Preventing unreachable error branches with type safety

Brandon Chinn

15 June 2021

Agenda

- Intro with personal project
- Application in LeapYear code
- Q + A

Personal Project

Personal Project

Intro to Scattergories

1. Get list of categories + a letter
2. Everyone has 3 minutes to write down answers
3. Everyone's answers are revealed
    - Get a point if you wrote unique answer

Personal Project

Scattergories Implementation v1

```haskell
type PlayerName = Text
type CategoryName = Text

data Round = Round
  { players ::
      Map PlayerName PlayerAnswers
  , letter :: Char
  }
```

```haskell
data PlayerAnswers = PlayerAnswers
  { answers ::
      Map CategoryName Answer
  }

data Answer = Answer
  { answer :: Maybe Text
  , isValid :: Maybe Bool
  }
```

Personal Project

Scattergories Implementation v1

```
-- while players are answering
"Food"   => Answer Nothing Nothing
"Animal" => Answer Nothing Nothing

-- after player has answered
"Food"   => Answer (Just "Beet") Nothing
"Animal" => Answer (Just "Bear") Nothing

-- after scoring
"Food"   => Answer (Just "Beet") (Just True)
"Animal" => Answer (Just "Bear") (Just False)
```

Personal Project

Scattergories Implementation v1

```haskell
-- should only be called after
-- everyone submits answers
getAnswers ::
  PlayerAnswers ->
  [(CategoryName, Text)]
getAnswers (PlayerAnswers answers) =
  Map.toList $ getAnswer <$> answers
  where
    getAnswer Answer{answer} =
      case answer of
        Just s -> s
        Nothing ->
          error "answer is Nothing"
```

```haskell
-- should only be called after
-- everyone is scored
getScore ::
  PlayerAnswers ->
  Int
getScore (PlayerAnswers answers) =
  sum $ score <$> answers
  where
    score Answer{isValid} =
      case isValid of
        Just True -> 1
        Just False -> 0
        Nothing ->
          error "isValid is Nothing"
```

Personal Project

Scattergories Implementation v2

```
data RoundStage = AnswersPending | AnswersDone | AnswersScored

data Round (stage :: RoundStage) = Round
  { players :: Map PlayerName (PlayerAnswers stage)
  , letter :: Char
  }

data PlayerAnswers (stage :: RoundStage) = PlayerAnswers
  { answers :: Map CategoryName (Answer stage)
  }

data Answer (stage :: RoundStage) where
  MaybeAnswer  :: Maybe Text    -> Answer 'AnswersPending
  Answer       :: Text          -> Answer 'AnswersDone
  ScoredAnswer :: Text -> Bool -> Answer 'AnswersScored
```

Personal Project

Scattergories Implementation v2

```
-- while players are answering
-- Map CategoryName (Answer 'AnswersPending)
"Food"   => MaybeAnswer Nothing
"Animal" => MaybeAnswer Nothing

-- after player has answered
-- Map CategoryName (Answer 'AnswersDone)
"Food"   => Answer "Beet"
"Animal" => Answer "Bear"

-- after scoring
-- Map CategoryName (Answer 'AnswersScored)
"Food"   => ScoredAnswer "Beet" True
"Animal" => ScoredAnswer "Bear" False
```

Personal Project

Scattergories Implementation v2

```haskell
getAnswers :: PlayerAnswers 'AnswersDone -> [(CategoryName, Text)]
getAnswers (PlayerAnswers answers) = Map.toList $ getAnswer <$> answers
  where
    getAnswer (Answer answer) = answer

getScore :: PlayerAnswers 'AnswersScored -> Int
getScore (PlayerAnswers answers) = sum $ score <$> answers
  where
    score (ScoredAnswer _ isValid) = if isValid then 1 else 0
```

Applying in LeapYear

Applying in LeapYear

Table schema in Data Manager

1. When a table is first created, schema types are `null`
2. When table is finished creating, schema types are populated (e.g. `INT(1, 10)`)
3. After table is finished creating, admin can edit schema bounds in Data Manager

Applying in LeapYear

Table schema in Data Manager

https://github.com/LeapYear/leapyear/pull/9136/files#r603658735

```
// the TableColumn type we use everywhere;
// corresponds with the graphql type
type TableColumn = {
  name: string
  type: ColumnType | null
  bounds: ColumnBounds | null
  nullable: boolean | null
}
```

```
// useEditTableSchema.ts
if (!column.type) {
  throw new Error(
    "unreachable: " +
    "table has null type when editing schema"
  )
}
```

Applying in LeapYear

Table schema in Data Manager

```
type TableColumnGeneric<IsReady extends 'READY' | 'NOT_READY'> = {
  name: string
  type: IsReady extends 'READY' ? ColumnType : null
  bounds: IsReady extends 'READY' ? ColumnBounds : null
  nullable: IsReady extends 'READY' ? boolean : null
}

type TableColumnReady = TableColumnGeneric<'READY'>
type TableColumnNotReady = TableColumnGeneric<'NOT_READY'>

type TableColumn = TableColumnReady | TableColumnNotReady
```

Appendix: Links + references

- https://github.com/brandonchinn178/categories-with-friends
- https://github.com/LeapYear/leapyear/pull/9136
- https://github.com/LeapYear/leapyear/pull/9141
- https://github.com/LeapYear/leapyear/pull/9140
- `data-manager/src/views/data/databasePage/components/TableDetails/SchemaPanel/`

Appendix: State transitions

```
-- | Attempt to finalize all the answers. If any answers
-- are still pending, returns Nothing.
finalizeAnswers ::
  PlayerAnswers 'AnswersPending ->
  Maybe (PlayerAnswers 'AnswersDone)

-- | Score answers with the given score sheet.
scoreAnswers ::
  PlayerAnswers 'AnswersDone ->
  Map CategoryName Bool ->
  PlayerAnswers 'AnswersScored
```

Appendix: Type narrowing

```
type TableHelper<TableColumn> = {
  id: string
  columns: TableColumn[]
}

type TableNotReady = TableHelper<TableColumnNotReady>
type TableReady = TableHelper<TableColumnReady>
type Table = TableNotReady | TableReady

function isTableReady(table: Table): table is TableReady {
  return isTableColumnReady(table.columns[0])
}

function isTableColumnReady(column: TableColumn): column is TableColumnReady {
  return column.type !== null
}
```