ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

IT3100

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 08. Lập trình tổng quát

Nội dung

1. Giới thiệu
   - Định nghĩa và sử dụng lập trình tổng quát
2. Cấu trúc dữ liệu tổng quát
   - Cấu trúc dữ liệu
   - Java collection framework
   - Các interface trong Java collection framework
   - Các cài đặt cho các interface – implementation
3. Ký tự đại diện (Wildcard)

1/ GIỚI THIỆU

1. GT lập trình tổng quát

- Tổng quát hóa chương trình để có thể hoạt động với các kiểu dữ liệu khác nhau, kể cả kiểu dữ liệu trong tương lai
  - Yêu cầu: thuật toán đã xác định
- Ví dụ:
  - C++: dùng template
  - Java: lợi dụng upcasting, lập trình tổng quát Generics (>1.5)

2. Lớp tổng quát

- Kiểu dữ liệu trở thành *tham số* cho phương thức, lớp, giao diện.
- Sử dụng <> để xác định tham số cho kiểu dữ liệu khi tạo lớp

```
class Test<T> {
    private T obj;
    Test(T obj)             {  this.obj = obj; }
    public T getObject()    { return this.obj; }
}
```

- Sử dụng đối tượng

```
Test <Integer> iObj = new Test<Integer>(15);
Integer num =  iObj.getObject();
Test <String> sObj = new Test<String>("GeeksForGeeks");
String  str = sObj.getObject());
```

3. Lớp tổng quát nhiều tham số

```
class Test<T, U> {
    T obj1;  // An object of type T
    U obj2;  // An object of type U
    Test(T obj1, U obj2){
         this.obj1 = obj1;
         this.obj2 = obj2;
    }
    public void print(){
         System.out.println(obj1);
         System.out.println(obj2);
     }
}

  // in main()
  Test <String, Integer> obj = new Test<String,
                                   Integer>("GfG", 15);
  obj.print();
```

4. Giao diện tổng quát

```
public interface GenericInterface<G> {
    public G doSomething();
}

public class GenericInterfaceImpl<G> implements
GenericInterface<G>{
    private G something;

    @Override
    public G doSomething() {
        return something;
    }
}
```

5. Phương thức tổng quát

```
public class KeyValue<K, V> { ... }
public class MyUtils {
    public static <K, V> K getKey(KeyValue<K, V> entry) {
         K key = entry.getKey();
         return key;
     }
}

// in main
 KeyValue<Integer, String> entry1 = new KeyValue<Integer,
String>(12000111, "Tom");
 Integer phone = MyUtils.getKey(entry1);
```

2/ CTDL TỔNG QUÁT TRONG JAVA

2.1. Cấu trúc dữ liệu

- Cấu trúc dữ liệu là cách tổ chức dữ liệu để giải quyết vấn đề.
- Một số cấu trúc dữ liệu phổ biến:
  - Mảng (Array)
  - Danh sách liên kết (Linked List)
  - Ngăn xếp (Stack)
  - Hàng đợi (Queue)
  - Cây (Tree)

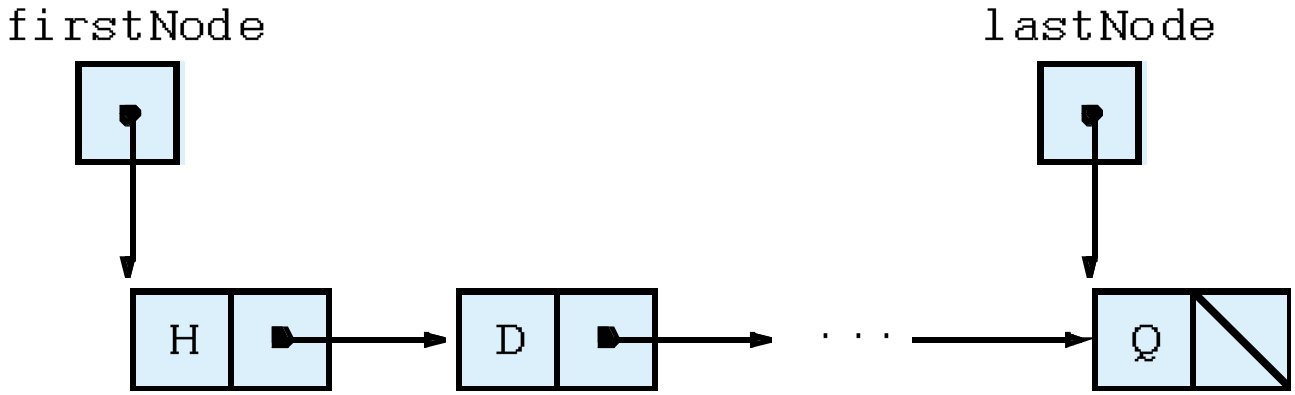
2.1.1 Danh sách liên kết

- DSLK (linked list) là cấu trúc gồm các nút liên kết với nhau thông qua các mối liên kết.
  - Nút cuối được đặt là null để đánh dấu kết thúc danh sách.
- DSLK giúp tiết kiệm bộ nhớ so với mảng trong các bài toán xử lý danh sách.
- Khi chèn/xoá một nút trên DSLK, không phải dãn/dồn các phần tử như trên mảng.
- Việc truy nhập trên DSLK luôn phải tuần tự.

2.1.1 Danh sách liên kết (2)

- Thể hiện Node thông qua lớp tự tham chiếu (self-referential class)

```
class Node
{
  private int data;
  private Node nextNode;
  // constructors and methods ...
}
```



2.1.2 Stack

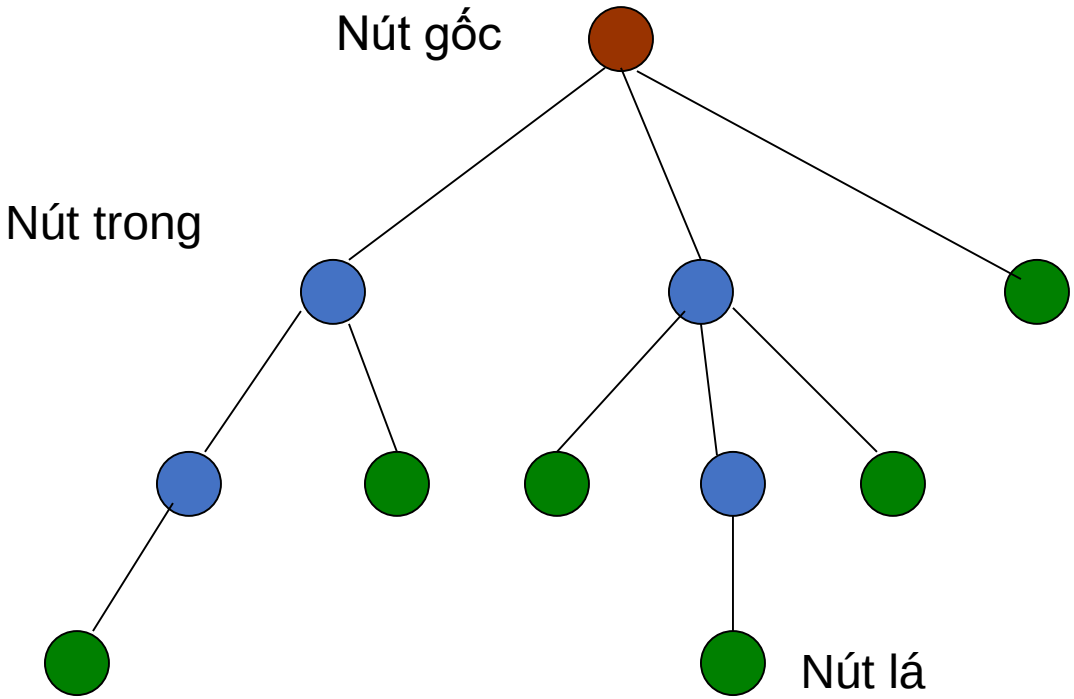
- Stack là một cấu trúc theo kiểu LIFO (Last In First Out), phần tử vào sau cùng sẽ được lấy ra trước.
- Hai thao tác cơ bản trên Stack
  - Chèn phần tử: Luôn chèn vào đỉnh Stack (push)
  - Lấy ra phần tử: Luôn lấy ra từ đỉnh Stack (pop)

2.1.3 Hàng đợi

- Queue (Hàng đợi) là cấu trúc theo kiểu FIFO (First In First Out), phần tử vào trước sẽ được lấy ra trước.
- Hai thao tác cơ bản trên hàng đợi
  - Chèn phần tử: Luôn chèn vào cuối hàng đợi (enqueue)
  - Lấy ra phần tử: Lấy ra từ đầu hàng đợi (dequeue)

2.1.4 Cây

- Cây là một cấu trúc phi tuyến (non-linear).
- Mỗi nút trên cây có thể có nhiều liên kết tới nút khác.

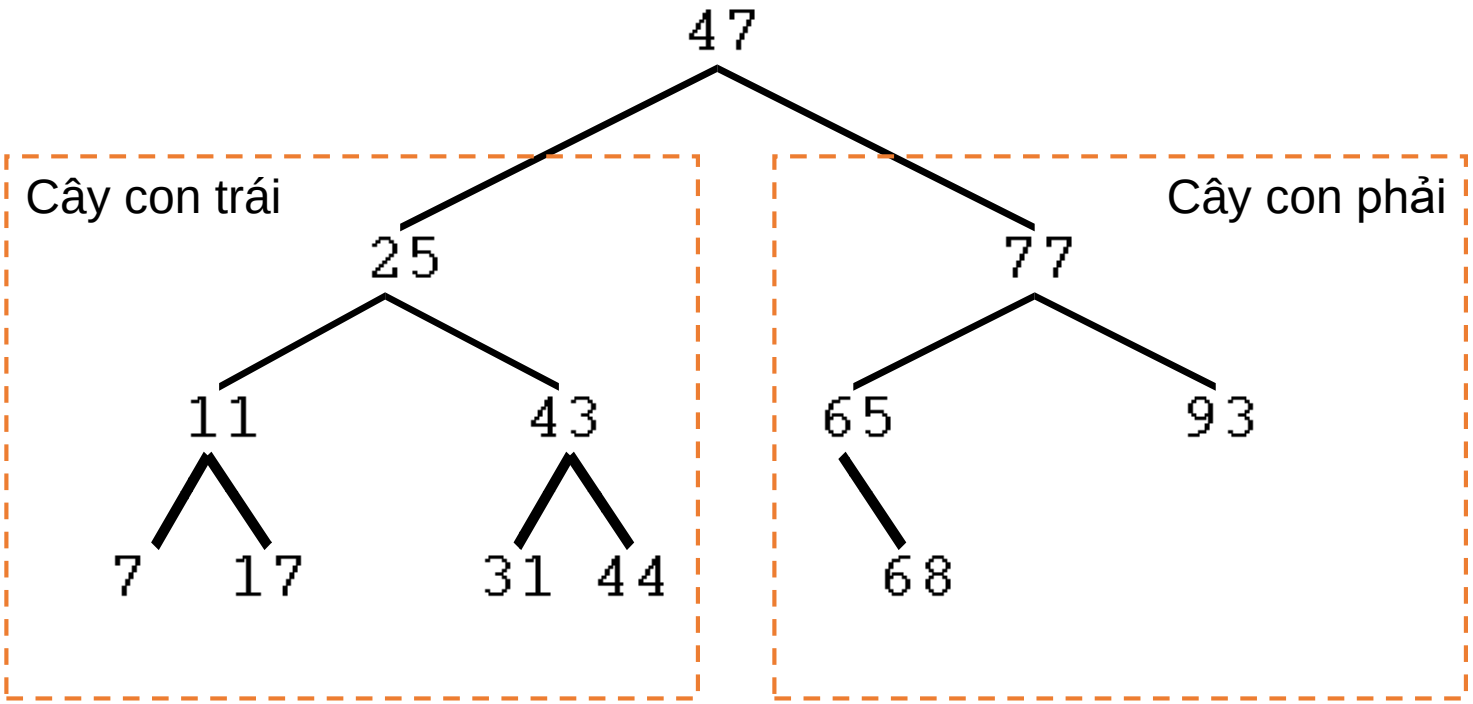


2.1.5 Cây nhị phân tìm kiếm

- Cây nhị phân là cây mà mỗi nút không có quá 2 nút con.
- Cây tìm kiếm nhị phân là cây nhị phân mà:
  - Giá trị các nút thuộc cây con bên trái nhỏ hơn giá trị của nút cha.
  - Giá trị các nút thuộc cây con bên phải lớn hơn giá trị của nút cha.
- Duyệt cây nhị phân
  - Inorder traversal
  - Preorder traversal
  - Postorder traversal

2.1.5 Cây nhị phân tìm kiếm (2)

- Ví dụ về Binary Search Tree



2.2. Java Collection Framework

- Collection là đối tượng có khả năng chứa các đối tượng khác.
- Các thao tác thông thường trên collection
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra một đối tượng có ở trong collection không
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection

2.2. Java Collection Framework (2)

- Các collection đầu tiên của Java:
  - Mảng
  - Vector: Mảng động
  - Hastable: Bảng băm
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.

2.2. Java Collection Framework (3)

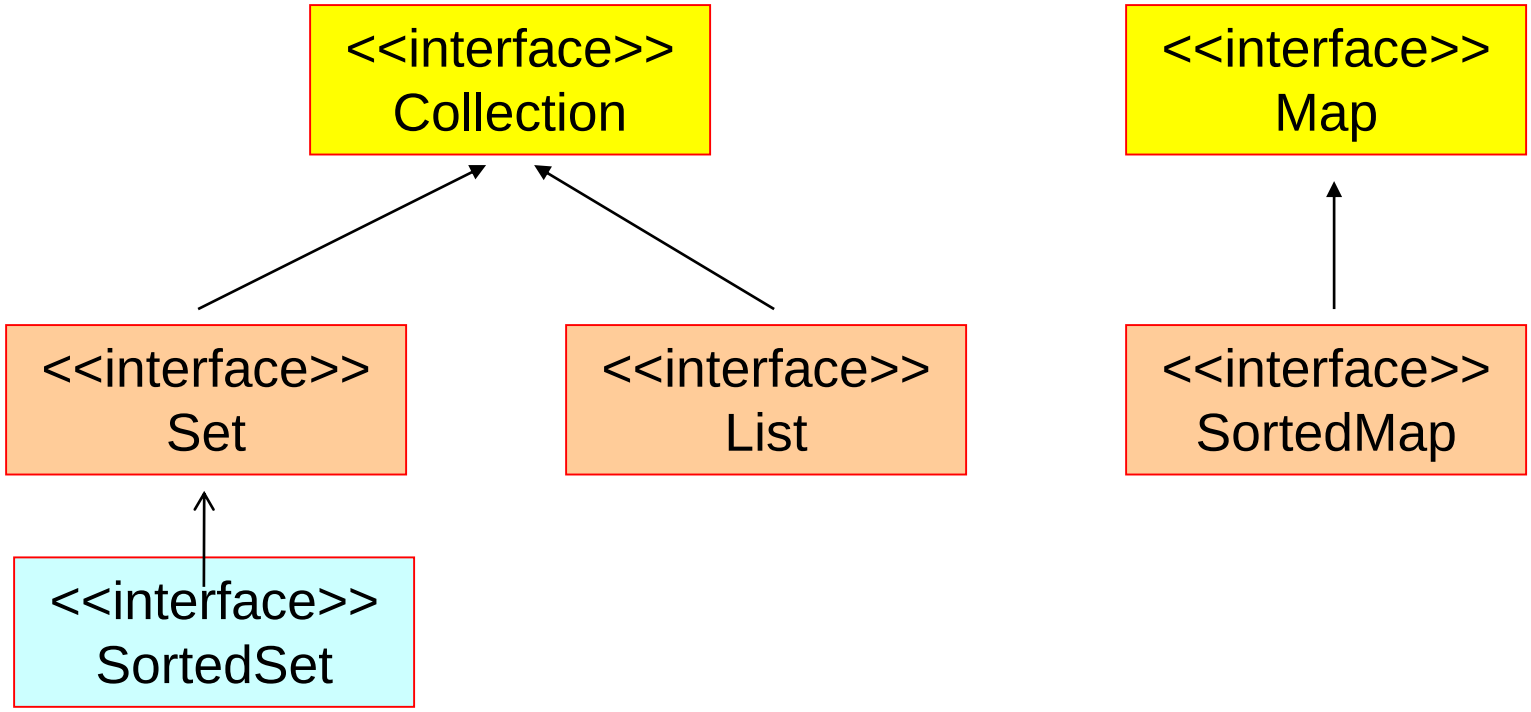
- Một số lợi ích của Collections Framework
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Khuyến khích việc sử dụng lại mã chương trình

2.2. Java Collection Framework (4)

- Collections Framework bao gồm
  - Interfaces: Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
  - Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
  - Algorithms: Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

2.2.1 Các interfaces

- List: Tập các đối tượng tuần tự, kế tiếp nhau, có thể lặp lại
- Set: Tập các đối tượng không lặp lại
- Map: Tập các cặp khóa-giá trị (key-value) và không cho phép khóa lặp lại



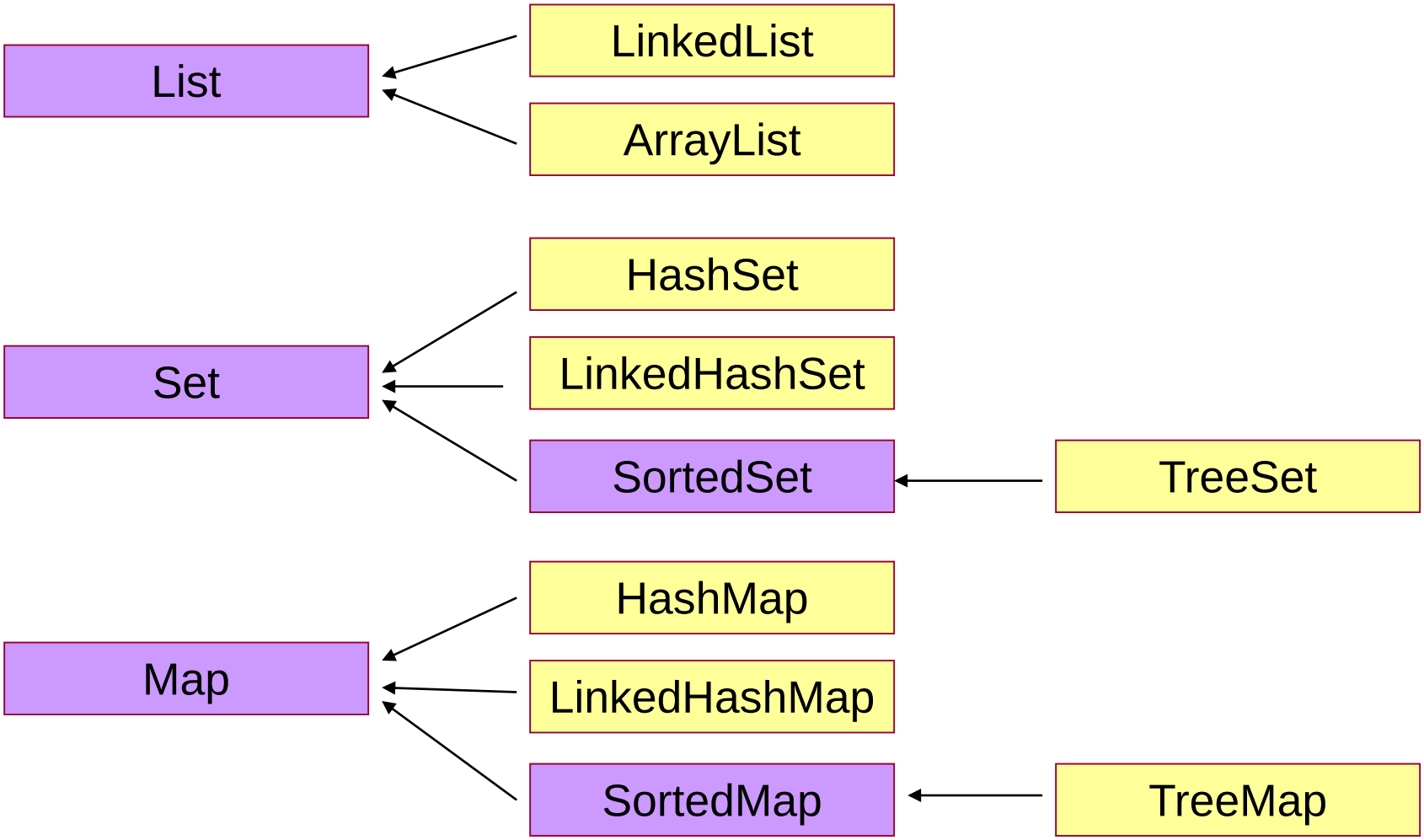
2.2.1 Các interfaces (2)

- Giao diện Collection
- Giao diện List
- Giao diện Set
- Giao diện SortedSet
- Giao diện Iterator
- Giao diện Map
- Giao diện SortedMap

2.2.2. Cài đặt interfaces

- Các cài đặt trong Collections Framework chính là các lớp collection có sẵn trong Java.
- Chúng cài đặt các collection interface ở trên để thể hiện các cấu trúc dữ liệu cụ thể
  - Ví dụ: mảng động, danh sách liên kết, bảng băm...

2.2.2. Cài đặt interfaces (2)



2.2.2. Cài đặt interfaces (3)

		IMPLEMENTATIONS				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Legacy
INTERFACES	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		HashTable, Properties

2.3 Sử dụng LTTQ trong các đối tượng Collection

- Cú pháp sử dụng generic collection

```
ClassOrInterface<Type>
```

Ví dụ sử dụng ArrayList

```
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        // Tạo ArrayList chứa các phần tử kiểu String
        ArrayList<String> userNames = new ArrayList<String>();

        // Thêm các String vào danh sách.
        userNames.add("tom");
        userNames.add("jerry");
        userNames.add(new Integer(100)); // Compile Error!

        String userName1 = userNames.get(0);
        System.out.println("userName1 = " + userName1);
    }
}
```

3. KÝ TỰ ĐẠI DIỆN (WILDCARD)

3.1. Ký tự đại diện (Wildcard)

- Trong LTTQ, dấu chấm hỏi (?), được gọi là một đại diện cho một loại/kiểu chưa rõ ràng.

3.2. Ý nghĩa ký tự đại diện

- "?": Xác định tập tất cả các kiểu hoặc bất kỳ kiểu nào.
- "? extends Type": Xác định một tập các kiểu con của Type.
- "? super Type": Xác định một tập các kiểu cha của Type
- Ví dụ:
  - ***Collection<?>*** mô tả một tập hợp chấp nhận tất cả các loại đối số (chứa mọi kiểu đối tượng).
  - ***List<? extends Number>*** mô tả một danh sách, nơi mà các phần tử là kiểu **Number** hoặc kiểu con của **Number**.
  - ***Comparator<? super String>*** Mô tả một bộ so sánh (***Comparator***) mà thông số phải là **String** hoặc cha của **String**.

3.3. Sử dụng ký tự đại diện

- ? được dùng trong LTTQ như kiểu của một tham số, trường (field), hoặc biến địa phương; đôi khi như một kiểu trả về
- không được dùng như là một đối số cho lời gọi một phương thức tổng quát, p/t khởi tạo đối tượng của lớp tổng quát, hoặc p/t lớp cha

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

Ví dụ

```
Collection<?> coll = new ArrayList<String>();

// Một tập hợp chỉ chứa kiểu Number hoặc kiểu con của Number
List<? extends Number> list = new ArrayList<Long>();

// Một đối tượng có kiểu tham số đại diện.
Pair<String,?> pair = new Pair<String,Integer>();
```

Một số khai báo không hợp lệ

```
// String không phải là kiểu con của Number, vì vậy lỗi.
List<? extends Number> list = new ArrayList<String>();

// Integer không phải là kiểu cha String của vì vậy lỗi
ArrayList<? super String> cmp = new ArrayList<Integer>();
```

Ví dụ

```
public class Example1 {
  public static void main(String[] args) {

    ArrayList<String> listString = new ArrayList<String>();
    listString.add("Tom");
    listString.add("Jerry");

    ArrayList<Integer> listInteger = new ArrayList<Integer>();
    listInteger.add(100);

    ArrayList<Object> list1 = listString; // ==> Error!

    // Một đối tượng kiểu tham số đại diện.
    ArrayList<? extends Object> list2;
    list2 = listString;  // ok
    list2 = listInteger; // ok

     }

}
```