



---

# **Bài 06**

## **Cấu trúc dữ liệu và giải thuật cơ bản**

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

---

# Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Interface và Abstract class”

Tóm tắt lại các phần đã học từ bài “Interface và Abstract class”

# Mục tiêu

---



- Trình bày được Java Collection Framework
- Phân biệt được các trường hợp sử dụng của ArrayList, LinkedList và Set
- Sử dụng được cấu trúc dữ liệu ArrayList
- Sử dụng được cấu trúc dữ liệu LinkedList
- Sử dụng được cấu trúc dữ liệu Set



---

# Thảo luận

Cấu trúc dữ liệu

# Cấu trúc dữ liệu

---



- Cấu trúc dữ liệu là hình thức tổ chức một nhóm dữ liệu:
  - Lưu trữ dữ liệu
  - Cung cấp các phương thức để thao tác với dữ liệu
- Các khái niệm:
  - Container: Lớp chứa dữ liệu
  - Elements: Các phần tử dữ liệu
- Ví dụ:
  - Lớp ArrayList là cấu trúc danh sách, lưu trữ nhiều giá trị
  - Các phương thức được cung cấp để thực hiện các thao tác: Thêm phần tử, xóa phần tử, duyệt phần tử, tìm kiếm...
- Việc lựa chọn cấu trúc dữ liệu và thuật toán phù hợp là rất quan trọng đối với hiệu năng của ứng dụng



# Các cấu trúc dữ liệu thông dụng

---

- Set (Tập hợp): Nhóm các phần tử không trùng nhau
- List (Danh sách): Nhóm các phần tử có thể trùng nhau
- Stack: Nhóm các phần tử theo trật tự first-in/last-out (vào trước/ra sau)
- Queue: Nhóm các phần tử theo trật tự first-in/first-out (vào trước/ra trước)
- Map (Bản đồ): Lưu trữ các cặp key/value
- Tree (Cây): Lưu trữ các phần tử theo mối quan hệ cha-con
- Graph (Đồ thị): Lưu trữ các phần tử theo mối quan hệ mạng lưới



---

# Thảo luận

Java Collection Framework



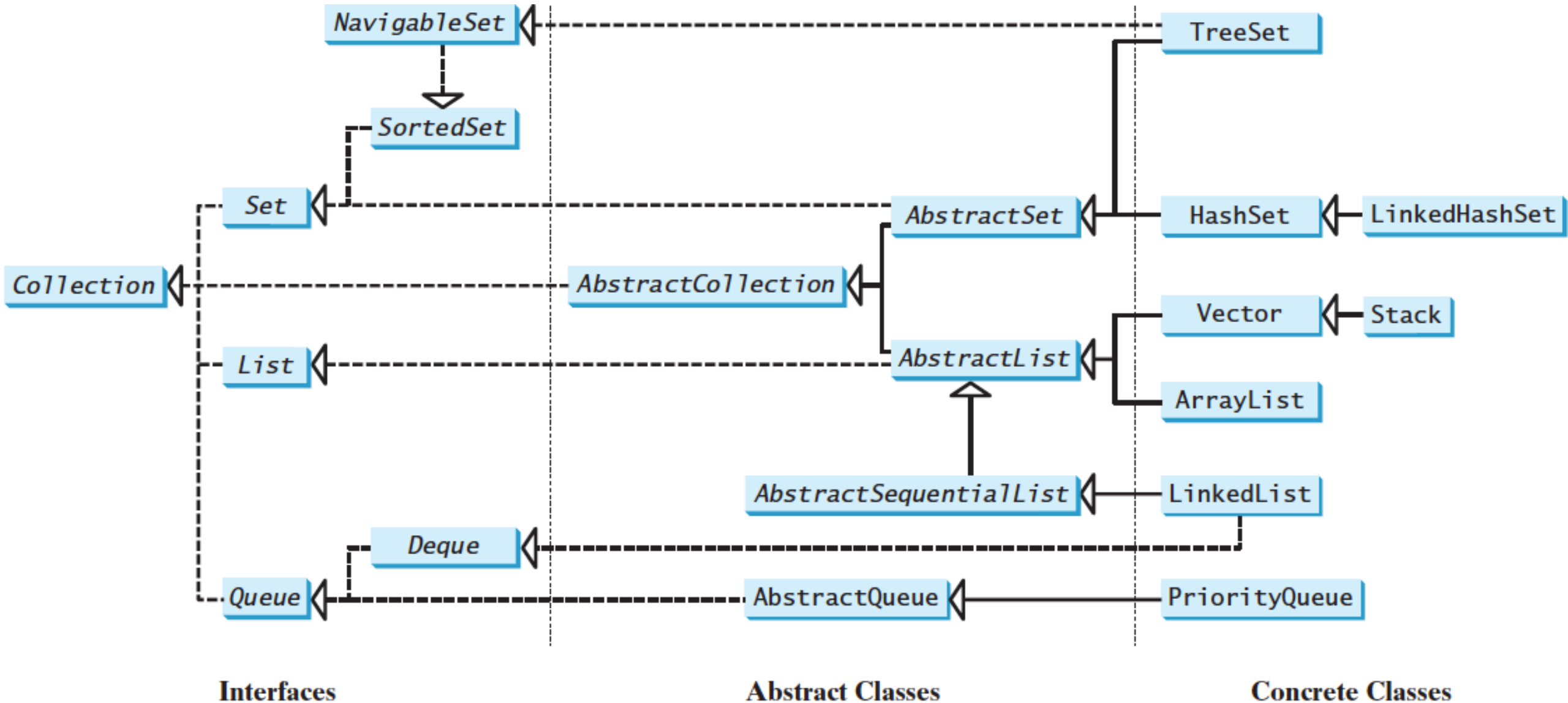
# Java Collection

---

- Java Collection Framework cung cấp các interface và lớp để thực hiện các thao tác với cấu trúc dữ liệu
- Các interface và lớp của Java Collection được đặt trong gói *java.util*
- Thiết kế của Java Collection:
  - Các interface mô tả các thao tác của từng loại cấu trúc dữ liệu
  - Các abstract class triển khai các phương thức chung
  - Các lớp triển khai các phương thức cụ thể của từng loại cấu trúc dữ liệu
- Interface Collection là interface gốc của tất cả các interface và lớp còn lại của Java Collection



# Java Collection Framework



# Interface Collection



**«interface»**  
*java.util.Collection<E>*

```
+add(o: E): boolean
+addAll(c: Collection<? extends E>): boolean
+clear(): void
+contains(o: Object): boolean
+containsAll(c: Collection<?>): boolean
+equals(o: Object): boolean
+hashCode(): int
+isEmpty(): boolean
+remove(o: Object): boolean
+removeAll(c: Collection<?>): boolean
+retainAll(c: Collection<?>): boolean
+size(): int
+toArray(): Object[]
```

# Iterator



- Collection kế thừa từ interface Iterable
- Phương thức iterator trả về đối tượng Iterator
- Iterator là cơ chế cho phép duyệt qua các phần tử của một cấu trúc dữ liệu

- Ví dụ:

```
Collection<String> collection = new ArrayList<>();  
collection.add("New York");  
collection.add("Atlanta");  
collection.add("Dallas");  
collection.add("Madison");
```

```
Iterator<String> iterator = collection.iterator();  
while (iterator.hasNext()) {  
    System.out.print(iterator.next().toUpperCase() + " ");  
}
```

«interface»  
*java.util.Iterator<E>*

+hasNext(): boolean  
+next(): E  
+remove(): void



---

# Demo

Iterator



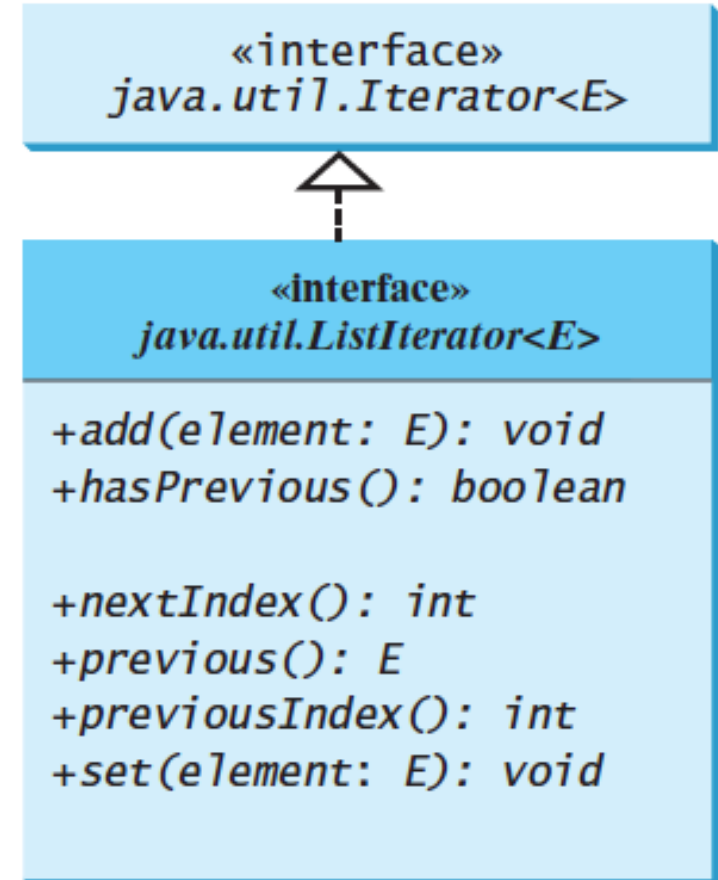
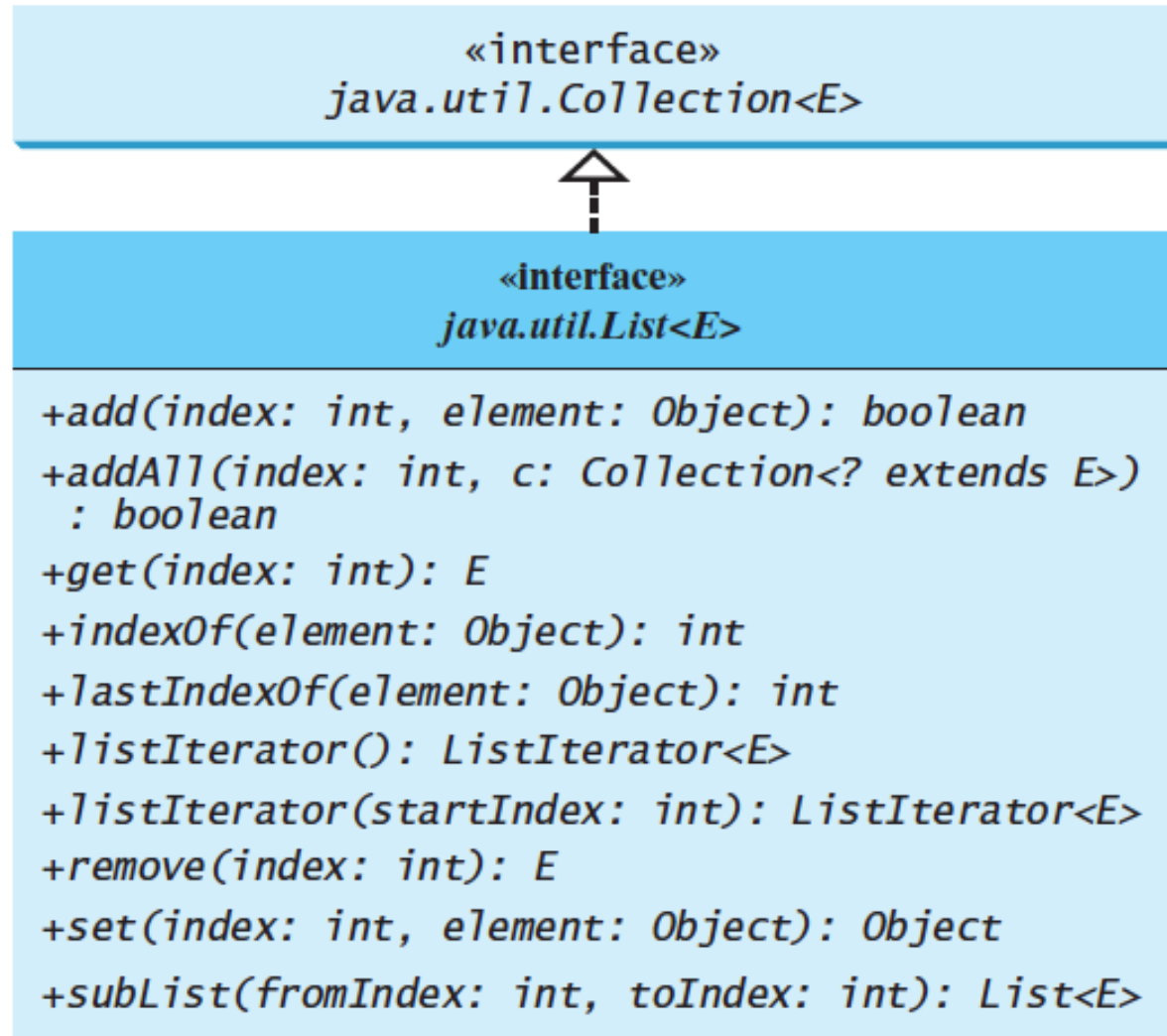
---

# Thảo luận

Interface List & ListIterator

Lớp ArrayList & LinkedList

# Interface List và ListIterator



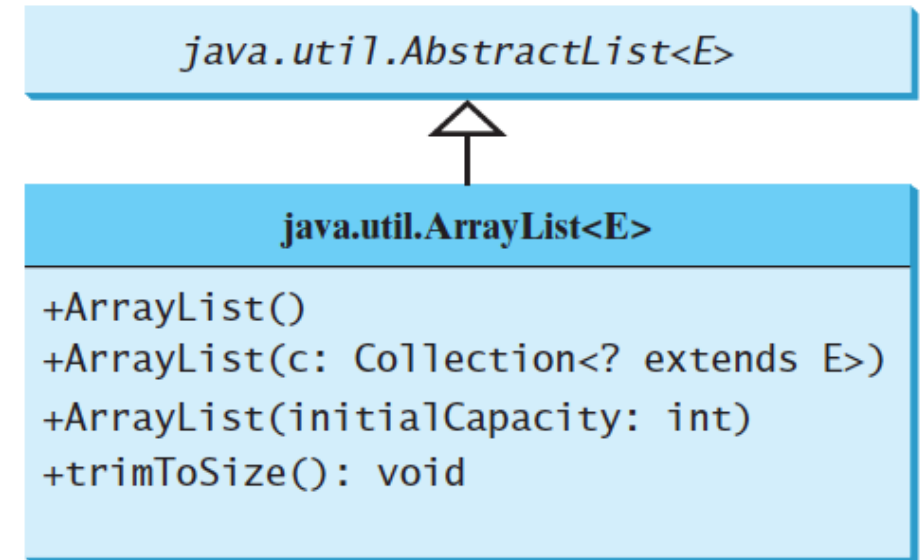


# Lớp ArrayList

- ArrayList lưu trữ các phần tử bên trong một mảng
- Kích thước của mảng được điều chỉnh tùy theo số lượng các phần tử
- Ví dụ:

```
List<String> arrayList = new ArrayList<>();  
arrayList.add("New York");  
arrayList.add("Atlanta");  
arrayList.add("Dallas");  
arrayList.add("Madison");
```

```
System.out.println(arrayList);
```



# Lớp LinkedList



- Lớp LinkedList lưu trữ các phần tử sử dụng cơ chế liên kết (link)
- Ví dụ:

```
List<String> linkedList = new LinkedList<>();  
linkedList.add("New York");  
linkedList.add("Atlanta");  
linkedList.add("Dallas");  
linkedList.add("Madison");
```

```
ListIterator<String> listIterator = linkedList.listIterator();  
while (listIterator.hasNext()) {  
    System.out.print(listIterator.next() + " ");  
}
```

*java.util.AbstractSequentialList<E>*



**java.util.LinkedList<E>**

```
+LinkedList()  
+LinkedList(c: Collection<? extends E>)  
+addFirst(element: E): void  
+addLast(element: E): void  
+getFirst(): E  
+getLast(): E  
+removeFirst(): E  
+removeLast(): E
```





# Lựa chọn ArrayList hay LinkedList

---

ArrayList	LinkedList
Truy xuất ngẫu nhiên nhanh	Truy xuất ngẫu nhiên chậm
Thêm, xoá chậm	Thêm, xoá nhanh

- ArrayList phù hợp với các bài toán cần thực hiện nhiều thao tác truy xuất ngẫu nhiên và ít thêm, xoá ở đầu danh sách
- LinkedList phù hợp với các bài toán cần thêm, xoá nhiều ở đầu danh sách và ít truy xuất ngẫu nhiên



---

# Demo

Lớp ArrayList & LinkedList



---

# Thảo luận

Triển khai ArrayList



# Các thao tác cơ bản của ArrayList

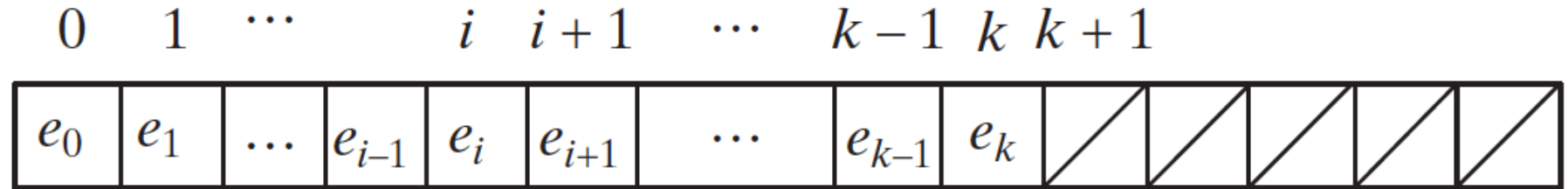
---

- `get()`: Lấy về một phần tử
  - `add()`: Thêm một phần tử
  - `remove()`: Xoá một phần tử
  - `size()`: Lấy về số lượng phần tử
  - `find()`: Tìm kiếm phần tử
  - `isEmpty()`: Kiểm tra rỗng
- 
- Sử dụng mảng để lưu trữ các phần tử

# Thêm phần tử vào ArrayList



Trước khi chèn  $e$   
vào vị trí  $i$

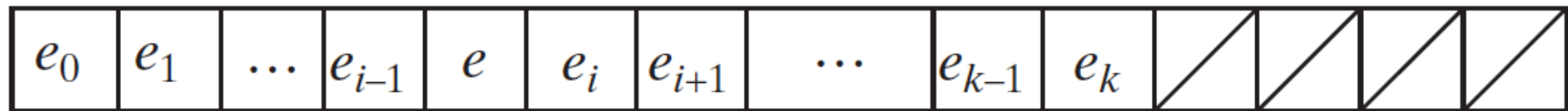


Vị trí chèn  $e$

...dịch chuyển...

$\text{data.length} - 1$

0 1 ...  $i$   $i+1$   $i+2$  ...  $k$   $k+1$



Sau khi chèn  $e$   
vào vị trí  $i$ , kích  
thước ArrayList  
tăng thêm 1

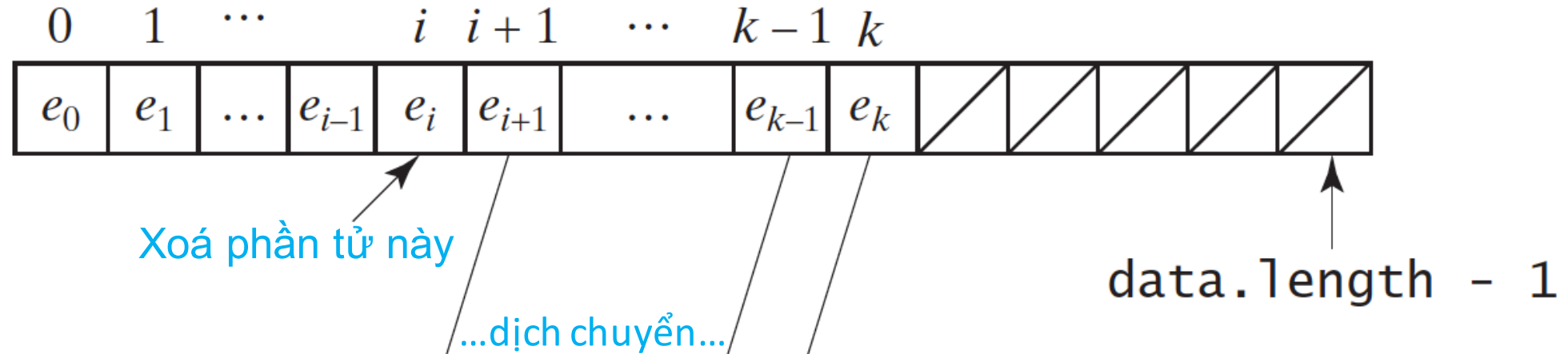
$e$  được chèn vào đây

$\text{data.length} - 1$

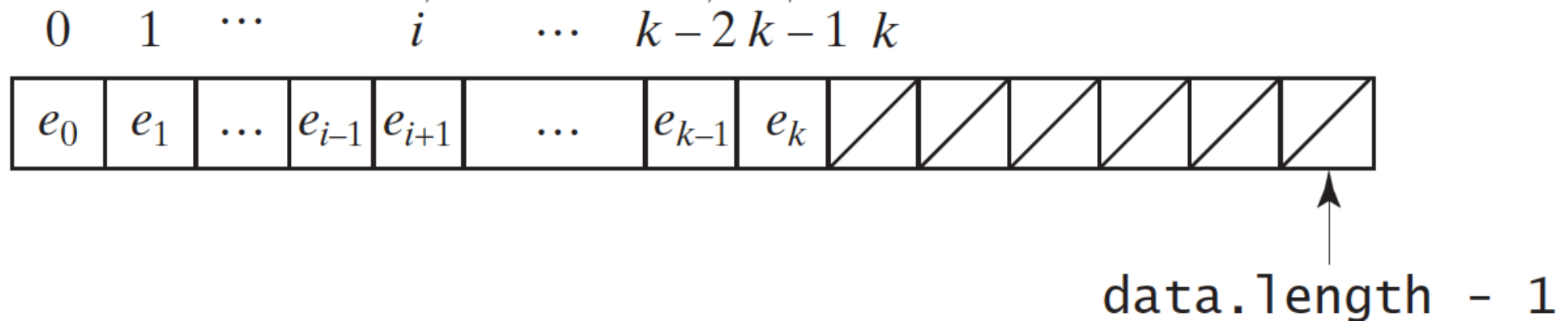
# Xoá phần tử khỏi ArrayList

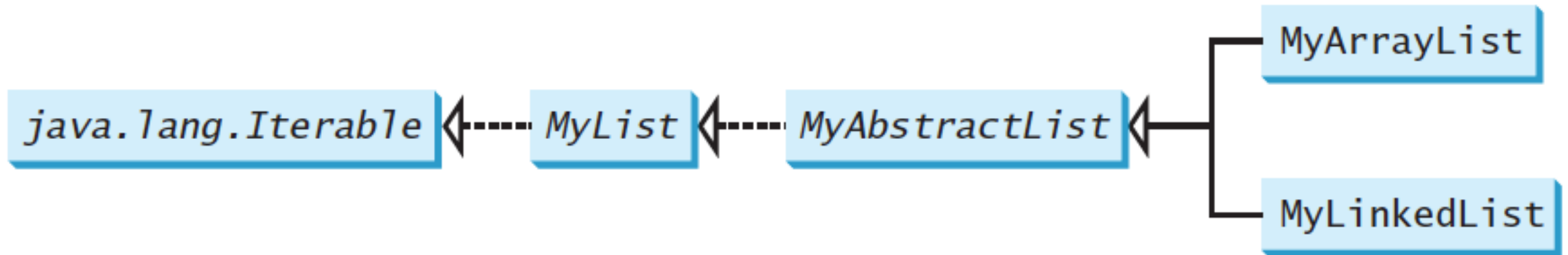


Trước khi xoá  
phần tử tại vị trí  $i$



Sau khi xoá phần tử  
tại vị trí  $i$ , kích thước  
của ArrayList giảm  
xuống 1





# Interface MyList và MyAbstractList



«interface»  
*MyList<E>*

```
+add(e: E): void  
+add(index: int, e: E): void  
+clear(): void  
+contains(e: E): boolean  
+get(index: int): E  
+indexOf(e: E): int  
+isEmpty(): boolean  
+lastIndexOf(e: E): int  
+remove(e: E): boolean  
+size(): int  
+remove(index: int): E  
+set(index: int, e: E): E
```

*MyAbstractList<E>*

#size: int

```
#MyAbstractList()  
#MyAbstractList(objects: E[])  
+add(e: E): void  
+isEmpty(): boolean  
+size(): int  
+remove(e: E): boolean
```



# Lớp MyArrayList



*MyAbstractList<E>*



**MyArrayList<E>**

-data: E[]

+MyArrayList()

+MyArrayList(objects: E[])

+trimToSize(): void

-ensureCapacity(): void

-checkIndex(index: int): void



---

# Demo

Triển khai ArrayList



---

# Thảo luận

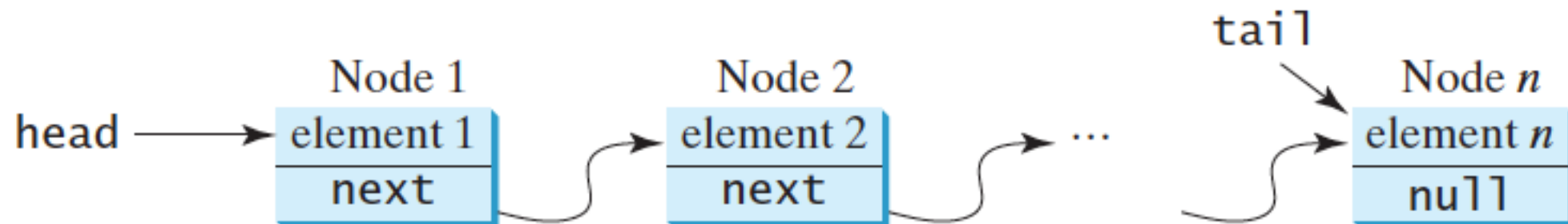
Triển khai LinkedList

# LinkedList

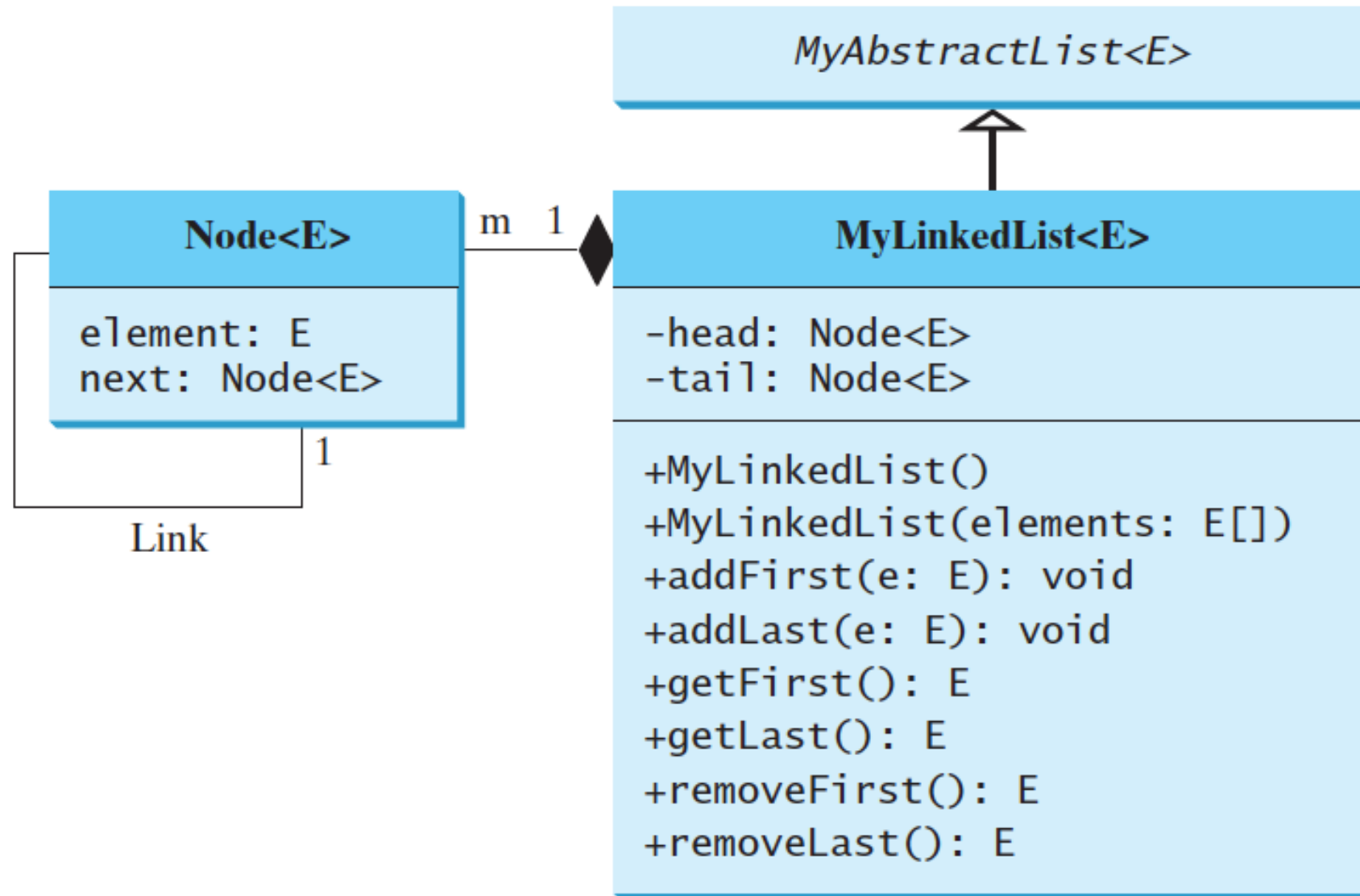


- LinkedList hoạt động dựa trên cơ chế liên kết giữa các Node
- Mỗi node chứa dữ liệu của node đó và liên kết đến node khác

```
class Node<E>{  
    E element;  
    Node<E> next;  
    public Node(E e) {  
        element = e;  
    }  
}
```



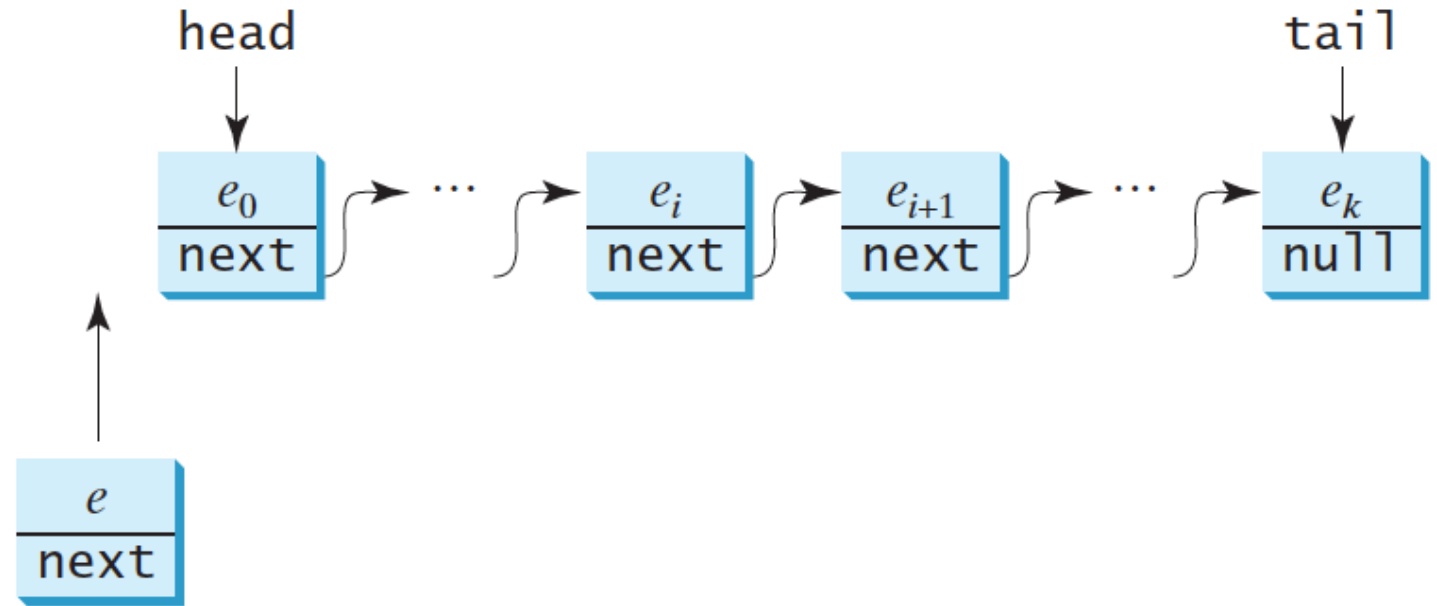
# Lớp MyLinkedList



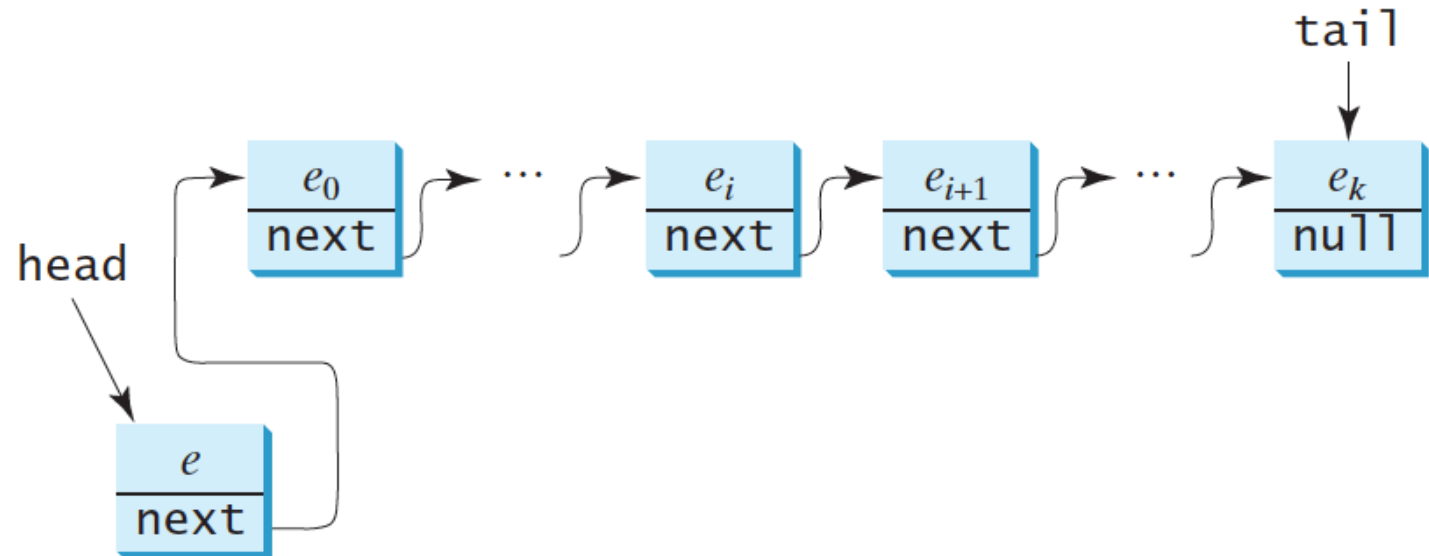
# Phương thức addFirst()



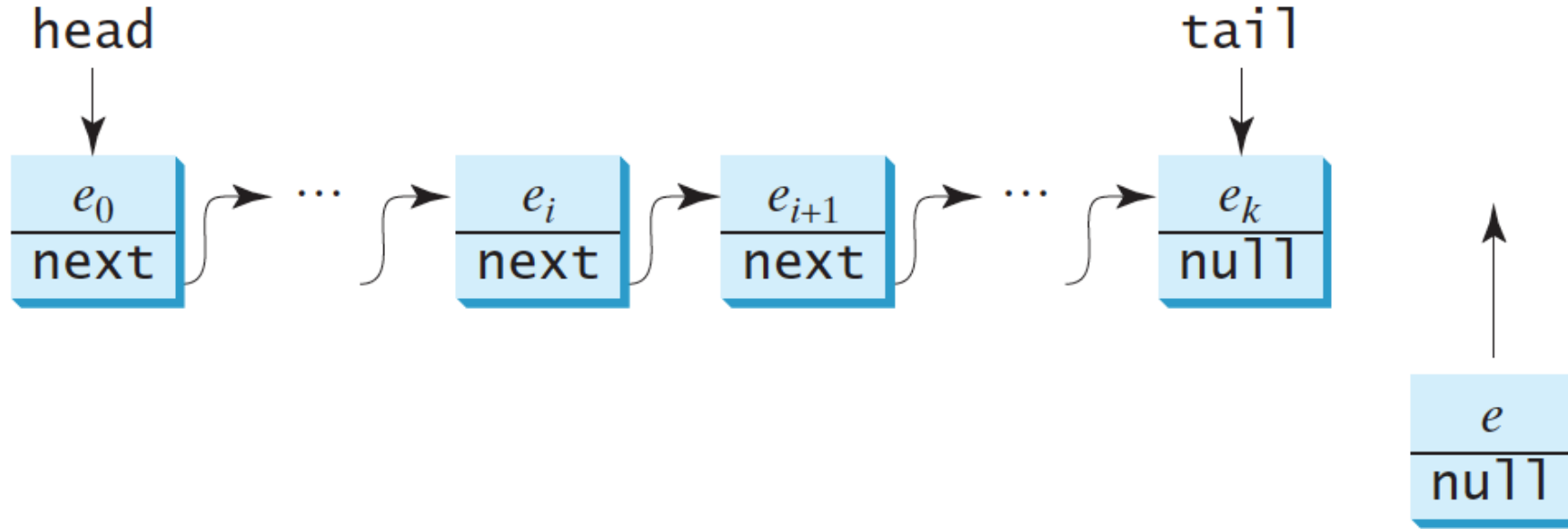
Trước khi chèn  
phần tử vào đầu



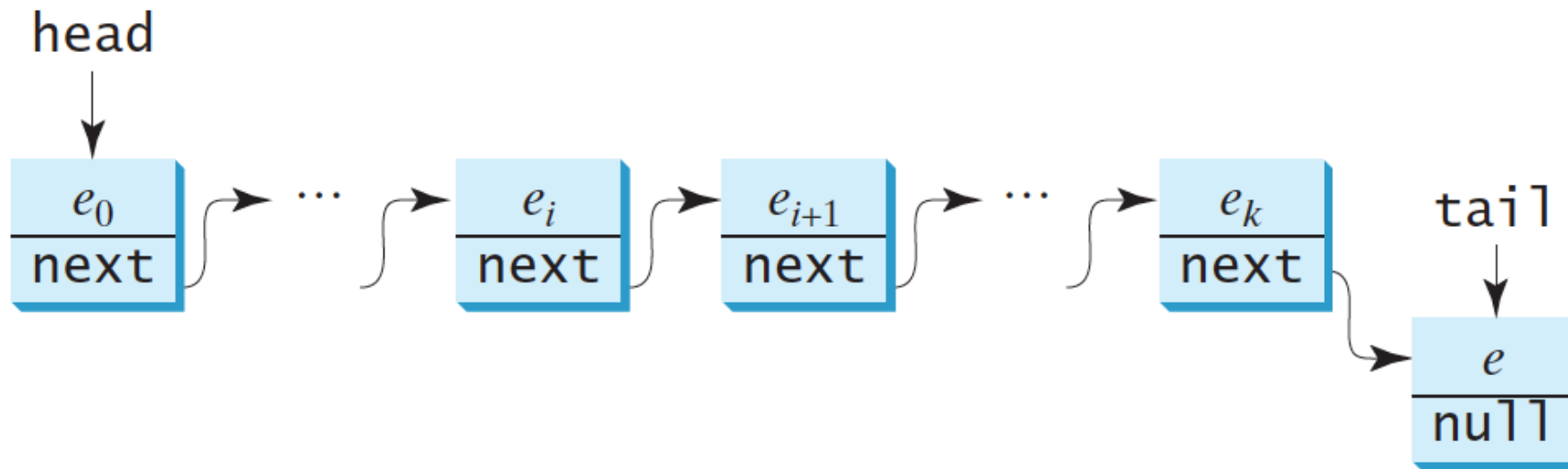
Sau khi được  
chèn vào đầu



# Phương thức addLast()

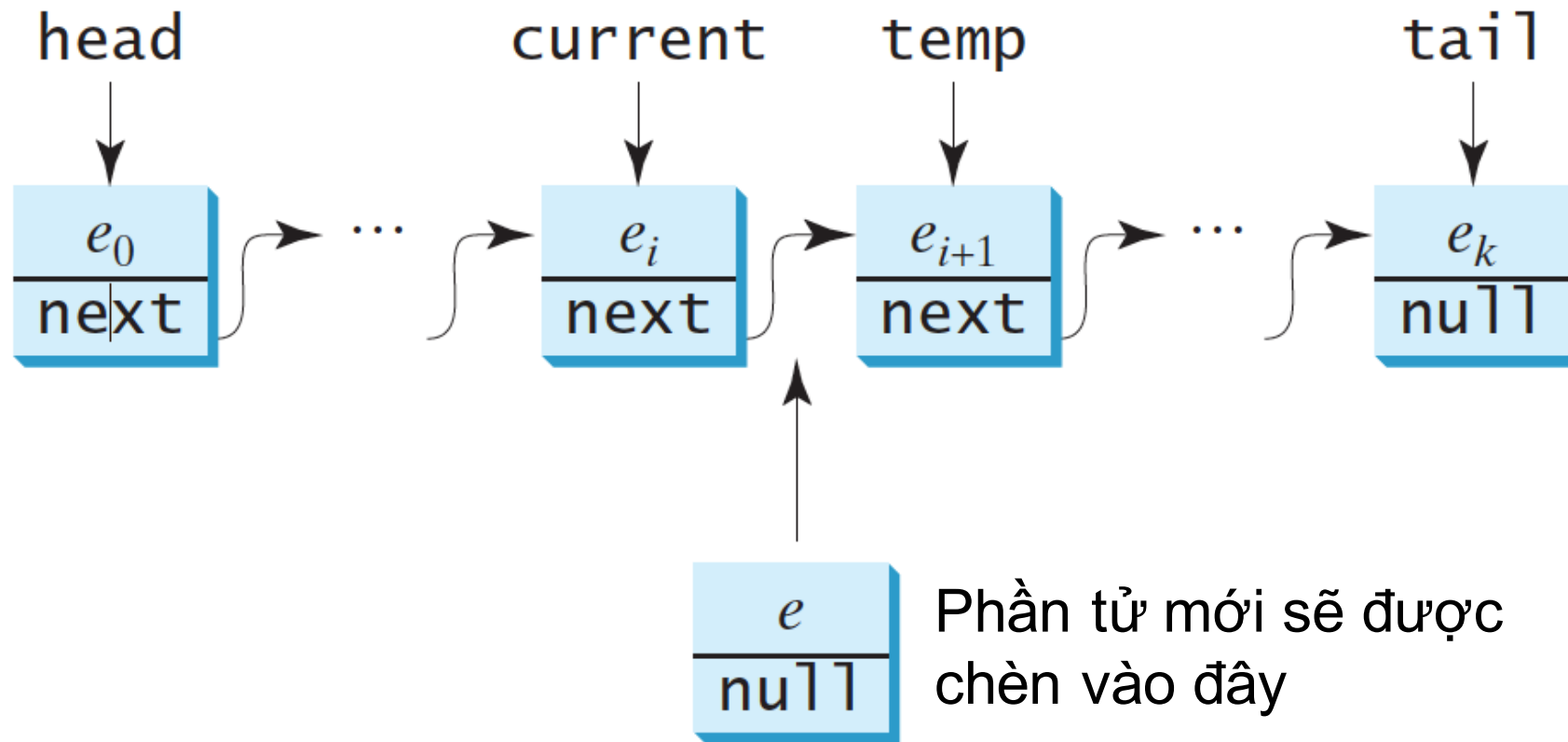


Trước khi chèn  
phần tử vào đuôi



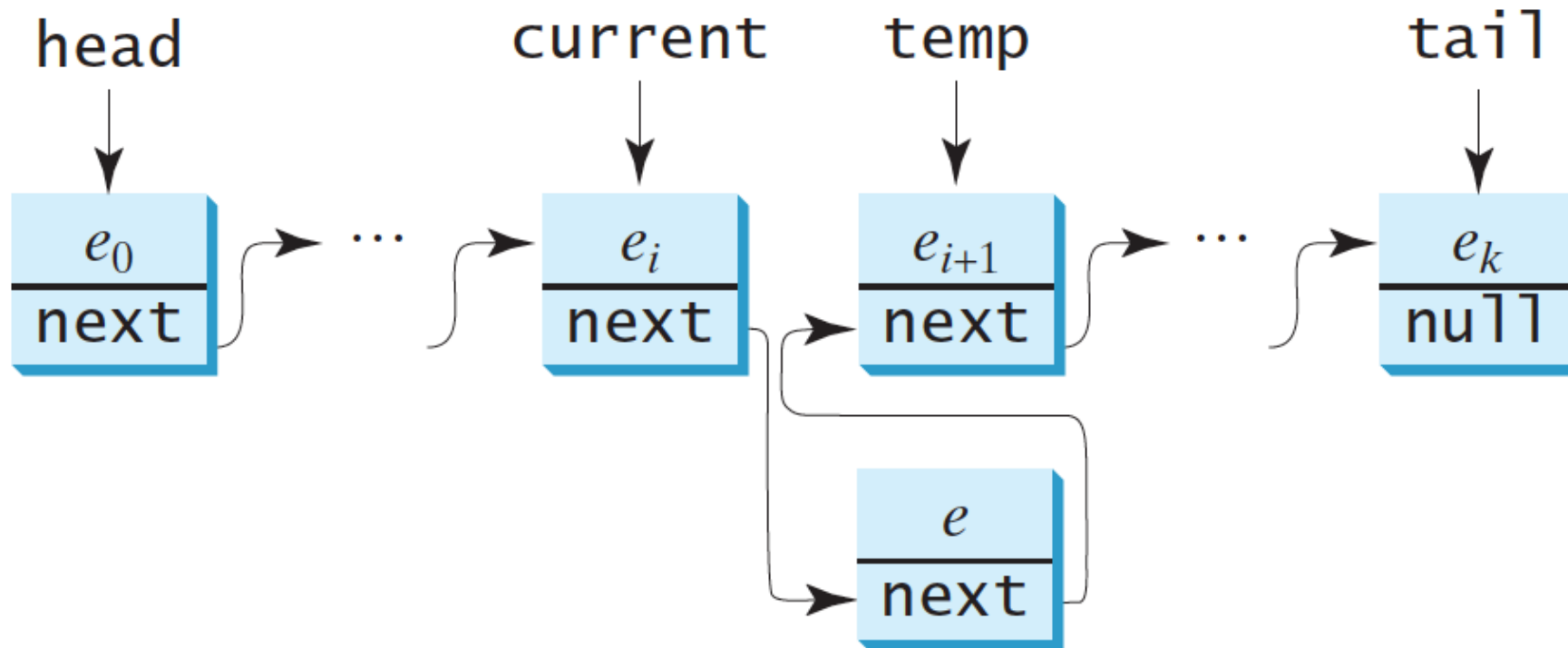
Sau khi chèn  
phần tử vào đuôi

# Phương thức add()



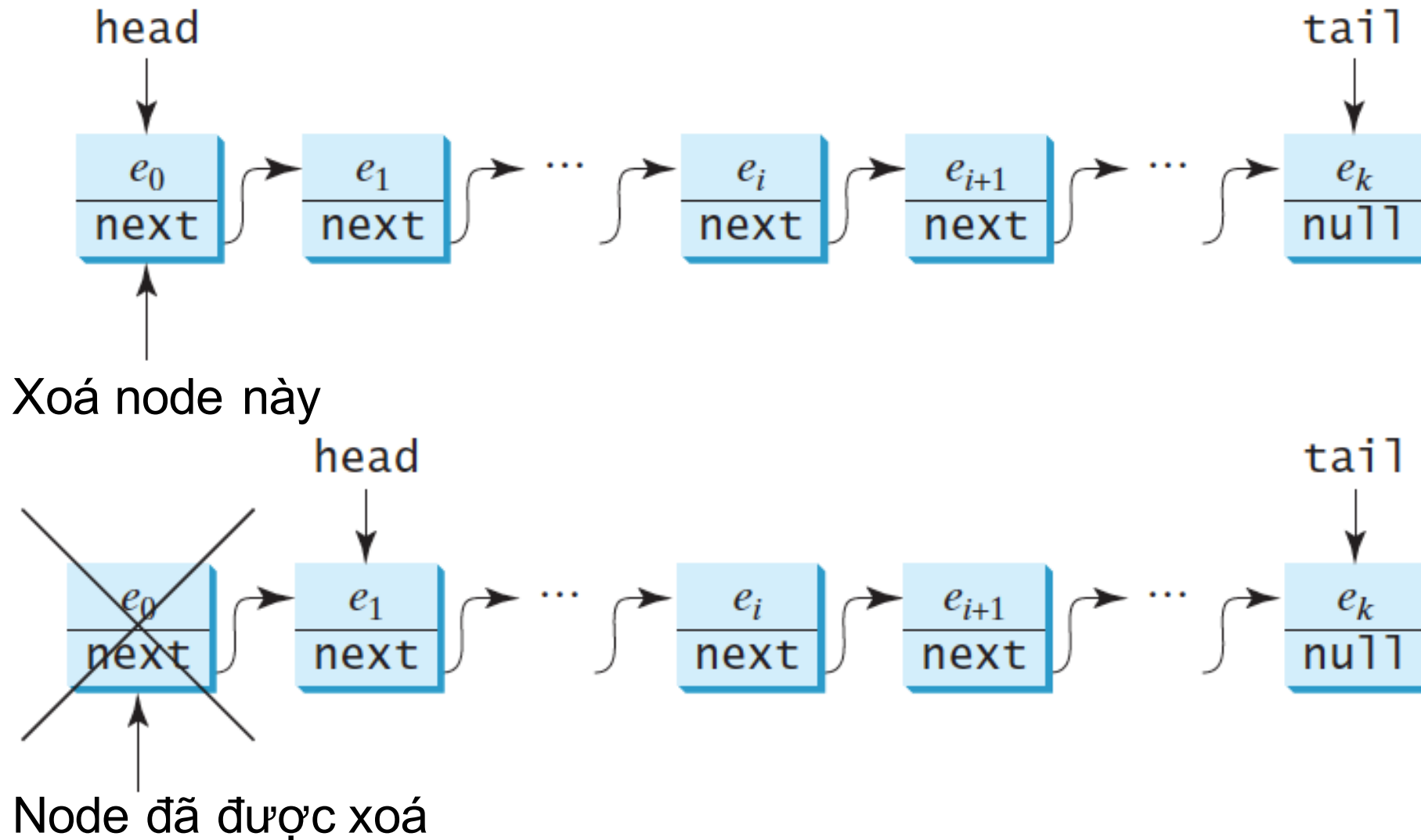


# Phương thức add()

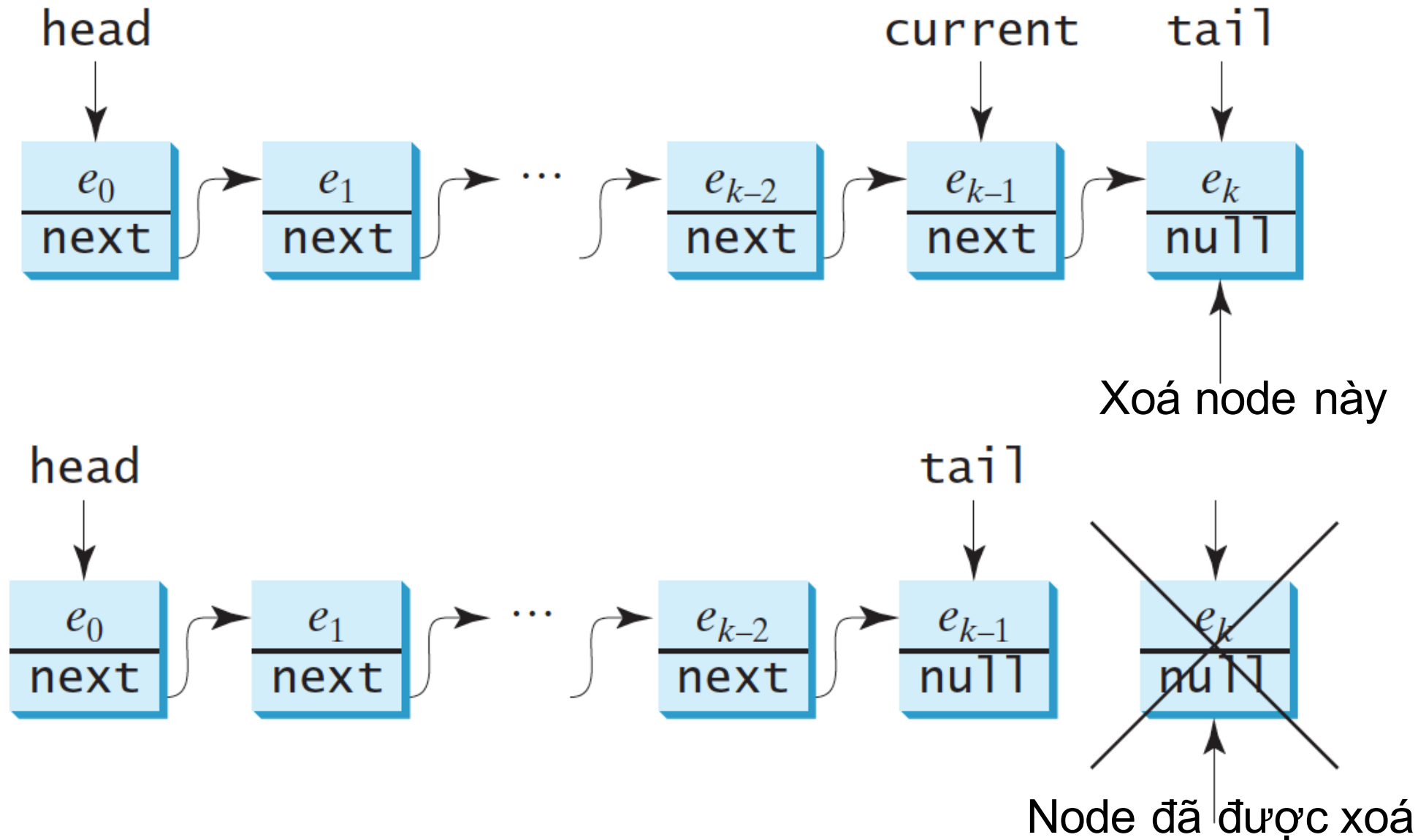


Phần tử mới đã được  
chèn vào LinkedList

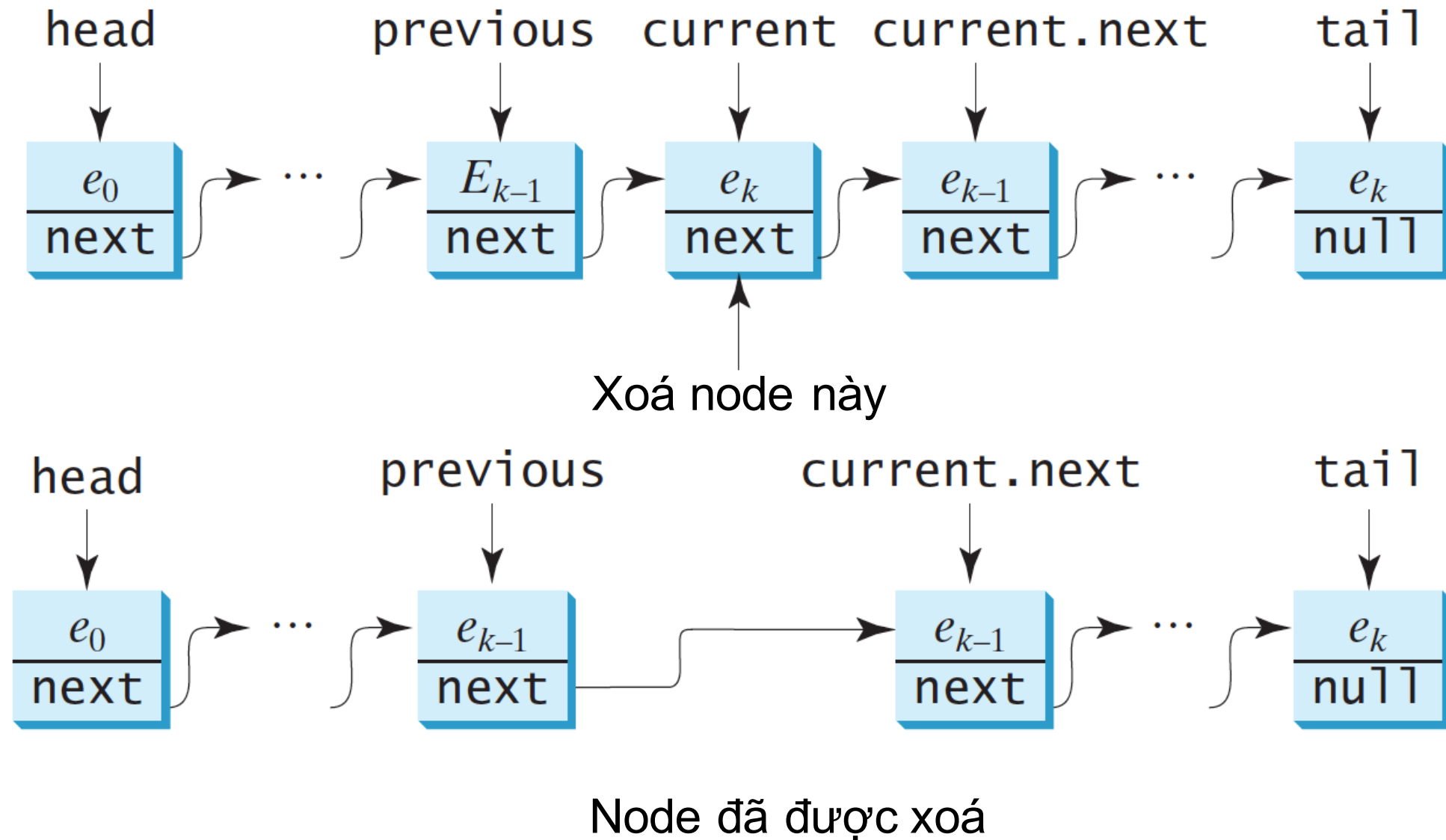
# Phương thức removeFirst()



# Phương thức removeLast()



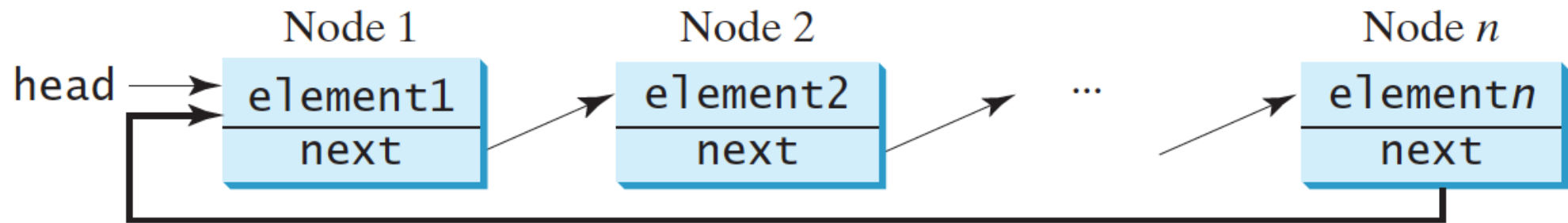
# Phương thức remove()



# Singly Linked List



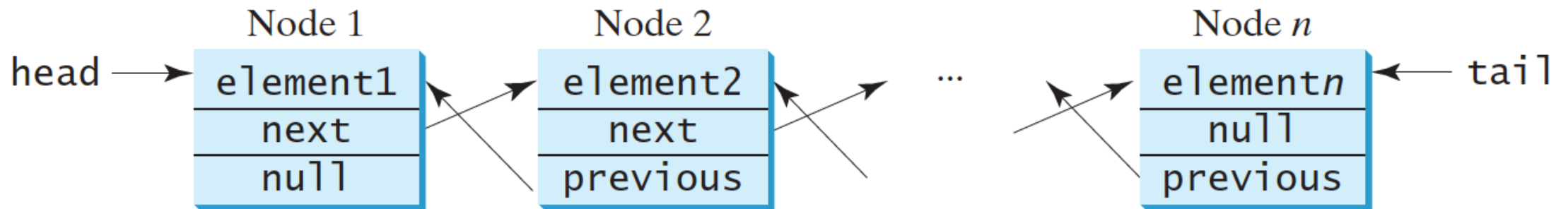
- Singly Linked List (Danh sách liên kết đơn):
  - Một node chỉ có một liên kết đến node phía sau nó
  - Node cuối cùng trở đến null
- Circular Singly LinkedList (Danh sách liên kết đơn vòng):
  - Node cuối vòng trở đến node đầu tiên



# Doubly Linked List



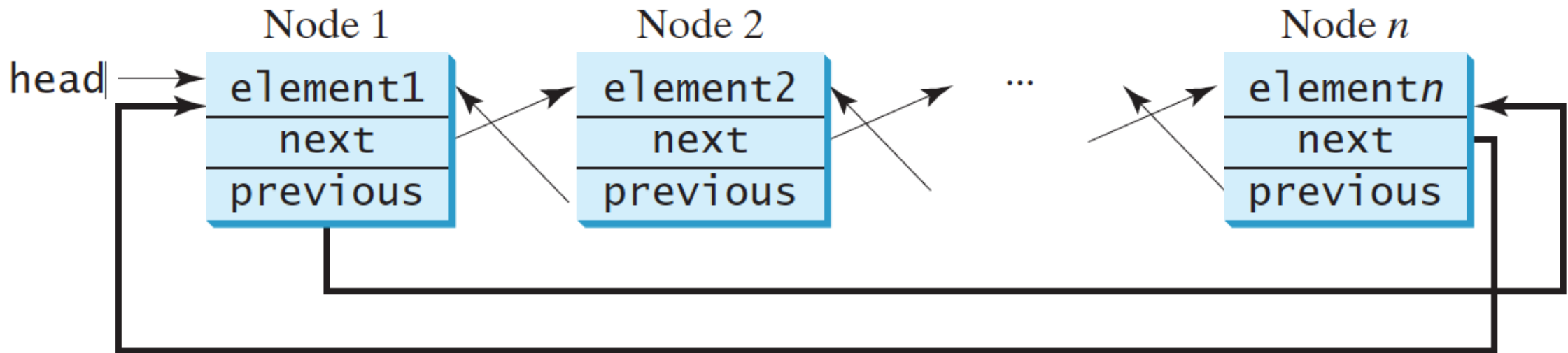
- Doubly Linked List (Danh sách liên kết đôi):
  - Một node chứa hai liên kết trỏ đến phần tử đứng trước và sau nó
  - Phần tử trước của phần tử đầu tiên là null
  - Phần tử sau của phần tử sau là null



# Circular Doubly Linked List



- Circular Doubly Linked List (Danh sách liên kết đôi vòng):
  - Node đầu tiên và node cuối cùng có liên kết trở đến nhau





---

# Demo

Triển khai LinkedList



---

# Thảo luận

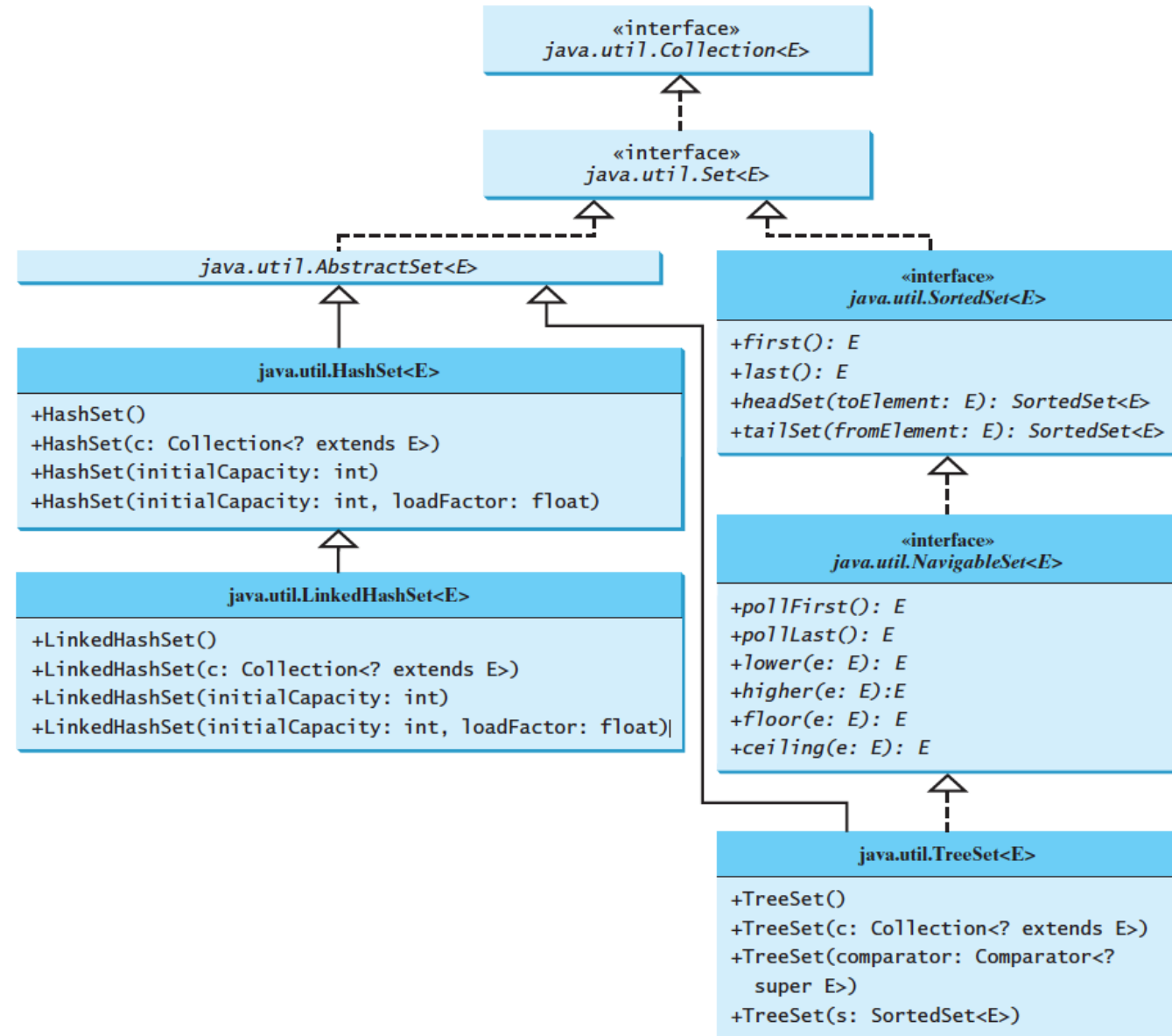
Interface Set

Các lớp trong tập Set

# Set



- Set (Tập hợp) là cấu trúc dữ liệu lưu trữ các phần tử không trùng lặp
- Có thể sử dụng một trong các lớp:
  - HashSet
  - LinkedHashSet
  - TreeSet



# Set: Ví dụ

---



```
public static void main(String[] args) {  
    Set<String> set = new HashSet<>();  
  
    set.add("London");  
    set.add("Paris");  
    set.add("New York");  
    set.add("San Francisco");  
    set.add("Beijing");  
    set.add("New York");  
  
    System.out.println(set);  
  
    for (String s: set) {  
        System.out.print(s.toUpperCase() + " ");  
    }  
}
```



---

# Demo

Sử dụng Set



# Tóm tắt bài học

---

- Cấu trúc dữ liệu là hình thức tổ chức dữ liệu và cung cấp các phương thức để thao tác với các phần tử
- Cấu trúc List chứa các phần tử cho phép trùng lặp
- Cấu trúc Set chứa các phần tử không trùng lặp
- ArrayList lưu trữ các phần tử trong mảng
- LinkedList lưu trữ các phần tử theo cơ chế liên kết giữa các phần tử
- Java Collection Framework cung cấp đầy đủ các interface và class thực hiện các thao tác với cấu trúc dữ liệu thông dụng



---

# Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Clean Code*