Class Evaluation Order

BRIAN TERLSON AND YEHUDA KATZ

ES6 Class Evaluation Order

1. A new lexical environment is created with an uninitialized binding for the class name.
2. Extends clause is evaluated and the class's prototype object is created.
3. The constructor is evaluated and the class's function object is created.
4. Each non-constructor element of the class is evaluated in-order by first evaluating the property name (possibly computed) and then creating the element functions' object and element accessor's getter/setters.
5. The lexical environment's binding for the class name is set to the class's function object.

In-order Evaluation: Unrealistic

(UNLESS MAX-MIN IS BASICALLY MAX-MAX)

Impossible scenarios with in-order evaluation

1. Instance data property initializers of any sort
   they must run at initialization time
2. Static data property initializers that refer to the class in any way
   the class is in TDZ until construction is finished
3. Class decorators
   obviously can't be run before you've even seen the class keyword
4. Member decorators
   can't apply right-to-left as is natural
   can't work on getter/setter pairs as they may be lexically separated
5. Integrity modifiers via decorators impossible
   can't add static fields to @sealed class without a decorator

Out-of-order Evaluation: Not Nonsense

Out-of-order Evaluation

Makes high-value features possible (eg. decorators, instance initializers)

Can make intuitive sense and be reasoned about, eg:

- Instance initializers must run at instance creation time
- Class decorators must run before static initializers (since initializers can see the class)

Exists in other languages with very static classes (eg. Java)

Exists in ES supersets (eg. TypeScript, Babel)

Gets lots of use in ES supersets without confusion

Class.next Evaluation Order Strawman

1. A new lexical environment is created with an uninitialized binding for the class name.
2. Extends clause is evaluated and the class's prototype object is created.
3. The constructor is evaluated and the class's function object is created.
4. Each non-constructor element of the class is evaluated in order
    a. Evaluate decorator expressions
    b. Evaluate computed property names
    c. Create necessary objects for the element (functions, getters/setters)
    d. Store the result in an element record list
5. Unobservably coalesce getter/setter pairs into a single element record. Half of a getter/setter pair is never exposed even when its corresponding elements are lexically separated in the class body
6. Class elements are decorated in order by executing their associated decorator functions.
7. Class itself is decorated by executing its associated decorator functions.
8. Class object representation is constructed from the possibly decorated pieces.
9. The lexical environment's binding for the class name is set to the class's function object.
10. Static elements installed onto the class in order by evaluating their initializers.
11. Apply decorator transformations

Object Evaluation Order

Makes sense to align Object evaluation order?

Breaking change:

```
{
  [expr1]: expr2,
  [expr3]: expr4
}
```

Would make decorating objects a possibility in the future