Ćwiczenia obiekty i klasy

Zadanie 1 – Magazyn Produktów

Treść zadania:

Utwórz klasę Produkt z prywatnymi polami:

- String nazwa,
- double cena,
- int ilosc.

Dodaj walidację:

- nazwa nie może być pusta ani null,
- cena > 0,
- ilość ≥ 0.

Dodaj metody:

- konstruktor ustawiający wszystkie pola z walidacją,
- public String getNazwa(),
- public double getCena(),
- public int getIlosc(),
- public void dodaj(int sztuk) – zwiększa ilość (tylko jeśli sztuk > 0),
- public void sprzedaj(int sztuk) – zmniejsza ilość (tylko jeśli sztuk ≤ aktualna ilość, inaczej wyjątek).

Następnie utwórz klasę Magazyn, która ma:

- prywatne pole ArrayList<Produkt> produkty,
- public void dodajProdukt(Produkt p),
- public Produkt znajdzProdukt(String nazwa) – wyszukuje produkt po nazwie, zwraca obiekt lub null,
- public double wartoscMagazynu() – zwraca sumę (cena * ilość) wszystkich produktów.

Wymagania techniczne:

- wszystkie pola prywatne, dostęp przez gettery,

- walidacja w konstruktorze i metodach (IllegalArgumentException dla złych wartości),
- użycie ArrayList<Produkt> do przechowywania w magazynie.

Kroki testowe / przykłady:

- Utwórz produkt Produkt p1 = new Produkt("Laptop", 3000, 10);
- Utwórz magazyn Magazyn m = new Magazyn();
- Dodaj produkt do magazynu m.dodajProdukt(p1);
- Sprawdź wartość magazynu → powinna być 30000.
- Sprzedaj 2 sztuki i sprawdź ilość → 8.
- Spróbuj sprzedać więcej niż masz → powinien być wyjątek.

Zadanie 2 – Rejestr Studentów

Treść zadania:

Utwórz klasę Student z polami:

- String imie, nazwisko,
- int numerIndeksu (stały – ustalany w konstruktorze, tylko getter),
- ArrayList<Double> oceny.

Dodaj metody:

- konstruktor ustawiający imię, nazwisko i numer indeksu,
- public void dodajOcene(double ocena) – walidacja: ocena od 2.0 do 5.0,
- public double srednia() – zwraca średnią ocen (0.0 jeśli brak ocen),
- public String toString() – zwraca dane studenta i jego średnią.

Utwórz klasę RejestrStudentow:

- prywatne pole ArrayList<Student> studenci,
- public void dodajStudenta(Student s) – sprawdza unikalność numeru indeksu (jeśli taki już istnieje → wyjątek),
- public Student znajdzPoIndeksie(int nr) – zwraca studenta lub null,
- public Student najlepszyStudent() – zwraca studenta o najwyższej średniej,
- public void posortujPoNazwisku() – sortuje listę studentów po nazwisku (użyj Collections.sort() + Comparator).

Wymagania techniczne:

- enkapsulacja (wszystko prywatne poza potrzebnymi getterami),
- walidacja ocen przy dodawaniu,
- unikalność indeksu w rejestrze,
- użycie ArrayList<Student> do przechowywania.

Kroki testowe / przykłady:

- Utwórz studentów:
  Student s1 = new Student("Jan", "Kowalski", 1001);
  Student s2 = new Student("Anna", "Nowak", 1002);
- Dodaj im oceny: s1.dodajOcene(5.0); s1.dodajOcene(4.0);
- Dodaj do rejestru: RejestrStudentow r = new RejestrStudentow(); r.dodajStudenta(s1); r.dodajStudenta(s2);
- Znajdź studenta po indeksie r.znajdzPoIndeksie(1002);
- Wyznacz najlepszego studenta – powinna być Anna lub Jan w zależności od średniej,
- Posortuj studentów po nazwisku.

Zadanie 3 – Biblioteka

Treść zadania:

Utwórz klasę Ksiazka z polami:

- String tytul,
- String autor,
- boolean wypozyczona.

Dodaj metody:

- konstruktor ustawiający tytuł i autora (początkowo książka jest dostępna),
- public void wypozycz() – ustawia wypozyczona = true (jeśli już wypożyczona → wyjątek),
- public void oddaj() – ustawia wypozyczona = false (jeśli książka nie była wypożyczona → wyjątek),
- public String status() – zwraca "dostępna" lub "wypożyczona",
- public String toString() – zwraca dane książki + status.

Następnie utwórz klasę Biblioteka, która ma:

- prywatne ArrayList<Ksiazka> ksiazki,
- public void dodajKsiazke(Ksiazka k),
- public void wypozyczKsiazke(String tytul) – wyszukuje książkę i wypożycza,
- public void oddajKsiazke(String tytul) – wyszukuje książkę i oddaje,
- public ArrayList<Ksiazka> listaDostepnych(),
- public ArrayList<Ksiazka> listaWypozyczonych().

Wymagania techniczne:

- wszystkie pola prywatne, dostęp tylko przez metody,
- walidacja operacji (np. nie można oddać książki, która nie jest wypożyczona),
- użycie ArrayList<Ksiazka> do przechowywania,
- wyszukiwanie książek po tytule.

Kroki testowe / przykłady:

- Utwórz książki:
  Ksiazka k1 = new Ksiazka("Pan Tadeusz", "Adam Mickiewicz");
  Ksiazka k2 = new Ksiazka("Lalka", "Bolesław Prus");
- Utwórz bibliotekę: Biblioteka b = new Biblioteka();
- Dodaj książki: b.dodajKsiazke(k1); b.dodajKsiazke(k2);
- Wypożycz "Lalka": b.wypozyczKsiazke("Lalka");
- Spróbuj wypożyczyć "Lalka" drugi raz → powinien być wyjątek,
- Oddaj książkę, sprawdź listę dostępnych i wypożyczonych.

Zadanie 4 – Skup Samochodów

Treść zadania:

Utwórz klasę Samochod z polami:

- String marka,
- String model,
- int rokProdukcji,
- double cenaZakupu.

Dodaj:

- konstruktor ustawiający wszystkie pola z walidacją (rokProdukcji > 1900, cenaZakupu > 0),
- gettery do odczytu,
- public String toString() – zwraca dane samochodu w czytelnej formie.

Utwórz klasę SkupSamochodow:

- prywatne ArrayList<Samochod> samochody,
- public void dodajSamochod(Samochod s),
- public void usunSamochod(String marka, String model) – usuwa pierwszy pasujący samochód,
- public Samochod najdrozszySamochod() – zwraca auto o najwyższej cenie zakupu,
- public ArrayList<Samochod> listaSamochodowZRoku(int rok) – zwraca wszystkie samochody z danego roku produkcji.

Wymagania techniczne:

- walidacja w konstruktorze,
- wszystkie pola prywatne, dostęp tylko przez metody,
- użycie ArrayList<Samochod> do przechowywania.

Kroki testowe / przykłady:

- Utwórz kilka samochodów: Audi A4 2018, Toyota Corolla 2010, BMW X5 2020.
- Dodaj je do skupu.

- Wyznacz najdroższy samochód.
- Usuń Toyotę po nazwie i modelu.
- Pobierz listę samochodów z roku 2018 → powinno być Audi A4.

Zadanie 5 – Wypożyczalnia Filmów

Treść zadania:

Utwórz klasę Film z polami:

- String tytul,
- String gatunek,
- boolean wypozyczony.

Dodaj:

- konstruktor ustawiający tytuł i gatunek,
- public void wypozycz() – ustawia wypozyczony = true (jeśli już wypożyczony → wyjątek),
- public void oddaj() – ustawia wypozyczony = false (jeśli nie był wypożyczony → wyjątek),
- public String status() – zwraca "dostępny" lub "wypożyczony",
- public String toString() – dane filmu i status.

Utwórz klasę Wypozyczalnia:

- prywatne ArrayList<Film> filmy,
- public void dodajFilm(Film f),
- public void wypozyczFilm(String tytul),
- public void oddajFilm(String tytul),
- public ArrayList<Film> filmyWGatunku(String gatunek) – zwraca listę filmów o danym gatunku,
- public ArrayList<Film> listaDostepnych().

Wymagania techniczne:

- walidacja wypożyczania i oddawania (nie można wypożyczyć już wypożyczonego filmu),
- wszystkie pola prywatne, dostęp przez metody,
- użycie ArrayList<Film> do przechowywania.

Kroki testowe / przykłady:

- Dodaj filmy: Matrix – Sci-Fi, Titanic – Dramat, Shrek – Animacja.
- Wypożycz Matrix.
- Spróbuj ponownie wypożyczyć Matrix → wyjątek.
- Oddaj film i sprawdź listę dostępnych.
- Pobierz wszystkie filmy gatunku "Dramat".

Zadanie 6 – Restauracja i Menu

Treść zadania:

Utwórz klasę Danie z polami:

- String nazwa,
- double cena,
- String kategoria (np. "przystawka", "danie główne", "deser").

Dodaj:

- konstruktor ustawiający wszystkie pola z walidacją (cena > 0, nazwa niepusta),
- public String toString() – zwraca dane dania w formie "nazwa – cena zł (kategoria)".

Utwórz klasę MenuRestauracji:

- prywatne ArrayList<Danie> dania,
- public void dodajDanie(Danie d),
- public ArrayList<Danie> daniaZKategorii(String kategoria),
- public Danie najdrozszeDanie(),
- public ArrayList<Danie> wyszukajPoFragmencie(String fraza) – wyszukuje wszystkie dania, których nazwa zawiera frazę (case-insensitive).

Wymagania techniczne:

- walidacja w konstruktorze (IllegalArgumentException dla złych danych),
- wszystkie pola prywatne, dostęp przez metody,
- wyszukiwanie po nazwie ignoruje wielkość liter (toLowerCase()).

Kroki testowe / przykłady:

- Dodaj dania: Pizza Margherita (30 zł, danie główne), Tiramisu (15 zł, deser), Bruschetta (12 zł, przystawka).
- Pobierz wszystkie desery → Tiramisu.
- Wyznacz najdroższe danie → Pizza Margherita.
- Wyszukaj po fragmencie "zza" → powinna znaleźć Pizza Margherita.