ASSIGNMENT 2

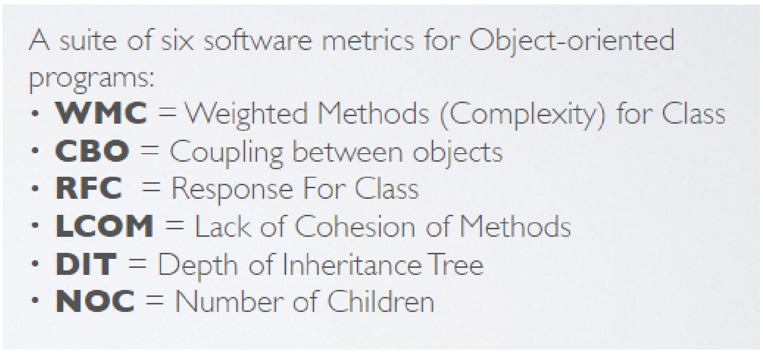
SEM GROUP 01a

Angelika Mentzelopoulou
Marina Serafeimidi
Jarno Berger
Daan Panis
Nicolae Filat
Toma Volentir

TABLE OF CONTENTS

INTRODUCTION

Our task was to refactor classes and methods that require improvement according to software metrics we were taught in class. We decided to analyse results according to the CK Metrics, which by S. Chidamber and C. Kemerer have been named “A Metric Suite for Object Oriented Design”. For the gathering of the values for the chosen metrics, we used *CodeMR,* an architectural software quality and static code analysis tool. After analysis, we ended up with the following data regarding our current code metrics, prior to any refactoring.

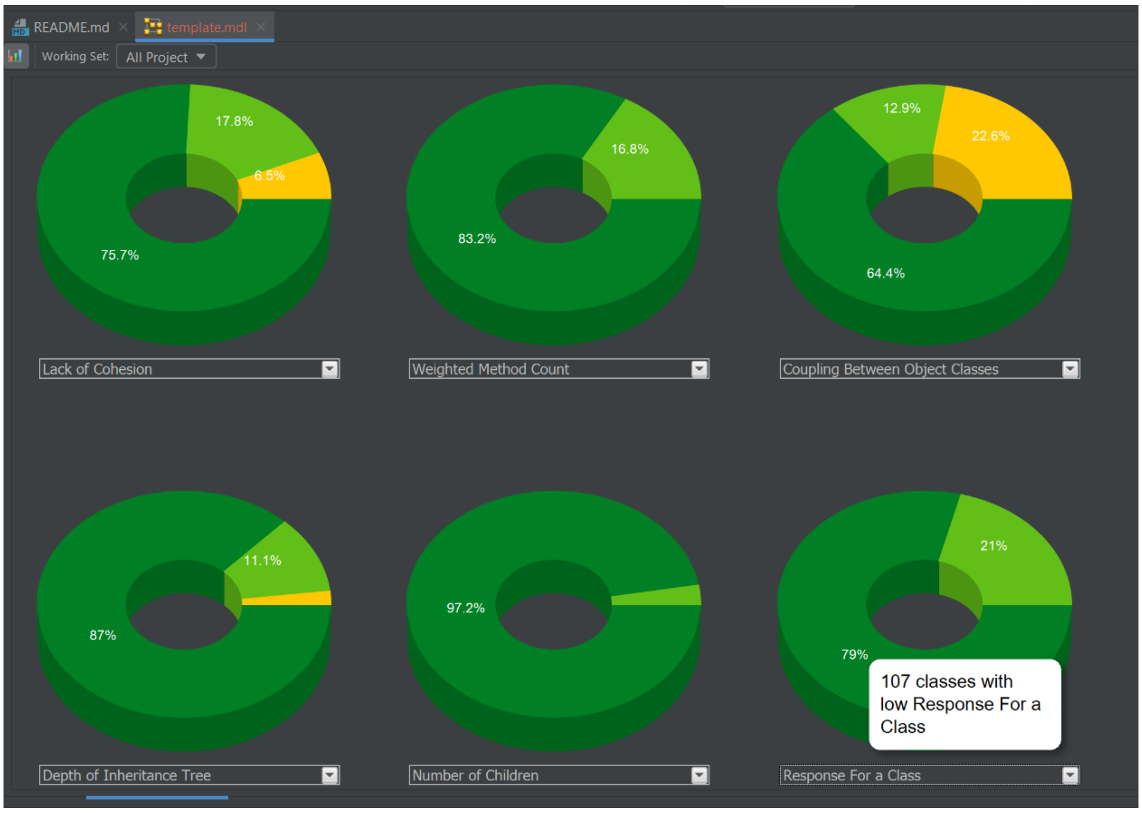


A suite of six software metrics for Object-oriented programs:

- **WMC** = Weighted Methods (Complexity) for Class
- **CBO** = Coupling between objects
- **RFC** = Response For Class
- **LCOM** = Lack of Cohesion of Methods
- **DIT** = Depth of Inheritance Tree
- **NOC** = Number of Children

CK metrics

BEFORE REFACTORING



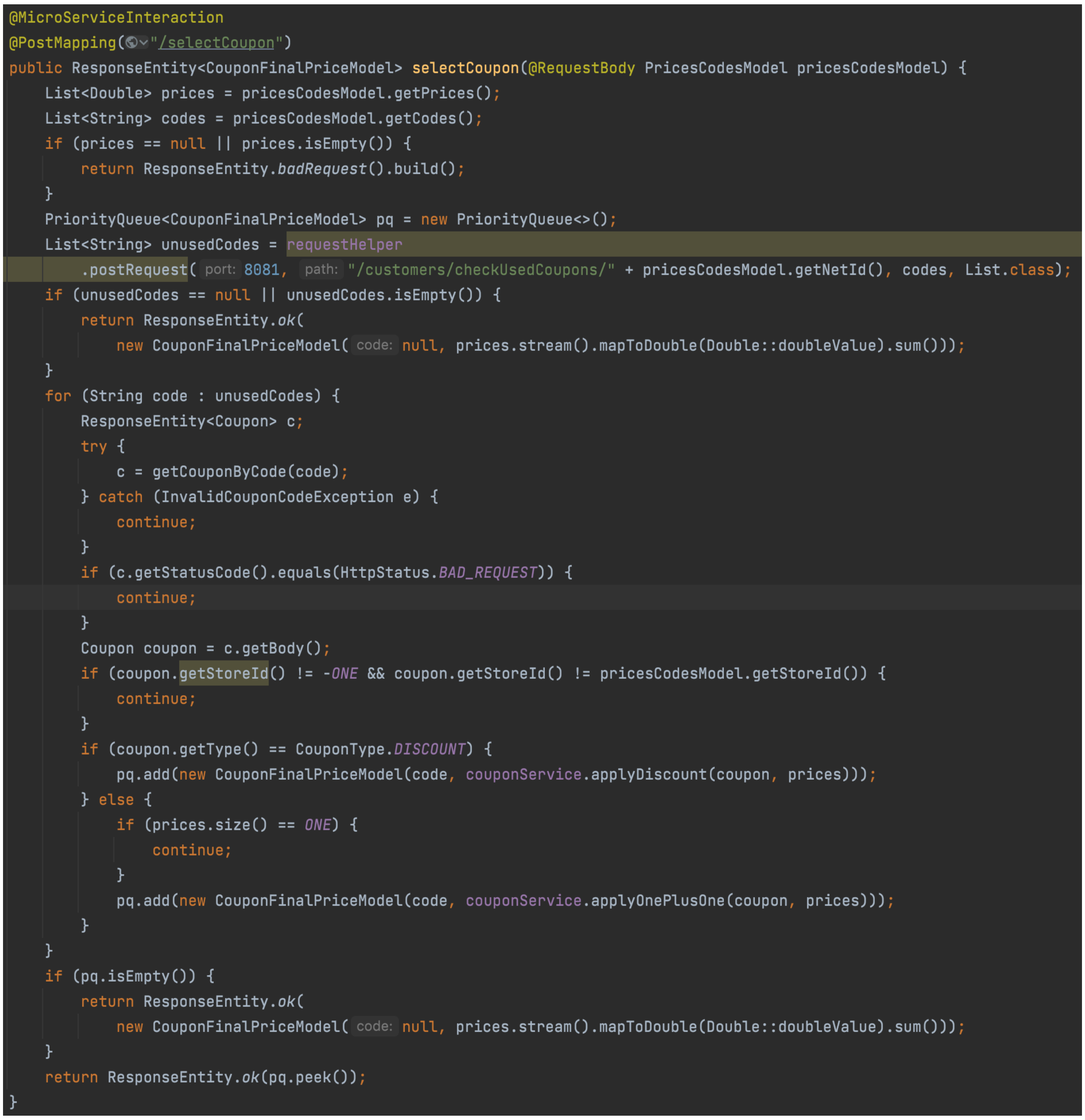
PROCESS

METHODS

Upon reviewing the results of the code analysis using the CodeMR tool, we observed good scores across all six software metrics. This made it challenging to pinpoint which methods and classes needed refactoring, meaning we had to be nitpicky with our decisions. As a result, we decided to focus on the LOC (Lines of Code) and CBO (Coupling Between Objects) metrics for our method refactoring as these seemed to have the most potential for possible improvement within our program. We also checked for Parameter Count on methods where it was relevant.
Based on these metrics, we isolated the parts of our code that were most in need of refactoring. We ended up with the following list of 5 methods:

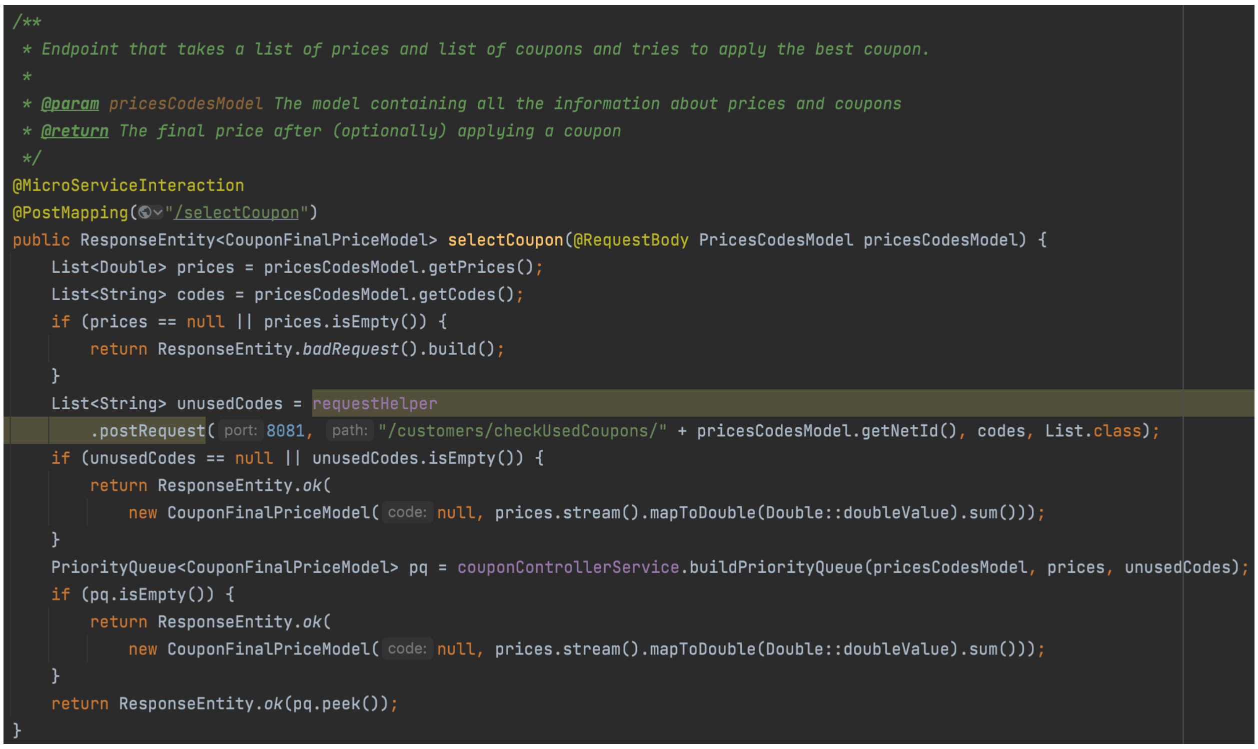
1. **selectCoupon (CouponController)**
    - Metrics Before: High LOC (34), Medium-high CBO (10)
    - Metrics After: Low LOC (17), Low-medium CBO (6)
    - Changes:
      Instead of happening within the endpoint, the iteration over the different eligible coupon codes (ones that have not been used by the user in the past) has been extracted to a new method buildPriorityQueue(). From there, another two methods are called, checkCouponValid(), to assure that the code exists and can be used, and applyCoupon() that gets the final price given to the order according to the coupon's type.

Before refactoring:

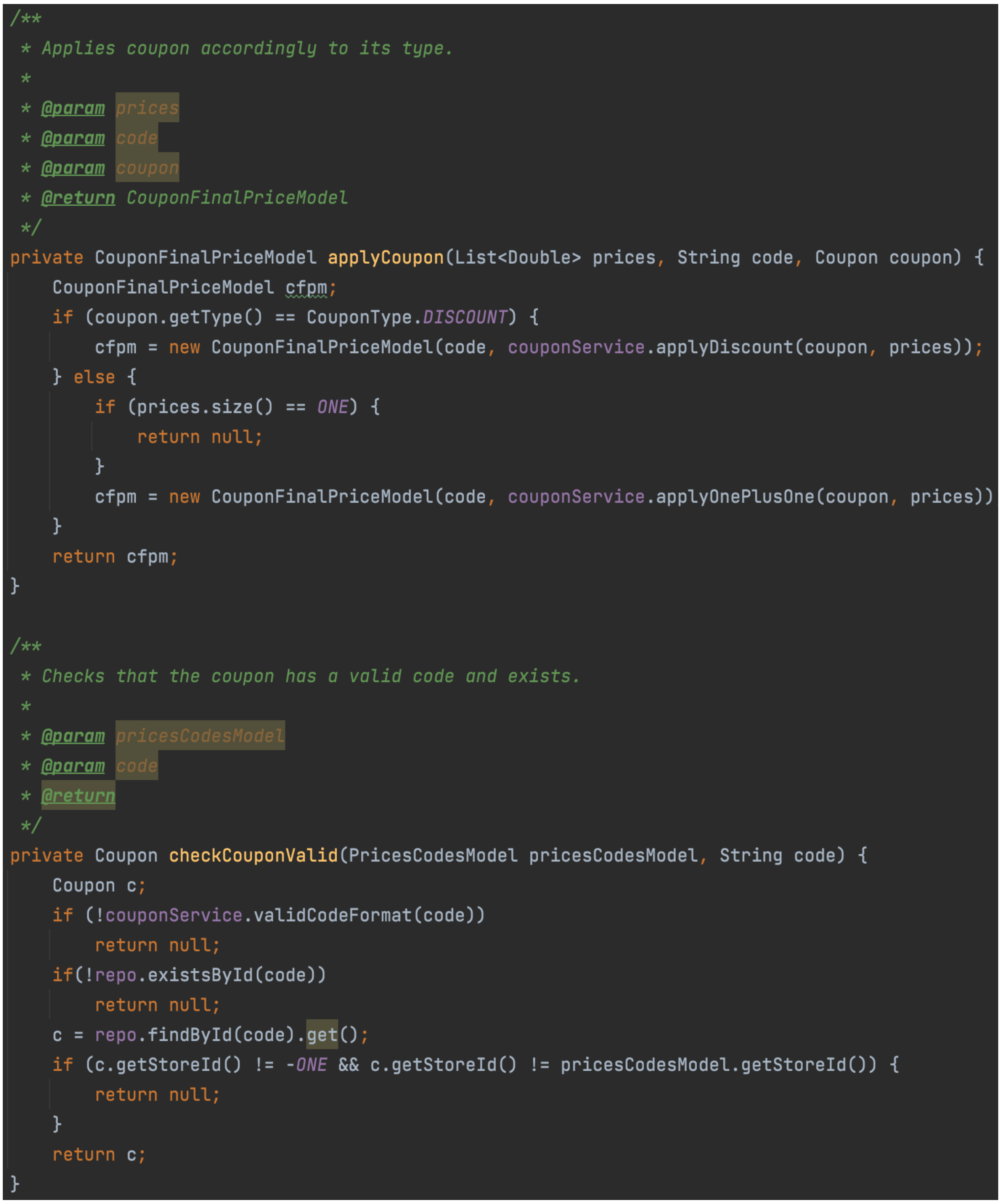


```java
@MicroServiceInteraction
@PostMapping("/selectCoupon")
public ResponseEntity<CouponFinalPriceModel> selectCoupon(@RequestBody PricesCodesModel pricesCodesModel) {
    List<Double> prices = pricesCodesModel.getPrices();
    List<String> codes = pricesCodesModel.getCodes();
    if (prices == null || prices.isEmpty()) {
        return ResponseEntity.badRequest().build();
    }
    PriorityQueue<CouponFinalPriceModel> pq = new PriorityQueue<>();
    List<String> unusedCodes = requestHelper
        .postRequest( port: 8081,  path: "/customers/checkUsedCoupons/" + pricesCodesModel.getNetId(), codes, List.class);
    if (unusedCodes == null || unusedCodes.isEmpty()) {
        return ResponseEntity.ok(
            new CouponFinalPriceModel( code: null, prices.stream().mapToDouble(Double::doubleValue).sum()));
    }
    for (String code : unusedCodes) {
        ResponseEntity<Coupon> c;
        try {
            c = getCouponByCode(code);
        } catch (InvalidCouponCodeException e) {
            continue;
        }
        if (c.getStatusCode().equals(HttpStatus.BAD_REQUEST)) {
            continue;
        }
        Coupon coupon = c.getBody();
        if (coupon.getStoreId() != -ONE && coupon.getStoreId() != pricesCodesModel.getStoreId()) {
            continue;
        }
        if (coupon.getType() == CouponType.DISCOUNT) {
            pq.add(new CouponFinalPriceModel(code, couponService.applyDiscount(coupon, prices)));
        } else {
            if (prices.size() == ONE) {
                continue;
            }
            pq.add(new CouponFinalPriceModel(code, couponService.applyOnePlusOne(coupon, prices)));
        }
    }
    if (pq.isEmpty()) {
        return ResponseEntity.ok(
            new CouponFinalPriceModel( code: null, prices.stream().mapToDouble(Double::doubleValue).sum()));
    }
    return ResponseEntity.ok(pq.peek());
}
```

After refactoring:



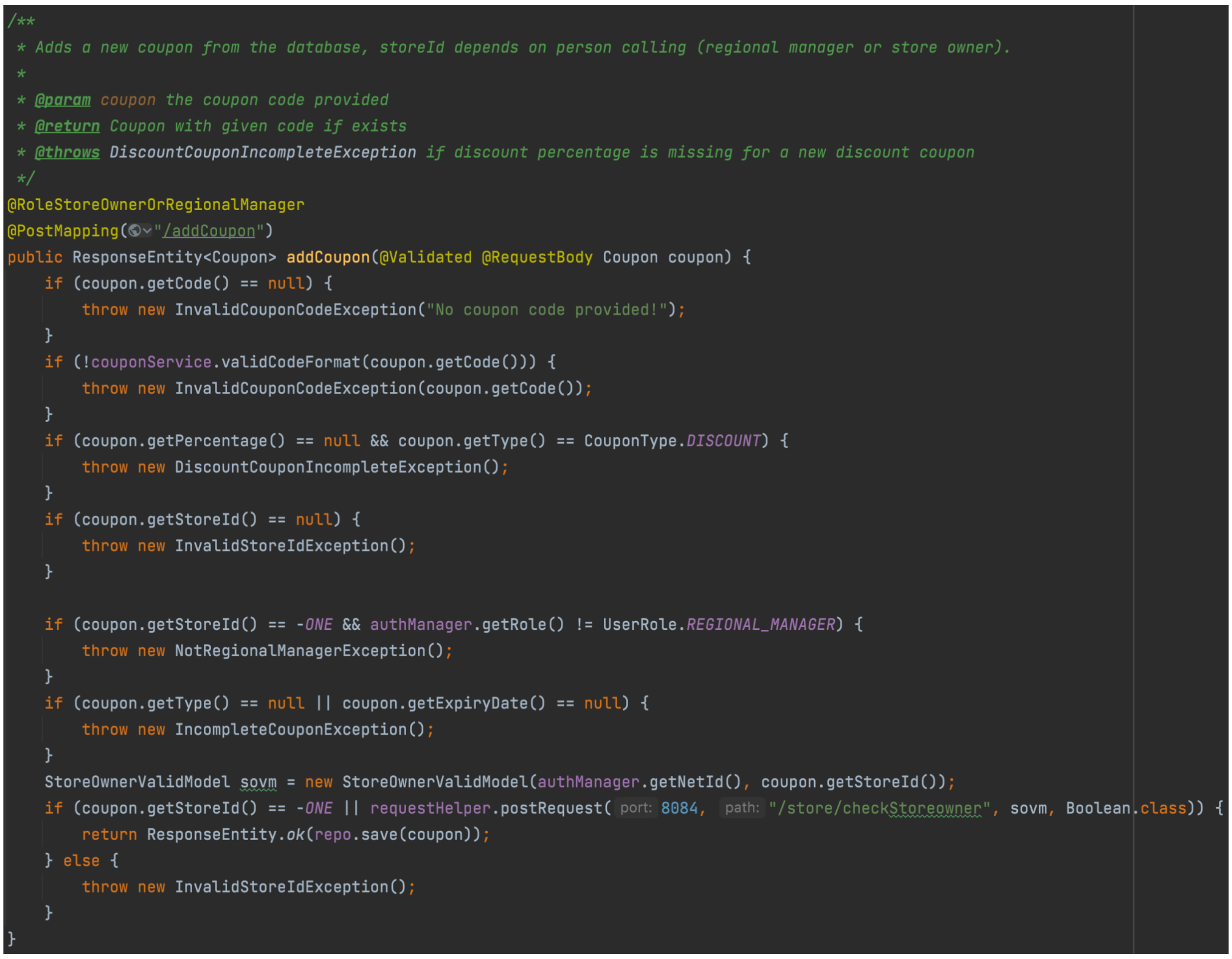
```java
/**
 * Endpoint that takes a list of prices and list of coupons and tries to apply the best coupon.
 *
 * @param pricesCodesModel The model containing all the information about prices and coupons
 * @return The final price after (optionally) applying a coupon
 */
@MicroServiceInteraction
@PostMapping("/selectCoupon")
public ResponseEntity<CouponFinalPriceModel> selectCoupon(@RequestBody PricesCodesModel pricesCodesModel) {
    List<Double> prices = pricesCodesModel.getPrices();
    List<String> codes = pricesCodesModel.getCodes();
    if (prices == null || prices.isEmpty()) {
        return ResponseEntity.badRequest().build();
    }
    List<String> unusedCodes = requestHelper
        .postRequest( port: 8081,  path: "/customers/checkUsedCoupons/" + pricesCodesModel.getNetId(), codes, List.class);
    if (unusedCodes == null || unusedCodes.isEmpty()) {
        return ResponseEntity.ok(
            new CouponFinalPriceModel( code: null, prices.stream().mapToDouble(Double::doubleValue).sum()));
    }
    PriorityQueue<CouponFinalPriceModel> pq = couponControllerService.buildPriorityQueue(pricesCodesModel, prices, unusedCodes);
    if (pq.isEmpty()) {
        return ResponseEntity.ok(
            new CouponFinalPriceModel( code: null, prices.stream().mapToDouble(Double::doubleValue).sum()));
    }
    return ResponseEntity.ok(pq.peek());
}
```



```java
/**
 * Applies coupon accordingly to its type.
 *
 * @param prices
 * @param code
 * @param coupon
 * @return CouponFinalPriceModel
 */
private CouponFinalPriceModel applyCoupon(List<Double> prices, String code, Coupon coupon) {
    CouponFinalPriceModel cfpm;
    if (coupon.getType() == CouponType.DISCOUNT) {
        cfpm = new CouponFinalPriceModel(code, couponService.applyDiscount(coupon, prices));
    } else {
        if (prices.size() == ONE) {
            return null;
        }
        cfpm = new CouponFinalPriceModel(code, couponService.applyOnePlusOne(coupon, prices))
    }
    return cfpm;
}

/**
 * Checks that the coupon has a valid code and exists.
 *
 * @param pricesCodesModel
 * @param code
 * @return
 */
private Coupon checkCouponValid(PricesCodesModel pricesCodesModel, String code) {
    Coupon c;
    if (!couponService.validCodeFormat(code))
        return null;
    if(!repo.existsById(code))
        return null;
    c = repo.findById(code).get();
    if (c.getStoreId() != -ONE && c.getStoreId() != pricesCodesModel.getStoreId()) {
        return null;
    }
    return c;
}
```

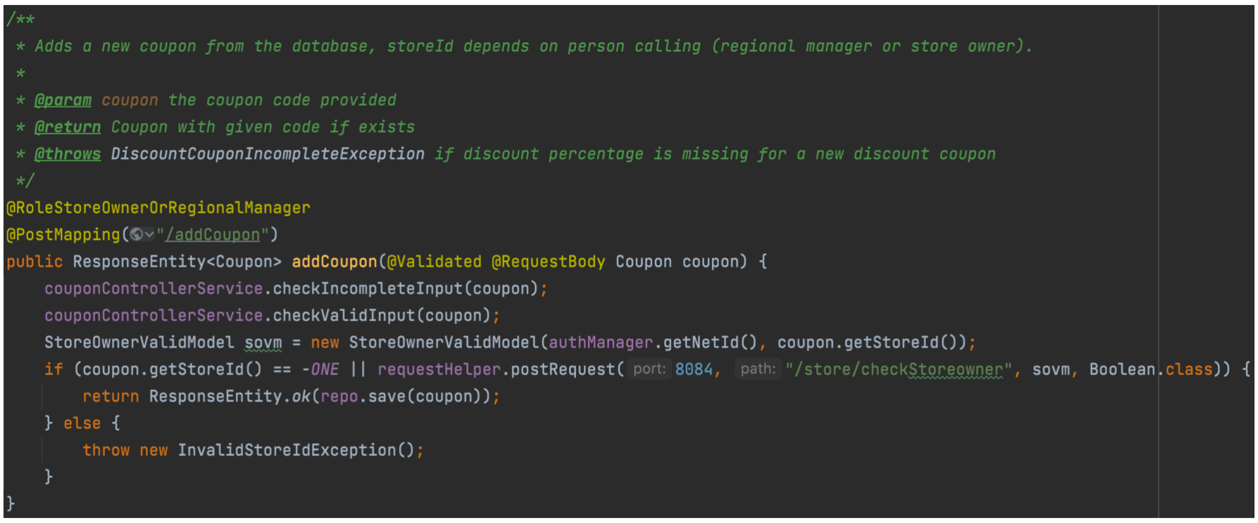
2. **addCoupon (CouponController)**
   - Metrics Before: High LOC (20), Medium-high CBO (14)
   - Metrics After: Low LOC (10), Medium-high CBO (8)
   - Changes:
     Various checks have been extracted to different methods. Specifically, all the checks regarding input fields being incomplete are gathered in one method, where the handling of any related exceptions take place. Additionally, any invalidly formatted input checks have been extracted to a second method, where they are handled respectively.

Before refactoring:

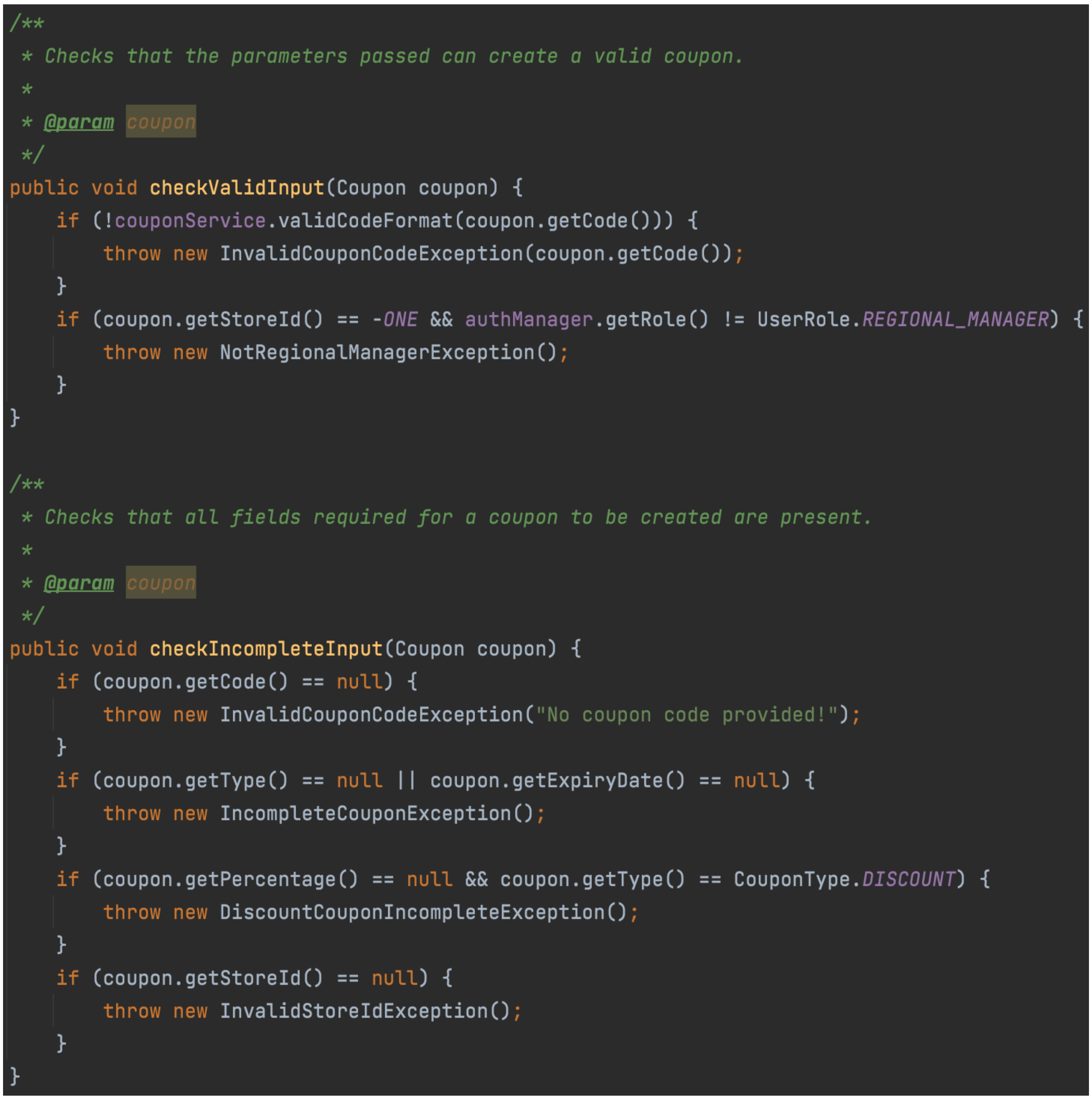


```java
/**
 * Adds a new coupon from the database, storeId depends on person calling (regional manager or store owner).
 *
 * @param coupon the coupon code provided
 * @return Coupon with given code if exists
 * @throws DiscountCouponIncompleteException if discount percentage is missing for a new discount coupon
 */
@RoleStoreOwnerOrRegionalManager
@PostMapping("/addCoupon")
public ResponseEntity<Coupon> addCoupon(@Validated @RequestBody Coupon coupon) {
    if (coupon.getCode() == null) {
        throw new InvalidCouponCodeException("No coupon code provided!");
    }
    if (!couponService.validCodeFormat(coupon.getCode())) {
        throw new InvalidCouponCodeException(coupon.getCode());
    }
    if (coupon.getPercentage() == null && coupon.getType() == CouponType.DISCOUNT) {
        throw new DiscountCouponIncompleteException();
    }
    if (coupon.getStoreId() == null) {
        throw new InvalidStoreIdException();
    }

    if (coupon.getStoreId() == -ONE && authManager.getRole() != UserRole.REGIONAL_MANAGER) {
        throw new NotRegionalManagerException();
    }
    if (coupon.getType() == null || coupon.getExpiryDate() == null) {
        throw new IncompleteCouponException();
    }
    StoreOwnerValidModel sovm = new StoreOwnerValidModel(authManager.getNetId(), coupon.getStoreId());
    if (coupon.getStoreId() == -ONE || requestHelper.postRequest( port: 8084,  path: "/store/checkStoreowner", sovm, Boolean.class)) {
        return ResponseEntity.ok(repo.save(coupon));
    } else {
        throw new InvalidStoreIdException();
    }
}
```

After refactoring:



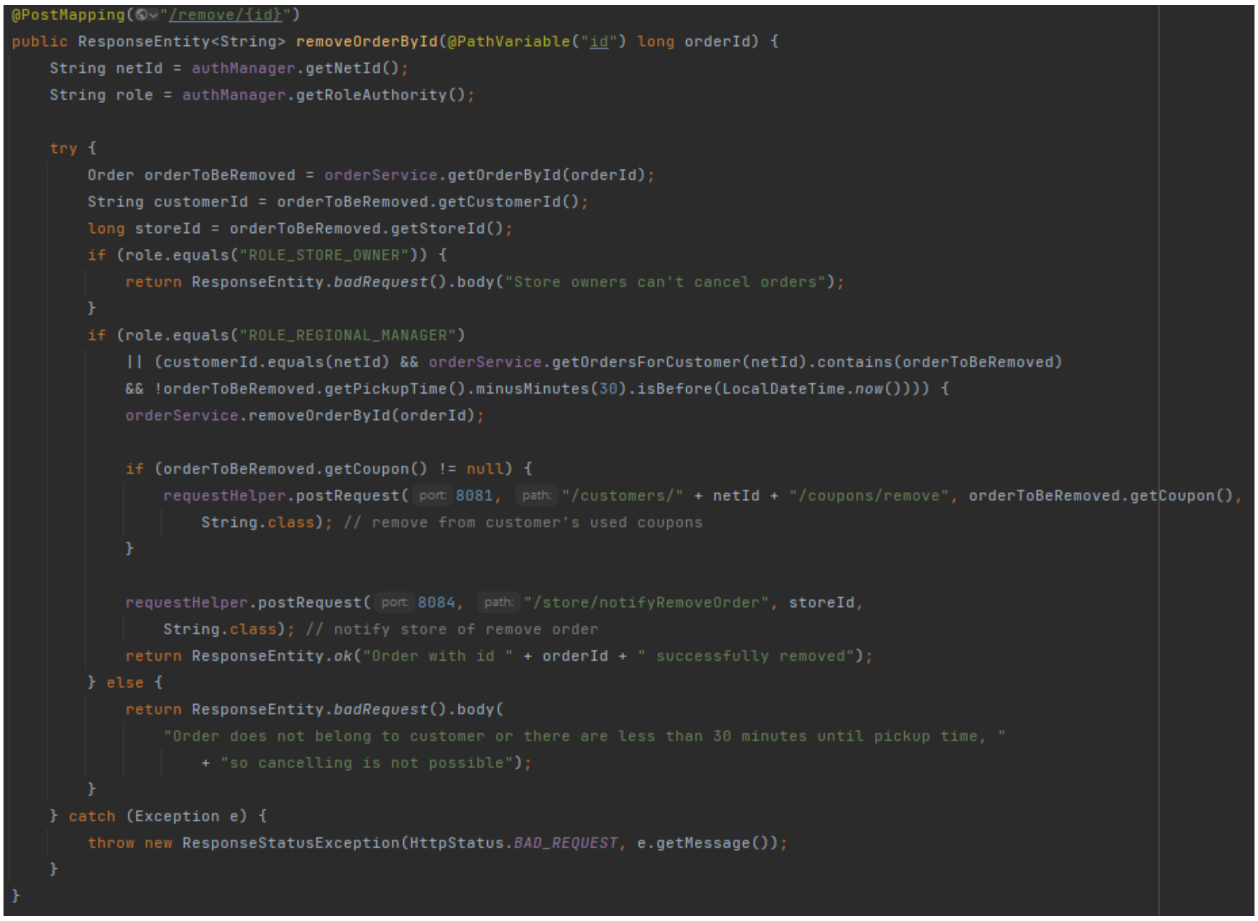
```java
/**
 * Adds a new coupon from the database, storeId depends on person calling (regional manager or store owner).
 *
 * @param coupon the coupon code provided
 * @return Coupon with given code if exists
 * @throws DiscountCouponIncompleteException if discount percentage is missing for a new discount coupon
 */
@RoleStoreOwnerOrRegionalManager
@PostMapping("/addCoupon")
public ResponseEntity<Coupon> addCoupon(@Validated @RequestBody Coupon coupon) {
    couponControllerService.checkIncompleteInput(coupon);
    couponControllerService.checkValidInput(coupon);
    StoreOwnerValidModel sovm = new StoreOwnerValidModel(authManager.getNetId(), coupon.getStoreId());
    if (coupon.getStoreId() == -ONE || requestHelper.postRequest( port: 8084,  path: "/store/checkStoreowner", sovm, Boolean.class)) {
        return ResponseEntity.ok(repo.save(coupon));
    } else {
        throw new InvalidStoreIdException();
    }
}
```



```java
/**
 * Checks that the parameters passed can create a valid coupon.
 *
 * @param coupon
 */
public void checkValidInput(Coupon coupon) {
    if (!couponService.validCodeFormat(coupon.getCode())) {
        throw new InvalidCouponCodeException(coupon.getCode());
    }
    if (coupon.getStoreId() == -ONE && authManager.getRole() != UserRole.REGIONAL_MANAGER) {
        throw new NotRegionalManagerException();
    }
}

/**
 * Checks that all fields required for a coupon to be created are present.
 *
 * @param coupon
 */
public void checkIncompleteInput(Coupon coupon) {
    if (coupon.getCode() == null) {
        throw new InvalidCouponCodeException("No coupon code provided!");
    }
    if (coupon.getType() == null || coupon.getExpiryDate() == null) {
        throw new IncompleteCouponException();
    }
    if (coupon.getPercentage() == null && coupon.getType() == CouponType.DISCOUNT) {
        throw new DiscountCouponIncompleteException();
    }
    if (coupon.getStoreId() == null) {
        throw new InvalidStoreIdException();
    }
}
```

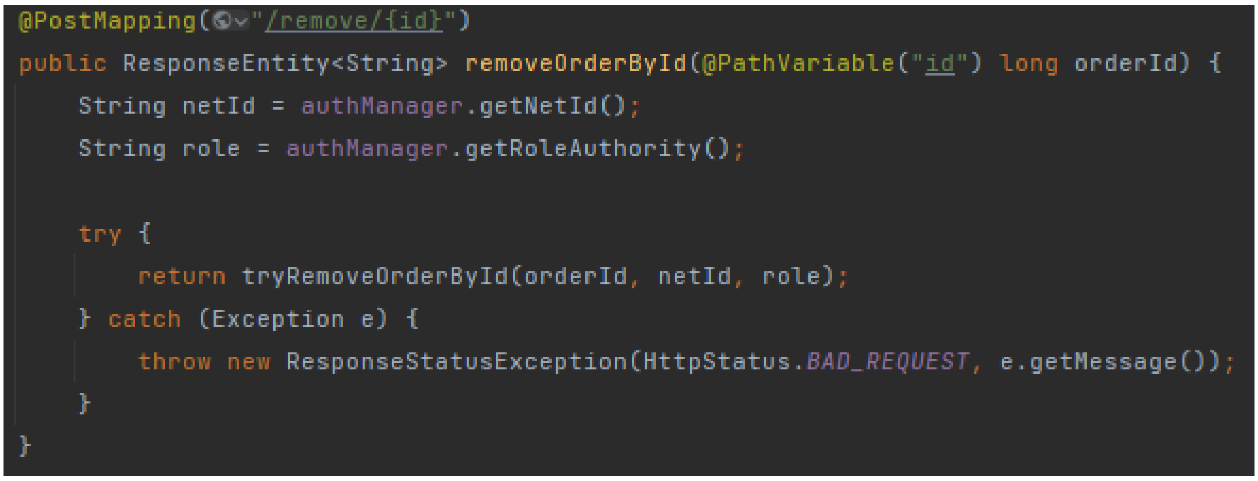
3. **removeOrderById (OrderController)**
   - Metrics Before: High LOC (26), Medium-high CBO (8)
   - Metrics After: Low LOC (8), Low-Medium CBO (4)
   - Changes:
     Extracted most of the code of this method into three separate methods, namely tryRemoveOrderById() which in turn calls an additional two new methods called isOrderRemovable() and removeOrderAndCoupon(). This spreads out the functionality of removeOrderById() in order to lower LOC and CBO.

Before Refactoring:

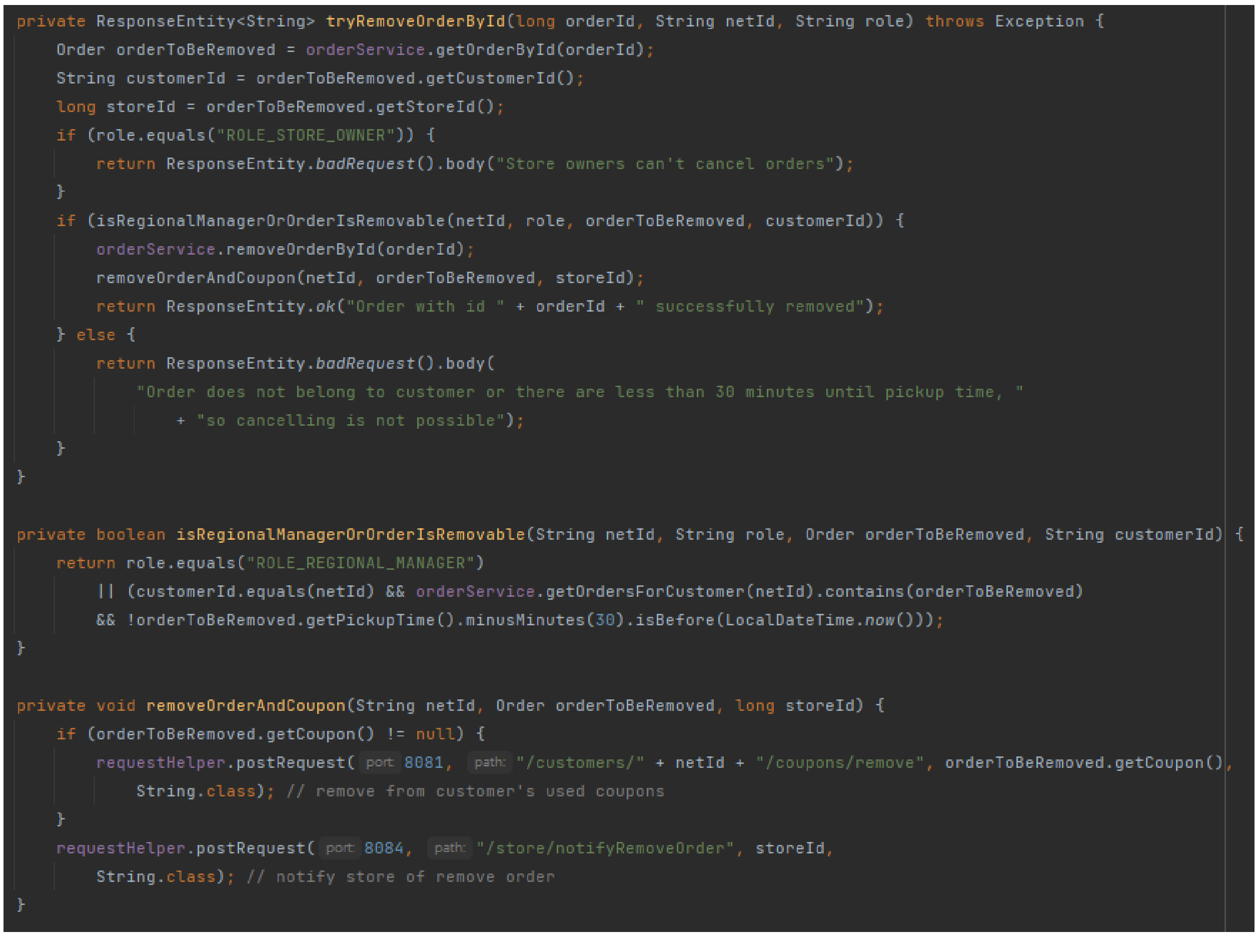


```java
@PostMapping("/remove/{id}")
public ResponseEntity<String> removeOrderById(@PathVariable("id") long orderId) {
    String netId = authManager.getNetId();
    String role = authManager.getRoleAuthority();

    try {
        Order orderToBeRemoved = orderService.getOrderById(orderId);
        String customerId = orderToBeRemoved.getCustomerId();
        long storeId = orderToBeRemoved.getStoreId();
        if (role.equals("ROLE_STORE_OWNER")) {
            return ResponseEntity.badRequest().body("Store owners can't cancel orders");
        }
        if (role.equals("ROLE_REGIONAL_MANAGER")
            || (customerId.equals(netId) && orderService.getOrdersForCustomer(netId).contains(orderToBeRemoved)
            && !orderToBeRemoved.getPickupTime().minusMinutes(30).isBefore(LocalDateTime.now()))) {
            orderService.removeOrderById(orderId);

            if (orderToBeRemoved.getCoupon() != null) {
                requestHelper.postRequest( port: 8081,  path: "/customers/" + netId + "/coupons/remove", orderToBeRemoved.getCoupon(),
                    String.class); // remove from customer's used coupons
            }

            requestHelper.postRequest( port: 8084,  path: "/store/notifyRemoveOrder", storeId,
                String.class); // notify store of remove order
            return ResponseEntity.ok("Order with id " + orderId + " successfully removed");
        } else {
            return ResponseEntity.badRequest().body(
                "Order does not belong to customer or there are less than 30 minutes until pickup time, "
                    + "so cancelling is not possible");
        }
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    }
}
```

After Refactoring:



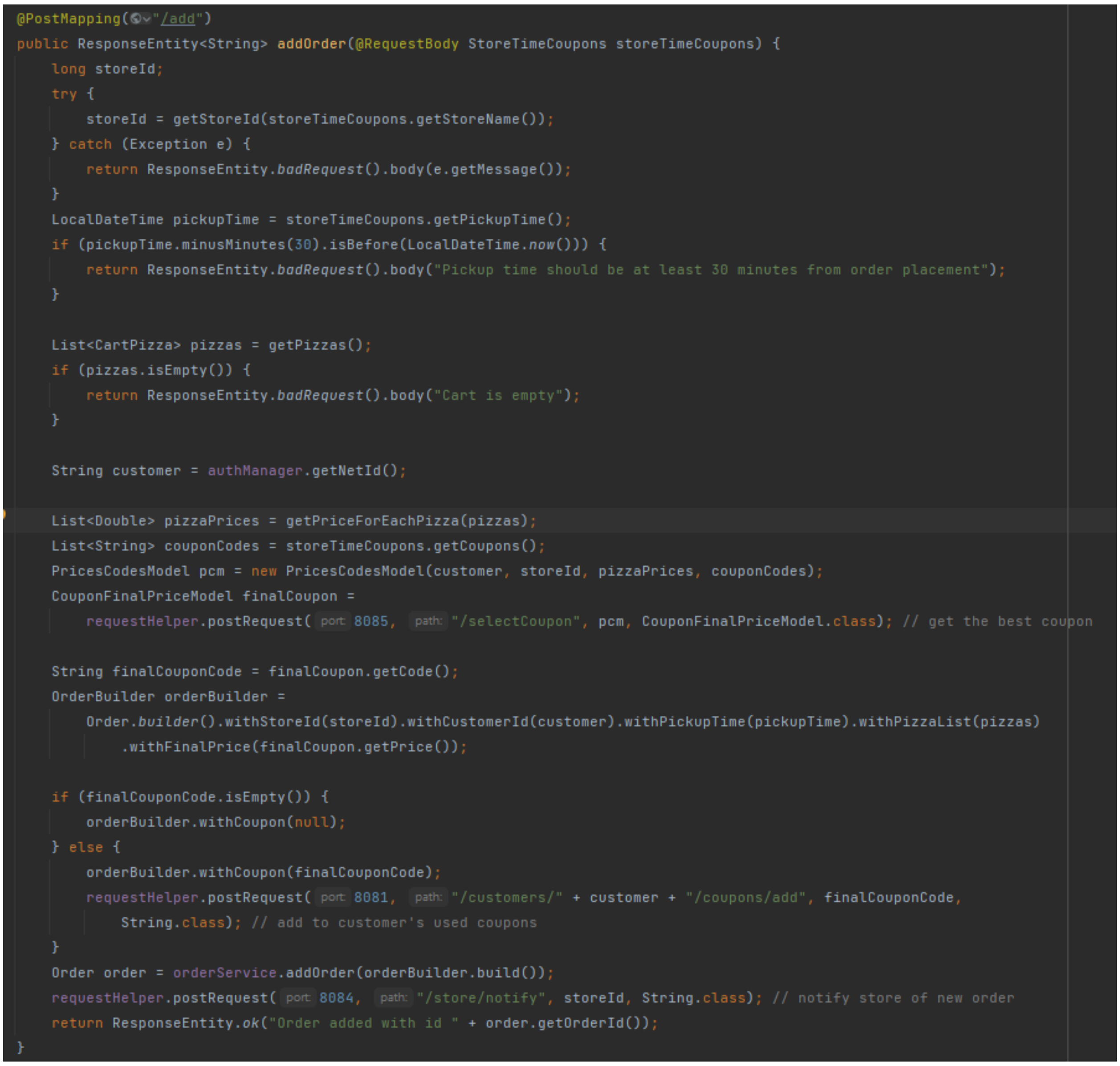
```java
@PostMapping("/remove/{id}")
public ResponseEntity<String> removeOrderById(@PathVariable("id") long orderId) {
    String netId = authManager.getNetId();
    String role = authManager.getRoleAuthority();

    try {
        return tryRemoveOrderById(orderId, netId, role);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    }
}
```



```java
private ResponseEntity<String> tryRemoveOrderById(long orderId, String netId, String role) throws Exception {
    Order orderToBeRemoved = orderService.getOrderById(orderId);
    String customerId = orderToBeRemoved.getCustomerId();
    long storeId = orderToBeRemoved.getStoreId();
    if (role.equals("ROLE_STORE_OWNER")) {
        return ResponseEntity.badRequest().body("Store owners can't cancel orders");
    }
    if (isRegionalManagerOrOrderIsRemovable(netId, role, orderToBeRemoved, customerId)) {
        orderService.removeOrderById(orderId);
        removeOrderAndCoupon(netId, orderToBeRemoved, storeId);
        return ResponseEntity.ok("Order with id " + orderId + " successfully removed");
    } else {
        return ResponseEntity.badRequest().body(
            "Order does not belong to customer or there are less than 30 minutes until pickup time, "
                + "so cancelling is not possible");
    }
}

private boolean isRegionalManagerOrOrderIsRemovable(String netId, String role, Order orderToBeRemoved, String customerId) {
    return role.equals("ROLE_REGIONAL_MANAGER")
        || (customerId.equals(netId) && orderService.getOrdersForCustomer(netId).contains(orderToBeRemoved)
        && !orderToBeRemoved.getPickupTime().minusMinutes(30).isBefore(LocalDateTime.now()));
}

private void removeOrderAndCoupon(String netId, Order orderToBeRemoved, long storeId) {
    if (orderToBeRemoved.getCoupon() != null) {
        requestHelper.postRequest( port: 8081,  path: "/customers/" + netId + "/coupons/remove", orderToBeRemoved.getCoupon(),
            String.class); // remove from customer's used coupons
    }
    requestHelper.postRequest( port: 8084,  path: "/store/notifyRemoveOrder", storeId,
        String.class); // notify store of remove order
}
```

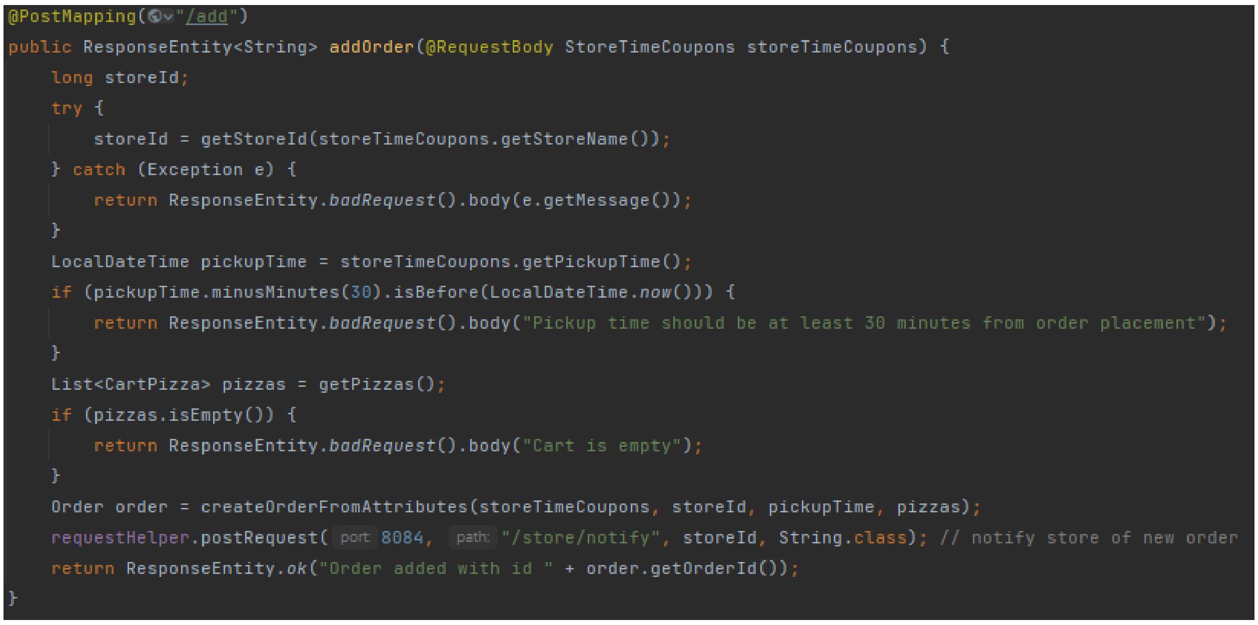
4. **addOrder (OrderController)**
   - Metrics Before: High LOC (32), Medium-High CBO (10)
   - Metrics After: Low LOC (16), Low-Medium CBO (5)
   - Changes:
     Split the functionality of the method into two different methods, in order to avoid having a single, unnecessarily lengthy method.

Before refactoring:

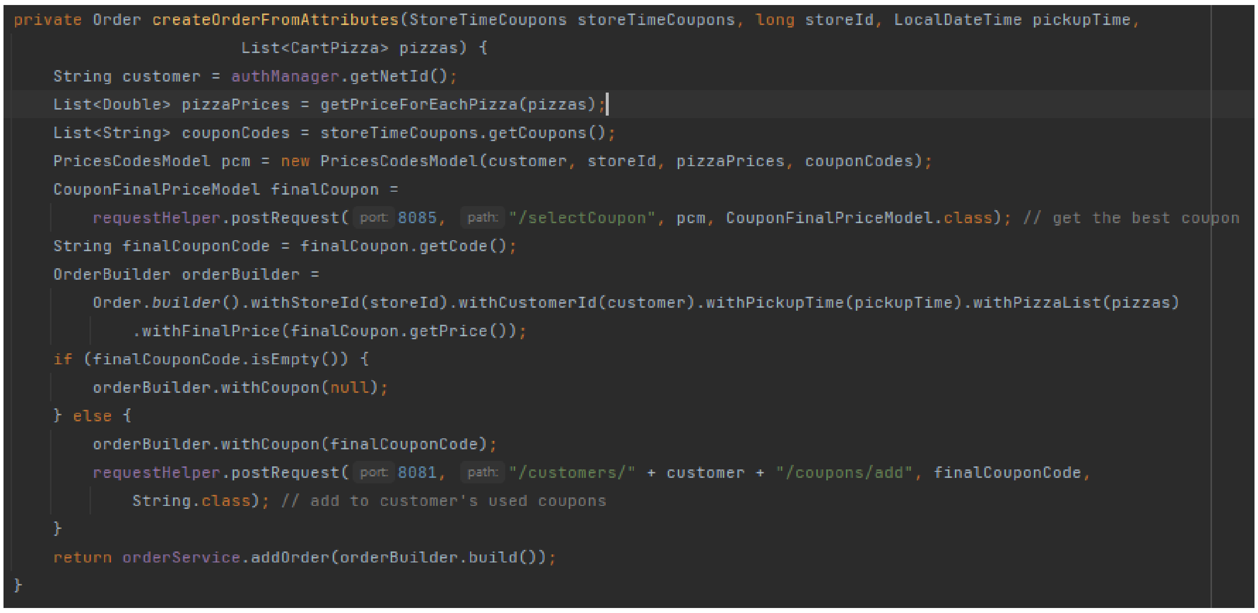


```java
@PostMapping("/add")
public ResponseEntity<String> addOrder(@RequestBody StoreTimeCoupons storeTimeCoupons) {
    long storeId;
    try {
        storeId = getStoreId(storeTimeCoupons.getStoreName());
    } catch (Exception e) {
        return ResponseEntity.badRequest().body(e.getMessage());
    }
    LocalDateTime pickupTime = storeTimeCoupons.getPickupTime();
    if (pickupTime.minusMinutes(30).isBefore(LocalDateTime.now())) {
        return ResponseEntity.badRequest().body("Pickup time should be at least 30 minutes from order placement");
    }

    List<CartPizza> pizzas = getPizzas();
    if (pizzas.isEmpty()) {
        return ResponseEntity.badRequest().body("Cart is empty");
    }

    String customer = authManager.getNetId();

    List<Double> pizzaPrices = getPriceForEachPizza(pizzas);
    List<String> couponCodes = storeTimeCoupons.getCoupons();
    PricesCodesModel pcm = new PricesCodesModel(customer, storeId, pizzaPrices, couponCodes);
    CouponFinalPriceModel finalCoupon =
        requestHelper.postRequest( port: 8085,  path: "/selectCoupon", pcm, CouponFinalPriceModel.class); // get the best coupon

    String finalCouponCode = finalCoupon.getCode();
    OrderBuilder orderBuilder =
        Order.builder().withStoreId(storeId).withCustomerId(customer).withPickupTime(pickupTime).withPizzaList(pizzas)
            .withFinalPrice(finalCoupon.getPrice());

    if (finalCouponCode.isEmpty()) {
        orderBuilder.withCoupon(null);
    } else {
        orderBuilder.withCoupon(finalCouponCode);
        requestHelper.postRequest( port: 8081,  path: "/customers/" + customer + "/coupons/add", finalCouponCode,
            String.class); // add to customer's used coupons
    }
    Order order = orderService.addOrder(orderBuilder.build());
    requestHelper.postRequest( port: 8084,  path: "/store/notify", storeId, String.class); // notify store of new order
    return ResponseEntity.ok("Order added with id " + order.getOrderId());
}
```

After refactoring:



```java
@PostMapping("/add")
public ResponseEntity<String> addOrder(@RequestBody StoreTimeCoupons storeTimeCoupons) {
    long storeId;
    try {
        storeId = getStoreId(storeTimeCoupons.getStoreName());
    } catch (Exception e) {
        return ResponseEntity.badRequest().body(e.getMessage());
    }
    LocalDateTime pickupTime = storeTimeCoupons.getPickupTime();
    if (pickupTime.minusMinutes(30).isBefore(LocalDateTime.now())) {
        return ResponseEntity.badRequest().body("Pickup time should be at least 30 minutes from order placement");
    }
    List<CartPizza> pizzas = getPizzas();
    if (pizzas.isEmpty()) {
        return ResponseEntity.badRequest().body("Cart is empty");
    }
    Order order = createOrderFromAttributes(storeTimeCoupons, storeId, pickupTime, pizzas);
    requestHelper.postRequest( port: 8084,  path: "/store/notify", storeId, String.class); // notify store of new order
    return ResponseEntity.ok("Order added with id " + order.getOrderId());
}
```

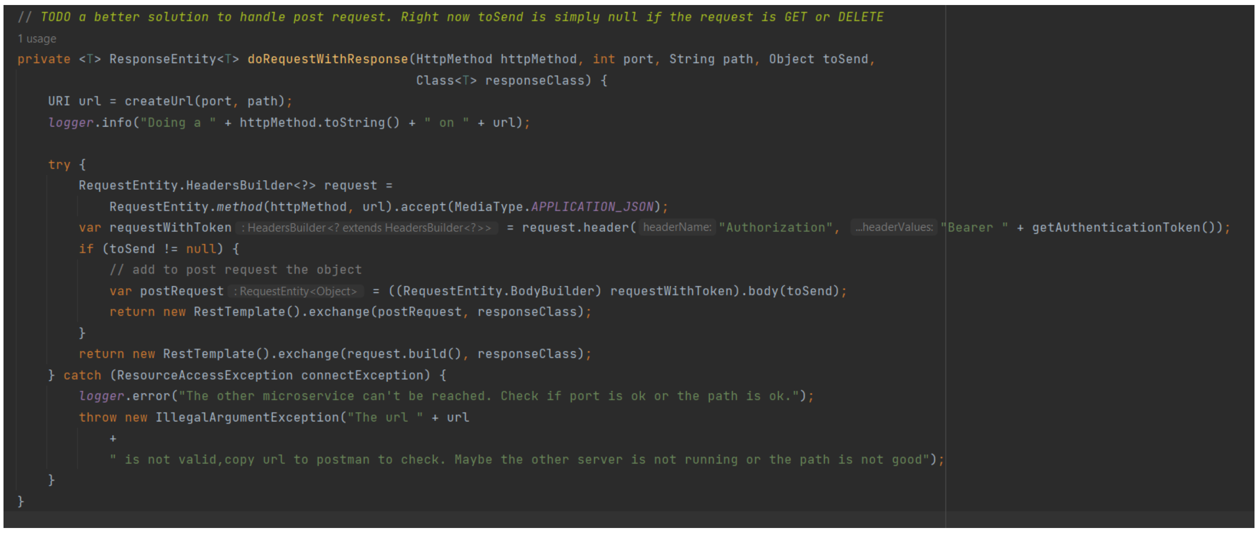


```java
private Order createOrderFromAttributes(StoreTimeCoupons storeTimeCoupons, long storeId, LocalDateTime pickupTime,
                                        List<CartPizza> pizzas) {
    String customer = authManager.getNetId();
    List<Double> pizzaPrices = getPriceForEachPizza(pizzas);
    List<String> couponCodes = storeTimeCoupons.getCoupons();
    PricesCodesModel pcm = new PricesCodesModel(customer, storeId, pizzaPrices, couponCodes);
    CouponFinalPriceModel finalCoupon =
        requestHelper.postRequest( port: 8085,  path: "/selectCoupon", pcm, CouponFinalPriceModel.class); // get the best coupon
    String finalCouponCode = finalCoupon.getCode();
    OrderBuilder orderBuilder =
        Order.builder().withStoreId(storeId).withCustomerId(customer).withPickupTime(pickupTime).withPizzaList(pizzas)
            .withFinalPrice(finalCoupon.getPrice());
    if (finalCouponCode.isEmpty()) {
        orderBuilder.withCoupon(null);
    } else {
        orderBuilder.withCoupon(finalCouponCode);
        requestHelper.postRequest( port: 8081,  path: "/customers/" + customer + "/coupons/add", finalCouponCode,
            String.class); // add to customer's used coupons
    }
    return orderService.addOrder(orderBuilder.build());
}
```

5. doRequestWithResponse (RequestHelper)

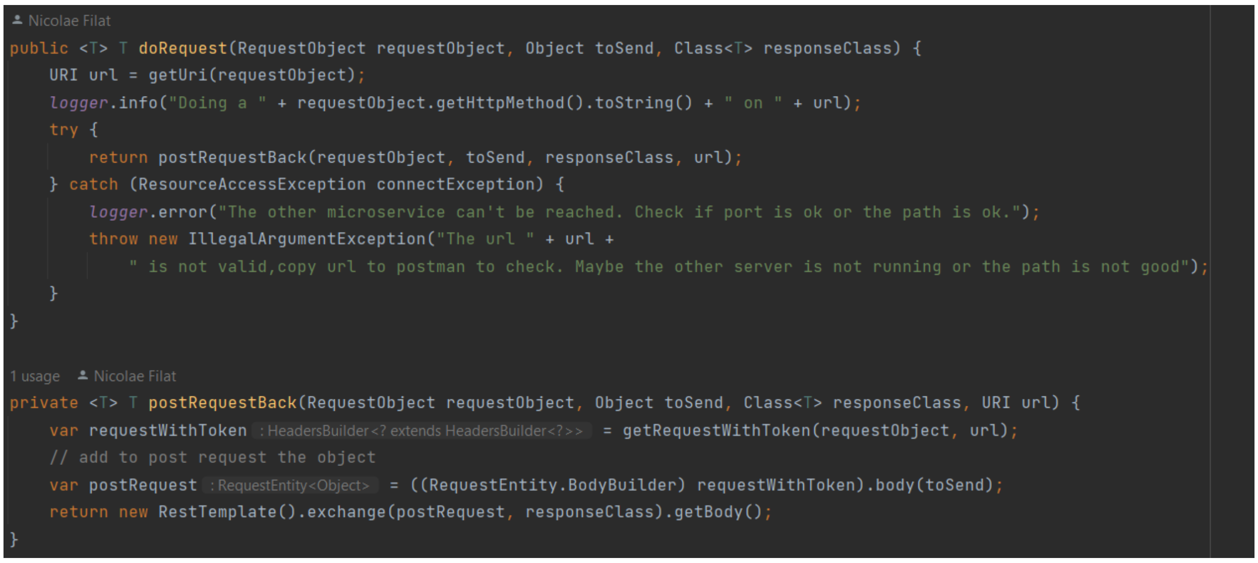
- Metrics Before: LOC (17), Medium-high CBO (9), Parameter count: 5
- Metrics After : LOC(10), CBO(7), Parameter count : 3
- Changes:
  Refactored the request type, port and path to a different class and removed the methods of getRequest(), postRequest(), deleteRequest(). I refactored this because we could just use the refactored method doRequest() that works. I basically splitted the big method into smaller methods do only build the request with the token and only do the POST request.
  I refactored the parameters to go to another class RequestObject and made the methods public so that users can directly interact with the class without using getRequest() or postRequest().

Before refactoring:

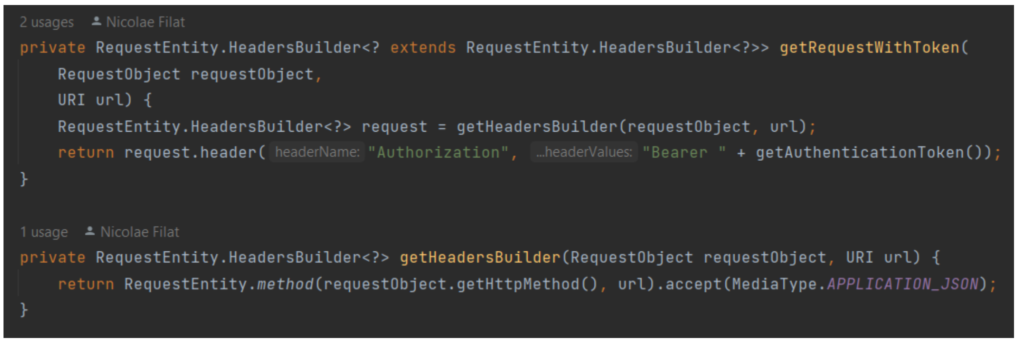


```java
// TODO a better solution to handle post request. Right now toSend is simply null if the request is GET or DELETE
1 usage
private <T> ResponseEntity<T> doRequestWithResponse(HttpMethod httpMethod, int port, String path, Object toSend,
                                                    Class<T> responseClass) {
    URI url = createUrl(port, path);
    logger.info("Doing a " + httpMethod.toString() + " on " + url);

    try {
        RequestEntity.HeadersBuilder<?> request =
            RequestEntity.method(httpMethod, url).accept(MediaType.APPLICATION_JSON);
        var requestWithToken : HeadersBuilder<? extends HeadersBuilder<?>> = request.header( headerName: "Authorization", ...headerValues: "Bearer " + getAuthenticationToken());
        if (toSend != null) {
            // add to post request the object
            var postRequest : RequestEntity<Object> = ((RequestEntity.BodyBuilder) requestWithToken).body(toSend);
            return new RestTemplate().exchange(postRequest, responseClass);
        }
        return new RestTemplate().exchange(request.build(), responseClass);
    } catch (ResourceAccessException connectException) {
        logger.error("The other microservice can't be reached. Check if port is ok or the path is ok.");
        throw new IllegalArgumentException("The url " + url
            +
            " is not valid,copy url to postman to check. Maybe the other server is not running or the path is not good");
    }
}
```

After refactoring:



```java
Nicolae Filat
public <T> T doRequest(RequestObject requestObject, Object toSend, Class<T> responseClass) {
    URI url = getUri(requestObject);
    logger.info("Doing a " + requestObject.getHttpMethod().toString() + " on " + url);
    try {
        return postRequestBack(requestObject, toSend, responseClass, url);
    } catch (ResourceAccessException connectException) {
        logger.error("The other microservice can't be reached. Check if port is ok or the path is ok.");
        throw new IllegalArgumentException("The url " + url +
            " is not valid,copy url to postman to check. Maybe the other server is not running or the path is not good");
    }
}

1 usage  Nicolae Filat
private <T> T postRequestBack(RequestObject requestObject, Object toSend, Class<T> responseClass, URI url) {
    var requestWithToken : HeadersBuilder<? extends HeadersBuilder<?>> = getRequestWithToken(requestObject, url);
    // add to post request the object
    var postRequest : RequestEntity<Object> = ((RequestEntity.BodyBuilder) requestWithToken).body(toSend);
    return new RestTemplate().exchange(postRequest, responseClass).getBody();
}
```



```
2 usages  Nicolae Filat
private RequestEntity.HeadersBuilder<? extends RequestEntity.HeadersBuilder<?>> getRequestWithToken(
    RequestObject requestObject,
    URI url) {
    RequestEntity.HeadersBuilder<?> request = getHeadersBuilder(requestObject, url);
    return request.header( headerName: "Authorization",  ...headerValues: "Bearer " + getAuthenticationToken());
}

1 usage  Nicolae Filat
private RequestEntity.HeadersBuilder<?> getHeadersBuilder(RequestObject requestObject, URI url) {
    return RequestEntity.method(requestObject.getHttpMethod(), url).accept(MediaType.APPLICATION_JSON);
}
```

CLASSES

In terms of refactoring Classes, we analyzed all six of the most significant software metrics for each class in our program and, again, similarly to the Methods, struggled to decide on which classes needed refactoring due to their respective metrics already being at a satisfactory level. Taking this into consideration, we chose the ones that had the highest metrics regardless and ended up isolating the following 5 Classes for refactoring:

CLASSES:

1. **OrderController**
   - Metrics Before: WMC (34), CBO (15), RFC (98), LCOM (0.333), DIT (1), NOC (0)
   - Metrics After: WMC (22), CBO (14), RFC (57), LCOM (0.519), DIT (1), NOC (0)
   - Changes:
     The metrics of this class were very high, especially the RFC. We lowered the values of the metrics by using move method refactoring on the following methods: tryRemove, getAllOrders, getOrderById, getOrderPrice. The methods were then moved to the OrderService which takes care of the interaction with the database. For the tryRemove only a smaller part of the method was moved to OrderService.

2. **CouponController**
   - Metrics Before: WMC (37), CBO (18), RFC (45), LCOM (0.667), DIT (1), NOC (0)
   - Metrics After: WMC (15), CBO (13), RFC (43), LCOM (0.583), DIT (1), NOC (0)
   - Changes:
     The coupon controller was a complex class mainly due to the relatively large amount of endpoints it contained, and the size and complexity of specifically two of them: the selectCoupon() and addCoupon() methods. After refactoring those methods as explained earlier, the class had been improved, but was still complex. That was caused by the fact that it now had a variety of helper methods to those two endpoints, which increased the values for the WMC and LCOM metrics. Therefore, to decrease those, and the rest even more, it was decided to create a brand new class called CouponControllerService, that could contain all those services for the endpoints in CouponController. To implement the refactoring, moving methods was required, while also accounting for the injection of a couponControllerService instance in the constructor of the controller class.

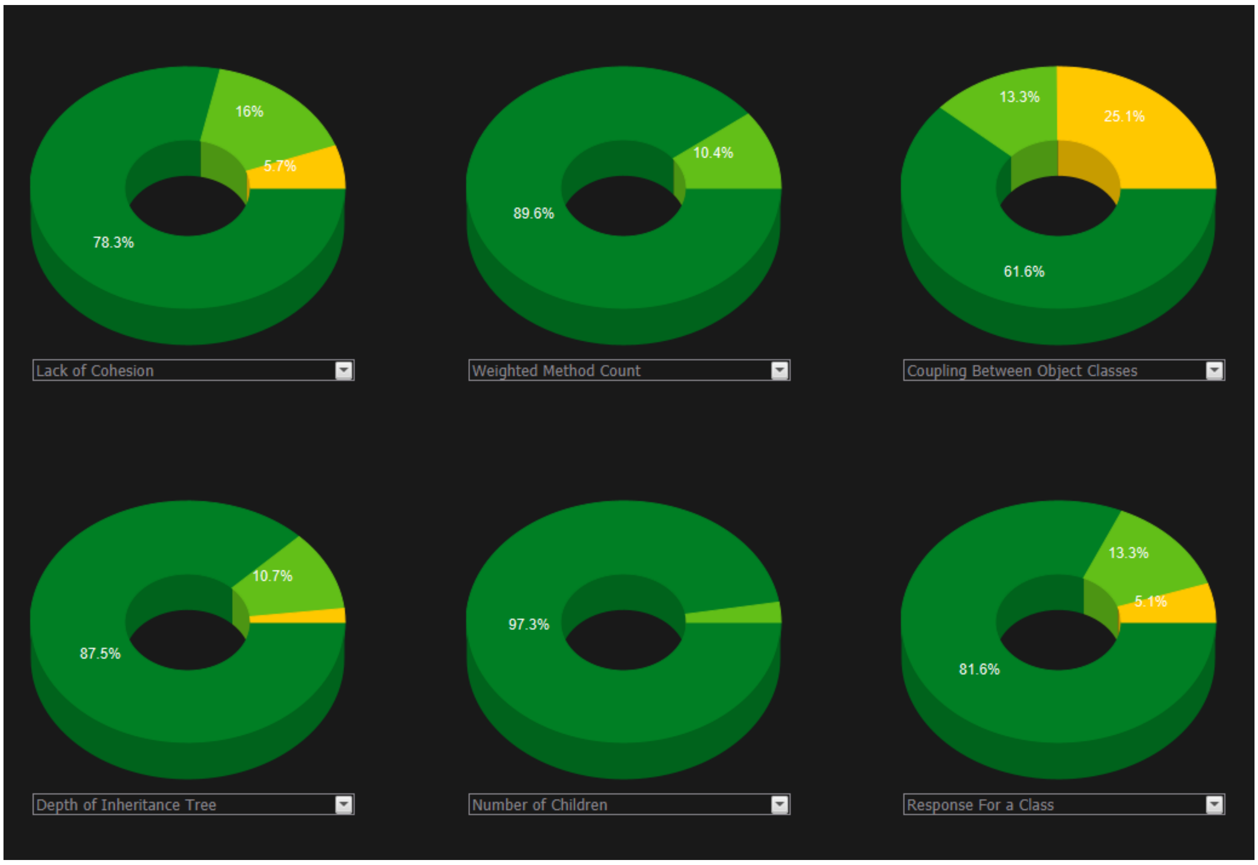
3. **CartController**
   - Metrics Before: WMC (27), CBO (17), RFC (95), LCOM (0.848), DIT (1), NOC (0)
   - Metrics After: WMC (14), CBO (13), RFC (43), LCOM (0.5), DIT (1), NOC (0)
   - Changes:
     The cart controller had a high coupling and low cohesion between methods. The idea was to group the methods that work on similar attributes into a service class. Helper methods like requireNotEmpty(), getCustomPizzza(), getDefaultPizza(), getCartFromSessionId(), assertInCart() I moved to the CartService class because they are mostly helper methods and do not have real business logic.
     Because we moved the methods to the service class the dependencies also changed because some dependencies could be moved to the service class (RequestHelper and DefaultPizzaRepository). This change decreased the coupling (CBO).

4. **CustomerController**
   - Metrics Before: WMC (11), CBO (5), RFC (37), LCOM (0.45), DIT (1), NOC (0)
   - Metrics After: WMC (11), CBO (4), RFC (23), LCOM (0.45), DIT (1), NOC (0)
   - Changes:
     The customer class had a high Response For Class stat. A small change to this class' addCustomer() method in which lines that were unnecessarily in the controller were moved to the corresponding service class managed to bring down that RFC stat, and in the process also lowered the CBO stat.

5. **AuthenticationController**
   - Metrics Before: WMC (5), CBO (18), RFC (65), LCOM (1), DIT (1) NOC (0)
   - Metrics After: WMC (3), CBO (11), RFC (65), LCOM (1), DIT (1) NOC (0)
   - Changes:
     Refactoring the AuthenticationController was hard to refactor and as you can see in the changes of the metrics we were only able to slightly decrease the WMC and the CBO. There were a couple of things we could do to decrease the CBO of the class.
     First of all, there were two try-catch statements in the authenticate endpoint to catch exceptions in case the credentials were incorrect, or the user's account was disabled. What we did is remove these two try-catches and create a new class (AuthenticationExceptionHandler) in which we use the @ExceptionHandler annotation that Spring Boot provides. This allowed us to offload the logic to return a specific http response when these exceptions occur to another class, thus decreasing the CBO of the class.
     Also, the AuthenticationRequestModel used string's for the netId and the password. This means that in the authenticate method, we're manually constructing the actual NetId and Password objects. What we did is change the netId and password in AuthenticationRequestModel to the actual NetId and Password objects. This allows us to offload the construction of these objects to the spring boot model parser, thus decreasing the CBO since we no longer need to interact with the NetId and Password class inside of AuthenticationController.
     Finally we extracted the logic of the authenticate endpoint to its own service (AuthenticationService). This service will now handle actually checking the

credentials of the user and generating a JWT token. This decreased the amount of fields of the controller by 2, and also decreases the CBO because instead of interacting with 3 fields (AuthenticationManager, JwtTokenGenerator & JwtUserDetailsService) it now only interacts with AuthenticationService inside of the authenticate endpoint.

RESULTS


Lack of Cohesion
Weighted Method Count
Coupling Between Object Classes
Depth of Inheritance Tree
Number of Children
Response For a Class