CÔNG NGHỆ JAVA

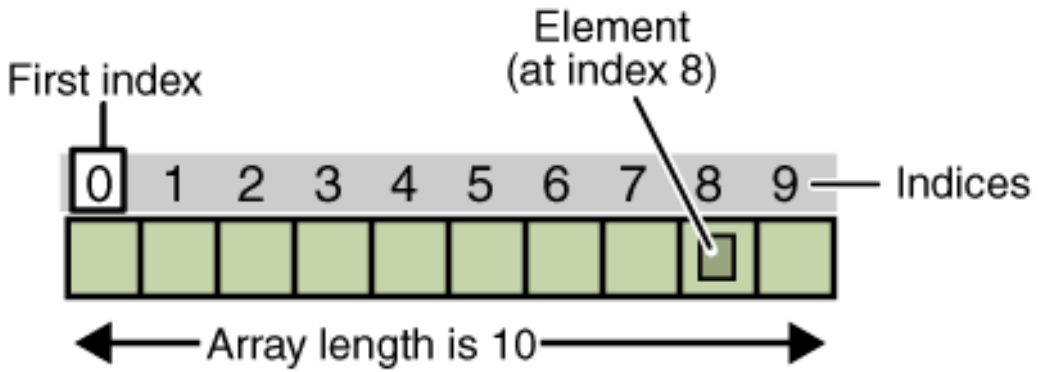
KHÁI NIỆM VỀ COLLECTIONS

Nội dung

- Collectión
- Map
- File & IO

Tập hợp và mảng

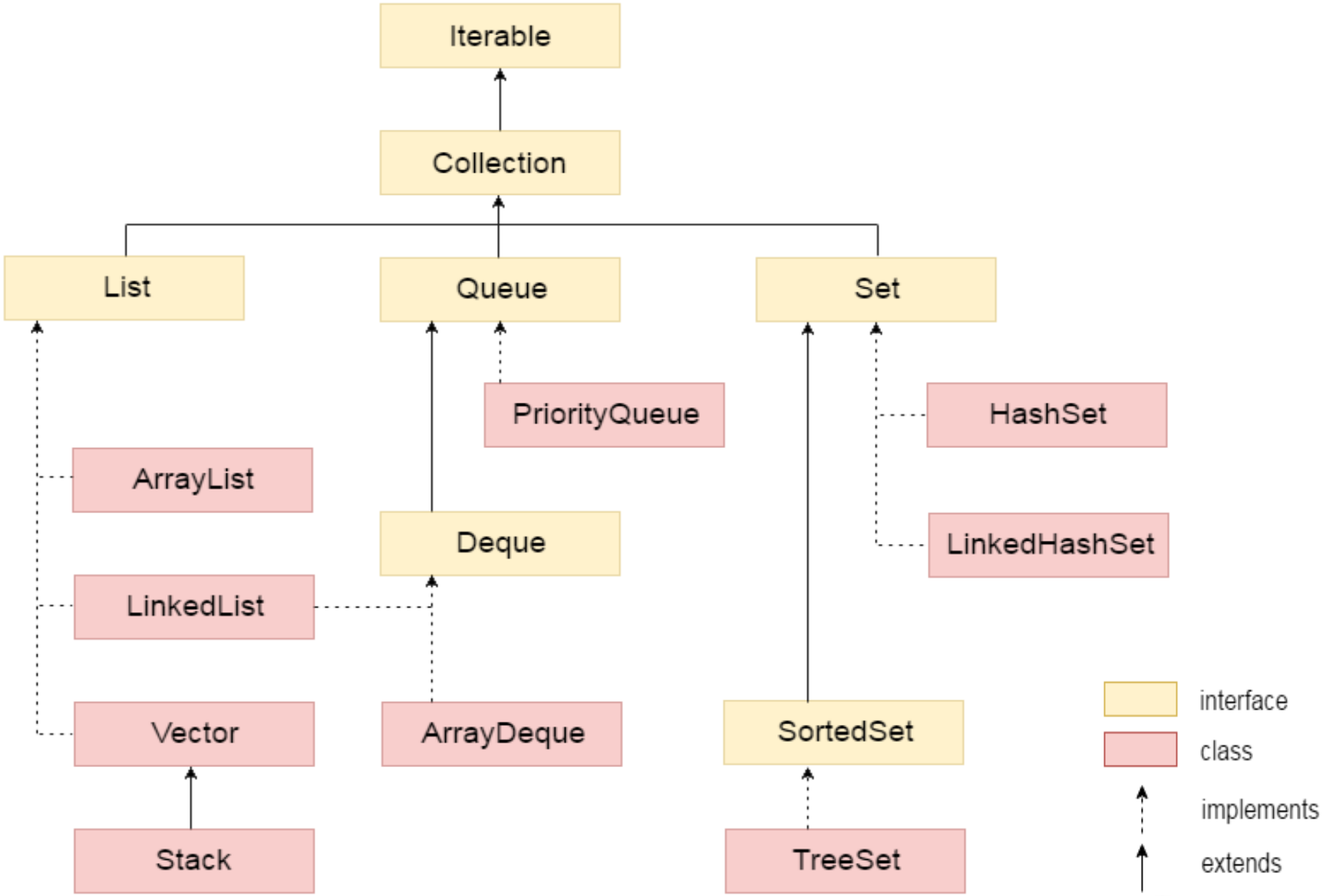
- **Một ứng dụng thường xuyên phải làm việc với tập hợp**
  - **Ví dụ: Lưu danh dách nhân viên, danh sách hóa đơn…**
- **Mảng chứa tập hợp các phần tử cùng kiểu**
- **Khi sử dụng phải khai báo số phần tử**
- **Việc thêm phần tử nằm ngoài kích thước của mảng hoặc**



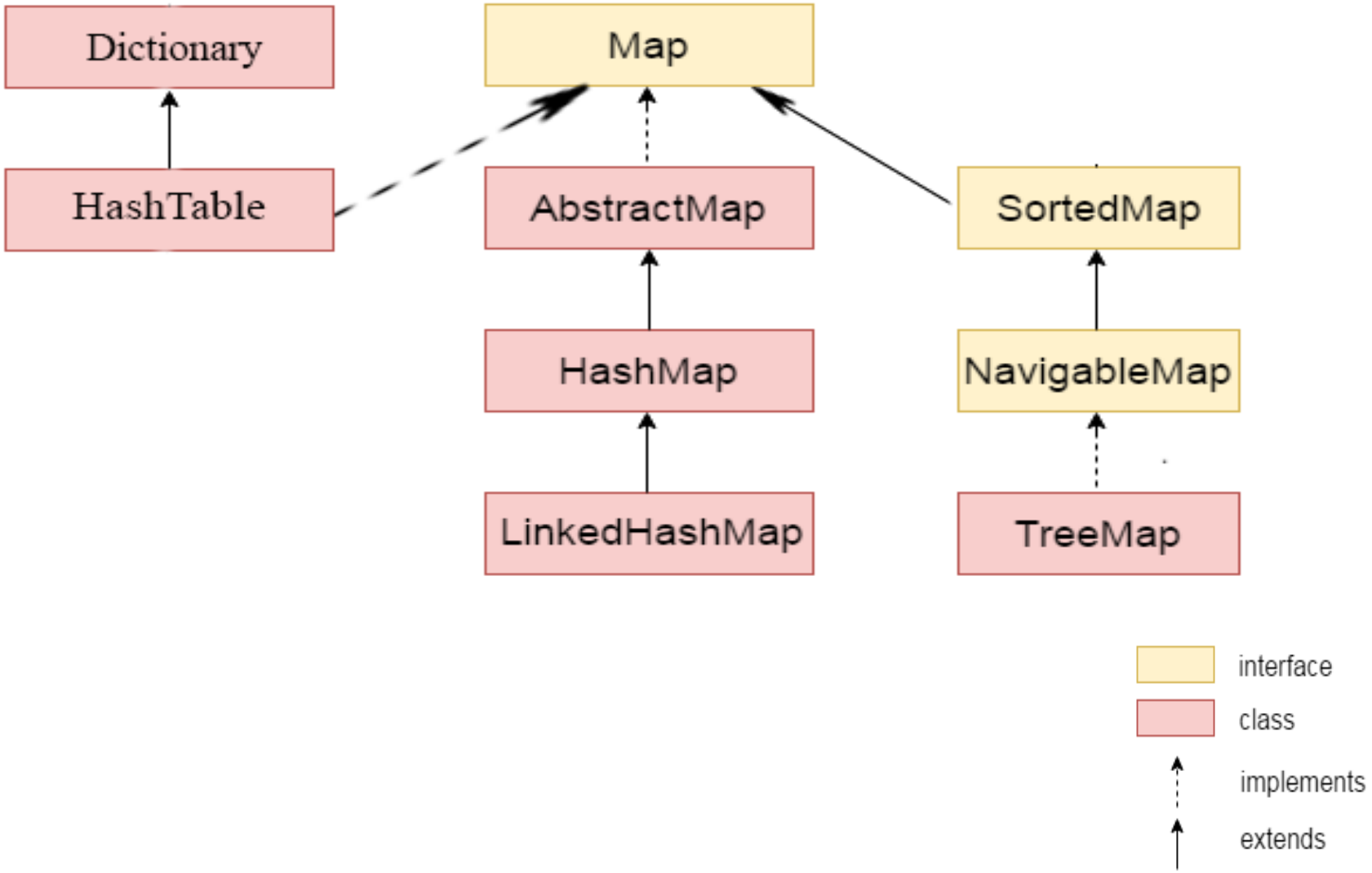
Collection trong Java

- **Collections trong java là một tập hợp các lớp (class) và các interface dùng để hỗ trợ việc thao tác trên tập các đối tượng.**
  - **Ví dụ: Tìm kiếm, sắp xếp, phân loại, thêm, sửa, xóa,… có thể được thực hiện bởi Java Collections.**
- **Collection trong java là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque...) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet, HashMap..).**
  - **Chú ý: Tất cả các interface và các lớp được định nghĩa trong java Collections Framework được nhóm lại trong package java.util**

Phân cấp Java collections



Phân cấp Java collections



Các interface chính

- **Set**: Là một collection không thể chứa 2 giá trị trùng lặp.
- **List**: là một collection có thứ tự. List có thể chứa các phần tử trùng lặp. Thường có thể kiểm soát chính xác vị trí các phần tử được chèn vào và có thể truy cập chúng bằng chỉ số.
- **Queue**: Là một collection cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Queue có thể được sử dụng như là FIFO
- **Deque**: là một queue nhưng có thể được sử dụng như là FIFO và LIFO. Tất cả các phần tử mới có thể được chèn vào, lấy ra và lấy ra ở cả hai đầu.
- **Map**: Chứa tập hợp ánh xạ mỗi key tương ứng với một giá trị. Map không thể chứa key trùng lặp. Mỗi key có thể ánh xạ đến nhiều nhất một giá trị.

Các interface chính

Phương thức	Mô tả
public boolean add(Object element)	Thêm một phần tử vào collection.
public boolean addAll(Collection c)	Thêm các phần tử collection được chỉ định vào collection gọi phương thức này.
public boolean remove(Object element)	Xóa phần tử ra khỏi collection.
public boolean removeAll(Collection c)	Xóa tất cả các phần tử từ collection được chỉ định ra khỏi collection gọi phương thức này.
public boolean retainAll(Collection c)	Xóa tất cả các thành phần từ collection gọi phương thức này ngoại trừ collection được chỉ định.
public int size()	Trả về tổng số các phần tử trong collection.
public void clear()	Xóa tất cả các phần tử trong Collection, sau khi thực hiện phương thức này, Collection sẽ rỗng (empty)
public boolean contains(Object element)	Kiểm tra một phần tử có nằm trong Collection không
public boolean containsAll(Collection c)	Kiểm tra một Collection có chứa tất cả các phần tử của một Collection khác
public Iterator iterator()	Trả về một iterator.
public Object[] toArray()	Chuyển đổi collection thành mảng (array).
public boolean isEmpty()	Kiểm tra Collection có rỗng hay không.
public boolean equals(Object element)	So sánh 2 collection.
public int hashCode()	Trả về số hashcode của collection.

Generic collection

- **Từ Java 1.5 collection trong Java là generic**
- **Generic cho phép chỉ định kiểu dữ liệu của phần tử**

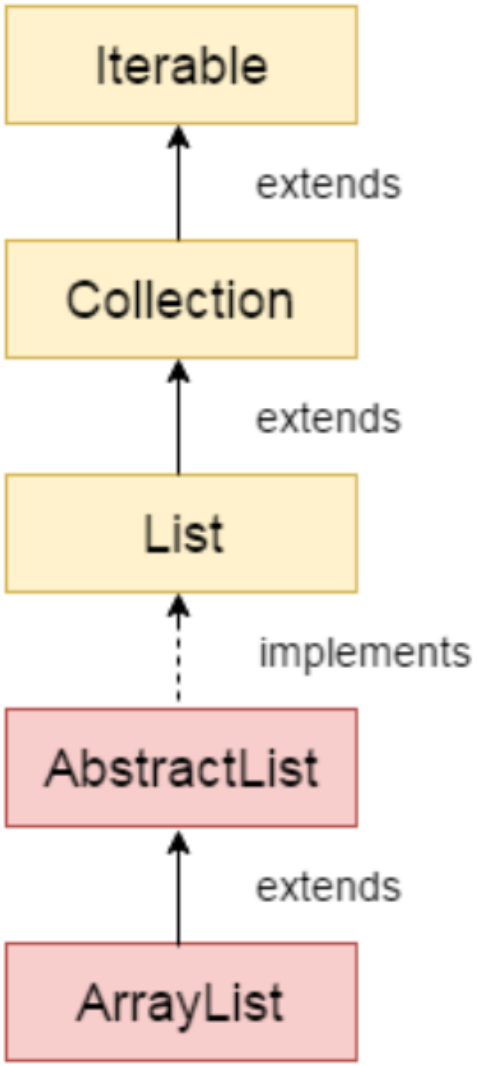
```
import java.util.ArrayList;
import java.util.List;

public class CreateListDemo {
	public static void main(String[] args) {
	List<String> list1 = new ArrayList<String>();
	list1.add("hello");
	list1.add("java");
	list1.add("collection");
    }
}
```

Duyệt collection

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class CreateListDemo {
    public static void main(String[] args) {
        List<String> list1 = new ArrayList<String>();
        list1.add("hello");
        list1.add("java");
        list1.add("collection");

        System.out.println("1. Using iterator");
        Iterator<String> it = list1.iterator();
        while(it.hasNext()) {
                System.out.println(it.next());
        }
        System.out.println("2. Using for");
        for(String s : list1) {
                System.out.println(s);
        }
        System.out.println("3. Using for with index");
        for (int i = 0; i < list1.size(); i++) {
                System.out.println(list1.get(i));
        }}}
```

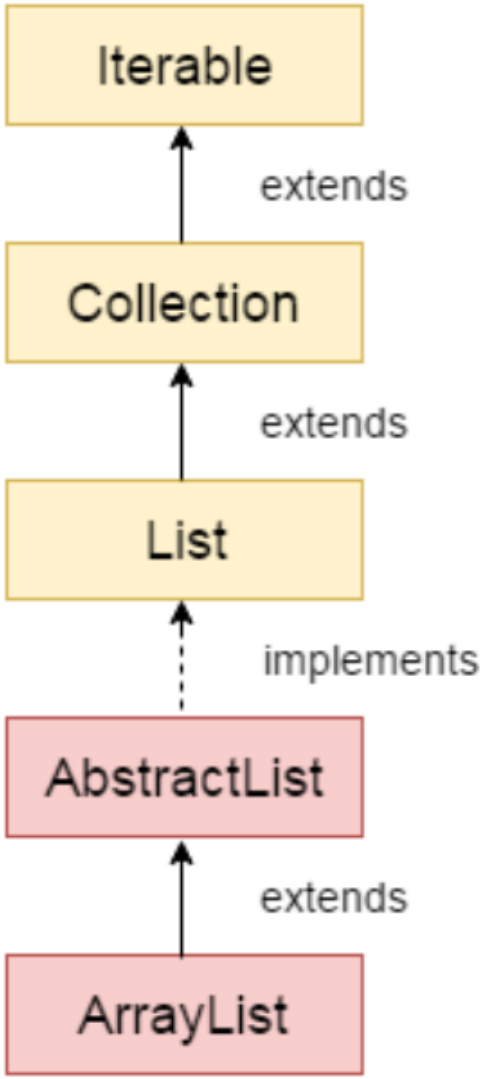
- **Có 3 cách:**
  - **Thông qua iterator**
  - **for**
  - **for-each loop**

ArrayList



- Lớp Java ArrayList sử dụng một mảng động để lưu trữ các phần tử. (giống như một mảng, nhưng không có giới hạn về kích thước)
- Có thể thêm/bớt các phần tử bất cứ lúc nào. Vì vậy, nó linh hoạt hơn nhiều so với mảng truyền thống.
- Nằm trong gói java.util (giống như Vector trong C ++)

ArrayList



- Đặc điểm:

❖ Có thể chứa các phần tử trùng lặp.

❖ Duy trì thứ tự chèn.

❖ Không được đồng bộ hóa.

❖ Cho phép truy cập ngẫu nhiên vì mảng hoạt động ở cơ sở chỉ mục.

❖ Thao tác chậm hơn so với LinkedList trong Java vì rất nhiều chuyển đổi cần xảy ra nếu bất kỳ phần tử nào bị xóa khỏi danh sách mảng.

ArrayList

```
// Khai báo ArrayList
// thì import gói thư viện java.util.ArrayList
import java.util.ArrayList;
public class TênClass {
    // ...
}

public static void main(String[] args) {
    // khai báo 1 ArrayList có tên là arrListString
    // có kiểu là String
    ArrayList<String> arrListString = new ArrayList<>();
}


public static void main(String[] args) {
    // khai báo 1 ArrayList có tên là arrListName
    // có kiểu là String và có 10 phần tử
    ArrayList<String> arrListName = new ArrayList<>(10);
}
```

ArrayList

Phương thức	Ý nghĩa
void add (int index, E element)	Thêm phần tử được chỉ định vào vị trí
boolean add (E e)	Thêm phần tử vào cuối danh sách.
boolean addAll (Collection<? expand E> c)	Nối All phần tử trong tập hợp đã chỉ định vào cuối danh sách theo thứ tự chúng được trả về bởi trình lặp của tập hợp đã chỉ định.
boolean addAll (int index, Collection <? expand E> c)	Nối All các phần tử trong tậPp hợp đã chỉ định, bắt đầu từ vị trí đã chỉ định của danh sách.
void clear ()	Xóa tất cả các phần tử khỏi danh sách
E get (int index)	Nó được sử dụng để tìm nạp phần tử từ vị trí cụ thể của danh sách.
boolean isEmpty ()	Nó trả về true nếu danh sách trống, ngược lại là false.
int lastIndexOf (Object o)	trả về chỉ mục xuất hiện cuối cùng của phần tử được chỉ định hoặc -1 nếu danh sách không chứa phần tử này.

ArrayList

Phương thức	Ý nghĩa
Object[] toArray ()	Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự.
<T> T [] toArray (T [] a)	Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự.
boolean chứa (Objecto)	Nó trả về true nếu danh sách chứa phần tử được chỉ định
int indexOf (Object o)	Nó được sử dụng để trả về chỉ mục trong danh sách này về lần xuất hiện đầu tiên của phần tử được chỉ định hoặc -1 nếu Danh sách không chứa phần tử này.
E remove (int index)	Nó được sử dụng để loại bỏ phần tử hiện diện ở vị trí được chỉ định trong danh sách.
boolean removeAll (Collection <?> c)	Nó được sử dụng để xóa tất cả các phần tử khỏi danh sách.

ArrayList

```java
package ArrayList;
import java.util.ArrayList;
public class ArrListToArr {
    public static void main(String[] args) {
// khai báo một ArrayList languages
    ArrayList<String> arr_list= new ArrayList<>();

//Sử dụng phương thức add() để thêm các phần tử cho languages
    arr_list.add("Java");
    arr_list.add("Python");
    arr_list.add("JavaScript");
    System.out.println("\n\nCác phần tử trong ArrayList: " + arr_list);

// khai báo một mảng arr với độ dài bằng độ dài của languages
    String[] arr = new String[arr_list.size()];
    //Sử dụng phương thức toArray() để chuyển đổi languages thành mảng arr
    arr_list.toArray(arr);

//dung vòng lặp for để in các phần tử trong mảng arr ra màn hình
    System.out.print("Các phần tử trong mảng array: ");
    for(String item:arr) {
      System.out.print(item+", ");
    }
```

ArrayList

VD: Tạo Array và chuyển đổi sang ArrayList

```
package Collection;
import java.util.ArrayList;
import java.util.Arrays;
public class ArrToArrList {
    public static void main(String[] args) {
// khai báo và khởi tạo giá trị cho các phần tử trong mảng array
    String[] array = {"Java", "Python", "C++"};

//hiển thị các phần tử trong mảng bằng phương thức toString()
    System.out.println("\n\nCác phần tử trong Array: "+ Arrays.toString(array));

//chuyển array thành ArrayList
    ArrayList<String> arr_list= new ArrayList<>(Arrays.asList(array));

//hiển thị ArrayList
    System.out.println("Các phần tử trong ArrayList: " + arr_list);
    System.out.println("\n-----------------------------------");
  }
}
```

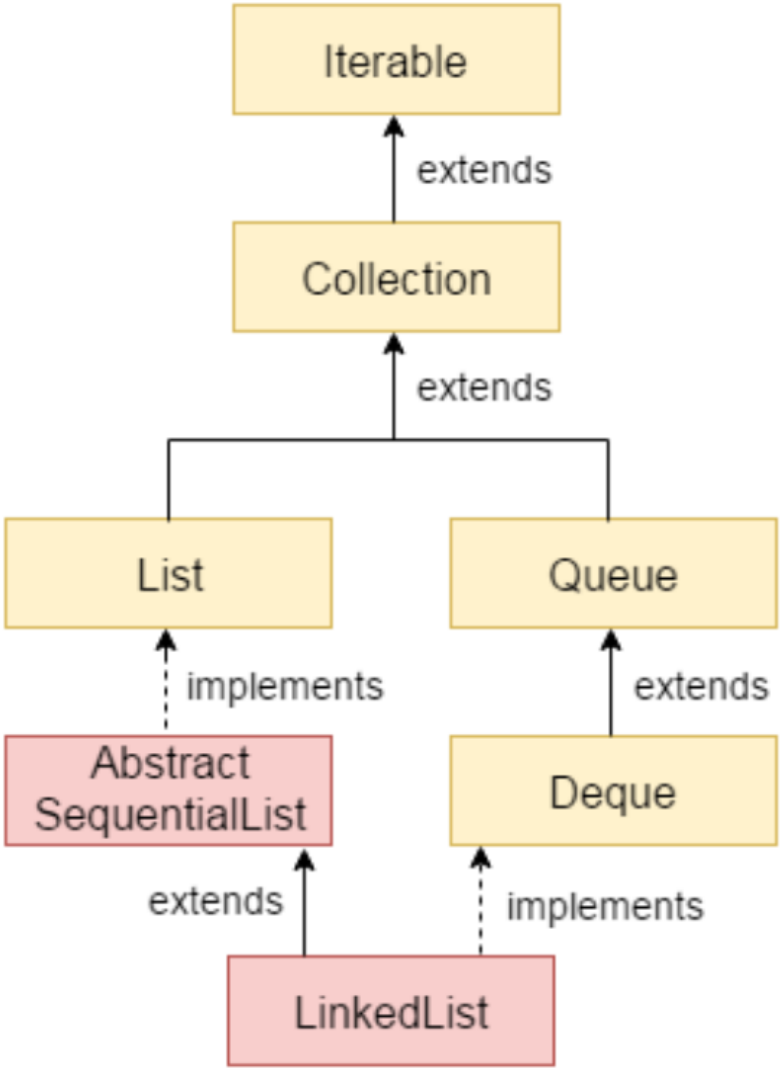
Get and Set ArrayList

```
package Collection;
import java.util.ArrayList;

public class Get_Set_ArrList {
    public static void main(String[] args) {
  ArrayList<String> al=new ArrayList<String>();
  al.add("Orange");
  al.add("Apple");
  al.add("Banana");
  al.add("Grapes");
  //lấy giá trị của phần tử 1
  System.out.println("Returning element: "+al.get(1));
  //gán giá trị cho phần tử 1
  al.set(1,"Dates");
  for(String fruit:al)
    System.out.println(fruit);
 }
}
```

Sort ArrayList

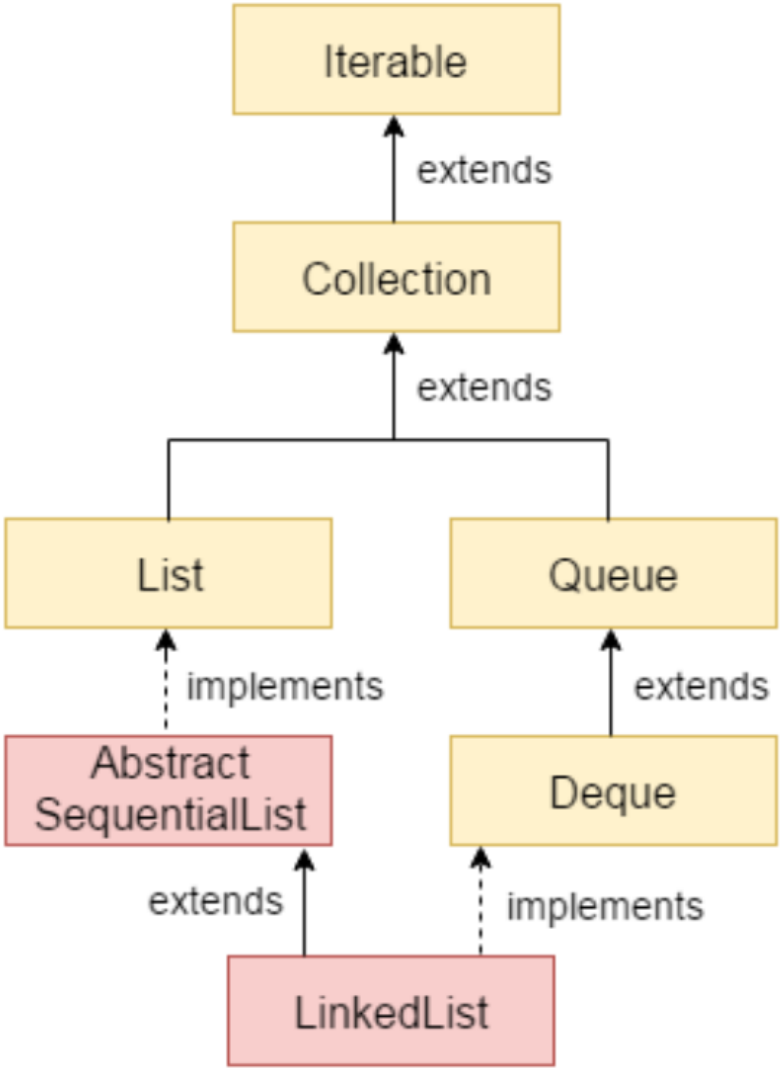
```
package Collection;
import java.util.ArrayList;
import java.util.Collections;

public class Get_Set_ArrList {
    public static void main(String[] args) {
  ArrayList<String> al=new ArrayList<String>();
  al.add("Orange");
  al.add("Apple");
  al.add("Banana");
  al.add("Grapes");
  //lấy giá trị của phần tử 1
  System.out.println("Returning element: "+al.get(1));
  //gán giá trị cho phần tử 1
  al.set(1,"Dates");
  Collections.sort(al);
  for(String fruit:al)
    System.out.println(fruit);
 }
}
```

LinkedList



- Lớp Java LinkedList sử dụng một danh sách được liên kết kép để lưu trữ các phần tử.
- Nó cung cấp một cấu trúc dữ liệu danh sách liên kết. Nó kế thừa lớp AbstractList và triển khai các giao diện List và Deque.

LinkedList



Đặc điểm:

- Có thể chứa các phần tử trùng lặp.
- Duy trì thứ tự chèn.
- Không được đồng bộ hóa.
- Thao tác nhanh vì không cần phải dịch chuyển.
- Có thể được sử dụng như một danh sách, ngăn xếp hoặc hàng đợi.

LinkedList

Có 2 hàm tạo

Constructor	Ý nghĩa
LinkedList ()	Nó được sử dụng để xây dựng một danh sách trống.
LinkedList (Collection<? Expand E> c)	Nó được sử dụng để tạo một danh sách chứa các phần tử của tập hợp được chỉ định, theo thứ tự

LinkedList

Ví dụ:

```
import java.util.LinkedList;

public class LinkedList_Demo {
    public static void main(String[] args) {
        LinkedList<String> ll=new LinkedList<String>();
        ll.add("Hello");
        ll.add("java");
        ll.add("C++");
        System.out.println("After invoking add(E e) method: "+ll);

    //Adding an element at the specific position
        ll.add(1, "Python");
        System.out.println("After invoking add(int index, E element) method: "+ll);
        ll.addFirst("C");
        ll.addLast("PHP");
        System.out.println("After invoking addFirst and addLast: "+ll);
    }
}
```

Viết đoạn chương trình xóa phần tử thứ 2, phần tử đầu tiên, phần tử cuối cùng

LinkedList

Ví dụ:

```
import java.util.LinkedList;

public class LinkedList_Demo {
    public static void main(String[] args) {
        LinkedList<String> ll=new LinkedList<String>();
        ll.add("Hello");
        ll.add("java");
        ll.add("C++");
        System.out.println("After invoking add(E e) method: "+ll);

    //Adding an element
        ll.add(1, "Python");
        System.out.println("After invoking add(int index, E element) method: "+ll);
        ll.addFirst("C");
        ll.addLast("PHP");
        System.out.println("After invoking addFirst and addLast: "+ll);
    //Removing an element
        ll.remove(2);
        ll.removeFirst();
        ll.removeLast();
        System.out.println("After invoking remove: "+ll);
    }
}
```

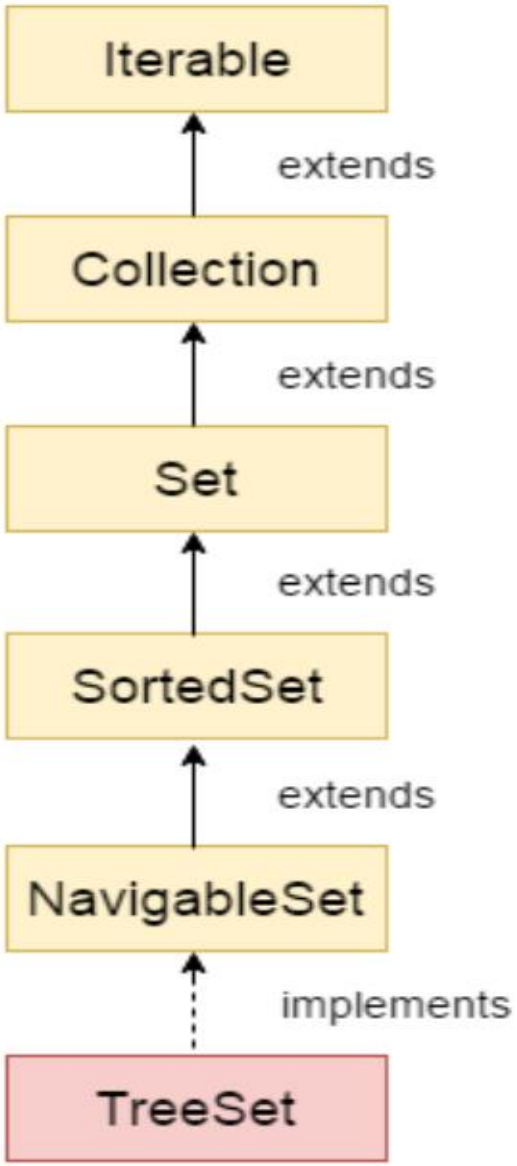
ArrayList & LinkedList

ArrayList	LinkedList
1) ArrayList sử dụng một **mảng động** để lưu trữ các phần tử.	LinkedList sử dụng **danh sách được liên kết kép** để lưu trữ các phần tử.
2) Thao tác với ArrayList **chậm** hơn	LinkedList **nhanh** hơn ArrayList
3) Một lớp ArrayList chỉ có thể **hoạt động như một danh sách** vì nó chỉ thực thi Danh sách.	Lớp LinkedList có thể **hoạt động như một danh sách và hàng đợi** vì nó triển khai các giao diện List và Deque.
4) ArrayList **tốt hơn để lưu trữ và truy cập** dữ liệu.	LinkedList **tốt hơn để thao tác** dữ liệu.

ArrayList & LinkedList

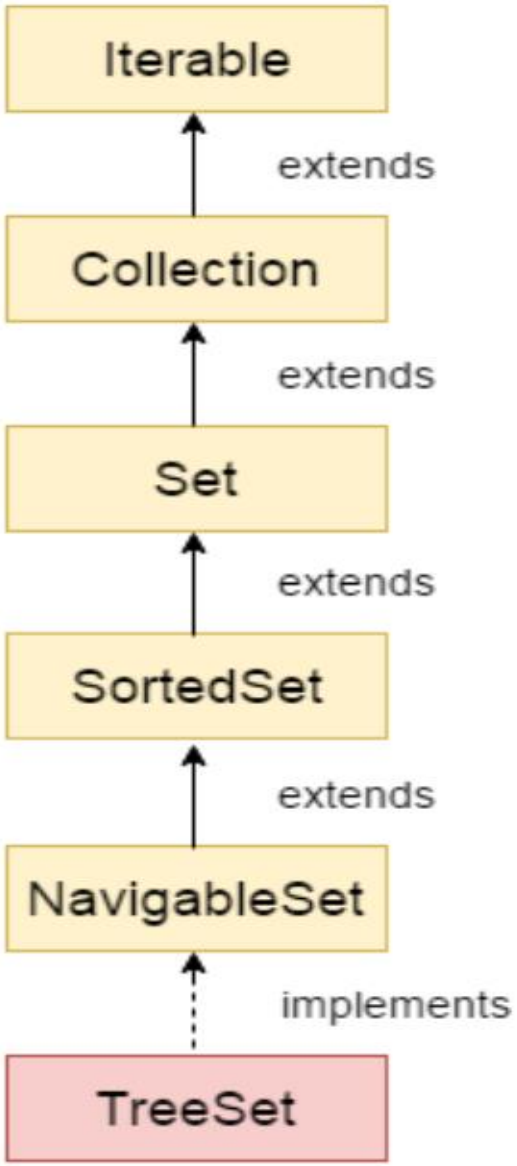
```
public class ArrList_LinkedList {
    public static void main(String[] args) {
        List<String> al=new ArrayList<String>();//creating arraylist
            al.add("Hello you!");
            al.add("Java");
            al.add("Python");
            al.add("PHP");

        List<String> al2=new LinkedList<String>();//creating linkedlist
            al2.add("Hello you!");
            al2.add("Java");
            al2.add("Python");
            al2.add("PHP");

        System.out.println("arraylist: "+al);
        System.out.println("linkedlist: "+al2);
 }
}
```

TreeSet



- Lớp Java TreeSet triển khai giao diện Set sử dụng một cây để lưu trữ.
- Nó kế thừa lớp AbstractSet và triển khai giao diện NavigableSet.
- Các đối tượng của lớp TreeSet được lưu trữ theo thứ tự tăng dần.

TreeSet



Đặc điểm:

- Lớp Java TreeSet chỉ chứa các phần tử duy nhất
- Thời gian truy cập và truy xuất lớp Java TreeSet rất nhanh.
- Lớp Java TreeSet không cho phép phần tử null.
- Không được đồng bộ hóa.
- Duy trì thứ tự tăng dần.

TreeSet

```
import java.util.Iterator;
import java.util.TreeSet;
public class TreeSet_Demo {
    public static void main(String[] args) {
        TreeSet<Integer> treeSetInteger = new TreeSet<>();
        treeSetInteger.add(15);
        treeSetInteger.add(90);
        treeSetInteger.add(40);
        treeSetInteger.add(22);
    // hiển thị các phần tử có trong treeSetInteger
    System.out.println("Các phần tử của treeSetInteger là: ");
    for(int number : treeSetInteger) {
        System.out.print(number + "\t");
    }   System.out.println();
     //hiển thị các phần tử có trong treeSet bằng cách sử dụng Iterator
    Iterator<Integer> iterator = treeSetInteger.iterator();
    System.out.println("Các phần tử có trong treeSet là: ");
    while (iterator.hasNext()) {
        System.out.print(iterator.next() + "\t");
    }
    System.out.println();
}
}
```

TreeSet

Thêm phần tử vào TreeSet:

```
System.out.println("Nhập phần tử cần thêm: ");
    Scanner sc = new Scanner(System.in);
    int number = sc.nextInt();

// thêm một phần tử mới vào treeSetInteger
if (!treeSetInteger.contains(number)) {
    treeSetInteger.add(number);
    System.out.println("Thêm thành công!");
    System.out.println("TreeSetInteger sau khi thêm: ");
    System.out.println(treeSetInteger);
} else {
    System.out.println("Phần tử " + number + " đã tồn tại!");
}
```

TreeSet

Thêm tất cả các phần tử của 1 TreeSet khác vào TreeSet:

```
TreeSet<Integer> treeSetInteger2 = new TreeSet<>();
    treeSetInteger2.add(33);
    treeSetInteger2.add(88);
 // thêm các phần tử của treeSetInteger2 vào treeSetInteger
    treeSetInteger.addAll(treeSetInteger2);
    System.out.println("TreeSetInteger sau khi thêm TreeSetInteger2:");
    System.out.println(treeSetInteger);
```

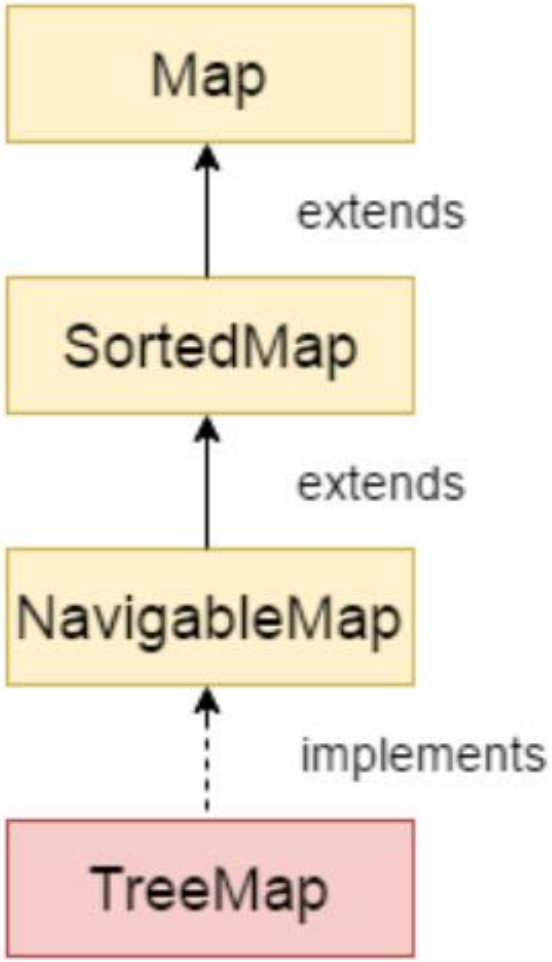
```
Các phần tử của treeSetInteger là:
15      22      40      90
Các phần tử có trong treeSet là:
15      22      40      90
Nhập phần tử cần thêm: 44
Thêm thành công!
TreeSetInteger sau khi thêm:
[15, 22, 40, 44, 90]
TreeSetInteger sau khi thêm TreeSetInteger2:
[15, 22, 33, 40, 44, 88, 90]
```

Viết đoạn chương trình xóa 1 phần tử

TreeSet

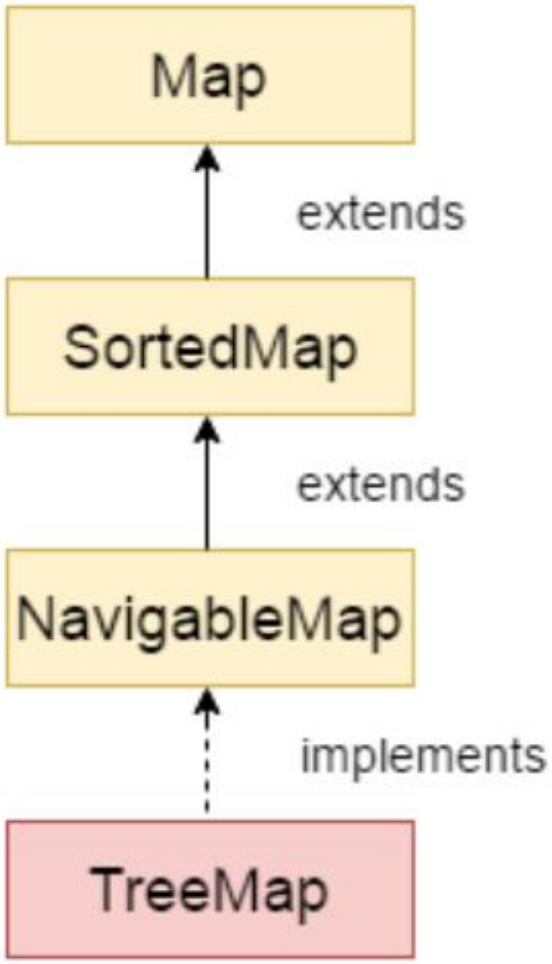
```
// xóa phần tử của treeSetInteger
        System.out.println("Nhập phần tử cần xóa: ");
        number = sc.nextInt();
    if (treeSetInteger.contains(number)) {
        treeSetInteger.remove(number);
        System.out.println("Xóa thành công!");
        System.out.println("Các phần tử còn lại trong treeSetInteger:");
        System.out.println(treeSetInteger);
} else {
    System.out.println("Xóa không thành công!");
}
```

TreeMap



- Lớp Java TreeMap là một triển khai dựa trên cây đỏ-đen.
- Nó cung cấp một phương tiện hiệu quả để lưu trữ các cặp khóa-giá trị theo thứ tự được sắp xếp.

TreeMap



Đặc điểm:

- Java TreeMap chứa các giá trị dựa trên khóa. Nó triển khai giao diện NavigableMap và mở rộng lớp AbstractMap.
- Chỉ chứa các phần tử duy nhất.
- Không được có khóa null nhưng có thể có nhiều giá trị null.
- Không được đồng bộ hóa.
- Duy trì thứ tự tăng dần.

TreeMap

```
import java.util.Map.Entry;
import java.util.Set;
import java.util.TreeMap;

public class TreeMap_Demo {
    public static void main(String[] args) {
    TreeMap<Integer, Character> treeMap = new TreeMap<>();

    // Thêm value vào trong treeMap với key tương ứng
    treeMap.put(6, 'A');
    treeMap.put(5, 'B');
    treeMap.put(1, 'C');
    treeMap.put(2, 'D');
    treeMap.put(8, 'E');

    // tạo 1 Set có tên là setTreeMap chứa toàn bộ các entry
    // (vừa key vừa value) của treeMap
    Set<Entry<Integer, Character>> setTreeMap = treeMap.entrySet();

    // các entry trong setTreeMap sẽ được sắp xếp tăng dần theo khóa
    System.out.println("Các entry có trong setTreeMap:");
    System.out.println(setTreeMap);
    }
}
```

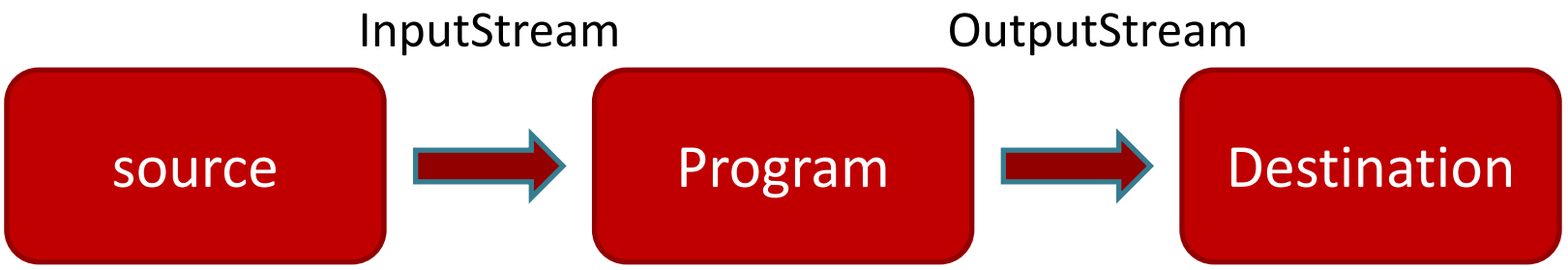
Bài tập

- **Chỉnh sửa lại bài tập ở phần OOP: Thay mảng bởi List**

FILE VÀ IO

Dòng dữ liệu (Stream)

- Nhập liệu là tác vụ đưa dữ liệu vào trong chương trình/module (bàn phím, file, module khác)
- Xuất dữ liệu là tác vụ đưa dữ liệu ra nơi chứa khác/module khác (file, màn hình, máy in, module khác)
- Một dòng (**stream**) có thể được định nghĩa là một chuỗi dữ liệu tuần tự, có 2 dòng dữ liệu:
  - **InputStream**: Dùng để đọc dữ liệu từ nguồn
  - **OutputStream**: Dùng để ghi dữ liệu ra đích



Byte Stream

- Byte stream dùng để đọc/ghi byte dữ liệu (8 bit)
- **FileInputStream** và **FileOutputStream** là 2 lớp được sử dụng thường xuyên nhất

Lớp OutputStream: là một lớp trừu tượng

Method	Description
1) public void write(int)throws IOException	được sử dụng để ghi một byte đến output stream hiện tại.
2) public void write(byte[])throws IOException	được sử dụng để ghi một mảng các byte đến output stream hiện tại.
3) public void flush()throws IOException	flush output stream hiện tại.
4) public void close()throws IOException	được sử dụng để đóng output stream hiện tại.

Byte Stream

- Byte stream dùng để đọc/ghi byte dữ liệu (8 bit)
- **FileInputStream** và **FileOutputStream** là 2 lớp được sử dụng thường xuyên nhất

Lớp InputStream: là một lớp trừu tượng

Method	Description
1. public abstract int read()throws IOException	Đọc byte kế tiếp của dữ liệu từ input stream. Nó trả về -1 khi đọc đến vị trí cuối tập tin.
2. public int available()throws IOException	Trả về một ước tính về số byte có thể đọc được từ input stream hiện tại.
3. public void close()throws IOException	được sử dụng để đóng input stream hiện tại.
4. public abstract int read()throws IOException	Đọc byte kế tiếp của dữ liệu từ input stream. Nó trả về -1 khi đọc đến vị trí cuối tập tin.

Byte Stream

Ví dụ: Copy file

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopyDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("C:\\java\\file1.txt");
            out = new FileOutputStream("C:\\java\\file2.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally { if (in != null) {in.close();}
                if (out != null) {    out.close();}
        }
        System.out.println("Done");
    }
}
```

Byte Stream

- Copy file sử dụng **BufferedInputStream** và **BufferedOutputStream** để tăng tốc độ đọc/ghi

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopyDemo {
    public static void main(String[] args) throws IOException {
        BufferedInputStream in = null;
        BufferedOutputStream out = null;
        try {
            in = new BufferedInputStream(
                new FileInputStream("C:\\java\\file1.txt"));
            out = new BufferedOutputStream(
                new FileOutputStream("C:\\java\\file2.txt"));
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {if(in != null) {in.close();}
            if(out != null) {out.close();}
        }
        System.out.println("Copied successfully!");
    }}
```

Tự đóng stream sử dụng try()

- Các lớp implements interface **Closeable** có thể được sử dụng với try để tự đóng (phương thức **close()** sẽ tự được gọi khi kết thúc khối *try*
- Tất cả các **InputStream**, **OutputStream**, **Reader**, **Writer** đều implements interface **Closeable** và có thể được sử dụng với *try*
- Ví dụ: Viết lại phần copy file

Tự đóng stream sử dụng try()

```
package stream_demo;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopyDemo {
    public static void main(String[] args) throws IOException {
        try (BufferedInputStream in = new BufferedInputStream(
            new FileInputStream("C:\\java\\file1.txt"));

            BufferedOutputStream out =new BufferedOutputStream(
                new FileOutputStream("C:\\java\\file2.txt"))){
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }}
        System.out.println("Copied successfully!");}}
```

Character Stream

- Character stream dùng để đọc/ghi 16 bit unicode
- **FileReader** và **FileWriter** là 2 lớp được sử dụng thường xuyên nhất
- Nội bộ **FileReader** sử dụng **FileInputStream**, **FileWriter** sử dụng **FileOutputStream**

Character Stream

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileCopyDemo {
    public static void main(String[] args) throws IOException {
        FileReader in = null;
        FileWriter out = null;

        try {
            in = new FileReader("C:\\java\\file1.txt");
            out = new FileWriter("C:\\java\\file2.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {in.close();}
            if (out != null) {out.close();}
        }
        System.out.println("Copied successfully!");
        }

}
```

Khi nào sử dụng Byte và Character Stream

Khi nào nên sử dụng Byte Stream

- Byte stream đọc từng byte, phù hợp khi cần xử lý dữ liệu binary (như file ảnh, video…)
- Các class đọc/ghi Byte Stream thường có tên …InputStream/…OutputStream

Khi nào nên sử dụng Character Stream

- Character stream có ích khi muốn xử lý file text
- Các class đọc/ghi Character Stream thường có tên …Reader/…Writer
- Việc đọc/ghi file binary sử dụng Character Stream có thể khiến hỏng dữ liệu

Đọc dữ liệu từ file vào biến String

- Sử dụng **BufferedReader**

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class BufferedReaderDemo {

    public static void main(String[] args) {
        StringBuilder contentBuilder = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new
                FileReader("C:\\java\\file1.txt"))) {

                String sCurrentLine;
            while ((sCurrentLine = br.readLine()) != null) {
                contentBuilder.append(sCurrentLine).append("\n");
                }
        } catch (IOException e) {
                throw new RuntimeException(e);
        }

        System.out.println(contentBuilder.toString());
    }
}
```

Đọc dữ liệu từ file vào biến String

- Sử dụng **Files.readAllBytes**

Chú ý: Chỉ phù hợp với file nhỏ. Dễ dẫn tới OutOfMemoryException với file lớn

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class ReadAllBytesDemo {

    public static void main(String[] args) {
        String content = "";

        try {
        byte[] bytes = Files.readAllBytes(Paths.get("C:\\java\\file1.txt"));
                content = new String(bytes);

        } catch (IOException e) {
                 throw new RuntimeException(e);
        }
        System.out.println(content);
    }

}
```

Bài tập

- Viết chương trình nhận vào đường dẫn đến 1 file text. Chương trình in ra danh sách các từ và số lần xuất hiện. Từ trùng nhau tính theo case insensitive (vd: Hello và hello coi là 1 từ). Báo lỗi khi file không tồn tại
  - Ví dụ: File “C:\\java\\word_count.txt” có nội dung: Hello this is the sample text file. This sample file contains 2 sentences
  - Chương trình sẽ in ra:

    hello: 1

    this: 2

    is: 1

    the: 1

    sample: 2

    …

Ghi String ra file

- Sử dụng **FileWriter** và **BufferedWriter**

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class StringToFile2 {
    public static void main(String[] args) throws IOException {
        String content = “I study\r\n Java \r\n";
        String path  = "C:\\java\\file2.txt";
        FileWriter fw = null;
        BufferedWriter bw = null;
        try {  fw = new FileWriter(path);
               bw = new BufferedWriter(fw);
               bw.write(content);
        } catch (IOException e) {throw new RuntimeException(e);
        }
        finally {if(bw != null) bw.close();
            if(fw != null)
            fw.close();
        }  System.out.println("Done");}
}
```

Ghi String ra file

- Sử dụng **Files.write** (Java 7)

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;

public class StringToFile {

    public static void main(String[] args) {
        String content = “I study\r\n Java \r\n";
        String path = "c:\\java\\file2.txt";
        try {
                Files.write(Paths.get(path),
        content.getBytes(StandardCharsets.UTF_8));
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
    }}
```

Ghi Object ra file

- Sử dụng **ObjectInputStream** và **ObjectOutputStream**

```
import java.io.Serializable;
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String name;

    public Student(String id, String name) {
        this.id = id; this.name = name;
    }
    public String getId() {return id;}
    public void setId(String id) {this.id = id;}
    public String getName() { return name;}
    public void setName(String name) {this.name = name;}
    @Override
    public String toString() {
        return "StudentDemo [id=" + id + ", name=" + name + "]";
    }
}
```

Ghi Object ra file

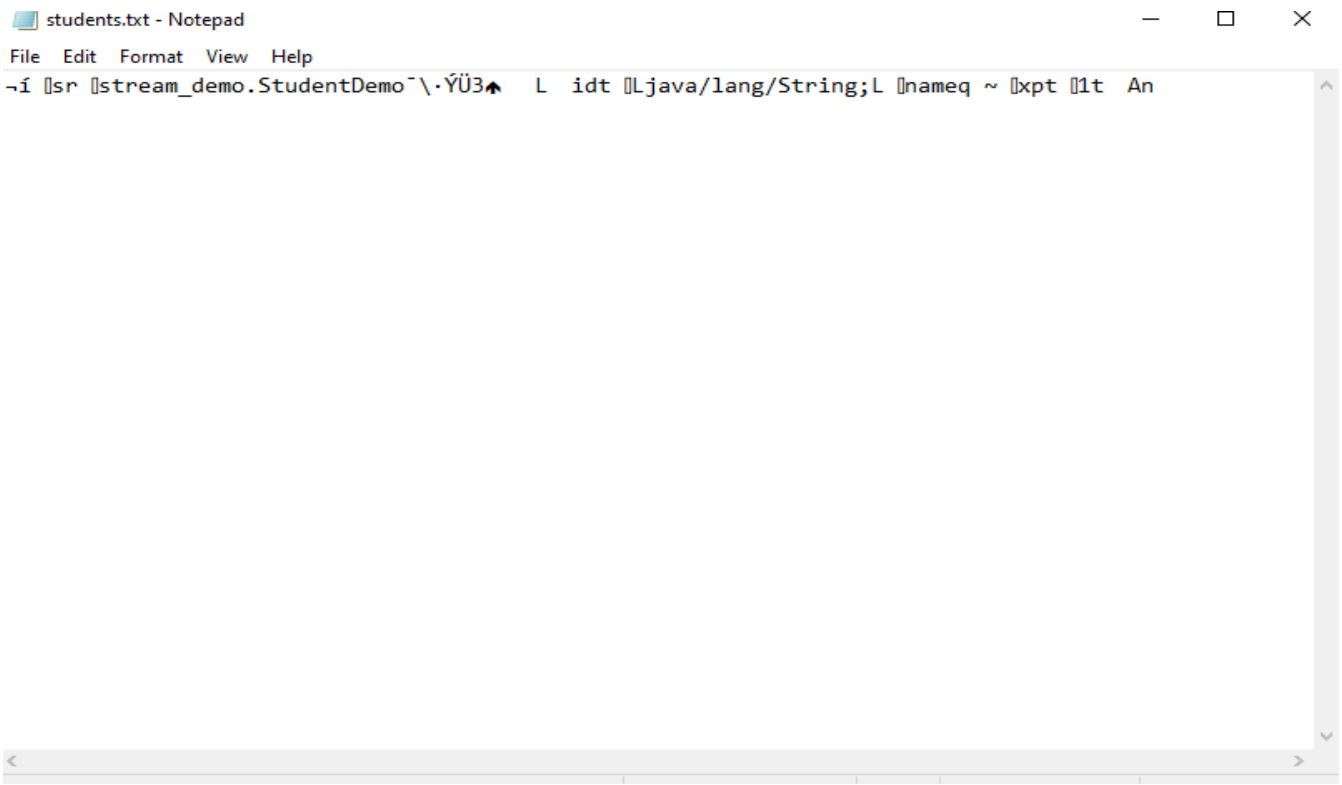
```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

public class StudentDemo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<Student>();
        students.add(new Student("1", "An"));
        students.add(new Student("2", "Binh"));
        // serialize
        try (ObjectOutputStream out = new ObjectOutputStream(
                new FileOutputStream("C:\\java\\students.dat"))) {
        for (Student s : students)
                out.writeObject(s);
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
```

Ghi Object ra file

```
    List<Student> inputStudents = new ArrayList<Student>();

      // deserialize
      // detect end of file using the FileInputStream object
      try (FileInputStream fi =
            new FileInputStream("C:\\java\\students.dat");
      ObjectInputStream in = new ObjectInputStream(fi)) {
         while (fi.available() > 0) {
            inputStudents.add((Student) in.readObject());
         }
      } catch (Exception e) {
            throw new RuntimeException(e);
      }
      for (Student s : inputStudents) {
            System.out.println(s);
      }
   }
}
```

Ghi Object ra file

- Chú ý: Khi sử dụng **ObjectInputStream** và **ObjectOutputStream**, file đầu ra không đọc được bởi người (not human readable)
- Nếu muốn file đầu ra đúng định dạnh text (csv, xml…) cần tự viết serialize và deserialize



Bài tập

- Thay đổi chương trình để serialize và deserialize danh sách Student theo định dạng csv (giá trị các trường cách nhau bởi dấu ,). Mỗi Student trên 1 dòng
  - Chú ý: Để cho đơn giản, không cần xử lý ký tự thoát