5.

Zadanie 5.1 Klasa z testami: Pracownik

Stwórz klasę **`Pracownik`**, której obiekt służy do zliczania godzin pracy jednego pracownika i wyliczania wypłaty zgodnie z modelem „timesheetowym" (płatność za godzinę zgodnie z ustaloną stawką), przy czym uwzględniana jest podwójna stawka za nadgodziny.

Postaraj się wykonać ten przykład zgodnie z metodyką pracy „Test-Driven Development", czyli najpierw pisz test, a dopiero później uzupełniaj implementację klasy w ten sposób, aby testy zadziałały.

Konstruktor **`Pracownik(int stawka)`** przyjmuje jedną wartość - stawkę godzinową pracownika.

```
Pracownik p1 = new Pracownik(100);
Pracownik p2 = new Pracownik(150);
```

Metody:

`void praca(int godziny)` – rejestruje jedną „dniówkę" pracy. Jeśli wartość podana jako parametr jest większa niż 8, to za godziny powyżej 8 naliczana jest podwójna stawka. Przepracowane godziny są w jakiś sposób zapamiętywane w obiekcie aż do momentu wypłaty.

`int wyplata()` – zwraca kwotę do wypłaty za przepracowane do tej pory godziny (jako iloczyn godzin i stawki godzinowej, z uwzględnieniem nadgodzin) i zeruje licznik - od tej pory godziny naliczane są od nowa. Jeśli nie ma aktualnie naliczonych godzin, zwraca zero.

Nadgodziny dotyczą tylko przekroczenia 8 godzin jednego dnia:

```
p1.praca(10);
int x = p1.wyplata(); // wynikiem jest 1200 : 8 godzin * 100 + 2 godziny * 200


p1.praca(5);
p1.praca(5);
int y = p1.wyplata();  // wynikiem jest 1000 : 10 godzin * 100
```

5.1.1 Opcjonalne rozszerzenia / utrudnienia dla chętnych:

1) Dla kwot pieniężnych zamiast **`int`** użyj **`BigDecimal`** z dwoma miejscami po przecinku.

2) Jako podklasę, dodaj klasę **`PracownikZPremia`**, który dodatkowo posiada metodę **`premia(int kwota)`** – do normalnie zarobionych pieniędzy dodaje podaną kwotę premii. Wiele premii jest sumowanych. Po wypłacie naliczane premie są zerowane.

Zadanie 5.2 Klasa Faktura

Zdefiniuj klasę służącą do wystawiania faktur. W jej implementacji wykorzystaj kolekcje (listy, mapy itp.). Możesz także stworzyć dodatkowe klasy (np. dla pozycji na fakturze).

Najlepiej utwórz zestaw testów JUnit, które będą sprawdzać, czy klasa działa prawidłowo. Możesz także napisać program konsolowy (dla chętnych i dysponujących czasem: okienkowy), który pozwoli wpisywać dane do faktury, a następnie wypisuje podsumowanie z wszystkimi pozycjami, sumą do zapłaty i podsumowaniem podatków.

Przykładowe operacje, które powinny działać:

```
Faktura faktura = new Faktura("1234/2019");
```

tworzy obiekt reprezentujący fakturę o podanym numerze. Początkowo faktura nie zawiera żadnych pozycji (wpisów).

```
faktura.dodajPozycję("Pralka automatyczna", 2, 1250.00, 23)
```

dodaje pozycję do faktury: nazwa towaru, ilość, cena jednostkowa netto, stawka VAT

```
int ile = faktura.iloscPozycji();
```

zwraca liczbę niepustych pozycji w fakturze.

```
double wartosc1 = faktura.wartoscPozycji(1);
```

zwraca wartość brutto z danej pozycji na fakturze (numeracja od 1).

```
double doZaplaty = faktura.doZaplaty();
```

zwraca sumę do zapłaty za całą fakturę.

```
faktura.wydrukuj();
```

wypisuje na ekran podsumowanie zawierające wszystkie pozycje, a dla każdej z nich:

- nr porządkowy (od 1),
- nazwa,
- cena jednostkowa netto,
- stawka VAT,
- wartość netto,
- wartość brutto

po wszystkich pozycjach podsumowanie zawiera sumę do zapłaty za całość.

Możliwe dalsze rozwinięcia:

1. Oprócz osobnych pozycji i jednej sumy na koniec wprowadź – jak na prawdziwych fakturach – osobne podsumowania dla poszczególnych stawek podatkowych. Zrób to wprowadzając osobną metodę faktura.sumaDlaStawki(stawkavat) i wypisz sumy dla poszczególnych stawek w `wydrukuj`().
2. Zamiast double użyj klasy `BigDecimal`. Zaokrąglaj wszystkie wypisywane kwoty do dwóch miejsc po przecinku.