@decorator export Ordering

Daniel Rosenwasser, Ron Buckton

Background

- In 2015, TypeScript implemented an early version of decorators behind a flag.
- TypeScript 5.0 Beta implements decorators as per stage 3.
  - Becomes the default – "old" decorators are still available via the same flag.
  - Full support planned for stable release in March
- TypeScript's early version of decorators slightly differs in syntax from the current proposal's.
  - We want to help our users transition to the standard.

Decorators first

```
@decorator(
  // ...
)
export class C {
    // ...
}
```

export first

```
export @decorator(
  // ...
)
class C {
    // ...
}
```

Why the change?

- Discussed extensively
  - https://github.com/tc39/proposal-decorators/issues/69
- One reason: theoretical ability to decorate a local, but not an export
  - https://github.com/tc39/proposal-decorators/issues/135
  - Idea is that a future proposal could use the decorators-before syntax.
  - But no motivating use-cases for this.

The issue with that

```
@decorator
export class Foo {
    static makeSpecialFoo() {
        return new Foo();
    }
}

let x = new Foo();
```

The issue with that

```
@decorator
export class Foo {
    static makeSpecialFoo() {
        return new Foo();
    }
}

let x = new Foo();
```

- We think this is **a major footgun**.
- It would be bad if **Foo** referred to the pre-decorated class.
- Too subtle.

Also: lack of positive feedback

- Feels like most of us preferred the original syntax
  - But we've resisted discussing it further.
  - Nobody wants to "deadlock" the proposal.
- TypeScript's syntax has shipped for almost 8 years.
  - Almost no demand for `export @decorator`
    - which was brought up at least 5 years ago.

So where are we?

- The TypeScript team would not support making a semantic distinction between

```
export @decorator
```

and

```
@decorator export
```

- We are okay with expanding the syntax, but **not** differing semantics.

Can we make a change?

- We believe there's no future for differing semantics based on ordering.
- The previous syntax is already widely-used – current syntax makes upgrades harder.
- Anecdotally, most library authors that shipped class decorators prefer the "old" ordering.

We would like to request one of the following changes to the decorators proposal.

- Option 1: Decorators are placed **before** the `export` keyword.
  - *Our preference*
- Option 2: Decorators can be placed **before or after** the `export` keyword.
  - Preference for exclusive-or

Appendix: Abridged Syntactic Modifications

StatementListItem and Declaration

StatementListItem[Yield, Await, Return] :

Declaration[?Yield, ?Await, +Decorators]

Declaration[Yield, Await, Decorators] :

ClassDeclaration[?Yield, ?Await, ~Default, ?Decorators]

Appendix: Abridged Syntactic Modifications

ExportDeclaration

ExportDeclaration[Yield, Await, Decorators] :

export *Declaration*[~Yield, ~Await, ~Decorators]

DecoratorList[~Yield, ~Await]
export *ClassDeclaration*[~Yield, ~Await, ~Default, ~Decorators]

DecoratorList[~Yield, ~Await]
export default *ClassDeclaration*[~Yield, ~Await, ~Default, ~Decorators]

Appendix: Abridged Syntactic Modifications

ClassDeclaration

ClassDeclaration[Yield, Await, Decorators] :

	class *BindingIdentifier ClassTail*[?Yield, ?Await]
[+Default]	**class** *ClassTail*[?Yield, ?Await]
[+Decorators]	*DecoratorList*[?Yield, ?Await] **class** *BindingIdentifier ClassTail*[?Yield, ?Await]
[+Decorators, +Default]	*DecoratorList*[?Yield, ?Await] **class** *ClassTail*[?Yield, ?Await]