- Zadanie 1
  Stwórz dwie klasy: *PoliczZnaki.java* oraz *PoliczZnakiMain.java*.

  PoliczZnaki ma wczytać zawartość jakiegoś pliku tekstowego. Twój program ma policzyć ile ten wczytany tekst zawiera:

  - Dużych liter
  - Małych liter
  - „Spacji" oraz „tabów"
  - Pozostałych znaków, takich jak kropki, przecinki, nawiasy, itp.

  Poniżej przykład demonstrujący jak uruchomienie programu może wyglądać:

  Ilość dużych liter: 52
  Ilość małych liter: 2063
  Ilość "spacji": 463
  Pozostałe znaki: 83

  Pamiętaj, aby twój program „złapał" *exceptions* które mogą wystąpić. Klasa *PoliczZnakiMain* ma zademonstrować jak działa klasa *PoliczZnaki*.

- **Zadanie 2**
  Dla ułatwienia i organizacji plików, stwórz nowy folder o nazwie *Stack*. Zapisz później wszystkie klasy związane z tym ćwiczeniem do tego folderu.

  Słowo „Stack" (po polsku stos) oznacza strukturę danych „LIFO" (Last-in, First-out). Pozwala ona na trzy operacje: push, pop oraz peek. Metoda push sprawia że kładziemy jakiś element na stosie. Dzięki Push możemy dodawać coraz to kolejne elementy do stosu. Pop oznacza że usuwamy i zwracamy górny element. Peek sprawia że zwracamy ten górny element, ale go nie usuwamy.

  Wyobraź sobie stos talerzy przy zlewozmywaku. Kiedy dodajesz talerze to je układasz jeden na drugim. Ale gdybyś chciał usunąć jakiś talerz, możesz wziąć tylko ten górny. Nie możesz usunąć talerza, co się znajduje w środku stosu. Struktura danych „Stack" ma symbolizować właśnie takie talerze w restauracji.

  Twoim zadaniem jest stworzyć klasę *StackClass.java* która wdraża poniższy interfejs:

```
public interface Stack {
int size();                          // Aktualny rozmiar stosu
boolean isEmpty();                   // Prawda jeśli stos jest pusty
void push(Object element);           // Dodaje element do stosu
Object pop();                        // Zwraca I usuwa element ze stosu
                                     // wyrzuca exception jeśli stos jest pusty
Object peek();                       // Zrwaca (ale nie usuwa) górny element
                                     // wyrzuca exception jeśli stos jest pusty
Iterator iterator();                 // Iterator
}
```

Iterator przechodzi z jednego elementu w stosie na drugi element. Stwórz klasę *StackMain.java* która demonstruje jak wszystkie metody działają. Uwaga: Do tego ćwiczenia nie wolno Ci używać już wbudowanych klas które są bibliotece Javy (nawet ArrayList). Ale za to możesz używać tabel.

- **Zadanie 3**

  Dla ułatwienia stwórz nowy folder o nazwie Miasta. Wszystkie pliki związane z tym ćwiczeniem zapisuj w tym folderze.

  Stwórz klasę *Miasto.java* która będzie reprezentowało jakieś Miasto.

  Napisz także klasę *SortujMiasta.java* która będzie wczytywała ilość miast z jakiegoś pliku tekstowego na Twoim komputerze. Ten plik tekstowy ma zawierać kod pocztowy danego miasta i nazwę miasta, które są przedzielone średnikiem. Dla przykładu zawartość tego pliku może wyglądać tak:

  32300;Olkusz
  32310;Klucze
  40209;Sosnowiec

  I tak dalej. Dla ułatwienia możesz usunąć myślnik między kodami pocztowymi.

  Klasa *SortujMiasta* ma wczytywać te dane z pliku tak, aby kod pocztowy był liczbą całkowitą a nazwa miasta była ciągiem znaków. Każde miasto w pliku tekstowym ma zostać zapisane, jako obiekt klasy *Miasto* (tak więc z każdym wczytanym miastem ma być tworzony nowy obiekt klasy *Miasto*).

  Klasa *Miasto* ma wdrażać interfejs *Comparable* oraz posiadać metodę *toString*. Po czytaniu pliku tekstowego na ekranie mają się pokazać wczytane miasta, **posortowane** wg. ich kodu pocztowego. Na przykład:

  Ilość znalezionych miast: 7
  05075 Warszawa
  15320 Białystok
  32300 Olkusz
  32310 Klucze
  40209 Sosnowiec
  52119 Wrocław
  61742 Poznań

- **Zadanie 4**

  Dla ułatwienia stwórz nowy folder o nazwie ZbiorDanych. Wszystkie pliki związane z tym ćwiczeniem zapisuj w tym folderze.

  Wśród plików, które otrzymałeś razem z tym kursem, znajdziesz katalog int_collection. Zawiera on abstrakcyjną klasę *AbstractIntCollection* oraz dwa interfejsy: *IntList* och *IntStack*.

Ta abstrakcyjna klasa zawiera metody, które zapewniają wsparcie do rozwijaniu struktur danych, które opierają się na tabelach. Te dwa interfejsy określają jak ma działać lista liczb całkowitych oraz stos liczb całkowitych.

Twoim zadaniem jest stworzyć klasy, które wdrażają te dwa interfejsy oraz dziedziczą z tej abstrakcyjnej klasy. Te dwie klasy mają mieć następujące sygnatury:

public class ArrayIntList extends AbstractIntCollection implements IntList
public class ArrayIntStack extends AbstractIntCollection implements IntStack

Stwórz także klasę *CollectionMain* która pokazuje jak te dwie klasy działają. Pamiętaj: Nie możesz nic zmieniać w obu interfejsach czy tej klasie abstrakcyjnej.

Koniec

I to już koniec modułu 4 oraz ćwiczeń w tym kursie. Porównaj swoje odpowiedzi z odpowiedziami załączonymi w tym kursie. Powodzenia i do zobaczenia!