Lecture 15 Summary

- Collections Framework
  - Iterable, Collections
  - List, Set
  - Map
- Collections class
- Comparable and Comparator

By the end of this lecture, you will be able to use different types of Collections and Maps in your Java code.

You will also be able to use Comparables and Comparators to simplify many algorithms.

In Python, what are lists, dictionaries, and tuples?

What benefit do they provide?

How would we get the same benefit in C?

Collections

- Store a bunch of stuff

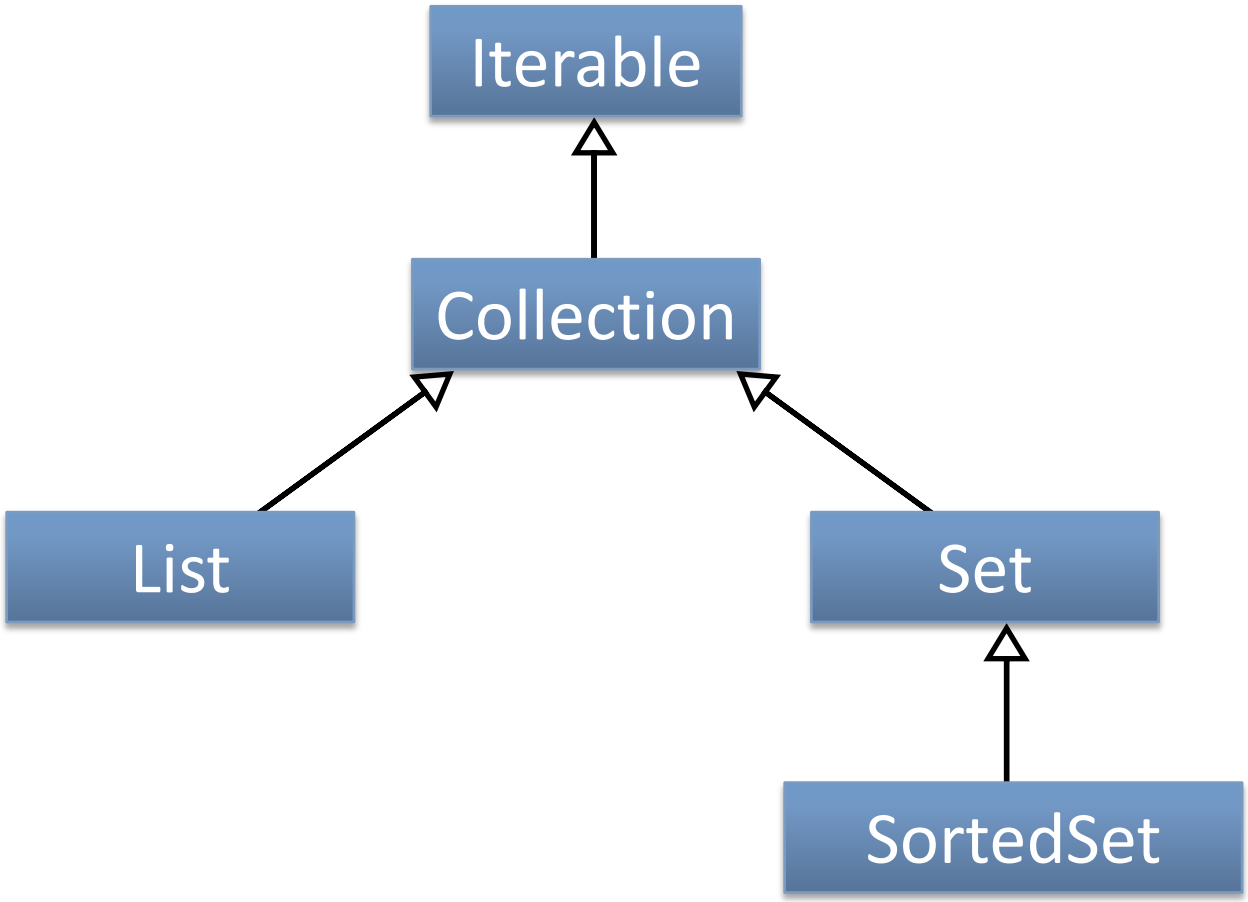
- In Python, how this stuff is stored is hidden from the programmer
- In C, have complete control over how stuff is stored (but we have to think about it)
- In Java, we have a choice – why?

Collections

- Actually, we have this choice in Python, C++, and Java

- Object-oriented programming provides a way of hiding when we want to (through inheritance)

Collection Framework in Java

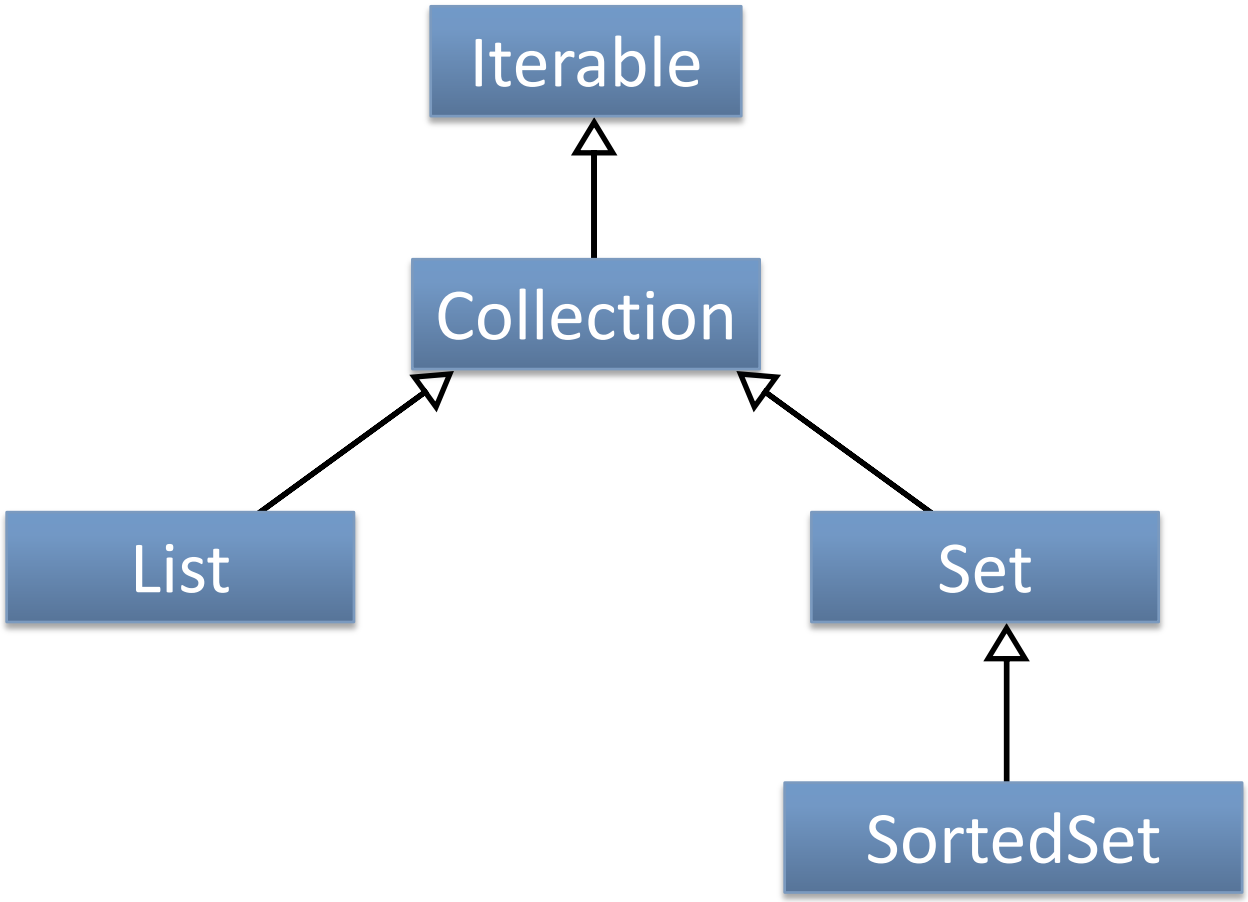


Iterable

```
public interface Iterable
{
    public Iterator iterator();
}
```

- What does this mean about anything that is an Iterable?

Collection Framework in Java



Collection

```
public interface Collection extends Iterable
{
    public boolean add(o);
    public boolean addAll(Collection c);
    public void clear();
    public boolean contains(Object o);
    public boolean containsAll(Collection c);
    public boolean equals(Object o);
    public int hashCode();
    public boolean isEmpty();
    public Iterator iterator();
    public boolean remove(Object o);
    public boolean removeAll(Collection c);
    public boolean retainAll(Collection c);
    public int size();
    public Object[] toArray();
}
```

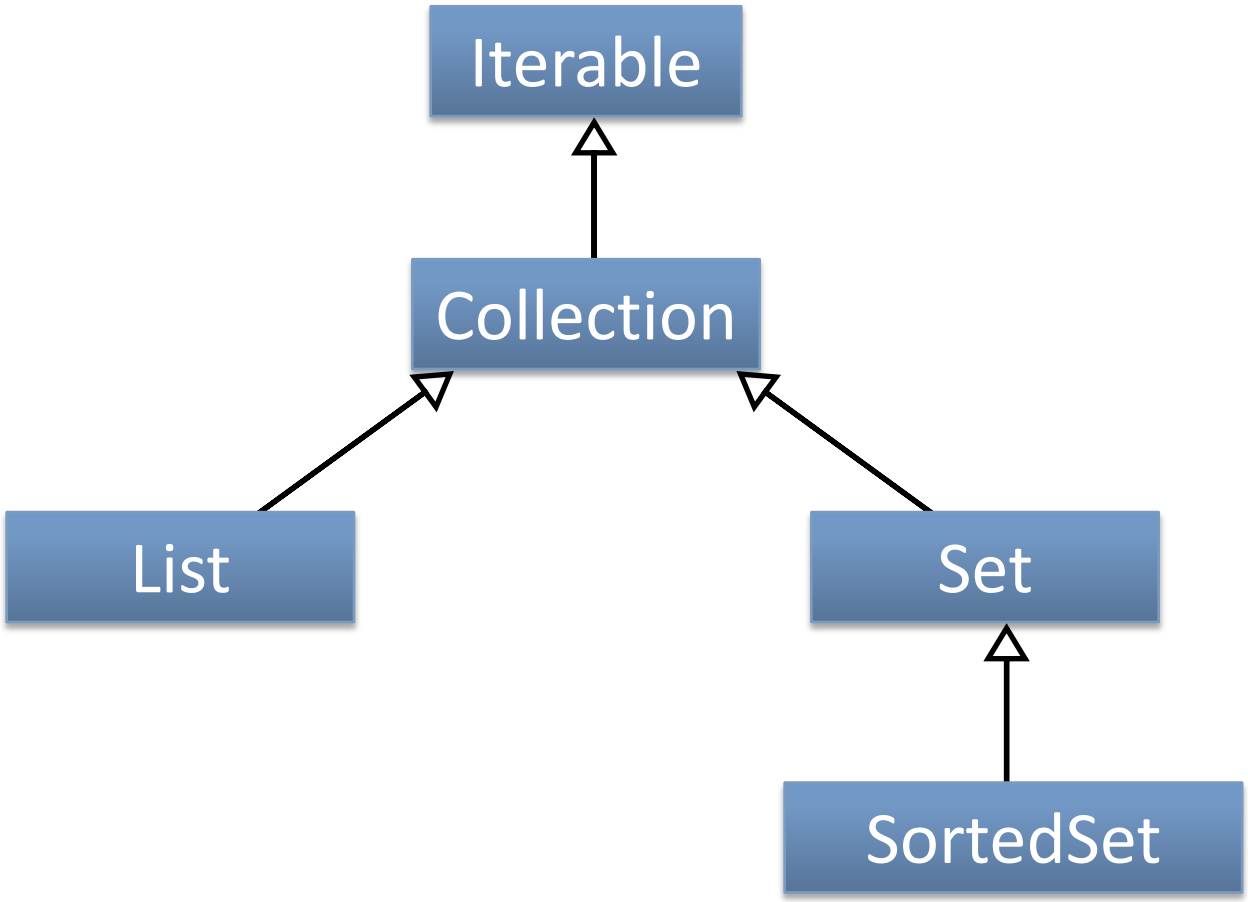
Collection

Method	Description
add	Add an entity to the collection
clear	Clear all entities from the collection
contains	Determine if the collection contains a specific entity
isEmpty	Indicates whether the Collection is empty or not
iterator	Provides an iterator which can be used to iterate through the entities contained within the collection
remove	Removes a specific entity from the Collection
size	Returns the number of entities currently contained within the Collection

How can collections be accessed?

```
public interface Collection extends Iterable
{
    public boolean add(o);
    public boolean addAll(Collection c);
    public void clear();
    public boolean contains(Object o);
    public boolean containsAll(Collection c);
    public boolean equals(Object o);
    public int hashCode();
    public boolean isEmpty();
    public Iterator iterator();
    public boolean remove(Object o);
    public boolean removeAll(Collection c);
    public boolean retainAll(Collection c);
    public int size();
    public Object[] toArray();
}
```

Collection Framework in Java



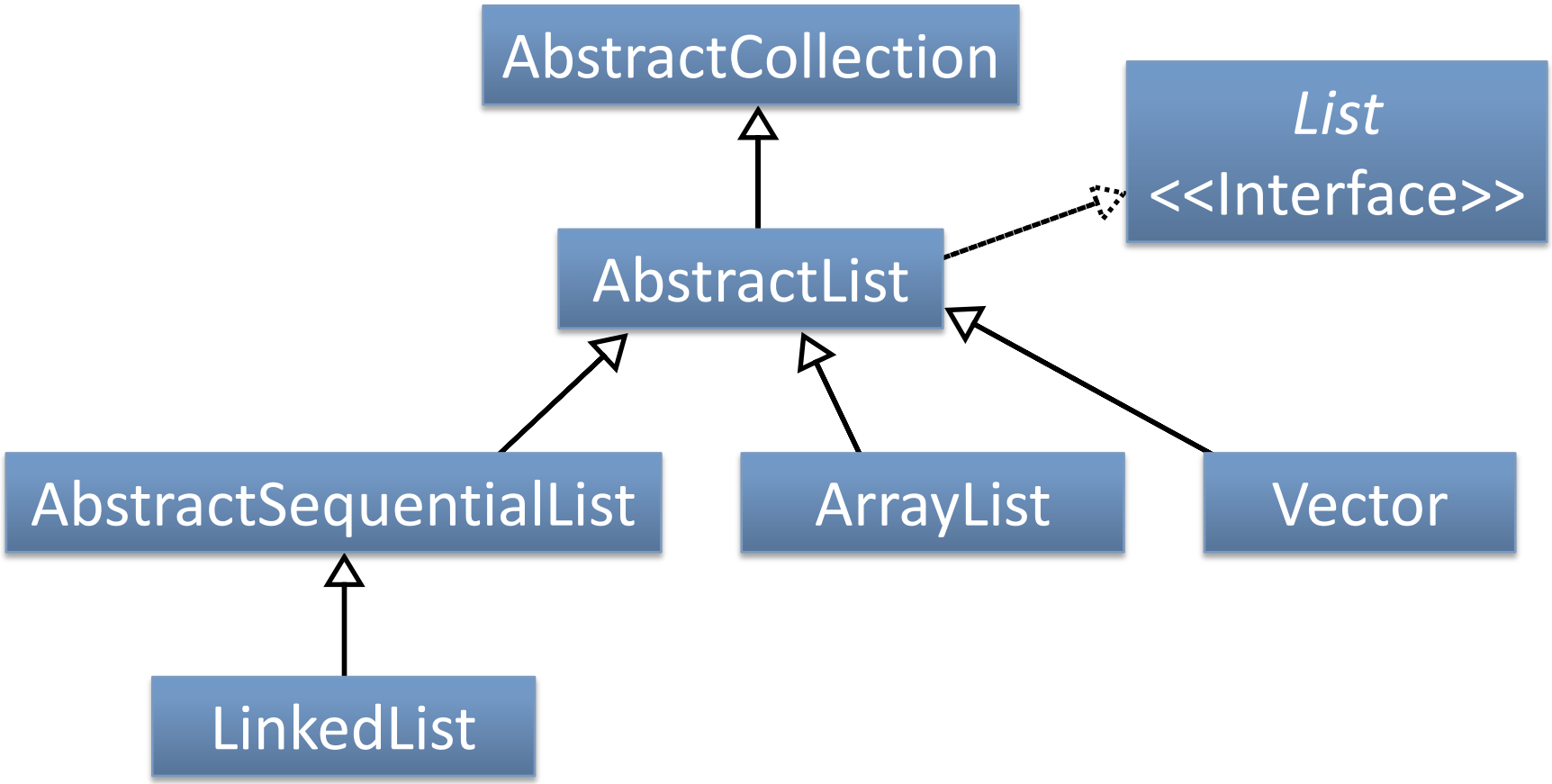
List

```
public interface List extends Collection
{
    public void add(int index, Object element);
    public boolean addAll(int index, Collection c);
    public Object get(int index);
    public int indexOf(Object o);
    public int lastIndexOf(Object o);
    public ListIterator listIterator();
    public ListIterator listIterator(int index);
    public Object remove(int index);
    public Object set(int index, Object element);
    public List<E> subList(int fromIndex, int toIndex);
}
```

List

Method	Description
add(int index, Object element)	Insert an object into the list at the specified index
get(int index)	Get a reference to the object at the specified index
indexOf(Object o)	Return the index of a specific object within the list
lastIndexOf(Object o)	Return the last index of a specific object within the list
listIterator()	Get an iterator which allows you to iterate over the list in forward or reverse direction
listIterator(int index)	Get an iterator which allows you to iterate over the list in forward or reverse direction starting at the specified location
remove(int index)	Remove the object at the specified index
set(int index, Object element)	Replace the object at a specific index with the newly specified element
subList(int fromIndex, int toIndex)	Obtain a sublist based on specified indices

Classes that implement List



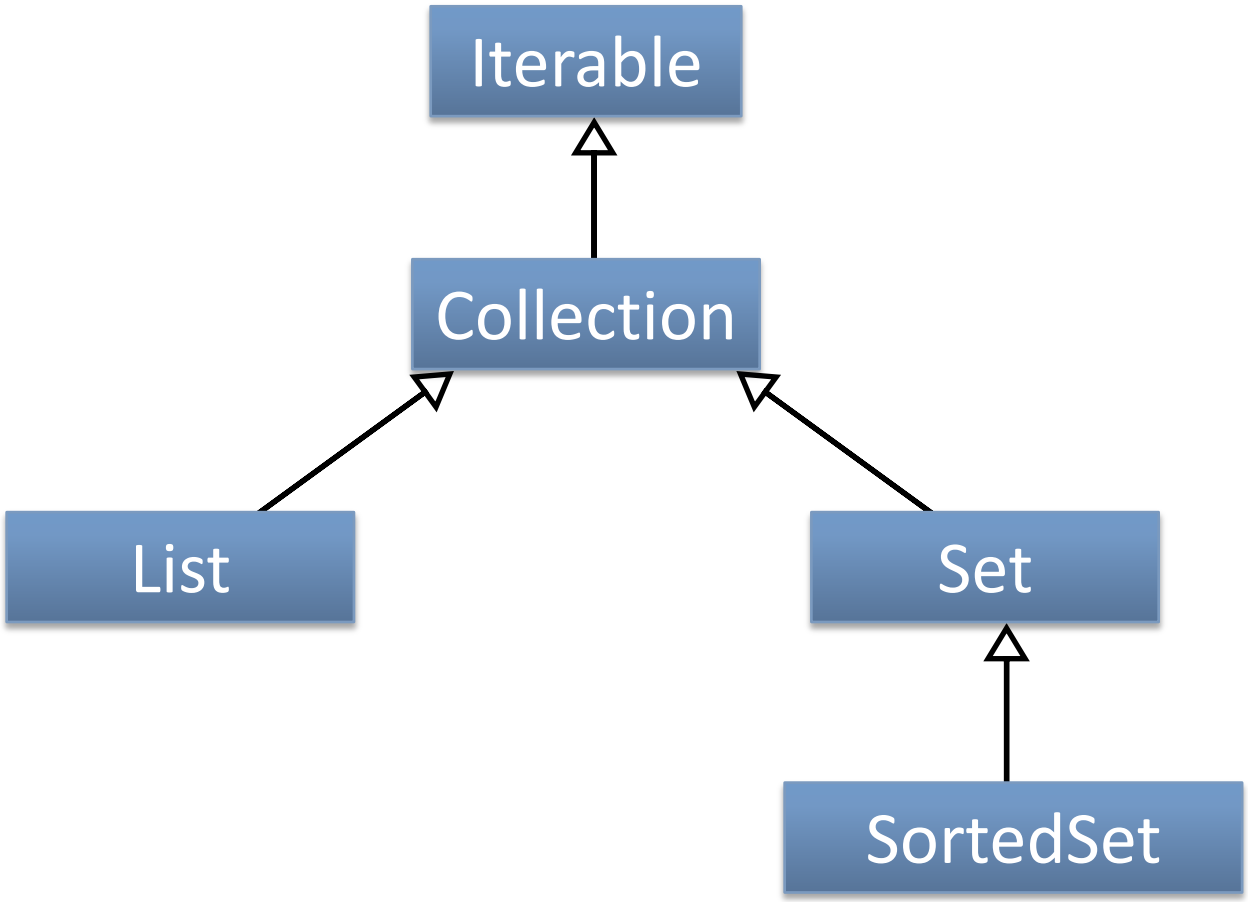
Classes that implement List

- This diagram has 3 abstract list classes and 3 concrete list classes
- ArrayList & Vector
  - internally implemented with arrays
  - their difference is beyond the scope of what you know so far
- LinkedList
  - internally implemented with linked lists

Exercise

- Create a program in Java that adds all of the strings from the command line to an ArrayList.

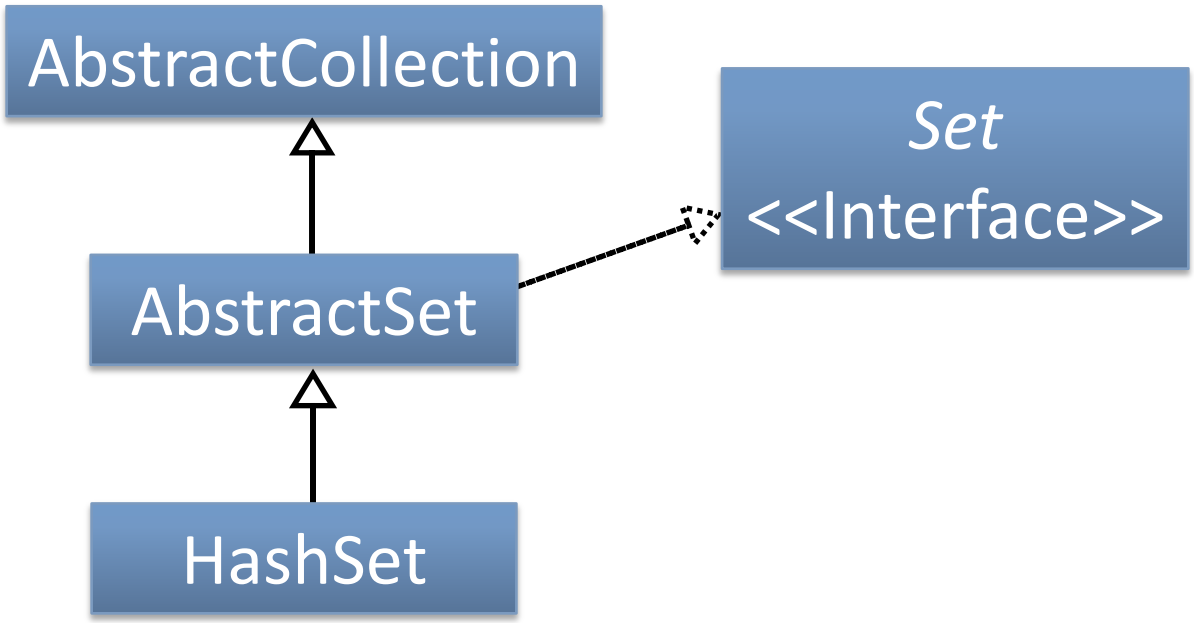
Collection Framework in Java



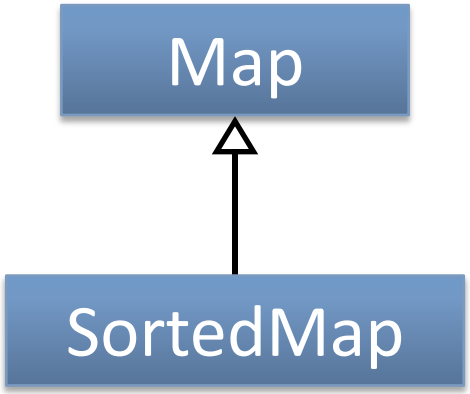
Set & SortedSet

- Set
  - no additional methods
  - adds restriction that no duplicate elements may be added
- SortedSet
  - additional methods (e.g., first, last)
  - elements have an order

HashSet (implements Set interface)



Map Interface



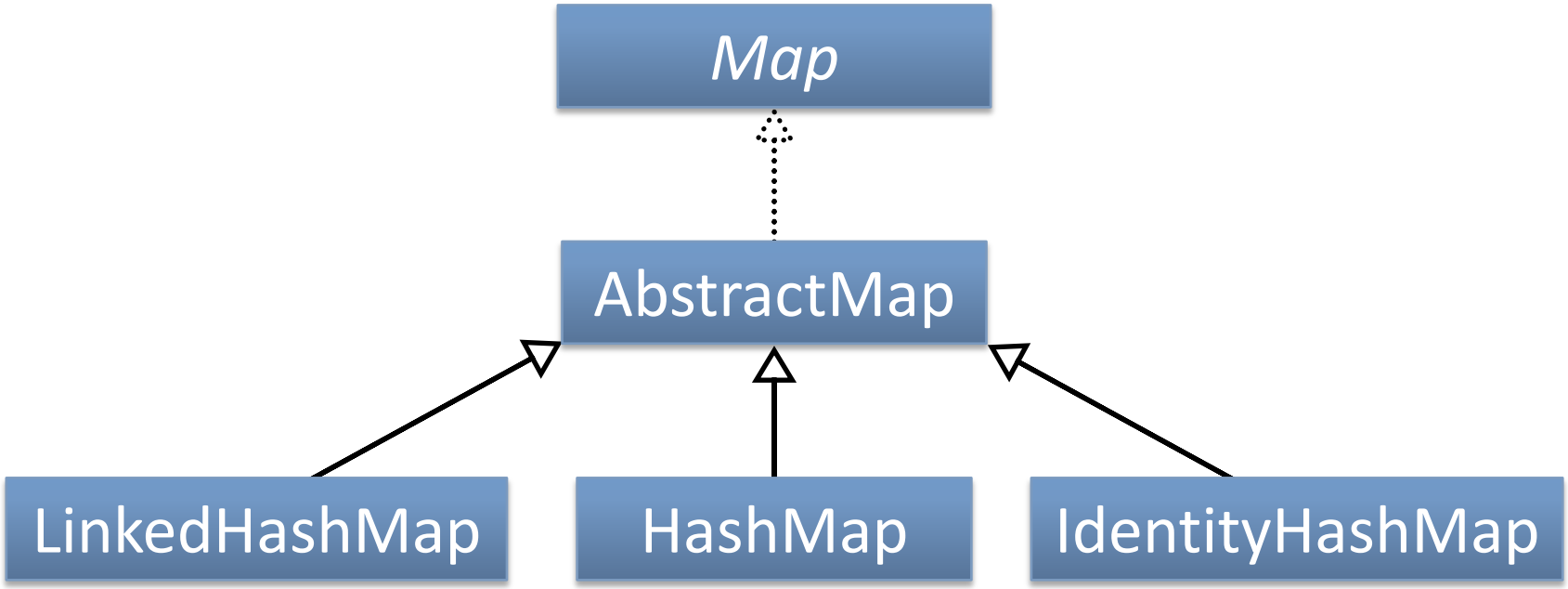
Map Interface

```
public interface Map
{
    public void clear();
    public boolean containsKey(Object key);
    public boolean containsValue(Object value);
    public Set entrySet();
    public boolean equals(Object o);
    public Object get(Object key);
    public int hashCode();
    public boolean isEmpty();
    public Set keySet();
    public Object put(Object key, Object value);
    public void putAll(Map t);
    public Object remove(Object key);
    public int size();
    public Collection values();
}
```

Map

Method	Description
clear()	Removes all mappings from the map
containsKey(Object key)	Returns true if map contains the specified key
containsValue(Object value)	Returns true if the map contains the specified value
get(Object key)	Returns the value which corresponds with the specified key
isEmpty()	Returns true if the Map is empty
put(Object key, Object value)	Puts the specified value into the map based on the specified key. If a previous object was in the map with the same key, that object is removed and returned
remove(Object key)	Remove the mapping which corresponds with the specified key
size()	Returns the number of mappings within the Map
values()	Returns a collection which contains the values in the Map.

Classes that implement Map



Exercise

- Create a program that adds the strings from the command line as (key, value) pairs to a HashMap.

- E.g.,

```
> java MyProgram a 5 b 2 c 10 d 6
```

  adds the pairs (a,5), (b,2), (c,10), and (d,6) to the HashMap.

How would you choose which class to use?

Example: Your assignment

- When reading in words from a file, what structure should you use to store those words?

- What about for your stop list?

Data Structures

- CPSC 331 describes in depth
- Beyond the scope of this course
- For your assignments and the exam, you are expected to know only how to use the data structures

Generics

- A topic far to advanced to be covered in depth
  - Introduced in Java 1.5

- Today (without generics):

```
ArrayList strs = new ArrayList();
```

- Before Today (with generics):

```
ArrayList<String> strs = new ArrayList<String>();
```

Generics

- These two statements are equivalent:

```
ArrayList strs = new ArrayList();
ArrayList<Object> strs = new ArrayList<Object>();
```

```
strs.add(new String("I CAN HAS STRIN?"));
String s = strs.get(0);
```


its..... so close...
ICANHASCHEEZBURGER.COM

Generics

```
strs.add(new String("I CAN HAS STRIN?"));
String s = (String)strs.get(0);
```

- Using generics removes the need to cast

Generics

```
for (String s : strs)
{
    // ...
}
```

- Using generics allows this kind of for loop

Collections Algorithms

- Look in Java API for the Collections class

Collections Algorithms

Method	Description
Collections.copy(List a, List b)	copy all of the elements from one list to another
Collections.frequency(Collection a, Object b)	returns the count of elements in the collection equal to b
Collections.max(Collection a)	returns the maximum element within the collection based on Natural Order
Collections.max(Collection a, Comparator b)	returns the maximum element within the collection as computed by the specified comparator
Collections.min(Collection a)	returns the minumum element within the collection based on Natural Order
Collections.min(Collection a, Comparator b)	returns the minimum element within the collection as computed by the specified comparator
Collections.replaceAll(List a, Object oldVal, Object newVal)	replaces all instances of oldVal in the list a with newVal
Collections.reverse(List a)	reverses the order of the elements in list a
Collections.shuffle(List a)	randomly order the elements in the list
Collections.sort(List a)	Sort the objects in list a based on Natural Order
Collections.sort(List a, Comparator b)	Sort the objects in list a based on the order computed by comparator b

Comparable & Comparator

- Allows the collection framework to implement some algorithms
- E.g., sort, max, min, frequency

Comparable

```
public interface Comparable<T>
{
    public int compareTo(T object);
}
```

Comparable

Return Value	Meaning
<0 (negative integer)	This object is *less than* the specified object (parameter)
0	This object is equal to the specified object (parameter)
>0 (positive integer)	This object is *greater than* the specified object (parameter)

Example: Cards

```
public class Card implements Comparable<Card>
{
    private String suit;
    private short value;

    // ...

    public int compareTo(Card card)
    {
        return this.value - card.value;
    }
}
```

Example: HonestPlayer

```
public class HonestPlayer extends Player
{
    public void passHighestCard(Player player)
    {
        Card maxCard = null;
        for (Card card : getCards())
        {
            if (maxCard == null)
            {
                maxCard = card;
            }
            else if (card.getValue() > maxCard.getValue())
            {
                maxCard = card;
            }
        }

        removeCard(maxCard);
        player.addCard(maxCard);
    }
}
```

Example: HonestPlayer

```
public class HonestPlayer extends Player
{
    public void passHighestCard(Player player)
    {
        Card maxCard = Collections.max(getCards());

        removeCard(maxCard);
        player.addCard(maxCard);
    }
}
```

Comparator

```
public interface Comparator<T>
{
    public int compare(T o1, T o2);
    boolean equals(Object obj);
}
```

Comparator

Return Value	Meaning
<0 (negative integer)	o1 is less than o2
0	o1 is equal to o2
>0 (positive integer)	o1 is greater than o2

- equals must return true if two comparators impose the same ordering
  - typically implemented as: `this.equals(obj);`

Example: TrumpComparator

```
public class TrumpComparator
   implements Comparator<Card>
{
  private String trump;

  public TrumpComparator(
                    String trump)
  {
    this.trump = trump;
  }

  public boolean equals(
                    Object obj)
  {
    return this.equals(obj);
  }
```

```
public int compare(Card c1, Card c2)
{
  if (c1.getSuit().equals(c2.getSuit())
    return c1.getValue()-c2.getValue();
  else if (trump.equals(c1.getSuit()))
    return 1;
  else if (trump.equals(c2.getSuit()))
    return -1;
  return 0;
}
```

Example: HonestPlayer

```
public class HonestPlayer extends Player
{
    public void passHighestCard(Player player)
    {
        Card maxCard = Collections.max(getCards(),
            new TrumpComparator("Hearts", "Spades");

        removeCard(maxCard);
        player.addCard(maxCard);
    }
}
```

Exercise (for home)

- Think of a card game that you have played before
- If you don't remember the rules, look them up
- Write comparator(s) for the card game:
  - do you need more than one?

Lecture 15 Summary

- Collections Framework
  - Iterable, Collections
  - List, Set
  - Map
- Collections class
- Comparable and Comparator

Next Class

- Unit Testing