Better Logging with Mango

Logging in Java typically involves sprinkling your code with calls to log.info() or log.debug() and creating messages with concatenation or string formatting.

The Mango logging framework is intended to formalise this tedious work and create consistent structured log messages.

Let’s first look at a simple Spring service using log4j directly.

Logging with plain ol log statements

```
@Service
public class CustomerService {
    public String checkout(Customer customer, double amount) {
        double tax = amount * 0.23;
        String transactionId;
        log.info("tax {}", tax);

        try {
            transactionId = auth(customer, amount+tax);
            log.info("transactionId {}", transactionId);
            capture(transactionId);
            log.info("capture {}", "successful");
        }
        catch (Exception e) {
            log.info("capture {}", "failed");
            throw e;
        }
    }
}
```

Seems simple enough, now let’s see what sort of log messages we get

Log output

As you can see we get three separate messages, and the only thing helping us see that they belong together is the "correlationId"


```
{
  "ts": "2023-11-12 20:11:780.651+0000",
  "level": "INFO",
  "type": "Business",
  "application": "customer-profile",
  "thread": "http-nio-8080-exec-1",
  "logger": "c.m..controllers.CustomerService",
  "correlationId": "e0dcef10-b290-43b1-9529-d5d17d58ac30",
  "requestId": "958eebbe-056b-4374-7554-e9b47cda4e01",
  "msg": {
    "tax": 4.78
  }
}
{
  "ts": "2023-11-12 20:11:780.651+0000",
  "level": "INFO",
  "type": "Business",
  "application": "customer-profile",
  "thread": "http-nio-8080-exec-1",
  "logger": "c.m..controllers.CustomerService",
  "correlationId": "e0dcef10-b290-43b1-9529-d5d17d58ac30",
  "requestId": "958eebbe-056b-4374-7554-e9b47cda4e01",
  "msg": {
    "transactionId": "01923847109238471"
  }
}
{
  "ts": "2023-11-12 20:11:780.651+0000",
  "level": "INFO",
  "type": "Business",
  "application": "customer-profile",
  "thread": "http-nio-8080-exec-1",
  "logger": "c.m..controllers.CustomerService",
  "requestId": "958eebbe-056b-4374-7554-e9b47cda4e01",
  "correlationId": "e0dcef10-b290-43b1-9529-d5d17d58ac30",
  "msg": {
    "capture": "successful"
  }
}
```

Logging with Mango

Let’s now look at a simple Spring service using Mango logging.

Annotate your data

```
@Getter
@Setter
@Builder
@Loggable // Use LoggerMapper instead of JacksonMapper
public class Customer {
    @Loggable // Log this field
    private final String id;
    @Loggable
    private final String name;
    @Loggable("pan")
    @Mask(char='*', prefix=6, postfix=4) // log this field masked
    private final String cardNumber;
}
```

Annotate your code

```
@Service
public class CustomerService {
    @Loggable // make this method loggable
    @LogReturn // log the return value
    @LogIsolated // log this method in it's own context
    public void checkout(
            @Loggable("customer") Customer customer, @Loggable("amount") double
amount) {
        double tax = amount * 0.23;
        String transactionId;
        LogContext.put("tax", tax);

        transactionId = auth(customer, amount+tax);
        capture(transactionId);
    }
}
```

As you can see we don't have any actual log statements in this code, we just gather information (via AOP) about the method called, the parameters passed, and any other useful data via calls to the LogContext.

Get rich structured logs

And voila! we get data rich, structured logs.


```
{
  "ts": "2023-11-321 09:11:577.1419+0000",
  "level": "INFO",
  "type": "Business",
  "node": "big-lemon-dog",
  "application": "customer-profile",
  "correlationid": "c3a124c8-b2fa-443e-a26a-001700c2b149",
  "thread": "appf-pool-1",
  "logger": "c.m..controllers.CustomerService",
  "logId": "138646a4-8a49-4123-a0a6-0d8ab5e0a011",
  "method": "CustomerService.checkout",
  "context": {
    "tax": 79.47
  },
  "params": {
    "customer": {
      "id": "249872364",
      "name": "John",
      "pan": "678912******5729"
    },
    "amount": 345.56
  },
  "called": [
    {
      "method": "Payments.auth",
      "params": {
        "customer": {
          "id": "249872364",
          "name": "John",
          "pan": "678912******5729"
        },
        "amount": 425.033
      },
      "response" : {
        "transactionId": "120938473120984"
      },
      "responsetime": 2
    },
    {
      "method": "Payments.capture",
      "params": {
        "transactionId": "120938473120984"
      },
      "responsetime": 2
    }
  ],
```

```
  "responsetime": 2
}
```