Generic Programming & Java Collection Framework

Pertemuan 11

Topics

1. Generic Programming
2. Java Collection Framework
   a) List
   b) Set
   c) Map

Generic Programming

- Generic programming artinya menuliskan kode yang dapat diguna-ulang (reuse) pada suatu objek dengan tipe yang berbeda.
- Sebelum adanya *generic programming* tipe yang digunakan adalah objek, namun tipe objek memiliki kendala terkait dengan *type casting*.

Enumeration

- Enums are a datatype that contains a fixed set of constants
- Enum are good to use when you already know all possibilities of the values
- Can be declared on their own or within another class

```
package generic;

public class EnumClass {
    enum Level {
        LOW, MEDIUM, HIGH
    };

    public static void main(String[] args) {
        Level myVar = Level.MEDIUM;
        System.out.println(myVar);

        switch (myVar) {
        case LOW:
            System.out.println("Low Level");
            break;
        case MEDIUM:
            System.out.println("Medium Level");
            break;
        case HIGH:
            System.out.println("High Level");
            break;
        }

        for (Level myLevel : Level.values()){
            System.out.println(myLevel);
        }
    }

}
```


Generic class

- A generic class is a special type of class that associates one or more non-specified Java types upon instantiation
- This removes the risk of the runtime exception “ClassCastException” when casting between different types
- Generic types are declared by using angled brackets <> around a holder return type. E.g. <T>

Contoh : Generic Class

```java
package generic;

public class Cell<T> {
    private T data;

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }


}
```

Class generic

```java
package generic;

public class CellDriver {

    public static void main(String[] args) {
        Cell<Integer> integerCell = new Cell<Integer>();
        Cell<String> stringCell = new Cell<String>();

        integerCell.setData(1);
        stringCell.setData("One");

        int num = integerCell.getData();
        String str = stringCell.getData();

        System.out.println("Integer value : " + num);
        System.out.println("String value : " + str);

    }
}
```

Class Main

Contoh : Generic Method

```
public class GenericMethodClass {

   public <T> void printArray(T[] array){
      for( T arrayitem : array ){
         System.out.println( arrayitem );
      }//endfor
   }//end method printArray

}//end class GenericMethodClass
```

Generic Method

```
public class GenericMethodDriver {
   public void main(String[] args) {
      GenericMethodClass gmc = new GenericMethodClass();

      Integer[] integerArray = {1, 2, 3};
      String[] stringArray = {"This","is","fun"};

      gmc.printArray(integerArray);
      gmc.printArray(stringArray);
   }//end method main
}//end class GenericMethodDriver
```

Class Main

```
1
2
3
This
is
fun
```

Output

Types Parameter

- The most commonly used types parameter names are :
  - E – Element (used extensively by JCF)
  - K – Key
  - N – Number
  - T – Type
  - V – Value
  - S,U,V etc. – 2nd, 3rd, 4th types

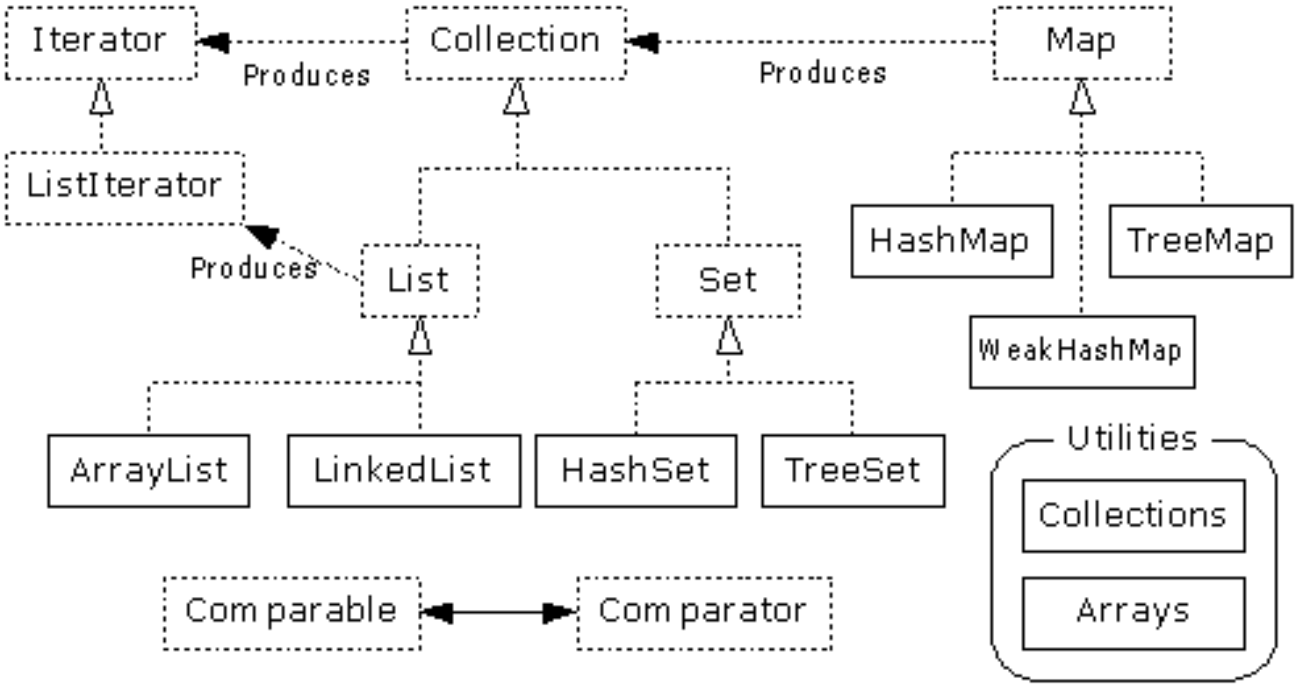
Java Collections

- Collection merupakan suatu objek yang mengelompokkan berbagai elemen kedalam satu unit
- Kegunaan :
  - Menyimpan, membaca dan memanipulasi data
  - Mengirimkan data dari satu method ke method lain
  - Struktur data

Collection Framework

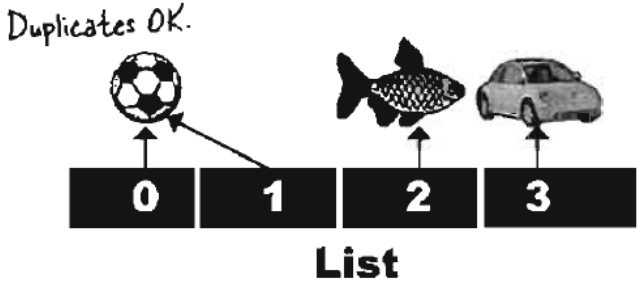
- Collection Framework merupakan suatu arsitektur untuk merepresentasikan dan mengelola suatu collection.
- Terdiri atas 3 (tiga) hal, yaitu :
  - Interfaces
  - Implementations
  - Algorithms

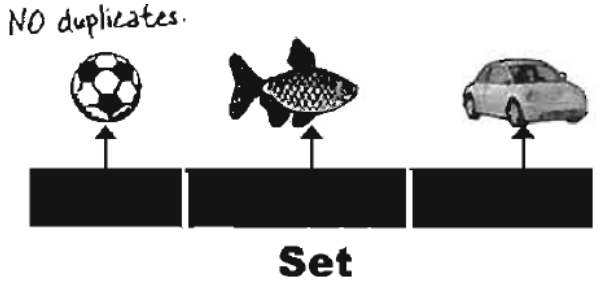
Collection Framework Diagram

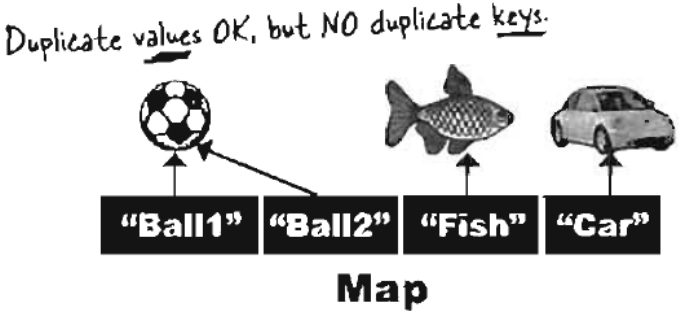


Beberapa tipe collection framework

- List :
  - Ketika urutan (sequence) menjadi pertimbangan
  - Ingin mengetahui posisi berdasarkan index
  - Boleh duplikasi value
- Set
  - Ketika keunikan (uniqueness) menjadi pertimbangan
  - Tidak boleh duplikasi value
- Map
  - Ketika mencari berdasarkan kunci menjadi pertimbangan
  - Boleh duplikasi value, namun tidak boleh duplikasi key







Collection Interface

- Defines fundamental methods
  - `int size();`
  - `boolean isEmpty();`
  - `boolean contains(Object element);`
  - `boolean add(Object element);    // Optional`
  - `boolean remove(Object element); // Optional`
  - `Iterator iterator();`
- These methods are enough to define the basic behavior of a collection
- Provides an Iterator to step through the elements in the Collection

Iterator Interface

- Defines three fundamental methods
  - `Object next()`
  - `boolean hasNext()`
  - `void remove()`
- These three methods provide access to the contents of the collection
- An Iterator knows position within collection
- Each call to next() “reads” an element from the collection
  - Then you can use it or remove it

Iterator Position

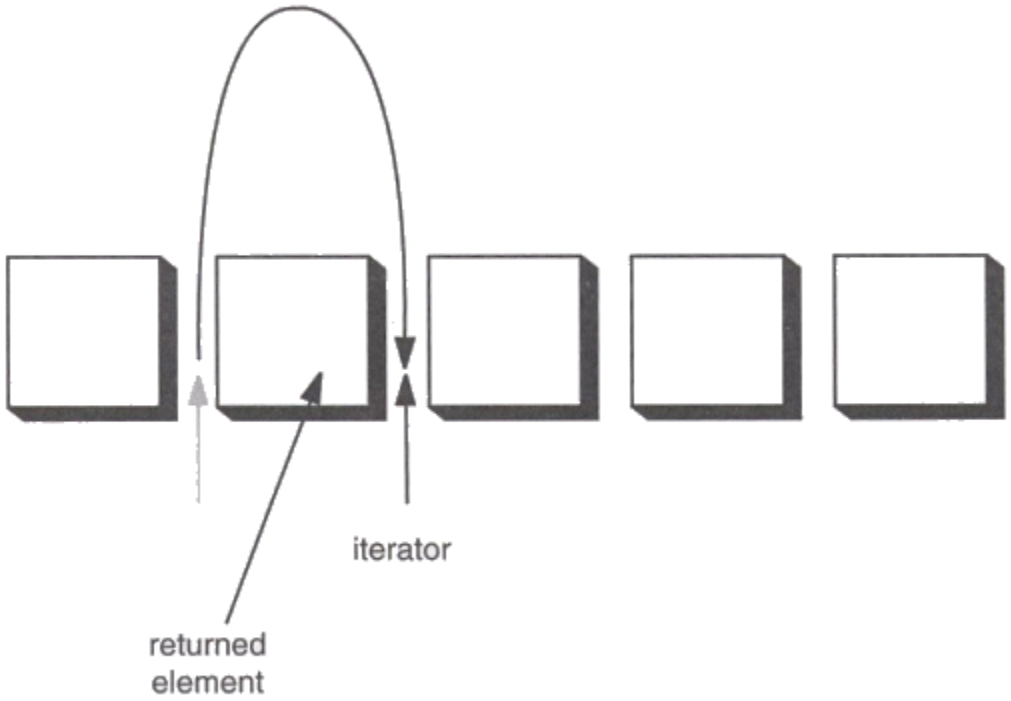
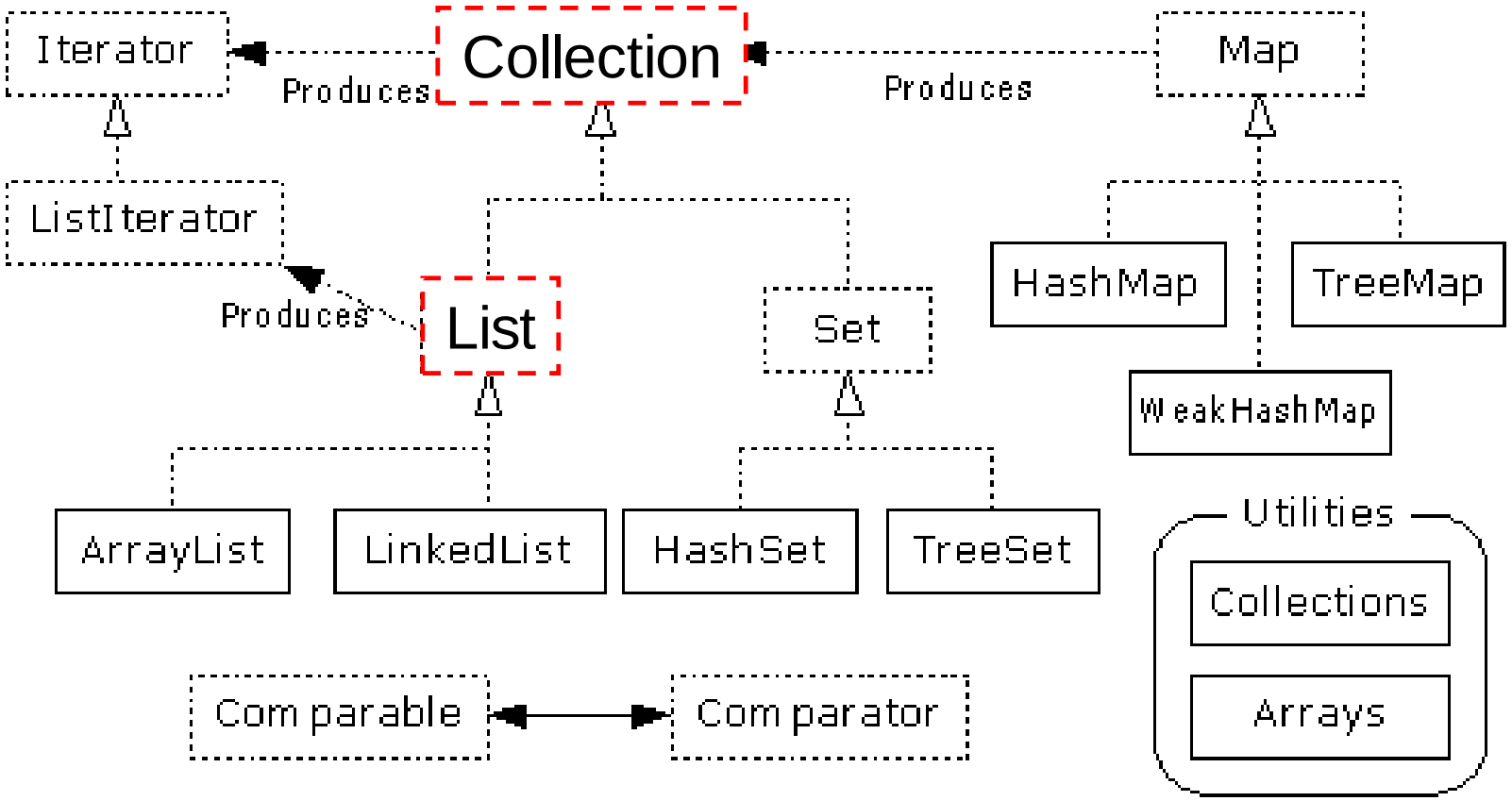


Figure 2–3: Advancing an iterator

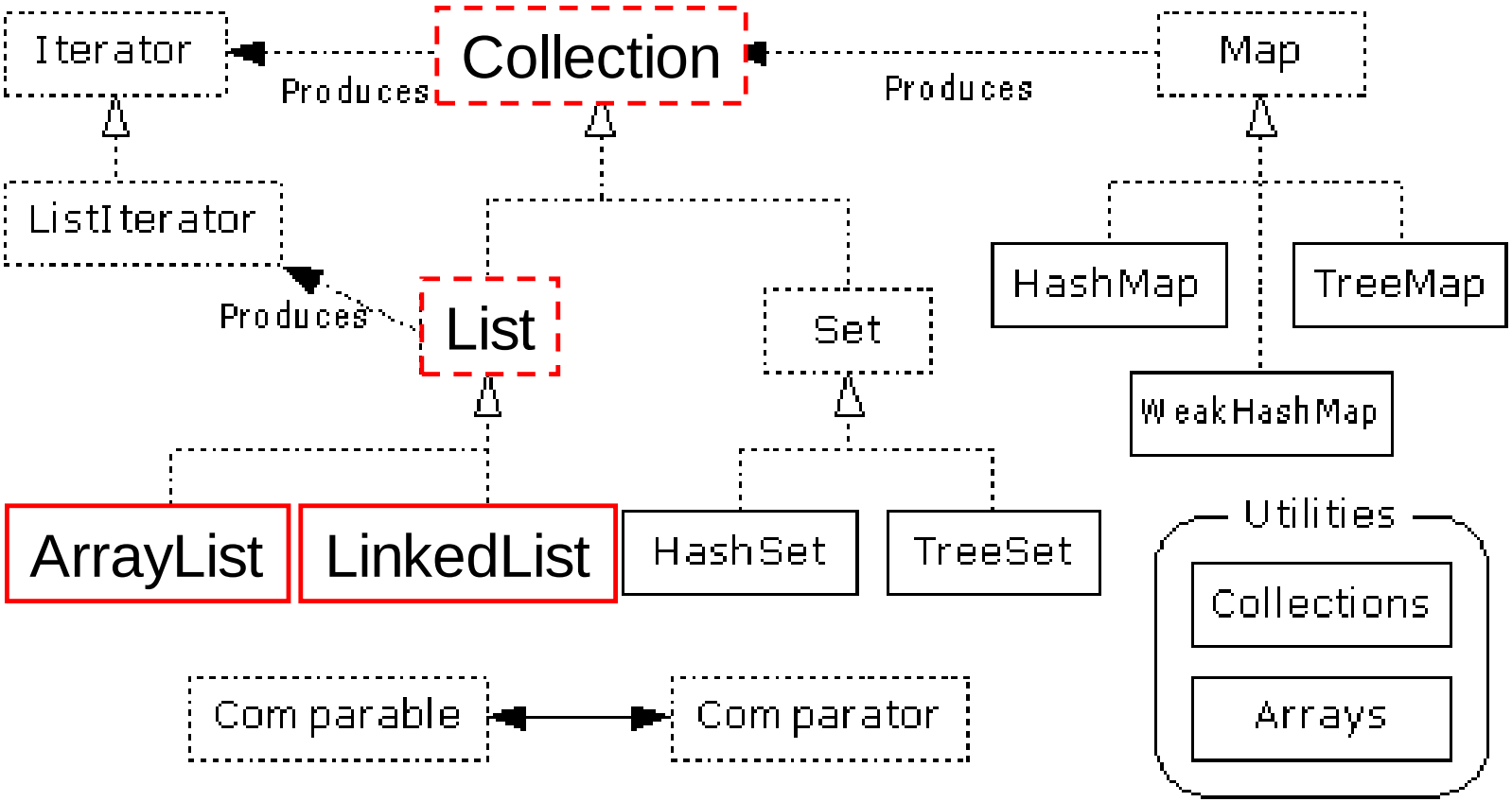
Example - SimpleCollection

```
public class SimpleCollection  {
  public static void main(String[] args) {
    Collection c;
    c = new ArrayList();
    System.out.println(c.getClass().getName());
    for (int i=1; i <= 10; i++) {
           c.add(i + " * " + i + " = "+i*i);
    }
    Iterator iter = c.iterator();
    while (iter.hasNext())
           System.out.println(iter.next());
  }
}
```


a. List


Iterator
Collection
Map
Produces
Produces
ListIterator
Produces
List
Set
HashMap
TreeMap
WeakHashMap
ArrayList
LinkedList
HashSet
TreeSet
Utilities
Collections
Arrays
Comparable
Comparator

ArrayList and LinkedList Context



List Implementations

- ArrayList
  - low cost random access
  - high cost insert and delete
  - array that resizes if need be
- LinkedList
  - sequential access
  - low cost insert and delete
  - high cost random access

ArrayList overview

- Constant time positional access (it's an array)
- One tuning parameter, the initial capacity

```
public ArrayList(int initialCapacity) {
   super();
   if (initialCapacity < 0)
    throw new IllegalArgumentException(
       "Illegal Capacity: "+initialCapacity);
   this.elementData = new Object[initialCapacity];
}
```


ArrayList methods

- The indexed get and set methods of the List interface are appropriate to use since ArrayLists are backed by an array
  - `Object get(int index)`
  - `Object set(int index, Object element)`
- Indexed add and remove are provided, but can be costly if used frequently
  - `void add(int index, Object element)`
  - `Object remove(int index)`
- May want to resize in one shot if adding many elements
  - `void ensureCapacity(int minCapacity)`

LinkedList overview

- Stores each element in a node
- Each node stores a link to the next and previous nodes
- Insertion and removal are inexpensive
  - just update the links in the surrounding nodes
- Linear traversal is inexpensive
- Random access is expensive
  - Start from beginning or end and traverse each node while counting

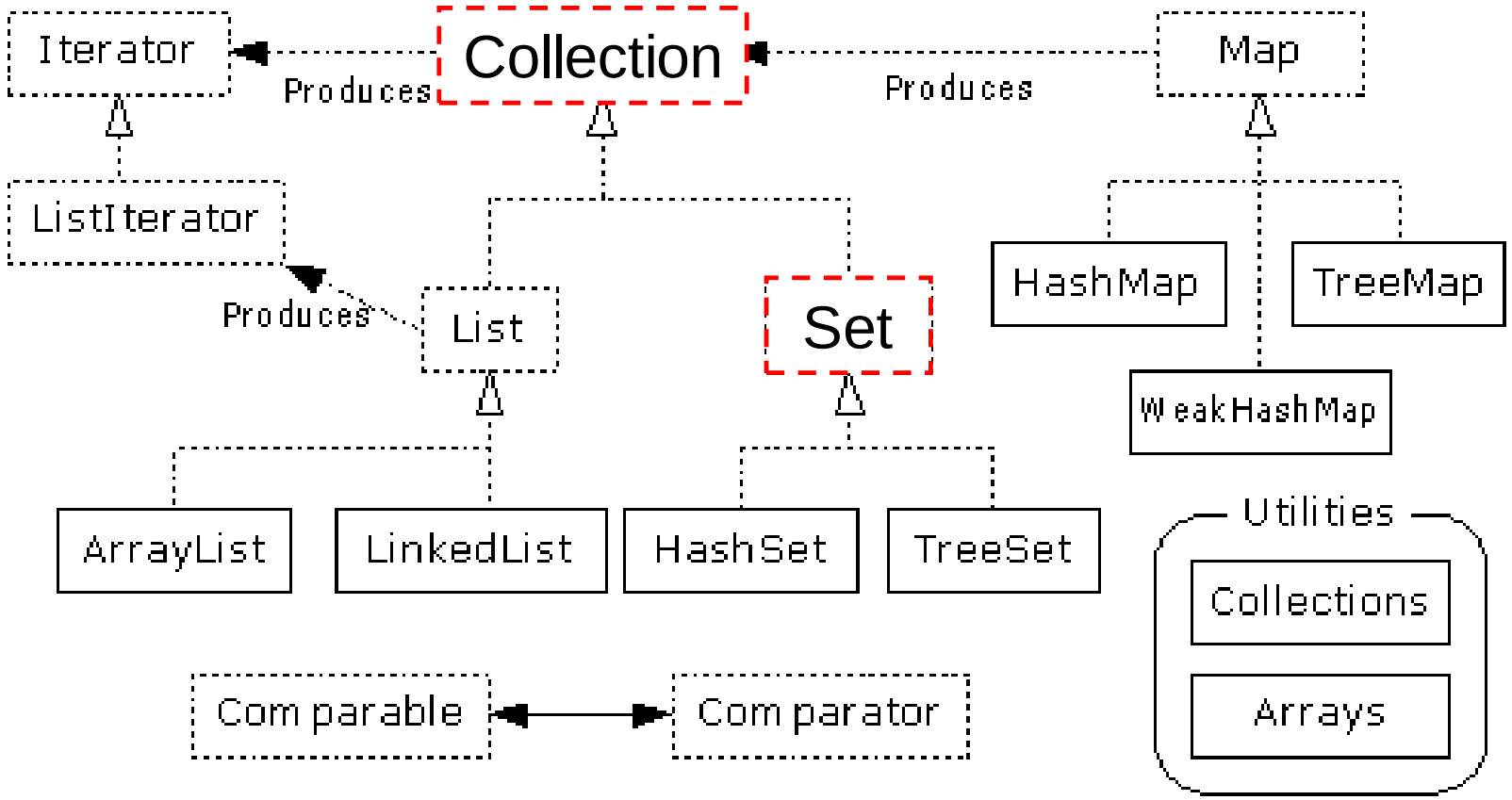
LinkedList entries

```
private static class Entry {
   Object element;
   Entry next;
   Entry previous;

   Entry(Object element, Entry next, Entry previous) {
        this.element = element;
        this.next = next;
        this.previous = previous;
   }
}

private Entry header = new Entry(null, null, null);

public LinkedList() {
   header.next = header.previous = header;
}
```


LinkedList methods

- The list is sequential, so access it that way
  - `ListIterator listIterator()`
- ListIterator knows about position
  - use `add()` from ListIterator to add at a position
  - use `remove()` from ListIterator to remove at a position
- LinkedList knows a few things too
  - `void addFirst(Object o), void addLast(Object o)`
  - `Object getFirst(), Object getLast()`
  - `Object removeFirst(), Object removeLast()`

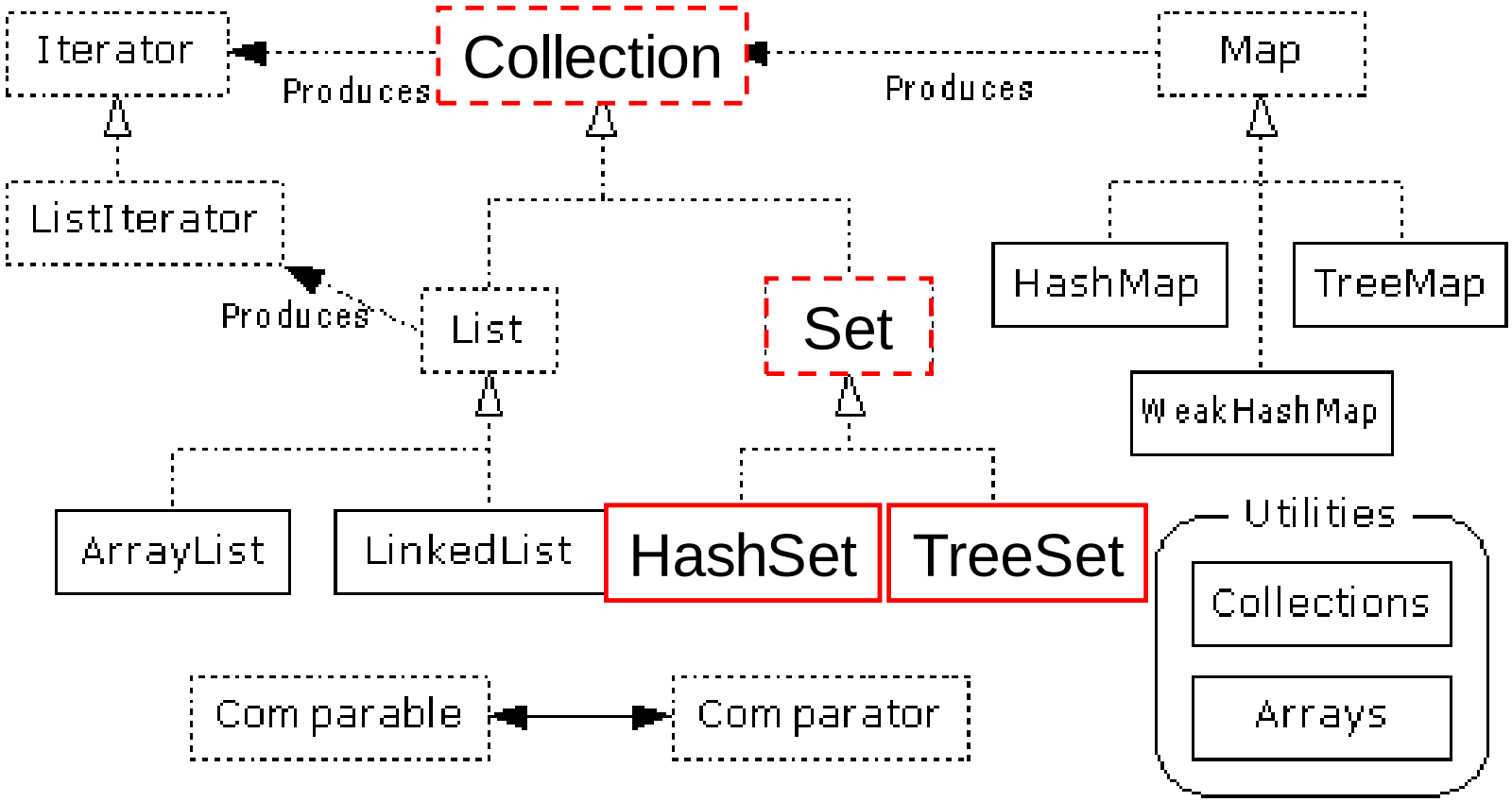
b. Set


Iterator
Collection
Map
Produces
Produces
ListIterator
Produces
List
Set
HashMap
TreeMap
WeakHashMap
ArrayList
LinkedList
HashSet
TreeSet
Utilities
Collections
Arrays
Comparable
Comparator

Set Interface

- Same methods as Collection
  - different contract - no duplicate entries
- Defines two fundamental methods
  - `boolean add(Object o)` - reject duplicates
  - `Iterator iterator()`
- Provides an Iterator to step through the elements in the Set
  - No guaranteed order in the basic Set interface
  - There is a SortedSet interface that extends Set

HashSet and TreeSet Context



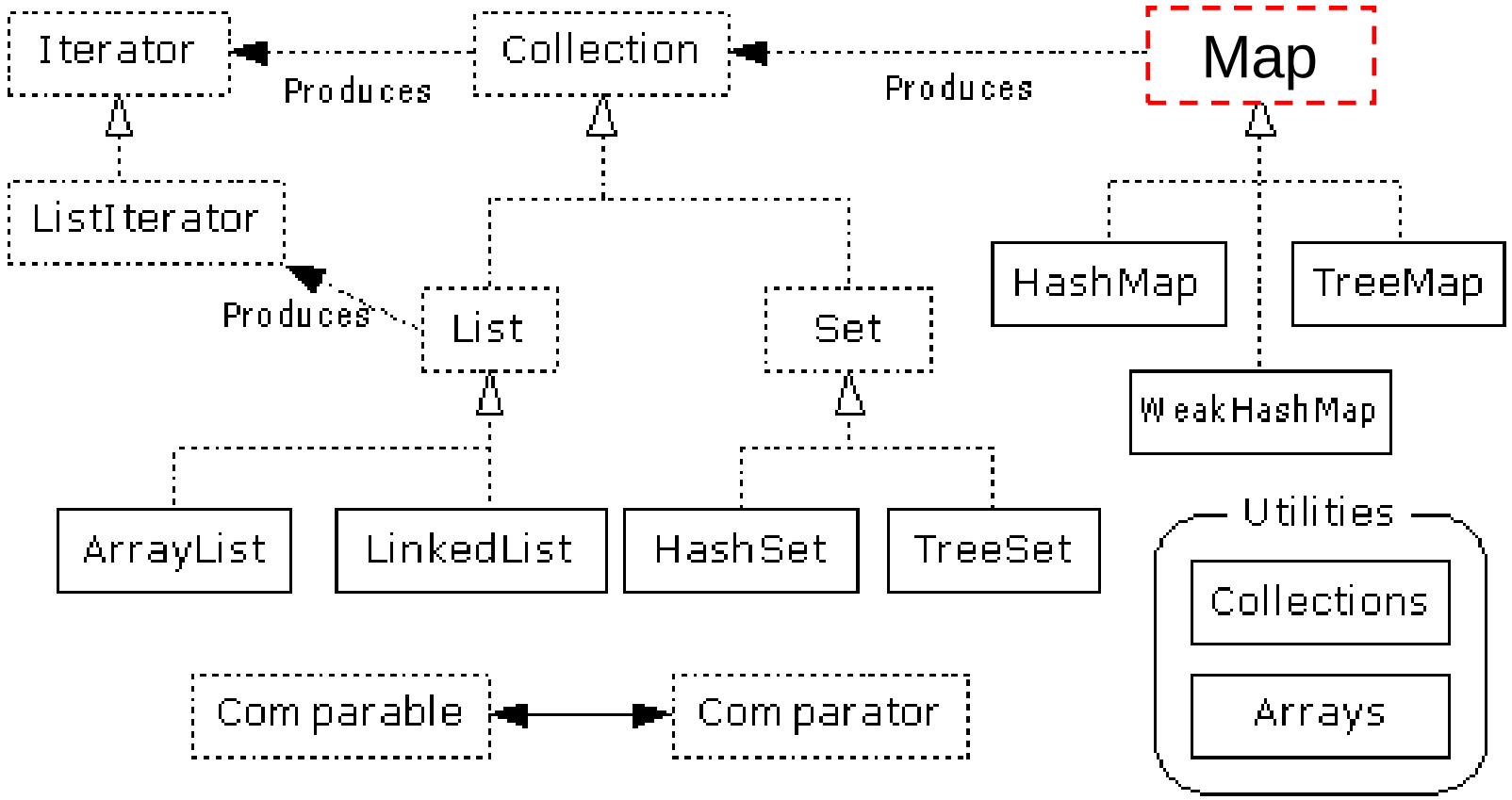
HashSet

- Prevent duplicates in the collection
- Can find and add that elements in the collections very quickly
- Hashing uses an array of linked lists
  - The `hashCode()` is used to index into the array
  - Then `equals()` is used to determine if element is in the (short) list of elements at that index
- The `hashCode()` method and the `equals()` method must be compatible
  - if two objects are equal, they must have the same `hashCode()` value

TreeSet

- Prevent duplicates in the collections
- Keep the element sorted
- Elements can be inserted in any order
- The TreeSet stores them in order
  - Red-Black Trees out of Cormen-Leiserson-Rivest
- An iterator always presents them in order
- Default order is defined by natural order
  - objects implement the Comparable interface
  - TreeSet uses `compareTo(Object o)` to sort
- Can use a different Comparator
  - provide Comparator to the TreeSet constructor

c. Map


Iterator
Collection
Map
Produces
Produces
ListIterator
Produces
List
Set
HashMap
TreeMap
WeakHashMap
ArrayList
LinkedList
HashSet
TreeSet
Utilities
Collections
Arrays
Comparable
Comparator

Map Interface

- Stores key/value pairs
- Maps from the key to the value
- Keys are unique
  - a single key only appears once in the Map
  - a key can map to only one value
- Values do not have to be unique

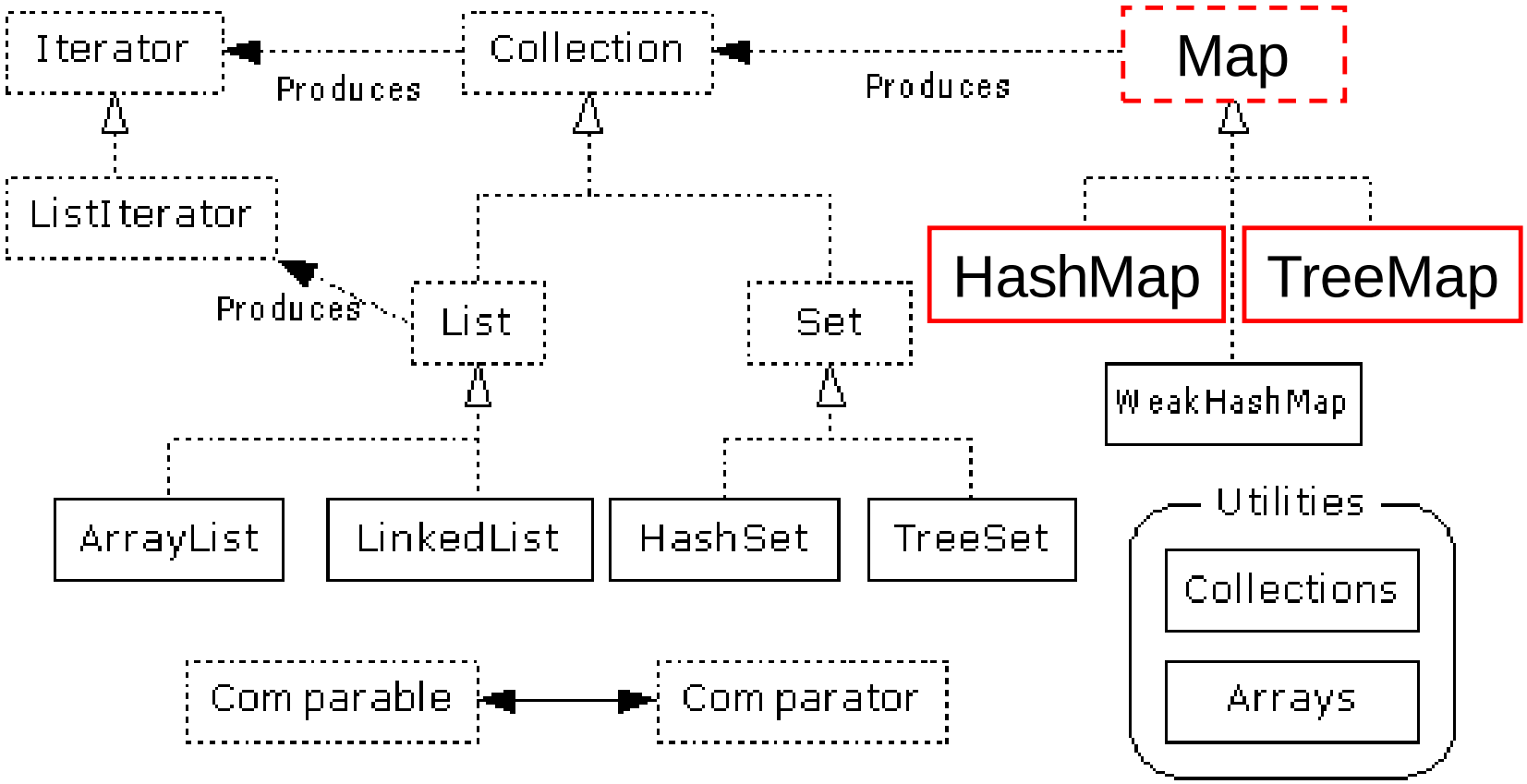
Map methods

```
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()
```


Map views

- A means of iterating over the keys and values in a Map
- `Set keySet()`
  - returns the Set of keys contained in the Map
- `Collection values()`
  - returns the Collection of values contained in the Map. This Collection is not a Set, as multiple keys can map to the same value.
- `Set entrySet()`
  - returns the Set of key-value pairs contained in the Map. The Map interface provides a small nested interface called Map.Entry that is the type of the elements in this Set.

HashMap and TreeMap Context



HashMap and TreeMap

- HashMap
  - The keys are a set - unique, unordered
  - Fast

- TreeMap
  - The keys are a set - unique, ordered
  - Same options for ordering as a TreeSet
    - *Natural order (Comparable, compareTo(Object))*
    - *Special order (Comparator, compare(Object, Object))*

Bulk Operations

- In addition to the basic operations, a Collection may provide "bulk" operations

```
boolean containsAll(Collection c);
boolean addAll(Collection c);     // Optional
boolean removeAll(Collection c); // Optional
boolean retainAll(Collection c); // Optional
void clear();                     // Optional
Object[] toArray();
Object[] toArray(Object a[]);
```


Kesimpulan

- Java Collection Framework mencakup : List, Set, Map
- List digunakan ketika urutan (sequences) menjadi pertimbangan.
- Set digunakan ketika keunikan (uniqueness) menjadi pertimbangan.
- Map digunakan ketika pencarian berdasarkan kunci (key) menjadi pertimbangan

Referensi

- Head first java 2nd Edition. Chapter 16 : Collections and Generic
- Java Collections. URL [http://www.cs.washington.edu/education/courses/403/03wi/]
- Oracle Academy, Section 3 – Data structures : Generic and Collections.