Generic - Collection

Giảng viên: Vũ Minh Sang
Phòng: E9.4
E-mail: sangvm@uit.edu.vn

Nội dung

Generics

Collections

Generics

Generics

- Xét phương thức cộng hai số nguyên kiểu `int`

```
public static int Cong(int a, int b)
{
    return a + b;
}
```

- Nhận xét:
  - Không thể dùng phương thức Cong trên để thực hiện cộng hai số kiểu `long, float` hoặc `double.`
  - Để cộng được các số kiểu `long, float` hoặc `double` cần viết code riêng cho từng kiểu dữ liệu.
  - Để sử dụng chung code cho nhiều kiểu dữ liệu, khi khai báo phương thức hoặc class có thể khai báo một kiểu dữ liệu chung, được gọi là kiểu Generic.

Generics

- Generics trong Java (Java Generics): là dạng tham số hóa kiểu dữ liệu.
- Cho phép tạo và sử dụng lớp, interface hoặc phương thức với nhiều kiểu dữ liệu khác nhau theo từng ngữ cảnh khác nhau.
- Xuất hiện từ Java 5.
- Kiểu Generic còn được gọi là tham số kiểu hoặc tham số biến hoặc kiểu dữ liệu tổng quát.
- Tham số biến có thể là các kiểu dữ liệu (trừ các kiểu dữ liệu cơ sở Primary type: `int, float, char`…)
- Khi sử dụng, thay thế tham số biến bằng các kiểu dữ liệu cụ thể.
- Có 2 loại generic: lớp Generic và phương thức Generic.

Generics

- Có thể sử dụng bất kỳ kí tự, viết hoa hoặc thường cho các tham số generic. Tuy nhiên, có một số quy ước đặt tên :
  - E – Element (phần tử, sử dụng trong Collection Framwork)
  - K – Key (khóa)
  - V – Value (giá trị)
  - N – Number (kiểu số: Interger, Long, Float, Double…)
  - T – Type (Kiểu dữ liệu bất kỳ, thuộc kiểu lớp bao: String, Interger, Long, Float…)
  - S, U, V… được sử dụng cho các kiểu loại T thứ 2, 3, 4....
- Ký tự Diamond <>: từ Java 7, có thể thay thế các đối số kiểu dữ liệu để gọi hàm khởi tạo của một lớp Generic bằng cặp dấu <>.

```
// Trước Java 7
List<Integer> listInt = new ArrayList<Integer>();
// Sử dụng cặp dấu <> từ phiên bản Java 7
List<Integer> listInt = new ArrayList<>();
```

Generics

- Ưu điểm của Generics:
  - Kiểm tra kiểu dữ liệu trong thời điểm biên dịch để đảm bảo tính chặt chẽ của kiểu dữ liệu.

```
class A{}
class B extends A{
    public void methodB(){}
}
class C {}
```

```
public class GenericEx {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add(new A());
        list.add(new B());
        list.add(new C());

        ((B)list.get(0)).methodB(); //compile OK
        ((B)list.get(1)).methodB(); //compile OK
        ((B)list.get(2)).methodB(); //compile OK
    }
```

```
Exception in thread "main" java.lang.ClassCastException: class genericex.A cannot be cast to class genericex.B (genericex.A and genericex.B are in unnamed module of loader 'app')
        at genericex.GenericEx.main(GenericEx.java:33)
D:\Hoc tap\Java\GenericEx\nbproject\build-impl.xml:1328: The following error occurred while executing this line:
D:\Hoc tap\Java\GenericEx\nbproject\build-impl.xml:948: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Generics

```
public class GenericEx {
    public static void main(String[] args) {

        List<A> list = new ArrayList<A>();
        list.add(new A());  //OK Compile
        list.add(new B());  //OK Compile
```

incompatible types: C cannot be converted to A

(Alt-Enter shows hints)

```
        list.add(new C());

        for (A item : list){
            if (item instanceof B) ((B)item).methodB();
        }
    }
```

Generics

- Loại bỏ việc ép kiểu dữ liệu.

```
//Phải ép kiểu
List list = new ArrayList();
list.add("abc");
String s = (String) list.get(0);


//Không phải ép kiểu
List<String> list = new ArrayList<>();
list.add("abc");
String s = list.get(0);
```

Generics

- Cho phép thực hiện các xử lý tổng quát: thực hiện các thuật toán tổng quát với các kiểu dữ liệu tùy chọn khác nhau.

```
public static <T extends Number> double add(T a, T b){
    return a.doubleValue() + b.doubleValue();
}

public static void main(String[] args) {
    System.out.println(add(1,2));
    System.out.println(add(7.5,15.0));
}
```

Generic Class

- Generic Class là dạng class có một hoặc nhiều tham số biến sử dụng trong class.
- Một class có thể tham chiếu bất kỳ kiểu đối tượng nào.
- Cú pháp Generic class:

```
class Tên_Class<T1,T2,…,Tn> {}
```

- Khi sử dụng, khai báo <T> với kiểu dữ liệu cụ thể nào thì trong generic class sẽ chỉ xử lý kiểu dữ liệu đó.
- Phạm vi và ý nghĩa của kiểu T sẽ là trong toàn class
- Sử dụng generic class khi:
  - Khi xây dựng class, chưa xác định được kiểu dữ liệu của biến thành viên, thuộc tính hoặc biến cục bộ của phương thức.
  - Khi nhiều class có cùng chung về mặt logic (các biến, các phương thức) chỉ khác biệt về kiểu dữ liệu.

Generics class

```
public class KeyValue<K, V> {
    private K key;
    private V value;

    public KeyValue(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey() {
        return key;
    }
    public void setKey(K key) {
        this.key = key;
    }
     public V getValue() {
        return value;
    }
     public void setValue(V value) {
        this.value = value;
    }
}
```

Generics class

```
public static void main(String[] args) {
    // TODO code application logic here
    //KeyValue với kiểu Integer và String
    KeyValue<Integer, String> keyValue = new KeyValue<Integer, String>(1234, "Test");
    Integer id = keyValue.getKey();
    String name = keyValue.getValue();
    System.out.println("ID: " + id + "Name =  " + name);

    //KeyValue với kiểu String và String
    KeyValue<String, String> keyString = new KeyValue<>("ABC","XYZ");
    String key = keyString.getKey();
    String value = keyString.getValue();
    System.out.println("Key: " + key + "Value =  " + value);
}
```

Generic Method

- Generic method (generic phương thức) là dạng phương thức có một hoặc nhiều tham số biến.
- Phương thức có thể được gọi với nhiều kiểu dữ liệu khác nhau.
- Cú pháp generic phương thức:

```
Tiền tố <T1,T2,…,Tn> Kiểu trả về Tên_Method([tham số]){}
```

- Khi sử dụng, khai báo <T> với kiểu dữ liệu cụ thể nào thì trong generic method sẽ chỉ xử lý kiểu dữ liệu đó.
- Phạm vi và ý nghĩa của kiểu T sẽ là toàn bộ trong method.
- Hai tham biến cùng kiểu dữ liệu chỉ cần khai báo một kiểu dữ liệu giả T.
- Thao tác trên kiểu dữ liệu <T> giống như là một kiểu dữ liệu bình thường.

Generic Method

- Kiểu <T> có thể được sử dụng làm kiểu trả về của phương thức.
- Sử dụng generic method khi logic của phương thức giống nhau và chỉ khác biệt nhau về kiểu dữ liệu thì có thể cài đặt phương thức theo generic.
- VD1:

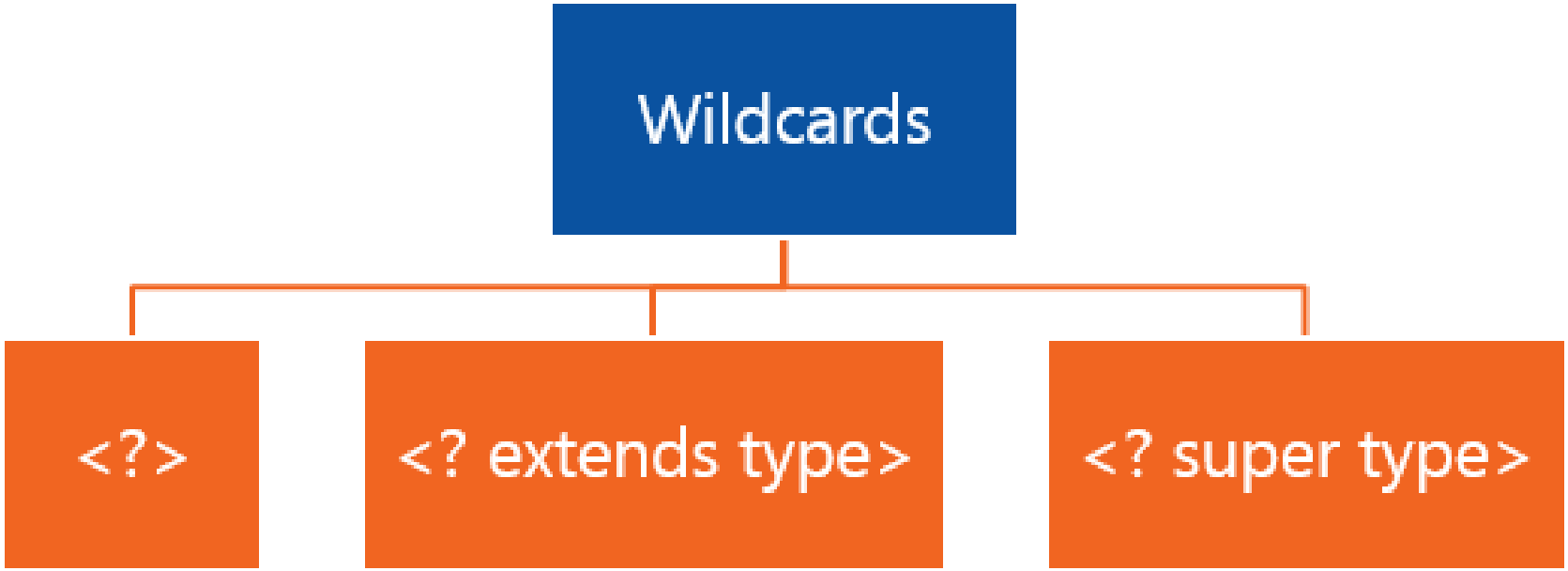
```
public static <T> T getMiddle(T... a){
    return a[a.length/2];
}

public static void main(String[] args) {
    String middle = getMiddle("ABC","DEF","IJK");
    System.out.println(middle);
    int midInt = getMiddle(5,8,1,17,19,20);
    System.out.println(midInt);
}
```

Generic Method

- VD2:

```
public static <T extends Comparable> T timMax(T a,T b,T c){
    T max = a;
    if (max.compareTo(b) < 0)
       max = b;
    if (max.compareTo(c) < 0)
        max = c;
    return max;
}

public static void main(String[] args) {
    int maxInt = timMax(8, 35, 20);
    System.out.println(maxInt);
    double max = timMax(5.7, 8.4, 30.0);
    System.out.println(max);
}
```

Ký tự đại diện Generic



Ký tự đại diện Generic

- Ký tự đại diện `<?> (wildcard)`: đại diện cho một kiểu không xác định.
- Có thể được sử dụng trong nhiều tình huống: tham số, biến cục bộ, thuộc tính hoặc có thể là một kiểu trả về.
- Không sử dụng như là một đối số cho lời gọi một phương thức generic, khởi tạo đối tượng class generic, hoặc kiểu cha.
- VD:

```
Collection<?> coll = new ArrayList<String>();
Pair<String,?> pair = new Pair<String,Integer>();
```

- Tham số ký tự đại diện không thể tham gia trong toán tử new

```
List<? extends Object> list= new ArrayList<?
    extends Object>(); //Lỗi
```

Ký tự đại diện Generic

- Có thể dùng để hạn chế kiểu dữ liệu của các tham số:
  - `<? extends type>:` kiểu dữ liệu kế thừa từ type hoặc đối tượng của type.

```
public static <T extends Comparable> T timMax(T a,T b,T c){
    T max = a;
    if (max.compareTo(b) < 0)
       max = b;
    if (max.compareTo(c) < 0)
        max = c;
    return max;
}
```

  - `<? super type>`: kiểu dữ liệu là kiểu cha type hoặc đối tượng của type

```
class A{}
class B extends A{}
class C {}
```

```
public static void methodB(List<? super B> param){}

public static void main(String[] args) {
    List<B> listB = new ArrayList<B>();
    methodB(listB);
    List<A> listA = new ArrayList<A>();
    methodB(listA);
    List<Object> listO = new ArrayList<Object>();
    methodB(listO);
    List<C> listC = new ArrayList<C>();
    methodB(listC);
}
```

Hạn chế của Generic

- Không thể khởi tạo generic với kiểu dữ liệu cơ sở

```
KeyValue<int, double> keyValue = new KeyValue<>(1234, 30.3); // Lỗi
```

- Không tạo được đối tượng của kiểu T

```
private T obj;
public GenExample(){
    obj = new T();
}
```

- Không là kiểu static trong class

```
class GenExample<T>
{
    //Lỗi
    static T obj;
    //Lỗi
    static T getObj(){
        return obj;
    }
}
```

Hạn chế của Generic

- Có thể khai báo một mảng generic nhưng không thể khởi tạo mảng Generic

```
public T[] arrayT; //OK
public T[] array = new T[10]; //Lỗi

//Lỗi
GenExample<String> gens[] = new GenExample<>[10];

//OK
GenExample<?> gens[] = new GenExample<?>[10];
gens[0] = new GenExample<Integer>(13);
gens[1] = new GenExample<String>("Gen");
```

- Không thể tạo class ngoại lệ là generic

```
public class GenException<T> extends Exception {}
```

a generic class may not extend java.lang.Throwable

(Alt-Enter shows hints)

Ví dụ mảng Generic

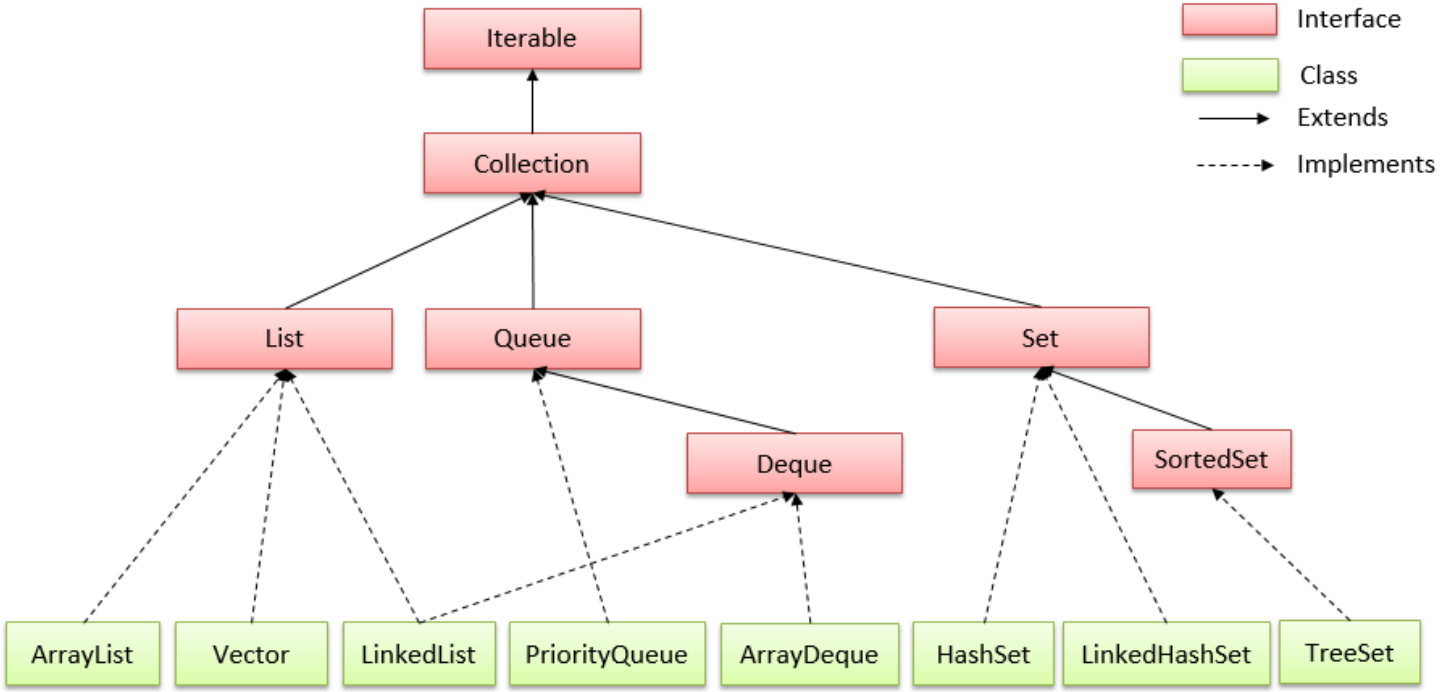
```
public class GenericArray<T> {
    public T[] array;

    public GenericArray(T[] arr){
       this.array = arr;
    }
    //Phương thức lấy phần tử tại vị trí index
    public T Get(int index){
        if (index >= 0 && index < array.length) return array[index];
        return null;
    }
}

public static void main(String[] args) {
    //Tạo một mảng String
    String[] country = new String[]{"US","UK","AU"};
    GenericArray<String> gArr = new GenericArray<>(country);
    String indCoun = gArr.Get(1);
    System.out.println(indCoun);


}
```

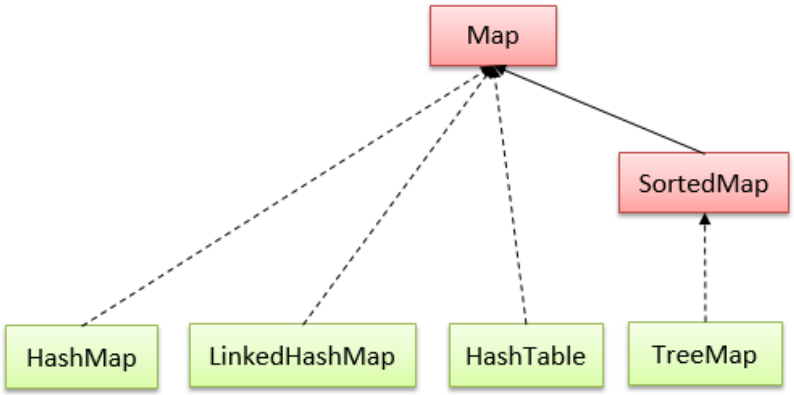
Collection Framework

Collection Framework

- Là tập hợp các interface và các lớp hỗ trợ thao tác trên tập hợp các đối tượng.
- Hỗ trợ thực hiện các thao tác trên dữ liệu: tìm kiếm, sắp xếp, phân loại, thêm, sửa, xóa…
- Cung cấp các thành phần sau:
  - *Interface*: là kiểu dữ liệu abstract biểu diễn collection; cho phép các collection thao tác độc lập.
  - *Implementations* (triển khai): là việc triển khai cụ thể của collection interface.
  - *Algorithms* (thuật toán): là các phương thức để thực thi các phép toán (tìm kiếm, sắp xếp…) trên các đối tượng đã triển khai các interface collection.
- Thuộc package **java.util.**
- Gồm 2 loại chính: *Interface Collections, Class Collections.*
- Ngoài ra, còn có *Map Interface và các class của Map* lưu trữ theo cặp key/value

Collection Framework





Collection Framework

- **Interface Collections:**
  - Là tập hợp đại diện cho nhóm các đối tượng.
  - Một số interface cho phép lưu trữ các phần tử giống nhau hoặc không giống nhau.
  - Tùy từng loại collection, các phần tử có thể có thứ tự hoặc không.
  - Bao gồm các phương thức: thêm (add), xóa (clear), so sánh (compare), duy trì (retaining) đối tượng.
  - Kế thừa lớp **Iterable interface**, có thể sử dụng **Iterator** để duyệt từng phần tử.
- **Class Collections:** là các lớp tiêu chuẩn dùng để thực thi các Interface Collections.
  - Trước JDK 1.5 là dạng non-generic; về sau là dạng generic.
  - Non-generic: `ArrayList arr = new ArrayList();`
  - Generic: `ArrayList<String> arr = new ArrayList<String>();`

Collection Framework

- **Iterable interface:** chứa phương thức tạo ra một Iterator.
- **Iterator interface:**
  - Là đối tượng có trạng thái lặp.
  - Truy xuất các phần tử từ đầu đến cuối của một collection.
  - Xóa phần tử khi lặp một collection.
  - Có 3 phương thức trong Iterator:

Phương thức	Mô tả
public boolean hasNext()	True nếu iterator còn phần tử kế tiếp phần tử đang duyệt.
public object next()	Trả về phần tử hiện tại và di chuyển con trỏ trỏ tới phần tử tiếp theo.
public void remove()	Loại bỏ phần tử cuối được trả về bởi Iterator.

Interface Collections

- Các Interface và Class Collections:

List*:* cấu trúc dữ liệu tuyến tính, các phần tử được sắp xếp theo thứ tự xác định và giá trị có thể trùng nhau. Gồm các class:

- *ArrayList*: kiểu danh sách sử dụng cấu trúc mảng (có kích thước thay đổi được) để lưu trữ phần tử; thứ tự các phần tử dựa theo thứ tự lúc thêm vào, giá trị có thể trùng nhau và không phân biệt kiểu dữ liệu của từng phần tử.
- *LinkedList:* danh sách liên kết đôi (double-linked list), duy trì thứ tự các phần tử được thêm vào và giá trị có thể giống nhau.
- *Vector*: tương tự ArrayList; kích thước có thể thay đổi được; là dạng synchronized (đồng bộ).
- *Stack*: lưu trữ trên cơ sở cấu trúc dữ liệu ngăn xếp (stack) LIFO.

Interface Collections

Set: mỗi phần tử chỉ xuất hiện một lần (giá trị các phần tử không được giống nhau). Gồm các class:

- *HashSet*: các phần tử được lưu trữ dưới dạng mảng băm (hash table); thứ tự các phần tử không dựa theo lúc thêm vào mà được sắp xếp ngẫu nhiên và giá trị các phần tử không trùng nhau.
- *LinkedHashSet*: kế thừa lớp HashSet và implement interface Set; chứa các phần tử duy nhất, đảm bảo thứ tự phần tử được thêm vào, cho phép chứa phần tử Null.

SortedSet*:* dạng riêng của Set Interface; giá trị các phần tử mặc định được sắp xếp tăng dần.

- *TreeSet:* các phần tử mặc định sắp xếp tăng dần và giá trị của các phần tử là duy nhất.

Interface Collections

Queue: được thực thi theo kiểu FIFO; có các loại queue: priority queue (queue có ưu tiên), interface deque (queue 2 chiều)…

- *LinkedList:* là LinkedList trong interface List.
- *PriorityQueue*: các phần tử được sắp xếp theo trật tự tự nhiên (các phần tử so sánh được với nhau – thi hành Comparable) hoặc theo một bộ so sánh Comparator được cung cấp cho PriorityQueue.
- *ArrayDeque*: là dạng queue 2 chiều; thực thi dựa trên mảng.

Interface Collections

Map *(đồ thị/ánh xạ):* dữ liệu của phần tử được quản lý theo dạng cặp key/value; key là duy nhất và ứng với mỗi key là một value. Không kế thừa từ Collection Interface.

- *HashMap*: giá trị mỗi phần tử bao gồm 2 phần: khóa (key – là duy nhất) được lưu trữ dưới dạng bảng băm và giá trị tương ứng (value); truy xuất trực tiếp dữ liệu bằng khóa; cho phép 1 key null và nhiều giá trị null.
- *LinkedHashMap*: có thể chứa 1 key là null và nhiều giá trị null; thứ tự các phần tử theo thứ tự thêm.
- *HashTable*: không cho phép key hoặc giá trị null

SortedMap: là dạng riêng của Map Interface, giá trị key được sắp xếp tăng dần.

- *TreeMap:* giá trị mỗi phần tử bao gồm 2 phần: khóa (key – là duy nhất) và giá trị tương ứng (value); key được sắp xếp tăng dần.

Interface Collections

Các phương thức trong Interface Collections:

Phương thức	Mô tả
boolean add(Object element)	Thêm một phần tử vào collection.
boolean addAll(Collection c)	Thêm các phần tử collection được chỉ định.
boolean remove(Object element)	Xóa phần tử từ collection.
boolean removeAll(Collection c)	Xóa tất cả các phần tử của collection được chỉ định.
boolean retainAll(Collection c)	Giữ lại các phần tử collection được chỉ định.
int size()	Tổng số các phần tử trong collection.
void clear()	Xóa tất cả các phần tử khỏi collection.
boolean contains(Object element)	True nếu collection chứa phần tử được chỉ định.
boolean containsAll(Collection c)	True nếu collection chứa collection con được chỉ chỉ định.
Iterator iterator()	Trả về một iterator.
Object[] toArray()	Trả về mảng chứa tất cả phần tử của collection.
boolean isEmpty()	True nếu collection rỗng.
boolean equals(Object element)	So sánh một đối tượng với collection.
int hashCode()	Trả về giá trị hashcode của collection.

ArrayList

- Là lớp thực thi của List Interface
- Sử dụng cấu trúc mảng để lưu trữ phần tử.
- Thứ tự các phần tử dựa theo lúc thêm vào và giá trị có thể trùng nhau.
- Chứa dữ liệu thuộc bất cứ kiểu dữ liệu nào
- Các phần tử có thể có kiểu dữ liệu khác nhau (non-generic).
- Thuộc **java.util.ArrayList**.
- Là loại không đồng bộ (non-synchronized).
- Cho phép truy cập ngẫu nhiên.
- Lưu trữ theo chỉ mục, tốc độ truy xuất (get) nhanh.
- Thêm (add), xóa (remove) chậm vì cần nhiều sự chuyển.
- Thường dùng khi cần truy xuất phần tử nhiều hơn cập nhật và xóa phần tử.

ArrayList

- Khai báo và khởi tạo ArrayList có 2 cách:
  - Cách 1: khởi tạo một ArrayList rỗng

```
ArrayList<String> list = new ArrayList<String>();
```

  - Cách 2: khởi tạo và cung cấp số lượng phần tử ban đầu

```
ArrayList<Integer> listInt = new ArrayList<>(10);
```

- Truy xuất phần tử dùng phương thức `get(int index)`.

```
Sring s = list.get(1);
Interger num = listInt.get(2);
```

ArrayList

- Duyệt ArrayList dùng vòng lặp For:

```
for (int i = 0; i < list.size(); i++){
    System.out.println(list.get(i));
}
Hoặc
for (int num : listInt){
    System.out.println(num);
}
```

- Duyệt ArrayList sử dụng Iterator:
  - Thuộc **java.util.Iterator**.
  - Khai báo Iterator cùng kiểu với ArrayList muốn duyệt.

```
Iterator<String> itr = list.iterator();
```

  - Dùng hàm `hasNext()` và hàm `next()` để duyệt.

```
while (itr.hasNext()){
    System.out.println(itr.next());
}
```

ArrayList

- Duyệt ArrayList sử dụng ListIterator
  - Thuộc **java.util.ListIterator.**
  - Khai báo ListIterator cùng kiểu với ArrayList muốn duyệt.

```
ListIterator<int> listItr = list.listIterator();
```

  - Dùng hàm `hasNext()` và `next()` để duyệt list từ đầu tới cuối

```
while (listItr.hasNext()){
    System.out.println(listItr.next());
}
```

  - Dùng hàm `hasPrevious()` và `previous()` để duyệt list từ cuối về đầu.

```
while (listItr.hasPrevious()){
    System.out.println(listItr.previous());
}
```

ArrayList

- Một số phương thức của ArrayList:
  - Thêm phần tử vào cuối danh sách: `add(Object o)`

```
list.add("XYZ");
```

  - Thêm collection vào cuối danh sách: `addAll(Collection c)`

```
list.addAll(listString);
```

  - Thêm phần tử vào vị trí bất kỳ trong danh sách:

`add(int index, object value)`

```
list.add(2, "XYZ");
```

  - Thêm collection vào vị trí bất ký trong danh sách

`addAll(int index, Collection c)`

```
list.addAll(3, listString);
```

  - Cập nhật giá trị phần tử: `set(int index, Object o)`

```
list.set(3, "ABC");
```

ArrayList

- Xóa phần tử tại vị trí index: `Remove(int index)`

```
list.Remove(3);
```

- Xóa tất cả các phần tử: `Clear()`

```
list.Clear();
```

ArrayList

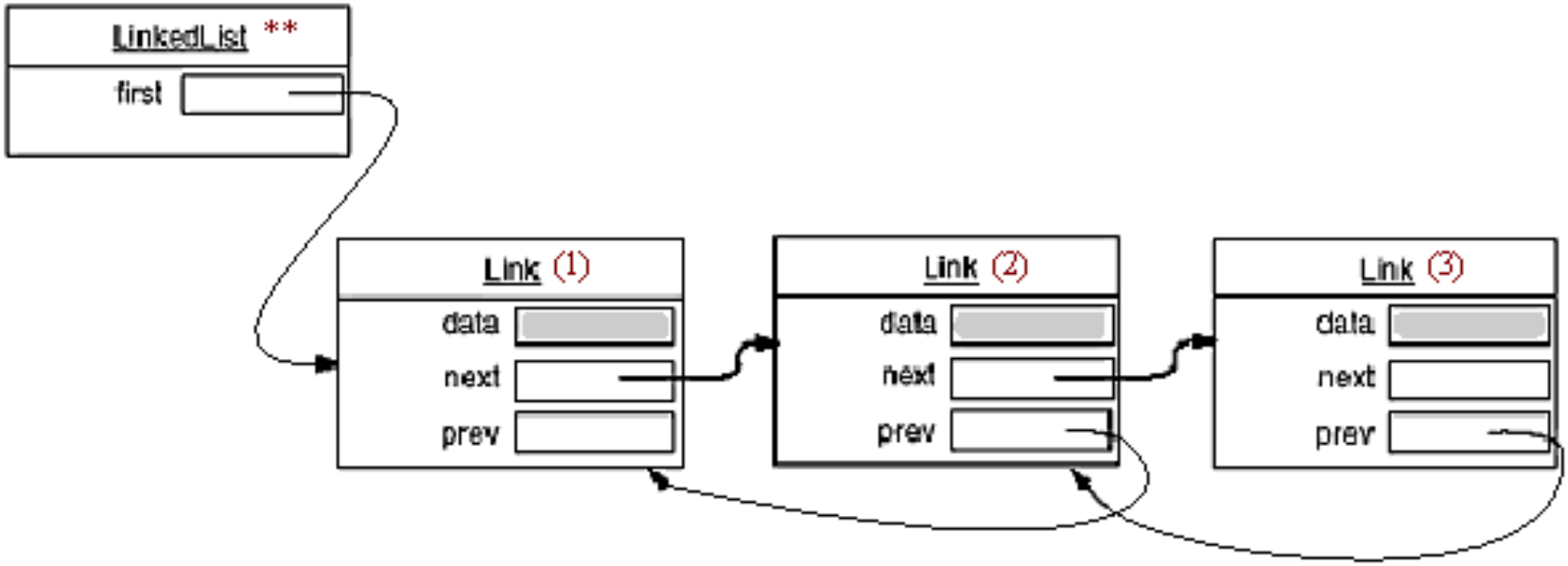
- Một số phương thức khác:

Phương thức	Mô tả
boolean isEmpty()	True nếu ArrayList rỗng.
int indexOf (Object o)	Trả về vị trí index trong list của phần tử o xuất hiện đầu tiên, hoặc -1 nếu List không chứa phần tử này
int lastIndexOf(Object o)	Trả về vị trí index của phần tử o cuối cùng, hoặc -1 nếu List không chứa phần tử này
boolean removeAll(Collection c)	Xóa tất cả các phần tử của ArrayList được chỉ định.
boolean retainAll(Collection c)	Giữ lại các phần tử ArrayList được chỉ định.
void removeRange(int fromIndex, int toIndex)	Gỡ bỏ từ list này tất cả phần tử từ vị trí fromIndex đến toIndex
int size()	Tổng số các phần tử trong ArrayList.
boolean contains(Object element)	True nếu ArrayList chứa phần tử được chỉ định.
void trimToSize()	Cắt kích thước của ArrayList này về kích thước hiện tại.
Object[] toArray()	Trả về một mảng chứa tất cả phần tử của list.
Object clone()	Trả về một bản copy của ArrayList này

LinkedList

- Là lớp thực thi của List Interface và Deque Interface
- Sử dụng cấu trúc danh sách liên kết **Doubly Linked List** để lưu trữ phần tử.
- Duy trì thứ tự các phần tử thêm vào và giá trị có thể trùng nhau.
- Thuộc **java.util.LinkedList**.
- Là loại không đồng bộ (non-synchronized).
- Thêm (add), xóa (remove) nhanh vì không cần phải dịch chuyển nếu bất kỳ phần tử nào thêm/xóa khỏi danh sách.
- Có thể sử dụng như danh sách (list), ngăn xếp (stack) hoặc hàng đợi (queue).
- Thường dùng khi cần làm việc với thêm/xóa lượng lớn phần.

LinkedList


LinkedList **
first
Link (1)
data
next
prev
Link (2)
data
next
prev
Link (3)
data
next
prev

LinkedList

- Khai báo và khởi tạo LinkedList:
  - Khởi tạo một LinkedList rỗng

```
LinkedList<String> list = new LinkedList<String>();
```

  - Khởi tạo với danh sách phần tử: `LinkedList(Collection c)`

```
LinkedList<Integer> listInt = new LinkedList<>(lInt);
```

- Truy xuất phần tử dùng phương thức `get(int index)`.

```
Sring s = list.get(1);
Interger num = listInt.get(2);
```

- Duyệt LinkedList dùng vòng lặp For hoặc Iterator hoặc ListIterator giống với ArrayList

LinkedList

- Một số phương thức của LinkedList:

Phương thức	Mô tả
boolean add(Object o)	Thêm phần tử vào cuối list
boolean add(int index, Object o)	Thêm phần tử vào vị trí index của list.
boolean addAll(Collection c)	Thêm một collection vào cuối list.
boolean addAll(int index, Collection c)	Thêm một collection vào vị trí index của list
boolean addFirst(Object o)	Thêm phần tử vào đầu list
boolean addLast(Object o)	Thêm phần tử vào cuối list
void clear()	Xóa tất cả các phần tử khỏi list.
boolean contains(Object o)	True nếu list chứa phần tử được chỉ định.
Object get(int index)	Trả về phần tử tại vị trí index của list
Object getFirst()	Trả về phần tử đầu tiên của list
Object getLast()	Trả về phần tử cuối của list
int indexOf(Object o)	Trả về vị trí index của phần tử o xuất hiện đầu tiên trong list

LinkedList

- Một số phương thức của LinkedList:

Phương thức	Mô tả
int lastIndexOf(Object o)	Trả về vị trí index của phần tử o cuối cùng, hoặc -1 nếu List không chứa phần tử này
Object remove(int index)	Xóa phần tử tại vị trí index trong list.
boolean remove(Object o)	Xóa phần tử o xuất hiện đầu tiên trong list.
Object removeFirst()	Xóa phần tử đầu tiên trong list.
Object removeLast()	Xóa phần tử cuối cùng trong list.
Object set(int index, Object o)	Thay thế phần tử tại vị trí index bằng phần tử o
int size()	Trả về số phần tử trong list
Object[] toArray()	Trả về một mảng chứa tất cả phần tử của list.

So sánh ArrayList và LinkedList

ArrayList	LinkedList
Sử dụng mảng động để lưu trữ phần tử	Sử dụng danh sách liên kết (Doubly Linked List) để lưu trữ phần tử.
Lưu trữ dữ liệu trên chỉ mục (index), mỗi phần tử (element) liên kết với một index.	Mỗi phần tử (node) lưu trữ 3 thông tin: giá trị phần tử, tham chiếu phần tử trước và tham chiếu phần tử sau.
Thao tác thêm, xóa chậm vì sử dụng nội bộ mảng; sau khi thêm, xóa cần sắp xếp lại	Thêm, xóa nhanh hơn ArrayList; không cần sắp xếp lại phần tử, chỉ cập nhật lại tham chiếu tới phần tử trước và sau.
Truy xuất nhanh (dựa trên index)	Truy xuất chậm hơn nhiều ArrayList vì phải duyệt lần lượt các phần tử từ đầu tới cuối
Truy xuất ngẫu nhiên nhiều phần tử	Không thể truy xuất ngẫu nhiên.
Chỉ hoạt động như một list vì là implements của List Interface	Hoạt động như list, stack hoặc queue, vì là implements của List và Deque Interface
Ít tốn bộ nhớ.	Cần nhiều bộ nhớ.
Tốt trong việc lưu trữ và truy xuất (get)	Tốt trong việc thực hiện thêm, xóa.

TreeSet

- Là lớp thực thi của SortedSet Interface
- Sử dụng TreeMap để lưu trữ phần tử.
- Các phần tử mặc định được sắp xếp tăng dần hoặc dựa trên bộ so sánh Comparator tùy chỉnh lúc khởi tạo.
- Giá trị phần tử là duy nhất và không null.
- Là loại không đồng bộ (non-synchronized).
- Thuộc **java.util.TreeSet**.
- Duyệt TreeSet dùng vòng lặp For hoặc Iterator.

TreeSet

- Khai báo và khởi tạo TreeSet:
  - Khởi tạo một TreeSet rỗng

```
TreeSet<String> list = new TreeSet <>();
```

  - Khởi tạo với danh sách phần tử: `TreeSet(Sorter<> s)`

```
TreeSet <Integer> listInt = new TreeSet <>(lInt);
```

  - Khởi tạo với với bộ so sánh Comparator tùy chỉnh.

```
TreeSet<String> list = new
      TreeSet<>(String.CASE_INSENSITIVE_ORDER);
```

Hoặc

```
TreeSet<String> list = new
      TreeSet<>(Comparator.reverseOrder());
```

Hoặc

```
TreeSet<String> list = new TreeSet<>(new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                  return s2.compareTo(s1);
            }
      });
```

TreeSet

- Các phương thức trong TreeSet

Phương thức	Mô tả
void add(Object o)	Thêm phần tử vào TreeSet
boolean addAll(Collection c)	Thêm một collection vào TreeSet.
boolean remove(Object o)	Xóa phần tử khỏi TreeSet
void clear()	Xóa tất cả các phần tử khỏi TreeSet
boolean contains(Object o)	True nếu TreeSet chứa phần tử được chỉ định.
Object first()	Trả về phần tử đầu tiên (nhỏ nhất) của TreeSet
Object last()	Trả về phần tử cuối cùng (lớn nhất) của TreeSet
SortedSet subSet(Object fromElement, Object toElement)	Trả về SortedSet từ fromElement đến phần tử đứng trước toElement
SortedSet headSet(E toElement)	Trả về SortedSet từ phần tử đầu tiên đến phần tử đứng trước toElement
SortedSet tailSet(E fromElement)	Trả về SortedSet từ phần tử lớn hơn hoặc bằng fromElement đến phần tử cuối cùng
int size()	Trả về số phần tử trong TreeSet
boolean isEmpty()	True nếu TreeSet rỗng.

Lựa chọn Collection

- Dựa trên cấu trúc dữ liệu tốt nhất và thuật toán hỗ trợ.
- Phần tử null?
- Trùng lặp phần tử?
- Truy cập phần tử theo index hay key?
- Thao tác thêm xóa hoặc sửa như thế nào?
- Sắp xếp, tìm kiếm?

TreeMap

- Là lớp thực thi của SortedMap Interface
- Lưu trữ phần tử (entry) dưới dạng key-value (khóa – giá trị) và key là duy nhất và không null.
- Có thể có nhiều giá trị null
- Các phần tử được sắp xếp theo key tăng dần.
- Thuộc **java.util.TreeMap**.
- Duyệt TreeMap dùng vòng lặp For hoặc Iterator.

TreeMap

- Khai báo và khởi tạo TreeMap:
  - Khởi tạo một TreeMap rỗng

```
TreeMap<Integer, String> list = new TreeMap <>();
```

  - Khởi tạo với danh sách phần tử:

```
TreeMap <Integer , String> list = new TreeMap <>(map);
```

- Hiển thị toàn bộ TreeMap: entry

TreeMap

- Các phương thức trong TreeMap

Phương thức	Mô tả
void add(Object o)	Thêm phần tử vào TreeSet
boolean addAll(Collection c)	Thêm một collection vào TreeSet.
boolean remove(Object o)	Xóa phần tử khỏi TreeSet
void clear()	Xóa tất cả các phần tử khỏi TreeSet
boolean contains(Object o)	True nếu TreeSet chứa phần tử được chỉ định.
Object first()	Trả về phần tử đầu tiên (nhỏ nhất) của TreeSet
Object last()	Trả về phần tử cuối cùng (lớn nhất) của TreeSet
SortedSet subSet(Object fromElement, Object toElement)	Trả về SortedSet từ fromElement đến phần tử đứng trước toElement
SortedSet headSet(E toElement)	Trả về SortedSet từ phần tử đầu tiên đến phần tử đứng trước toElement
SortedSet tailSet(E fromElement)	Trả về SortedSet từ phần tử lớn hơn hoặc bằng fromElement đến phần tử cuối cùng
int size()	Trả về số phần tử trong TreeSet
boolean isEmpty()	True nếu TreeSet rỗng.