Record & Tuple

Unified equality, an important objective.

Unified equality, an important objective.

Concern: Records and Tuples need to be compared, but by which semantics should they be compared? **Identity? Contents? Does the language decide this, or does the developer?**

Question: Should Record & Tuple comparisons be implemented with a function/method or the === operator?

“Unified equality” is an important property to uphold for records/tuples.

Unified equality, an important objective.

What does “unified equality” mean?

- One view of equality
- Equality is reliable
- Common operations share semantics
- Hiding identity for coherence.

Invariant: one view of equality.

ECMA262 provides a single, coherent model of equality for every value.

Objects only define equality by “identity”

```
{}  !==  {}
```

Primitives only define equality by “contents”

```
"foo" === "foo"
```

Invariant: Equality is reliable

ECMA262 does not describe other methods of determining equality:

- no operator overloading (for ===)
- no hash codes

This makes `Map` and `Set` possible, as they rely on stable equality.

Developers have come to rely on the “reliability” of equality in a language where most builtins can be modified.

Invariant: common ops share equality semantics.

For example: strings are compared via the same equality semantics in all circumstances.

```
const a = "foo bar";
const b = "foo bar";
a === b; // true

const set = new Set();
set.add(a);
set.has(b); // true
```

Invariant: Hiding identity for coherence.

Developers are often confused by accidental identity comparisons in other languages: https://stackoverflow.com/questions/513832/how-do-i-compare-strings-in-java/513839#513839

- Objects are “owned” by the things that point to them.
- Nothing “owns” a String, it’s existence is self-describing.

Identity is a Pandora’s Box; once it is exposed, it is impossible to remove, preventing future interning/optimization.

Side Note: Read-Only Collections

Read-Only Collections and Record/Tuple can share design where the proposals overlap, but we don't think it makes sense to explain one in terms of the other.

Read-Only Collections have different usage patterns:

- Created, mutated, and snapshotted over time
- Uncommon to compare equality
- Tend to be larger than objects (in # of keys); interning less effective