ZADANIA Z AI

Zadanie 1: Usuwanie Duplikatów Pracowników

Nauka pracy z kolekcjami Set i implementacja metod equals/hashCode.

Prompt dla AI

```
W klasie ClassEmployee zaimplementuj metodę removeDuplicates() która:
- usuwa duplikaty pracowników na podstawie imienia i nazwiska
- zachowuje pierwsze wystąpienie pracownika
- zwraca liczbę usuniętych duplikatów
- używa Set do identyfikacji duplikatów
- nie modyfikuje oryginalnej listy, tworzy nową

Dodatkowo sprawdź czy klasa Employee ma poprawnie zaimplementowane equals() i
hashCode()
```

Kod Startowy

```
public class ClassEmployee {
    private List<Employee> employees;

    // TODO: Metoda do wygenerowania przez AI
    public int removeDuplicates() {
        // AI wygeneruje implementację
    }
}
```

Checklist Weryfikacji

- ☐ Metoda zwraca poprawną liczbę usuniętych duplikatów,
- ☐ Zachowuje pierwsze wystąpienie pracownika,
- ☐ Nie modyfikuje oryginalnej listy,
- ☐ equals() i hashCode() w Employee działają poprawnie,
- ☐ Obsługuje przypadek pustej listy.

Zadanie 2: Grupowanie Pracowników Według Stanu

Praca z Map i grupowanie danych, wykorzystanie Stream API.

Prompt dla AI

```
W klasie ClassEmployee dodaj metodę groupByCondition() która:
- grupuje pracowników według ich stanu (EmployeeCondition)
- zwraca Map<EmployeeCondition, List<Employee>>
- używa Stream API z Collectors.groupingBy()
- obsługuje przypadek gdy dla niektórych stanów nie ma pracowników
- sortuje pracowników w każdej grupie alfabetycznie po nazwisku
```

Kod Startowy

```
import java.util.stream.Collectors;
import java.util.Map;

public class ClassEmployee {
    // TODO: Metoda do wygenerowania przez AI
    public Map<EmployeeCondition, List<Employee>> groupByCondition() {
        // AI wygeneruje implementację używając Stream API
    }
}
```

Pytania Kontrolne

1. Czy metoda wykorzystuje Stream API?
2. Czy pracownicy w grupach są posortowani?
3. Jak zachowuje się metoda dla pustej listy?
4. Czy wszystkie stany enum są reprezentowane w wyniku?

Zadanie 3: Obliczanie Mediany Wynagrodzenia

Algorytmy statystyczne, sortowanie, praca z liczbami.

Prompt dla AI

```
W klasie ClassEmployee zaimplementuj metodę medianSalary() która:
- oblicza medianę wynagrodzeń wszystkich pracowników
- dla parzystej liczby pracowników zwraca średnią z dwóch środkowych wartości
- dla nieparzystej liczby zwraca środkową wartość
- zwraca 0.0 dla pustej listy pracowników
- nie modyfikuje oryginalnej listy (tworzy kopię do sortowania)
- używa precyzji double
```

Kod Startowy

```
public class ClassEmployee {
    // TODO: Metoda do wygenerowania przez AI
    public double medianSalary() {
        // AI wygeneruje algorytm obliczania mediany
    }
}
```

Analiza Kodu AI

Po wygenerowaniu kodu przez AI, sprawdź: 1. Czy algorytm poprawnie obsługuje przypadki parzyste i nieparzyste? 2. Czy sortowanie nie wpływa na oryginalną listę? 3. Czy precyzja obliczeń jest odpowiednia?

Zadanie 4: TreeMap dla Automatycznego Sortowania Grup

Różnice między implementacjami Map, automatyczne sortowanie.

Prompt dla AI

```
Zmodyfikuj klasę ClassContainer aby używała TreeMap<String, ClassEmployee> za
miast HashMap:
- grupy będą automatycznie sortowane alfabetycznie według nazwy
- zachowaj wszystkie istniejące metody (addClass, removeClass, findEmpty, sum
mary)
- dodaj metodę getGroupsInOrder() zwracającą posortowaną listę nazw grup
- dodaj konstruktor pozwalający na własny Comparator dla kluczy
- obsłuż przypadki gdy nazwy grup różnią się tylko wielkością liter
```

Kod Startowy

```java
import java.util.*;

public class ClassContainer {
    // TODO: Zmień na TreeMap
    private Map<String, ClassEmployee> groups;

    // TODO: Konstruktory do wygenerowania przez AI
    public ClassContainer() {
        // Domyślne sortowanie alfabetyczne
    }

    public ClassContainer(Comparator<String> keyComparator) {
        // Własny komparator
    }

    // TODO: Nowa metoda
    public List<String> getGroupsInOrder() {
        // AI wygeneruje implementację
    }
}
```

Zadanie 5: Liczenie Pracowników w Grupach

Agregacja danych, praca z Map i Stream API.

Prompt dla AI

```
W klasie ClassContainer dodaj metodę countEmployeesInGroups() która:
- zwraca Map<String, Integer> gdzie klucz to nazwa grupy, wartość to liczba p
racowników
- używa Stream API do przetwarzania danych
- uwzględnia tylko grupy które mają pracowników (pomija puste grupy)
- sortuje wynik malejąco według liczby pracowników
- używa LinkedHashMap aby zachować kolejność sortowania
```

Kod Startowy

```java
public class ClassContainer {
    // TODO: Metoda do wygenerowania przez AI
    public Map<String, Integer> countEmployeesInGroups() {
        // AI wygeneruje implementację z Stream API
    }

    // Pomocnicza metoda do wyświetlania statystyk
    public void printEmployeeStatistics() {
        Map<String, Integer> stats = countEmployeesInGroups();

        System.out.println("Statystyki pracowników w grupach:");
        stats.forEach((groupName, count) ->
            System.out.println(groupName + ": " + count + " pracowników"));
    }
}
```

Rozszerzenie - Dodatkowe Statystyki

Po wygenerowaniu podstawowej metody, poproś AI o rozszerzenie:

```
Dodaj do ClassContainer metodę getDetailedStatistics() która zwraca Map<Strin
g, GroupStatistics>
gdzie GroupStatistics to klasa zawierająca:
- liczbę pracowników
- średnie wynagrodzenie
- najwyższe wynagrodzenie
- najniższe wynagrodzenie
- rozkład według stanów pracowników
```

Zadanie 6: Najstarszy i Najmłodszy Pracownik

Algorytmy znajdowania ekstremów, Optional, Stream API.

Prompt dla AI

```
W klasie ClassEmployee dodaj metody oldestEmployee() i youngestEmployee() któ
re:
- zwracają Optional<Employee> (mogą nie znaleźć pracownika)
- używają Stream API z metodami min()/max()
- porównują pracowników według roku urodzenia
- obsługują przypadek pustej listy pracowników
- dodaj także metody getAverageAge() i getAgeStatistics()

Dodatkowo stwórz klasę AgeStatistics zawierającą:
- najmłodszy wiek, najstarszy wiek, średni wiek
- liczbę pracowników w różnych przedziałach wiekowych
```

Kod Startowy

```
import java.util.Optional;
import java.time.Year;

public class ClassEmployee {
    // TODO: Metody do wygenerowania przez AI
    public Optional<Employee> oldestEmployee() {
        // AI wygeneruje z Stream API
    }

    public Optional<Employee> youngestEmployee() {
        // AI wygeneruje z Stream API
    }

    public double getAverageAge() {
        // AI wygeneruje obliczanie średniego wieku
    }

    public AgeStatistics getAgeStatistics() {
        // AI wygeneruje szczegółowe statystyki
    }
}

// TODO: Klasa do wygenerowania przez AI
class AgeStatistics {
    // Pola i metody dla statystyk wiekowych
}
```

Zadanie 7: LinkedHashMap - Zachowanie Kolejności

Różnice między implementacjami Map, znaczenie kolejności wstawiania.

Prompt dla AI

```
Zmodyfikuj klasę ClassContainer aby oferowała trzy tryby przechowywania grup:
1. HashMap - bez gwarancji kolejności (domyślny)
2. LinkedHashMap - zachowuje kolejność dodawania
3. TreeMap - sortowanie alfabetyczne

Stwórz enum StorageMode i konstruktor przyjmujący ten parametr.
Dodaj metodę changeStorageMode() która konwertuje między trybami zachowując d
ane.
Dodaj metodę demonstrateOrderDifferences() pokazującą różnice w kolejności.
```

Kod Startowy

```
public enum StorageMode {
    HASH_MAP,           // Bez gwarancji kolejności
    LINKED_HASH_MAP,    // Kolejność wstawiania
    TREE_MAP            // Sortowanie alfabetyczne
```

```
}

public class ClassContainer {
    private Map<String, ClassEmployee> groups;
    private StorageMode currentMode;

    // TODO: Konstruktory i metody do wygenerowania przez AI
    public ClassContainer(StorageMode mode) {
        // AI wygeneruje inicjalizację odpowiedniego typu Map
    }

    public void changeStorageMode(StorageMode newMode) {
        // AI wygeneruje konwersję między typami Map
    }

    public void demonstrateOrderDifferences() {
        // AI wygeneruje demo pokazujące różnice w kolejności
    }
}
```

Eksperyment Do Wykonania

```
@Test
public void testOrderDifferences() {
    // Test zachowania kolejności w różnych implementacjach
    String[] groupNames = {"Zespół C", "Zespół A", "Zespół B", "Zespół D"};

    for (StorageMode mode : StorageMode.values()) {
        ClassContainer container = new ClassContainer(mode);

        // Dodaj grupy w określonej kolejności
        for (String name : groupNames) {
            container.addClass(name, 10);
        }

        System.out.println("Tryb: " + mode);
        System.out.println("Kolejność: " + container.getGroupsInOrder());
        System.out.println();
    }
}
```

Zadanie 8: Filtrowanie po Wynagrodzeniu

Filtrowanie danych, Stream API, predykaty.

Prompt dla AI

```
W klasie ClassEmployee dodaj metodę filterByMinSalary(double minSalary) która
:
- zwraca List<Employee> z pracownikami zarabiającymi co najmniej minSalary
- używa Stream API z filter()
- sortuje wynik malejąco według wynagrodzenia
- dodaj także metodę filterBySalaryRange(double min, double max)
- stwórz klasę SalaryFilter z metodami do tworzenia różnych filtrów

Dodatkowo dodaj metody:
- getTopEarners(int count) - N najlepiej zarabiających
- getBottomEarners(int count) - N najgorzej zarabiających
- filterByPercentile(double percentile) - pracownicy powyżej percentyla
```

Kod Startowy

```java
import java.util.function.Predicate;

public class ClassEmployee {
    // TODO: Metody filtrowania do wygenerowania przez AI
    public List<Employee> filterByMinSalary(double minSalary) {
        // AI wygeneruje z Stream API
    }

    public List<Employee> filterBySalaryRange(double min, double max) {
        // AI wygeneruje filtrowanie zakresu
    }

    public List<Employee> getTopEarners(int count) {
        // AI wygeneruje top N zarabiających
    }

    public List<Employee> filterByPercentile(double percentile) {
        // AI wygeneruje filtrowanie według percentyla
    }
}

// TODO: Klasa pomocnicza do wygenerowania przez AI
class SalaryFilter {
    public static Predicate<Employee> minSalary(double min) {
        // AI wygeneruje predykat
    }

    public static Predicate<Employee> salaryRange(double min, double max) {
        // AI wygeneruje predykat zakresu
    }
```

```
    public static Predicate<Employee> topPercent(List<Employee> allEmployees,
double percent) {
        // AI wygeneruje predykat dla top %
    }
}
```

Zadanie 9: Sprawdzenie Delegacji

Predykaty boolean, Stream API anyMatch/allMatch/noneMatch.

Prompt dla AI

```
W klasie ClassEmployee dodaj metodę hasEmployeesOnDelegation() która:
- zwraca true jeśli przynajmniej jeden pracownik jest w delegacji
- używa Stream API z anyMatch()
- dodaj także metody:
  * allEmployeesPresent() - wszyscy obecni
  * noEmployeesSick() - nikt nie jest chory
  * getConditionSummary() - podsumowanie wszystkich stanów

Stwórz klasę ConditionAnalyzer z metodami statycznymi do analizy stanów praco
wników.
```

Kod Startowy

```
public class ClassEmployee {
    // TODO: Metody do wygenerowania przez AI
    public boolean hasEmployeesOnDelegation() {
        // AI użyje anyMatch()
    }

    public boolean allEmployeesPresent() {
        // AI użyje allMatch()
    }

    public boolean noEmployeesSick() {
        // AI użyje noneMatch()
    }

    public Map<EmployeeCondition, Long> getConditionSummary() {
        // AI wygeneruje zliczanie z groupingBy i counting
    }
}

// TODO: Klasa pomocnicza
class ConditionAnalyzer {
    public static boolean hasCondition(List<Employee> employees, EmployeeCond
ition condition) {
        // AI wygeneruje uniwersalną metodę sprawdzania
```

```
    }

    public static double getConditionPercentage(List<Employee> employees, Emp
loyeeCondition condition) {
        // AI wygeneruje obliczanie procentu
    }
}
```

Zadanie 10: Grupowanie według Wynagrodzenia

Złożone grupowanie, praca z kluczami liczbowymi w Map.

Prompt dla AI

```
W klasie ClassEmployee dodaj metodę groupBySalary() która:
- zwraca Map<Double, List<Employee>> grupując według dokładnego wynagrodzenia
- dodaj metodę groupBySalaryRange(double rangeSize) grupującą według przedzia
łów
- stwórz metodę getSalaryDistribution() zwracającą rozkład wynagrodzeń
- użyj Stream API z Collectors.groupingBy()

Dodatkowo stwórz klasę SalaryAnalyzer z metodami:
- analyzeDistribution() - analiza rozkładu
- findOutliers() - znajdowanie wartości odstających
- calculateGiniCoefficient() - współczynnik nierówności
```

Kod Startowy

```
public class ClassEmployee {
    // TODO: Metody grupowania do wygenerowania przez AI
    public Map<Double, List<Employee>> groupBySalary() {
        // AI użyje groupingBy z dokładnym wynagrodzeniem
    }

    public Map<String, List<Employee>> groupBySalaryRange(double rangeSize) {
        // AI wygeneruje grupowanie według przedziałów (np. "4000-5000")
    }

    public SalaryDistribution getSalaryDistribution() {
        // AI wygeneruje analizę rozkładu
    }
}

// TODO: Klasy pomocnicze do wygenerowania przez AI
class SalaryDistribution {
    private double min, max, mean, median, standardDeviation;
    private Map<String, Integer> ranges;

    // gettery, toString, analiza rozkładu
}
```

```
class SalaryAnalyzer {
    public static List<Employee> findOutliers(List<Employee> employees, doubl
e threshold) {
        // AI znajdzie wartości odstające (np. > 2 odchylenia standardowe)
    }

    public static double calculateGiniCoefficient(List<Employee> employees) {
        // AI wygeneruje obliczanie współczynnika Giniego
    }
}
```

ZADANIA KONTROLNE I TESTY

Test Integracyjny - Wszystkie Funkcjonalności

```
public class ComprehensiveTest {

    @Test
    public void testCompleteWorkflow() {
        // 1. Stwórz system z grupami
        ClassContainer container = new ClassContainer();
        container.addClass("IT", 10);
        container.addClass("HR", 5);

        ClassEmployee itGroup = container.getGroup("IT");
        ClassEmployee hrGroup = container.getGroup("HR");

        // 2. Dodaj pracowników
        itGroup.addEmployee(new Employee("Jan", "Kowalski", OBECNY, 1990, 800
0));
        itGroup.addEmployee(new Employee("Anna", "Nowak", DELEGACJA, 1985, 90
00));
        itGroup.addEmployee(new Employee("Piotr", "Kowalski", OBECNY, 1988, 7
500)); // duplikat nazwiska

        hrGroup.addEmployee(new Employee("Maria", "Zielińska", OBECNY, 1992,
5500));
        hrGroup.addEmployee(new Employee("Tomasz", "Wiśniewski", CHORY, 1980,
6000));

        // 3. Testuj wszystkie funkcjonalności AI

        // Grupowanie według stanu
        Map<EmployeeCondition, List<Employee>> byCondition = itGroup.groupByC
ondition();
        assertTrue(byCondition.containsKey(OBECNY));
        assertEquals(2, byCondition.get(OBECNY).size());
```

```
        // Mediana wynagrodzeń
        double median = itGroup.medianSalary();
        assertEquals(8000.0, median, 0.01); // środkowa z [7500, 8000, 9000]

        // Filtrowanie po wynagrodzeniu
        List<Employee> highEarners = itGroup.filterByMinSalary(8000);
        assertEquals(2, highEarners.size());

        // Najstarszy/najmłodszy
        Optional<Employee> oldest = itGroup.oldestEmployee();
        assertTrue(oldest.isPresent());
        assertEquals("Nowak", oldest.get().getLastName());

        // Sprawdzenie delegacji
        assertTrue(itGroup.hasEmployeesOnDelegation());
        assertFalse(hrGroup.hasEmployeesOnDelegation());

        // Statystyki kontenerów
        Map<String, Integer> stats = container.countEmployeesInGroups();
        assertEquals(Integer.valueOf(3), stats.get("IT"));
        assertEquals(Integer.valueOf(2), stats.get("HR"));
    }
}
```

Zadania Diagnostyczne

Zadanie A: Debugowanie Kodu AI

```
// Ten kod został wygenerowany przez AI, ale zawiera błędy
// Znajdź i popraw wszystkie problemy

public List<Employee> getEmployeesSortedByAge() {
    return employees.stream()
        .sorted((e1, e2) -> e1.getBirthYear() - e2.getBirthYear()) // Problem
1: kolejność
        .filter(emp -> emp != null) // Problem 2: kolejność operacji
        .collect(Collectors.toList());
}

public Map<String, Double> getAverageSalaryByLastName() {
    return employees.stream()
        .collect(Collectors.groupingBy(
            Employee::getLastName,
            Collectors.averagingDouble(Employee::getSalary)
        )); // Problem 3: brak obsługi pustej listy
}
```

Zadanie B: Optymalizacja Wydajności

```
// Zoptymalizuj ten kod wygenerowany przez AI
public boolean hasHighEarner(double threshold) {
```

```
    List<Employee> highEarners = employees.stream()
        .filter(emp -> emp.getSalary() > threshold)
        .collect(Collectors.toList()); // Niepotrzebne collect

    return highEarners.size() > 0; // Można uprościć
}

// Twoja zoptymalizowana wersja:
public boolean hasHighEarner(double threshold) {
    // TODO: Napisz optymalną wersję
}
```

PYTANIA DO AUTOREFLEKSJI

1. **Jakość współpracy z AI:**
   - Czy Twoje prompty były wystarczająco szczegółowe?
   - Ile iteracji potrzebowałeś/-aś do uzyskania dobrego kodu?
   - Które zadania były najłatwiejsze/najtrudniejsze z AI?
2. **Weryfikacja i testowanie:**
   - Czy zawsze testowałeś/-aś kod przed zaakceptowaniem?
   - Jakie błędy najczęściej generowało AI?
   - Jak poprawiłeś/-aś jakość generowanego kodu?
3. **Nauka i zrozumienie:**
   - Czy rozumiesz cały kod który zaakceptowałeś?
   - Które koncepty lepiej zrozumiałeś/-aś dzięki AI?
   - Gdzie AI Ci przeszkadzało w nauce?