Ćwiczenie: Blue-Green Refactor

Obecny kod warstwy ACL pozwala wybrać za pomocą toggla:
- Czy liczymy oryginalnym kodem legacy
- Czy liczymy nowo zrefaktoryzowanym kodem obiektowym

Przepisz logikę warstwy ACL tak by:
- Zawsze liczyła za pomocą oryginalnego kodu legacy
- Opcjonalnie przy włączonym toggle:
  - Dodatkowo liczy nowo zrefaktoryzowanym kodem obiektowym
  - Porównywała wyniki w zakresie: daty i ilości niedoboru
  - Logowała:
    - Informację: sukces / błąd
    - Scenariusz: stock (level), productions (data + output), demands (data + level)
    - Różnicę w wynikach

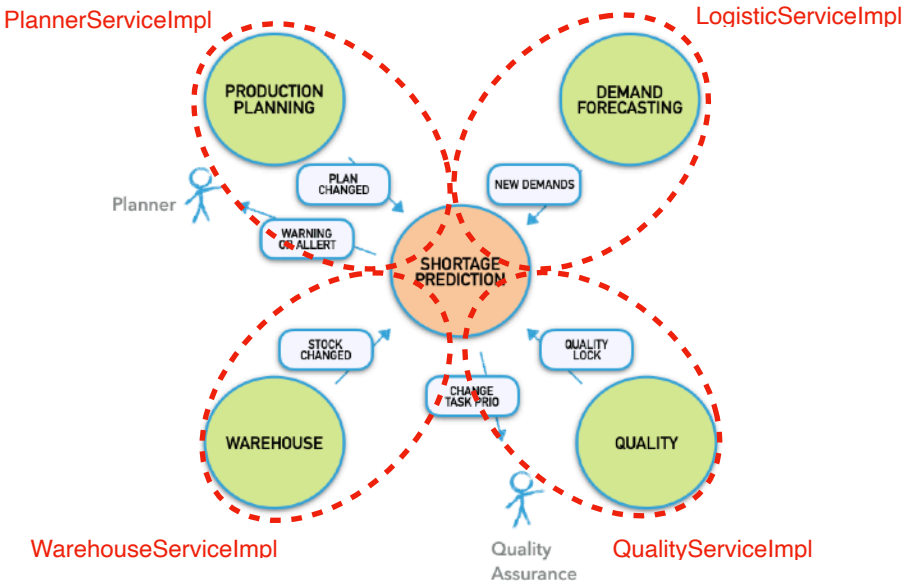
UWAGI:
Na potrzeby ćwiczenia porównywanie wyników i logowanie różnic ogranicz wyłącznie do rozmiaru kolekcji List<ShortageEntity>. W realnym przypadku było by to pełne porównanie (data + missing).
Na potrzeby ćwiczenia możesz „logować” na standardowe wyjście.

Ćwiczenie: Rozszerzenie odpowiedzialności modułu shortages

Obecny zakres modułu **shortages** ogranicza się wyłącznie do wyliczenia algorytmu ShortagePrediction.

Jednak w innych serwisach istnieją nadal fragmenty logiki związanej z przewidywaniem i alarmowaniem o niedoborach:
- **PlannerServiceImpl** metoda processShortages
- **LogisticServiceImpl** fragment metody adjustDemand
- **WarehouseServiceImpl** metoda processShortages
- **QualityServiceImpl** metoda processShortages

Tym samym zakres odpowiedzialności serwisów NIE pokrywa się 1 do 1 z logicznym podziałem na obszary biznesowe:



Do wymienionych serwisów dodaj zależność typu **ShortagePredictionService** a metody:
- **PlannerServiceImpl** metoda **processShortages**
- **WarehouseServiceImpl** metoda **processShortages**
- **QualityServiceImpl** metoda **processShortages**

przenieś do serwisu **ShortagePredictionService**, przenieś również niezbędne zależności.

Wyekstrahuj do metody właściwy fragment metody **LogisticServiceImpl.adjustDemand** (odpowiedzialny z niedobory) i przenieś w analogiczny sposób do serwisu **ShortagePredictionService**.

W serwisach: **PlannerServiceImpl, WarehouseServiceImpl, QualityServiceImpl, LogisticServiceImpl**, w odpowiednim miejscu wywołaj właściwą metodę z **ShortagePredictionService**.

Ćwiczenie: Czyszczenie Anti-corruption layer i legacy

Zastąp wszystkie obecne wywołania ShortageFinderACL:

```
public void processShortages(String productRefNo) {
    LocalDate today = LocalDate.now(clock);
    CurrentStock currentStock = stockService.getCurrentStock(productRefNo);

    List<ShortageEntity> shortages = ShortageFinderACL.findShortages(
            today, confShortagePredictionDaysAhead,
            currentStock,
            productionDao.findFromTime(productRefNo, today.atStartOfDay()),
            demandDao.findFrom(today.atStartOfDay(), productRefNo)
    );

    List<ShortageEntity> previous = shortageDao.getForProduct(productRefNo);
    if (!shortages.isEmpty() && !shortages.equals(previous)) {
        notificationService.softNotifyPlanner(shortages);
        if (currentStock.getLocked() > 0 &&
                shortages.get(0).getAtDay()
                        .isBefore(today.plusDays(confIncreaseQATaskPriorityInDays))) {
            jiraService.increasePriorityFor(productRefNo);
```

Wywołanie ACL

Wywołaniem kodu nowego modelu:

```
ShortagePrediction shortagePrediction = factory.create(
                                productRefNo, today, confShortagePredictionDaysAhead);
List<ShortageEntity> shortages = shortagePrediction.predict();
```

Przy tej okazji przenieś zależność :

```
private ProductionDao productionDao;
private StockService stockService;
private DemandDao demandDao;
```

oraz logikę pobierania danych:

```
stockService.getCurrentStock(productRefNo)
productionDao.findFromTime(productRefNo, today.atStartOfDay())
demandDao.findFrom(today.atStartOfDay(), productRefNo)
```

do ShortagerPredictionFactory.

Usuń kod ACL i kod legacy **ShortageFinder**.

Ćwiczenie: Zamiana adaptera na Value Object

Przekształć klasę **ProductionOutputs** tak by wewnętrznie nie zależała od **ProductionEntity**. Docelowo wewnętrzną strukturą powinno być:

Map<LocalDate, Long> **outputs**;

Zamiast obecnego

Map<LocalDate, List<ProductionEntity>> **outputs**;

Logika mapowania **List<ProductionEntity>** na **Map<LocalDate, Long>** powinna znaleść się poza klasą **ProductionOutputs**, a konkretnie w metodzie: **ShortagerPredictionFactory.create**.

Do przemalowania encji możesz posłużyć się strumieniem:

```
productions.stream().collect(groupingBy(
        e -> e.getStart().toLocalDate(),
        summingLong(ProductionEntity::getOutput))
);
```

Ćwiczenie: Zamiana adaptera na Value Object (2. przypadek)

Przekształć klasę **Demands i DailyDemand** tak by wewnętrznie nie zależały od **DemandEntity**. Docelowo wewnętrzną strukturą powinno być:

Map<LocalDate, DailyDemand> **demandsPerDay**;

Zamiast obecnego

Map<LocalDate, DemandEntity> **demandsPerDay**;

Logika mapowania **List<DemandEntity>** na **Map<LocalDate, DailyDemand>** wraz z wykorzystaniem klasy **Util** powinna znaleść się poza klasami **Demands** i **DailyDemand**, a konkretnie w metodzie: **ShortagerPredictionFactory.create**.