Testowanie w czystej Java (bez bibliotek zewnętrznych)

□ CZĘŚĆ I – Teoria

1. Odpowiedz na pytania:

1. Co to jest test jednostkowy?
2. Dlaczego warto testować kod?
3. Czym różni się testowanie manualne od automatycznego?
4. Co to jest przypadek testowy?

□ CZĘŚĆ II – Testowanie bez JUnit (czysta Java)

W czystej Javie testy możemy realizować poprzez:

- instrukcje `if`
- rzucanie wyjątków
- wypisywanie komunikatów
- użycie słowa kluczowego `assert`

□ Zadanie 1 – Prosta klasa

Utwórz klasę:

```
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }

    public int divide(int a, int b) {
        return a / b;
    }
}
```

□ Zadanie 2 – Testowanie przez metodę main()

Utwórz klasę testową:

```
public class CalculatorTest {

    public static void main(String[] args) {

        Calculator calc = new Calculator();

        // Test 1 - dodawanie
        if (calc.add(2, 3) == 5) {
            System.out.println("Test dodawania OK");
        } else {
            System.out.println("Test dodawania NIEPOPRAWNY");
        }

        // Test 2 - dzielenie
        if (calc.divide(10, 2) == 5) {
            System.out.println("Test dzielenia OK");
        } else {
            System.out.println("Test dzielenia NIEPOPRAWNY");
        }

    }
}
```

✏ Odpowiedz:

- Czy to jest test automatyczny?
- Jakie są wady takiego rozwiązania?

CZĘŚĆ III – Test negatywny (obsługa wyjątku)

Dodaj test dzielenia przez 0:

```
try {
    calc.divide(5, 0);
    System.out.println("BŁĄD - powinien wystąpić wyjątek!");
} catch (ArithmeticException e) {
    System.out.println("Test dzielenia przez 0 OK");
}
```

✏ Wyjaśnij:

- Dlaczego testy negatywne są ważne?
- Co sprawdzamy w tym teście?

CZĘŚĆ IV – Użycie słowa kluczowego assert

Java posiada wbudowane `assert`.

Przykład:

```
public class CalculatorAssertTest {

    public static void main(String[] args) {

        Calculator calc = new Calculator();

        assert calc.add(2, 3) == 5 : "Błąd w dodawaniu!";
        assert calc.divide(10, 2) == 5 : "Błąd w dzieleniu!";

        System.out.println("Wszystkie testy przeszły poprawnie.");
    }
}
```

 Aby uruchomić assert należy włączyć flagę:

```
java -ea CalculatorAssertTest
```

✏ Pytania:

1. Co się stanie, jeśli warunek `assert` będzie fałszywy?
2. Dlaczego `assert` domyślnie jest wyłączony?

CZĘŚĆ V – Rozszerzenie klasy

Dodaj do klasy `Calculator` metodę:

```
public int factorial(int n) {
    if (n < 0) {
        throw new IllegalArgumentException("Liczba nie może być ujemna");
    }

    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Zadanie testowe

Napisz testy dla:

- factorial(5) → 120
- factorial(0) → 1
- factorial(-3) → wyjątek

□ CZĘŚĆ VI – Zadanie problemowe

Zaprojektuj własną klasę:

`BankAccount`

z metodami:

- deposit()
- withdraw()
- getBalance()

Następnie napisz testy sprawdzające:

- poprawne zwiększanie salda,
- brak możliwości wypłaty większej kwoty niż saldo,
- brak możliwości wpłaty wartości ujemnej.

□ CZĘŚĆ VII – Refleksja

Odpowiedz:

1. Czy testowanie bez frameworków jest wygodne?
2. Jakie problemy mogą pojawić się w dużych projektach?
3. Dlaczego w praktyce używa się narzędzi takich jak JUnit?

⋆ Zadanie dodatkowe (dla chętnych)

Stwórz prostą klasę `SimpleTestRunner`, która:

- zlicza liczbę testów,
- zlicza liczbę testów poprawnych,
- wyświetla podsumowanie na końcu programu.