ES6 Subclassing Built-ins

TC39 – July 2012
Allen Wirfs-Brock
Mozilla
The Problem

> function MyDate(timeValue) { this.setTime(timeValue); }  
> MyDate.prototype.__proto__ = Date.prototype;

> var md = new MyDate(Date.now());
TypeError on line 3: Date.prototype.setTime called on incompatible Object

Why is this an error?

• The specification says so:

“...properties of the Date prototype object... none of these functions are generic; A TypeError exception is thrown if the this value is not an object for which the value of the [[Class]] internal properties is “Date”. Also, the phrase “this time value” refers to ...the value of the [[PrimitiveValue]] internal property of this Date object.”
What’s the intention of this requirement

“...Also, the phrase “this time value” refers to ...the value of the [[PrimitiveValue]] internal property of this Date object.”

• It ensures that that methods that reference internal properties actually have that required internal property.
Typing within the Specification

• Currently nominal type is used to ensure the safety of such methods:
  “(if the) value of the [[Class]] internal property is ‘Date’... “

• Behavior typing would be just as safe:
  If this object has a [[[PrimitiveValue]]] internal property...

Switching to behavior typing eliminates dependencies upon a particular built-in object/
[[Class/[[NativeBrand]]]
How do built-ins instances acquire internal properties?

• They are described in the specification.
• Implicitly (and sometimes explicitly) they are added to instances by the built-in constructors.
• But they are not inherited.
How to add inherited internal properties to subclass instances.

- Run the inherited constructor:
  ```javascript
  function MyDate(timeValue) {
    Date.call(this);
    this.setTime(timeValue);
  }
  ```

- Constructor must added the internal properties to the passed this value.
- Internal properties must be implemented as “expandos”.