Using raw objects or typescript interfaces

On dynamic languages, it's likely that you will use directly some primitive objects that can represent the JSON. This is the anti-pattern `primitive obsession`.

This will most likely increase code duplication, and, everywhere you use this object you will ask `obj.type`.

With typescript interfaces, you are just putting a mask on a raw object...

Using raw objects or typescript interfaces

```
const my_object = {} as IMyObject;

if (my_object.type == SOME_CONSTANT) {
  ...
} else if (my_object.type == OTHER_CONSTANT) {
  ...
}
```

When you are dealing with a state machine

You always need to ask which state the application is.

```
;; lisp
(case current-step
   (:welcome (welcome-do-something))
   (:form (form-do-something)))

# ruby
case current_step
when :welcome
welcome_do_something()
when :form
form_do_something()
end
```

When you are dealing with a state machine

Use classes to represent the states of application.

```
;; lisp
(do-something current-step)

// any other language
current_step.do_something()
```

When you are transitioning within a state machines

When you try to execute a transition using the variables in the scope:

```
{ // of course there are better
  // examples than this :)
  if (a && b) transition_to_d()
  if (a) transition_to_b()
  if (b) transition_to_c()
 ...
}
```

When you are transitioning within a state machines

In this case, it is possible that every transition happens
when this block is executed.

To clean that, it shouldn't be possible to represent invalid transitions.

```
case (next_state(a, b))
when Transition::B
  transition_to_b()
when Transition::C
  transition_to_c()
when Transition::D
  transition_to_d()
end
```