

Handling Failures and Input Errors in a Functional Way



Vladimir Khorikov

PROGRAMMER

@vkhorikov www.enterprisecraftsmanship.com



Traditional Approach to Handling Failures and Input Errors

Handle all input errors at the boundaries of the domain model

Catch all expected failures at the lowest level possible



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    Customer customer = _database.GetById(customerId);  
    customer.Balance += moneyAmount;  
    _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    _database.Save(customer);  
  
    return "OK";  
}
```



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    customer.Balance += moneyAmount;  
    _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    _database.Save(customer);  
  
    return "OK";  
}
```



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    if (customer == null) {  
        return "Customer is not found";  
    }  
    customer.Balance += moneyAmount;  
    _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    _database.Save(customer);  
  
    return "OK";  
}
```



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    if (customer == null) {  
        return "Customer is not found";  
    }  
    customer.Balance += moneyAmount;  
    try {  
        _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    }  
    catch (ChargeFailedException) {  
        return "Unable to charge the credit card";  
    }  
    _database.Save(customer);  
  
    return "OK";  
}
```



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    if (customer == null) {  
        return "Customer is not found";  
    }  
    customer.Balance += moneyAmount;  
    try {  
        _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    }  
    catch (ChargeFailedException) {  
        return "Unable to charge the credit card";  
    }  
    try {  
        _database.Save(customer);  
    }  
    catch (SqlException) {  
        _paymentGateway.RollbackLastTransaction();  
        return "Unable to connect to the database";  
    }  
  
    return "OK";  
}
```



```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        _logger.Log("Money amount is invalid");  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    if (customer == null) {  
        _logger.Log("Customer is not found");  
        return "Customer is not found";  
    }  
    customer.Balance += moneyAmount;  
    try {  
        _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    }  
    catch (ChargeFailedException) {  
        _logger.Log("Unable to charge the credit card");  
        return "Unable to charge the credit card";  
    }  
    try {  
        _database.Save(customer);  
    }  
    catch (SqlException) {  
        _paymentGateway.RollbackLastTransaction();  
        _logger.Log("Unable to connect to the database");  
        return "Unable to connect to the database";  
    }  
    _logger.Log("OK");  
    return "OK";  
}
```





```
public string RefillBalance(int customerId, decimal moneyAmount) {  
    if (!IsMoneyAmountValid(moneyAmount)) {  
        _logger.Log("Money amount is invalid");  
        return "Money amount is invalid";  
    }  
    Customer customer = _database.GetById(customerId);  
    if (customer == null) {  
        _logger.Log("Customer is not found");  
        return "Customer is not found";  
    }  
    customer.Balance += moneyAmount;  
    try {  
        _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);  
    }  
    catch (ChargeFailedException) {  
        _logger.Log("Unable to charge the credit card");  
        return "Unable to charge the credit card";  
    }  
    try {  
        _database.Save(customer);  
    }  
    catch (SqlException) {  
        _paymentGateway.RollbackLastTransaction();  
        _logger.Log("Unable to connect to the database");  
        return "Unable to connect to the database";  
    }  
    _logger.Log("OK");  
    return "OK";  
}
```



Introducing Railway-oriented Programming

Straying from the happy path

*"As a user I want to update my name and email address"
- and see sensible error messages when something goes wrong!*



```
type Request = {  
  userId: int;  
  name: string;  
  email: string  
}
```

Flowchart steps:

- Receive request
- Validate and canonicalize request
- Update existing user record
- Send verification email
- Return result to user

Error messages from the flowchart:

- From "Validate and canonicalize request": Name is blank, Email not valid
- From "Update existing user record": User not found, Db error

2014 NDC new DevelopersConference 1-5 December · London, UK

Inspiring Developers SINCE 2008

59:25

NDC HD

<https://vimeo.com/113707214>



```
public string RefillBalance(int customerId, decimal moneyAmount) {
    if (!IsMoneyAmountValid(moneyAmount)) {
        _logger.Log("Money amount is invalid");
        return "Money amount is invalid";
    }
    Customer customer = _database.GetById(customerId);
    if (customer == null) {
        _logger.Log("Customer is not found");
        return "Customer is not found";
    }
    customer.Balance += moneyAmount;
    try {
        _paymentGateway.ChargePayment(customer.BillingInfo, moneyAmount);
    }
    catch (ChargeFailedException) {
        _logger.Log("Unable to charge the credit card");
        return "Unable to charge the credit card";
    }
    try {
        _database.Save(customer);
    }
    catch (SqlException) {
        _paymentGateway.RollbackLastTransaction();
        _logger.Log("Unable to connect to the database");
        return "Unable to connect to the database";
    }
    _logger.Log("OK");
    return "OK";
}
```



Introducing Railway-oriented Programming

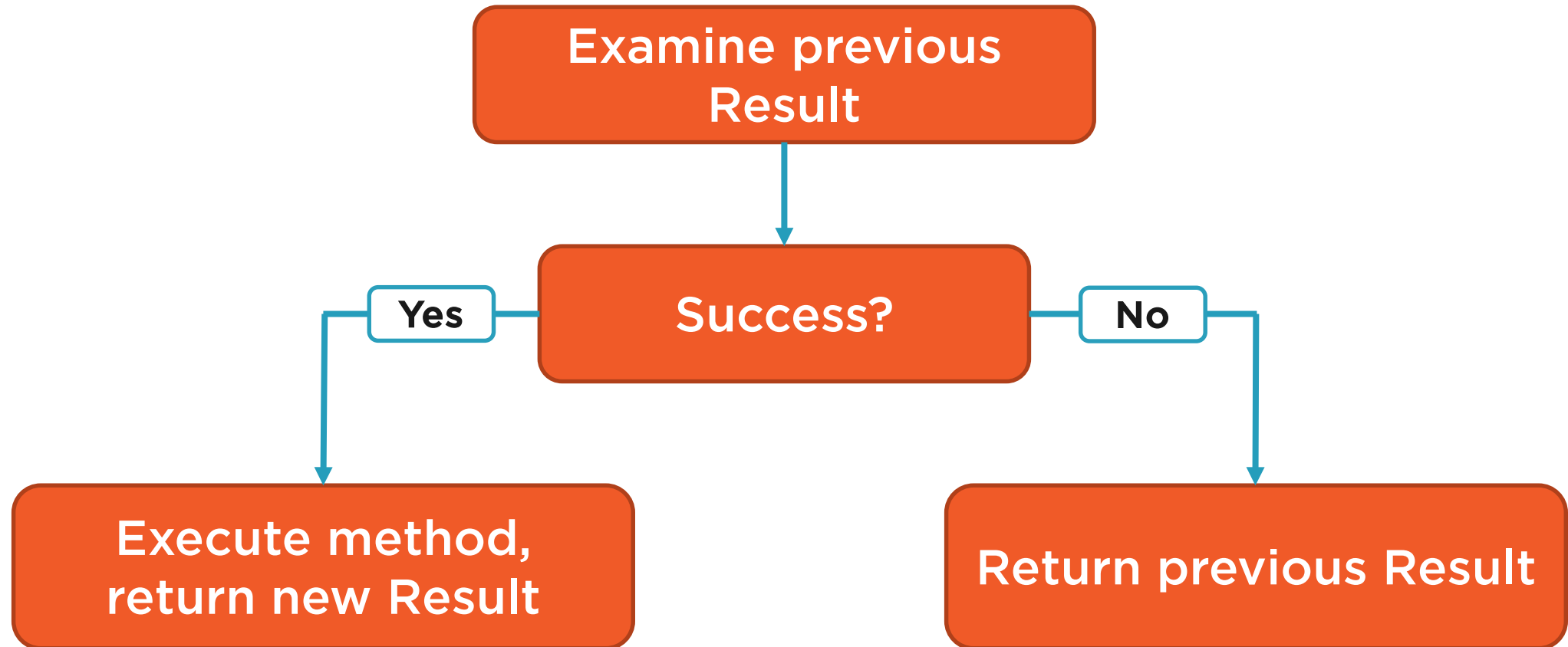
OnSuccess

OnFailure

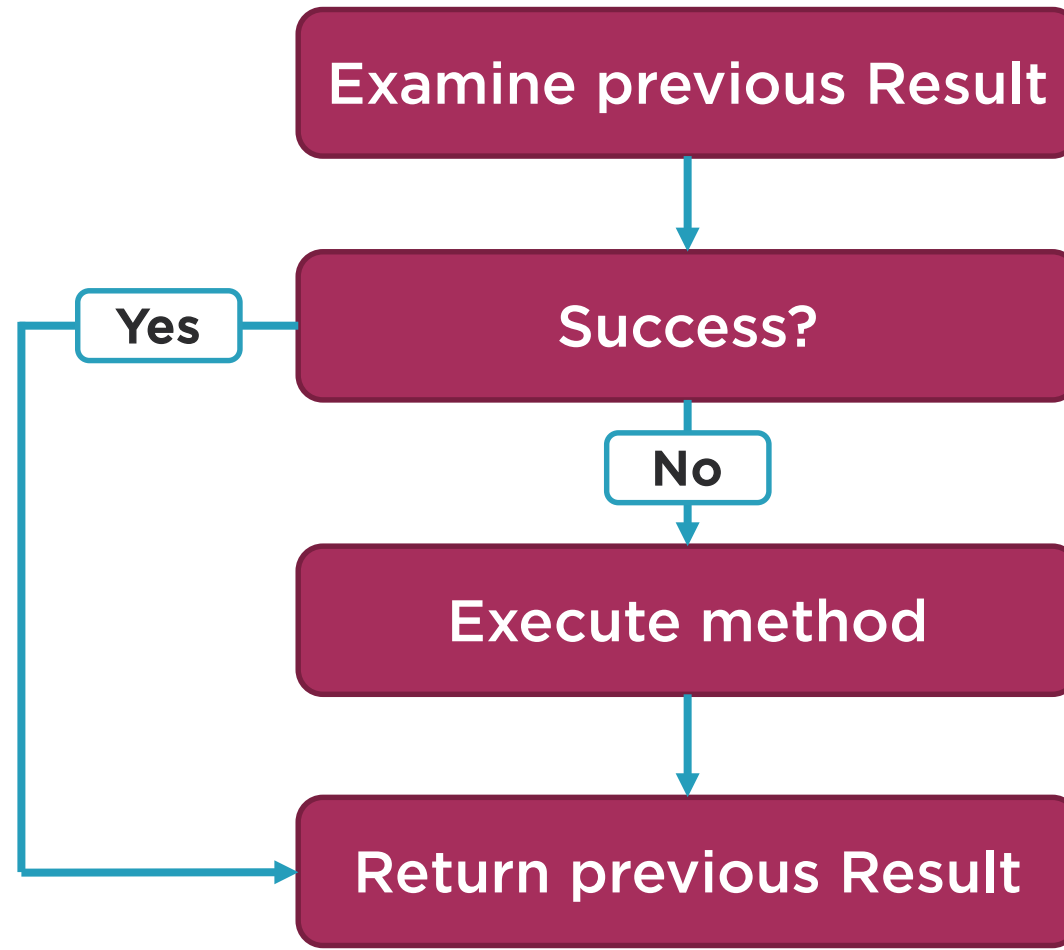
OnBoth



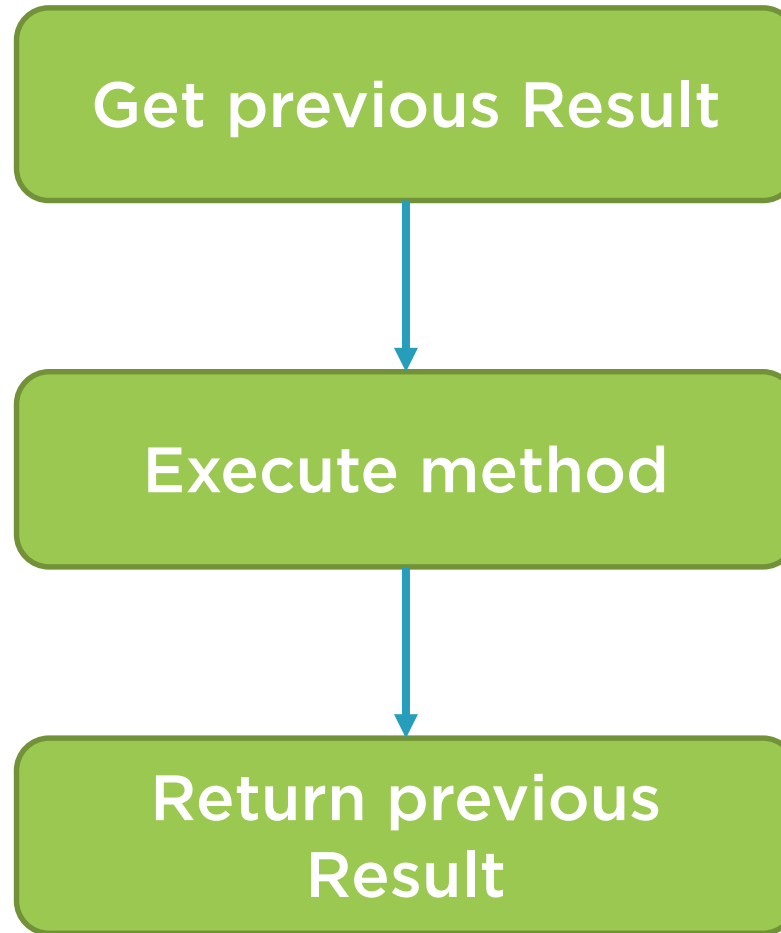
OnSuccess Extension Method



OnFailure Extension Method



OnBoth Extension Method



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(
            () => _database.Save(customer.Value)
                .OnFailure(() => _paymentGateway.RollbackLastTransaction()))
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(
            () => _database.Save(customer.Value)
                .OnFailure(() => _paymentGateway.RollbackLastTransaction()))
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(
            () => _database.Save(customer.Value)
                .OnFailure(() => _paymentGateway.RollbackLastTransaction()))
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(
            () => _database.Save(customer.Value)
                .OnFailure(() => _paymentGateway.RollbackLastTransaction()))
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(() => _database.Save(customer.Value))
        .OnFailure(() => _paymentGateway.RollbackLastTransaction())
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Recap: Introducing Railway-oriented Programming

```
public string RefillBalance(int customerId, decimal moneyAmount)
{
    Result<MoneyToCharge> moneyToCharge = MoneyToCharge.Create(moneyAmount);
    Result<Customer> customer = _database.GetById(customerId).ToResult("Customer is not found");

    return Result.Combine(moneyToCharge, customer)
        .OnSuccess(() => customer.Value.AddBalance(moneyToCharge.Value))
        .OnSuccess(() => _paymentGateway.ChargePayment(customer.Value.BillingInfo, moneyToCharge.Value))
        .OnSuccess(
            () => _database.Save(customer.Value)
                .OnFailure(() => _paymentGateway.RollbackLastTransaction()))
        .OnBoth(result => Log(result))
        .OnBoth(result => result.IsSuccess ? "OK" : result.Error);
}
```



Isn't suitable for sophisticated scenarios



Extension methods can be moved to Result



Summary



Handling failures and input errors in a functional way

Railway-oriented approach

Extension methods

- onSuccess, onFailure, onBoth
- Work on top of Result



In the Next Module

Putting it all together



Conversion between Maybe and nulls



Using the Result class



Employing Value Objects



Saving the domain model into a database

