Laboratorium 3

W tym ćwiczeniu należy zbudować aplikację z okienkowym interfejsem użytkownika, która ilustruje sposób działania kolekcji zawartych w pakiecie „*Java Collections Framework*". Kolekcje są to obiekty tworzące abstrakcyjne struktury danych, w których możliwe jest gromadzenie innych obiektów. Na kolekcjach można wykonywać podstawowe operacje takie jak dodawanie, usuwanie oraz przeglądanie elementów . Główne cele ćwiczenia to:

- nabycie umiejętności tworzenia aplikacji wykorzystujących graficzny interfejs użytkownika,
- nabycie umiejętności implementacji interfejsów,
- porównanie właściwości kolekcji różnych typów,
- nabycie umiejętności tworzenia i operowania na kolekcjach z pakietu „*Java Collections Framework*",

Program przykładowy

Program przykładowy jest rozwinięciem programu z poprzedniego ćwiczenia i jest podzielony na dwa bloki:

- **model danych:** plik `Person.java` oraz `GroupOfPeople.java`
- **okienkowy interfejs użytkownika:** pliki `GroupManagerApp.java` i `GroupOfPeopleWindowDialog.java` oraz `PersonWindowDialog.java`

W skład modelu danych oprócz klas z poprzedniego ćwiczenia (typ wyliczeniowy `enum PersonJob`, klasa `PersonException,` klasa `Person`) wchodzi typ wyliczeniowy `enum GroupType` oraz klasa `GroupOfPeople.` Typ `GroupType` reprezentuje typy kolekcji, które mogą być wykorzystane do tworzenia grupy osób. W programie można wybrać dwa rodzaje kolekcji: Listy i Zbiory. Każdy rodzaj kolekcji może być implementowany przy pomocy różnych klas:

- Listy: klasa `Vector`, klasa `ArrayList`, klasa `LinnkedList`;
- Zbiory: klasa `TreeSet`, klasa `HashSet`.

Typ wyliczeniowy `GroupType` zawiera m.in. metodę `createCollection()`, której zadaniem jest tworzenie obiektu kolekcji przy pomocy konstruktora tej klasy, która jest skojarzona z konkretnym typem kolekcji. Metoda ta powinna być wywoływana zawsze wtedy gdy jest tworzona nowa grupa osób lub modyfikowany jest typ istniejącej grupy.

Główną klasą modelu danych w tym programie jest klasa `GroupOfPeople`, która reprezentuje grupy osób opisane za pomocą trzech atrybutów:

- `name` – nazwa grupy wybierana przez użytkownika (musi zawierać niepusty ciąg znaków),
- `type` – typ kolekcji, która ma być użyta do zapamiętania danych osób należących do tej grupy.
- `collection` - kolekcja obiektów klasy `Person`, w której pamiętane są dane osób należących do tej grupy (musi być to obiekt utworzony za pomocą metody `createCollection` z typu wyliczeniowego `GroupType`).

Klasa zawiera typowe konstruktory, gettery i settery dla atrybutów `name` i `type`. Dla atrybutu `collection` zamiast gettera klasa udostępnia trzy tzw. metody delegowane z interfejsu `Collection<Person>` (`add`, `iterator` oraz `size`), które umożliwiają wykonywanie operacji na kolekcji obiektów. Ponadto zdefiniowano cztery pomocnicze

metody pozwalające na sortowanie listy osób według różnych kryteriów. Pierwsza metoda do porównywania obiektów wykorzystuje metodę *compareTo*, będącą implementacją interfejsu *Comparable<Person>* w klasie *Person*. Pozostałe trzy metody do porównywania obiektów wykorzystują dedykowany komparator utworzony jako obiekt klasy anonimowej, która implementuje interfejs *Comparator<Person>*.

Usługowy charakter mają statyczne metody *printToFile* i *readFromFile*, które umożliwiają zapis i odczyt danych grupy do/z pliku tekstowego.

Ostatnią częścią definicji klasy są cztery statyczne metody do tworzenia specjalnych grup, które są wynikiem wykonania wybranych operacji na dwóch grupach źródłowych. Możliwe są następujące operacje:

- SUMA – grupa osób zawierająca wszystkie osoby z grupy pierwszej oraz wszystkie osoby z grupy drugiej;
- ILICZYN – grupa osób, które należą zarówno do grupy pierwszej jak i do grupy drugiej;
- RÓŻNICA – grupa osób, które należą do grupy pierwszej i nie ma ich w grupie drugiej
- RÓŻNICA SYMETRYCZNA – grupa osób, które należą do grupy pierwszej i nie ma ich w grupie drugiej oraz te osoby, które należą do grupy drugiej i nie ma w grupie pierwszej.

Nazwa grupy specjalnej jest tworzona według następującego wzorca:

"(nazwa1 NNN nazwa2)"

gdzie:

- *nazwa1* – nazwa pierwszej grupy osób,
- *nazwa2* – nazwa drugiej grupy osób,
- *NNN* – symbol operacji wykonywanej na grupach osób:
  OR – dla operacji typu SUMA,
  AND – dla operacji typu ILOCZYN,
  SUB – dla operacji typu RÓŻNICA,
  XOR – dla operacji typu RÓŻNICA SYMETRYCZNA.

Typ grupy specjalnej zależy od typu grup źródłowych i jest wybierany według następujących reguł:

- jeśli obie grupy źródłowe są tego samego rodzaju (lista lub zbiór) to grupa wynikowa ma taki typ jak pierwsza grupa źródłowa,
- jeśli grupy źródłowe różnią się rodzajem (jedna jest listą, a druga zbiorem) to grupa wynikowa ma taki sam typ jak grupa źródłowa, która jest zbiorem.

Cała klasa *GroupOfPeople* została zdefiniowana w ten sposób by nie zawierała jakichkolwiek operacji związanych implementacją interfejsu użytkownika. Wszystkie metody w razie potrzeby zgłaszają wyjątek klasy *PersonException* z komunikatem tekstowym wskazującym przyczynę błędu. Komunikat ten będzie wyświetlany w modułach realizujących dialog z użytkownikiem.

Moduły interfejsu użytkownika pozwalają na przetestowanie wszystkich operacji wykonywanych na obiektach klasy *Person* oraz *GrupOfPeople*. Plik `PersonWindowDialog.java` (skopiowany z poprzedniego ćwiczenia) zawiera implementację okna dialogowego umożliwiającego tworzenie i modyfikowanie obiektu reprezentującego pojedynczą osobę. Plik `GroupOfPeopleWindowDialog.java` zawiera implementację okna dialogowego umożliwiającego wykonywanie wszystkich operacji na pojedynczej grupie osób. Główne okno aplikacji (klasa *GroupManagerApp*) umożliwia następujące operacje:

1) tworzenie nowej grupy osób,

2) modyfikację istniejącej grupy osób,
3) usuwanie wybranej grupy osób
4) wczytanie pojedynczej grupy osób z pliku tekstowego,
5) zapis wybranej grupy do pliku tekstowego
6) tworzenie grup specjalnych, które są wynikiem wykonania opisanych wcześniej operacji na dwóch wskazanych przez użytkownika grupach.

Przy zamykaniu aplikacji `GroupManagerApp.jar` program automatycznie zapisuje w pliku binarnym o nazwie `LISTA_GRUP.BIN` wszystkie dane o aktualnie utworzonych grupach. Dane te zostaną automatycznie wczytane przy ponownym uruchomieniu aplikacji. Do zapisu został wykorzystany mechanizm serializacji. By to umożliwić klasy `Person` oraz `GroupOfPeople` implementują interfejs `Serializable`.

Zadanie 0 (nie wymaga wysyłania do oceny)

Proszę zapoznać się z podstawowymi interfejsami zawartymi w pakiecie Java Collections Framework oraz klasami implementującymi te interfejsy. Główne interfejsy `to Collection`, `List`, `Set`. Najważniejsze klasy implementujące te interfejsy to: `Vector`, `ArrayList`, `LinkedList`, `TreeSet`, `HashSet`. Przystępne omówienie właściwości poszczególnych kolekcji jest w książce Krzysztofa Barteczko pt. „Java, Programowanie praktyczne od podstaw”, rozdział 6.2 „Wprowadzenie do kolekcji”. Proszę zapoznać się z zasadami, które należy przestrzegać przy przedefiniowywaniu metod `equals` i `hashCode` oraz przy implementacji interfejsu `Comparable` w klasach obiektów, które mają być zapamiętywane w kolekcjach. Zagadnienia te są bardzo precyzyjnie omówione w książce Joshua Blocha pt. „Java Efektywne programowanie”, Temat 8: Zachowanie założeń w trakcie przedefiniowywania metody `equals`, Temat 9: Przedefiniowywanie metody `hashCode` wraz z `equals`, Temat 12: Implementacja interfejsu `Comparable`

Proszę uruchomić program **`GroupManagerApp.jar`** i szczegółowo zapoznać się z jego działaniem. W czasie testów proszę utworzyć kila różnych grup osób. W grupach proszę wpisać po kilka osób. Niektóre osoby powinny się powtarzać. Po utworzeniu grupy typu Lista w której niektóre osoby się powtarzają, proszę zmienić typ grupy na Zbiór . Następnie proszę przywrócić typ Lista. Proszę wyjaśnić przyczynę zaistniałych zmian. Proszę zaobserwować różnice między działaniem kolekcji typu `TreeSet` i `HashSet`. Proszę wyjaśnić przyczyny zaobserwowanych różnic.

Proszę przeanalizować szczegółowo kod źródłowy Typu wyliczeniowego `enum GroupType` i klasy `GroupOfPeople` oraz klasy `GroupManagerApp`.

Zadanie 1 (obowiązkowe)

1. Proszę rozbudować klasę reprezentującą wybrane obiekty, utworzoną w ramach poprzedniego ćwiczenia. W klasie proszę zaimplementować interfejs `Serializable` oraz przedefiniować następujące metody (które były pierwotnie zdefiniowane w klasie `Object`):
   – `toString` – metoda, która zwraca reprezentację tekstową obiektu w postaci łańcucha,

– *hashCode* – metoda, która zwraca wartość kodu mieszania obiektu obliczoną na podstawie wartości kodu mieszania atrybutów jednoznacznie identyfikujących obiekt,
– *equals* – metoda, która porównuje obiekty tej klasy (zwraca wartość *true* dla obiektów reprezentujących takie same obiekty na podstawie porównania atrybutów jednoznacznie identyfikujących obiekt,
– *compareTo* (implementacja interfejsu *Comparable*), – metoda, która porównuje naturalny porządek obiektów (np. uporządkowanie alfabetyczne).

2. Wzorując się na klasie *GroupOfPeople* proszę napisać własną klasę umożliwiająca reprezentację grup obiektów, z użyciem kolekcji typu Lista i kolekcji typu Zbiór

3. Wzorując się na programie **GroupOfPeopleWindowDialog.jar** proszę napisać własną aplikację (okno dialogowe) umożliwiającą tworzenie i modyfikowanie pojedynczej grupy obiektów. Okno powinno umożliwiać następujące operacje:

   - zmianę nazwy grupy
   - zmianę typu kolekcji
   - dodawanie nowego obiektu do grupy,
   - modyfikację wybranego w grupie obiektu
   - usuwanie wybranego obiektu
   - zapis oraz odczyt obiektu do/z pliku tekstowego lub binarnego
   - sortowanie listy obiektów według wybranego kryterium.

4. Wzorując się na programie **GroupManagerApp.jar** proszę napisać własną aplikację umożliwiającą zarządzanie grupami. Aplikacja powinna umożliwiać następujące operacje:
   - tworzenie nowej grupy,
   - modyfikację wybranej grupy (należy wykorzystać okno dialogowe z poprzedniego punktu)
   - usuwanie wybranej grupy,
   - zapis oraz odczyt pojedynczej grupy do/z pliku tekstowego lub binarnego.

Zadanie 2 (dla ambitnych)

Proszę rozbudować program z poprzedniego zadania, tak by umożliwiał tworzenie grup specjalnych będących wynikiem wykonania operacji typu SUMA, ILOCZYN, RÓŻNICA, RÓŻNICA SYMETRYCZNA na dwóch wskazanych grupach obiektów.

Zadanie 3 (dla bardzo ambitnych)

Program w wersji podstawowej umożliwia utworzenie grupy specjalnej. Ale tak utworzona grupa nie jest później aktualizowana, gry modyfikacji ulegnie któraś z grup źródłowych. Proszę rozbudować program tak, by po każdej modyfikacji grupy źródłowej, każda grupa specjalna utworzona na podstawie tej modyfikowanej grupy źródłowej była aktualizowana.

Wskazówki pomocnicze:

W tym celu należy zaimplementować wzorzec projektowy „Obserwator”. Można do tego wykorzystać gotową klasę *Observable* oraz interfejs *Observer* z pakietu JDK.

Klasa reprezentujące grupy obiektów powinna dziedziczyć po klasie *Observable*. W tej klasie po każdej zmianie stanu należy wywołać metody *setChanged()* oraz *notifyObservers().*

Klasa reprezentująca grupy specjalne powinna dziedziczyć po klasie reprezentującej zwykłe grupy i dodatkowo implementować interfejs *Observer*. Interfejs ten zawiera metodę *update*, która będzie wywoływana, gdy w obiekcie obserwowanym nastąpi zmiana stanu. Konstruktor dla grupy specjalnej powinien zapamiętać w dodatkowych polach referencje do grup źródłowych oraz dla tych grup powinien wywołać metodę *addObserver* z parametrem *this*. Metoda *update* powinna być tak zaimplementowana, by za każdym razem korzystając z zapamiętanych referencji do grup źródłowych aktualizować zawartość grupy.

Przed usunięciem grupy specjalnej, należy dla grup źródłowych wywołać metodę *deleteObserver*.