Bài 10
Cấu trúc dữ liệu và giải thuật cơ bản

Module: ADVANCED PROGRAMMING WITH JAVA

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài "TestFirst & TDD"

Tóm tắt lại các phần đã học từ bài "TestFirst & TDD"

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
- Trình bày được khái niệm Generic
- Sử dụng được cơ chế Generic

Thảo luận

Cấu trúc dữliệu

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, xoá 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 ác 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

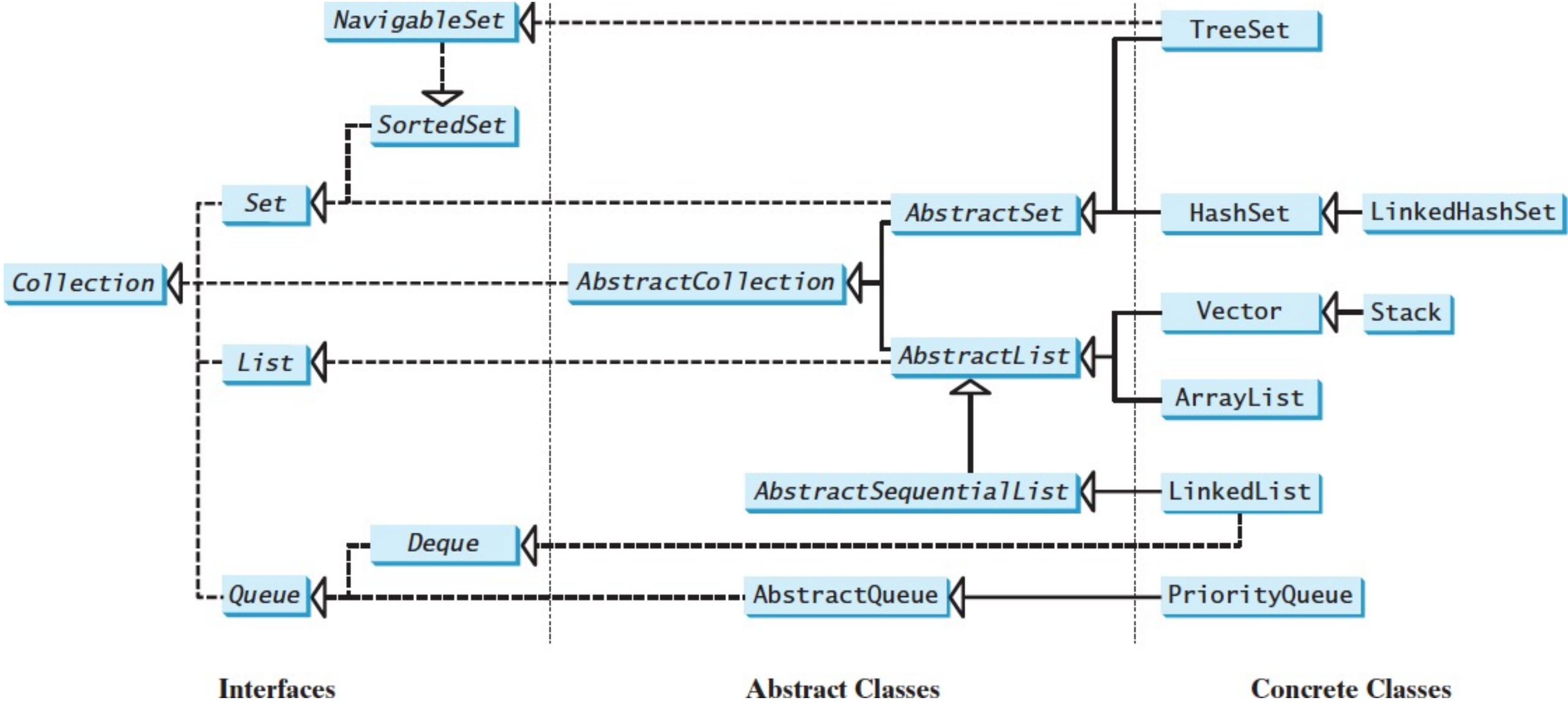
Thảo luận

Java Collection Framework

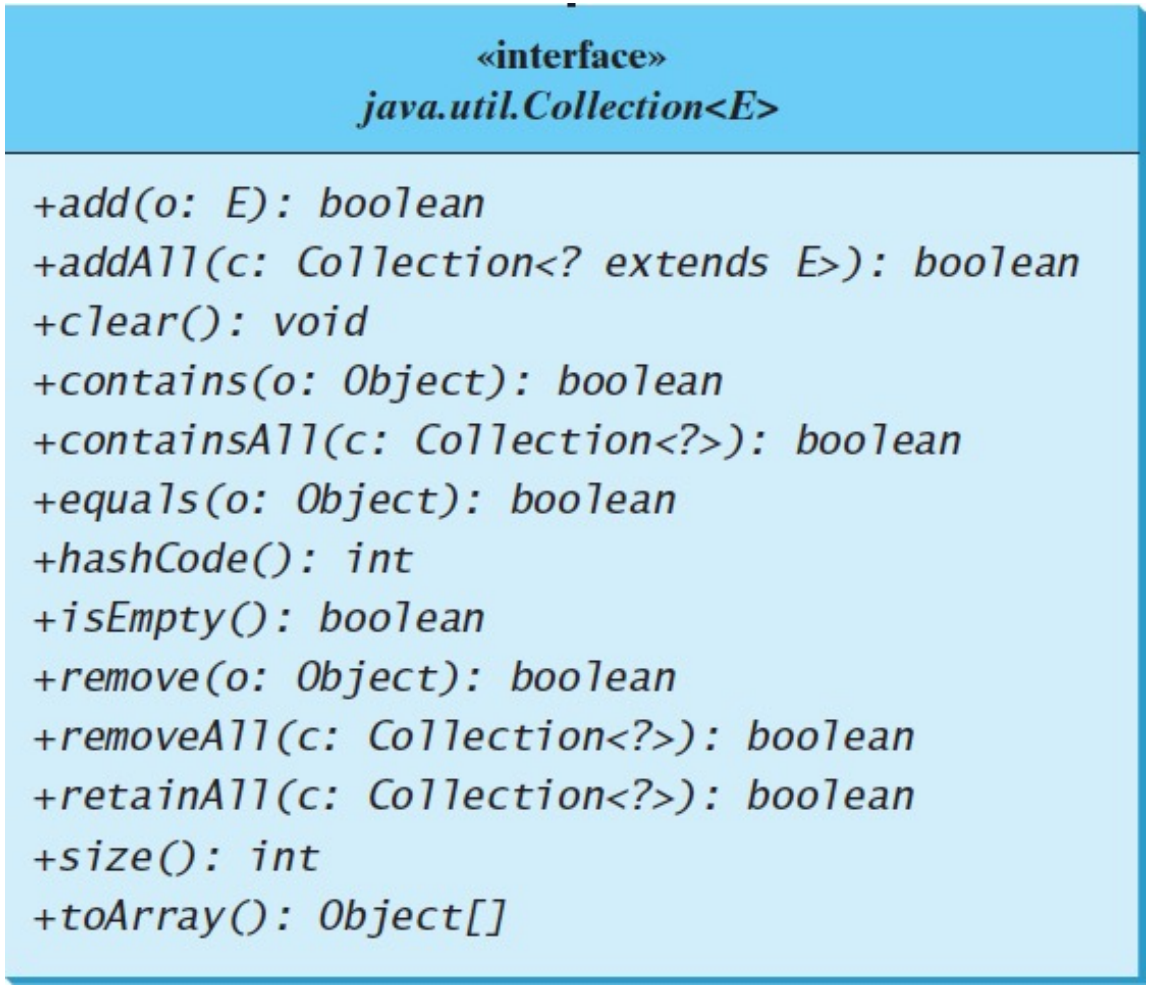
Java Collection

- 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

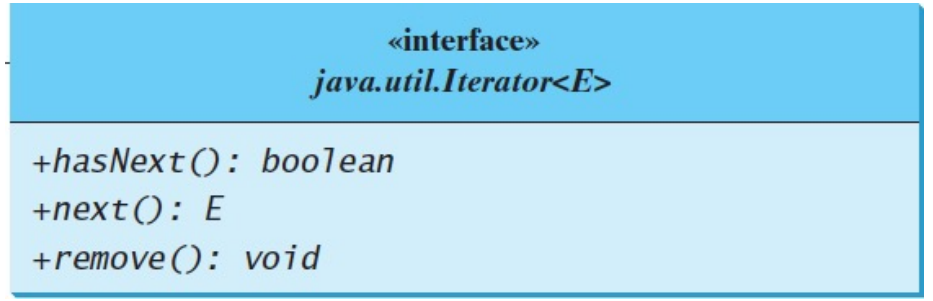


Iterator

- Collection kế thừa từ interface Interable
- Phương thức interator 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() + " ");
}
```



Demo

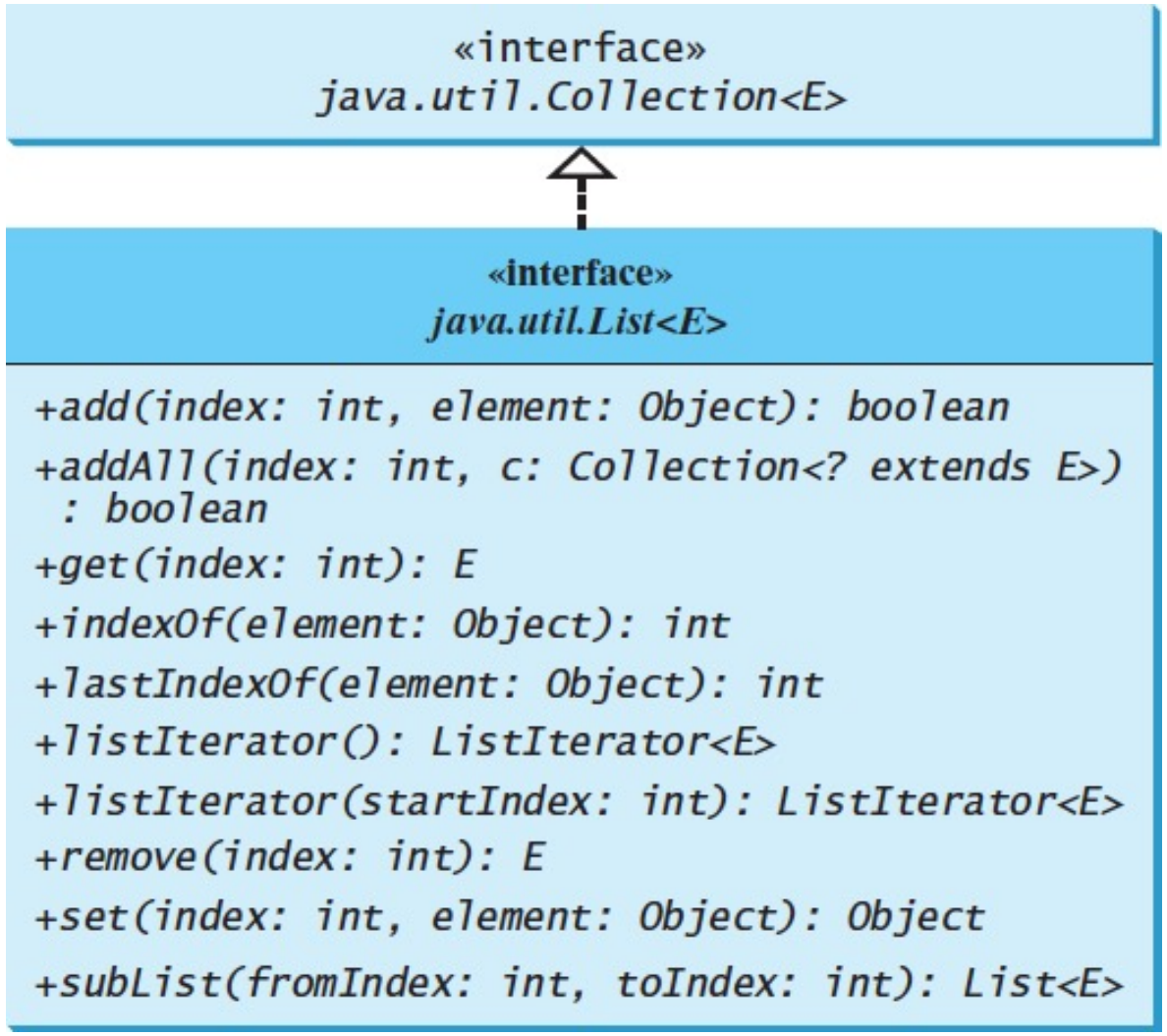
Iterator

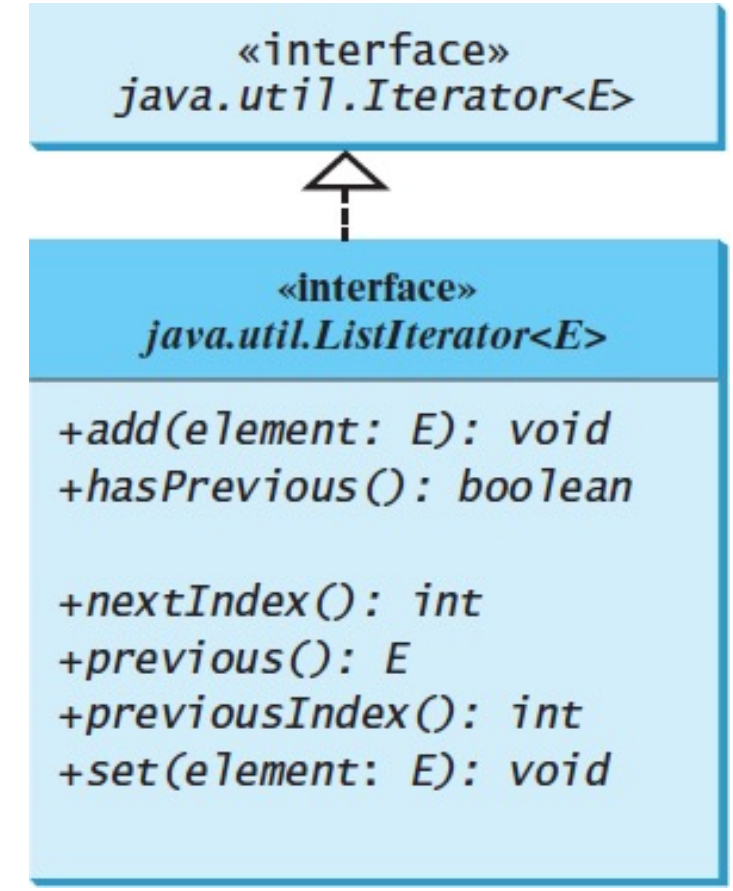
Thảo luận

Interface List& ListIterator

Lớp ArrayList& LinkedList

Interface List và ListIterator



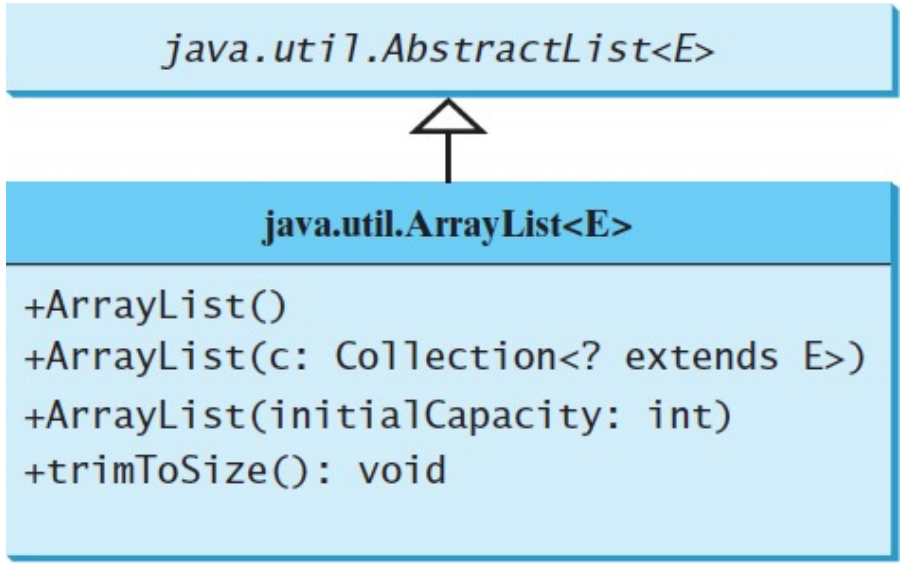


Lớp ArrayList

- ArrayList lưu trữ các phần tử bên trong một mảng
- Kích thước của mảng được điều chỉnh tuỳ 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);
```

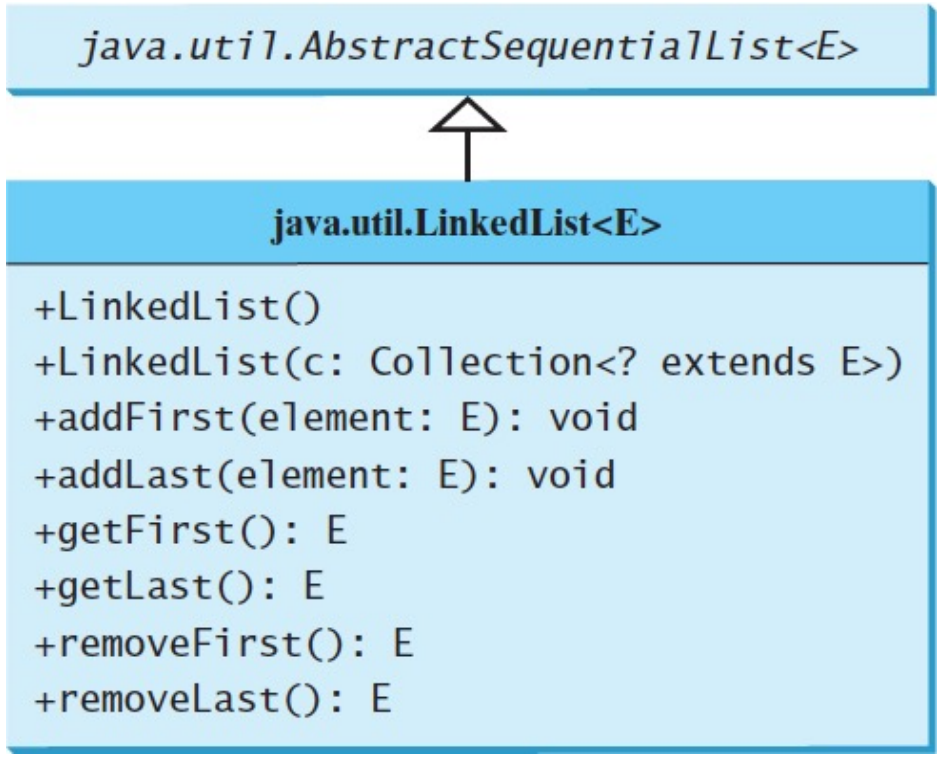


Lớp LinkedList

- 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() + " ");
}
```



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

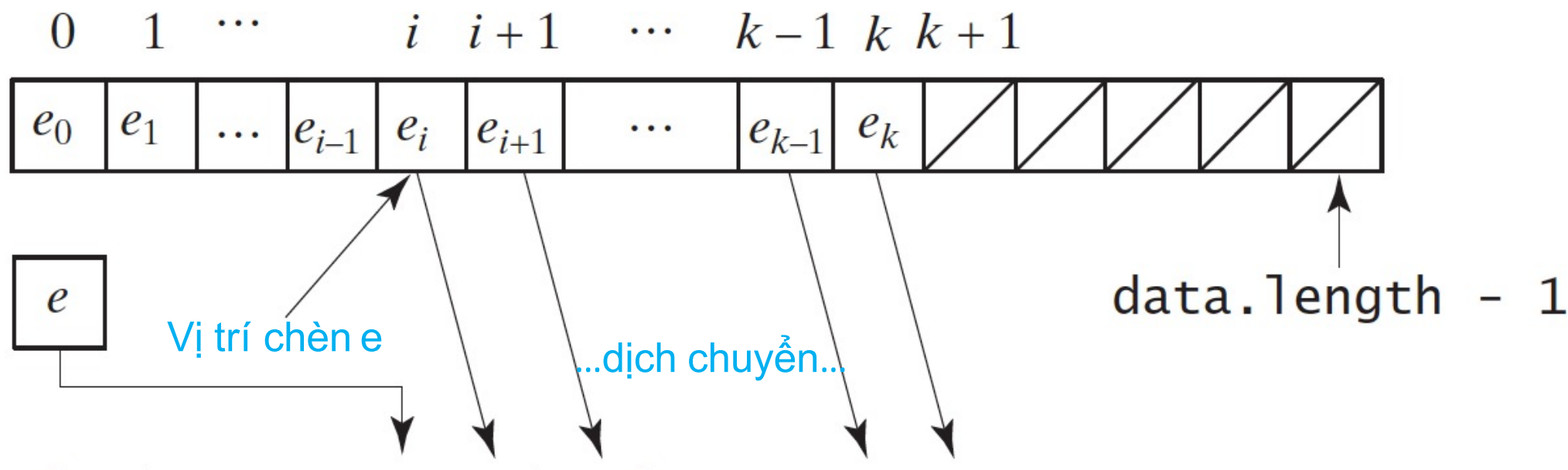
Các thao tác cơ bản của ArrayList

- 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

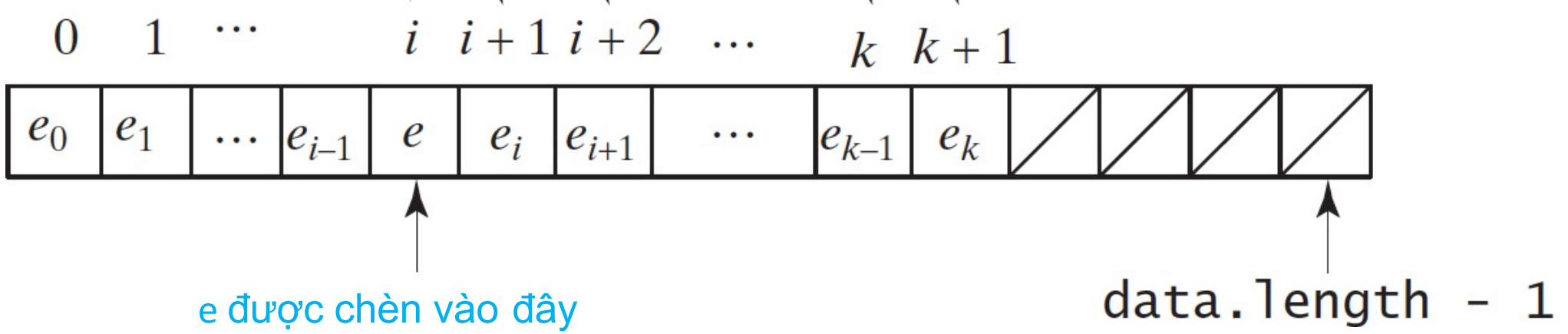
- Sử dụng mảng để lưu trữ các phần tử

Thêm phần tử vào ArrayList

Trước khi chèn *e* vào vị trí *i*

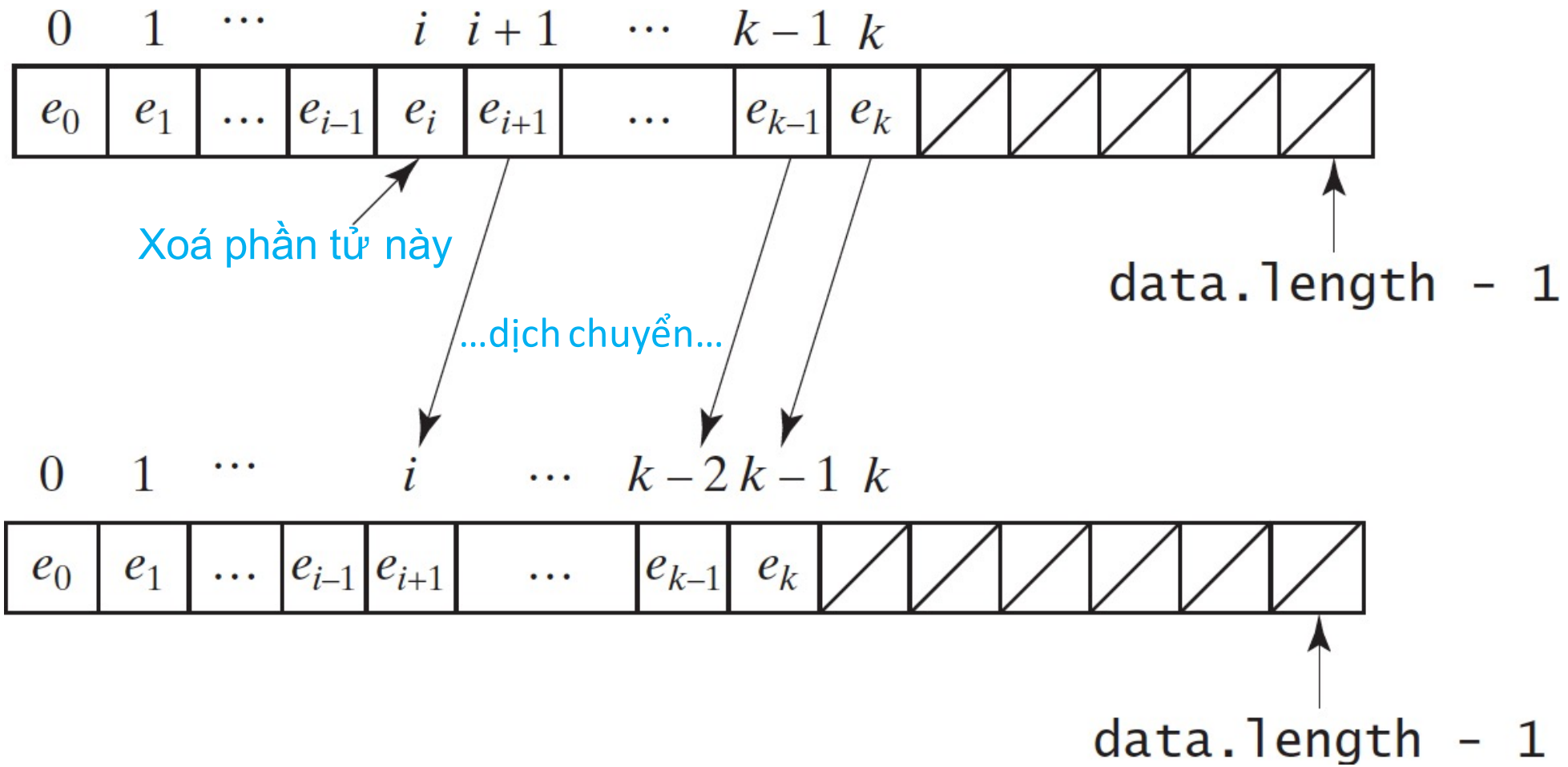


Sau khi chèn *e* vào vị trí *i, kích thước ArrayList tăng thêm 1*



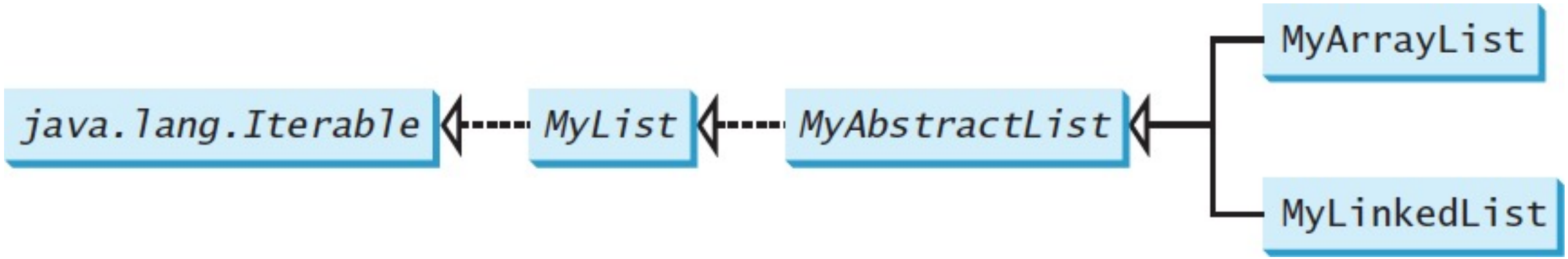
Xoá phần tử khỏi ArrayList

Trước khi xoá phần tử tại vị trí *i*

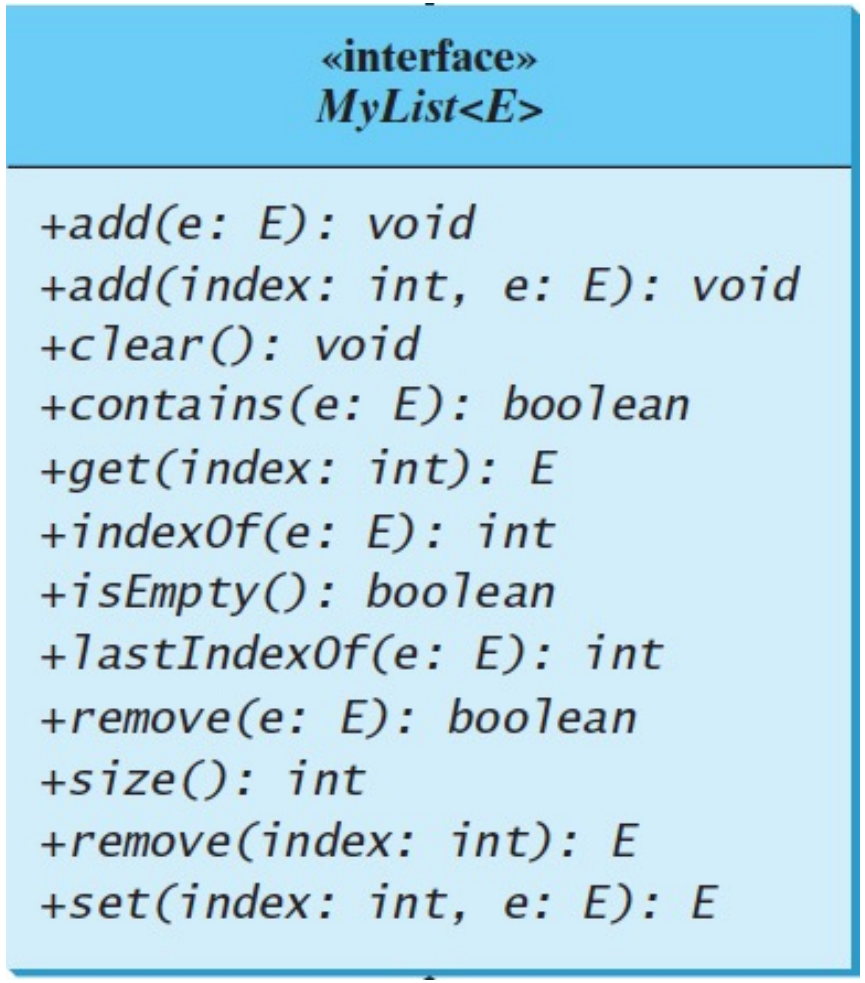


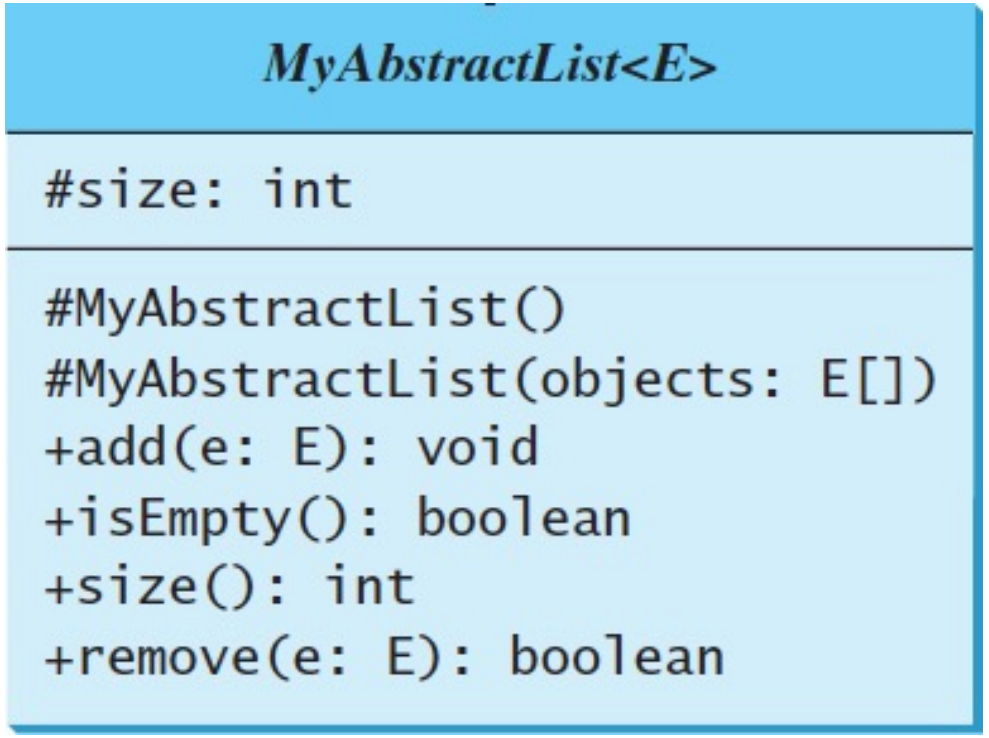
Sau khi xoá phần tử tại vị trí *i*, kích thước của ArrayList giảm xuống 1

Thiết kế

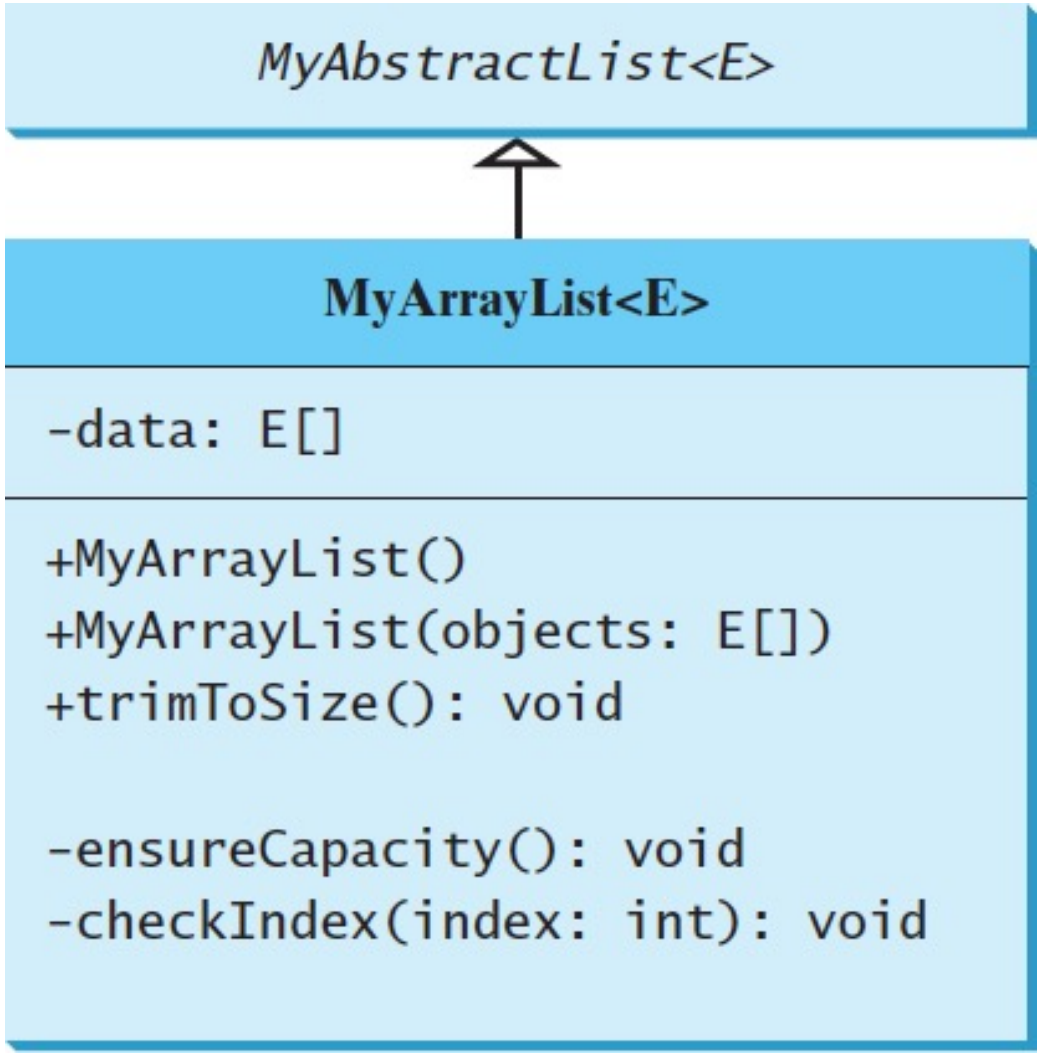

java.lang.Iterable
MyList
MyAbstractList
MyArrayList
MyLinkedList

Interface MyList và MyAbstractList





Lớp MyArrayList



Demo

Triển khai ArrayList

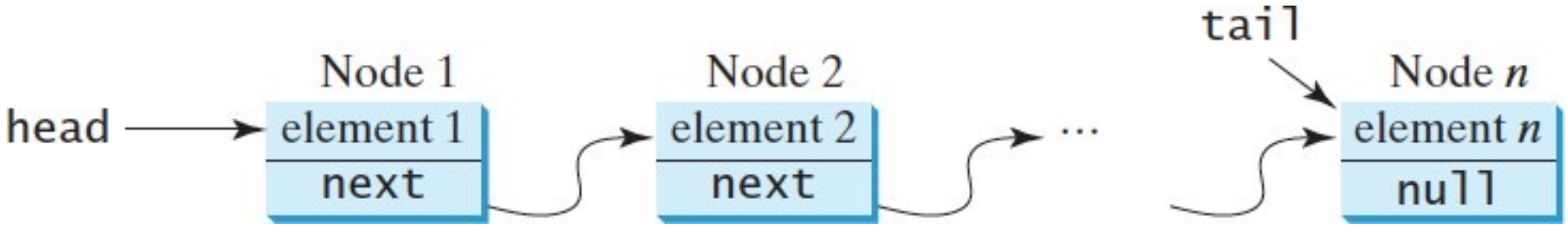
Thảo luận

Triển khai LinkedList

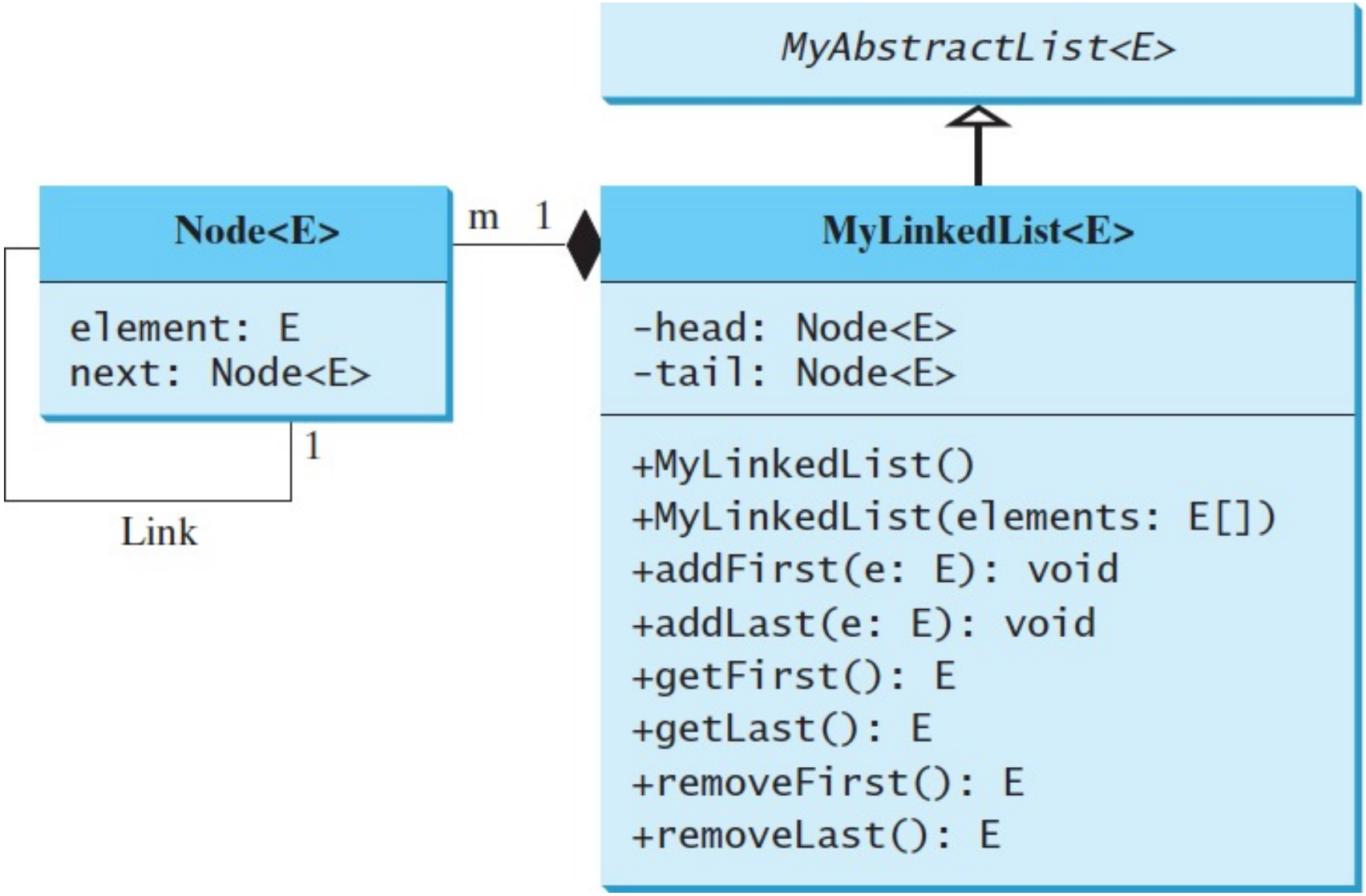
LinkedList

- 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;
    }
}
```

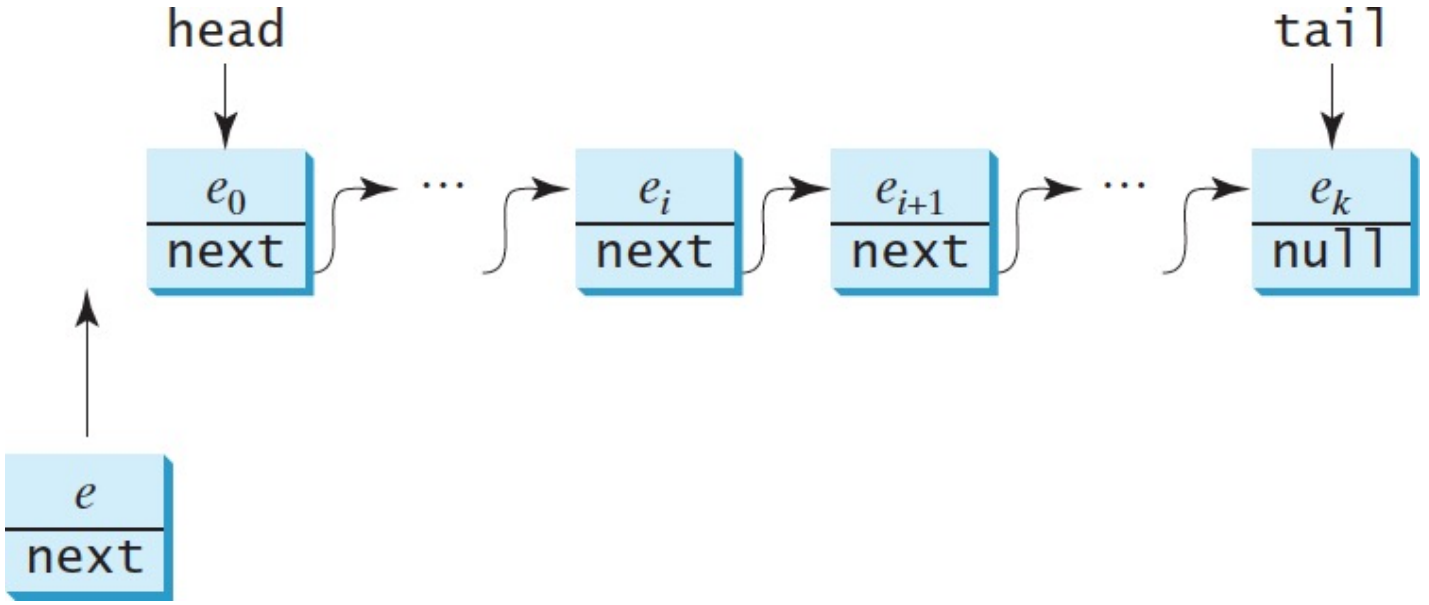


Lớp MyLinkedList

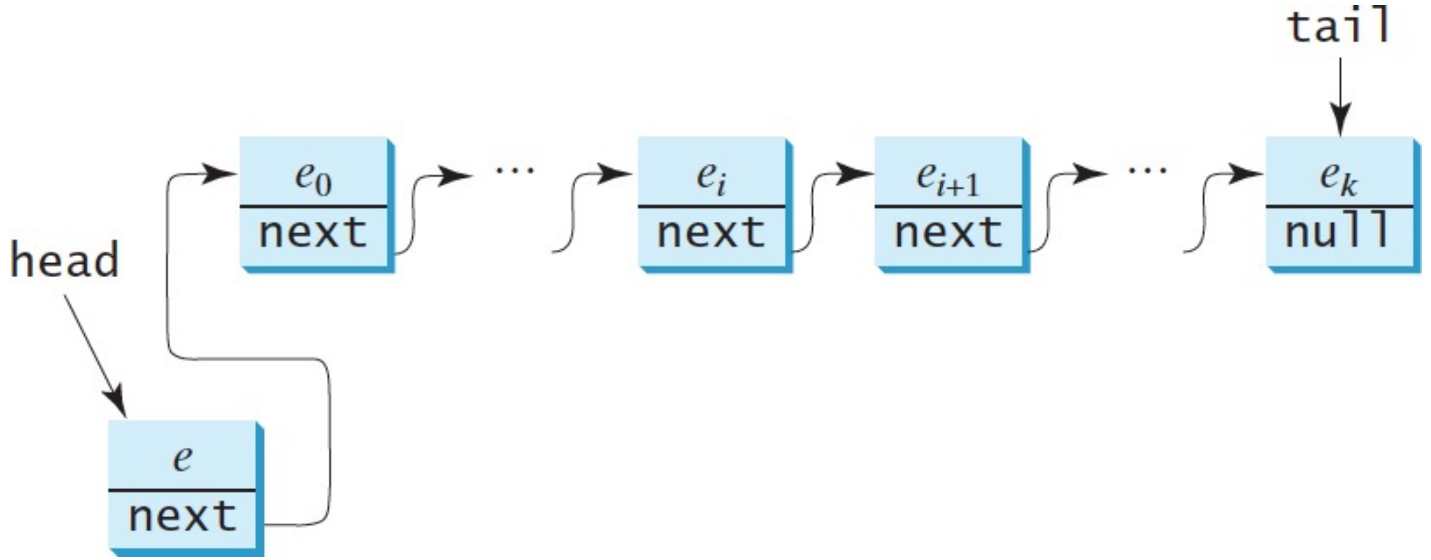


Phương thức addFirst()

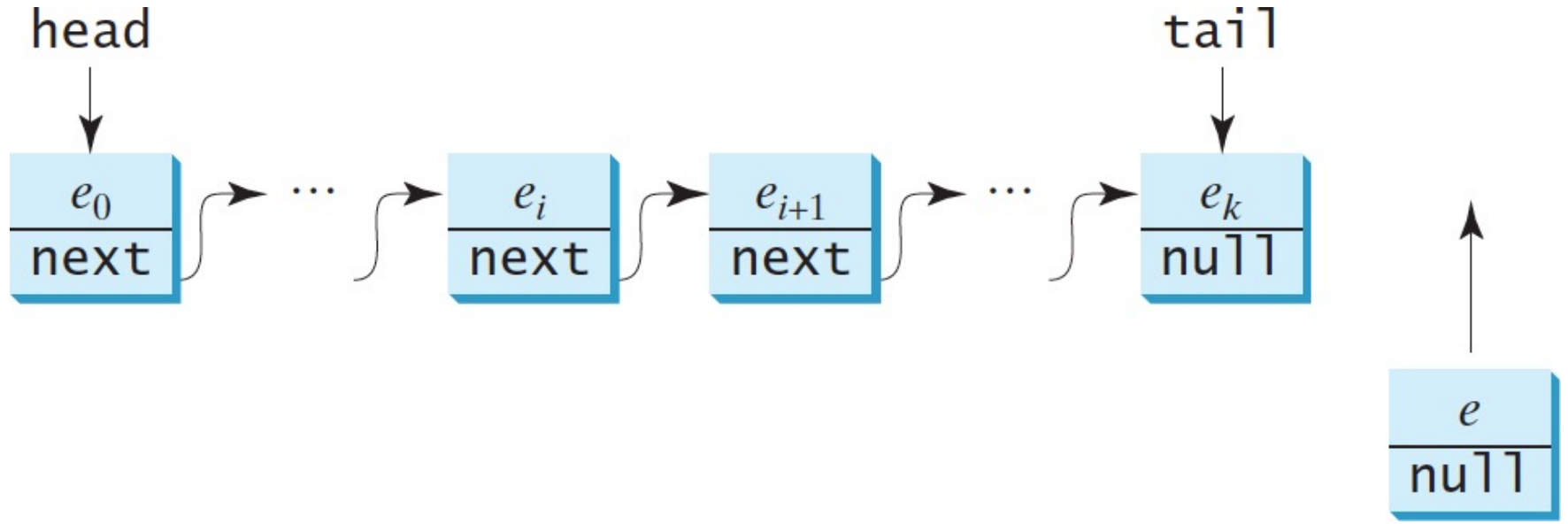
Trước khi chèn phần tử vào đầu



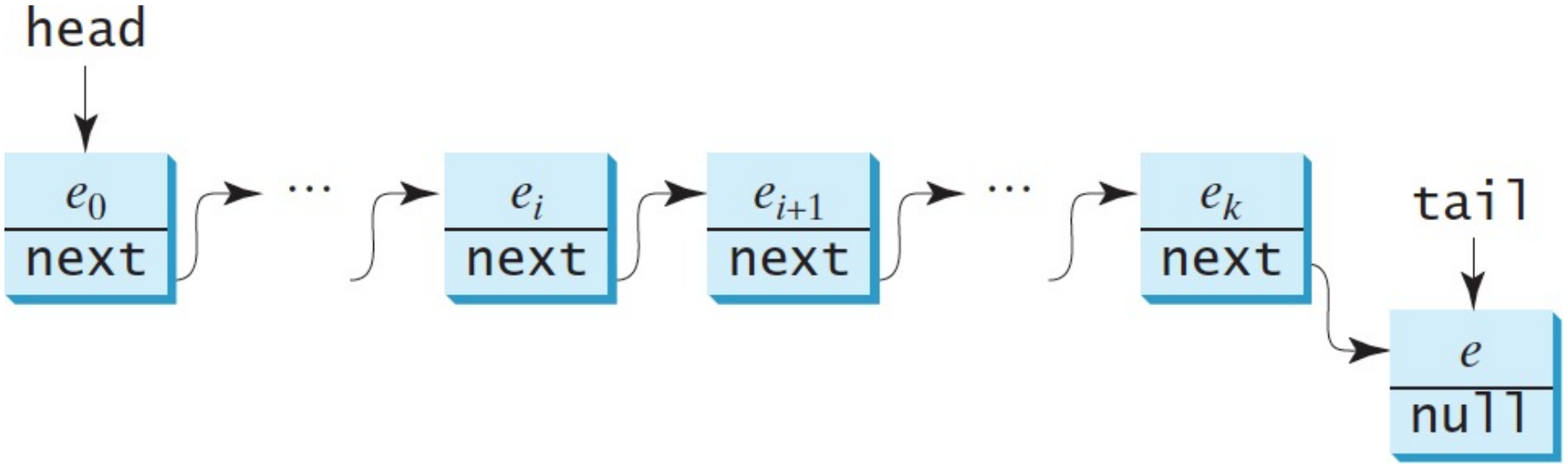
Sau khi được chèn vào đầu



Phương thức addLast()

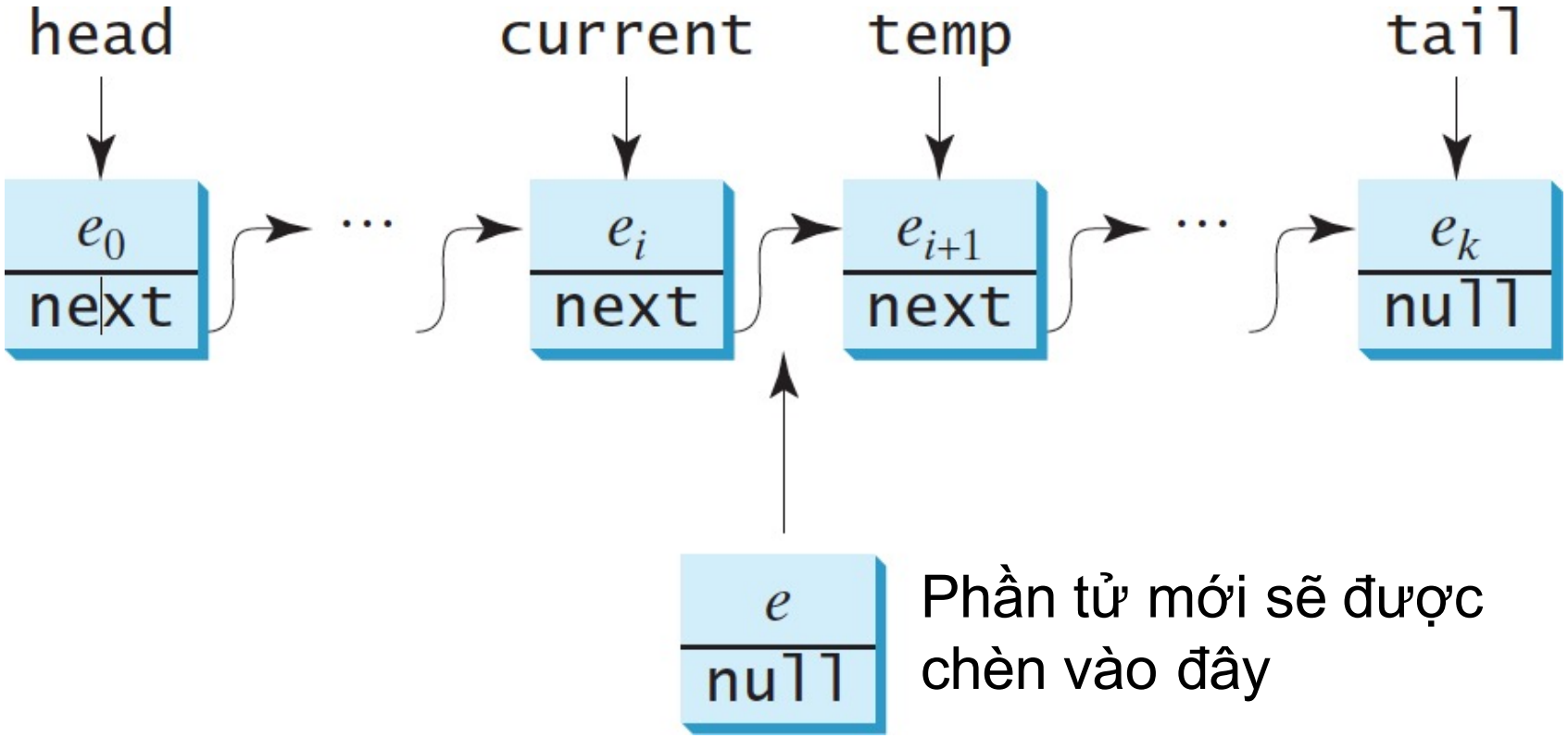


Trước khi chèn phần tử vào đuôi

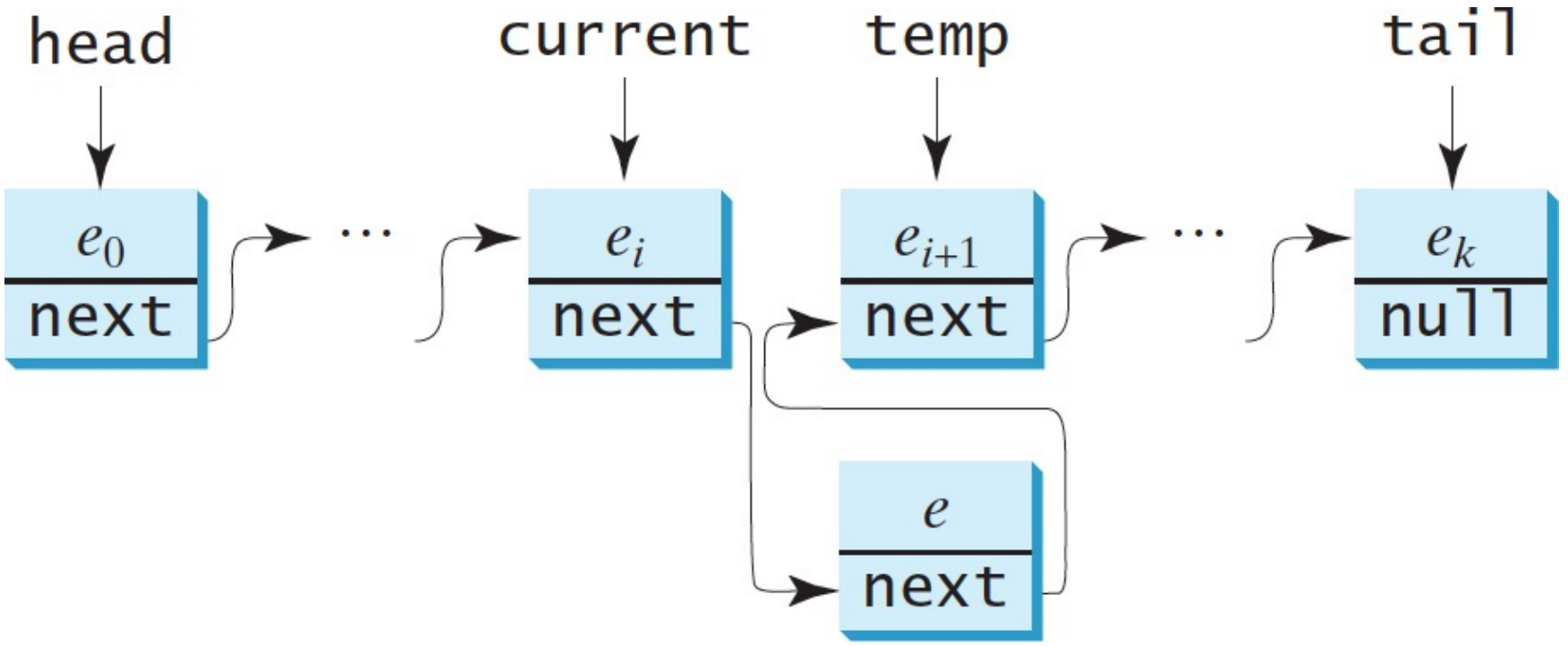


Sau khi chèn phần tử vào đuôi

Phương thức add()

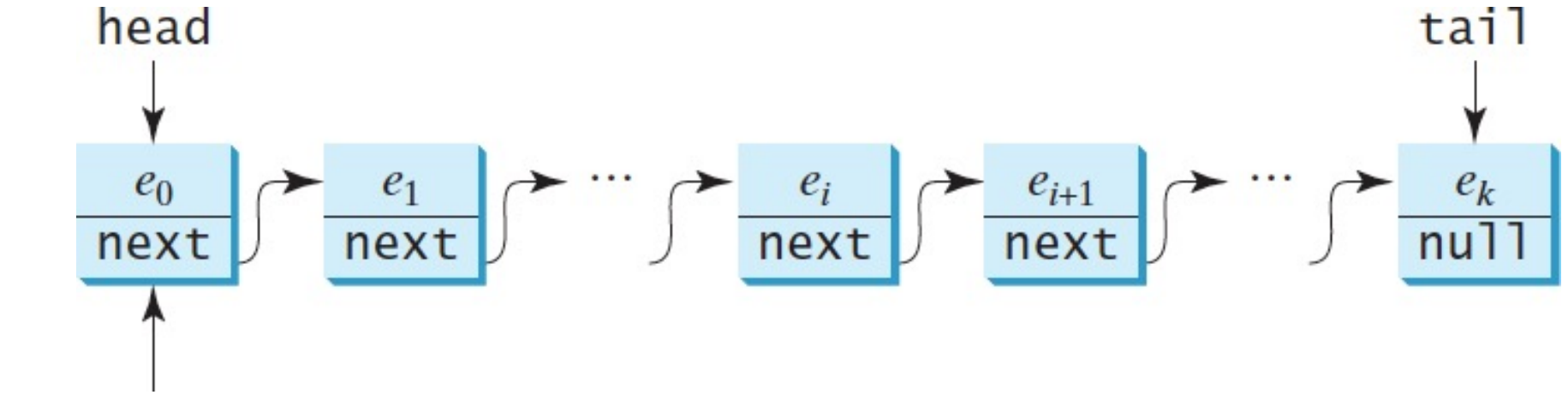


Phương thức add()

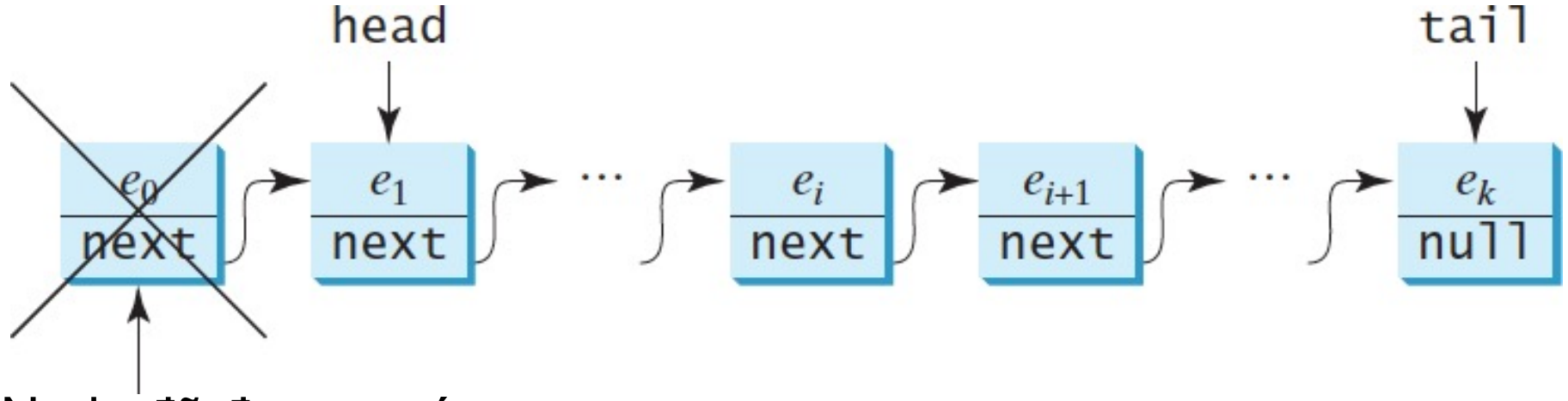


Phần tử mới đã được chèn vào LinkedList

Phương thức removeFirst()

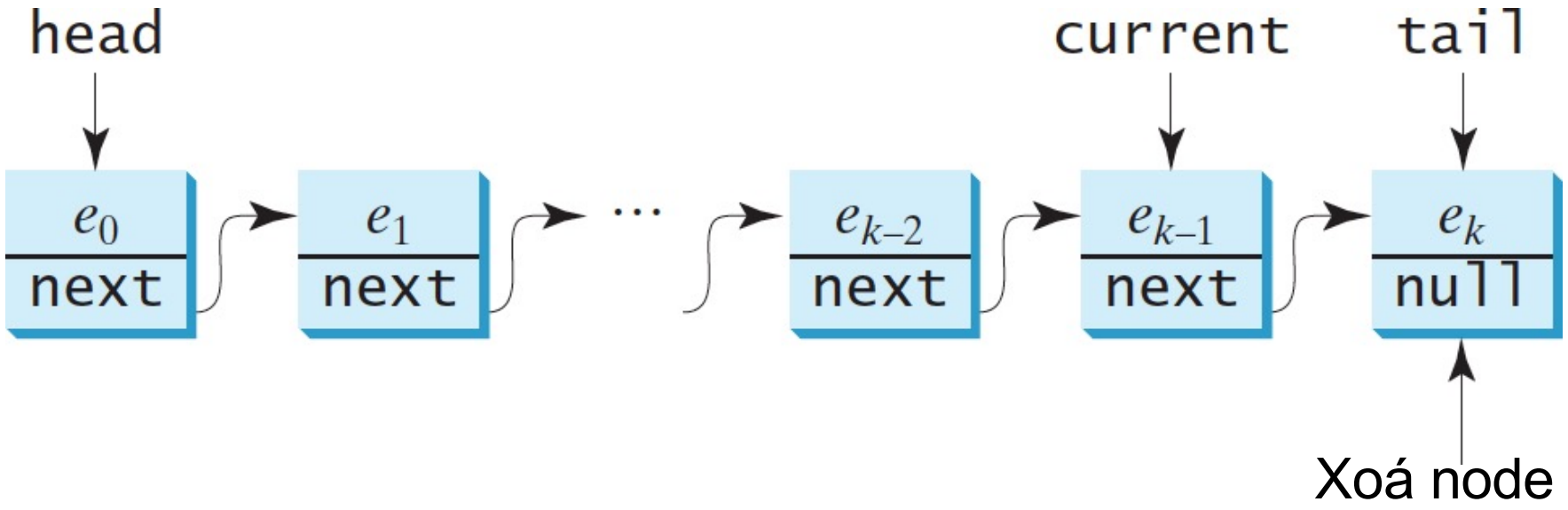


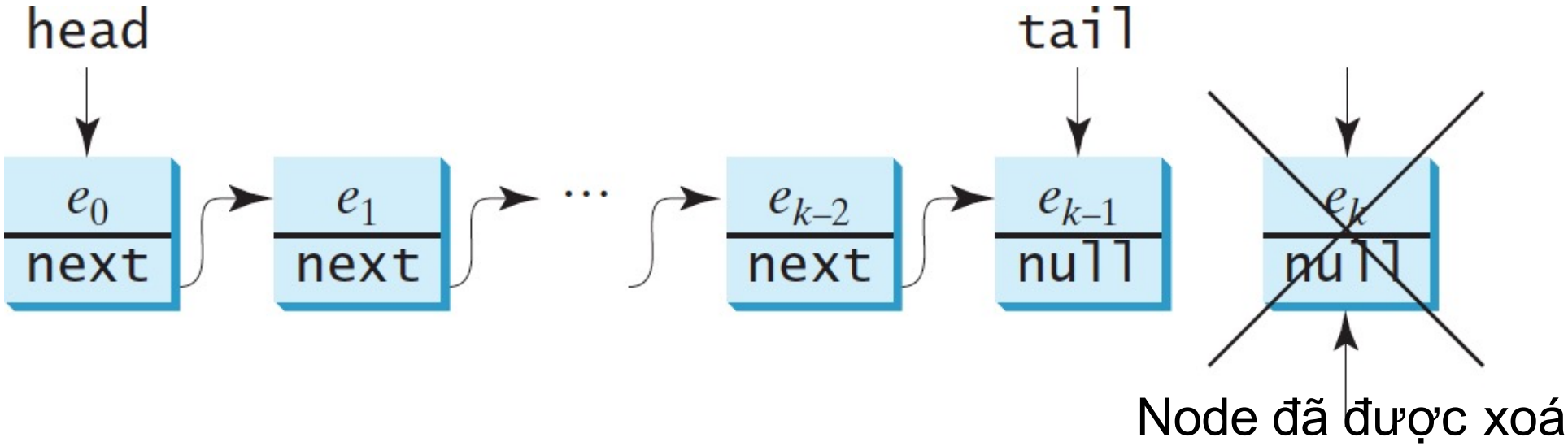
Xoá node này



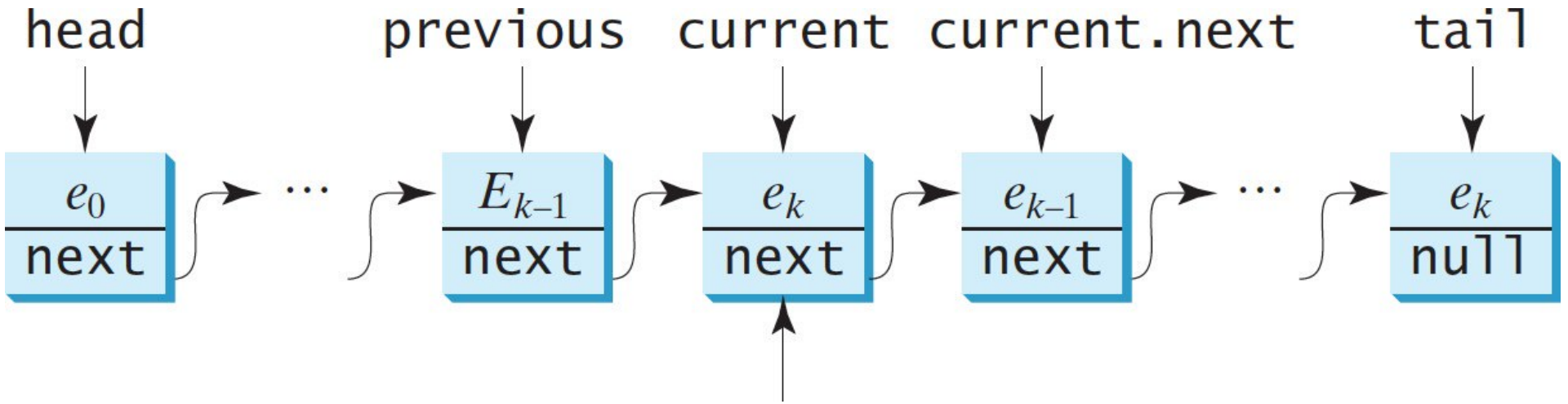
Node đã được xoá

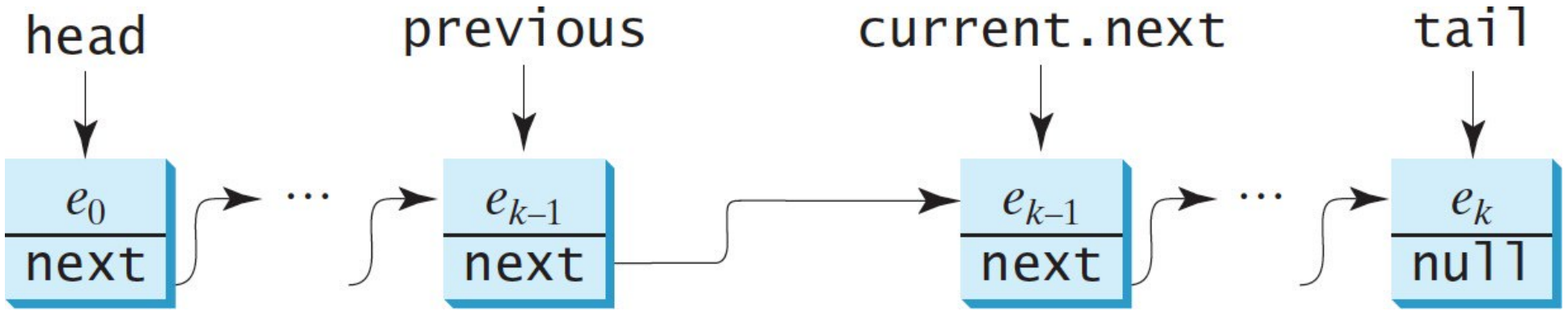
Phương thức removeLast()





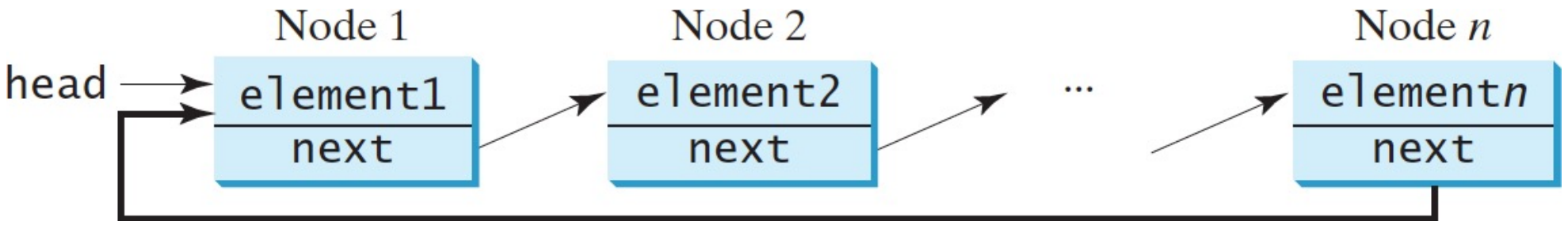
Phương thức remove()





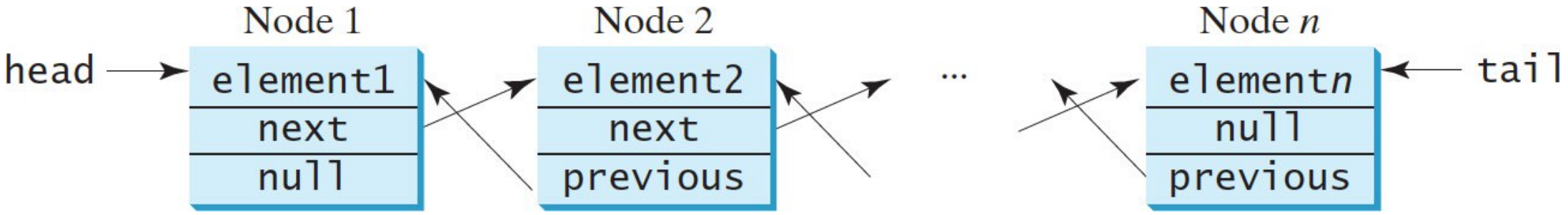
Singly Linked List

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



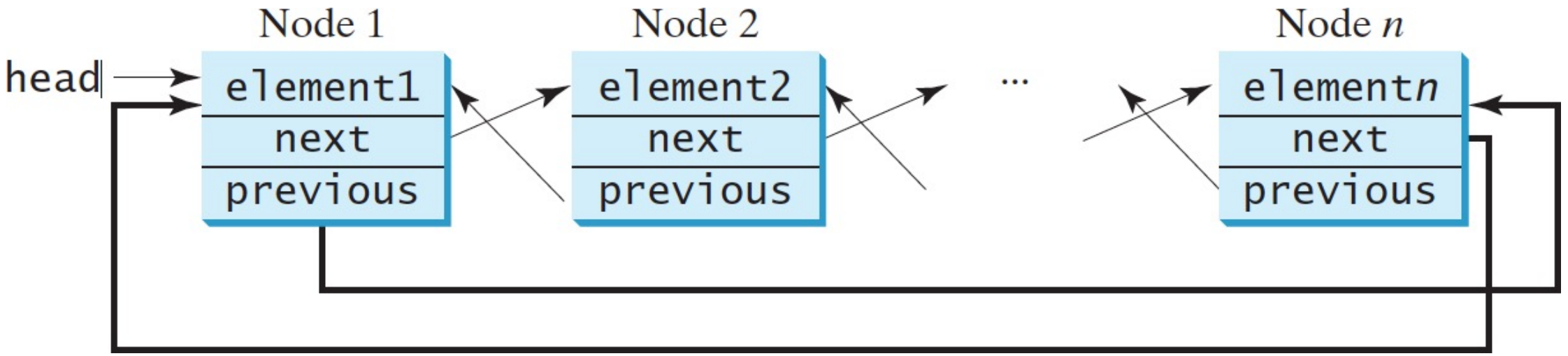
Doubly Linked List

- 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 trỏ đến nhau



Demo

Triển khai LinkedList

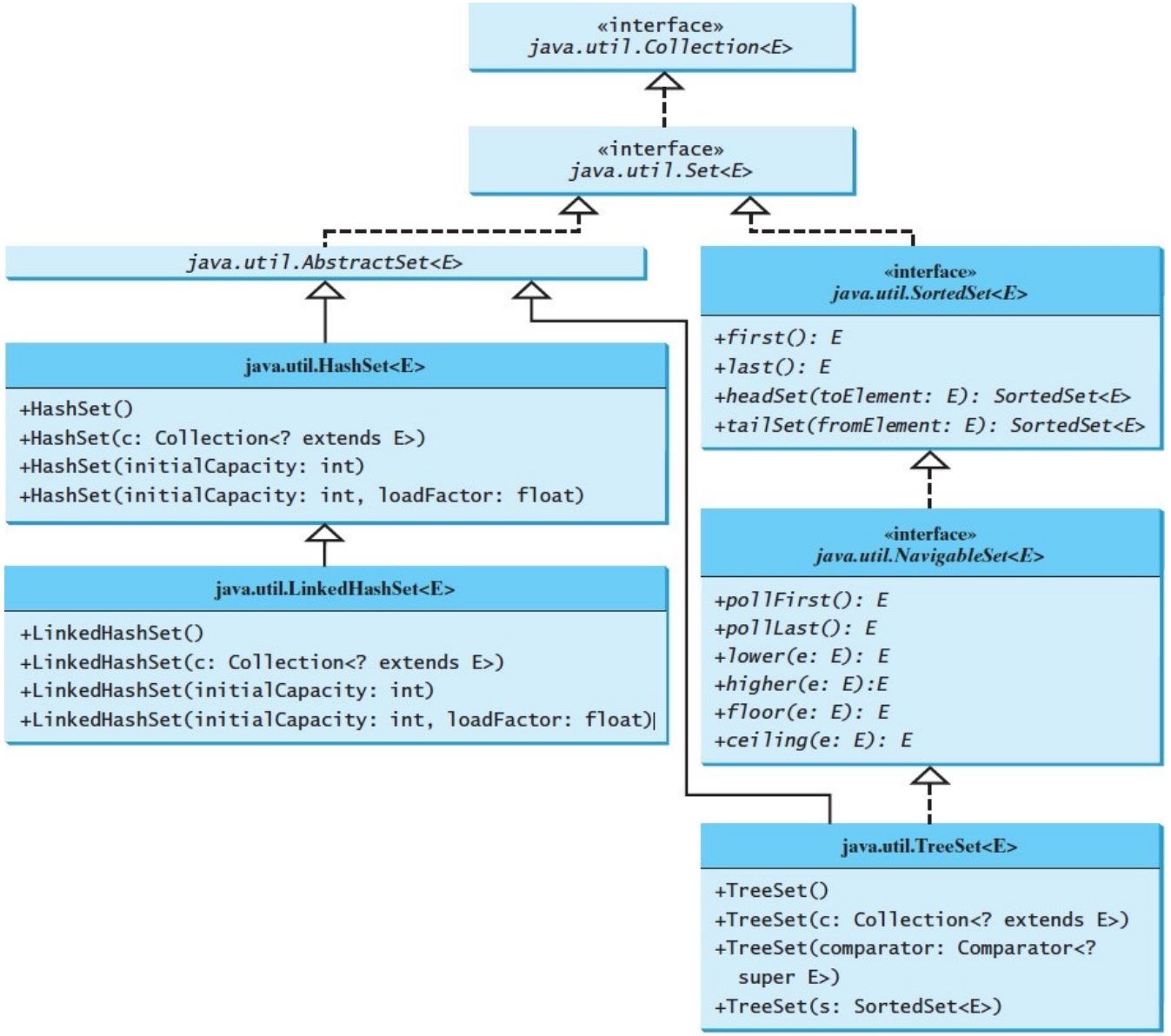
Thảo luận

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

Generic

Generic

- Generic là cơ chế cho phép sử dụng Kiểu dữ liệu như là tham số
- Có thể định nghĩa Lớp và Phương thức với một kiểu dữ liệu generic, sau đó, compiler sẽ thay thế kiểu dữ liệu generic với một kiểu dữ liệu cụ thể
- Ví dụ:
  - Khai báo lớp ArrayList:

```
class ArrayList<E> {

}
```

  - Sử dụng lớp ArrayList:

```
ArrayList<String> strings = new ArrayList<>();
ArrayList<Customer> customers = new ArrayList<>();
```

  - E đại diện cho một kiểu dữ liệu generic
  - String và Customer là các kiểu dữ liệu cụ thể

Lợi ích của generic

- Giúp phát hiện lỗi ngay tại thời điểm biên dịch, thay vì tại thời điểm thực thi nếu không dùng generic
- Generic cho phép quy định các kiểu dữ liệu được phép sử dụng ở trong một lớp hoặc phương thức
- Nếu kiểu dữ liệu không phù hợp được sử dụng thì sẽ được phát hiện

Lợi ích của generic: Ví dụ

Không sử dụng Generic

```
ArrayList numbers = new ArrayList();
numbers.add(1);
numbers.add("a");

int total = 0;
for (int i = 0; i < numbers.size(); i++){
    total += (int)numbers.get(i);
}
System.out.println("Total: " + total);
```

Cần ép kiểu

Có lỗi xảy ra tại thời điểm thực thi bởi vì không thể ép kiểu từ String sang int

Có sử dụng Generic

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add("a");

int total = 0;
for (int i = 0; i < numbers.size(); i++){  total
    += numbers.get(i);
}
System.out.println("Total: " + total);
```

Không cần ép kiểu

Thông báo lỗi được hiển thị ngay tại thời điểm compile. String không được add vào ArrayList Integer

So sánh lớp ArrayList có generic và không

java.util.ArrayList
+ArrayList() +add(o: Object): void +add(index: int, o: Object): void +clear(): void +contains(o: Object): boolean +get(index:int): Object +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: Object): Object

java.util.ArrayList<E>
+ArrayList() +add(o: E): void +add(index: int, o: E): void +clear(): void +contains(o: Object): boolean +get(index:int): E +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: E): E

Khai báo lớp và interface generic

- Cú pháp:

```
class ClassName<T> {

}
interface InterfaceName<T> {

}
```

  - Trong đó:
    - ClassName và InterfaceName là tên của lớp và interface
    - T là kiểu dữ liệu Generic. Có thể dùng bất cứ chữ cái nào.

Khai báo lớp generic: Ví dụ

```
class GenericArrayList<T> {
    private static final int INITIAL_SIZE = 16;
    private  T[] elements;
    private int count = 0;

    public GenericArrayList(){
        this.elements = (T[])new Object[INITIAL_SIZE];
    }

            public void add(T element){
                //TODO: Ensure capacity
        this.elements[count++] = element;
    }
    ....
}
```

Generic với nhiều kiểu dữ liệu

- Có thể định nghĩa lớp và interface với nhiều kiểu dữ liệu generic
- Các kiểu dữ liệu cách nhau bởi dấu phẩy (,)
- Ví dụ:

```
class GenericMap<K, V>{

}
```

Trong đó K và V là các kiểu dữ liệu generic

Phương thức generic

- Có thể sử dụng generic cho các phương thức static
- Cú pháp:

```
static <T> data_type MethodName(){

}
```

Trong đó:

- T là kiểu dữ liệu generic
- data_type là kiểu dữ liệu trả về
- MethodName là tên của phương thức

Phương thức generic: Ví dụ

```
public class GenericMethodDemo {
    public static void main(String[] args ) {  Integer[]
        integers = {1, 2, 3, 4, 5};
        String[] strings = {"London", "Paris", "New York", "Austin"};

        GenericMethodDemo.<Integer>print(integers);
        GenericMethodDemo.<String>print(strings);
    }

    public static <E> void print(E[] list) {
        for (int i = 0; i < list.length; i++){
            System.out.print(list[i] + " ");
        }
        System.out.println();
    }
}
```

Có thể gọi rút gọn, không cần chỉ rõ kiểu:

GenericMethodDemo.print(integers);
GenericMethodDemo.print(strings);

Ràng buộc cho kiểu Generic

- Có thể quy định kiểu generic là subtype của một kiểu dữ liệu khác
- Ràng buộc này được gọi là Bounded Type
- Ví dụ:

```
public class BoundedTypeDemo {
    public static void main(String[] args ) {  Rectangle
        rectangle = new Rectangle(2, 2);  Circle circle =
        new Circle(2);

        System.out.println("Same area? " + equalArea(rectangle, circle));
    }

    public static <E extends GeometricObject> boolean equalArea(E object1, E object2) {
        return object1.getArea() == object2.getArea();
    }
}
```

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: *DSA Stack Queue*