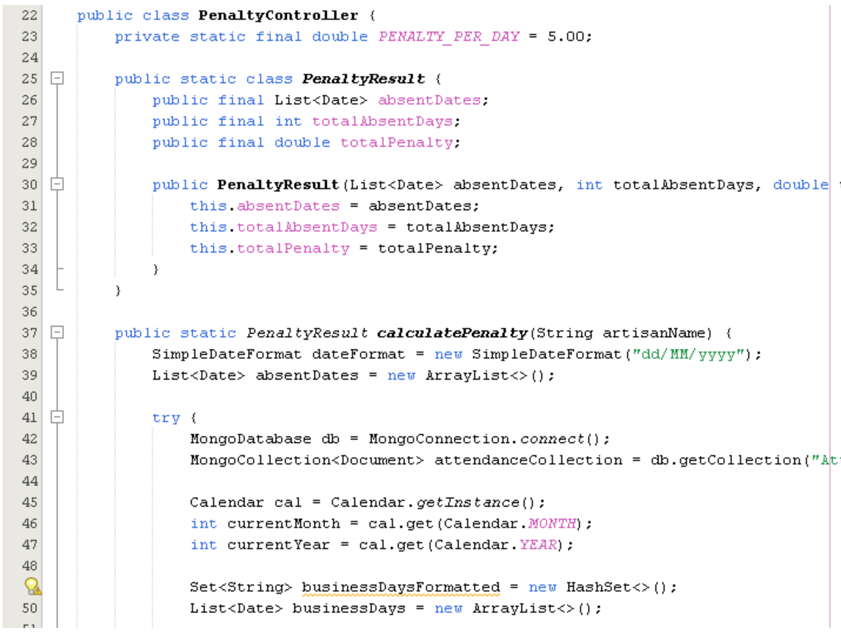
SOLID PRINCIPLES

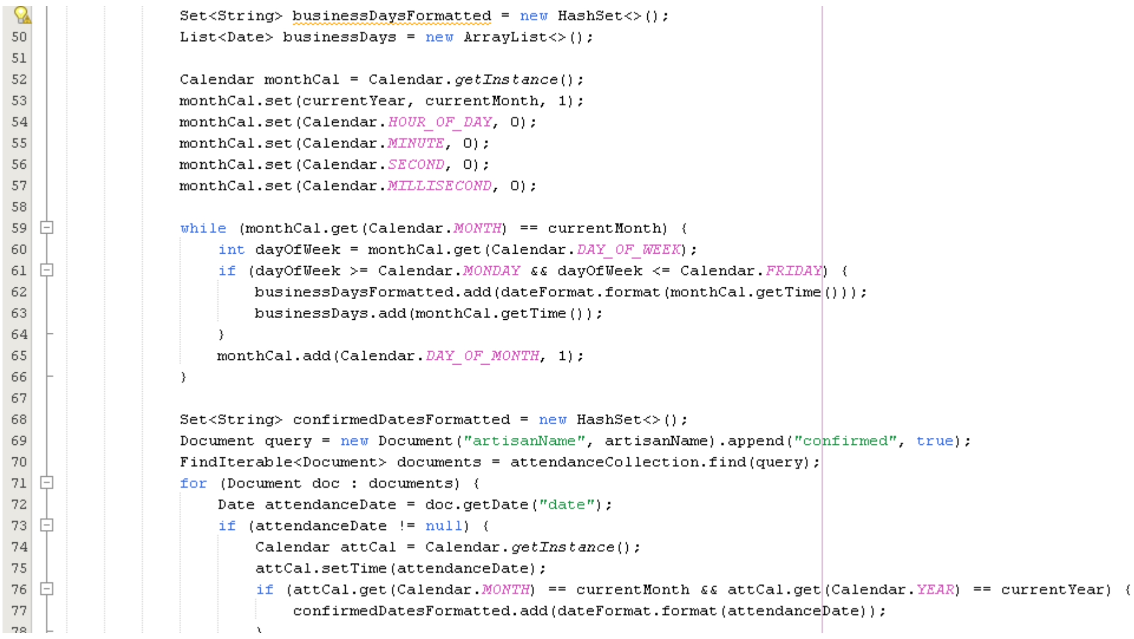
SIGLE RESPONSABILITY:

Explanation: It has date logic, connection to MongoDB, penalty calculation, and a mix of business logic and persistence. Perform date calculations, formatting, attendance filters, and result presentation, all in a single class.

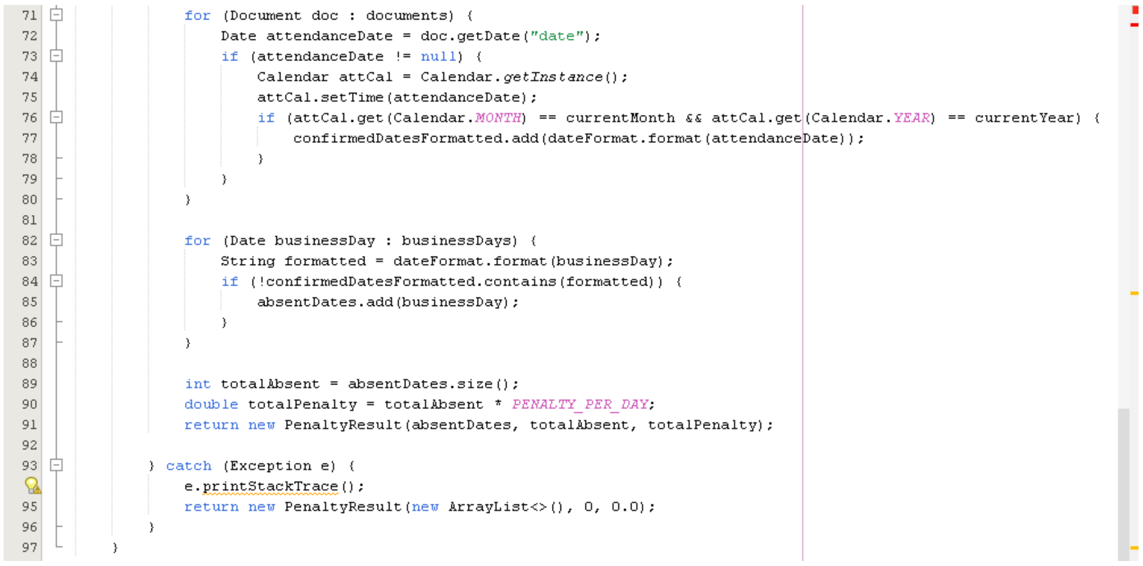
It has a direct dependency on a specific data source (Mongo).



```java
public class PenaltyController {
    private static final double PENALTY_PER_DAY = 5.00;

    public static class PenaltyResult {
        public final List<Date> absentDates;
        public final int totalAbsentDays;
        public final double totalPenalty;

        public PenaltyResult(List<Date> absentDates, int totalAbsentDays, double totalPenalty) {
            this.absentDates = absentDates;
            this.totalAbsentDays = totalAbsentDays;
            this.totalPenalty = totalPenalty;
        }
    }

    public static PenaltyResult calculatePenalty(String artisanName) {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
        List<Date> absentDates = new ArrayList<>();

        try {
            MongoDatabase db = MongoConnection.connect();
            MongoCollection<Document> attendanceCollection = db.getCollection("Attendance");

            Calendar cal = Calendar.getInstance();
            int currentMonth = cal.get(Calendar.MONTH);
            int currentYear = cal.get(Calendar.YEAR);

            Set<String> businessDaysFormatted = new HashSet<>();
            List<Date> businessDays = new ArrayList<>();
```



```java
            Set<String> businessDaysFormatted = new HashSet<>();
            List<Date> businessDays = new ArrayList<>();

            Calendar monthCal = Calendar.getInstance();
            monthCal.set(currentYear, currentMonth, 1);
            monthCal.set(Calendar.HOUR_OF_DAY, 0);
            monthCal.set(Calendar.MINUTE, 0);
            monthCal.set(Calendar.SECOND, 0);
            monthCal.set(Calendar.MILLISECOND, 0);

            while (monthCal.get(Calendar.MONTH) == currentMonth) {
                int dayOfWeek = monthCal.get(Calendar.DAY_OF_WEEK);
                if (dayOfWeek >= Calendar.MONDAY && dayOfWeek <= Calendar.FRIDAY) {
                    businessDaysFormatted.add(dateFormat.format(monthCal.getTime()));
                    businessDays.add(monthCal.getTime());
                }
                monthCal.add(Calendar.DAY_OF_MONTH, 1);
            }

            Set<String> confirmedDatesFormatted = new HashSet<>();
            Document query = new Document("artisanName", artisanName).append("confirmed", true);
            FindIterable<Document> documents = attendanceCollection.find(query);
            for (Document doc : documents) {
                Date attendanceDate = doc.getDate("date");
                if (attendanceDate != null) {
                    Calendar attCal = Calendar.getInstance();
                    attCal.setTime(attendanceDate);
                    if (attCal.get(Calendar.MONTH) == currentMonth && attCal.get(Calendar.YEAR) == currentYear) {
                        confirmedDatesFormatted.add(dateFormat.format(attendanceDate));
                    }
```



```java
                for (Document doc : documents) {
                    Date attendanceDate = doc.getDate("date");
                    if (attendanceDate != null) {
                        Calendar attCal = Calendar.getInstance();
                        attCal.setTime(attendanceDate);
                        if (attCal.get(Calendar.MONTH) == currentMonth && attCal.get(Calendar.YEAR) == currentYear) {
                            confirmedDatesFormatted.add(dateFormat.format(attendanceDate));
                        }
                    }
                }

                for (Date businessDay : businessDays) {
                    String formatted = dateFormat.format(businessDay);
                    if (!confirmedDatesFormatted.contains(formatted)) {
                        absentDates.add(businessDay);
                    }
                }

                int totalAbsent = absentDates.size();
                double totalPenalty = totalAbsent * PENALTY_PER_DAY;
                return new PenaltyResult(absentDates, totalAbsent, totalPenalty);

            } catch (Exception e) {
                e.printStackTrace();
                return new PenaltyResult(new ArrayList<>(), 0, 0.0);
            }
        }
```

Possible solution:

SRP Single responsibility:

Break the class down into more specific components:

PenaltyCalculator: This is responsible for calculating the penalty from a list of absence dates.

BusinessCalendar: Returns the business days of the month.

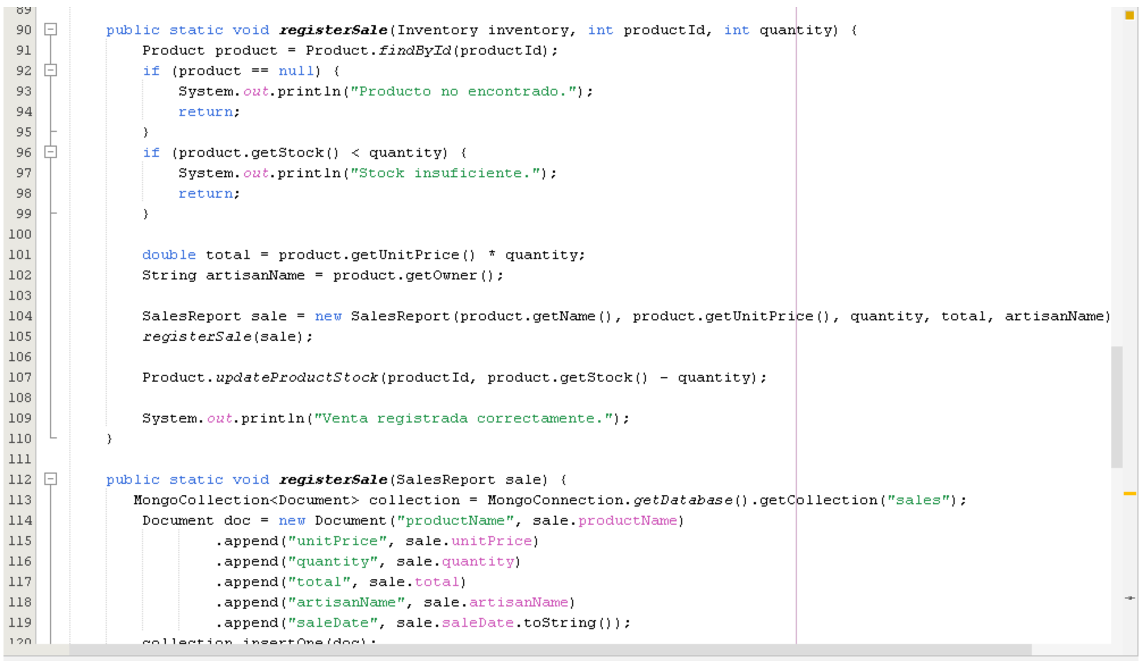
DIP – Dependency Inversion:

Instead of PenaltyController depending directly on MongoConnection, it could depend on an abstraction.

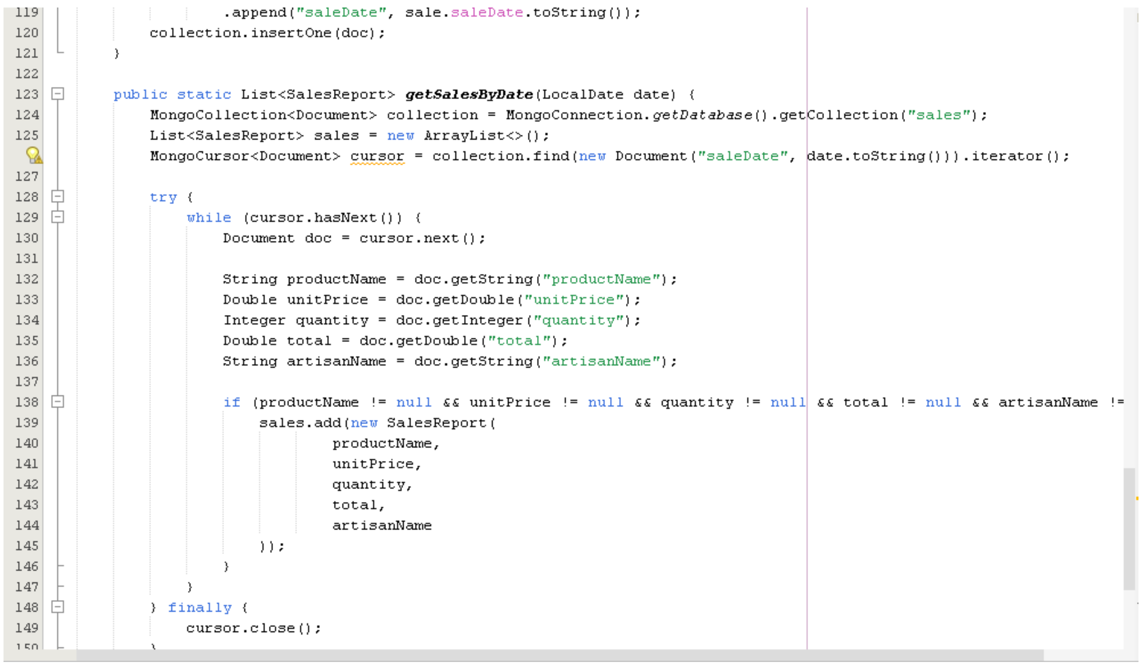
That way, you could easily change the logic without touching the controller.

Open/Closed Principle

Explanation: If you want to change the database you would have to directly modify SalesReport, SearchReport breaking the SOLID principles.



```java
    public static void registerSale(Inventory inventory, int productId, int quantity) {
        Product product = Product.findById(productId);
        if (product == null) {
            System.out.println("Producto no encontrado.");
            return;
        }
        if (product.getStock() < quantity) {
            System.out.println("Stock insuficiente.");
            return;
        }

        double total = product.getUnitPrice() * quantity;
        String artisanName = product.getOwner();

        SalesReport sale = new SalesReport(product.getName(), product.getUnitPrice(), quantity, total, artisanName)
        registerSale(sale);

        Product.updateProductStock(productId, product.getStock() - quantity);

        System.out.println("Venta registrada correctamente.");
    }

    public static void registerSale(SalesReport sale) {
       MongoCollection<Document> collection = MongoConnection.getDatabase().getCollection("sales");
        Document doc = new Document("productName", sale.productName)
                .append("unitPrice", sale.unitPrice)
                .append("quantity", sale.quantity)
                .append("total", sale.total)
                .append("artisanName", sale.artisanName)
                .append("saleDate", sale.saleDate.toString());
```



```java
                .append("saleDate", sale.saleDate.toString());
        collection.insertOne(doc);
    }

    public static List<SalesReport> getSalesByDate(LocalDate date) {
        MongoCollection<Document> collection = MongoConnection.getDatabase().getCollection("sales");
        List<SalesReport> sales = new ArrayList<>();
        MongoCursor<Document> cursor = collection.find(new Document("saleDate", date.toString())).iterator();

        try {
            while (cursor.hasNext()) {
                Document doc = cursor.next();

                String productName = doc.getString("productName");
                Double unitPrice = doc.getDouble("unitPrice");
                Integer quantity = doc.getInteger("quantity");
                Double total = doc.getDouble("total");
                String artisanName = doc.getString("artisanName");

                if (productName != null && unitPrice != null && quantity != null && total != null && artisanName !=
                    sales.add(new SalesReport(
                            productName,
                            unitPrice,
                            quantity,
                            total,
                            artisanName
                    ));
                }
            }
        } finally {
            cursor.close();
```

Possible solution: Using a database abstraction.

Interface Segregation Principles:

Explain: High dependency on concrete classes

Controllers depend directly on:

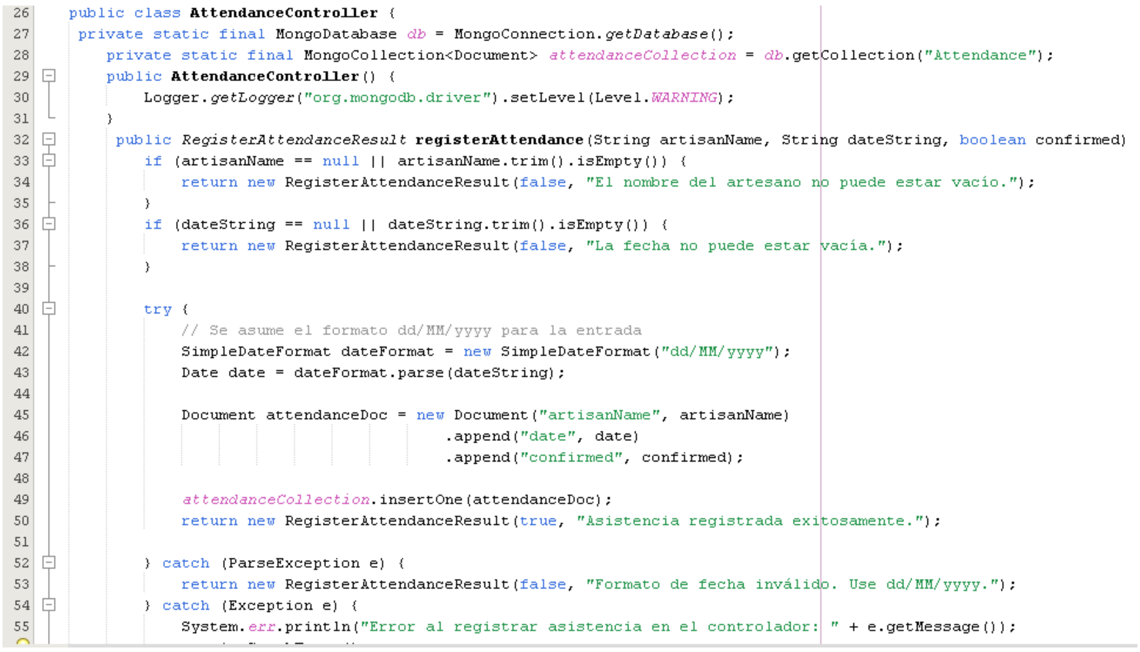
MongoConnection

Product This creates tight coupling, preventing:

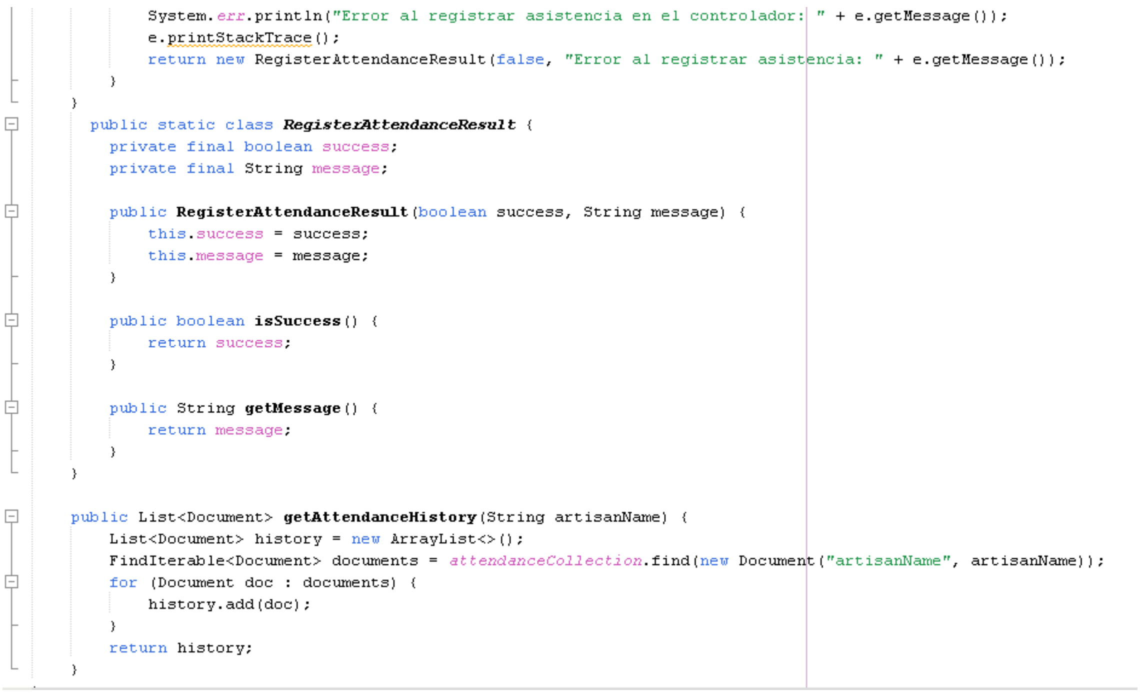
Easily changing the database

Reusing logic in a different context

Performing unit tests without touching the real database



```java
public class AttendanceController {
 private static final MongoDatabase db = MongoConnection.getDatabase();
    private static final MongoCollection<Document> attendanceCollection = db.getCollection("Attendance");
    public AttendanceController() {
        Logger.getLogger("org.mongodb.driver").setLevel(Level.WARNING);
    }
     public RegisterAttendanceResult registerAttendance(String artisanName, String dateString, boolean confirmed)
        if (artisanName == null || artisanName.trim().isEmpty()) {
            return new RegisterAttendanceResult(false, "El nombre del artesano no puede estar vacío.");
        }
        if (dateString == null || dateString.trim().isEmpty()) {
            return new RegisterAttendanceResult(false, "La fecha no puede estar vacía.");
        }

        try {
            // Se asume el formato dd/MM/yyyy para la entrada
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
            Date date = dateFormat.parse(dateString);

            Document attendanceDoc = new Document("artisanName", artisanName)
                                            .append("date", date)
                                            .append("confirmed", confirmed);

            attendanceCollection.insertOne(attendanceDoc);
            return new RegisterAttendanceResult(true, "Asistencia registrada exitosamente.");

        } catch (ParseException e) {
            return new RegisterAttendanceResult(false, "Formato de fecha inválido. Use dd/MM/yyyy.");
        } catch (Exception e) {
            System.err.println("Error al registrar asistencia en el controlador: " + e.getMessage());
```



```java
            System.err.println("Error al registrar asistencia en el controlador: " + e.getMessage());
            e.printStackTrace();
            return new RegisterAttendanceResult(false, "Error al registrar asistencia: " + e.getMessage());
        }
    }
     public static class RegisterAttendanceResult {
        private final boolean success;
        private final String message;

        public RegisterAttendanceResult(boolean success, String message) {
            this.success = success;
            this.message = message;
        }

        public boolean isSuccess() {
            return success;
        }

        public String getMessage() {
            return message;
        }
    }

    public List<Document> getAttendanceHistory(String artisanName) {
        List<Document> history = new ArrayList<>();
        FindIterable<Document> documents = attendanceCollection.find(new Document("artisanName", artisanName));
        for (Document doc : documents) {
            history.add(doc);
        }
        return history;
    }
```

Possible solution:

Introduce interfaces and abstractions:

Creating interfaces for key operations, such as saving attendance, retrieving products, or calculating penalties, allows:

Easily replace implementations.

Invert dependencies: Controllers no longer know the internal details of Mongo or other services.