Interfejsy `SortedSet` i `SortedMap` udostępniają obiekt komparatora użyty do sortowania oraz definiują metody tworzące widoki podzbiorów kolekcji. Widoki te opisujemy w kolejnym podrozdziale.

Wreszcie, w Java SE 6 wprowadzono interfejsy `NavigableSet` i `NavigableMap`, które zawierają dodatkowe metody służące do przemierzania i przeszukiwania uporządkowanych zbiorów i map (w idealnej sytuacji metody te powinny się znajdować w interfejsach `SortedSet` i `SortedMap`). Interfejsy te są implementowane przez klasy `TreeSet` i `TreeMap`.

Kolej na klasy implementujące wymienione interfejsy. Wiemy już, że niektóre metody interfejsów kolekcyjnych można z łatwością zaimplementować na bazie bardziej podstawowych metod. Wiele z tych implementacji znajduje się w klasach abstrakcyjnych:

```
AbstractCollection
AbstractList
AbstractSequentialList
AbstractSet
AbstractQueue
AbstractMap
```

Klasy te można rozszerzać przy tworzeniu własnych klas kolekcyjnych, dzięki czemu dziedziczy się po nich wiele rutynowych procedur.

Konkretne klasy dostępne w bibliotece Javy to:

```
LinkedList
ArrayList
ArrayDeque
HashSet
TreeSet
PriorityQueue
HashMap
TreeMap
```

Rysunek 13.11 przedstawia relacje zachodzące między tymi klasami.

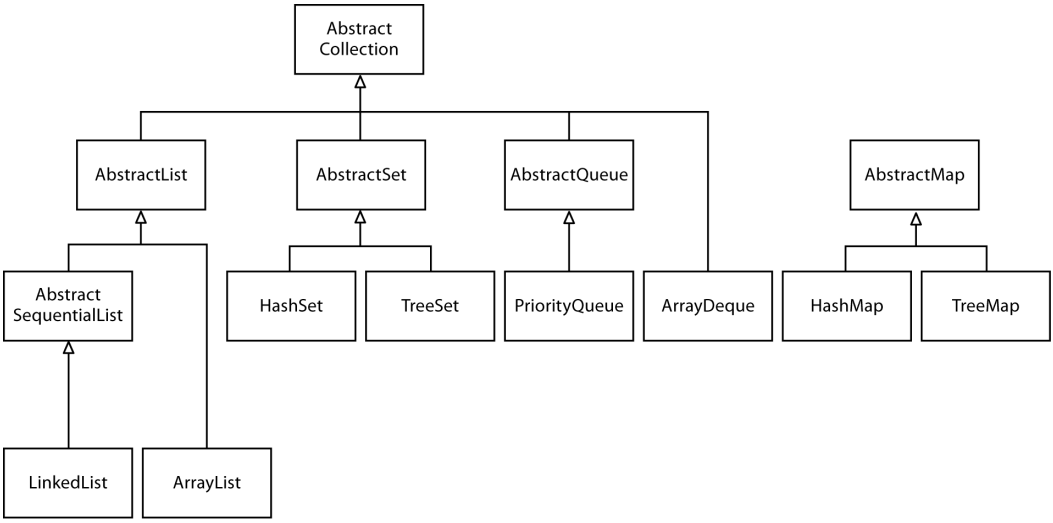
Na koniec należy jeszcze wymienić kilka starszych klas kontenerowych, które są dostępne w Javie od początku, zanim jeszcze powstała architektura kolekcji:

```
Vector
Stack
Hashtable
Properties
```

Zostały one wcielone do kolekcji — rysunek 13.12. Omawiamy te klasy nieco dalej.

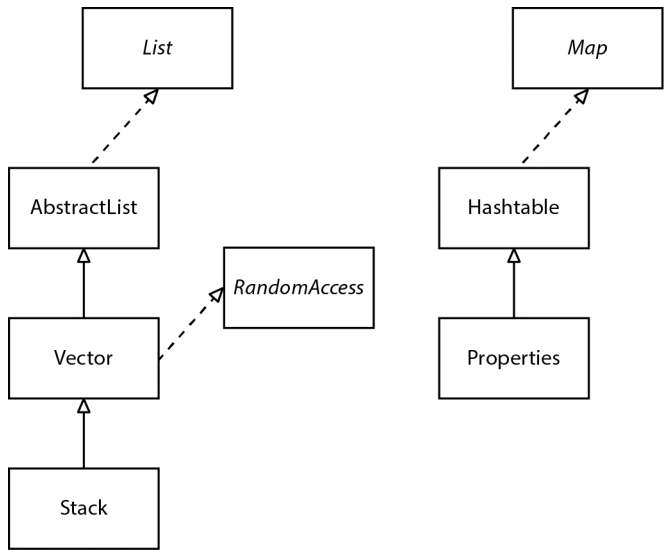
13.3.1. Widoki i obiekty opakowujące

Z rysunków 13.10 i 13.11 można wyciągnąć wniosek, że projektanci Javy wpadli w lekką przesadę, tworząc tak wiele interfejsów i klas abstrakcyjnych do implementacji raczej umiarkowanej liczby konkretnych klas kolekcyjnych. Jednak rysunki te nie pokazują wszystkiego.



Rysunek 13.11. Klasy w architekturze kolekcji

Rysunek 13.12. Starsze klasy w architekturze kolekcji



Za pomocą tak zwanych **widoków** (ang. *view*) można tworzyć inne obiekty implementujące interfejsy `Collection` i `Map`. Przykład takiego działania zaprezentowaliśmy, kiedy użyliśmy metody `keySet` mapy. Na pierwszy rzut oka wydaje się, że metoda ta tworzy nowy zbiór, zapełnia go wszystkimi kluczami z mapy i zwraca go. Nie jest to jednak prawda. Metoda `keySet` zwraca obiekt klasy implementującej interfejs `Set`, którego metody operują na oryginalnej mapie. Taka kolekcja nazywana jest **widokiem**.

Widoki mają kilka ważnych zastosowań w architekturze kolekcji. Opisujemy je w poniższych podrozdziałach.