Zadania na zajęcia Live Coding do bloku “Java Zaawansowana - programowanie”

Zadanie 1.

Stwórz metodę, która jako parametr przyjmuje listę stringów, następnie zwraca tą listę posortowaną alfabetycznie od Z do A.

Zadanie 2.

Stwórz metodę, która jako parametr przyjmuje listę stringów, następnie zwraca tą listę posortowaną alfabetycznie od Z do A nie biorąc pod uwagę wielkości liter.

Zadanie 3.

Stwórz metodę, która jako parametr przyjmuje mapę, gdzie kluczem jest string, a wartością liczba, a następnie wypisuje każdy element mapy do konsoli w formacie: **Klucz: <k>, Wartość: <v>**. Na końcu każdego wiersza poza ostatnim, powinien być przecinek, a w ostatnim kropka.

Przykład:

Klucz: Java, Wartość: 18**,**
Klucz: Python, Wartość: 1**,**
…
Klucz: PHP, Wartość: 0**.**

Zadanie 4.

Stwórz klasę **Storage**, która będzie miała prywatne pole typu **Map**, publiczny konstruktor oraz metody:

addToStorage(String key, String value) → dodawanie elementów do przechowywalni

printValues(String key) → wyświetlanie wszystkich elementów pod danym kluczem

findValues(String value) → wyświetlanie wszystkich kluczy, które mają podaną wartość

Klasa **Storage** powinna pozwalać na przechowywanie wielu wartości pod jednym kluczem.

Zadanie 5.

Zaimplementuj klasę **SDAHashSet<E>** , która będzie implementować logikę **HashSet<E>**. W tym celu zaimplementuj obsługę metod:

- add
- remove
- size
- contains
- clear

Zadanie 6.

Stwórz metodę, która przyjmuje **TreeMap** i wypisuje w konsoli pierwszy i ostatni **EntrySet**.

Zadanie 7.

Stwórz klasę imitującą magazynek do broni. Klasa powinna mieć możliwość definiowania rozmiaru magazynka za pomocą konstruktora. Zaimplementuj metody:

loadBullet(String bullet) → dodawanie naboju do magazynka, nie pozwala załadować więcej naboi niż wynosi pojemność magazynka

isLoaded() → zwraca informację o tym czy broń jest naładowana (przynajmniej jeden nabój) czy nie

shot() → jedno wywołanie wystrzeliwuje (wypisuje w konsoli wartość string) jeden - ostatni załadowany nabój i przygotowuje kolejny, załadowany przed ostatnim, jeżeli nie ma więcej nabojów to wypisuje w konsoli “pusty magazynek”

Zadanie 8.

Zaimplementuj interfejs **Validator**, który będzie zawierać w swojej deklaracji metodę **boolean validate(Parcel input)**. Stwórz klasę **Parcel** z parametrami:

- int **xLength**
- int **yLength**
- int **zLength**
- float **weight**
- boolean **isExpress**

Validator powinien weryfikować czy suma wymiarów (x, y, z) nie przekracza 300, czy każdy z rozmiarów nie jest mniejszy niż 30, czy waga nie przekracza 30.0 dla isExpress=false lub 15.0 dla isExpress=true. W przypadku błędów, powinien o nich poinformować użytkownika.

Zadanie 9.

Stwórz klasę **Point2D** posiadającą pola **double x**, **double y**, gettery, settery oraz konstruktor. Następnie stwórz klasę **Circle**, która będzie miała konstruktor:

Circle(Point2D center, Point2D point)

Pierwszy parametr określa środek koła, drugi jest dowolnym punktem na okręgu. Na podstawie tych punktów, klasa **Circle** ma określać:

- promień okręgu przy wywołaniu metody **double getRadius()**
- obwód okręgu przy wywołaniu metody **double getPerimeter()**
- pole okręgu przy wywołaniu metody **double getArea()**
- *(trudniejsze) trzy punkty na okręgu co 90 stopni od punktu podanego przy wywołaniu metody **List<Point2D> getSlicePoints()**

Zadanie 10.

Stwórz klasę **MoveDirection** posiadającą pola **double x**, **double y** oraz gettery, settery i konstruktor. Stwórz interfejs **Movable** posiadający metodę **move(MoveDirection moveDirection)**.

Zaimplementuj interfejs w klasach z poprzedniego zadania (**Point2D** oraz **Circle**). Przy wywołaniu metody **move(MoveDirection moveDirection)**, obiekty mają zmienić swoje położenie na podstawie przekazanego kierunku (**MoveDirection**).

Sprawdź poprawność przesunięcia wywołując pozostałe metody klasy **Circle**.

Zadanie 11.

Stwórz interfejs **Resizable** posiadający metodę **resize(double resizeFactor)**.

Zaimplementuj interfejs w klasie z poprzedniego zadania (**Circle**). Przy wywołaniu metody **resize(double resizeFactor)**, okrąg ma zmienić swoje rozmiary o zadany współczynnik (1.5, 0.5, 10.0, itp).

Sprawdź poprawność zmiany rozmiaru wywołując pozostałe metody klasy **Circle**.

Zadanie 12.

Stwórz klasę **Manufacturer**, która będzie zawierać pola: nazwa, rok założenia, kraj. Uwzględnij wszystkie niezbędne meotody oraz parametry konstruktora. Zaimplementuj metody **hashCode()** i **equals()** .

Stwórz klasę **Car** , która będzie zawierać pola: nazwa, model, cena, rok produkcji, lista producentów (**Manufacturer**), oraz typ silnika (reprezentowany jako klasa enum, np. V12, V8, V6, S6, S4, S3). Uwzględnij wszystkie niezbędne metody oraz parametry konstruktora. Zaimplementuj metody **hashcode()** i **equals()**.

Zadanie 13.

Stwórz klasę **CarService** , która będzie zawierać w sobie listę aut, oraz będzie realizować poniższe metody:

1. dodawanie aut do listy,
2. usuwanie auta z listy,
3. zwracanie listy wszystkich aut,
4. zwracanie aut z silnikiem V12,
5. zwracanie aut wyprodukowanych przed rokiem 1999,
6. zwracanie najdroższego auta,
7. zwracanie najtańszego auta,
8. zwracanie auta z co najmniej 3 producentami,
9. zwracanie listy wszystkich aut posortowanych zgodnie z przekazanym parametrem: rosnąco/malejąco,
10. sprawdzanie czy konkretne auto znajduje się na liście,
11. zwracanie listy aut wyprodukowanych przez konkretnego producenta,
12. zwracanie listy aut wyprodukowanych przez producenta z rokiem założenia <,>,<=,>=,=,!= od podanego.

Zadanie 14.

Na podstawie 100000 elementowej tablicy z losowo wybranymi wartościami zaimplementuj następujące funkcjonalności:

1. zwróć listę unikalnych elementów,
2. zwróć listę elementów, które co najmniej raz powtórzyły się w wygenerowanej tablicy,
3. zwróć listę 25 najczęściej powtarzających się elementów.

Zaimplementuj metodę, która deduplikuje elementy w liście. W przypadku znalezienia duplikatu, zastępuje go nową losową wcześniej nie występującą wartością. Sprawdź czy metoda zadziałała poprawnie wywołując metodę numer 2.

Zadanie 15.

Stwórz klasę **enum Car** ze stałymi FERRARI, PORSCHE, MERCEDES, BMW, OPEL, FIAT, TOYOTA, itp. Każdy z pojazdów ma własne parametry np. cena, moc, itp. Enum powinien zawierać metody **boolean isPremium()** oraz **boolean isRegular()**. Metoda **isPremium()** powinna zwracać rezultat przeciwny od rezultatu wywołania metody **isRegular()**.

Dodatkowo w ramach klasy enum powinna być zadeklarowana i zaimplementowana metoda **boolean isFasterThan()**. Metoda ta powinna przyjmować obiekt typu **Car** oraz wyświetlać informacje o tym, że wskazany pojazd jest szybszy bądź nie od pojazdu przekazanego w argumencie. W tym celu skorzystaj z metody **compareTo()**.

Zadanie 16.

Stwórz klasę **enum Runner** ze stałymi BEGINNER, INTERMEDIATE, ADVANCED. Enum powinien posiadać dwa parametry:

- minimalny czas przebiegnięcia maratonu w minutach
- maksymalny czas przebiegnięcia maratonu w minutach

Dodatkowo **enum Runner** powinien zawierać metodę statyczną **getFitnessLevel()**, która przyjmuje na wejściu dowolny czas przebiegnięcia maratonu, a jako rezultat powinna zwracać konkretny obiekt **Runner** na podstawie przekazanego czasu.

Zadanie 17.

Stwórz klasę **enum ConversionType** ze stałymi METERS_TO_YARDS, YARDS_TO_METERS, CENTIMETERS_TO_ICHES, INCHES_TO_CENTIMETERS, KILOMETERS_TO_MILES, MILES_TO_KILOMETERS. Enum powinien posiadać parametr typu **Converter** służący do przeprowadzania obliczeń dla danego typu.

Następnie stwórz klasę **MeasurementConverter**, która będzie posiadała metodę **convert(int value, ConvertionType conversionType)** i na podstawie przekazanej wartości oraz typu konwersji, korzystała z **Convertera** danego typu i zwracała wynik.

Zadanie 18.

Stwórz klasę **Computer** posiadającą pola określające cechy komputera: procesor, ram, karta grafiki, firma oraz model. Zaimplementuj settery, gettery, konstruktor z wszystkimi polami, metody **toString()** oraz **equals()** i **hashcode()**.

Zainstancjuj kilka obiektów i sprawdź działanie metod.

Zadanie 19.

Stwórz klasę **Laptop** rozszerzającą klasę **Computer** z poprzedniego zadania. Klasa **Laptop** powinna dodatkowo zawierać parametr bateria.

Zaimplementuj dodatkowe gettery, settery, konstruktor oraz odpowiednio nadpisz metody **toString()** oraz **equals()** i **hashcode()**.

Użyj odniesienia do metod klasy nadrzędnej.

Zadanie 20.

Stwórz abstrakcyjną klasę **Shape** wraz z abstrakcyjnymi metodami **calculatePerimeter()** służącą do obliczania obwodu oraz **calculateArea()** służącą do obliczania powierzchni.

Stwórz klasy **Rectangle, Triangle, Hexagon**, rozszerzając klasę **Shape**, odpowiednio implementując metody abstrakcyjne. Sprawdź poprawność działania.

Zadanie 21.

Stwórz abstrakcyjną klasę **3DShape** rozszerzającą klasę **Shape** z poprzedniego zadania. Dodaj dodatkową metodę **calculateVolume()**.

Stwórz klasy **Cone** oraz **Qube** rozszerzając klasę **3DShape**, odpowiednio implementując metody abstrakcyjne. Sprawdź poprawność działania.

Zadanie 22.

Stwórz interfejs **Fillable** posiadający metodę **fill()**. Zaimplementuj metodę w klasie **3DShape** z poprzedniego zadania lub osobno w klasach **Cone** oraz **Qube**.

Metoda **fill()** powinna przyjmować parametr np. int i sprawdzać, czy po, akcja napełnienia figury:

- wleje za dużo wody do figury i przeleje,
- napełni figurę wodą po brzegi,
- wleje za mało wody.

Dla każdej sytuacji powinna zapisać informację o stanie w konsoli. Użyj metody **calculateVolume()**.

Zadanie 23.

Stwórz klasę **Zoo**, która będzie posiadała zbiór zwierząt oraz pozwalała na otrzymywanie statystyk na temat posiadanych zwierząt:

int getNumberOfAllAnimals() → zwraca liczbę wszystkich zwierząt

Map<String, Integer> getAnimalsCount() → zwraca liczbę zwierząt każdego gatunku

Map<String, Integer> getAnimalsCountSorted() → zwraca liczbę zwierząt każdego gatunku posortowaną na podstawie ilości zwierząt danego gatunku, gdzie pierwszym elementem zawsze jest gatunek z największą ilością zwierząt

void addAnimals(String, int) → dodaje n zwierząt danego gatunku

Zadanie 24.

Stwórz klasę **Basket**, która imituje koszyk i przechowuje aktualną ilość elementów w koszyku. Dodaj metodę **addToBasket()**, która dodaje element do koszyka (zwiększając aktualny stan o 1) oraz metodę **removeFromBasket()**, która usuwa element z koszyka (zmniejszając aktualny stan o 1).

Koszyk może przechowywać od 0 do 10 elementów. W przypadku, kiedy użytkownik chce wykonać akcję usunięcia przy stanie 0 lub dodania przy stanie 10, rzuć odpowiedni runtime exception (**BasketFullException** lub **BasketEmptyException**).

Zadanie 25.

Zamień wyjątki **BasketFullException** oraz **BasketEmptyException** z poprzedniego zadania na wyjątki typu checked exception.

Obsłuż je.

Zadanie 26a.

Wykorzystując mechanizmy programowania funkcyjnego na podstawie zadanej struktury wyświetl:

1. listę wszystkich Modelów,
2. listę wszystkich aut,
3. listę wszystkich nazw producentów,
4. listę wszystkich lat założenia producentów,
5. listę wszystkich nazw modeli,
6. listę wszystkich lat startu produkcji modeli,
7. listę wszystkich nazw aut,
8. listę wszystkich opisów aut,
9. tylko modele z parzystym rokiem startu produkcji,
10. tylko auta producentów z parzystym rokiem założenia,
11. tylko auta z parzystym rokiem startu produkcji modelu oraz nieparzystym rokiem założenia producenta,
12. tylko auta typu CABRIO z nieparzystym rokiem startu produkcji modelu i parzystym rokiem założenia producenta,
13. tylko auta typu SEDAN z modelu nowszego niż 2019 oraz rokiem założenia producenta mniejszym niż 1919

Zadanie 26b.

```
enum CarType {

COUPE, CABRIO, SEDAN, HATCHBACK

}

class Car {
   public String name;
   public String description;
   public CarType carType;


public Car(String name, String description, CarType carType) {

this.name = name;

this.description = description;

this.carType = carType;

}}
```

```
class Model {
   public String name;
   public int productionStartYear;
   List<Car> cars;


public Model(String name, int productionStartYear, List<Car> cars) {

this.name = name;

this.productionStartYear = productionStartYear;

this.cars = cars;

} }

class Manufacturer {
   public String name;
   public int yearOfCreation;
   List<Model> models;


public Manufacturer(String name, int yearOfCreation, List<Model> models) {

this.name = name;

this.yearOfCreation = yearOfCreation;

this.models = models;

} }
```

Zadanie 27.

Zaprojektuj klasę **Joiner<T>**, która w konstruktorze będzie przyjmowała separator (string) oraz posiadała metodę **join()** pozwalającą na podanie dowolnej ilości obiektów typu **T**. Metoda ta będzie zwracać stringa łącząc wszystkie przekazane elementy wywołując ich metodę **toString()** i oddzielać je separatorem.

np. dla Integerów oraz separatora “-” będzie zwracała: 1-2-3-4…

Zadanie 28.

Rozszerz klasę **ArrayList<E>** implementując metodę **getEveryNthElement()**. Metoda ta zwraca listę elementów oraz przyjmuje dwa parametry: indeks elementu od którego zaczyna oraz liczbę określającą co który element ma wybierać.

Dla listy: [a, b, c, d, e, f, g] oraz parametrów: startIndex=2 i skip=3 zwróci listę [c, g]

Zadanie 29.

Zaimplementuj generyczną metodę **partOf**, która na podstawie tablicy dowolnego typu oraz wskazanej funkcji jako parametrów zwróci **%** elementów spełniających warunek.

Zadanie 30.

Napisz program, który odczyta dowolny plik i zapisze go w tym samym folderze. Zawartość jak i tytuł nowego pliku powinny być odwrócone (od tyłu).

Zadanie 31.

Napisz program, który policzy wystąpienia każdego słowa w pliku tekstowym a następnie wyniki zapisze w formie tabelki w nowym pliku.

Zadanie 32.

Napisz program, który będzie w stanie zapisać listę elementów (np. aut) do pliku w takim formacie, aby był również w stanie odczytać ten plik i stworzyć nową listę elementów i wypisać ją w konsoli.

Zadanie 33.

Napisz program, który wyświetli wszystkie zdjęcia (.png, .jpg) w danym katalogu i podkatalogach.

Zadanie 34.

Utwórz klasę rozszerzającą klasę **Thread** np. **MyThread**, przeciąż metodę **run()**, w której wyświetlisz w konsoli nazwę wątku odczytując ją ze statycznej metody aktualnego wątku:

```
Thread.currentThread().getName()
```

Utwórz klasę z metodą public **static void main(String[] args)**. Wewnątrz metody main utwórz zmienną typu naszej klasy stworzonej powyżej np. **MyThread** oraz zainicjuj klasę. Na zmiennej wykonać metodę **start()**.

Zadanie 35.

Utwórz klasę implementującą **Runnable** np. **MyRunnable**. Zaimplementuj metodę **run()**, która wyświetli nam nazwę aktualnego wątku analogicznie jak w poprzednim ćwiczeniu.

Utwórz klasę z metodą **public static void main(String[] args)**. Wewnątrz metody main utwórz zmienną typu klasy stworzonej powyżej np. **MyRunnable** oraz zainicjuj klasę.

Utwórz zmienną typu **Thread** i zainicjuj ją przekazując jako parametr konstruktora implementację interfejsu **Runnable**. Na zmiennej typu **Thread** wykonać metodę **start()**.

Zadanie 36.

Utwórz klasę **ThreadPlaygroundRunnable** implementującą interfejs **Runnable** posiadająca pole **name** typu **String** z konstruktorem publicznym dla tego pola. Klasa powinna zaimplementować metodę **run()** z interfejsu Runnable. Metoda ta powinna zawierać pętlę liczącą do 10 oraz drukować nazwę aktualnego wątku korzystając z **Thread.currentThread().getName()**, nazwę nadaną w konstruktorze oraz numer aktualnie wykonywanej iteracji pętli.

Utwórz klasę, która ma dwa prywatne statyczne pola typu **Thread** (wątek) oraz standardową metodę **public static void main(String[] args)**. Następnie zainicjuj pola typu **Thread** używając konstruktora przyjmującego obiekt **Runnable** i przekaż **ThreadPlaygroundRunnable** tworząc go za pomocą konstruktora, za każdym razem podając inną nazwę.

Na każdym z wątków (Thread) użyj metody **start()**.

Zadanie 37.

Stwórz klasę zawierającą standardową metodę statyczną main oraz zmienną typu **Executor** i korzystając z metody fabrykującej **newFixedThreadPool** klasy **Executors** utwórz pulę executorów o rozmiarze 2.

W iteracji dodaj do Executora 10 obiektów klasy **ThreadPlaygroundRunnable** z poprzedniego zadania. Jako nazwy użyj dowolnego ciągu znaków oraz numeru aktualnej iteracji.

Zadanie 38.

Napisz aplikację, która będzie symulować maszynę do robienia kawy. W przypadku, gdy dowolna cykliczna usługa parzenia kawy zastanie pusty zbiornik na wodę, powinno nastąpić wstrzymanie wątku. W momencie, gdy w maszynie zostanie uzupełniona woda, powinno nastąpić wzbudzenie wątku.

Zadanie 39.

Napisz program, który ma za zadanie rozwiązać poniższy problem.

Istnieje system przechowujący aktualne wyniki w zawodach. Wiele ekranów odczytuje aktualne wyniki, natomiast kilka czujników aktualizuje te wyniki. Napisz program, który pozwala na nieprzerwany odczyt danych przez wiele ekranów (wątków) oraz aktualizację danych przez wiele czujników (wątków).

Aby zaktualizować wyniki, czujnik musi przekazać aktualne wyniki wraz z nowymi. System następnie weryfikuje czy czujnik miał aktualne dane i aktualizuje dane. Jeżeli w tym czasie dane w systemie zmieniły się, aktualizacja danych przez czujnik jest odrzucona.

Dla ułatwienia możesz przyjąć, że czujnik odczytuje dane, czeka losową ilość czasu i zwiększa dane o losową wartość.