Bài 11

Collections

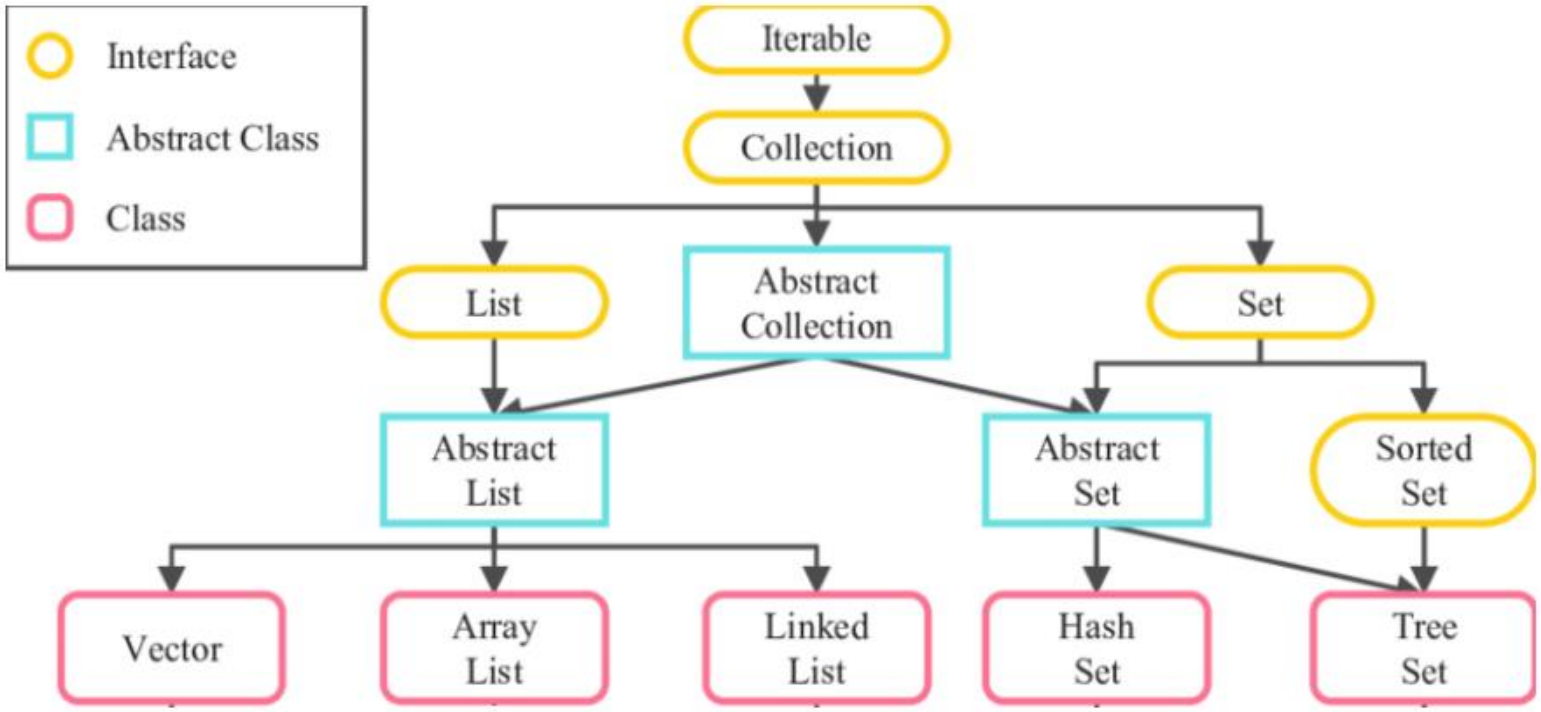
Tổng quan

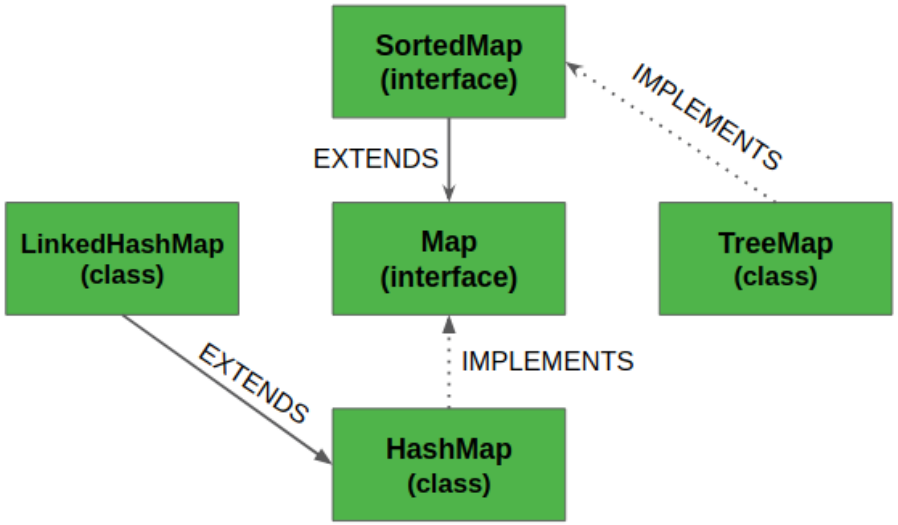
- Là cấu trúc dữ liệu cho phép lưu trữ và thao tác với tập hợp các đối tượng
  - String, Integer, Item, Employee
  - int, float, double, char

- Collections được sử dụng trong hầu hết các ứng dụng Java. Là một trong những phần chính của Java.

- Thành phần trong collections
  - Interfaces: Chứa các hàm trừu tượng
  - Implementation classes: Chứa phần thực thi của các hàm trừu tượng
  - Algorithms: Thuật toán có sẵn hỗ trợ sorting, shuffle

Collection architecture



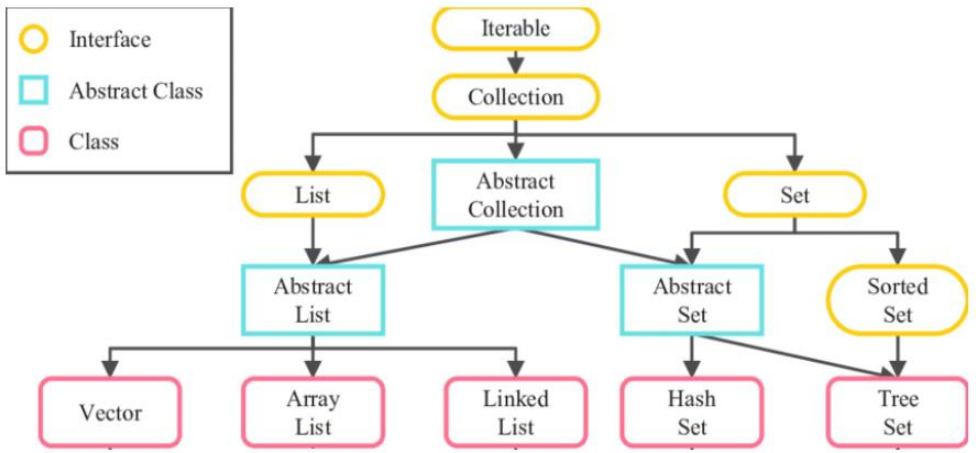


MAP Hierarchy in Java

Collection architecture

Interface

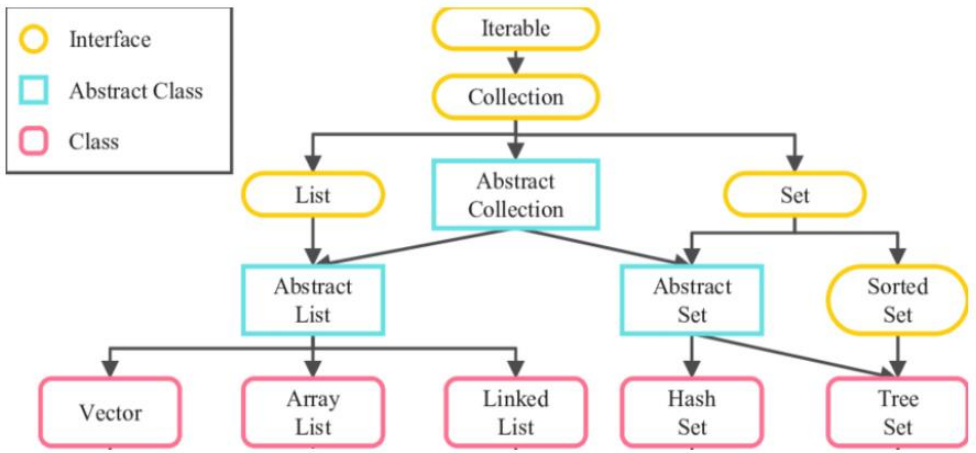
- Là tập các kiểu dữ liệu trừu tượng như abstract class, interface
- Chứa các phương thức trừu tượng trong collection size(), iterator(), add(), remove()
- Sử dụng strategy, factory design pattern
- Một số interface quan trọng như
  - List
  - Set
  - Map



Collection architecture

Implementation classes

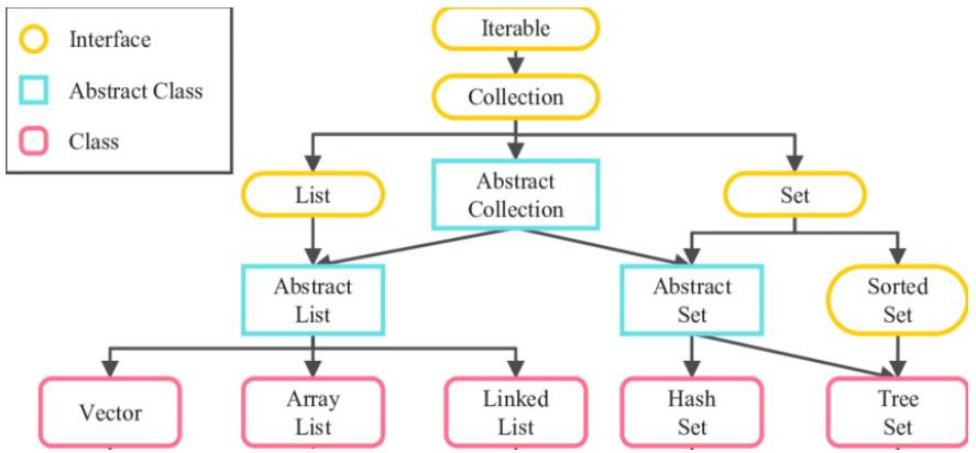
- Là tập hợp các lớp thực thi cho collection interface
- Implement chi tiết thuật toán trong các phương thức của mỗi cấu trúc dữ liệu
- Sử dụng factory design pattern
- Một số lớp thực thi quan trọng như
  - ArrayList, LinkedList
  - HashMap, TreeMap, LinkedHashMap
  - HashSet, TreeSet, LinkedHashSet



Collection benefits

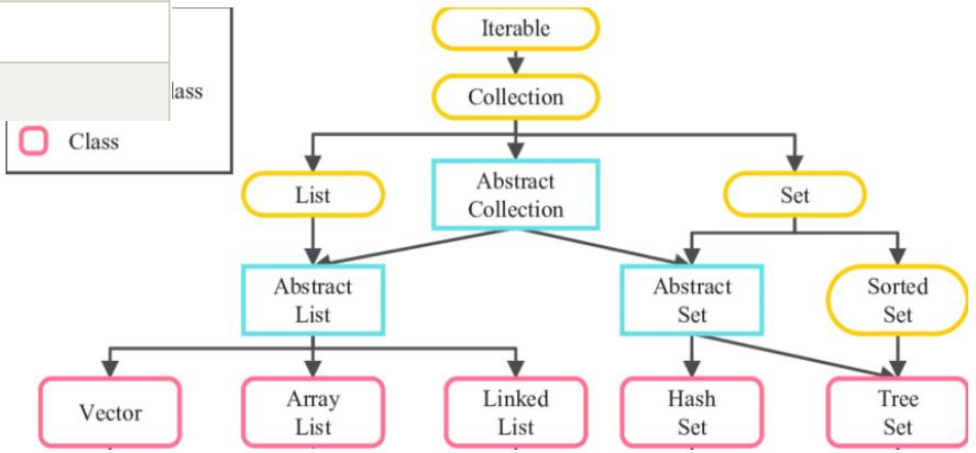
- Hỗ trợ tập hợp nhiều phương thức xử lý hữu ích. Tập trung chức năng của dự án
- Tăng hiệu suất, chất lượng ứng dụng với những phương thức được hỗ trợ bởi ngôn ngữ
- Gọi thông qua các API. Việc còn lại ngôn ngữ sẽ tự xử lý

```
List<Item> items = new ArrayList<>();

Item item1 = new Item(1, 10, "A10", 100);
Item item2 = new Item(2, 20, "A20", 200);
Item item3 = new Item(3, 30, "A30", 300);
Item item4 = new Item(1, 12, "A12", 120);

items.add(item1);
items.add(item2);
items.add(item3);
items.add(item4);
```



Collection helpful methods

1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.



LIST INTERFACE

LIST

Interface

- List là một interface chứa tập hợp các đối tượng có thể lưu trữ các giá trị trùng nhau.
- Cho phép lưu trữ tập hợp phần tử với dynamic size
- List interface được thực thi bởi ArrayList, LinkedList, Vector, Stack.
- Generic Type: Từ Java 1.5 xuất hiện khái niệm Generics, chúng ta có thể khai báo kiểu dữ liệu trực tiếp cho List

JDK 1.4

```
List listA = new ArrayList();
List listB = new LinkedList();
```

JDK 1.5 1.7

```
List<Object> listA = new ArrayList<>();
List<Object> listB = new LinkedList<>();
```

JDK 1.5 1.7

```
List<T> listA = new ArrayList<>();
List<T> listB = new LinkedList<>();
```

ARRAYLIST

Implementation classes

- Lớp thực thi từ interface List

```
List<T> listA = new ArrayList<>();
List<T> listB = new LinkedList<>();
```

- Sử dụng bản chất của cấu trúc dữ liệu mảng với số lượng phần tử có thể thay đổi được
- Không bắt buộc khai báo trước số lượng phần tử như mảng
- Có thể khai báo trước sức chứa(opacity) cho lần khởi tạo mảng đầu tiên của ArrayList
- Phương thức size, isEmpty, get, set, iterator được thực thi với thời gian cố định O(1)
- Phương thức add, remove sẽ phụ thuộc vào vị trí xử lý. Cách hoạt động tương tự với mảng. Và sẽ có sự nhanh, chậm hơn khi so sánh với cấu trúc dữ liệu LinkedList

ARRAYLIST

Implementation classes

Khai báo và khởi tạo

- ArrayList()
- ArrayList(Collection collection)
- ArrayList(int capacity)

Phương thức thường dùng

- add
- remove
- removeIf
- get
- size
- contains

```
List<T> listA = new ArrayList<>();
List<T> listB = new LinkedList<>();
```

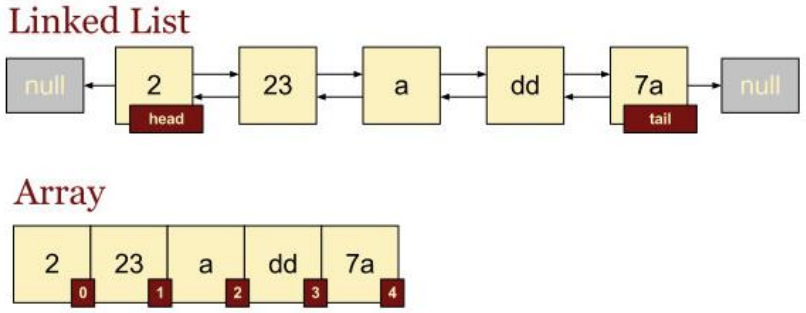
ARRAYLIST

DEMO

1. List ArrayList and raw type problems
2. List ArrayList with Single Object Type(SOT) – Integer, String
3. Basic operations SOT: construct, add, get, set, iterate, for each, remove, contains
4. Basic operations SOT: add all, removeAll, retainAll
5. Basic operation with own Object type such as Item
    1. Item: storeId, itemId, name, price
    2. Add with mock data
    3. Contains, remove and internal build with equals method
    4. Sorting

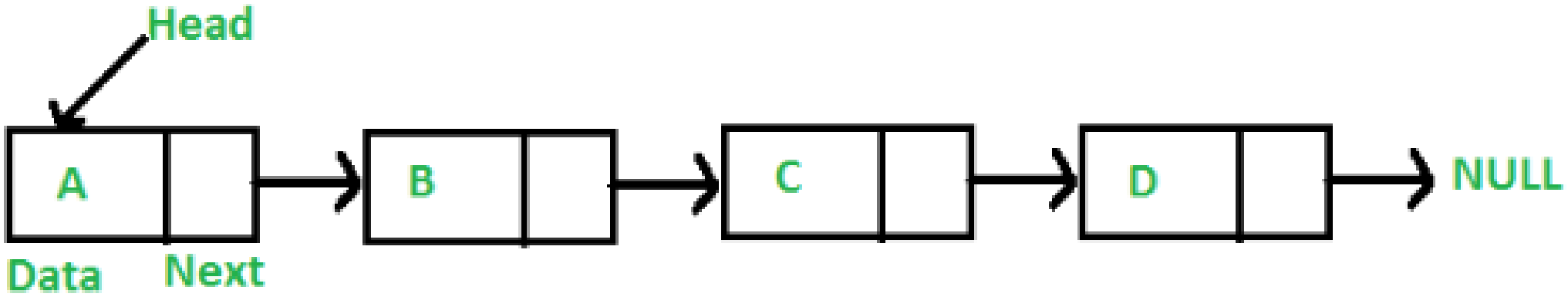
LINKEDLIST

- Là lớp thực thi của interface List
- Sử dụng cấu trúc danh sách liên kết đôi để lưu trữ các phần tử
- Thực thi từ List và Deque hỗ trợ truy cập phần tử từ 2 đầu của danh sách
- Các phương thức xử lý hầu hết tương tự với ArrayList
- Tốc độ xử lý các phương thức get, set, add, remove khác với ArrayList vì code base bên trong ArrayList(sử dụng mảng) và LinkedList(sử dụng danh sách liên kết đôi)



Data structure

- Singly Linked List

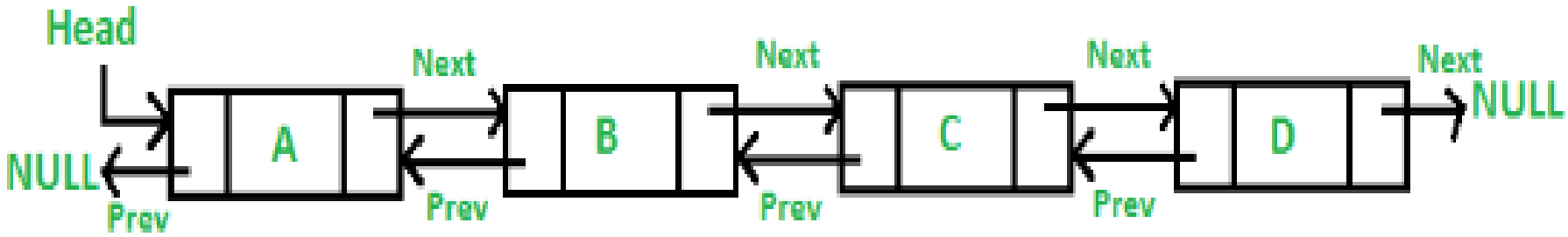



```
struct Node {
    Data data;
    Node* next
}
```

```
class Node {
    Data data;
    Node next
}
```

Data structure

- Doubly Linked List

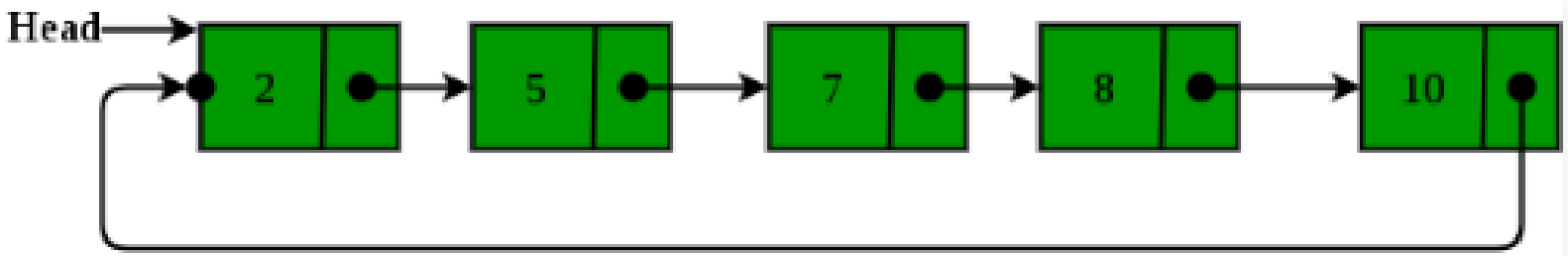


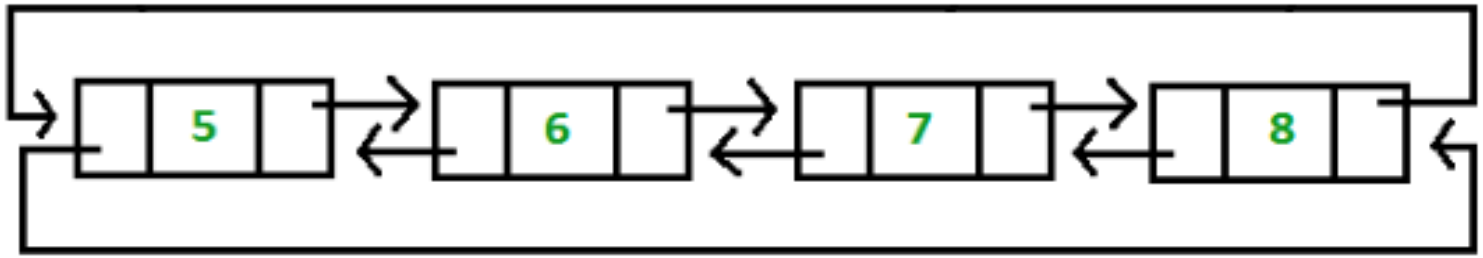

```
struct Node {
    Data data;
    Node* left;
    Node* right;
}
```

```
class Node {
    Data data;
    Node left;
    Node right;
}
```

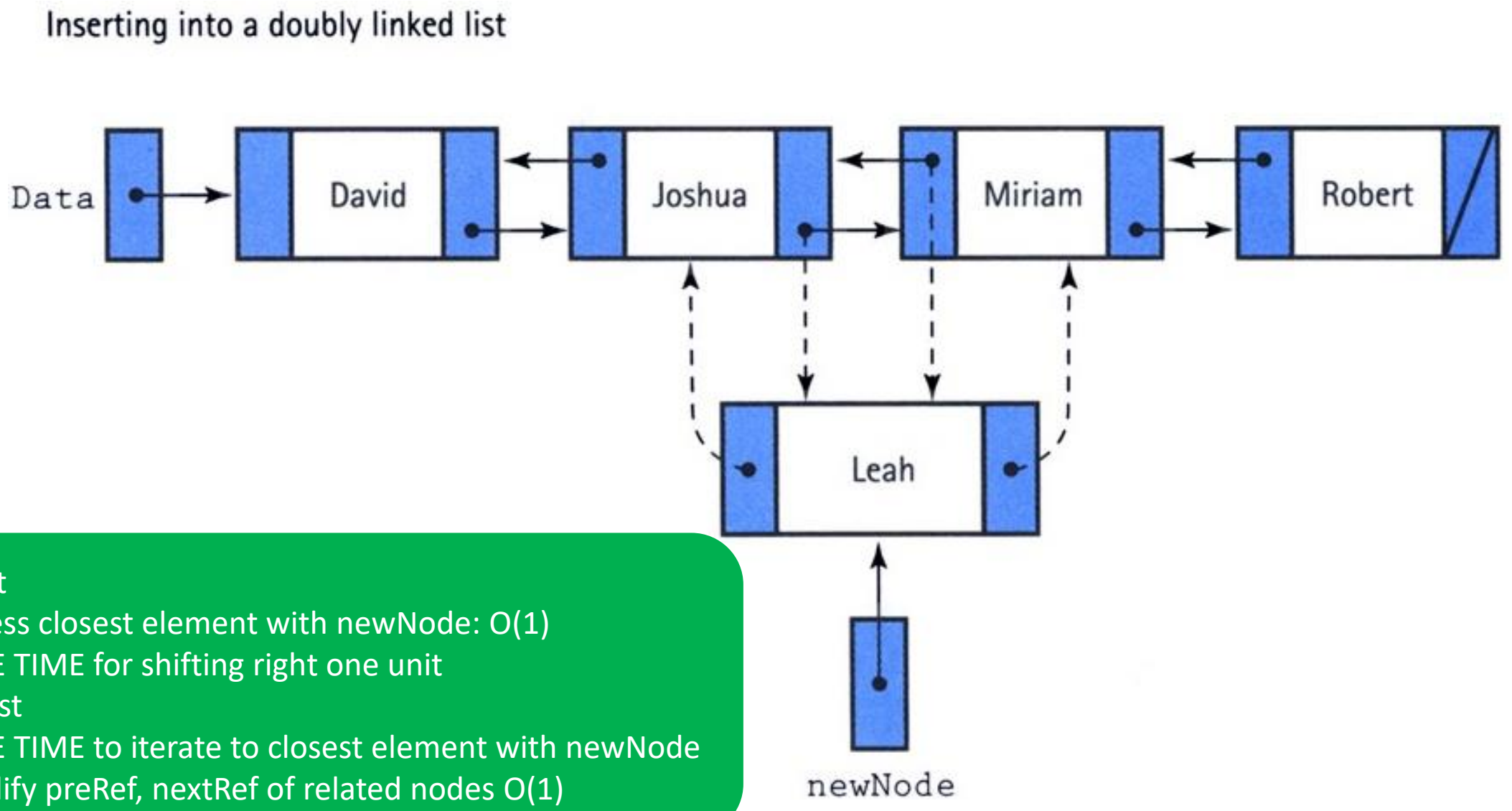
Data structure

- Circular Linked List





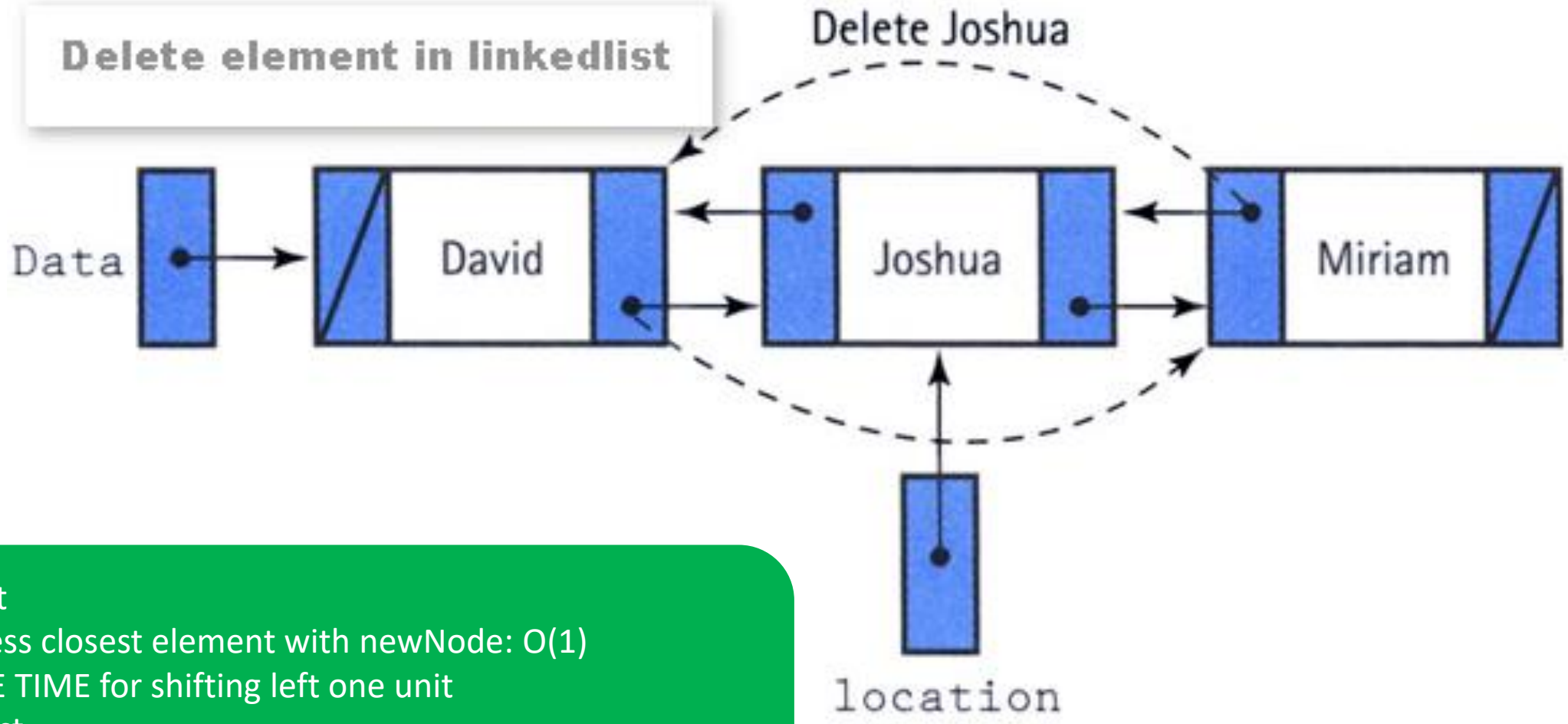
LINKEDLIST



ArrayList
1. Access closest element with newNode: O(1)
2. TAKE TIME for shifting right one unit

LinkedList
1. TAKE TIME to iterate to closest element with newNode
2. Modify preRef, nextRef of related nodes O(1)

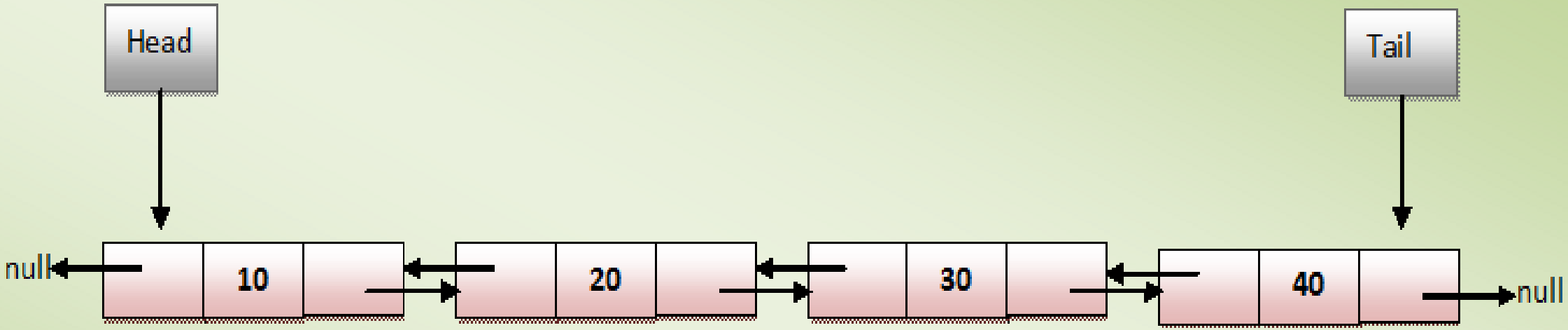
LINKEDLIST



ArrayList
1. Access closest element with newNode: O(1)
2. TAKE TIME for shifting left one unit

LinkedList
1. TAKE TIME to iterate to closest element with newNode
2. Modify preRef, nextRef of related nodes O(1)

LINKEDLIST



Add, Remove

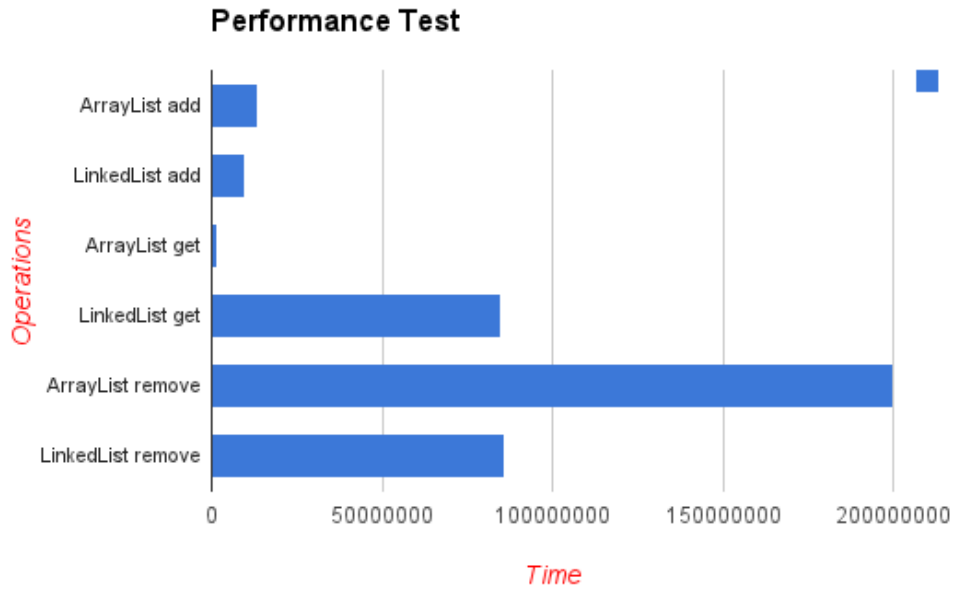
Get, Set

Insertions and Removals in LinkedList are faster than ArrayList. Because, You don't need to resize LinkedList after each insertions and removals.

Retrieval operations in LinkedList are slow compared to ArrayList. Because, it needs traversing from the beginning or from end to reach the element.

LINKEDLIST AND ARRAYLIST

Operation \ List	LinkedList	ArrayList
get(int index)	O(n/4) average	O(1)
add(E element)	O(1)	O(1) amortized O(n) worst-case
add(int index, E element)	O(n/4) average O(1) if index is 0	O(n/2) average
remove	O(n/4) average	O(n/2) average
Iterator.remove()	O(1)	O(n/2) average
ListIterator.add(E element)	O(1)	O(n/2) average



MAP INTERFACE

Tổng quan về MAP

❖ Khái quát

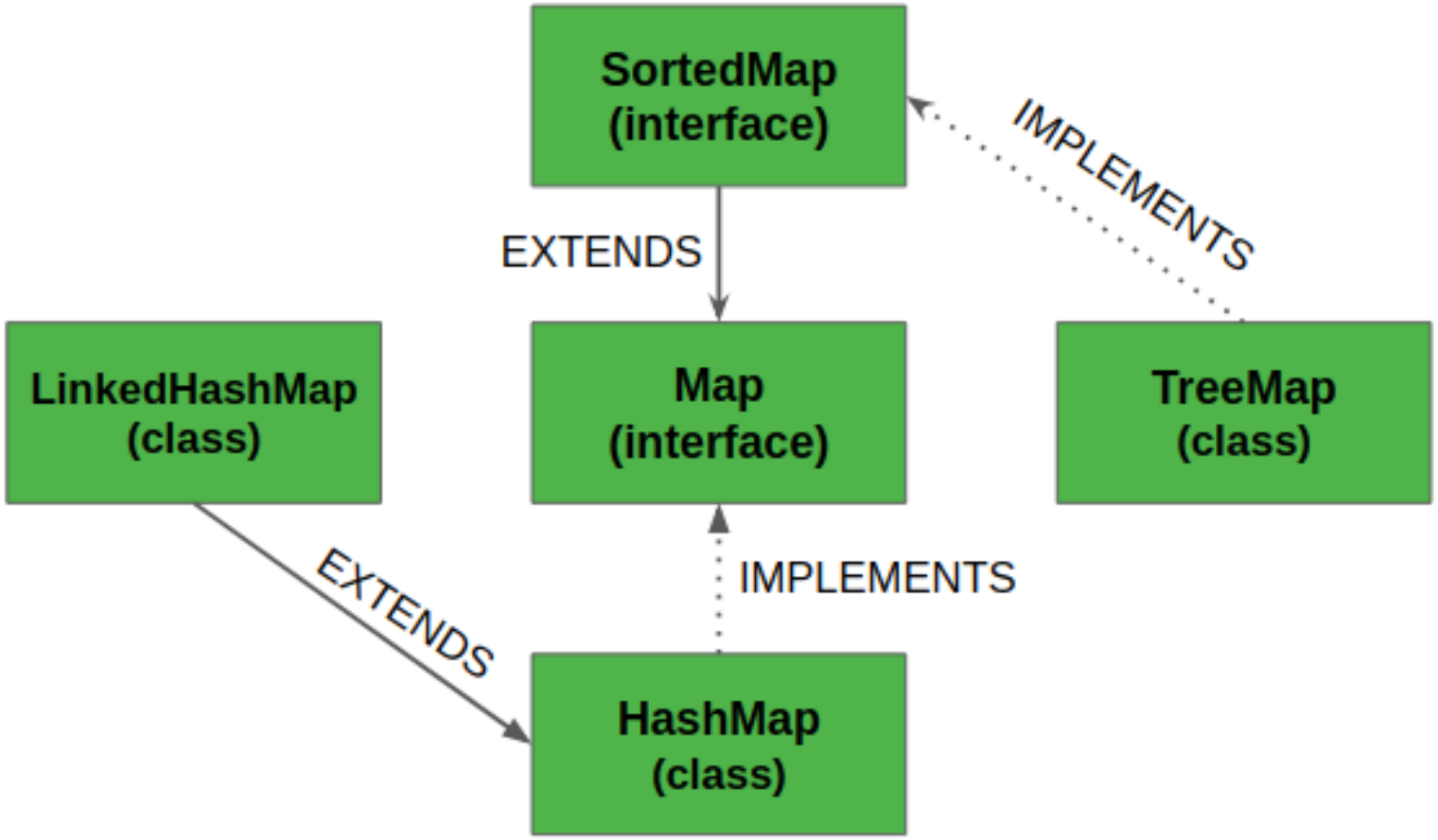
- Map là cấu trúc lưu trữ data dưới dạng **key-value** tương tự như dictionaries
- Không thể chứa duplicate keys
- Lớp thực thi: HashMap, TreeMap, LinkedHashMap

Why and **When** Use Maps:

Maps are perfectly for key-value association mapping such as dictionaries. Use Maps when you want to retrieve and update elements by keys, or perform lookups by keys. Some examples:

- A map of error codes and their descriptions.
- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value).

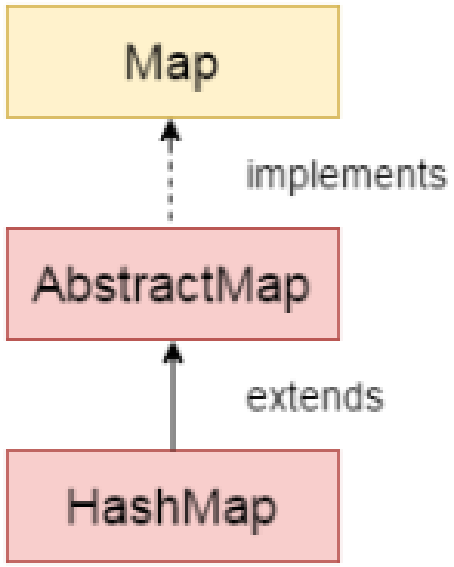
Tổng quan về MAP



MAP Hierarchy in Java

HashMap

- Giới thiệu về HashMap
- Phương thức khởi tạo
- Các phương thức chính
- Ví dụ minh họa



HashMap

- **HashMap lưu trữ giá trị dựa trên key**
- **Key có thể mang giá trị NULL nhưng là duy nhất - không trùng nhau**
- **Không duy trì thứ tự vị trí các phần tử đưa vào**
- **Các phần tử khi đưa vào HashMap chưa được sắp xếp – unsorted yet**
- **public class HashMap<K,V> extends AbstractMap<K,V>**

  implements Map<K,V>, Cloneable, Serializable

  K: key type

  V: value mapped type

HashMap

❖ Phương thức khởi tạo

- HashMap()
- HashMap(Collection c)
- HashMap(int capacity)

❖ Các phương thức chính

- put(Object key, Object value)
- get(Object key)
- remove(Object key)
- containsKey(Object key)
- containsValue(Object value)
- size()
- isEmpty
- .....
- budget -> . -> ctrl+space

HashMap

- Duyệt Entry
  - for(Map.Entry<Key, Value> item: map.entrySet())
- Duyệt Key
  - for(Key key: map.keyset())
- Duyệt Value
  - for (Value value: map.values())

HashMap

Tạo cấu trúc dữ liệu lưu trữ các hoạt động chi tiêu cá nhân trong ngày

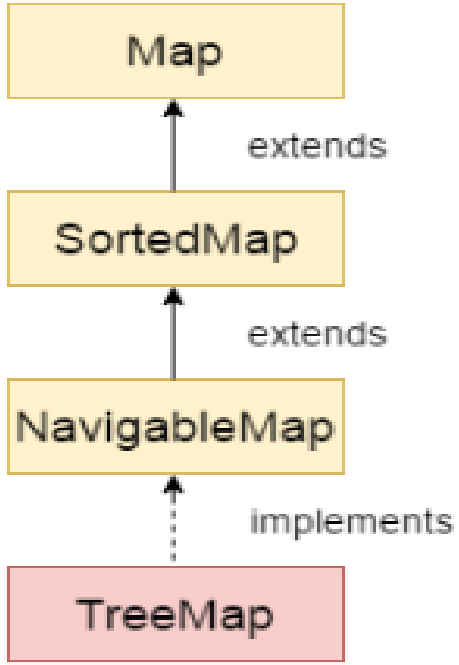
- Tạo danh sách dữ liệu
  - Breakfast
  - Café
  - Football
  - Food
  - Relaxing
- Thêm mới hoạt động
- Tìm kiếm thông tin hoạt động đá bóng
- Xóa thông tin relaxing
- Sửa chi tiêu café thành 20K
- Duyệt và in ra danh sách hoạt động
- Sắp xếp thứ tự các phần tử theo K/V

Tạo ứng dụng từ điển đơn giản cho phép dịch từ English sang Vietnamese

- Tạo danh sách dữ liệu
  - E/V
  - E/V
- Thêm mới từ vựng
- Tra cứu từ vựng
- Xóa từ vựng
- Hiển thị số từ vựng trong từ điển
- Sắp xếp danh sách từ vựng
  - Vietnamese
  - English

TreeMap

- Giới thiệu về TreeMap
- Phương thức khởi tạo
- Các phương thức chính
- Ví dụ minh họa



TreeMap

❖ Giới thiệu về TreeMap

- TreeMap class thực thi từ map interface, lưu trữ dữ liệu key/value pairs theo thứ tự đã được sắp xếp – sorted.
- **Không thể chứa null key,** có thể **multiple null values**
- Các phần tử đã được **sắp xếp theo thứ tự tăng dần theo keys**
- **public class** TreeMap<K,V> **extends** AbstractMap<K,V>

  implements NavigableMap<K,V>, Cloneable, Serializable

K: key type

V: value mapped type

HashMap vs TreeMap

Property \ Map	HashMap	TreeMap
Ordering	not guaranteed	sorted, natural ordering
get / put / remove complexity	O(1)	O(log(n))
Inherited interfaces	Map	Map NavigableMap SortedMap
NULL values / keys	allowed	only values

LinkedHashMap

- **HashMap lưu trữ giá trị dựa trên key**
- **Key có thể mang giá trị NULL nhưng là duy nhất - không trùng nhau**
- **Duy trì thứ tự vị trí các phần tử đưa vào**
- **Các phần tử khi đưa vào HashMap chưa được sắp xếp – unsorted yet**
- **public class HashMap<K,V> extends AbstractMap<K,V>**

  implements Map<K,V>, Cloneable, Serializable

  K: key type

  V: value mapped type

Set

❖ Overview

- Basically, Set is a type of collection that does not allow duplicate elements. That means an element can only exist once in a Set.
- Focus on hashcode, equals method

❖ Characteristics

- Duplicate elements are not allowed.
- Elements are not stored in order. That means you cannot expect elements sorted in any order when iterating over elements of a Set.

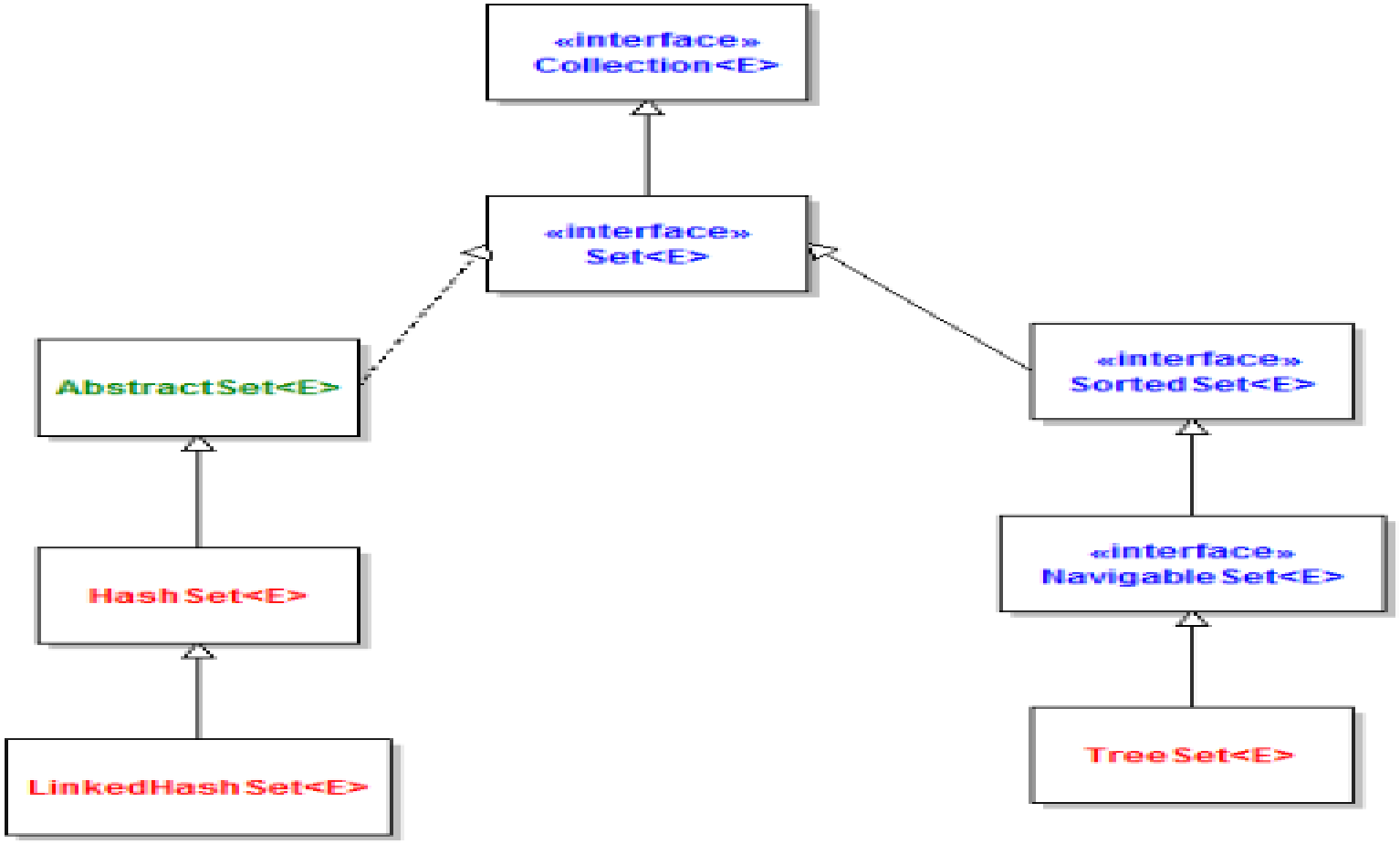
❖ Why and when uses Sets

- You want to store elements distinctly without duplication, or unique elements
- You don’t care about order of elements

Set Interface

❖ Set implementations

- provides three major implementations of the Set interface: HashSet, TreeSet ..



Set – Implementation classes

- **HashSet**: is the best-performing implementation and is a widely-used Set implementation. It represents the core characteristics of sets: **no duplication and unordered**.

- **LinkedHashSet**: This implementation orders its elements based on insertion order. So consider using a LinkedHashSet when you want to store **unique elements in order**.

- **TreeSet**: This implementation orders its elements based on their values, either by their natural ordering, or by a Comparator provided at creation time.

Technique – Hashcode - Equals

- Các phần tử trong cấu trúc dữ liệu Set và các key trong Map yêu cầu không được trùng nhau

- Internal built: Mặc định khi add một phần tử vào Set, key vào Map – nó sẽ gọi phương thức Equals, Hashcode của Object mới để kiểm tra xem đã tồn tại phần tử(Set), key(Map) có Equals và có HashCode bằng với phần tử mới không

- Nếu chưa tồn tại thì sẽ thêm vào, ngược lại báo trùng hoặc ghì đè lên phần tử cũ

- Equals: So sánh địa chỉ 2 đối tượng
- Hashcode: Địa chỉ của đối tượng và tên package, class

Technique – Hashcode - Equals

- Mặc định
  - Equals: So sánh địa chỉ 2 đối tượng
  - Hashcode: Địa chỉ của đối tượng và tên package, class

- Override
  - Equals: So sánh đối tượng theo giá trị các thuộc tính mong muốn
  - Hashcode: Dãy số - là kết quả sau khi hashing giá trị của các thuộc tính, không còn là địa chỉ

- Yêu cầu
  - Các thuộc tính trong equals và hashcode phải giống nhau
  - Nếu 2 đối tượng equals với nhau thì hashcode sẽ bằng nhau

---

If two objects are equal according to the *equals(Object)* method, then calling the hashcode() method on each of the two objects must produce the same integer result.

Technique – Collections

List<E>

1. Các phần tử có thể trùng nhau
2. Phải override phương thức equals khi thực hiện remove, contants

ArrayList
LinkedList
Stack
Queue

Map<K, V>

1. Mỗi phần tử là 1 Entry chứa 2 thuộc tính Key, Value
2. Phải override phương thức equals, hashcode để đảm bảo các key không trùng nhau khi put vào

HashMap
TreeMapp
LinkedHasMap

Cơ chế mã băm

Set<E>

1. Mỗi phần tử là 1 đối tượng
2. Phải override phương thức equals, hashcode để đảm bảo các key không trùng nhau khi add vào

HashSet
TreeSet
LinkedHashSet

Technique – Collections

Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	Yes	Yes	No	Yes	Yes	No
LinkedList	Yes	No	No	Yes	Yes	No
HashSet	No	No	No	No	Yes	No
TreeSet	Yes	No	No	No	No	No
HashMap	No	Yes	Yes	No	Yes	No
TreeMap	Yes	Yes	Yes	No	No	No
Vector	Yes	Yes	No	Yes	Yes	Yes
Hashtable	No	Yes	Yes	No	No	Yes
Properties	No	Yes	Yes	No	No	Yes
Stack	Yes	No	No	Yes	Yes	Yes
CopyOnWriteArrayList	Yes	Yes	No	Yes	Yes	Yes
ConcurrentHashMap	No	Yes	Yes	No	No	Yes
CopyOnWriteArraySet	No	No	No	No	Yes	Yes

DEFINITION OF GENERIC TYPE JAVA 1.5

Problems

```
<R> Stream<R> map(Function<? super T, ? extends R> mapper);

public static <T> void showData(List<T> list) {
    for (T t: list) {
        System.out.print(t + "   ");
    }
}
```

Sometimes, we see some strange words
such as T – R – U - ? – super, extends. What is that ?

Tổng quan generic type

- Xuất hiện từ phiên bản Java 1.5
- Là kiểu dữ liệu đối tượng đại diện cho các class, interface, method chung có thể được tham số hóa cho hầu hết các kiểu dữ liệu đối tượng.
- Tương tự Object và nhiều tiện ích khác
- Hỗ trợ compile-time type safety, cho phép bắt lỗi invalid type tại compile time.
- Giải quyết vấn đề: chúng ta có thể viết duy nhất một phương thức sắp xếp, tìm độ dài mảng … Phương thức này có thể sắp xếp, tìm độ dài các phần tử của một mảng Integer, String, Employee hoặc bất kỳ kiểu dữ liệu gì hỗ trợ sắp xếp.

Lợi ích khi sử dụng generic type

- Hỗ trợ kiểm tra tại compile time
- Kiểm tra errors tại compile time là dễ dàng và nhanh hơn so với việc kiểm tra runtime errors, khó khăn trong tìm kiếm lỗi.
- Loại bỏ casts exception at runtime
- Sử dụng generics, chúng ta có thể thực thi generic algorithm được sử dụng với collection, type safe và code dễ đọc hơn

Lợi ích khi sử dụng generic type

Without Generic Type

```
// required: list of integers
List salaries = new ArrayList(); // default: list of objects

// able to add any type: Integer, String, Double, Item
salaries.add(new Integer(2000));
salaries.add(new Integer(4000));
salaries.add("wrong_type");

// compile time >> no error
// runtime      >> java.lang.NumberFormatException: For input string: "wrong_type"

// for each : JDK 1.5
for (Object salary: salaries) {
    System.out.println(Integer.parseInt(salary.toString()) * 2);
}
```

1. Display warning message for raw type(no pass a parameter for List)
2. Allow to add any type in List >>
   >> compile time will no show error
   >> appear the error at runtime >> hard to find the root cause
3. Expect error at compile time

Lợi ích khi sử dụng generic type

```
// Required: List<Integer> List<String>
//         : error at compile time
List<Integer> digits = new ArrayList<>();
digits.add(14);
digits.add(22);
digits.add(36);
digits.add("wrong_type"); // mandatory is Integer
```

With Generic Type

```
List<String> students = new ArrayList<>();
students.add("John smith");
students.add("Rivaldo");
students.add(123); // mandatory is String
```

Catch the errors at compile time

Generic Topics

- Generic Method, Class, Interface
- Generic Type
- Generics Bounded Type Parameters
- Generics, Inheritance and sub types
- Generics Wildcards

Generic Method

- Viết chương trình in ra danh sách các phần tử trong mảng: Interger, Double, String

```
// generic method printArray
public static <E> void printArray(E[] inputArray) {
    // Display array elements
    for (E element : inputArray) {
        System.out.printf("%s ", element);
    }
    System.out.println();
}


public static void main(String args[]) {
    // Create arrays of Integer, Double and String
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    String[] stringArray = { "Hello", "BKIT", "JAVA", "OX" };

    System.out.println("Array integerArray contains:");
    printArray(intArray); // pass an Integer array

    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray); // pass a Double array

    System.out.println("\nArray characterArray contains:");
    printArray(stringArray); // pass a String array
}
```

Generic Method – Thực hành

- Duyệt và in ra danh sách các số trong List<T>
- Với T có thể là: Integer, Float, Double

- Khác biệt giữa tạo danh sách Object và danh sách Generic

Generic Class

- Khái niệm
- Chúng ta có thể định nghĩa các class với generic type
- Một generic class là một class hoặc interface that is parameterized over types
- Là một class có tham số là 1 KDL khác
- Sử dụng angle bracket <> để có thể có kiểu tham số.

Generic Class

```
public class GenericsTypeOld {
    private Object t;

    public Object get() {
        return t;
    }

    public void set(Object t) {
        this.t = t;
    }

    public static void main(String args[]) {
        GenericsTypeOld type = new GenericsTypeOld();
        type.set("Hello");
        type.set(1);
        String str = (String) type.get();
        // type casting, error prone and can, cause ClassCastException
        System.out.println(str);
    }
}
```

Generic Class – Thực hành

- Tạo cấu trúc dữ liệu CustomList
  - Cho phép lưu trữ danh sách các phần tử là đối tượng
  - Dựa trên cấu trúc mảng, based index
  - Có các chức năng add, remove, get, set, size, isEmpty, contains, removeIf

Generic Class – Thực hành

B1 – Khởi tạo cấu trúc

```java
public interface IList<E> {
    boolean isEmpty();
    boolean add(E t);
    boolean remove(int i);

    void set(int i, E t);
    void show();

    int count(Predicate<E> predicate);
    int size();

    E get(int i);
}
```

```java
public class JavaList<E> implements IList<E> {

    private int initialCapacity = 4;
    private int size;
    private E[] elements;

    public JavaList() {
        elements = createGenericArray(initialCapacity);
    }

    public JavaList(int capacity) {
        if (capacity <= 0) {
            throw new IllegalArgumentException("Capacity should be greater than 0");
        }
        if (capacity > initialCapacity) {
            initialCapacity = capacity;
        }
        elements = createGenericArray(initialCapacity);
    }

    @SuppressWarnings("unchecked")
    private E[] createGenericArray(int size) {
        return (E[])Array.newInstance(Object.class, size);
    }
```

Generic Class – Thực hành

B2 – Thực hiện các chức năng

```
public boolean add(E e) {
    if (size < initialCapacity) {
        elements[size++] = e;
        return true;
    }
    elements = grow(e);
    return true;
}

public boolean remove(int i) {
    return false;
}

public E get(int i) {
    if (i < 0 || i >= size) {
        throw new ArrayIndexOutOfBounds
    }
    return elements[i];
}

public void set(int i, E e) {
    if (i < 0 || i >= size) {
        throw new ArrayIndexOutOfBounds
    }
    elements[i] = e;
}
```

```
private E[] grow(E e) {
    E[] newElements = createGenericArray(size+1);
    for (int i = 0; i < size; i++) {
        newElements[i] = elements[i];
    }
    newElements[size++] = e;
    return newElements;
}

public int count(Predicate<E> predicate) {
    int count = 0;
    for (int i = 0 ; i < size; i++) {
        if (predicate.test(elements[i])) {
            count++;
        }
    }
    return count;
}
```

Generic Type

- Generic Type Naming convention giúp hiểu code dễ dàng hơn thay vì sử dụng object.

- Thông thường tên của Generic Type là single, uppercase letters để dễ dàng phân biệt với biết trong Java. Tên của generic type thường được đặt như sau:

  - E: Element (Sử dụng nhiều trong Collection)

  - K,V: Key/Value (Sử dụng trong Map)

  - T: Type

  - N: Number

  - S, U, V etc: 2nd, 3rd, 4th types

Generic Type - Extend

- Giới hạn kiểu dữ liệu truyền vào cho generic type parameter

```
private static <Element extends Number> void printf(Element[] elements) {
    for (Element element: elements) {
        System.out.print(element + " ");
    }
    System.out.println();
}

public static <T extends Comparable<? super T>> void sort(List<T> list) {
    list.sort(null);
}
```

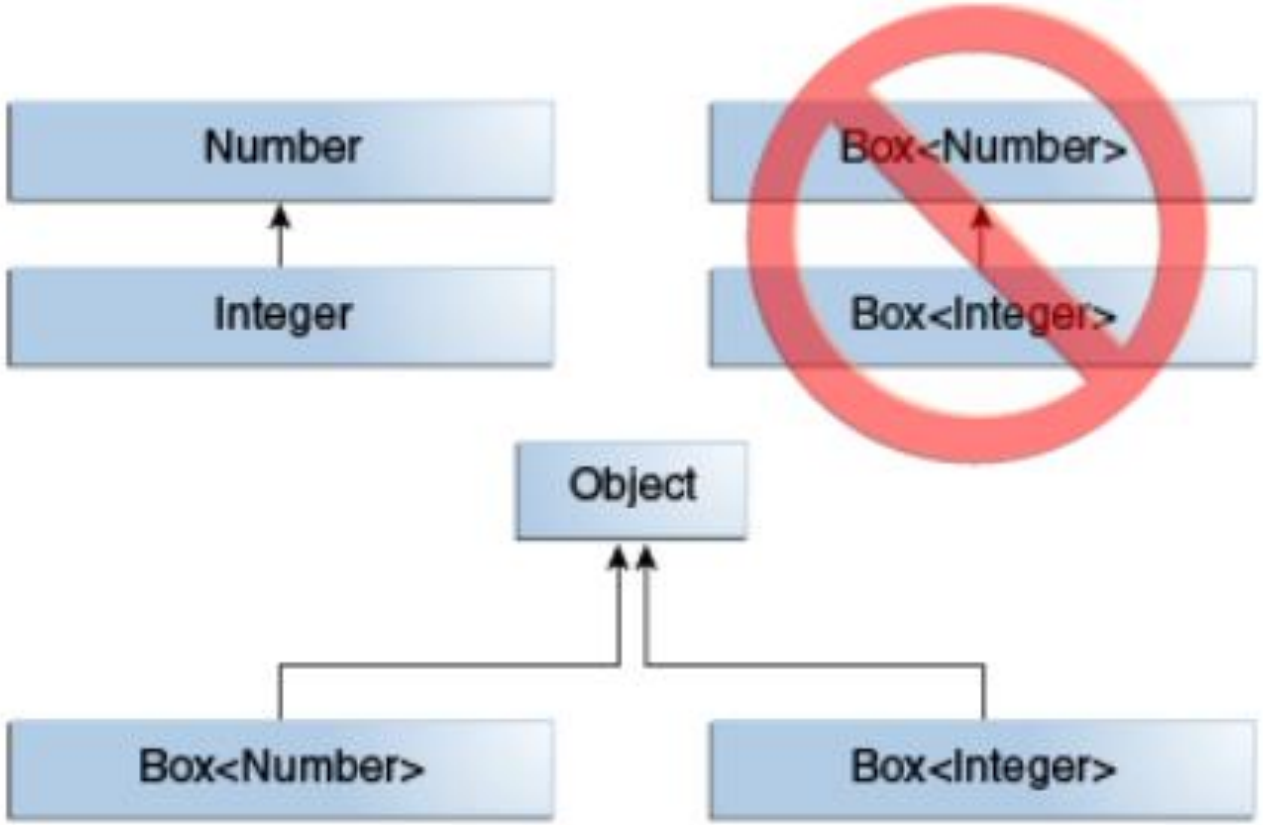
- Lợi ích – Hỗ trợ bắt lỗi tại thời điểm compile
- Ví dụ: Khi sử dụng Collections.sort(List<T>) – Yêu cầu T phải là Comparable hoặc kế thừa (con) Comparable<T> thì mới cho phép

- Khác với Arrays.sort(Object[]) – Bắt lỗi convert tại runtime

Generic Type – Extend – Thực hành

- Sử dụng cấu trúc dữ liệu JavaList – Viết phương thức hỗ trợ
  - Đếm số lượng phần tử thỏa mãn yêu cầu truyền vào – count(Predicate<T> predicate)
  - Giới hạn điều kiện truyền vào chỉ cho phép chuỗi

Generic - Relational



`Box<Integer>` is not a subtype of `Box<Number>` even though `Integer` is a subtype of `Number`.

Generic - Willcard

- Trong generic - question mark (?) được gọi là wildcard(kí tự đại diện) thể hiện một unknown type. Một wildcard có thể được sử dụng ở nhiều tình huống khác nhau
- Willcard có thể là parameter, field, local variable, a return type
- Willcard không bao giờ được sử khi đứng một mình – giúp gọi và truyền tham số unknow type
- List<T> = List<?> = List<Object>
- void show(? list) ✗
- void show(List<? super Integer> list)
- ? A = 10 ✗
- public class List<?> ✗

Generic Class, Interface với tham số là generic
Gọi Generic Class, Interface và mình chưa biết sẽ truyền, trả về kiểu dữ liệu gì cho generic parameter thì mình sẽ dùng dấu ? Để truyền vào

Generic – Willcard - Types

- Unbound – Collections.reverse

```
public static void reverse(List<?> list) {
    int size = list.size();
    if (size < REVERSE_THRESHOLD || list instanceof RandomAccess) {
        for (int i=0, mid=size>>1, j=size-1; i<mid; i++, j--)
            swap(list, i, j);
```

- Lower – Collections.addAll

```
public boolean addAll(int index, Collection<? extends E> c) {
    synchronized (mutex) {return list.addAll(index, c);}
}
```

- Upper – Collections.sort

```
public void sort(Comparator<? super E> c) {
    synchronized (mutex) {list.sort(c);}
}
```

Generic - Willcard

- Phân biệt Generic Willcard(?) và Generic Type(E)
- Trong một số trường hợp, willcard và parameterized type(T) đều được sử dụng giống nhau.
- Nếu phương thức chỉ có one parameterized type argument, có thể sử dụng willcard, mặc dù type parameter cũng có thể - giảm code khai báo
- Generic type hỗ trợ multiple parameters
- Willcard hỗ trợ upper(extends) and lower(super) bounds, type parameters chỉ hỗ trợ upper bounds(extends).
- Trong một phương thức, lớp – T và T sẽ cùng kiểu nhưng ? Và ? Có thể khác kiểu dữ liệu

```
public static <T extends Number> void copy(List<T> dest, List<T> src)
```

```
public static void copy(List<? extends Number> dest, List<? extends Number> src)
```

Technique – Generic

Parameterized Type

1. Generic Method
2. Generic Type Class, Interface
3. Upper Bound with extends

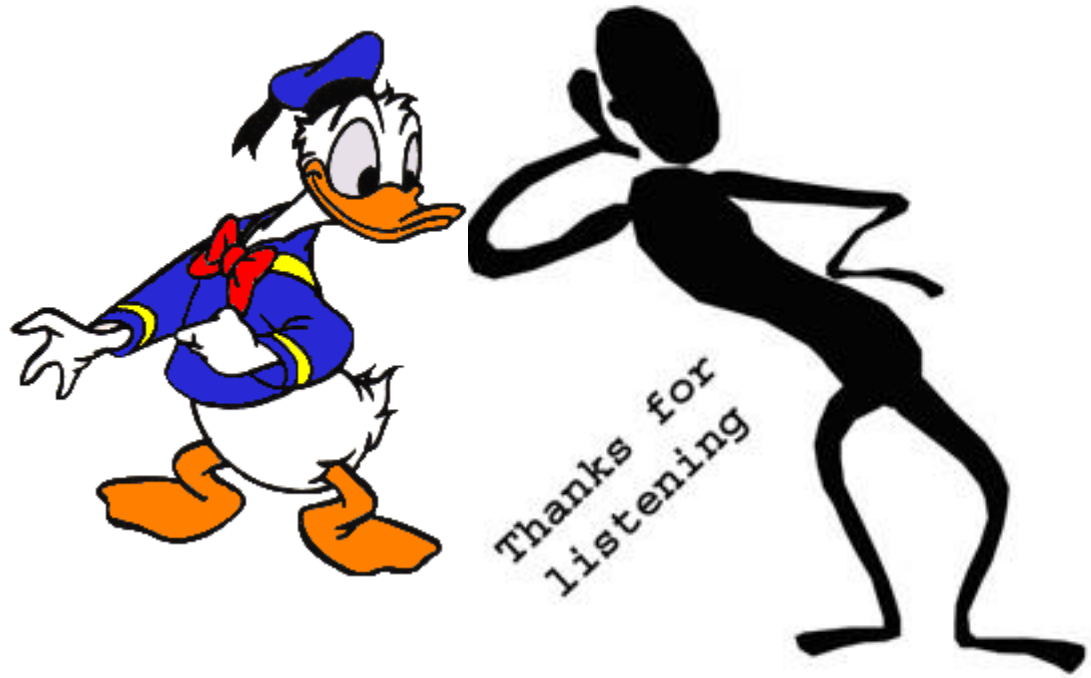
```
private static <Element extends CharSequence> void printg(List<Element> elements) {
    for (Element element: elements) {
        System.out.print(element + "  ");
    }
    System.out.println();
}
```

```
public class CustomList<E> implements IList<E>  {

    private int initialCapacity = 4;
    private int size;
    private E[] elements;
```

WillCard

1. Represent for unknown type
2. Real type at runtime
3. Upper Bound with extends
4. Lower Bound with super

```
public static void printw(List<? super String> elements) {
    for (Object element: elements) {
        System.out.println(element);
    }
}
```



END