**Section 16.1 Introduction**

• The Java collections framework provides prebuilt data structures and methods to manipulate them.

#### Section 16.2 Collections Overview

• A collection is an object that can hold references to other objects.

• The classes and interfaces of the collections framework are in package java.util.

#### Section 16.3 Type-Wrapper Classes

• Type-wrapper classes (e.g., Integer, Double, Boolean) enable programmers to manipulate primitive-type values as objects (p. [687](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec4_html#page_687)). Objects of these classes can be used in collections.

#### Section 16.4 Autoboxing and Auto-Unboxing

• Boxing (p. [687](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec4_html#page_687)) converts a primitive value to an object of the corresponding type-wrapper class. Unboxing (p.[687](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec4_html#page_687)) converts a type-wrapper object to the corresponding primitive value.

• Java performs boxing conversions and unboxing conversions automatically.

#### Section 16.5 Interface Collection and ClassCollections

• Interfaces Set and List extend Collection (p. [686](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec2_html#page_686)), which contains operations for adding, clearing, comparing and retaining objects in a collection, and method iterator (p. [691](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec5_html#page_691)) to obtain a collection’sIterator (p. [687](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec4_html#page_687)).

• Class Collections (p. [688](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec5_html#page_688)) provides static methods for manipulating collections.

#### Section 16.6 Lists

• A List (p. [694](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_694)) is an ordered Collection that can contain duplicate elements.

• Interface List is implemented by classes ArrayList,LinkedList and Vector. ArrayList (p. [688](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec5_html#page_688)) is a resizable-array implementation. LinkedList (p. [688](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec5_html#page_688)) is a linkedlist implementation of a List.

• Java SE 7 supports type inferencing with the <>notation in statements that declare and create generic type variables and objects.

• Iterator method hasNext (p. [691](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec5_html#page_691)) determines whether a Collection contains another element. Method next returns a reference to the next object in the Collection and advances the Iterator.

• Method subList (p. [694](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_694)) returns a view into a List. Changes made to this view are also made to the List.

• Method clear (p. [694](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_694)) removes elements from a List.

• Method toArray (p. [695](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_695)) returns the contents of a collection as an array.

#### Section 16.7 Collections Methods

• Algorithms sort (p. [697](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec7_html#page_697)), binarySearch, reverse (p.[702](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec8_html#page_702)), shuffle (p. [700](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec7_html#page_700)), fill (p. [702](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec8_html#page_702)), copy, addAll(p. [693](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_693)), frequency and disjoint operate on Lists. Algorithms min and max (p. [703](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec9_html#page_703)) operate onCollections.

• Algorithm addAll appends all the elements in an array to a collection (p. [706](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec10_html#page_706)), frequency (p. [706](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec10_html#page_706)) calculates how many elements in the collection are equal to the specified element, and disjoint (p. [706](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec10_html#page_706)) determines whether two collections have elements in common.

• Algorithms min and max find the smallest and largest items in a collection.

• The Comparator interface (p. [697](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec7_html#page_697)) provides a means of sorting a Collection’s elements in an order other than their natural order.

• Collections method reverseOrder (p. [698](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec7_html#page_698)) returns aComparator object that can be used with sort to sort a collection’s elements in reverse order.

• Algorithm shuffle (p. [700](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec7_html#page_700)) randomly orders the elements of a List.

• Algorithm binarySearch (p. [704](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec9_html#page_704)) locates an Object in a sorted List.

#### Section 16.8 Stack Class of Package java.util

• Class Stack (p. [708](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_708)) extends Vector. Stack methodpush (p. [709](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_709)) adds its argument to the top of the stack. Method pop (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) removes the top element of the stack. Method peek returns a reference to the top element without removing it. Stack method isEmpty (p.[710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) determines whether the stack is empty.

#### Section 16.9 Class PriorityQueue and InterfaceQueue

• Interface Queue (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) extends interface Collectionand provides additional operations for inserting, removing and inspecting elements in a queue.

• PriorityQueue (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) implements interface Queueand orders elements by their natural ordering or by aComparator object that’s supplied to the constructor.

• PriorityQueue method offer (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) inserts an element at the appropriate location based on priority order. Method poll (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) removes the highest-priority element of the priority queue. Method peek(peek) gets a reference to the highest-priority element of the priority queue. Method clear (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) removes all elements in the priority queue. Method size (p. [710](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec8_html#page_710)) gets the number of elements in the priority queue.

#### Section 16.10 Sets

• A Set (p. [711](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec9_html#page_711)) is an unordered Collection that contains no duplicate elements. HashSet (p. [711](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec9_html#page_711)) stores its elements in a hash table. TreeSet (p. [711](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec9_html#page_711)) stores its elements in a tree.

• Interface SortedSet (p. [712](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec10_html#page_712)) extends Set and represents a set that maintains its elements in sorted order. Class TreeSet implements SortedSet.

• TreeSet method headSet (p. [712](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec10_html#page_712)) gets a TreeSet view containing elements that are less than a specified element. Method tailSet (p. [713](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec12_html#page_713)) gets a TreeSet view containing elements that are greater than or equal to a specified element. Any changes made to these views are made to the original TreeSet.

#### Section 16.11 Maps

• Maps (p. [714](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec11_html#page_714)) store key–value pairs and cannot contain duplicate keys. HashMaps (p. [714](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec11_html#page_714)) and Hashtables (p.[714](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec11_html#page_714)) store elements in a hash table, and TreeMaps (p.[714](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec11_html#page_714)) store elements in a tree.

• HashMap takes two type arguments—the type of key and the type of value.

• HashMap method put (p. [717](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_717)) adds a key–value pair to a HashMap. Method get (p. [717](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_717)) locates the value associated with the specified key. Method isEmpty (p.[718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) determines if the map is empty.

• HashMap method keySet (p. [718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) returns a set of the keys. Map method size (p. [718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) returns the number of key–value pairs in the Map.

• Interface SortedMap (p. [714](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec11_html#page_714)) extends Map and represents a map that maintains its keys in sorted order. Class TreeMap implements SortedMap.

#### Section 16.12 Properties Class

• A Properties object (p. [718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) is a persistent subclass ofHashtable.

• The Properties no-argument constructor creates an empty Properties table. An overloaded constructor receives a Properties object containing default property values.

• Properties method setProperty (p. [718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) specifies the value associated with its key argument. MethodgetProperty (p. [718](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#page_718)) locates the value of the key specified as an argument. Method store (p. [720](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec12_html#page_720)) saves the contents of a Properties object to specifiedOutputStream. Method load

(p. [721](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec12_html#page_721)) restores the contents of a Properties object from the specified InputStream.

#### Section 16.13 Synchronized Collections

• Collections from the collections framework are unsynchronized. Synchronization wrappers (p. [721](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec12_html#page_721)) are provided for collections that can be accessed by multiple threads simultaneously.

#### Section 16.14 Unmodifiable Collections

• Unmodifiable collection wrappers (p. [721](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec12_html#page_721)) throwUnsupportedOperationExceptions (p. [694](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#page_694)) if attempts are made to modify the collection.

#### Section 16.15 Abstract Implementations

• The collections framework provides various abstract implementations of collection interfaces from which you can quickly flesh out complete customized implementations.

### Self-Review Exercises

[**16.1**](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec19_html#ch16ans1) Fill in the blanks in each of the following statements:

a) A(n) \_\_\_\_\_\_\_\_\_ is used to iterate through a collection and can remove elements from the collection during the iteration.

b) An element in a List can be accessed by using the element’s \_\_\_\_\_\_\_\_\_.

c) Assuming that myArray contains references to Doubleobjects, \_\_\_\_\_\_\_\_\_ occurs when the statement "myArray[0] = 1.25;" executes.

d) Java classes \_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_ provide the capabilities of arraylike data structures that can resize themselves dynamically.

e) If you do not specify a capacity increment, the system will \_\_\_\_\_\_\_\_\_ the size of the Vector each time additional capacity is needed.

f) You can use a(n) \_\_\_\_\_\_\_\_\_ to create a collection that offers only read-only access to others while allowing read–write access to yourself.

g) Assuming that myArray contains references to Doubleobjects, \_\_\_\_\_\_\_\_\_ occurs when the statement "double number = myArray[0];" executes.

h) Collections algorithm \_\_\_\_\_\_\_\_\_ determines if two collections have elements in common.

[**16.2**](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec19_html#ch16ans2) Determine whether each statement is true or false. Iffalse, explain why.

a) Values of primitive types may be stored directly in a collection.

b) A Set can contain duplicate values.

c) A Map can contain duplicate keys.

d) A LinkedList can contain duplicate values.

e) Collections is an interface.

f) Iterators can remove elements.

g) With hashing, as the load factor increases, the chance of collisions decreases.

h) A PriorityQueue permits null elements.

### Answers to Self-Review Exercises

[**16.1**](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec18_html#ch16que1)

a) Iterator.

b) index.

c) autoboxing.

d) ArrayList, Vector.

e) double.

f) unmodifiable wrapper.

g) auto-unboxing.

h) disjoint.

[**16.2**](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec18_html#ch16que2)

a) False. Autoboxing occurs when adding a primitive type to a collection, which means the primitive type is converted to its corresponding type-wrapper class.

b) False. A Set cannot contain duplicate values.

c) False. A Map cannot contain duplicate keys.

d) True.

e) False. Collections is a class; Collection is aninterface.

f) True.

g) False. As the load factor increases, fewer slots are available relative to the total number of slots, so the chance of a collision increases.

h) False. Attempting to insert a null element causes aNullPointerException.

### Exercises

**16.3** Define each of the following terms:

a) Collection

b) Collections

c) Comparator

d) List

e) load factor

f) collision

g) space/time trade-off in hashing

h) HashMap

**16.4** Explain briefly the operation of each of the following methods of class Vector:

a) add

b) set

c) remove

d) removeAllElements

e) removeElementAt

f) firstElement

g) lastElement

h) contains

i) indexOf

j) size

k) capacity

**16.5** Explain why inserting additional elements into a Vectorobject whose current size is less than its capacity is a relatively fast operation and why inserting additional elements into a Vector object whose current size is at capacity is a relatively slow operation.

**16.6** By extending class Vector, Java’s designers were able to create class Stack quickly. What are the negative aspects of this use of inheritance, particularly for class Stack?

**16.7** Briefly answer the following questions:

a) What is the primary difference between a Set and aMap?

b) What happens when you add a primitive type (e.g.,double) value to a collection?

c) Can you print all the elements in a collection without using an Iterator? If yes, how?

**16.8** Explain briefly the operation of each of the followingIterator-related methods:

a) iterator

b) hasNext

c) next

**16.9** Explain briefly the operation of each of the following methods of class HashMap:

a) put

b) get

c) isEmpty

d) containsKey

e) keySet

**16.10** Determine whether each of the following statements is true or false. If false, explain why.

a) Elements in a Collection must be sorted in ascending order before a binarySearch may be performed.

b) Method first gets the first element in a TreeSet.

c) A List created with Arrays method asList is resizable.

**16.11** Explain the operation of each of the following methods of the Properties class:

a) load

b) store

c) getProperty

d) list

**16.12** Rewrite lines 16–25 in [Fig. 16.3](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec6_html#ch16fig03) to be more concise by using the asList method and the LinkedList constructor that takes a Collection argument.

**16.13 (Duplicate Elimination)** Write a program that reads in a series of first names and eliminates duplicates by storing them in a Set. Allow the user to search for a first name.

**16.14 (Counting Letters)** Modify the program of [Fig. 16.18](http://proquest.safaribooksonline.com/9780133813036/ch16lev2sec13_html#ch16fig18)to count the number of occurrences of each letter rather than of each word. For example, the string "HELLO THERE"contains two Hs, three Es, two Ls, one O, one T and one R. Display the results.

**16.15 (Color Chooser)** Use a HashMap to create a reusable class for choosing one of the 13 predefined colors in classColor. The names of the colors should be used as keys, and the predefined Color objects should be used as values. Place this class in a package that can be imported into any Java program. Use your new class in an application that allows the user to select a color and draw a shape in that color.

**16.16 (Counting Duplicate Words)** Write a program that determines and prints the number of duplicate words in a sentence. Treat uppercase and lowercase letters the same. Ignore punctuation.

**16.17 (Inserting Elements in a** ***LinkedList*** **in Sorted Order)** Write a program that inserts 25 random integers from 0 to 100 in order into a LinkedList object. The program should sort the elements, then calculate the sum of the elements and the floating-point average of the elements.

**16.18 (Copying and Reversing** ***LinkedList*s)** Write a program that creates a LinkedList object of 10 characters, then creates a second LinkedList object containing a copy of the first list, but in reverse order.

**16.19 (Prime Numbers and Prime Factors)** Write a program that takes a whole number input from a user and determines whether it’s prime. If the number is not prime, display its unique prime factors. Remember that a prime number’s factors are only 1 and the prime number itself. Every number that’s not prime has a unique prime factorization. For example, consider the number 54. The prime factors of 54 are 2, 3, 3 and 3. When the values are multiplied together, the result is 54. For the number 54, the prime factors output should be 2 and 3. Use Sets as part of your solution.

**16.20 (Sorting Words with a** ***TreeSet*)** Write a program that uses a String method split to to-kenize a line of text input by the user and places each token in a TreeSet. Print the elements of the TreeSet. [Note: This should cause the elements to be printed in ascending sorted order.]

**16.21 (Changing a** ***PriorityQueue*’s Sort Order)** The output of [Fig. 16.15](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec9_html#ch16fig15) shows that PriorityQueue ordersDouble elements in ascending order. Rewrite [Fig. 16.15](http://proquest.safaribooksonline.com/9780133813036/ch16lev1sec9_html#ch16fig15) so that it orders Double elements in descending order (i.e., 9.8should be the highest-priority element rather than 3.2).