Collections

An introduction to the Java Programming Language

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhan Drohan (sdrohan@wit.ie)

Waterford Institute *of* Technology

INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Agenda

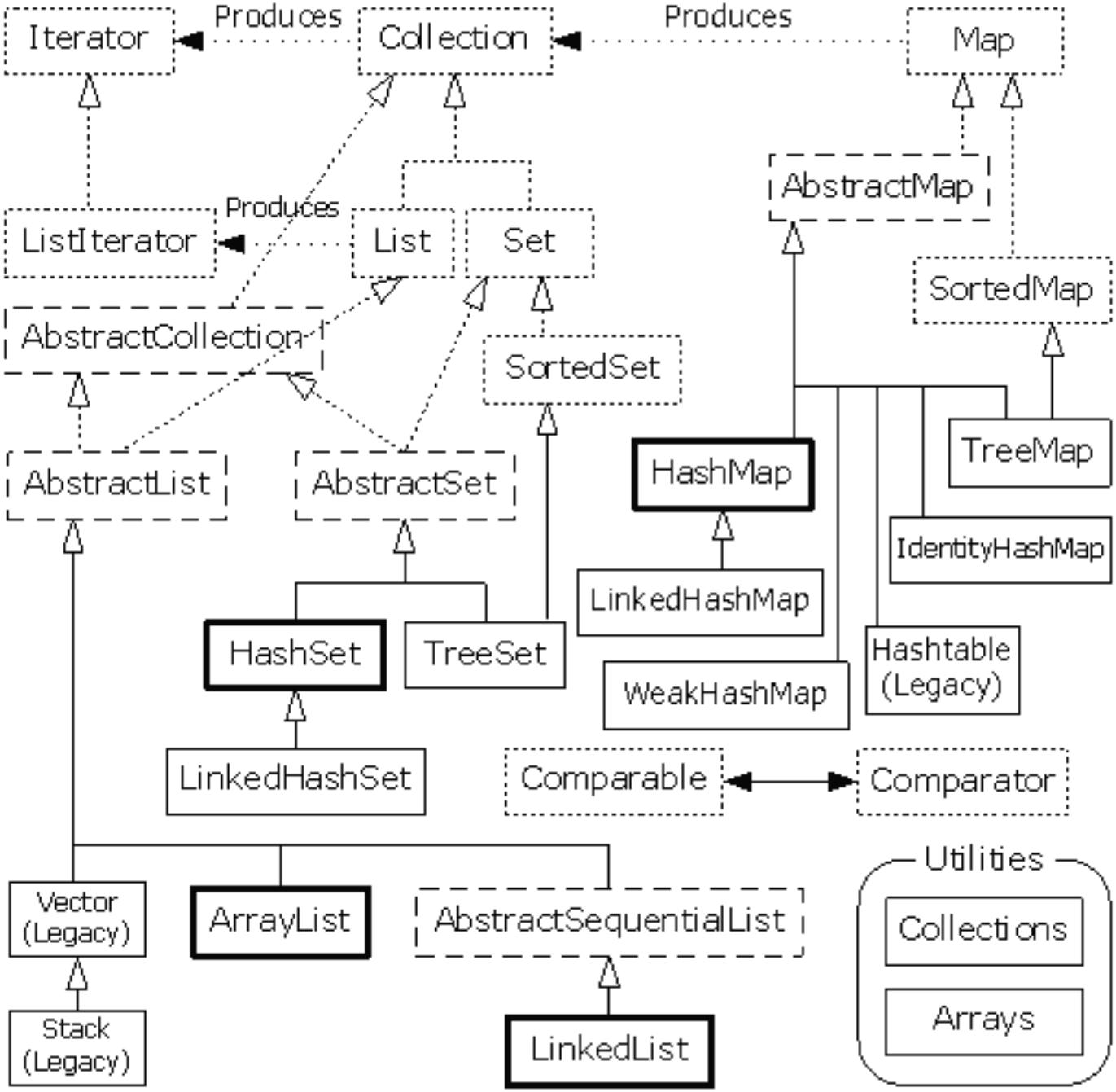
- The Collection Framework
- Interfaces
  - Collection
  - List
  - Set
  - Map
  - Iterator
- Implementations
  - ArrayList
  - HashMap
  - HashSet

What are Collections?

- Collections are Java objects that group multiple elements into a single unit.
  - Represent data items that form a natural group e.g. users, locations, activities.
- Collections store, retrieve, and manipulate other Java objects
  - Any Java object may be part of a collection, so collection can contain other collections.
- Collections do not store primitives.

Benefits:

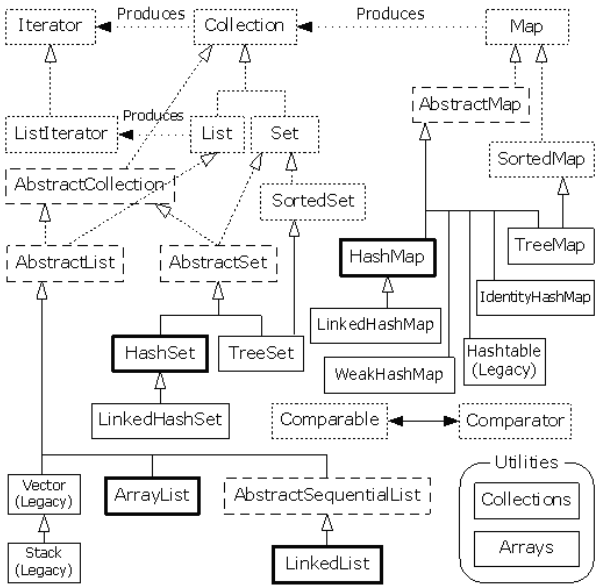
- Reusability
- Uniformity
- Faster development
- Higher quality
- Interoperability
- Less programming


Iterator
Produces
Collection
Produces
Map
AbstractMap
ListIterator
Produces
List
Set
SortedMap
AbstractCollection
SortedSet
AbstractList
AbstractSet
HashMap
TreeMap
IdentityHashMap
LinkedHashMap
HashSet
TreeSet
Hashtable
(Legacy)
WeakHashMap
LinkedHashSet
Comparable
Comparator
Utilities
Collections
Arrays
Vector
(Legacy)
ArrayList
AbstractSequentialList
Stack
(Legacy)
LinkedList

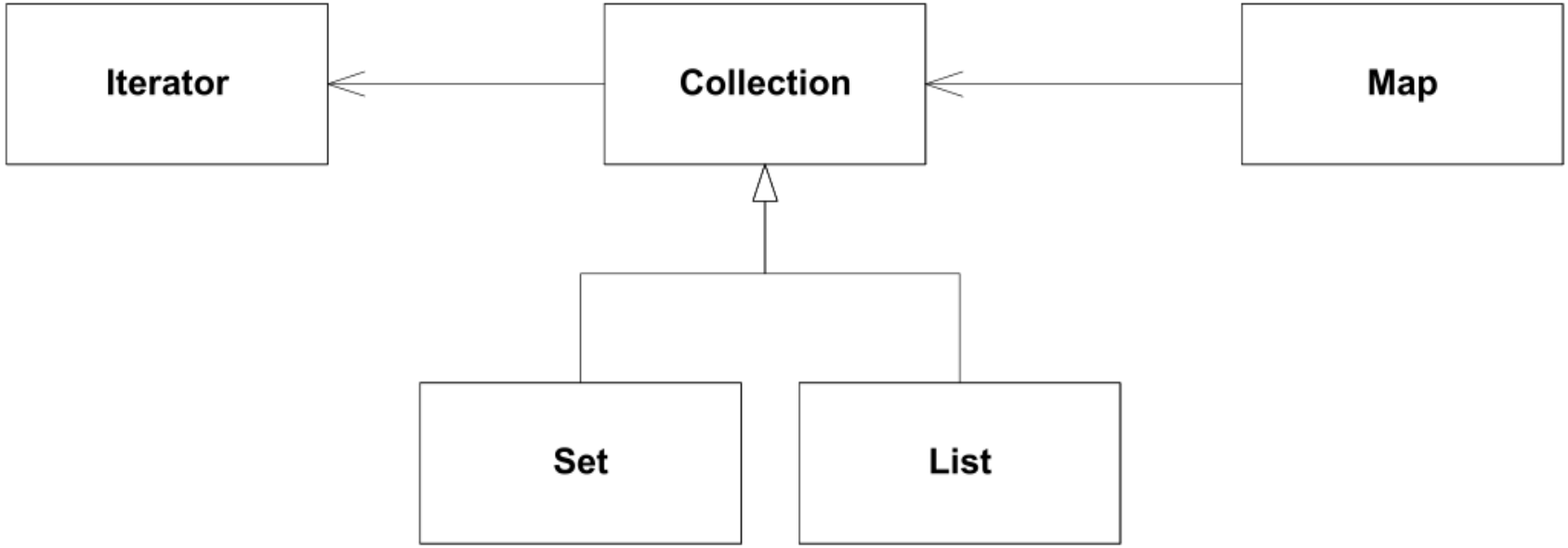
?

Collection Architecture

- **Interfaces** - abstract data types representing collections
- **Implementation** - concrete implementation of collection interfaces
- **Algorithms** - methods for manipulating collection objects (e.g. sorting, searching, shuffling, etc).



Interfaces



- Collection “uses” Iterator
- Map “uses” Collection
- Set extends Collection (subtyping)
- List extends Collection (subtyping)

Collection Interface

- `Collection` represents a group of objects
  - These collection objects are known as collection elements
- There is no direct implementation of this interface in JDK
  - Concrete implementations are provided for subtypes
- Collections in general can allow duplicate elements, and can be ordered
  - Unordered collections that allow duplicate elements should implement directly Collection interface

Adding Elements

In general two methods are defined for adding elements to the collection:

```
interface Collection
{
  //…
  /**
   * Adds element to the receiver.
   * Returns true if operation is successful, otherwise returns false.
   */
  boolean add(Object element);

  /**
   * Adds each element from collection c to the receiver.
   * Returns true if operation is successful, otherwise returns false.
   */
  boolean addAll(Collection c);
}
```

Removing Elements

Similarly to adding protocol, there are two methods are defined for removing elements from the collection:

```
interface Collection
{
  //…
  /**
   * Removes element from the receiver.
   * Returns true if operation is successful, otherwise returns false.
   */
  boolean remove(Object element);

  /**
   * Removes each element contained in collection c from the receiver.
   * Returns true if operation is successful, otherwise returns false.
   */
  boolean removeAll(Collection c);
}
```

Other Collection Methods

Includes methods for:

- Checking how many elements are in the collection
- Checking if an element is in the collection
- Iterating through collection

```
boolean contains(Object element);
boolean containsAll(Collection c);
int size();
boolean isEmpty();
void clear();
boolean retainAll(Collection c);
Iterator iterator;
```

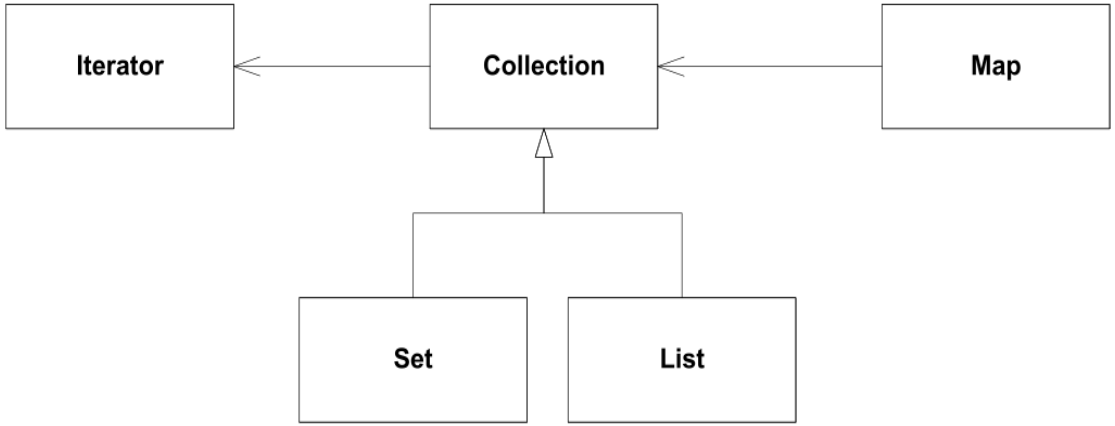
Iterator Interface

- Defines a protocol for iterating through a collection:

```
public interface Iterator
{
  /**
   * Returns whether or not the underlying collection has next
   * element for iterating.
   */
  boolean hasNext();

  /**
   * Returns next element from the underlying collection.
   */
  Object next();

  /**
   * Removes from the underlying collection the last element returned by next.
   */
  void remove();
}
```

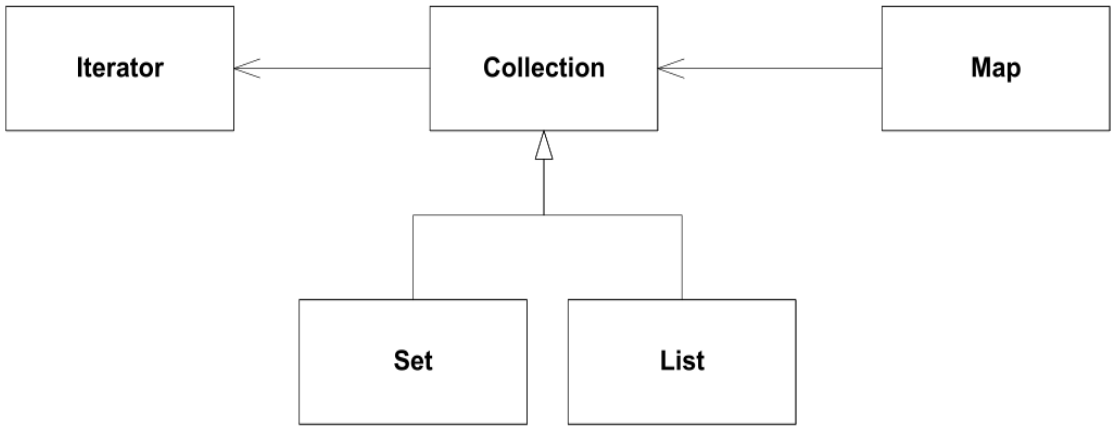
Set Interface

- Set is a collection that does not contain duplicate elements
  - This is supported by additional behavior in constructors and add(), hashCode(), and equals() methods
  - All constructors in a set must create a set that does not contain duplicate elements
- It is not permitted for a set to contain itself as an element
- If set element changes, and that affects equals comparisons, the behavior of a set is not specified



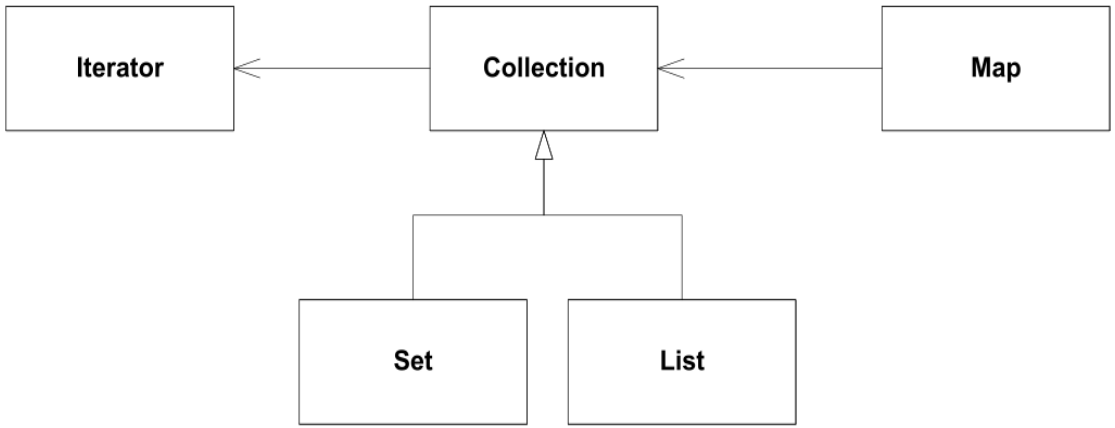
List Interface

- List represents an ordered collection
  - Also known as sequence
- Lists may contain duplicate elements
- Lists extend behavior of collections with operations for:
  - Positional Access
  - Search
  - List Iteration
  - Range-view

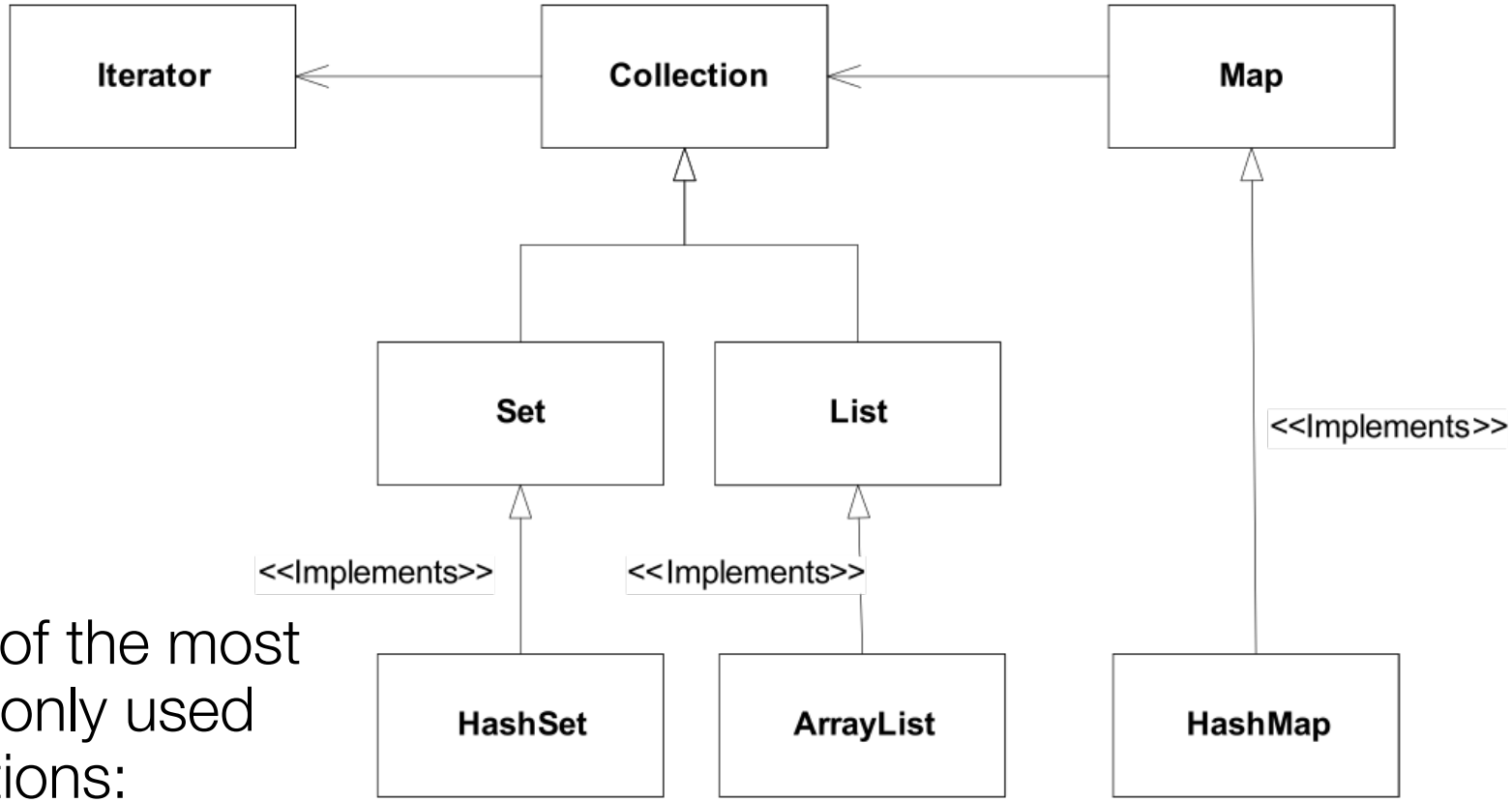


Map Interface

- Map is an object that maps keys to values
  - Keys must be unique, i.e. map cannot contain duplicate keys
  - Each key in the map can map to most one value, i.e. one key cannot have multiple values
- Map interface defines protocols for manipulating keys and values



Most Commonly Used Collections



- Three of the most commonly used collections:
  - HashSet
  - ArrayList
  - HashMap

ArrayList

- Represents resizable-array implementation of the List interface
  - Permits all elements including null
- It is generally the best performing List interface implementation
- Instances of this class have a capacity
  - It is size of the array used to store the elements in the list, and it's always at least as large as the list size
  - It grows as elements are added to the list

ArrayList Examples

```java
//declare list
ArrayList list = new ArrayList();

//add elements to the list
list.add("First element");
list.add("Second element");

//get the list size
int listSize = list.size();

//print the list size and the first element
System.out.println(listSize);
System.out.println(list.get(0));

//add first element in the list
list.add(0,"Added element");

//get the list iterator
Iterator iterator = list.iterator();
while (iterator.hasNext())
{
   String element = (String)iterator.next();
   System.out.println(element);
}
```

Console

```
2
First element
```

Console

```
Added element
First element
Second element
```

HashMap

- Collection that contains pair of objects
  - Values are stored at keys
- It is a hash table based implementation of the Map interface
  - Permits null values and null keys
  - The order of the map is not guaranteed
- Two parameters affect performance of a hash map:
  - Initial capacity, indicates capacity at the map creation time
  - Load factor, indicates how full the map should be before increasing its size
    - 0.75 is the default

HashMap Example

```
//create a number dictionary
HashMap numberDictionary = new HashMap();
numberDictionary.put("1", "One");
numberDictionary.put("2", "Two");
numberDictionary.put("3", "Three");
numberDictionary.put("4", "Four");
numberDictionary.put("5", "Five");

//get an iterator of all the keys
Iterator keys = numberDictionary.keySet().iterator();
while (keys.hasNext())
{
  String key = (String)keys.next();
  String value = (String)numberDictionary.get(key);
  System.out.println("Number: " + key + ", word: " + value);
}
```

Console

```
Number: 5, word: Five
Number: 4, word: Four
Number: 3, word: Three
Number: 2, word: Two
Number: 1, word: One
```

HashSet

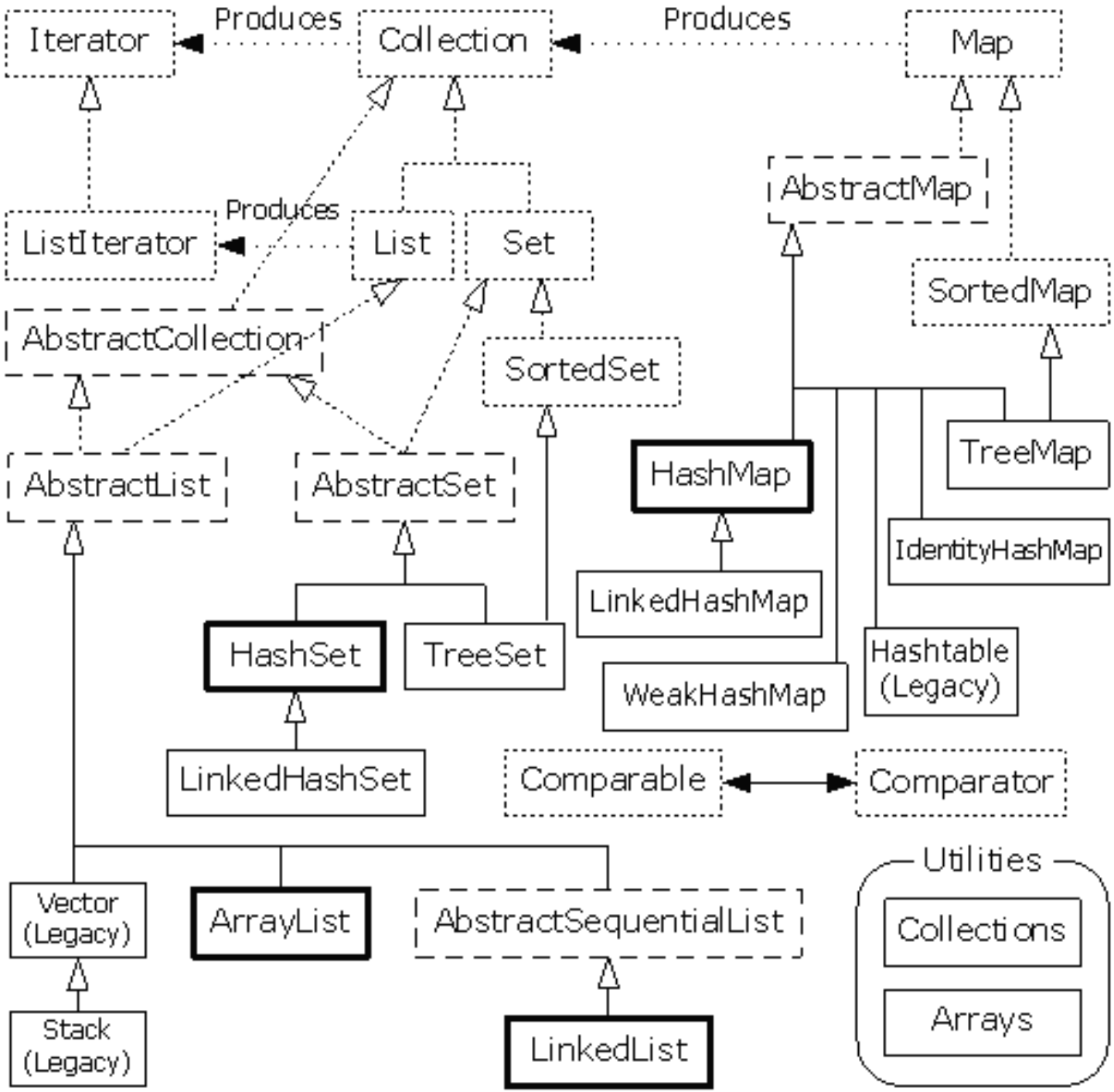
- Concrete implementation of the Set interface
  - Backed up by an instance of HashMap
  - Order is not guaranteed
- Performance of the set is affected by size of the set and capacity of the map
  - It is important not to set the initial capacity too high, or the load factor too low if performance of iteration is important
- Elements in the set cannot be duplicated

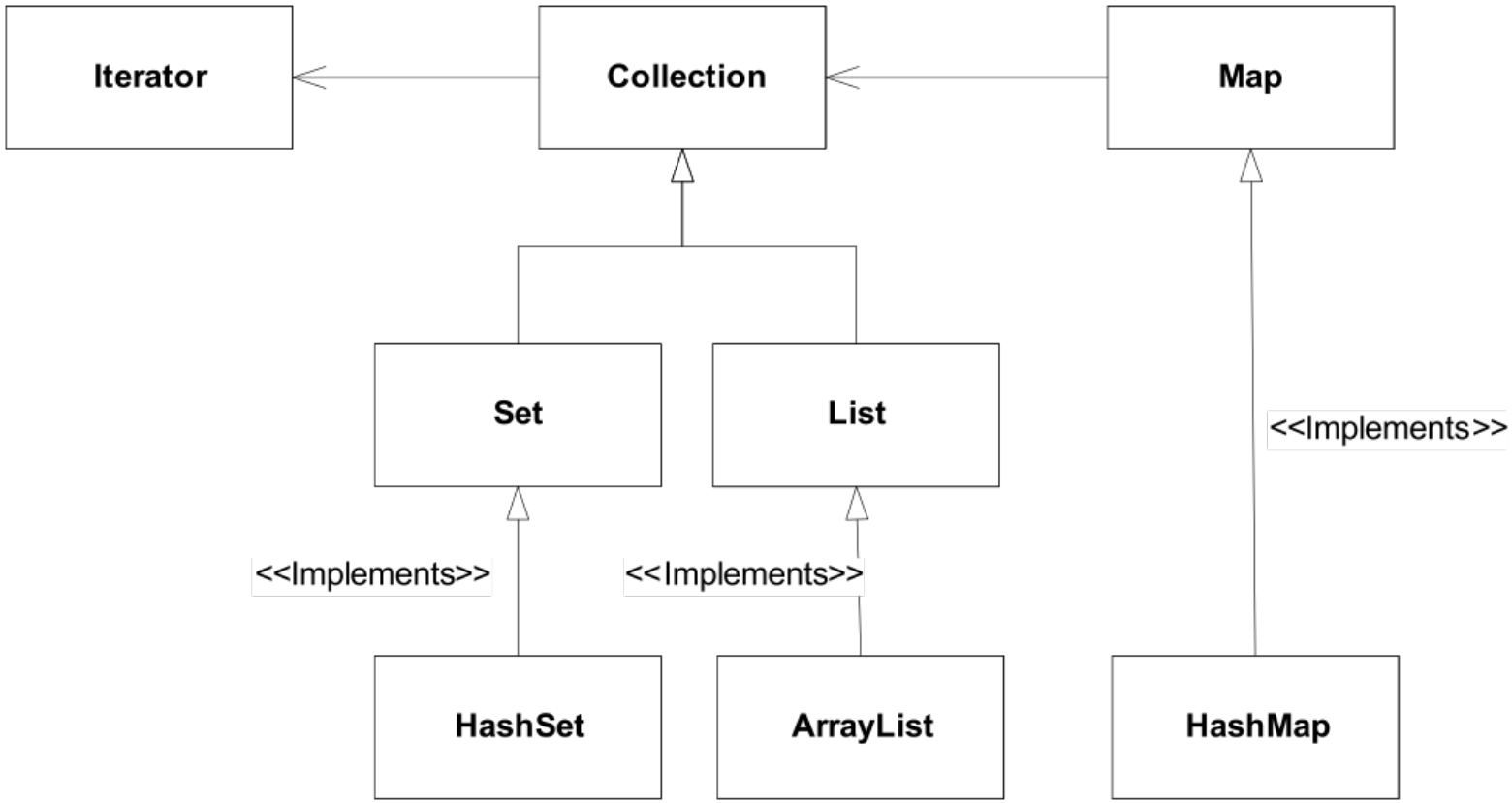
HashSet Example

```java
//create new set
HashSet set = new HashSet();

//add elements to the set
set.add("One");
set.add("Two");
set.add("Three");

//elements cannot be duplicated in the set
set.add("One");

//print the set
System.out.println(set);
```

Console

```
[One, Three, Two]
```


Iterator
Produces
Collection
Produces
Map
AbstractMap
ListIterator
Produces
List
Set
SortedMap
AbstractCollection
SortedSet
AbstractList
AbstractSet
HashMap
TreeMap
IdentityHashMap
LinkedHashMap
HashSet
TreeSet
WeakHashMap
Hashtable
(Legacy)
LinkedHashSet
Comparable
Comparator
Utilities
Collections
Arrays
Vector
(Legacy)
ArrayList
AbstractSequentialList
Stack
(Legacy)
LinkedList


Iterator
Collection
Map
Set
List
<<Implements>>
<<Implements>>
<<Implements>>
HashSet
ArrayList
HashMap