U5PMCBK6V : ah thanks. Stream of messages terminology is relatable to me, coming from js world
U5PMCBK6V : and the elm compiler checks that each possible message maps to an update function
U23SA861Y : mm, there is only a single update function, but it checks that the update function handles all possible messages
U5PMCBK6V : ah thanks
U5X2ZRFDF : Is it considered bad style to create a type with a single constructor wrapping a record? The down side is that you have to pattern match the constructor to get at the record in order to use dot notation. The up side is the strong nominal typing guarantee.
U5X2ZRFDF : E.g. `type Foo = Foo { x : Int, y : Int }`
U5X2ZRFDF : I guess Evan also pointed out that wrapping the record can be used to hide the representation from library users. I probably have no need for that in my case.
U2UGVS24E : <@U5X2ZRFDF> I would not consider it bad style at all. In our application we tried to do that as much as possible where it made sense, for the reason you described. Elm has great pattern matching, so it's very easy to access said fields in a function definition, etc.
E.g

```
myFunc (Foo {x, y}) =
    x + y
```

U2UGVS24E : If I recall correctly, you don't need the parentheses if you only have one argument
U2UGVS24E : And if you need to reference the entire value again, you can use the `as` keyword:
```
myFunc (Foo {x, y} as fooValue) =
   ...
```

U5X2ZRFDF : Good point, thanks. I have been accessing the record via:```
myFunc (Foo foo) =
    foo.x + foo.y
```
...so that's another option. What's tricky is if you have a list of foos and you want to map an accessor over them.

U153UK3FA : <@U5X2ZRFDF> The common pattern for that is to define a `map` function for your type that applies functions to the unwrapped record value
U5X2ZRFDF : That makes total sense, considering it's a separate type, and Elm likes to keep its map functions clearly distinct.
U5X2VC483 : <@U0JFEBK6F> <@U2M4VPZ9D> thanks for the heads-up about fluxxu's elm-hot-loader and create-elm-app
U2M4VPZ9D : No problem <@U5X2VC483> I was at a Meetup and some ClojureScript people were talking about how cool ClojureScript is at hot code reloading. Elm is not at that level, but with the hot loader, it is close enough.
U5WD40ZA9 : I'm trying to create a input autocomplete dropdown like this one:
<http://gregziegan.com/elm-autocomplete/>
I've managed to do so, but I would like to add the feature that when the dropdown suggestions is shown and the user presses outside of the dropdown, the dropdown closes. Any suggestions :slightly_smiling_face:?

U0CLDU8UB : The classical JS solution is to make a click handler that closes it for the entire page behind the dropdown.
U4872964V : <@U0CLDU8UB> but not for clicks in the input field, right? I wonder how to best do that in Elm
U0JFEBK6F : oscarevert: The way I've done it: Keep track of the rect for your input/dropdown (i.e in your model), you could use something like <http://package.elm-lang.org/packages/debois/elm-dom/1.2.3/DOM#boundingClientRect>) and then set up a subscription to <http://package.elm-lang.org/packages/elm-lang/mouse/1.0.1/Mouse#clicks> to check for clicks outside your input/dropdown
U2D5SAEMN : I'm building a filterable list of events and I'm wondering if I'm on the right track:
<https://ellie-app.com/3xVrLFYmdnra1/3> Specifically:1. As `EventType` is a union type, how can I filter the list of events on e.g. `Birthday` without taking it's extra `Int` parameter (age) into account?
2. When I pass `Birthday` as the `EventType` to the `SetTypeFilter` message, I need to pass in an extra value for the age as well. Would there be a better way to do this?
```

U0CLDU8UB : Yeah, this seems odd and could lead to bugs: `(\_ -&gt; SetTypeFilter (Birthday 0))`
U0CLDU8UB : What do the numbers represent in the types?
U2D5SAEMN : with Birthday it's the age he/she will reach, with Employment it's the work anniversary (e.g. working for 5 years at the company), with Marriage it's the wedding anniversary (e.g. being married for 10 years)
U0CLDU8UB : oooh, okay
U2D5SAEMN : I use these in the rendering, e.g. "2017-06-21 It's the birthday of Pete Promo, becoming 42 years old"
U2D5SAEMN : Since the numbers belong to the event type, I figured using a union type.
U0CLDU8UB : Hmm, this makes me feel like putting a function in the SetTypeFilter message, but that's never a good solution (and it also makes the debugger not work)...
U0CLDU8UB : There's probably an easier modeling of this problem.
U2D5SAEMN : I could place the numbers in a separate property on the Event, e.g. `payload: Maybe Int`? Then the EventType union type doesn't need the extra parameter.
U0CLDU8UB : I have an idea that would work rather nicely, but involves understanding extensible records. Your idea works too, and it's not bad at all.
U0CLDU8UB : I'll update the Ellie to the idea that I have just as a reference, in case you're interested
U2D5SAEMN : really? wow, thanks! :raised_hands:
U4872964V : A simple option is to have a separate union type for filters, like `BirthdayFilter` etc, and then have a filter function that does `case` on that type. More boilerplate, but type safe
U2D5SAEMN : How would the filter function work then?
U0CLDU8UB : Yeah, the idea that I had is actually much more cumbersome in code than I thought it would be, so let's scratch that. :slightly_smiling_face:
U4872964V : So, I'd do like this (but this is just a preference)```
```
filterEvent filter event =
  case (filter, event) of
    (ActiveFilter, Active) -&gt; True
    (ActiveFilter, _) -&gt; False
    (BirthdayFilter, Birthday _) -&gt; True
    (BirthdayFilter, _) -&gt; False
    ...
```
I generally do it like this so I don't have to have a "catch all"

U4872964V : or the arguments in the other order, maybe, for your usage of the filter
U2D5SAEMN : `event` would be the `eventType` property of the event record, I guess?
U4872964V : another option is to do it like this```
```
filterEvent filterList event =
  case event of
    Active -&gt; List.member ActiveFilter filterList
    Birthday _ -&gt; List.member BirthdayFilter filterList
    ...
```
sorry if I've not understood your problem correctly

U2D5SAEMN : ohanhi: Thanks anyway!
U2D5SAEMN : nice, that would support multiple filters as well. Thanks a lot!
U4872964V : Generally I find that making specific types pay off in simplicity and type safety, even if it leads to more code
U4872964V : you're welcome!
U4872964V : you could of course break it out to make a function `: Event -&gt; EventFilter` and make that `filterEvent` function more general, it's just refactoring at that point
U2D5SAEMN : Sorry, I don't understand what you mean by that? :slightly_smiling_face: