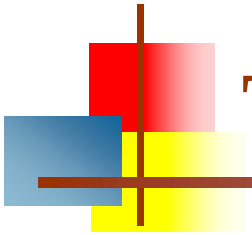
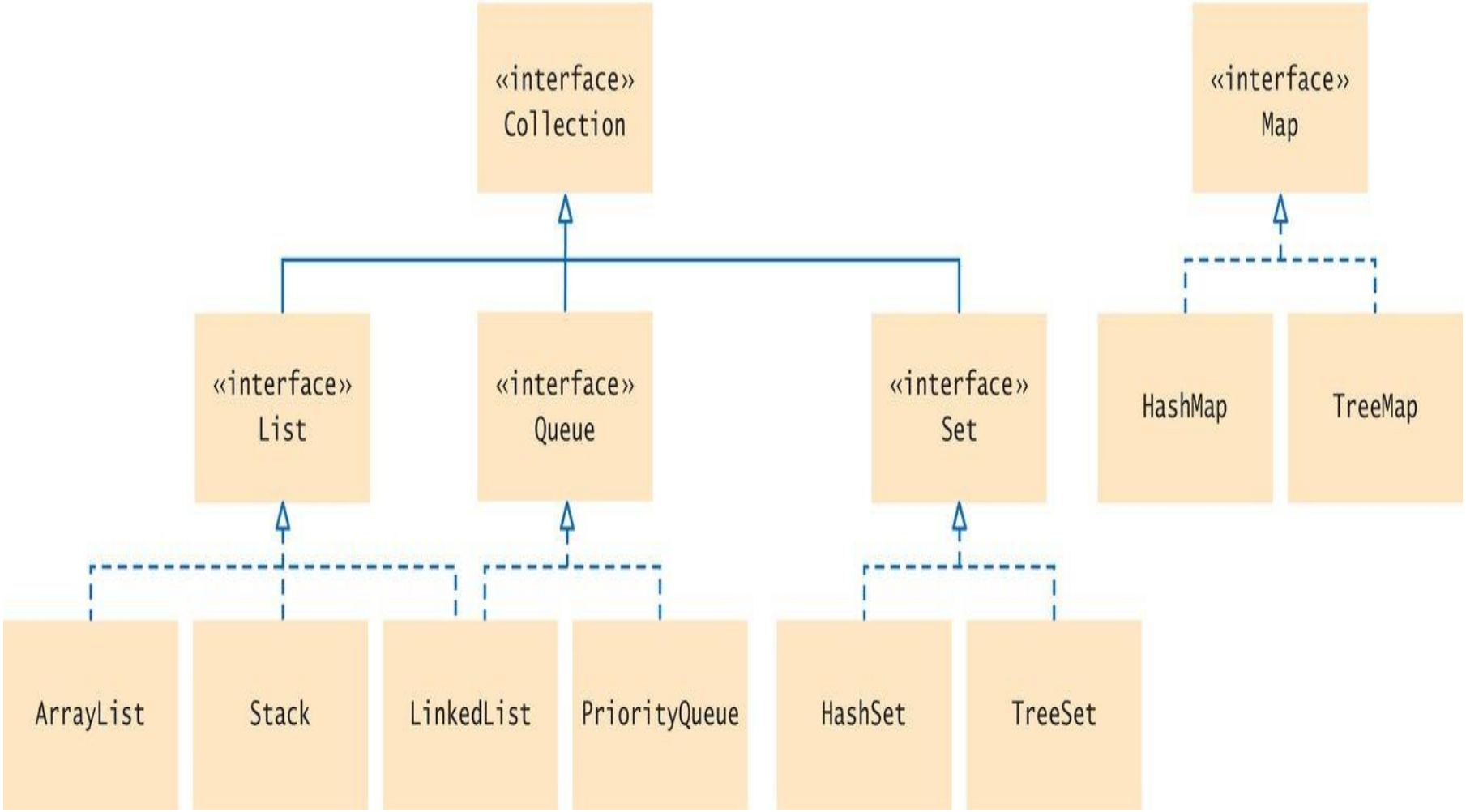
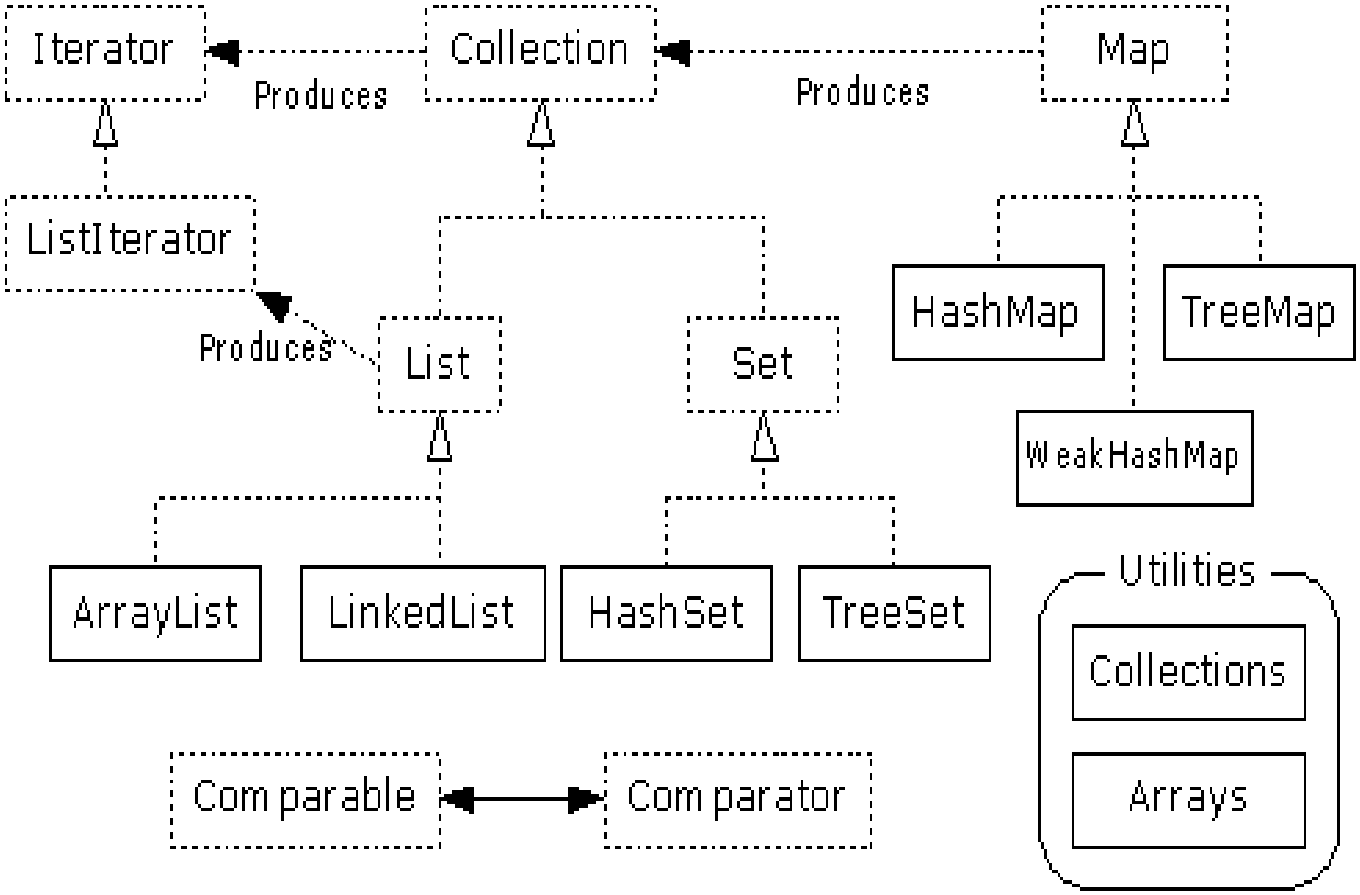


The Java Collections Framework

An Overview of the Collections Framework



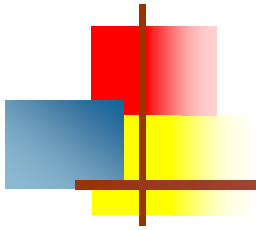
Collection Interface



Stack overview

- Remembers the order of elements
- But you can only add and remove at the top

- Removes items in the opposite order than they were added Last-in, first-out or LIFO order
- Add and remove methods are called push and pop.



Stack overview

- Stack class provides push, pop and peek methods.

Table 7 Working with Stacks	
`Stack<Integer> s = new Stack<>();`	Constructs an empty stack.

`s.push(1);`
`s.push(2);`
`s.push(3);`
	Adds to the top of the stack; s is now `[1, 2, 3]`. (Following the `toString` method of the `Stack` class, we show the top of the stack at the end.)

`int top = s.pop();`	Removes the top of the stack; `top` is set to 3 and s is now `[1, 2]`.
`head = s.peek();`	Gets the top of the stack without removing it; `head` is set to 2.

Stack overview

```
public class StackDemo {
        public static void main(String arg[])
        {
                Stack   s = new Stack ();
                s.push("A");
                s.push("B");
                s.push("C");  // s =  C, B, A
                String word = (String) s.peek();  //C
                System.out.println(word);
                while (s.size() > 0)
                {
                        System.out.print(s.pop() + " "); // Prints C B A
                }
        }
}
```

Calculator.java

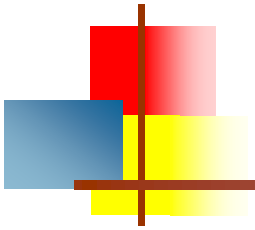

```
4           /**
5           This calculator uses the reverse Polish notation.
6           */
7           public class Calculator
8           {
9           public static void main(String[] args)
10          {
11                StaScanner in = new Scanner(System.in);
12                Stack<Integer> results = new Stack<>();
13                System.out.println("Enter one number or operator per line, Q to quit.
");
14                boolean done = false;
15                while (!done)
16                {
17                   String input = in.nextLine();
18
19
20            // If the command is an operator, pop the arguments and push the result
21
22                   if (input.equals("+"))
23                   {
24                   results.push(results.pop() + results.pop());
25                   }
26                   else if (input.equals("-"))
27                   {
28                   Integer arg2 = results.pop();
29                   results.push(results.pop() - arg2);
30                   }
31                   else if (input.equals("*") || input.equals("x"))
32                   {
33                   results.push(results.pop() * results.pop());
34                   }
35                   else if (input.equals("/"))
36                   {
                     Integer arg2 = results.pop();
```

Queue Overview

- Add items to one end (the tail) and remove them from the other end (the head)
- A queue of people

Photodisc/Punchstock.

- A priority queue
  - an unordered collection
  - has an efficient operation for removing the element with the highest priority



Queue Overview

- The Queue interface in the standard Java library has:
  - an add method to add an element to the tail of the queue,
  - a remove method to remove the head of the queue, and
  - a peek method to get the head element of the queue without removing it.

- The LinkedList class implements the Queue interface. When you need a queue, initialize a Queue variable with a `LinkedList` object:

```
Queue<String> q = new LinkedList<>();
 q.add("A");
q.add("B");
q.add("C");
while (q.size() > 0) {
        System.out.print(q.remove() + " ");   // Prints A B C
 }
```

Queue Overview

Table 8 Working with Queues	
`Queue<Integer> q = new LinkedList<>();`	The `LinkedList` class implements the `Queue` interface.

`q.add(1);`
`q.add(2);`
`q.add(3);`
	Adds to the tail of the queue; `q` is now `[1, 2, 3]`.

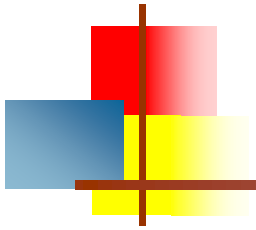
`int head = q.remove();`	Removes the head of the queue; `head` is set to 1 and `q` is `[2, 3]`.
`head = q.peek();`	Gets the head of the queue without removing it; `head` is set to 2.

Priority Queue Overview

- A priority queue collects elements, each of which has a priority.
- Example: a collection of work requests, some of which may be more urgent than others.
- Does not maintain a first-in, first-out discipline.
- Elements are retrieved according to their priority.
- Priority 1 denotes the most urgent priority.
- Each removal extracts the minimum element.
- When you retrieve an item from a priority queue, you always get the most urgent one.

Priority Queue Overview

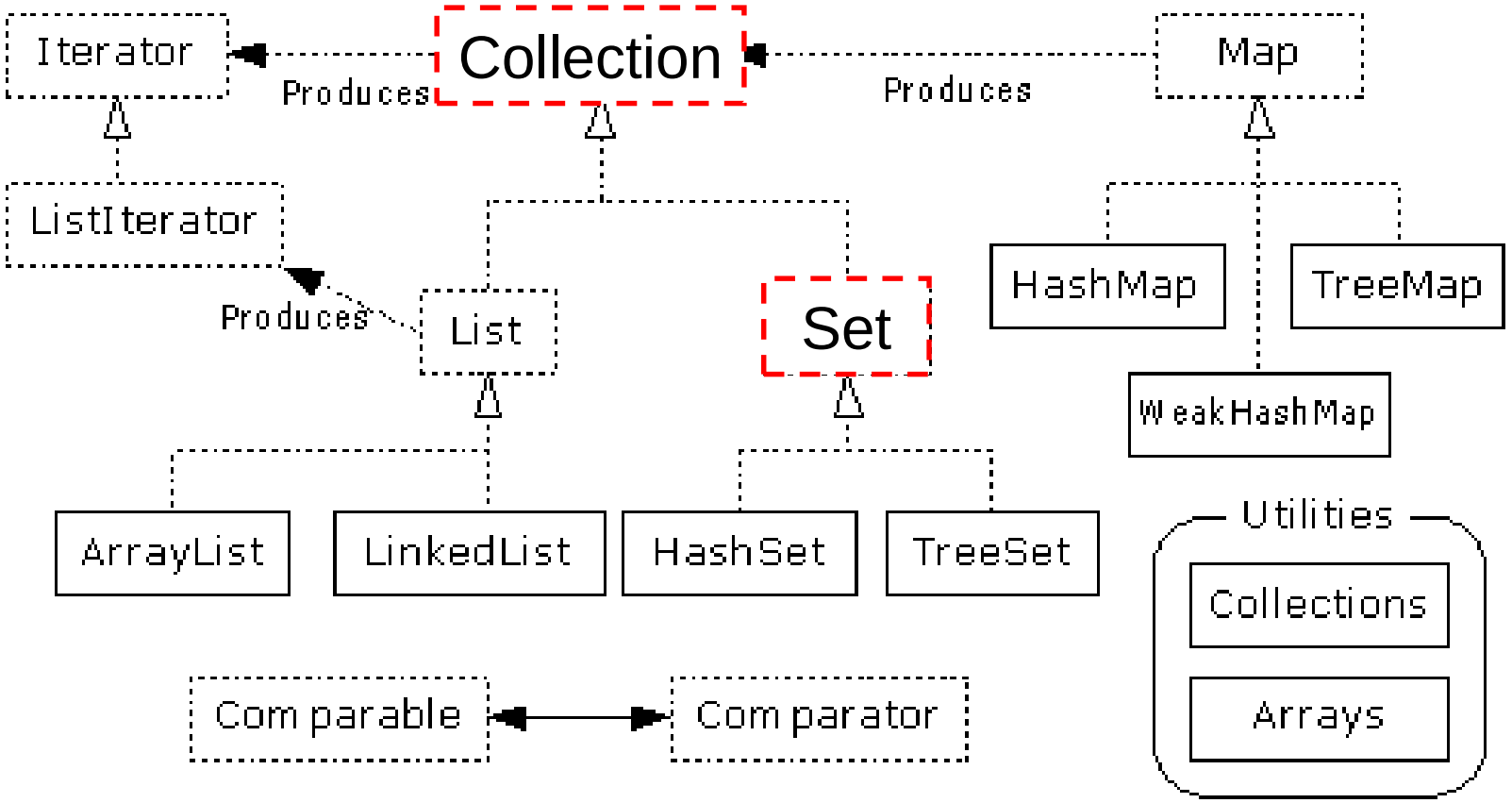
```
4  public class PriorityQ {
5      public static void main(String arg[]){
6          PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
7          pq.add(10);
8          pq.add(7);
9          pq.add(3);
10         pq.add(6);
11         pq.add(2);
12         System.out.println(pq.peek());
13         System.out.println(pq);
14         pq.poll();
15         System.out.println(pq);
16         PriorityQueue<String> spq = new PriorityQueue<String>();
17         spq.add("Zoo");
18         spq.add("Book");
19         spq.add("Apple");
20         spq.add("Yello");
21         System.out.println(spq.peek());
22         System.out.println(spq);
23         spq.poll();
24         System.out.println(spq);
25     }
26
27 }
```

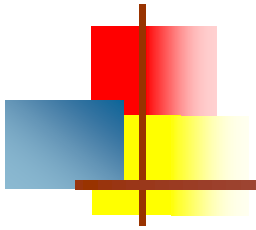


Priority Queue Overview

```
PriorityQueue<WorkOrder> q = new PriorityQueue<>();
q.add(new WorkOrder(3, "Shampoo carpets"));
q.add(new WorkOrder(1, "Fix broken sink"));
q.add(new WorkOrder(2, "Order cleaning supplies"));
```

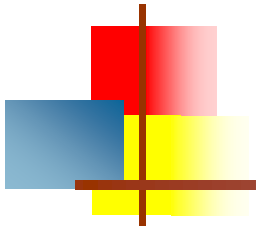
Set Interface Context





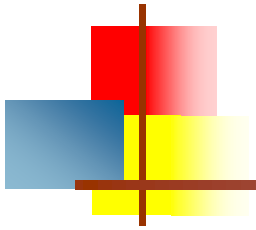
Set Interface

- Set is a child interface of Collection.
- The Set interface present group of individual objects as a single entity where duplicate are not allowed and insertion order is not preserved.
- Inserting and removing elements is more efficient with a set than with a list.



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



Set Interface

- Testing if `s2` is a *subset* of `s1`

  ```
  s1.containsAll(s2)
  ```

- Setting `s1` to the *union* of `s1` and `s2`

  ```
  s1.addAll(s2)
  ```

- Setting `s1` to the *intersection* of `s1` and `s2`

  ```
  s1.retainAll(s2)
  ```

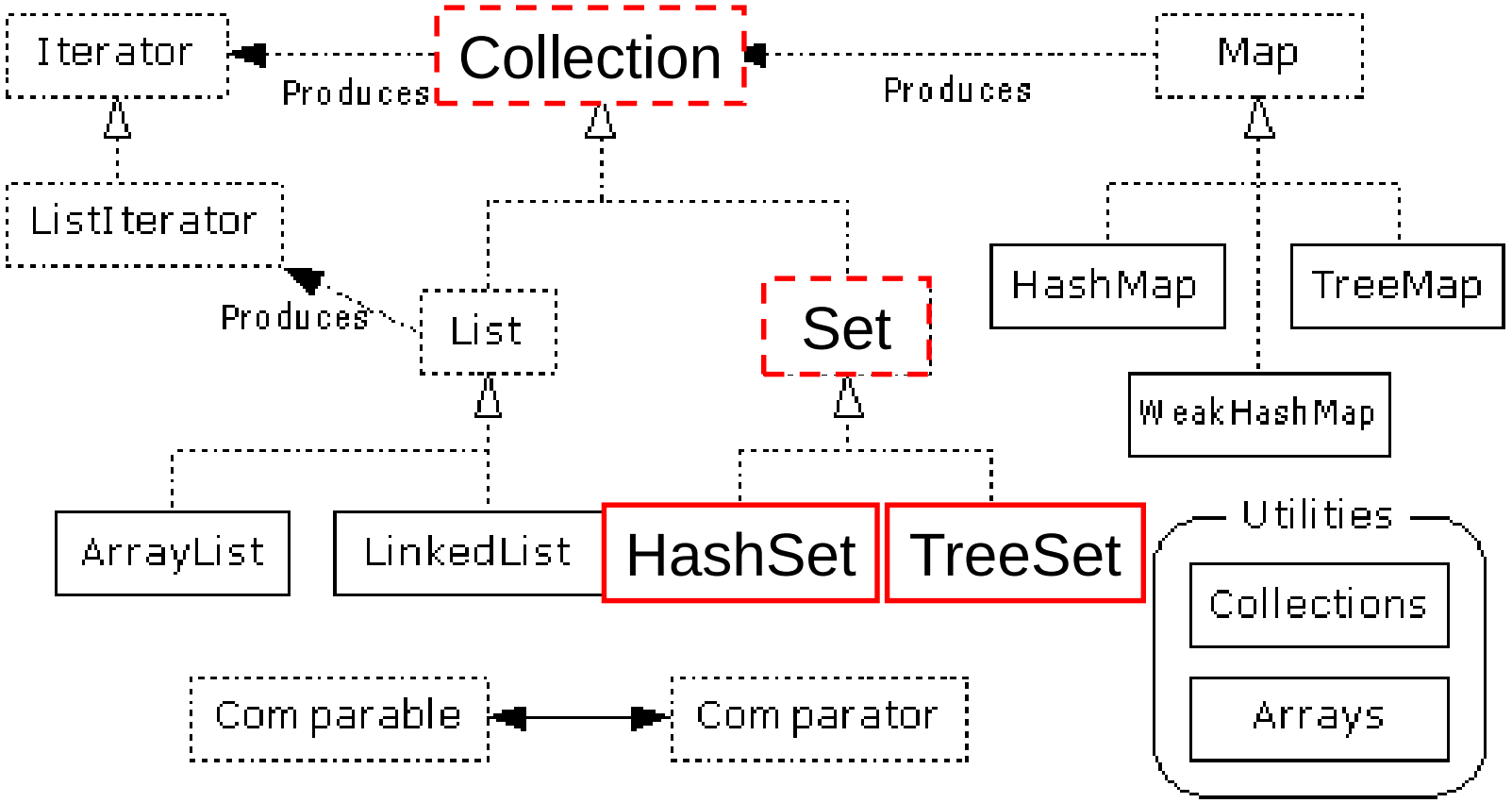
- Setting `s1` to the *set difference* of `s1` and `s2`

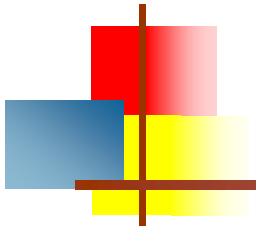
  ```
  s1.removeAll(s2)
  ```

The difference between List and Set?

List	Set
Duplicate is allowed	Duplicate is not allowed
Insertion order is preserved	Insertion order is not preserved

HashSet and TreeSet Context





Hashset overview

- The underlying structure is Hashtable.
- Duplicates are not allowed. What happened if you insert duplicate?
- Insertion order is not preserved and all the objects will be inserted based on hash-code object.
- Set elements are grouped into smaller collections of elements that share the same characteristic.
  - Grouped by an integer hash code that is computed from the element.
- Heterogeneous objects are allowed (int, string, etc).
- Null insertion is allowed.
- Hashset is the best option for frequent search operations.
- Demo