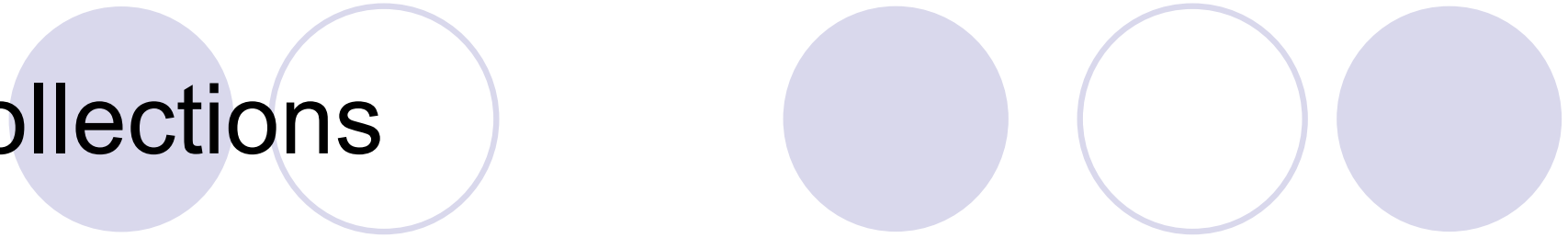
Cisco Java Chapter 14

BeiJing University of Posts & Telecommunications

Tang Xiaosheng
txs@bupt.edu.cn

- This chapter introduces the many uses of utility classes.
- These include managing a collection of objects, manipulating date data and obtaining information about system settings.

Collections



- The java.util Package
- Collections
- The Collections Framework
- Casy Study: JBANK Application

Key terms

- Collections Framework
- elements collection array LinkedList
- doubly linked Tree Hash table
- bag, multiset, sequence
- ordered, unordered , Set Map dictionaries
- Collection Interface , ArrayList Vector Treeselt Hashset
- Iterator ListIterator , Enumaeration Property
- System properties , key-value pairs sorting shuffling

14.1 The java.util Package

- What does java.util provided?
  - Classes to manage a collection of objects
  - Classes to manage messaging between objects when changes occur in an object
  - Classes to manipulate date information

The java.util Package

- The java.util package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (such as a string tokenizer, a random-number generator, and a bit array).

14.2 Collections

- 14.2.1 Understanding collections
- 14.2.2 Collection storage technologies
- 14.2.3 Properties of collections
- 14.2.4 Types of collections

14.2.1 Understanding collections

- A **collection** is similar to an array. It is a single object that represents a group of objects, i.e. references to other objects.
- The objects in the collection are called **elements**.
- Collections typically deal with many types of objects, all of which descend from a common parent type (Object class).
- Collections **only** maintain references to objects of type Object.

Collection vs. array

- Collection objects provide a great deal of flexibility over array objects.
  - can grow dynamically
  - can maintain the objects in the collection in a sorted order
  - can move through the elements of the collection
  - and can add and remove elements of the structure efficiently
- Methods to perform these operations are provided in most collection classes.

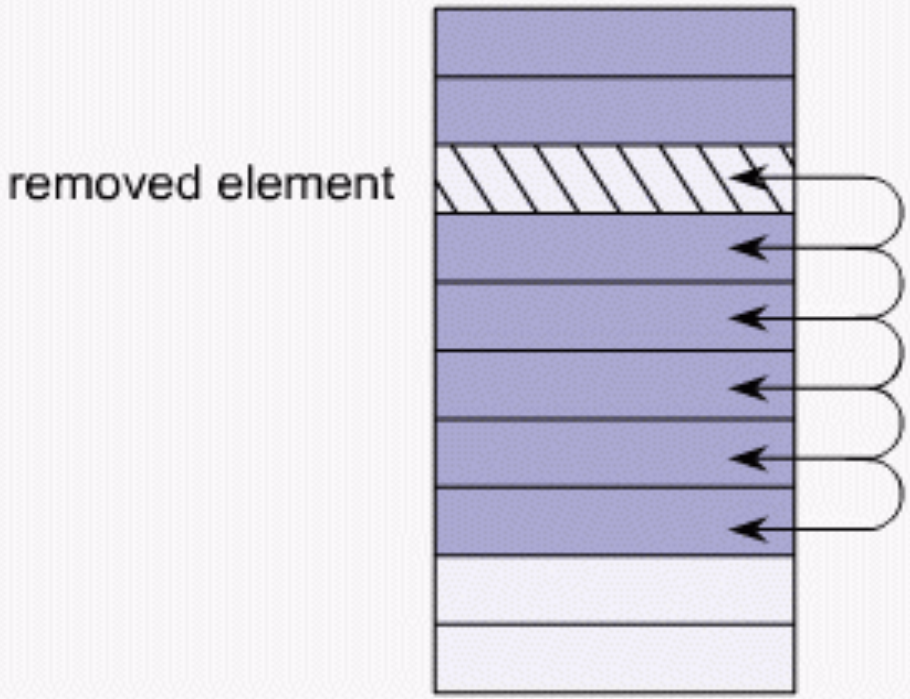
Three important concepts

- Storage technologies available to store and manage a collection of objects
- Properties of collections
- Types of collection objects

14.2.2 Collection storage technologies

- **Array, Linked-list, Tree, HashTable**
- **Array**
- Used for **fixed number** of elements. It is **fast and efficient**. It is very **difficult to add or remove elements** from an array. The array would have to be copied to create a new array. This can be memory intensive and inefficient.

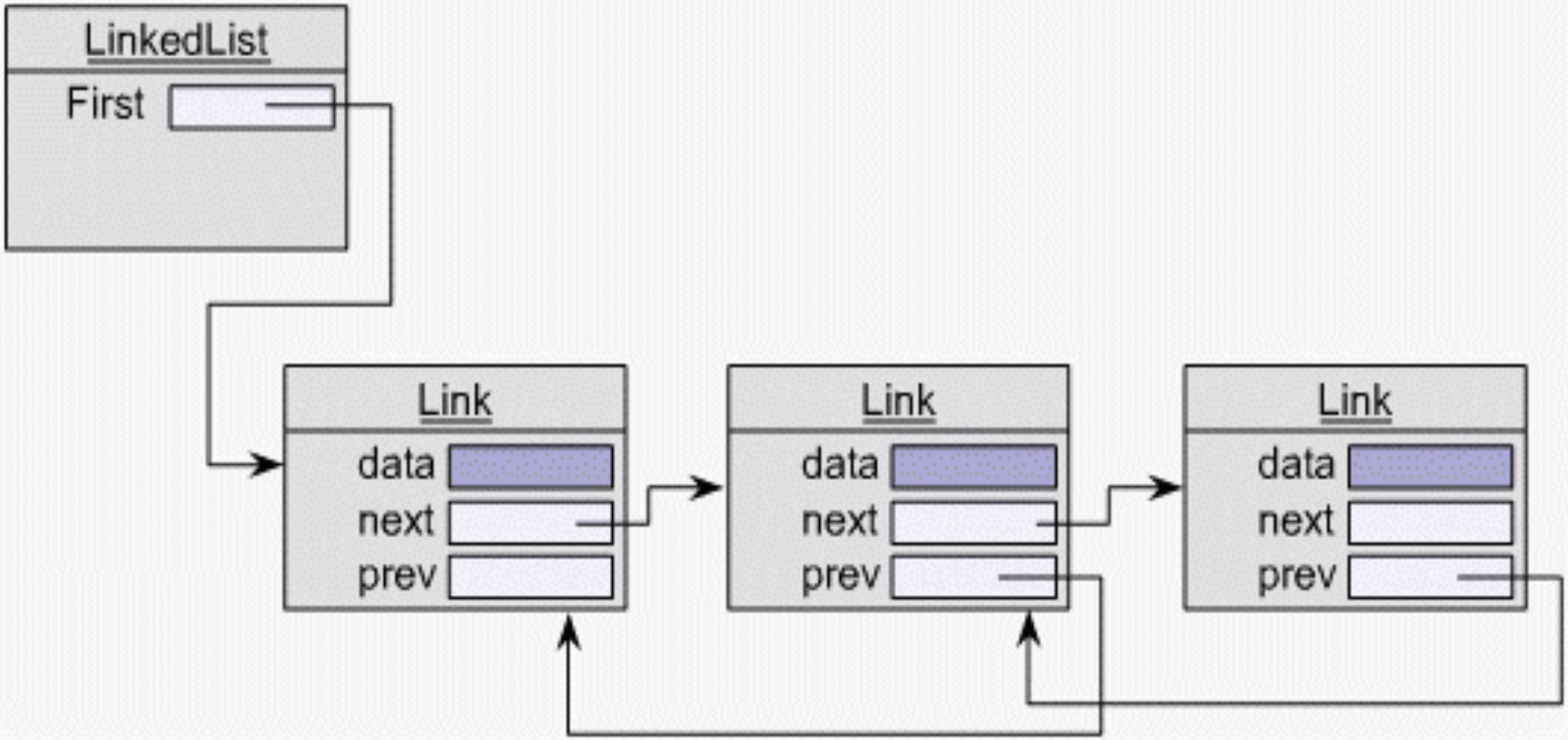
Array, Removing an Element from an Array



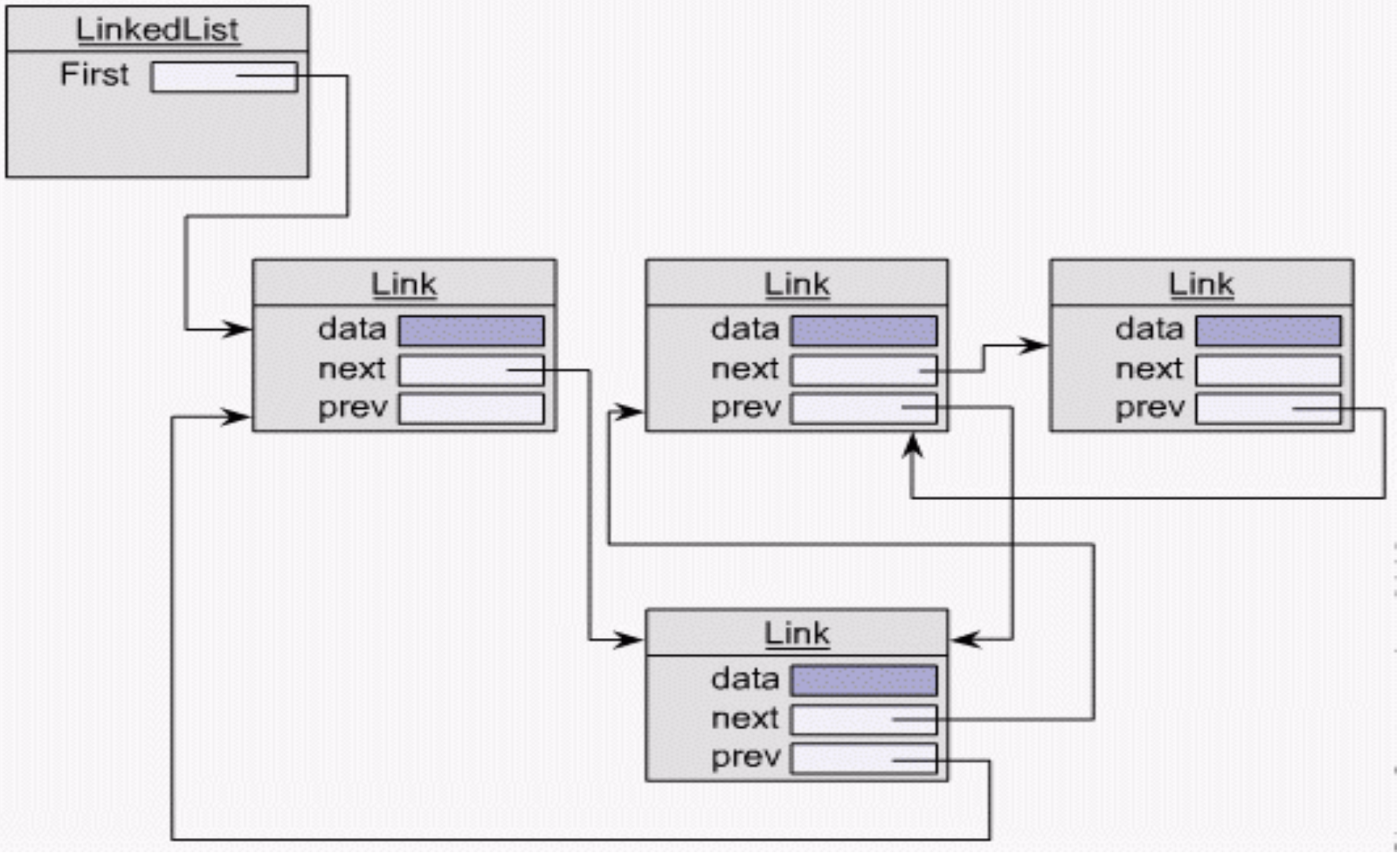
Linked-list

- The items are **ordered**, and they **can have duplicates**.
- The **size can grow** dynamically.
- **Accessing** linked lists **can be slow**, and lists do not provide a search mechanism.
- Linked lists in Java are **doubly linked**.

Linked List, Doubly Linked



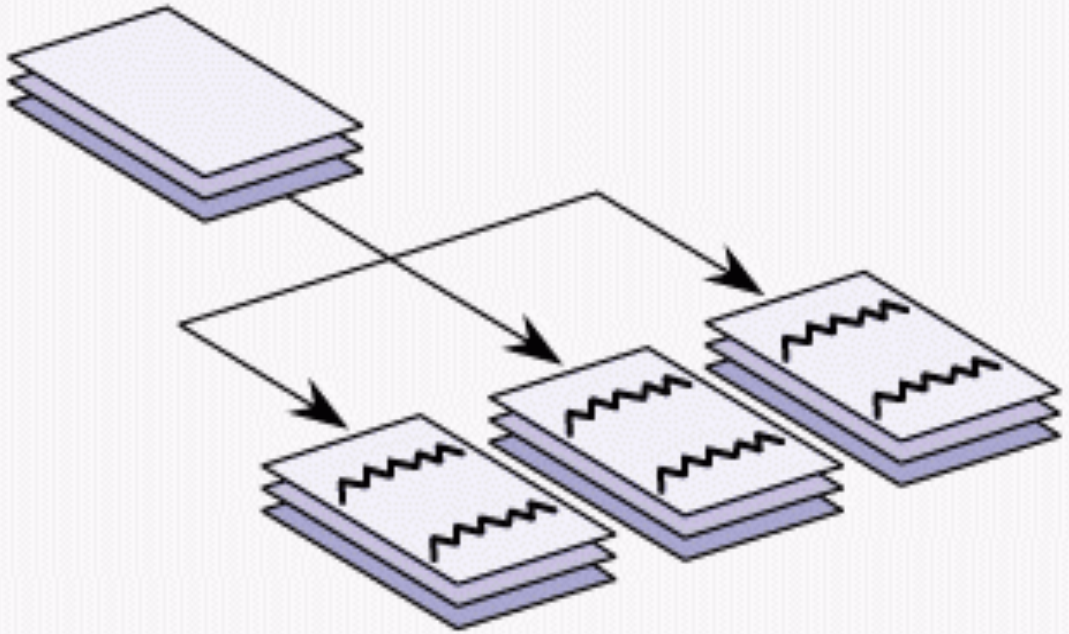
Adding Elements to a List



Tree

- Trees provide storage for items that are sorted in **ascending order**.
- If the objects are placed in a natural order that can be sorted, an index is used to search for the objects.

Tree

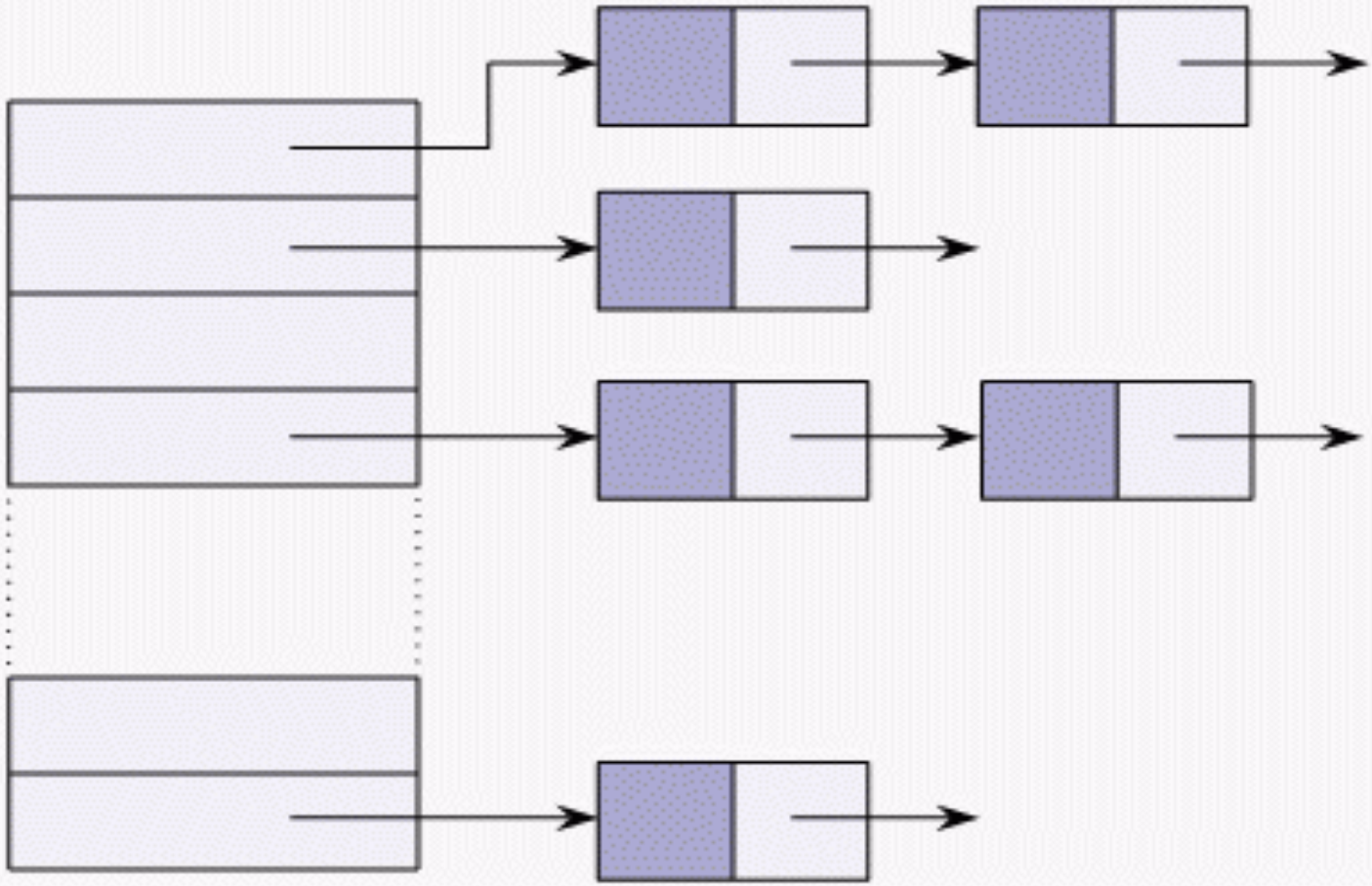


A tree structure

HashTable

- Each item in the collection consists of a **key or identifier** and the **item**. This is also called a **key-value pair**.
- The storage mechanism uses the key value to locate each item. **Accessing** items in a collection that stores items using a hashtable **is fast**, however **additional memory** is needed to maintain the key information.

Hash-Table



14.2.3 Properties of collections

Property	Description
Sorted	Ascending order, or sorted naturally using the equals() method of the object An example would be a List of courses offered or a list of teachers in a school.
Ordered/ Unordered	Collection keeps track of the determining where to place the object in the collection. A deck or cards would an ordered collection
Allows Duplicates	Duplicate orders can be added to the collection. An Example is a collection of birthdays for students in a class.
Uses keys	A key object is used to reference the stored object. Names of customers stored by unique customer ids

14.2.4 Types of collections

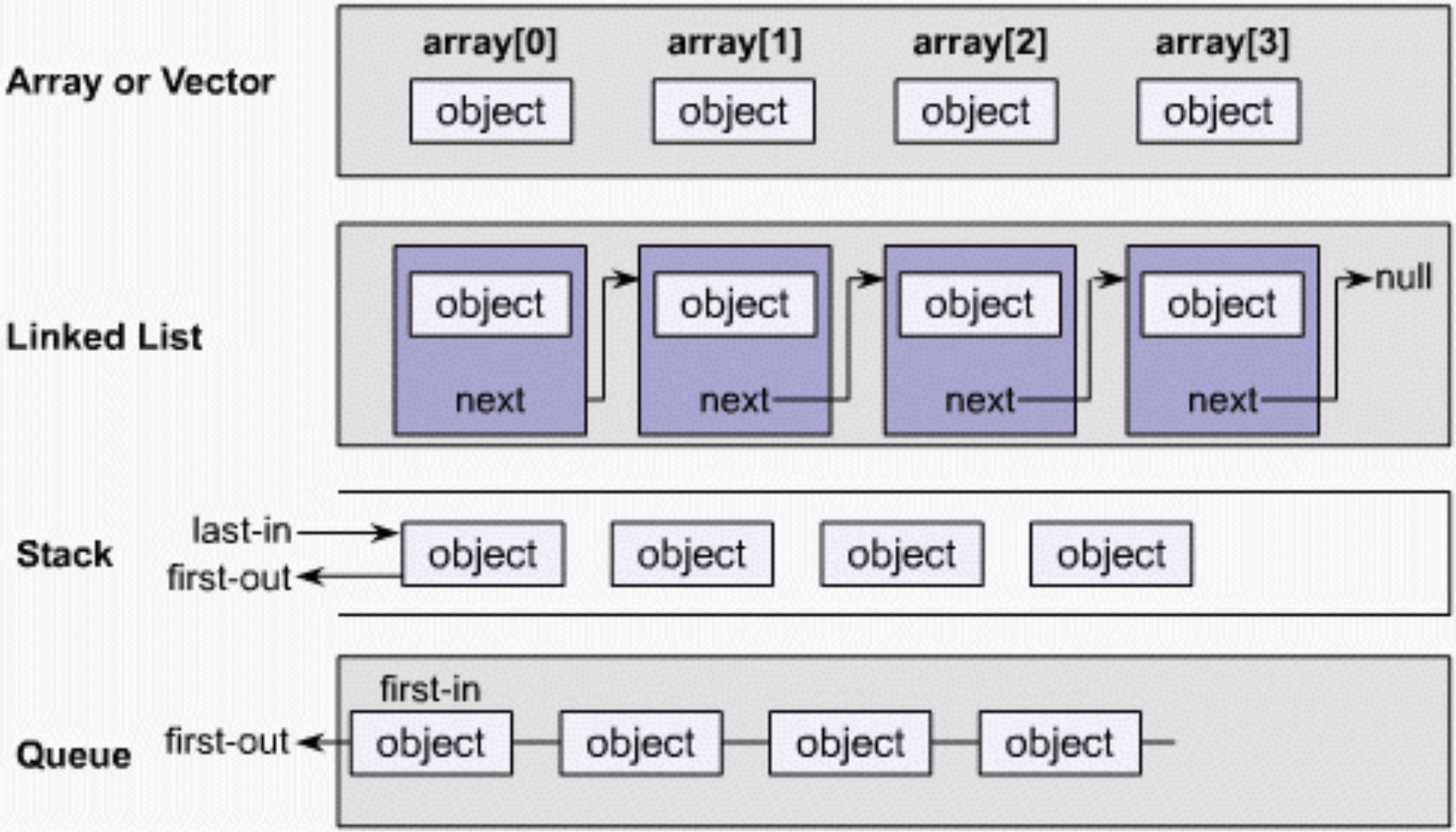
- **Collection, lists, Sets, Maps**

- **Collection**
- A simple container. The objects in a collection can be **unordered**, and **duplicates are permitted**.

Lists

- Lists are **ordered** collections, and they **can have duplicates**.
- The order can be the **natural order**, which is the order in which the objects were added to the list. Because the list is ordered, objects in a list **can be indexed**.
- This type of storage is also known as a **bag** or **multiset**.
- Examples: Array, Vector, Linked List, Stack, Queue

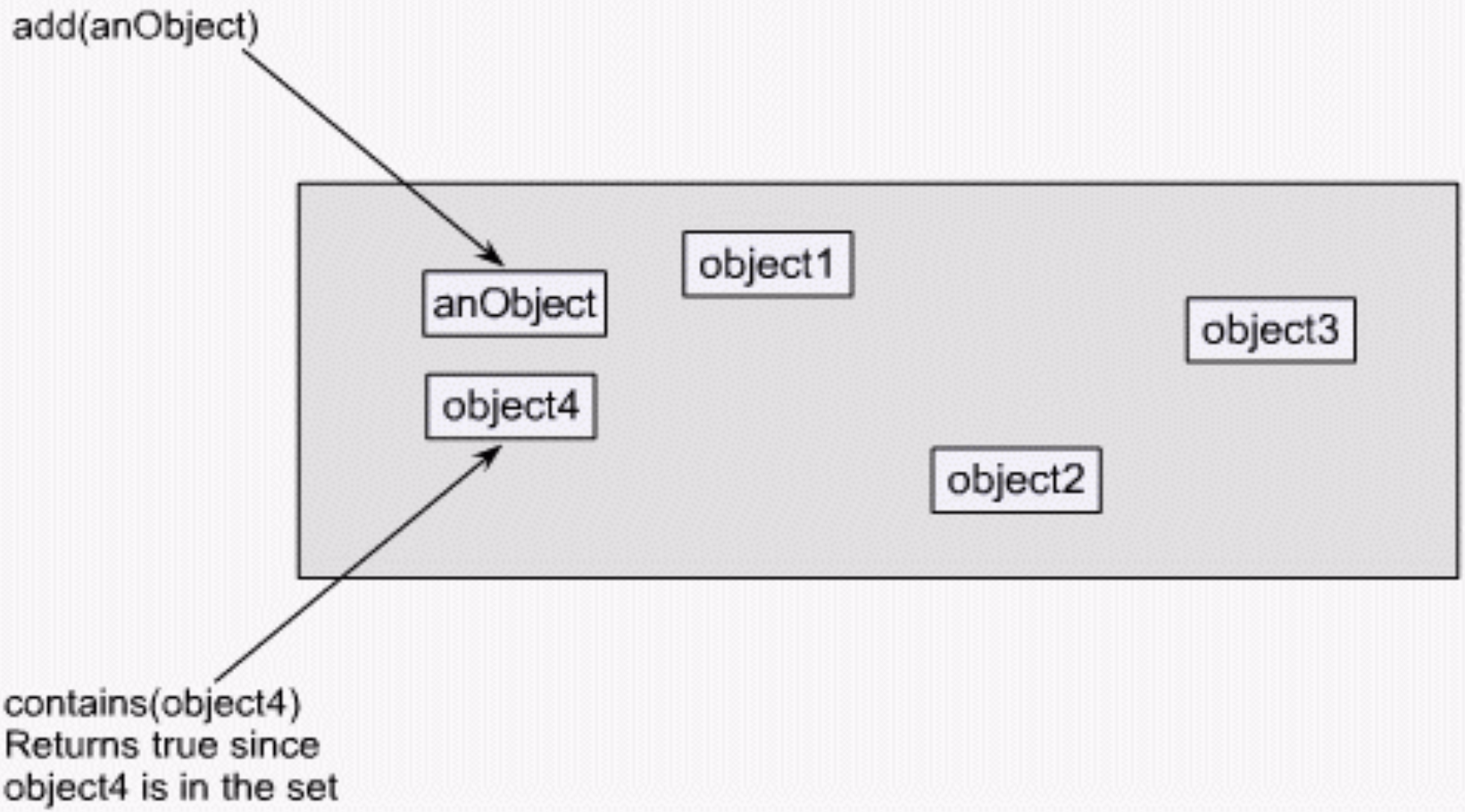
Lists


Array or Vector
array[0]
array[1]
array[2]
array[3]
object
object
object
object
Linked List
object
next
object
next
object
next
object
next
null
Stack
last-in
first-out
object
object
object
object
Queue
first-in
first-out
object
object
object
object

Sets

- A set is an **unordered** collection of objects. **Duplicates are not permitted**.
- The collection can add and remove objects.

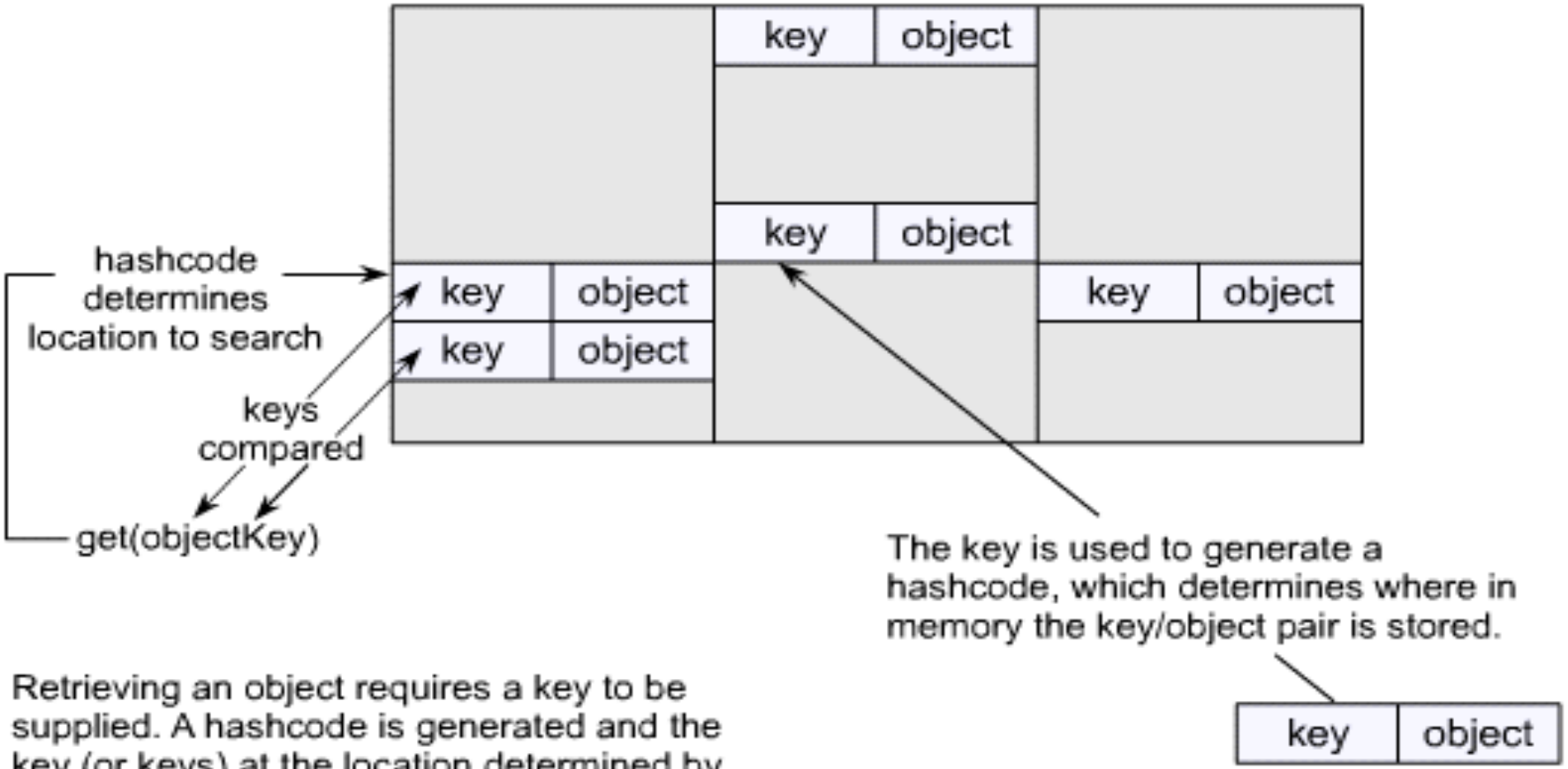
Sets


add(anObject)
anObject
object1
object3
object4
object2
contains(object4)
Returns true since
object4 is in the set

Maps

- A map is a collection of arbitrary associations between a **key object and a value object**.
- With maps, users can search on key data. Maps are also known as dictionaries.

Maps



Retrieving an object requires a key to be supplied. A hashcode is generated and the key (or keys) at the location determined by the hashcode is compared with the supplied key.

14.3 The Collections Framework

- 14.3.1 Overview
- 14.3.2 Collection Interface
- 14.3.3 Collection Classes
- 14.3.4 Set objects
- 14.3.5 List objects
- 14.3.6 Map objects
- 14.3.7 Iterators
- 14.3.8 Sorting and shuffing list objects

14.3.1 Overview

- Framework is **a set of rules or guidelines**.
- The idea behind the collection framework classes was to **define a set of behaviors for objects that contain other objects**.
- These include container behaviors for the collection objects. The collections framework consists of these features.

Collections Framework

Collection Interfaces	Represent different types of collections, such as sets, lists, and maps. These interfaces form the basis of the framework.
General-purpose Implementations	Primary implementations of the collection interfaces.
Legacy Implementations	The collection classes from earlier releases, Vector and Hashtable, have been retrofitted to implement the collection interfaces.
Wrapper Implementations	Add functionality, such as synchronization, to other implementations.
Convenience Implementations	High-performance "mini-implementations" of the collection interfaces.
Abstract Implementations	Partial implementations of the collection interfaces to facilitate

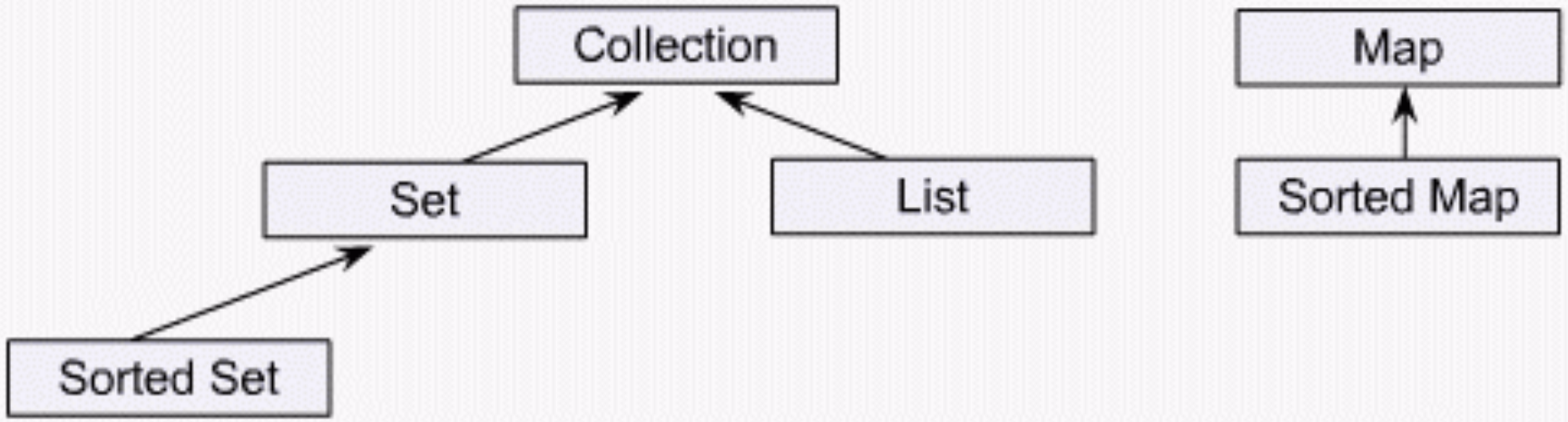
Collections Framework

	implementations" of the collection interfaces.
Abstract Implementations	Partial implementations of the collection interfaces to facilitate custom implementations.
Algorithms	Static methods that perform useful functions on collections, such as sorting a list.
Infrastructure	Interfaces that provide essential support for the collection interfaces.
Array Utilities	Utility functions for arrays of primitives and reference objects. Not, strictly speaking, a part of the Collections Framework, this functionality is being added to the Java platform at the same time and relies on some of the same infrastructure.

14.3.2 Collection interfaces

- The collection framework defines **six** interfaces.
- There are two root classes at the top of the hierarchy. These are the **collection** and the **map** interfaces.

Collections Framework, Map and Collection Interface Hierarchy



Collection

- The collection interface is used for any collection that is **unordered**, and **can have duplicates**.
- This interface supports methods for **adding**, **removing**, **counting**, and **checking items** in a collection.
- A collection is sometimes referred to as a bag, since they do not impose any rules.

List

- The List Interface extends the collection interface to implement an **ordered set** of collections.
- The ordered lists **can be indexed** and **can have duplicate values**.

Set

- The Set interface extends the collection interface to implement sets that are **finite**.
- Sets **do not allow duplicate values**. If they permit a null value, only one null value can occur.
- SortedSet, which extends from Set is an interface for sets whose values are **sorted in ascending order**.

Map

- The **Map** interface provides the basic methods for storing and retrieving data **using key values**. The key values must be **unique**.
- The Map.Entry is an **inner interface** that provides for working with key-value pairs.
- **Sorted Map**, which extends from Map, is an interface for creating a map whose elements are **sorted in ascending order**.

Other Helper Interfaces

- These provide functionality for **traversing** (retrieving a collection of objects in some specific order).
- There are **two specific interfaces** that are available:
- **Iterator** - Provides a basic mechanism for iterating (**looping**) through the elements of a collection. **Only moves forward** through the list.
- **ListIterator** - Provides support of iteration through a list. Provides for scanning **both forward and backward through** a list.

Methods of the Collection Interfaces

Add and Remove methods - used to add - **`boolean add(Object element)`**, and remove - **`boolean remove(Object element)`**, individual elements. If the collection has successfully changed the method will return true.

Query Methods - to find useful information about the collection, such as **`size of collection int size()`**, **`boolean isEmpty()`**, **`boolean contains(Object element)`**.

Iterator methods - to iterate through one element at a time. The iterator methods create an iterator object that has methods to iterate through the object. The iterator object can access the next element using the **`Object next()`** method, check to see if there are any more objects in the collection using the **`boolean hasNext()`** method and remove objects from a collection using the **`void remove()`** method.

Group methods - to deal with the entire collection. With these you programmers can add or remove more than one element with one method call. These include boolean **`containsAll(collection collection)`**,

Methods of the Collection Interfaces

Query Methods - to find useful information about the collection, such as `size of collection int size()`, `boolean isEmpty()`, `boolean contains(Object element)`.

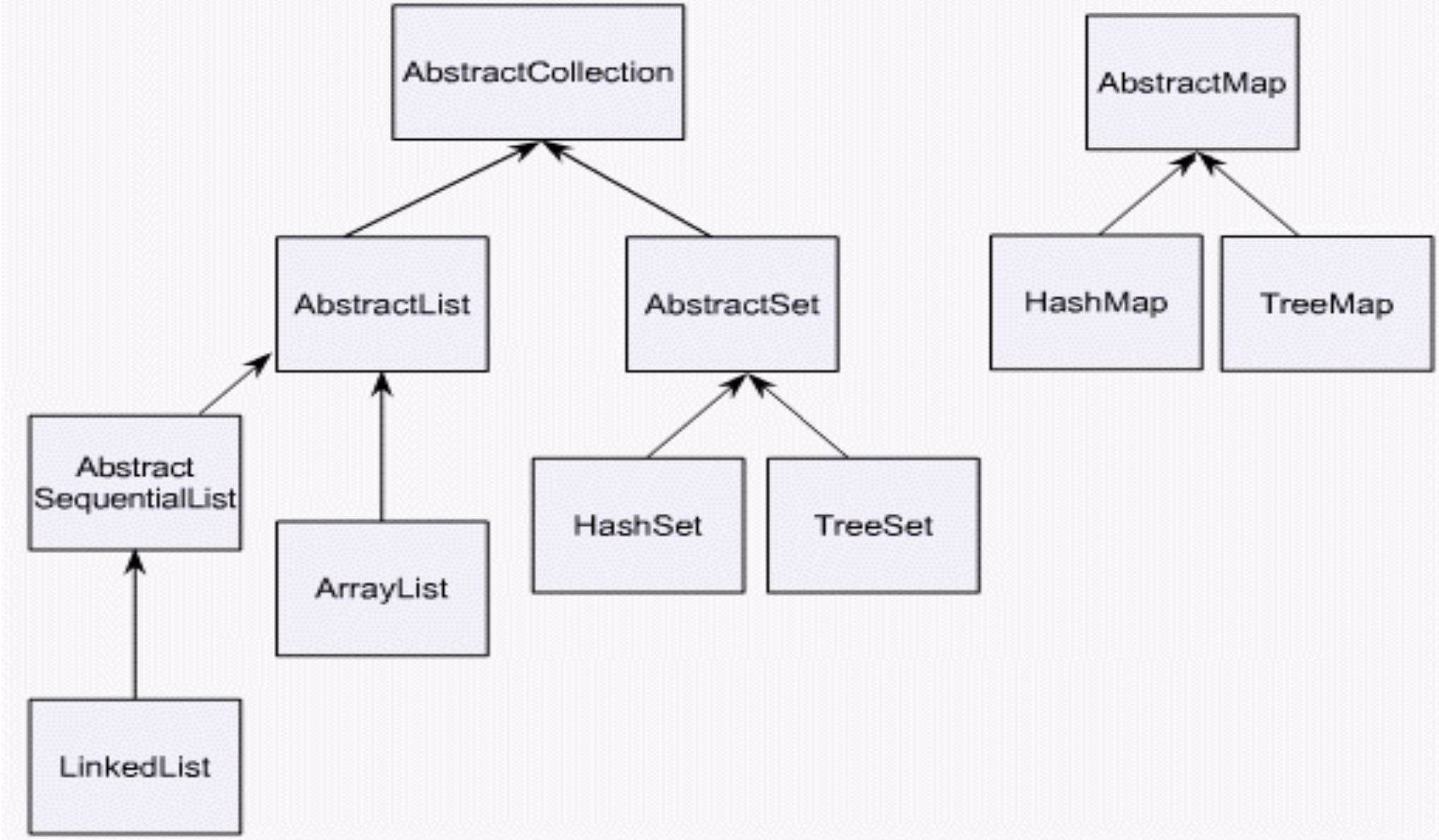
Iterator methods - to iterate through one element at a time. The iterator methods create an iterator object that has methods to iterate through the object. The iterator object can access the next element using the `Object next()` method, check to see if there are any more objects in the collection using the `boolean hasNext()` method and remove objects from a collection using the `void remove()` method.

Group methods - to deal with the entire collection. With these you programmers can add or remove more than one element with one method call. These include boolean `containsAll(collection collection)`, `boolean addAll(Collection collection)`, `void clear()`, `void removeAll(Collection collection)`, `void retainAll(Collection collection)`

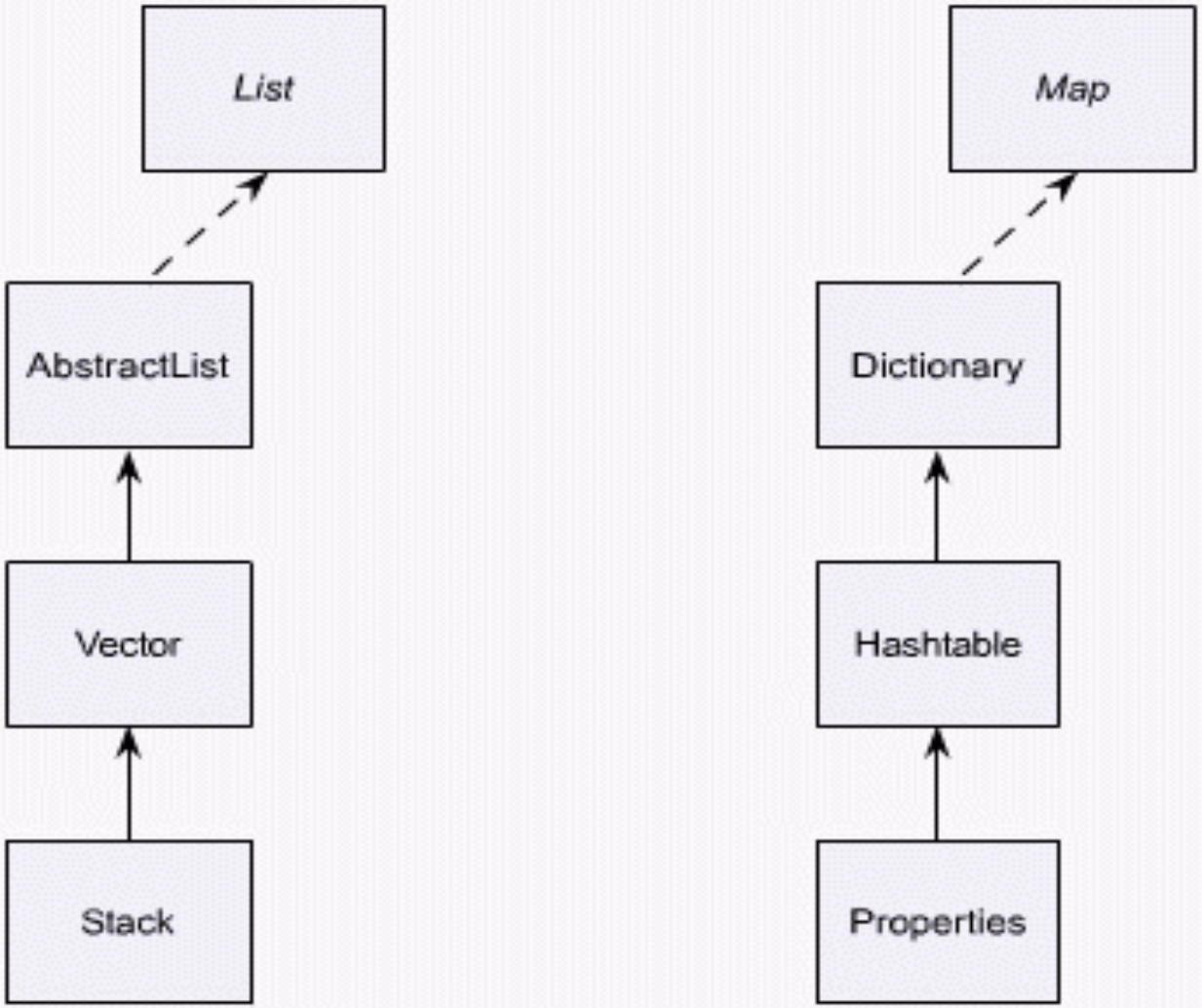
14.3.3 Collection classes

- One purpose of this chapter is to help programmers **select appropriate collection classes to contain their objects**.
- In general, the name of the class includes a reference to the **type of storage technology** that is implemented in the class.
- For example: A TreeSet class is therefore a collection of the **type Set** implementing a **Tree storage** technology.

Hierarchy of Collection Classes



Hierarchy of Legacy Classes



Legacy classes are older implementations of the Collection Framework

Hierarchy of Legacy Classes

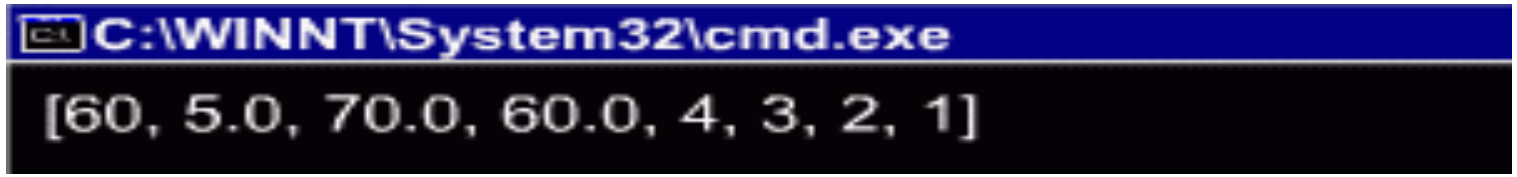
			Collection Properties			
Collection Class	**Storage Technologies**	**Collection Interface**	**Sorted**	**Ordered**	**Duplicates**	**Key**
LinkedList	Linked-List	List		x	x	
ArrayList	Array	List		x	x	
Vector	Array	List		x	x	
HashSet	HashTable	Set				
TreeSet	Tree	SortedSet	x			
HashMap	HashTable	Map				x
TreeMap	Tree	SortedMap	x			x
Hashtable/ Properties	HashTable	Map				x

14.3.4 Set objects(HashSet TreeSet)

- **HashSet**
- The HashSet class implements the Set interface.
- In the following example, the program declares a variable (set) of type Set and is initialized to a new HashSet object.
- It then adds a few elements and prints the set to standard output. A Set object **does not allow duplicate** objects to enter the collection.
- The collection is **unordered and unsorted**.

HashSet

- Set set = new HashSet();
- set.add(new Byte((byte)1));
- set.add(new Short((short) 2));
- set.add(new Integer(3));
- set.add(new Integer(4));
- set.add(new Float(5.0F));
- set.add(new Long(60));
- set.add(new Double(60.00));
- set.add(new Float(70.00));
- set.add(new Double(60));
- System.out.println(set);



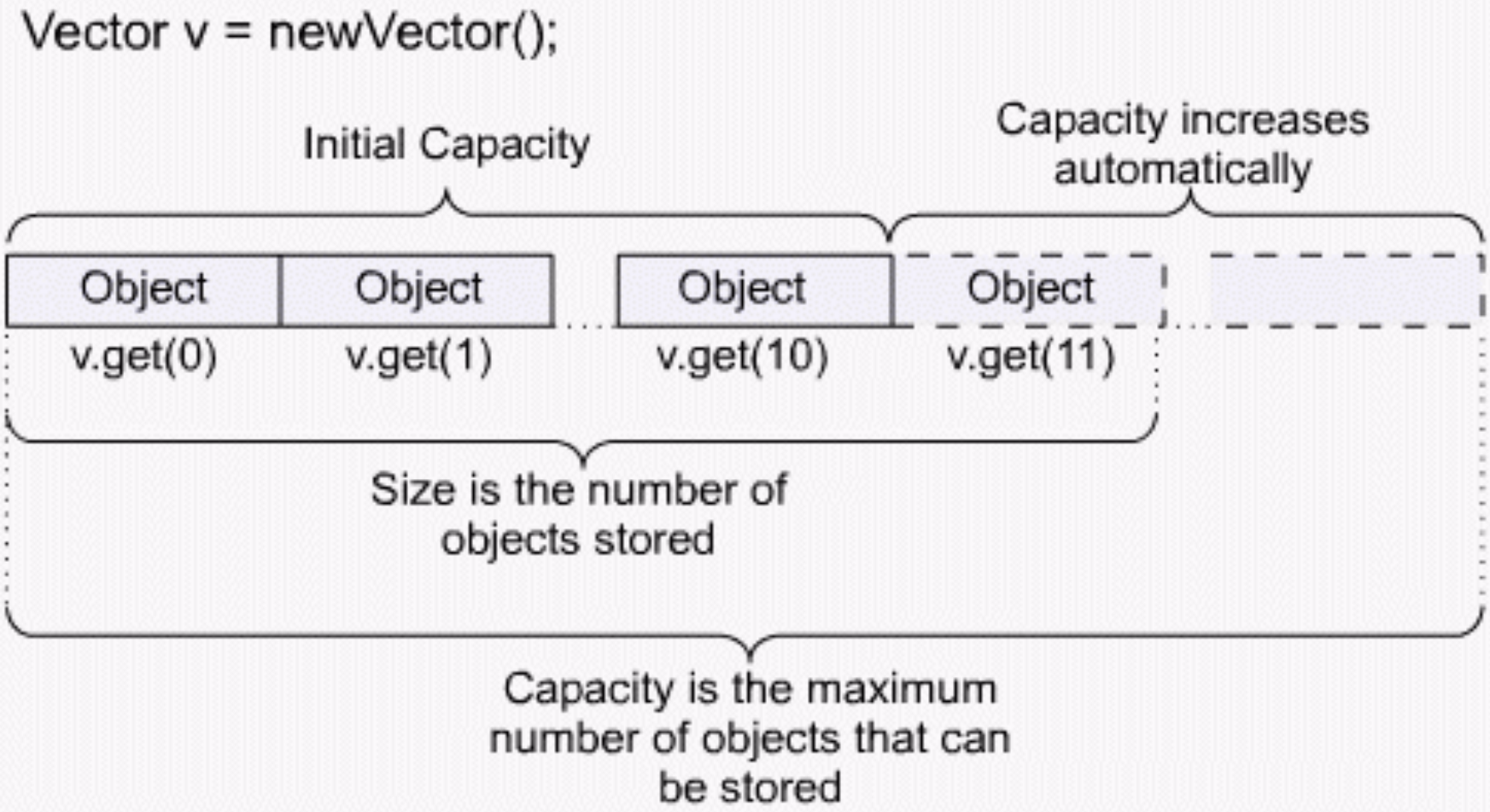
TreeSet

- The TreeSet class **implements the SortedSet** interface.
- These objects place the **elements in ascending order**.
- The elements must be **non-duplicating**.
- Because the elements are sorted, additional methods are available to take advantage of the order.
- **Example**: TreeSetExample

14.3.5 List objects

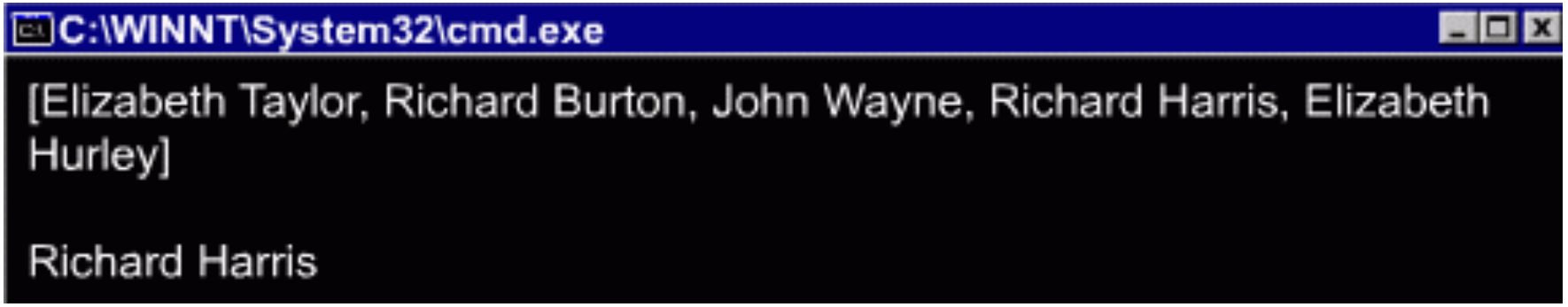
- **Vector, ArrayList, LinkedList**
- **Vector**
- The Vector class **implements the List interface**.
- The dimensions of the Vector object include its **capacity** (how many elements can be added) and its **initial size**.
- The capacity of a Vector object can be increased after the object has been created.
- The **number of objects represents the length** or elements of the vector.

Vector


Vector v = newVector();
Initial Capacity
Capacity increases automatically
Object
Object
Object
Object
v.get(0)
v.get(1)
v.get(10)
v.get(11)
Size is the number of objects stored
Capacity is the maximum number of objects that can be stored

Vector

- List v = new Vector();
- v.add("John Wayne");
- v.add(0, "Elizabeth Taylor");
- v.add("Richard Harris");
- v.add("Elizabeth Hurley");
- v.add(1, "Richard Burton");
- System.out.println(v);
- String name = (String) v.get(3);
- System.out.println(name);

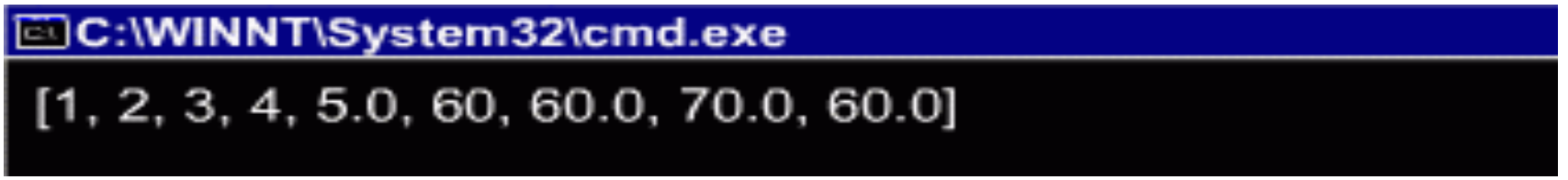


ArrayList

- The ArrayList class **implements the List** interface.
- Because lists **allow duplicates**, ArrayList also allow duplicates.

ArrayList

- List list = new ArrayList();
- list.add(new Byte((byte)1));
- list.add(new Short((short) 2));
- list.add(new Integer(3));
- list.add(new Integer(4));
- list.add(new Float(5.0F));
- list.add(new Long(60));
- list.add(new Double(60.00));
- list.add(new Float(70.00));
- list.add(new Double(60));
- System.out.println(list);

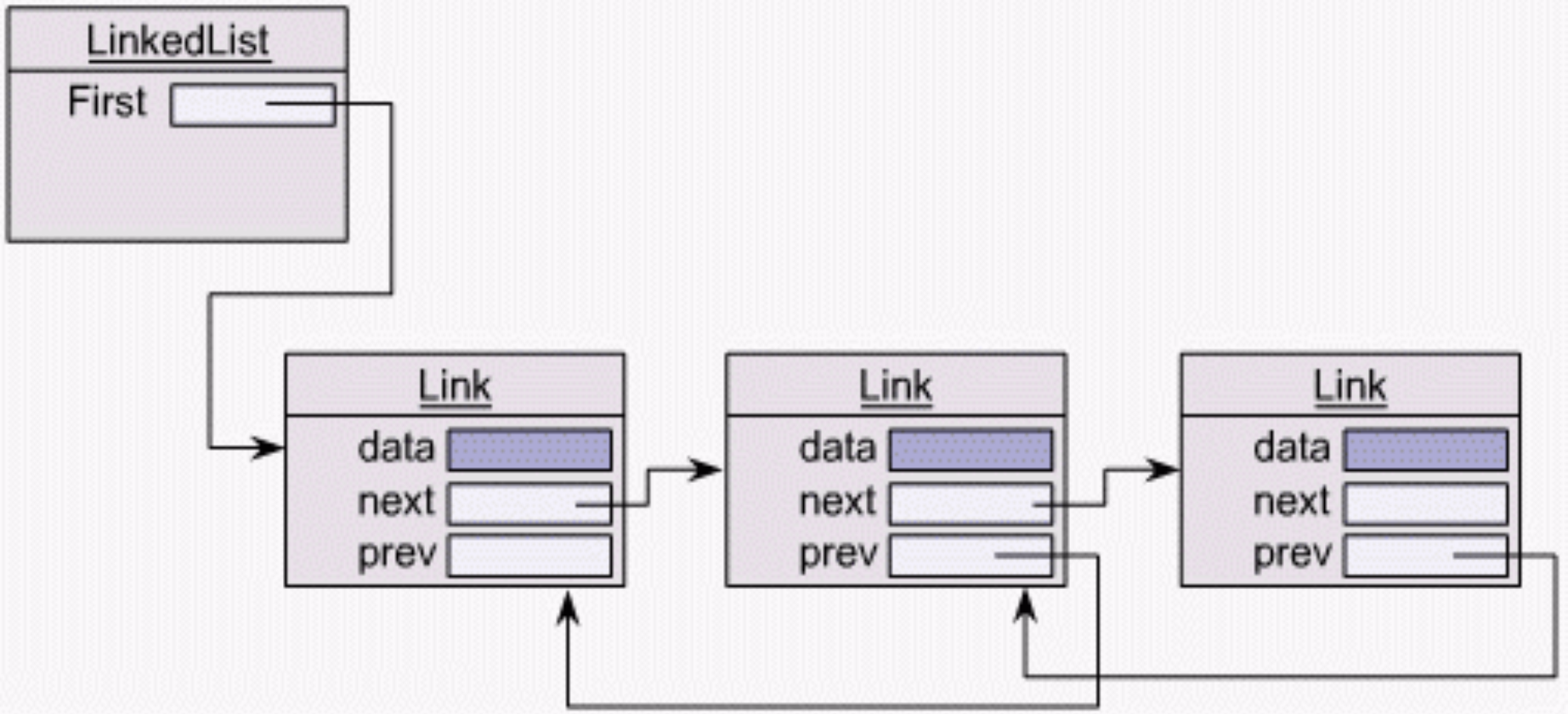


LinkedList

- The LinkedList class **implements the List** interface.
- **Removing and adding** elements in the middle of arrays **is time consuming**. This requires moving all the elements beyond the element. Each insertion or deletion will require copying and recopying the array because the array stores data in locations next to each other.

LinkList Objects

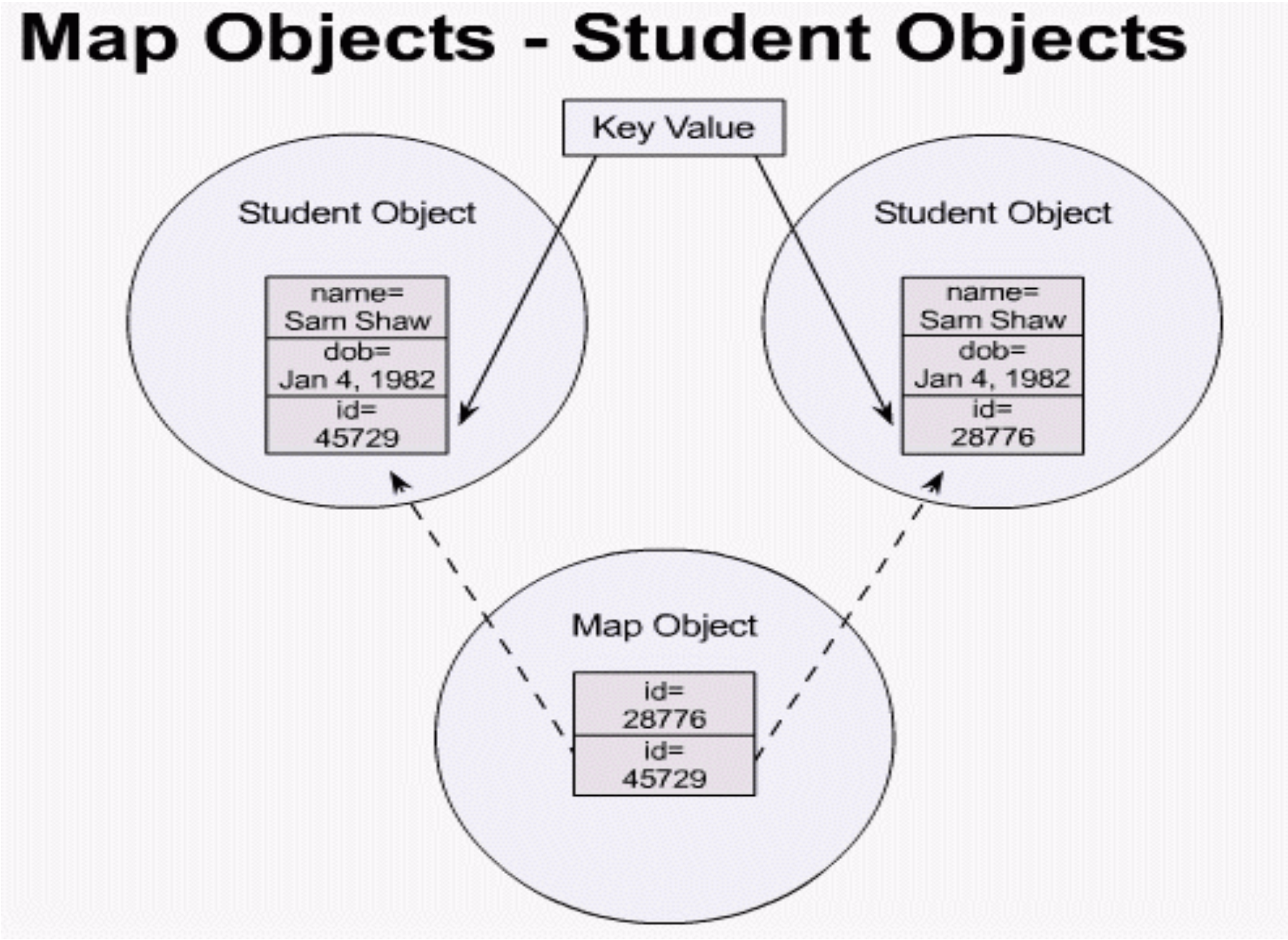
Example: LinkListExample



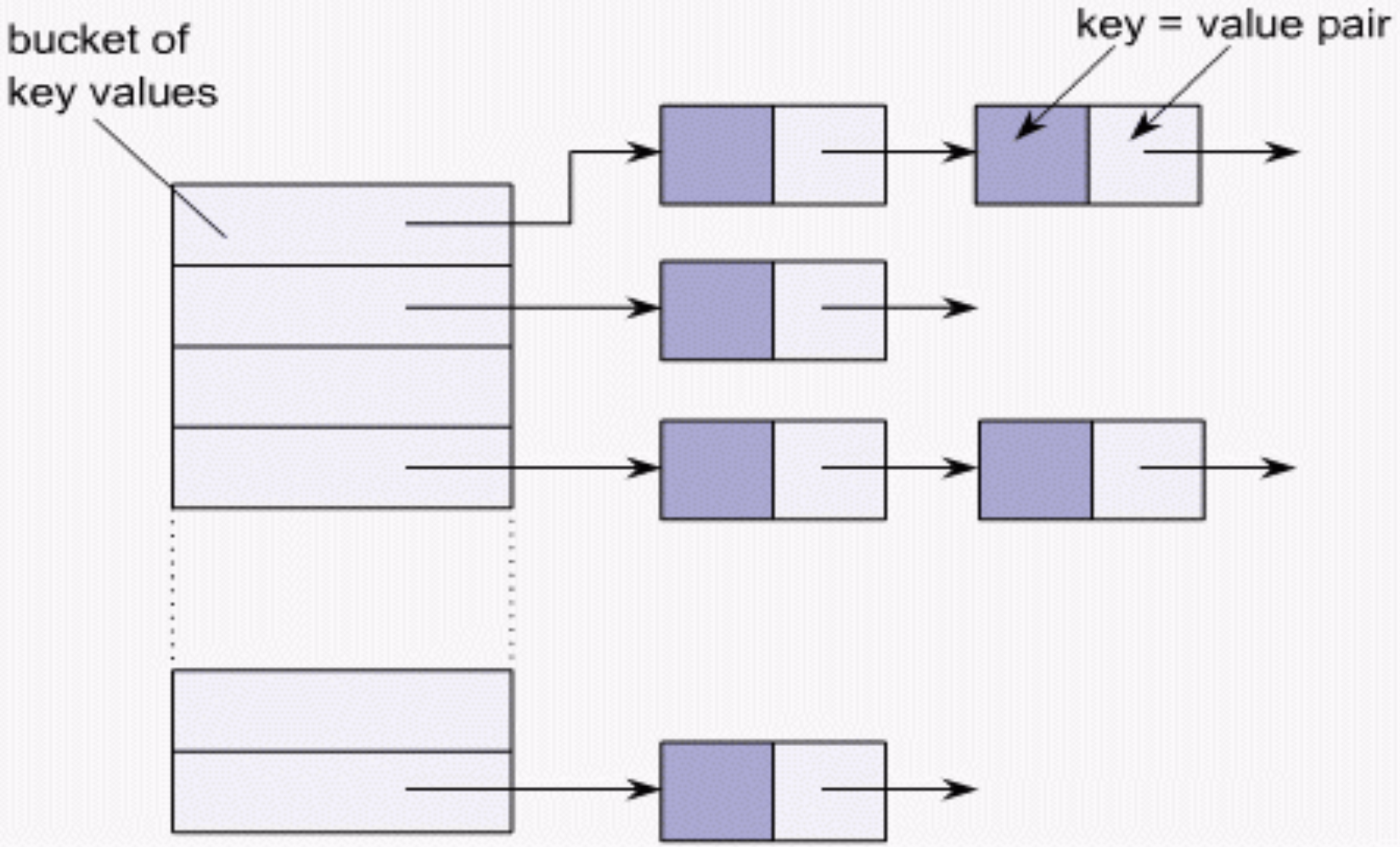
14.3.6 Map objects

- The Map interface **does not extend from the Collection interface**.
- Technically, it is not a collection, but in general terms it is part of the collections framework.
- It can be used to store and manage a collection of objects **using hashtable storage methods**.
- **A map is a collection of arbitrary associations between a key object and a value object.**

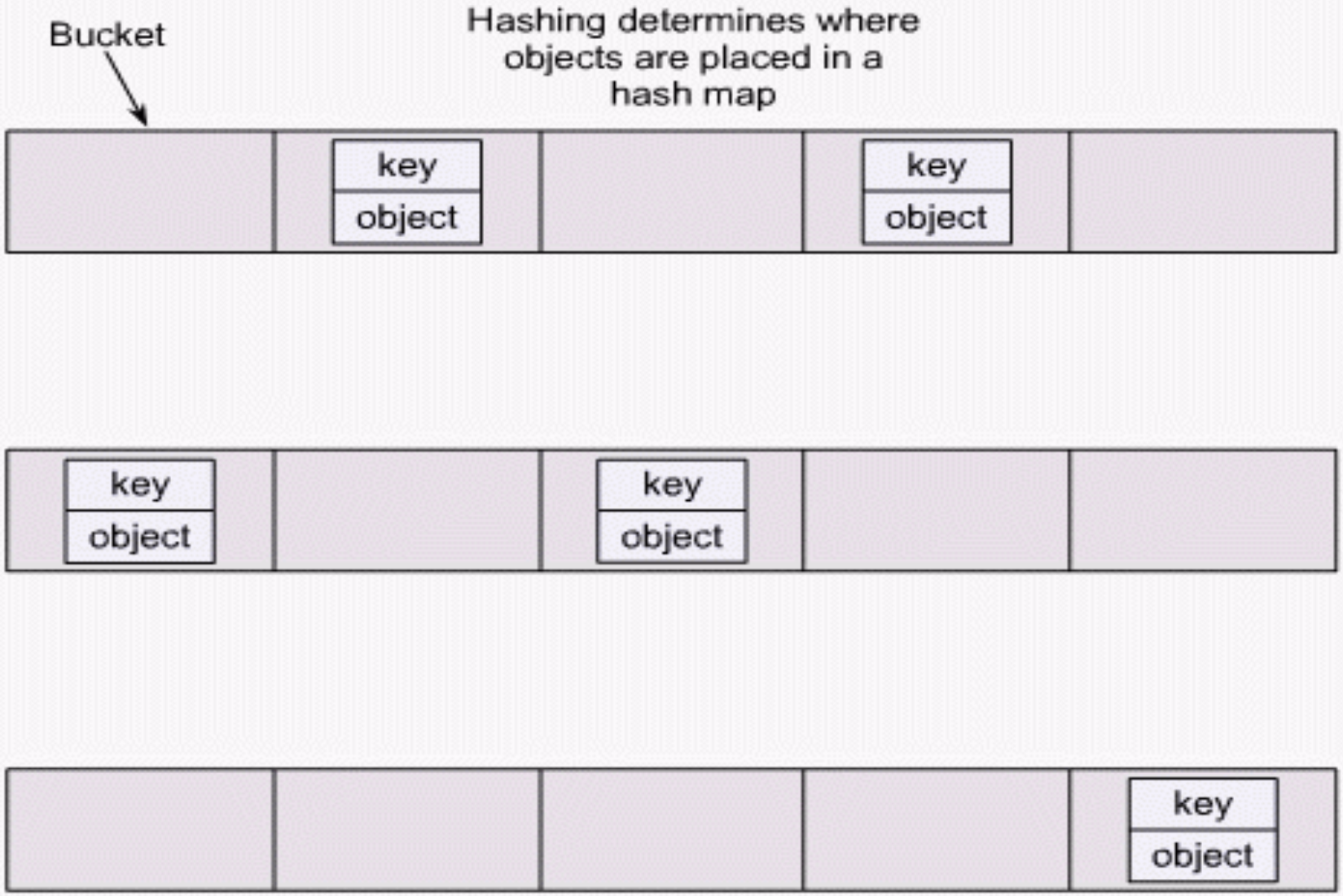
Map Objects - Student Objects



Bucket and hashvalue1



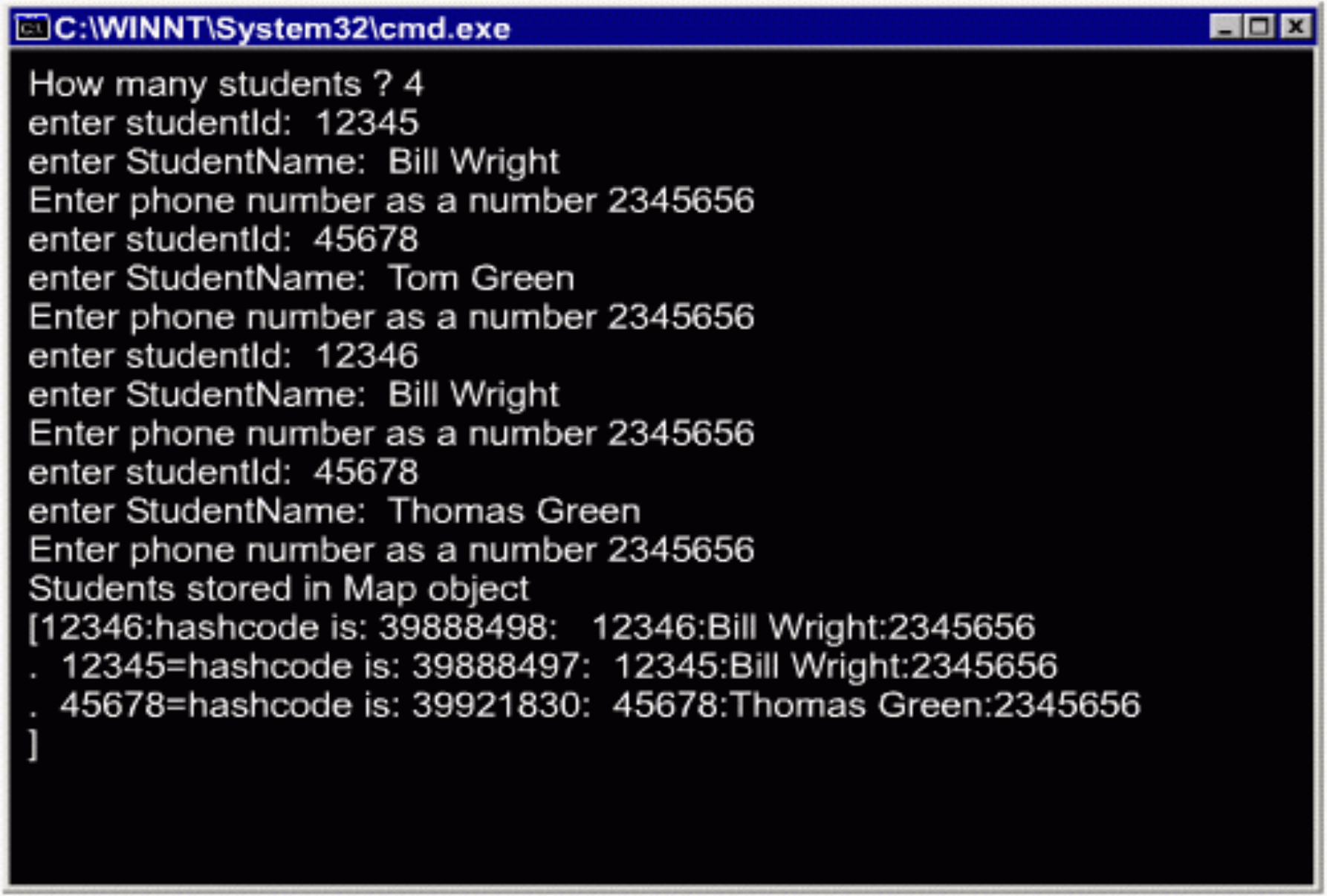
Bucket and hashvalue2



Example

- StudentHasHashValue.java
- MappedStudent.java

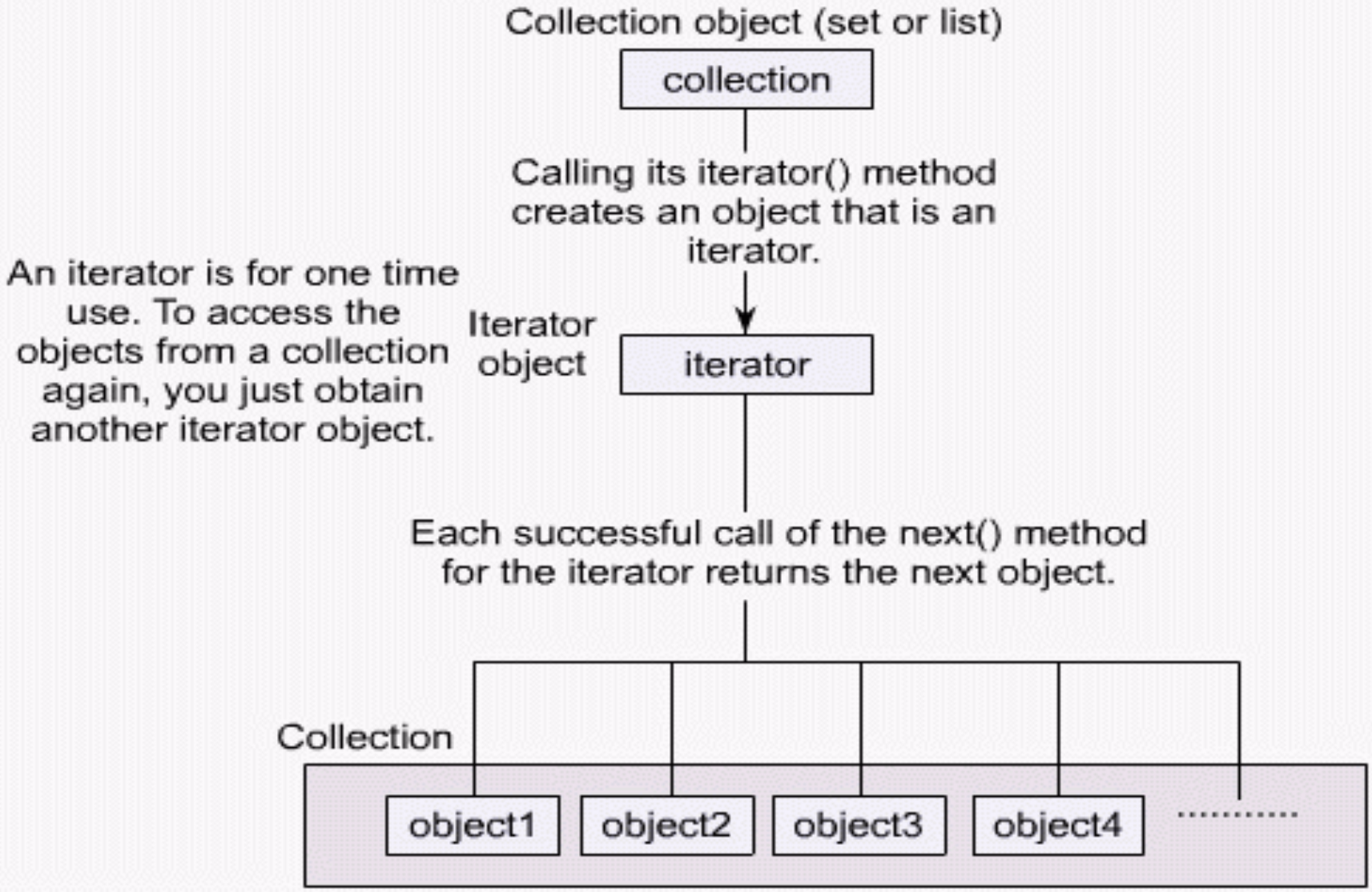
MappedStudent.java



14.3.7 Iterators

- A collection **can be scanned using an iterator**.
- Two iterator interfaces are available in the collection framework: the **Iterator** and its subclass the **ListIterator**.
- Iterators provide methods for scanning through **any** collection.
- Iterator objects provide for **scanning through** the list and for **adding and removing** elements from the collection.

Iterator Processing

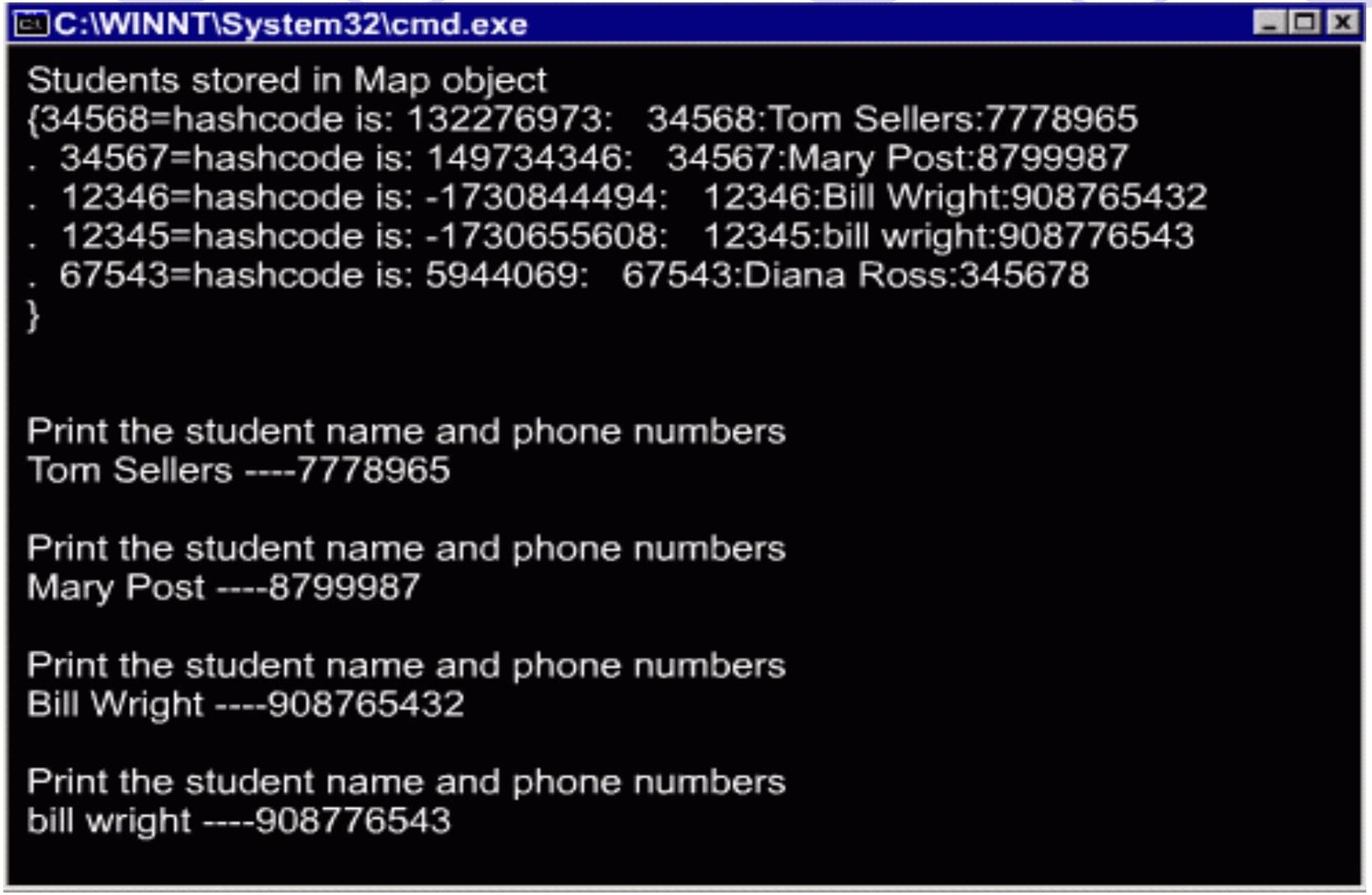


Common Iterator Methods

- **`public boolean hasNext()`** - returns true if a subsequent call to the next() method returns an element. True is there is another element on the list and false if there is none.
- **`public Object next()`** - returns the next element in the list.
- **`public void remove()`** - removes the last element returned by the next() method.

StudentIterator.java

Example: StudentIterator



14.3.8 Sorting and shuffing list objects

- **These two method are very efficient.**
- **Sorting**
- Sorting is fundamental to any complex business application, especially when generating reports.
- The **Collections class** provides a set of **static** methods for sorting:

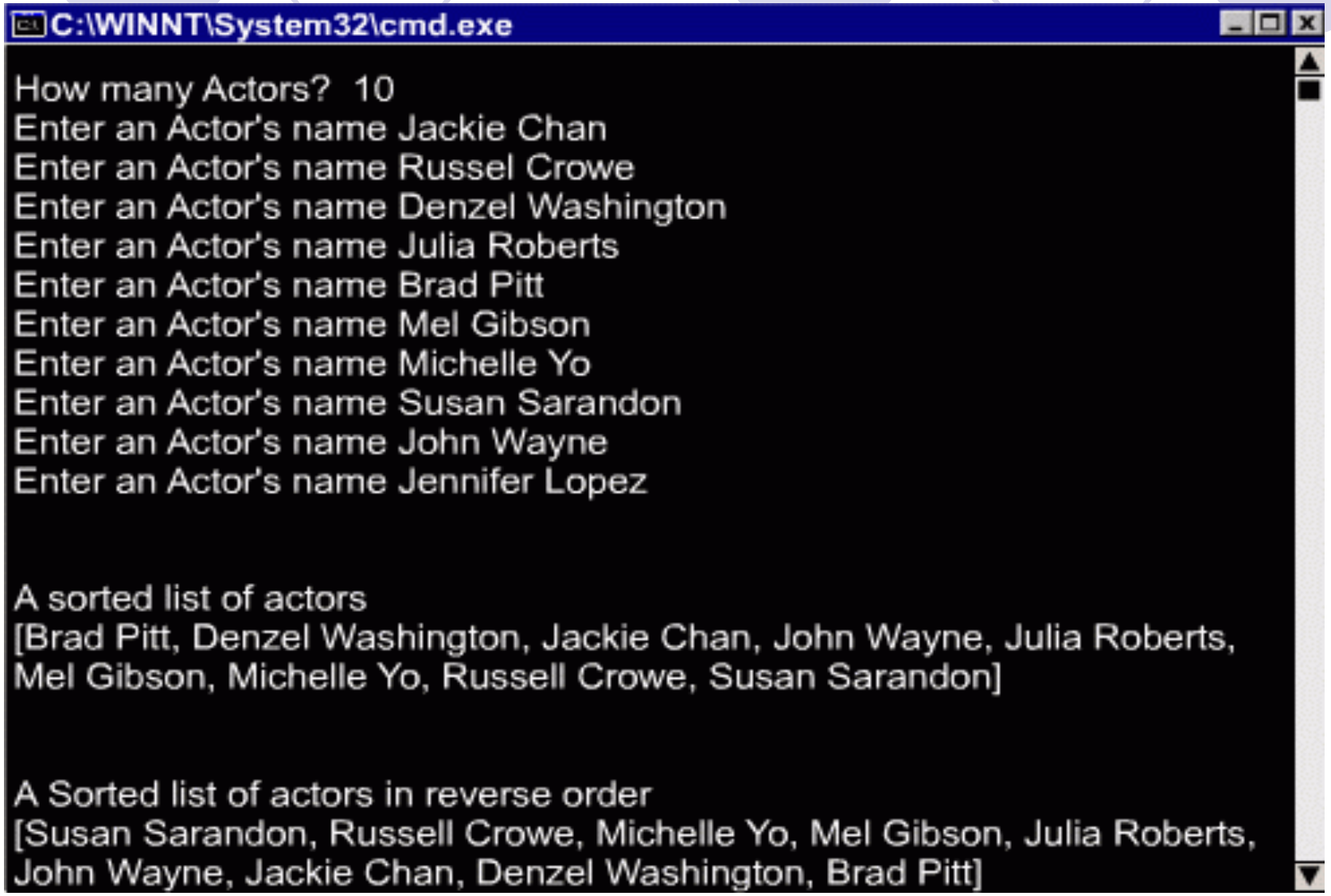
  Collections.sort()
  Collections.reverse().

Shuffing

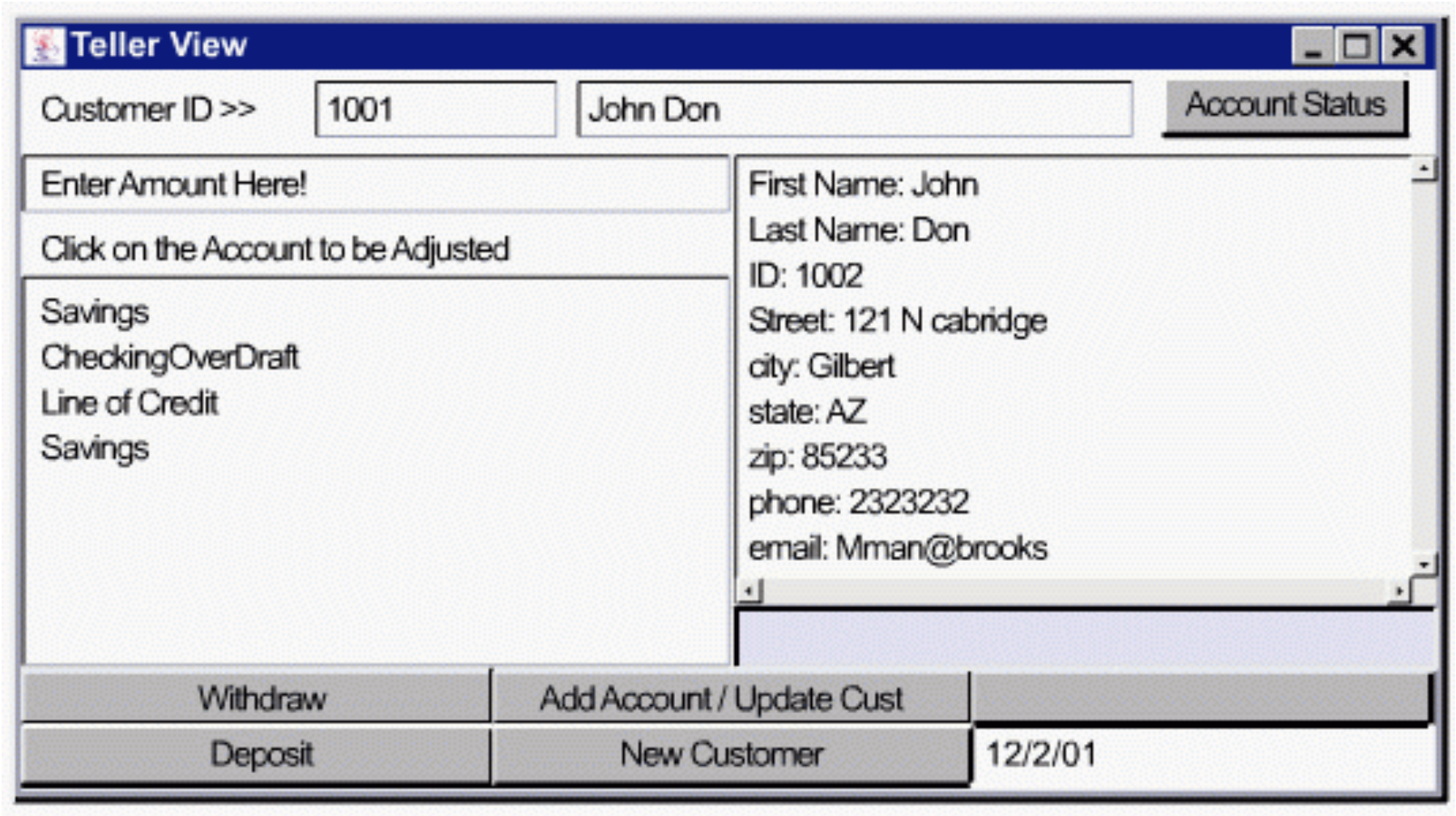
- Shuffling **rearranges the elements in a List object in some random order.**
- The **Collections.shuffle()** method is overloaded.
- One version uses a **default source for determining the random order**.
- The second version **accepts a Random object to determine the randomness** for positioning the elements in the array.

ActorsList

Example: ActorsList



14.4 Case Study: JBANK Application



The JBANK Application