Monads

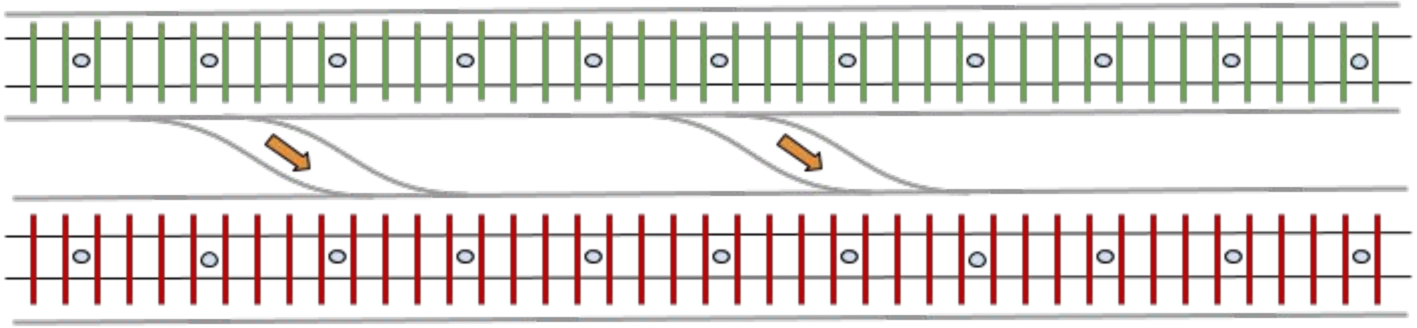
Excuse to play with generics

Filip

Ruby Developer @ SuperSaas

Railway Oriented Programming

Has nothing to do with
Ruby On Rails



Types of monads

```
func subscribe(r *http.Request)  (int, error) {
        token, err := getToken(r)
        if err != nil {
                return 0, err
        }
        data, err := decrypt(token)
        if err != nil {
                return 0, err
        }
        user, err := fetchUser(data.UserID)
        if err != nil {
                return 0, err
        }
        subscriptionID, err := createSubscription(user)
        if err != nil {
                return 0, err
        }
        return subscriptionID, nil
}
```

```
subscribe request =
        getToken request
                |> decrypt
                |> fetchUser
                |> createSubscription
```

OKAFOR
ULC
LINZ
SPORT-THIEME

```go
type Result[T any] struct {
	value T
	err   error
}

func (r Result[T]) Ok() bool {
	return r.err == nil
}

func (r Result[T]) Value() T {
	return r.value
}

func (r Result[T]) ValueOr(val T) T {
	if r.Ok() {
		return r.value
	}
	return val
}

func (r Result[T]) Error() error {
	return r.err
}
```

```go
func (r Result[T]) Unwrap() (T, error) {
	return r.value, r.err
}

func Ok[T any](val T) Result[T] {
	return Result[T]{value: val, err: nil}
}

func Err[T any](err error) Result[T] {
	return Result[T]{err: err}
}

func Fmap[T, U any](r Result[T], f func(T) Result[U])
Result[U] {
	if !r.Ok() {
		return Err[U](r.Error())
	}
	return f(r.Value())
}
```

```
token := Fmap(Ok(request), getToken)

data := Fmap(token, decrypt)

user := Fmap(data, fetchUser)

subscriptionID := Fmap(user, createSubscription)
```

```
func getToken(r *http.Request) Result[string] {
    token := r.Header.Get(“Authorization”)
    if token == “” {
        return Err[string](errors.New(“Missing auth header”))
    }
    splitToken := strings.Split(token, “Bearer”)
    if len(splitToken) != 2 {
        return Err[string](errors.New(“Bearer token required”)
    }
    return Ok(splitToken[1])
}
```

```
return Ok(request)
                .Fmap(getToken)
                .Fmap(decrypt)
                .Fmap(fetchUser)
                .Fmap(createSubscription)
```

Conclusion

Current generics implementation makes it difficult to apply the Fmap method on the monad itself.

Things are moving in a good direction.

Thanks