**Listy**

Cechy wspólne:

* Elementy mogą być zduplikowane;
* Lista zachowuje kolejność elementów taka, w jakiej dodawane były elementy;
* Możemy uzyskać dostęp do wszystkich elementów listy po ich indeksach.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| **Różnice** | |
| 1. Pod spodem ma tablicę (ma ustalony rozmiar), której rozmiar jest automatycznie zwiększany w razie takiej konieczności; 2. Wyszukiwanie dowolnego elementu jest stałe w czasie; 3. Powinna być używana wtedy kiedy dużo razy odczytujesz dane; 4. Dodawanie/usuwanie elementu jest wolniejsze niż uzyskiwanie dostępu do tego elementu (listę trzeba przeorganizować). | 1. Pod spodem ma Deque (może być używana jako kolejka); 2. Wyszukiwanie elementów nie jest w stałym czasie i zależy od miejsca rozlokowania elementu; 3. Powinna być używana, kiedy mamy konieczność częstego dodawania elementów; 4. Dodawanie/usuwanie elementu jest szybsze niż uzyskiwanie dostępu do tego elementu |

**Set**

Cechy wspólne:

* Może przechowywać różne od siebie obiekty w rozumieniu **Equals** (nie możemy przechowywać elementów zduplikowanych);
* Elementy nie mają swoich indeksów\* (elementy przechowywane są w sposób losowy);
* Bez dobrze napisanej metody HashCode, działa bardzo wolno;
* Aby wyświetlić posortowanego Set’a należy zamienić go na listę, a następnie ułożyć za pomocą komparatora;

|  |  |  |
| --- | --- | --- |
| **HashSet** | **LinkedHashSet** | **TreeSet** |
| **Różnice** | | |
| 1. Nie mamy pewności w jakiej elementy zostaną ułożone | 1. Zachowuje kolejność dodawanych elementów (występuje podwójnie linkowana LinkedList) | 1. Sortuje elementy, które dodajemy do Set’a; 2. Można w nim zastosować komparator; 3. Przy dodawaniu elementów, TreeSet sprawdza gdzie powinien on zostać dodany 4. Można do niego dodawać obiekty, które są comparable. Jeżeli nie implementujemy w naszej klasie interfejsu Comparable to musimy własnoręcznie napisać lambdę dla Compare. 5. Implementuje NavigableSet |