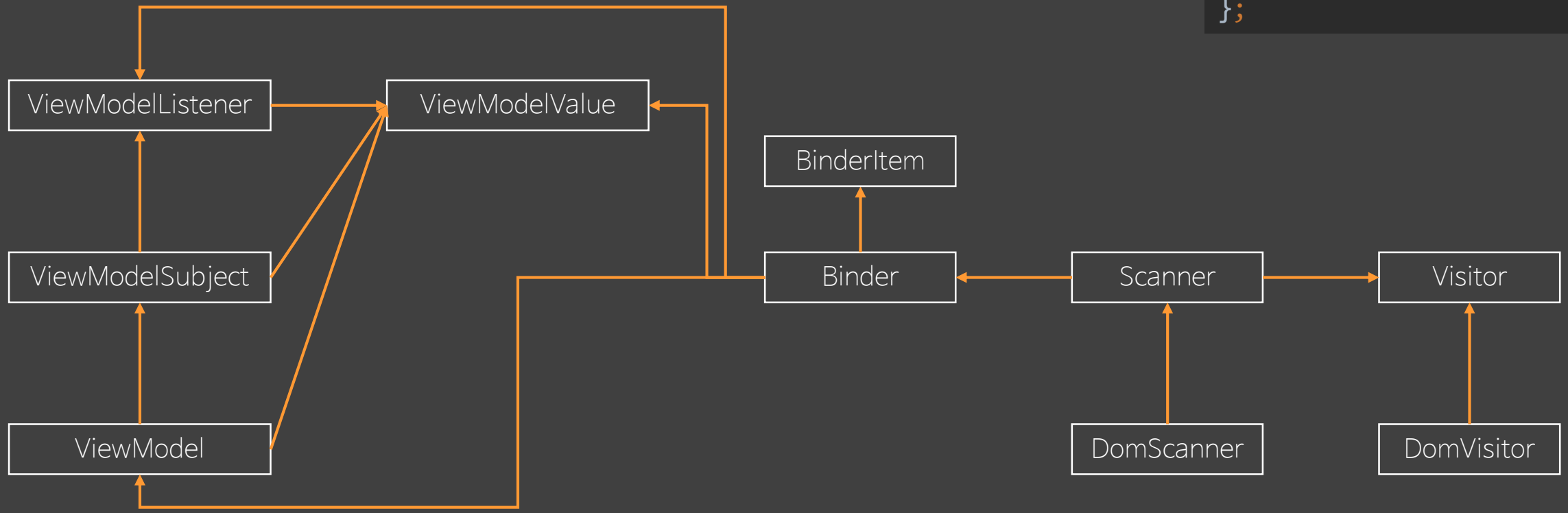

```
const Binder = class extends ViewModelListener{
    #items = new Set; #processors = {};
    add(v, _ = type(v, BinderItem)){this.#items.add(v);}
    viewmodelUpdated(updated){…}
    addProcessor(v, _0=type(v, Processor)){…}
    watch(viewmodel, _ = type(viewmodel, ViewModel)){…}
    unwatch(viewmodel,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



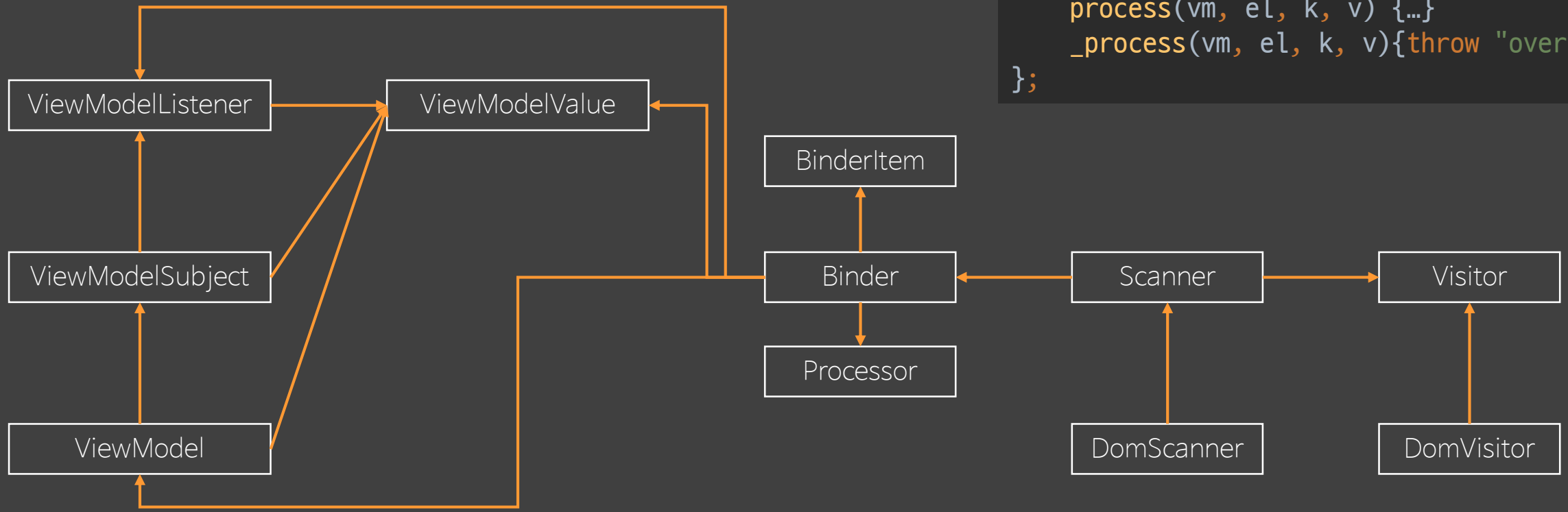



```
const Binder = class extends ViewModelListener{
    #items = new Set; #processors = {};
    add(v, _ = type(v, BinderItem)){this.#items.add(v);}
    viewmodelUpdated(updated){…}
    addProcessor(v, _0=type(v, Processor)){…}
    watch(viewmodel, _ = type(viewmodel, ViewModel)){…}
    unwatch(viewmodel,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

```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel){…}
};
```

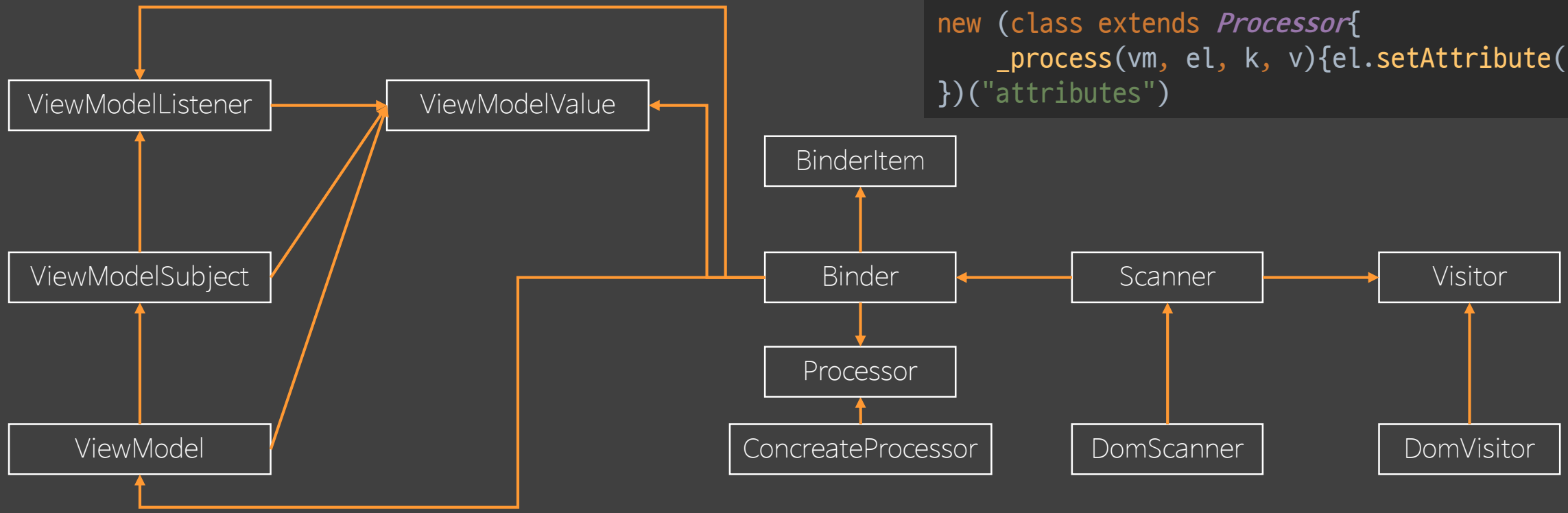


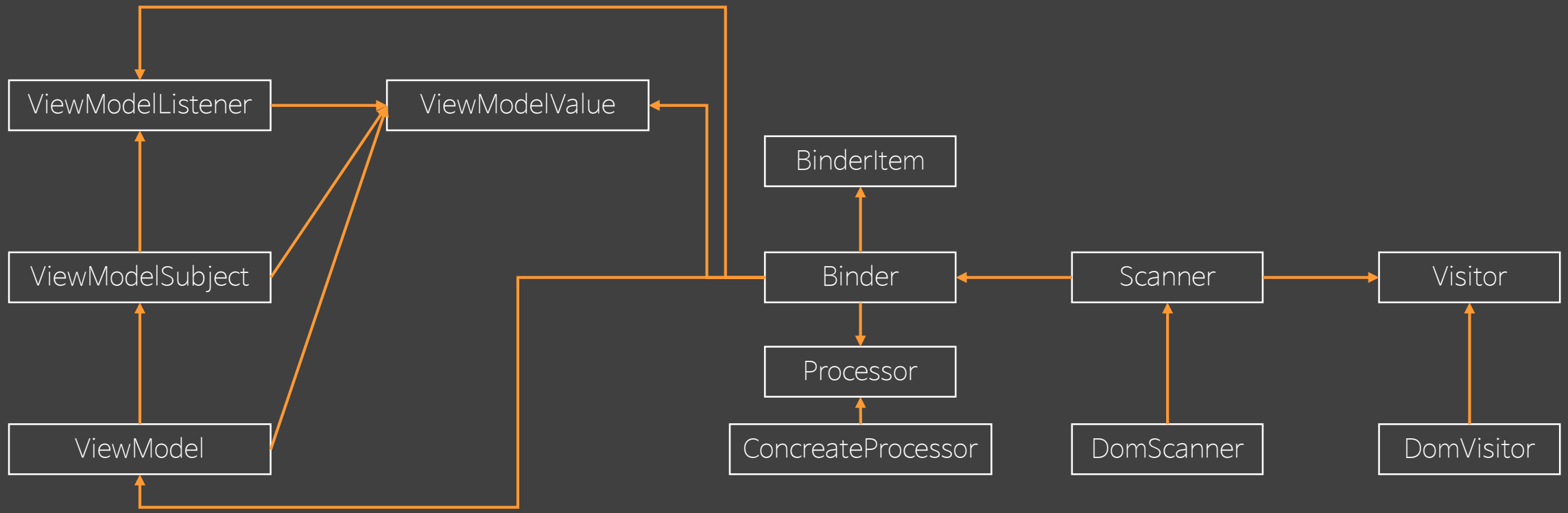
```
const Processor = class{
    cat;
    constructor(cat){…}
    process(vm, el, k, v) {…}
    _process(vm, el, k, v){throw "override";}
};
```

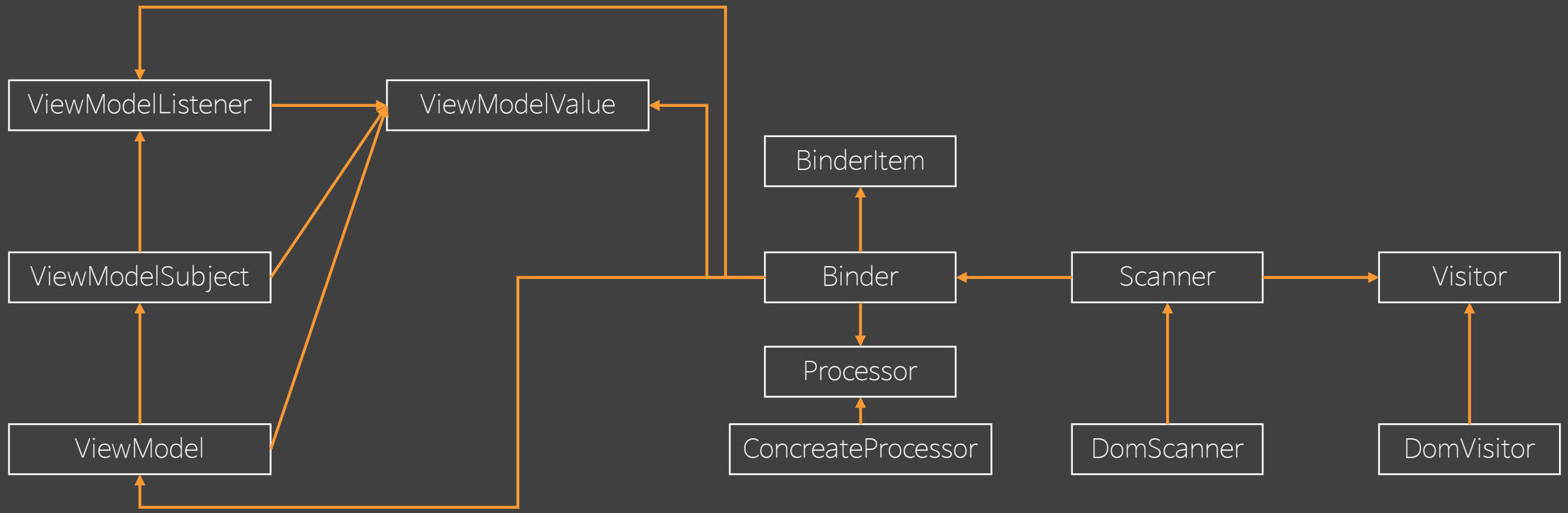


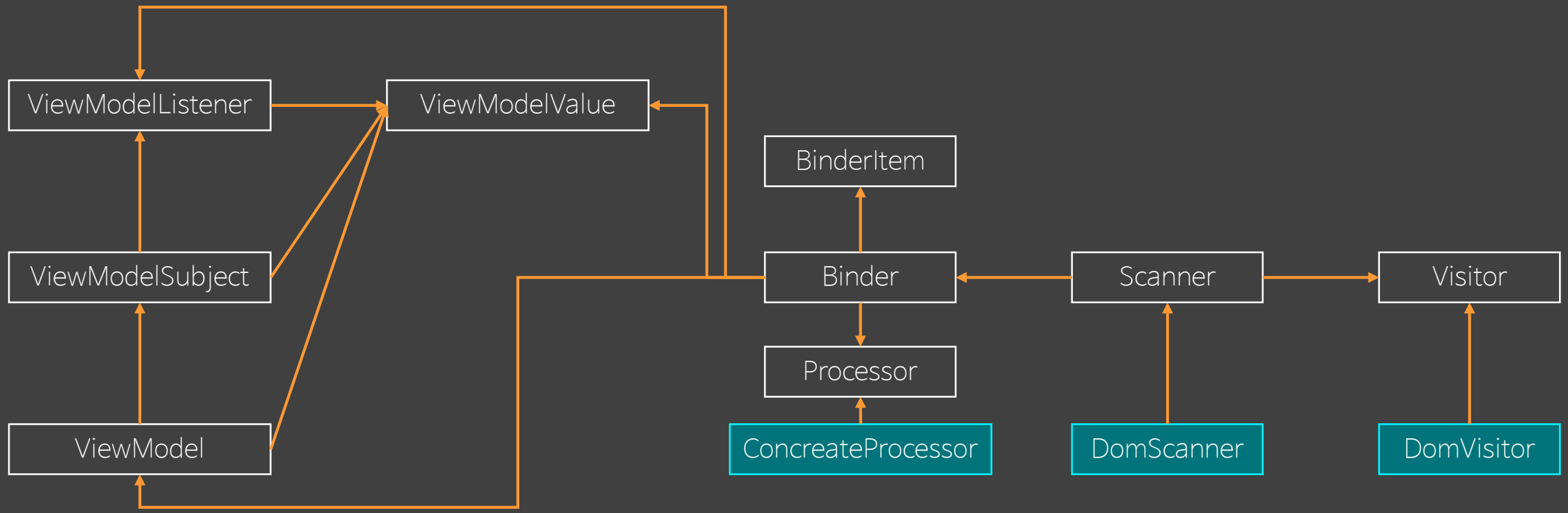
```
new (class extends Processor{
    _process(vm, el, k, v){el.style[k] = v;}
})("styles")
```

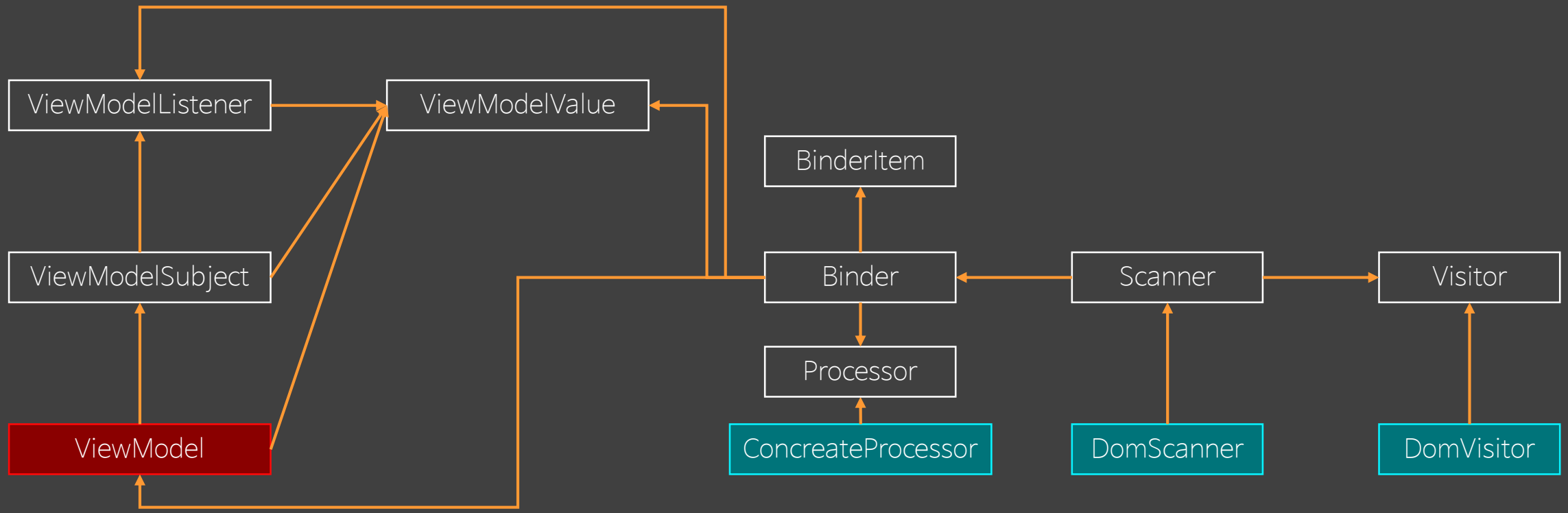
```
new (class extends Processor{
    _process(vm, el, k, v){el.setAttribute(k, v);}
})("attributes")
```




ViewModelListener
ViewModelValue
BinderItem
ViewModelSubject
Binder
Scanner
Visitor
Processor
ViewModel
ConcreateProcessor
DomScanner
DomVisitor


ViewModelListener
ViewModelValue
BinderItem
ViewModelSubject
Binder
Scanner
Visitor
Processor
ViewModel
ConcreateProcessor
DomScanner
DomVisitor


ViewModelListener
ViewModelValue
BinderItem
ViewModelSubject
Binder
Scanner
Visitor
Processor
ViewModel
ConcreateProcessor
DomScanner
DomVisitor


ViewModelListener
ViewModelValue
BinderItem
ViewModelSubject
Binder
Scanner
Visitor
Processor
ViewModel
ConcreateProcessor
DomScanner
DomVisitor