

# Kolokwium #1 - Programowanie obiektowe - Zestaw W22

Imię i nazwisko, numer albumu .....

## Informacje wstępne

- Łącznie do zdobycia max 40 punktów. Próg zaliczenia: 25 pkt (bez innych punktów).
- **Czas: 90 minut. Po zakończeniu można wyjść, dalszych zajęć nie będzie.**
- **Kolokwium należy wykonać na komputerach zamontowanych na stałe w pracowniach.**
- Student przysyłając rozwiązania oświadcza, że rozwiązał je samodzielnie.
- W trakcie kolokwium nie można korzystać z żadnych materiałów pomocniczych w żadnej formie. Wszelkie kody powinny być napisane manualnie bez wspomaganie się dodatkami automatycznie generującymi kod (np. Copilot, chat GPT itp.).
- Publikowanie poleceń i rozwiązań w internecie jest zabronione do czasu napisania kolokwium przez wszystkie grupy ćw.
- Należy zwracać uwagę na właściwe umieszczenie kodu (luzem lub w pakiecie).
- Kod musi się kompilować, aby był sprawdzany.
- Należy oddzielać klasę z definicjami od klasy testującej (z main) zgodnie z poleceniami.
- Jeśli w poleceniu nie jest podany typ zmiennej, można go wybrać dowolnie.
- Jeśli w danej metodzie nie ma sprecyzowanej „walidacji”, to można ją pominąć.
- Metody nie powinny wykonywać nadmiarowych, nielogicznych czynności.
- Poza zmiennymi/polami w klasie wymienionym w poleceniach zabronione jest tworzenie innych pól w klasie. Stworzenie dodatkowych metod jest dopuszczalne, ale nie należy tego nadużywać.
- W pierwszym kolokwium nie występują zagnieżdżone klasy w żadnym z poleceń.
- Jeśli w poleceniu nie są sprecyzowane modyfikatory dostępu, należy dostępować zgodnie z zasadami hermetyzacji (pola prywatne, przy metodach najmniejszy z możliwych, klasy – dostęp pakietowy).
- Rozwiązania (projekt z IntelliJ) należy w całości spakować jako archiwum zip. Następnie ustawić nazwę. Rozwiązania należy umieścić na pendrive przekazany przez prowadzącego kolokwium.
- **Nazwa archiwum powinna być wg schematu NUMERZESTAWU\_NUMERALBUMU.zip gdzie numer zestawu znajduje się na górze kartki z poleceniami. np. A23\_123456.zip.**
- Archiwum powinno być bez hasła.
- Kod zakomentowany nie będzie sprawdzany.
- Zawartość pendrive będzie pusta. Udostępniony będzie tylko w celu zgrania rozwiązań. Umieszczenie poleceń na pendrive powinno odbyć się w czasie kolokwium. Rozwiązania po czasie mogą nie być sprawdzane.
- Jeśli w poleceniu pojawia się informacja o konieczności zachowania formatowania napisów (np. wielkość znaków, znaki interpunkcyjne), to należy to bezwzględnie wykonać.
- Podpunkty będą oceniane kaskadowo – wykonanie ich bez wykonania wcześniejszych podpunktów może oznaczać zero punktów.
- O ile nie zaznaczono w poleceniu inaczej, każdą z metod należy wywołać co najmniej jeden raz (może być bardzo trywialnie). Warto zwrócić uwagę, że samo tworzenie obiektów w każdym zdefiniowanym samodzielnie typie nie jest wymagane (chyba że polecenie tego wymaga).
- Należy zachowywać kolejność argumentów w konstruktorach i metodach. Należy dążyć do tego, że nazwy argumentów metod powinny pokrywać się z nazwami pól w klasie, gdzie to ma sens.
- Warto zwracać uwagę na typ zwracany metod – jeśli metoda ma „coś” zwrócić, będzie to wskazane w poleceniu.
- Po kartkach z poleceniami można pisać i traktować jako brudnopis.

## Zadanie 1. Klasa `Clothing` (Ubranie) (13pkt max.)

A. (1pkt) Klasa `Clothing` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.clothing`.

B. (1pkt) Klasa powinna posiadać prywatne pola:

- `material`, (materiał), typ `String`
- `size`, (rozmiar), typ `String`
- `price`, (cena), typ `double`

C. (3pkt) Napisz trzyargumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B. Zapewnij niezależnie warunki sprawdzające poprawność:

- `String material` nie może być `null` - w przypadku błędnej wartości ustaw `"Cotton"`.
- `String size` nie może być `null` - w przypadku błędnej wartości ustaw `"Medium"`.
- Cena `price` musi być liczbą nieujemną, w przeciwnym wypadku ustaw ją na `50.0`.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Powinny mieć taką samą walidację jak w konstruktorze.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu zgodnie z następującym schematem (zwróć uwagę na wielkość znaków i znaki interpunkcyjne):

```
[Clothing]. Material: [material]. Size: [size]. Price: [price].
```

Uwaga. Jeśli materiał lub rozmiar jest napisem pustym (równe `" "`), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwa ubrania są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam materiał i cenę. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Napisz metodę (zwykłą) `wash` (pol. pranie) bez argumentów. Metoda zwiększa cenę o `5.50`.

H. (2pkt) Napisz metodę statyczną `changePrice` (pol. zmień cenę), której argumentem jest obiekt typu `Clothing` i zmienna typu `double`. Metoda ma zmienić cenę w przekazanym jako argument obiekcie na wartość przekazaną przez drugi argument metody. Jeśli drugi argument jest liczbą ujemną, to metoda ma nic nie robić. Jeśli pierwszy argument metody jest `null`, to metoda też ma nic nie robić.

## Zadanie 2. Klasa `Shirt` (Koszula) (13pkt max.)

A. (1pkt) Klasa `Shirt` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.clothing` w innym pliku niż klasa `Clothing`.

B. (2pkt) Klasa `Shirt` dziedziczy po klasie `Clothing`. Klasa powinna posiadać prywatne pola:

- `sleeveLength`, typu `int` (długość rękawa)
- `collarType`, typu `String` (typ kołnierza)

C. (2pkt) Napisz pięcio-argumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B (najpierw z klasy bazowej, potem pochodnej). Zapewnij niezależnie warunki sprawdzające poprawność dodatkowo:

- Długość rękawa `sleeveLength` musi być liczbą dodatnią - w przeciwnym wypadku ustaw ją na `32`.
- Typ kołnierza `collarType` nie może być `null` - w przeciwnym wypadku ustaw go na `"Classic"`.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Powinny mieć taką samą walidację jak w konstruktorze.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu zgodnie z następującym schematem (zwróć uwagę na wielkość znaków, łamanie linii i znaki interpunkcyjne):

```
[Shirt]. Material: [material]. Size: [size]. Price: [price].  
Sleeve Length: [sleeveLength]. Collar Type: [collarType].
```

Uwaga. Podobnie jak w klasie bazowej, jeśli materiał lub rozmiar jest napisem pustym (równe ""), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwie koszule są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam materiał, cenę oraz długość rękawa. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Nadpisz metodę `wash` tak, aby zmniejszała cenę o 15.

H. (2pkt) Napisz metodę statyczną `changePrice` (pol. zmień cenę), której argumentem jest obiekt typu `Shirt` i zmienna typu `double`. Metoda ma zmienić cenę w przekazanym jako argument obiekcie na wartość przekazaną przez drugi argument metody. Jeśli drugi argument jest liczbą ujemną, to metoda ma nic nie robić. Jeśli pierwszy argument metody jest `null`, to metoda też ma nic nie robić.

### **Zadanie 3. Klasa `TestClothing` (pol. klasa testująca dla ubrania) (9pkt max.)**

A. (2pkt) Klasę `TestClothing` umieść bezpośrednio w katalogu `src`. Umieść w tej klasie tylko metodę `main`.

B. (7pkt) W metodzie `main` stwórz 5 obiektów w typach definiowanych w zadaniu 1 i 2. Następnie sprawdź poprawność działania metody `equals` i `hashCode` na co najmniej 5 różnych sposobów.

### **Zadanie 4. Klasa `WaterVehicle` (pol. pojazd wodny) (5pkt max.)**

A. (1pkt) Stwórz abstrakcyjną klasę `WaterVehicle` zawierającą publiczną abstrakcyjną metodę `navigate()`, która przyjmuje jako argument `String destination` (pol. cel podróży) i zwraca `String`. Klasę umieść w pakiecie `maritime`.

B. (2pkt) Utwórz dwie klasy pochodne od `WaterVehicle`: `Boat` i `Submarine`. W obu klasach nadpisz metodę `navigate()`. Dla `Boat` metoda powinna zwracać napis "Navigating to [destination] on the surface", gdzie [destination] to argument metody. Dla `Submarine` metoda powinna zwracać napis "Navigating to [destination] underwater", gdzie [destination] to argument metody.

C. (2pkt) W klasie `TestWaterVehicle` w pakiecie `maritime` utwórz tablicę typu `WaterVehicle` i zainicjalizuj ją różnymi instancjami `Boat` i `Submarine`. Iteruj po tablicy wywołując metodę `navigate()` dla każdego pojazdu wodnego, przekazując jako argument przykładowe destynacje.

