Maybe

Let me have the great honour of introducing `Maybe` to you. `Maybe` will soon become your best friend. They are always there for you when times are tough. Why is `Maybe` so great? Well when you start learning about the more complex and some say "scary" concepts in Haskell (they aren't actually scary, they are really just warm fuzzy things. Horribly misunderstood creatures) `Maybe` is always a good example to start with.

So let's get to know `Maybe`:

```
data Maybe a  =  Nothing
              |  Just a
              deriving Show
```

`Maybe` can take on two values. It can be `Nothing` or `Just a`. This magical `a` is polymorphic and can be anything. You can have any type of `Maybe` you want, from `Maybe Int` to `Maybe [[String]]`. The defining quality of `Maybe` is how it encapsulates failure. If one of your functions fails, `Maybe` is like a shock blanket you can wrap it in, reassuring it that sometimes it is okay to fail and fall flat on your face.

Take the terrible twins: `head` and `tail`. These kids blow up and throw exceptions when you give them the empty list. What even is the first element of the empty list? You don't even know so why on earth so we expect `head` to? That's just mean. Unrealistic expectations. Instead we could be good parents and tell both `head` and `tail` how to deal with the empty list. Give them a way of throwing their hands up and saying "I don't know man".

```
head :: [a] -> Maybe a
head []      =  Nothing
head (x:xs)  =  Just x

tail :: [a] -> Maybe [a]
tail []      =  Nothing
tail (x:xs)  =  Just xs
```

Much better wouldn't you agree? Nobody wants a Haskell program to throw a runtime exception! Unconscionable!

In the question sheet you are asked to transform a list of `Maybe`s into a `Maybe [a]`. One way to think about this conceptually is dominoes: if one element of the list is `Nothing` then the whole list is `Nothing`; if one domino falls then so do the rest.