WRITING INTERFACE CODE IS HARD

```javascript
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    },
draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```
STATE MANAGEMENT
AUTOMATA ARE ACTUALLY USED TO MODEL UI
AUTOMATA ARE ACTUALLY USED TO MODEL UI

```javascript
export default (reduxState: State = defaultValue, action: Actions): State => {
  switch (action.type) {
    case "SUBMIT_FRUIT":
      return {
        ...reduxState,
        state: "fruit_loading",
        form: action.form
      };
    case "SUBMIT_FRUIT_ERROR":
      return {
        ...reduxState,
        state: "fruit_error",
        error: action.error
      };
    case "SUBMIT_FRUIT_OK":
      return {
        ...reduxState,
        state: "fruit_ok",
        response: action.response
      };
    default:
      exhaustiveCheck(action.type);
      return reduxState;
  }
};
```
AUTOMATA ARE ACTUALLY USED TO MODEL UI

```javascript
export default (reduxState: State = defaultState, action: Actions): State => {
  switch (action.type) {
    case "SUBMIT_FRUIT":
      return {
        ...reduxState,
        state: "fruit_loading",
        form: action.form
      };
    case "SUBMIT_FRUIT_ERROR":
      return {
        ...reduxState,
        state: "fruit_error",
        error: action.error
      };
    case "SUBMIT_FRUIT_OK":
      return {
        ...reduxState,
        state: "fruit_ok",
        response: action.response
      };
    default:
      exhaustiveCheck(action.type);
      return reduxState;
  }
};
```

- **initial**
  - SUBMIT_FRUIT → fruit_loading (1)
  - SUBMIT_FRUIT_ERROR → ? (2)
  - SUBMIT_FRUIT_OK → ? (2)

- **fruit_loading**
  - SUBMIT_FRUIT → fruit_loading (3)
  - SUBMIT_FRUIT_ERROR → fruit_error (4)
  - SUBMIT_FRUIT_OK → fruit_ok (5)

- **fruit_error**
  - SUBMIT_FRUIT → fruit_loading (6)
  - SUBMIT_FRUIT_ERROR → ? (7)
  - SUBMIT_FRUIT_OK → ? (7)

- **fruit_ok**
  - SUBMIT_FRUIT → fruit_loading (6)
  - SUBMIT_FRUIT_ERROR → ? (7)
  - SUBMIT_FRUIT_OK → ? (7)
VISUAL DEBUGGERS
VISUAL DEBUGGERS

- **No drag**
  - **mouse down**
  - **double click**

- **Drag**
  - **on('mousedown', this)**
  - **on('mouseup')**
  - **on('dblclick', this)**

- **Drag Lock**
  - **on('click')**

- **Add Field**
  - **prototypes (div)**
  - **dom.div**
  - **x**: 313
  - **y**: 763
  - **fill**: 'black'

- **Draggable**
  - **copies**: 1

- **Own**
  - **drag**
  - **drag_lock**

- **Mouse Up**
VISUAL DEBUGGERS
```javascript
x = cjs(1)
y = cjs(() => {
    x.get() + 1;
})
x.set(2)
print(y.get()) // 3
```
Constraint.js like things are used in industry.

Conceptually Mobx treats your application like a spreadsheet. ... everything that can be derived from the application state, will be derived. Automatically.

- Mobx
IMPLEMENTING IDEAS IN A LANGUAGE

- Inheritance
- Good debugging
- JS interoperability
INHERITANCE

- Target implementation uses prototype model
- Multiple Inheritance is essential

A.value = 1
B.value = 2
C.value = 3
D.value = ?
Proposal: C# Stateless Runtime-Polymorphic Mixins (Extension methods 2.0). #303

dmitryse opened this issue on Mar 20, 2017 · 2 comments

dmitryse commented on Mar 20, 2017 · edited ·

This proposal allows to apply general code (from any assembly) to some existing class/struct/interfaces without any modification in their source code.

```csharp
// The same general code but implemented via C# mixin (extension methods 2.0)
public mixin AMyEnumerable<T>
    extends IEnumerable<T>
{
    int GetMySum<T>()
    {
        int sum = 0;
        // In this line "this" is extension target object (of type IEnumerable<T>)
        foreach (T item in this)
        {
            // This is meaningless, only for example.
            sum+=item.GetHashCode();
        }
        return sum;
    }
}
```
DEBUGGING

- Constraint based systems are notoriously hard to debug.
- Ensure that the system is in a consistent state before breaking.

```java
largestPrimeLessThan(x)
let k = x;
while(k > 1) {
    if isPrime(k) {
        return k;
    }
    k--;
}
```
**EVALUATION**

- 20 programmers
- Task: Add a specified feature to given code
- Randomly counterbalanced between native JS and InterState
- JS: $28.2 \pm 7.6$ min
- IS: $14.7 \pm 5.5$ min ($p < 0.01$)
THINGS TO THINK ABOUT

- A whole new language??
  - Elm
  - TypeScript & JSX made it
  - Dart didn’t
THINGS TO THINK ABOUT

- A whole new language??
  - Elm
  - TypeScript & JSX made it
  - Swift style syntactic extensions

```swift
import Foundation

prefix operator √
prefix func √(lhs: Double) -> Double {
    return sqrt(lhs)
}

let someVal: Double = 25
let squareRoot = √someVal
print(squareRoot)
```
THINGS TO THINK ABOUT

- A whole new language??
  - Elm
  - TypeScript & JSX made it
  - Swift style syntactic extensions
  - Mixins

```javascript
let calculatorMixin = Base => class extends Base {
    calc() {}
};

let randomizerMixin = Base => class extends Base {
    randomize() {}
};
```

A class that uses these mix-ins can then be written like this:

```javascript
class Foo {}
class Bar extends calculatorMixin(randomizerMixin(Foo)) {}
```
THINGS TO THINK ABOUT

- A whole new language??
  - Elm
  - TypeScript & JSX made it
  - Swift style syntactic extensions
  - Mixins

```javascript
let calculatorMixin = Base => class extends Base {
  calc() {}
};

let randomizerMixin = Base => class extends Base {
  randomize() {}
};
```

A class that uses these mix-ins can then be written like this:

```javascript
class Foo {}

class Bar extends calculatorMixin(randomizerMixin(Foo)) {}
```

```javascript
import {observer} from "mobx-react"
import {observable} from "mobx"

@observer class Timer extends React.Component {
  @observable secondsPassed = 0

  componentWillMount() {
    setInterval(() => {
      this.secondsPassed++
    }, 1000)
  }
```
THANK YOU