Collections

Bob Singh

Data structures

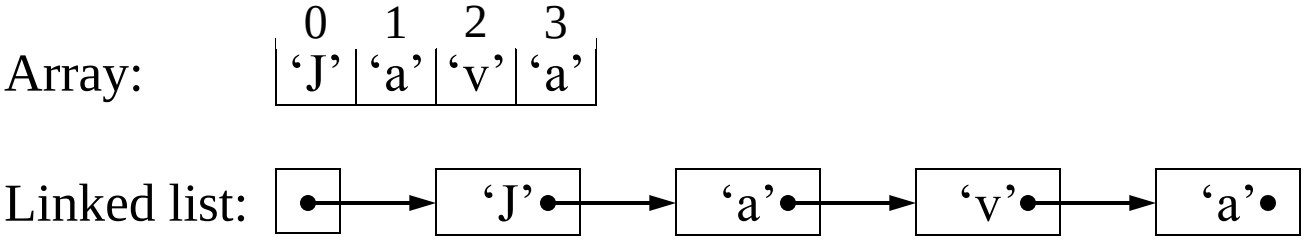
- A **data structure** is a systematic way of organizing a collection of data.
- A **static data structure** is one whose capacity is fixed at creation.
  E.g.: array.
- A **dynamic data structure** is one whose capacity is variable, so it can expand or contract at any time.
  E.g.: linked list, binary tree.
- For each data structure we need algorithms for insertion, deletion, searching, etc.

Generics

- What are generics? (Program 1)
- Multiple Type Definitions (Program 2)
- T extends Y (Program 3)
- ? in argument for comparing generic types (Program 4)

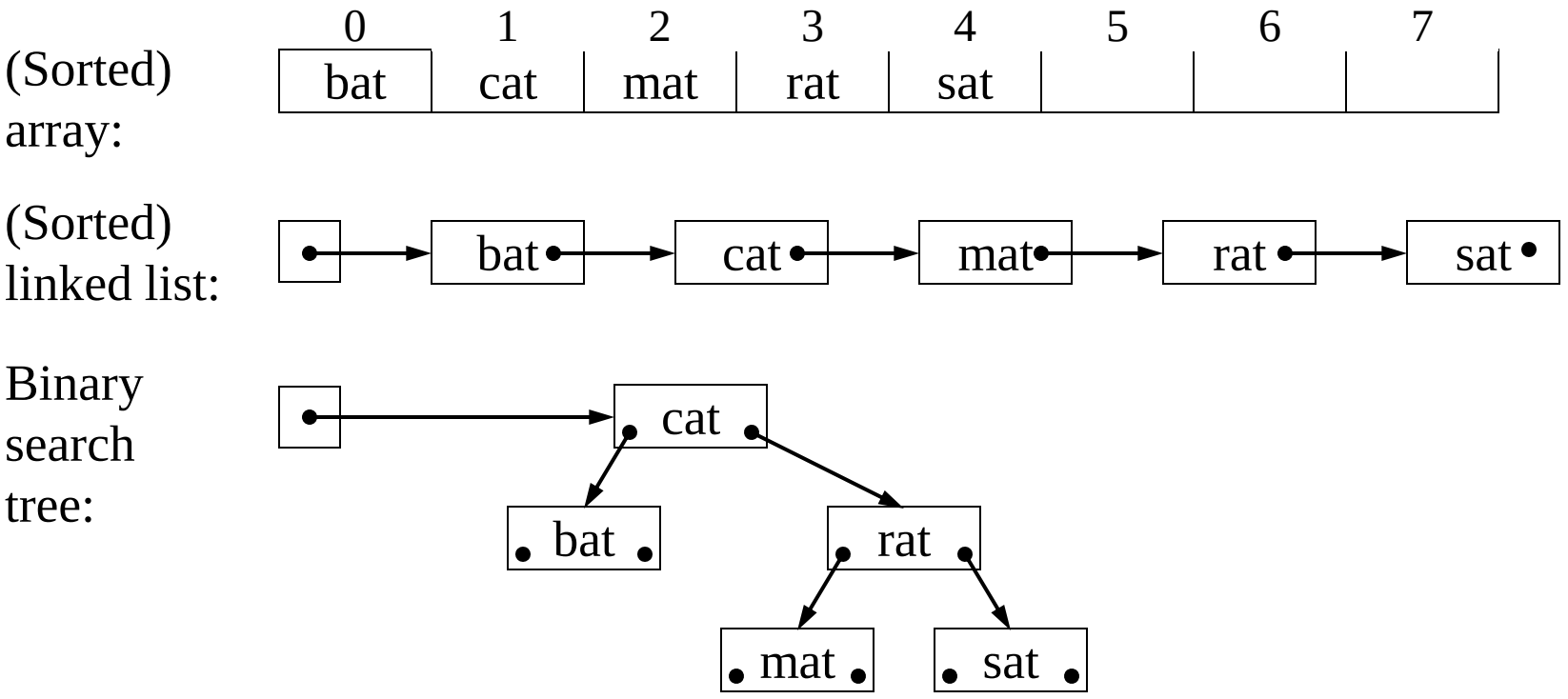
Example: representing strings

- Possible data structures to represent the string “Java”:



Example: representing sets

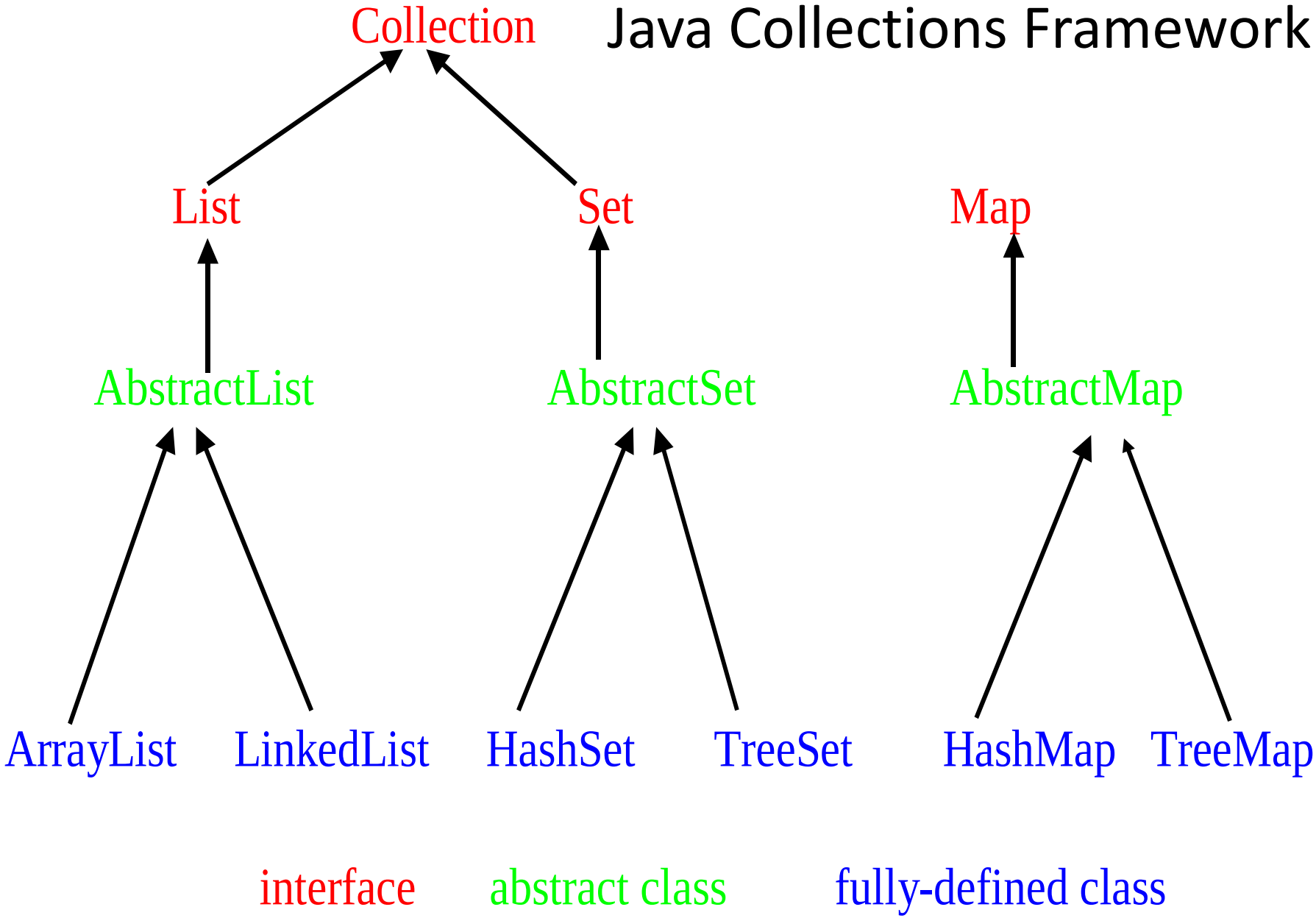
- Possible data structures to represent the set of words {bat, cat, mat, rat, sat}:



Collection

- An object, which groups multiple elements into a single unit.
- Used to store, retrieve and manipulate data
- Used to transmit data from one method to another.

Java Collections Framework



Java Collections Framework

Advantages

- Reduces Programming by facilitating interoperability among unrelated APIs
- Increases Program Speed and Quality by implementations of useful data structures and algorithms.
- Reduces the effort to design new APIs
- Software reuse

Java Collections Framework

Disadvantages

- Complex Implementation

Collection

- Root of the Collection hierarchy
- Represents a group of objects, known as its elements
- Some Collection implementations allow duplicate elements and others do not
- Some are ordered and others unordered
- The JDK doesn't provide any direct implementations of this interface

Collection

- List and Set extend Collection
  - SortedSet extends Set
- Map is a top-level interface
  - SortedMap extends Map
- Comparator stands alone
- Iterator is a top-level interface
  - ListIterator extends Iterator

Collection

- Supports basic operations like adding and removing.
- When you try to remove an element, only a single instance of the element in the Collection is removed, if present.

```
boolean add(Object element);
boolean remove(Object element);
```

Collection

Basic Collection Operations

```
public interface Collection
{
   boolean add(Object element);
   boolean remove(Object element);
   int size();
   boolean isEmpty();
   boolean contains(Object element);
   Iterator iterator();
}
```

List

- Two List implementations
  - **`ArrayList`** (good for random access)
  - **`LinkedList`** (good for inserts and removal)
- LinkedList implements java.util.List interface
  - Uses linked list for storage
  - Allows elements to be added, removed from the collection at any location
  - Elements accessed sequentially
  - Can be referred as a Stack or Queue (depending on the implementation)

List (Program 5)

```
import java.util.*;
public class LinkedListTest
{
  public static void main(String args[])
  {
    LinkedList queue = new LinkedList();
    queue.addFirst("John");
    queue.addFirst("James");
    System.out.println(queue);
    queue.removeLast();
    System.out.println(queue);
   }}
```

```
Output John,James
John
```

List

- ArrayList implements java.util.List interface
- Uses array for storage
  - Array storage is generally faster but it has limitations like, cannot insert, and delete entries in middle of the list.

List (Program6a, Program6b)

```
import java.util.*;
public class TestArrayList
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    list.add("John");
    list.add("James");
    System.out.println(list);
    System.out.println("2: “ +list.get(2));
    System.out.println("1: " +list.get(1));
   }
}
```

Set

- In Set implementations null is valid entry, but allowed only one time.
- The JDK contains two general-purpose Set implementations
  - **`HashSet`** implements java.util.Set
    - Stores its elements in a hash table and is faster
  - **`TreeSet`** implements java.util.Set interface
    - Provides an ordered set, uses tree for storage. It allows elements to be added, removed at any location in the container by insisting means of ordering.

Set (Program 7)

```
import java.util.*;
public class TestTreeSet
{
    public static void main(String args[])
    {
                Set testSet = new TreeSet();
                testSet.add("John");
                testSet.add("Jim");
                System.out.println(testSet);
    }
}
Output: John,Jim
```

Set (Program 8)

- Example of HashSet, TreeSet and LinkedHashSet

Map

- java.util.Map
- Set with no duplicates – must have unique keys
- Maps the key values against the records in the database
- Key values used to lookup, or index the stored data
- Has it's own hierarchy.
- In Map interface, null is a valid entry, but allowed only one time

Map

- Two Map implementations available in the Collections Framework:
  - HashMap implements java.util.Map interface
    - Uses hashing for storage
    - Does not permit duplicates
  - TreeMap implements java.util.Map interface
    - uses tree for storage.
    - Provides the ordered map

Map

```
public interface Map
{
    // Basic Operations
    Object put(Object key, Object value);
    Object get(Object key);
    Object remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
}
```

Iterator

- Can traverse a collection from start to end and safely remove elements from the underlying Collection.
- The iterator() method generally used in query operations. Basic methods:
    - iterator.remove();
    - iterator.hasNext();
    - iterator.next();
- The ListIterator interface extends the Iterator interface to support bi-directional access

Map (Program 9)

- HashMap

Map (Program 10)

- TreeMap

Composite Data Structure (See Program 8)

- LinkedHashSet

Composite Data Structures (Program 11)

- LinkedHashMap