1. Klasa Arrays

Umożliwia operacje na tablicach. Metody udostępnione przez tą klasę:

- `fill()`
- `equals()`
- `sort()`
- `binarySearch()`

2. Kolekcja.

Kolekcja *Collection* jest grupą odrębnych elementów podlegających określonym regułom. Są to obiekty służące do przechowywania innych obiektów i udostępniające mechanizmy pozwalające na wykonywanie rożnych operacji na zbiorze tych obiektów. Do przechowywania zmiennych prostych oraz obiektów posłużyć mogą tablice, jednak w wielu zadaniach do składowania obiektów i ich przetwarzania tablice okazują się nie wystarczające. W Javie zostały zaprojektowane odpowiednie interfejsy oraz klasy pozwalające na efektywne przechowywanie takich obiektów.

Interfejs Collection jest interfejsem sparametryzowany, w postaci:

```
interface Collection<Typ>
```

Klasa *Typ* określa typ obiektów przechowywanych w kolekcji. Uniemożliwia to dodawanie do kolekcji obiektów innego typu co mogło by powodować błędy. Interfejs ten jest rozszerzeniem Interfejsu *Iterable*, można do niego zastosować pętlę *for-each*. Tylko klasy implementujące ten interfejs można przeglądaćza pomocą tej pętli. Poniżej przedstawione są najważniejsze metody tego interfejsu:

- `add()`
- `remove()`
- `size()`
- `iterator()`
- `toArray()`

3. Interfejs *List*

Interfejs *List* jest interfejsem sparametryzowanym postaci:

```
Interface List<Typ> extends Collection<Typ>
```

List służy do przechowywania elementów w określonej kolejności. Elementy mogą być pobierane oraz wstawiane na podstawie ich położenia. W przypadku list stosuje się indeksowanie podobne jak w przypadku tablic.

Klasa *Array List* implementuje interfejs List i rozszerza klasę *AbstractList. ArrayList* jest klasą sparametryzowaną:

```
Class ArrayList<Typ>
```

ArrayList implementuje interfejs *List* jako tablic (rozszerzonej). Pozwala na szybkii swobodny dostęp do elementów. Nie jest dobry rozwiązaniem jeżeli chcemy aby służyła do wstawiania lub usuwania elementów z wnętrza tablicy. Starszą klasą o podobnym zastosowaniu jest klasa *Vector*.

Klasa *LinkedList* implementuje interfejs *List* i rozszerza klasę *AbstractList*, również jest klasą sparametryzowaną. Dostosowana jest ona do sekwencyjnego przetwarzania. Kolekcję tą mającą strukturę listy można z powodzeniem zastosować do wstawiania i usuwania elementów z początku i końca listy. Dzięki udostępnianym przez siebie metodom klasa ta nadaje się do budowania stosu list jedno i dwukierunkowych oraz kolejek.

4. Interfejs *Set*

Interfejs *Set* jest interfejsem sparametryzowanym i rozszerzającym interfejs *Collection*, postaci:

```
Interface Set <Typ> extends Collection <Typ>
```

Set oraz *SortedSet*, definiują zbiory i służą do przechowywania pojedynczych egzemplarzy, ten drugi w postaci uporządkowanej. Nie przyjmują zduplikowanych egzemplarzy, w związku z tym w klasach wywiedzionych z nich musi być zdefiniowana metoda *equals()*.

Klasa HashSet stosowana jest w przypadku zbiorów dla których jest istotny krótki czas lokalizacji. Do przechowywania danych w tej kolekcji służy tablica haszująca. Do wyszukiwania danych stosowane są funkcje mieszające. Funkcje mieszające na podstawie klucza liczą unikatową wartość, tzw. skrót. Skrót ten służy do indeksowania zawartości.

LinkedHashSet jest klasą podobną do *HashSet*, przechowuje informacje o kolejności wstawiania danych, odtworzenie tej kolejności wymaga utworzenia iteratora.

TreeSet przechowuje zbiór w postaci uporządkowanej (rosnąco), dokładniej w strukturze drzewa dzięki temu można pobierać elementy uporządkowane.

Queue jest interfejsem sparametryzowanym dodanym w Javie 5. Pozwala na wydajne działanie w strukturze kolejek w szczególności kolejek FIFO. Zawiera metody pozwalające na działaniu na elementach początkowych kolejki.

PriorityQueue tworzy kolejkę priorytetową, priorytety ustala komputer.

5. Map

Map *(nie jest kolekcją)* **–** interfejs sparametryzowany pozwala na operowanie grupą obiektów typu klucz-wartość. Kontenery *Map,* tak jak tablice i kolekcje *Collection*, mogą być łatwo rozbudowywane do wielu wymiarów; wystarczy utworzyć kontener *Map*, którego elementami są inne kontenery Map. Można zatem w prosty sposób utworzyć kontenery o całkiem pokaźnych strukturach danych. Kontener *Map* może zwracać zbiór (*Set*) kluczy, kolekcję (*Collection*) wartości albo zbiór par.

HashMap implementacja jest oparta na tablicy mieszającej. Zapewnia wstawianie i lokalizację par w stałym czasie.

TreeMap implementacja oparta jest na drzewach. Przechowuje pary w postaci uporządkowanej. Pozwala na dostęp do fragmentu drzewa.

Metody interfejsu *Map*:

- `put()`
- `get()`
- `remove()`
- `size()`
- `entrySet()`
- `keySet()`

6. Iteratory

Iterator jest obiektem pozwalającym poruszać się po innym obiekcie (kolekcji). Jest to tak popularne i pożyteczne narzędzie że doczekało się nawet specjalnego wzorca projektowego.

Metody *Itteratora*:

- `hasNext()`
- `next()`
- `remove()`

Zadanie:

- Zaimplementować aplikację z wykorzystaniem tablic I klasy Arrays
- Zaimplementować aplikację z wykorzystaniem kontera HashSet lub HashMap
- Przesłonić metody equals oraz hashCode